# CODING THEORY: INTRODUCTION TO LINEAR CODES AND APPLICATIONS

**Jay Grossman***
**Undergraduate Student, B.A. in Mathematics Program, Rivier College**

## Coding Theory Basics

Coding theory is an important study which attempts to minimize data loss due to errors introduced in transmission from noise, interference or other forces. With a wide range of theoretical and practical applications from digital data transmission to modern medical research, coding theory has helped enable much of the growth in the 20th century. Data encoding is accomplished by adding additional information to each transmitted message to enable the message to be decoded even if errors occur. In 1948 the optimization of this redundant data was discussed by Claude Shannon from Bell Laboratories in the United States, but it wouldn't be until 1950 that Richard Hamming (also from Bell Labs) would publish his work describing a now famous group of optimized linear codes, the Hamming Codes. It is said he developed this code to help correct errors in punch tape. Around the same time John Leech from Cambridge was describing similar codes in his work on group theory.

### *Notation and Basic Properties*

To work more closely with coding theory it is important to define several important properties and notation elements. These elements will be used throughout the further exploration of coding theory and when discussing its applications.

There are many ways to represent a code, but perhaps the simplest way to describe a given code is as a set of codewords, i.e. {000,111}, or as a matrix with all the codewords forming the rows, such as:

$$\begin{bmatrix} 000 \\ 111 \end{bmatrix}$$

This example is a code with two codewords 000 and 111 each with three characters per codeword. In future examples it will be shown that there are more efficient methods of describing a code than to list all of its codewords. There are many mathematical advantages that can be derived simply by representing a code in a specific way.

Several notation elements will be used consistently and will be defined now. Let $n$ be the number of characters or symbols making up each codeword (vector) of the code and let M represent the number of total codewords. A code with properties $n$ and M will form a $M \times n$ matrix. In the previous example, the code {000,111} forms a $2 \times 3$ matrix. Let $q$ be the total number of possible characters in each codeword position. This is known as the alphabet of the code. Examples: A binary code uses 0 and 1, hence $q = 2$, whereas a ternary code uses 0, 1 and 2, hence $q = 3$. Codes with larger alphabets are referred to as $q$-ary codes. For example 5-ary codes use the symbols 0, 1, 2, 3 and 4.

Copyright © 2008 by Jay Grossman. Published by Rivier College, with permission.
ISSN 1559-9388 (online version), ISSN 1559-9396 (CD-ROM version).

There are also important relation functions involving codewords. Perhaps the most important is the minimum distance. The minimum distance between codewords is the minimum number of positions in which each codeword must be different from any other codeword. Example: $d(0010,0100) = 2$ with the changed positions highlighted.

With this notation defined, a code can now be described by its properties. For example, {000,111} is a binary (3,2)-code, or by specifying its minimum distance it is a binary (3,2,3)-code.

Besides notation, these properties describe important aspects about how the code will work. Since the point of coding theory is to improve the accuracy of received messages, it is obviously concerned with how many errors can be detected or corrected. It is the minimum distance between any two codewords of a given code which tells how many errors the code is able to detect and how many it can correct. It can be shown that to detect *s* errors in a codeword then $d(C) \geq s + 1$, or a code can detect one less error than the minimum distance between any two of its codewords. Similarly, a code can correct errors in up to *t* positions if $d(C) \geq 2t + 1$. In the example {000,111} there is $d(C) = 3$ thus this code can detect $s \leq 2$ errors or correct exactly $t = 1$ errors. Of course, with only two codewords there is a significant limit on the diversity of data that can be transmitted using this code.

One of the most basic types of decoding is based solely on the minimum distance between a received vector and one of the codewords in the code. This is known as *nearest neighbor decoding*. If vector V is received, the minimum distance is calculated relative to each codeword, $d(V, C_1), d(V, C_2)$, etc. The codeword having the least distance from the received vector is determined to be the intended codeword. Thus, V is decoded as $C_h$. If the codeword was received without error, of course there would be $d(V, C_h) = 0$. Back to the example of {000,111}: if 010 was received, then $d(010,000) = 1$ and $d(010,111) = 2$, thus the received vector is closer to 000 than to 111 ($1 < 2$), and the vector is decoded as 000. Note that in this example if there are more than two positions in error in the received codeword that error cannot be reliably corrected.

Equivalent codes have different codewords, but share common properties and structures. Strictly speaking, equivalent codes are those which can be derived from one other by either swapping the positions of codewords, or by swapping the position of symbols in the codeword.

Another important function is the weight of given codeword. This is the number of non-zero elements in the codeword, or the minimum distance between a codeword and the 0 vector of length *n*. In {000,111}, codeword 000 has weight 0, 111 has weight 3. This is typically denoted $w(000) = 0$ and $w(111) = 3$.

Codewords can be compared and combined using sums and intersections. To calculate the sum of two vectors simply add the values in corresponding positions.

For example: $1100 + 1010 = (1 + 1, 1 + 0, 0 + 1, 0 + 0) = 0110$

To calculate the intersection of two vectors multiply corresponding positions.

For example: $1100 \cap 1010 = (1 \times 1, 1 \times 0, 0 \times 1, 0 \times 0) = 1000$

Note that addition and multiplication are calculated modulo q which in binary examples is always $q = 2$. Further analysis of the concept of modular arithmetic is left to a future section.

### *Code Capacity and Efficiency*

Much in coding theory discusses the optimization of codes for specific purposes. If a certain number of errors need to be reliably detected or corrected then coding theory seeks to find the smallest code that can fulfill this task. Smaller codes and smaller codewords result in more efficient encoding, transmission and decoding. To better understand code efficiency some familiarity with binomial coefficients is required. Binomial coefficients are written in the form $\binom{n}{m}$ and are read aloud as '*n choose m*'. Mathematically $\binom{n}{m} = \frac{n!}{m!(n-m)!}$ with $n! = n(n-1)(n-2)\ldots 321$ and $0! = 1$.

For example:

$$\binom{5}{3} = \frac{5!}{3!(2)!} = \frac{5 \times 4 \times 3 \times 2 \times 1}{(3 \times 2 \times 1)(2 \times 1)} = \frac{120}{12} = 10$$

The binomial coefficient represents the number of *unordered* selections of distinct objects (*m*) from a set of unique objects (*n*). In the previous example, the coefficient $\binom{5}{3}$ means that from a set of 5 possible distinct objects it is possible to make 10 *unordered* selections of 3 objects. If the 5 distinct objects are 1, 2, 3, 4 and 5 then the possible *unordered* selections are:

$$(1,2), (1,3), (1,4), (1,5), (2,3), (2,4), (2,5), (3,4), (3,5), (4,5).$$

This concept has immediate applications to coding theory in determining the number of possible codes meeting specific criteria. For example, to determine the number of binary codes with 4 codewords where each codeword is 6 positions ($M = 4, n = 6$) calculate $\binom{24}{6} = \binom{16}{6} = 8008$. Binomial coefficients are also useful for determining the number of codewords of a given weight in a specific alphabet. For example, to find the number of possible binary codewords of length 5 and also of weight 3 ($n = 5, w = 3$) calculate $\binom{5}{3} = 10$. There is an obvious one-to-one correspondence between these vectors and the *unordered* selections listed in the previous example above.

The term *sphere* is used to describe the vectors of minimum distance $\leq r$ where *r* is the radius of the sphere. Spheres are denoted as $S(\boldsymbol{x}, r)$. The example sphere: $S(101,1) = \{101, 001, 111, 100\}$. Since this set is by definition the vectors of minimum distance *r* from $\boldsymbol{x}$, is follows that if any of these vectors are received they will be decoded as $\boldsymbol{x}$ using nearest neighbor decoding. It also follows that any two spheres of radius r centered on different codeword cannot have any vectors in common. There are $\binom{n}{r}$ possible ways of choosing the *r* positions where each position differs in $(q-1)$ ways. In the next position they can differ in $\binom{n}{1}(q-1)$ possible ways and so forth. Eventually it is seen that any sphere contains exactly $\binom{n}{0} + \binom{n}{1}(q-1) + \binom{n}{2}(q-1)^2 + \cdots + \binom{n}{r}(q-1)^r$ vectors.

From this definition of binomial coefficients and spheres a *sphere-packing* or *Hamming bound* can be derived which shows the maximum number of possible codewords in a code with certain parameters. Take a *q*-ary (*n, M,* 2*t*+1) code, remember that any two spheres or radius *t* are unique and the total number of possible codewords in a given alphabet are $q^n$. Then it is easily seen that

$M\{\binom{n}{0} + \binom{n}{1}(q-1) + \binom{n}{2}(q-1)^2 + \cdots + \binom{n}{t}(q-1)^t\} \leq q^n$. With $q = 2$, the equation for a binary (*n, M,* 2*t*+1) code is $M\{\binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{t}\} \leq 2^n$.

Note that while using these equations it is possible to determine the theoretical maximum number of codewords, this does not imply that such a code actually exists. In fact, a code which exactly matches the sphere-packing bound is known as a *perfect code*. In such a code, all $q^n$ possible vectors are included in a sphere. Each is a distance $\leq t$ from exactly one codeword. As will be shown very few perfect codes are known to exist up to equivalence.

## *Block Design*

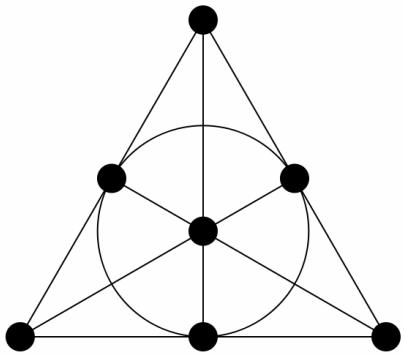
*Balanced block designs* are structures that can be used to describe the relationships of elements in a set. The $u$ individual elements in such a design are called points. The set is organized into subsets called blocks, $b$. A block contains precisely $k$ elements, and each element is in $r$ blocks. Every pair of elements is contained in $\lambda$ blocks. With each of these properties defined, it is possible to describe the block design as $(b, u, r, k, \lambda) - design$.

An example that is frequently used is the $(7, 7, 3, 3, 1) - design$. The elements of this design are $\{1, 2, 3, 4, 5, 6, 7\}$ divided into blocks $\{1, 2, 4\}, \{2, 3, 5\}, \{3, 4, 6\}, \{4, 5, 7\}, \{5, 6, 1\}, \{6, 7, 2\}, \{7, 1, 3\}$.

By listing each block as a column an incidence matrix can be used to illustrate which elements occur in which blocks:

$$\begin{array}{c} Incidence\ Matrix \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{bmatrix} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

Geometrically this is known as the projective plane of order 2 or the Fano plane:



Numerous applications of coding theory use balanced block designs, for example the Solomon-Reed codes used in the data storage and transmission of digital data.

## *Fields and Vector Spaces*

Fields are algebraic structures that define the operations of addition and multiplication (and by proxy, subtraction and division). While these are commonly considered as standard mathematical operations, the elements of a field need not be numbers. Although fields can be infinite or finite, for the purposes of coding theory the latter is more appropriate.

Specifically, fields have the following characteristics by definition. If a$\in F$ *and* $b \in F$ *then*:

i. $a + b \in F \; and \; a \times b \in F$ (Closed under addition and multiplication)
ii. $a + b = b + a \; and \; a \times b = b \times a \; (Commutative \; Law)$
iii. $(a + b) + c = a + (b + c) and \; a \times (b \times c) = (a \times b) \times c \; (Associative \; Law)$
iv. $a \times (b + c) = a \times b + a \times c \; (Distributive \; Law)$

Additive and multiplicative inverses exist (except for 0 where no multiplicative inverse exists) such that for all $a \in F$:

v. $a + 0 = a$
vi. $a \times 1 = a$
vii. $a + (-a) = 0$
viii. $a \times a^{-1} = 1, a \neq 0$

Some important fields in mathematics include the set of all complex numbers under the usual addition and multiplication and the set of all real numbers under the usual additional and multiplication. Fields are also sometimes referred to as division rings or commutative division rings. A ring is an algebraic structure in which there is no requirement for a multiplicative inverse (viii above need not be true).

Earlier it was mentioned that only finite fields would be discussed in relation to linear codes. The exact number of elements in a field is known as its *order*. An important theorem in the study of finite fields was proven by Evariste Galois. He demonstrated that for a field of order $q$ to exist, $q$ must be a prime power. That is to say $q=p^n$ with p as a prime number and n a positive integer. Per this theorem a field of order q is said to be a Galois field and is expressed using the notation GF($q$).

Important to the study of coding theory, a finite field GF($q$) is the set of integers $\{1,2,3 \ldots q-1\}$ with addition and multiplication modulo $p$. For a brief review of modular arithmetic it is necessary only to look at what it means to be congruent modulo $p$. Two integers, $a$ and $b$, are said to be congruent modulo $q$ if $a - b$ is divisible by $q$. This would be notated $a \equiv b (mod \; q)$. For example, under addition modulo 7 the following uncanny result appears: $5 + 6 = 4$. Another way of notating this would be $11 \equiv 4 (mod \; 7)$. Working with addition modulo 2 it is clear that $1 + 1 = 0$ or $2 \equiv 0 (mod 2)$.

The elements in GF($q$) form subsets of ordered $n$-tuples known as vector spaces and denoted $V(n, q)$.

## Linear Codes

### *Definition and Basic Properties*

Of particular importance in the study of coding theory is a group of codes known as *linear codes*. Specifically, a linear code is a vector subspace of a Galois field GF($q$). From the earlier definition of a vector subspace, a subset $C$ of GF($q$) is a linear code if and only if:

ix. $\boldsymbol{u} + \boldsymbol{v} \in C \; for \; all \; \boldsymbol{u} \; and \; \boldsymbol{v} \; in \; C$ (closed under vector addition)

x. $a\boldsymbol{u} \in C \; for \; all \; \boldsymbol{u} \in C \; and \; all \; a \in GF(q)$ (closed under scalar multiplication)

For binary codes requirement ii above is self evident (since scalar multiplication can only be by 0 or 1), meaning that the only real requirement is closure under addition.

The dimension $k$ of this vector subspace is the number of codewords, and the linear code is denoted as a $q$-ary [$n$, $q^k$]-*code*. If a minimum distance is specified this is a [$n$, $q^k$, $d$]-*code*. Note the use of square brackets to denote the linear aspect of the code.

An interesting feature of linear codes is the correlation between the weight of the codewords and the minimum distance of the code itself. In fact it can be shown that the weight of the smallest, non-trivial codeword is the minimum distance of the code itself. To prove this, note first that the minimum distance of any two binary codewords is equal to the weight of their sum. This is obvious since the resultant codeword can only have a 1 in places where the original codewords had different values. Symbolically: $d(\boldsymbol{x}, \boldsymbol{y}) = w(\boldsymbol{x} + \boldsymbol{y})$. Let $w(C)$ be the smallest weight of any codeword in the code, then it follows that the minimum distance of the code and the combination of any two codewords will be equal to or greater than this minimum weight: $d(\boldsymbol{x}, \boldsymbol{y}) = w(\boldsymbol{x} + \boldsymbol{y}) \geq w(C)$. Also, by the definition of $w(C)$ at least one codeword has this minimal weight and that the distance from this codeword to the $\mathbf{0}$ vector (included in every linear code) must be at least the minimum distance of the entire code. In other words, let $\boldsymbol{z}$ be the codeword with the smallest weight, $w(C)$, then $w(C) = d(\boldsymbol{z}, \mathbf{0}) \geq d(C)$. Using these equations see that $w(C) \leq d(C)$ and $w(C) \geq d(C)$ which can only mean $w(C) = d(C)$. Thus it is proven that the smallest weight of any codeword of a linear code is equal to the minimum distance of the code itself.

### *Generator Matrices*

Another special feature of linear codes is that they can be entirely defined by listing only the basis vectors of the code's subspace, as opposed to listing every single codeword. This generator matrix can be obtained from the matrix of the entire code by the use of row reduction. A binary [$n$, $2^k$]-*code* will reduce to a generator matrix of dimensions $k \times n$. The entire code can then be "re-generated" by expressing all linear combinations of the rows of the generator matrix. Also, any generator matrix of the form $[I_k|A]$, with $I_k$ being the identity matrix of dimension k, is said to be in standard-form. For example:

The [4, 8]-code $\begin{bmatrix}0000\\1100\\1010\\1001\\0110\\0101\\0011\\1111\end{bmatrix}$ has $2^k = 8 \rightarrow k = 3$ and reduces to the generator matrix $\begin{bmatrix}1001\\0101\\0011\end{bmatrix}$

The code can then be "re-generated" by creating all linear combinations of the rows of the generator matrix:

$$[xyz] \times \begin{bmatrix}1001\\0101\\0011\end{bmatrix} = x[1001] + y[0101] + z[0011]$$

As an example, letting $x = 1, y = 1$ and $z = 0$ generates the second codeword in the original codeword list as follows:

$$[1001] + [0101] = [1 + 0, \quad 0 + 1, \quad 0 + 0, \quad 1 + 1] = [1100]$$

Repeating this process for all possible combinations of values 0 and 1 for *x*, *y* and *z* will eventually result in the generation of the entire linear code. The discussion will return to the subject of encoding after a brief discussion of equivalence.

The use of the generator matrix to represent the code brings up the notion of the equivalence of linear codes. Equivalent codes share the same minimum distance, codeword length and number of codewords, but contain at least some different codewords. However, these characteristics are not sufficient to show that two codes are equivalent. By definition, two linear codes are said to be equivalent if it is possible to obtain the generator matrix for one through the following permutations:

i. Swapping the order of rows
ii. Multiplying any row by a non-zero scalar (in the binary case this is generally discarded)
iii. Addition of a scalar from one row to another
iv. Swapping the order of columns
v. Multiplying any column by a non-zero scalar (in the binary case this is generally discarded)

If only the first three operations are used (the row operations), then the resultant matrix is simply another basis of the same subspace and it will generate the exact same code. If the last two operations (the column operations) are used then the resultant matrix will generate an equivalent, but potentially different, code.

Take for example the following binary [7, 4]-code, in random order, which can be reduced to its generator matrix and further permutated into standard form:

$$\begin{bmatrix}1101001\\1100110\\1011010\\1010101\\0111100\\0110011\\0001111\\0000000\end{bmatrix} \begin{matrix}Replace \rightarrow with\\ r_2 \rightarrow r_1 + r_2\\ r_4 \rightarrow (r_5 + r_6) - (r_1 + r_2)\\ r_6 \rightarrow (r_3 + r_4) - (r_1 + r_2)\\ r_7 \rightarrow r_7 - (r_1 + r_2)\end{matrix} \begin{bmatrix}1101001\\0001111\\1011010\\\cancel{0000000}\\0111100\\\cancel{0000000}\\\cancel{0000000}\\\cancel{0000000}\end{bmatrix}$$

After one more permutation the generator matrix (basis) for the code (vector subspace) remains:

$$\begin{bmatrix}1101001\\0001111\\1011010\\0111100\end{bmatrix} \begin{matrix}Replace \rightarrow with\\ r_3 \rightarrow r_1 + r_2 + r_3\end{matrix} \begin{bmatrix}1101001\\0001111\\\cancel{0000000}\\0111100\end{bmatrix}$$

Using column operations the resultant matrix can be put into standard form. This standard form will produce an equivalent, but potentially different, code from the original:

$$\begin{bmatrix}1101001\\0001111\\0111100\end{bmatrix} \begin{matrix}Swap\\ c_2 \leftrightarrow c_6\end{matrix} \begin{bmatrix}1001011\\0101101\\0011110\end{bmatrix} is\ standard\ form \begin{bmatrix}I_3\\1011\\1101\\1110\end{bmatrix} where\ A = \begin{bmatrix}1011\\1101\\1110\end{bmatrix}$$

The resultant generator matrices produce two different, but equivalent, [7, 4, 4]-codes:

$$\begin{bmatrix}1101001\\0001111\\0111100\end{bmatrix} \begin{matrix}generates\\ \rightarrow\end{matrix} \begin{bmatrix}1101001\\1100110\\1011010\\1010101\\0111100\\0110011\\0001111\\0000000\end{bmatrix} while \begin{bmatrix}1001011\\0101101\\0011110\end{bmatrix} \begin{matrix}generates\\ \rightarrow\end{matrix} \begin{bmatrix}1111000\\1100110\\1010101\\0110011\\1001011\\0101101\\0011110\\0000000\end{bmatrix}$$

Codewords themselves are encoded messages. Part of the codeword is the message digits, and the additional portion of the codeword is redundancy to allow for error detection and correction. Earlier examples demonstrated the encoding process which will now be further discussed.

Let $v_1v_2v_3 \ldots v_k$ be the message vectors to be encoded. Multiplying these with the generator matrix $G$ on the right produces the codeword. The rows of the generator matrix G are represented as $r_1r_2r_3 \ldots r_k$. Symbolically: $vG = \sum_{i=1}^{k} v_i r_i$

Encoding message vector $v = 111$ using the example generator matrix from the earlier example:

$$[111] \times \begin{bmatrix}1101001\\0001111\\0111100\end{bmatrix} = [1011010]$$

If the generator matrix is in standard form the resulting codeword can be further analyzed. Encoding message vector $v = 111$ using the example standard form generator matrix from the earlier example:

$$[111] \times \begin{bmatrix} 1001011 \\ 0101101 \\ 0011110 \end{bmatrix} = [1111000]$$

Because this codeword is generated from a matrix in standard form, the first $k^{th}$ digits of the codeword will match the message vector. The remainder of the codeword is redundancy bits. Dissecting the codeword generated in the previous example shows:

$$1111000 = \left[\begin{matrix} 111 \\ Message\ vector \end{matrix} \middle| \begin{matrix} 1000 \\ Redundancy\ bits \end{matrix}\right]$$

### *Coset Array Decoding*

Decoding the received vector can be done in several ways. One basic way of decoding is done using cosets and *coset arrays.* A *coset* is a set of vectors that results when a specific vector is added to each codeword. Continuing with the standard-form-generated [7, 4, 4]-code in the previous example:

| *coset* → | | | | | |
|---|---|---|---|---|---|
| | 0000000 | 1111000 | 1100110 | ··· | 0000000 |
| | 0000001 | 1111001 | 1100111 | ··· | 0000001 |
| | 0000010 | 1111010 | 1100100 | ··· | 0000010 |
| | 0000100 | 1111100 | 1100010 | ··· | 0000100 |
| | 0001000 | 1110000 | 1101110 | ··· | 0001000 |
| | 0010000 | 1101000 | 1110110 | ··· | 0010000 |
| | 0100000 | 1011000 | 1000110 | ··· | 0100000 |
| | 1000000 | 0111000 | 0100110 | ··· | 1000000 |
| | ↑ | | | | |
| | *coset leaders* | | | | |

Suppose a vector is received as $1100100$ it is located in the array. This vector is located in the third row, second column above. The codeword is then decoded by subtracting the coset leader of that row from the received vector, thus $1100100$ is decoded as $1100100 - 0000010 = 1100110$.

The following is a ternary example:

$$\begin{bmatrix} 1022 \\ 0121 \end{bmatrix} \begin{matrix} generates \\ \rightarrow \end{matrix} \begin{bmatrix} 1022 \\ 0121 \\ 1110 \\ 2011 \\ 0212 \\ 2220 \\ 2102 \\ 1201 \\ 0000 \end{bmatrix} thus$$

| *coset* → | | | | | |
|---|---|---|---|---|---|
| 0000 | 1022 | 0121 | 1110 | ··· | 0000 |
| 0001 | 1020 | 0122 | 1111 | ··· | 0001 |
| 0010 | 1002 | 0101 | 1120 | ··· | 0010 |
| 0100 | 1122 | 0221 | 1210 | ··· | 0100 |
| 1000 | 2022 | 1121 | 2110 | ··· | 1000 |
| 0002 | 1021 | 0120 | 1112 | ··· | 0002 |
| 0020 | 1012 | 0111 | 1100 | ··· | 0020 |
| 0200 | 1222 | 0021 | 1010 | ··· | 0200 |
| 2000 | 0022 | 2121 | 0110 | ... | 2000 |
| ↑ | | | | | |
| *coset leaders* | | | | | |

If a vector $1121$ is received, it is located in the fifth row and coset leader 1000 is subtracted to decode as follows: $1121 - 1000 = 0121$. Another example: $1010 - 0200 = 1110$.

### *Dual-codes and Parity Check Matrices*

For any linear code, the set of all vectors orthogonal to every codeword is known as the *dual-code* and denoted $C^{\perp}$. For example $C = \begin{bmatrix}000\\110\\011\\101\end{bmatrix}$ $then\ C^{\perp} = \begin{bmatrix}000\\111\end{bmatrix}$. The generator matrix for $C^{\perp}$ is a parity check matrix for $C$ itself. The *parity-check* matrix $H$ can be obtained from $G$, the generator matrix of $C$, as follows:

$$G = \left[ I_k \middle| \begin{matrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n-k} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n-k} \\ \vdots & \vdots & \vdots & \vdots \\ a_{k,1} & a_{k,2} & \cdots & a_{k,n-k} \end{matrix} \right]$$

$then$

$$H = \left[ \begin{matrix} -a_{1,1} & -a_{2,1} & \cdots & -a_{k,1} \\ -a_{1,2} & -a_{2,2} & \cdots & -a_{k,2} \\ \vdots & \vdots & \vdots & \vdots \\ -a_{1,n-k} & -a_{2,n-k} & \cdots & -a_{k,n-k} \end{matrix} \middle| I_{n-k} \right]$$

In the binary case the negative signs have no effect since $-1 = 1$. Using the standard form matrix from the previous example:

$$G = \begin{bmatrix}1001011\\0101101\\0011110\end{bmatrix}, then\ A = \begin{bmatrix}1011\\1101\\1110\end{bmatrix}$$

$thus$

$$H = [-A^T | I_4] = \begin{bmatrix}1111000\\0110100\\1010010\\1100001\end{bmatrix}$$

Using the parity-check matrix it is possible to describe the entire code *C* as follows:

$$for\ a\ vector\ x\ of\ length\ n, \{x \in C | xH^T = 0\}$$

### *Syndrome Decoding*

The parity-check matrix is especially useful for generating the *syndrome* of a given coset. The syndrome a vector $x$ length $n$ is defined as $S(x) = xH^T$. Any two vectors in the same coset will have the same syndrome. The proof for this is straightforward. Let $x$ and $y$ be vectors in a given coset such that $x + C = y + C$ (from the definition of coset). It follows that $x - y \in C$. From the definition of the parity check matrix $(x - y)H^T = 0$ which means that $xH^T = yH^T$. Thus from the definition of syndrome this proof has shown that $S(x) = S(y)$.

Using this concept of a syndrome the coset array can be augmented using the parity check matrix:

$$H^T = \begin{bmatrix} 1011 \\ 1101 \\ 1110 \\ 1000 \\ 0100 \\ 0010 \\ 0001 \end{bmatrix}, then$$

| $coset \rightarrow$ | | | | | |
|---|---|---|---|---|---|
| 0000000 | 1111000 | 1100110 | ··· | 0000000 | 0000 |
| 0000001 | 1111001 | 1100111 | ··· | 0000001 | 0001 |
| 0000010 | 1111010 | 1100100 | ··· | 0000010 | 0010 |
| 0000100 | 1111100 | 1100010 | ··· | 0000100 | 0100 |
| 0001000 | 1110000 | 1101110 | ··· | 0001000 | 1000 |
| 0010000 | 1101000 | 1110110 | ··· | 0010000 | 1110 |
| 0100000 | 1011000 | 1000110 | ··· | 0100000 | 1101 |
| 1000000 | 0111000 | 0100110 | ··· | 1000000 | 1011 |
| ↑ *coset leaders* | | | | | ↑ *syndrome* |

It then becomes possible to represent the decoding scheme using only two columns which greatly speeds up the decoding process. Instead of searching a possibly large coset array for exact position of every codeword received, it is now possible to decode by simply calculating the syndrome of the received vector and then subtract from it the corresponding coset leader. The standard coset array is truncated into a syndrome lookup table with only two columns:

| *Coset leader* | *Syndrome* |
|---|---|
| 0000000 | 0000 |
| 0000001 | 0001 |
| 0000010 | 0010 |
| 0000100 | 0100 |
| 0001000 | 1000 |
| 0010000 | 1110 |
| 0100000 | 1101 |
| 1000000 | 1011 |

Thus to decode 1011011:

$$S(1011011) = [1011011] \times \begin{bmatrix} 1011 \\ 1101 \\ 1110 \\ 1000 \\ 0100 \\ 0010 \\ 0001 \end{bmatrix} = 1011 + 1110 + 1000 + 0010 + 0001 = 1110$$

Using only the syndrome and coset leader columns this vector is decoded as $1011011 - 0010000 = 1001011$.

One key principle of the parity-check matrix is what it reveals about the minimum distance of the code. If a linear code has minimum distance $d$ then the parity-check matrix of this code has precisely $d-1$ linearly independent columns, but there are at least $d$ dependant columns. The proof for this is based on earlier theorems discussed. Based on the definition of the parity-check matrix if a vector $\boldsymbol{x} \in C$ then $\boldsymbol{xH^T = 0}$. This implies $x_1\boldsymbol{H_1} + x_2\boldsymbol{H_2} + x_3\boldsymbol{H_3} + \cdots + x_n\boldsymbol{H_n} = \boldsymbol{0}$ with $\boldsymbol{H}$ representing the column vectors of the parity check matrix. Therefore for any vector $\boldsymbol{x}$ of weight $d$ there exists at least $d$ linealy dependent columns of $H$. If there were only $d-1$ linearly dependent columns it would imply that there exists a codeword of weight less than $d$ such that $\boldsymbol{xH^T = 0}$. It was proven earlier however that the weight of the smallest codeword of a linear code is in fact its minimum distance. Thus, the parity check-matrix must have at least $d$ linearly dependent columns and has exactly $d-1$ linearly independent columns.

### *Hamming Codes*

Hamming codes are linear codes with parity-check matrices of dimension $r \times 2^r - 1$ such that the columns of $H$ are exactly the unique, non-zero vectors of length $r$. Hamming codes are denoted as $\boldsymbol{Ham(r,q)}$ with $q$ prime. Binary Hamming codes are denoted similarly as $\boldsymbol{Ham(r,2)}$. The length of the codewords in a binary Hamming code will be $n = 2^r - 1$. Listing the columns of the parity-check in a different order results in a simple permutation of the rows and columns of the generator matrix which will generate an equivalent Hamming code (as shown in an earlier example). Hamming codes always achieve their sphere-packing bounds making every Hamming code also a perfect code.

From this definition it is easy to construct the parity-check matrix representing $\boldsymbol{Ham(3,2)}$ by simply listing every possible column vector of length 3:

$$H = \begin{bmatrix} 1011100 \\ 1101010 \\ 0111001 \end{bmatrix}, thus\ G = \begin{bmatrix} 1000110 \\ 0100011 \\ 0010101 \\ 0001111 \end{bmatrix}\ and\ n = 2^3 - 1 = 7$$

Another example is a $Ham(4,2)$ code:

$$H = \begin{bmatrix} 100010011011111 \\ 010011000111011 \\ 001001110010111 \\ 000100101101101 \end{bmatrix}, thus\ G = \begin{bmatrix} \ldots & I_{11} & \ldots \end{bmatrix} \begin{bmatrix} 1100 \\ 0110 \\ 0011 \\ 1010 \\ 1001 \\ 0101 \\ 1110 \\ 1101 \\ 1011 \\ 0111 \\ 1111 \end{bmatrix}\ and\ n = 2^4 - 1 = 15$$

In this previous example only the four columns with exactly one entry of 1 are linearly independent, which implies that $d = 3$ and this code can detect up to one error in each codeword.

The same basic principles apply to $q$-ary Hamming codes such as the $Ham(2,5)$ code:

$$H = \begin{bmatrix} 04310 \\ 32001 \end{bmatrix}, thus\ G = \left[ I_3 \middle| \begin{matrix} 02 \\ 13 \\ 20 \end{matrix} \right] \ and\ n = 5$$

Binary Hamming codes become easy to manipulate using only their parity check matrix if the columns are listed in order of increasing binary value. Rearranging the columns of the parity-check matrix for the $Ham(4,2)$ code in the previous example in increasing binary order yields:

$$H = \begin{bmatrix} 0000000011111111 \\ 0000111100001111 \\ 0110011001100110 \\ 1010101010101010 \end{bmatrix} \ arranged\ from\ 1\ (0001)\ through\ 15\ (1111)$$

Since each of the column vectors are also the syndromes of $C$, it is possible to determine the digit in error simply by calculating the syndrome of a received vector. For example, the received vector $X$:

$$x = 100000000001110 \ \rightarrow 100000000001110 \times H^T = 0001 + 1100 + 1101 + 1110 = 1110$$

Since $1110$ is the 14th vector of $H$ this implies that an error has occurred in the 14th digit and the vector is decoded as the codeword $100000000001100$.

Another example using the same parity-check matrix:

$$x = 11100000000000 \rightarrow S(x) = 0001 + 0010 + 0011 = 0000 \xrightarrow{decode\ as} x\ is\ a\ codeword$$

### *Extended Hamming Codes*

Binary Hamming codes can be extended by adding a row and column and are denoted $\hat{H}am(r, 2)$. This modification increases the minimum distance from 3 to 4 which slows down the transmission of the codewords, but also enables the code to simultaneously detect two errors while it corrects one. For a binary Hamming code with parity-check matrix , the parity-check $\hat{H}$ of the extended code is:

$$\hat{H} = \left[ \begin{array}{ccc|c} & H & & \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \\ \hline 1 & \cdots & 1 & 1 \end{array} \right]$$

Taking the earlier example of a $Ham(3,2)$ code, extending it and putting it in increasing binary order yields:

$$H = \begin{bmatrix} 1011100 \\ 1101010 \\ 0111001 \end{bmatrix}, \text{thus } \hat{H} = \begin{bmatrix} 10111000 \\ 11010100 \\ 01110010 \\ 11111111 \end{bmatrix}$$

Decoding with an extended code uses a process called incomplete decoding. When a vector x is received, its syndrome is calculated as normal: $S(x) = x\hat{H}^T = s_1 s_2 s_3 \dots s_n$. The the following scenarios describe how to decode the received vector:

i. If $s_1 s_2 s_3 \dots s_{n-1} = 0$ *and* $s_n = 0$, then there were no errors and $x$ is a codeword
ii. If $s_1 s_2 s_3 \dots s_{n-1} \neq 0$ *and* $s_n = 0$, then at least two errors have occurred and the vector will need to be retransmitted
iii. If $s_1 s_2 s_3 \dots s_{n-1} = 0$ *and* $s_n = 1$, then a single error has occurred in the last digit
iv. If $s_1 s_2 s_3 \dots s_{n-1} \neq 0$ *and* $s_n = 1$, then a single error has occurred in the digit represented by the binary number $s_1 s_2 s_3 \dots s_{n-1}$

Note that although this extended binary Hamming code can detect two errors in a single codeword it is unable to correct both errors and must seek retransmission of the vector.

The following examples decode sample vectors $w$, $x$, $y$, and $z$ using the extended binary Hamming code in the previous example:

$$w = 0111000 \rightarrow S(w) = 0101 + 0111 + 1001 = 1011 \xrightarrow{decode\ as} 0111100$$
$$x = 11100000 \rightarrow S(x) = 0011 + 0101 + 0111 = 0001 \xrightarrow{decode\ as} 11100001$$
$$y = 11000000 \rightarrow S(y) = 0011 + 0101 = 0110 \xrightarrow{decode\ as} 2\ errors, retransmit$$
$$z = 00110011 \rightarrow S(z) = 0111 + 1001 + 1111 + 0001 = 0000 \xrightarrow{decode\ as} z\ is\ a\ codeword$$

### *Perfect Codes*

It has been mentioned that the Hamming codes $Ham(r, q)$ are perfect codes for any prime number $q$. In fact, the only non-trivial perfect codes besides the Hamming codes are two codes known as the Golay codes. They were discovered by Marcel Golay in 1949. The first perfect Golay code is a binary [23, 12, 7]-code. Using the definition of a perfect code described earlier it is easy to see: $2^{12}\left\{1 + 23 + \binom{23}{2} + \binom{23}{3}\right\} = 2^{23}$. The other perfect Golay code is a ternary [11, 6, 5]-code. Perfect codes are of interest because they are the most efficient codes for their corresponding codeword length and minimum distance. For this reason they are commonly used in the transmission of digital data.

## Applications and Examples

There are many applications of coding theory in the modern world. In computer science, where coding theory originated, powerful error detection and correction codes are used in the transmission of digital data.

### *Hamming codes and DRAM*

Traditional DRAM (dynamic random access memory) uses Hamming codes for error correction purposes. However, Hamming codes have a minimum distance $d = 3$ which enables them to correct only one bad bit per codeword. As computers have progressed from 8-bit machines to 16-bits, 32-bits or

even 64-bits, the ability to correct only a single bit error introduces the increasing possibility of data corruption. In the presence of ever increasing data throughput even extended Hamming codes seem to fall short of the required error correction.

### *Reed-Solomon Codes*

A more powerful family of codes is the Reed-Solomon codes which are used to protect data from defects in the storage or transmission media. The Reed-Solomon code is crucial to the success of the audio CD, allowing CD players to interpolate data lost due to scratches or dust on the disks. The code allows the CD player to correct over 2mm of damaged or missing disk surface. This means that almost any "skips" the user hears when listening to a CD are almost certainly related to tracking errors by the laser itself. Although CDs were the first commercially marketed use of error correcting codes, they are most certainly not the last. Cellular telephones employ these codes to overcome interference from high-strength radio transmitters. Digital televisions, high-speed DSL modems, the Space Communications Protocol and common bar code scanners also rely on these powerful codes. Protocols used in computer networks almost universally apply redundancy to data transmissions to protect against errors. One of the most common networking protocols used in Windows networks is Transmission Control Protocol or TCP.

Reed-Solomon codes are denoted as $RS(n,k)$ with $s$ symbol bits. Each codeword consists of $k$ data symbols of $s$ bits each for a total codeword length of $n$. These codes can correct up to $t$ incorrect symbols in each codeword with $2t = n - k$. A very common Reed-Solomon code in digital applications is the $RS(255,223)$ code with 8-bit symbols. Since $2t = 255 - 223 = 32 \rightarrow t = 16$ this implies that this code can correct up to 16 symbol errors per codeword. This is a significant improvement over the single-error correcting Hamming codes.

The following diagram illustrates a sample Reed-Solomon codeword:

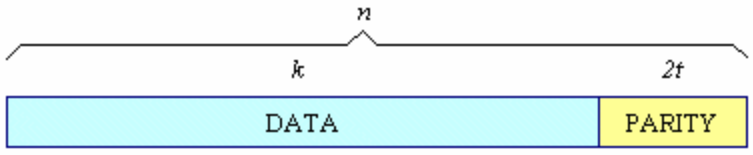



**Figure 1: Reed-Solomon Codeword diagram (Harries, 2007)**

### *Coding Theory and Genetic Research*

There are many new frontiers of science that coding theory is finding applications in. One such application which arouses great excitement is the use of coding theory in the study of evolution and genetic mutations. In their short paper, *A Coding Theory Framework for Genetic Sequence* Analysis, M.A Vouk, D.L. Bitzer and D.I. Rosnick summarize how mathematical code models are used to interpret and predict various processes in modern genetics and evolution.

Genetic information in the form of DNA is taken as input, transmitted via the process of replication and then ultimately output as amino acid proteins. Errors are introduced by fluxuations in heat, radioactivity and other factors. One scientist referenced in the paper, Eigen, suggests that some type of

error-correcting code must be employed in this process to ensure the survival of the species. DNA molecules contain "complementary interactions" which serve as the alphabet of the code. These DNA sequences are mapped to a space of discrete points. This space is similar in concept to the idea of a sphere, with *n* length sequences surrounding an *n*-length codeword. The symbol difference between the *n*-length sequences and the *n*-length codewords they surround is the minimum distance of the code. The following figure describes the proposed structure of the encoding/decoding process:

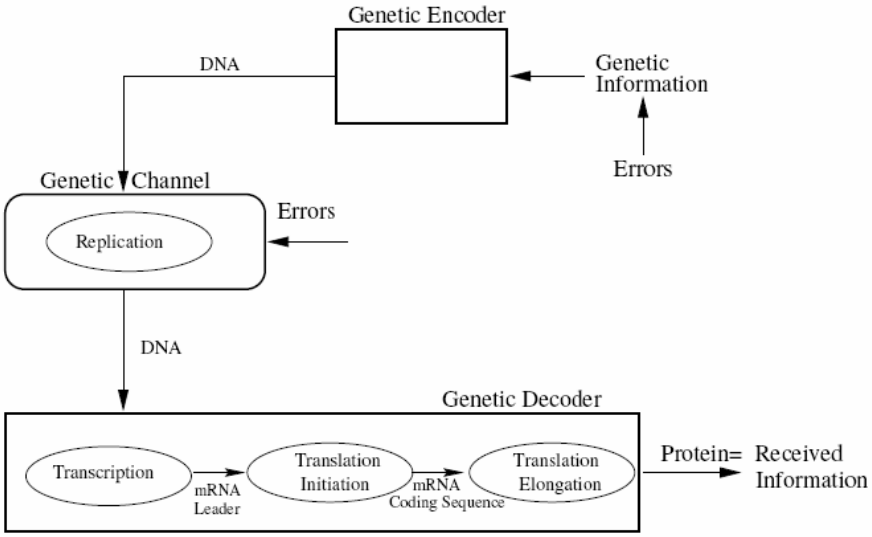



**Figure 2: Gene Sequence Encoding/Decoding (May, 2004)**

A major challenge in defining a coding theory approach to genetics is the ability to encode and manipulate genetic data in DNA. In what is known as DNA computing, a problem or question is encoded into a strand of DNA. Various laboratory experiments are used to manipulate the DNA in an attempt to "solve" the problem. Coding theory played a central role in determining which combinations of genetic sequences could serve as codewords as well as identifying the "reading frames" or sections of the DNA which need to be specifically identified.

One model used in an attempt to validate the above processes was a $(n,k)$ block code which outputted a parity check code. Based on the known genetic bases, codewords of length $n = 5$ and $n = 8$ were developed and evaluated based on a minimum distance (nearest neighbor) decoding scheme. Another model used was a convolutional code model. Convolutional coding allows for immediate past and future information to be used in the encoding/decoding process. Both models were evaluated on several criteria: recognition of certain sections of mRNA sequences, the ability to differentiate between sequences which had been translated and those which had not, and the synchronization of reading frames. While both models did surprising well in laboratory studies, the block based code model seemed to fit the observations closer.

The results of these studies seem to suggest that it is possible to develop more powerful coding theory approaches to genetic analysis. Also, it is considered likely that organisms use some type of error-correcting mechanism in the replication of their genetic materials.

### *Network Communications*

Recent work on two families of codes known as Tornado codes and Luby Transform codes are improving the effectiveness and efficiency of network communications. One of the most common network protocols, TCP, maintains a single connection with each attached computer and transmits data in chunks called packets. Reed-Solomon codes are primarily used encode and decode packets. Receiving computers then send an acknowledgement back to the sender once they have successfully decoded the packet. If no acknowledgement arrives the sender assumes the packet was lost and it is resent. Another method of distributing data is known as multicasting. In this model packets are sent out to all computers simultaneously. This method is cheaper and more efficient than standard TCP, but the sender is quickly overwhelmed when a large number of acknowledgements are received at once. A better method of ensuring reliable data transmission was needed (Yamaguchi, 2000).

Tornado codes and Luby codes are improvements which better fit the needs of enhancing network communication. Luby codes encode packets into metapackets. Metapackets are assigned a weight, and then a probability is associated with that weight. So for a metapacket of length $n$ and of weight $w$, the probability is $\frac{1}{w}(w-1)$. To encode a metapacket $w$ packets of weight $w$ are chosen and added modulo two (as in previous binary examples). The result is a metapacket whose header contains data about which packets are encoded within it.

Receiving machines collect metapackets and extrapolate on the packets they need. By collecting slightly more packets than is necessary the receiver is able to decode the packets with an extremely good success rate (Robinson, 1996).

## Bibliography


Hanley, S. (2002, April 24). *Reed-Solomon Codes and CD Encoding*. Retrieved 2008, from U.S. Navy Website: http://www.usna.edu/Users/math/wdj/reed-sol.htm

Harries, C. (2007). *Error Correction Techniques for Solid State Disk.* Retrieved from Imperial Technologies Website: http://www.imperialtechnology.com/technology_whitepapers_ecc.htm

Hill, R. (1986). *A First Course in Coding Theory.* Clarendon Press.

May, E. e. (2004). An Error-Correcting Code Framework for Genetic Sequence Analysis. *Journal of the Franklin Institute* , Volume 341, Issues 1-2, Pages 89-109.

Pinch, R. (1997). *Coding theory: the first 50 years.* Retrieved from Plus Magazine: Living Mathematics Website: http://plus.maths.org/issue3/codes/index.html

Riley, M., & Richardson, I. (1998). *Reed-Solomon Codes.* Retrieved from 4i2i Communications Ltd Website: http://www.cs.cmu.edu/afs/cs/project/pscico-guyb/realworld/www/reedsolomon/reed_solomon_codes.html

Robinson, S. (1996). Beyond Reed–Solomon: New Codes for Internet Multicasting Drive Silicon Valley Start-up. *SIAM News*.

Yamaguchi, A. A. (2000). Evaluation of multicast error recovery using convolutional codes. *IEEE Explore*, 37-44.


* **JAY GROSSMAN** is a senior at Rivier College pursuing a Bachelor of Arts in Mathematics. He currently lives in Portland, OR and works full time as a software designer.