

Kenyon College

Digital Kenyon: Research, Scholarship, and Creative Exchange

Faculty Publications

Mathematics and Statistics

8-2007

An Introduction to Coding Theory via Hamming Codes: A Computational Science Model

Nuh Aydin

Kenyon College, aydinn@kenyon.edu

Follow this and additional works at: https://digital.kenyon.edu/math_pubs



Part of the [Mathematics Commons](#)

Recommended Citation

Aydin, Nuh, "An Introduction to Coding Theory via Hamming Codes: A Computational Science Model" (2007). *Faculty Publications*. Paper 4.

https://digital.kenyon.edu/math_pubs/4

This Article is brought to you for free and open access by the Mathematics and Statistics at Digital Kenyon: Research, Scholarship, and Creative Exchange. It has been accepted for inclusion in Faculty Publications by an authorized administrator of Digital Kenyon: Research, Scholarship, and Creative Exchange. For more information, please contact noltj@kenyon.edu.

An Introduction to Coding Theory via Hamming Codes

A Computational Science Module

Supported by

National Science Foundation, NSF CCLI Grant (DUE 0618252)

Module Author: Nuh Aydin

Department of Mathematics

Kenyon College

aydinn@kenyon.edu

August 2007

To the Instructor

The only prerequisite for this module is a course on linear algebra. It can be used in a linear algebra course after students learn necessary background. In fact, it would be an excellent project in a linear algebra course. Typically, in a first course on linear algebra, students study vector spaces over the real numbers. For this module, they need to study vector spaces over the binary field. So, that will provide a level of abstraction (but a manageable one). Additionally, it can be used in any computational science course where introducing error-correct codes is appropriate or desired. Finally, another course that can use this module would be an Abstract Algebra course. Once students learn general finite fields, they can define and implement Hamming codes over an arbitrary finite field (of course they will still benefit from studying Hamming codes over the binary field first). Typically, students before taking an abstract algebra course are familiar with the field of integers module p , for a prime p , but they are not familiar with more general finite fields. The software used in this module is Maple version 10 (classical worksheet mode).

Abstract

The theory of error-correcting codes is a relatively recent application of mathematics to information and communication systems. The theory has found a wide range of applications from deep space communication to quality of sound in compact disks. It turns out that a rich set of mathematical ideas and tools can be used to design good codes. The set of mathematical tools used in the field generally comes from algebra (linear and abstract algebra). The purpose of this module is to introduce the basics of the subject to students with an elementary knowledge of linear algebra, via a well-known class of codes known as Hamming codes. Interesting properties and projects related to Hamming codes are introduced.

KEYWORDS: Coding theory, error-correcting codes, linear codes, Hamming codes, perfect codes

Introduction

The Main Problem

Coding theory is concerned with reliability of communication over noisy channels. Error correcting codes are used in a wide range of communication systems from deep space communication, to quality of sound in compact disks and wireless phones.

In computers, data of any kind is stored and processed as binary digits (or bits for short). A bit is a 0 or a 1. Every letter has an ASCII code. For example, the ASCII code of the letter 'A' is 01000001. Typically, data consists of billions of bits. Digital data is transmitted over a channel (which could be a wire, network, space, air etc.), and there is often *noise* in the channel. The noise may distort the messages to be sent. Therefore, what the receiver receives may not be the same as what the sender sends. The goal of coding theory is to improve the reliability of digital communication by devising methods that enable the receiver to decide whether there have been errors during the transmission (*error detection*), and if there are, to possibly recover the original message (*error correction*). Sometimes, error detection could be good enough in practice, if we can easily retransmit the data such as scanning UPC's in retail stores. However, in some application we want to be able to correct errors because retransmission is too expensive or infeasible, such as in deep space communication.

The General Idea

The main method used to recover messages that might be distorted during transmission over a noisy channel is to employ redundancy. We make use of redundancy present in human languages to detect or correct errors. This happens in both oral and written communication. For example, if you read the sentence "There is a miscake in this sentence" you can tell that something is wrong. So we can *detect* an error. Moreover we can even *correct* it. We are achieving two things here: error detection and error correction. What are the principles that we are using to achieve these goals? First, because the string "miscake" is not a valid word in English, we know that there is an error. Here, the redundancy manifests itself in the form of the fact that not every possible string is a valid word in the language. Secondly, the word "miscake" is closest to the valid word "mistake" in the language, so we conclude that it is the most likely word intended. Of course we can also use the context and meaning to detect and/or correct errors but that is an additional feature, not available to computers. When I enter the string "mistaky" to Merriam-Webster online dictionary (<http://www.m-w.com>), no entry can be found for it, however, the computer comes up with a list of suggested words, the first of which is "mistake". So, the computer is telling me that "mistake" is the most likely word to be intended, because it is *closest* to the given string. This is called the *maximum likelihood principle*. As I typed this article on my computer I witnessed many instances of this principle used by my word processor. For instance, when I mistakenly typed "fistr" it automatically corrected it to "first".

There are also other ways redundancy is used in natural languages. As already pointed out, redundancy in context often enables us to detect and correct errors, vagueness, and ambiguities. When humans communicate, redundancy, either explicitly introduced by the speaker/author or built into the language, comes into play to help the audience understand the message better by overcoming such obstacles as noise, accent, hearing difficulties etc. Shetter [9] gives a number of examples in which redundancy is manifest and useful in languages.

Mathematical Use of Redundancy in Digital Communication

As we see, redundancy is present and useful in human languages in a number of different ways. Our next problem is whether computers can use some of the same principles to achieve error detection and/or correction in digital communication. Since computers have more limited capabilities than humans, in particular, they cannot make sense of words, it is the method of employing redundancy that can be used to achieve this goal in computers using the precise language of mathematics.

To illustrate the use of redundancy in a mathematical way in digital communication systems, consider the following example. Suppose we want to communicate with another party in a simple manner: sending messages that represent a “Yes” or a “No”. Let’s agree that a 1 represents “Yes” and a 0 represents “No”. Unfortunately, there is often noise in the communication channel which may distort messages by flipping the binary bit. If we just send the messages as they are, do we have any way of knowing if an error occurred during the transmission? Note that the reason we can do nothing against errors is that all possible strings (that all have length 1 in this simple example) are valid *codewords*. Codewords in digital communication correspond to valid words in a language.

To get an idea of how to employ redundancy in digital communication, we first model our channel then consider and compare several encoding schemes.

The Binary Symmetric Channel

One of the most fundamental models of a communication channel is the *binary symmetric channel* (BSC), pictured below. In this channel, every bit of a transmitted message has the same probability p of being changed to the other bit. So, a 0 will change to a 1 with probability p and will stay unchanged with probability $1-p$. The number $1-p$ is called the reliability of the BSC. We also assume that errors occur randomly and independently (not always a realistic assumption but will work well for many situations). We will always assume that we work with a BSC with a fixed reliability.

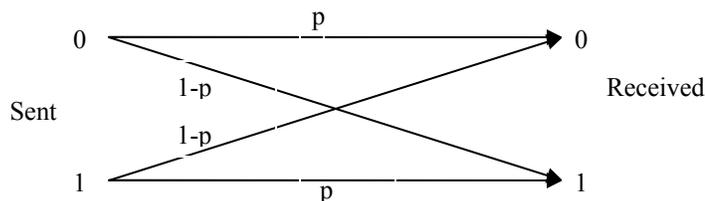


Figure 1: The Binary Symmetric Channel

In block coding theory, original data is broken into blocks of a fixed length. In the examples below, the block size is 4. Then, certain amount of redundancy is added to data.

Scheme 1 Perhaps the most intuitive way of adding redundancy to a message is simply to repeat the original message. For example, instead of sending 1011, we send 1011**1011**. Here 1011 is the original message and 10111011 is the *codeword*. The string obtained after adding redundancy is called a codeword. The set of all codewords is called the code. We shall call this code *simple repetition code*. What does this code buy us? Do we get any error detection or correction capability? If you think about this for a moment, you can see that if there is a single error, then it can be detected. We simply break the received word in half, and compare the two halves. If there is exactly one error, the two halves won't be the same. We also note, however, that we cannot *correct* any errors.

Problem 1 There are some double errors this code cannot detect. Can you come up with an example of such an error? Can you describe which double errors can be detected and which ones cannot be detected?

To quantify what we gain by employing an encoding scheme, let us assume that the probability of error for a single bit for a certain channel is $p = 0.001$, errors occur randomly and independently, and there are about 3000 bits in a page. If we do not employ any encoding scheme, on average we expect to have 3 words (a word is an 8-bit block) in error per page (The expected number of bits in error is $3000 * p = 3$). If we use the simple repetition code on the other hand, there must be at least 2 errors per word in order for an error to go unnoticed. This decreases the expected number of incorrect words to 1 in about 50 pages. Can we do better?

Problem 2 Show the probability calculations to verify the claim that this scheme decreases the expected number of incorrect words to 1 in about 50 pages.

Scheme 2 This scheme repeats everything 3 times. So the original message 1011 is encoded as 1011**10111011**. What are the pros and cons of this scheme? It is easy to see that not only can we detect single or double errors, we can also correct single errors by using the majority opinion, where if at least 2 out of 3 repetitions of a bit agree, then the third one is changed to that majority value. This improvement comes with a cost however: only 1 out of 3 bits sent are information bits (so 2 out of 3 are redundancy bits). We say that the rate (or information rate) of this code is $1/3$. In general, the rate of a code is the ratio of the length of an original message before encoding to the length of a codeword. The rate of the simple repetition code is $1/2$. With this improved error correction capability, the expected number of incorrect words is 1 in about 6115 pages.

Problem 3 Verify the last claim.

Scheme 3 This is a well-known and commonly used encoding scheme that adds a single *parity check* bit at the end so that the number of 1's in the resulting codeword is even. Therefore, the original information 1011 is encoded as the codeword 1011**1**. Another way of describing this method is that the modulo 2 sum of all bits (including the redundancy bit) is 0. It is easy to see that this scheme detects any single errors, but cannot correct any.

Problem 4 Find the rate of this parity check code.

Scheme 4 This is also a well-known example of an error correcting code that was one of the earliest codes designed. It was introduced by Hamming [3]. In this scheme, 3 bits of redundancy are added to information bits. If we denote the original data bits by $x_1x_2x_3x_4$ then the codeword corresponding to the data is $x_1x_2x_3x_4x_5x_6x_7$, obtained by adding three redundancy bits according to the equations $x_5 = x_1 + x_2 + x_4$, $x_6 = x_1 + x_3 + x_4$, and $x_7 = x_2 + x_3 + x_4$ where all computations are done modulo 2. According to this scheme, the information string 1011 is encoded as 1011**010**. Notice that the rate of this Hamming code is 4/7: 4 out of 7 bits are the original information bits. Although it is not obvious (but easy to show after learning a little bit of coding theory), this code corrects any single error. Therefore, compared to the Scheme 2 above, the Hamming code achieves the same error correction ability in a more efficient way: The information rates are 1/3 vs. 4/7 (Scheme 2 vs. Scheme 4).

The following is a pictorial representation of digital communication over a noisy channel with error correcting codes.

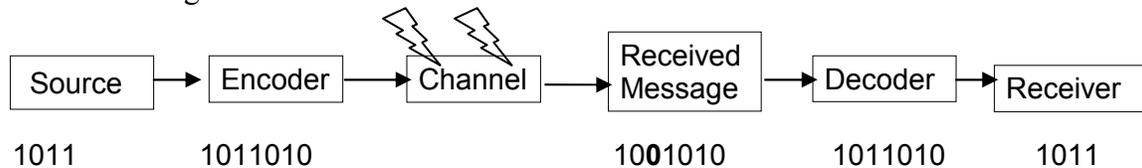


Figure 2: A general picture of communication over a noisy channel

A visual representation of the Hamming Code

The figure below gives an alternative representation of the Hamming code. Each one of the three big circles in the figure corresponds to the three parity check equations defining the Hamming code, and the seven regions correspond to the seven bits in a codeword. The defining equations require that there be an even number of ones in each big circle. If not, we know that there is an error. Moreover, when there is a single error in transmission, we can fix it by flipping a single bit in the appropriate region, corresponding to the location of the error. For instance, distribute the bits in the received vector 1001010 to the regions in the figure. Are the conditions satisfied? If not, can you fix it?

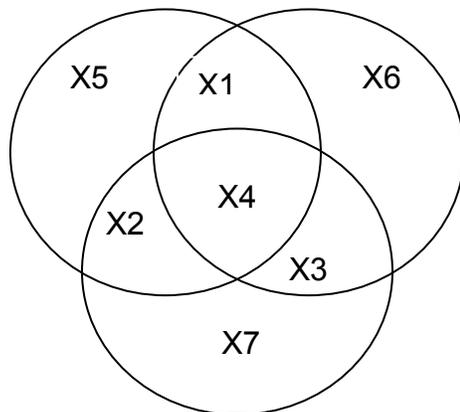


Figure 3: An alternative description of the Hamming code

Although codes used in practice are longer and more sophisticated, the basic principles are the same. These examples show that there are different ways of employing redundancy, some more efficient than others. The question is, therefore, not whether or not redundancy can be useful but how to best use it. Researchers are still looking for more clever ways of employing redundancy to design even more efficient codes. Surprisingly, a lot of theoretical mathematics can be used in the design of good codes. In the rest of this unit, we will be studying some of the formal mathematical methods used in coding theory.

Formal Set-up and Mathematical Definitions

Definition A code C of length n over an alphabet F is a subset of $F^n = Fx_1x_2 \cdots x_n$, the Cartesian product of F with itself n times, which consists of n -tuples from the elements of F .

The elements of C are called codewords. Informally, you can think of a code as the set of all valid words of a language, and a codeword as a particular word in the language. Just like a language does not contain all possible strings as valid words, a code is only a subset of F^n , not all of it (theoretically, it can be all of it but that would not be a useful code. We call this code a trivial code). The alphabet of a code is a finite set. In practice, the most important alphabet is the binary alphabet $B = \{0,1\}$. A code over the binary alphabet is referred to as a binary code. In most of this unit, we will be dealing with binary codes only. However, mathematicians do study codes over other alphabets, especially over finite fields. For example, a ternary code is a code over the field $Z_3 = \{0,1,2\}$, the integers modulo 3.

As an example, let us consider the code $C = \{1010, 0101, 1111\}$ of length 4; it is a subset of B^4 . This code has 3 codewords. Suppose, using this code over a channel, we receive the vector 0100. What can we say about possible transmission errors? Is there an error? Can we correct it? What is the most likely codeword that was sent? We see that the most likely codeword that was sent is 0101. This is because the received vector 0100 is closest to 0101. This brings us to the formal definition of Hamming distance (or simply distance):

Definition The Hamming distance $d(u,v)$ between two vectors u and v is the number of positions they differ. The Hamming weight $wt(u)$ of a vector is the number of non-zero components it has.

In the example above, if we label the codewords c_1, c_2 , and c_3 and the received vector u , the Hamming distances are $d(u,c_1) = 3$, $d(u,c_2) = 1$ and $d(u,c_3) = 3$. Also, $d(c_1,c_2) = 4$, $wt(c_1) = 2$.

Problem 5 Show that for binary vectors u , and v , we have $d(u,v) = wt(u+v)$, where $u+v$ is the usual binary (modulo 2) sum of two vectors.

The Hamming distance is a fundamental notion in coding theory. Error detection/correction is based on Hamming distances. Another important and related notion is the minimum distance of a code.

Definition The minimum distance of a code C is $\min\{d(u,v) : u, v \in C, u \neq v\}$, that is the smallest of the distances between distinct pairs of codewords. Similarly, the minimum weight

of a code C is $\min\{wt(u) : u \in C, u \neq \text{the zero vector}\}$, the smallest weight among non-zero codewords.

The minimum distance of a code determines its error detection/correction capacity, as the following theorem shows:

Theorem 1 Let C be a code with minimum distance d . Then C can detect any $d-1$ errors, and can correct any $\left\lfloor \frac{d-1}{2} \right\rfloor$ (the floor of $(d-1)/2$) errors. Moreover, there are some d errors C cannot detect and there are $\left\lfloor \frac{d-1}{2} \right\rfloor + 1$ errors C cannot correct.

Problem 6 Convince yourself that Theorem 1 is true.

Problem 7 Show that the Hamming distance is a *metric*, i.e. it satisfies the following properties: For all vectors u, v , and w

- i) $d(u, v) \geq 0$
- ii) $d(u, v) = 0$ if and only if $u = v$
- iii) $d(u, v) = d(v, u)$
- iv) $d(u, v) \leq d(u, w) + d(w, v)$ (This is called the triangle inequality)

Linear Codes

Although a binary code is defined as a subset of B^n , it is more convenient to consider subsets with certain mathematical structure instead of dealing with arbitrary subsets. The first observation in that direction is to notice that B^n is a *vector space* over the binary field B . Let us recall from linear algebra that this means we can add vectors in B^n (using component-wise mod 2 sum), and there is a scalar multiplication (and a there are a few properties these operations satisfy). Since there are only two scalars in the binary field, that is 0 and 1, the scalar multiplication is particularly simple.

Since B^n has the structure of a vector space, we can consider subspaces of it. This is exactly what we do in the study of linear codes. We can then use what we know from linear algebra for coding theory. Again recall that a subspace of a vector space is a subset that is a vector space on its own. For binary linear codes, we consider the vector space B^n . In fact, all the codes in Scheme 1 through Scheme 4 are examples of linear codes.

Definition A binary linear code of length n is a vector subspace of B^n .

Problem 8 Show that each one of the codes considered so far (Scheme 1 through Scheme 4) is a linear code. How about the code $C = \{1010, 0101, 1111\}$?

We again recall from linear algebra that every vector space has a dimension. We define the dimension of a linear code as its dimension as a vector space.

Problem 9 Find the dimension of each code in Problem 7.

Problem 10 Let C be a binary code of length k . What is the size of C ? (i.e. how many codewords does C contain?)

The length and the dimension are two of the most important parameters of a linear code. Another important parameter is its minimum distance. A linear code C that has length n , dimension k , and minimum distance d is referred to as an $[n,k,d]$ -code.

Now you found that the Hamming code (Scheme 4) is a linear code. It has length 7, and dimension 4. So it contains $2^4 = 16$ codewords. What is its minimum distance? Recall, that we claimed that it can correct single errors. What does that imply about its minimum distance? How can we go about finding the minimum distance of the Hamming code? An obvious and brute-force strategy would be to list all the codewords and compute all the distances between codewords. The next exercise asks you to do exactly that.

Problem 11 List all the codewords of the Hamming code. Then look at the distances between distinct codewords. How many distances are there to consider?

You probably agree that there are too many distances to check by hand. Later we will learn better methods to determine the minimum distance of a linear code. To this end, let us do a similar exercise which is a little less tedious.

Problem 12 Compute the minimum weight of the Hamming code. How does it compare to the minimum distance? How many weights did you need to compute?

Although Problems 10 and 11 are a little tedious, they are instructive. This kind of tedious work is exactly where we want to make use of a computer algebra system, like Maple. But, it was not a bad idea to first do it by hand for couple of reasons: first we know exactly what we are doing, secondly we really appreciate the power of such systems. Now, let us see how to do these computations in Maple. Although, this example was doable by hand, there are much larger codes for which computations by hand is prohibitive.

```
> with(linalg):
```

```
> with(LinearAlgebra):
```

```
#include these commands to do linear algebra in Maple
```

Next, enter the equations defining the Hamming code

```
> eq1:=x1+x2+x4+x5=0 mod 2;
```

```
eq1 := x1 + x2 + x4 + x5 = 0
```

```
> eq2:=x1+x3+x4+x6=0 mod 2;
```

```
eq2 := x1 + x3 + x4 + x6 = 0
```

```
> eq3:=x2+x3+x4+x7=0 mod 2;
```

Then let Maple compute the matrix of this system of equations

```
> A:=genmatrix({eq1,eq2,eq3},[x1,x2,x3,x4,x5,x6,x7],'b');
A:=
$$\begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

```

Now, let Maple solve this system using matrix form

```
> C:=Linsolve(A,b,'r',t) mod 2;
C:=[t3+t5+t7,t3+t5+t6,t3,t5+t6+t7,t5,t6,t7]
```

Note that, the rank of the matrix is 3 and there are 4 free variables (which Maple calls t3,t5,t6,t7).

Now, letting the free variables take all possible values (0 or 1), we can generate all solutions, hence all vectors (codewords) of the Hamming code as follows:

```
> ls:=[]; # store solutions in a list, start with an empty list
ls:=[]
> nops(ls); # gives the length of the list
0
> for t[3] from 0 to 1 do
  for t[5] from 0 to 1 do
    for t[6] from 0 to 1 do
      for t[7] from 0 to 1 do
        ls:=op(ls),[C[1] mod 2,C[2] mod 2,C[3] mod 2,C[4] mod 2,C[5]
mod 2,C[6] mod 2,C[7] mod 2]];
      od:
    od:
  od:
od:
> nops(ls);
16
```

This means 16 vectors are stored in the list. We can access the individual elements in the list as follows:

```
> ls[5];
[1, 1, 0, 1, 1, 0, 0]
And here is all of the elements in the list, hence all codewords of the Hamming code.
> ls[1..16];
[[0, 0, 0, 0, 0, 0, 0], [1, 0, 0, 1, 0, 0, 1], [0, 1, 0, 1, 0, 1, 0], [1, 1, 0, 0, 0, 1, 1],
 [1, 1, 0, 1, 1, 0, 0], [0, 1, 0, 0, 1, 0, 1], [1, 0, 0, 0, 1, 1, 0], [0, 0, 0, 1, 1, 1, 1],
 [1, 1, 1, 0, 0, 0, 0], [0, 1, 1, 1, 0, 0, 1], [1, 0, 1, 1, 0, 1, 0], [0, 0, 1, 0, 0, 1, 1],
 [0, 0, 1, 1, 1, 0, 0], [1, 0, 1, 0, 1, 0, 1], [0, 1, 1, 0, 1, 1, 0], [1, 1, 1, 1, 1, 1, 1]]
```

Next, we define a procedure to compute the distance between two vectors.

```
> distance:=proc(u,v)
  local n,d,i;
  n:=vectdim(u);
  d:=0;
  for i from 1 to n do
```

```

    if u[i]<>v[i] then d:=d+1; fi;
  od;
  return d;
end:

```

And another one, to compute the weight of a vector.

```

> weight:=proc(u)
  local n,w,i:
  n:=vectdim(u):
  w:=0:
  for i from 1 to n do
    if u[i]<>0 then w:=w+1; fi;
  od;
  return w;
end:

```

Finally, we can use all of above, to find the minimum distance and the minimum weight of the Hamming code:

```

> mindist:=7;
                                     mindist := 7

> for i from 1 to nops(ls) do
  for j from i+1 to nops(ls) do
    if (distance(ls[i],ls[j])< mindist) then mindist:=
distance(ls[i],ls[j]): fi:
  od:
od:
> mindist;
3

```

So, the minimum distance is 3. Now, check the minimum weight.

```

> minweight:=7;
minweight := 7
> for i from 1 to nops(ls) do
>   if (weight(ls[i])>0 and weight(ls[i])< minweight) then
minweight:= weight(ls[i]): fi:
> od:
> minweight;
3

```

Maple confirms that minimum weight is 3 as well. Although, this example was small enough to be doable by hand, there are many codes that are much larger. We need computers to do computations for such codes.

Fortunately, the fact that the minimum distance of the Hamming code is equal to its minimum weight is not a coincidence! This is true for any linear code. So, let us record this important result as a Theorem.

Theorem 2 For a linear code C , its minimum distance is equal to its minimum weight.

Problem 13 It is not hard to prove this theorem. Can you do that? Let us consider only binary codes. Here is some hint: You want to show that the minimum of the set $\{d(u,v) : u, v \text{ in } C, u \neq v\}$ is the same as the minimum of the set $\{wt(u) : u \text{ in } C, u \neq \text{the zero vector}\}$. Use the linearity of C to show that every number in the first set appears in the second, and vice versa.

Problem 14 Let C be linear code of dimension k . How many pairs does one need to consider when computing the minimum distance directly from the definition? How many weights need to be computed to find the minimum weight directly from the definition?

So, you found that the minimum distance of the Hamming code is 3. Therefore, it is a $[7,4,3]$ -code. It can detect any double errors and correct any single errors.

Some History on Hamming Codes

Coding theory is a subfield of a larger field called information theory. Cryptography, sometimes confused with coding theory, is another, distinct subfield of information theory. The whole subject started with the seminal paper of Claude Shannon [8] in the middle of 20th century. Shannon established theoretical limits on how good a code can be, and he proved that “good codes” exist. However, his proofs were probabilistic and existential, not constructive. Shortly after Shannon’s theoretical results, Richard Hamming [3] and Marcel Golay [2] gave the first explicit constructions of error-correcting codes, known as Hamming codes and Golay codes. There is actually an infinite family of Hamming codes. What we have studied is the smallest non-trivial Hamming code. Richard Hamming was working at Bell Laboratories when he invented the Hamming code. He was frustrated that those early computers were detecting errors and halting, hence wasting a lot of computations. He thought, if the computer can detect errors, why can it not correct them? This led him to discover his single-error correcting code in mid 1940’s. More about Richard Hamming and Hamming codes can be found at [3], [4]. There has been a vast amount of research since then to explicitly construct efficient codes that are promised by the theory. Mathematicians invented many other and more sophisticated codes over the years. In recent years, researchers came close to attaining the best possible theoretical results in practice [6].

Generator and Parity Check Matrices

We have learned that a linear code is a subspace of B^n , hence a vector space by itself. There are other ways of obtaining or defining vector spaces that we recall from linear algebra. For example, the row space of a matrix defines a vector space. So does the null space. We can therefore define linear codes via matrices.

Definition A generator matrix of a linear code C is a matrix G whose rows span C . A parity check matrix H of a linear code is a matrix whose null space is C . Therefore, the code (of dimension k) can be defined as either $C = \{ u * G : u \text{ in } B^k \}$ or $C = \{ u \text{ in } B^n : H * u = 0 \text{ vector} \}$. The rank of G or the nullity of H give the dimension of C .

Problem 15 Find a generator and a parity check matrix for the Hamming code.

We again recall from linear algebra that a k by n matrix over B defines a linear transformation from B^k to B^n . So, the vector-matrix multiplication $u * G$ corresponds to encoding: The information vector u of length k is transformed into a codeword $v = u * G$ of length n . The redundancy is added through the vector-matrix multiplication. The parity check matrix is useful for checking for errors. Suppose the code has a parity check matrix H . If a vector w is received, we compute the product $H * w$, called the *syndrome* of w . If the syndrome is the zero vector, we assume that there was no error. If not, we know that there is an error. In that case, can we correct the error? If yes, how? In general, decoding is a more difficult problem. There is an alternative description of the Hamming code based on a parity check matrix that has a very elegant way of decoding. We are about to learn this version of the Hamming code. We first note another useful property of the parity check matrix:

Theorem 3 Let H be a parity check matrix of a code C such that any set of $d-1$ columns of H are linearly independent, and there is a set of d columns of H that is linearly dependent. Then the minimum weight (hence the minimum distance) of C is d .

Proof The proof of this is based on the following observation: Let H be a $k \times n$ matrix with columns h_1, h_2, \dots, h_n so, write H as $H = [h_1 \ h_2 \ \dots \ h_n]$, and $v = (v_1, v_2, \dots, v_n)$ be a vector, then the product $H * v$ gives the linear combination $v_1 * h_1 + v_2 * h_2 + \dots + v_n * h_n$. We proceed by contradiction. Suppose there is a vector v of weight less than d in C . Let v have non-zero components at positions i_1, i_2, \dots, i_r , where $r < d$. Then, we have $H * v = 0$ hence $v_{i_1} * h_{i_1} + \dots + v_{i_r} * h_{i_r} = 0$. This means that the set of $r < d$ columns h_{i_1}, \dots, h_{i_r} is linearly dependent, but this contradicts the assumption. Hence there is no codeword of weight less than d . On the other hand, existence of a set of s linearly dependent columns implies the existence of a codeword of weight d .

Problem 16 Use theorem 3 to show that the Hamming code has minimum weight 3.

Decoding with the Hamming Code and Equivalent Codes

When we use an error-correcting code in a communication system, it is important to have an efficient way of correcting errors (this is called *decoding*) when errors do occur. Earlier in this unit we have seen a visual method to decode the Hamming code using circular regions (page 8). Although that is a neat way of illustration for humans, it is not as useful for computers. We now give an alternative description of the Hamming code that is very convenient for decoding using computers. We will also introduce the concept of equivalent codes.

Consider the numbers 1 through 7 and their binary representations. For example, the binary representation of 6 is 110 because $1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 6$. We define a 3 by 7 matrix whose columns are binary representations of the numbers 1-7. We will use that matrix as a parity check matrix of the Hamming code. Let us use Maple to construct this matrix.

First, Maple can find binary representations of numbers as follows. The command

```
> binrep:=convert(6,base,2);
binrep := [0, 1, 1]
```

converts the number 6 to binary. However, the order of bits is reverse of what we normally use, but this is easy to fix. Now, we can write a loop to construct the matrix.

```
> H:=[]:
> for s from 1 to 7 do
>   binrep:=convert(s,base,2):
>   column:=vector(3,0):
>   for t from 1 to vectdim(binrep) do # reversing the bits
>     column[3-t+1]:=binrep[t]:
> od:
> H:=augment(op(H),column):
> od:
> evalm(H);
```

$$\begin{matrix} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} \\ \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \end{matrix}$$

Notice that this parity check matrix is not exactly the same as what you found in Problem 15. Although, both matrices have the same set of columns, they are in different order. If the order of the columns in the latter matrix is 1,2,3,4,5,6,7 then the order in the original matrix is 7,6,3,5,2,1,4. It turns out that permuting the rows of a parity check matrix does not change the code in any essential way. The effect of such a permutation is exactly to permute the coordinates of codewords. We call two versions of such codes equivalent codes. Equivalent codes have the same essential characteristics.

Now let us list the codewords in this version of the Hamming code.

The command

```
> nH:=Nullspace(H) mod 2;
nH := {[1, 1, 1, 0, 0, 0, 0], [1, 0, 0, 1, 1, 0, 0], [0, 1, 0, 1, 0, 1, 0], [1, 1, 0, 1, 0, 0, 1]}
```

gives a basis for the nullspace of H, hence a basis for the Hamming code.

Using this basis, we can generate all the vectors in the code:

```
> v1:=Transpose(Vector(7,0)):
> v2:=Transpose(Vector(7,0)):
> v3:=Transpose(Vector(7,0)):
> v4:=Transpose(Vector(7,0)):
> for i from 1 to 7 do
>   v1[i]:=nH[1][i]:
```

```

> od:

> for i from 1 to 7 do
  v2[i]:=nH[2][i]:
> od:

> for i from 1 to 7 do
  v3[i]:=nH[3][i]:
> od:
> for i from 1 to 7 do
  v4[i]:=nH[4][i]:
> od:

> ls2:=[]:

> for i1 from 0 to 1 do
>   for i2 from 0 to 1 do
>     for i3 from 0 to 1 do
>       for i4 from 0 to 1 do
> ls2:=[op(ls2), [VectorScalarMultiply(v1,i1) mod
2+VectorScalarMultiply(v2,i2) mod 2+VectorScalarMultiply(v3,i3)
mod 2+VectorScalarMultiply(v4,i4) mod 2]]:
> od:
> od:
> od:
> od:
> for i from 1 to nops(ls2) do
  ls2[i];
> od;
(output written in compact form for space)
[[0, 0, 0, 0, 0, 0, 0]], [[1, 1, 0, 1, 0, 0, 1]], [[0, 1, 0, 1, 0, 1, 0]], [[1, 0, 0, 0, 0, 1, 1]]
[[1, 0, 0, 1, 1, 0, 0]], [[0, 1, 0, 0, 1, 0, 1]], [[1, 1, 0, 0, 1, 1, 0]], [[0, 0, 0, 1, 1, 1, 1]]
[[1, 1, 1, 0, 0, 0, 0]], [[0, 0, 1, 1, 0, 0, 1]], [[1, 0, 1, 1, 0, 1, 0]], [[0, 1, 1, 0, 0, 1, 1]]
[[0, 1, 1, 1, 1, 0, 0]], [[1, 0, 1, 0, 1, 0, 1]], [[0, 0, 1, 0, 1, 1, 0]], [[1, 1, 1, 1, 1, 1, 1]]

```

Problem 17 Verify that if the permutation 1->7, 2->6, 3->3, 4->5, 5->2, 6->1, 7->4 is applied to the coordinates of each of the vectors in this version of the Hamming code, we do get the set of vectors in the original version of the Hamming code.

Let us also obtain a generator matrix for the Hamming code that we will be used for encoding.

```

> G:=[]:
> for i from 1 to nops(nH) do
>   G:=stackmatrix(op(G), nH[i]):
> od:
> evalm(G);

```

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Now, let us encode the original data 1,0,1,1. We use the generator matrix for encoding.

```
> v:=vector([1,0,1,1]);
v := [1, 0, 1, 1]
> c:=map(x->x mod 2, evalm(v&*G));
c := [0, 1, 1, 0, 0, 1, 1]
```

(Explanation: `evalm(v&*G)` does the vector-matrix multiplication, and `x->x mod 2` reduces the result mod 2)

So, the resulting codeword is $c = (0,1,1,0,0,1,1)$. To see if there is an error in the received vector r , we look at the syndrome $H*r$. If it is zero, there is no errors. If not, there is. Let us verify this: Suppose there is no error, so $r = c$.

```
> r:=c;
> syn:=map(x->x mod 2, evalm(H&*r));
syn := [0, 0, 0]
```

Which means there is no error. Suppose now that an error occurs at position 6. That means that a 1 is added to that position, or the vector $(0,0,0,0,0,1,0)$ is added to the entire codeword. So the received vector is $r = (0,1,1,0,0,0,1)$. Let see what we obtain then.

```
> r[6]:=(r[6]+1) mod 2; # introduce error at position 6
r_6 := 0
> syn:=map(x->x mod 2, evalm(H&*r));
syn := [1, 1, 0]
```

We see that the syndrome is not the zero vector, so we know there is an error. The next question is, do we know where the error is? The elegant thing about this version of the Hamming code is that the syndrome gives the location of the error in binary. The vector 1,1,0 is the binary representation of the number 6, which means that the error is at location 6. Neat huh?

Problem 18 Show that in general when there is a single error in the transmitted codeword of a Hamming code, the syndrome of the received vector is the position of the error in binary.

In summary, notice that generator matrices are useful for encoding and parity-check matrices are useful for decoding.

General Binary Hamming Codes

It turns out that there are infinitely many Hamming codes. We just had fun with the smallest (non-trivial) Hamming code. Let us learn how to define all binary Hamming codes. A binary Hamming code has length $2^r - 1$ for some positive integer r . Their length have this particular form, they do not exist for arbitrary lengths. In the example we studied, r was equal 3. They are defined through parity check matrices, similarly to the way we defined length 7 code. The parity check matrix H has binary representations (vectors of length r) of the numbers 1 through $2^r - 1$ as its columns. So it is an r by $2^r - 1$ matrix. Note that this means the rows of H give the set of all non-zero binary vectors of length r . So, the length of a Hamming code is $n = 2^r - 1$. Let us determine the other two important parameters of a Hamming code. The dimension is equal to the nullity of H , which is $2^r - 1 - r$ - the rank of H . Notice that the rank of H is at most r . We claim that it is in fact equal to r .

Problem 19 Verify this claim.

Therefore, the dimension of a Hamming code is $2^r - 1 - r$.
The last important parameter to determine is its minimum distance. We claim that it is 3.

Problem 20 Verify this claim. (Hint: Use Theorem 3)

Therefore, the parameters of a general Hamming code are $[2^r - 1, 2^r - 1 - r, 3]$ for some positive integer r . For the Hamming code we studied in detail, $r = 3$. Hamming codes are 1-error-correcting codes: they can correct any single errors.

Problem 21 Use Maple to construct a parity check matrix H and generator matrix G for the Hamming code for $r = 4$. Use G to encode the message 1,1,1,0,0,0,1,1,1,0,1. Introduce single errors at selected positions and see if you can correct them using H .

Hamming Codes Are Perfect

There are more useful properties of Hamming codes we are yet to explore. In a mathematically well-defined sense, the Hamming codes are “perfect”. This notion of perfectness is related to the concept of correctable errors. Recall that we are using the maximum likelihood principle in correcting errors. If a received vector is not a codeword, then we know there is an error. If there is a unique codeword the received vector is closest to, then we correct it to that codeword. There are situations where there are more than one codeword with the same distance to a given vector. In those cases, we cannot correct errors unambiguously.

Problem 22 Consider the code from Problem 7, that is $C = \{1010, 0101, 1111\}$. Give an example of vector that has the same distance to more than one codewords.

Problem 23 Let C be a code with an even minimum distance. Show that there exists a vector that has the same distance to more than one codewords.

Definition A code of length n is said to be e -perfect if every vector in B^n is either a codeword or is within e units of a unique codeword.

Problem 23 shows that the minimum distance of a perfect code cannot be an even number, therefore it is an odd number. If every vector in B^n is within e units of a unique codeword of a code C , then C is said to be e -perfect (also e error-correcting). Having an odd minimum distance is a necessary condition for a code to be perfect, but is it also sufficient?

Problem 24 Show that having an odd minimum distance is not sufficient for a code to be perfect by considering the code whose generator matrix is

$$G := \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Use Maple to find the minimum distance (recall minimum weight is equal to minimum distance of a linear code). Then find a vector that has the same distance to more than one codeword. Hence, show that this code is not perfect (even though it has an odd minimum distance).

Now, we will characterize perfectness using some equations. First, we define a sphere of radius r around a vector u as $S_r(u) = \{v \in B^n : d(u, v) \leq r\}$, which is the set of all vectors in the space B^n whose distance from u is less than or equal to r , i.e., those vectors within r units of u .

In Theorem 1, we showed that a code of minimum distance d can correct $\left\lfloor \frac{d-1}{2} \right\rfloor$ errors. This

is equivalent to saying that spheres of radius $\left\lfloor \frac{d-1}{2} \right\rfloor$ around codewords are disjoint. We can represent this fact pictorially as follows:

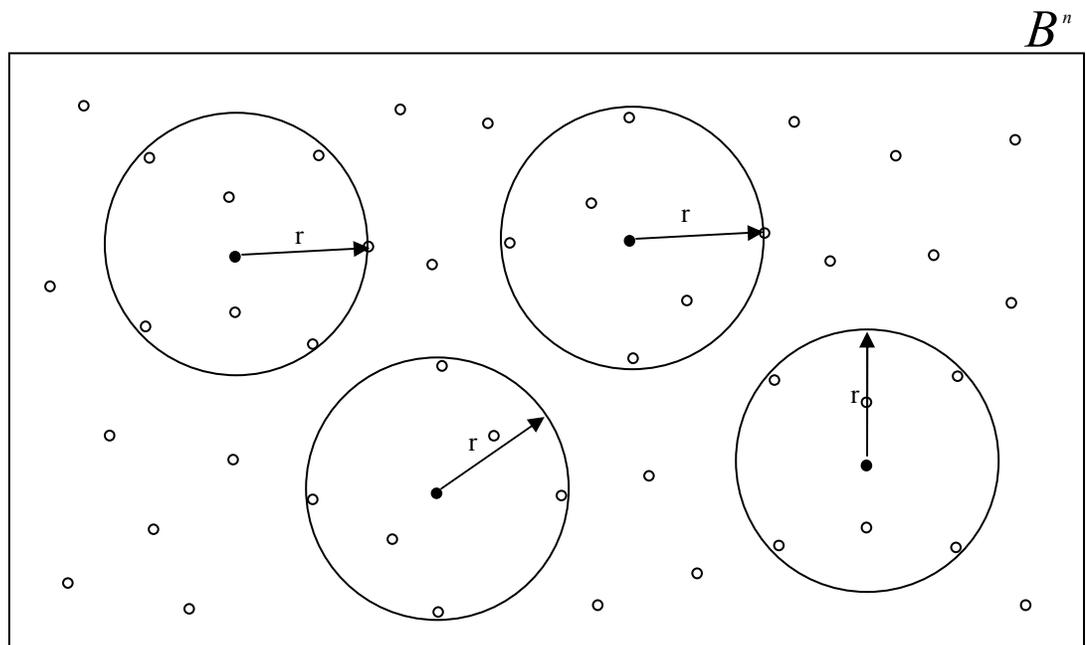


Figure 4: A graphical representation of the sphere packing bound

In the figure above, the solid dots represent codeword. Other vectors in the space B^n are depicted by small circles. Big circles represent spheres of radius r . (A more correct term to use would be “ball” rather than “sphere”, but historically they are called spheres in this context).

For a code of minimum distance d , spheres of radius $r = \left\lfloor \frac{d-1}{2} \right\rfloor$ are disjoint, i.e., they do not overlap.

Our next question is to determine the number of vectors in a sphere of radius r . This is a counting problem. To solve this problem, we can proceed as follows. Given a vector u in B^n , let us first determine the number of vectors that have distance exactly 1 to u . Such a vector v will have all but 1 position the same as vector u . So, to count all these vectors, we just find all possible positions that v will differ from u . Obviously, that number is n .

Next, what is the number of vectors that have distance 2 from u . To determine such a vector v , we just need to pick 2 positions at which v will differ from u . The total number of such

choices is $\binom{n}{2}$ (pronounced “n choose 2”) which is equal to $\frac{n(n-1)}{2}$. Proceeding in this

manner, we find that the total number of vectors in $S_r(u)$, or “the volume of the sphere” is

$\binom{n}{0} + \binom{n}{1} + \dots + \binom{n}{r}$ (the term $\binom{n}{0} = 1$ is for u itself, the center of the sphere), and this number is

independent of the vector at the center. Now, if C is a code of length n , and minimum distance d , then we know that the spheres of radius $e = \left\lfloor \frac{d-1}{2} \right\rfloor$ are disjoint. Therefore, the total number

of vectors in all of the spheres around codewords is no more than the total number of vectors in the whole space B^n . The latter number is 2^n . Noting that the volumes of the spheres are all the same and there are $|C|$ spheres around codewords, the number of vectors in all of the spheres

around codewords is $|C| \cdot \left[\binom{n}{0} + \binom{n}{1} + \dots + \binom{n}{e} \right]$. This number is no more than 2^n . This gives

us the well-known bound known as sphere packing bound.

Theorem 4 For a code of length n , and minimum distance $d = 2e+1$ (or $2e+2$), we have

$$|C| \cdot \left[\binom{n}{0} + \binom{n}{1} + \dots + \binom{n}{e} \right] \leq 2^n.$$

To say that a code is perfect is equivalent to saying that there are no vectors left outside the spheres of radius e , i.e. those spheres cover the whole space. Hence, the equality is attained in sphere packing bound. Graphically, this means that there are no vectors outside the spheres in

Figure 4 if radii of the spheres are all equal to $\left\lfloor \frac{d-1}{2} \right\rfloor$.

Problem 25 Show that the $[6,3,3]$ is not perfect and $[7,4,3]$ Hamming code is perfect using the sphere packing bound.

Problem 26 Show that the general the Hamming codes are all perfect.

Hence, Hamming codes provide an infinite family of perfect codes. It turns out that perfect codes are rare. Besides Hamming codes, there exist only few other perfect codes. Only other linear perfect codes are what are known as Golay codes. They are also well-known codes that were invented around the same time as Hamming codes [2]. For more information on Golay codes see [5]. It took mathematicians a great deal of effort to prove the fact that there exist few perfect codes. To learn more about perfect codes and their classification, see page 24 of [5].

Additional Projects Related to Hamming Codes

Project 1: Hamming Codes and the Hat Puzzle

The binary Hamming code of length 7 has also an interesting connection to the well-known puzzle “The Hat Problem”, featured in the *New York Times* [7], with a more technical treatment in [1]. The puzzle is as follows:

At a mathematical show with 7 players, each player receives a hat that is red or blue. The color of each hat is determined by a coin toss. So the hat colors of players are determined randomly and independently. Each person can see the other players’ hats but not his own. When the host signals, all players must simultaneously guess the color of their own hats or pass. The group shares a \$1 million prize if at least one player guesses correctly and no players guess incorrectly.

No communication of any sort is allowed between the players during the show, but they are given the rules in advance and they can have an initial strategy session before the game starts. What should they do to maximize their chances?

Question 1 There is an obvious strategy with a 50% chance of winning. What is that?

Question 2 Is there a strategy with a higher chance of winning?

As you might predict, there is a strategy with a much higher chance of winning than 50%, and it involves [7,4,3] Hamming code. The fact that the Hamming code is perfect and 1-error correcting makes it a good solution, in fact the best possible solution. Describe how the Hamming code can be used to solve this puzzle with a winning probability of 7/8.

Project 2: Hamming Codes over Integers Modulo p .

We studied only binary codes so far. However, we can use any finite set as the alphabet. If we are interested in linear codes, the alphabet must have the structure of a field. We know that Z_p , integers modulo p , is a field when p is a prime number. For instance, Z_3 is a field. A code whose alphabet is Z_3 is referred to as a ternary code. A linear code of length over Z_p is a

vector subspace of Z_p^n . A code over Z_p is referred to as a p-ary code. Most concepts for binary linear codes are defined in the same way for codes over other finite fields, such as the dimension and the Hamming distance. A code of length n , dimension k , and minimum Hamming distance d over Z_p is referred to as an $[n, k, d]_p$ -code.

It turns out that Hamming codes can be defined over any finite field. The purpose of this project is to introduce general Hamming codes over Z_p for any prime p . Here is the definition:

Definition Let r be a positive integer. The p-ary Hamming code is defined as the code whose generator matrix H contains all vectors of length r over Z_p whose first non-zero component is 1.

Question 1 What is the number of columns of H , i.e. what is the length of the Hamming code?

Questions 2 What is the dimension of the Hamming code?

Question 3 Show that the minimum distance of the Hamming code is 3.

Question 4 Obtain the general version of the sphere packing bound for Z_p . Then show that Hamming codes are perfect.

Question 5 Implement the Hamming code with $p = 5$, and $r = 2$ in Maple (find a parity check matrix and a generator matrix). Verify the code parameters using Maple. Encode the message $m=(1,0,2,4)$ into a codeword c , then introduce an error at position 5 of c by adding 4 to that position. Then, come up with a decoding algorithm for this (and general) Hamming code. Notice that there are two things to figure out in decoding: The position of the error, and the magnitude of the error.

Project 3: Hamming Codes Over an Arbitrary Finite Field.

Integers modulo p , for a prime p , are well-known examples of finite fields. These are called prime fields. However, there exist other finite fields. It is well-known that there is a unique finite field for every prime power $q = p^n$, called the Galois field of order q , denoted by $GF(q)$. For instance, there is a finite field with $2^2 = 4$ elements (and it is not integers modulo 4). Typically, students are only familiar with the prime fields until they learn about more general finite fields in Abstract Algebra. This project is the same as project 2, but over non-prime finite fields. So, define Hamming codes over an arbitrary finite field. As a concrete example, implement the Hamming code for $q = 4$, and $r = 2$.

REFERENCES

1. Bernstein, M. 2001. The Hat Problem and Hamming Codes, *Focus*. 21(8): 4-6
2. Golay, M. J. E. 1949. Notes on digital coding. *Proc. IRE*. 37: 637
3. Hamming, R. W. 1950. Error-detecting and error-correcting codes. *Bell System Technical Journal*. 29: 147-160
4. Morgan S. P. 1998. Richard Wesley Hamming (1915-1998), *Notices of the American Mathematical Society* 45 (8): 972-977
5. Pless, V. *Introduction to the theory of error-correcting codes*, Third edition, 1998. John Wiley & Sons, New York.
6. Richardson T. J., Shokrollahi M. A., and Urbanke R. L. Design of capacity-approaching irregular low-density parity-check codes. *IEEE Transactions on Information Theory*. 47 (2): 619-637.
7. Robinson, S. 2001. Why Mathematicians now care about their hat color. *The New York Times*. Available: <http://www.cecs.csulb.edu/~ebert/hatProblem/nyTimes.htm>
8. Shannon, C. E. 1948. A mathematical theory of communication. *Bell System Technical Journal*. 27: 379-423 and 623-656
9. Shetter, W. Z. 2000. This essay is redundant. Available: <http://home.bluemarble.net/~langmin/miniatures/redund.htm>