OPTIMIZING SIZES OF CODES

LINDA MUMMY

Abstract. This paper shows how to determine if codes are of optimal size. We discuss coding theory terms and techniques, including Hamming codes, perfect codes, cyclic codes, dual codes, and parity-check matrices.

1. Introduction to Coding Theory

Error correcting codes and check digits were developed to counter the effects of static interference in transmissions. For example, consider the use of the most basic code. Let 0 stand for "no" and 1 stand for "yes." A mistake in one digit relaying this message can change the entire meaning. Therefore, this is a poor code. One option to increase the probability that the correct message is received is to send the message multiple times, like 000000 or 111111, hoping that enough instances of the correct message get through that the receiver is able to comprehend the original meaning. This, however, is an inefficient method of relaying data.

While some might associate "coding theory" with cryptography and "secret codes," the two fields are very different. Coding theory deals with transmitting a *codeword*, say $\mathbf{x}$, and ensuring that the receiver is able to determine the original message $\mathbf{x}$ even if there is some static or interference in transmission. Cryptography deals with methods to ensure that besides the sender and the receiver of the message, no one is able to encode or decode the message.

We begin by discussing a real life example of the error checking codes (ISBN numbers) which appear on all books printed after 1964. We go on to discuss different types of codes and the mathematical theories behind their structures. Some codes we discuss are Hamming codes, perfect codes, cyclic codes, and linear codes, along with the methods of their generation.

2. ISBN Numbers and Basic Coding Theory Definitions

All books published after 1964 have a 10 digit ISBN number. This vector of digits $\mathbf{i} = a_1a_2a_3a_4a_5a_6a_7a_8a_9a_{10}$, where $0 \leq a_i \leq 9$, satisfies the condition

$$\sum_{j=1}^{10} ja_j \equiv 0 \pmod{11}. \tag{1}$$

Take, for example, the vector $\mathbf{i} = 0743277708$. Then

$$\big((10)\mathbf{0} + (9)\mathbf{7} + (8)\mathbf{4} + (7)\mathbf{3} + (6)\mathbf{2} + (5)\mathbf{7} + (4)\mathbf{7} + (3)\mathbf{7} + (2)\mathbf{0} + (1)\mathbf{8}\big) \pmod{11} = 0,$$

therefore $\mathbf{i}$ could be a valid ISBN. Suppose Alice tries to send $\mathbf{i}$, the ISBN number of a book for Bob to buy online, to Bob. Instead, because of static or interference,

Bob receives $\mathbf{i}' = 0743227708$, which is Alice's message with the 5th digit incorrect. To check if this could be a valid ISBN, Bob multiplies the ISBN term by term with the sequence

$$\mathbf{w} = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10$$

and considers the sum modulo 11. He finds:

$$\begin{aligned}\mathbf{w} \cdot \mathbf{i}' \pmod{11} = \big((10)\mathbf{0} + (9)\mathbf{7} + (8)\mathbf{4} + (7)\mathbf{3} + (6)\mathbf{2} + (5)\mathbf{2} \\ +(4)\mathbf{7} + (3)\mathbf{7} + (2)\mathbf{0} + (1)\mathbf{8}\big) \pmod{11} = 8.\end{aligned}$$

This ISBN is invalid since Equation 1 does not hold, so Bob needs to determine what could be wrong and fix it. The most common error made, either through interference in transmission or through human error, is one digit of the ISBN being changed to an incorrect number.

Operating under the assumption that one digit is incorrect, it is possible to iterate though all the valid ISBNs which are one digit different than Alice's message and satisfy the Equation 1 test for ISBN numbers. This gives:

$$\begin{aligned}\mathbf{i}_1 &= 8743227708 \\ \mathbf{i}_2 &= 0043227708 \\ \mathbf{i}_3 &= 0733227708 \\ \mathbf{i}_4 &= 0745227708 \\ \mathbf{i}_5 &= 0743827708 \\ \mathbf{i}_6 &= 0743277708 \\ \mathbf{i}_7 &= 0743225708 \\ \mathbf{i}_8 &= 0743227808 \\ \mathbf{i}_9 &= 0743227778 \\ \mathbf{i}_{10} &= 0743227700 \\ \mathbf{i}_{11} &= 0743227708\end{aligned}$$

Out of these numbers, only $\mathbf{i}_6$, $\mathbf{i}_7$, and $\mathbf{i}_9$ are ISBN numbers of books available on Amazon.com. We have narrowed down the possibilities for the message Alice originally sent from every book ever written to one of three books. Of these, $\mathbf{i}_6$ is the vector Alice originally sent. We are able to detect that the ISBN number Bob received is not valid, but we do not have an algorithm for taking an invalid ISBN and figuring out exactly which ISBN Alice meant to send. If we had such a method, it would be denoted as error correction. From this, we conclude that Bob's attempt at error correcting, while not perfect, drastically narrows down the possibilities for the original message.

We now show that no valid codeword $\mathbf{x}$ with one digit wrong or with two digits interchanged to form $\mathbf{x}'$ can possibly satisfy Equation 1. Therefore $\mathbf{x}'$ is not a codeword.

Theorem 1. *Let $A = a_1 a_2 \dots a_{10}$ satisfy the congruence $\sum_{i=1}^{10} i a_i \equiv 0 \pmod{11}$. Replace a_j $(1 \leq j \leq 10)$ with b $(0 \leq b \leq 9)$ in A to get the codeword Y. Then Y satisfies $\sum_{i=1}^{10} i a_i \equiv 0 \pmod{11}$ only if $a_j = b$.*

Proof. Assume Equation 1. Replacing a_j with b we want to find b such that

$$\sum_{i=1}^{10} ia_i + jb - ja_j \equiv 0 \pmod{11}. \tag{2}$$

Since A is a valid ISBN number, we know Equation 1 is true, so

$$jb - ja_j \equiv 0 \pmod{11} \tag{3}$$

which implies $11 \mid j(b - a_j)$. Since $1 \leq j \leq 10$, and thus $\gcd(11, j) = 1$, we know $11 \mid (b - a_j)$. Since $0 \leq b \leq 9$ and $0 \leq a_j \leq 9$, then $b - a_j = 0$ and so $a_j = b$. □

Theorem 2. *Let $A = a_1a_2 \dots a_{10}$ satisfy $\sum_{i=1}^{10} ia_i \equiv 0 \pmod{11}$. If $Y = b_1b_2 \dots b_{10}$ then choose $j \neq k$ and then, for $i \neq j$ and $i \neq k$, $a_i = b_i$, $a_j = b_k$ and $a_k = b_j$. Then $\sum_{i=1}^{10} ib_i \equiv 0 \pmod{11}$ only if $a_j = a_k$.*

Proof. Assume $\sum_{i=1}^{10} ia_i \equiv 0 \pmod{11}$. Then

$$\sum_{i=1}^{10} ib_i = \sum_{i=1}^{10} ia_i + ja_k + ka_j - ja_j - ka_k. \tag{4}$$

Because $\sum_{i=1}^{10} ia_i \equiv 0 \pmod{11}$, we can rewrite Equation 4 as

$$\begin{aligned} ja_k + ka_j - ja_j - ka_k &= j(a_k - a_j) + k(a_j - a_k) \\ &= (j - k)(a_k - a_j). \end{aligned}$$

When is $(j - k)(a_k - a_j) \equiv 0 \pmod{11}$? We know $11|(j - k)(a_k - a_j)$. Since 11 is prime, $11|(j - k)$ or $11|(a_k - a_j)$. Because $1 \leq k \leq 10$ and $1 \leq j \leq 10$, $|j - k| \leq 9$ but $j \neq k$, and thus $11 \nmid (j - k)$. We know $11|(a_k - a_j)$. Since $a_j, a_k \in \mathbb{Z}_{10} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ we have $11|(a_k - a_j)$ if and only if $a_k - a_j = 0$. Thus $a_k = a_j$. □

2.1. **Definitions.** In order to discuss codes, we need to have proper notation to deal with them. Let us define $F_q = \{0, 1, 2, \dots, q - 1\}$. We can think of F_q as the alphabet we use to form codewords, which we form by grouping elements of F_q together. We use $(F_q)^n$ to denote the set of all possible codewords formed using n elements from F_q:

$$(F_q)^n = \{a_1a_2 \dots a_n | a_i \in F_q \, \forall \, 0 \leq i \leq n\}.$$

A code is any subset of $(F_q)^n$. When we explore ISBN numbers we deal with elements of F_{10}, since ISBN numbers are formed as strings of the integers 0 to 9. Since all ISBN numbers are of length 10, the valid ISBN numbers are all elements of $(F_{10})^{10}$. There are elements of $(F_{10})^{10}$ which are not valid ISBN numbers, like $\mathbf{i}_1 = 0743227708$, from Section 2. Thus we can conclude that the set of valid ISBN numbers is a *subset* of $(F_{10})^{10}$.

Now that we have a method for defining codes and codewords, we define a metric to measure distances between codewords. We call this metric the Hamming Distance.

Definition 1. *The Hamming Distance measures the distance between two vectors* $\mathbf{x}$ *and* $\mathbf{y}$ *by counting the number of places in which their digits differ.*

Theorem 3. *The Hamming Distance satisfies the following conditions:*

(1) $d(\mathbf{x}, \mathbf{y}) = 0$ *if and only if* $\mathbf{x} = \mathbf{y}$.
(2) $d(\mathbf{x}, \mathbf{y}) \geq 0$ *for all* $\mathbf{x}, \mathbf{y} \in (F_q)^n$.
(3) $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$ *for all* $\mathbf{x}, \mathbf{y} \in (F_q)^n$.
(4) $d(\mathbf{x}, \mathbf{y}) \leq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{z}, \mathbf{y})$ *for all* $\mathbf{x}, \mathbf{y}, \mathbf{x} \in (F_q)^n$.

Proof. See Hill, page 5. □

Definition 2. *The* length *(n) of a code is the number of digits in each codeword.*

All measurements of distance refer to the Hamming Distance unless otherwise specified. The best codes are those with large distances between all their codewords. This way, if Bob receives an invalid codeword it usually much closer to the codeword Alice sent than to any other, incorrect codeword. A very simplistic example is to consider a code C with two codewords, $\mathbf{x} = 1200$ and $\mathbf{y} = 1201$. Since the codewords differ only in the last place, $d(\mathbf{x}, \mathbf{y}) = 1$. Consider one digit of $\mathbf{x}$ being corrupted in transmission, producing $\mathbf{x}' = 1202$. Notice now that $d(\mathbf{x}, \mathbf{x}') = d(\mathbf{y}, \mathbf{x}')$ and so the corrupted transmission is the same distance from the un-corrupted transmission as it is from another codeword in C . This is a poor code because it has two codewords which are only one unit away from each other.

In order to create the best possible code, we want C to have many codewords, and have a large distance between codewords to prevent errors, but with length short enough to be easily manageable.

Definition 3. *The minimum distance of a code is the Hamming distance between the two closest codewords.*

We seek to optimize the quantities

$$(n, M, d)$$

where n is the length of a code containing M codewords and having minimum distance d. That is, we want to maximize M while minimizing n.

2.2. **Permutations.**

Definition 4. *A permutation is a one-to-one mapping from a set S onto itself.*

Definition 5. *Two codes are equivalent if one can be obtained from the other by permutations on positions of the code or permutations of the symbols appearing in a fixed position.*

One of the most important results stemming from Definition 5 is the following theorem:

Theorem 4. *Any (n, M, d)-code on the alphabet of size q is equivalent to another (n, M, d)-code on the same alphabet which contains the zero vector.*

Proof. Assume a code of length n where all the codewords are of the form $x_1x_2x_3 \ldots x_n$. Choose a codeword and a $x_i \neq 0$ which appears in the codeword. Perform the mapping

$$0 \mapsto x_i$$
$$x_i \mapsto 0$$
$$j \mapsto j$$

for each nonzero x_i in the chosen codeword. This mapping is also given through the diagram

$$\begin{pmatrix} 0 & x_i & j & \\ \downarrow & \downarrow & \downarrow & \forall j \neq 0, x_i \\ x_i & 0 & j & \end{pmatrix}$$

We repeat the permutation choosing different all the nonzero values for x_i in our chosen codeword. Then the codeword has been permuted to be $\mathbf{0}$ and so the original code is equivalent to a code which contains $\mathbf{0}$. □

Example 1. *Consider a code of length* 4*:*

$$C = \begin{cases} 1010 \\ 0110 \\ 0101 \end{cases}.$$

We apply a permutation on the first column, sending 1 *to* 0 *and* 0 *to* 1*, resulting in*

$$C' = \begin{cases} 0010 \\ 1110 \\ 1101 \end{cases}.$$

Next we apply the same permutation to the third column, resulting in

$$C'' = \begin{cases} 0000 \\ 1100 \\ 1111 \end{cases}$$

and thus C is equivalent to a code containing the zero vector.

Definition 6. *$A_q(n, d)$ is the largest value of M such that there exists a q-ary (n, M, d)-code. $A_q(n, d)$ counts the maximum number of codewords available of length n, separated by a minimum Hamming Distance of d.*

Example 2. *The following code C_1 is a $(3, 4, 3)$-code over $(F_4)^3$:*

$$C_1 = \begin{cases} 000 \\ 111 \\ 222 \\ 333 \end{cases}$$

while C_2 is an example of a binary $(3, 4, 2)$-code:

$$C_2 = \begin{cases} 000 \\ 101 \\ 110 \\ 011 \end{cases}.$$

The set of all possible binary codewords of length 3 *is*

$$C_3 = \begin{cases} 000 \\ 001 \\ 010 \\ 011 \\ 100 \\ 101 \\ 110 \\ 111 \end{cases}.$$

It is easy to see that C_3 is all of $(F_2)^3$ and so we have found a $(3, 8, 1)$ code.

3. Finding Codes of Optimal Size

Speed is the biggest issue in finding large sets of codewords for any given n and d on the alphabet F_q. An exhaustive search algorithm is not efficient. Cycling through all the possible codewords in $(F_q)^n$ involves comparing q^n codewords to every other codeword in the code. Testing every possible combination of codewords for every possible value for M gives

$$s = \sum_{i=1}^{d^n} \binom{d^n}{i} \tag{5}$$

where s is the number of possible ways to choose a subset of codewords of in $(F_q)^n$. When considering $A_2(9,3)$, we quickly calculate that $s = 10^{154}$. As the values for n and d increase, this already huge number grows quickly, making an exhaustive search impractical.

3.1. **Finding Values for $A_2(n,d)$.** In order to determine the number of binary codewords of length n which are d units apart, we turn to a greedy algorithm. Because of permutations and equivalence of codes, given in Theorem 4, we can assume that the zero vector is in our code. From there, we cycle through every other possible codeword. The distance between each potential codeword and every other codeword in the code is computed, and, if the potential codeword is of a distance greater than or equal to d away from every other codeword, it is added to the code. This method of selecting a set of codewords allows us to choose a portion of all possible codewords, but does not, as we will see, find the optimal number in every case.

3.2. **Results from Raymond Hill's "First Course in Coding Theory".** In Raymond Hill's *First Course in Coding Theory*, he provides a table containing values of $A_q(n,d)$ for small of q, n, and d. We compare values found in the binary case using the greedy algorithm to those found in Hill. Part of Hill's table is shown in Table 1, for codewords of lengths 5 through 11. For example, when the minimum distance between codewords is $d = 5$, $n = 8$ and $A_q(n,d) = 4$.

n	d=3	d=5	d=7
5	4	2	-
6	8	2	-
7	16	2	2
8	20	4	2
9	40	6	2
10	72-79	12	2
11	144-158	24	4

Table 1. Hill's table of values for a binary $A_2(n,d)$ for small values of n and d.

The results using my greedy algorithm are shown in Table 2. The MATLAB code used to generate Table 2 can be found in Appendix A.

Because these some of these values are less than the values found in Hill, application of the greedy algorithm clearly does not find the optimal size code in these

cases. We have, however, generated a list of codewords belonging to this smaller set.

3.3. **Exhaustive search of possible $\mathbf{A_2(n, d)}$.** Let s be the number of possible subsets in $(F_2)^n$. Then from Equation 5 the number of possible subsets of codewords of size k is

$$s = \sum_{k=1}^{2^n} \binom{2^n}{k}. \tag{6}$$

Values of s for small values of n are shown in table 3.

From the tables 1 and 2, note that $n = 8$ is the point at which our lower bound for $A_2(n, d)$ computed through the greedy algorithm, $A_2(8, 3)' = 16$, first differs from the known value of $A_2(8, 3) = 20$. From the data above, an exhaustive search of all possible subsets is computationally impossible, from the magnitude of comparisons required. We remember that, by Theorem 4, every code is equivalent to a code containing the **0** codeword. Assuming that **0** exists inside the code reduces the number of possible codewords by 1. For a binary code with $n = 8$, the number of possible subsets of codewords becomes

$$s = \sum_{k=1}^{2^n-1} \binom{2^n - 1}{k} = 5.79e76$$

which is a negligible improvement over $s = 1.16e77$.

n	d=3	d=5	d=7
5	4	2	-
6	8	2	-
7	16	2	2
8	16	4	2
9	32	4	2
10	64	8	2
11	128	16	4

TABLE 2. Lower bounds for $A_2(n, d)$ found using the greedy algorithm. Note the differences between some of the values and their respective values in Hill's table.

n	s
2	16
3	256
4	65536
5	42e9
6	1.84e19
7	3.4e38
8	1.16e77

TABLE 3. The number of possible subsets of $(F_2)^n$ as a function of n.

We know from Hill's chart that $A_2(8,3) = 20$. Therefore, if we are looking for a list of the codewords in $(8, 20, 3)$ we only have to consider sets smaller than the upper bound in order to find one possibility for the set of codewords in $A_2(8,3)$. We can use Hill's upper bound as a limit in order to find the exact codewords in a code of optimal size:

$$s = \sum_{k=1}^{19} \binom{2^n - 1}{k} = 2.38e28.$$

We must consider, however, that this is merely the quantity of sets and not the number of comparisons required. This number of comparisons is

$$C = \sum_{k=1}^{2^8-1} \binom{2^8 - 1}{k} \frac{k(k+1)}{2} = 4.48e30$$

when $n = 8$. Performing this many operations is far beyond our capabilities, we must, therefore, consider alternate methods of computing $A_q(n, d)$.

4. Error Correction and the Sphere Packing Bound

Suppose an error occurs in a codeword during transmission. If the codeword received is not in the code, the receiver can instantly tell that an error has occurred, and has detected the error. If the error in the codeword turns one codeword into other codeword, error detection is impossible. We have two questions to consider upon receiving a codeword. Can we tell if the codeword is valid, and can we fix the codeword if it is invalid? Let an "error" be one digit in the codeword that is wrong. The following theorem, which we will state without proof, shows how many errors we can correct in messages of different lengths:

Theorem 5. Detection and Correction

(1) *A code C can detect up to s errors in any codeword if $d(C) \geq s + 1$.*
(2) *A code C can correct up to s errors in any codeword if $d(C) \geq 2s + 1$.*

Proof. See Hill, page 7. □

One method used to find upper bounds for the sizes of codes is the *sphere packing bound*. Visualize the codewords as points centered in spheres of radius t on a plane. The metric on this plane is the Hamming Distance. Each codeword is the center of a sphere such that all elements of $(F_q)^n$ which are less than t units away from

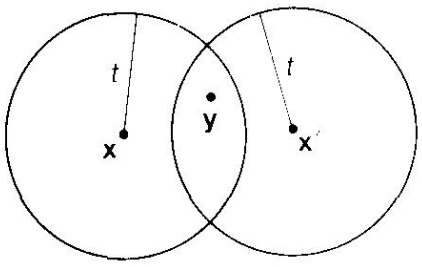

Figure 1. Overlapping spheres in a poor code [8]

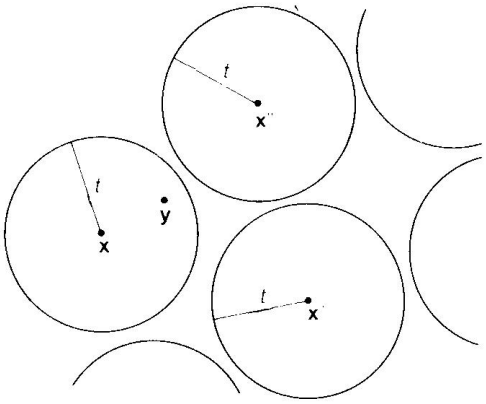

FIGURE 2. Good codes have spheres that do not overlap [8]

the codeword are contained in the sphere. Any distorted message $\mathbf{x}'$ of $\mathbf{x}$ which lies within t units of $\mathbf{x}$ can be mapped back onto $\mathbf{x}$, since it is obviously closer to $\mathbf{x}$ than to any other codeword. Messages which do not fall into one of these circles are indecipherable (see Figure 1, where the spheres are simplified to two dimensions). If any circles overlap we have a poor code, since a distorted codeword $\mathbf{x}'$ appearing in the intersection of multiple spheres could map back onto any of two or more points.

Since we can correct up to t errors, by Theorem 5, this sphere is of radius t. The two spheres in Figure 1 are too close together to be useful, since $\mathbf{y}$ appears in the spheres centered at both $\mathbf{x}$ and $\mathbf{x}'$. We are looking for our spheres to be "packed" closely in space without touching, as in Figure 2. No two spheres overlap, but the space not enclosed in a circle is minimized. The idea of perfect codes, codes in which every distorted codeword is within r units of a codeword, is described in Section 5. We wish to determine the maximum number of spheres that can be packed into our codespace, $(F_q)^n$. This gives us a reasonable upper bound for how many codewords are in any code, although it will not give us the actual maximum.

In order to determine this bound, we determine how many codewords exist inside each sphere of radius t. To find the number of vectors in each sphere we consider vectors at distances of $d = 0, 1, 2, \ldots, t$. Working out from the codeword at the center of each sphere, there is $\binom{n}{0} = 1$ vector at 0 distance, the codeword itself. There are $\binom{n}{1} = n$ vectors at distance 1 in a binary code. For larger alphabets, there are $\binom{n}{1}(q-1)$, since there are now $q-1$ ways for an entry to be different, instead of just one. Building a pattern, we can determine that the upper bound for a q-ary $(n, M, 2t+1)$ code. If there are M codewords in our code, then there are M spheres each containing

$$\binom{n}{0} + \binom{n}{1}(q-1) + \ldots + \binom{n}{t}(q-1)^t$$

codewords. Since there are only q^n possible codewords we get

$$M\left[\binom{n}{0} + \binom{n}{1}(q-1) + \ldots + \binom{n}{t}(q-1)^t\right] \leq q^n$$

which solved for M gives

$$M \leq \frac{q^n}{\sum_{x=0}^{t} \binom{n}{x}(q-1)^x}. \tag{7}$$

Inequality 7 is commonly referred to as the sphere packing bound.

Table 4 shows sphere-packing upper bounds for several lengths of codes and Hamming distances in a binary code (over F_2).

n	d=3	d=5	d=7	d=9
4	3	1	1	1
5	5	2	1	1
6	9	2	1	1
7	16	4	2	1
8	28	6	2	1
9	51	11	3	2
10	93	18	5	2
11	170	30	8	3
12	315	51	13	5

TABLE 4. Length upper bounds of binary codes found using Equation 7

Table 5 shows upper bounds for codes over F_3:

n	d=3	d=5	d=7
4	16	7	5
5	40	15	9
6	104	33	17
7	273	75	34
8	729	177	70
9	1968	427	151
10	5368	1054	335
11	14762	2643	763
12	40880	6727	1777

TABLE 5. Length upper bounds of tercery codes found using the greedy algorithm

5. Perfect Codes

When the upper bound in (7) is an equality (M is as large as possible) we say the code *satisfies the sphere packing bound.* Codes that satisfy the sphere packing bound are known as perfect codes. For example, the binary code with $d = 3$ and $n = 7$ has a sphere packing bound of 16, from Table 4. We can easily find a $(7, 16, 3)$

code using the greedy algorithm:

$$C = \begin{cases} 0000000 \\ 0000111 \\ 0011001 \\ 0011110 \\ 0101010 \\ 0101101 \\ 0110011 \\ 0110100 \\ 1001011 \\ 1001100 \\ 1010010 \\ 1010101 \\ 1100001 \\ 1100110 \\ 1111000 \\ 1111111 \end{cases}$$

This the $(7, 16, 3)$ code C is perfect because it contains 16 codewords and thus satisfies the sphere packing bound. We can envision this as 16 spheres of radius 1 filling up all of $(F_2)^7$. These spheres partition the set. Consider the binary case in which $d = 1$. Since $t = 0$, we know $M = q^n$. Since $\binom{n}{0} = 1$, this bound can be satisfied for any q and n, and the perfect code in this case is all of $(F_q)^n$. There is another trivial binary case in which $n = d$. Then

$$M = \frac{q^{2t+1}}{\sum_{x=0}^{t} \binom{2t+1}{x}(q-1)^x}.$$

Since

$$\binom{2t+1}{0} + \binom{2t+1}{1} + \ldots + \binom{2t+1}{t} = 2^{2t},$$

$$M = \frac{2^{2t+1}}{2^{2t}} = 2.$$

The code is therefore just

$$\{\underbrace{00\ldots0}_{n}, \underbrace{11\ldots1}_{n}\}.$$

We now consider some more interesting perfect codes besides the trivial cases.

Theorem 6. *Let q be a prime. q-ary codes which satisfy the sphere packing bound have size q^r, where $r \in \mathbb{Z}^+$.*

Proof. Assume we have a q-ary code which satisfies the sphere packing bound. We want

$$M = \frac{q^{2t+1}}{\sum_{x=0}^{t} \binom{2t+1}{x}(q-1)^x}$$

for our perfect code, we deduce that $\sum_{x=0}^{t} \binom{2t+1}{x}(q-1)^x$ must be a power of q and thus M must also be a power of q. Thus all codes which satisfy the sphere-packing bound have q^r elements for some integer r. □

Example 3. *Can there be perfect binary code of length* 7 *and distance* 3*? Consider the sphere packing bound for binary codes:*

$$M = \frac{2^n}{\sum_{x=0}^{t} \binom{n}{x}}.$$

Thus

$$M = \frac{2^7}{\binom{7}{0} + \binom{7}{1}} = \frac{128}{8} = 16.$$

Since $16 = 2^4$*, it is possible that there exists a perfect binary code. This is the perfect code we found earlier.*

Example 4. *Can there be a perfect binary code of length* 10 *and distance* 5*? We need* M *to be a power of* 2*:*

$$M = \frac{2^{10}}{\binom{10}{0} + \binom{10}{1} + \binom{10}{2}} = \frac{2^{10}}{56}.$$

Since M *is not an integer, there cannot exist a perfect code of length* 10 *and minimum distance* 5*.*

6. Finite Fields

For our purposes, the exact properties of fields are less important than understanding the general idea of a field and how it relates to the alphabet of a code.

Theorem 7. *A field* F *is a set of elements with two operations* $+$ *(called addition) and* $\cdot$ *(called multiplication) satisfying the following properties.*

1. F *is closed under* $\cdot$ *and* $+$*.*
2. *If* $a, b \in F$*,* $ab = ba$ *and* $a + b = b + a$ *(*F *is commutative under addition and multiplication.*
3. *If* $a, b, c \in F$*,* $a + (b + c) = (a + b) + c$ *and* $a(bc) = (ab)c$ *(*F *is associative under addition and multiplication).*
4. *For* $a, b, c \in F$ $a \cdot (b+c) = a \cdot b + a \cdot c$ *(multiplication distributes over addition in* F*.)*
5. *There exists* $0 \in F$ *such that* $a + 0 = a$ *for all* $a \in F$*.*
6. *There exists* $1 \in F$ *such that* $a \cdot 1 = a$ *for all* $a \in F$*.*
7. *If* $a \in F$*, there exists* $-a \in F$ *such that* $a + (-a) = 0$*.*
8. *If* $a \in F$ *and* $a \neq 0$ *there exists* $a^{-1} \in F$ *such that* $aa^{-1} = 1$*.*

In order to simplify notation and prevent redundancy we will make several assumptions. When q is a prime, $F_q = \{0, 1, 2, , 3, \ldots, q-1\}$ is a field with operations addition and multiplication modulo q. We will primarily consider codes on a binary alphabet. Therefore, unless otherwise specified, $q = 2$.

6.1. Vector Spaces Over Finite Fields .

Before we can begin our discussion of the properties of vector spaces we must first define the two operations, element-wise addition and scalar multiplication. Assume $\mathbf{x}, \mathbf{y} \in (F_q)^n$. If $\mathbf{x} = x_1 x_2 \ldots x_n$ and $\mathbf{y} = y_1 y_2 \ldots y_n$, then

$$\mathbf{x} + \mathbf{y} = (x_1 + y_1, x_2 + y_2, \ldots, x_n + y_n).$$

Additionally, if $a \in F_q$ then $a\mathbf{x} = (ax_1, ax_2, \ldots, ax_n)$ It is easy to prove the following properties which make $(F_q)^n$ a vector space, although we will omit the proofs here:

1. $(F_q)^n$ is closed under addition mod n.

(2) $(F_q)^n$ is associative under addition mod n.
(3) The 0 vector is an additive identity.
(4) There is an additive inverse for every element in $(F_q)^n$.
(5) If $a, b \in (F_q)^n$, then $a + b = b + a$ (commutativity).
(6) If $a \in (F_q)^n$ and $c \in F_q$ then $ac \in (F_q)^n$.
(7) Distributive laws hold with elements in $(F_q)^n$ and scalars in F_q.
(8) Multiplication is associative with elements in $(F_q)^n$ and scalars in F_q.
(9) The multiplicative identity in F_q is also the multiplicative identity in $(F_q)^n$.

So $(F_q)^n$ is a vector space composed of codeword vectors of length n under addition modulo q and codes are a subset of $(F_q)^n$.

7. Linear Codes

One type of code which appears frequently throughout coding theory is the linear code. To understand linear codes we recall that codewords are vectors living in a vector space. So $(F_q)^n$ is a vector space composed of vectors of length n under addition modulo q and the codes are just subsets of $(F_q)^n$. If there exists some codewords $k_1, k_2, \ldots, k_i \in (F_q)^n$ such that $C =$Span $\{k_1, \ldots, k_i\}$ then C is a subspace of the vector space $(F_q)^n$ and thus is a linear code.

Example 5. *Is the code*

$$C_1 = \begin{cases} 111 \\ 101 \\ 001 \end{cases}$$

linear? Since every vector added to itself in binary is $\mathbf{0}$*, every spanning set includes* $\mathbf{0}$ *and so the zero vector must be in all linear codes. Since* $\mathbf{0} \notin C_1$*,* C_1 *is not a linear code.*

Example 6. *Let*

$$C_2 = \begin{cases} 000 \\ 001 \\ 011 \\ 010 \end{cases}$$

Is C_2 *a linear code? There are multiple possible spanning sets, including* $\{000, 001, 011, 010\}$*,* $\{001, 011\}$*,* $\{000, 001, 011\}$*, and* $\{001, 011, 010\}$*. Therefore* C_2 *is a linear code.*

In Example 6 there are multiple spanning sets (not all the same size) for C_2. The smallest possible spanning set (the basis for C_2) is of size 2, the basis can be either $\{001, 011\}$, $\{001, 010\}$, or $\{011, 010\}$, since we can easily determine that each of these pairs of codewords spans C_2.

Definition 7. *Let* C *be a linear code. If the rows (vectors) of a matrix* G *form a basis for* C *then* G *is a* generator matrix *for* C*.*

To find a generating matrix for a code C, we use standard linear algebra techniques to find a basis for C and use these vectors as the rows of G.

Example 7. [8] *Find* C*, the binary code having the generator matrix*

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

To determine the set of codewords which is spanned by the rows of G we take every possible linear combination (using coefficients in $F_2 = \{0, 1\}$) of the rows of G.

$$C = \left\{ \begin{array}{ccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{array} \right.$$

Example 8. *Suppose C is the tertiary code generated by*

$$G = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \end{bmatrix}.$$

The codewords in C are every possible linear combination of the codewords in the generator matrix using $F_3 = \{0, 1, 2\}$ as coefficients. They correspond to

$$C = \left\{ \begin{array}{c} 0 \\ g_1 \\ g_2 \\ 2g_1 \\ 2g_2 \\ 2g_1 + g_2 \\ g_1 + 2g_2 \\ g_1 + g_2 \\ 2g_1 + 2g_2 \end{array} \right.$$

where g_1 and g_2 are the rows of G. Thus the codewords in C are

$$C = \left\{ \begin{array}{l} 0000 \\ 1011 \\ 0112 \\ 2022 \\ 0221 \\ 2101 \\ 1202 \\ 1120 \\ 2210 \end{array} \right. .$$

We know C is a linear code because it is the spanning set of the rows of G. Since the minimum distance in a linear code is the same as the minimum weight of non-zero vectors in the code, $d = 3$. Since $d = 3$, we know $t = 1$ from Theorem 5. In combination with Equation 7, we can determine that the sphere-packing bound gives

$$M \leq \frac{3^4}{\sum_{x=0}^{1} \binom{4}{x}(3-1)^x}$$

$$M \leq \frac{81}{\binom{4}{0}(2)^0 + \binom{4}{1}(2)^1}$$

$$M \leq \frac{81}{1+8} = 9.$$

Since the sphere packing bound $M = 9$ and $|C| = 9$ coincide, this code is perfect as well as linear.

8. Encoding

Suppose we have a linear code $C \subseteq (F_q)^n$ with a generator matrix G. What does this have to do with transmitting codes? G allows us to convert messages of length k to messages of length n. (Since the rows of G are linearly independent, we are guaranteed that $k \leq n$.) These extra digits are the redundancy in the encryption and serve as the error detection and correction.

Since C is linear we know there are q^k codewords in the code, where k is the number of rows in the generator matrix (k is also commonly referred to as the *dimension* of the code). Consider codewords of of length k, rather than length n. We define encoding a codeword vector $\mathbf{x}$ using C as right multiplication by G; the encoded message is $\mathbf{y} = \mathbf{x}G$. This gives us a linear combination of the rows of G with elements of $(F_q)^n$ as weights. Since this is the definition of elements of C, we know $\mathbf{y} \in C$.

Any codeword in $(F_q)^k$ can be encoded using the linear code, that is, the message is mapped to a codeword in C.

Example 9. *Suppose C is generated by*

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

To encode the message 0000 *we multiply*

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

and so we get 0000000 *while encoding* 0011 *gives us*

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

8.1. **Dual Codes.** The dual of C, denoted $C^\perp$, is the set of all vectors in $(F_q)^n$ that are orthogonal to every codeword in C. In symbols,

$$C^\perp = \{\mathbf{v} \in V(n, q) | \mathbf{v} \cdot \mathbf{u} = 0 \ \forall \, \mathbf{u} \in C\}$$

Example 10. *Here is an example of a code (C_1) and its dual code ($C^\perp$):*

$$C_1 = \begin{Bmatrix} 100 \\ 011 \\ 110 \end{Bmatrix},$$

$$C_1^\perp = \{\ 000\ .$$

Additionally,

$$C_2 = \begin{cases} 100 \\ 110 \\ 010 \end{cases}$$

and

$$C_2^{\perp} = \begin{cases} 000 \\ 001 \end{cases}$$

form a code/dual pair. If

$$C_3 = \begin{cases} 1001 \\ 0110 \\ 0000 \end{cases}$$

then

$$C_3^{\perp} = \begin{cases} 1111 \\ 1001 \\ 0110 \\ 0000 \end{cases}.$$

It is interesting to note that in this case, $C_3 \subset C_3^{\perp}$.

8.2. **Parity Check Matrices.**

Definition 8. *The parity check matrix for C is a matrix H such that the columns of H form a basis for $C^{\perp}$. In symbols, $C = \{\mathbf{x} \in V(n,q) |\, \mathbf{x}H^T = \mathbf{0}\}$ where H is the parity check matrix.*

9. Cyclic Codes

Recall from basic number theory that the division algorithm for integers states that given any two integers a and $b \geq 0$ there exists integers r and q such that

$$a = bq + r$$

with $0 \leq r < b$. The following theorem extends the idea of division algorithm to polynomials. Let us define $F_q[x]$ as the set of polynomials in x with coefficients in F_q.

Theorem 8. [10] *For any $g(x)$ and nonzero $f(x)$ in $F_q[x]$ there exist unique polynomials $q(x)$ and $r(x)$ such that*

$$g(x) = q(x)f(x) + r(x)$$

where deg $r(x)$ <deg $f(x)$.

Proof. See Appendix B. □

Using the polynomial division algorithm we can extend our idea of modular arithmetic to polynomials. We say polynomials $a(x)$ and $b(x)$ are congruent modulo $q(x)$ if $q(x)|(a(x) - b(x))$. That is, if $r(x) = 0$ through the division algorithm.

Definition 9. *If $f(x)$ is not the product of some two nonconstant polynomials in $F_q[x]$, then the set of polynomials in $F_q[x]$ of degree less than deg $f(x)$ is a field under addition and multiplication modulo $f(x)$. We denote this field as R_n.*

Theorem 9. *A code C is cyclic if it satisfies the following properties:*

(1) *C is linear.*

(2) *Any cyclic shift of a codeword is also a codeword (that is, whenever* $a_0a_1...a_{n-1} \in C$*,* $a_{n-1}a_0a_1...a_{n-2} \in C$*.)*

Theorem 10. *Let* C *be the set of vectors formed by taking the coefficients from a set of polynomials,* P *over* R_n*. If* P *satisfies:*

(1) $a(x), b(x) \in C$ *implies* $a(x) + b(x) \in C$*;*
(2) $a(x) \in C$ *and* $r(x) \in R_n$ *implies* $r(x)a(x) \in C$

then C *is a cyclic code.*

Note that property 2 of Theorem 10 is *not* the same as C being closed under multiplication. Property 2 requires that the product of an element in C with any element in R_n is still in C.

9.1. **Generating Cyclic Codes.** Let us denote

$$< f(x) >= \{r(x)f(x)|r(x) \in R_n\}.$$

Theorem 11. *For any* $f(x) \in R_n$*, the set* $< f(x) >$ *is the cyclic code generated by* $f(x)$*.*

Proof. Assuming $a(x), b(x) \in R_n$, this produces the following two implications: If $a(x)f(x) + b(x)f(x) \in < f(x) >$ then

$$a(x)f(x) + b(x)f(x) = (a(x) + b(x))f(x) \in < f(x) > .$$

Additionally, if $a(x)f(x) \in < f(x) >$ and $r(x) \in R_n$ then

$$r(x)(a(x)f(x)) = (r(x)a(x))f(x) \in < f(x) > .$$

Thus the set satisfies the two requirements stipulated in Theorem 3, and so $< f(x) >$ is a cyclic code. □

Example 11. *Consider the code* $C =< 1 + x^2 >$ *in the field* R_n*. Multiplying each of the elements in* $R_3 = \{0, 1, 1 + x, x, 1 + x + x^2, x + x^2, 1 + x^2, x^2\}$ *by* $1 + x^2$ *and reducing modulo* $x^3 - 1$ *yields the set* $\{0, 1 + x, 1 + x^2, x + x^2\}$*. Thus the set* $C = \{000, 110, 101, 011\}$*, formed using the coefficients of the polynomials, is cyclic.*

9.2. **Hamming Code.**

Definition 10. *A code* G *is a* Hamming Code *if the rows of its parity check matrix* H *are* $(F_q)^n \setminus \mathbf{0}$*.*

Then any Hamming Code G is a perfect code which can correct single errors and detect two errors, by Theorem 5. The code generated by

$$G = \begin{bmatrix} 1000111 \\ 0100110 \\ 0010101 \\ 0001011 \end{bmatrix}$$

is a Hamming Code because its parity check matrix H is all of $(F_2)^3$:

$$H = \begin{bmatrix} 111 \\ 110 \\ 101 \\ 011 \\ 100 \\ 010 \\ 001 \end{bmatrix}, .$$

which can be used to generate the entire code C:

$$C = \left\{ \begin{array}{l} 0000000 \\ 1000111 \\ 0100110 \\ 0010101 \\ 0001011 \\ 1100001 \\ 1010010 \\ 1001100 \\ 1110100 \\ 1101010 \\ 1011001 \\ 1111111 \\ 0110011 \\ 0101101 \\ 0111000 \\ 0011110 \end{array} \right.$$

10. Fixed Weight Codes

Recall from Definition 6 that $A_2(n, d)$ is the maximum number of codewords of length n where every codeword is at least d distance away from every other codeword. All codewords are binary.

Definition 11. *Let us define the* weight *of a binary codeword to be the number of* 1*s in the codeword. The notation $A(n, w, d)$ is the maximum number of codewords in F_q which are all at least d distance apart and of weight w.*

We can extend this notation and let $A(n_1, w_1, n_2, w_2, d)$ be the maximum number of binary codewords that can exist when, in addition to all being at least d units apart, the first n_1 digits have weight w_1 and the next n_2 digits have weight w_2. The length of the code is $n_1 + n_2$.

10.1. **Finding fixed-weight codes.** In an attempt to determine the correct value for $A_2(8, 3)$, we again use the greedy algorithm, this time limiting our search to vectors of weight 3. That is, for all $\mathbf{x} \in C$, where C is our code, $w(\mathbf{x}) = 3$. The greedy algorithm requires an initial vector. We enter a variety of initial vectors into the MATLAB script, but each time the result in a code of only 11 codewords, like C:

$$C = \left\{ \begin{array}{l} 11010001 \\ 00001111 \\ 00110011 \\ 00111100 \\ 01010110 \\ 01100101 \\ 01101010 \\ 10011010 \\ 10100110 \\ 10101001 \\ 11001100 \end{array} \right.$$

Since this is far less than the 16 codewords we found using the regular greedy algorithm, we will discard using the greedy algorithm as a device to find codes of weight 3 which maximize $A_2(n, w, d)$. Perhaps the upper bound for $A_2(n, w, d)$ is considerably smaller than the upper bound for $A_2(n, d)$. This would be an interesting topic for the reader to explore further.

11. Conclusion

We use ISBN number to introduce the topic of error correcting codes, and then investigate methods for determining optimal sizes for these codes. Defining vector spaces over finite fields allows us to determine how many errors codes are culpable of correcting. We also briefly touched on many different kinds of error correcting codes, perfect codes, linear codes, cyclic codes, dual codes, Hamming codes, and fixed weight codes, giving definitions and showing how to find them and decrypt messages. This paper only scratches the surface of coding theory. Polynomials play a disproportionally small role in this paper in relation to the role they play in coding theory as a whole. The interested reader might explore polynomials in codes other than cyclic codes (see Hoffman et al 1992). Some knowledge of abstract algebra is required, but a cursory exploration, like that provided by Hill, deals only with the aspects of algebra necessary to understand basic coding theory (see Hill 1986). An interesting example of a code which uses polynomials but is not cyclic is the Reed-Solomon codes (see Hoffman et al 1992 p. 139-170).

One might expand on the theory in other ways, such as using weighted codes to determine upper bounds for codes (see Best, et al. 1978 p. 81-92). Coding theory techniques and ideas can also be used in cryptography and data compression.

Alternately, if the reader is more interested in applications, he or she might explore the uses of coding theory in real life applications like music on compact discs (see Hoffman, et al 1992 p. 182-184, p. 249-252) or numbers on driver's licenses (see Gallian 1996 p. 504-517).

References

[1] Best, M.R. et al. "Bounds for Binary Codes of Length Less Than 25." 1978, 81-92.

[2] Blum, Manuel, Michael Luby and Ronitt Rubinfeld. "Self-Testing/Correcting with Applications to Numerical Analysis." Journal of Computer and System Sciences. Volume 47, Issue 3 (December 1993), 73-83.

[3] Connor, Stever. "The invisible border guard." New Scientist, 5 January 1984, 9-14.

[4] Gallian, Joseph. "Assigning Driver's License Numbers". Mathematics Magazine, Vol. 64, No. 1, February 1991 13-22.

[5] Gallian, Joseph. "How Computers Can Read and Correct ID Numbers." Math Horizons, Winter 1993, 14-15.

[6] Gallian, Joseph. "The Zip Code Bar Code". UMAP Journal, 7 (1986) 191-195.

[7] Gallian, Joseph A and Steven Winters. "Modular Arithmetic in the Marketplace." The Teaching of Mathematics, v.95 n.6, p.549-551, June/July 1988.

[8] Hill, Raymond. A First Course in Coding Theory. New York: Oxford University Press, 1986.

[9] Hoffman, D.G. et al. Coding Theory: The Essentials. New York: Marcel Dekker, Inc., 1992.

[10] Stewart, Ian. Galois Theory. New York: Chapman & Hall/CRC, 2004.

[11] Tuchinsky, Philip M. "International Standard Book Numbers." The UMA Journal. 5, 1989, 41-54.

[12] Wood, Eric. "Self-Checking Codes - An Application of Modular Arithmetic". Mathematics Teacher, 80, 1987, 312-316

12. Appendix A: MATLAB Code for Greedy Algorithm

```
function numCodes=Aq()

n=5;
d=3;
q=10;

codes(1,1:n)=zeros(1,n);

%check=zeros(1,n);

for i=1:q^n-1

    [r,c]=size(codes);
    baseq=dec2base(i,q);
    x=str2num(baseq);
    comp=toArray(x,n)
    for j=1:r
        %distVect stores the distances of each vector to comp
        distVect(j)=hamming(comp,codes(j,1:n));
    end

    rep=1;
    for j=1:r
        if (distVect(j)<d)
            %rep=0 if this vector is too close to the other vectors
            rep=0;
        end

    end

    % adds codeword to the list if it is sufficiently far away
    if (rep==1)
        codes(r+1,1:n)=comp;
    end

end

codes

size(codes)

end
```

```
%determines the Hamming Distance between vectors a and b
function y=hamming(a,b)

%makes the vectors the same length, adding 0s to the shorter of the two
%vectors
%aArray=toArray(a);
%bArray=toArray(b);

aArray=a;
bArray=b;

la=length(aArray);
lb=length(bArray);

if (la<lb)
   aArray(lb-la+1:lb)=aArray(1:la);
   aArray(1:lb-la)=0;
end

if (la>lb)
   bArray(la-lb+1:la)=bArray(1:lb);
   bArray(1:la-lb)=0;
end

% aArray and bArray are now the same length
ln=length(bArray);

count=0;

for i=1:ln
   if (aArray(i)~=bArray(i))
       count=count+1;
   end
end

y=count;

end
```

13. Appendix B: Proof of Theorem 8

Theorem 8: For any $g(x)$ and nonzero $f(x)$ in $F_q[x]$ there exist unique polynomials $q(x)$ and $r(x)$ such that

$$g(x) = q(x)f(x) + r(x)$$

where deg $r(x)$ <deg $f(x)$.

Proof. The existence proof is by induction on the degree of $g(x)$.

Base Case: If deg $g(x) = -\infty$ then $g(x) = 0$ and so $q(x) = r(x) = 0$. If deg $g(x) = 0$ then $g(x) = r(x)$ and $q(x) = 0$. If deg $f(x) = 0$ then $f(x) = g(x)$. Else, deg $f(x) > 0$ and thus $r(x) = g(x)$ and $q(x) = 0$.

Induction Step: Assume the hypothesis is true for cases where deg $g(x) < n$. (The degree of $g(x)$ must be positive, 0, or $-\infty$ by the definition of a polynomial.) Let

$$g(x) = b_n x^n + \cdots + b_0$$
$$f(x) = a_m x^m + \cdots + a_0$$

where the leading coefficients are nonzero and deg $f(x) <$ deg $g(x)$. We can now introduce the polynomial $g_1(x)$:

$$g_1(x) = b_n a_m^{-1} x^{n-m} f(x) - g(x).$$

The term of highest degree in $g(x)$ cancels and thus deg $g_1(x) <$ deg $g(x)$. Then we can apply the induction hypothesis and write $g_1(x)$ in the following unique factorization:

$$g_1(x) = f(x)q_1(x) + r_1(x).$$

Now let

$$q = b_n a_m^{-1} x^{n-m} - q_1(x)$$

and

$$r = -r_1.$$

Then

$$g = f(x)q(x) + r = f(x)(b_n a_m^{-1} x^{n-m} - q_1(x)) - r_1$$

To prove uniqueness, suppose that a polynomial $g(x)$ has two different possible factorizations:

$$g(x) = f(x)q_1(x) + r_1(x)$$

and

$$g(x) = f(x)q_2(x) + r_2(x).$$

We subtract these polynomials to get

$$\begin{aligned} g(x) - g(x) &= f(x)q_1(x) + r_1(x) - f(x)q_2(x) - r_2(x) \\ 0 &= f(x)(q_1(x) - q_2(x)) + r_1(x) + r_2(x) \\ r_2(x) - r_1(x) &= f(x)(q_1(x) - q_2(x)) \end{aligned} \tag{8}$$

Since deg$f(x)(q_1(x) - q_2(x))$ >deg$r_2(x) - r_1(x)$, the degrees of both must be 0. Then $r_1(x) = r_2(x)$ and $q_1(x) = q_2(x)$ and the division algorithm produces unique polynomials.

□