The Coding Theory Workbook

A companion to EE 567

Version 1.1
Spring 2016

Matthew C. Valenti, Ph.D., P.E.
West Virginia University

Contents

Chapter 1

Finite Fields

1.1 Sets (Ryan and Lin Section 2.1)

A *set* is a collection of *elements*. The convention is to use uppercase letters to denote sets, such as X or Y. The elements of a set may be represented by lowercase letters, which could be subscripted, e.g. x_1 or y_1. A *finite set* has a finite number of elements, for instance:

$$\begin{aligned} X &= \{x_1, x_2\} \\ Y &= \{y_1, y_2, y_3\} \end{aligned}$$

are two finite sets. The *cardinality* of a finite set is the number of elements in the set. The cardinality of a set S is denoted $|S|$. For the above examples, determine $|X|$ and $|Y|$:

$$\begin{aligned} |X| &= \\ |Y| &= \end{aligned}$$

A set X is said to be a *subset* of another set Y if all the elements in X are also in Y. For instance, if

$$\begin{aligned} X &= \{4, 5\} \\ Y &= \{4, 5, 6\} \end{aligned}$$

then X is a subset of Y and we would express this relationship as $X \subseteq Y$. X is said to be a *proper* subset of Y if $|X| < |Y|$, and we indicate this by $X \subset Y$. Note that any set X is a subset of itself but not a proper subset of itself, therefore we can write $X \subseteq X$ but we cannot write $X \subset X$.

We are often interested in the following set operations:

Union: $X \cup Y$ contains all elements in X or Y (or both).

Intersection: $X \cap Y$ contains all elements in X and Y.

Difference: $Y \setminus X$ contains the elements in Y that are not in X.

Two sets are said to be *disjoint* if they have no elements in common, i.e. $X \cap Y = \varnothing$.

1.2 Binary Operations (Ryan and Lin Section 2.1)

A *binary operation* is a rule that assigns a value to a pair of elements called *operands*. The word *binary* means that there are exactly two operands. While a variety of symbols may be used for the operator, we will use the notation '$*$' to denote an arbitrary operator. Thus, we would write a binary operator as:

$$a * b \quad = \quad c$$

Properties: A binary operator defined over a set S may or may not satisfy the following properties:

Closure: for all $a, b \in S$, $a * b = c \in S$.

Associativity: for all $a, b, c \in S$, $(a * b) * c = a * (b * c)$.

Commutativity: for all $a, b \in S$, $a * b = b * a$.

Examples: Some typical binary operations:

1. Arithmetic operations on scalars: Addition, multiplication, subtraction, division.
2. Exponentiation: a^b.
3. Matrix addition; matrix multiplication.
4. Modulo-m addition $a \boxplus b$; modulo-m and multiplication $a \boxdot b$.[1]
5. Logicals: AND, OR, XOR, NOR, NAND.

1.3 Groups (Ryan and Lin Section 2.2)

A *group* is an algebraic system consisting of a set G on which a binary operation $*$ is defined with properties:

1. *Closure:* if $a, b \in G$ then $a * b = c \in G$.
2. *Associativity:* for all $a, b, c \in G$, $(a * b) * c = a * (b * c)$.
3. *Identity:* there exists a unique element $e \in G$ such that $a * e = e * a = a$.
4. *Inverse:* for all $a \in G$ there exists a unique element $a' \in G$ such that $a * a' = a' * a = e$.

A group is said to be *commutative* or *abelian* if it satisfies the above properties as well as:

5. *Commutativity:* for all $a, b \in G$, $a * b = b * a$.

The *order* of a group is the number of elements it contains, i.e. $|G|$, which is the *cardinality* of set G. If the order is finite, then the group is said to be finite.

[1] $a \boxplus b$ is defined as the remainder that results when (a+b) is divided by m. The result is contained in the set $\{0, ..., m-1\}$. Similarly, $a \boxdot b$ is the remainder of ab/m.

1.3.1 Examples

Which of the following are groups? Abelian groups?

1. The integers $G = \mathbb{Z}$ under addition?

2. The integers $G = \{0, 1, 2\}$ under addition?

3. The integers $G = \{0, 1, 2\}$ under modulo-3 addition?

4. The integers $G = \mathbb{Z}$ under multiplication?

5. The integers $G = \{0, 1, 2\}$ under modulo-3 multiplication?

6. The integers $G = \{1, 2\}$ under modulo-3 multiplication?

7. The integers $G = \{1, 2, 3\}$ under modulo-4 multiplication?

8. The set of N by N real-valued non-singular matrices under matrix multiplication?

9. The integer 0 under addition.

1.3.2 Modulo-m Addition

Theorem: The integers $G = \{0, ..., m-1\}$ form a finite abelian group of order m under modulo-m addition for any positive integer m. Such a group is an instance of an *additive group*.

Proof:

1. *Closure:* Assured by modularity, since $a \boxplus b \in G$.
2. *Associativity:* Follows from integer addition.
3. *Identity:* The identity element is $e = 0$.
4. *Inverse:* The inverse of a is $a' = m - a$.
5. *Commutativity:* Follows from integer addition.

Subtraction by a is defined to be the addition with $-a$, where $-a$ is the additive inverse of a.

1.3.3 Modulo-m Multiplication

Theorem: The integers $G = \{1, ..., m-1\}$ form a finite abelian group of order $m-1$ under modulo-m multiplication if and only if m is prime. Such a group is an instance of a *multiplicative group.*

Proof:

1. *Closure:* Because $0 \notin G$, modularity does not assure closure. G cannot be a field if there exists some $a, b \in G$ such that $a \boxdot b = 0$. This condition occurs iff m is not prime, because there will exist two positive integer factors $a, b < m$ such that $ab = m$ and therefore $a \boxdot b = 0$. If m is prime, then no such factors exist and the group is closed.

2. *Associativity:* Follows from integer multiplication.

3. *Identity:* The identity element is $e = 1$.

4. *Inverse:* If G is closed, then for each $a \in G$ its product with each $b \in G$ is distinct, i.e. $a \cdot b_i \neq a \cdot b_j$ if $b_i \neq b_j$. Otherwise $a \cdot b_i = a \cdot b_j$ implies $a \cdot (b_i - b_j) = 0$ and closure would not be satisfied. Therefore one of the b's must be the inverse.

5. *Commutativity:* Follows from integer multiplication.

Division by a is defined to be the multiplication with a^{-1}, where a^{-1} is the multiplicative inverse of a.

1.3.4 Cyclic Groups

A multiplicative group is said to be *cyclic* if there exists an element $a \in G$ such that every element $b \in G$ is some power of a, i.e. $b = a^i$ for an integer i. The element a is called a *generator* for the group. Note that exponentiation follows the rules of modulo-m multiplication, so $a^2 = a \boxdot a$ and $a^i = a \boxdot a^{i-1}$.

Example: Is the group $G = \{1, 2, 3, 4, 5, 6\}$ under modulo-7 multiplication cyclic? If so, determine a generator.

1.3.5 Subgroups

Let $H \subseteq G$, where G is a group under a binary operator '$*$'. H is said to be a *subgroup* if it is also a group under the same binary operator.
Example: Given $G = \{0, 1, 2, 3, 4, 5\}$ under modulo-6 addition. Identify four different subgroups of G.

1.3.6 Cosets

Let G be an abelian group under binary operator $*$. Let H be a subgroup of G. Pick an element $a \in G$. Define the set $a * H$ to be the set containing the results $a * h$ for all $h \in H$. The set $a * H$ is called a *coset* of H.

Example: Let $G = \{0, 1, 2, 3, 4, 5\}$ under modulo-6 addition and let $H = \{0, 3\}$. The coset $a \boxplus H$ is found by picking an element $a \in G$ and adding it (modulo-6) to each element $h \in H$. Determine all the distinct cosets of H.

Properties of cosets:

1. All cosets of H have the same number of elements.

2. The distinct cosets of H are disjoint.

3. Every element in G appears in exactly one distinct coset of H.

1.4 Fields (Ryan and Lin Section 2.3)

A *field* is a set F over which two operations are defined: "+" and "$\cdot$" with properties:

1. F forms an abelian group under "+" with (additive) identity element "0".
   The additive inverse of a is denoted $-a$.
2. $F\backslash\{0\}$ forms an abelian group under "$\cdot$" with (multiplicative) identity element "1".
   The multiplicative inverse of a is denoted a^{-1}.
3. "+" and "$\cdot$" distribute: $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$, for all $a, b, c \in F$.

The *order* of a field is the number of elements it contains, i.e. $|F|$. A field is *finite* if it has finite order.

1.4.1 Examples

Which of the following are fields? If the operation is not specified, assume that "+" is addition and "$\cdot$" is multiplication.

1. The real numbers $\mathbb{R}$?

2. The complex numbers $\mathbb{C}$?

3. The integers $\mathbb{Z}$?

4. The rational numbers?

5. The set of integers $\{0, ..., m-1\}$ under modulo-m addition and multiplication?

6. The set of integers $\{0, 1, 2, 3\}$ with addition and multiplication as defined below.

+	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

·	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

1.4.2 Subfields

Let F be a field under '$+$' and '$\cdot$'. Let $K \subseteq F$. Then K is a *subfield* of F if it is also a field under '$+$' and '$\cdot$'. If K is a subfield of F, then F is an *extension field* of K. If F has no proper subfields, then it is called a *prime field*.

Exercise: Identify subfield/extension field relationships in the preceding examples.

1.4.3 Galois Fields

Finite fields are called *Galois Fields* after Evariste Galois (1811-1832). The Galois field of order q is denoted $GF(q)$ for short. All finite fields of order q have the same mathematical properties up to the labelling of their elements, so the shorthand $GF(q)$ is not ambiguous.

One way to create a Galois field is to use the set of integers $\{0, ..., m-1\}$ where m is prime and $+$ and $\cdot$ are modulo-m addition and modulo-m multiplication, respectively. But is this the only way to create a Galois field?

We know that the integers $\{0, ..., m-1\}$ cannot form a field under modulo-m multiplication if m is not prime. However this does not mean that $GF(q)$ does not exist if q is not prime ... it just means we can't use modulo-m multiplication. We will soon see that it is possible to create $GF(q)$ as long as $q = p^m$ for p prime and m a positive integer. But this requires *extending* the integers to m-dimensional space just as the complex numbers are a 2-D extension of the real numbers. Before we can do this, we must establish some more properties of Galois Fields, which will require some intense discrete math.

Reed Solomon (RS) codes, which are perhaps the most commercially successful code ever devised, use Galois Fields of the form $GF(2^m)$ with m often equal to 8. So to understand RS codes, we must first understand Galois Fields.

However, there are many codes that are binary, i.e. defined over the binary field $GF(2) = \{1, 0\}$. The main concepts in this class are best explained with the simpler case of $GF(2)$. Once these principles are established, we can go back and see how things change for nonbinary codes, i.e those defined over the field $GF(p^m)$ where $p > 2$ and/or $m > 1$.

Chapter 2

Vector Spaces

2.1 Definition of a Vector Space (Ryan and Lin Section 2.4)

Let $\mathcal{V}$ be a set of elements called *vectors* on which addition "+" is defined. Let F be a field. Elements in F are called *scalars*. Multiplication "$\cdot$" is defined between elements in F and $\mathcal{V}$. The set $\mathcal{V}$ is a *vector space* over the field F if the following properties are satisfied:

1. $(\mathcal{V}, +)$ is an abelian group with identity $\mathbf{0}$.

2. *Scalar multiplication:* For any element $a \in F$ and $\mathbf{v} \in \mathcal{V}$, then $a \cdot \mathbf{v} \in \mathcal{V}$.

3. *Associative rule:* If $\mathbf{v} \in \mathcal{V}$ and $a, b \in F$, then

$$(a \cdot b) \cdot \mathbf{v} \quad = \quad a \cdot (b \cdot \mathbf{v}).$$

4. *Distributive rule #1:* If $\mathbf{u}, \mathbf{v} \in \mathcal{V}$ and $a \in F$, then

$$a \cdot (\mathbf{u} + \mathbf{v}) \quad = \quad a \cdot \mathbf{u} + a \cdot \mathbf{v}$$

5. *Distributive rule #2:* If $\mathbf{v} \in \mathcal{V}$ and $a, b \in F$, then

$$(a + b) \cdot \mathbf{v} \quad = \quad a \cdot \mathbf{v} + b \cdot \mathbf{v}.$$

6. Let "1" be the multiplicative identity (i.e. unit element) of F. Then for any $\mathbf{v} \in \mathcal{V}$,

$$1 \cdot \mathbf{v} \quad = \quad \mathbf{v}.$$

Examples:

1. Let $\mathcal{V}_n = \{0, 1\}^n$ be the set of binary vectors of length n and $F = \{0, 1\}$ be a field under modulo-2 addition and multiplication. Vector addition "+" is performed by adding pairs of vectors element by element modulo-2. Is $\mathcal{V}_n$ a vector space over F?

2. Let $\mathcal{C}$ contain the following elements:

$$\mathcal{C} = \left\{ \begin{array}{ccccc} (0 & 0 & 0 & 0 & 0), \\ (0 & 0 & 1 & 1 & 1), \\ (1 & 1 & 0 & 1 & 0), \\ (1 & 1 & 1 & 0 & 1) \end{array} \right\} \tag{2.1}$$

and again $F = \{0, 1\}$ is a field under modulo-2 addition and multiplication. Is $\mathcal{C}$ a vector space over F?

2.2 Linear Dependence

Linear Combination: A *linear combination* of ℓ vectors $\mathbf{v}_0, ..., \mathbf{v}_{\ell-1} \in \mathcal{V}_n$ is an expression of the form

$$\mathbf{v} = \sum_{i=0}^{\ell-1} a_i \mathbf{v}_i \tag{2.2}$$

where $a_i = \{0, 1\}$ and $a_i \mathbf{v}_i = a_i \cdot \mathbf{v}_i$.

Linear dependence: A set of vectors $\{\mathbf{v}_0, ..., \mathbf{v}_{\ell-1}\}$ is *linearly dependent* if there exists a nontrivial[1] linear combination equal to the all-zeros word, i.e. if there exists some combination

$$\sum_{i=0}^{\ell-1} a_i \mathbf{v}_i = \mathbf{0} \tag{2.3}$$

other than the *trivial* $a_0 = ... = a_{\ell-1} = 0$.

Linear independence: If a set of vectors is not linearly dependent, then it is said to be *linearly independent*.

Example #1: Is the set $\{(0, 1, 0), (1, 0, 1), (1, 1, 1)\}$ linearly independent?

Example #2: How about $\{(1, 0, 0), (1, 1, 0), (1, 1, 1)\}$?

Example #3: How about $\{(0, 0, 0)\}$?

Example #4: How about any set containing the all-zeros word?

[1] *Nontrivial* means that at least one of the a_i coefficients in equation (2.2) is nonzero.

2.3 Vector Subspaces

Let $\mathcal{C} \subset \mathcal{V}$ be a nonempty subset of the vector space $\mathcal{V}$. Then $\mathcal{C}$ is a *subspace* of $\mathcal{V}$ if it satisfies:

1. *Closure under vector addition:* If $\mathbf{u}, \mathbf{v} \in \mathcal{C}$, then $\mathbf{u} + \mathbf{v} \in \mathcal{C}$.
2. *Closure under scalar multiplication:* If $\mathbf{u} \in \mathcal{C}$ and $a \in F$, then $a \cdot \mathbf{u} \in \mathcal{C}$.

Note that $\mathcal{C}$ is in itself a vector space over F.

Examples:

1. Is $\mathcal{C}$ from the last example is a subspace of $\mathcal{V}_5$?

2. For $\mathcal{C} = \{\mathbf{v} : \mathbf{v}$ is a linear combination of $\mathbf{v}_0, ..., \mathbf{v}_{\ell-1}\}$, is $\mathcal{C}$ is a *vector subspace* of $\mathcal{V}_n$?

2.4 Bases and Dimensionality

2.4.1 Some Definitions

Spanning Set: The set $\mathcal{G} = \{\mathbf{g}_0, ..., \mathbf{g}_{\ell-1}\}$ *spans* $\mathcal{C}$ if every vector in $\mathcal{C}$ can be found be taking a linear combination of vectors in $\mathcal{G}$, i.e. $\mathcal{C} = \{\mathbf{v} : \mathbf{v}$ is a linear combination of $\mathbf{g}_0, ..., \mathbf{g}_{\ell-1}\}$.

Minimal Spanning Set: A spanning set $\mathcal{G}$ is *minimal* if no (strict) subset $\mathcal{G}$ is also a spanning set, i.e. there is no spanning set $\mathcal{T} : \mathcal{T} \subset \mathcal{G}$. In other words, the minimal spanning set contains the minimum number of vectors required to span $\mathcal{C}$.

Theorem #1: The set of vectors in a minimal spanning set are linearly independent. *Why?*

Basis: A minimal spanning set for the space $\mathcal{C}$ is called a *basis* for $\mathcal{C}$. Note that a space $\mathcal{C}$ could have several different bases, although each basis contains the same number of vectors.

Dimension: The *dimensionality* of $\mathcal{C}$, denoted $dim(\mathcal{C})$, is the number of vectors in any of its bases, i.e. if $\mathcal{G}$ is a basis for $\mathcal{C}$, then $dim(\mathcal{C}) = |\mathcal{G}| = k$.

Linear independence and bases: Any set of k linearly independent vectors $\{\mathbf{g}_0, ..., \mathbf{g}_{k-1}\} \in \mathcal{C}$ forms a basis for $\mathcal{C}$.

2.4.2 Binary Linear Codes

Definition: An (n, k) *binary linear code* $\mathcal{C}$ is a k-dimensional subspace of $\mathcal{V}_n$.

A note about notation: We generally use the notation $\mathbf{c}$ to denote a codeword; i.e., element of $\mathcal{C}$.

2.4.3 Finding a Basis

Finding a basis for $\mathcal{C}$ amounts to finding a set of k linearly independent vectors in $\mathcal{C}$. A somewhat brute-force method is as follows:

Initialization: We begin with the set $\mathcal{G} = \emptyset$, which is initially empty but will eventually contain all the basis vectors. We assume that we know all the vectors in $\mathcal{C}$.

Step 1: Let the first basis vector $\mathbf{g}_0$ equal any non-zero vector $\mathbf{c}_i \in \mathcal{C}$. Place this vector into $\mathcal{G}$.

Step 2: Now pick another nonzero vector $\mathbf{c}_i \in \mathcal{C}$ that has not yet been considered. Ask if $\mathbf{c}_i$ can be formed as a linear combination of the vectors currently in $\mathcal{G}$?

Yes: "Throw away" $\mathbf{c}_i$ and repeat step 2 using the next vector ($\mathbf{c}_i$ will not be considered in future steps of the algorithm).

No: Add $\mathbf{c}_i$ to $\mathcal{G}$. If the dimensionality of $\mathcal{C}$ is known, then repeat step 2 if $|\mathcal{G}| < k$ and stop once $|\mathcal{G}| = k$. If the dimensionality is not known, then keep repeating step 2 until all non-zero codewords in $\mathcal{C}$ have been considered.

Example #1: Determine the *canonical basis* for $\mathcal{V}_4$.

Example: Find a basis for the following subspace of $\mathcal{V}_7$,

$$\mathcal{C} = \left\{ \begin{array}{l} (0000000) \\ (0011101) \\ (0100111) \\ (0111010) \\ (1001110) \\ (1010011) \\ (1101001) \\ (1110100) \end{array} \right\}$$

2.5 Vector and Matrix Multiplication

Inner Product: The *inner product* of two vectors $\mathbf{v}, \mathbf{w} \in \mathcal{V}_n$ is defined by:

$$\mathbf{v} \cdot \mathbf{w} = \mathbf{v}\mathbf{w}^T \tag{2.4}$$

$$= \sum_{i=0}^{n-1} v_i w_i \tag{2.5}$$

where the sum is modulo-2, and thus $\mathbf{v} \cdot \mathbf{w} \in \{0, 1\}$.

Example: Compute the inner product of (1110) and (1011).

Orthogonality: Two vectors $\mathbf{v}, \mathbf{w} \in \mathcal{V}_n$ are *orthogonal* if their inner product is zero, i.e. $\mathbf{v} \cdot \mathbf{w} = 0$.

Example: Determine if the set $\{(0101), (1010), (1111)\}$ is mutually orthogonal.

Vector-Matrix Multiplication and Generator Matrices: Let $\mathbf{u} \in \{0, 1\}^k$ and let G be a k by n matrix with elements from the binary finite field $\{0, 1\}$. Then $\mathbf{u}G = \mathbf{c}$ is found by letting c_i equal the inner product of $\mathbf{u}$ with the i^{th} column of G. Note that $\mathbf{c}$ is a linear combination of the rows of G, and that the set of all $\mathbf{c}$ forms a subspace $\mathcal{C}$ of $\mathcal{V}_n$. If the rows of G are linearly independent, then G is called a *generator matrix* for $\mathcal{C}$.

Example: Let $\mathbf{u} = (101)$ and

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

Find $\mathbf{u}G$.

Matrix-Matrix Multiplication: Let $C = AB$, where A, B, and C are all matrices with binary entries. Then, $C_{i,j}$ is found by taking the inner product of the i^{th} row of A with the j^{th} column of B.

2.6 Hamming Space

Let $F = \{0, 1\}$ be a field over modulo-2 addition and multiplication. let $\mathcal{V}_n$ be the set of all length-n vectors with elements drawn from F (i.e. *binary* vectors). Define the following operations:

1. *Vector addition:* To add vector $\mathbf{u} = (u_0,\ u_1,\ \ldots\ , u_{n-1})$ with vector $\mathbf{v} = (v_0,\ v_1,\ \ldots\ , v_{n-1})$, simply add them element by element under the rules of the underlying field F, i.e.

$$\mathbf{u} + \mathbf{v} \quad = \quad (u_0 + v_0,\ , u_1 + v_1,\ \ldots\ , u_{n-1} + v_{n-1}),$$

   where addition is modulo-2. The all-zeros vector $\mathbf{0}$ is the identity under vector addition, i.e.

$$\mathbf{v} + \mathbf{0} = \mathbf{0} + \mathbf{v} \quad = \quad \mathbf{v}.$$

   It follows from the rules of vector and modulo-2 addition that

$$\mathbf{v} + \mathbf{v} \quad = \quad \mathbf{0},$$

   and thus $\mathbf{v}' = \mathbf{v}$ under vector addition, i.e. a vector is its own inverse.

2. *Scalar multiplication:* since $|F| = 2$, there are only two ways to perform scalar multiplication:

$$\begin{aligned} 1 \cdot \mathbf{v} &= \mathbf{v} \\ 0 \cdot \mathbf{v} &= \mathbf{0}. \end{aligned}$$

The n-dimensional vector space defined by the algebraic system above is called *Hamming Space* and is compactly denoted by $\mathcal{V}_n$.

2.7 Subspaces of $\mathcal{V}_n$

Let $\mathcal{C} \subset \mathcal{V}_n$ be a nonempty subset of the vector space $\mathcal{V}_n$.
$\mathcal{C}$ is a *subspace* of $\mathcal{V}_n$ if it is closed under vector addition; i.e.,

   If $\mathbf{c}_i \in \mathcal{C}$ and $\mathbf{c}_j \in \mathcal{C}$, then $\mathbf{c}_i + \mathbf{c}_j \in \mathcal{C}$.

Note that $\mathcal{C}$ is in itself a vector space over the binary field, and that it is closed under scalar multiplication (as a consequence of the way scalar multiplication is defined for $\mathcal{V}_n$ and the existence of the all-zeros word in $\mathcal{C}$).

2.7.1 Binary Linear Codes

The subspace $\mathcal{C}$ defined above is a *binary linear code*. Vectors $\mathbf{c} \in \mathcal{C}$ are called *codewords*. The code has the following properties:

1. The sum of any two codewords is itself a codeword, i.e. if $\mathbf{c}_1, \mathbf{c}_2 \in \mathcal{C}$, then $\mathbf{c}_1 + \mathbf{c}_2 \in \mathcal{C}$.
2. The code must contain an all-zeros codeword, i.e. $\mathbf{0} \in \mathcal{C}$.
3. $\mathcal{C}$ can be formed by taking all linear combinations of k linearly independent vectors from $\mathcal{V}_n$, in which case we can say that $\mathcal{C}$ is a *k-dimensional* subspace of $\mathcal{V}_n$, and that the set of k linearly independent codewords forms a *basis* for $\mathcal{C}$.
4. If $\mathcal{C}$ has dimension k, then $|\mathcal{C}| = 2^k$.
5. There is a one-to-one correspondence between *messages* and *codewords*, and the code may be used to convey k bits of information.

2.7.2 Generator Matrices

Generator Matrix: We know that any codeword $\mathbf{c} \in \mathcal{C}$ can be found by forming a linear combination of basis vectors $\mathbf{g} \in \mathcal{G}$. Furthermore, we know that multiplying a vector by a matrix forms a linear combination of the rows of the matrix. Thus, we can form any codeword in $\mathcal{C}$ by multiplying a k-dimensional vector $\mathbf{u} \in \{0,1\}^k$ by the *generator matrix* G, where the rows of G are the basis vectors

$$G = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_{k-1} \end{bmatrix} \tag{2.6}$$

That is, $\mathbf{c} = \mathbf{u}G$.

Message: Since the vector $\mathbf{u}$ may be any binary k-tuple, we can assign k data bits to it. $\mathbf{u}$ is called the *data word* or *message*.

Example: Recall the example given in Section 4.3 of Note Set #2. The basis was found to be:

$$\mathcal{G} = \left\{ \begin{array}{l} (0011101) \\ (0100111) \\ (1001110) \end{array} \right\}$$

Using this basis as the rows for a generator matrix, encode the message $\mathbf{u} = (101)$.

Row Space: The k-dimensional subspace of $\mathcal{V}_n$ formed by multiplying all possible $\mathbf{u} \in \{0,1\}^k$ by G is called the *row space* of G. Since the row space is a vector subspace of $\mathcal{V}_n$, it is a binary linear code.

Manipulating G: Any matrix G may be transformed by elementary row operations (swapping rows, adding distinct rows) into another matrix G' that has the same row space (although the mapping from messages to codewords will be different). *Why?*

Combinatorially Equivalent Codes: Now consider a matrix G' formed by performing elementary row operations on G and permuting the columns. G' is said to be *combinatorially equivalent* to G, which means their row spaces are identical except that the ordering of the bits within codewords might be different. Combinatorially equivalent codes have the same distance properties, and hence can correct/detect the exact same number of errors.

Example: Are the codes generated by the following two generator matrices combinatorially equivalent?

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

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

2.8 Dual Spaces

Definition: Let $\mathcal{C}$ be a subspace of $\mathcal{V}_n$. Then its *dual space* $\mathcal{C}^\perp$ is the set of all vectors $\mathbf{w} \in \mathcal{V}_n$ that are orthogonal to every vector $\mathbf{v} \in \mathcal{C}$. Thus for every $\mathbf{v} \in \mathcal{C} \subset \mathcal{V}_n$ and every $\mathbf{w} \in \mathcal{C}^\perp \subset \mathcal{V}_n$, $\mathbf{v} \cdot \mathbf{w} = 0$.

Properties: The dual space is in itself a subspace of $\mathcal{V}_n$, and therefore has all the properties of a vector subspace.

Dimensionality Theorem: If $dim(\mathcal{C}) = k$, then $dim(\mathcal{C}^\perp) = n - k$.

Not Disjoint: Note that $\mathcal{C}$ and $\mathcal{C}^\perp$ are not disjoint sets. *Why* (think about what element is always common to both)?

2.8.1 Dual Codes

Dual Code: Since $\mathcal{C}^{\perp}$ is a subspace of $\mathcal{V}_n$, it is in itself a linear code. In particular, the code $\mathcal{C}^{\perp}$ is called the *dual code* of $\mathcal{C}$.

Example: Find the dual code for the code generated by

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

Self-Dual Codes: If $\mathcal{C} = \mathcal{C}^{\perp}$, then the code is called *self-dual*. How must k and n be related for a self-dual code?

2.8.2 Parity-Check Matrices

Parity-check Matrix: Since the dual code $\mathcal{C}^{\perp}$ is a $(n-k)$ dimensional subspace of $\mathcal{V}_n$, we can find a basis for it. Denote the basis $\mathcal{H} = \{\mathbf{h}_0, ..., \mathbf{h}_{n-k-1}\}$. We can create a generator matrix for the dual code as follows:

$$H = \begin{bmatrix} \mathbf{h}_0 \\ \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_{n-k-1} \end{bmatrix} \tag{2.7}$$

Thus, codewords $\mathbf{w} \in \mathcal{C}^{\perp}$ can be generated by the matrix multiplication $\mathbf{w} = \mathbf{u}H$, where now $\mathbf{u} \in \{0,1\}^{n-k}$.

Syndrome: Let $\mathbf{r} \in \mathcal{V}_n$, then $\mathbf{s} = \mathbf{r}H^T$ is called a *syndrome. What is the dimensionality of* $\mathbf{s}$?

Error Detection: If $\mathbf{r}$ is a valid codeword, then what is its syndrome? i.e. if $\mathbf{r} \in \mathcal{C}$; i.e., it is a valid codeword, then determine $\mathbf{s} = \mathbf{r}H^T$.

G times H-transpose: Applying the last result, what is GH^T equal to?

Example: Recall the example given in Section 4.3 of Note Set #2. The basis was found to be:

$$\mathcal{G} \quad = \quad \left\{ \begin{array}{l} (0011101) \\ (0100111) \\ (1001110) \end{array} \right\}$$

Find the parity-check matrix for this code.

2.9 Systematic Codes

Systematic Form: A linear block code can be designed so that each code word can be partitioned as follows:

$$\mathbf{c} \quad = \quad (\mathbf{u}, \mathbf{p}) \tag{2.8}$$

where **u** is the message and **p** contains the n-k *parity bits*. A code whose codewords and messages are related in this way is said to be in *systematic form*. Note that to determine if a code is in systematic form, we need to know how messages are mapped to codewords.

Systematic-form for G: What is the form of the G matrix if the code is in systematic form?

Putting G into systematic form: Every linear code with generator matrix G has a combinatorially equivalent code with generator matrix G' which is in systematic form. The systematic G' is found by performing elementary row operations on G and permuting columns, when necessary.

Example: Put the following G matrix into systematic form:

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

Relationship between G and H: If G is in systematic form, then what is the form for its parity check matrix H?

Chapter 3

Binary Linear Codes

3.1 Binary Codes

3.1.1 Hamming Space

Hamming Space $\mathcal{V}_n = \{0,1\}^n$ is the set of all n-dimensional vectors with elements from the binary field, along with the corresponding vector addition (add corresponding elements of the vectors modulo-2) and scalar multiplication ($0 \cdot \mathbf{v} = \mathbf{0}$ and $1 \cdot \mathbf{v} = \mathbf{v}$) operations.

3.1.2 Binary Codes

A *binary code* $\mathcal{C}$ is any subset of $\mathcal{V}_n$, i.e. $\mathcal{C} \subset \mathcal{V}_n$. Vectors $\mathbf{c} \in \mathcal{C}$ are called *codewords*. Binary codes may be *linear* or *non-linear*.

3.1.3 Linear Codes

A code $\mathcal{C}$ is a *linear* code if it forms a vector subspace of $\mathcal{V}_n$, which is assured as long as $\mathcal{C}$ is closed under vector addition (closure under scalar multiplication is automatically assured for binary codes).

3.1.4 Nonlinear Codes

If a code is not linear, then it is said to be a *nonlinear* code.

3.1.5 Cosets of $\mathcal{C}$

Let $\mathcal{C}$ be a linear code. Let $\mathbf{v}_0 \in \mathcal{V}_n$ be any vector from n-dimensional Hamming space. The set $\mathbf{v}_0 + \mathcal{C}$, which is found by adding $\mathbf{v}_0$ to every vector in $\mathcal{C}$, is called a *coset* of $\mathcal{C}$. Some properties of cosets:

1. $\mathbf{v}_0$ is called the *coset leader*. Any element from a coset may serve as the leader.
2. If $\mathbf{v}_0 \in \mathcal{C}$, then the coset $\mathbf{v}_0 + \mathcal{C}$ will be $\mathcal{C}$ itself (Why?).
3. Every vector in $\mathcal{V}_n$ is in exactly one coset.
4. The cosets are disjoint (this follows from 3).

5. Except for the coset that is $\mathcal{C}$ itself, all cosets are nonlinear codes (Why?).

6. There are 2^{n-k} distinct cosets (including $\mathcal{C}$, which is a coset of itself).

3.1.6 Dual Code

The binary code $\mathcal{C}$ has a dual code $\mathcal{C}^\perp$, which contains all of the vectors that are orthogonal to all of the codewords in $\mathcal{C}$, i.e. for every $\mathbf{v} \in \mathcal{C} \subset \mathcal{V}_n$ and every $\mathbf{w} \in \mathcal{C}^\perp \subset \mathcal{V}_n$, $\mathbf{v} \cdot \mathbf{w} = 0$.

The duality relationship is reciprocal, so it follows that $\mathcal{C}$ is the dual of $\mathcal{C}^\perp$. Since $\mathcal{C}^\perp$ contains every vector that is orthogonal to all vectors in $\mathcal{C}$, it follows that $\mathcal{C}$ contains every vector that is orthogonal to all vectors in $\mathcal{C}^\perp$. Therefore, any vector that is not in $\mathcal{C}$ will *not* be orthogonal to every vector in $\mathcal{C}^\perp$ (though it may be orthogonal to some vectors). This fact is crucial to the detection and correction of errors, as will become apparent on the next page.

3.2 Binary Linear Codes

The following concepts pertain to *linear* codes. Because they do not satisfy closure, nonlinear codes cannot be represented by a basis.

3.2.1 Bases and Generators

The set $\mathcal{G} \subset \mathcal{C}$ is said to *span* the linear code $\mathcal{C}$ if every codeword $\mathbf{c} \in \mathcal{C}$ may be expressed as a linear combination of the vectors in $\mathcal{G}$. If the vectors in $\mathcal{G}$ are linearly independent, then $\mathcal{G}$ is a *minimal spanning set* or *basis*. Given a particular basis, the linear combination of basis vectors used to form a codeword is unique (one-to-one).

3.2.2 Dimension of a Code and its Dual

Let $\mathcal{G}$ be a basis for $\mathcal{C}$. The *dimension* k of $\mathcal{C}$ is equal to $|\mathcal{G}|$. Note that the number of codewords in $\mathcal{C}$ is $|\mathcal{C}| = 2^k$.

Now let $\mathcal{C}^\perp$ be the dual of $\mathcal{C}$. If $\mathcal{C}$ is linear, then so is $\mathcal{C}^\perp$. From the *dimensionality theorem*, $|\mathcal{C}^\perp|$ has dimension $n - k$, and thus it may be generated from the linear combinations of a set $\mathcal{H}$ of $n - k$ basis vectors.

3.2.3 Generator and Parity-Check Matrices

A *generator matrix* is a $k \times n$ full-rank[1] matrix G whose rows form a basis for $\mathcal{C}$.

A *parity-check matrix* is a rank $(n - k)$ matrix H whose rows span $\mathcal{C}^\perp$. Usually (but not necessarily), the rows of H are linearly independent (i.e. H is full rank), in which case H is a generator matrix for $\mathcal{C}^\perp$.

Note that if H is the generator matrix for $\mathcal{C}^\perp$ and a parity-check matrix for $\mathcal{C}$, it follows that G is the generator matrix for $\mathcal{C}$ and a parity-check matrix for $\mathcal{C}^\perp$.

[1]The (row) *rank* of a matrix is the maximum number of linearly independent rows. The rank is *full* if it is equal to the number of rows.

3.2.4 Systematic Codes

A *systematic* code is one whose codewords may be expressed as:

$$\mathbf{c} = (\mathbf{u}, \mathbf{p}) \tag{3.9}$$

where $\mathbf{u}$ is the message and $\mathbf{p}$ contains the n-k *parity bits*.

The generator matrix for a linear systematic code will have the following form:

$$G = [I \;\; P] \tag{3.10}$$

where I is an $k \times k$ identity matrix and P is a $k \times (n-k)$ matrix.

If a code is in systematic form, then one of its parity-check matrices is

$$H = [P^T \;\; I] \tag{3.11}$$

where I is an $(n-k) \times (n-k)$ identity matrix. Proof:

.

Every linear code has a *combinatorially equivalent* code that may be put into systematic form.

Exercise: Put the following G matrix into systematic form:

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

Exercise: Find a parity-check matrix for the code with generator matrix G. Be careful to account for column permutations (we want the parity-check matrix for the *original* code generated by G, not its the combinatorially equivalent code generated by G').

3.2.5 Syndromes

Let $\mathbf{r} \in \mathcal{V}_n$. The vector

$$\mathbf{s} = \mathbf{r}H^T \tag{3.12}$$

is called the *syndrome* of $\mathbf{r}$. It will be a row vector with as many elements as H has rows. If H is full rank, then $\mathbf{s}$ will have $(n-k)$ elements. Syndromes are useful for both error detection and correction.

All members of a particular coset of $\mathcal{C}$ generate the same syndrome. To see this, recall that each element of the coset may be represented by the sum of a particular vector $\mathbf{v}_0 \in \mathcal{V}_n$ (called a *coset leader*) with one of the codewords $\mathbf{c} \in \mathcal{C}$. The syndrome is

$$\begin{aligned} \mathbf{s} &= (\mathbf{v}_0 + \mathbf{c})H^T \\ &= \mathbf{v}_0 H^T + \mathbf{c}H^T \\ &= \mathbf{v}_0 H^T \end{aligned} \tag{3.13}$$

where the third line follows from the second because $\mathbf{c}$ is orthogonal to every row of H. Thus, the syndrome of all elements of a coset are equal to the syndrome of the coset leader. Furthermore, each distinct coset has a distinct syndrome. Since there are 2^{n-k} cosets of $\mathcal{C}$, there are also 2^{n-k} distinct syndromes (one for each linear combination of $\mathcal{H}$).

Only one of the cosets will generate an all-zeros syndrome, and it is the coset that is $\mathcal{C}$ itself. For this coset, the coset leader may be taken to be $\mathbf{v}_0 = \mathbf{c}_0 = \mathbf{0}$, in which case equation (3.13) is simply $\mathbf{s} = \mathbf{0}H^T = \mathbf{0}$.

3.2.6 Error Detection

Error detection is a simple matter: All one does is compute the syndrome and check to see if it is all zeros. If it is not all zeros, then declare that an error has been detected. If it is all zeros, then assume that the codeword is correct. Note, however, that it is possible to have an *undetected* error as follows.

Suppose that the codeword $\mathbf{c} \in \mathcal{C}$ is transmitted over a *binary symmetric channel* (BSC). A BSC channel will independently invert bits with probability p. It can be modeled by defining another vector $\mathbf{e} \in \mathcal{V}_n$ such that the received vector is $\mathbf{r} = \mathbf{c} + \mathbf{e}$. The vector $\mathbf{e}$ is called the *error indicator vector*. The syndrome is then:

$$\begin{aligned} \mathbf{s} &= (\mathbf{c}+\mathbf{e})H^T \\ &= \mathbf{c}H^T + \mathbf{e}H^T \\ &= \mathbf{e}H^T \end{aligned} \tag{3.14}$$

which will be a non-zero vector as long as $\mathbf{e} \notin \mathcal{C}$. However, $\mathbf{s} = \mathbf{0}$ if $\mathbf{e} \in \mathcal{C}$, i.e. if the error pattern is in itself a codeword. Such an error pattern is said to be *undetectable*.

Now consider the probability of an undetectable error. This is the probability that $\mathbf{e}$ is a codeword. The probability of any particular error pattern of weight i (i.e. with i ones) is $p^i(1-p)^{n-i}$. Suppose that there are A_i codewords of weight i. The set $\{A_i\}$ is called the *weight distribution* of $\mathcal{C}$. Derive an expression for the probability of undetected error in terms of the set $\{A_i\}$ [See equation (3.16) in book].

Exercise: Consider the code spanned by the set

$$\mathcal{G} = \left\{ \begin{array}{l} (0011101) \\ (0100111) \\ (1001110) \end{array} \right\}$$

Derive an expression for the probability of undetected error (as a function of the BSC crossover probability):

3.2.7 Error Correction

The goal of error correction is to form an estimate $\hat{\mathbf{c}}$ of the transmitted codeword $\mathbf{c}$, given the received word $\mathbf{r}$. The most likely estimate will be the one that assumes the minimum number of errors occurred on the channel. Equivalently, one could estimate the error pattern $\hat{\mathbf{e}}$ and add it to the received codeword $\mathbf{r}$,

$$\begin{aligned} \hat{\mathbf{c}} &= \mathbf{r} + \hat{\mathbf{e}} \\ &= (\mathbf{c} + \mathbf{e}) + \hat{\mathbf{e}} \\ &= \mathbf{c} + (\mathbf{e} + \hat{\mathbf{e}}) \end{aligned}$$

which will be the correct codeword $\mathbf{c}$ provided that $\hat{\mathbf{e}} = \mathbf{e}$. If however, the wrong error pattern was selected, then $\hat{\mathbf{e}} \neq \mathbf{e}$, and $\hat{\mathbf{c}} \neq \mathbf{c}$ meaning that the decoder failed to properly correct the error.

The question then becomes how to determine the most appropriate $\hat{\mathbf{e}}$? We can use the syndrome to help us with this selection. The idea is that the syndrome will tell us what coset the received vector $\mathbf{r}$ is in. Let $\hat{\mathbf{e}}$ be the coset leader, then every vector $\mathbf{r}$ in that coset can be expressed as the sum of the leader with a codeword, i.e. $\hat{\mathbf{e}} + \mathbf{c}$. Since all elements of the coset have the same syndrome, we can come up with a decoder that works as follows:

1. Compute the syndrome $\mathbf{s} = \mathbf{r}H^T$.
2. Using the syndrome, identify the coset that $\mathbf{r}$ belongs to.
3. Identify the leader $\hat{\mathbf{e}}$ of the coset, which will be the ML error pattern.
4. Correct the error (assuming the ML error pattern): $\hat{\mathbf{c}} = \mathbf{r} + \hat{\mathbf{e}}$.

The main issue in designing such a decoder is identifying the set of coset leaders, which will correspond to the set of correctable error patterns. This is not trivial, since any member of a coset may be chosen to be its leader. However, the ML principle suggests that we should use the minimum weight vector of each coset to be its leader.

More details and an example of how to use cosets for decoding will follow, but we first need to establish some more terminology.

3.3 Minimum Distance

3.3.1 Definitions

The following definitions are useful:

Hamming Weight: The number of ones in a binary vector, denoted $w(\mathbf{v})$. It can be found by adding up the elements of $\mathbf{v}$, i.e. $w(\mathbf{v}) = \sum_{j=0}^{n-1} v_j$. This is usually just called the *weight*. Combinatorics can be used to determine $|\{\mathbf{v} \in \mathcal{V}_n : w(\mathbf{v}) = i\}|$.

Hamming Distance: The number of positions that two vectors differ, $d(\mathbf{v}, \mathbf{u}) = w(\mathbf{v} + \mathbf{u})$. Often this is just called *distance*.

Hamming Sphere: The Hamming sphere of radius ℓ centered at $\mathbf{v}$ is the set of all binary vectors within distance ℓ of $\mathbf{v}$:

$$S_\ell(\mathbf{v}) \quad = \quad \{\mathbf{u} \in \mathcal{V}_n : d(\mathbf{u}, \mathbf{v}) \leq \ell\} \tag{3.15}$$

Using combinatorics, it is possible to determine $|S_\ell(\mathbf{v})|$ as a function of n and ℓ.

Minimum (Hamming) Distance: The minimum Hamming distance among all pairs of distinct codewords,

$$d_{min} \quad = \quad \min_{\mathbf{c}_i, \mathbf{c}_j \in \mathcal{C}} \{d(\mathbf{c}_i, \mathbf{c}_j) : \mathbf{c}_i \neq \mathbf{c}_j\} \tag{3.16}$$

A code $\mathcal{C} \subset \mathcal{V}_n$ with $|\mathcal{C}| = 2^k$ and minimum distance d_{min} is referred to a (n, k, d_{min}) code for short.

3.3.2 Error Correction and Detection Capability

Error Detection: The process of determining if the received codeword is valid, i.e. checking if $\mathbf{r} \in \mathcal{C}$. If this is true, then either $\mathbf{r}$ is the transmitted codeword, or the channel has transformed the transmitted codeword into a different valid codeword (if this occurs, then we have *undetectable errors*).

Error Detection Capability: Since the closest pair of codewords are d_{min} apart there exists some error indicator vector $\mathbf{e}$ of weight $w(\mathbf{e}) = d_{min}$ that transforms some valid codeword $\mathbf{c}_i$ into some other valid codeword $\mathbf{c}_j$, i.e. $\mathbf{r} = \mathbf{c}_i + \mathbf{e} = \mathbf{c}_j$. Conversely, there is no vector $\mathbf{e}$ of weight $w(\mathbf{e}) < d_{min}$ that will transform any valid codeword into another valid codeword. Thus the maximum weight of $\mathbf{e}$ for which errors can *always* be detected is $t_d = d_{min} - 1$. The value $d_{min} - 1$ is the *maximum error detection capability of the code.*

ML Error Correction: The process of determining which codeword is closest to the received vector,

$$\hat{\mathbf{c}} \quad = \quad \arg\min_{\mathbf{c}_i \in \mathcal{C}} d(\mathbf{c}_i, \mathbf{r}) \tag{3.17}$$

Once we have an estimate of the transmitted codeword $\hat{\mathbf{c}}$, it is straightforward (at least conceptually) to find the corresponding estimate of the data word $\hat{\mathbf{u}}$, since $\mathbf{u} \Leftrightarrow \mathbf{c}$.

Decoding Rule: A rule for choosing an estimate $\hat{\mathbf{u}}$ for every possible received $\mathbf{r}$. Note that while the mapping $\mathbf{u} \Rightarrow \mathbf{c}$ is one-to-one, the mapping $\mathbf{r} \Rightarrow \hat{\mathbf{u}}$ is many-to-one.

Decoding Error: The decoder produces erroneous results whenever $\hat{\mathbf{u}} \neq \mathbf{u}$, or equivalently when $\hat{\mathbf{c}} \neq \mathbf{c}$.

Error Correcting Capability: Let t be the maximum error weight $w(\mathbf{e})$ for which errors can *always* be *corrected*, regardless of what $\mathbf{c}$ was transmitted. t is the largest radius of nonoverlapping Hamming spheres around all distinct codewords:

$$t \quad = \quad \max_{\mathbf{c}_i, \mathbf{c}_j \in \mathcal{C}} \{\ell : S_\ell(\mathbf{c}_i) \cap S_\ell(\mathbf{c}_j) = \emptyset, \mathbf{c}_i \neq \mathbf{c}_j\} \tag{3.18}$$

Correction and Detection: It is possible to combine detection and correction. The idea is to use some of the code's redundancy to *correct* a small number of errors, while using the remaining redundancy to *detect* a larger number of errors. Note that errors must first be detected before they can be corrected, so the set of correctable error patterns is a subset of the set of detectable error patterns. Let t_c indicate the maximum number of correctable errors and $t_d \geq t_c$ indicate the maximum number of detectable errors. Then t_c and t_d must satisfy $t_c + t_d < d_{min}$. We can now consider two extreme cases:

Detection Only: Here $t_c = 0$ and thus $t_d < d_{min}$ which is satisfied by $t_d = d_{min} - 1$ which agrees with our earlier discussion.

Correction Only: Here all error patterns that are uncorrectable are also undetectable. This implies that $t_c = t_d = t$, since correctable error patterns must also be detectable. This gives $2t < d_{min}$. For odd d_{min} this is satisfied by $t = (d_{min} - 1)/2$, while for even d_{min} it requires $t = (d_{min} - 2)/2$. These two conditions can be succinctly represented as:

$$t = \left\lfloor \frac{d_{min} - 1}{2} \right\rfloor \tag{3.19}$$

where the floor operator $\lfloor . \rfloor$ rounds down to the closest integer.

3.3.3 Minimum Distance of a Linear Code

The minimum distance of a linear code $\mathcal{C}$ is the minimum Hamming weight of all the nonzero codewords in $\mathcal{C}$,

$$d_{min} = \min_{\mathbf{c} \in \mathcal{C}} \{w(\mathbf{c})\} \tag{3.20}$$

Proof:

3.4 Finding the Minimum Distance from H

Theorem: There exists a codeword in $\mathcal{C}$ of weight w if and only if w of the columns in H are linearly dependent.

Proof :

Corollary: The minimum distance d_{min} is the smallest nonzero number of columns in H for which a nontrivial linear combination sums to zero.

Example: Find the minimum distance of the code with the following parity-check matrix:

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

Chapter 4

The Many-Uses of H: Design, Encoding, and Decoding

4.1 Properties of Parity-Check Matrices

Let $\mathcal{C}$ be a (n, k) linear code and $\mathcal{C}^{\perp}$ its $(n-k, n)$ dual. A *parity-check matrix* is a rank $(n-k)$ matrix H whose rows *span* $\mathcal{C}^{\perp}$. Usually, but not necessarily, the rows of H are linearly independent (i.e. H is full rank), in which case H is a *generator* matrix for $\mathcal{C}^{\perp}$.

4.1.1 Non-uniqueness of the H matrix:

While the generator matrix G *uniquely* specifies the mapping of messages $\mathbf{u}$ to codewords $\mathbf{c}$, the parity-check matrix of a code is *not* unique. A code $\mathcal{C}$ may be represented by any of a number of parity-check matrices. All that is required for each H is that its rows span $\mathcal{C}^{\perp}$.

Example: Suppose we have a systematic code $\mathcal{C}$ generated by:

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

Then, the parity-check matrices for this code include:

$$H_1 = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$$H_2 = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

$$H_3 = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

$$H_4 = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

$$H_5 = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Notice that H_1, H_2, and H_3 are full rank, but H_4 and H_5 are not.

4.1.2 Finding the Minimum Distance from H

Three Theorems (using the book's numberings):

(3.1) There exists a codeword in $\mathcal{C}$ of weight w if and only if w of the columns in H sum to zero.

(3.2) The minimum distance d_{min} is the smallest nonzero number of columns in H that sum to zero.

(3.3) If no d or fewer columns sum to zero, then $d_{min} > d$.

4.2 Code Design

4.2.1 Optimizing Codes of Particular n and d_{min}

Suppose we require a particular code length n and minimum distance d_{min}. Once those parameters are set, the parameter left to optimize is the code *rate* R. In particular, given n and d_{min} we wish to *maximize* R. How do we do this?

We will be designing the code using the parity-check matrix H. Suppose that the H matrix is full rank and has $n - k = m$ rows. It follows that the code rate is:

$$R = \frac{k}{n} = \frac{n-m}{n} = 1 - \frac{m}{n} \tag{4.21}$$

Since n is already fixed, it follows that to maximize the rate, we want to *minimize* m, which is the number rows in our full rank H.

Alternatively, if $m = n - k$ is fixed (the amount of redundancy in the code) along with d_{min}, then we will want to *maximize* the code length n.

4.2.2 Designing Codes with $d_{min} \geq 3$

Rules that follow from Theorem (3.3):

1. If H does not contain a column of all zeros, then $d_{min} > 1$.
2. If, in addition to rule 1, H does not contain repeated columns, then $d_{min} > 2$.

Thus, if follows that if rules 1 and 2 are satisfied, the code will have $d_{min} \geq 3$.

4.2.3 Hamming Codes

Let's optimize a code for rate $d_{min} = 3$. To design this code, we simply have to make sure H does not have an all-zeros column (so that $d_{min} > 1$), and that H has no repeated columns (so that $d_{min} > 2$).

If we fix m, then to maximize the rate, we will want to maximize n. This is done by picking H to contain the maximum possible number of distinct nonzero columns. How many distinct nonzero columns are there? Since each column has a length of m and contains binary elements, it follows that there are 2^m distinct columns possible for a rank m parity-check matrix. However, one of these columns is all-zeros, and thus the number of distinct *nonzero* columns is $n = 2^m - 1$.

The $(n, k, d_{min} = 3)$ code of maximum rate is constructed by creating a parity-check matrix containing all distinct nonzero columns of length $m = n - k$. Such a code is called a *Hamming* code. Hamming codes are characterized by the following properties:

$$\begin{aligned} d_{min} &= 3 \\ n &= 2^m - 1 \\ k &= n - m \end{aligned}$$

Note that the code rate will increase with increasing m, as demonstrated in the following table:

m	n	k	R
2			
3			
4			
5			
10			

4.2.4 Shortening

Shortening is the process of deleting columns from the H matrix in such a way that the rank is maintained. Let H be the parity-check matrix of code $\mathcal{C}$, and let $\mathcal{C}'$ be the shortened code with parity-check matrix H' created by deleting one column from H. The deleted column is selected so that $rank(H') = rank(H)$. If $\mathcal{C}$ is a (n, k) code, then $\mathcal{C}'$ is a $(n-1, k-1)$ code, and the rate is decreased from $R = k/n$ to $R' = (k-1)/(n-1)$.

Equivalently, shortening is the process of deleting one column *and* one row from the G matrix (while maintaining full rank). If G is in systematic form, then this may be achieved by deleting one of the weight-1 columns of the G matrix (which for $d_{min} > 2$ codes, will be a systematic bit position) and the row that contained the single 1 in that column.

Let d_{min} be the minimum distance of the original code $\mathcal{C}$ and d'_{min} be the minimum distance of the shortened code $\mathcal{C}'$. It follows that:

$$d'_{min} \quad \geq \quad d_{min} \tag{4.22}$$

Why?

Example: Construct the parity-check matrix for a $(7, 4)$ Hamming code $\mathcal{C}$. By repeatedly shortening $\mathcal{C}$, come up with a code $\mathcal{C}'$ with $d'_{min} = 4$.

4.2.5 The Singleton Bound

From the last section, it is evident that we can continually increase the minimum distance by deleting more and more columns from the H matrix. We need to make sure that we maintain the rank of H as we do this. The following interesting question arises: For an (n, k) code, what is the maximum d_{min}? The answer is d_{min} must be limited by:

$$d_{min} \leq \underbrace{n-k}_{m}+1 \tag{4.23}$$

Why?

The inequality in (4.23) is called the *Singleton bound*, and any code that satisfies this bound with equality is called a *minimum distance separable* (MDS) code[1].

4.2.6 Codes of $d_{min} = 4$

Theorem: Let $=\{\mathbf{v}_0, \mathbf{v}_1, ..., \mathbf{v}_{d-1}\}$ be a set of d vectors from $\mathcal{V}_n$, each of *odd* weight. Then:

$$\sum_{i=0}^{d-1} \mathbf{v}_i \neq \mathbf{0} \tag{4.24}$$

if d is odd.
Proof:

Corollary: If H has all odd-weight columns, then d_{min} will be even.
This idea can be used to construct a (rate optimal) code of $d_{min} = 4$ by deleting all even-weight columns from the H matrix of a Hamming code.

[1] Repetition codes and Reed Solomon codes are examples of MDS codes.

Exercise: Starting with the parity-check matrix of a $(15, 11)$ Hamming code, come up with a $d_{min} = 4$ code of maximal rate. You may specify the new code in terms of an H matrix.

4.3 Systematic Encoding Using the H Matrix

In Section 4.1.1 we saw that a systematic code may be represented by one of several H matrices. It turns out that a systematic code may be encoded by using *any* of its full-rank parity-check matrices that have the following form:

$$H \quad = \quad [A,\ L]$$

where L is a rectangular $(n-k) \times (n-k)$ lower-triangular matrix with all 1's along the diagonal, and A is an arbitrary $(n-k) \times k$ matrix. Encoding is performed by *back substitution. Example:* Encode the message $\mathbf{u} = (\ 1\ 1\ 0\ 1\ )$ by using parity-check matrix H_2 from Section 4.1.1:

Why would we want to encode using an alternative H matrix instead of the systematic-form G matrix?

4.4 Decoding Using H

4.4.1 Standard Array Decoding

In the last set of notes, we saw that we may partition $\mathcal{V}_n$ into 2^{n-k} *cosets*, and that each vector $\mathbf{r} \in \mathcal{V}_n$ will be in exactly one of these cosets. We can use this idea to formulate an error correcting decoder by assigning a correctable error pattern to each of the cosets. The main detail to work out is what error patterns to assign to each coset. This may be done in a methodical fashion by constructing a *standard array*. A standard array is a $2^{n-k} \times 2^k$ array of containing all vectors from $\mathcal{V}_n$, i.e. each element of the standard array is a vector. The columns of the standard array correspond to the codewords of $\mathcal{C}$, while the rows of the standard array correspond to the cosets of $\mathcal{C}$.

The standard array is constructed as follows (Section 3.1.4 of text):

1. Place the codewords $\mathbf{c} \in \mathcal{C}$ into the first row of the standard array, with the all-zeros codeword $\mathbf{c}_0 = \mathbf{0}$ at the top of the first column.

2. Move on the the next row, which we call row j.

3. For row j pick a coset leader from the set of vectors in $\mathcal{V}_n$ that have not appeared in any of the previous rows of the standard array. The coset leader should be a vector of minimum weight. Place the coset leader in the first entry of the j^{th} row. Call this coset leader $\mathbf{e}_j$.

4. Now form the rest of the j^{th} row by adding the coset leader $\mathbf{e}_j$ to the codeword located at the top of the column. If $\mathbf{c}_i$ is the codeword located at the top of the i^{th} column, then the $(j,i)^{th}$ entry of the standard array is $\mathbf{e}_j + \mathbf{c}_i$.

5. If $j = 2^{n-k}$, then the last row of the standard array has been reached, in which case it is complete. Otherwise, go to set 2.

Having completed the standard array, it may be used for decoding as follows:

1. Locate the received $\mathbf{r}$ in the table.

2. Find the codeword $\mathbf{c}_i$ located at the top of the column containing $\mathbf{r}$. This is the most likely codeword.

If the actual error pattern was the $\mathbf{e}_j$ that leads the row containing $\mathbf{r}$, then the codeword will be correctly decoded. Otherwise, if the error pattern was some other element of that row, then $\mathbf{r}$ will *not* be correctly decoded.

Exercise: Suppose that the code $\mathcal{C}$ is generated by:

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

construct the standard array.

4.4.2 Syndrome Table Decoding

Recall that there is a one-to-one correspondence between *cosets* and *syndromes*. Instead of storing the entire standard array at the decoder, it is sufficient to just maintain a list of syndromes and the corresponding correctable error patterns.

- For any vector $\mathbf{r} \in \mathcal{V}_n$, the parity-check matrix H may be used to compute the *syndrome* $\mathbf{s} = \mathbf{r}H^T$.
- All members of a particular coset have the same syndrome, $\mathbf{s}_j = \mathbf{e}_j H^T$, where $\mathbf{e}_j$ is the coset leader.
- Instead of storing the entire standard array, store a two-column array containing each distinct syndrome $\mathbf{s}_j$ and the corresponding coset leader $\mathbf{e}_j$.
- To decode, use the syndrome $\mathbf{c}_j$ to lookup the error pattern $\mathbf{e}_j$ in the table.
- The decoded codeword is then $\hat{\mathbf{c}} = \mathbf{r} + \mathbf{e}_j$.

Exercise: Create the syndrome-table for the last exercise:

4.4.3 A Simple Decoder for the Hamming Code

It is easy to design a digital circuit capable of performing the following mapping:

$$
\begin{aligned}
(0\,0\,0) &\longrightarrow (0\,0\,0\,0\,0\,0\,0)\\
(0\,0\,1) &\longrightarrow (1\,0\,0\,0\,0\,0\,0)\\
(0\,1\,0) &\longrightarrow (0\,1\,0\,0\,0\,0\,0)\\
(0\,1\,1) &\longrightarrow (0\,0\,1\,0\,0\,0\,0)\\
(1\,0\,0) &\longrightarrow (0\,0\,0\,1\,0\,0\,0)\\
(1\,0\,1) &\longrightarrow (0\,0\,0\,0\,1\,0\,0)\\
(1\,1\,0) &\longrightarrow (0\,0\,0\,0\,0\,1\,0)\\
(1\,1\,1) &\longrightarrow (0\,0\,0\,0\,0\,0\,1)
\end{aligned}
$$

Construct a Hamming code (in terms of an H matrix) that may be decoded by passing the syndrome $\mathbf{s}$ through a logical circuit with the above I/0 relationship.

4.5 The Hamming Bound

Each distinct coset of $\mathcal{C}$ (row in the standard array) corresponds to exactly one correctible error pattern (including the all-zeros error pattern). The total number of length n correctible error patterns of weight $w_H(\mathbf{e}) \leq t$ cannot exceed the number of distinct cosets (otherwise the standard array would not be able to correct them all). Since there are 2^{n-k} distinct cosets of $\mathcal{C}$, this leads to the following inequality:

$$|w_H(\mathbf{e}) \leq t| \quad \leq \quad 2^{n-k} \tag{4.25}$$

The number of length n correctible error patterns of weight $w_H(\mathbf{e}) \leq t$ is

$$|w_H(\mathbf{e}) \leq t| \quad = \quad \sum_{i=0}^{t} \binom{n}{i}. \tag{4.26}$$

Substituting (4.26) into (4.25) results in

$$\sum_{i=0}^{t} \binom{n}{i} \quad \leq \quad 2^{n-k} \tag{4.27}$$

which is called the *Hamming Bound*.

4.6 Perfect Codes

Definition: A *perfect* code is a code that satisfies the Hamming bound with equality:

$$\sum_{i=0}^{t} \binom{n}{i} \quad = \quad 2^{n-k} \tag{4.28}$$

- Perfect codes can be identified by looking for codes that satisfy the Hamming bound. There are only two known non-trivial binary perfect codes (Hamming and Golay) and a trivial binary perfect code (repetition code of odd length).
- Hamming codes
- Golay codes
- Repetition codes of odd length

4.7 Summary of Terms

- **Shortening** - Deleting columns from the parity-check matrix to increase d_{min}.
- **Singleton Bound** - The maximum possible d_{min} for an (n, k) code.
- **Standard Array** - $2^{n-k} \times 2^k$ array s.t. the first row contains all codewords from $\mathcal{C}$, and all additional rows contain cosets of $\mathcal{C}$.
- **Syndrome Table** - Table of error patterns and syndromes used for decoding.
- **Hamming Bound** - A bound that arises from the fact that the number of error patterns of weight t or less cannot exceed the number of rows in the standard array.
- **Perfect Code** - Any code which satisfies the Hamming Bound with equality.

Chapter 5

Cyclic Codes

5.1 Cyclic Codes: Motivation

Based on what we have seen *so far*:

- The encoder can be specified in terms of either G or H. However, this requires the storage of the entire matrix, which for practical codes can be quite large.
- An error-detecting decoder requires storage of H, which again could be large.
- An error-correcting decoder requires storage of not only H, but also a syndrome table which can be very large.

These observations lead to the following conclusions:

- It is desirable to represent the code in a more compact form instead of specifying the entire G matrix, H matrix, and/or syndrome table.
- By imposing more structure onto the code (aside from linearity), it should be able to represent codes in a more compact manner.
- *Cyclic codes* are a subclass of the linear codes that enable a more compact representation. The generators can be efficiently specified by a polynomial of degree $n-k$ with coefficients from GF(2) and the parity-checks may be expressed as a polynomial (or vector) of degree k. With cyclic codes, the matrix multiplications $\mathbf{c} = \mathbf{u}G$ and $\mathbf{s} = \mathbf{r}H^T$ are replaced with a type of discrete-time *convolution* to be described below.

5.2 Cyclic Codes: Definitions

Cyclic codes are covered in Sections 3.2 and 3.3 of the Ryan and Lin text (pp. 106-121). See also section 2.5 (pp. 51-56) which covers polynomials over finite fields. Please read these sections to fill in some of the finer points that we will not have the luxury of completely covering in class.

Cyclic shift: Given a vector $\mathbf{v} = (v_0, v_1, ..., v_{n-2}, v_{n-1})$. Its (right) cyclic shift is $\mathbf{v}^{(1)} = (v_{n-1}, v_0, v_1, ..., v_{n-2})$. It follows that if $\mathbf{v}$ is shifted i times the result is $\mathbf{v}^{(i)} = (v_{n-i}, v_{n-i+1}, ..., v_{n-1}, v_0, v_1, ..., v_{n-i-1})$.

Cyclic code: A linear code $\mathcal{C}$ is *cyclic* if every cyclic shift of a codeword in $\mathcal{C}$ is also a codeword in $\mathcal{C}$. In other words, for every $\mathbf{c} \in \mathcal{C}$ then $\mathbf{c}^{(1)} \in \mathcal{C}$.

Example: Is the following code cyclic?

$$\mathcal{C} = \left\{ \begin{array}{cccccccc} (& 0 & 0 & 0 & 0 & 0 & 0 & 0 &) \\ (& 1 & 1 & 0 & 1 & 0 & 0 & 0 &) \\ (& 0 & 1 & 1 & 0 & 1 & 0 & 0 &) \\ (& 1 & 0 & 1 & 1 & 1 & 0 & 0 &) \\ (& 1 & 1 & 1 & 0 & 0 & 1 & 0 &) \\ (& 0 & 0 & 1 & 1 & 0 & 1 & 0 &) \\ (& 1 & 0 & 0 & 0 & 1 & 1 & 0 &) \\ (& 0 & 1 & 0 & 1 & 1 & 1 & 0 &) \\ (& 1 & 0 & 1 & 0 & 0 & 0 & 1 &) \\ (& 0 & 1 & 1 & 1 & 0 & 0 & 1 &) \\ (& 1 & 1 & 0 & 0 & 1 & 0 & 1 &) \\ (& 0 & 0 & 0 & 1 & 1 & 0 & 1 &) \\ (& 0 & 1 & 0 & 0 & 0 & 1 & 1 &) \\ (& 1 & 0 & 0 & 1 & 0 & 1 & 1 &) \\ (& 0 & 0 & 1 & 0 & 1 & 1 & 1 &) \\ (& 1 & 1 & 1 & 1 & 1 & 1 & 1 &) \end{array} \right\} \tag{5.29}$$

5.3 Polynomial Representation of Vectors

The vector $\mathbf{v} = (v_0, v_1, v_2, ..., v_{n-1})$ can be represented by the *polynomial* [book equation (3.24)]:

$$\begin{aligned} v(X) &= v_0 + v_1 X + v_2 X^2 + ... + v_{n-1} X^{n-1} \\ &= \sum_{i=0}^{n-1} v_i X^i, \end{aligned}$$

where the coefficients $v_i \in \{0, 1\}$. This polynomial is said to be "a polynomial over GF(2)" since its coefficients are from GF(2). The *degree* of a polynomial is the largest power of X with a nonzero coefficient. Clearly, the degree of $v(X) \leq n-1$, and it is equal to $n-1$ if and only if $v_{n-1} = 1$.

- How many polynomials of degree ℓ over GF(2) are there?

- For the remainder of this set of notes, all polynomials are assumed to be over GF(2).

5.4 Polynomial Arithmetic

A polynomial $f(X) = f_0 + f_1 X + f_2 X^2 + ... + f_n X^n$ may be added, subtracted, multiplied, or divided by a second polynomial $g(X) = g_0 + g_1 X + g_2 X^2 + ... g_m X^m$, as follows:

Addition: To add $f(X)$ and $g(X)$, add the coefficient of the same power of X. If $m \leq n$, then

$$\begin{aligned} f(X) + g(X) &= (f_0 + g_0) + (f_1 + g_1)X + (f_2 + g_2)X^2 + ... \\ &\quad + (f_m + g_m)X^m + f_{m+1} X^{m+1} + ... + f_n X^n \end{aligned}$$

Example: Add $f(X) = 1 + X + X^4 + X^5 + X^6$ to $g(X) = 1 + X + X^3$

Subtraction: When subtracting one polynomial from another, the coefficients of the same power of X are subtracted. However, since the coefficients are from GF(2), and in GF(2) subtraction is equivalent to addition, then we find that $f(X) - g(X) = f(X) + g(X)$.

Multiplication: Polynomial multiplication proceeds as it would for ordinary polynomials (those whose coefficients are real-valued) only now the operations on the coefficients are over GF(2).

Example: Multiply the $f(X)$ and $g(X)$ from the previous example.

Multiplication and Convolution: In general, we find that

$$f(X)g(X) = c_0 + c_1X + c_2X^2 + ... + c_{n+m}X^{n+m}$$

where

$$\begin{aligned} c_0 &= f_0g_0 \\ c_1 &= f_0g_1 + f_1g_0 \\ c_2 &= f_0g_2 + f_1g_1 + f_2g_0 \\ c_i &= \sum_{\{j,k\}:j+k=i} f_jg_k \\ c_{n+m} &= f_ng_m, \end{aligned}$$

and the multiplication and addition are all over GF(2). If we define $g_i = 0$ for $i < 0$, then this can be expressed as

$$c_i = \sum_{j=0}^{n} f_jg_{i-j}.$$

This may look familiar. It is the equation for discrete-time convolution. In fact, the coefficients of $c(X)$ are obtained by performing a discrete-time convolution over GF(2) of the coefficients of $f(X)$ with the coefficients of $g(X)$. Matlab could be used to do the convolution *(conv)*. However, to make it over GF(2) it is necessary to use the syntax *c=mod(conv(f,g),2)*. Alternatively, if you have the communication toolbox, then use the "gf" command to make f and g of type gf before convolving them.

Division: When $f(X)$ is divided by $g(X)$, we get a unique pair of polynomials over GF(2), namely $q(X)$ which is called the *quotient* and $r(X)$ which is called the *remainder*. The four polynomials are related by:

$$f(X) \quad = \quad q(X)g(X) + r(X),$$

and the degree of $r(X)$ is less than that of $g(X)$. Generally, $q(X)$ and $r(X)$ must be obtained through *long division*.

Example: Divide $f(x)$ by $g(X)$ from the previous example.

5.5 Factoring Polynomials

Factors: If the remainder $r(X) = 0$ when $f(X)$ is divided by $g(X)$, then we say that $f(X)$ is *divisible* by $g(X)$ and that $g(X)$ is a *factor* of $f(X)$.

Roots: "a" is a *root* of the polynomial $f(X)$ if $f(a) = 0$. For now we only consider two possible real-valued roots, $a \in \{0, 1\}$.

- For $a = 0$ to be a root, then $f_0 = 0$ and X must be a factor of $f(X)$.
- For $a = 1$ to be a root, then $X + 1$ must be a factor, and therefore $f(X)$ must be divisible by $X + 1$. If the polynomial has an even number of nonzero terms, then it will be divisible by $X + 1$ and hence have $a = 1$ as a root.

Irreducible polynomials: A polynomial $f(X)$ of degree n is said to be *irreducible* over GF(2) if $f(X)$ is not divisible by any polynomial $g(X)$ of degree m, where $0 < m < n$.

Examples: Which of the following polynomials are irreducible:

1. $f(X) = X + X^3 + X^4 + X^5$

2. $f(X) = 1 + X^2 + X^3 + X^4$

3. $f(X) = 1 + X + X^3$

4. $f(X) = 1 + X + X^4$

5. List the set of all irreducible polynomials of degree 2.

6. List the set of all irreducible polynomials of degree 3.

Theorem (2.9): Any irreducible polynomial over GF(2) of degree m divides $X^n + 1$ where $n = 2^m - 1$.

Factored form: A polynomial $f(X)$ of degree n may be expressed in *factored form* as

$$f(X) = \prod_{i=1}^{m} g_i(X)$$

where the $g_i(X)$ are all irreducible polynomials, and $m \leq n$ is the number of irreducible factors.

Example: Factor the polynomial $f(X) = 1 + X^7$

5.6 Cyclic Shift of Polynomials

Let $v(X) = v_0 + v_1X + v_2X^2 + ... + v_{n-1}X^{n-1}$. Now multiply $v(X)$ by X,

$$Xv(X) = v_0X + v_1X^2 + v_2X^3 + ... + v_{n-1}X^n$$

Add $(v_{n-1} + v_{n-1}) = 0$ to get:

$$\begin{aligned} Xv(X) &= v_{n-1} + v_0X + v_1X^2 + v_2X^3 + ... + v_{n-1} + v_{n-1}X^n \\ &= v_{n-1} + v_0X + v_1X^2 + v_2X^3 + ... + v_{n-1}(1 + X^n) \end{aligned}$$

Recall that $\mathbf{v}^{(1)} = (v_{n-1}, v_0, v_1, ..., v_{n-2})$, so

$$Xv(X) = v^{(1)}(X) + q(X)(1 + X^n),$$

where $q(X) = v_{n-1}$. Comparing against the definition of polynomial division, we see that if we divide $Xv(X)$ by $1 + X^n$, the remainder is $v^{(1)}(X)$. More generally, if we divide $X^iv(X)$ by $1 + X^n$, the remainder is $v^{(i)}(X)$.

Example: Let $v(X) = 1 + X^3 + X^5 + X^6$. Determine $v^{(2)}(X)$.

5.7 Code, Generator, and Parity Polynomials

Code Polynomials: Each codeword $\mathbf{v} \in \mathcal{C}$ may be represented as a polynomial of degree no more than $n-1$ called a *code polynomial*.

Generator Polynomials: Let $g(X)$ be the nonzero code polynomial of smallest degree m. There will be only one such polynomial of smallest degree. If $g(X)$ has degree $m = n-k$, then it can be used to generate a (n,k) code through the polynomial multiplication $v(X) = u(X)g(X)$, where $u(X) = u_0 + u_1 X + u_2 X^2 + ... + u_{k-1} X^{k-1}$ is the message polynomial of degree no more than $k-1$ (polynomial whose coefficients come from the k-tuple $\mathbf{u}$). Since $g(X)$ may be used to generate a code, it is called a *code polynomial*.

What is the maximum degree of $v(X)$?

Example: Determine the code polynomial g(X) for the code given by (5.29). Represent each codeword by its corresponding code polynomial and show how it can be obtained by multiplying the message polynomial by the generator polynomial.

Generator Matrix: The generator matrix of a cyclic code has the form [equation (3.27) from the text]:

$$G = \begin{bmatrix} g_0 & g_1 & g_2 & \cdots & g_{n-k} & 0 & 0 & \cdots & 0 \\ 0 & g_0 & g_1 & \cdots & g_{n-k-1} & g_{n-k} & 0 & \cdots & 0 \\ \vdots & & & & & & & & \vdots \\ 0 & 0 & 0 & \cdots & g_0 & g_1 & \cdots & g_{n-k-1} & g_{n-k} \end{bmatrix} \tag{5.30}$$

That is, the first row contains the coefficients of the generator polynomial, and each subsequent row is a shift of the previous row (by one position to the right). The generator can be put in systematic form by performing elementary row operations *without* any column permutations. If no columns are permuted, the systematic code will still be cyclic.

Error Detection: Every code polynomial in a cyclic code is a multiple of $g(X)$. Therefore, a polynomial $v(X)$ of degree $n-1$ or less is a code polynomial if and only if it is a multiple of $g(X)$. Long division can be used to determine if a polynomial is a code polynomial or not. This is done by dividing $f(X)$ and $g(X)$ and checking to see if the remainder is zero.

Example: Let $g(X) = 1 + X + X^3$. Is $X^2 + X^3 + X^4$ a valid code polynomial?

Parity-check Polynomial: The generator polynomial $g(X)$ must be a factor of $1 + X^n$. [1] Let the polynomial $1 + X^n$ be factored as follows:

$$1 + X^n = g(X)f(X) \tag{5.31}$$

where $g(X)$ is of degree $n - k$ and $f(X)$ is of degree k. Note that $g(X)$ and $f(X)$ need not be irreducible. Given $g(x)$ and n, then $f(X)$ may be found by dividing $1 + X^n$ by $g(X)$. Define the reciprocal polynomial of $f(X)$ to be [book equation (3.29)]:

$$h(X) = X^k f\left(X^{-1}\right) \tag{5.32}$$

The polynomial $h(X)$ is called the parity-check polynomial of $\mathcal{C}$, and it is the generator polynomial for $\mathcal{C}^{\perp}$.

Example: Determine the parity-check polynomial for the code given by (5.29).

Example: How many distinct cyclic codes of length $n = 7$ are there?

Example: Determine the generator polynomial for the dual of the code given by (5.29).

Parity-check Matrix: The parity-check matrix may be expressed as [equation (3.30) from the text]:

$$H = \begin{bmatrix} h_0 & h_1 & h_2 & \ldots & h_k & 0 & 0 & \ldots & 0 \\ 0 & h_0 & h_1 & \ldots & h_{k-1} & h_k & 0 & \ldots & 0 \\ \vdots & & & & & & & & \vdots \\ 0 & 0 & 0 & \ldots & h_0 & h_1 & \ldots & h_{k-1} & h_k \end{bmatrix} \tag{5.33}$$

[1] The proof is Theorem 5.5 of S. Lin and D. Costello, Error Control Coding, second edition.

Chapter 6

BCH Codes

6.1 Review of Cyclic Codes

1. Every codeword $\mathbf{c} = (c_0, c_1, c_2, ..., c_{n-1})$ in $\mathcal{C}$ may be represented as a *code polynomial* $c(X) = c_0 + c_1X + c_2X^2 + ... + c_{n-1}X^{n-1}$ of degree $n-1$ or less.

2. The code polynomial $g(X) = g_0 + g_1X + ... + g_mX^m$ that has smallest degree is called the *generator polynomial*.

3. If the code is *cyclic*, then all valid code polynomials are multiples of $g(X)$.

4. $g(X)$ must divide the polynomial $1 + X^n$. The proof follows.

5. If $f(X)g(X) = 1 + X^n$, then $h(X) = X^k f(X^{-1})$ is the parity-check polynomial, which is the generator for $\mathcal{C}^\perp$.

6. If $\mathcal{C}$ is cyclic, then so is $\mathcal{C}^\perp$.

Proof of 4 (Theorem 5.5 in Lin and Costello, 2e):

6.2 Overview of BCH Codes

For a given length n and dimension k, there may be several (n, k) cyclic codes to choose from (namely, any for which the degree $m = n - k$ generator divides $1 + X^n$). As n and k increase, so does the number of possible generators. Furhermore, for a given n and k, we would like to know what the error correcting capability t is. We would like to now turn our attention to picking the best $g(X)$ to satisfy our design requirements. The code design problem may be posed as follows:

1. Specify the length n of the code.
2. Specify the required error correcting capability of the code t'. It is okay if the actual error correcting capability $t > t'$.
3. Pick a generator $g(X)$ of minimum degree that satisfies requirements 1 and 2.

Since the actual t might be greater than t', we call t' the *designed* error correction capability. By picking the *minimal* degree $g(X)$, we will have the minimum redundancy, i.e. the fewest parity bits if the code is systematic, and yet will meet (or exceed) the required error correcting capability. By minimizing the amount of redundancy, we will be maximizing the code rate for the speficied n and t.

If, as with the Hamming code, we limit the code length n to be of the form $n = 2^\ell - 1$, for integer ℓ, then the design problem can be solved by using BCH codes[1]. The key properties of BCH codes are as follows:

1. For parameter $\ell \geq 3$, the length of the BCH code is $2^\ell - 1$.
2. A BCH code exists with an error correcting capability of t or greater which requires $m = \ell t$ parity-check bits or fewer.

We would now like to determine how to design BCH codes by picking the appropriate generator polynomial. In order to design binary BCH codes, we must first understand some more advanced polynomial theory.

6.3 Advanced Polynomial Theory

6.3.1 Primitive Polynomials

Recall that all irreducible polynomials of degree ℓ divide $X^n + 1$ for $n = 2^\ell - 1$ [Theorem 2.9, Ryan and Lin, page 55]. An irreducible polynomial $p(X)$ of degree ℓ is *primitive* if the smallest n for which it divides $X^n + 1$ is $n = 2^\ell - 1$ [Definition 2.21, Ryan and Lin, page 55].

Exercise: Which of the degree 4 irreducible polynomials are primitive? For those that are not primitive, determine the minimum value of n for which the polynomial divides $X^n + 1$.

[1] BCH codes are cyclic codes that are named after Bose, Chaudhuri, and Hocquenghem, who invented the codes in 1959-1960.

6.3.2 Constructing $GF(2^\ell)$

Let α be a *root* of the degree-ℓ primitive polynomial $p(X)$, i.e. $p(\alpha) = 0$. Because $p(X)$ is irreducible, α does not come from $GF(2)$, but rather it comes from the *extension field* $GF(2^\ell)$. Let us consider how to construct the field $GF(2^\ell)$.

Let's start with multiplication. Recall that the elements of the field excluding zero must form an abelian group under multiplication. Use α as a generator for a cyclic multiplicative group $\{\alpha^0, \alpha^1, \alpha^2, \alpha^3, ...\}$. There will be no more than $2^\ell - 1$ distinct elements in this group because $\alpha^{2^\ell - 1} = 1 = \alpha^0$ [Proof follows]. Because $p(X)$ is primitive, the number of distinct elements in the group is *exactly* $2^\ell - 1$ [Proof follows]. Thus, we can use as our multiplicative group the $2^\ell - 1$ distinct powers of α along with the multiplication rule $\alpha^i \alpha^j = \alpha^{i+j} = \alpha^{(i+j) \mod (2^\ell - 1}0$ with multiplicative identity $\alpha^0 = 1$. This group is isomorphic to the integers $\{0, ..., 2^\ell - 2\}$ under modulo-$(2^\ell - 1)$ addition.

Proof that $\alpha^{2^\ell - 1} = 1$ [p.57, Ryan and Lin]

Now let's consider addition. We need to show that the entire set of elements in the field form an abelian group under a suitably defined addition operation. This means we need an additive identity, which we will denote as '0'. Let α^i be the i^{th} power of α. We can represent α^i as a polynomial $a_i(X)$ by using the remainder that comes as a result of dividing X^i by $p(X)$, i.e. [equation (2.34), Ryan and Lin, p. 57]

$$X^i = q_i(X)p(X) + a_i(X) \tag{6.34}$$

where $q_i(X)$ is the quotient of the division operation.

Each of the $2^\ell - 1$ nonzero elements of $GF(2^\ell)$ may be represented by a *distinct* polynomial $a_i(X)$ [theorem 2.10, Ryan and Lin, p. 58].

Proof:

We see that there is a one-to-one mapping of powers of α to polynomials,

$$\alpha^i \quad \Leftrightarrow \quad a_i(X) \tag{6.35}$$

Thus, each element in $GF(2^\ell)$ may be represented using either its *power representation* or its *polynomial representation*. The zero element is represented by the zero polynomial. Because the polynomials are distinct and of degree no more than $\ell - 1$, *all* 2^ℓ polynomials of degree $\ell - 1$ or less are used to represent the 2^ℓ elements of the field. Each element may also be compactly expressed using an *integer representation*

$$\sum_{i=0}^{\ell-1} 2^i a_i \tag{6.36}$$

Exercise: Determine the polynomial and integer representations of all the nonzero elements of $GF(2^3)$. Use primitive polynomial $p(X) = 1 + X + X^3$.

Having represented each field element as a polynomial, we may simply use polynomial addition as our addition operation. We know that polynomial addition is associative and commutative. Furthermore, since all 2^ℓ polynomials of degree less than ℓ are in the field's set of elements, the set will be closed under addition.

Exercise: Create the corresponding addition and multiplication tables for $GF(2^3)$ (see also HW 1 problem 6).

The solution is (numerical values are the integer representation):

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	0	3	2	5	4	7	6
2	2	3	0	1	6	7	4	5
3	3	2	1	0	7	6	5	4
4	4	5	6	7	0	1	2	3
5	5	4	7	6	1	0	3	2
6	6	7	4	5	2	3	0	1
7	7	6	5	4	3	2	1	0

·	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	3	1	7	5
3	0	3	6	5	7	4	1	2
4	0	4	3	7	6	2	5	1
5	0	5	1	4	2	7	3	6
6	0	6	7	1	5	3	2	4
7	0	7	5	2	1	6	4	3

6.3.3 Minimal Polynomials

Let α^i be an element of $GF(2^\ell)$. Let $\phi_i(X)$ be the polynomial of smallest degree that has α^i as a root. The polynomial $\phi_i(X)$ is called the *minimal* polynomial of α^i [Definition 2.21, Ryan and Lin, page 66]. The minimal polynomial is irreducible (theorem 2.14) and unique (theorem 2.15). A table of minimal polynomials may be found in Appendix B of Lin and Costello, second edition.

Because every α^i is a root of $1 + X^{2^\ell - 1}$ [Theorem 2.13, Ryan and Lin, p. 66], it follows that $\phi_i(X)$ is a factor of $1 + X^{2^\ell - 1}$. This suggests a way to find the minimal polynomial of α^i:

- Express $1 + X^{2^\ell - 1}$ in factored form,
- For each factor of $1 + X^{2^\ell - 1}$, determine if α^i is a root.
- If more than one factors are identified in 2, pick the one of smallest degree.

Exercise: Consider $GF(2^3)$ formed by primitive polynomial $p(X) = 1 + X + X^3$. Find $\phi_i(X)$ for $i = \{1, 2, 3, 4\}$.

6.3.4 Conjugates

If β is a root of a polynomial $f(X)$, then β^2 is also a root of $f(X)$ [Theorem 2.11, Ryan and Lin, p. 64]. β and β^2 are said to be *conjugates*. As a consequence, both β and β^2 will have the same minimal polynomial.

6.4 BCH Code Generators

Let α be a primitive element of $GF(2^\ell)$. The length $n = 2^\ell - 1$ BCH code with *designed* error correcting capability t' has a generator polynomial $g(X)$ with $2t'$ consecutive powers of α as roots. For instance [equation (3.39), Ryan and Lin, p. 111]

$$\alpha, \alpha^2, \alpha^3, ..., \alpha^{2t'} \tag{6.37}$$

may be the roots of $g(X)$. If the *first* $2t'$ powers are used (as in the above equation), the code is said to be a *narrow-sense* BCH code, but more generally *any* set of $2t'$ consecutive powers of α may be used.

The generator is found as

$$g(X) \quad = \quad LCM\{\phi_1(X), \phi_2(X), ..., \phi_{2t'}(X)\} \tag{6.38}$$

where LCM is the *least common multiple* operator and $\phi_i(X)$ is the minimal polynomial of α^i. The idea behind using the LCM operator is that repeated factors may be discarded. Two roots α^i and α^j will have the same minimal polynomial if they are conjugates, in which case they will only contribute one factor to $g(X)$.

Exercises:

1. Design a BCH code of length $n = 7$ capable of correcting at least $t' = 1$ errors.
2. Design a BCH code of length $n = 7$ capable of correcting at least $t' = 2$ errors.
3. Design a BCH code of length $n = 15$ capable of correcting at least $t' = 2$ errors.