MATH 422

CODING THEORY

NOLAN ZUREK

Chapter 1 - Introduction to Codes

> This is where most (though not all) students realize this course isn't about programming

Code Definitions

A **code** C is given set of M **codewords**, which are in turn finite sequences of symbols from the code's **Alphabet** F_q. E.g. $C_0 = \{000, 010, 101, 111\}$ is a code.

- A **q-ary code** is a code with alphabet $F_q = \mathbb{Z}_q$
- A **binary code** has alphabet $\{0, 1\}$
- If each codeword has the same length n, the code is a **block code**.

A **word** or **vector** is any sequence of symbols from F_q, so the set of words of size n is $F_q \times F_q \times \cdots \times F_q = (F_q)^n$. Not all words are codewords, i.e. $C \subseteq F_q \times F_q \times \cdots \times F_q$. Generally, we endeavour to *encode* words into codewords.

A code can be written as an $M \times n$ *array* where the rows are the codewords of C. E.g. C_0 from earlier is represented by the matrix $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$.

> **Basic Code Parameters**
>
> An (n, M, d) code has M codewords of length n (i.e. a *length of* n) with *minimum distance* d

Distance

The **Hamming distance** $d(\vec{x}, \vec{y})$ between vectors $\vec{x}, \vec{y} \in (F_q)^n$ is defined as the number of places in which they differ.

The Hamming distance is a *distance function* because it satisfies

1. $d(\vec{x}, \vec{y}) = 0$ if and only if $\vec{x} = \vec{y}$
2. $d(\vec{x}, \vec{y}) = d(\vec{y}, \vec{x})$ for all $\vec{x}$ and $\vec{y}$
3. $d(\vec{x}, \vec{y}) \leq d(\vec{x}, \vec{z}) + d(\vec{z}, \vec{y})$ for all $\vec{x}, \vec{y}, \vec{z}$ (the *triangle inequality*)

The **minimum distance** $d(C)$ of code C is the smallest distance between any two codewords in the code. So, $d(C_0) = 1$ because 000 and 010 have distance 1.

- Formally, $d(C) = \min\{d(\vec{x}, \vec{y}) \mid \vec{x}, \vec{y} \in C, \vec{x} \neq \vec{y}\}$

Error Detection and Correction

Theorem 1.9

A code C can detect up to s errors in a given codeword if $d(C) \geq s+1$, and correct up to t errors if $d(C) \geq 2t+1$.

By corollary (1.10) a code C can detect up to $d(C)-1$ errors and correct up to $\left\lfloor \frac{d(C)-1}{2} \right\rfloor$ errors

Channels

We assume vector $\vec{x}$ is transmitted and vector $\vec{y}$ is received, possibly having been distorted.

A communication channel is **q-ary symmetric** if each symbol has the same probability $p < \frac{1}{2}$ of being received in error, and each of the $q-1$ possible errors for a given symbol is equally likely.

Pages: 1-10

Chapter 2 - The main coding theory problem

> The main problem of coding theory is that I decided to take coding theory

What Makes a Good Code?

"Good" codes generally have

- *Small* n so that transmission is fast
- *Large* M to require less codewords per message
- *Large* d to correct many errors

> **Coding Theory**
>
> The **main problem of coding theory** is the optimization of one parameter n, M, d of a code given values for the other two.

Equivalent Codes

Two q-ary codes are **equivalent** of one can be obtained from the other by

1. Permutation of the *positions* of the code → permutation of the columns of the code's matrix
2. Permutation of the symbols of the code → (internally) relabelling the symbols in a column of the code's matrix

Distances between codewords are invariant under this operation, so *equivalent codes have the same parameters* n, M, d, and thus have the *same error detection and correction capabilities*.

`LEMMA 2.3` Any q-ary (n, M, d) code over alphabet $F_q = \{0, 1, \ldots, q-1\}$ is equivalent to an (n, M, d) code containing the *zero vector* $\vec{0} = 00\ldots0$

- It is often helpful to assume WLOG that a code contains $\vec{0}$ when answering questions regarding A_q.

Optimizing M

We use $A_q(n, d)$ to denote the *largest* M such that a q-ary (n, M, d) code exists, i.e. the number of codewords that can exist in a code given n and d.

- $A_q(n, 1) = q^n$, namely when $C = (F_q)^n$

- $A_q(n,n) = q$, namely when C is (or is equivalent to) the q-ary **repetition code** of length n

Aside: the number of q-ary $(n, M, \_)$ codes is $\binom{2^n}{M}$

Binary Codes

Theorem 2.7

For odd d, a binary (n, M, d) code exists if and only if a binary $(n+1, M, d+1)$ code exists.

Corollary 2.8

For odd d, $A_2(n+1, d+1) = A_2(n, d)$. Thus, for even d, $A_2(n, d) = A_2(n-1, d-1)$

Spheres and Sphere Packing

A **sphere** $S(\vec{u}, r)$ of radius r around vector $\vec{u}$ is the set of vectors in $(F_q)^n$ whose distance from $\vec{u}$ is less than r, i.e. $S(\vec{u}, r) = \{\vec{v} \in (F_q)^n \mid d(\vec{u}, \vec{v}) \leq r\}$

For t error-detecting codes, we have $d(C) \geq 2t+1$, implying that the spheres with radius t centered on the codewords of C are *disjoint*. This implies that we can simply pick the (closest) sphere a received vector is in to decode it. This is an instance of **nearest neighbour decoding**.

A sphere of radius r in $(F_q)^n$ contains $\sum_{i=0}^{r} \binom{n}{i} (q-1)^i$ vectors

- Aside: the terms in this series correspond to the number of vectors of distance i from the center of the sphere.

Sphere Packing Bound

A q-ary $(n, M, 2t+1)$-code satisfies $M \times \sum_{i=0}^{t} \binom{n}{i} (q-1)^i \leq q^n$

- This clearly follows from the sphere population definition

Codes that reach the sphere packing bound are **perfect codes**; the spheres of radius t centered at a perfect code's codewords "fill" all of $(F_q)^n$ without overlapping.

- E.g. The binary repetition codes of length n are perfect codes

Balanced Block Designs

Balanced Block Design

A **balanced block design** is a set S of v *points/varieties* with a collection of b subsets of itself, called **blocks**. For fixed k, r, λ, we have

- Each block contains k points
- Each point lies in r blocks
- Each pair of points occurs together in λ blocks

- We define block designs by their parameters, i.e. we would say "a (b, v, r, k, λ)-design"
- E.g. the *seven-point plane* represents a balanced block design

The parameters (b, v, r, k, λ) are not independent; we find the following constraints (among others)

- $bk = vr$ is the *total number of points* in the design
- $r(k-1) = \lambda(v-1)$ is the number of pairwise occurrences of a given point with any other point

A balanced block design is **symmetric** if $v = b$, which implies $k = r$ as well.

We can describe a balanced block design by an **incidence matrix**, where the columns correspond to blocks, rows correspond to points, and each entry is 0 or 1 depending on whether a particular point is in a particular block.

Aside: balanced block designs have applications beyond coding theory, e.g. statistical testing combinations of fertilizers on different crops.

Pages: 11-29

Chapter 3 - Finite Fields

> It's pronounced "gal-WUAH"

Recap: Algebraic Structures

Field

A **field** F is a set of elements equipped with addition $+$ and multiplication $\cdot$ operations that satisfies the following properties:

1. *Closure* under $+$ and $\cdot$
2. Commutative $+$ and $\cdot$
3. Associative $+$ and $\cdot$
4. Distributivity: $a \cdot (b + c) = a \cdot b + a \cdot c$

A field must also have the *identity elements* 0 and 1, satisfying for all $a \in F$:

1. *Additive Identity*: $a + 0 = a$
2. *Multiplicative Identity*: $a \cdot 1 = a$
3. *Additive inverse*: $-a$ exists where $a + (-a) = 0$
4. *Multiplicative Inverse*: a^{-1} exists where $a \cdot a^{-1} = 1$

The following properties are implied by this definition:

- *Zero absorption/annihilation*: $a0 = 0$ for all $a \in F$
- *Cancellation law*: $ab = 0 \implies a = 0 \text{ or } b = 0$

Aside: combining $+$ and $\cdot$ with inverses lets us define operations like $-$ and $\div$

Abelian Ring (in terms of field)

A **abelian ring** is also a set equipped with $+$ and $\cdot$ that has the same properties as a field except the guarantee of multiplicative inverses for all elements.

Finite Fields: Basic Definitions

A **finite field** with **order** n is a field with a finite number n of elements.

- E.g. the *ring* $\mathbb{Z}_n$ is a *field* (and thus a finite field) if and only if n is a prime number.

Theorem 3.2

If a field of order q exists, q must be a *prime power*, i.e. $q = p^h$ for some prime p.

All fields for a given q share the same structure; the structure in general is known as the **Galois field of order** q, denoted $\mathrm{GF}(q)$.

Modular Arithmetic

Integers a and b are **congruent modulo** m (denoted $a \equiv b \mod m$) if $a = km + b$ for some integer k. Informally, a and b are congruent if they have the same *remainder* when divided by m.

- Aside: this mirrors the structure of a quotient space

We find that for $a \equiv a'$ and $b \equiv b'$, we get $a + b \equiv a' + b'$ and $ab \equiv a'b'$, which further implies $a^n = (a')^n$ (all $\mod m$).

- This can be encapsulated into a field Z_m if and only if m is prime, since otherwise we could find some $ab \equiv 0$, which cannot happen for nonzero a, b in a field.

Euler Totient Function

We define the **Euler totient function** or **Euler indicator** as the function $\varphi(n) := |\{m \in \mathbb{N} \mid a \leq m \leq n, \mathrm{GCD}(m, n) = 1\}|$

- So, $\varphi(n)$ is the number of integers less than or equal to n that are relatively prime with n.
- If p is a prime number, then $\varphi(p) = p - 1$ and $\varphi(p^r) = p^r - p^{r-1}$ for any $r \in \mathbb{N}$
  - The second fact is true because $p, 2p, 3p, \ldots, (p^{r-1} - 1)p$ all have a factor in common with p^r
- If we denote $\mathbb{Z}_n^*$ as the set of integers in $\mathbb{Z}_n$ that are not 0-divisors, then $|\mathbb{Z}_n^*| = \varphi(n)$ since every number sharing a factor with n is by definition a 0-divisor (i.e. can be multiplied with another element of $\mathbb{Z}_n$ to yield 0).

The **Chinese remainder theorem** states that $\varphi(mn) = \varphi(m)\varphi(n)$ if and only if $\mathrm{GCD}(m, n) = 1$.

- This implies that $\sum_{d|n} \varphi(d) = n$ for all $n \in \mathbb{N}$

Primitive Elements

The **order** of an element α of a finite field $\mathbb{F}$ is the smallest natural number e such that $\alpha^e = 1$.

The nonzero elements of any finite field can be written as powers of a single element

A **primitive element** α is an element of order $q-1$ in a finite field F_q

- Thus, successive powers of α eventually generate every member of F_q, so $F_q = \{0, \alpha^{0,}\alpha^{1,}\ldots, \alpha^{q-2}\}$
- So, since every element in the field can be written this way, we can write any multiplication in F_q as $\alpha^i\alpha^j$
- Primitive elements aren't necessarily unique; F_{q^r} will contain $\varphi(p^r-1)$ primitive elements, namely a^i for all i that are relatively prime to p^r-1
- In F_q, we (clearly) have $\alpha^q = \alpha$ since α is by definition of order $q-1$
- If α is primitive in F_q, then $\alpha^{-1} = \alpha^{q-2}$. α^{-1} is also a primitive element

Polynomials

Minimal Polynomials

Every element β of a finite field F_{q^r} is a root of the equation $\beta^q - \beta = 0$ and is a root of some polynomial $f(x) \in F_p[x]$.

For element $\beta \in F_{p^r}$, **minimal polynomial of** β is the *monic* polynomial $m(x) \in F_p[x]$ of least degree with β as a root.

- Existence of the minimal polynomial can be proven with the division algorithm
- $m(x)$ must be irreducible in $F_p[x]$.

For $\beta \in F_{p^r}$ and $f(x) \in F_p[x]$ with β as a root, then $f(x)$ is divisible by the minimal polynomial of β.

- By corollary, the minimal polynomial of $\beta \in F_q$ must divide $x^q - x$.

Primitive Polynomials

A **primitive polynomial** of a field is a the minimal polynomial of a primitive element of a field.

- If $f(x) \in F_p[x]$ then $f(x^p) = [f(x)]^p$
- If α is a root of $f(x) \in F_p[x]$, then α^p is also a root of $f(x)$

Reciprocal Polynomials

The following statements are equivalent:

1. If $\alpha \in F_{q^r}$ is a nonzero root of $f(x) \in F_p[x]$, then α^{-1} is a root of the **reciprocal polynomial** of $f(x)$
2. Polynomial is irreducible $\iff$ reciprocal polynomial is irreducible
3. If $m(x)$ is the minimal polynomial of some nonzero $\alpha \in F_{p^r}$, then a scalar multiple of the reciprocal polynomial of α is a minimal polynomial of α^{-1}
4. A polynomial is primitive $\implies$ a scalar multiple of its reciprocal polynomial is primitive

Alternate Interpretation of Finite Fields

Consider $F_4 = F_2[x]/(x^2+x+1)$, with elements $\{0, 1, x, x+1\}$. $\alpha = x$ is a primitive element in this field (since $x^2+x+1=0 \implies x=1$), we can "solve" $x^2+x+1=0$ using the quadratic formula to find $\alpha = \dfrac{-1+\sqrt{1-4}}{2} = -\dfrac{1}{2} + i\dfrac{\sqrt{3}}{2}$, i.e. we treat α like a complex number.

- The other root is $\overline{\alpha}$, so we have $x^2+x+1 = (x-\alpha)(x-\overline{\alpha})$
- So, it follows that $1 = \alpha\overline{\alpha} \implies |\alpha|^2 = 1$, implying that $\alpha = e^{i\theta} = \cos\theta + i\sin\theta$ for some $\theta \in \mathbb{R}$
- By inspection, we find $\theta = \dfrac{2\pi}{3}$ works, so $\alpha^3 = e^{2i\pi} = 1$; α is a primitive *third root of unity*.

Being a third root of unity is equivalent to being a primitive element of F_4; we can think of F_4 as $\{0, 1, x, x+1\}$ or $\{0, 1, e^{2\pi i/3}, e^{-2\pi i/3}\}$

- Similarly, $F_3 = \{0, 1, 2\} \cong \{0, 1, -1\}$ and $F_5 = \{0, 1, 2, 3, 4\} \cong \{0, 1, i, -1, -i\}$.

Aside: $F_4 = F_2[x]/(x^2+x+1)$ (or more accurately, $F_4 \cong F_2[x]/(x^2+x+1)$) because, as mentioned, every field with the same number elements is isomorphic. By the quotient construction of $F_2[x]/(x^2+x+1)$, it has 4 elements (namely $\{0, 1, x, x+1\}$), so it behaves the same was as any "other" field with 4 elements.

Application: ISBN Codes

An **ISBN-Code** is a 10-digit number $x_1x_2\dots x_{10}$ satisfying $x_{10} = \sum_{i=1}^{9} ix_i \mod 11$

- If a single digit is unknown, we can figure out what it should be; there can only be one digit that satisfies the equation

---

Textbook pages: 31-40, Notes pages: 47-55.

Chapter 4 - Vector Spaces over Finite Fields

> Linear Algebra I Speedrun

For future chapters, we will find it useful to perform operations on codewords themselves, specifically the operations defined in a *vector space*.

For prime power q, we define *scalars* as $\mathrm{GF}(q)$ and *vectors* as $V(n, q) = \mathrm{GF}(q)^n$. We define *vector addition* and *multiplication* as we do for column vectors in linear algebra.

Vector Space Axioms

A **vector space** is a set V (e.g. $V(n, q)$) with operations $+$ and $\cdot$ satisfying the following properties:

1. Closure under $+$
2. Associative $+$
3. Additive identity $\vec{0}$
4. Additive inverse $= \vec{u}$
5. Commutative $+$
6. Closure under $\cdot$
7. Distributive law: $a(\vec{u} + \vec{v}) = a\vec{u} + a\vec{v}, (a + b)\vec{u} = a\vec{u} + b\vec{u}$
8. Associative $\cdot$
9. Multiplicative identity

Note that commutative multiplication and multiplicative inverses were *not* defined

Aside: properties 1-5 define a vector space as an *abelian group under* $\cdot$.

A **subspace** is a subset of a vector space that is also a vector space. A subset $V_0 \subseteq V$ of a vector space is a *subspace* iff it is closed under $+$ and $\cdot$.

- The set of all linear combinations of a subset of vectors in $V(n, q)$ is clearly a subspace of $V(n, q)$.

A set $\{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_r\}$ are **linearly dependent** if there exist scalars $a_1, a_2, \ldots, a_r$ such that $a_1\vec{v}_1 + a_2\vec{v}_2 + \cdots + a_r\vec{v}_r = \vec{0}$.

- Therefore one of the vectors in $\{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_r\}$ can be written as a linear combination of the others

- If such scalars $a_1, a_2, \ldots, a_r$ don't exist, $\{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_r\}$ is **linearly independent**. If this is the case, we have the implication $a_1\vec{v}_1 + a_2\vec{v}_2 + \cdots + a_r\vec{v}_r = \vec{0} \implies a_1, a_2, \ldots, a_r = 0$.

A **basis** of vector space C is a linearly independent set of vectors in C that generate C, i.e. a *minimal generating set*.

- E.g. $\{\begin{bmatrix}1\\0\\\vdots\\0\end{bmatrix}, \begin{bmatrix}0\\1\\\vdots\\0\end{bmatrix}, \ldots, \begin{bmatrix}0\\0\\\vdots\\1\end{bmatrix}\}$ is a basis of $V(n, q)$.
- Every vector in C can be *uniquely* represented as a linear combination of basis vectors.
- If C is a non-trivial subspace of $V(n, q)$, then any generating set of C contains a basis of C; this basis is formed by removing redundant vectors from the generating set until it is linearly independent.

The **dimension** of C (denoted $\dim C$) is k if a basis for C has k vectors.

- Then, C itself will have q^k vectors since we operate over the field $\mathrm{GF}(q)$.
- So, $\dim V(n, q) = n$.

Pages: 41-45

Chapter 5 - Linear Codes

Linear Code Definitions

Linear Code

A **linear code** C_ℓ over $\mathrm{GF}(q)$ is a subspace of $V(n,q)$ for positive integer n. So, a linear code is *closed under addition and scaling*: for any words $\vec{u}, \vec{v} \in C_\ell$, $\vec{u}+\vec{v} \in C_\ell$ and $a\vec{u} \in C_\ell$ for scalar $a \in \mathrm{GF}(q)$.

- E.g. $C_{\ell 0} = \{000, 011, 101, 110\}$ is a *binary linear code*

A **linear** $[n,k]$ **code** is a k-dimensional subspace of $V(n,q)$. We may also refer to this as a linear $[n,k,d]$ code to specify minimum distance.

- E.g. $C_{\ell 0}$ defined above is a $[3,2,2]$ linear code.

A linear code must contain $\vec{0}$ by the definition of a vector (sub)space.

Weight

The **weight** $w(\vec{v})$ of a vector $\vec{v}$ in a linear code is the number of non-zero components of $\vec{v}$, i.e. $w(\vec{v}) = d(\vec{0}, \vec{v})$

- For $\vec{u}, \vec{u}$ in a linear code, $d(\vec{v}, \vec{y}) = w(\vec{x} - \vec{y})$
- `THEOREM 5.2` Thus, the minimum distance $d(C)$ of a linear code is the *smallest weight of non-zero codeword*, i.e. $d(C) = w(C)$

But why Tho?

Advantages of Linear Codes

- Finding the minimum distance $d(C)$ of the code requires checking $M-1 \in \Theta(M)$ codeword weights instead of making $\binom{M}{2} \in \Theta(M^2)$ comparisons
- We can specify a linear code by providing a *basis* for it, instead of listing all the codewords like we would for a general code
- Encoding and decoding linear codes is elegant; decoding a general code can be clunky

Disadvantages of Linear Codes

- Linear q-ary codes are only defined when q is a prime power.
  - In practice, selecting a slightly larger q then necessary isn't a big issue though
- There exist strong(er) codes that aren't linear, so a linear code might not be optimal (e.g. $A_q(n, d)$ might be defined by a non-linear code)

Generator Matrices

The **generator matrix** G_C of a linear code C_ℓ is a $k \times n$ matrix whose *rows* form the *basis* of a linear $[n, k]$ code.

- E.g. $C_{\ell 0}$ has 2×3 generator matrix $\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$
- E.g. a q-ary repetition code of length n is a $[n, 1, n]$ code with generator matrix $\begin{bmatrix} 1 & 1 & \dots & 1 \end{bmatrix}$

Equivalence of Linear Codes

Two linear codes are **equivalent** if one can be obtained from the other by *permuting the positions of the code* or *scaling symbols in a fixed position*

> **Theorem 5.4**
>
> Two $k \times n$ matrices generate *equivalent linear codes* over $\mathrm{GF}(q)$ if one matrix is obtainable from the other by
>
> 1. `R1` Permuting the rows
> 2. `R2` Scaling a row
> 3. `R3` Adding a scaled row to another row
> 4. `C1` Permuting the columns
> 5. `C2` Scaling a column

- The *row operations* `R1`, `R2`, and `R3` simply modify the basis for the *same code*, i.e. they preserve the *code itself,* not just equivalence
  - Note: since these operations define row reduction, row reduction preserves equivalence
- The *column operations* convert the generator matrix to one for an *equivalent code*

Standard Form

Standard Form

The **standard form** of a generator matrix G_C for code C is $[I_k \mid A]$, where I_k is the $k \times k$ *identity matrix* and A is a $k \times (n-k)$ matrix. Standard form

`THEOREM 5.5` Standard form can be obtained by performing the operations `R1`, `R2`, `R3`, `C1`, `C2` on the generator matrix in question.

- In general, we can find the standard form by row reducing until columns for each *standard basis vector* exist, then permuting the columns as necessary
- Another algorithm is outlined in the text on page 51.

Pages: 47-54

Chapter 6 - Encoding and Decoding Linear Codes

Encoding

Let $\vec{u} = u_1u_2 \dots u_k \in V(k, q)$ be one of the q^k possible words. We **encode** $\vec{u}$ by multiplying it by the *generator matrix* G_C for our code C. So, our encoded message is $\vec{u}G = \sum_{i=1}^{k} u_i\vec{r}_i$ where $\vec{r}_i$ is the ith row of G_C.

- So, our encoding is a *function* $E : V(k, q) \to C$ that maps $\vec{u} \mapsto \vec{u}G_C$

When G is in standard form (i.e. $G = [I_k|A]$), then $\vec{u}G = x_1x_2 \dots x_k\tilde{x}_{k+1}\tilde{x}_{k+2} \dots \tilde{x}_{k+n}$, where $\tilde{x}_{k+i} = \sum_{j=1}^{k} a_{ji}u_j$, a_{ji} being the (j, i)th entry of A.

- The first k digits are just the message itself (*message digits*); the rest of the digits are **check digits** that exist as redundancies to protect the message against noise. This clearly illustrates the purpose of encoding.

Decoding

For sent vector $\vec{x}$ and received vector $\vec{y}$, we define the **error vector** $\vec{e}$ as $\vec{y} - \vec{x}$.

Cosets

For $[n, k]$ linear code over $\mathrm{GF}(q)$ and vector $\vec{a} \in V(n, q)$, we define the **coset** $\vec{a} + C$ of C as $\vec{a} + C = \{\vec{a} + \vec{x} \mid \vec{x} \in C\}$.

- E.g. the cosets of $C_{\ell 0} = \{000, 011, 101, 110\}$ are $000 + C_{\ell 0} = C_{\ell 0}$ (i.e. just $C_{\ell 0}$ itself) and $100 + C_{\ell 0} = \{100, 111, 001, 010\}$. Note how every vector in $V(k, q) = \{0, 1\}^3$ is in one of these cosets.
- Aside: *cosets* and *equivalence classes* are different terms for the same thing; $\vec{a} + C$ is the **equivalence class** $[\vec{a}]$ of $\vec{a}$ with respect to C.

Lagrange's Theorem (Theorem 6.4)

For $[n, k]$ code C over $\mathrm{GF}(q)$:

- Every vector $\vec{\_} \in V(n, q)$ is in some coset of C
- Every coset of C contains exactly q^k vectors

- There is no partial overlap of cosets: either cosets are the same or entirely disjoint.

- This implies that $V(n,q)$ is *partitioned* by cosets of any of its subspaces

For a given coset, the vector with the smallest weight is the **coset leader**.

- E.g. the coset leader of $C_{\ell 0}$ is 000, and the coset leader of $100 + C_{\ell 0}$ is 100 (or 001)
- Multiple vectors may be of this minimum weight; picking one at random to be the coset leader suffices

Slepian Array

Slepian / Standard Array

The **Slepian** or **standard array** of a linear $[n,k]$ code C is the (a) $q^{n-k} \times q^k$ array of containing all the vectors in $V(n,q)$ where

- The first row consists of the codewords of C, starting with $\vec{0}$
- The first row consists of the coset leaders of each coset defined by C
- Each row is a coset $\vec{a_i} + C$

In particular, the we order the cosets such that $A[i,j] = A[i,1] + A[1,j]$

The Slepian can be constructed as follows

1. List the codewords of C, starting with $\vec{0}$
2. Chose the word $\vec{a} \in V(n,q)$ of the smallest weight that isn't already in the array. List the coset $\vec{a} + C$ in that row, where $\vec{a} + \vec{x}$ is under $\vec{x}$ for each $\vec{x}$ in the first row.
3. Keep repeating 2) until the array is complete.

E.g. the Slepian of $C_{\ell 0}$ is $\begin{bmatrix} 000 & 011 & 101 & 110 \\ 100 & 111 & 101 & 110 \end{bmatrix}$

Decoding a Linear Code

> Finally

We **decode** received vector y by finding it in the Slepian. The vector at the beginning of its row is the *error vector* $\vec{e} = \vec{y} - \vec{x}$, so the first vector in its column will be the *nearest neighbour in* C, and thus the decoded vector since $\vec{x} + \vec{e} = \vec{y}$

- So, the *decoded vector* is the first vector in $\vec{y}$'s column.

Probability of Error Correction

For binary $[n, k]$ code C with α_i coset leaders of *weight* i (for $i \in \{0, 1, \ldots, n\}$), then the *probability* $P_{\text{corr}}(C)$ *that an arbitrary codeword is decoded correctly* is $P_{\text{corr}}(C) = \sum_{i=0}^{n} \alpha_i p^i (1-p)^{n-i}$, where p is the probability of a bit being flipped due to channel noise.

- The **error rate** $P_{\text{err}}(C)$ of C is defined as $P_{\text{err}}(C) = 1 - P_{\text{corr}}(C)$

Pages: 55-61

Chapter 7 - Dual Codes, Parity-Check Matrices and Syndrome Decoding

Dual Codes

The **dual code** $C^\perp$ of linear $[n, k]$ code C is the set of vectors $\vec{v} \in V(n, q)$ that are *orthogonal* to every codeword of C, i.e. $C^\perp = \{\vec{v} \in V(n, q) \mid \vec{v} \cdot \vec{u} = 0 \text{ for all } \vec{u} \in C\}$

- E.g. for $C_{\ell 0} = \{000, 011, 101, 110\}$, $C^\perp = \{000, 111\}$ can be found by inspection.

`LEMMA 7.2` If such C has generator matrix G, then $\vec{v} \in C^\perp$ if and only if $\vec{v}G^T = \vec{0}$, where G^T is the *transpose* of G.

`THEOREM 7.3` $C^\perp$ is a *linear code of dimension* $n - k$, i.e. $C^\perp$ is a linear $[n, n-k]$ code.

`THEOREM 7.5` For any linear $[n, k]$ code C, $(C^\perp)^\perp = C$

Parity Check Matrices

The **parity-check matrix** H_C for $[n, k]$ code C is a *generator matrix of* $C^\perp$.

- So, H_C is an $(n-k) \times n$ matrix satisfying $G_C {H_C}^T = \mathcal{O}$.
- We can equate $C = \{\vec{x} \in V(n, q) \mid \vec{x} {H_C}^T = \mathcal{O}\}$; thus, we can completely define a linear code by a parity-check matrix, much like we can with its generator matrix.
- E.g. $C_{\ell 0}$ has parity-check matrix $[1 \quad 1 \quad 1]$.

The rows of the parity-check matrix are *parity checks* on the codewords. Namely, they constrain certain linear combinations to be 0, encoding the additional structure built into the codewords.

- E.g. Parity-check matrix $H = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$ defines the code $\{(x_1, x_2, x_3, x_4) \in V(4, 2) \mid x_1 + x_2 = 0, x_3 + x_4 = 0\}$

Finding a Parity-check Matrix

Theorem 7.6

If $G_C = [I_k | A]$ is the standard form of a generator matrix for linear $[n, k]$ code C, then the *parity-check matrix* H_C is defined as $H_C = [-A^T | I_{n-k}]$

- E.g. $C_{\ell 0}$ has 2×3 generator matrix $\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$, so it has parity check matrix $\begin{bmatrix} 1 & 1 \mid I_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$

A parity check matrix H_C is in *standard form* if $H_C = [B \mid I_{n-k}]$

- `THM 7.6` finds parity-check matrices in standard form.
- We can reduce parity-check matrices to standard form like we did for generator matrices

Syndromes

Syndrome Definitions and Theorems

For vector $\vec{y} \in V(n, q)$, its **syndrome** $S(\vec{y})$ is defined as the $1 \times (n-k)$ *row vector* $S(\vec{y}) = \vec{y}{H_C}^T$, where H_C is the *parity-check matrix* of linear $[n, k]$ code C.

- If $S(\vec{y}) = \vec{0}$, then $\vec{y} \in C$, and vice versa

`LEMMA 7.8` Two vectors $\vec{u}$ and $\vec{v}$ are in the same coset of C if and only if they have same syndrome, i.e. $S(\vec{u}) = S(\vec{v})$

- So, there is a *bijection* between cosets and syndromes

Syndrome Decoding

For large n, array decoding is inefficient because it requires searching every entry in the array. As $n \to \infty$, *syndrome decoding* becomes more efficient compared to array decoding because it leverages `LEMMA 7.8` to find the coset of $\vec{y}$ in $O(n)$ time.

First, we must *augment* the standard array by appending the syndrome $S(\vec{e})$ of each coset leader $\vec{e}$ to the end of its corresponding row.

Syndrome Decoding

The **syndrome decoding** algorithms is as follows for received vector $\vec{y}$

1. Calculate the syndrome $S(\vec{y}) = \vec{y}H^T$ of $\vec{y}$.
2. Locate $S(\vec{y})$ in the syndromes column of the array.
3. In the row where $S(\vec{y})$ is located, find $\vec{y}$ and decode as normal, i.e. the column header of this column is the decoded vector.

- Aside: when implementing this, we can get even more efficient: we calculate $S(\vec{y})$, find its coset leader $c(S(\vec{[y]}))$ in the *syndrome lookup table* that has columns for each syndrome and its corresponding coset leader. Then, the decoded vector $\vec{x}$ is $\vec{y} - c(S(\vec{[y]}))$; the structure of the whole Slepian is implied here.

Pages: 67-74

Chapter 8 - Hamming Codes

- Obligatory 3b1b plug: https://www.youtube.com/watch?v=X8jsijhllIA
- Encoding Simulator: https://visualizer-tan.vercel.app/#/heyming

Hamming codes are a family of linear, single-error-correcting codes over any $\mathrm{GF}(q)$ with elegant encoding and decoding schemes. Hamming codes are most conveniently defined by their parity check matrices.

Hamming Code Definition

Binary Hamming Code

The **binary Hamming code** $\mathrm{Ham}(r, 2)$ is the code whose parity-check matrix H has dimensions $r \times (2^r - 1)$ and whose columns are the distinct non-zero vectors of $V(r, 2)$.

- Note that the columns can be in any order; all codes with the same columns are *equivalent*.
  - In general, we write them in increasing order for simplicity
- E.g. A parity-check matrix for $\mathrm{Ham}(2,2) = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$; we see that the corresponding G is $\mathrm{Ham}(2,2) = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$, so $\mathrm{Ham}(2,2)$ is the *binary repetition code*.
- E.g. A parity-check matrix for $\mathrm{Ham}(3,2) = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$.

The **redundancy** $r = n - k$ of the code is the number of check-symbols the code has.

Theorem 8.2

For $r \geq 2$, the binary Hamming code $\mathrm{Ham}(r, 2)$:

1. is a $[2^r - 1, 2^r - 1 - r]$ code
2. has minimum distance 3, and is thus *single-error correcting*
3. Is a *perfect code*

- Proof sketch: 2) follows from every nonzero codeword having a weight of 3 or higher, 3) follows from the sphere packing bound directly

Decoding Hamming Codes

$\text{Ham}(r, 2)$ being perfect implies the following properties

- There are $2^r = n + 1$ coset leaders, which are precisely the vectors of $V(n, 2)$ with a weight of 1 or lower (i.e. $\vec{0}$)
- Thus, the syndrome of $\vec{v} = 0 \dots 010 \dots 0$ where the 1 is at place j is the jth column of H

> **Decoding Hamming Codes**
>
> 1. Calculate the syndrome $S(\vec{y}) = \vec{y}H^T$ of the received vector $\vec{y}$
> 2. If $S(\vec{y}) = \vec{0}$, then (we assume) $\vec{y}$ was the codeword sent, so no error occurred
> 3. If $S(\vec{y}) \neq \vec{0}$, we assume one error occurred; $S(\vec{y})$ is the binary number indicating the position of the error.

- E.g. for the $\text{Ham}(3, 3)$ parity check matrix given earlier, if we receive $\vec{y} = 1101011$, then $S(\vec{y}) = 110$, indicating the error is at position 6. So, y is *decoded* as 1101001.

Extended Binary Hamming Codes

We obtain the **extended binary Hamming code** $\hat{\text{Ham}}(r, 2)$ by adding a parity-check to $\text{Ham}(r, 2)$. These codes are no better at decoding completely (in fact, they are worse because they use an extra bit), but provide more error *detection*, making them better for *incomplete decoding*.

The parity-check matrix $\hat{H}_C$ for $\hat{\text{Ham}}(r, 2)$ is created by right-appending a column of 0s, then bottom-appending a row of 1s to to the parity-check matrix H_C for $\text{Ham}(r, 2)$.

The decoding process is as follows:

- If the *parity bit* (i.e. last bit) of $S(\vec{y})$ is 0
  - If the rest of the bits are also 0, then no errors occurred
  - Otherwise, we assume at least two errors have occurred, which we cannot correct
- If the parity bit of $S(\vec{y})$ is 1
  - If the rest of the bits are 0, assume a single error at the last place
  - Otherwise, there is an error at the place indicated by the binary interpretation of $S(\vec{y})$, like before

Relating $d(C)$ and Linear Independence

> **Theorem 8.4**

For $[n, k]$ linear code C over $\mathrm{GF}(q)$ with parity-check matrix H_C any $d(C)-1$ columns of H_C are *linearly independent*, but any set of $d(C)$ columns of H_C are *linearly dependent*.

- Proof: follows from the property that $\vec{x} \in C \iff \vec{x}H^T = \vec{0}$
- This property characterizes $d(C)$, so we can establish $d(C)$ for any C given H_C.

q-ary Hamming Codes

For $d(C) = k$, any k columns of H_C must be linearly independent. So, for given redundancy r, a $[n, n-r, k]$ code can be constructed by finding a set of nonzero vectors in $V(r, q)$ where any k columns are linearly independent.

For $q = 3$, a vector $\vec{v} \in V(r, q)$ has $q - 1 = 2$ nonzero scalar multiples, so can be partitioned into $\dfrac{q^r - 1}{q - 1}$ *equivalence classes*, where $\vec{u} \sim \vec{v} \iff \vec{u} = \lambda\vec{v}$ for some λ, i.e. $\vec{u}$ and $\vec{v}$ are linearly dependent. We form H_C by taking one column from each equivalence class.

- Any different matrices generated this way are equivalent.
- Aside: this is a quotient structure.

Finding H_C for a q-ary Hamming code

A parity-check matrix H_C for $\mathrm{Ham}(r, q)$ can be formed by listing all the nonzero r-tuples in $V(r, q)$ whose first nonzero entry is 1.

- E.g. $\mathrm{Ham}(2, 3)$ has parity-check matrix $H_C = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 2 \end{bmatrix}$.
- E.g. $\mathrm{Ham}(2, 11)$ has parity-check matrix $H_C = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{bmatrix}$.
- E.g. $\mathrm{Ham}(3, 3)$ has the parity-check matrix $H_C = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\ 1 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \end{bmatrix}$

`THEOREM 8.6` $\mathrm{Ham}(r, q)$ is a *perfect single-error-correcting code*.

- `COROLLARY 8.7` For prime power q and $n = \dfrac{q^r - 1}{q - 1}$, $A_q(n, 3) = q^{n-r}$ for some $r \geq 2$.

Decoding q-ary Hamming Codes

Hamming codes are perfect, single-error correcting codes, so its nonzero coset leaders are the vectors of weight 1 in $V(r,q)$. So, $S(\vec{y})=\vec{0}$ implies no errors and $S(\vec{y})\neq\vec{0}$ implies an (assumed) single error. A coset leader for $\vec{y}$ looks like $0\dots0b0\dots0$, where the b is at the jth entry. So, $S(\vec{y})=b\vec{H}_j$, where H_{Cj} is the jth column of H_C. So, the error is corrected by subtracting b from the jth entry of $\vec{y}$.

- E.g. For $q=5$, $H_C=\begin{bmatrix}0&1&1&1&1&1\\1&0&1&2&3&4\end{bmatrix}$ and received vector $\vec{y}=203031$, we find $S(\vec{y})=\begin{bmatrix}2\\3\end{bmatrix}=2\begin{bmatrix}1\\4\end{bmatrix}$. $\begin{bmatrix}1\\4\end{bmatrix}$ is at the 6th position of H_C, so we decode $\vec{y}$ as 203034.

Shortening Codes

We can **shorten** a code C of length n to code C' of length $n-1$ by selecting any codewords in C with symbol λ at position j (both fixed), then deleting the jth entry from each word to form C'.

- If C is $[n,k,d]$, then C' will be $[n-1,k-1,d']$, where $d\geq d'$
- We get the corresponding parity-check matrix $H_{C'}$ by deleting the corresponding column of H_C

Pages: 81-90

Chapter 12 - Cyclic Codes

> Insert Ring Cycle pun here

A linear code C is a **cyclic code** if each *cyclic shift* of a codeword is also a codeword, i.e. for any codeword $a_1a_2a_3a_4\dots a_n = \vec{a} \in C$, $a_na_1a_2a_3\dots a_4 \in C$ as well.

- E.g. $C_{\ell 0} = \{000, 101, 011, 110\}$ is a cyclic code
- E.g. $\mathrm{Ham}(3, 2)$ is a cyclic code

Often, for non cyclic C, we can find an *equivalent* cyclic C' by interchanging coordinates.

Polynomials

$\mathrm{GF}(q)[x]$, now denoted $F[x]$ is the set of polynomials with coefficients in F.
$f(x) = f_0 + f_1x + \cdots + f_mx^m \in F[x]$ has *degree* m, denoted $\deg f(x) = m$, and leading coefficient f_m.
$F[x]$ is a *vector space*, but *not* a field since multiplicative inverses do not exist.

Division Algorithm

For any polynomials $a(x), b(x) \in F[x]$, there exists a *unique* **quotient** $q(x)$ and **remainder** $r(x)$ such that $a(x) = q(x)b(x) + r(x)$, where $\deg r(x) < \deg b(x)$.

- Aside: this is the same structure as the division algorithm for $\mathbb{Z}$ (ring shenanigans…)

The Ring of Polynomials $\mod f(x)$

Polynomials $g(x)$ and $h(x)$ are **congruent** $\mod f(x)$, denoted $g(x) \equiv h(x) \mod f(x)$, if $g(x) - h(x)$ is divisible by $f(x)$, i.e. $f(x) \mid [g(x) - h(x)]$.

We define $F[x]/f(x)$ as the **ring of polynomials over** F **modulo** $f(x)$. This ring's domain comprises every polynomial in $p(x) \in F[x]$ such that $\deg p(x) < \deg f(x)$ (i.e. "smaller" polynomials), and addition and multiplication are "carried out $\mod f(x)$".

- It follows that $|F_q[x]/f(x)| = q^n$.
- E.g. the ring $F_2[x]/(x^2 + x + 1)$ has domain $\{0, 1, x, 1 + x\}$; these are the values that must populate the addition and multiplication tables.

Reducibility

Polynomial $f(x)$ is **reducible** in field $F[x]$ iff there exist $a(x), b(x) \in F[x]$ satisfying $\deg a(x), \deg b(x) < \deg f(x)$ where $f(x) = a(x)b(x)$. Informally, $f(x)$ is reducible if it can be "reduced" into smaller factors.

- $F[x]/f(x)$ is only a field when $f(x)$ is *irreducible* in $F[x]$.
- Irreducibility for polynomials is like primality for integers: any monic polynomial can be factored into a unique set of irreducible polynomials

Lemma 12.3: Useful Observations for Factoring Polynomials

- A polynomial $f(x)$ has linear factor $(x-a)$ iff $f(a) = 0$
- A polynomial $f(x)$ in $F[x]$ of degree 2 or 3 is irreducible if and only if $f(a) \neq 0$ for all a in F.
- Over any field, $x^n - 1 = (x-1)(x^{n-1} + x^{n-2} + \cdots + x + 1)$

Cyclic Codes

Definition and Characterization as Polynomials

We consider the ring $F[x]/(x^n - 1)$, i.e. polynomials modulo $x^n - 1$.

- $x^n \equiv 1$, so we can reduce any polynomial by replacing x^n with 1, x^{n+1} by x, x^{n+1} by x^2, etc
- Multiplying by x corresponds to a cycle shift, multiplying by x^m corresponds to a cycle shift through m positions.
- Polynomials $F[x]/(x^n - 1) \ni a(x) = a_0 + a_1 x + \cdots + a_{n-1}x^{n+1}$ act like (and correspond to) vectors $(a_1, a_2, \ldots, a_{n-1}) \in V(n, q)$. So, we can interpret a code to be a subset of either space; it is algebraically useful to interpret it as a polynomial.

Theorem 12.6 - Characterizing cyclic codes

A code C in R_n is a **cyclic code** if and only if we have, for polynomials in C,

- $a(x), b(x) \in C \implies a(x) + b(x) \in C$
- $a(x) \in C \text{ and } r(x) \in R_n \implies r(x)a(x) \in C$ (note that this is stronger than C simply being closed under multiplication since $r(x)$ is arbitrary in R_n)

- In ring theory terms, cyclic codes are the ideals of the ring R_n.
- We prove $\implies$ by considering $r(x) = 1$ and $r(x) = x$, respectively

For polynomial $f(x)$ in R_n, we define a **cyclic code** $\langle f(x) \rangle$ as the subset of R_n consisting of all (polynomial) multiples of $f(x)$, reduced $\mod x^n - 1$, i.e. $\langle f(x) \rangle = \{r(x)f(x) \mid r(x) \in R_n\}$

- E.g. the code $\langle 1+x^2 \rangle$ in R_3 where $F = \mathrm{GF}(2)$ produces the distinct codewords 0, $1+x$, $1+x^2$, $x+x^2$, so $C = C_{\ell 0} = \{000, 110, 101, 011\}$ from before.

Generator Polynomials

Theorem 12.9

If C is a non-zero cyclic code in R_n, then

- A unique monic polynomial of smallest degree $g(x)$ exists in C
- $C = \langle g(x) \rangle$
- $g(x)$ is a factor of $x^n - 1$

This $g(x)$ is the **generator polynomial** of C

- C may contain other polynomials of larger degree that *also* generate itself, e.g. the generator of $\langle 1+x^2 \rangle$ in R_3 from before is actually $g(x) = 1+x$; both polynomials generate the same code
- In ring theory terms, every ideal in R_n is a *principal ideal*

We find all the cyclic codes in R_3 are $V(3,2)$ in its entirety, $\{000, 110, 011, 101\}$, $\{000, 111\}$, $\{000\}$ with respective generator polynomials 1, $x+1$, x^2+x+1, $x^3-1=0$

- The generator polynomials are pretty good for indicating how the codewords themselves are actually formed; this is somewhat reverse-engineerable.

Theorem 12.12

If C is a cyclic code with generator polynomial $g(x) = g_0 + g_1 x + \cdots + g_r x^r$, then the generator matrix G_C of C is the $n-r \times n-r$ matrix $G = \begin{bmatrix} g_0 & g_1 & g_2 & \cdots & g_r & 0 & 0 & \cdots & 0 \\ 0 & g_0 & g_1 & \cdots & g_{r-1} & g_r & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & g_{r-4} & g_{r-3} & g_{r-2} & g_{r-1} & g_r \end{bmatrix}$,

where row $i < n-r$ is formed by cycling $(g_0, g_1, \ldots, g_r, 0, \ldots, 0)$ i times.

To find all the q-ary cyclic codes of length k, we

- Factorize $x^k - 1$ over $\mathrm{GF}(q)$ into irreducible polynomials
- If $x^k - 1$ has n distinct factors (remember, these are polynomials), it has 2^n divisors (since they each can either be part of the divisor's factorization or not), each generates a cyclic code.
- By Theorem 12.9, these are the only such codes (i.e. the generator polynomial)

- E.g. we find the ternary (so $\mathrm{GF}(3)$) cyclic codes of length 4 by factoring $x^4 - 1 = (x-1)(x+1)(x^2+1)$, implying that there are $2^3 = 8$ divisors of $x^4 - 1$ in $F_3(x)$. Each of these generate a cyclic code.

Check Polynomials and Parity-Check Matrices

The **check polynomial** of C generated by $g(x)$ is the polynomial $h(x)$ such that $g(x)h(x) = x^n - 1$.

- This must exist by Theorem 12.9

For any polynomial $c(x) \in R_n$ is a codeword of code C if and only if $c(x)h(x) = 0$, where $h(x)$ is the parity-check polynomial of C.

- Note how this is how a *parity-check matrix* works; what structure underlies the two concepts?

Note: Although it is true that $\dim \langle h(x) \rangle = \dim C^\perp = n - k$, $h(x)$ does *not* generate the dual code $C^\perp$ of C. Namely, the fact that $h(x)c(x) = 0$ in R_n does *not* carry the same meaning as the corresponding vectors in $V(n, q)$ being orthogonal.

If C is an $[n, k]$ code with parity-check polynomial $h(x) = h_0 + h_1 x + \cdots + h_k x^k$, then:

- $H = \begin{bmatrix} h_k & h_{k-1} & h_{k-2} & \dots & h_0 & 0 & 0 & \dots & 0 \\ 0 & h_k & h_{k-1} & \dots & h_1 & h_0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & h_4 & h_3 & h_2 & h_1 & h_0 \end{bmatrix}$ is a parity-check matrix for C (and thus a generator matrix for $C^\perp$). Note that the structure is similar to the generator matrix for C itself, but with the polynomial's coefficients in reverse order with respect to the generator polynomial $g(x)$.
- The dual code $C^\perp$ is also cyclic and is generated by the **reciprocal polynomial** $\bar{h}(x) = x^k h(x^{-1}) = h_k + h_{k-1}x + \cdots + h_0 x^k$
  - In the non-binary case, we should multiply $\bar{h}(x)$ by h_0^{-1} to make it monic
  - The polynomial $h(x^{-1}) = x^{n-k}\bar{h}(x)$ is a member of $C^\perp$

Hamming Codes Are Cyclic

Irreducible polynomial $p(x)$ of degree r is a **primitive polynomial** iff x is a primitive element in $F[x]/p(x)$. Informally, a primitive element of a finite field is an element that "generates" every member of the field if raised to a high enough power (we will formally define this later).

If $p(x)$ is a primitive polynomial of degree r over $\mathrm{GF}(2)$, then the cyclic code $\langle p(x) \rangle$ is the Hamming code $\mathrm{Ham}(r, 2)$.

- The columns of the parity-check matrix are formed by the binary representations of $1, \alpha, \alpha^2, \alpha^3, \dots$ where $\alpha = x$ is the primitive element of the field

- E.g. $p(x) = x^3 + x + 1$ is irreducible over $\mathrm{GF}(2)$, so $F_2[x]/(x^3 + x + 1)$ is a field of order 8. We note that x is a primitive element of the field since $F_2[x]/(x^3 + x + 1) = \{0, 1, x, x^2, x^3 = x + 1, x^4 = x^2 + x, x^5 = x^2 + x + 1, x^6 = x^2 + 1\}$. From this, we find the parity-check matrix $H = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$, which is clearly a (cyclic version of a) Hamming code parity-check matrix, namely $\mathrm{Ham}(3, 2)$. Notice the columns are ordered as described in the first point

More generally, $\mathrm{Ham}(r, q)$ is a cyclic code if r and $q - 1$ are relatively prime.

Pages 141-144, 146-153

Chapter xx - BCH Codes

> Chernousov goes off the rails!

> Hamming codes are cool, but they can only correct one error. We need more…

Basic Definitions

BCH Code Definition

Let element α be of order n in a finite field F_{q^s}. A $[n, d]$ BCH code has length n and *design distance* d is a *cyclic code* generated by the product of distinct minimal polynomials in $F_q[x]$ of elements $\alpha, \alpha^2, \ldots, \alpha^{d-1}$.

- Usually, we take α to be a primitive element of F_{q^s}, so $n = q^s - 1$.
- A BCH code of odd design distance d can correct at least $\frac{d-1}{2}$ errors.

To encode codeword $a_0 a_1 \ldots a_{k-1}$, we represent it by polynomial $f(x) = a_0 + a_1 x + \ldots$ and multiply it by its generator polynomial $g(x)$ to get codeword $c(x) = f(x)g(x)$.

- For the binary case $q = 2$, $g(x)$ is the product of the distinct minimal polynomials of odd powers of primitive element α from 1 to $d - 1$, i.e. $g(x) = \sum_{i=1}^{d-1} m_i(x)$

Example

The field $F_{2^4} = F_2[x]/(x^4 + x + 1)$ has primitive element $\alpha = x$. We can construct a $[15, 7]$ code that can correct 2 errors by finding a generator polynomial $g(x)$ with roots $\alpha, \alpha^2, \alpha^3, \alpha^4$. We find such a $g(x)$ in the product of the minimal polynomials of α and α^3:
$g(x) = m_1(x)m_3(x) = (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1) = x^8 + x^7 + x^6 + x^4 + 1$.

Decoding BCH Codes

Decoding BCH codes is polynomially analogous to syndrome decoding: for sent codeword $c(x)$ and received codeword $y(x)$, we define the **error polynomial** $e(x) = y(x) - c(x)$.

- We can write $e(x)$ as $x^{\ell_1} + x^{\ell_2} + \cdots + x^{\ell_t}$ for some powers $\ell_1, \ell_2, \ldots, \ell_t$.

The **first syndrome** S_1 is computed by substituting α into $y(x)$: $S_1 := y(\alpha) = c(\alpha) + e(\alpha)$

- We know $c(\alpha) = 0$ by the definition of a codeword since α is primitive
- So $S_1 = \cdots = e(\alpha) = e_1 + e_2 + \cdots + e_t$ where $e_i = \alpha^{\ell_i}$ for $i \leq t$.

We can define each subsequent syndrome (up to syndrome $d-1$) by using the corresponding power of α: for $k \geq d-1$, $S_k = y(\alpha^k) = c(\alpha^k) + e(\alpha^k) = e(\alpha^k) = e_1^k + e_2^k + \cdots + e_t^k$

BCH Decoding Scheme

To decode a BCH code, we must determine if there is a value of t and choices of field elements $e_1, e_2, \ldots, e_t$ that are consistent with all the syndromes, i.e. $S_1, S_2, \ldots, S_{d-1}$.

- If a solution exists, the powers in $\ell_1, \ldots, \ell_t$ where $e_i = \alpha^{\ell_i}$ tell us directly which bits need be toggled

We define the **error locator polynomial**
$\sigma(x) := (e_1x - 1)(e_2x - 1)\ldots(e_tx - 1) = b_tx^t + b_{t-1}x^{t-1} + \cdots + b_1x + 1$.

- Notice that the roots of this polynomial are the inverses of $e_1, e_2, \ldots, e_t$

Example

We wish to decode the $[15, 7]$ code generated by $g(x) = x^9 + x^6 + x^5 + x^4 + x + 1$. Assume our message is $110\,0000$, so we transmit $110\,011\,100\,100\,000$ (i.e. $c(x) = (1 + x)g(x)$). Say we receive $110\,010\,101\,100\,000$, which has 2 errors.

- From the received word, we get $y(x) = x^9 + x^8 + x^6 + x^4 + x + 1$
- We compute the syndromes and reduce using the power table for $F_2[x]/(x^4 + x + 1)$: $S_1 = y(\alpha) = \alpha^9 + \alpha^8 + \alpha^4 + \alpha + 1 = \cdots = \alpha^4$, $S_2 = \cdots = \alpha^8$, etc. up to S_4
- We note that $S_1S_3 - S_2^2 = S_1(S_3 - S_1^3) = S_1(\alpha^7 - \alpha^{12}) \neq 0$. So, we get the system of equations $\begin{bmatrix} \alpha^4b_2 + \alpha^8b_1 = \alpha^7 \\ \alpha^8b_2 + \alpha^7b_1 = \alpha \end{bmatrix}$. We can solve this to define b_1, b_2 in terms of powers of α
- We find the error polynomial is $\sigma(x) = \alpha^{13}x^2 + \alpha^4 + x + 1 = (e_1x - 1)(e_2x - 1)$. We find the roots by simply searching for i where $\alpha^{2i-2} + \alpha^{i+4} = 1$. In this case, $i = 7$
- The inverse of this root is α^8, and since we also know $e_1e_2 = b_2 = \alpha^{13}$, we have $e_2 = \alpha^5$.
- So, the errors are in positions 5 and 8.

Chapter yy - Golay Codes

> Hamming didn't finish his homework

There exist other nontrivial, non-Hamming codes with the same parameters as Hamming codes that also satisfy the sphere-packing bound $M\sum_{k=0}^{t}\binom{n}{k}(q-1)^k = q^n$ (i.e. are perfect codes).

- Two such codes were discovered by Golay in 1949

Non-Hamming Triples

Golay discovered three other triples (n, M, d) satisfying the sphere-packing bound that are not parameters of a Hamming code:

- $(23, 2^{12}, 7)$ for $q = 2$
- $(90, 2^{78}, 5)$ for $q = 2$
- $(11, 3^6, 5)$ for $q = 3$

(non-perfect) linear codes for $(23, 2^{12}, 7)$ (binary) and $(11, 3^6, 5)$ (ternary) exist; these are the **Golay codes**.

These triples also appear to be combinatorial results in addition to coding-theoretical (algebraic) ones

The $[23, 12, 7]$ Binary Golay Code G_{24}

It is convenient to extend the Golay $[23, 12, 7]$ code into the *extended Golay* $[23, 12, 8]$ code by adding an extra parity bit that makes the weight of every codeword even.

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

> The extended Golay $[23, 12, 8]$ code (I didn't want to copy this into TeX myself)

- Note: we can express G_{24} as $[I_{12}|A]$ where A is a 12×12 symmetric matrix (i.e. $A = A^\top$)
- Note: every row of G_{24} is orthogonal to every other

The extended Golay $[23, 12, 8]$ code generated by G_{24} has distance 8.

Self-Orthogonality

Any two rows of the matrix representing G_{24} are orthogonal to each other.

- This can be proven by showing the first row is orthogonal to itself, then using the cyclic symmetry of A' (formed from "A" by removing the first row and column).

A linear code C is **self-orthogonal** iff $C \subset C^\perp$ and **self-dual** iff $C = C^\perp$

We find that since $C_{24}, C_{24}^\perp$ have the same dimension and $C_{24} = C_{24}^\perp$, the parity check matrix $H_{24} = [A|I_{12}]$ is a generator matrix for C_{24}.

The $[11, 6]$ Ternary Golay Code G_{12}

$$G_{12} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 2 & 2 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 2 & 2 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 2 & 1 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 2 & 2 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 1 & 0 \end{bmatrix}$$

- Minimum distance: 5.