ALPHABET SOUP

ROBERT GREEN

Abstract. This paper is an expository piece on algebraic methods in the theory of error-correcting codes. We will assume familiarity with Galois theory and linear algebra as well as basic properties of (finite) groups, rings, fields, and probability spaces.

Algebraic structure is important in coding theory because, by assigning structure to sets by defining operations on them, we can describe sets of encoded messages by their "generators" instead of naively, by listing out each of the elements. Taking advantage of this algebraic structure that we assign to sets allows us to create much more efficient "interpretation" or "decoding" algorithms that take (possibly slightly corrupted) messages and correct them to the original messages that were sent.

"Computer science today is a race between software engineers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning."

- Rick Cook

Date: August 2016.

Contents

1. Introduction & Motivation

1.1. **Hat Problem.** There are n people in a room, and each person is given a black or white hat chosen uniformly at random and independently from all other choices. Each person can see every other person's hat, but not his/her own. We play a game as follows: each player has the option to guess his/her hat colour, or to abstain from guessing. In order to win, at least one player must guess, and every player who guesses must guess correctly. If a single person guesses incorrectly, or if everyone abstains, the game is lost. Players may strategise before hats are handed out, though once hats are assigned they may not communicate.

It is trivial to come up with a strategy that wins this game with probability $\frac{1}{2}$: one player guesses and all others abstain. The question is, can we win this game with probability better than $\frac{1}{2}$? At a glance, the answer would seem to be no – if all of the choices are independent, then looking at other players' hats conveys no information about your own hat, and having more than one person guess can only reduce the probability of winning... right?

Wrong. In fact, for all $n > 2$ that the players can win with probability strictly greater than $\frac{1}{2}$. Furthermore, for infinitely many n, the players can win with probability $\frac{n}{n+1}$. Let's see how this works in the case of $n = 3$, and then once we've studied the Hamming Codes, see if we can generalize.

First, we look at all possible outcomes of hat assignments. Encoding white hats as 0 and black hats as 1, the $2^3 = 8$ possible outcomes are:

$$000\ ,\ 001\ ,\ 010\ ,\ 100$$
$$111\ ,\ 110\ ,\ 101\ ,\ 011$$

Now, we make the crucial observation that, since each guess on its own has probability $\frac{1}{2}$ of being correct, it follows that any strategy must produce an equal number of correct and incorrect guesses when summed across all games. So, what we want to do here is to come up with a strategy that concentrates all of the incorrect guesses in very few games, while spreading the correct guesses over a large number of games. By symmetry, the winning strategy should not regard the specific colours of the hats; rather, their similarity to other hats. We note two games that stand out from the other six in that they do not have an "odd man out": 000 and 111. Can we somehow concentrate the incorrect guesses in these two games? The answer is yes, and after some playing around, we realize the following strategy:

Algorithm 1.1. (Guessing strategy for all players)

```
IF other two players have same hat colour
    Guess that your hat colour is the opposite
ELSE
    Abstain
```

Observe that this strategy loses games 000 and games 111, with every player guessing and guessing incorrectly. Thus, those games are lost. However, in the other six games, the only player who guesses is the "odd man out", and (s)he guesses correctly. Thus, these games are won, and we have produced a strategy that wins this game with probability $\frac{6}{8} = \frac{3}{4}$. We cannot do any better than this because of our earlier observation: we cannot concentrate 7 incorrect guesses into a single game, since each game has a maximum of 3 people guessing.

1.2. **The Paradigm of Coding Theory.** So what are codes and why do we care about them? Here's the idea: we have a sender and a recipient, and they are separated by a communication channel, along which they will send information in chunks called "codewords" (think of sending radio waves across space, or sending electrical pulses through a circuit, or storing data on a hard drive to be retrieved later). The codewords are strings of letters of fixed length (denoted by n) over some alphabet that have been designated to "represent" words. Sometimes, though, the message that is sent gets slightly "corrupted" (for instance, radio waves interfere with each other; circuits may short or lose an electrical signal to resistance, and the materials that comprise hard drives may degrade over time). In terms of the codewords, this means that some of the letters may get erased or changed to other letters. Thus, we want to create sets of codewords that are immune to this sort of "corruption", i.e. even if the words are slightly corrupted, it is still possible for the recipient to recover the set of words that were sent.

1.3. **What does it mean for a code to be "good"?** For fixed word length n, there is a tradeoff between the number of errors that a code can correct and the number of words in the code. This should be obvious: the "farther away" we require the codewords to be from each other, the fewer codewords we can have. Intuitively, we want a "good" code to be one which achieves the best ratio of words to distance. We will see below some basic bounds on this ratio.

1.4. **Notation & Definitions.**

Definition 1.2. Let Σ be a finite set. For ease of notation, we will refer to the elements of Σ as $0, 1, 2, 3, ..., (|\Sigma| - 1)$ unless otherwise specified. A *code* C of length n over Σ is a subset $C \subset \Sigma^n$. In this case, Σ is referred to as the *alphabet.* The elements of C are called codewords. When $|\Sigma| = q$, we call C a $q-$ary code of length n.

Above, we appealed to a notion of "distance" between codewords. We now formalize this notion of distance.

Definition 1.3. Let $\mathbf{x},\ \mathbf{y} \in \Sigma^n$. We define $\Delta(\mathbf{x}, \mathbf{y})$, the *Hamming distance* between $\mathbf{x}$ and $\mathbf{y}$:

$$\Delta(\mathbf{x},\ \mathbf{y}) = |\{i : x_i \neq y_i\}|$$

The Hamming distance between $\mathbf{x}$ and $\mathbf{y}$ is the number of entries in which $\mathbf{x}$ and $\mathbf{y}$ differ. It is trivial to check that Δ defines a metric. We may use d instead of Δ when it is clear that the metric we are talking about is the Hamming metric.

Definition 1.4. The *Hamming weight* of a codeword $\mathbf{x}$ is defined

$$w(\mathbf{x}) = |\{i : x_i \neq 0\}|$$

Note that $w(\mathbf{x}) = \Delta(\mathbf{x},\ \mathbf{0})$ and $\Delta(\mathbf{x}, \mathbf{y}) = w(\mathbf{x} - \mathbf{y})$.

Definition 1.5. The *relative distance* between $\mathbf{x}$ and $\mathbf{y} \in C$ is defined by

$$\delta(\mathbf{x},\ \mathbf{y}) = \frac{\Delta(\mathbf{x},\ \mathbf{y})}{n}$$

Definition 1.6. The *distance* or *minimum distance* of a code C, denoted $\Delta(C)$, is the minimum over all pairs $\mathbf{x},\ \mathbf{y} \in C$ of $\Delta(\mathbf{x},\ \mathbf{y})$.

The minimum distance measures a code's "resilience" to error. Observe that a code of distance d can uniquely correct errors in up to $\frac{d-1}{2}$ entries.

Definition 1.7. The *rate* of a code C, denoted $R(C)$, is given by

$$R(C) = \frac{\log |C|}{n \cdot \log |\Sigma|}$$

The rate measures a code's level of "redundancy": the rate is the amount of non-redundant information per bit.

Definition 1.8. The *dimension* of a code C, denoted $\dim(C)$, is defined as

$$\dim(C) = \frac{\log |C|}{\log |\Sigma|}$$

The notion of dimension makes a lot more sense in the context of algebraic codes.

Notation: A code of length n, dimension k, and distance d over an alphabet of size q is called an $[n, k, d]_q$ code. If the distance is clear or irrelevant, we will use the notation $[n, k]_q$.

In general, the best way to describe codes is simply by listing out all of their elements. However, if we can assign some sort of algebraic structure to Σ^n and specify that codes must be substructures of Σ^n, then we can describe codes much more succinctly with a "small" set of generators. This brings us to our next section, and ultimately to the focus of this paper: algebraic coding theory.

2. Linear Codes

Unfortunately, for most codes, we can only describe them by listing out all of their elements. However, when $|\Sigma| = p^k$ for prime p and positive integer k, we can define addition and multiplication operations, endowing Σ with field structure and giving Σ^n the structure of a vector space. We then restrict codes to being sub*spaces* of Σ^n (instead of sub*sets*) so that we can describe our codes by basis vectors and linear transformations.

Note: there is no inherent reason to study algebraic codes over any other sort of codes, and in fact in many cases, the "best" codes are not algebraic. It just so happens that algebraic codes are easier to describe in general settings, and thus easier to decode. We have not discovered good methods for studying and decoding non-algebraic codes.

Definition 2.1. Let Σ be a (Galois) field. Then a *linear code* of length n over Σ is a subspace $C \leq \Sigma^n$.

From now on, we will restrict our focus to linear codes and algebraic methods. We will denote the Galois field on q elements by $\mathbb{F}_q$ or $GF(q)$.

Observation: The minimum distance of a linear code is equal to the minimum Hamming weight over all codewords. This is because linear codes are closed under subtraction, and $w(\mathbf{x} - \mathbf{y}) = \Delta(\mathbf{x}, \mathbf{y})$.

Once we have adopted this notion of codes as subspaces, we can define codes by their *generator matrices* and their *parity check matrices.* Typically generator matrices will be referred to by the letter G, and parity check matrices by the letter H.

Definition 2.2. A *generator matrix* for a linear code C is a matrix whose rows form a basis for C.

Definition 2.3. A *parity check matrix* for a linear code C is a matrix H such that $H\mathbf{x} = \mathbf{0}$ if and only if $\mathbf{x}$ is a codeword of C. That is, C is precisely the nullspace of the linear transformation defined by H.

Definition 2.4. The vector $H\mathbf{y}$ is called the *syndrome* of $\mathbf{y}$.

Observe that neither generator matrices nor parity check matrices are unique.

Lemma 2.5. *If C is a linear code and H is a parity check matrix for C, then $\Delta(C)$ is the minimum number of columns of H that are linearly dependent.*

Proof. Suppose WLOG that the first d columns of H are linearly dependent. Construct a codeword $\mathbf{x}$ that has ones in the first d entries and zeros elsewhere. Then $w(\mathbf{x}) = d$ and $H\mathbf{x} = \mathbf{0}$. Conversely, if there were a codeword of weight less than d, the support of that codeword would correspond to a linearly dependent set of columns of size less than d. □

The statement we have proven here is in fact stronger than our lemma, because its proof is constructive.

Theorem 2.6. *A code C contains a nonzero codeword of Hamming weight w or less if and only if there is a linearly dependent set of w columns of H.*

2.1. **Systematic & Non-systematic Encoding Schemes.** Suppose we have a set $R \subset \Sigma^k$ of raw data in its most compressed form, and we want to encode the data as codewords over Σ^n so as to send it through a communication channel.

Definition 2.7. An encoding scheme $E : \Sigma^k \to \Sigma^n$ is said to be *systematic* if a dataword $\mathbf{c} \in \Sigma^k$ is mapped to a vector whose first k coordinates are the k coordinates of $\mathbf{c}$, and its last $n-k$ coordinates are linear combinations of the first k coordinates. The last $n-k$ coordinates are called *check coordinates.* An encoding scheme is said to be *non-systematic* if it is not systematic.

Theorem 2.8. *Every linear code has a generator matrix that defines a systematic encoding scheme.*

Proof. This is, in fact, simply another way to view change-of-basis, or Gaussian elimination. Use Gaussian elimination to convert G to the form

$$\begin{bmatrix} I_k \\ B \end{bmatrix}$$

where B is a $(n-k) \times k$ matrix. □

2.2. **Orthogonal Codes.** Since linear codes are subspaces of $GF(q)^n$, we have motivation to define orthogonal codes and investigate their properties.

Definition 2.9. Let $C \leq GF(q)^n$ be an $[n,k]_q$ linear code. Then $C^\perp$, the orthogonal code of C, is defined as follows:

$$C^\perp = \{\mathbf{x} \in GF(q)^n : \mathbf{x} \cdot \mathbf{c} = \mathbf{0}, \forall \mathbf{c} \in C\}$$

where $\cdot$ is the standard dot product.

Several properties follow quickly from this definition:

(1) $C^\perp$ is an $[n, n-k]_q$ linear code
(2) $(C^\perp)^\perp = C$
(3) Let G be a generator matrix for C and let H be a parity check matrix for C. Then G^T is a parity check matrix for $C^\perp$ and H^T is a generator matrix for $C^\perp$.

However, since the standard dot product is not positive-definite over fields with finite characteristic, we do not necessarily get the nice properties of Euclidean space where $C \cap C^\perp = \{\mathbf{0}\}$ and $C \cup C^\perp = \mathbb{F}^n$. In fact, there are many codes where $C^\perp \subset C$ or even $C^\perp = C$. An example of this type of code is as follows:

$$C = \{\mathbf{x} \in GF(2)^n : \sum_i x_i \equiv 0\}$$

It is easy to verify that this code is self-orthogonal.

3. Bounds on Codes

3.1. **Hamming Bound.** Let C be a binary code of length n and distance d. Then

$$|C| \leq \frac{2^n}{\sum_{i=0}^{\lfloor \frac{d}{2} \rfloor} \binom{n}{i}}$$

Proof. The proof of this statement follows from the pigeonhole principle: We are trying to cover as much of the set Σ^n as possible with disjoint balls of constant radius. We note that the volume of each ball will be $\sum_{i=0}^{\lfloor \frac{d}{2} \rfloor} \binom{n}{i}$: we are summing the volume of each "shell" around the ball of integer radius up to $\lfloor \frac{d}{2} \rfloor$ - which is the maximum (constant) radius that these balls may have so that all pairs of balls around pairs of points are disjoint. 2^n is the volume of binary n-dimensional space. Thus we divide the volume of the space by the volume of the balls that we can "squeeze" into the space to get an upper bound on the number of codewords. $\square$

It is not hard to extend the Hamming bound to q-ary codes:

Let C be a q-ary code of length n and distance d. Then

$$|C| \leq \frac{q^n}{\sum_{i=0}^{\lfloor \frac{d}{2} \rfloor} \binom{n}{i}(q-1)^i}$$

The reasoning for this is the same as above.

Codes for which the Hamming bound is tight are called *perfect codes.*

Theorem 3.1. ***(Tietavainen, Van Lindt)*** *There are three classes of perfect binary codes:*

1. Hamming Codes
2. The $[23, 12, 7]_2$ Golay Code
3. Trivial codes: Σ^n, single-word codes, and the code $\{00...0, 11...1\}$

We will encounter Hamming codes and Golay codes later in this paper.

3.2. **Gilbert-Varshamov Bound.**

Definition 3.2. Let $A(q, n, d)$ denote the maximum size of a q-ary code of length n and distance d.

Definition 3.3. Let $V(q, n, r)$ denote the volume of a ball of radius r in Σ^n, where $|\Sigma| = q$.

We note that $V(q, n, r)$ does not depend on the centre of the ball.

Lemma 3.4.

$$V(q, n, r) = \sum_{i=0}^{r} \binom{n}{i} (q-1)^i$$

Proof. Fix any string $\mathbf{x}$. The number of ways to deviate from $\mathbf{x}$ in i coordinates is $\binom{n}{i}$, and for each deviation there are $(q-1)$ other letters of the alphabet to choose from. □

Gilbert and Varshamov gave a lower bound on $A(q, n, d)$ in terms of $V(q, n, d-1)$ using a greedy algorithm for constructing codes of distance $\geq d$.

Theorem 3.5. ***(Gilbert, Varshamov)***

$$A(q, n, d) \geq \frac{q^n}{\sum_{i=0}^{d-1} \binom{n}{i} (q-1)^i}$$

Proof. Choose codewords from Σ^n at distance at least d from each other, until it is no longer possible to do so. Once this procedure halts and picks a code C, the balls of radius $d-1$ around codewords must cover the entire space – otherwise there is a point at distance $\geq d$ from all codewords, and it can be added to C. Thus we have $|C| \cdot V(q, n, d-1) \geq q^n$. □

3.3. **Singleton Bound.**

We now give an upper bound on $A(q, n, d)$:

$$A(q, n, d) \leq q^{n-d+1}$$

Proof. Let C be a code of length n and distance d. By assumption, we can delete the first $d-1$ entries of every codeword, and still have a code C' of length $n-d+1$ and distance ≥ 1. Thus all elements of C' are distinct. □

Linear codes for which the Singleton Bound is tight are called *maximum distance separable* (MDS) codes.

Theorem 3.6. *Let C be an $[n, k, d]$ code over $GF(q)$. Then the following are equivalent:*

(1) C is a maximum distance separable code
(2) $C^{\perp}$ is a maximum distance separable code
(3) Any set of k columns of a generator matrix for C are linearly independent
(4) Any set of $n-k$ columns of a parity check matrix for C are linearly independent
(5) Given any d coordinate positions, there exists a minimum weight codeword whose support is precisely those d positions

This is not so much a theorem as an observation. The reader should think about why this observation is true.

3.4. **Johnson Bound.**

Definition 3.7. Let $A(q, n, d, w)$ denote the size of the largest $q-$ary code of length n, distance d, and constant weight w. Let $A'(q, n, d, w)$ denote the size of the largest $q-$ary code of length n, distance d, and weight at most w.

Observe:

$$A(q, n, d) \leq \frac{q^n}{\binom{n}{w}(q-1)^w} A(q, n, d, w)$$

because $\binom{n}{w}(q-1)^w$ is the number of ways to stay on the "shell" of weight w around the origin. This, along with the following two geometric lemmas (stated without proof) help us to prove the Johnson bound:

Lemma 3.8. *Let $\boldsymbol{v}_1, ..., \boldsymbol{v}_m$ be m unit vectors in $\mathbb{R}^n$.*

(1) Suppose $\boldsymbol{v}_i \cdot \boldsymbol{v}_j \leq \varepsilon$ for all $i < j$. Then $m \leq 1 + \frac{1}{\varepsilon}$.
(2) Suppose $\boldsymbol{v}_i \cdot \boldsymbol{v}_j \leq 0$ for all $i < j$. Then $m \leq 2n$.

Lemma 3.9. *Let C be a binary code of length n and distance d.*

(1) If $d > \frac{n}{2}$ then $|C| \leq \frac{2d}{2d-n}$
(2) If $d \geq \frac{n}{2}$ then $|C| \leq 2n$

Theorem 3.10. *(Binary Johnson Bound) For integers $1 \leq w \leq d \leq \frac{n}{2}$, if $w \leq \frac{1}{2}(n - \sqrt{n(n-2d)})$, then $A'(2, n, d, w) \leq 2n$.*

Proof. Let $C = \{\mathbf{c}_1, ..., \mathbf{c}_m\}$ be a binary code of length n, distance d, and maximum weight w. We map the codewords c_i into vectors $v_i \in \mathbb{R}^n$ as follows:

$$\mathbf{v}_i = ((-1)^{c_i[1]}, (-1)^{c_i[2]}, ..., (-1)^{c_i[n]})$$

where $c_i[j]$ is the j^{th} entry of the i^{th} codeword.

Let $\mathbf{1}$ be the all 1's vector. Let $\alpha > 0$ be a parameter to be specified later so that all dot products $(\mathbf{v}_i - \alpha\mathbf{1}) \cdot (\mathbf{v}_j - \alpha\mathbf{1}) \leq 0$ for $i < j$. We have

$$(\mathbf{v}_i - \alpha\mathbf{1}) \cdot (\mathbf{v}_j - \alpha\mathbf{1}) = n - 2\Delta(\mathbf{c}_i, \mathbf{c}_j) + \alpha^2 n + \alpha(2w(\mathbf{c}_i) + 2w(\mathbf{c}_j) - 2n)$$
$$\leq n - 2d + \alpha^2 n + 2\alpha(2w - n)$$

which is at most zero, so long as

$$4w \leq 2n - \left(\alpha n + \frac{n-2d}{\alpha}\right)$$

Choosing $\alpha = \sqrt{n(n-2d)}$ maximizes the right hand side, and gives

$$w \leq \frac{1}{2}(n - \sqrt{n(n-2d)})$$

By the second part of Lemma 3.8, we have $A'(q, n, d, w) \leq 2n$. □

4. Hamming Codes

Hamming codes are a class of binary linear codes. They all have distance 3, and thus can correct only single-bit errors. They tend not to be very useful for this reason, though they do exhibit some beautiful mathematical properties.

Definition 4.1. Define H_r as the $r \times 2^r - 1$ matrix whose i^{th} column is the binary representation of i. The r^{th} Hamming code, C_r is defined as the nullspace of the parity check matrix, H_r.

Lemma 4.2. *C_r is a $[2^r - 1, 2^r - 1 - r, 3]_2$ code.*

Proof. Clearly H_r takes input vectors of length $2^r - 1$. The rank of H_r is r, showing that the dimension of C_r is $2^r - 1 - r$ by rank-nullity. To see that the minimum distance is 3 we observe that the binary representations of 1, 2, and 3 add up to $\mathbf{0}$ (thus satisfying the conditions of Lemma 2.4). □

Note that the Hamming codes meet the Hamming bound, since $|C_r| = \frac{2^{2^r}}{2^r}$ and the length of C_r is $2^r - 1$.

Perhaps the most beautiful characteristic of the Hamming codes, though, is the algorithm that decodes them. Suppose $\mathbf{y}$ is the syndrome of a codeword $\mathbf{x}$, and H is the parity check matrix that we used to define the Hamming codes. Then $\mathbf{y} = \mathbf{x} + \mathbf{e}_i$, where $\mathbf{e}_i$ is some unit vector. Then

$$H(\mathbf{y}) = H(\mathbf{x} + \mathbf{e}_i) = H(\mathbf{x}) + H(\mathbf{e}_i) = H(\mathbf{e}_i)$$

And because of how H is defined, $H(\mathbf{e}_i)$ is the binary representation of i! If there is an error that this code must correct, running a received string through the parity check matrix will tell you the coordinate in which you have an error!

4.1. **Return to the Hat Problem.**

It turns out that the solution to the very first problem presented in this paper relies on binary Hamming codes, and uses the fact that Hamming codes are *perfect* – disjoint balls of radius 1 around each of the codewords cover the entire space $\mathbb{F}_2^n$. The winning strategy is as follows:

Algorithm 4.3. (Guessing strategy for all players)

```
IF other $n - 1$ players have a hat configuration that can be turned into a
Hamming codeword by guessing your hat colour appropriately:
    Guess that your hat colour is the opposite colour necessary to
    complete a Hamming codeword

ELSE
    Abstain
```

We now must show that this strategy is optimal for n of the form $n = 2^r - 1$.

Claim 4.4. For any n, the probability of winning, regardless of strategy, cannot be greater than $\frac{n}{n+1}$

Proof. We appeal to the observation we made earlier regarding the sum over all games of correct and incorrect guesses, and apply the pigeonhole principle. Suppose the max number of games we can win is N. Then $2^n - N$ is the minimum number of games that must be lost. We observe that $2^n - N \geq \frac{N}{n}$ – equality would only hold if we managed to pack all of the incorrect guesses into games where *nobody* guessed correctly, and if each winning game were one with a single guess. Incidentally, this is what we will use the Hamming codes to do to achieve equality, and thus optimal probability of winning.

$$\begin{aligned} 2^n - N &\geq \frac{N}{n} \\ n(2^n - N) &\geq N \\ n2^n &\geq nN + N \\ N &\leq \frac{n}{n+1} \cdot 2^n \end{aligned}$$

□

In order for equality to hold (i.e. for N to be an integer), we require that $n+1$ is a power of 2, since $n+1$ will never divide n. Conveniently enough, this is precisely the constraints on n for which a Hamming code of length n exists. This can be observed by noting that there are $2^r - 1$ nonzero $r-$tuples, and every pair of binary r-tuples is linearly independent because the characteristic of the base field is two.

Claim 4.5. We can use Hamming codewords to construct a strategy that wins with optimal probability for all n of the form $2^r - 1$.

Proof. We log the results of Algorithm 4.3. There are two cases:

(1) The game forms a Hamming codeword
(2) The game does not form a Hamming codeword

In case 1, each player sees that a Hamming codeword can be completed. Each player chooses to guess the letter that does not complete the Hamming codeword. Thus n people guess incorrectly.

In case 2, only one player sees that a Hamming codeword can be completed (this is the player whose bit is "corrupted"). One player must see the possibility for a Hamming codeword to be completed because every string of length n is within Hamming distance 1 of a Hamming codeword. There cannot be more than one player who sees that a Hamming codeword can be completed because if that were the case, the word would in fact be a Hamming codeword, contradicting the condition of case 2.

□

This problem is beautiful and worth mentioning in this paper because it shows the ubiquity of scenarios in which coding theory can be applied, and is easily accessible to anyone who knows a bit of linear algebra. Other examples of applications of coding theory include the proof of the PCP theorem in complexity theory – the statement that every NP problem has a probablistically checkable proof of constant query complexity and logarithmic randomness complexity. In addition, many deep results in group theory and finite projective geometry rely on coding theory.

4.2. Generalized Hamming Codes.

It turns out that Hamming codes can be generalized to $q-$ary alphabets, and do not just exist in the binary case. By Theorem 2.5, the idea is to construct a check matrix such that all pairs of columns are linearly independent. However, for $q \neq 2$, we cannot use all nonzero $r-$tuples of field elements, since some of them will be multiples of each other. We force linear independence by requiring that (for instance) the topmost nonzero element of each column is (for instance) a 1. This adds an extra constraint, and thus the number of r-tuples that satisfy the above constraint is

$$\frac{q^r - 1}{q - 1}$$

Thus, because H_r has rank r, the r^{th} $q-$ary Hamming code has parameters

$$\left[\frac{q^r - 1}{q - 1}, \frac{q^r - 1}{q - 1} - r, 3\right]$$

5. Cyclic Codes and Ideals

It turns out random linear codes attain bound with high probability but no surefire way to decode them efficiently, so we need to construct codes where we can take advantage of more algebraic structure so as to decode efficiently. Cyclic codes are important because they allow us to define a "multiplication" operation on vector spaces, giving the vector space additional structure (specifically that of a field) which can be exploited both for constructing and for decoding good codes. We study cyclic codes over $GF(q)$ by stepping up into an extension field $GF(q^m)$ through the vector space $GF(q)^m$ – just as stepping from $\mathbb{R}$ to $\mathbb{C}$ can yield useful information about $\mathbb{R}$.

Definition 5.1. A cyclic code over $GF(q)$ is a linear code that is closed under cyclic permutation of the coordinate indices. That is, if $(c_0, c_1, \dots, c_{n-1})$ is a codeword in a cyclic code, then $(c_{n-1}, c_0, \dots, c_{n-2})$ is a codeword as well.

Recall that linear codes over $GF(q)$ are defined by their parity check matrices, and a string $\mathbf{c}$ is a codeword if and only if $H\mathbf{c} = \mathbf{0}$.

Take, for example, the parity check matrix for the [7,4] Hamming code:

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

We can identify the columns of H with elements of $GF(8)$ in the following way: the top element represents the coefficient of x^0, the middle entry represents the coefficient of x^1, and the bottom element represents the coefficient of x^2. Now consider $GF(2)[x]/(x^3 + x + 1)$. This is a Galois field because the quotient polynomial is irreducible over $GF(2)$. Taking α to be the primitive element of $GF(8)$ represented by x, the parity check matrix becomes a 1×7 matrix over $GF(8)$:

$$H = \begin{bmatrix} \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \end{bmatrix}$$

Now, by the definition of a parity check matrix, we have defined codewords $\mathbf{c}$ so that the products $H\mathbf{c}$ are $\mathbf{0}$. But in fact, this product is just

$$\sum_{i=0}^{6} c_i\alpha^i = 0$$

.

This motivates us to define a representation of codewords by polynomials:

Definition 5.2. Given a codeword $\mathbf{c}$ of length n, its polynomial representation (called a *codeword polynomial*) is given by

$$\mathbf{c}(x) = \sum_{i=0}^{n-1} c_i x^i$$

This is a fairly natural encoding of codeword vectors as polynomials, and it gives us a a way to "multiply" codeword vectors with each other by multiplying their corresponding polynomials. In the case of the Hamming code example above, the operation of multiplying H with $\mathbf{c}$ becomes evaluating the polynomial $\mathbf{c}(x)$ at $x = \alpha$, and the condition that $\mathbf{c}$ is a codeword becomes the condition that $\mathbf{c}(\alpha) = 0$. Thus, in the example above, the $[7, 4]$ Hamming code is the set of polynomials over $GF(2)$ of degree ≤ 6 that have α as a zero in $GF(8)$.

In general, we can use this technique for compactification of H whenever the number of rows of the parity check matrix, $(n-k)$, is divisible by the degree of the field extension, m. Each set of m rows can be grouped together, encoded naturally as polynomials over an extension field, and represented as a single row over that extension field. The following transformation takes place:

$$H = \begin{bmatrix} h_{11} & \dots & h_{1n} \\ \vdots & \ddots & \vdots \\ h_{(n-k)1} & \dots & h_{(n-k)n} \end{bmatrix} \longmapsto H = \begin{bmatrix} \eta_{11} & \dots & \eta_{1n} \\ \vdots & \ddots & \vdots \\ \eta_{r1} & \dots & \eta_{rn} \end{bmatrix}$$

where $r = \frac{n-k}{m}$ and η_{ij} is the polynomial encoding of $h_{((i-1)m+1)j}$ thru $h_{(im)j}$. We will see that this section of the paper – cyclic codes – concerns the cases in which the modified parity check matrix is a Vandermonde matrix of the form

$$H = \begin{bmatrix} \eta_0^0 & \dots & \eta_0^{n-1} \\ \vdots & \ddots & \vdots \\ \eta_{r-1}^0 & \dots & \eta_{r-1}^{n-1} \end{bmatrix}$$

5.1. **Polynomial Description of Cyclic Codes.**

Recall the definition of a cyclic code, and that each vector in $GF(q)^n$ is canonically represented as a polynomial of degree less than n with coefficients in $GF(q)$. We start with a characterization of cyclic codes in terms of polynomials:

Theorem 5.3. *A code C of length n over $GF(q)$ is cyclic if and only if the following hold:*

(1) The set of codeword polynomials of C (which we will refer to from here on as just "C") is an additive subgroup of $GF(q)[x]/(x^n - 1)$
(2) If $c(x) \in C$ and $a(x) \in GF(q)[x]/(x^n - 1)$, then $c(x)a(x) \in C$

Proof. Suppose C satisfies the above properties. Then it forms a vector subspace over $GF(q)$. It is also closed under multiplication by x, and so C is cyclic.

Now suppose C is cyclic. Then it is closed under linear combinations and is thus a vector subspace and an additive subgroup. So it is closed under multiplication by x, and by linear combinations of the powers of x. Thus C satisfies the above properties. □

There is a unique monic codeword polynomial of smallest degree in C, and we will denote its degree by $n-k$. This polynomial is called a **generator polynomial**, and is usually denoted by $g(x)$.

Corollary 5.4. *Every cyclic code consists of all multiples of a generator polynomial $g(x)$ by polynomials of degree $k - 1$ or lower.*

For those familiar with ring theory, the above theorem is precisely the statement that the set of codeword polynomials of cyclic codes are ideals of the ring $GF(q)[x]/(x^n - 1)$. The corollary follows because this ring is a Euclidean domain with degree norm, so every ideal is generated by a single element.

Theorem 5.5. *There is a cyclic code of length n and generator polynomial $g(x)$ if and only if $g(x)$ divides $x^n - 1$.*

This also follows from the fact that $GF(q)[x]/(x^n - 1)$ is a Euclidean domain and cyclic codes are ideals. Conversely, every polynomial dividing $(x^n - 1)$ can be used to define a cyclic code over $GF(q)$.

We now have motivation to define a check polynomial:

$$x^n - 1 = g(x)h(x)$$

for some polynomial h. We will call h the *check polynomial*, for the reason that it checks the parity of codeword polynomials in the same way that a check matrix checks the parity of codewords - that is, any codeword $\mathbf{c}(x)$ has product zero with $h(x)$ because

$$\mathbf{c}(x)h(x) = a(x)g(x)h(x) = a(x)(x^n - 1) \equiv 0$$

for some polynomial $a(x)$. In this case, $a(x)$ is called the **data polynomial**, since it is, in some sense, the raw, unencoded data.

The most natural way to encode the "raw" $k-$dimensional data is through the map

$$a(x) \mapsto \mathbf{c}(x) = a(x)g(x)$$

The problem with this encoding is that it is nonsystematic – i.e. $a(x)$ is not immediately visible in $\mathbf{c}(x)$. A better, systematic encoding scheme would be

$$a(x) \mapsto \mathbf{c}(x) = a(x) \cdot x^{n-k} + t(x)$$

where $t(x)$ is chosen so that $\mathbf{c}(x) \equiv 0 \pmod{g(x)}$, so

$$t(x) = -R_{g(x)}[x^{n-k}a(x)]$$

.

5.2. **Minimal Polynomials & Conjugates.**

Definition 5.6. A code over $GF(q)$ is said to have *primitive blocklength* if $n = q^m - 1$ for some integer m.

By unique prime factorization,

$$x^{q^m-1} - 1 = f_1(x) \cdot f_2(x) \cdot \dots \cdot f_s(x)$$

Alternatively, the nonzero elements of $GF(q^m)$ are precisely the roots of $x^{q^m-1} - 1$, so

$$\prod_{j=1}^{q^m-1} (x - \beta_j)$$

where the β_j range over $GF(q^m) \setminus \{0\}$.

Theorem 5.7. *Suppose $g(x)$ generates a cyclic code C, and g has zeros at $\beta_1, \dots, \beta_r$ in $GF(q^m)$. Then $\mathbf{c}(x)$ is a codeword polynomial if and only if*

$$\boldsymbol{c}(\beta_1) = \dots = \boldsymbol{c}(\beta_r) = 0$$

.

Proof. Suppose $\mathbf{c}$ is a codeword polynomial. Then $c(\beta_j) = a(\beta_j)g(\beta_j) = 0$.

Conversely, suppose $\mathbf{c}(\beta_j) = 0$. By the division algorithm,

$$\mathbf{c}(\beta_j) = Q(\beta_j)x_j(\beta_j) + r(\beta_j) = r(\beta_j)$$

where Q is the quotient polynomial, r is the remainder polynomial of strictly lower degree than c, and x_j is the minimal polynomial of β_j. Since $c(\beta_j) = 0$, we have that $r(x)$ is identically zero, so $\mathbf{c}(x)$ is divisible by all of the $f_j(x)$, and is thus divisible by $\text{LCM}_j[f_j(x)] = g(x)$. □

Definition 5.8. The set $S = \{\beta_1, \dots, \beta_r\}$ in the above context is called a *complete defining set* of the code C generated by the polynomial $g(x)$. A subset of S consisting of at least one zero of each minimal polynomial is called a *defining set.*

Observe: Suppose α is primitive in $GF(q^m)$. Then to get a complete defining set from a defining set, take all cyclic permutations of the q-ary representations of the exponents of α in the defining set, and raise α to these powers.

We now state some lemmas that will be useful:

Lemma 5.9. *Let $GF(q)$ have characteristic p. Then for any $m \in \mathbb{N}$,*

$$\left[\sum_i c_i x^i\right]^{p^m} = \sum_i c_i^{p^m} x^{ip^m}$$

Proof. The case $m = 1$ follows from the fact that for all j besides $j = 0$ and $j = p$, the binomial coefficient $\binom{p}{j}$ is divisible by p and thus is congruent to 0 mod p. The general case for $m \in \mathbb{N}$ follows by induction. □

Lemma 5.10. *Suppose $f(x)$ is the minimal polynomial of $\beta \in GF(q^m)$ over $GF(q)$. Then f is also the minimal polynomial of β^q.*

Proof. Because q is a power of p, we have

$$(f(x))^q = \sum_{i=0}^{\deg(f)} f_i^q x^{iq} = \sum_{i=0}^{\deg(f)} f_i x^{iq} = f(x^q)$$

where the second equality follows from Fermat's little theorem. Thus we have

$$0^q = (f(\beta))^q = f(\beta^q) = 0$$

□

Lemma 5.11. *The minimal polynomial of β is*

$$f(x) = (x - \beta)(x - \beta^q)(x - \beta^{q^2})...(x - \beta^{q^{r-1}})$$

where r is the minimum integer such that $\beta^{q^r} = \beta$. So $f(x)$ has no zeros besides β and its conjugates.

Proof. By the previous lemma, such a minimal polynomial must have $\beta, \beta^q, ..., \beta^{q^{r-1}}$ as zeros. It now remains to show that this is in fact a polynomial over $GF(q)$:
Consider that

$$(f(x))^q = (x - \beta)^q (x - \beta^q)^q (x - \beta^{q^2})^q ... (x - \beta^{q^{r-1}})^q$$
$$= (x^q - \beta^q)(x^q - \beta^{q^2})(x^q - \beta^{q^3})...(x^q - \beta^{q^{r-1}})(x^q - \beta)$$
$$= f(x^q)$$

We have shown that all of the coefficients f_i lie in $GF(q)$, which implies that this is in fact a minimal polynomial. □

Lemma 5.12. *Suppose n and q are relatively prime. Then $\exists m \in \mathbb{N}$ such that $(x^n - 1)$ divides $(x^{q^m - 1} - 1)$, and $(x^n - 1)$ has n distinct zeros in $GF(q^m)$.*

Proof. We only need to show that n divides $q^m - 1$ for some m, and then use general factorization of $z^b - 1$ with $z = x^n$ to prove the lemma.
Use the division algorithm to write the following equations:

$$\begin{aligned} q &= Q_1 n + r_1 \\ q^2 &= Q_2 n + r_2 \\ \vdots \quad & \quad \vdots \qquad \vdots \\ q^n &= Q_n n + r_n \\ q_{n+1} &= Q_{n+1} n + r_{n+1} \end{aligned}$$

By pigeonhole principle, two of the remainders, r_i and r_j (WLOG $i < j$) must be the same. Then n divides $q^j - q^i = q^i(q^{j-i} - 1)$. By assumption, $\gcd(n, q) = 1$ so n cannot divide q^i and must divide $(q^{j-i} - 1)$. $\square$

The main takeaway from these properties is that we can describe codes of length n over $GF(q)$ in an extension field $GF(q^m)$ when n and q are relatively prime. Suppose $q^m - 1 = nb$ and let $\alpha \in GF(q^m)$ be primitive. Set $\beta = \alpha^b$. Then all zeros of $x^n - 1$ and thus of $g(x)$ are powers of β, so we can use powers of β instead of powers of α to define a cyclic code of length $n = \frac{(q^m-1)}{b}$.

5.3. Hamming Codes as Cyclic Codes.

We finish this section by returning to the Hamming codes and framing them as cyclic codes. The example used to start this section, the $[7, 4]_2$ Hamming code, has generator polynomial $g(x) = x^3 + x + 1$. g has a zero at the primitive element $\alpha \in GF(8)$, and so all codewords satisfy $c(\alpha) = 0$.

In general, to create a binary Hamming code of length $n = 2^r - 1$, we take $g(x)$ to be a polynomial of degree r with a zero at a primitive element $\alpha \in GF(2^r)$.

6. Reed-Solomon and BCH Codes

6.1. Reed-Solomon Codes.

Reed-Solomon codes are a nice class of linear codes which in some cases are cyclic. They exhibit many beautiful and useful properties, though they require large alphabet size. However, Reed-Solomon codes are still used widely because they lend themselves much better to decoding bursts of errors – errors that occur all at once in "bursts" of consecutive indices (think scratches on CDs or bouts of radiowave interference).

Definition 6.1. Let $GF(q)$ be a Galois field on q elements. Let $S = \{\alpha_0, ..., \alpha_{n-1}\} \subset GF(q)$ be distinct points. We define the Reed-Solomon Code $RS[n, k, q, S]$ by its encoding map

$$E : GF(q)^k \to GF(q)^n$$

where a raw message $\mathbf{m} = (m_1, ..., m_k)$ is mapped according to

$$(m_1, ..., m_k) \mapsto (f_m(\alpha_0), ..., f_m(\alpha_{n-1}))$$

where f_m is the data polynomial

$$f_m(x) = \sum_{i=0}^{k-1} m_i x^i$$

In most cases, we will take $S = \mathbb{F}^*$, the nonzero elements of $GF(q)$, and the encoding of a message $\mathbf{m}$ will be given by

$$\mathbf{m} \mapsto V\mathbf{m}$$

where V is the Vandermonde matrix on $\alpha_0, ..., \alpha_{n-1}$:

$$V = \begin{bmatrix} \alpha_0^0 & \dots & \alpha_0^{n-1} \\ \vdots & \ddots & \vdots \\ \alpha_{n-1}^0 & \dots & \alpha_{n-1}^{n-1} \end{bmatrix}$$

Lemma 6.2. *Reed-Solomon Codes meet the Singleton bound: $d = n + k - 1$*

Proof. Every nonzero polynomial of degree $k - 1$ has at most $k - 1$ roots, so if two polynomials agree in more than $k - 1$ places, they are the same polynomial. Suppose $f_m(x)$ and $f_{m'}(x)$ are data polynomials of two distinct (unencoded) words $\mathbf{m}, \mathbf{m'} \in GF(q)^k$. Then $E(\mathbf{m})$ and $E(\mathbf{m'})$ are at distance $n - a$ where a is the number of zeros in $E(\mathbf{m}) - E(\mathbf{m}')$, which is the number of roots that $f_m - f_{m-1}$ have amongst the $\alpha_0, ..., \alpha_{n-1}$. Thus $a \leq k - 1$, and so $d \geq n - k + 1$. By the Singleton bound, $d = n - k + 1$. □

We now turn to a special case of Reed-Solomon codes – ones whose evaluation set is all powers of a primitive element $\alpha \in GF(q)[x]/(x^n - 1)$. We characterize them by their parity check as follows:

Theorem 6.3. *If we have a Galois field on $q = n+1$ elements, a primitive element α, and an evaluation set $S = \{1, \alpha, \alpha^2, ..., \alpha^{n-1}\}$, then*

$$RS[n, k, q, S] = \{\boldsymbol{c} \in GF(q)^n : c(\alpha) = c(\alpha^2) = ... = c(\alpha^{n-k})\}$$

Proof. For a message $\mathbf{m} \in GF(q)^k$, we have

$$f_m(\alpha^j) = m_0 \sum_{i=0}^{n-1} \alpha^{ij} + m_1 \sum_{i=0}^{n-1} \alpha^{i(j+1)} + ... + m_{k-1} \sum_{i=0}^{n-1} \alpha^{i(j+k-1)}$$

and for all $x \in GF(q)^*$ such that $x \neq 1$, we have

$$\sum_{i=0}^{n-1} x^i = \frac{x^n - 1}{x - 1} \equiv 0$$

where arithmetic is done mod $(x^n - 1)$. Because α is primitive, all powers of it up to $n = q - 1$ are not equal to 1. □

Definition 6.4. For any subset of indices $S \subset [n]$, let C_S denote the projection of C onto the indices of S.

Definition 6.5. Codes that meet the Singleton bound are called *Maximum Distance Separable (MDS) codes.* They have the following nice property:

Theorem 6.6. *Suppose C is a MDS code of dimension k and length n over $GF(q)$. Then for all $S \subset [n]$ such that $|S| = k$, we have $|C_S| = q^k$. Moreover, $C_S = GF(q)^k$ for all such S.*

Proof. Because C is maximum distance separable, we can project onto $d - 1 = n - k$ coordinates and still retain the property that distinct codewords have distinct images. Thus, for all such $S \subset [n]$, we have

$$|C_S| = |GF(q)|^k = q^k$$

Since $C_S \subset GF(q)^k$, that implies that in fact $C_S = GF(q)^k$.

□

The above property of MDS codes gives many important properties in terms of finding t-wise independent subsets of random variables over $GF(q)$. Let's look into this:

Definition 6.7. Suppose $X_1, ..., X_n$ are random variables over $GF(q)$. The X_i are said to be t-wise independent if for every $S \subset \{X_1, ..., X_n\}$ with $|S| = t$, are independent random variables over $GF(q)$.

Definition 6.8. A subset $S \subset GF(q)^n$ is a t-wise independent source if, for any uniformly random sample $(X_1, ..., X_n)$, the random variables X_i are t-wise independent.

We observe that any linear code over a finite field is a 1-wise independent source, and moreover any MDS code of dimension k is a $k-$wise independent source. This can be used to show that there exist k-wise independent sources over $GF(2)$ of size $\leq 2^k \cdot n^k$, implying that $k(\log(n)+1)$ random bits chosen uniformly are enough to compute n random bits that are $k-$wise independent. We will not go into the proofs of these claims for the sake of brevity, though it is important to note this powerful combinatorial result that we get from MDS codes.

6.2. **Bose-Chaudhuri-Hocquenghem (BCH) Codes.** As mentioned earlier, the main downside of Reed-Solomon codes is that they require a rather large alphabet size in order for the codes to be nontrivial. This is unavoidable for MDS codes, unfortunately – any code that meets the Singleton bound must have a large alphabet – the size of the alphabet must grow with block length because of the constraint that $n = k - d - 1$.

Now, we ask ourselves, "what have we done in the past that reduces alphabet size?". Suppose $q = p^m$. Then, viewing elements of $GF(q)$ as polynomials, we can naturally convert them to $m-$bit vectors in $GF(p)^m$ and end up with a code of length mn over $GF(p)$.

In general, if we have a $[N, K, D]_q$ code, the above reduction gives a code with parameters $[N\log(N), K\log(N), D']_p$ where $D' \geq D$. These codes, though decent, have rates that can be improved. The BCH codes we will see have dimension $N\log(N) - \lceil \frac{D-1}{q} \rceil \cdot \log(N\log(N)+1)$.

Definition 6.9. For a blocklength $n = q^m - 1$, distance d, and a primitive element $\alpha \in GF(q^m)^*$, we define the BCH code:

$$BCH[q, n, d] = \{(c_0, c_1, ..., c_{n-1}) \in GF(q)^n : c(\alpha) = c(\alpha^2) = ... = c(\alpha^{d-1}) = 0\}$$

This definition should look familiar to the parity check characterization of cyclic Reed-Solomon codes. In fact, they are extremely similar, with the distinction here being that the coefficients c_i must take values in the base field $GF(q)$ instead of the extension field $GF(q^m)$. As such, it follows that BCH codes are linear and cyclic, and the above definition gives a parity-check view of them.

Now, each constraint $c(\alpha) = 0$ is a single constraint over $GF(q^m)$. However, viewing $GF(q^m)$ as a vector space over $GF(q)$ means that such a constraint can also be viewed as m linear constraints over $GF(q)$. This follows from the fact that, using the notion of cyclic shifts in $GF(q)^m$ as multiplication in $GF(q^m)$, we have that multiplication by α for fixed α defines M_α, a $GF(2)-$linear map. The coefficients $c_i \in GF(2)$ correspond to the vectors $\mathbf{v}_i := (c_i, 0, 0, ..., 0)$, and the constraint

$$c(\alpha) = c_0 + c_1\alpha + ... + c_{n-1}\alpha^{n-1} = 0$$

is equivalent to the constraint

$$\mathbf{v}_0 + M_\alpha \mathbf{v}_1 + M_\alpha^2 \mathbf{v}_2 + ... + M_\alpha^{n-1}\mathbf{v}_{n-1} = \mathbf{0}$$

Theorem 6.10. *The dimension of $BCH[q,n,d]$ is at least $n - \lceil \frac{d-1}{q} \rceil \cdot \log_q(n+1)$*

Proof. Observe that, since we are working over $GF(q)$, for all $\gamma \in GF(q)$, if $c(\gamma) = 0$, we also have $c(\gamma^q) = 0$. Thus, the constraints $c(\alpha^{qj})$, $j \in [\lfloor \frac{d-1}{q} \rfloor]$ are redundant. Thus we can remove this set of constraints and be left with $\lceil \frac{d-1}{q} \rceil \cdot \log_q(n+1)$ constraints. □

6.3. **Reed-Muller Codes.** It is worth mentioning, albeit briefly, a class of codes that strictly generalizes Reed-Solomon by considering multivariable polynomials instead of single variable ones. They allow us to use many nice properties of Reed-Solomon codes without requiring a large alphabet size. Efficient decoding of Reed-Solomon implies efficient decoding of Reed-Muller so we will not devote much attention to these.

Definition 6.11. Fix a number of variables $m \in \mathbb{N}$, a degree bound r, and a Galois field $GF(q)$. The Reed-Muller code $RM[q,m,r]$ is defined by its encoding map

$$f(x_1, ..., x_m) \longmapsto \langle f(\alpha) \rangle |_{\alpha \in GF(q)^m}$$

where f is a data polynomial in $GF(q)[x_1, ..., x_m]$ of total degree $\leq r$.

7. Decoding Reed-Solomon Codes

We will now present some algorithms that efficiently decode up to $e < \frac{d}{2}$ errors in Reed-Solomon Codes in $O(n^3)$ time. The unique decoding algorithm can be extended quite easily to a list decoding algorithm that meets the Johnson bound.

7.1. **Unique Decoding.** Suppose we have a $[n, k, (n-k+1)]$ Reed-Solomon code whose evaluation set is $S = \{\alpha_1, ..., \alpha_n\}$. Let $\mathbf{y} \in GF(q)^n$ be a received string. In order to better visualize our data and algorithmic process geometrically, it is useful to think of $\mathbf{y}$ as a set of ordered pairs in the plane $\{(\alpha_1, y_1), (\alpha_2, y_2), ..., (\alpha_n, y_n)\}$. Assume in addition that we have a data polynomial $P(x)$ of degree $\leq k-1$ such that $\Delta(\mathbf{y}, P(\alpha_i)_{i=1}^n) \leq e$.

The idea behind this unique decoding algorithm is to reverse-engineer $P(x)$ by assuming that we know what it is, prove some identities involving the polynomial, and use the identities to then solve for P. To this end, suppose we also have a polynomial $E(x)$ such that

$$E(\alpha_i) = 0 \Longleftrightarrow y_i \neq P(\alpha)$$

Such a polynomial E is called an *error-locator polynomial.* Note that the degree of E is at most e. A solid choice for such a polynomial would be

$$E(x) = \prod_{i: y_i \neq P(\alpha_i)} (x - \alpha_i)$$

Claim 7.1. For every $i \in [n]$, we have $y_i \cdot E(\alpha_i) = P(\alpha_i) \cdot E(\alpha_i)$

Proof. If $y_i = P(\alpha_i)$ then equality is obvious. If not, then $E(\alpha_i) = 0$ and both sides of the equation become zero. □

It may seem as if we have not made any progress here since we know neither P nor E, but we have established a relationship between them that we will be able to exploit later on. In fact, we have another relationship between P and E that we can exploit:

Claim 7.2. If we know $E(x)$ and $\mathbf{y}$, we can uniquely find $P(x)$

Proof. Supposing we know E, we also have the values of i for which $y_i \neq P(\alpha_i)$. Because $\deg(E) < n - k + 1$, we know at least k evaluation points for P, which, in a space of polynomials of degree at most $k - 1$, uniquely determines P. □

If we think of the coefficients of P (of which there are k) and of E (of which there are $e+1$) as variables, then claim 7.1 yeilds n equations in $e+k+1$ variables. If we obey our bound on e, this gives more equations than variables, and so solving for these unknowns would give us a unique decoding.

Unfortunately, though, the equations that we get from claim 7.1 are quadratic and not linear, and systems of quadratic equations are NP hard to solve. If we're clever, we can find a way to introduce new variables that convert our quadratic equations to linear equations. It turns out that this process of linearization requires $e+k-1 \ll n$. Once this linearization step is complete, we only need to use Gaussian elimination (a process requiring $O(n^3)$ time) to solve for the coefficients of a new polynomial we will call $N(x)$.

To do this linearization, define $N(x) = P(x)E(x)$, a polynomial of degree $\leq e + k - 1$. If we can find N and E, then by claim 7.2 we can solve for P by polynomial long division, which can be done in $O(n^3)$ time. Note that at this point we have reduced our algorithm to Gaussian elimination followed by polynomial long division. We now present the algorithm in all of its glory:

Algorithm 7.3. (Welch, Berlekamp)

```
Input:  $n \geq k \geq 1$,
        $e < \frac{n-k+1}{2}$, and
        $n$ pairs $\{(\alpha_i, y_i)\}_{i=1}^n$

Output: Polynomial $P(x)$ of degree $\leq k-1$, or
        FAIL

    1: Compute a nonzero polynomial $E(x)$, and a polynomial $N(x)$ of degree
       $\leq e+k-1$ such that $y_i \cdot E(\alpha_i) = P(\alpha_i) \cdot E(\alpha_i)$
    2: IF $E(x)$ and $N(x)$ do not exist or do not satisfy degree bounds,
           THEN return FAIL
    3: $P(x) \leftarrow \frac{N(x)}{E(x)}$
    4: IF $\Delta(\mathbf{y}, P(\alpha_i)_{i=1}^n) \leq e$
           THEN return $P(x)$
       ELSE
           return FAIL
```

Observe that, if this algorithm does not output **FAIL**, then it produces a correct output. Thus, all we need to do is show that if $E(x)$ and $N(x)$ satisfy the above conditions, then their ratio must be $P(x)$. This can be seen easily by supposing we have two different solutions $\frac{N_1(x)}{E_1(x)}$ and $\frac{N_2(x)}{E_2(x)}$. Define

$$R(x) = N_1(x)E_2(x) - N_2(x)E_1(x)$$

a polynomial of degree $\leq 2e + k - 1$. By definition, R has n roots (at each of the α_i), but by our bound on e, R has degree lower than n, so $R(x) = 0$. Thus:

Theorem 7.4. *All Reed-Solomon codes can be uniquely decoded in $O(n^3)$ time, so long as $e < \frac{n+k-1}{2}$.*

Acknowledgments. I would like to thank my mentor, John Wilmes, for his outstanding guidance and patience in the readings which helped me to produce this paper. I would also like to thank Venkatesan Guruswami for his clear course notes and textbook draft. Coding theory is a difficult topic to teach, and both my mentor and Prof. Guruswami did an excellent job of it.

References

[1] Venkatesan Guruswami, *Carnegie Mellon University, Course 15-859Y Coding Theory Lecture Notes, Problem Sets, & Textbook Draft*, `http://www.cs.cmu.edu/~venkatg/teaching/codingtheory-au14/`

[2] Sarah Spence Adams, *Introduction to Algebraic Coding Theory*, `http://www.math.niu.edu/~beachy/courses/523/coding_theory.pdf`

[3] J.H. Van Lint *Introduction to Coding Theory* `http://www.math.harvard.edu/~gsmith/vanLint.pdf`