2022

Decoding Cyclic Codes via Gröbner Bases

Eduardo Sosa

Decoding Cyclic Codes via Gröbner Bases

Eduardo Sosa

Thesis Presented for Mathematics with Honors

Mathematics Department
Colby College
May 11th, 2022

Abstract

In this paper, we analyze the decoding of cyclic codes. First, we introduce linear and cyclic codes, standard decoding processes, and some standard theorems in coding theory. Then, we will introduce Gröbner Bases, and describe their connection to the decoding of cyclic codes. Finally, we go in depth into how we decode cyclic codes using the key equation, and how a breakthrough by A. Brinton Cooper on decoding BCH codes using Gröbner Bases gave rise to the search for a polynomial-time algorithm that could some day decode any cyclic code. We discuss the different approaches taken towards developing such an algorithm and their success in creating a practical decoding process.

Acknowledgments

I want to thank Katie Andre for her endless continuous support. I also want to thank Prof. Nora Youngs for her motivational guidance and for advising this thesis. Lastly, I want to thank my reader, Prof. Noha Abdelghany, for taking time out of her schedule to read my thesis.

Contents

Chapter 1

Introduction

Whether it be inside a DVD-player, or on a satellite orbiting Earth, encoded information flows through noisy channels to be decoded by a receiver. The encoding aspect of information is important (we need an efficient way to transmit information), but the decoding aspect can become tricky. Information flows through this noisy channel and is thus subject to corruption (errors). If too many errors have occurred, it is impossible to be sure that we can decode the original encoded piece of information. How many errors is too many? How do we encode information, and most importantly, how do we decode it? How can we be so sure we have decoded accurately? In this paper, we will dive into a specific class of codes, namely cyclic codes. "Cyclic code" is an umbrella term for many famous and widely-used codes such as BCH codes, Golay Codes, RS-Codes, and others. Specifically, BCH codes are known for how efficiently we can decode them, while arbitrary cyclic codes are much harder to decode.

The decoding of BCH codes is considered efficient, with polynomial time decoding algorithms by Berlekamp-Massey [3, 13] that have been in use since the 1960's. However, as the length of a BCH code increases, the efficiency of the Berlekamp-Massey algorithm decreases [11]. Thus, large BCH codes do not make very good codes for decoding. On the other hand, cyclic codes, in general, are not known to suffer from the same flaw and so decoding general cyclic codes efficiently is of great interest.

Unfortunately, at this time there does not exist a polynomial-time algorithm for decoding cyclic codes, but empirical evidence suggests that cyclic codes can be decoded in polynomial time regardless of length or minimum distance [15]. Several algorithms have been proposed on the decoding of cyclic codes, some of which make use of Gröbner Bases to find a general error locator polynomial. Gröbner Bases are a finite representation of a given ring ideal whose structure we can exploit to find the locations of the errors of a given encoded piece of information.

While these algorithms can decode most cyclic codes of length less than 63 and error correction capability less than 3, the question remains unanswered for cyclic codes that do not fit these parameters. In this thesis, we strive to investigate these proposed algorithms and their computational complexity further.

Chapter 2

Coding Theory and Gröbner Bases

Linear codes are more common in our everyday lives than we think, whether it's to save a file on your computer, to send a message over any channel or to play a CD. We use linear codes to take information, encode it, send it over a channel, and then the information can be decoded by the receiver. Often, the channel may be noisy in the sense that the information that we send over gets corrupted. Coding theory allows us to encode our messages and be able to decode them with great accuracy up to a certain number of errors that may have occurred.

The method of decoding, however, depends on the structure of our system and its algebraic foundations. In this section, we will introduce some algebraic terminology, basic coding theory, and Gröbner Bases.

2.1 Algebraic Background

Let $\mathbb{F}_2 = \{0, 1\}$ be the standard binary field. We use $\mathbb{F}_2[x]$ to denote all polynomials in x with coefficients in $\mathbb{F}_2$. Thus, let $x^n - 1 \in \mathbb{F}_2[x]$ for a positive integer n, and let $\mathbb{F} = \mathbb{F}_{2^m}$ represent the splitting field of $x^n - 1$ over $\mathbb{F}_2$ for some m. The following theorem regarding our splitting field is fairly known (and thus, we won't derive) [16]:

Theorem 2.1. *Given the splitting field $\mathbb{F}$, $x^n - 1$ can be written as a product of all $x - \zeta$ where ζ is an element of our splitting field. That is,*

$$x^n - 1 = \prod_{\zeta_i \in \mathbb{F}} (x - \zeta_i)$$

for all nonzero $\zeta_i \in \mathbb{F}$.

With this in mind, we can define a primitive element:

Definition 1. *A primitive element is an nonzero element $\alpha \in \mathbb{F}$ such that every element $\zeta \in \mathbb{F}$ can be rewritten as $\zeta = \alpha^i$ for some integer i.*

It turns out that every field $\mathbb{F} = \mathbb{F}_{2^m}$ has at least one primitive element, α that generates the entire field, and that the set $\{\alpha^i \mid 0 \leq i \leq n-1\}$ where $n = |\mathbb{F}|$, are all distinct elements [10]. Taking this into consideration, we can write the following:

$$x^n - 1 = \prod_{i=0}^{n-1}(x - \alpha^i)$$

for $0 \leq i \leq n-1$

In other words, every nonzero element of $\mathbb{F}$ is a root of the equation $x^n - 1$.

2.2 Linear Codes

Before we define *linear* codes, we should define what a code is to begin with. Here we define a code as simply a collection or a set of words. Therefore, $\{a, b, c\}$ is a code if we want it to be. Although it is a useless code since it only contains three words (a, b and c), it helps us understand what a code is. For the remainder of this thesis, we will focus on binary codes. A *binary code* of length one would be $\{0, 1\}$, and a binary code of length two would be $\{00, 01, 10, 11\}$. A word within a binary code can be thought of as a vector of size n. Essentially, binary codes are a subset of the binary vector space of length n. The binary code of length two therefore has words of length $n = 2$. With this said, we need to find a way to define just how different two words are from one another:

Definition 2. *Given two words of length n that belong to a code, the Hamming Distance between the words is the number of positions by which the two words differ.*

We consider two very simple examples:

Example 2.1. *Consider the binary code of length two above. If we take the words 00 and 01, we see that they are different in the second position and so the Hamming distance is one.*

Example 2.2. *If we have a binary code of length two and two words, 00 and 11, the Hamming distance is now two since we have differences in the first position and the second.*

Theorem 2.2. *The Hamming distance is a metric.*

This theorem can be easily derived by the reader and found in [16], but it's important in the sense that the lowest Hamming distance (other than the trivial distance between a word and itself) is able to tell us a lot about the error-correcting capability of a code. From now on, we will refer to Hamming distance as distance.

We now introduce a few more definitions that relate to distance between two code words:

Definition 3. *The minimum distance of a code, denoted by the symbol d, is the shortest distance between any two distinct words in a code.*

Definition 4. *The Hamming weight, or just weight, of a word v is defined as the distance between v and the all 0 word. We denote the weight of v as $w(v)$.*

For the purpose of this paper, our codes will have words whose positions contain *bits*. *Bits* are elements of a field (in our case, mostly $\mathbb{F}_2$). For example, the binary code of length two is one such example where the bits are elements of $\mathbb{F}_2$. It is easy to see how we can interpret these words as vectors, where each bit in a word is a position in a vector. For example, all words in the binary code of length two can be thought of as vectors $v \in \mathbb{F}_2^2$ so that 01 becomes $(0, 1)$ and 00 becomes $(0, 0)$. We say that the binary code of length two is a code over $\mathbb{F}_2$.

It is also helpful to associate words in a code C to polynomials so that a word $(a_0, a_2, ...a_{n-1})$ becomes the polynomial $a_0 + a_1x + ... + a_{n-1}x^{n-1}$ [2]. By interpreting words as such, we can use the algebraic structure of codes to our advantage so that we understand the structure of these words and how they can be used to transmit information easily from one end to another.

We are now ready to introduce linear codes.

Definition 5. *A code $C \subseteq \mathbb{F}^n$ is linear if the code forms a subspace of $\mathbb{F}^n$ over $\mathbb{F}$. That is, for any two words v_1 and v_2 in C and any two scalars a_1 and a_2 in $\mathbb{F}$, $a_1v_1 + a_2v_2$ is also a word that belongs to the code.*

Linear codes allow for a more succinct definition of minimum distance in a linear code C:

Proposition 2.1. *If C is a linear code then*

$$d = \min_{c \in C \setminus \{0\}} w(c)$$

Proof. The proof of this follows from the fact that if v_1 and v_2 are in C, then $v_1 - v_2$ is also in C (due to C being represented by a linear code) and so if two words v_1 and v_2 produce the minimum distance then $v_1 - v_2 = c \in C$ so that

$$d(v_1 - v_2) = w(v_1 - v_2) = w(c) = d(c)$$

as desired. □

Each linear code has a few defining parameters. The first is the length of the word which we denote as n. The second is the minimum distance of the code which we denote as d. The third is the dimension of the subspace (how many words can span the entire code). This is known as k. We thus denote a linear code with these three parameters as a $[n, k, d]$ code.

The strong relationship between linear codes and vector spaces allows us to use matrices to represent linear codes [2].

Definition 6. *G is called the generator matrix of C if*

$$C = \{vG \mid v \in \mathbb{F}^k\}$$

As we recall, the dimension of a linear code C is k and so we can take a vector in $\mathbb{F}^k$ and map it onto a vector in $\mathbb{F}^n$ such that the vector belongs to C. It follows that $rank(G) = k$ and that the dimensions of G are $k \times n$.

There is also a matrix H that maps words in C to the 0 vector so that every word in C is in the kernel of H.

Definition 7. *The parity-check matrix, denoted by H, is a matrix such that for every $v \in C$ we have*

$$Hv^T = 0$$

The converse is also true: If $Hv^T = 0$ for some vector v, then $v \in C$.

It follows that

$$rank(H) = n - k$$

The parity-check matrix will become very important in the definition of syndromes, which are central to the decoding of transmitted messages. For now, the parity-check matrix is an easy way to check for any errors:

Proposition 2.2. *Let C be a linear code and H its parity matrix. If $Hv^T \neq 0$, then an error has occurred.*

Example 2.3. *A [7,4,3] linear code over $\mathbb{F}_2$ is famously known as the Hamming Code and has generator matrix*

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

Conversely, its parity-check matrix is

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

Proposition 2.2 is very helpful in our ability to detect errors. If an error were to occur, we see that our received word (call it y) would be the sum of the real message v and an error vector e so that

$$y = v + e.$$

If we apply the parity-check matrix to y^T we see that

$$Hy^T = Hv^T + He^T = 0 + He^T = s.$$

This leads us to our next definition:

Definition 8. *The syndrome of a received word y is*

$$s = He^T = Hy^T$$

where e is the error incurred by the channel.

It follows that $s = 0$ if and only if no error has occurred - or so many errors have occurred that we have arrived at another word.

Syndromes are central to the decoding process. Next, we will go over a very naive way of decoding linear codes known as coset decoding. Before going into the decoding process, we need to state and prove two powerful theorems:

Theorem 2.3 (Error-Correction Capability)**.** *Given a code C, the amount of errors we can correct μ is bounded so that*

$$\mu \leq \lfloor \frac{d-1}{2} \rfloor$$

Proof. If we can prove that there does not exist two codewords that are $\mu \leq \lfloor \frac{d-1}{2} \rfloor$ apart from a received word y, then we prove that we can correct $\lfloor \frac{d-1}{2} \rfloor$ errors. Assume that there are two codewords c and c' both $\lfloor \frac{d-1}{2} \rfloor$ distance from y. Then, we have that the distance between c and c' is at most $\frac{d-1}{2} + \frac{d-1}{2} = d - 1$ from each other, by the triangle inequality. However, this implies that two codewords are $d - 1$ distance from each other which contradicts the definition of minimum distance. Thus, if there is a codeword c that is $\lfloor \frac{d-1}{2} \rfloor$ away from y, it is unique and correcting $\lfloor \frac{d-1}{2} \rfloor$ errors will get us at that codeword, only. Now, assume that $\lfloor \frac{d-1}{2} \rfloor + 1$ errors occur. Then, we can have more than two codewords at this distance from y since $\frac{d-1}{2} + 1 + \frac{d-1}{2} + 1 = d - 1 + 2 = d + 1$ which is more than the minimum distance. Thus, we can correct at most $\lfloor \frac{d-1}{2} \rfloor$ errors. □

This is a well-known and important proof since it allows us to determine how many errors we can correct.

Theorem 2.4 (Syndrome Uniqueness)**.** *If the number of errors that have occurred given a word y is less than or equal to μ then the syndrome is unique.*

One consequence of the Syndrome Uniqueness Theorem is that it tells us we can correct the word. However, if more than μ errors have occurred, it is impossible to tell what the error was since there is more than one word in C that has the same syndrome[2].

Example 2.4. *Given the linear binary code of length 2, where $C = \{00, 01, 10, 11\}$, the minimum distance is $d = 1$ so that*

$$\mu \leq \lfloor \frac{d-1}{2} \rfloor = \lfloor \frac{1-1}{2} \rfloor = 0$$

and so we can correct at most 0 errors. This is true since given an arbitrary word, say 01, we can never decode such a word since if an error has occurred, the correct word could either be 00 or 11.

Decoding a word y follows the following steps:

1. Generate the standard array for your code.
2. Receive a word.

3. Compute the syndrome of the word using the equation $s = Hy^T$.
4. Find the coset leader e' of the syndrome using the standard array and s.
5. The decoded word is $v = y - e'$.

Note that the above works if and only if $w(e') \leq \mu$, otherwise more than μ errors have occurred and we no longer know what the original word is. By the Syndrome Uniqueness theorem, however, we know that if $w(e') \leq \mu$ the syndrome is unique and thus $e' = e$, the actual error.

To build the standard array we first start with the first row which consists of **all words in the code**, starting with the 0 word (note that the 0 word is part of all codes). Every row thereafter will begin with $e' \in \mathbb{F}^n$ such that e' is of the smallest possible weight and such that e' has not appeared in a previous row. The rest of the row will consist of $e' + c$ where c is the column leader in the first row (this means that $c \in C$).

The first column of the standard array will be the coset leaders. Some of the coset leaders may not be unique in the sense that other choices could have been made in replacement of the current coset leader, however, we do not need to worry since this only happens in the case where the weight of the coset leader is greater than μ.

Example 2.5. *Consider the [5,2,3] linear code over $\mathbb{F}_2$ which has generator matrix*

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

The standard array here would be

00000	10110	01011	11101
00001	10111	01010	11100
00010	10100	01001	11111
00100	10010	01111	11001
01000	11110	00011	10101
10000	00110	11011	01101
00101	10011	01110	11000
10001	00111	11010	01100

Thus, if we received the word 01111, we know that the coset leader is 00100 (in the first column) and thus our original word is 01011 (in the first row).

The dimensions of the array turn out to be $q^{n-k} \times q^k$ where q is the size of our field. The process of building the standard-array and then looking up your word on the array could be tedious, especially if n and k are large. For example, if we let $n = 63$ and $k = 21$, we get a $2^{42} \times 2^{21}$ array. Building such an array is just not feasible. Another approach already proposed is to find the syndrome and then the coset leader for the syndrome. In other words, find e' such that $s = He'$ where $w(e')$ is minimal. This problem is known to be computationally difficult [2], and so taking this approach

would result in a very slow decoding algorithm. For this reason, we turn to other approaches.

Before moving on, we introduce some more terminology on the bounds of linear codes in order to classify certain types of codes and their error-correcting capabilities.

Definition 9. *Given a linear [n, k, d] code, we have that the singleton bound of a code is*

$$d \leq n - k + 1$$

We can prove that the inequality is valid:

Proof. Suppose that we are working over $\mathbb{F}_q$ and that C is a $[n, k, d]$ linear code. Then, since the generator matrix has rank k, we know that the basis for generating the code C consists of k codewords. The linear combinations of such codewords would then result in q^k codewords so that $|C| = q^k$. Rearranging the inequality we have that $k \leq n - d + 1$. Suppose this is false.

Then, we have that $|C| > q^{n-d+1}$. This implies, by the pigeonhole principle, that there exist two codewords c_1 and c_2 in C such that they agree on the first $n - d + 1$ positions. However, note that this means that the distance between c_1 and c_2 must be at most $d-1$ since there is a possibility that the other $d-1$ positions are different (remember that the length of c_1 and c_2 is n and $(n - d + 1) + (d - 1) = n$). This implies that $d(c_1, c_2) \leq d - 1$ which is a contradiction to the definition of minimum distance.

Thus, $|C| \leq q^{n-d+1}$ so $k \leq q^{n-d+1}$ which implies that $d \leq n - k + 1$. □

When the singleton bound is attained, meaning that we have

$$d = n - k + 1,$$

the code is called Maximum Distance Separable or MDS. An example of a MDS code is the very famous Reed-Solomon codes, more about these can be read in [16].

2.3 Cyclic Codes

In this section, we introduce an important subset of the set of linear codes:

Definition 10. *A linear $[n, k, d]$ code C is cyclic if*

$$(c_0, c_1, ..., c_{n-1}) \in C \Rightarrow (c_{n-1}, c_0, ..., c_{n-2}) \in C$$

In other words, a cyclic shift of the bits in any word in C results in a word also in C.

We recall once again that we can view a word $c \in C$ as a polynomial via the representation

$$(c_0, c_1, ..., c_{n-1}) \to c_0 + c_1 x + ...c_{n-1} x^{n-1} = c(x)$$

Thus, a cyclic shift is equivalent to multiplying by x and modding by $x^n - 1$ so that we have

$$c(x) \in C \Rightarrow c(x)x \bmod x^n - 1 \in C$$

Thus, it makes sense to think of cyclic codes in terms of polynomials under this framework. The equivalent of a generator matrix under this framework is a generator polynomial, which we define below:

Definition 11. *A generator polynomial $g(x)$ is a polynomial of degree $n - k$ such that for all $c \in C$*

$$c(x) \in C \Rightarrow g(x)|c(x)$$

It follows that if $c(x) \in C$ then that means that $c(x) = g(x)h(x)$ for some $h(x)$ where $h(x)$ has degree k. However, this also implies that $c(x)$ has degree greater than $g(x)$ and that every root of $g(x)$ is a root of $c(x)$.

Proposition 2.3. *The generator polynomial $g(x)$ above exists and is unique and monic.*

The proof for this proposition can be found in [16].

The following remark is a consequence of the uniqueness of such polynomial:

Remark 1. *If $g(x)|c(x)$ then $c \in C$.*

This leads us to the following alternate definition of a cyclic code C.

Definition 12. *Given a $[n, k, d]$ cyclic code over $\mathbb{F}$ and a generator polynomial $g(x)$ of degree $n - k$ so that $g(x)|x^n - 1$, we can define the code as the set $C = \{g(x)u(x) \mid u(x) \in \mathbb{F}_k[x]\}$*

Since the degree of $g(x)$ is $n - k$ then it follows that every word in the code is of length n.

Note that definition asserts that $g(x)|x^n - 1$, we prove that the generator polynomial of a cyclic code must divide $x^n - 1$:

Proposition 2.4. *If the generator polynomial of a $[n, k, d]$ cyclic code C is $g(x)$ then $g(x)|x^n - 1$.*

Proof. First, write

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

for some polynomial $h(x)$ and some remainder polynomial $r(x)$ such that $\deg(r(x)) < \deg(g(x))$. This must mean that

$$r(x) = -g(x)h(x) \bmod x^n - 1$$

This implies that $g(x)|r(x)$ and thus, following Remark 1 and Definition 11, we have that $r(x)$ must be in the code C. Recall, however, that $r(x)$ must be such that

$$\deg(r(x)) > \deg(g(x))$$

except in the trivial case where $r(x) = 0$. Thus, it follows that $r(x) = 0$ and so this must mean that

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

□

2.4 Examples of Cyclic Codes

Here, we introduce two different types of cyclic codes that will be studied in Chapter 3. This section goes into Reed-Solomon (RS) codes and BCH codes, and though we will mainly focus on the latter, it turns out that RS codes and BCH codes are related to one another [16].

Definition 13. *Let $\mathbb{F} = \mathbb{F}_q$ be the field we are working over with primitive element α. An RS code is a code that has parity-check matrix of the form*

$$H = \begin{bmatrix} 1 & \alpha^b & ... & \alpha^{(n-1)b} \\ 1 & \alpha^{b+1} & ... & \alpha^{(n-1)(b+1)} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha^{b+d-2} & ... & \alpha^{(n-1)(b+d-2)} \end{bmatrix}$$

where b is any integer and α is a primitive element of order n.

RS-codes have the following important property: Let $\alpha_1, \alpha_2, ..., \alpha_n$ be distinct elements within $\mathbb{F}$. It follows that in RS-codes, if the α_i can be written as consecutive powers of α:

$$\alpha_i = \alpha^i \quad 0 \leq i \leq n-1$$

We can call these α_i are called the code locators.

Code locators are important, since, as we will see, they will allow us to tell the position of where an error has occurred. If an error has occurred at position i, this will then correspond to α^i.

From this definition, we can derive the following:

Theorem 2.5. *If C is an RS code then*

$$c(x) \in C \iff c(\alpha^i) = 0 \quad i = b, b+1, ..., b+d-2$$

Thus, elements of RS-codes have at least $d-1$ consecutive roots. It follows from the above that since the generator polynomial $g(x)$ divides $c(x)$ for all $c(x) \in C$, then we can define the generator polynomial as

$$g(x) = (x - \alpha^b)(x - \alpha^{b+1})...(x - \alpha^{b+d-2})$$

From RS-codes we can construct BCH-Codes.

Definition 14. *Let $\mathbb{F}$ be a field of size n and C be a $[n, k, d]$ RS code over an extension field $\mathbb{F}_{q^m}$.*

Bose–Chaudhuri–Hocquenghem (BCH) codes are the intersection of the code C and $\mathbb{F}^n$ so that

$$C_{BCH} = C \cap \mathbb{F}^n$$

BCH codes are very similar to their underlying RS code. In fact,the length of a word in a BCH code is the same as the length of a word in its corresponding RS code. However, the minimum distance of the BCH code may be different:

Definition 15. *Let C_{BCH} be a BCH code and let C be the underlying $[n, k, d]$ RS code. Let D be the distance of C_{BCH}. Then, $d \leq D$. d is known as the minimum designed distance of C_{BCH}.*

One of the things we are interested on with BCH codes is, how do we decode them? Before we answer this question, we need to introduce some important tools in decoding cyclic codes.

2.5 Tools to Decode Cyclic Codes

Decoding cyclic codes is complicated. As we saw, while we know a very naive way of decoding any linear code, this method includes finding the coset leaders and building a standard array. The size of this standard array can be extremely large for relatively small codes. Indeed, the problem of decoding a linear code is known to be NP-Complete [2]. This means that no polynomial-time algorithm for it exists.

Is this the case with cyclic codes? While at the moment, there exists no algorithm, certain mathematicians have tried to derive an algorithm that can decode cyclic codes in polynomial time. In this section, we will introduce some methods used in decoding BCH codes and Reed-Solomon codes and how this may expand to decoding cyclic codes.

The first step in decoding is to compute the syndromes of a received word. We know that a word, y could have some error e such that $y = v + e$ where v is the intended word. To compute the syndromes given a word y, simply apply the parity check matrix on y so that

$$S = \begin{bmatrix} S_0 \\ S_1 \\ \vdots \\ S_{d-2} \end{bmatrix} = H\vec{y} \tag{2.1}$$

Applying the parity check matrix on the received word y would result in $d - 1$ syndromes. These form what we know as our system of syndromes.

Through algorithms such as the BM algorithm and the algorithms presented in Chapter 3, this system of syndromes is used to derive the following important equation:

Definition 16. *The error locator polynomial is defined as*

$$\Lambda(x) = \prod_{j \in J} 1 - \alpha_j x$$

for all a_i such that the $i-th$ position in the error word e is nonzero. Furthermore, define

$$J = \{i \mid i^{th} \text{ position is nonzero}\}$$

.

As we can see, the error-locator polynomial tells us the exact location of the errors where they have occurred since the error polynomial is nonzero exactly where an error has occurred. Finding the error-locator polynomial would help since from there, we can simply plug in all elements in the splitting field α_i to find the roots of such a polynomial, and since the roots tell us that an error occurred at a position, then if $\Lambda(\alpha_i) = 0$ we know an error has occurred at position i. This process is known as the Chien Search.

We will go over how we can find the error locator polynomial from the system of syndromes using Gröbner bases in the next chapter. Though there are other ways to find the error locator polynomial of a given word for an RS-code, these are out of the scope of this thesis. Some of these methods can be found in [16].

One of these algorithms for finding the error-locator polynomial is restricted to BCH codes and is known as the Berlekamp-Massey algorithm [3, 13]. The Berlekamp Massey Algorithm is known for being very efficient (quadratic in time) and helps derive the error locator polynomial after the syndromes have been computed. Another method for deriving the error-locator polynomial is the Euclidean Algorithm [16] which is cubic in time. It seems that decoding RS-codes or BCH codes is easily done in polynomial time. In the next chapter, we will explore how there may exist a polynomial time algorithm for *any* cyclic code.

2.6 Motivation for Gröbner Bases

In this section, we will briefly introduce Gröbner Bases as they can be used to decode cyclic codes. More on the topic can be found here [8].

In 1990, A Brinton Cooper believed that we could also decode BCH codes and find the error locator polynomial by exploiting the algebraic structure of the ideal formed by the system of syndromes. His idea was to use Gröbner Bases to represent the ideal and therefore derive the error locator polynomial that way. In order to define Gröbner Bases, we will need to divide multivariate polynomials. It is unclear as to how we can perform long division on multivariate polynomials, since as we will see, ordering variables (like x,y,z) has consequences on the outcome of division. We introduce

monomial orderings to aid us in determining an algorithm for dividing multivariate polynomials.

First, we need some sort of order in our polynomials. For example, given the simple polynomial $xy + z$, we need to figure out how order the elements xy, z.

We define an ordering on a set of elements Γ to be linear if given two elements α and β, either $\alpha > \beta, \alpha = \beta$, or $\alpha < \beta$.

Definition 17. *A monomial ordering, $>$, on $k[x_1, x_2, ..., x_n]$ is a relation on the set of monomials x^α for $\alpha \in \mathbb{Z}^n_{\geq 0}$, which satisfies three properties:*

1. *$>$ must be a linear ordering on $\mathbb{Z}^n_{\geq 0}$.*
2. *If $\alpha, \beta, \gamma \in \mathbb{Z}^n_{\geq 0}$ and $\alpha > \beta$ then $\alpha + \gamma > \beta + \gamma$.*
3. *Given a nonempty subset $\Gamma \subseteq \mathbb{Z}^n_{\geq 0}$, there always is a least element α so that if $\beta \in \Gamma$ and $\beta \neq \alpha$ then $\alpha < \beta$. This is called the well-ordering property.*

Property three of a monomial ordering means that given a sequence of elements in $\mathbb{Z}^n_{\geq 0}$, there is always a least element in the set, and so an infinite sequence of descending elements is impossible. This is an important property in that, as we will soon see, it allows the division algorithm to eventually terminate since some monomial is always decreasing in degree.

Example 2.6. *Consider the set of positive integers $\mathbb{Z}_{>0}$. Then the usual ordering of numbers is a potential monomial ordering. Properties 1 and 2 follow easily, and Property 3 follows from the fact that given any subset of numbers $\{a_1, a_2, a_3, ...\}$, there will always be a least number.*

We now introduce a more complicated example:

Example 2.7 (Lex Ordering)**.** *We can create an ordering on tuples of size n. Say that we have two tuples $\alpha = (\alpha_1, \alpha_2, ..., \alpha_n)$ and $\beta = (\beta_1, \beta_2, ..., \beta_n)$ both in $\mathbb{Z}^n_{\geq 0}$. How would we order them and say that one of them is greater than the other? One way would be to take the difference of both tuples $\alpha - \beta$ and see if the leftmost nonzero value is positive. If so, we can declare that $\alpha > \beta$. If we get the all zeroes vector, we declare that $\alpha = \beta$ and otherwise, we declare $\alpha < \beta$.*

The above example is actually known as the lexicographic monomial ordering. Below we see some more practical examples that use the lexicographic monomial ordering (lex, for short).

Example 2.8. *Assuming the lex ordering on $\mathbb{Z}^3_{\geq 0}$, we have that*

$$(3, 0, 0) > (2, 0, 0)$$

because the difference between $(3, 0, 0)$ and $(2, 0, 0)$ is $(1, 0, 0)$.

Example 2.9. *We have that*

$$(3, 2, 0) > (3, 1, 5)$$

since $(3, 2, 0) - (3, 1, 5) = (0, 1, -5)$.

We can now use this ordering and apply it to monomials. Consider the two monomials x^2y^3z and x^2y^2z. We can map their exponents to $\mathbb{Z}^3_{\geq 0}$ and have $(2,3,1)$ and $(2,2,1)$, respectively. Thus, using lex ordering, we know that

$$x^2y^3z > x^2y^2z$$

How do we know that lex monomial ordering is a monomial ordering? We need to check that lex ordering actually follows the definition of a monomial ordering. A proof that that lex ordering is a monomial ordering can be found in [8].

We will now introduce two more monomial orderings: Graded Lex Ordering and Graded Reverse Lex Ordering.

Definition 18. *Graded Lex Ordering or grlex for short, is a monomial ordering in which given two elements $\alpha, \beta \in \mathbb{Z}^n_{>0}$, we say that $\alpha > \beta$ if:*

$$|\alpha| = \sum_{i=1}^{n} \alpha_i > |\beta| = \sum_{i=1}^{n} \beta_i$$

If, however, $|\alpha| = |\beta|$ then this means that

$$\alpha > \beta$$

by lex ordering.

In other words, given two vectors, we define that one is greater than the other by grlex ordering if the total sum of terms of one is greater than the total sum of terms of the other. If the total sum of terms is the same for both vectors, then break the tie using lex ordering.

Example 2.10. *Given that $\alpha = (0,1,3)$ and $\beta = (1,0,0)$ and that we are working over grlex ordering, then*

$$|\alpha| = 0 + 1 + 3 = 4 > |\beta| = 1 + 0 + 0 = 1$$

Note that if we were working over lex ordering, $\beta > \alpha$. Now let $\alpha = (1,0,1)$ and $\beta = (0,1,1)$. Then

$$|\alpha| = 1 + 0 + 1 = 2 = |\beta| = 0 + 1 + 1 = 2$$

Thus, we must break ties using lex ordering, so $\alpha > \beta$ since $\alpha - \beta = (1,-1,0)$.

Finally, we introduce the Graded Reverse Lex Ordering also known as grevlex.

Definition 19. *Graded Reverse Lex Ordering or grevlex for short, is a monomial ordering in which given two elements $\alpha, \beta \in \mathbb{Z}^n_{>0}$, we say that $\alpha > \beta$ if:*

$$|\alpha| = \sum_{i=1}^{n} \alpha_i > |\beta| = \sum_{i=1}^{n} \beta_i$$

If, however, $|a| = |b|$, then $\alpha > \beta$ if $\alpha - \beta$ results in the rightmost nonzero value being negative.

Grevlex is similar to grlex in that they both favor higher total sum of terms. However, both break ties differently.

Example 2.11. *Let $\alpha = 1, 0, 1$ and $\beta = 0, 1, 1$. Then*

$$|\alpha| = 1 + 0 + 1 = 2 = |\beta| = 0 + 1 + 1 = 2$$

Thus, we must break the tie. Compute $\alpha - \beta = (1, -1, 0)$. Since the rightmost nonzero value is -1, then $\alpha > \beta$. Note that $\alpha > \beta$ in both grevlex and grlex, but for different reasons.

While most computer systems implement lex ordering, it turns out that grevlex and grlex can be really useful and much more computationally efficient under different scenarios. We will use all three in the forthcoming chapters.

Given that a polynomial is the sum of a finite number of monomials, we can use monomial orderings to reorder the monomials of a polynomial from greatest to least.

Example 2.12. *Consider the polynomial*

$$f(x, y, z) = x^2y^5 + x^3y + z^4 + z^3$$

Then, under the lex ordering for monomials where $x > y > z$, we have that

$$f(x, y, z) = x^3y + x^2y^5 + z^4 + z^3$$

Under grlex we have

$$f(x, y, z) = x^2y^5 + x^3y + z^4 + z^3$$

and under grevlex we have

$$f(x, y, z) = x^2y^5 + x^3y + +z^3 + z^4$$

Finally, we will introduce some important terminology regarding polynomials and monomial orderings:

Definition 20. *Given a polynomial $f = \sum a_\alpha x^\alpha \in k[x_1, ..., x_n]$, and a monomial ordering $>$ in $\mathbb{Z}^n_{>0}$, we define*

the <u>multidegree</u> of f to be

$$multideg(f) = max(\alpha \in \mathbb{Z}^n_{>0} | a_\alpha \neq 0)$$

with respect to $>$,

the <u>leading coefficient</u> to be

$$LC(f) = a_{multideg(f)} \in k,$$

the leading monomial to be

$$LM(f) = x^{multideg(f)},$$

and the leading term to be

$$LT(f) = LC(f) * LM(f).$$

Given an ideal $I \subseteq k[x]$, which is spanned by a set of polynomials $f_1, f_2, ..., f_n$ we can determine whether a function f belongs to the ideal if the remainder upon long division by the basis is zero [8]. Namely,

$$f \in I \iff f = f_1q_1 + f_2q_2 + ...f_nq_n$$

where $q_1, q_2, ..., q_n \in k[x]$. However, when it comes to multivariate ideals, this is no longer the case. As we will see, the division algorithm for multivariate polynomials brings about several problems. If a function f is divided by a basis for an ideal and the remainder r is nonzero, that does not necessarily imply that the polynomial f is not in the ideal.

The division algorithm brings similar problems: depending on the type of monomial ordering we use, we may get different remainders. It turns out both of these problems can be solved via the use of a Gröbner Basis. First, we show this problem of different remainders using multivariate polynomial long division:

Example 2.13. *Suppose we want to divide the polynomial* $x^2y+xy^2+y^2$ *by* $f_1 = xy-1$ *and* $f_2 = y^2 - 1$*. On the left hand side, we have our division process where* a_1 *and* a_2 *represent the dividend after division. The general process is we first divide the polynomial* $x^2y + xy^2 + y^2$ *by* $xy - 1$ *until we no longer can and we subtract the product just as in division in one variable. Then, we divide by* f_2 *and repeat the process. We keep track of the remainder on the right-side.*

$$\begin{array}{rll}
a_1: & x + y & \\
a_2: & 1 & \textit{remainder} \\
xy - 1 & & \\
y^2 - 1 & \sqrt{x^2y + xy^2 + y^2} & \\
 & x^2y - x & \\
 & \hline
 & \quad xy^2 + x + y^2 & \\
 & \quad xy^2 - y & \\
 & \quad \hline
 & \qquad x + y^2 + y & \\
 & \qquad y^2 + y & \\
 & \qquad y^2 - 1 & \to x \\
 & \qquad \hline
 & \qquad\quad y + 1 & \\
 & \qquad\quad \hline
 & \qquad\quad 1 & \to x + y \\
 & \qquad\quad \hline
 & \qquad\quad 0 & \to x + y + 1
\end{array}$$

As we saw above, out remainder turned out to be $x + y + 1$ so that we can rewrite $x^2y + xy^2 + y^2$ as:

$$x^2y + xy^2 + y^2 = (x + y) * (xy - 1) + (y^2 - 1) + x + y + 1$$

What happens if we divide by f_2 first instead?

Example 2.14. *Suppose we want to divide the polynomial $x^2y+xy^2+y^2$ by $f_1 = y^2-1$ and $f_2 = xy - 1$ with lex ordering $x > y$. We do as above and compute*

$$\begin{array}{rll}
a_1: & x + y & \\
a_2: & 1 & \textit{remainder} \\
y^2 - 1 & & \\
xy - 1 & \sqrt{x^2y + xy^2 + y^2} & \\
 & x^2y - x & \\
 & \hline
 & \quad xy^2 + x + y^2 & \\
 & \quad xy^2 - x & \\
 & \quad \hline
 & \qquad 2x + y^2 & \\
 & \qquad \hline
 & \qquad\quad y^2 & \to 2x \\
 & \qquad\quad y^2 - 1 & \\
 & \qquad\quad \hline
 & \qquad\qquad\quad 1 & \\
 & \qquad\qquad\quad \hline
 & \qquad\qquad\quad 0 & \to 2x + 1
\end{array}$$

Now, we can rewrite the polynomial as

$$x^2y + xy^2 + y^2 = (xy - 1) + (x + y)(y^2 - 1) + 2x + 1$$

Note that even though we divided using the same ordering (lex ordering), but in different orders, we still got different remainders. In other words, different monomial orderings and different polynomial orderings give us different remainders.

Gröbner Bases fix this by giving us a concrete answer to the ideal membership problem: Given a polynomial f and an ideal I, the polynomial f is in I if and only if division by the Gröbner Basis results in a remainder of 0 under any monomial ordering [8].

2.7 Gröbner Bases

Now that we have introduced long division, we can now answer the question of how we can find out whether a polynomial f is in an ideal I spanned by a set of polynomials. First, we introduce monomial ideals:

Definition 21. *An ideal I is a monomial ideal if I consists of polynomials of the form $\sum_{\alpha \in A} h_\alpha x^\alpha$ where $h_\alpha \in k[x_1, ..., x_n]$ so that $I = \langle x^\alpha | \alpha \in A \rangle$ (not necessarily finite).*

Monomial ideals possess a series of really useful properties regarding whether a given polynomial $f \in k[x_1, x_2, ..., x_n]$ belongs in the monomial ideal:

Theorem 2.6. *Let I be a monomial ideal, and let $f \in k[x_1, x_2, ..., x_n]$. Then the following are equivalent:*

1. *$f \in I$*
2. *Every term of f lies in I*
3. *f is a k-linear combination of monomials that lie in I.*

Thus, if we have a finite basis, we can determine whether a polynomial belongs to an ideal by simply dividing every term of f with every term in the basis and seeing if all terms are divided by one of the terms in the basis.

An example of a monomial ideal would be $I = \langle x^6, x^2y^3, xy^7 \rangle$. As it turns out, if I is a monomial ideal, then a finite basis exists such that $I = \langle x^{\alpha_1}, x^{\alpha_2}, ...x^{\alpha_s} \rangle$. The proof of this lemma (known as Dickson's Lemma) can be found in [8].

We can relate this idea of monomial ideals to any ideal I. If we take the leading terms of every polynomial in the ideal I, these leading terms also form an ideal so that $\langle LT(I) \rangle$ is also an ideal [8]. Then by Dickson's Lemma, we can derive that there exists a finite basis for said ideal so that $\langle LT(I) \rangle = \langle x^{\alpha_1}, x^{\alpha_2}, ...x^{\alpha_s} \rangle$.

We now introduce the Hilbert Basis Theorem:

Theorem 2.7. *(Hilbert Basis Theorem) Every ideal $I \subseteq k[x_1, ..., x_n]$ has a finite generating set. In other words, $I = \langle g_1, ..., g_t \rangle$ for some $g_1, g_2, ..., g_t \in I$.*

We are now assured that given any ideal, we can produce a finite basis. What follows next is the definition of a Gröbner Basis:

Definition 22. *Fix a monomial order on the polynomial ring $k[x_1, ..., x_n]$. A finite subset $G = \{g_1, ..., g_t\}$ of an ideal $I \subseteq k[x_1, ..., x_n]$ different from $\{0\}$ is said to be a Gröbner basis if $\langle LT(g_1), ..., LT(g_t) \rangle = \langle LT(I) \rangle$.*

Given a Gröbner Bases, we can now determine whether a polynomial f belongs in an ideal I.

Theorem 2.8 ([8])**.** *Let $G = \{g_1, ..., g_t\}$ be a Gröbner basis for an ideal $I \subseteq k[x_1, ..., x_n]$ and let $f \in k[x_1, ..., x_n]$. Then $f \in I$ if and only if the remainder on division of f by the basis G is zero.*

If division of f by G is 0, then this means that we can write

$$f = h_1g_1 + h_2h_2 + ... + h_tg_t,$$

implying that $f \in I$ since it can be generated by the basis G (there exists a linear combination of the terms in G that form f). The converse is more complicated, but uses the lemma that if there exists a remainder r such that

1. no term of r is divisible by any $LT(g_1), LT(g_2), ..., LT(g_t)$, and
2. there is $g \in I$ such that $f = g + r$

then that remainder is unique on division of f by G no matter the order of the elements in G when using the division algorithm [8].

So suppose that $f \in I$, then $f = f + 0$ and since $f \in I$ and 0 satisfies the two properties above, then 0 is the remainder upon division by G.

In the next chapter, we will see how Gröbner Basis representations can be used to decode any BCH codes, and the various attempts to use them to decode cyclic codes.

Chapter 3

The Search for an Efficient Cyclic Decoding Algorithm

3.1 The Cooper Philosophy

A. Brinton Cooper studied BCH codes and how they could be decoded efficiently (in a polynomial-bounded time complexity)[6]. More specifically, he theorized that Gröbner Bases could be used as a way of finding the error locations. To see what his idea was, recall that BCH codes have length $n = 2^m - 1$ for some positive integer m and that a generator polynomial for such a code has $2t$ roots whose powers are consecutive powers of a primitive element in $\mathbb{F}_{2^m}$. This allows us to correct at most t errors.

Recall that given a word $y(x)$ that has been received, we know that

$$y(x) = v(x) + e(x)$$

where $y(x)$ is the received word, $v(x)$ is the intended word, and $e(x)$ is the noise inflicted on the word by the channel (or the error polynomial).

Suppose that we know the $2t$ consecutive roots of our generator polynomial so that these roots are as follows:

$$\alpha_j = \alpha^j \quad j \in \{1, 2, ..., 2t\}.$$

Then, because every root of the generator polynomial is also a root of a given word in a code, plugging each of the roots above into $y(x)$ should generate the following system of syndromes [6]:

$$\begin{aligned} y(\alpha_1) &= r(\alpha_1) + e(\alpha_1) = e(\alpha_1) = S_1 \\ y(\alpha_2) &= r(\alpha_2) + e(\alpha_2) = e(\alpha_2) = S_2 \\ &\vdots \\ y(\alpha_j) &= r(\alpha_j) + e(\alpha_j) = e(\alpha_j) = S_j \end{aligned} \tag{3.1}$$

Naturally, these syndromes are really polynomials whose roots are elements of the splitting field of the BCH code. Remember that every polynomial e holds the error locations of where an error has occurred so that we now have:

$$\begin{aligned} e(\alpha_1) = S_1 &= e_{i_1}\alpha^{i_1} + e_{i_2}\alpha^{i_2} + ... + e_{i_t}\alpha^{i_t} \\ e(\alpha_2) = S_2 &= e_{i_1}\alpha^{2i_1} + e_{i_2}\alpha^{2i_2} + ... + e_{i_t}\alpha^{2i_t} \\ &\vdots \\ e(\alpha_j) = S_t &= e_{i_1}\alpha^{2ti_1} + e_{i_2}\alpha^{2ti_2} + ... + e_{i_t}\alpha^{2ti_t} \end{aligned} \tag{3.2}$$

A few notes are made about the above system of syndromes. The first of these is that since e_k is either 0 or 1 and since we are considering only those that are 1, we can ignore every e_k in the system of syndromes above. Second is that since $\mathbb{F} = \mathbb{F}_{2^m}$ and thus has characteristic 2 [10], we can assume that the Freshman's Dream is true so that $(a+b)^2 = a^2 + b^2$. Therefore, every syndrome computed from even powers is already accounted for by some other syndrome [14]. For example, $S_2 = S_1^2$, so solving for these is redundant and adds nothing. Finally, Cooper finds it convenient to substitute a^{i_j} for X_{i_j} Cooper then reduces this down to the following system of equations:

$$\begin{aligned} e(\alpha_1) = S_1 &= X_1 + X_2 + ... + X_t \\ e(\alpha_3) = S_3 &= X_1^3 + X_2^3 + ... + X_t^3 \\ &\vdots \\ e(\alpha_{2t-1}) = S_{2t-1} &= X_1^{2t-1} + X_2^{2t-1} + ... + X_t^{2t-1} \end{aligned} \tag{3.3}$$

We can further simply this to form the following system:

$$\begin{aligned} X_1 + X_2 + ... + X_t - \alpha_{r_1} &= 0 \\ X_1^3 + X_2^3 + ... + X_t^3 - \alpha_{r_3} &= 0 \\ &\vdots \\ X_1^{2t-1} + X_2^{2t-1} + ... + X_t^{2t-1} - \alpha_{r_{2t-1}} &= 0 \end{aligned} \tag{3.4}$$

As we can see above, if we let the set of equations, define them as F, be the subset of $K[X] = K[X_1, X_2, ..., X_j]$ and then generate an ideal from this set of equations so that we have an ideal $I(F)$, finding the roots of F comes down to analyzing the structure of the variety $V(I(F))$ [14, 6]. For this, it would be extremely useful if we could derive the structure of the variety using a finite basis (like a Gröbner Basis) so that finding the roots of such a a variety comes down to finding the roots of a finite amount of polynomials. Indeed, we will make use of an extremely useful theorem:

Theorem 3.1 ([8])**.** *If $I = \langle f_1, f_2, ...f_s \rangle$ then $V(I) = V(f_1, f_2, ..., f_s)$*

Thus, given a Gröbner Basis representation, it suffices to find the zeros of the variety of the finite Gröbner Basis of I. However, multiple Gröbner Bases exist for a given ideal, and we want to focus on one particular type of Gröbner Basis:

Definition 23. *A reduced Gröbner Basis is a Gröbner Basis such that each polynomial has coefficient of 1 and no element can be reduced modulo the rest of the polynomials [8].*

Cooper then claims that every reduced Gröbner Basis is triangularized as follows:

$$\begin{aligned} g_1 &= g_1(X_1) \\ g_2 &= g_2(X_1, X_2) \\ &\vdots \\ g_t &= g_t(X_1, X_2, ..., X_t) \end{aligned} \tag{3.5}$$

This next lemma proven by Cooper [6] follows

Lemma 3.1. *$g_1(X_1)$ is, within a multiplicative constant, the error locator polynomial $\Lambda(x)$ of the BCH code.*

Once we find the error locator polynomial, all that remains is to perform the Chien Search, which will yield the error locations.

3.2 Cooper's philosophy on cyclic codes

Though Cooper's idea was successful in decoding BCH codes, certain algorithms like the Berlekamp-Massey algorithm were a lot faster, with the Berlekamp-Massey algorithm running in polynomial time. In contrast, Cooper's idea was not poly-time, since in order to calculate the Gröbner Basis of a given ideal, one needs to go through Buchberger's algorithm [4], which is unfortunately, *not* polynomial in time.

Cooper theorized that in the future, one could modify Buchberger's algorithm to produce a polynomial in time algorithm with the purpose of specifically decoding BCH codes. Other mathematicians, such as Chen et al., focused on expanding Cooper's philosophy beyond the scope of BCH codes. Could we possibly decode any cyclic code by creating some sort of ideal, finding its Gröbner Basis, and then deducing the error locator polynomial from there?

3.3 Chen's Attempt

Chen et al. [9] wanted to expand this idea of using Gröbner Bases to decode BCH codes to the decoding of any cyclic code. The ideal built in Cooper's paper would not work for any cyclic code since any cyclic code did not necessarily have $2t$ consecutive roots in its generator polynomial. Instead, Chen et al. wanted to build a special ideal, like Cooper's, that exploited any cyclic code's system of syndromes using Gröbner Bases to then find the general error locator polynomial. The following are the results from Chen et al.'s paper, more can be found in [9].

First we define the system of syndromes. In a BCH code, we have $2t$ consecutive roots, but in any cyclic code, we just have a finite set of roots which can be represented as powers of some primitive roots α of $\mathbb{F}$. We can define such roots as

$$Q = \{\alpha^{i_1}, \alpha^{i_2}, ..., \alpha^{i_j}\}.$$

Thus, computing the syndromes would result in something like:

$$\begin{aligned} y(\alpha_1) = r(\alpha_1) + e(\alpha_1) = e(\alpha_1) = S_1 \\ y(\alpha_2) = r(\alpha_2) + e(\alpha_2) = e(\alpha_2) = S_2 \\ \vdots \\ y(\alpha_j) = r(\alpha_j) + e(\alpha_j) = e(\alpha_j) = S_j \end{aligned} \tag{3.6}$$

and further simplification as in Cooper's case:

$$\begin{aligned} f_{i_1} &= X_1^{i_1} + X_2^{i_1} + ... + X_s^{i_1} - Z_1 = 0 \\ f_{i_2} &= X_1^{i_2} + X_2^{i_2} + ... + X_s^{i_2} - Z_2 = 0 \\ &\vdots \\ f_{i_j} &= X_1^{i_j} + X_2^{i_j} + ... + X_s^{i_j} - Z_j = 0 \end{aligned} \tag{3.7}$$

where $\alpha_i \in Q$ where we have $\mathbb{F} = \mathbb{F}_{2^m}$. Also note that the subscript s above denotes the number of errors that have occurred in that given word.

Chen et al. also introduce one other system of equations defined as

$$\begin{aligned} h_1 &= X_1^{n+1} - X_1 = 0 \\ h_2 &= X_2^{n+1} - X_2 = 0 \\ &\vdots \\ h_s &= X_s^{n+1} - X_s = 0 \end{aligned} \tag{3.8}$$

so that X_i is strictly in $\mathbb{F}$.

So, much like Cooper, we can define the next set of polynomials F over $\mathbb{F}_q$:

$$F = \{f_{i_l} \mid 1 \leq l \leq j\} \cup \{h_u \mid 1 \leq u \leq t\}$$

where t is the minimum distance of the cyclic code C (since we do not know the number of errors s that have occurred).

A very important step in Chen et al.'s approach is determining the *number* of errors that have occurred in a given word. Note that Cooper's Philosophy did not make use of this and that Cooper assumes that the number of errors that have ocurred is exactly t. Thus, Chen et al. try to figure out the number of errors before the true decoding process even begins. If we generate an ideal from the set F, we have

$$I(F) \subseteq K[Z_j, Z_{j-1}, ...Z_1, X_t, X_{t-1}, ...X_1] = K[Z, X].$$

Using lex ordering with $Z > X$, from now on denoted $<_{lex}$, we generate a Gröbner Basis G of $I(F) \subseteq K[Z, X]$

An important note is that Chen et al. generalize this algorithm to underlying fields that are not the binary fields, and so the coefficients as seen in (3.2) are not either 0 or 1. To see this generalization, please refer to [9].

As with Cooper, we can also derive the normalized reduced Gröbner Basis G of $I(F)$.

Chen et el. then state the following proposition:

Proposition 3.1 ([5]). *If G is a normalized reduced Gröbner Basis with respect to $<_{leq}$ and $0 < |V(G)| < \infty$ then G has exactly one polynomial which contains only the variables $Z, X_t, X_{t-1}, ..., X_{t-i+1}$ for a fixed i and $i \leq t$. Let us denote these polynomials as follows :*

$$g_i(Z, X_t, X_{t-1}, ..., X_{t-i+1}), ..., g_2(Z, X_t, X_{t-1}), g_1(Z, X_t).$$

In other words, Chen et al. believed that the normalized reduced Gröbner Basis G of $I(F)$ was of a specific structure, with one of the polynomials in the basis G being in exactly the variables $Z, X_t, X_{t-1}, ..., X_{t-i+1}$ for a fixed i. Also note that one of the requirement is that $0 < |V(G)| < \infty$. Since the polynomials in F are finite and since the number of polynomials in F are finite as well, then because $V(F) = V(I(F)) = V(G)$ [8], we can rest assured that for our specific case of cyclic code, the statement that $|V(G)| < \infty$ is always true. Is $0 < |V(G)|$ always true? Unfortunately, this is left unclear in [5].

We will come back to this proposition in Chen's paper. For now, we will suppose it is true. Chen et al. then build off the following polynomials from the given Gröbner Basis polynomials above:

$$\begin{aligned} w_1(Z, X_t) = g(Z, X_t) &= \sum_{v=0}^{n_1} c_{1,j}(Z) X_t^j \\ w_2(Z, X_{t-1}) = g(Z, X_t, X_{t-1})|_{X_t=0} &= \sum_{v=0}^{n_2} c_{2,j}(Z) X_{t-1}^j \\ &\vdots \\ w_t(Z, X_1) = g(Z, X_t, X_{t-1}, ...X_1)|_{X_t=...=X_2=0} &= \sum_{v=0}^{n_t} c_{t,j}(Z) X_1^j \end{aligned} \tag{3.9}$$

Above, $c_{i,j}(Z)$ implies a monomial with variables in Z. $|_{X_i=0}$ implies that the polynomial was reduced by being evaluated with $X_i = 0$. These polynomials are used to prove the next theorem. This theorem would determine the number of errors and from there determine the error-locator polynomial.

Theorem 3.2 ([5]). *Let C be a cyclic code over F_2 of length n and true minimum distance t. If the number of errors v satisfies $0 < v \leq t$ and $Z = \{S_{i_l} \mid 1 \leq l \leq t\}$ per the system of syndromes which is then derived from the received word $v(x)$, then*

1. *There are exactly s errors if and only if $c_{1,0}(Z) = c_{2,0}(Z) = ... = c_{t-s,0}(Z) = 0$ and $c_{t-s+1,0}(Z) \neq 0$*

2. If there are s errors then $\Lambda(z) \mid GCD(z^n - 1, h_{t-s+1}(X, z))$

The proof can be found in [5]. The above implies that we can find the exact number of errors that have occurred in a word $y(x)$ by computed the polynomials described in (3.9) and evaluating the point at which $c_{i,0} \neq 0$ for the first time. The second point then implies that our error locator polynomial is within a multiplicative polynomial of the GCD between $z^n - 1$ and $h_{t-s+1}(X, z)$.

Therefore, a basic outline of the decoding algorithm follows:

A Method for Decoding any Cyclic Code:

1. Preprocessing Step:
   (a) Compute the normalzed reduced Gröbner Basis of $I(F)$ as described above.
   (b) From this Gröbner Basis, obtain the sets of equations described in Proposition 3.1 and in (3.9)
2. Online Decoding Algorithm:
   (a) Compute the syndromes
   (b) Substitute $Z = \{S_{i_l} \mid 1 \leq l \leq j\}$ into $c_{i,0}(Z)$ for $i = 1, 2, ...t$ and find the first integer s such that $c_{1,0}(Z) = c_{2,0}(Z) = ... = c_{t-s,0}(Z) = 0$ and $c_{t-s+1,0}(Z) \neq 0$
   (c) Use Proposition 3.1 and the extended euclidean algorithm [16] to find the univariate polynomial $Q(z) = GCD(z^n - 1, h_{t-s+1}(Z, z))$.
   (d) Find all the roots of $Q(z)$ via Chien Search.
   (e) Finally, find the error locations by computing $g_i(Z, X_t, ..., X_{s+1}, X_s, ..., X_{s-i+1})|_{X_t=...=X_{s+1}=0}$ for $i = 2, 3, ...s$.

We can analyze the time complexity of this method: As we can see, this algorithm is an online algorithm which means that we can decode words as they come, as long as we complete the preprocessing step. The preprocessing step is clearly not polynomial since we have not solved the problem of Buchberger's Algorithm, which again, is *not* polynomial time. Thus, the preprocessing step has not solved our dilemma.

The online decoding algorithm computes the syndromes which is known to be quadratic in time [16], then substituting as indicated by step 2b is a polynomial process since the computation of finite polynomials is polynomial in time. Next, we use the extended euclidean algorithm to find $Q(z)$. The extended euclidian algorithm is cubic in time, which is still polynomial [16]. Finally, Chien search and step 2e are both polynomial for the reasons stated above. Thus, the online decoding algorithm is bounded by the extended euclidean algorithm which is cubic in time and therefore the online decoding algorithm is itself polynomial in time.

Unfortunately, the algorithm proposed by Chen et al. has been proven wrong. As it turns out, Proposition 3.1 is too ambitious, and the claim that there exists only one polynomial in G which spans $I(F)$ such that $g_i(Z, X_t, X_{t-1}, ..., X_{t-i+1})$ is false. This

was proven false by Loustaunau [12], and so the algorithm is invalid. Loustaunau then tries to fix the problem in his paper, but as [14] point out, this algorithm is not as efficient in terms of computational complexity.

However, the idea proposed involves finding a "one size fits all" Gröbner Basis such that the Gröbner Basis only needs to be computed once and then that same Gröbner Basis can be used with the syndromes of a received word to decode the word. Coding theorists have used this idea to not only compute the Gröbner Basis once, but come up with a general error locator polynomial that only relies on computing the syndromes.

3.4 General Error Locator Polynomials

We will not go into depth about general error locator polynomials, but only introduce them. More on these polynomials can be found here [14, 15].

First, we define a defining set:

Definition 24. *The complete defining set of a cyclic code C is the set*

$$S_C = \{i \mid g(\alpha^i) = 0, 0 \leq i \leq n-1\}$$

where α is a primitive element of $\mathbb{F} = \mathbb{F}_{2^m}$ and g is the generator polynomial of g.

A defining set, S is any subset of S_C such that S is a cyclotomic class of S_C.

Sala et al. [15] fix the parameters of a cyclic code so that the length n of a code is less than 63 and the error correcting capability t is less than or equal to two. For cyclic codes with $n < 63,\ t \leq 2$, the authors were able to break up all codes into a few categories (listed below) and derive general error locator polynomials for these cases:

1. When $S = \{1, 2i+1\}$ for some i and $t = 2$.
2. When $S = \{1, n-1, l\}$ for some l and $t = 2$.
3. When $n < 63$ and $t \leq 1$

The idea is to introduce a general error locator polynomial that given a syndrome vector, will give us the error locator polynomial associated with the syndrome vector. Sala, Orsini, and Mora define a syndrome ideal much like Cooper and Chen et al. whose normalized reduced Gröbner Basis leads to a general error locator polynomial. They then analyze the cyclic codes which follow the above categories and attempt to find a error locator polynomial $\Lambda(x)$ for each of them. After receiving a word $y(x)$, it is only a matter of computing the syndromes, and using the general error locator polynomial to find the actual locator polynomial, whose roots can be found via Chien Search and thus, the error locations follow afterwards.

One can look at [15] for more information.

Chapter 4

Conclusion

Long BCH codes are bad. As has been proven, as the length of a BCH code increases, the efficiency at which we can decode such a code also worsens. In [11], the ratio $\frac{k}{d}$ is proven to go to zero as the length of a code increases. This means that we can correct fewer errors with respect to the length of the code. One workaround is to work with long cyclic codes instead, since these are not known to suffer from the same problem.

Decoding cyclic codes is therefore extremely important, since a general way to decode cyclic codes could allow us to be able to use extensive, complex and superior cyclic codes in technology. At the moment, however, there does not exist a way to decode cyclic codes efficiently (i.e, there does not exist a polynomial-time algorithm to decode cyclic codes). While some have focused on finding an algorithm to improve the decoding of BCH codes [17], others have turned to Gröbner Bases for possible solutions [14, 15, 5, 6].

In this thesis, we explored Gröbner Bases and how we can apply them to coding theory to perhaps find an efficient algorithm that decodes any cyclic codes. Cooper found a way to apply them to BCH codes, and though the algorithm itself relies on an exponential [7] in time algorithm (Buchberger's algorithm), it gave raise to the idea that such a method can work on cyclic codes.

Chen et al. then attempted to generalize the idea of building a syndrome ideal, and then using its Gröbner Basis to find an error locator polynomial through an online decoding algorithm. Unfortunately, a costly mistake invalidated the algorithm, and so other coding theorists tried to fix the mistake.

Orsini, Sala, and Mora, for example, attempted to find a general error locator polynomial, and have succeeded to do so for codes with length less than 63 and error correcting capability less than 2. Unfortunately, neither of these algorithms answers the question of decoding cyclic codes of any length in a polynomial amount of time, but one can only hope that this may some day be solved. It seems that for now, we will stick with long BCH codes and the Massey-Berlekamp algorithm, as bad as long BCH codes may be.

Selected Bibliography Including Cited Works

[1] D. Augot, M. Bardet, and J.-C. Faugere, *Efficient decoding of (binary) cyclic codes above the correction capacity of the code using grobner bases*, in IEEE International Symposium on Information Theory, 2003. Proceedings., 2003, pp. 362–362.

[2] D. Augot, E. Betti, and E. Orsini, *An Introduction to Linear and Cyclic Codes*, 05 2009, pp. 47–68.

[3] E. Berlekamp, *Nonbinary bch decoding*, IEEE Transactions on Information Theory, 14 (1968), pp. 242–242.

[4] B. Buchberger, *A theoretical basis for the reduction of polynomials to canonical forms*, SIGSAM Bull., 10 (1976), p. 19–29.

[5] X. Chen, I. S. Reed, T. Helleseth, and T. A. Truong, *Algebraic decoding of cyclic codes: A polynomial ideal point of view*, 1993.

[6] A. B. Cooper, *Toward a new method of decoding algebraic codes using gröbner bases*, Army Research Laboratory, (1993).

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, The MIT Press, 3rd ed., 2009.

[8] D. Cox, J. Little, and D. O'Shea, *Ideals, varieties, and algorithms. an introduction to computational algebraic geometry and commutative algebra*, (2007).

[9] C. et al., *Algebraic decoding of cyclic codes: A polynomial ideal point of view*, Contemporary Mathematics, 168 (1994).

[10] J. Gallian, *Contemporary Abstract Algebra*, Cengage Learning, 2012.

[11] S. Lin and E. Weldon, *Long bch codes are bad*, Information and Control, 11 (1967), pp. 445–451.

[12] P. Loustaunau and E. V. York, *On the decoding of cyclic codes using gröbner bases*, Appl. Algebra Eng. Commun. Comput., 8 (1997), pp. 469–483.

[13] J. Massey, *Shift-register synthesis and bch decoding*, IEEE Transactions on Information Theory, 15 (1969), pp. 122–127.

[14] T. Mora and E. Orsini, *Decoding cyclic codes: the cooper philosophy*, Gröbner Bases, Coding, and Cryptography, (2008).

[15] E. Orsini and M. Sala, *General error locator polynomials for binary cyclic codes with and*, Information Theory, IEEE Transactions on, 53 (2007), pp. 1095 – 1107.

[16] R. Roth, *Introduction to Coding Theory*, Cambridge University Press, 2006.

[17] D. Schipani, M. Elia, and J. Rosenthal, *On the decoding complexity of cyclic codes up to the bch bound*, in 2011 IEEE International Symposium on Information Theory Proceedings, 2011, pp. 835–839.