Algebraic Coding Theory

Michael Toymil
University of Puget Sound
Math 434, Spring 2010

1 Introduction

Passing Notes
Suppose you are sitting in your British Literature 223 class (suppose *real* hard...) and you are handed a note which reads 'THIS CLPSS IS BORING'. You immediately notice the error in the note, and in addition to wondering how the author of the note ever got into college, you wonder what the actual intended message was. Very quickly you replace 'CLPSS' with 'CLASS' and the meaning becomes clear. You've just applied some coding theory in your English class by *detecting* and *correcting* an error, but how did you do it? In order to detect and correct the error so quickly, you made three assumptions:

1. That a correct word will be an English word.
2. That the correct word contains five letters.
3. That it is more likely that only one letter was incorrect as opposed to two or more.

For now, we will ignore the assumption that the correct word should make sense in the context of entire message. Assumption 2 allowed us to make sense of the errant message, but it does have a weakness. In the above example 'CLASS' was the only English word we could make by changing one letter in 'CLPSS', but what if the received word was 'CLASK'? Now by changing one letter we can make words like 'FLASK' and 'CLASP' as well as 'CLASS'. Using only our assumptions (1 and 2) we cannot justify choosing one word over another, that is, in this case we cannot correct the error. In some cases our two assumptions may not even let us catch errors, for example if the received word was an English word, but the wrong one, say 'GLASS'.

The above situation can be thought of as a simple example of interpreting a message with a decoding algorithm. This is an absolutely crucial process in fields like telecommunication, electrical engineering, and computing where data is sent over a noisy "channel" where it may be altered before it is received. The main aims of Coding Theory are to detect and correct transmission errors as thoroughly and rapidly, and thus as efficiently as possible. Note that Coding Theory is NOT cryptography, that is it doesn't protect data from malicious eyes. The typical model of this system is as follows:

And now with our note-passing example:

We will now begin a mathematical treatment of Coding Theory in order to understand the development of this fascinating (and useful!) field of study.

2 Preliminaries

2.1 Terms

Coding Theory requires some terminology and definitions. Assume the familiar notions of **groups**, **rings**, **fields**, and **Galois Fields** with their standard notation. As in linear algebra, let $V(n, q)$ be the vector space of dimension n over $GF(q)$ containing sequences of length n over $GF(q)$. For simplicity denote $GF(2)$ by $\mathbb{B}$ (for binary).

Let $\mathbb{B}^n$, for $n \in \mathbb{N}$ be the set of ordered n-tuples with entries in $\mathbb{B}$. Define the sum in $\mathbb{B}^n$ component-wise, and note that $\mathbb{B}^n$ is an Abelian group over this operation with the the identity as the zero sequence and each element as its own inverse. Now for some coding theory language.

Definition 1. *Define* $\mathbf{z}$ *as the binary n-tuple of all zeroes, and* $\mathbf{o}$ *as the n-tuple of all ones.*

Definition 2. *A* ***binary block*** (m, n)***-code*** *consists of an* ***encoding function*** $E : \mathbb{B}^m \to \mathbb{B}^n$ *and a* ***decoding function*** $D : \mathbb{B}^n \to \mathbb{B}^m$*. The elements of* $Im(E)$ *are called* ***code words****.*

An example of a code similar to a binary block is the ASCII character set. Each character is mapped to an element of $\mathbb{B}^8$ in a text file before it is decoded and presented to the user.

Definition 3. *If* $a, b \in \mathbb{B}^n$*, we define the* ***distance*** $d(a, b)$ *between* a *and* b *to be:*

$$d(a,b) = \sum_{i=1}^{n} x_i \qquad \begin{cases} x_i = 0 & if \quad a_i = b_i \\ x_i = 1 & if \quad a_i \neq b_i \end{cases}$$

where $a = a_1 a_2 ... a_n$ *and* $b = b_1 b_2 ... b_n$*.*

For example, if $a = 1001$ and $b = 1100$ then $d(a, b) = 2$. Note $d(a, b) = d(b, a)$ for all $a, b \in \mathbb{B}^n$

Definition 4. *If* $a \in \mathbb{B}^n$*, define the* ***weight*** $wt(a)$ *of* a *as the number of non-zero components of* a*.*

For example, if $a = 11001$ then $wt(a) = 5$.

Lemma 1. *If* $a, b \in \mathbb{B}^n$*, then* $d(a, b) = wt(a + b)$*.*

Proof. This is a consequence of addition in $\mathbb{B}$. Let $a = a_1 ... a_n$ and $b = b_1 ... b_n$. Note for $1 \leq i \leq n$, $a_i + b_i = 1$ if and only if $a_i \neq b_i$. Hence, (a_i, b_i) contributes 1 to $wt(a+b)$ if and only if it contributes 1 to $d(a, b)$. □

With the following definition we can understand our note-passing decoding function in mathematical terms.

Definition 5. *The* ***nearest-neighbor decoding principle*** *states that if a word* $r \in \mathbb{B}^n$ *is received and it is a code word, then* $D(r) = r$*. If* r *is not a code word, then we take the distance of* r *from all of the code words and find the least distance, call it* d*. There exists a code word* a*, such that* $d(a, r) = d$*. If* a *is the only code word with* $d(a, r) = d$*, then* $D(r) = a$*. If there is another code word, say* b*, such that* $d(b, r) = d$*, then there is a* ***decoding failure****.*

When a word of length n is transmitted and k components of this word are incorrect, we say that k transmission errors have occured. We build an n-tuple by taking a 1 at each of these k positions and zeros everywhere else, and call an **error vector/word**, e. Note $wt(e) = k$.

Definition 6. *An error word e is **detected** by a code if $a + e$ is not a code word for any code word a. If $a + e$ is a code word, then e is **undetected**.*

Definition 7. *An error word e is **corrected** by a code if the decoding function D gives $D(b + e) = b$ for every code word b.*

2.2 Elementary Codes

Definition 8. *For a given code, let q be the number of individual detectable error words, r be the number of correctable error words, and s be the total number of error words. Define the **performance proportions** to be $\mathcal{P}_d = \frac{q}{s}$ and $\mathcal{P}_c = \frac{r}{q}$.*

The proportions defined above will serve as measurements of a code's error-detecting and error-correcting performance as we examine different types of codes. High performance proportions are desirable.

Example - The Parity Check
One of the first ways devised to detect an error in a received word was the parity check. A $(m, m+1)$ parity check is a binary block$(m, m+1)$-code. The encoding function simply adds up the components of the input word (an m-tuple) (mod 2) and appends the result to create a code word (an $m+1$-tuple). Thus the codewords are always of even weight. The decoding function can *detect* all error words of odd weight, since the addition of such a vector to a code word, would not be a valid code word. An example is shown below

$$1001001 \mapsto a = 10010011, e = 10100001, a + e = 00110010$$
$$\text{but } a + e \text{ is not a codeword since } wt(a + e) \text{ is odd.}$$

Now let us dig a little deeper. Consider the $(2, 3)$ parity check code. E encodes the message words as follows:

$$00 \mapsto 000, 01 \mapsto 011, 10 \mapsto 101, 11 \mapsto 110$$

The set of code words, C is $\{000, 011, 101, 110\}$ (isomorphic to a subgroup of $\mathbb{Z}_2 X \mathbb{Z}_2 X \mathbb{Z}_2$, in case you were dying for some group theory.)

Let us notice some things. Note that the distance between any two code words in C is 2. There are 3 possible error vectors of weight 1: $001, 010, 100$. Any of these error vectors added to a code word will not produce a code word and are thus all detected. However, any error vector of weight 2 is a code word, and will thus go undetected. Our decoding function cannot correct any errors, however, since for $a \in C$ and e an error vector of weight 1, $a + e$ will be equidistant from 3 code words, and so we have a decoding failure.

It is clear that the parity check is not a very good code:

$$\mathcal{P}_d = \frac{1}{2} \text{ and } \mathcal{P}_c = 0$$

However, this example still has some importance. It turns out that it is not a coincidence that the weight of the detectable errors was less than the minimum distance of C. Error-correcting capabilities also depend upon the minimum distance between code words. This is generalized and solidified by the following theorem.

Theorem 1. *A code can detect all errors of weight k or less if and only if the minimum distance between any two code words is $k+1$ or more.*

Proof. Let C be the set of all code words of length n of a given code.
($\Leftarrow$) First suppose that for all $a, b \in C, d(a,b) \geq k+1$. Let $c \in C$ be the transmitted code word and suppose that the channel introduces the error word e with

$$wt(e) \leq k$$

Then the received word is $c+e$ and

$$\begin{aligned} d(c+e,c) &= wt(c+e+c) && \text{(Lemma 1)} \\ &= wt(e+(c+c)) = wt(e) \leq k \end{aligned}$$

Thus $c+e$ is not in C and therefore e is detected.
($\Rightarrow$) Suppose that the code can detect all error words of weight k or less. That is, for all e with $wt(e) \leq k$ and a given $c \in C$, $c+e$ is not a code word. Now suppose $a, b \in C$ and that $d(a,b) \leq k$. Let $e = a+b$. Then $wt(e) = wt(a+b) = d(a,b) \leq k$. Note $a+e = a+a+b = b$ which is a code word. This proves that e goes undetected, a contradiction. Therefore

$$d(a,b) \geq k+1 \text{ for all } a, b \in C$$

□

Theorem 2. *A code can correct all error words of weight k or less, if and only if the minimum distance between code words is at least $2k+1$ (given that the nearest neighbor decoding principle holds).*

We will omit the proof for the sake of space.

Example - The Binary Repetition Code
Naturally, we want to detect and correct more errors. With the previous theorems in mind we know we need to increase the minimum distance between code words. One solution is to simply repeat bits n times. The **binary repetition code** of length n or $BRC(n)$ is defined by the encoding function E

$$0 \mapsto \mathbf{z},\ 1 \mapsto \mathbf{o}$$

Lemma 2. *For a given $BRC(n)$, $\mathcal{P}_d = 1$ and $\mathcal{P}_c = \frac{1}{2}$ (for odd n).*

Proof. The set of code words is $Im(E) = \{\mathbf{z}, \mathbf{o}\}$. The minimum distance is simply the distance between these two elements, given by:

$$d(\mathbf{z}, \mathbf{o}) = wt(\mathbf{z}+\mathbf{o}) = wt(\mathbf{o}) = n$$

Theorem 1 ensures that the code can detect error words of weight $n-1$ or less. That is all of $\mathbb{B}^n$ except z and o. Consequently, the $BRC(n)$ will detect every error word. Thus

$$\mathcal{P}_d = 1$$

Now, Theorem 2 gives that $BRC(n)$ can correct errors of weight $\frac{n-1}{2}$ or less. The total number of correctable error words, r, can be derived combinatorially by the following:

$$\text{for } i \in \mathbb{Z} \text{ such that } 1 \leq i \leq (n-1)/2,\ r = \sum_{i=1}^{(n-1)/2} M(i),$$

where $M(i) = \dfrac{n!}{i!(n-i)!}$ is the number of ways to place i ones in an n-tuple with the rest of the components zero. Some messy work is omitted to show that

$$r = 2^{n-1} - 1$$

Finally, since the total number of detectable errors is $2^n - 2$, we have

$$\mathcal{P}_c = \tfrac{2^{n-1}-1}{2^n-2} = \tfrac{1}{2} \text{ for odd } n. \text{ An interesting result.}$$

□

It appears error detection and correction performance remains constant as we increase n for a $BRC(n)$. This seems counter-intuitive, however, as one would expect some compensation for inflating the size of our code words. Indeed, there is a piece of the puzzle missing. It turns out that as we increase n, the *probability* of encountering error words with high enough weights to go uncorrected is very low. We will not address these probabilities here as we are more concerned by the theory of coding, not the implementation.

Also, there is an optimization problem at hand as we attempt to balance code word length (and thus transmission time) and correction/detection performance. The optimal code for a situation depends on many factors including time constraints, algorithm complexity, and desired "robustness" of the code. As we continue our treatment of codes, it will be helpful collect characterizing information of a code in one place so that we may more easily compare them.

2.3 Measuring and Comparing the Efficiency of Codes

Definition 9. *An (n, M, d)-code is a code of length n, containing M codewords, and having minimum distance d.*

In coding theory, a good (n, M, d)-code has small n for fast transmissions, large M to enable a variety of messages, and a large d to correct many errors. The trick is to optimize these values for each transmission situation. Looking back at our previous examples note that the $BRC(n)$ is an $(n, 2, n)$-code and that the $(m, m+1)$ parity check code is an $(m+1, 2^m, 2)$*code*. Moving forward we will use this characterization as well as the performance proportions to analyze codes.

3 Algebraic Coding Theory

3.1 Linear Codes

Definition 10. *A subspace L of $V(n,q)$ is called a **linear code** of length n over $F = GF(q)$.*

Lemma 3. *The minimum distance d of a linear code L equals the minimum among with weights of non-zero code words.*

This is a result from group codes, which I have chosen to omit the details of.

Let L be a linear code of length n over F. Let $k \leq n$ be the dimension of L over F and choose a basis

$$X^1, X^2, ..., X^k$$

of L over F. Then any element in L is of the form

$$a_1X^1 + a_2X + ... + a_kX^k$$

that is, a linear combination of the basis elements. A message vector $a = (a_1a_2...a_k)$ is thus encoded. A $[n,k,d]$ linear code has length n, dimension k, and minimum distance d.

Example - A linear Code
Examine the linear code L of length over $\mathbb{B}$ with basis

$$B = \{\begin{pmatrix}1\\1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\1\\1\end{pmatrix}, \begin{pmatrix}1\\0\\1\\0\end{pmatrix}\}$$

then the encoding function maps the message words using linear combinations of elements of B as follows:

$$000 \mapsto 0000, 001 \mapsto 1010, 010 \mapsto 0111, 100 \mapsto 1100, 110 \mapsto 1011, 101 \mapsto 0110, 011 \mapsto 1101, 111 \mapsto 0001$$

Notice that the set of code words is thus generated by B. Note the minimum distance is 1 since $wt(0001) = 1$. Thus L is a (4, 8, 1)-code and a $[4,3,1]$ linear code.

$$\mathcal{P}_d = \mathcal{P}_c = 0$$

Lemma 4. *If L is a $[n,k,d]$ linear code over F, then $d \leq n - k + 1$.*

Assuming that linear codes exist such that $d = n - k + 1$, let L be such a $[n,k,d]$ linear code over $\mathbb{B}$. Then L is an $(n, 2^k, n-k+1)$*code* that can thus detect error words of weight $n - k$ or less and correct error words of weight $\frac{n-k-1}{2}$ or less.

3.2 Cyclic Codes

Cyclic codes are special sub-class of Linear Codes. They offer a rich algebraic structure as well as practical advantages in efficiency. Unfortunately their performance will not be analyzed in general here, as finding lower bounds on the minimum distance is a lengthy process. Instead, we will explore a nice connection between cyclic codes and ideals of polynomial rings to illustrate the potential for algebra in coding theory.

Definition 11. *A linear code L of length n over $\mathbb{B}$ is called* ***cyclic*** *if any cyclic shift of a code word is again a code word, i.e, if $(a_0, a_1, ..., a_{n-1}) \in L$ then $(a_{n-1}, a_0, ..., a_{n-2}) \in L$*

An Algebraic Description of Cyclic Codes
Define a map

$$\theta : V(n, 2) \to \mathbb{B}[x]/\langle x^n - 1\rangle$$

where $\langle x^n - 1\rangle$ denotes the ideal of the polynomial ring $\mathbb{B}[x]$ generated by $x^n - 1$,by

$$\theta(a_0, a_1, ..., a_{n-1}) = a_0 + a_1x + ... + a_{n-1}x^{n-1} + \langle x^n - 1\rangle$$

Observe that $\mathbb{B}[x]/\langle x^n - 1\rangle$ is also a vector space over $\mathbb{B}$ (a subspace of vector space of polynomials over $\mathbb{B}$). It is easy to show θ is a vector space isomorphism. Let L be a linear code of length n over $\mathbb{B}$, i.e. L is a subspace of $V(n, q)$. Then, because θ is an isomorphism $Im(L)$ is a subspace of $\mathbb{B}[x]/\langle x^n - 1\rangle$. Let $(a_0, a_1, ..., a_{n-1}) \in L$. Then $(a_{n-1}, a_0, ..., a_{n-2}) \in L$ if and only if

$$a_{n-1} + a_0x + ... + a_{n-2}x^{n-1} + \langle x^n - 1\rangle = x(a_0 + a_1x + ... + a_{n-1}x^{n-1}) + \langle x^n - 1\rangle$$

is in $Im(L)$. Denote $a_0 + a_1x + ... + a_{n-1}x^{n-1} = f(x)$. Then if both $f(x)$ and $xf(x)$ are in $Im(L)$, $x^2f(x)$ is in $Im(L)$ and for $0 \leq i \leq n - 1$, $x^if(x)$ is in $Im(L)$. Since $Im(L)$ is a vector space, any linear combination of the vectors $f(x), xf(x), ..., x^{n-1}f(x)$ is also in $Im(L)$. Therefore, for every polynomial $p(x) = b_0 + b_1x + ... + b_{n-1}x^{n-1}$ in $\mathbb{B}[x]$,

$$p(x)f(x) = (b_0 + b_1x + ... + b_{n-1}x^{n-1})f(x) = b_0f(x) + b_1xf(x) + ... + b_{n-1}x^{n-1}f(x)$$

which is a sum of elements of $Im(L)$ and is thus in $Im(L)$.

$$\text{Hence, } Im(L) \text{ is an ideal in } \mathbb{B}[x]/\langle x^n - 1\rangle$$

And we can thus regard L as an ideal of $\mathbb{B}[x]/\langle x^n - 1\rangle$.

This generalizes easily to the ring of polynomials over any finite field.

3.3 CRC32 - A real-world example

If I may break from strict mathematical rigor for a moment, I'd like to point out the kind of coding performance that is possible using a real-world example, the cyclic redundancy check or CRC, which is used to encode messages over a wireless ethernet.

Cyclic Codes in Terms of Polynomials
We can offer a second definition a cyclic code in terms of a generator polynomial $P(x)$ of degree $n-k$, by stating that a polynomial of degree less than n is a code polynomial if and only if it is divisible by $P(x)$, with x^n-1 evenly divisible by $P(x)$. Note that this is a group code since the sum of two code polynomials will be divisible by $P(x)$.

With this definition we could form code polynomials simply by multiplying our message polynomials by a suitable $P(x)$. However, in a real transmission situation, this would convolute our data thus making decoding computationally expensive. The following clever method of encoding is implemented by the CRC.

A CRC code forms code polynomials in which high-order coefficients are the message symbols and low-order coefficients are "check" symbols, which behave much like the parity check bits do. To encode a message polynomial $M(x)$ of degree k, we divide $x^{n-k}M(x)$ by a generator polynomial $P(x)$ of degree $n-k$, and then add the remainder $R(x)$ to $x^{n-k}M(x)$. The result

$$F(x) = x^{n-k}M(x) + R(x) = Q(x)P(x)$$

is a code polynomial since it is divisible by $P(x)$.

To check for errors, all we need to do is divide the code polynomial by $P(x)$ and check to see that the remainder is zero. If so, we assume the message is error-free and we pull off the k high-order coefficients that represent the original message.

The standard CRC-32 code uses the generator polynomial

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

This code has remarkable error-detection properties. For only adding 32 bits onto the message word, the following are gained:

1. All errors of weight 1 are detected.
2. All burst errors of length 31 or less are detected.
3. All errors of odd weight are detected.
4. Will detect a fraction of $1 - 2^{-32}$ of ALL longer error bursts.

4 Conclusion

In this paper we developed a broad sense of the purpose and practice of Coding Theory. The general feel of things is that you cannot get all of the best performance traits out of a single code. Developing codes is a balancing act of redundancy and efficiency. With the aid of Abstract and Linear Algebra, the search for the optimal code is made much more precise. You'll never think about that phone call or text message the same way again!

References

1. Peterson, W. W. and Brown, D. T. (January 1961). "Cyclic Codes for Error Detection". *Proceedings of the IRE* 49: 228.

2. "CRC-32" on *http://www.wikipedia.com/*

3. Hill, R. A First Course in Coding Theory, Oxford, New York, 1986.

4. Judson, Abstract Algebra: Theory and Applications, 1997.

5. Vermani, L. R. Elements of Algebraic Coding Theory, Chapman and Hall, London, UK, 1996.

6. C.J.Salwach, Codes That Detect and Correct Errors, *The College Mathematics Journal*, Vol. 19, No. 5 (Nov., 1988), pp. 402-416