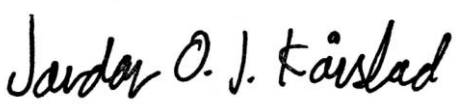
University of Stavanger

Faculty of Science and Technology

MASTER'S THESIS

Study program/Specialization: Lektorutdanning for trinn 8-13 i realfag	Spring semester, 2021 Open / ~~Restricted access~~
Writer: Jardar Olav Jøsendal Kårstad	 (Writer's signature)
Faculty supervisor: David Ploog External supervisor(s):	
Thesis title: Error Correcting Codes and BCH Codes	
Credits (ECTS): 30	
Key words: Information theory, error detection, error correction, codes.	Pages: ……31…… + enclosure: …… Stavanger, 15/06/2021

Frontpage for master thesis
Faculty of Science and Technology

Error Correcting Codes and BCH codes

Jardar Olav Jøsendal Kårstad

15/06/2021

Abstract

One of the central topics in coding theory is the study of error detection and correction, where codes are used to ensure that the correct information can be extracted from a message or storage, even if an error occurs during transmission, or a storage device is damaged. We give an overview of the background of information theory, before learning the necessary theory to understand a communication system and the structure of codes. One central property of codes is the minimum distance, which determines how many errors a code can detect and correct. We learn about some important classifications of codes, like the linear codes, which have a more systematic way of finding the minimum distance. A subclass of the linear codes is the cyclic codes, which have more effective ways of construction by using polynomials. We look further into specific cyclic codes like the BCH codes, which is constructed by choosing a defining minimum distance, the Reed-Solomon codes, which is a special case of BCH code, and the Golay codes, which are examples of perfect codes.

Contents

Chapter 1

Introduction

Whenever we send and receive information between each other, there is always a risk of that information getting distorted in some way. Whether it is a loud noise drowning out our speech, or water smudging ink on a paper, it is hard to guarantee that someone will always receive our intended messages. In digital communication, information gets converted into bits of ones and zeroes and gets transferred either wirelessly or through some physical medium like a cable. It is during these transfers that our messages are exposed to external interference, such as electromagnetic waves, that can change ones into zeroes and zeroes into ones.

The purpose of error detection and correction is to find effective ways of making sure that a potential receiver will be able determine if any errors have occurred, and deduce the original message if any such errors did occur. Certain families of codes have internal mathematical structures that make construction and analysis of the codes more manageable, and these structures will be the main topics of interest in this thesis.

The first chapter will cover the basic theory necessary to understand how one constructs a code, as well as look into the parameter called the minimum distance, and see how this determines the error-correcting capabilities of a code. The second chapter will cover different classes of structured codes, such as the Hamming code, which is the first error-correcting code. It will then introduce cyclic codes, and look at some subclasses within these. The final chapter will briefly discuss the possibilities of introducing some of the mathematical concepts used in this thesis to pupils at different grades in school.

Chapter 2

Basic Theory

2.1 Motivation and focus

For my bachelor thesis, I wrote about the card game SET and finite geometry. The thesis focused mainly on how the cards in the game can be used to represent finite fields, and how the way the game is played is actually based on algebraic principles. For the final part of the thesis, I barely touched upon the concepts of error detection and correction, where I looked at ways the properties of the cards can be used to determine if any mistakes have been made during a game, and then correct them. This turned to be an interesting field with important real life applications, and served as a great motivation to write this thesis, where I will get a chance to look further into the mathematical concepts behind error detection and correction.

The focus of this thesis will mainly be to learn some of the tools and terminology of coding theory, and give a thorough explanation some important mathematical proofs. The theory in this thesis will closely follow the books *CODING THEORY a first course* by Henk. C.A. van Tilborg [3], and *Introduction to Coding Theory Lecture Notes* by Yehuda Lindell [2]. All theorems and proofs can be found in van Tilborg (1993) [3].

2.2 Information Theory

The mathematical theory of communication was first introduced by Claude Shannon in 1948, when he presented his *noisy-channel coding theorem*. Shannon states that good codes can be designed, so that information can safely be exchanged with a low risk of transmission errors [3].

In the context of this paper, we think of communication as the process where information is being exchanged between a *sender* and a *receiver*. The information must be transferred across some medium called the *channel*, and this can be the cables that signals travel through, or a disk that information is written on to be read at a later time. The channel will consist of an input alphabet X and an output alphabet Y.

Definition 1. *An alphabet* $\mathrm{Q} = \{q_1,\ q_2,\ q_3, ...,\ q_n\}$ *is a collection of symbols that can be arranged in different orders to form words.*

If the sets X and Y consist of the same symbols, the channel is said to be *symmetric.* A channel that uses symbols from an alphabet with q elements is said to be q-ary, so for example the channel that uses the alphabet $Q = \{0, 1\}$ is the binary channel. For the majority of this thesis we will be working with the *Binary Symmetric Channel*, or BSC, which is the language used by our computers. Here, the symbols of Q = {0,1} form the basic unit of information, and are called bits.

2.3 Codes

When we send a message, we put together a string of bits and send it across a channel. Noise in the channel refers to all the factors that may alter the symbols in our string, and our goal is to find ways of making our messages more robust before sending them through a noisy channel. The way we do this is to add some extra bits to them. These extra bits carry no information, but will instead serve as a way for the receiver to check whether any errors have occurred, and are therefore referred to as *redundancy.*

Definition 2. *The algorithm used to systematically add redundancy to a message is called the* encoder.

When an encoder has added some redundancy to a message, we have what is referred to as a *codeword.* This codeword is then sent through the channel, and when it arrives at the receivers end, they will have to decode it and check if any errors have occurred.

Definition 3. *The algorithm used to convert a received codeword back into the original message is called the* decoder.

The reason we know that something is wrong when we see the word "universiky" is that this is not a valid word in the English language. We conclude that an error must have occurred, and it is quite easy to decode it into a word

that makes sense. The same idea is used in coding theory. Whenever we receive a non-valid word, we use our decoding algorithm to correct it into a valid one.

Example 1. Suppose we wish to send a simple yes/no message, where 0 symbolizes "no" and 1 symbolizes "yes". These two bits are representing the information we wish to convey, and as such they are called information bits or data bits. All it takes is for one error to occur, before the meaning of our message changes completely.

In order to improve our chances that the correct message is received, we can add some redundancy to our information bits. Instead of sending only a 0 or a 1, we can repeat the the intended bit three times. That way, "no" becomes 000 and "yes" becomes 111. This particular way of making the message more robust is called the *repetition code*. If the receiver then gets a message reading something like 101 or 001, they can determine that an error has occurred, since these combinations of symbols do not represent any valid words. In order to correct such an error, they will use a maximum likelihood algorithm. This simply means to look at what symbol appears the most in a received word, and then change the others accordingly. The word 101 would then be corrected to 111, while 001 would be corrected to 000.

With this, we have seen our first example of a *block code*, sometimes simply referred to as a *code*. The next step will be to define the different parameters that describe the characteristics and capabilities of codes. The number of symbols used to form a codeword is its length. If a code C uses an alphabet Q, and all codewords have a fixed length n, then all valid codewords form a subset of Q^n. The number of codewords in a code is its size or cardinality, and is usually denoted by M. Out of the n symbols used to form a codeword, only k symbols carry information. This is called the dimension of the code, and the remaining symbols $r = n - k$ are the redundancy of the codeword. The relation $R = k/n$ is called the *information rate* of a code, and tells us how effective a code is based on how many of the transferred bits actually carry any information.

2.4 The minimum distance

In this section, we will define a few parameters that describe the error-correcting capabilities of codes. The first one is the Hamming distance, which is the number of coordinates in which two codewords differ.

Definition 4. *The* Hamming distance $d(\boldsymbol{x}, \boldsymbol{y}) = |\{1 \leq i \leq n \mid x_i \neq y_i\}|$.

As an example, the Hamming distance between the codewords

$$c_1 = 110101$$
$$c_2 = 111100$$

is 2 because the third and sixth coordinates are different. The smallest distance between any two codewords in a code is called the minimum distance, and this parameter is crucial for determining how many errors a code is able to detect and correct.

Definition 5. *The* minimum distance d *of a code* C *is given by*

$$d = min\{d(\boldsymbol{x}, \boldsymbol{y}) \mid \boldsymbol{x} \epsilon\ C,\ \boldsymbol{y} \epsilon\ C,\ \boldsymbol{x} \neq \boldsymbol{y}\} \tag{2.1}$$

A code of length n, cardinality M, and minimum distance d is typically labeled as an (n, M, d) code. Another way of thinking about the minimum distance is that it represents the number of coordinates we can change in a codeword before it becomes another valid codeword. Let us consider a binary code C consisting of the codewords

$$\mathbf{x} = 0$$
$$\mathbf{y} = 1$$

The minimum distance d in this case is 1. If a sender transmits the codeword 0, and the receiver receives a 1, then they have no way of knowing that an error occurred since what they received is also a valid codeword. Let us now consider a binary code C with the codewords

$$\mathbf{x} = 00$$
$$\mathbf{y} = 11$$

In this code, the minimum distance d is equal to two, because we must change two coordinates in a codeword in order for it to become another codeword. If a sender now transimts the codeword 00, and the receiver receives

either a 01 or a 10, then they can determine that an error must have occurred since neither of these vectors are valid words in our code. In general, if a code has minimum distance $d = t + 1$, then as long as no more than $d - 1$ errors occur we can detect up to t errors.

If a code has minimum distance

$$d = 2e + 1,$$

then the code can correct up to e errors.

Definition 6. *The* error-correcting capability *e is given by*

$$e = \left\lfloor \frac{d-1}{2} \right\rfloor \tag{2.2}$$

Even though we can determine that an error has occurred when we receive either 01 or 10, we have no way of knowing whether the intended message was 00 or 11. For this, we need a larger minimum distance between the codewords.

Minimum distance d	Detectable errors t	Correctable errors e
1	0	0
2	1	0
3	2	1
4	3	1
5	4	2

When creating a more complex system for communication where large quantities of information will be transferred, it becomes necessary to have a code with a sufficient number of codewords in it. We will now look at a few definitions required to understand how many codewords we can fit in a code.

A *Hamming ball* is a visualization of the error-correcting capability of a code. We can imagine a ball of radius e around a codeword $\mathbf{x}$, where the elements inside this ball are all the non-valid words at distance at most e from $\mathbf{x}$. For the purpose of error correcting, it is important that all such balls are disjoint, meaning that any element in Q^n appears in at most one ball.

Definition 7 (Hamming ball)**.** *A* Hamming ball *of radius r around a codeword $\boldsymbol{x}$ is denoted by $B_r(\boldsymbol{x})$, where the cardinality of such a ball is given by*

$$|B_r(\boldsymbol{x})| = \sum_{i=0}^{r} \binom{n}{i} (q-1)^i. \tag{2.3}$$

To determine the cardinality of a Hamming ball of radius r, is to determine how many words lie at distance at most r from a codeword $\mathbf{c}$. To do this, we must first choose i out of the n coordinates in the codeword to change. This gives us $\binom{n}{i}$ options. We must then replace the symbols in those coordinates with one of the other $(q-1)$ symbols in our alphabet. The number of words at distance i from $\mathbf{x}$ is therefore $\binom{n}{i}(q-1)^i$. By adding these up for all $0 \leq i \leq r$, we find the total number of elements in the Hamming ball of radius r.

The set of all possible words Q^n can be partitioned into $|C|$ disjoint Hamming balls, one for each valid codeword. As all the balls are disjoint, we can add up all the elements in every Hamming ball and get a maximum of q^n elements.

Theorem 1 (Hamming bound)**.** *Let C be a code with error-correcting capability e. Then*

$$|C| \sum_{i=0}^{e} \binom{n}{i} (q-1)^i \leq q^n. \tag{2.4}$$

We may get values less than q^n, because certain words may not lie in any Hamming ball. This is the case whenever the minimum distance of a code is an even number, since each Hamming ball must contain the same number of elements and not overlap.

Another essential parameter is the *covering radius* of a code. This refers to the smallest distance any word $\mathbf{x}$ in Q^n can lie from a codeword $\mathbf{c}$. In other words, if $d(\mathbf{x}, C) = \min\{d(\mathbf{x},\ \mathbf{c})|\ \mathbf{c}\ \epsilon C\}$ is the distance between any word $\mathbf{x}$ and the code C, then the covering radius is given by the following definition:

Definition 8. *The* covering radius ρ *of a code C is given by*

$$\rho = max\{d(\boldsymbol{x}, C) \mid \boldsymbol{x}\ \epsilon\ Q^n\}. \tag{2.5}$$

Knowing that every word in Q^n lies at distance at most ρ from a codeword, we can draw a Hamming ball of radius ρ around every codeword and be sure that every element in Q^n is contained in at least one of these balls.

Theorem 2.

$$|C| \sum_{i=0}^{\rho} \binom{n}{i} (q-1)^i \geq q^n. \tag{2.6}$$

In the special case when $\rho = e$, we have what is called a *perfect code*. These are codes where the Hamming balls of radius e cover all of Q^n, meaning every element in Q^n is contained in exactly one ball.

Theorem 3 (Sphere packing bound)**.**

$$|C| \sum_{i=0}^{e} \binom{n}{i} (q-1)^i = q^n. \tag{2.7}$$

The repetition code of length n is an example of a perfect code. This will have a minimum distance equal to n, but the rate will decrease as we increase n. As it turns out, there are only two other families of codes that are perfect, which we will cover later in the thesis.

The final theorem in this section gives us an upper bound on how many codewords we can have in a code. A proof of the existence of a lower bound can be found in van Tilborg [2] on page 12.

Theorem 4 (Singleton bound)**.** *Let C be a (n, M, d) code, then* $M \leq q^{n-d+1}$.

Proof. If we delete the last $d-1$ coordinates in every valid codeword, their new length becomes $n - (d-1) = n - d + 1$. Because all codewords are at distance at least d from each other, these new words will still be distinct, but there are only q^{n-d+1} distinct words of this length over the alphabet of size q. The number of valid codewords therefore not exceed this number. □

Codes where the cardinality meets this bound, meaning $M = q^{n-d+1}$, are called *maximum-distance-separable codes*, or *MDS codes*.

2.5 Fields

It will be useful go through some of the fundamentals of field theory before moving on to looking at the construction of specific codes. As we shall see, there are many useful algebraic tools we can use to build effective and reliable codes.

Some well known examples of fields are the rational numbers $\mathbb{Q}$, the real numbers $\mathbb{R}$ and the complex numbers $\mathbb{C}$. These are all sets of elements closed under the operations of addition, subtraction, multiplication and division.

Definition 9. *A field is a set $\mathbb{F}$ that together with the operations of addition and multiplication satisfy a set of axioms.*

- If x, y ϵ $\mathbb{F}$ then x + y ϵ $\mathbb{F}$, and xy ϵ $\mathbb{F}$.
- Commutativity: x + y = y + x, and xy = yx.
- Associativity: (x + y) + z = x + (y + z), and (xy)z = z(yz).
- $\mathbb{F}$ contains the additive and multiplicative identity elements, such that x + 0 = x and 1x = x.
- $\mathbb{F}$ contains the additive and multiplicative inverses, such that x + (-x) = 0 and for each x $\neq$ 0, x$\mathbf{x}^{-1}$ = 1.
- Distributivity: x(y + z) = xy + xz

The sets $\mathbb{R}$, $\mathbb{Q}$ and $\mathbb{C}$ are all infinite fields, meaning they contain an infinite number of elements. A finite field, or Galois field, has only a finite number of elements, but still satisfy all the field axioms. Some well known Galois fields are the integers mod p^m, where p is a prime number. In fact, for each prime power p^m, there exists a unique field with p^m elements.

Every finite field contains at least one element called a *primitive element* α, such that every other element except zero can be expressed as a power of α, i.e. $GF(q^m) = \{0,\ 1,\ \alpha,\ \alpha^2, ...\}$. The field $GF(5) = \{0,\ 1,\ 2,\ 3,\ 4\}$ can for example be generated by both two and three when raised to different powers modulo five.

$$\begin{aligned} 2^0 &= 1 & 3^0 &= 1 \\ 2^1 &= 2 & 3^1 &= 3 \\ 2^2 &= 4 & 3^2 &= 9 \equiv 4 \pmod 5 \\ 2^3 &= 8 \equiv 3 \pmod 5 & 3^3 &= 27 \equiv 2 \pmod 5. \end{aligned}$$

2.6 Linear codes

The properties and capabilities of our codes depend on the structures we choose when we construct them. There are certain structures we can choose that will make finding parameters like the minimum distance, and thereby the error-correcting capability, much easier. One way of achieving this is to make our codes linear. This involves giving our alphabet the structure of a Galois field $GF(q)$ for some $q = p^m$, where p is a prime number. We also consider all words in Q^n as vectors in an n-dimensional vector space, denoted by $V_n(q)$. This means that codewords can also be referred to as code vectors

from now on, and will be denoted in bold text.

Definition 10. *A linear, q-ary code C of length n is any linear subspace of $V_n(q)$. If C has dimension k and minimum distance d, then we can label C as a $[n, k, d]$ code. The cardinality of such a code is q^k.*

As for the minimum distance of linear codes, it is not necessary to compare all $\binom{M}{2}$ pairs of codewords and see which one is smallest.

Theorem 5. *The minimum distance of a linear code C is equal to the lowest non-zero Hamming weight in C.*

Proof. For linear codes, the sum of two valid code vectors will result in another valid code vector. If **x** and **y** are in C, then **x** + **y** and **x** - **y** are also in C. This lets us observe the following:

$$d(\mathbf{x}, \mathbf{y}) = d(\mathbf{x} - \mathbf{y}, 0) = w(\mathbf{x} - \mathbf{y}). \tag{2.8}$$

Considering that $w(x) = d(x,\ 0)$, we see how the distance between two codewords in C is equal to the weight of some other codeword in C.

□

For this reason, it is sufficient to find the lowest Hamming weight in a linear code to know its minimum distance. This is a very useful property, especially for codes of lower cardinality. For codes with a large number of codewords, sometimes in the thousands, it becomes necessary to find other ways of finding the minimum distance, which is something that will not be covered in this thesis.

When it comes to the construction and representation of linear codes, there are some elegant methods here as well. A k-dimensional linear subspace can be represented by a set of k independent vectors, called basis vectors. A selection of codewords can be chosen to serve as the basis vectors for a code, and together they form a *generator matrix* that is used to generate the remaining codewords.

Definition 11. *The generator matrix G of an $[n, k, d]$ code C is a $k \times n$ matrix, where the k rows form the basis of C.*

The generator matrix is used to transform k-dimensional message vectors **a** into n-dimensional code vectors **c**. Once the generator matrix has been formed, it can be multiplied with all q^k message vectors to generate a code.

$$C = \{\mathbf{a}G \mid \mathbf{a} \in V_k(q).\}$$

For this reason, a linear code can be represented by a single matrix, instead of a long list of codewords. When a generator matrix is written on the form $(I_k \ P)$, it is called *standard form*, where I_k is the $k \times k$ identity matrix.

Example 2. The binary [5, 2, 3] code can be generated by the generator matrix

$$G = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}. \tag{2.9}$$

The cardinality of this code will be $q^k = 2^2 = 4$, so we can multiply all four message vectors of length 2 with our matrix to generate our codewords.

Message vector $\mathbf{a}$	Code vector $\mathbf{c} = \mathbf{a}$G
00	00000
01	01110
10	10101
11	11011

With the generator matrix being in standard form $(I_k \ P)$, the first two bits of the codewords are equal to the message bits, while the last three bits are the redundancy. As this is a smaller linear code, it is quite easy to determine that the minimum distance is equal to 3, as this is the lowest non-zero Hamming weight of the code.

Another way of describing a k-dimensional vector space such as this is with $n - k$ linearly independent equations. This allows us to form another type of matrix that describes a linear code, called the *parity check matrix.* Acting as a kind of compliment to the generator matrix, the parity check matrix is essential in the decoding process of received codewords.

Definition 12. *The parity check matrix H of an $[n, k, d]$ code is the $(n-k) \times n$ matrix, that when multiplied with a transposed codeword returns the zero vector.*

$$\boldsymbol{c} \in C \iff H\boldsymbol{c}^T = 0^T. \tag{2.10}$$

If the generator matrix was written on the form $(I_k \ P)$, then the parity check matrix can be written as $(-P^T \ I_{n-k})$. This is so that the product $GH^T = 0_{k,n-k}$ gives the all-zero matrix of size $k \times (n - k)$.

The definition of the parity check matrix states that a received codeword is only valid if it returns the zero vector when multiplied by the parity check matrix. This leads us the concept of decoding and the following definition:

Definition 13. *Let C be a q-ary $[n, k, d]$ code with parity check matrix H, and let $\boldsymbol{x}$ be a vector in $V_n(q)$. The vector $\boldsymbol{s} = Hx^T$ is a vector in $V_{n-k}(q)$ and is called the syndrome of $\boldsymbol{x}$.*

If the vector $\mathbf{r}$ is received, and the syndrome of $\mathbf{r}$ is computed to be $\mathbf{0}$, then one assumes that no error has occured and that $\mathbf{r}$ is a valid codeword. If the code vector $\mathbf{c}$ is sent, and the vector $\mathbf{r}$ is received, then an error has occurred and the syndrome will be non-zero. We let the vector $\mathbf{e}$ denote the error vector, and let $\mathbf{r} = \mathbf{c} + \mathbf{e}$. We can then show the following:

$$H\mathbf{r}^T = H(\mathbf{c} + \mathbf{e})^T = H\mathbf{c}^T + H\mathbf{e}^T = H\mathbf{e}^T. \tag{2.11}$$

We know from (2.10) that $H\mathbf{c}^T = \mathbf{0}$, so the syndrome of $\mathbf{r}$ depends only on the error vector. Once we have determined that an error has occurred, the next step becomes to find the intended message $\mathbf{c}$, by first finding the vector $\mathbf{e}$ of minimal weight such that $\mathbf{r} - \mathbf{e} = \mathbf{c}$.

The way this is done is to consider the solutions of the system of linear equations $\mathbf{s} = Hx^T$. Not only will $\mathbf{r}$ be a solution, but all vectors $\mathbf{r}+\mathbf{c}$, where $\mathbf{c}$ is in C, will form the solution space. This means that the set $\mathbf{r} + \mathbf{c}$ forms a *coset*, which is a set of words in $V_n(q)$ which all have the same syndrome. In order to correct the error, we must find the vector $\mathbf{e}$ in this coset with the lowest Hamming weight, called the *coset leader*.

When $\mathbf{e}$ has been found, we have a good maximum-likelihood estimate for the correct codeword with $\mathbf{c} = \mathbf{r} - \mathbf{e}$. A coset leader does not have to be unique, and in that case we have no sure way of estimating the intended message.

Going back to our example with the [5, 2, 3] code, we can start by constructing the parity check matrix, and then use it to correct an error in a transmitted codeword. Writing the parity check matrix on the form $(-P^T\ I_{n-k})$ we get

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{2.12}$$

Now suppose we receive the vector $\mathbf{r}$ =(0,1,1,1,1). Transposing this and multiplying it with H gives us

$$\mathbf{s} = \mathbf{r}^T H = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}. \tag{2.13}$$

We get a non-zero syndrome, and conclude that an error has occurred. In order to correct this error, it is necessary to look at all possible words in Q^n, and group them together in the right cosets. In this example, there are four words sharing the syndrome $\mathbf{s} = (0, 0, 1)^T$, where $(0, 0, 0, 0, 1)$ is the one with the lowest weight. We therefore determine this to be our error vector $\mathbf{e}$, and we can complete the decoding.

$$\mathbf{c} = \mathbf{r} - \mathbf{e} = (0, 1, 1, 1, 0). \tag{2.14}$$

This method of decoding is clearly not suited for long codes, as this requires us to write unreasonably long lists of words to compare the cosets. Instead, it gives us a nice glimpse into the some of the ideas behind error correction.

Before ending this chapter, we should mention one of the most important families of linear codes, namely the Hamming codes. We will not be able to dedicate a section to this topic, but it is worth mentioning that this is the second family of codes which is perfect. The third and final family of perfect codes will be discussed at end of the following chapter.

Chapter 3

Code classes

3.1 Cyclic Codes

The next step is look at a special subclass of linear codes, namely the cyclic codes. This is the point where we will start thinking a little differently about our codewords, when we start representing them as q-ary polynomials over the field $GF(q)$ instead of vectors. A polynomial defined over a field $GF(q)$ simply refers to a polynomial where all coefficients are elements in $GF(q)$. The set of all such polynomials is denoted by $GF(q)[x]$. The reason for this change is to take advantage of the mathematical utilities that become available when working with polynomials, and we shall see how these are particularly useful for expressing and performing calculations with cyclic codes.

Definition 14 (Cyclic codes)**.** *A code C is* cyclic *if it is linear, and for every codeword* $(c_0,\ c_1,\ c_2, ..., c_{n-1})$ *in C, the cyclic shift* $(c_{n-1},\ c_0,\ c_1, ..., c_{n-2})$ *is also in C.*

What this means is that if we take the last coordinate of any codeword in a cyclic code C and move it to the front, we get another valid codeword in C. If we keep shifting a codeword like this, we will eventually end up with the initial codeword again.

$C = \{000, 101, 110, 011\}$ is an example of such a code. The cyclic shift of one codeword, as well as the sum of any two codewords, results in another valid codeword in C.

The code vector $\mathbf{c} = (c_0,\ c_1,\ c_2, ..., c_{n-1})$ will now be represented as the

polynomial

$$c(x) = c_0 + c_1x + c_2x^2 + ... + c_{n-1}x^{n-1},$$

where c_0 represents the constant coefficient, c_1 represents the linear coefficient and so on.

To show why this notation is well suited for expressing cyclic codes, we shall consider the case where we multiply $c(x)$ by x. As we shall see, this will give us the cyclic shift of $c(x)$ if we reduce modulo $x^n - 1$ after multiplying.

$$\begin{aligned} xc(x) &\equiv x(c_0 + c_1x + c_2x^2 + ... + c_{n-1}x^{n-1}) \equiv \\ &\equiv c_0x + c_1x^2 + c_2x^3 + ... + c_{n-2}x^{n-1} + c_{n-1}x^n \equiv \\ &\equiv c_{n-1} + c_0x + c_1x^2 + ... + c_{n-2}x^{n-1} \pmod{x^n - 1}. \end{aligned}$$

When calculating mod $x^n - 1$, then

$$x^n \equiv 1 \rightarrow c_{n-1}x^n \equiv c_{n-1}. \tag{3.1}$$

This gives us an n-dimensional basis for the vector space denoted by $GF(q)[x]/(x^n - 1)$, that refers to the polynomials of $GF(q)[x]$ when calculated modulo $x^n - 1$.

Multiplying a codeword $c(x)$ by x corresponds to one cyclic shift to the right, and since the cyclic shift of any codeword must result in a new codeword, we see that $x^2c(x)$, $x^3c(x)$, and so on, are all valid codewords. Since C is linear, we also know that any linear combination of these polynomials is in C, and so we conclude that we can multiply a codeword by any polynomial as long as we reduce modulo $x^n - 1$. Specifically, for any polynomial $f(x)$, we have $f(x)c(x) \pmod{x^n - 1} \in C$.

One of the benefits of working with cyclic codes is their effectiveness when it comes to describing different codes. More specifically, instead of having to write out long lists of valid codewords, or constructing large generator matrices, we can describe a specific code with just one single polynomial, called the *generator polynomial*.

Theorem 6. *Let C be a cyclic code in $V_n(q)$, then there exists a unique monic polynomial $g(x)$ over $GF(q)$ dividing $x^n - 1$ with the property*

$$c(x) \text{ is in } C \iff g(x) \text{ divides } c(x). \tag{3.2}$$

Proof. Let $g(x)$ be the non-zero monic polynomial of lowest degree in C. If $g(x)$ divides $c(x)$, then we can write $c(x) = f(x)g(x) + r(x)$, where $r(x)$ is the residue after division. The degree of $r(x)$ must then be lower than that of $g(x)$. When $g(x)$ is in C, we have shown that $f(x)g(x) \pmod{x^n - 1}$ is also in C, and when $c(x)$ is in C, we know through linearity that $r(x)$ must be in C. Because $g(x)$ is of the lowest degree, $r(x)$ must be equal to zero, and we conclude that $c(x) = f(x)g(x)$. □

In practice this means that if we can factorize $x^n - 1$ over some field $GF(q)$ into its irreducible factors, then we can use these factors to generate all cyclic codes of length n.

Example 3. If we wanted to create a binary cyclic code of length 7, then our generator polynomial would divide $x^7 - 1$. This is factorized over $GF(2)$ as:

$$x^7 - 1 = (x + 1)(x^3 + x^2 + 1)(x^3 + x + 1). \tag{3.3}$$

Knowing that $g(x)$ divides $x^7 - 1$, we see that any one of these factors, or any product of them, can be used as our generator polynomial. This gives us a total of eight different generator polynomials, thereby defining eight cyclic codes.

Generator polynomial g(x)	Generated code C
1	$V_7(2)$
$x + 1$	[7,6]
$x^3 + x + 1$	[7,4,3] Hamming code
$x^3 + x^2 + 1$	[7,4,3] Hamming code
$(x + 1)(x^3 + x + 1)$	[7,3]
$(x + 1)(x^3 + x^2 + 1)$	[7,3]
$(x^3 + x + 1)(x^3 + x^2 + 1)$	$[7, 1, 7]$ repetition code
$(x + 1)(x^3 + x^2 + 1)(x^3 + x + 1)$	[7,0]

Some of these codes are equivalent to each other, like the [7,4] Hamming codes. The equivalence of codes is explained in van Tilborg [2] on page 11. The [7,0] zero code is trivial since it only contains the zero vector, and when $g(x) = 1$ we generate the entire vector space.

Once we have our generator polynomial, the next step is to determine the dimension k of our code. Before we can use our generator polynomial to encode a message, we need to how many of the n bits will be message bits, and how many will be redundancy. For this we have the following theorem:

Theorem 7. *Let C be a k-dimensional cyclic code in $V_n(q)$ with generator polynomial $g(x)$. The degree of $g(x)$ is then equal to $n-k$.*

Given the generator polynomial $g(x) = g_0 + g_1x + ... + g_{n-k}x^{n-k}$, the generator matrix is defined as

$$G = \begin{pmatrix} g_0 & g_1 & \cdots & \cdots & g_{n-k} & 0 & \cdots & \cdots & 0 \\ 0 & g_0 & g_1 & \cdots & \cdots & g_{n-k} & 0 & \cdots & 0 \\ 0 & 0 & g_0 & g_1 & \cdots & \cdots & g_{n-k} & \cdots & 0 \\ \vdots & \vdots & & \ddots & & & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & g_0 & g_1 & \cdots & \cdots & g_{n-k} \end{pmatrix}$$

Proof. Let the degree of $g(x)$ be equal to l, and let G^* be the $(n-l) \times n$ matrix, whose rows are the codewords $x^i g(x),\ 0 \leq i < n-l$. The rank of G^* will then be equal to $n-l$, because g(x) is monic.

Every codeword $c(x)$ is a multiple of $g(x)$, so we can write $c(x) = u(x)g(x)$, where the degree of u(x) must be lower than $n-l$ because the degree of c(x) is lower than n. With the code C having dimension k, we conclude that $k = n-l$ and $G^* = G$. □

The next step will be to briefly look at an alternative way of describing a cyclic code. Knowing that $g(x)$ divides x^n-1, we can write $x^n-1 = g(x)h(x)$, where $h(x)$ is called the *parity check polynomial*.

Theorem 8. *Let C be a cyclic code in $V_n(q)$ with generator polynomial $g(x)$. C will then have a* parity check polynomial $h(x) = (x^n-1)/g(x)$ *with the property*

$$c(x) \ \in \ C \iff c(x)h(x) \equiv 0 \pmod{x^n-1}.$$

Proof. First to prove that $c(x) \ \in \ C \implies c(x)h(x) \equiv 0\ (mod\ x^n-1)$. When $c(x)$ is in C, we can write it as $c(x) = f(x)g(x)$. This way, $c(x)h(x)$ becomes $f(x)g(x)h(x) \ = \ f(x)(x^n-1) = 0$ in $GF(q)[x]/(x^n-1)$.

The next step is to prove that $c(x) \ \in \ C \impliedby c(x)h(x) \equiv 0\ (mod\ x^n-1)$. If $c(x)h(x) \equiv 0 \pmod{x^n-1}$, then we have $c(x)h(x) = f(x)(x^n-1)$. We can replace (x^n-1) by $g(x)h(x)$, and we get $c(x)h(x) = f(x)g(x)h(x)$. This shows that $g(x)$ divides $c(x)$, meaning $c(x) \ \in \ C$. □

Finally, we can look at how one determines the minimum distance of a cyclic code. For this, we need an extension field of the base field $GF(q)$

where $x^n - 1$ can be factorised completely into linear factors. If we consider the field $GF(8)$, then another way of describing this is
$GF(2)[x]/(x^3 + x + 1)$. Here, $x^3 + x + 1$ is the irreducible polynomial generating the extension field $GF(8)$, and as such is called a *primitive polynomial*.

If we let α be a zero of $x^3 + x + 1$, then α is a *primitive element* of the multiplicative group of $GF(2)[x]/(x^3 + x + 1)$, meaning every element in the extension field except 0 can be generated by multiplying α by itself. In other words, $GF(2)[x]/(x^3 + x + 1) = \{0,\ 1,\ \alpha,\ \alpha^2, ...,\ \alpha^6\}$. The relation $\alpha^3 = \alpha + 1$ is used to reduce the degree of the polynomials to less than 3 for the multiplication.

From (3.3) we saw that $x^7 - 1$ is factorised in $GF(2)$ as

$$x^7 - 1 = (x + 1)(x^3 + x^2 + 1)(x^3 + x + 1).$$

If we were to generate a code using $x^3 + x + 1$ as the generator polynomial, then g(x) would factor into $(x - \alpha)(x - \alpha^2)(x - \alpha^4)$ over $GF(2^3)$. The parity check polynomial would then factor into $(x - 1)(x - \alpha^3)(x - \alpha^5)(x - \alpha^6)$ over $GF(2^3)$.

Over all, we have that

$$x^{q^m} - 1 = \prod_{\xi \in GF(q^m),\ \xi \neq 0} (x - \xi). \tag{3.4}$$

We can factorise $x^n - 1$ into linear factors over $GF(q^m)$ as long as n divides $(q^m - 1)$. If we let $n|(q^m - 1)$, and let ω be a primitive element in $GF(q^m)$, the the n different powers of $\alpha = \omega^{(q^m-1)/n}$ will all be zeroes of $x^n - 1$. We can write the following:

$$x^n - 1 = \prod_{i=0}^{n-1} (x - \alpha^i). \tag{3.5}$$

Here, α is a zero of $x^n - 1$, but will also generate all other zeroes. It is therefore called a *primitive n-th root of unity.* In order for $n|(q^m - 1)$ to be possible, we will from now on always assume that $gcd(q, n) = 1$.

From here we can write the generator polynomial as

$$g(x) = \prod_{i \in I} (x - \alpha^i), \tag{3.6}$$

where I is a subset of {0, 1,...,n-1} called the *defining set* of C with respect to α. Now let $m_i(x)$ be the *minimal polynomial* of α^i. What this means is that $m_i(x)$ is an irreducible polynomial dividing $x^n - 1$, and that α^i is a zero of $m_i(x)$. This means that $(\alpha^i)^q$ is also a zero, because $(\alpha^i)^q = \alpha^{iq}$. In fact, $\alpha^{iq^2}, ..., \alpha^{iq^{m-1}}$ are all zeroes, and the exponents of α modulo n, written as iq^j, form a set called the *cyclotomic coset* $\mathcal{C}_i$ of i modulo n. We can reduce modulo n because $\alpha^n = 1$.

The minimal polynomial can then be factorised as

$$m_i(x) = \prod_{l \in \mathcal{C}_i} (x - \alpha^l). \tag{3.7}$$

When generating a cyclic code, we choose a minimal polynomial, or a product of minimal polynomials, to serve as our generator polynomial. In order to find these minimal polynomials, we first find the corresponding cyclotomic cosets. This is done by using the following property of the defining set:

$$i \in I \rightarrow qi \in I. \tag{3.8}$$

We multiply i by q and reduce modulo n.

We can from this derive the following theorem.

Theorem 9. *Let $V_n(q)$ be a vector space with $gcd(q, n) = 1$. Let m satisfy $n|(q^m - 1)$, and let ω be a primitive element in $GF(q^m)$. Then $\alpha = \omega^{(q^m-1)}$ is a primitive n-th root of unity.*

Let $I = \{i_1, i_2, ..., i_l\}$ be the defining subset of a q-ary, cyclic code C of length n, and let $m_i(x)$ be the minimal polynomial of α^i. We then have the following relations:

$$C = \{c(x) \mid m_i(x) \text{ divides } c(x) for \text{ all } i \in I\}, \tag{3.9}$$

$$C = \{c(x) \mid c(\alpha^i) = 0 \text{ } for \text{ all } i \in I\}. \tag{3.10}$$

Making a cyclic code from scratch

We now have the tools necessary to make our own cyclic code from scratch! We first need to determine the length of the code, and the base field we will be working with. Let us make a binary code of length 9, and use the base field $GF(2)$. We know from Theorem 1 that a generator polynomial will divide $x^9 - 1$, so any factor of this expression can be used as our generator polynomial. We can also write down the set of exponents of the n-th root

of unity as $\{0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8\}$. From this set we can generate the cyclotomic cosets, by starting with 1 and multiplying by $q = 2$ and reducing modulo 9.

$$\begin{aligned}\mathcal{C}_0 &= \{0\}\\ \mathcal{C}_1 &= \{1,\ 2,\ 4,\ 8,\ 7,\ 5\}\\ \mathcal{C}_3 &= \{3,\ 6\}\end{aligned}$$

$\mathcal{C}_0$ gives rise to the irreducible polynomial $m_0 = (x + 1)$, meaning we can write

$$x^9 - 1 = (x + 1)(x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1). \tag{3.11}$$

The other cosets tell us that $(x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1)$ can be factored into two polynomials of order six and two. Starting with $\mathcal{C}_3$, it contains two elements and should therefore define an irreducible polynomial of degree two. Our code is binary, so this leaves us with the following options:

$$\begin{aligned}&x^2\\ &x^2 + x\\ &x^2 + 1\\ &x^2 + x + 1\end{aligned}$$

Observe that $f(0)$ or $f(1)$ yields zero for all of these polynomials except for $x^2 + x + 1$, meaning this is the only second degree polynomial which is irreducible over $GF(2)$. Our second irreducible polynomial is therefore $m_1(x) = x^2 + x + 1$. From this we can work out that

$$\frac{x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1}{x^2 + x + 1}$$

will give us the last factor of $x^9 - 1$. This polynomial division will give us

$$\frac{x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1}{x^2 + x + 1} = x^6 + x^3 + 1. \tag{3.12}$$

We have now reduced $x^9 - 1$ to its primitive polynomials:

$$x^9 - 1 = (x + 1)(x^2 + x + 1)(x^6 + x^3 + 1). \tag{3.13}$$

Any one of these, or any product of them, can now be used to define a cyclic code. Let us use $x^6 + x^3 + 1$ as our generator polynomial, and create

the appropriate generator matrix. The polynomial $x^6 + x^3 + 1$ in $GF(2)$ corresponds to the code vector 100100100 in $V_9(2)$. This gives us the following generator matrix:

$$\mathrm{G} = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

We know that the degree $deg\ g(x) = n - k$, meaning the dimension of our code is $k = n - deg\ g(x) = 9 - 6 = 3$. This corresponds nicely to our three rows in our generator matrix. We can now take all binary message vectors of length three and encode them using our matrix to construct the following cyclic code:

Message bits	Codeword
000	000000000
001	001001001
010	010010010
011	011011011
100	100100100
101	101101101
110	110110110
111	111111111

Another way of generating this code without using the matrix would be to write all the message vectors as message polynomials and multiply them with our generator polynomial and reduce modulo $x^9 - 1$.

Message bits	Message polynomial m(x)	Codeword m(x)g(x)
000	0	0
001	x^2	$x^2 + x^5 + x^8$
010	x	$x + x^4 + x^7$
011	$x + x^2$	$x + x^2 + x^4 + x^5 + x^7 + x^8$
100	1	$1 + x^3 + x^6$
101	$1 + x^2$	$1 + x^2 + x^3 + x^5 + x^6 + x^8$
110	$1 + x$	$1 + x + x^3 + x^4 + x^6 + x^7$
111	$1 + x + x^2$	$1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^7 + x^8$

3.2 BCH Codes

As we have seen, the minimum distance of a code is one of its most important parameters, as this determines the error-correcting capability of the code. Up until now, the codes we have been looking at have minimum distances that depend on the other parameters we set for the code, meaning the minimum distance is something we have to calculate after the code is made. We will now move on to looking at a certain group of codes with a very nice property. This new group is called the *BCH codes*, and is characterized by the fact that we choose a designed distance first, and develop the rest of the code around it. This will allow us to make sure that a code is able to correct as many errors as we need it to, and this makes the BCH codes a very useful subclass of the cyclic codes.

Definition 15. *Let C be a cyclic code in $V_n(q)$, with defining set I. If I contains $d_{BCH}-1$ consecutive integers, C is said to be a* BCH code *of* designed minimum distance d_{BCH}.

If I contains $\{1, 2, ..., d_{BCH}-1\}$ as a subset, the code is a called a narrow-sense *BCH code.*

As mentioned, the benefit of using BCH codes is that we start by choosing a designed minimum distance d_{BCH} and develop the code around this. The following theorem explains how the actual minimum distance such a code will be greater or equal to the designed minimum distance.

Theorem 10 (BCH bound)**.** *Let C be a BCH code of designed minimum distance d_{BCH}. The minimum distance d will then satisfy*
$d \geq d_{BCH}$.

Proof. Let α be an n-th root of unity. We start by considering any non-zero codeword c(x) and defining the following polynomial:

$$C(X) = \sum_{i=1}^{n} c(\alpha^i) X^{n-i}.$$

Having $c(\alpha^i) = 0$ for all $1 \leq i \leq d_{BCH} - 1$, we see that the first non-zero term in our sum is when $i = d_{BCH}$. Knowing that $c(\alpha^i) \neq 0$ for all $i \geq d_{BCH}$, we see that $C(X)$ has at most $n - d_{BCH}$ zeroes, meaning $C(X)$ has degree at most $n - d_{BCH}$.

Now consider $C(\alpha^l)$ for $0 \leq l \leq n-1$. We write

$$C(\alpha^l) = \sum_{i=1}^{n} c(\alpha^i)(\alpha^l)^{n-i}.$$

Here we can rewrite $(\alpha^l)^{n-i} = \alpha^{ln-il}$, where $\alpha^{ln} = (\alpha^n)^l = 1$. Putting this into our expression we get

$$C(\alpha^l) = \sum_{i=1}^{n} c(\alpha^i)\alpha^{-il}.$$

We can also rewrite

$$c(\alpha^i) = c_0 + c_1\alpha^i + ... + c_{n-1}(\alpha^i)^{n-1} = \sum_{j=0}^{n-1} c_j\alpha^{ij}.$$

This gives us

$$C(\alpha^l) = \sum_{i=1}^{n}\sum_{j=0}^{n-1} c_j\alpha^{ij}\alpha^{-il} = \sum_{j=0}^{n-1}\sum_{i=1}^{n} c_j\alpha^{i(j-l)} = \sum_{j=0}^{n-1} c_j \sum_{i=1}^{n} \alpha^{i(j-l)}.$$

As $c_j = 0$ for $j \neq l$, we have

$$\sum_{j=0}^{n-1} c_j = c_l.$$

Considering the geometric series $1 + z + ... + z^{n-1} = \frac{1-z^n}{1-z}$, we can say the following about this sum:

Let $u = j - l, then$

$$\sum_{i=1}^{n} \alpha^{iu} = \alpha^u + \alpha^{2u} + ... + \alpha^{nu} = \sum_{i=1}^{n} \alpha^{iu} = 1 + \alpha^u + \alpha^{2u} + ... + \alpha^{(n-1)u} = \frac{1-\alpha^{nu}}{1-\alpha} = 0.$$

This is true for all $0 \leq j \leq n-1$, except for when $j = l$. Then we have

$$\sum_{i=1}^{n} \alpha^{iu} = \sum_{i=1}^{n} \alpha^0 = n.$$

Finally, we conclude that $C(\alpha^l) = nc_l$, and since the degree of $C(X)$ is at most $n - d_{BCH}$, it follows that at most $n - d_{BCH}$ coordinates c_l can be zero. This lets us conclude that $c(x)$ has a weight at least equal to d_{BCH}.

□

Example 4. We will now look at how to construct a binary narrow-sense BCH code of length n = 93 and designed minimum distance $d_{BCH} = 13$. From our definition of a narrow-sense code, we know that our defining set must contain the elements {1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ,11, 12}. The smallest extension field of GF(2) that contains 93-th roots of unity is $GF(2^m)$ such that 93 divides $2^m - 1$. We find that m = 10, so our extension field will be $GF(1024)$.

We create the cyclotomic cosets by multiplying our elements iq (mod 93):

$$\mathcal{C}_1 = \{1, 2, 4, 8, 16, 32, 64, 35, 70, 47\}$$
$$\mathcal{C}_3 = \{3, 6, 12, 24, 48\}$$
$$\mathcal{C}_5 = \{5, 10, 20, 40, 80, 67, 41, 82, 71, 49\}$$
$$\mathcal{C}_7 = \{7, 14, 28, 56, 19, 38, 76, 59, 25, 50\}$$
$$\mathcal{C}_9 = \{9, 18, 36, 72, 51\}$$
$$\mathcal{C}_{11} = \{11, 22, 44, 88, 83, 73, 53, 13, 26, 52\}.$$

Our generator polynomial can then be written as $g(x) = m_1(x)m_3(x)m_5(x)m_7(x)m_9(x)m_{11}(x)$. The cyclotomic cosets contain fifty elements in total, meaning fifty zeroes. The degree of $g(x)$ is therefore equal to fifty, and the dimension of our code will then be $k = n - deg(g(x)) = 43$. It follows from the BCH bound that the minimum distance of this code must be equal to or greater than the designed minimum distance. Looking at the cyclotomic cosets, we see that they contain the zeroes $I = \{1, 2, ..., 14\}$, meaning the minimum distance is actually at least 15. The parameters of this code is therefore $[93, 43, \geq 15]$.

3.3 Reed-Solomon Codes

In 1977, NASA launched their Voyager program by sending two interstellar probes into space. Their objective was to collect data and transmit this to Earth as they moved through and eventually out of the solar system. As of 2021, the probes have entered interstellar space and are still collecting data. The pictures sent back to Earth by these probes are encoded by certain error-correcting codes, one of them being a special type of BCH code called a *Reed-Solomon code* [1].

Definition 16. *A* Reed-Solomon code*, or* RS(n,k) code*, is a narrow-sense BCH code with length $n = q - 1$ and dimension k.*

Because of this definition, the Reed-Solomon codes are good examples of non-binary codes, as you need larger alphabets in order to create useful codes. The code used by the Voyager probes is the $RS(255, 223)$ code, with an excellent information rate of 0.8745, or 87%.

When constructing RS codes, we consider extension fields $GF(q^m)$ of some base field $GF(q)$ that contain n-th roots of unity. When $n = q - 1$, the smallest extension field that contains such roots is $GF(q)$ itself. If we let α be a primitive element in our field $GF(q)$, and we choose a designed minimum distance d_{BCH}, then the generator polynomial for the corresponding RS code is given by

$$\prod_{i=1}^{d_{BCH}-1} (x - \alpha^i).$$

Being a BCH code, we know from the BCH bound that the actual minimum distance of the Reed-Solomon codes must be equalt to or greater than the designed minimum distance d_{BCH}. As it turns out, the true minimum distance of a Reed-Solomon code is also upper bounded by the Singleton bound, meaning it is actually equal to the designed minimum distance.

The following table shows a few examples of Reed-Solomon codes and their parameters. The table lists the error-correcting capabilities t, dimensions k, minimum distances d and information rates r.

$q = 2^m$	n = q - 1	t	k	d	r
4	3	1	1	3	0.3333
8	7	1	5	3	0.7143
		2	3	5	0.4286
		3	1	7	0.1429
16	15	1	13	3	0.8667
		2	11	5	0.7333
		5	5	11	0.3333
		7	1	15	0.0667
32	31	1	29	3	0.9355
		5	21	11	0.6774
		8	15	17	0.4839
256	255	5	245	11	0.9608
		50	155	101	0.6078

Looking at the table, we can observe that the $RS(255, 245)$ code would make a particularly effective code, as it can correct up to five errors, and has

an impressive information rate of 0.9608, or 96%. The price we have to pay is the long code length of 255.

3.4 Golay Codes

Finally, there exists a unique set of codes called the *Golay codes*. There are two codes in this family, one binary, and one ternary. We will focus on the binary version, which has the parameters [23, 12, ≥ 5]. Following the usual algorithm for constructing cyclic codes, we know that a generator polynomial $g(x)$ must divide $x^{23}-1$, meaning we can write $x^{23}-1 = (x-1)m_1(x)m_2(x)$. Considering that we are working in $GF(2)$, we can think of $m_2(x)$ as $m_{-1}(x)$, and if we write out the cyclotomic cosets for $m_1(x)$ and $m_{-1}(x)$ (mod 23), something interesting occurs:

$$\mathcal{C}_1 = \{1,\ 2,\ 4,\ 8,\ 16,\ 9,\ 18,\ 13,\ 3,\ 6,\ 12\}$$
$$\mathcal{C}_{-1} = \{-1,-2,-4,-8,-16,-9,-18,-13,-3,-6,-12\}.$$

It turns out that the cosets $\mathcal{C}_1$ and $\mathcal{C}_{-1}$ are symmetric, and that $m_1(x) = x^{11}m_{-1}(1/x)$. This can be shown by considering the following:

$$m_{-1}(x) = \Big(\frac{1}{x}-\frac{1}{\alpha}\Big)\Big(\frac{1}{x}-\frac{1}{\alpha^2}\Big)\Big(\frac{1}{x}-\frac{1}{\alpha^4}\Big)\ \ldots\ \Big(\frac{1}{x}-\frac{1}{\alpha^{12}}\Big).$$

$$x^{11}m_{-1}(x) = x^{11}\Big(\frac{1}{x}-\frac{1}{\alpha}\Big)\Big(\frac{1}{x}-\frac{1}{\alpha^2}\Big)\Big(\frac{1}{x}-\frac{1}{\alpha^4}\Big)\ \ldots\ \Big(\frac{1}{x}-\frac{1}{\alpha^{12}}\Big) =$$

$$= \Big(1-\frac{x}{\alpha}\Big)\Big(1-\frac{x}{\alpha^2}\Big)\Big(1-\frac{x}{\alpha^4}\Big)\ \ldots\ \Big(1-\frac{x}{\alpha^{12}}\Big).$$

This last expression has a total of 92 alphas in it. With α being a 23rd root of unity, we have that $\alpha^{n\cdot 23} = 1$. We can therefore multiply by $\alpha^{4\cdot 23}$ to cover all the fractions and get

$$\alpha^{4\cdot 23}x^{11}m_{-1}(x) = (\alpha-x)(\alpha^2-x)(\alpha^4-x)\ \ldots\ (\alpha^{12}-x) =$$
$$= (-1)(x-\alpha)(x-\alpha^2)(x-\alpha^4)\ \ldots\ (x-\alpha^{12}) = (-1)m_1(x) = m_1(x)$$

in $GF(2)$.

Let $c(x)$ be a codeword in the binary code C, generated by $m_1(x)$, and let $c(x)$ have an even Hamming weight w. For the binary Golay code, we have the following implication:

$$w(c(x)) \equiv 0 \ (mod\ 2) \implies w(c(x)) \equiv 0 \ (mod\ 4).$$

In words, this means that if a codeword in the binary Golay code has even Hamming weight, then not only is it divisible by two, but it must actually be divisible by four.

Proof. A codeword of even weight w can be written as $c(x) = x^{i_1} + x^{i_2} + ... + x^{i_w}$. If $c(x)$ was generated by $m_1(x)$, then $c(x) \equiv 0 \bmod m_1(x)$. With $c(x)$ having an even weight, we can determine that $c(1)$ will give us an even number of ones, which is equal to zero when added up in $GF(2)$. Having 1 as a zero of $c(x)$ means that $(x-1)$ divides $c(x)$, and since $(x-1)$ and $m_1(x)$ are coprime, we conclude that $c(x) \equiv 0 \bmod (x-1)m_1(x)$.

Next, $c\big(\frac{1}{x}\big)$ can be written as $x^{-i_1} + x^{-i_2} + ... + x^{-i_w}$. We can then let $p(x) = x^{23}c\big(\frac{1}{x}\big) = x^{23-i_1} + x^{23-i_2} + ... + x^{23-i_w}$. We are working with the set of polynomials $GF(2)/x^{23} - 1$, so $x^{23} \equiv 1$. If $c(x) = f(x)m_1(x)$ for some polynomial $f(x)$, then $c\big(\frac{1}{x}\big) = f\big(\frac{1}{x}\big)m_1\big(\frac{1}{x}\big)$. We can substitute this into $p(x)$ and get the following expression:

$$\begin{aligned} p(x) &= x^{23}c\Big(\frac{1}{x}\Big) \\ &= x^{23}f\Big(\frac{1}{x}\Big)m_1\Big(\frac{1}{x}\Big) \\ &= x^{12}f\Big(\frac{1}{x}\Big)x^{11}m_1\Big(\frac{1}{x}\Big) \\ &= x^{12}f\Big(\frac{1}{x}\Big)m_{-1}(x). \end{aligned} \tag{3.14}$$

So $p(x)$ is a multiple of $m_{-1}(x)$, meaning $p(x) \equiv 0 \ (\text{mod}\ m_{-1}(x))$. Having $p(1) = 0$, we also have that $p(x) \equiv 0 \ (\text{mod}\ (x-1))$. With $m_{-1}(x)$ and $(x-1)$ being coprime, we can say that $p(x) \equiv 0 \bmod m_{-1}(x)(x-1)$. If we multiply $p(x)$ by $c(x)$ we get

$$c(x)p(x) = c(x)x^{23}c(\frac{1}{x}) \equiv 0 \ (\text{mod}\ (x-1)m_1(x)m_{-1}(x) = x^{23} - 1.) \tag{3.15}$$

$$c(x)c(\frac{1}{x}) = \sum_{u=1}^{w} x^{i_u} \sum_{v=1}^{w} x^{-i_v} = \sum_{u,v=1}^{w} x^{i_u - i_v}$$

When $u = v$, then

$$\sum_{u=v=1}^{w} x^{i_u - i_v} = w \equiv 0 \pmod 2,$$

so all such terms cancel. We can therefore define the sum

$$\sum_{u \neq v, u, v=1}^{w} x^{i_u - i_v} =: \sum_{i=1}^{22} s_i x^i = s_1 x + s_2 x^2 + ... + s_{22} x^{22}$$

This sum adds up to zero in $GF(2)/(x^{23}-1)$, so each s_i is an even number. Another important property of this sum is that $s_i = s_{23-i}$. This can be shown by considering the product

$$\begin{aligned} c(x)c\Big(\frac{1}{x}\Big) &= (x^a + x^b)(x^{-a} + x^{-b}) \\ &= 1 + x^{a-b} + x^{b-a} + 1 \\ &\equiv x^{a-b} + x^{b-a} \pmod 2 \end{aligned} \tag{3.16}$$

If we let $a > b$, then $a-b$ is positive, while $b-a$ is negative. By factorizing out (-1) we get $x^{-(a-b)}$, and since $x^{23} = 1$, we can multiply by this and get $x^{a-b} + x^{23-(a-b)} = x^i + x^{23-i}$.

Having removed all terms where $u = v$, the sum $\sum_{u \neq v, u, v=1}^{w} x^{i_u - i_v}$ has $w(w-1)$ terms. This lets us show the following:

$$w(w-1) = \sum_{i=1}^{22} s_i = \sum_{i=1}^{11} 2s_i \equiv 0 \bmod 4. \tag{3.17}$$

The sum can be changed because $s_i = s_{23-i}$, and since s_i is already even, $2s_i$ must be a multiple of 4. With $(w-1)$ being odd, we can conclude that $w \equiv 0 \pmod 4$.

□

This proves that any codeword of even weight is zero modulo 4. The next step is to use this to find the actual minimum distance of the code, and show that the code is indeed perfect.

Let A_i be the number of codewords of weight i. For this specific code, we have a bijection between $A_i = |\{c \,\epsilon\, C \mid w(c) = i\}|$ and $A_{23-i} = |\{c \,\epsilon\, C \mid w(c) = 23 - i\}|$, namely that $A_i = A_{23-i}$. This is due to the fact the all-one vector is contained in the code, which can we can be shown by rewriting the vector as

$$c = (1, 1, ..., 1) = \sum_{i=0}^{22} x^i = \frac{x^{23} - 1}{x - 1} = m_1(x)m_{-1}(x).$$

Now, if a codeword $c(x)$ has weight i, then the codeword $c'(x) = c(x) + (1, 1, ..., 1)$ will be the inverse codeword and have weight $23 - i$. Since we can do this for all codewords $c(x)$, we have shown that $A_i = A_{23-i}$. This tells us the following about the minimum distance:

$$A_5 = A_{18} = 0,$$

because 18 is not divisible by 4, and $A_6 = 0$ for the same reason. The minimum distance is therefore at least equal to 7. Finally,

$$2^k \sum_{i=0}^{e} \binom{n}{i} (q-1)^i \;=\; 2^{12} \sum_{i=0}^{3} \binom{23}{i} \;=\; 2^{23},$$

proving that the minimum distance is exactly 7, and that the code is perfect. The parameters for this code are therefore [23,12,7].

Chapter 4

Coding theory in school

The renewal of the curriculum in Norwegian schools started in 2020, and aims to deepen pupils understanding in various subjects, as well as highlight the connection between them. As for mathematics, computer science and programming will have a more prominent role, as improving the digital skills of the pupils is a part of the new learning goals (Utdanningsdirektoratet, 2020).

Certain mathematical concepts used in this thesis are already part of the learning goals of different grade levels. Others could be introduced, given that it happens in the right context. In the 10th grade, pupils are expected to learn polynomial multiplication, and during the 12th grade they learn polynomial division. At this point, it could be possible show them codewords represented as polynomials, and let pupils try to generate their own codewords. Modular arithmetic with polynomials could prove a bit challenging for some, but could be presented to a more advanced science class.

In elementary school, children learn how to tell the time, meaning they are introduced to modular arithmetic at a very young age. Whenever we do calculations involving time on a clock, we work with groups under addition modulo 12 and 24. The concept of finite groups and fields is something that could be introduced more formally at some point, without going into too much detail on group theory. Instead, pupils could be introduced to other finite groups $\mathbb{Z}/n\mathbb{Z}$, and practise modular addition with different numbers. These kinds of computations are something most people are able to perform.

One of the benefits of introducing these concepts to pupils is that it shows a clear connection between the mathematics they learn at school, and real life problems that need solving. If anyone are interested in pursuing a career in computer science, then this can serve as a nice demonstration of the types

of challenges that arise in digital communication and storing of information.

Bibliography

[1]K. S. Andrews, D. Divsalar, S. Dolinar, J. Hamkins, C. R. Jones and F. Pollara, (2007), *The Development of Turbo and LDPC Codes for Deep-Space Applications*,IEEE.

[2] Lindell, Y. (2010). *Introduction to Coding Theory Lecture Notes*. Bar-Ilan University, Israel.

[3] van Tilborg, H. C. A. (1993). *CODING THEORY a first course.*

[4] Utdanningsdirektoratet (2020) *Læreplan i matematikk 1.–10. trinn (MAT01-05)*. Fastsatt som forskrift. Læreplanverket for Kunnskapsløftet 2020.