Partitions of Codes

Karl Michael Vincent Waugh

for a DPhil in Mathematics

University Of Sussex

March 2013

Declaration

I hereby declare that this thesis has not been and will not be, submitted in whole or in part to another University for the award of any degree.

Signature:.....

Dedication

For Chloe, with love

Acknowledgements

I would like to thank my advisor, Prof. James W.P. Hirschfeld for his confidence in my mathematical skill, his support, his patience and his advice throughout my entire research period, this thesis would not have happened without his faith in me. I would like to thank my parents for all the love and support they have given me, and for being constantly baffled that I could do a doctorate in mathematics, and yet still being proud of me. I would like to thank *all* of my friends for their ability to bring me back to earth and for telling me not to worry when things seemed heavy, I might just have gone mad without you. Finally I would like to thank my fiancée and the love of my life, Chloe, for her unwavering support and love whilst I committed to something as foolhardy as a doctorate, I love you so much, and to our cat, Bettie, for purring and cuddles, and for not even knowing what mathematics is, I think she just wishes I would bring home more fish.

Abstract

In this thesis we look at coding theory wherein we introduce the concept of perspective, a generalisation on the minimum distance of a code, which naturally leads to a partition of the code. Subsequently we introduce focused splittings, which shall be shown to be a generalisation of perfect codes. We investigate the existence of such objects, and address questions such as the complexity of finding a focused splittings, which we show to be NP-Complete. We analyse the symmetries of focused splittings. We use focused splittings to address the problem of error correction and we construct an encoding method based on them. Finally we test this construction for various classes of focused splittings.

Contents

List of Figures

A man who does not think for
himself, does not think at all

Oscar Wilde

Chapter 1

Introduction To The Problem

1.1 In This Thesis

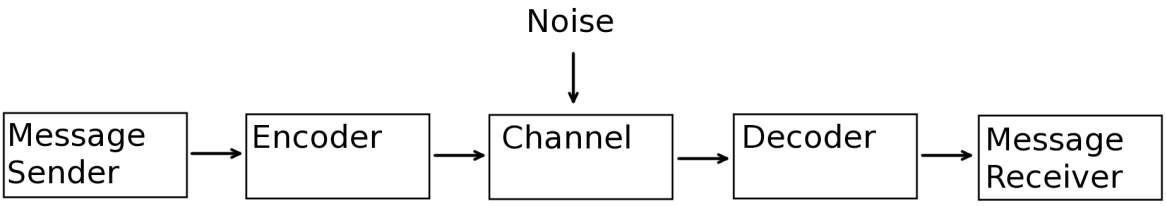
In this thesis we are looking at coding theory, we shall go over the core ideas and themes and hopefully show how they lead onto the work we shall present. Chapter 1 shall introduce all the relevant previous work in coding theory, roughly following the work in [17] and [36], giving an overview of the subject to inform the subsequent Chapters. In Chapter 2 we will be introducing the concept of *perspective*, and shall show how this generalises the traditional concept of the minimum distance of a code. We show how perspective leads to a partition of the code into distinct codes and we distinguish those which we consider to be most natural, which we subsequently call *focused splittings*, this assertion, although ultimately a value judgement, will be further reinforced by demonstrating that in the extremal cases of perspective all the partitions formed will be focused splittings, and also that for the case where the code in question is the whole space that the focused splittings are precisely the perfect codes on that space and that perfect codes will induce a focused splitting on the space. We follow this up by giving a few constructions of focused splittings for codes with specific parameters or properties. In Chapter 3 we give an overview of complexity theory, including an explanation and proof of Cook's Theorem, we follow this by proving that the question of deciding whether a code has a focused split-

ting is **NP-complete**. Chapter 4 deals with the inevitable question of automorphisms of focused splittings and how they are related to various structural properties of the focused splittings themselves. For Chapter 5 we move our attention on to the uses of focused splittings in an error correcting capacity, we start by briefly examining alternative methods of error correction and we then move on to a construction of a code, with a potentially higher probability of a correct decoding. Chapter 6 follows the work of Chapter 5 by testing various constructions of focused splittings with the construction given in Chapter 5, these are compared under both normal random error channels and burst error channels giving various results.

1.2 What is Coding Theory?

In many scenarios in the world we may wish to transfer information from one computer, machine, or other electronic device to another. Regardless of the medium of transmission, be it wires, radio waves or something else, there is a potential for interference to disrupt the message being sent, there will be errors introduced. Obviously an error in a digital message can change the entire meaning of the message and this is clearly undesirable. The original objective of coding theory, and of codes, is to be able to correct these errors which is achieved by introducing redundancy, that is transmitting more than the original message so that the original message can be recovered assuming that there are not "too many" errors. Thus we introduce an message encoder and a message decoder to add redundancy and correct errors respectively, giving the situation given in Figure 1.1.

Figure 1.1: Message transmission schematic



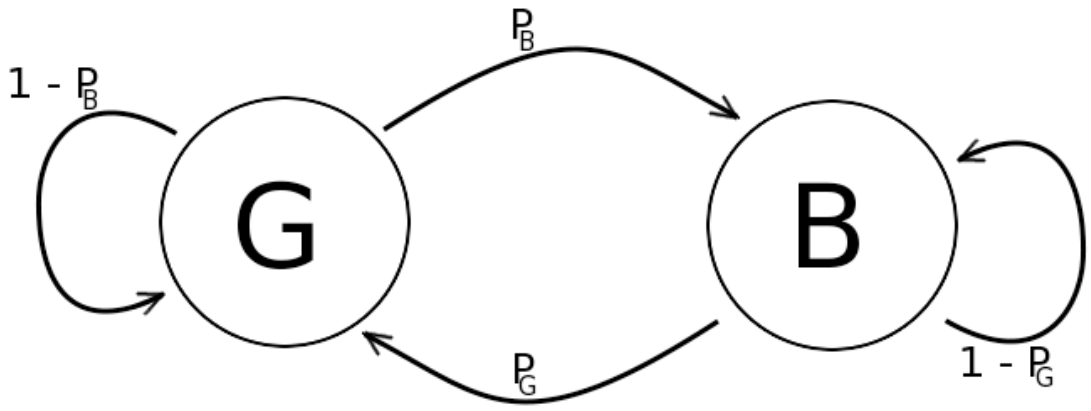
The message encoder will take an information message, that is one where every symbol counts and is important to the receiver and encode it as one of a number of longer messages which are hopefully robust against a certain number of errors, the noisy channel will change some of the symbols of the message, depending on the channel the number and distribution of these will change, we shall discuss this further in Section 1.3, wherein the decoder will either detect the errors and if possible correct the errors such that the original information message is received and presented to the user. Codes have found many other applications in cryptography, see [5] and [31] as well as in projective geometry [18] and design theory [2], but we will not be focusing on them but shall instead just consider codes as a method of correcting errors in information and as objects of pure mathematical interest.

1.3 Channel Noise

It is clear that every different medium of transferring information will have different properties and quirks. Noise on a channel is a process which introduces errors to a message, an error is where a symbol being transmitted gets changed to a different symbol. We shall be looking at two of the main models of noise on a channel, random noise and burst noise. Random noise does essentially what it says, it is where errors are caused at each co-ordinate of the transmitted message independently with a fixed probability p, thus the location of some errors has no effect on the location of other errors, thus a model

channel with random noise can be described just by the probability p and simulated by use of a random number generator to see if an event occurs at each co-ordinate with probability p. Burst noise is where the errors come in bursts, that is they are clustered together, this could occur if a cable got disconnected or if there was a lightning storm affecting radio waves or other such intermittent phenomena, it is where errors would occur frequently for a number of co-ordinates before becoming less frequent afterwards, we model burst errors by the Gilbert-Elliot model [15] [12] wherein we view the channel as a Markov chain consisting on two states, traditionally called G and B, for "Good" and "Bad". When the channel is in state G there is a probability of P_g of an error occurring, and when the channel is in state B there is a probability of P_b of an error occurring, with the usual proviso that $P_b > P_g$ to fit our description of G being the "Good" state and B being the "Bad" state. Transitions between the states occur with probabilities P_G, for going from state B to state G, and with probability P_B, for going from state G to state B. Thus the probabilities of staying in state G or B would be $1-P_B$ and $1-P_G$ respectively, this is all shown in Figure 1.2

Figure 1.2: State transitions for a Gilbert-Elliot model



Moreover for some models [26] we take the simpler approach and set values of P_g and

P_b to be 0 and 0.5 respectively, this is so that in state G no errors occur, and in state B errors occur with a probability of 0.5. The reason we consider 0.5 to be the error rate in the "Bad" state is because with an error rate of 0.5 an error is as likely to happen as not, and as such the channel becomes completely unreliable. If an error rate was 1 then for a binary channel the message received is the negative of the message sent, because every 0 would be changed to a 1 and every 1 would be changed to a 0. For similar reasons with the random noise channel we allow the error probability p to range from 0 to 0.5. The reason we take this simpler approach to the Gilbert-Elliot model is because it reduces the number of variables required to describe the channel whilst still giving us the *bursty* nature we are after. In more detailed models we could consider there to be a certain amount of noise even when not in the "Bad" state and as such we would set $P_g \geq 0$ but this is superfluous to our requirements. For the purpose of this thesis we only require either the unbiased scenario of a random error channel or the bursty nature of the burst error channel.

For the burst error channel with $P_g = 0$ and $P_b = 0.5$, from hereon these value will be fixed as this, we can work out the probability $\mathfrak{P}$ of an error. By use of the transition matrix for the Markov chain, which is

$$\begin{pmatrix} 1 - P_B & P_B \\ P_G & 1 - P_G \end{pmatrix}.$$

Thus the stationary distributions [16], π_G and π_B, can be found by

$$(\pi_G, \pi_B) \begin{pmatrix} 1 - P_B & P_B \\ P_G & 1 - P_G \end{pmatrix} = (\pi_G, \pi_B)$$

and thus

$$(1 - P_B)\pi_G + P_G\pi_B = \pi_G$$

which implies

$$\pi_G - P_B\pi_G + P_G\pi_B = \pi_G.$$

Thus we find

$$\frac{\pi_G}{\pi_B} = \frac{P_G}{P_B}$$

which allows us to calculate the overall probability of an error $\mathfrak{P}$ which can be worked out as

$$\begin{aligned}
\mathfrak{P} &= 0.5\big(\frac{\pi_B}{\pi_B + \pi_G}\big) \\
&= 0.5\big(\frac{\pi_B + \pi_G}{\pi_B}\big)^{-1} \\
&= 0.5\big(1 + \frac{\pi_G}{\pi_B}\big)^{-1} \\
&= 0.5\big(1 + \frac{P_G}{P_B}\big)^{-1} \\
&= 0.5\big(\frac{P_B + P_G}{P_B}\big)^{-1} \\
&= 0.5\big(\frac{P_B}{P_B + P_G}\big).
\end{aligned}$$

Subsequently we can see that $\mathfrak{P} = 0.5(\frac{P_B}{P_B+P_G})$ and thus we can notice that $\mathfrak{P}$ depends on the ratio of P_B and P_G rather than specific values, as well as the fact that an increase in P_B, for a fixed value of P_G will increase $\mathfrak{P}$. Moreover for any fixed value of P_G, we see that $\mathfrak{P}$ can range from 0 to $0.5(\frac{1}{1+P_G})$ and thus a natural question to ask is "what would be the difference between channels which admit the same value of $\mathfrak{P}$ but for which P_B and P_G are different?". The answer to this question, in a non-rigourous manner, is that the smaller the value of P_G the "burstier" the channel becomes, this means that the errors will be clumped together. We can see this by considering the expected amount, $E(G)$ of time before a "Bad" state B changes to a "Good" state G, this gives us the expected number of time steps that the system is in state B in one go. When we are in state B changing to state G happens with probability P_G then the expected time for this to occur

is

$$E(G) = \frac{1}{P_G}$$

and thus a lower value for P_G will lead to a higher value of $E(G)$, and thus for a given probability p and a given value P_G then by setting

$$P_B = \frac{2pP_G}{1-2p}$$

we get that

$$\begin{aligned}
\mathfrak{P} &= 0.5\big(\frac{P_B}{P_B + P_G}\big) \\
&= 0.5\big(\frac{\frac{2pP_G}{1-2p}}{\frac{2pP_G}{1-2p} + P_G}\big) \\
&= 0.5\big(\frac{\frac{2pP_G}{1-2p}}{\frac{2pP_G + P_G - 2pP_G}{1-2p}}\big) \\
&= 0.5\big(\frac{2pP_G}{2pP_G + P_G - 2pP_G}\big) \\
&= 0.5\big(\frac{2pP_G}{P_G}\big) \\
&= 0.5\big(2p\big) \\
&= p
\end{aligned}$$

and thus we can fix $\mathfrak{P}$ to any value we choose. Thus if the expected number of errors is the same, but the expected time spent in state B in any one sitting is greater then the errors must be clumped together more. Thus a lower value of P_G leads to a "burstier" channel.

1.4 Block Codes

In this section we shall be going over the basics of the theory of block codes, we shall define everything from first principles and prove the relevant theorems, giving a basic

understanding of the area. We will mention results or interesting asides but not prove them if they are not necessary for our need. A fully and more rounded view point can be gained from textbooks such as [17] and [36].

If we wish to protect information against potential errors, we need a way of sending that information so that as long as there are not too many errors then we can work out what the original message was, and thus what the original information was. The method we shall be following is that of block codes, the other main method of error correction is that of convolutional codes which we shall briefly remark on in Chapter 5. In block codes we have a space from which all possible messages can be sent, we shall be using an alphabet, Σ of q symbols, and we shall restrict q to a prime power so that we can consider our alphabet to be $\mathbb{F}_q$. This restriction becomes relevant later on. If we send messages of length n, then our messages belong to Σ^n, usually $(\mathbb{F}_q)^n$, and we want our message to be from a subset of Σ^n such that they are "far apart" and can not be easily mistaken, this shall be clarified later. Therefore we shall require a method to measure how far apart two messages are, this is the *Hamming distance*. We shall be using the notation $c_1(i)$ to refer to the i-th coordinate of c_1.

Definition 1.1. The *Hamming distance*, $d(c_1, c_2)$ for $c_1, c_2 \in \Sigma^n$ is

$$d(c_1, c_2) = |\{i : c_1(i) \neq c_2(i)\}|$$

that is the number of differences in co-ordinates.

Lemma 1.2. *The Hamming distance is a metric.*

Proof (i) As $d(c_1, c_2)$ is defined as the cardinality of a set thus $d(c_1, c_2) \geq 0$ for all c_1, c_2.

(ii) If $d(c_1, c_2) = 0$ then there would be no differences in co-ordinates and thus $c_1 = c_2$.

(iii) It is clear that as the i-th co-ordinate differs between c_1 and c_2 that we get that $d(c_1, c_2) = d(c_2, c_1)$.

(iv) For three words c_1, c_2 and c_3 in Σ^n, if we know $d(c_1, c_3)$, then we can see for each co-ordinate i such that $c_1(i)$ is not equal to $c_3(i)$ then either $c_1(i)$ is not equal to $c_2(i)$ or $c_2(i)$ is not equal to $c_3(i)$, otherwise we would have that $c_1(i)$ is equal to $c_2(i)$ and that is equal to $c_3(i)$, causing a contradiction. Therefore

$$\{i : c_1(i) \neq c_3(i)\} \subseteq \{i : c_1(i) \neq c_2(i)\} \cup \{i : c_2(i) \neq c_3(i)\}$$

thus we get that

$$\begin{aligned} |\{i : c_1(i) \neq c_3(i)\}| &\leq |\{i : c_1(i) \neq c_2(i)\} \cup \{i : c_2(i) \neq c_3(i)\}| \\ &\leq |\{i : c_1(i) \neq c_2(i\}| + |\{i : c_2(i) \neq c_3(i)\}| \\ d(c_1, c_3) &\leq d(c_1, c_2) + d(c_2, c_3) \end{aligned}$$

thus showing the triangle inequality and completing the proof.

□

We now define what we mean by a code.

Definition 1.3. A code C is an $(n, M, d)_q$ code, where n, M, d and q are all positive integers, if and only if

(i) C is a subset of Σ^n, where $|\Sigma| = q$. Note we will often take $\Sigma = \mathbb{F}_q$.

(ii) $|C| = M$.

(iii) For every pair of words $c_1, c_2 \in C$ we have that $d(c_1, c_2) \geq d$, moreover there is a pair such that equality holds. We will refer to d as the *minimum distance* of the code.

Definition 1.4. We say a code C is i error *detecting* if and only if for every codeword $c \in C$, if c is transmitted and has j errors in the received word, where $1 \leq j \leq i$, then we can systematically *detect* that errors have occurred, that is for all j, when j errors occur, we can be certain that some errors have occurred.

Definition 1.5. We say a code C is i error *correcting* if and only if for every codeword $c \in C$ if c is transmitted and has j errors in the received word, where $1 \leq j \leq i$, then we can systematically can *correct* the errors that have occurred, that is we can calculate c from the received words when j errors occur.

Lemma 1.6. *A code $C = (n, M, d)_q$ can detect $d-1$ errors or correct $\lfloor \frac{d-1}{2} \rfloor$ errors.*

Proof As C has a minimum distance of d, then by changing $\leq d-1$ co-ordinates in a codeword, the received word can not have been changed to another word in C, as this would take a minimum of d changes, and thus will belong to $(\mathbb{F}_q)^n \setminus C$, as we know that the sent word will belong to C, thus we can see that some errors have occurred, thus C is $d-1$ error detecting.

If $\lfloor \frac{d-1}{2} \rfloor$ errors occur in a codeword c, giving us a word $w \in (\mathbb{F}_q)^n$ then we can see that $d(c', w) > \lfloor \frac{d-1}{2} \rfloor$ for all codewords $c' \neq c$. By contradiction if $d(c', w) \leq \lfloor \frac{d-1}{2} \rfloor$ then we get $d(c, c') \leq d(c, w) + d(c', w) \leq \lfloor \frac{d-1}{2} \rfloor + \lfloor \frac{d-1}{2} \rfloor \leq d-1$ which is a contradiction with the minimum distance being $d-1$. Thus we can see c is the only codeword within a distance of $\leq \lfloor \frac{d-1}{2} \rfloor$ of w and by assuming that at most $\lfloor \frac{d-1}{2} \rfloor$ errors occurred we can correct w to c. □

Locating c from w as in the above proof is not always a simple calculation, as codes can be quite big in terms of the number of codewords to check, this is the crux of the decoding problem in coding theory, whereby we wish to know the most efficient method of decoding, we shall discuss this aspect later. We would obviously want a code to both correct a lot of errors, and also to transmit a lot of information, and clearly this is a bit ambitious.

Definition 1.7. The *rate* of a code $C = (n, M, d)_q$ is $R(C) = \frac{\log_q M}{n}$. Thus the higher the rate of the code, the more efficient the code is.

It is clear that if we wish to find "good" codes then we wish to be able to find codes which have a high minimum distance, and thus error correction, and also have a high rate. To further this search we now discuss linear codes; linear codes are codes which are linear subspaces of $(\mathbb{F}_q)^n$, now although this gives no reason why they should provide the most efficient codes, it does allow us to use a lot of machinery to examine to possibilities.

Definition 1.8. A *linear code* $C = [n, k, d]_q$ is a linear subspace of $(\mathbb{F}_q)^n$ of dimension k, with a minimum distance of d. Thus C is a subset of $(\mathbb{F}_q)^n$ such that for all c_1, $c_2 \in C$ we have $c_1 + c_2 \in C$ and $ic_1 \in C$ for all $i \in \mathbb{F}_q$.

Now clearly an $[n, k, d]_q$ code is also an $(n, q^k, d)_q$ code, although this is obviously only one way. The advantages of restricting our attention to linear codes are that they can be simpler to examine, as checking the minimum distance of a linear code involves only checking the distances between each word and the 0^n word, which is always in a linear code, we can describe a linear code by just describing a basis for it, and we can use the theory of linear algebra to give us various properties more easily than we could otherwise. We also note that in a linear code the rate is the ratio of the number of symbols holding information to the total length of the code.

Definition 1.9. For a linear code C the *weight* of a codeword c is the number of non-zero co-ordinates and thus

$$w(c) = d(c, 0^n).$$

Definition 1.10. A $k \times n$ matrix whose rows form a basis for a linear $[n, k, d]_q$ code is called a *generator matrix* of the code.

We can thus see that the rate of a linear code can be more simply described as $R(C) = \frac{k}{n}$ since $\log_q M = \log_q q^k = k$.

Definition 1.11. Two linear codes are *equivalent* if and only if one can be obtained from the other by a combination of permuting the co-ordinates of the code, and multiplying the symbols appearing in a fixed co-ordinate by a non-zero scalar.

Lemma 1.12. *Two $k \times n$ matrices generate equivalent linear codes over $(\mathbb{F}_q)$, if and only if one can be obtained from the other by a combination of permuting the rows, multiplying the rows by a non-zero scalar, addition of a scalar multiple of one row to another, permuting the columns and multiplying any column by a non-zero scalar,*

Proof We note that permuting, multiplying and addition of the rows preserves the fact that matrix is a basis for the subspace. Permuting the columns and multiplying the columns is equivalent to permuting the co-ordinates of the code and multiplying a fixed co-ordinate. Thus the codes are equivalent. □

We also briefly note that the generator matrix of code C can always be transformed to the form

$$[I_k|A]$$

where I_k is the $k \times k$ identity matrix and A is a $k \times (n-k)$ matrix. This is done by a combination of the above moves and can be seen in detail in [17].

We recall that for two vectors $c_1 = (x_1, x_2, \ldots, x_n)$ and $c_2 = (y_1, y_2, \ldots, y_n)$ the *inner product* of them $c_1 \cdot c_2 = x_1y_1 + x_2y_2 + \ldots x_ny_n$.

Definition 1.13. For a linear code $C = [n, k, d]_q$ the code

$$C^{\perp} = \{w \in (\mathbb{F}_q)^n : w \cdot c = 0 \ \forall c \in C\}$$

that is the set of words in $(\mathbb{F}_q)^n$ orthogonal to every word of C, we call $C^{\perp}$ the *dual* of C.

Dual codes are very interesting in coding theory and are studied a lot, we mainly introduce them for something attaining completeness and for the following development.

Lemma 1.14. *For a linear code $C = [n, k, d]_q$ with a generator matrix G and a dual code $C^\perp$ we get that $c \in C^\perp$ if and only if $cG^T = 0^k$.*

Proof As the rows of G are codewords of C then the columns of G^T are codewords, and as such $cG^T = 0^k$. Assuming that $cG^T = 0^k$, let the rows of G be $g_1, g_2, \ldots, g_k$ and thus we get that $c \cdot g_i = 0$ for all g_i. As G forms a basis for C, let w be in C and as such $w = \sum_{i=1}^{k} \lambda_i g_i$, for λ_i in $\mathbb{F}_q$, and thus

$$\begin{aligned} c \cdot w &= c \cdot \sum_{i=1}^{k} \lambda_i g_i \\ &= \sum_{i=1}^{k} \lambda_i (c \cdot g_i) \\ &= \sum_{i=1}^{k} \lambda_i (0) = 0 \end{aligned}$$

and thus c is in $C^\perp$. □

Lemma 1.15. *For a linear code $C = [n, k, d]_q$ we get that the dual code $C^\perp = [n, n-k, d']_q$.*

Proof We first show $C^\perp$ is linear, let $c_1, c_2 \in C^\perp$, $w \in C$ and $\lambda_1, \lambda_2 \in \mathbb{F}_q$, then we consider $\lambda_1 c_1 + \lambda_2 c_2$. We can see that

$$\begin{aligned} (\lambda_1 c_1 + \lambda_2 c_2) \cdot w &= \lambda_1 (c_1 \cdot w) + \lambda_2 (c_2 \cdot w) \\ &= \lambda_1 (0) + \lambda_2 (0) = 0 \end{aligned}$$

and thus $(\lambda_1 c_1 + \lambda_2 c_2)$ belongs to $C^\perp$ and $C^\perp$ is linear. We now show that for a code C with length n and dimension k we get $C^\perp$ having dimension $n - k$. For induction we take a base case of $k = 1$, thus C has a generator matrix

$$G = (g_1)$$

and thus for a word c to belong to $C^{\perp}$ we must have $c \cdot g_1 = 0$, we write

$$g_1 = (g_1(1), g_1(2), \ldots, g_1(n))$$

and thus

$$\begin{aligned} c \in C^{\perp} &\iff c \cdot g_1 = 0 \\ &\iff \sum_{i=1}^{n} c(i) g_1(i) = 0 \\ &\iff \sum_{i=1}^{n-1} c(i) g_1(i) = -c(n) g_1(n) \\ &\iff \frac{1}{-g_1(n)} \sum_{i=1}^{n-1} c(i) g_1(n) = c(n) \end{aligned}$$

and thus we get that $c(n)$ is determined by the set of $c(i)$ for $1 \leq i \leq n-1$ and thus $C^{\perp}$ has dimension $n-1$. For our inductive step we assume that the statement of the theorem hold for a code C with dimension $k-1$ and consider a code with dimension k.

Let C_1 be the code generated by the matrix

$$G_1 = \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_{k-1} \end{pmatrix}$$

and let C_2 be the code generated by the matrix

$$G_2 = \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_{k-1} \\ g_k \end{pmatrix}$$

thus C_1 has dimension $k-1$ and C_2 has dimension k, moreover as $G_1 \subseteq G_2$ then all the restrictions of G_1 on $C_1^\perp$ are also restricting $C_2^\perp$ and thus $C_2^\perp \subseteq C_1^\perp$. Subsequently the dimension of $C_2^\perp$ is less than the dimension of $C_1^\perp$ which by the inductive hypothesis is $n-k+1$. We consider a word $c \in C_1^\perp$ and consider when $c \in C_2^\perp$, that is when

$$cG_2^T = 0^k$$

but we know that $c \cdot g_i = 0$ for $1 \leq i \leq k-1$ so we only need to consider $c \cdot g_k$, that is c will belong to $C_2^\perp$ when $c \cdot g_k = 0$. We write

$$g_k = (\lambda_1, \lambda_2, \ldots, \lambda_n)$$

and thus

$$\begin{aligned} c \in C_2^\perp &\iff c \cdot g_k = 0 \\ &\iff \sum_{i=1}^{n} c(i)\lambda_i = 0 \\ &\iff \sum_{i=1}^{n-k+1} c(i)\lambda_i + \sum_{i=n-k+2}^{n} c(i)\lambda_i = 0 \end{aligned}$$

now we know that as $C_1^\perp$ has dimension $n-k+1$, and as $c \in C_1^\perp$ that a choice of $c(i)$ for

$i \leq n-k+1$ will fix $c(i)$ for $i \geq n-k+2$ and thus we know that the value

$$\sum_{i=n-k+2}^{n} c(i)\lambda_i$$

is determined by c and g_k as $c \in C_1^{\perp}$. We shall call this value $\Omega(c, g_k)$. Thus we get

$$\begin{aligned} c \in C_2^{\perp} &\iff \sum_{i=1}^{n-k+1} c(i)\lambda_i + \Omega(c, g_k) = 0 \\ &\iff \sum_{i=1}^{n-k} c(i)\lambda_i + \lambda_{n-k+1}c(n-k+1) + \Omega(c, g_k) = 0 \\ &\iff \frac{1}{-\lambda_{n-k+1}}\big(\sum_{i=1}^{n-k} c(i)\lambda_i + \Omega(c, g_k)\big) = c(n-k+1) \end{aligned}$$

and thus we get that $c(n-k+1)$ is determined by $c(i)$ for $1 \leq i \leq n-k$ and thus $C_2^{\perp}$ has dimension $n-k$. This completes our inductive step and thus proves the theorem. □

Lemma 1.16. *The dual of a dual is the code itself, that is* $(C^{\perp})^{\perp} = C$.

Proof We can see that $C \subseteq (C^{\perp})^{\perp}$ as every word in C is orthogonal to every word in $C^{\perp}$. As the dimension of $(C^{\perp})^{\perp}$ is $n-(n-k) = k$ is the dimension of C, we see that $C = (C^{\perp})^{\perp}$. □

Definition 1.17. A *parity check matrix* H for a linear code $C = [n, k, d]_q$ is a $(n-k) \times n$ matrix over $\mathbb{F}_q$ such that H is the generator matrix for $C^{\perp}$.

Lemma 1.18. *We can define* C *by a parity check matrix such that*

$$C = \{c \in (\mathbb{F}_q)^n : cH^T = 0^{n-k}\}$$

that is every word is orthogonal to H.

Proof By Lemma 1.16 we get $C = (C^{\perp})^{\perp}$ and by Lemma 1.14 we have that

$$C^{\perp} = \{c \in (\mathbb{F}_q)^n : cG^T = 0^k)\}$$

where G is the generator matrix of C, thus

$$C = (C^{\perp})^{\perp} = \{c \in (\mathbb{F}_q)^n : cH^T = 0^{n-k}\}$$

where H is the generator matrix for $C^{\perp}$. □

We now discuss how knowing structural properties about H lead us to knowing properties about C, most important of these is the following theorem wherein we connect the linear dependancy and independency of columns of H with the minimum distance of C.

Theorem 1.19. *Let $C = [n, k, d]_q$ with a parity check matrix H. Then C has minimum distance d if and only if any $d-1$ columns of H are linearly independent and some d columns are linearly dependent.*

Proof As C is linear then we know that if the minimum distance is d then there exists a codeword c with exactly d non-zero co-ordinates. If we label the columns of H as $H_1, H_2, \ldots, H_n$ then for every codeword $c \in C$ we get that

$$c(1)H_1 + c(2)H_2 + \ldots + c(n)H_n = 0$$

and thus for a codeword c such that $d(c, 0) = d$ then this gives a set of d linearly dependant columns of H. Similarly if there did exists a set of $d-1$ columns of H which were linearly dependant, columns $H_{\phi(1)}, H_{\phi(2)}, \ldots, H_{\phi(d-1)}$, where ϕ is a permutation of 1 to n, then there would exist scalars $\lambda_1, \lambda_2, \ldots \lambda_n$, not all zero, such that

$$\lambda_1 H_{\phi(1)} + \lambda_2 H_{\phi(2)} + \ldots + \lambda_{d-1} H_{\phi(d-1)} = 0$$

and thus we could create the word $x = (0, 0, \ldots, 0, \lambda_1, 0, \ldots, 0, \lambda_{d-1}, 0, \ldots, 0)$ with λ_i in the $\phi(i)$-th co-ordinate, and this would show $xH = 0^{n-k}$ and thus x would belong to C, but $d(x, 0) \leq d-1$ which is a contradiction of C having minimum distance d. Thus every set of $d-1$ columns must be linearly independent. □

One consequence of Theorem 1.19 is that it means that the quest for good codes is instead the search for good parity check matrices, that is a parity check matrix that produces a good code. The size and shape of the parity check matrix gives the length and dimension of the code, and the dependancies between columns gives us the minimum distance, so attention should be paid to methods of creating such parity check matrices rather than just searching for good codes arbitrarily.

When constructing a code, looking for a high rate of transmission and high error correction are clearly the best initial aims, but the method of decoding is also highly important [3], [4], [37]. The reason for this is that in practical application of codes, the dimension of a code may be fairly high, and as such we would not want to search through all the codewords to ascertain which is closest to the received word, therefore some process which automatically leads us to that conclusion in a shorter computation is necessary. One general method is that of syndromes.

Definition 1.20. For a code $C = [n, k, d]_q$ with a parity check matrix H, for a word $w \in (\mathbb{F}_q)^n$ we define the *syndrome* of w as

$$S(w) = wH^T.$$

We first note that we already know that a word belongs to the code C if and only if $cH^T = 0^{n-k}$ that is, we know that the syndrome of a codeword is the zero word.

Lemma 1.21. *Two words w_1, w_2 have the same syndrome if and only if they differ by a codeword c in C.*

Proof If w_1 and w_2 have the same syndrome we get that

$$\begin{aligned} S(w_1) = S(w_2) &\iff w_1H^T = w_2H^T \\ &\iff (w_1 - w_2)H^T = 0 \\ &\iff w_1 - w_2 = c \in C \end{aligned}$$

$$\iff \quad w_1 = w_2 + c$$

that is that they differ by a codeword c in C, and thus words in $(\mathbb{F}_q)^n$ which differ by a codeword will have the same syndrome. $\square$

We can class the words of $(\mathbb{F}_q)^n$ into cosets of the code C, each coset differing from C by fixed amount. Therefore to each syndrome we can associate a coset of C.

Definition 1.22. In a coset of C we define the *coset leader* to be a word of minimum weight, the coset leader is not necessarily unique but where it is not we choose one at random from the words of minimum weight.

We shall denote the coset leader of a coset associated with a syndrome as $f(S(w))$.

Lemma 1.23. *Every word w in $(\mathbb{F}_q)^n$ differs from its nearest neighbour in C by a word z where z is the coset leader of the coset associated with the syndrome of w. That is, let $z = f(S(w))$ then we decode w to $w - z$.*

Proof If two words w_1 and w_2 differ by a codeword then we know they have the same syndrome, $S(w_1) = S(w_2)$, and thus for a word w, which has a nearest neighbour c in the code C, there will exist a word z in the same coset as w where z will have minimum weight. As $(w - c)$ is in the same coset as w, and as c is w's nearest neighbour, thus $(w - c)$ is of minimum weight amongst words in that coset. Thus $c = w - z$, where z is in the same coset as w and of minimum weight thus z will be a coset leader and thus will be the coset leader associated with the syndrome of w. $\square$

Thus for a linear code we can construct a look up table of coset leaders and syndromes and merely refer to this when decoding a code, decoding a received word w to the codeword $w - f(S(w))$.

1.5 The Limits of Coding Theory

Between balancing the transmission rate of a code and the error correcting capabilities there are general questions about how much of one we can do if we wish to do a certain amount of the other. In this section we present a few bounds which give us a good idea of the limitations of block codes. We also introduce what is known as "the main coding theory problem".

Definition 1.24. The value $A_q(n,d)$ is the smallest value such that if $C = (n,M,d)_q$ then $M \leq A_q(n,d)$, that is $A_q(n,d)$ is the maximum number of codewords in a code of length n and minimum distance d over $\mathbb{F}_q$.

Definition 1.25. Thus we can say "the main coding theory problem" is to find a code $C = (n,M,d)_q$ such that $M = A_q(n,d)$, that is given q, n and d we want to determine $A_q(n,d)$.

Lemma 1.26. *If d is odd then a $(n,M,d)_2$ code exists if and only if a $(n+1,M,d+1)_2$ code exists. Thus we get that $A_2(n,d) = A_2(n+1,d+1)$.*

Proof Let $C = (n,M,d)_2$ code, and we define $\hat{C}$ as follows

$$\hat{C} = \{\hat{c} = (x_1,x_2,\ldots,x_n,w) : c = (x_1,\ldots,x_n) \in C, w = w(c) \mod 2\}.$$

As we have increased the length and not the number of codewords, we have that $\hat{C} = (n+1,M,d')_2$ code where $d \leq d' \leq d+1$, as the minimum distance can only have increased in the one new co-ordinate, but as $w(\hat{c}) = w(c) + (w(c) \mod 2)$ we have that if $w(c)$ is even then $(w(c) \mod 2) = 0$ and $w(\hat{c})$ is even, and if $w(c)$ is odd then $(w(c) \mod 2) = 1$ thus $w(\hat{c})$ is even, therefore we get that all weights in $\hat{C}$ are even and thus $\hat{C}$ has an even minimum distance, therefore $d' = d+1$. Thus the existence of a $(n,M,d)_2$ code implies the existence of a $(n+1,M,d+1)_2$ code.

Let $C = (n+1, M, d+1)_2$ code, as the minimum distance is $d+1$ there must exist two codewords c_1 and c_2 such that $d(c_1, c_2) = d+1$ and subsequently there must be $d+1$ co-ordinates which disagree, we let i be a co-ordinate where c_1 and c_2 disagree and we define

$$\bar{C} = \{\bar{c} = (x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) : c \in C\}.$$

We notice that this gives $\bar{C}$ a length of n, and as no two distinct words in C will be the same in $\bar{C}$, as $d+1$ is at least 2, thus $\bar{C}$ will have M words. The minimum distance of $\bar{C}$ can be at least d as only one co-ordinate has been removed, but as we choose i such that $d(\bar{x}, \bar{y}) = d(x, y) - 1 = d$ this shows that $\bar{C} = (n, M, d)_2$ code. Thus the existence of a $(n+1, M, d+1)_2$ code implies the existence of a $(n, M, d)_2$ code. This proves the lemma.

□

Lemma 1.27. *If there exists an $(n, M, d)_q$ code then there exists an $(n-1, M', d')_q$ code with $M' \geq \frac{M}{q}$ and $d' \geq d$. Thus we get that $A_q(n, d) \leq qA_q(n-1, d)$.*

Proof Let $C = (n, M, d)_q$ and we partition C into q classes, $C^{(i)}$ depending on the n-th co-ordinate of c. That is

$$C^{(i)} = \{c \in C : c(n) = i\}$$

and thus we see that $C = \bigcup_{i \in \mathbb{F}_q} C^{(i)}$, thus we can see that

$$\max\{|C^{(1)}|, |C^{(2)}|, \ldots, |C^{(q)}|\} \geq \frac{M}{q}$$

and thus picking j such that $|C^{(j)}| \geq \frac{M}{q}$ we define

$$\bar{C} = \{\bar{c} = (x_1, x_2, \ldots, x_{n-1}) : c \in C^{(j)}\}.$$

Subsequently $|\bar{C}| = |C^{(j)}|$, and as none of the words in $C^{(j)}$ did not differ in the n-th co-ordinate we can see that the minimum distance of $\bar{C}$ is equal to the minimum distance of

$C^{(j)}$, which by definition is greater or equal to the d. Thus we see that $\bar{C} = (n-1, M', d')_q$ code with $M' \geq \frac{M}{q}$ and $d' \geq d$. Also noting that $A_q(n, d') \leq A_q(n, d)$ for all values of n and $d' \geq d$ by virtue of the fact any $(n, M, d')_q$ code could be considered as a $(n, M, d)_q$ as all distances are by definition greater than d. Thus by letting our original code C have $M = A_q(n, d)$ we can see that

$$\frac{1}{q} A_q(n, d) \leq M' \leq A_q(n-1, d') \leq A_q(n-1, d)$$

and thus

$$A_q(n, d) \leq q A_q(n-1, d).$$

□

We now prove the *Plotkin* bound.

Theorem 1.28. *The following five statements are known as the Plotkin bound.*

(i) *For $q > 2$ then*

$$A_q(n, d) \leq q \left\lfloor \frac{d}{q(d-n)+n} \right\rfloor.$$

(ii) *For $q = 2$ if d is even and $2d > n$, then*

$$A_2(n, d) \leq 2 \left\lfloor \frac{d}{2d-n} \right\rfloor.$$

(iii) *For $q = 2$ if d is odd and $2d + 1 > n$, then*

$$A_2(n, d) \leq 2 \left\lfloor \frac{d+1}{2d+1-n} \right\rfloor.$$

(iv) *For $q = 2$ if d is even, then*

$$A_2(2d, d) \leq 4d.$$

(v) *For $q = 2$ if d is odd, then*

$$A_2(2d+1, d) \leq 4d + 4.$$

Proof We begin with the proof of (i). Let C be a $(n, M, d)_q$ code, and let us consider the value of

$$\sum_{c_1, c_2 \in C, c_1 \neq c_2} d(c_1, c_2)$$

and note that as d is the minimum distance thus $d(c_1, c_2) \geq d$ and that there are $M(M-1)$ ways to choose c_1 and c_2 and thus

$$M(M-1)d \leq \sum_{c_1, c_2 \in C, c_1 \neq c_2} d(c_1, c_2)$$

giving us a lower bound. We consider a $M \times n$ matrix T consisting of all the words of C. Let $t_{i,\alpha}$ be the number of occurrences of α in the i-th column of T. Thus by summing over each column and symbol of $\mathbb{F}_q$ we get that

$$\sum_{c_1, c_2 \in C, c_1 \neq c_2} d(c_1, c_2) = \sum_{\alpha in \mathbb{F}_q} \sum_{i=1}^{n} t_{i,\alpha}(M - t_{i,\alpha})$$

and noting that $M - t_{i,\alpha} = \sum_{\beta \neq \alpha} t_{i,\beta}$ we can see that

$$\sum_{\alpha in \mathbb{F}_q} \sum_{i=1}^{n} t_{i,\alpha}(M - t_{i,\alpha}) \; le \sum_{\alpha in \mathbb{F}_q} \sum_{i=1}^{n} t_{i,\alpha}(\sum_{\beta in \mathbb{F}_q, \beta \neq \alpha} t_{i,\beta})$$

and noting that $\sum_\alpha t_{i,\alpha} = M$ we can see that the right hand term is maximised when $t_{i,\alpha} = \frac{M}{q}$ and thus

$$\sum_{c_1, c_2 \in C, c_1 \neq c_2} d(c_1, c_2) \leq nq \frac{M}{q} \frac{(q-1)M}{q} = \frac{q-1}{q} nM^2.$$

Combining this with our lower bound we get

$$M(M-1)d \leq \frac{q-1}{q} nM^2$$

and thus

$$\begin{array}{rcl} M^2d - Md - \frac{q-1}{q}nM^2 & \leq 0 \\ \implies \quad Md - d - \frac{q-1}{q}nM & \leq 0 \\ \implies \quad M(d - \frac{q-1}{q}n) & \leq d \\ \implies \quad M & \leq \dfrac{d}{d - \frac{q-1}{q}n} \\ \implies \quad M & \leq \dfrac{qd}{qd-(q-1)n} \end{array}$$

and as M is even then M must be less than the lowest even integer below $fracqdqd-(q-1)n$ thus

$$M \leq q\lfloor \frac{qd}{q(d-n)+1} \rfloor.$$

.

For the proof of (ii) we substitute q for 2.

For the proof of (iii), we note by Lemma 1.26 that $A_2(n,d) = A_2(n+1,d+1)$ and if d is odd then $d+1$ is even and by (ii) we have

$$A_2(n,d) = A_2(n+1,d+1) \leq 2\lfloor \frac{d+1}{2(d+1)-(n+1)} \rfloor = 2\lfloor \frac{d+1}{2d+1-n} \rfloor$$

completing the proof for (ii).

If d is even then by Lemma 1.27 we can see that $A_2(2d,d) \leq 2A_2(2d-1,d)$ and by (ii) we can see that $A_2(2d-1,d) \leq 2\lfloor \frac{d}{2d-(2d-1)} \rfloor = 2d$ thus $A_2(2d,d) \leq 4d$. This proves (iiii).

If d is odd, then by Lemma 1.26 we have $A_2(2d+1,d) = A_2(2d+2,d+1)$ and the by (iv) we have that $A_2(2d+2,d+1) \leq 4(d+1) = 4d+4$. This proves (v). $\square$

Theorem 1.29. *The* Singleton *bound states that*

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

Proof Let $C = (n, M, d)_q$ code with $M = A_q(n,d)$, if we construct

$$\hat{C} = \{\hat{c} = (x_1, x_2, \ldots, x_{n-d+1}) : c = (x_1, x_2, \ldots, x_n) \in C\}$$

then we can see that as C had a minimum distance of d and we have deleted $d-1$ coordinates from C, thus $\hat{C}$ has a minimum distance of at least 1 and that $|\hat{C}| = |C|$ as no two words in C will have been mapped to the same word in $\hat{C}$ and so their cardinality is conserved. We finally note that as $\hat{C}$ has length $n-d+1$ that $|\hat{C}| \leq q^{n-d+1}$ which completes the proof. □

Another bound we can construct is know as the "Hamming bound", also known as the "sphere packing bound". The basic idea is that if we have a minimum distance of d, then every word within a distance of $e = \lfloor \frac{d-1}{2} \rfloor$ of a codeword can not be in the code, and thus we can bound the maximum number of codewords.

Theorem 1.30. *For a code $C = (n, M, d)_q$ we get that*

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

which can be written

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

which bounds the total number of words in C. Similarly for linear codes we get that

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

which thus bounds the dimension of the code k.

Proof If the code C has minimum distance d, then for $e = \lfloor \frac{d-1}{2} \rfloor$ and $c \in C$ we define the sets

$$B_e(c) = \{w \in (\mathbb{F}_q)^n : d(w, c) \leq e\}$$

and we can see that these sets do not intersect, for if they did then there would exist an element $w \in B_e(c_1)$ and $w \in B_e(c_2)$ and thus $d(w, c_1) \leq e$ and $d(w, c_2) \leq e$ thus

$$d(c_1, c_2) \leq d(w, c_1) + d(w, c_2)$$

which implies that

$$d(w, c_2) \geq d(c_1, c_2) - d(w, c_1) \geq d - d(w, c_2) \geq d - e > e$$

as $d(c_1, c_2) \geq d$ and $d(w, c_2) \leq e$ and as $d - e = d - \lfloor \frac{d-1}{2} \rfloor \geq \frac{d+1}{2} > e$ and this is a contradiction with $d(w, c_2) \leq e$. Thus the sets $B_e(c)$ do not intersect.

Counting the number of words in $B_e(c)$ for each c. We start by counting the number of words for a fixed value i of errors, we can see that for i errors there will be $\binom{n}{i}(q-1)^i$ words, as there have to be i co-ordinates where the words differ, and thus $\binom{n}{i}$ ways to choose these co-ordinates, and if they differ the symbol in that position must be one of the other $(q-1)$ symbols. Thus the total number of words is

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

and as there are $|C|$ sets $B_e(c)$ which do not intersect, we can see that $|C||B_e(c)| \leq |(\mathbb{F}_q)^n$. This completes the proof. $\square$

Now obviously it is useful to have lower bounds as well as upper bounds, so we can get a more thoroughly defined idea of the actual values of $A_q(n, d)$.

Theorem 1.31. *The* Gilbert Varshamov *bound states*

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

that is we can always construct a code with these values.

Proof Let $C = (n, M, d)_q$ code with $M = A_q(n, d)$. We can see that for every word w in $(\mathbb{F}_q)^n$ that there exists a word c in C such that $d(w, c) \leq d - 1$, if there did not then we could add w to C without decreasing the minimum distance and thus create a code C' with a greater value of M than $A_q(n, d)$, which is by definition the maximum. Thus

$$(\mathbb{F}_q)^n = \bigcup_{c \in C} B_{d-1}(c)$$

and subsequently

$$\begin{aligned} |(\mathbb{F}_q)^n| &= |\bigcup_{c \in C} B_{d-1}(c)| \\ &\leq \sum_{c \in C} |B_{d-1}(c)| \\ &= |C||B_{d-1}(c)| \\ &= |C| \sum_{i=0}^{d-1} \binom{n}{i} (q-1)^i \end{aligned}$$

and thus as $|(\mathbb{F}_q)^n| = q^n$ we get that

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

which completes the proof. □

There are of course other more sophisticated bounds on the parameters of codes, such as the linear programming bound. We shall not deal with these due to only wishing to outline the basics required by this thesis.

1.6 Perfect Codes

An important area in coding theory is that of *perfect codes*.

Definition 1.32. A code $C = (n, M, d)_q$ is a *perfect code* if and only if the spheres of radius $e = \lfloor \frac{d-1}{2} \rfloor$ around each codeword exactly fill the space $(\mathbb{F}_q)^n$. Equivalently

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

and thus for a linear code we get

$$\sum_{i=0}^{e} \binom{n}{i} (q-1)^i = q^{n-k}$$

subsequently these are codes which meet the Hamming bound in Theorem 1.30.

We note that as these codes meet the Hamming bound, perfect codes are important because they are, in some sense, the most efficient codes we can create as the spheres fit exactly into the space. They also admit some interesting properties, see [9].

It is possible to completely classify linear perfect codes, as seen in [35], and we shall briefly outline the results, without proving all the theorems themselves.

Firstly we shall deal with a number of "trivial" perfect codes. The complete space $(\mathbb{F}_q)^n$ with a minimum distance of 1, satisfies the criteria in Definition 1.32 as the sum totals 1 and $|C| = q^n$, any code consisting of a single codeword, as the value of e can be as large as we want, and thus equal to n and so the set $B_e(c) = C$, we also have binary repetition codes of an odd length n, that is a code $C = \{0^n, 1^n\}$, this is as n is odd and as the minimum distance is n we get $e = \lfloor \frac{d-1}{2} \rfloor = \frac{n-1}{2}$ as a whole number, and thus every word in $(\mathbb{F}_q)^n$ either consists of more 0s than 1s or visa versa and thus every word in $(\mathbb{F}_q)^n$ is closer to either 0^n or 1^n and thus C is perfect.

Now we move onto the known constructions for linear perfect codes, the main example for these are the "Hamming codes" [17] which we present here.

Definition 1.33. Fix r a positive integer and consider $(\mathbb{F}_q)^r$, the r-dimensional vector space of $\mathbb{F}_q$, we choose a set $S \subseteq (\mathbb{F}_q)^r$ of non-zero vectors which are not multiples of each other, as large as possible. There are $q^r - 1$ non-zero vectors in $(\mathbb{F}_q)^r$, and each one has $q - 1$ non-zero multiples, thus we can partition $(\mathbb{F}_q)^r$ in to $\frac{q^r-1}{q-1}$ classes

$$\{iv : i \in \mathbb{F}_q, v \in (\mathbb{F}_q)^r\}$$

and as such two vectors are scalar multiples of each other if and only if they are in the same class. Thus by choosing 1 vector from each class we can obtain a set S with $\frac{q^r-1}{q-1}$ vectors, and as any further vectors will require 2 vectors from some class then this is the maximal size of S. We now construct a parity check matrix H, where the columns of H are the vectors of S. We define the code Ham (r, q) as the code generated by the parity check matrix H.

Theorem 1.34. *The code* Ham (r, q) *over* $\mathbb{F}_q$ *is a perfect code.*

Proof By the construction in Definition 1.33 we get H as a $r \times \frac{q^r-1}{q-1}$ matrix, and thus Ham (r, q) is a $(\frac{q^r-1}{q-1}, \frac{q^r-1}{q-1} - r, d)_q$ code. If a set of 2 vectors are linearly dependant then one is a multiple of the other, and by definition none of the column vectors of H are

multiples of each other and therefore all sets of 2 columns are linearly independent and thus $d > 2$. To show that $d = 3$ we note that when choosing the classes in Definition 1.33 that there will be a class where $v_1 = (0, 0, \ldots, 0, 1)$, and all words in that class are multiples of that word, similarly for $v_2 = (0, 0, \ldots, 0, 1, 0)$ and $v_3 = (0, 0, \ldots, 0, 1, 1)$ and regardless of the representatives from each class taken we can show that these are linearly dependant. Let the representatives of each class be $w_1 = \lambda_1 v_1$, $w_2 = \lambda_2, v_2$ and $w_3 = \lambda_3 v_3$ then we take the linear combination

$$\begin{aligned}(q-1)\lambda_1^{-1}w_1 + (q-1)\lambda_2^{-1}w_2 + \lambda_3^{-1}w_3 &= (q-1)v_1 + (q-1)v_2 + v_3 \\ &= (0, 0, \ldots, 0, 0, -1) + (0, 0, \ldots, 0, -1, 0) + (0, 0, \ldots, 0, 1, 1) \\ &= (0, 0, \ldots, 0, 0, 0)\end{aligned}$$

and thus a set of 3 columns are linearly dependant and thus by Theorem 1.19 we see that $d = 3$. Now to prove the Ham (r, q) is perfect we note that

$$\begin{aligned}|Ham(r,q)|\sum_{i=0}^{e}\binom{n}{i}(q-1)^i &= q^{\frac{q^r-1}{q-1}-r}\sum_{i=0}^{1}\binom{\frac{q^r-1}{q-1}}{i}(q-1)^i \\ &= q^{\frac{q^r-1}{q-1}-r}(1 + \binom{\frac{q^r-1}{q-1}}{1}(q-1)^1) \\ &= q^{\frac{q^r-1}{q-1}-r}(1 + q^r - 1) \\ &= q^{\frac{q^r-1}{q-1}-r+r} \\ &= q^{\frac{q^r-1}{q-1}} = q^n\end{aligned}$$

and thus Ham (r, q) is perfect. □

The other main construction for perfect codes, and in fact the only other linear perfect codes, are the *Golay* codes, these are codes with parameters

$$G_{23} = [23, 12, 7]_2$$

and

$$G_{11} = [11, 6, 5]_3,$$

each of these codes also has an extended version, that is with an added parity check co-ordinate making the weight of every word even, with parameters

$$G_{24} = [24, 12, 8]_2$$

and

$$G_{12} = [12, 6, 6]_3$$

and although the extended versions are not perfect they are very interesting and are useful in the construction of the perfect Golay codes. We shall only be displaying the Golay codes and suggest the interested reader refer to [17], [36] or [9] for further details and proofs of relevant statements.

Definition 1.35. The code G_{23} is generated by a generating matrix over $\mathbb{F}_2$ in the form

$$[I_{12}|A]$$

where A is a 12×11 matrix of the form

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

We note that $G_{23} = [23, 12, 7]_2$ satisfies

$$2^{12}\left(1 + \binom{23}{1} + \binom{23}{2} + \binom{23}{3}\right) = 2^{12}\Big(1 + 23 + 253 + 1771\Big) = 2^{12}(2048) = 2^{23}$$

which are the conditions to be a perfect code and as such G_{23} is perfect. Note that we have left the proof that G_{23} has minimum distance of 7 due to their lack of necessity for our following work. We note however that the relevant calculation would be to check the linear dependance of the columns.

Definition 1.36. The code G_{11} is generated by a generating matrix over $\mathbb{F}_3$ in the form

$$[I_6|A]$$

where A is a 6×5 matrix of the form

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 2 & 1 \\ 1 & 0 & 1 & 2 & 2 \\ 2 & 1 & 0 & 1 & 2 \\ 2 & 2 & 1 & 0 & 1 \\ 1 & 2 & 2 & 1 & 0 \end{pmatrix}$$

We note that $G_{11} = [11, 6, 5]_3$ satisfies

$$3^6\left(1 + \binom{11}{1}(3-1) + \binom{11}{2}(3-1)^2\right) = 3^6\Big(1 + 22 + 220\Big) = 3^6 3^5 = 3^{11}$$

which are the conditions to be a perfect code and as such $G_{11} = [11, 6, 5]_3$ is perfect. Note we have left the proof out of the minimum distance and this can be checked by checking the linear dependance of the columns in A.

There are many papers building to a proof of the non-existence of codes with different parameters, including [34], [33] and culminating in [35]. It has been shown in [13] and [28] that there exist non-linear codes with the same parameters as the Hamming codes. Moreover it can be shown that any perfect codes will have the parameters of a Hamming or are one of the Golay codes [35].

This work culminates in the following Theorem, which we will omit the proof of.

Theorem 1.37. *A nontrivial perfect code over an alphabet of prime power must either have the parameters of a Hamming code, or be a Golay code.*

There are no known examples of perfect codes over non-prime power alphabets, moreover it is known that Lloyds Theorem holds for non-prime power alphabets and that certain classes of codes do not exist over non-prime power alphabets, namely group codes [21].

1.7 Concluding

In this Chapter we have introduced the problem of transmission of messages over a noisy channel and have shown one of the most effective methods of battling this problem, block codes. We have given the basic definitions and properties of block codes, shown some of their limitations and have exhibited some of the properties of the highly practical and also theoretical interesting perfect codes. We have only scratched the surface of the interesting and wonderful properties of many block codes, and mean this chapter as a repository of only the basic preliminary information.

Other wonderful aspects of coding theory that can be investigated include higher weights [32], which is a generalisation of the weight of a codeword to that of the weight of a general linear subspace and thus questions of the type "what is the minimum distance/ minimum weight of a code" can be more generally tackled as questions of the type "what is the minimum weight of a subspace of dimension i in a code" this can lead to some interesting answers. Other further directions include the tie in between other combinatorial objects and that of codes, this can be particularly fruitful in the search for 'good' parity check matrices and the work of Goppa [6] ties in the search for, and maximum size of, arcs from projective geometry with MDS codes, that is codes which meet the Singleton bound from Theorem 1.29.

In the following Chapters we hope to shine a light on a different approach at looking at codes, to establish other properties that they can have, to examine these thoroughly and then to attempt to integrate this back into an approach with the view of error correction in mind.

I never saw an ugly thing in my life: for let the form of an object be what it may, light, shade and perspective will always make it beautiful

John Constable

Chapter 2

Focused Splittings

2.1 Overview

In this Chapter we introduce the concept of *perspective*, which is a generalisation on the Hamming distance, and show that it leads to a natural partition of the codewords into distinct codes. We shall then identify which of these partitions we consider to be nice, these are the *focused splittings*, and show that they are a natural choice in two ways, firstly for their appearance at the extremal situations of perspective as well as their relationship to perfect codes. We follow this up by examining the existence of focused splittings for various parameters as well as giving a couple of specific constructions for focused splittings. Furthermore we look at design theory and show a relationship between focused splittings and certain forms of designs.

2.2 Perspective and Focused Splittings

Definition 2.1. A code with *perspective* $(\theta, \delta) \in \mathbb{N}^2$ is a set $C \subseteq (\mathbb{F}_q)^n$ such that

$$|\{b \in C : d(b, c) < \delta\}| \leq \theta$$

for all $c \in C$, where we generally assume δ is the largest such integer satisfying the relation for a given θ, and that there exists some element $c \in C$ such that

$$|\{b \in C : d(b, c) < \delta\}| = \theta$$

We call θ the *width* of the perspective and δ the *distance*.

Thus, for a width of 1, the distance of the perspective is the minimum distance of the code. As such, perspective presents a way of generalising the concept of minimum distance.

Definition 2.2. A code has a *balanced perspective* (θ, δ) if and only if

$$|\{b \in C : d(b, c) < \delta\}| = \theta \text{ for all } c \in C.$$

A *balanced code* is a code which admits a balanced perspective for some value of δ.

Notation 2.3. The set $\{b \in C : d(b, c) < \delta\}$ will be referred to as the *society* of c and denoted $S(c)$. We shall use the word *neighbour* to express when two words in a code have a distance less than δ, that is when two words are in the same society.

Theorem 2.4. *Linear codes are always balanced.*

Proof Let C be a linear code. To show C is balanced, we need to prove that there are the same number of elements in the societies of b and c, that is $|S(b)| = |S(c)|$ for all b, c in C. We do this by constructing a one-to-one correspondence between elements of $S(b)$ and $S(c)$. Let $d \in S(c)$, this happens when $d(c, d) < \delta$. As C is linear, this means there exists $e \in C$ such that $c + e = d$; so $d(c, c + e) < \delta$, and so $d(b, b + e) < \delta$. Now let $d' = b + e$. So for every $d \in S(c)$ there is a unique $d' \in S(b)$, and as c and b are arbitrary, thus $|S(c)| = |S(b)|$ for all b, c in C as required. □

Now we show that this idea of a perspective of a code leads to a natural partition of the code into θ distinct codes, each with a minimum distance of at least δ.

Theorem 2.5. *A code C with perspective (θ, δ) can be partitioned into θ codes, not necessarily non empty, each with a minimum distance of at least δ.*

Proof We prove this by induction on θ.

Base Case $\theta = 1$

The code is already partitioned into 1 code and this code has a minimum distance of at least δ by the definition of perspective.

Inductive Step

Assume that "A code C with perspective (θ, δ) can be partitioned into θ codes each with a minimum distance of at least δ" is true for $\theta = i - 1$, that is, $D_1, D_2, \ldots, D_{i-1}$ can be constructed. For the case of $\theta = i$, we select our first split code D_i as follows:

(1) set $D = C$ and D_i as empty;

(2) select an element $c \in D$ and put $c \in D_i$;

(3) set $D = D \backslash S(c)$.

Repeat steps (2) and (3) until D is empty.

As C is finite this process must terminate. As D is empty at the end of the process, every element of C must appear in one of the societies of an element in D_i as these will be the only words that are ever removed.

Now we claim that $C \backslash D_i$ is a code with perspective $(i - 1, \delta)$.

In C each word has at most i words at a distance $< \delta$, but as each word in C belongs to the society of a word in D_i, one of these words must belong to D_i. So each word in C has at most $i - 1$ words at a distance $< \delta$ inside $C \backslash D_i$, and thus each word in $C \backslash D_i$ has at most $i - 1$ words at a distance $< \delta$ inside $C \backslash D_i$. Thus $C \backslash D_i$ is a code with perspective (θ', δ) where $\theta' \leq i - 1$. As we have $\theta' \leq i - 1$ we know by our definition of perspective that

$$|\{b \in C \backslash D_i : d(b, c) < \delta\}| \ \leq \theta' \leq i - 1$$

and we can thus consider $C \backslash D_i$ as a code with perspective $(i-1, \delta)$, and so by the inductive hypothesis can be split into $i-1$ codes, not necessarily non-empty, $D_1, D_2, \ldots, D_{i-1}$ each with a minimum distance of at least δ. Adding D_i to this we get the splitting for C. As we artificially extended the perspective of $C \backslash D_i$ we note that the codes are not necessarily non-empty, we demonstrate this by example of a code with a single word, thus the code has perspective of $(1, \delta)$ and we could thus consider it as a code with perspective of $(2, \delta)$ but by virtue of their being just one word we know we can not partition it into two non-empty codes each with a minimum distance of at least δ. □

Definition 2.6. A *splitting* of a code C with perspective (θ, δ) is a partition into θ codes, not necessarily non-empty, each with a minimum distance of at least δ. We shall refer to an individual code in the splitting as a split code.

Definition 2.7. A code has a *focused splitting* if and only if given $c \in C$ and D_i a split code, then $|S(c) \cap D_i| \leq 1$.

Also note that if a code is balanced then this is equivalent to $|S(c) \cap D_i| = 1$.

As this definition requires every word in a code to have a particular property, we can talk about particular words having that property, we shall say a code is "focused around a word" if a particular word does have that property and "not focused around a word" if it does not. We shall also speak of a "word being focused" where we mean that the code is focused around that word, and similarly with a "word being not focused". It is clear that to show that a splitting is not focused, it will be enough to show that a particular (or arbitrary) word is not focused within it and that to show a splitting is focused we will have to show that all words are focused.

2.3 Structure of Focused Splittings

Lemma 2.8. *Let C be a code with a balanced perspective of (θ, δ) having a focused splitting of $D_1, D_2, \ldots, D_\theta$ and let M be the number of words in C. Then $|D_i| = |D_j|$ for all i and*

j, and $|D_i| \times \ \theta = M$. Thus $\theta \mid M$.

Proof Let C be a code with a balanced perspective of (θ, δ) having a focused splitting of $D_1, D_2, \ldots, D_\theta$. As C has a balanced perspective, $c \in C$ will have exactly θ neighbours in C and as the splitting is focused at most 1 of these is in each D_i. As there are θ split codes there must be precisely 1 neighbour of c per D_i. Now we need to show that for different words in the same split code they will have distinct neighbours in another split code. If $b, c \in D_i$ both have the neighbour $d \in D_j$ then the splitting of C would not be focused, because d would have more than 1 neighbour in D_i. Thus for each element in D_i there is 1 in D_j thus $|D_i| \leq |D_j|$, but as D_i and D_j are arbitrary this means that $|D_i| = |D_j|$ for all i, j. As there are θ split codes all together and they partition C this means there must be $\theta \ \times |D_i|$ words in C. Thus $\theta \mid M$. □

Lemma 2.9. *Let C be a linear $[n, k, d]_q$ code, with a perspective of (θ, δ). Then $(q-1) \mid (\theta - 1)$*

Proof As $\mathbb{F}_q$ is a field thus for all $i, j \in \mathbb{F}_q \setminus \{0\}$ we get $ij \neq 0$. Also note by linearity, all words in $(\mathbb{F}_q)^n$ will have the same size society, so without loss of generality we shall deal with $S(0)$.

A word c is in $S(0) \setminus \{0\}$ if and only if $0 < w(c) < \delta$. Now we note that the words ic for $i \in \mathbb{F}_q \setminus \{0\}$ will all distinct and as c has at least one non-zero entry k then in the words ic for $i \in \mathbb{F}_q \setminus \{0\}$ there will be non-zero entries of ik. Thus note that the weights of the words ic for $i \in \mathbb{F}_q \setminus \{0\}$ will all be the same as the weight of the word c and thus they will belong to $S(0)$. So for each words $c \in S(0) \setminus \{0\}$ there will be $q-2$ other words ic for $i \in \mathbb{F}_q \setminus \{0, 1\}$ in $S(0) \setminus \{0\}$.

Also note as $\mathbb{F}_q$ is a field, for any word $d = j \times c$, where $j \in \mathbb{F}_q \setminus \{0\}$, the set of words $\{di : i \in \mathbb{F}_q \setminus \{0\}\} = \{cji : i \in \mathbb{F}_q \setminus \{0\}\} = \{ci : i \in \mathbb{F}_q \setminus \{0\}\}$ and thus the multiples of a word c in $S(0) \setminus \{0\}$ will produce distinct classes of words.

Thus $\theta - 1 = |S(0) \setminus \{0\}| = (q-1) \times \phi$ where ϕ is the number of distinct class of multiples of words. Thus $q - 1 \mid \theta - 1$. □

Now in the same way that the sphere packing bound gives a bound on the number of words in a code, we can use the same machinery to give a bound on the value of θ in relation to the dimension of the code k.

Theorem 2.10. *Given a linear code with dimension k and perspective (θ, δ) then*

$$\theta \leq \sum_{i=0}^{\delta-1} \binom{k}{i} (q-1)^i.$$

Proof Given that the code has dimension k, then by choosing k independent co-ordinates of the n co-ordinates of any word, the other $n-k$ co-ordinates will be fixed by the choice of the first k. In a k-dimensional space, the number of words in a sphere of size $\delta - 1$ is

$$\sum_{i=0}^{\delta-1} \binom{k}{i} (q-1)^i$$

that is, at a distance of i there will be $\binom{k}{i}$ ways to choose i co-ordinates to change and $(q-1)^i$ ways in which to change them.

Because our k-dimensional space is a subspace of an n-dimensional space, not all the words in the sphere will be of Hamming distance $< \delta$ from the centre word, as the $n-k$ other co-ordinates may contribute to the distance. Noting that any words not counted in the sphere, will have a distance of at least δ from the k co-ordinates, thus we get

$$\theta = |\{c \in A : d(a,c) < \delta\}| \leq \sum_{i=0}^{\delta-1} \binom{k}{i} (q-1)^i.$$

□

We now look at what form of regularity a focused splitting can have.

Theorem 2.11. *Let C be a $[n,k,d]_q$ code with perspective (θ, δ) and a focused splitting $D_1, D_2, \ldots, D_\theta$.*

(i) *Let $S(0)$ be labelled $\{e_1, e_2, \ldots, e_\theta\}$.*

(ii) *Fix* $j \in [1, \theta]$.

(iii) *Define* $E_i = D_j + e_i = \{(c_t + e_i) : c_t \in D_j\}$.

Then $E_1, E_2, \ldots, E_\theta$ *is a focused splitting for* C.

Proof Firstly we count the elements in $E_1, E_2, \ldots, E_\theta$. As the size of E_i is irrespective of i thus no word will get counted twice. If a word did get counted twice then some word is in E_i and E_s for some i and s and so $c_t + e_i = c_v + e_s$ for some t and v with $c_t, c_v \in D_j$. Subsequently the distance from the word $c_v + e_s$ to both c_t and c_v is less than δ and so $c_t, c_v \in (S(c_v + e_s) \bigcap D_j)$. This implies that $|S(c_v + e_s) \bigcap D_j| \geq 2$, which is a contradiction to the fact that D_j is a split code from a focused splitting. Thus we can see that

$$\theta \times |E_i| = \theta \times |D_j| = \theta \times \frac{M}{\theta} = M.$$

Secondly each E_i has a minimum distance $> \delta$ as D_j does.

Thirdly $|E_i \bigcap S(c)| = 1$ for all $i \in [1, \theta]$ and for all $c \in C$. If this was not the case then there would exist $b, d \in E_i$, with $b \neq d$ such that $b, d \in S(c)$ and thus $(b - e_i), (d - e_i) \in D_j$ and $(b - e_i), (d - e_i) \in S(c - e_i)$. This implies $|S(c - e_i) \bigcap D_j| \geq 2$, which is a contradiction to the fact that D_j is a split code from a focused splitting.

Thus $E_1, E_2, \ldots, E_\theta$ is a focused splitting for C. □

Note that we shall refer to this as the linearity of focused splittings, because it shows that there is always a focused splitting where every split code has exactly the same structure.

2.4 Which Codes have a Focused Splitting

In this section we investigate when a code has a focused splitting. We start by looking at the case where the code is in fact the whole space, and we get the lovely result that these are precisely the perfect codes in that space.

Theorem 2.12. *Let $C = (\mathbb{F}_q)^n$ with perspective (θ, δ) and a focused splitting $D_1, D_2, \ldots, D_\theta$. Then D_i is a perfect code for all i.*

Proof As C is balanced and focused, thus $|D_i| = |D_j|$ for all i, j and $\theta \mid q^n$.

Choose D_i and count the words in the societies around the words of D_i. No word will get counted twice, because if it did that word would appear in 2 societies around words in D_i, and thus two words of D_i would appear in that word's society, which would mean that the code would not be focused. Now there are $|D_i|$ words, each with a society of $|S(c)|$ and we know that

$$|D_i| \times |S(c)| = |D_i| \times \theta = M = q^n$$

and thus the spheres of size δ around the words of D_i exactly fill the space $(\mathbb{F}_q)^n$; thus D_i is a perfect code. □

Theorem 2.13. *A perfect code P over $(\mathbb{F}_q)^n$ induces a focused splitting of $(\mathbb{F}_q)^n$.*

Proof Let P be a perfect code with minimum distance d and let e be such that $2e+1 = d$. As the code P is perfect, spheres around the words of size e will fill the space. Let T be all the words in $(\mathbb{F}_q)^n$ of weight at most e and label these words $e_1, e_2, \ldots, e_\phi$, and set $D_i = \{p + e_i : p \in P\}$. Now to prove that $D_1, D_2, \ldots, D_\phi$ is a focused splitting we must show that

(1) it partitions the space $(\mathbb{F}_q)^n$;

(2) each D_i has a minimum distance of greater than e, and thus is a splitting of $(\mathbb{F}_q)^n$;

(3) it is focused; that is, that given $c \in (\mathbb{F}_q)^n$ and D_i, then $|S(c) \cap D_i| \leq 1$.

We show each of these in the following ways.

(1) There are ϕ words of weight at most e and so there are ϕ words in each sphere around a word in P. Thus there are a total of $\phi \times |P|$ words in $(\mathbb{F}_q)^n$. Now there are $|P|$

words in each D_i and so there are $\phi \times |P|$ in $D_1, D_2, \ldots, D_\phi$ and so $D_1, D_2, \ldots, D_\phi$ partitions the space.

(2) P has a minimum distance greater than e, and so any translation of P is going to have minimum distance greater than e.

(3) If $|S(c) \cap D_i| \geq 2$ for some c and i, then there exists $b_1, b_2 \in D_i$ such that $b_1 \in S(c)$ and $b_2 \in S(c)$ which implies $c \in S(b_1)$ and $c \in S(b_2)$. Noting that $b_1 = p_1 + e_i$ and $b_2 = p_2 + e_i$ for appropriately labelled p_1 and $p_2 \in P$, then $S(b_1) \cap S(b_2) \neq \emptyset$ which implies $S(p_1 + e_i) \cap S(p_2 + e_i) \neq \emptyset$ which implies $S(p_1) \cap S(p_2) \neq \emptyset$, which contradicts the fact that P is a perfect code.

Therefore $D_1, D_2, \ldots, D_\phi$ is a focused splitting. □

Theorem 2.14. *All splittings for codes with a perspective of $\theta = 1$ are focused.*

Proof A code with perspective $\theta = 1$ will split into just one code with a minimum distance of δ and so will automatically be focused. □

Theorem 2.15. *All splittings for codes with a perspective of $\theta = 2$ are focused.*

Proof For a proof by contradiction, assume that C with perspective $(2, \delta)$ has an unfocused splitting into two codes D_1 and D_2, that is $|S(c) \bigcap D_i| \geq 2$ for some $c \in C$ and $i \in \{1, 2\}$. As $|S(c)| \leq 2$ thus for $|S(c) \bigcap D_i| \geq 2$ we have that $S(c) \subseteq D_i$, but as D_i has a minimum distance $\geq \delta$ and $S(c)$ consists of 2 words with a distance $< \delta$ we can see that this causes a contradiction. Thus the splitting is focused. □

With $\theta = 3$ there are codes which have no focused splitting, and codes which have both focused and unfocused splittings. There are also conditions on whether a code has a focused splitting. We shall start by introducing an idea which will help us reduce the number of cases we need to look at.

Definition 2.16. The *component* of a code containing $c \in C$ is the set of all words that can be reached by a finite number of steps from one word to its neighbour starting at c. We shall consider a component to have the same perspective as the whole code it originates from, even if when the component is considered separately it could have a lower value of θ.

This is an equivalent idea to that of components in graphs and as such we can use it to reduce problems to the case of codes made of single components, using the following lemma.

Lemma 2.17. *A code C has a focused splitting if and only if every component of the code has a focused splitting. Moreover we show how to construct one from the other.*

Proof To prove the forward implication. Let a code C have a focused splitting $D_1, D_2, \ldots D_\theta$ such that $|S(c) \bigcap D_i| \leq 1$. Let C' be an arbitrary component of the code, and let $D'_i = D_i \cap C'$. Then we claim that $D'_1, D'_2, ...D'_\theta$ is a focused splitting of C'.

Then $\bigcup \{D'_i\}_{i=1}^{\theta} = \bigcup \{D_i \cap C'\}_{i=1}^{\theta} = (\bigcup \{D_i\}_{i=1}^{\theta}) \bigcap C' = C \cap C' = C'$ and so $D'_1, D'_2, \ldots, D'_\theta$ partitions the component, and also each D'_i will inherit a minimum distance $\geq \delta$ and so $D'_1, D'_2, \ldots, D'_\theta$ is a splitting of C'. To show that the splitting is focused we must show that each word has at most 1 neighbour in each D'_i. In C' each word has exactly the same neighbours that it did in C, and as $D'_i \subseteq D_i$, thus each $c \in C'$ will have at most one neighbour per D'_i because in C it had at most one neighbour per D_i, and it can not have any more. Thus each component has a focused splitting.

To prove the the reverse implication, consider a code C, which has components

$$C', C'', \ldots, C^{(\alpha)}$$

which each have focused splittings

$$(D'_1, D'_2, \ldots, D'_\theta), (D''_1, D''_2, \ldots, D''_\theta), \ldots, (D_1^{(\alpha)}, D_2^{(\alpha)}, \ldots, D_\theta^{(\alpha)})$$

respectively. Then we can construct a focused splitting for C as follows:

Set $D_i = \bigcup\{D_i^{(j)}\}_{j=1}^{\alpha}$ for all $i \in [1, \theta]$. Then we claim that $D_1, D_2, \ldots, D_\theta$ is a focused splitting for C. First note that

$$\bigcup\{D_i\}_{i=1}^{\theta} = \bigcup\{\bigcup\{D_i^{(j)}\}_{j=1}^{\alpha}\}_{i=1}^{\theta} = \bigcup\{\bigcup\{D_i^{(j)}\}_{i=1}^{\theta}\}_{j=1}^{\alpha} = \bigcup\{C^{(j)}\}_{j=1}^{\alpha} = C.$$

So $D_1, D_2, \ldots, D_\theta$ is a partition of C, and as each of the $C_i^{(j)}$ was a splitting for each of the components each $C_i^{(j)}$ has a minimum distance $\geq \delta$, and as words in different components must be at least δ distance apart by virtue of being in separate components, then the words inside D_i must be at least δ distance apart; thus $D_1, D_2, \ldots, D_\theta$ is a splitting of C. To show that it is focused, we take an arbitrary word and an arbitrary split code and show that the word has at most 1 neighbour in the split code. Let $c \in C$ and let D_i be a split code of c. Without loss of generality say that $c \in C' \subseteq C$. As all of the neighbours of c are in C' we know that if c has a neighbour in D_i it must also be in $D_i \cap C' = D_i'$. As $D_1', D_2', \ldots, D_\theta'$ is a focused splitting of C' we know that there is at most 1 neighbour of c in D_i' and thus at most 1 neighbour of c in D_i □

Lemma 2.18. *A code C with a balanced perspective with $\theta = 3$ made up of a single component has a focused splitting if and only if the size of the code $M \equiv 0$ mod 3*

Proof The forward implication is covered by Lemma 2.8. To prove the reverse implication, if $M \equiv 0 \bmod 3$ and C is made of a single component, then the words of C can be labelled as follows.

Choose an arbitrary word, label it 1. As there are 3 neighbouring words, one of which is itself, there are 2 other neighbours, arbitrarily label these 0 and 2. Take the word labelled 2, it will have one unlabelled neighbour, label this 3, label 3's unlabelled neighbour 4 and so on. As there are only a finite number of words this will eventually terminate, and as C is a single component thus every word of C will have been labelled. The words of C will be labelled from 0 to $M-1$, and as $M \equiv 0 \bmod 3$ then $M - 1 \equiv 2 \bmod 3$. Partition

the words into equivalence classes mod 3, giving $\theta = 3$ codes. This gives a splitting of C as there are $\theta = 3$ codes and in each code, the words have minimum distance $\geq \delta$, as otherwise they would be neighbours, and no neighbours are in the same code by the definition of how they were partitioned. To show the splitting is focused, consider a word labelled i, it will have two neighbours, $i-1$ and $i+1$, and these 3 words will be in separate equivalence classes mod 3, and thus the splitting is focused. □

If a code C, consisting of a single component, has an unbalanced perspective with $\theta = 3$, then there must be at least one word with either 1 or 2 neighbours. In the first case, then that word is its only neighbour, and thus that word is the entire component, and any splitting of such is automatically focused.

Lemma 2.19. *If a code C, consisting of a single component with an unbalanced perspective of $\theta = 3$ and has 1 word with precisely 2 neighbours (including itself) then it must have exactly 2 such words.*

Proof Let there be M words in A.

Take the first word with only two neighbours and label it 0, and its neighbour 1. If 1 has only two neighbours then we have proved the lemma as 0 and 1 will form the component. Otherwise label 1's unlabelled neighbour 2, label 2's unlabelled neighbour 3 and so on.

As M is finite this must terminate. No word can have only 1 neighbour because it would form its own component, so each word must have 2 or 3 neighbours. As the labelling must terminate, the final word labelled $M-1$ must have 2 neighbours, for if it had 3 then one would be $M-2$, and there would not be an unlabelled word to label M, as any labelled word already had all its neighbours accounted for. Any word before $M-1$, except 0, can not have exactly 2 neighbours, as that would mean that the words 0 up to that word would form a component. Thus 0 and $M-1$ are the only words with exactly 2 neighbours and all other words have 3 neighbours. □

Corollary 2.20. *Any code C, made of a single component with an unbalanced perspective of $\theta = 3$, has a focused splitting.*

Proof There are two cases.

(i) If there is a word with exactly one neighbour, then that word forms its entire component and is thus automatically focused.

(ii) If there is a word with exactly two neighbours, thus by Lemma 2.19 there are precisely 2 such words. Label the words, as in Lemma 2.19, 0 to $M-1$. Partition them according to equivalence classes mod 3. This will give $\theta = 3$ codes. For any word i, not 0 or $M-1$, i has two other neighbours $i-1$ and $i+1$ and these are in separate equivalence classes and thus separate codes. As no two neighbours are in the same code, the words in one code have to be at least δ apart; otherwise they would be neighbours. The word 0 will have one other neighbour, the word 1, and this will be in a separate code, similarly the word $M-1$, will have one other neighbour, the word $M-2$, which will be in a separate code. Thus the splitting is focused.

□

Theorem 2.21. *A Code C with perspective $\theta = 3$ has a focused splitting if and only if each component C' of C has one of the following properties:*

(i) *an unbalanced perspective of $\theta = 3$;*

(ii) *a balanced perspective of $\theta = 3$ with $M' \equiv 0 \ mod\ 3$ words, where M' is the cardinality of C'.*

Proof By Lemma 2.17, a code C has a focused splitting if and only if each component has a focused splitting. If a component C' has an unbalanced perspective of $\theta = 3$ then by Corollary 2.20 that the component will have a focused splitting. Also if C' has a balanced

perspective of $\theta = 3$ we have seen in Lemma 2.18 that it has a focused splitting if and only if it has $M' \equiv 0 \bmod 3$, where M' is the cardinality of C'. □

Now we consider higher values of θ.

Theorem 2.22. *A code C with a perspective with width of $\theta = M$ will always have a focused splitting. Moreover there is only one splitting of a code C with perspective $\theta = M$.*

Proof Let C be a code with perspective $\theta = M$, then a splitting of this code will produced M codes, each with only one word in it. So the splitting is focused as each word will have only up to one neighbour per code, as there is only up to one word per code. The splitting is unique as every word has to go into a separate way, and this can only be done one way. □

Theorem 2.23. *A code C where $M/2 < S(c) < M$, for all $c \in C$ has no focused splittings.*

Proof Proof by contradiction. Suppose that there was a focused splitting for C. As $\theta < M$ the code would split into less than M codes, and as there are M words, by the pigeon hole principle there will be at least one split code with at least 2 words in it. Let b, c be two words in the same split code. As $S(c) > M/2$ for all $c \in C$ so $|S(b) \cap S(c)| \geq 1$, thus there exists a word d which is neighbours with both b and c, and thus the code would not be focused around that word. □

Corollary 2.24. *All codes with a balanced perspective of $M/2 < \theta < M$ have no focused splittings.*

Proof For a code with a balanced perspective, $S(c) = \theta$, for all $c \in C$ and by Theorem 2.23 it will have no focused splittings. □

Theorem 2.25. *Let $C = [n, k, d]_2$ be a binary linear code, let n be odd, and let $1^n \in C$, where $1^n = (1, 1, \ldots, 1) \in (\mathbb{F}_2)^n$. Then C has a focused splitting for $\theta = 2^{k-1}$ and $\delta = \lceil \frac{n}{2} \rceil$.*

Proof As n is odd, $\frac{n}{2}$ is not an integer and so all words in C have weight either strictly less than or strictly more than $\frac{n}{2}$.

As $1^n \in C$ and C is linear, then for each $c \in C$ with $w(c) < \frac{n}{2}$ there is a word $b \in C$ with $b = 1^n + c$ and thus $w(b) > \frac{n}{2}$ and similarly for $w(c) > \frac{n}{2}$ we have $w(b) < \frac{n}{2}$. Thus $|\{c \in C : w(c) < \frac{n}{2}\}| = 2^{k-1}$.

We label $\{c \in C : w(c) < \frac{n}{2}\} = \{c_1, c_2, \ldots, c_\theta\}$ with $c_1 = 0^n$, we set $D_i = \{c_i, 1^n + c_i\}$, and we claim that $D_1, D_2, \ldots, D_\theta$ is a focused splitting for C.

(i) $D_1, D_2, \ldots, D_\theta$ is a partition of C as there are $\theta = 2^{k-1}$ codes and each contains 2 words, so a total of 2^k words.

(ii) Each split code has a minimum distance of $n > \lceil \frac{n}{2} \rceil$ and so the form a splitting of C.

(iii) If $|D_i \cap S(c)| \geq 2$ then $\exists b, e \in D_i$ such that $b, e \in S(c)$ Thus $\{b, e\} = D_i$ and so $e = 1^n + b$, thus $d(b, e) = n$. Now as $b, e \in S(c)$ then $d(b, c) < \frac{n}{2}$ and $d(b, e) < \frac{n}{2}$, so by the triangle inequality $d(b, e) \leq d(b, c) + d(c, e) < \frac{n}{2} + \frac{n}{2} = n$ which contradicts that $d(b, e) = n$, thus $|D_i \cap S(c)| \leq 1$ and thus the splitting is focused.

□

Theorem 2.26. *Let $C = [n, k, d]_2$ be a linear code with perspective (θ, δ) and with the set $S(0) \setminus \{0\}$ linearly independent. Then there exists a focused splitting on C if and only if there exists a perfect single-error correcting code on $(\mathbb{F}_2)^{\theta-1}$.*

Proof First note, that as C is a linear code, and $S(0) \setminus \{0\}$ is linearly independent then $|S(0) \setminus \{0\}| = \theta - 1 \leq k$, and thus we can choose a set G such that $S(0) \setminus \{0\} \subseteq G$ and $\langle G \rangle = C$, that is G forms a generator matrix for C, and so every word in C can be uniquely represented as the sum of words in G.

Labelling the elements of G, $g_1, g_2, \ldots, g_k$ such that $g_1, g_2, \ldots, g_{\theta-1}$ are the elements in $S(0) \setminus \{0\}$, thus $g_1, g_2, \ldots, g_{\theta-1}$ are the only non-zero words of weight $< \delta$.

We can represent each word in C uniquely as the sum of the words in the generator matrix, that is for $c \in C$ we can write $c = \sum_{i=1}^{k} \alpha_i g_i$ where $\alpha_i \in \mathbb{F}_2$. Thus we can represent each word in C as $(\alpha_1, \alpha_2, \ldots, \alpha_k)$ where α_i is the coefficient of g_i. Note that each combination of different values of α_i will appear as some word in C, and as there are 2^k words in C and 2^k different combinations of $(\alpha_1, \alpha_2, \ldots, \alpha_k)$ this will be unique .

Now if two words b and c in C differ in any of $(\alpha_\theta \ldots, \alpha_k)$ then they are not neighbours in C as each of $g_\theta \ldots, g_k$ has weight $\geq \delta$, and as G is linearly independent $g_1 \ldots, g_{\theta-1}$ are the only words of weight $< \delta$.

If two words b and c have 1 difference in $\alpha_1 \ldots, \alpha_{\theta-1}$ and no differences in $\alpha_\theta \ldots, \alpha_k$ then they will be neighbours in C.

If two words b and c have more than 1 difference in $\alpha_1 \ldots, \alpha_{\theta-1}$ then they will not be neighbours in C as $\alpha_1 \ldots, \alpha_{\theta-1}$ is linearly independent and consists of all words of weight $< \delta$ thus the difference between b and c will be $\geq \delta$.

Now if there was a focused splitting on C and we restrict all words in C to $(\alpha_1 \ldots, \alpha_{\theta-1})$ then there are $2^{\theta-1}$ different words, each equivalent to one in $(\mathbb{F}_2)^{\theta-1}$. Each word in $(\mathbb{F}_2)^{\theta-1}$ has $2^{k-\theta+1}$ words in C mapped to it under the restriction, that is 1 for each of the variations of $(\alpha_\theta, \ldots, \alpha_k)$.

As there is a focused splitting on C the words are all partitioned so that no two neighbours are in the same split code, that is, no two words differing by one value in $\alpha_1, \ldots, \alpha_{\theta-1}$. So taking one of the split codes, say D_1, representing it as $(\alpha_1, \alpha_2, \ldots, \alpha_k)$ and restricting it to $(\mathbb{F}_2)^{\theta-1}$, we see that for each word in D_1 the sphere of distance 1 around a word is empty, as the words of distance 1 in $(\mathbb{F}_2)^{\theta-1}$ are the neighbours of the word in C. We can also see that every word in $(\mathbb{F}_2)^{\theta-1}$ appears in one of these spheres of distance 1 as for any $c \in C$ we have $|S(c) \bigcap D_1| = 1$ and as each c represents some word in $(\mathbb{F}_2)^{\theta-1}$ and $S(c)$ consists of words which are of distance 1 from c when considered as

words in $(\mathbb{F}_2)^{\theta-1}$. Thus this forms a perfect single error correcting code on $(\mathbb{F}_2)^{\theta-1}$.

Now if there is a perfect single-error correcting code P on $(\mathbb{F}_2)^{\theta-1}$, and we can assume that P contain the zero word $(0, 0, \ldots, 0)$, then set $\hat{D}_1$ to be P translated to $(\alpha_1, \ldots, \alpha_{\theta-1})$ and set D_1 to be $\hat{D}_1$ extended to $(\alpha_1, \ldots, \alpha_k)$ with all possible combinations of $(\alpha_\theta, \ldots, \alpha_k)$.

Now setting $D_2, \ldots, D_\theta$ to be D_1 translated by a word of weight 1 in $(\mathbb{F}_2)^k$, that is by a word $(0, \ldots, 0, \alpha_i, 0, \ldots, 0)$ for $i \leq \theta - 1$ and α_i in the i-th position.

We claim $D_1, D_2, \ldots, D_\theta$ forms a focused splitting for C.

(i) Each word in C is represented by a unique word in $(\alpha_1, \ldots, \alpha_k)$ and all such words represent some word in C. As P was perfect, it had no words in spheres of size 1 around words of P and so when extended to $(\alpha_1, \ldots, \alpha_k)$ there were no words which differed by 1 in the first $\theta - 1$ positions. So by translating D_1 by words of weight 1 where the 1 appears in the first $\theta - 1$ positions then there would be no duplicated words. Counting the number of words in $D_1, D_2, \ldots, D_\theta$, we get

$$\begin{aligned}
\theta \times |D_1| &= \theta \times |P| \times (2^{k-\theta+1}) \\
&= \theta \times \frac{2^{\theta-1}}{1 + (\theta-1)(2-1)} \times (2^{k-\theta+1}) \\
&= \theta \times \frac{2^{\theta-1+k-\theta+1}}{1 + (\theta-1)} \\
&= \theta \times \frac{2^k}{\theta} \\
&= 2^k = |C|.
\end{aligned}$$

Thus $D_1, D_2, \ldots, D_\theta$ is a partition of C.

(ii) As $D_1, D_2, \ldots, D_\theta$ is a partition of C and D_i's are all translates of D_1. Consider $S(c) \cap D_1$ for some $c \in C$. Now c belongs to one split code, say D_i, and as D_i is a translate of D_1 then for some word $e \in C$ with weight $< \delta$, then $D_i = D_1 + e$; so there exists a word $b \in D_1$ such that $c = b + e$ thus $b \in S(c)$ thus $|S(c) \cap D_1| \geq 1$. If $|S(c) \cap D_1| \geq 2$ then there exists $b, f \in D_1$ such that $b, f \in S(c)$ but if $b, f \in S(c)$

then b and f differ in the first $\theta - 1$ positions when represented as $(\alpha_1, \ldots, \alpha_k)$, and thus b and f differ when restricted to $(\alpha_1, \ldots, \alpha_{\theta-1})$ and so are different words in P. But in P each word has non-intersecting spheres of size 1 but b and f would both have the restriction of c in their spheres of size 1. Thus we reach a contradiction and so $|S(c) \cap D_1| = 1$.

Thus $D_1, D_2, \ldots, D_\theta$ forms a focused splitting for C. □

2.5 Applications to Design Theory

In the following we shall take a side step into design theory. Design theory is, initially at least, the study of how to construct (or how to design) efficient experiments. The parable goes that a company had seven different types of coffee they wished to compare, presumably they were going to invest money into the winning coffee type or some other such prize. They were not going to trust one person's opinion so they knew they would need several volunteers to drink the coffee, but it soon became clear that if they made each person drink seven cups of coffee and then asked them to rate them, all that would happen is they would have an extremely hyperactive person, possibly on the brink of having a coronary since the coffee was reportedly very strong, who could not really tell the differences between the coffees because they had all started to taste the same after a few. So it was decided that each person should only drink a few cups of coffee, and to make this judgement fair it was also noted that each coffee should be compared to every other coffee and that each coffee should only be tested equally often so that no type of coffee gets an advantage by being tested more than any other. A couple of solutions were proposed, each pair could be tested by a different person, and this would lead to 21 different people being required or they could be arranged in triples, with each person testing a 3 different coffees and comparing them, the triples being arranged as such

$$\{1, 2, 3\}, \quad \{1, 4, 5\}, \quad \{1, 6, 7\},$$

$$\{2,5,7\}, \quad \{2,4,6\}, \quad \{3,4,7\},$$
$$\{3,5,6\}.$$

Which we may notice is the same arrangement as the Fano plane [2]. Thus the question got asked: "What if we had more types of coffee? How could we still arrange a fair experiment?" and this is the question that design theory answers. Incomplete block designs and partial incomplete block designs are studied widely [2], [30], with both practical and pure motivation. We shall continue by formally defining the objects we are interested in. Note we shall be following definitions and theorems from both [2] and [30].

Definition 2.27. For t, n, k and λ all positive integers with $t \geq 2$ then a *partial* $t-(n,k,\lambda)$ *design* is a pair of sets $(X, \mathfrak{B})$ which contain *points* and *blocks* respectively. We couple this with an incidence structure and we require that

(i) X contains n points;

(ii) each block $B \in \mathfrak{B}$ is incident with precisely k points of X;

(iii) every t-tuple of points in X is incident with at most λ blocks.

A pair $(X, \mathfrak{B})$ is a $t-(n,k,\lambda)$ *design* if we replace 'at most' in part (iii) of the condition with 'exactly'. By convention a block is described by the points it is incident with, and as such we can describe $\mathfrak{B}$ as a subset of the power set of X, that is each $B \subseteq X$. If $|X| = |\mathfrak{B}|$ then we say it forms a symmetric partial block design.

Theorem 2.28. *For a linear code $C = [n,k,d]_q$ with perspective (θ, δ), let C be the set of points, let $\mathfrak{B} = \{S(c) : c \in C\}$ be the set of blocks and set $t = \max_{b,c \in C} |S(b) \bigcap S(c)|$. Then $(C, \mathfrak{B})$ is a partial $t-(q^n, \theta, t)$ design. Moreover as $|C| = |\mathfrak{B}|$ it forms a symmetric partial block design.*

Proof We know $|C| = q^n$ and as each block $B \in \mathfrak{B}$ is $S(c)$ for some $c \in C$, thus $|B| = |S(c)| = \theta$ and B is thus incident with precisely θ points of C. To complete the

proof we thus need to show that ever t-tuple of points in C is incident with at most t blocks. We do this by proving that the intersection of any $t+1$ blocks must contain 0 or 1 points.

Suppose for contradiction that there exists $t+1$ points $b_1, b_2, \ldots b_{t+1}$ and 2 other points c_1, c_2 such that

$$\bigcap_{i=1}^{t+1} S(b_i) \supseteq \{c_1, c_2\}.$$

Note that we do not know whether $c_1 = b_j$ for some j or not, similarly for c_2. Thus we know that for all i we have

$$c_1, c_2 \in S(b_i),$$

which as we know $b \in S(c)$ if and only if $c \in S(b)$, thus for all i we have

$$b_i \in S(c_1) \text{ and } b_i \in S(c_2)$$

and so this would give

$$\{b_1, b_2, \ldots b_{t+1}\} \subseteq S(c_1) \bigcap S(c_2),$$

which contradicts that $t = \max_{b,c \in C} |S(b) \bigcap S(c)|$. Thus the intersection of any $t+1$ blocks must contain 0 or 1 points.

Thus if the intersection of any $t+1$ blocks contains either 0 or 1 points then any t-tuple of points can only appear in at most t blocks. □

Definition 2.29. A *parallelism* of a block design $(X, \mathfrak{B})$ is a partition of $\mathfrak{B}$ into classes $\mathfrak{B}_i$ for an index set $i \in I$ such that

(i) $\bigcup_{i \in I} \mathfrak{B}_i = \mathfrak{B}$;

(ii) $\mathfrak{B}_i \bigcap \mathfrak{B}_j = \emptyset$ for all $i \neq j$;

(iii) $\bigcup_{B \in \mathfrak{B}_i} B = X$ for all i;

(iv) $B_j \bigcap B_j = \emptyset$ for all $B_j, B_k \in \mathfrak{B}_\mathfrak{i}$ for all $i \in I$.

That is a partition of blocks into classes such that each class is a partition of the point set. Note that a block design which admits a parallelism is sometimes called a *resolvable* block design.

Theorem 2.30. *For a linear code C with a perspective (θ, δ), a focused splitting $D_1, D_2, \ldots, D_\theta$ induces a parallelism of the block design $(C, \mathfrak{B})$.*

Proof Let $\mathfrak{B}_\mathfrak{i} = \{S(c) : c \in D_i\}$ and we wish to prove the criteria in Definition 2.29.

(i)

$$\begin{aligned} \bigcup_{i \in I} \mathfrak{B}_\mathfrak{i} &= \bigcup_{i \in I} \{S(c) : c \in D_i\} \\ &= \{S(c) : c \in C\} \\ &= \mathfrak{B}. \end{aligned}$$

(ii) $\mathfrak{B}_\mathfrak{i} \bigcap \mathfrak{B}_\mathfrak{j} = \{S(c) : c \in D_i\} \bigcap \{S(c) : c \in D_j\}$ and as $D_i \bigcap D_j = \emptyset$ for all $i \neq k$ then $\mathfrak{B}_\mathfrak{i} \bigcap \mathfrak{B}_\mathfrak{j} = \emptyset$.

(iii) For a fixed i we have

$$\bigcup_{B \in \mathfrak{B}_\mathfrak{i}} B = \bigcup_{c \in D_i} S(c)$$

and by counting the words we can see that $|\bigcup_{c \in D_i} S(c)| = |S(c)||D_i| = \theta \frac{M}{\theta} = M$ which as every word contained is in C and are distinct then $\bigcup_{B \in \mathfrak{B}_\mathfrak{i}} = C$.

(iv) For all $B_j, B_k \in \mathfrak{B}_\mathfrak{i}$ for all $i \in I$ we have $B_j \bigcap B_j = S(c_1) \bigcap S(c_2)$ for $c_1, c_2 \in D_i$ and by definition of D_i we have $S(c_1) \bigcap S(c_2) = \emptyset$.

□

2.6 Concluding

In this Chapter we have introduced the concept of perspective and the subsequent concept of focused splittings and have shown several relationships between the latter and perfect codes, specifically that a focused splitting on the complete space $(\mathbb{F}_q)^n$ will be equivalent to a perfect code existing on the complete space. We have gone on to give several constructions of focused splittings, and to establish the existence of focused splittings for extremal values of θ and to show connections with design theory. We shall leave the question of the existence of focused splittings for other parameters, specifically those not covered by Theorem 2.26 where the set $S(0) \setminus \{0\}$ is not linearly independent, and in subsequent Chapters we shall be exploring other aspects of focused splittings, namely the hardness of the problem of finding a focused splitting, the symmetry of focused splittings and also their use within error correction.

Everything is complicated; if that were not so, life and poetry and everything else would be a bore

Wallace Stevens

Chapter 3

Finding a Focused Splitting is NP-complete

3.1 Overview

In this Chapter we discuss complexity theory, a theory in which we try to distinguish different classes of problems and in which we find that some problems can be classed as 'hard', we shall define what we mean for a problem to be hard, that is that a problem is **NP-complete**, and we shall also give a proof of Cook's Theorem. We shall discuss what ramifications this has for the problem and furthermore we shall show that the problem of finding a focused splitting for an arbitrary code is **NP-complete**.

3.2 Complexity Theory

Complexity theory [14] is the study of algorithms and the maximum number of steps they take to return an answer, this is calculated by analysis of the worst case scenario for an arbitrary sized input . The theory generally deals with decision problems, that is questions which have a 'yes/no' answer, as opposed to optimisation problems which look for the smallest or largest of something or problems which ask for an instance of

something satisfying certain criteria. Optimisation problems can be converted to decision problems by asking whether something of 'at least' or 'at most' a certain size exists. So a problem such as "Given a graph G and two vertices u and v what is the length of the shortest path between them?" can be posed as a set of questions: "Given a graph G, two vertices u and v and an integer k is there a path between u and v of length at most k?", and we do this for a range of k to find the shortest length, this thus turns the optimisation problem into a set of decision problems. We note that at most $\lceil \log_2 n \rceil$ values of k need to be considered, where n is the length of the longest path in G, as we can home in on the value of the shortest length by repeatedly halving our search space.

Definition 3.1. A *decision problem* is a function f from the set of instances of a problem I to the set $\{0, 1\}$ that is $f : I \to \{0, 1\}$.

Thus a decision problem takes an instance of a problem $i \in I$ and says whether that problem has a solution, by returning $f(i) = 1$, or if they problem does not have a solution $f(i) = 0$. For example with the problem "Given a graph G, two vertices u and v and an integer k is there a path between u and v of length at most k?", an instance of the problem would be $i = (G, u, v, k)$ and $f(i) = 1$ if there is such a path and $f(i) = 0$ if there is not. Obviously if we are going to be computing with instances of a problem, we need to be able to encode the information in an efficient manner.

Informally an *algorithm* is a collection of instructions which with an instance of a problem $i \in I$ will calculate $f(i)$. To formalise the idea we have to define a *Turing machine* [14], a *Turing Machine* is an abstract concept of what we now call a computer, and it is generally accepted, but ultimately impossible to prove, that anything that is 'effectively calculable' is computable by a Turing machine [23]. Informally a Turing machine is a machine that has an infinite strip of tape, split up into blocks, each of which contains one of a finite number of symbols. The machine has a state q, and depending on the combination of state and symbol the machine changes the symbol on that section of the tape, changes the state the machine is in, and moves one position left or right on the tape.

Definition 3.2. A *Turing machine* is a 7-tuple,

$$(Q, \Sigma, \Gamma, \delta, q_1, q_{accept}, q_{reject})$$

where

1. Q is a finite set of states for the machine,
2. Σ is the input alphabet, not containing a blank symbol $\sqcup$,
3. Γ is the tape alphabet, with $\sqcup \in \Gamma$ and $\Sigma \subset \Gamma$,
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{-1, +1\}$ is the transition function,
5. $q_1 \in Q$ is the starting state,
6. q_{accept} is the accept state,
7. q_{reject} is the reject state and $q_{reject} \neq q_{accept}$.

A Turing machine is fed an infinite input $L \in \Gamma^{\mathbb{Z}}$ where the blank symbol is the only symbol appearing infinitely often. We use $L(p)$ to denote the symbol on the tape in position p. The machine starts with a position vector of $p = 0$ and a state $q = q_1$, and at each step of the computation the machine computes $\delta(q, L(p)) = (q', r, s)$, and then changes q to q', $L(p)$ to r and p to $p + s$. If $q' = q_{accept}$ or $q' = q_{reject}$ then the machine halts, otherwise it continues with the next step of the computation.

We say a Turing machine *accepts* an input if the machine eventually halts with q_{accept} and we say the Turing machine *rejects* an input if the machine halts with q_{reject}.

We say an Turing machine *solves* a decision problem if the Turing machine accepts the input precisely when the decision problem has a solution, that is $f(i) = 1$, and the Turing machine rejects the input precisely when the decision problem does not have a solution, that is $f(i) = 0$.

For a decision problem we can talk about a signature of a solution, which is information relating to the solution of a given instance of the problem, usually the signature of a solution will be an instance of the object having specific properties, for example a path between two vertices in a graph which has a length less than the required distance. Thus if we have a Turing machine which solves the decision problem, we can create a Turing machine which can check whether a given signature is the signature of the solution to a given instance of the decision problem. Using this observation we can define a non-deterministic Turing machine.

Definition 3.3. A *non-deterministic Turing machine* is a Turing machine coupled with a random function $r : \mathbb{N} \to \Sigma$, that is a function which maps on to the alphabet with every symbol appearing with a positive probability, and a given integer n. The random function writes to the tape for positions from -1 to $-n$ a random symbol from Σ, where n is the desired length of a signature and then returns to the starting position before running as a Turing machine.

We require the non-deterministic Turing machine to accept the random input if and only if it is the signature to a solution of the instance, which is given to the machine in the normal manner, and rejects the random input if it is not. We say a non-deterministic Turing machine *solves* a decision problem if for a given instance and random input it accepts the random input precisely when the input is a solution of the instance.

Now that we have the concepts of a Turing machine and a non-deterministic Turing machine we can consider the complexity of algorithms. The two classes that we are primarily going to be interested in are **P** and **NP**, and we shall also be discussing **NP-complete** problems. Informally we can think of the class **P** to be problems with an efficient algorithm for solving them, the class of **NP** to be problems that a solution to can be checked efficiently but not necessarily computed efficiently, and the class of **NP-complete** problems to be the hardest problems within **NP**. Before we can formally define such concepts we must define what we mean by the complexity of an algorithm,

which although formally we are talking about Turing machines, we can keep the colloquial concept of an algorithm at hand.

Definition 3.4. Let $T = (Q, \Sigma, \Gamma, \delta, q_1, q_{accept}, q_{reject})$ be a Turing machine or a non-deterministic Turing machine solving a decision problem f and let i be an instance of the problem, then we define $\#T(i)$ to be the number of time steps T takes before returning either q_{accept} or q_{reject}.

Informally we can think of this as the number of steps an algorithm will take to solve a specific case of the problem.

Definition 3.5. Let $T = (Q, \Sigma, \Gamma, \delta, q_1, q_{accept}, q_{reject})$ be a Turing machine or a non-deterministic Turing machine solving a decision problem f, then for a function $g(n)$ we say T has complexity $O(g(n))$ if for $h(n)$ defined as

$$h(n) = \sup\{\#T(i) : |i| = n\}$$

there exists a real number M such that $h(n) \leq Mg(n)$ for all values of n.

Informally this can be thought of as the worst case scenario, that is what is the longest that the algorithm could take to solve an instance of the problem. We will often refer to a problem having a given complexity, by which we shall mean that a machine solving that problem will have that complexity.

It is clear that when evaluating an algorithm it is simple enough to just consider the overall behaviour of $h(n)$ and to ignore all terms in any finite expansion of $h(n)$ except for the one with the quickest growth rate, see [11] for details.

Now we can define the complexity classes of **P** and **NP**.

Definition 3.6. A decision problem f is in **P** if for a Turing machine T solving f, and for a polynomial $g(n)$, T has complexity $O(g(n))$. We say f is of *polynomial time*.

Definition 3.7. A decision problem f is in **NP** if for a non-deterministic Turing machine T solving f, and for a polynomial $g(n)$, T has complexity $O(g(n))$. We say f is of *non-deterministic polynomial time*

We can see that every problem f in **P** is in **NP** by constructing a non-deterministic Turing machine which solves f. We do this by letting it run as the Turing machine which solves f and then comparing this solution to the guessed solution, this forms a non-deterministic Turing machine as it runs on the deterministic input and then compares the signature from the computation to the guessed solution formed by random input. Moreover the non-deterministic Turing machine will only accept the random input if it agrees with the calculated output. Thus the complexity of the non-deterministic Turing machine is at most the complexity of the Turing machine, as the process of checking the random input against the calculated output is linear and thus $\mathbf{P} \subseteq \mathbf{NP}$. Such an inclusion begs the question $\mathbf{P} = \mathbf{NP}$ and unfortunately this remains an open, and famous, problem see [14], but we can make further progress with the problem via the seminal Cook's Theorem [14] and with the theory of reductions, see [14] and [22].

If we have two problems, one of which we know about, a useful concept for the analysis of the unknown problem is the concept of reductions. A reduction is a polynomial algorithm which converts an instance of the understood problem into an instance of the new problem, such that any solution to the new problem gives a solution to the understood problem, and thus if there existed a polynomial algorithm solving the new problem it could be converted to a polynomial algorithm solving the understood problem. This would be done by running the algorithm for the new problem and then interpreting the result for the understood problem. Thus the new problem can be seen to be at least as hard as the understood problem.

Definition 3.8. For two decision problems f_1 and f_2, with instance classes of I_1 and I_2 respectively, we say that f_1 transforms to f_2, written $f_1 \propto f_2$ if and only if there exists a function $t : I_1 \to I_2$ such that $f_1(i) = f_2(t(i))$ for all $i \in I_1$, providing there is a Turing

machine which computes t in polynomial time.

Lemma 3.9. *For two decision problems f_1 and f_2, if $f_1 \propto f_2$ and $f_2 \in \mathbf{P}$ then $f_1 \in \mathbf{P}$.*

Proof As $f_1 \propto f_2$ then we know that there exists a function t such that $f_1(i) = f_2(t(i))$ for all $i \in I_1$ and that t can be computed by a Turing machine in polynomial time, thus the size of $t(i)$ is bounded by a polynomial in the size of i. As we know $f_2 \in \mathbf{P}$ then f_2 can be computed by a Turing machine in polynomial time. Thus $f_2(t(i))$ can be calculated by a Turing machine in polynomial time by first calculating $i' = t(i)$ and then calculating $f_2(i')$, each of which can be done in a polynomial amount of time, and as the product of two polynomials is a polynomial, f_1 must have polynomial time. □

Thus we can see that if $f_1 \propto f_2$ then we can say that f_2 is at least as hard as f_1. It is clear that if $f_1 \propto f_2 \propto f_3$ then $f_1 \propto f_3$, that is $\propto$ is a transitive relation and as such reductions can be used to provide a hierarchy on the space of decision problems. In 1971 Stephen Cook showed in [10] that the problem of Boolean satisfiability (herein referred to as SAT) is at least as hard as any problem in **NP**, that is he showed that for every problem $f \in \mathbf{NP}$ that $f \propto SAT$. In doing this Cook gave the first example of a problem being **NP-complete**, this is as SAT was shown to be at least as hard as any problem in **NP**. This means that if $SAT \in \mathbf{P}$ then it would show that $\mathbf{P} = \mathbf{NP}$. Similarly if SAT reduced to another problem $f \in \mathbf{NP}$ then the same could be said for f, as $f \in \mathbf{P}$ would imply $SAT \in \mathbf{NP}$ which would imply $\mathbf{P} = \mathbf{NP}$.

Definition 3.10. A decision problem f is **NP-complete** if and only if

(i) $f \in \mathbf{NP}$;

(ii) for all $f' \in \mathbf{NP}$ we have that $f' \propto f$.

It should be clear that by the transitivity of $\propto$ that condition (ii) in the above definition could be considered as there existing some known **NP-complete** problem that transforms to f. Before we can prove Cook's Theorem we must first define what we mean by the

problem SAT and before we can do this we must define a few concepts which we shall subsequently use.

Definition 3.11. Let $U = \{u_1, u_2, ...u_n\}$ be a set of *variables* and then a *truth assignment* for U is a function $b : U \to \{T, F\}$, where we say u *is true under* b if and only if $b(u) = T$ and u *is false under* b if and only if $b(u) = F$.

Definition 3.12. If $u \in U$, then u and $\bar{u}$ are *literals* over U. The literal u is true under b if and only if u is true under b, the literal $\bar{u}$ is true under b if and only if u is false under b.

Definition 3.13. A *clause* over U is a set of literals over U, which represents the disjunction of those literals. Thus a clause is *satisfied* under b if and only if at least one of the literals is true under b.

Definition 3.14. A set of clauses S is *satisfiable* if and only if there exists a truth assignment b such that each clause is satisfied.

Definition 3.15. The problem SAT is defined as such "Given a set of clauses S over a set of variables U, is S satisfiable?"

Theorem 3.16. *The problem SAT is* **NP-complete**.

Proof This will only be a sketch proof giving the overall flavour of the full proof, for complete details see [14] or [10]. The main concept is that we consider an arbitrary problem f in **NP** and we prove that f transforms to SAT. As f is in **NP** we know that there exists a non-deterministic Turing machine which solves f with polynomial complexity. Thus to show f transforms to SAT we create a set of boolean clauses which are satisfied precisely when the non-deterministic Turing machine which solves f accepts a given input, and this is done by emulating the actions of the non-deterministic Turing machine.

For a given size of input n, we know that on a non-deterministic Turing machine f has a complexity of $O(g(n))$ and thus there exists a polynomial $p(n) = Mg(n)$ which bounds the total number of times steps taken by the non-deterministic Turing machine to solve f. Similarly if the non-deterministic Turing machine only takes $p(n)$ time steps, it can only feasibly use $p(n)$ tape spaces, and as the non-deterministic Turing machine starts at position 0, the only tape positions used can be in the range $-p(n)$ to $p(n) + 1$.

To construct our variable set U we must consider how to represent the non-deterministic Turing machine, this can be done by at any given time representing the state of the machine, the position of the read head, and the contents of each tape position. Thus we create the following variables with the following conditions

$$\begin{aligned} Q[i,k] = T &\iff \text{at time } i \text{ machine is in state } q_k \\ H[i,j] = T &\iff \text{at time } i \text{ read head is in position } j \\ S[i,j,l] = T &\iff \text{at time } i \text{ position } j \text{ contains symbol } s_l \end{aligned}$$

We do this for all values $0 \leq i \leq p(n)$, $1 \leq k \leq r$, $-p(n) \leq j \leq p(n) + 1$ and $0 \leq l \leq v$ where $r = |Q|$ and $v = |\Gamma| - 1$. Moreover we specify state q_r to be q_{accept}.

Now we create our set of clauses, we break the set into several subsets depending on the purpose the clauses have in emulating the non-deterministic Turing machine. There are six subsets in total which we shall denote S_1 to S_6, each of which has a specific purpose and places a specific restriction on the way the variables in U can interact such that they emulate the non-deterministic Turing machine. We need to make sure that the machine is in exactly one state at any one time, this is the purpose of S_1. We need to make sure that the read head is in exactly one position at any one time, this is the purpose of S_2. We need to make sure that each position on the tape contains exactly one symbol at any one time, this is the purpose of S_3. We need to make sure that at time 0 that the machine is in it's initial configuration, this is the purpose of S_4, here we don't specify the contents

of the part of the tape which would have been the guess, because then the satisfiability of the clauses says whether there would exist some guess which would satisfy the initial conditions. We need to check that by time $p(n)$ the machine will have accepted the input, this is the purpose of S_5. Most importantly we need to check that moving from each time step to the next, that the only differences in the the state of the machine, the contents of the tape, and the position of the read head, are differences that would be caused by the non-deterministic Turing machine which solves f moving forward in time with the transition function, this is the purpose of S_6. It is easy to see that subsets S_1, S_2, S_3 and S_6 ensure that the emulated machine would act as a machine would, that is having only one state at any time, reading only one tape position at any one time, each tape square containing only one symbol at any one time and the change in states, symbols and position of read head only happens as dictated by the transition function. Subset S_4 ensures that the machine starts off in an initial configuration, otherwise it would be calculating a different instance of the problem. Subset S_5 ensures that the machine will have accepted the input, because not only do we require the the set of clauses to emulate a working machine but we also require the input to be accepted.

We now display how each of clauses in these subsets are formed. S_1 is formed by taking clauses of the form

$$\{Q[i,0], Q[i,1], \ldots, Q[i,r]\} \text{ for all } 0 \leq i \leq p(n)$$

and clauses of the form

$$\{\overline{Q[i,j]}, \overline{Q[i,j']}\} \text{ for all } 0 \leq i \leq p(n) \text{ and for all } 1 \leq j < j' \leq r.$$

The first of these ensures that for each time step there is a state, and the second ensures

that there is at most one per time step. S_2 is formed by taking clauses of the form

$$\{H[i,-p(n)], H[i,-p(n)+1], \ldots, H[i,p(n)+1]\} \text{ for all } 0 \leq i \leq p(n)$$

and clauses of the form

$$\{\overline{H[i,j]}, \overline{H[i,j']}\} \text{ for all } 0 \leq i \leq p(n) \text{ and for all } -p(n) \leq j < j' \leq p(n)+1.$$

The first of these ensures that at each time step the read head is in some position, and the second ensures that there is at most one per time step. S_3 is formed by taking clauses of the form

$$\{S[i,j,0], S[i,j,1], \ldots, S[i,j,v]\} \text{ for all } 0 \leq i \leq p(n) \text{ and for all } -p(n) \leq j \leq p(n)+1$$

and clauses of the form

$$\{\overline{S[i,j,l]}, \overline{S[i,j,k']}\} \text{ for all } 0 \leq i \leq p(n), \text{ for all } -p(n) \leq j \leq p(n)+1 \text{ and for all } 0 \leq k < k' \leq v.$$

The first ensures that each tape square contains at least one symbol, and the second ensures that there is at most one. S_4 is formed by taking the following clauses

$$\{Q[0,1]\}, \{H[0,0]\}$$

$$\{S[0,0,0]\}\{S[0,1,l_1]\}, \{S[0,2,l_2\}, \ldots, , \{S[0,n,l_n\}$$

and

$$\{S[0,n+1,0\}, \{S[0,n+2,0\}, \ldots, \{S[0,p(n)+1,0\}$$

where the information about the instance is encoded $s_{l_1} s_{l_2} \ldots s_{l_n}$. The first three of these clauses are to make sure the state and read head start off in the position in which a

machine would start, the next n are to ensure that the information about the instance is encoded and the final $p(n)+1-n$ are to ensure no further information is encoded as the instance. S_5 is formed by taking the clause

$$\{Q[p(n), r]\}.$$

This is to ensure that by the maximum amount of time we know a computation would take, that the machine has entered the accept state. S_6 is formed of clauses of two different types. The first type of clauses in S_6 are of the form

$$\{\overline{S[i,j,l]}, H[i,j], S[i+1,j,l]\} \text{ for all } 0 \leq i < p(n),\ -p(n) \leq j \leq p(n)+1 \text{ and } 0 \leq l \leq v.$$

This is such that if $S[i,j,l]$ is true, that is at time i tape position j contains symbol s_l, then either the read head is at position j and a time i, and thus a change may occur, or that $S[i+1,j,l]$. Thus these clauses are true if the tape doesn't change other than via the transition function. The second type of clauses in S_6 are of the form

$$\{\overline{H[i,j]}, \overline{Q[i,k]}, \overline{S[i,j,l]}, H[i+1, j+s],$$

$$\{\overline{H[i,j]}, \overline{Q[i,k]}, \overline{S[i,j,l]}, Q[i+1,k']\}$$

and

$$\{\overline{H[i,j]}, \overline{Q[i,k]}, \overline{S[i,j,l]}, S[i+1,j,l']\}$$

for every quadruple of (i,j,k,l) with $0 \leq i < p(n)$, $-p(n) \leq j \leq p(n)+1$, $1 \leq k \leq r$ and $0 \leq l \leq v$ and where if $q_k \in Q \setminus \{q_{accept}, q_{reject}\}$ then $\delta(q_k, s_l) = (q_{k'}, s_{l'}, s)$ and if $q_k \in \{q_{accept}, q_{reject}\}$ then $s = 0$, $k' = k$, $l' = l$. This is such that for each each possible combination of time(i), position of read head (j), state (k) and symbol being read (l) either the machine is not in the specified state, position and symbol or the state, position and

symbol at the next time step is as implied by the transition function. For $S_1, S_2, \ldots, S_6$ it is fairly easy to see that each of these subsets contains only a polynomial of the input size number of clauses, and thus the entire set S is polynomial of the size of the input. Also we note that an accepting computation for a guessed input on the non-deterministic Turing machine would give a truth assignment to U which would satisfy S, and that any truth assignment satisfying S could be used to calculating a guess, the tape positions -1 to $-p(n)$ at time 0 of the accepting Turing machine, which would be accepted by the non-deterministic Turing machine. Thus the satisfying truth assignments are in one-to-one correspondence with the accepting guesses for f and thus $f \propto SAT$. □

3.3 Finding a Focused Splitting is NP-Complete

In the following we show that finding a focused splitting in a code C is **NP-complete** by a reduction from the problem of finding the total chromatic number of a graph, which is known to be **NP-complete** [29]. We start with some definitions.

Definition 3.17. A *graph* G is a pair (V, E) where V is the vertex set, and E the edge set, each vertex is just a single element and each edge is incident to 2 vertices, and as such is sometimes represented as a pair of vertices. We shall represent an edge by the pair of vertices it is incident to in either order. A *simple* graph is a graph where there is at most one edge between any two vertices. A pair of graphs are *isomorphic*, written $G \cong G'$ if a relabelling of the vertices in one gives the other.

Definition 3.18. The *perspective graph* of a code C is a graph where the vertices are the words in C, that is $V = C$ and an edge exists between two distinct vertices if and only if the words in C have a distance less than δ, that is

$$E = \{(w_i, w_j) : w_i, w_j \in C, d(w_i, w_j) < \delta\}.$$

We denote the perspective graph of a code C by $G(C)$.

Definition 3.19. The distance between two vertices on a graph G is the number of edges in the shortest path between them. If no such path exists we say the distance is infinite. For two vertices v and w we denote the distance $d(v, w)$ with

$$d(v, w) = \min\{|\{e_1, e_2, \ldots, e_t\}| : e_1 = (v, w_1), e_2 = (w_1, w_2), \ldots, e_t = (w_{t-1}, w)\}.$$

Definition 3.20. A distance-2 vertex colouring on a graph G is a colouring of the vertices of G such that any two vertices with a distance less than or equal to 2 have distinct colours.

For analysis of the question of finding focused splittings, and distance-2 colourings, we must formalise the questions.

Definition 3.21. The problem *F.S.* is defined as "Given a code C, with perspective (θ, δ), does there exist a focused splitting on C?"

Definition 3.22. The problem $D2$ is defined as "Given a graph G does there exist a distance-2 vertex colouring on G?"

Theorem 3.23. *A code C has a focused splitting if and only if the perspective graph $G(C)$ has a distance-2 vertex colouring in θ colours.*

Proof We start by proving the forwards implication. Let C have a focused splitting $D_1, D_2, \ldots, D_\theta$, we label the words in $C = \{w_1, w_2, \ldots, w_M\}$ and we construct a colouring on $G(C)$ by labelling the vertices in $G(C) = \{v_1, v_2, \ldots, v_M\}$ respectively and colouring v_i colour j if and only if $w_i \in D_j$.

We claim that this gives a distance-2 vertex colouring on $G(C)$. We prove this by contradiction. If it did not give a distance-2 vertex colouring on $G(C)$ then there exists 2 vertices, v_i and v_j such that $d(v_i, v_j) \leq 2$ with v_i and v_j coloured the same colour (say colour k). This then breaks down into 2 cases:

(1) The first case is where the distance is 1, that is $d(v_i, v_j) = 1$ which implies the equivalent words in C will be neighbours, that is $w_i \in S(w_j)$ and $w_i, w_j \in D_k$. Thus

$w_i, w_j \in (D_k \cap S(w_j))$ which implies $|D_k \cap S(w_j)| \geq 2$ and thus $D_1, D_2, \ldots, D_\theta$ is not a focused splitting. This is a contradiction.

(2) The second case is where the distance is 2, that is $d(v_i, v_j) = 2$ which implies the existence of v_L such that $d(v_i, v_L) = 1$ and $d(v_j, v_L) = 1$, that is there is a interim vertex between v_i and v_j and thus the equivalent words in C both belong to the interim word's society, that is, $w_i, w_j \in S(w_L)$ and $w_i, w_j \in D_k$, as they are the same colour. Thus $w_i, w_j \in (D_k \cap S(w_L))$ which implies $|D_k \cap S(w_L)| \geq 2$ and thus $D_1, D_2, \ldots, D_\theta$ is not a focused splitting. This is again a contradiction, which completes the claim and proves the forwards implication.

Now we prove the reverse implication. Let $G(C)$ have a distance-2 vertex colouring in θ colours. Then we place the words in C in split codes depending on the colour of the associated vertex. Thus $w_i \in D_j$ if and only if v_i is coloured colour j.

We claim that this gives C a focused splitting. We prove this by contradiction. If it did not give C a focused splitting then there exists i, j such that $|D_j \cap S(w_i)| \geq 2$ and this implies there exists w_k, w_m with w_k, $w_m \in S(w_i)$ and w_k, $w_m \in D_j$ (although we require w_k and w_m to be distinct words in C, there is no requirement for either of them to be distinct from w_i) and we can see that this implies that there exists v_k, $v_m \in G(C)$ such that v_k, v_m both coloured j and that $d(v_k, v_m) \leq d(v_k, v_i) + d(v_i, v_m) \leq 1 + 1 = 2$ and thus the colouring on $G(C)$ is not a distance-2 colouring, as v_k and v_m have a distance less than 2 and are coloured the same, which is a contradiction, and thus we prove the claim.

Thus a code C has a focused splitting if and only if the perspective graph $G(C)$ has a distance-2 vertex colouring in θ colours. □

Lemma 3.24. *For any simple graph G, there exists a code C, such that $G(C) \cong G$.*

Proof Proof is given by a construction of C from G such that $G(C) \cong G$.

Let $G = (V, E)$. We construct C over an alphabet of size $|V|$. From the definition we will require $|C| = |V|$ as each word of C will give a vertex in $G(C)$, and we let $M = |C|$.

We label $V = \{v_1, v_2, \ldots, v_M\}$ and we consider C to be made up of M words which we label $C = \{w_1, w_2, \ldots, w_M\}$ without yet knowing what each word looks like. We do this so that v_i will map to w_i which will map to v_i. Set the length of the code $n = |E|$ and label $E = \{e_1, e_2, \ldots, e_n\}$. Then we define each word $w_i = \alpha_{i,1}, \alpha_{i,2}, \ldots, \alpha_{i,n}$ with

$$\alpha_{i,j} = \begin{cases} 0 & \text{if } v_i \in e_j, \\ i & \text{otherwise.} \end{cases}$$

As the graph G is simple each pair of words will have a maximum of one edge joining them and each co-ordinate will contribute a value of 1 to the distance between two words unless those two words are joined by the relevant edge, which happens at most once. Thus

$$d(w_i, w_j) = \begin{cases} n-1 & \text{if } v_i v_j \in E, \\ n & \text{if } v_i v_j \notin E. \end{cases}$$

So by setting $\delta = n$, only the words that are adjacent in G will have a distance of $n-1$ between them and thus be in the same society in C and thus adjacent words in G will be adjacent in $G(C)$. Words that are non-adjacent in G will have a distance of n between them in C and so will be not be in the same society in C and so will be non-adjacent in the perspective graph $G(C)$. Thus G and $G(C)$ have the same vertex set and the same adjacency between vertices and thus the same edge set and so $G(C) \cong G$. □

Corollary 3.25. *$D2 \propto F.S.$*

Proof By Theorem 3.23 a focused splitting of a code C and a distance-2 colouring in θ colours of a graph $G(C)$ are equivalent, and by Lemma 3.24 for every graph G there exists a code C such that $G(C) \cong G$. Therefore all we need to show is that this transformation from G to C can be performed in polynomial time. We note that an instance of the problem $D2$ can be described by the pair (V, E), and thus has size at most $n + \frac{n(n-1)}{2}$ where n is the size of V, we also note that an instance of C can be described by the set of co-ordinates C and the numbers θ and δ and we observe that in our construction C has n words each of length of $|E|$, which is at most $\frac{n(n-1}{2}$ from an alphabet of size n, and thus C has at most size $n^{2\frac{n(n-1)}{2}}$ which is clearly bounded by a polynomial in the size of

$n + \frac{n(n-1)}{2}$. To build C from G, we must look at each edge, and then set $\alpha_{i,j}$ accordingly, therefore we need perform only $|E||V|$ operations and thus this is polynomial in the size of G. Thus $D2 \propto F.S.$ □

This construction of C from G is by no means the only one, and although it creates a code with a high value of q this is by no means necessary.

Lemma 3.26. *Any simple graph G can be transformed to a binary code C where the perspective graph of the code is the initial graph, that is $G(C) \cong G$.*

Proof Having generated a code C', with an alphabet of size q as in Lemma 3.24, replace every symbol α with q symbols, all 0 except the α-th which would be 1, with the symbol 0 getting mapped to 0^q. All distances between non-zero co-ordinates will be doubled, as what was originally symbols i and j will now differ in the i-th and j-th positions and thus distances between non-zero co-ordinates in C' will be doubled, subsequently setting $\delta = 2n - 1$ gives the same perspective graph. □

We now look at *total colourings* of a graph [29], the question of whether a total colouring exists and we tie that in to our question of focused splittings.

Definition 3.27. A total colouring on a graph G is a colouring of the vertices and edges such that no adjacent vertices share the same colour, no adjacent edges share the same colour, and no vertices with their adjacent edges share the same colour.

Definition 3.28. The problem $TCOL$ is defined as "Given a graph G does there exists a total colouring on G?"

Theorem 3.29. *The problem of whether a graph G has a total colouring in n colours can be reduced to the problem of whether a related graph $\bar{G}$ has a distance-2 colouring in n colours. That is $TCOL \propto D2$.*

Proof We prove the theorem by constructing $\bar{G}$ from G. Let $G = (V, E)$ and label $V = \{v_1, v_2, \ldots, v_V\}$ then set $V^* = \{v_{(i,j)} : v_i v_j \in E\}$, this is the set of new vertices, one

for each edge in E. We set $\bar{G} = (\bar{V}, \bar{E})$ where $\bar{V} = V \cup V^*$ and $\bar{E} = \{v_i v_{(i,j)} : v_i v_j \in E\}$ that is the edges of $\bar{G}$ are representing the adjacency of vertices and edges in the original graph.

Now we claim a total colouring exists on G in n colours if and only if a distance-2 colouring exists on $\bar{G}$ in n colours. To prove the claim, consider if there was a total colouring on G in n colours, then each pair of the form:

(1) (v_i, v_j) for every edge $v_i v_j \in E$, so this is pairs of adjacent vertices;

(2) $(v_i, v_i v_j)$ for every edge $v_i v_j \in E$, so this is pairs of vertices and edges which are adjacent;

(3) $(v_i v_j, v_i v_k)$ for every pair of edges $v_i v_j$ and $v_i v_k \in E$, that is pairs of edges which share a vertex;

would have distinct colours.

Similarly if there was a distance-2 vertex colouring on $\bar{G}$ in n colours then as the only edges in $\bar{G}$ are between vertices in $V \subset \bar{V}$ and vertices in $V^* \subset \bar{V}$ thus the only paths of distance-2 or less in $\bar{G}$ are

(i) v_i to v_j, for $v_i v_j \in E$, these are paths of length 2 starting at a vertex in V;

(ii) v_i to $v_{(i,j)}$ for $v_i v_j \in E$, these are paths of length 1 in $\bar{G}$;

(iii) $v_{(i,j)}$ to $v_{(i,k)}$ for all pairs $v_i v_j$, and $v_i v_k \in E$, these are paths of length 2 starting at a vertex in V^*;

and thus the vertices at the ends of each of these paths would be coloured distinct colours.

Now as the pairs (1), (2) and (3) are equivalent to the paths (i), (ii) and (iii) we can see that if we colour the vertices $v_i \in V \subset \bar{V}$ the colour of $v_i \in V$, and the vertices $v_{(i,j)} \in V^* \subset \bar{V}$ the colour of $v_i v_j \in E$. Then a total colouring in G would produce a distance-2 vertex colouring in $\bar{G}$.

Equivalently if we colour the vertices $v_i \in V$ the colour of $v_i \in V \subset \bar{V}$ and if we colour the edges $v_i v_j \in E$ the colour of the vertices $v_{(i,j)} \in V^* \subset \bar{V}$. Then a distance-2 vertex colouring in $\bar{G}$ would produce a total colouring in G.

Thus a total colouring exists on G in n colours if and only if a distance-2 colouring exists on $\bar{G}$ in n colours. □

Corollary 3.30. *The problem of whether a graph has a distance-2 colouring in n colours and the problem of whether a code C has a focused splitting are both* **NP-complete**.

Proof We can see that the problem of whether a graph G has a distance-2 vertex colouring is in **NP** because we could be given a colouring and check whether it was a distance-2 vertex colouring by looking at each vertex and checking all of the vertices at distance-2 from it. Let d be the maximum degree of a vertex in G, then you could check each of the V vertices, and each vertex has at most d neighbours and so less than d^2 other vertices at distance-2 and so this would have a complexity less than $V \times d^2$, which is polynomial. Thus the problem is in **NP**.

Moreover as the problem of whether a graph has a total colouring in n colours is known to be **NP-complete** [29], thus the problem of whether a graph has a distance-2 colouring in n colours is NP-complete.

The problem of whether a code C has a focused splitting is equivalent to the problem of whether a graph $G(C)$ has a distance-2 vertex colouring in θ colours, as shown in Theorem 3.23, and for any graph G there exists a code C with $G(C) = G$ as shown in Lemma 3.24 and the problem of whether a graph G has a distance-2 colouring in n colours was shown to be **NP-complete** in Theorem 3.29, and thus the problem of whether a code C has a focused splitting is NP-complete. That is as $TCOL \propto D2$ and $D2 \propto F.S.$ thus $TCOL \propto F.S.$ □

3.4 The Ramifications of Being NP-Complete

As we now know that determining whether a given code has a focused splitting is, in general, **NP-complete** then we know that we are highly *unlikely* to find a polynomial algorithm for doing so and that if we ever did it would be a ground breaking result. As such we shall be leaving the question of when does a given code have a focused splitting and shall instead be focusing, excuse the pun, on the consequences of a given code having a focused splitting, and on the properties a focused splitting may have, such as the automorphisms of one.

To find a form that accommodates
the mess, that is the task of the artist
now

Samuel Beckett

Chapter 4

Automorphisms of Focused Splittings

4.1 Overview

In this Chapter we shall be investigating the automorphisms that a focused splitting can have. We shall show how the automorphism group can be decomposed into smaller groups depending on the structural properties of the focused splitting and perspective and we shall use this to determine the automorphism groups for certain classes of focused splitting. We shall start by defining a group action [24].

4.2 Preliminaries

Definition 4.1. For a group G and a set S the *action of G on S* is a map, $G \times S \to S$ such that

(i) $es = s \quad \forall s \in S$ where $e \in G$ is the identity element;

(ii) $(\alpha_1\alpha_2)s = \alpha_1(\alpha_2 s) \quad \forall \alpha_1, \alpha_2 \in G$, $s \in S$, that is the action is associative.

We shall now define the 'natural' permutation group of a code C.

Definition 4.2. For a code $C = (n, M, d)_q$ we label the elements of $C = \{c_1, c_2, ...c_M\}$. Let S_M be the symmetric group, that is, all permutations on M elements and we define the *natural action of* S_M on C in the following manner:

For an element $\alpha \in S_M$ which permutes an element i to an element j, for i, j in the range 1 to M, we define $\alpha c_i = c_j$ where $c_i, c_j \in C$.

It is clear from the definition that the labelling of elements of C does not affect the action of the permutation group as S_M is all permutations between M elements. Note we shall also refer to αS where $S \subseteq C$, by which we shall mean the set of elements you get to from applying α to S, that is $\alpha S = \{c \in C : \alpha s = c, s \in S\}$.

We now define an automorphism of a focused splitting.

Definition 4.3. Let C be a $(n, M, d)_q$ code, with a focused splitting $D_1, D_2, \ldots D_\theta$. Let S_M act on C in the natural way. Then $\alpha \in S_M$ is *an automorphism of the focused splitting of* C if and only if,

(i) for all $b, c \in C$ then $b, c \in D_i$ if and only if $\alpha b, \alpha c \in D_j$ for some j;

(ii) for all $b, c \in C$ then $c \in S(b)$ if and only if $\alpha c \in S(\alpha b)$.

That is α preserves the structure of the split codes and of the societies of C.

In a simplification of notation we shall be binding a focused splitting $D_1, D_2, \ldots, D_\theta$ to the code C, and as such in this chapter where we refer to a code C with a focused splitting it shall be assumed that a given focused splitting is known and is the focused splitting being considered. We do this to avoid lengthy, and ultimately unnecessary notation from cluttering the work. If we were to consider one code with two focused splittings on it then we would treat this as two distinct codes each with an associated focused splitting, this is because it is the structure of the focused splitting that we are interested in looking at and not the structure of the underlying code. This is also why we only address the general permutations of the set of C that preserve the structure of the focused splitting rather

than the automorphisms of the code C, that is permutations of co-ordinates and alphabet which preserve the codewords of C, Had we taken this latter definition we would have all kinds of groups appearing as automorphisms of focused splittings which themselves may have a simplistic structure, purely by virtue of the underlying code having an interesting structure. It is of course perfectly valid to consider the interaction of these two definitions and consider the automorphisms of the underlying code which also preserve the structure of the focused splitting, but but being this the first investigation into this realm we feel it better to concentrate attention on the structure of the focused splittings themselves.

Thus we define the automorphism group of a focused splitting to be the set of of all automorphisms of the focused splitting.

Definition 4.4. $\Gamma(C) = \{\alpha \in S_M : \alpha \text{ is an automorphism of the focused splitting of } C\}$.

Lemma 4.5. $\Gamma(C)$ *is a group.*

Proof We prove this by showing that $\Gamma(C)$ is a subgroup of S_M. Now by definition $\Gamma(C) \subseteq S_M$ so to show it is a subgroup we must show it is closed and that it contains inverses, since S_M is finite this reduces to just showing that $\Gamma(C)$ is closed. To show $\Gamma(C)$ is closed we must check that for α_1 and α_2 in $\Gamma(C)$ that $\alpha_1\alpha_2$ meets the criteria for membership of $\Gamma(C)$:

(i) If $\alpha_1, \alpha_2 \in \Gamma(C)$ thus for all $b, c \in C$ we have $b, c \in D_i$ if and only if $\alpha_1 b, \alpha_1 c \in D_j$ for some j and we have $b, c \in D_i$ if and only if $\alpha_2 b, \alpha_2 c \in D_j$ for some j. Thus $b, c \in D_i$ if and only if $\alpha_2 b, \alpha_2 c \in D_j$ for some j which happens if and only if $\alpha_1\alpha_2 b, \alpha_1\alpha_2 c \in D_t$ for some t.

(ii) If $\alpha_1, \alpha_2 \in \Gamma(C)$ thus for all $b, c \in C$ we have $c \in S(b)$ if and only if $\alpha_1 c \in S(\alpha_1 b)$ and we have $c \in S(b)$ if and only if $\alpha_2 c \in S(\alpha_2 b)$. Thus $c \in S(b)$ if and only if $\alpha_2 c \in S(\alpha_2 b)$ which happens if and only if $\alpha_1\alpha_2 c \in S(\alpha_1\alpha_2 b)$.

Thus $\Gamma(C)$ is closed and this completes the proof. □

Lemma 4.6. *Let C be a code consisting of components $C_1, C_2, \ldots, C_t$. Then for $\alpha \in \Gamma(C)$ the image of a component is a component, that is $\alpha C_i = C_j$ for some $j \in [1, t]$.*

Proof If $b, c \in C_i$ then there exists a finite sequence of elements $b_1, b_2, ...b_y$ such that $b \in S(b_1)$, $b_y \in S(c)$ and for all $i \in [1, y-1]$ we have $b_i \in S(b_{i+1})$. Thus as $\alpha \in \Gamma(C)$ then we must have that $\alpha b \in S(\alpha b_1)$ and $\alpha b_y \in S(\alpha c)$ and for all $i \in [1, y-1]$ we have $\alpha b_i \in S(\alpha b_{i+1})$, thus αb and αc are still in the same component, thus $\alpha C_i \subseteq C_j$. Assume there was an element $d \in C_j$ such that $\alpha^{-1} d \notin C_i$, and let $\alpha c = c'$. By the first part of this proof, as c' and d are in the same component, then $\alpha^{-1} c'$ and $\alpha^{-1} d$ must be. However $\alpha^{-1} c' = \alpha^{-1} \alpha c = c \in C_i$; thus $\alpha^{-1} d \in C_i$. Thus we conclude $\alpha C_i = C_j$. □

4.3 Decomposing the Automorphism Group

Before we can deal with specific cases we have to be able to break down the calculation for codes with a different number of components.

Definition 4.7. Let C be a code with a perspective (θ, δ) which has t components $C_1, C_2, \ldots, C_t$ and a focused splitting $D_1, D_2, \ldots, D_\theta$. For $\alpha \in \Gamma(C)$ we define *an automorphism of the components and split codes* $\alpha \in \gamma(C)$ as an element that satisfies the condition: For all $\alpha' \in \gamma(C)$ and all $c \in C$ we have $\alpha c, \alpha' c \in (C_i \bigcap D_j)$ if and only if $\alpha c = \alpha' c$.

By Lemma 4.6 and Definition 4.3 we know that for $\alpha \in \Gamma(C)$ that α takes components to components, takes split codes to split codes and societies to societies. For a successful decomposition of $\Gamma(C)$ we require the intersection of split codes and components to be taken uniquely to other intersection of split code and component in a unique way.

If we were going to build up a set of elements satisfying Definition 4.7 then we can see that the order in which we pick such elements could affect the structure of the resulting set, especially as the inclusion of one element denies the inclusion of other elements, and as there is no initial non-trivial element we see that we could build up different sets. We

wish to consider maximal sets, that is ones where there are no further valid elements to be included, As there may be many such maximal sets of elements satisfying this criteria, we define $\gamma(C)$ to be one such set, we shall prove shortly that this choice is irrelevant as all such groups will be isomorphic but first we need to show that any choice of maximal set will be a group.

Lemma 4.8. *$\gamma(C)$ is a group.*

Proof As $\gamma(C) \subseteq \Gamma(C)$ and by Lemma 4.5 we know $\Gamma(C)$ is a finite group, then we need only check that $\gamma(C)$ is closed. To prove closure we must check the criteria. For $\alpha_1, \alpha_2, \alpha' \in \gamma(C)$, let $(\alpha_1\alpha_2)c, \alpha' c \in (C_i \bigcap D_j)$ and set $\alpha'' = \alpha_1^{-1}\alpha'$. By Lemma 4.6 we know that $\alpha_2 c, \alpha'' c \in (C_k \bigcap D_l)$ for some k and l and as $\alpha_2, \alpha'' \in \gamma(C)$ we know that $\alpha_2 c = \alpha'' c$ and by multiplying on the left by α_1 thus $\alpha_1\alpha_2 c = \alpha_1\alpha'' c = \alpha' c$. Thus $\gamma(C)$ is closed and thus $\gamma(C)$ is a group. □

Lemma 4.9. *Let $C = (n, M, d)_q$ be a code and let $\gamma(C)$ and $\gamma'(C)$ be two distinct automorphism groups of the split codes and components, then $\gamma(C) \cong \gamma'(C)$.*

Proof We know from Definition 4.3 and from Lemma 4.6 that for an element α in $\Gamma(C)$ that $\alpha C_i = C_k$ for some k and that $\alpha D_j = D_L$ for some L and as such we can see that $\alpha(C_i \bigcap D_j) = C_k \bigcap D_L$. From Definition 4.7 we know that if for α and β in $\gamma(C)$ that if $\alpha(C_i \bigcap D_j) = C_k \bigcap D_L$ and $\beta(C_i \bigcap D_j) = C_k \bigcap D_L$ then we have $\alpha c = \beta c$ for all c in $C_i \bigcap D_j$ and thus we can describe an α in $\gamma(C)$ by how it permutes $(i, j) \in [1, t] \times [1, \theta]$. Thus we can see that there exists a group G such that $\gamma(C) \cong G_\gamma \leq S_t \times S_\theta$.

Now we claim that if there exists an element α in $\Gamma(C)$ then there exists an element $\bar{\alpha}$ in $\gamma(C)$ such that $\alpha(C_i \bigcap D_j) = \bar{\alpha}(C_i \bigcap D_j)$ for all pairs (i, j). First we note that we defined $\gamma(C)$ as elements α in $\Gamma(C)$ such that if an element α' was also in $\gamma(C)$ then we get that $\alpha c, \alpha' c \in (C_i \bigcap D_j)$ implies that $\alpha c = \alpha' c$ for all pairs (i, j). Also note that we require $\gamma(C)$ to be maximal, that is that any element in $\Gamma(C) \setminus \gamma(C)$ added to $\gamma(C)$ would force the condition to break. We can consider $\gamma(C)$ to be constructed by repeatedly adding

elements $\beta \in \Gamma(C)$ until all such attempts would cause the condition to break. Note that if α takes each $(C_i \bigcap D_j)$ to itself then we can take $\bar{\alpha}$ to be the identity, thus we are only considering $\alpha \in \Gamma(C)$ which permute the sets $(C_i \bigcap D_j)$. If an element α is not in $\gamma(C)$ then there must exist an element β_1 such that $\beta_1(C_i \bigcap D_j) = \alpha(C_i \bigcap D_j)$ but there exists $c \in (C_i \bigcap D_j)$ such that $\beta_1 c \neq \alpha c$, moreover we are considering $\alpha(C_i \bigcap D_j) \neq (C_i \bigcap D_j)$. Now considering $\beta_1^{-1}\alpha$ we see that this takes $(C_i \bigcap D_j)$ to itself and also $\beta_1^{-1}\alpha$ does not belong to $\gamma(C)$ because $\gamma(C)$ is a closed group. Now we repeat the process and find β_2 such that $\beta_2(C_i \bigcap D_j) = \beta_1^{-1}\alpha(C_i \bigcap D_j)$ but there exists a $c \in (C_i \bigcap D_j)$ such that $\beta_2 c \neq \beta_1^{-1}\alpha c$ for (i,j) such that $\beta_1^{-1}\alpha(C_i \bigcap D_j) \neq (C_i \bigcap D_j)$ and then we consider the element $\beta_2^{-1}\beta_1^{-1}\alpha$. We repeat this process until we have a sequence of $\beta_1, \beta_2, \ldots, \beta_y$ such that $\beta_y^{-1} \ldots \beta_1^{-1}\alpha$ takes $(C_i \bigcap D_j)$ to itself for all pairs (i,j), we note that the process must end as there are finite number of pairs. If we set $\Delta = \beta_y^{-1} \ldots \beta_1^{-1}\alpha$ and we know that Δ takes $(C_i \bigcap D_j)$ to itself for all pairs (i,j) and thus so does Δ^{-1}, thus $\alpha\Delta^{-1}$ performs the same permutation of the sets of the form $(C_i \bigcap D_j)$ as α, but we can see that

$$\begin{aligned} \alpha\Delta^{-1} &= \alpha(\beta_y^{-1} \ldots \beta_1^{-1}\alpha)^{-1} \\ &= \alpha(\alpha^{-1}\beta_1\beta_2 \ldots \beta_y) \\ &= \beta_1\beta_2 \ldots \beta_y \ \in \gamma(C). \end{aligned}$$

So by setting $\bar{\alpha} = \alpha\Delta^{-1}$ we've shown that for each α in $\Gamma(C)$ there exists $\bar{\alpha}$ in $\gamma(C)$, and thus there exists an element $\hat{\alpha}$ in $G \leq S_t \times S_\theta$.

Thus for $\gamma(C)$ and $\gamma'(C)$ we know that there exists groups G and $G' \leq S_t \times S_\theta$ such that $\gamma(C) \cong G$ and $\gamma'(C) \cong G'$ but as G and G' can be determined by $\Gamma(C)$ we can see that $G \cong G'$ thus $\gamma(C) \cong \gamma'(C)$. □

Thus the idea of the automorphism group of the split codes and components is well defined and we can unambiguously talk about $\gamma(C)$.

To get a better idea of the structure of the automorphisms of focused splittings we

couple these concepts with the automorphisms of the individual components.

Definition 4.10. For a code C with a perspective (θ, δ) consisting of t components $C_1, C_2, \ldots, C_t$, we define $\Gamma'(C_i)$ as the automorphism group of the focused splitting when restricted to the component C_i, that is for $\alpha \in S_M$ we define $\alpha \in \Gamma'(C_i)$ if and only if

(i) $c \in C_i$ if and only if $\alpha c \in C_i$;

(ii) for all $b, c \in C_i$ then $b, c \in D_j$ if and only if $\alpha b, \alpha c \in D_j$;

(iii) for all $b, c \in C_i$ then $c \in S(b)$ if and only if $\alpha c \in S(\alpha b)$.

We note that as there are no interactions between the actions on different components of a code thus we get that $\Gamma'(C_i)$ as described by the conditions above really is the automorphism group of the focused splitting when restricted to the component C_i. We also note that $\alpha \in \Gamma'(C_i)$ takes $C_i \cap D_j$ to itself.

Lemma 4.11. $\Gamma'(C_i)$ *is a group.*

Proof We prove that $\Gamma'(C_i)$ is a group by showing that $\Gamma'(C_i)$ is a subgroup of S_M. Now by definition $\Gamma'(C_i) \subseteq S_M$ and as S_M is finite to show it is a subgroup we must show it is closed. To show $\Gamma'(C_i)$ is closed we must check the criteria:

(i) If $\alpha_1, \alpha_2 \in \Gamma'(C_i)$ then for all $c \in C_i$ we know $\alpha_1 c \in C_i$ and $\alpha_2 c \in C_i$ and if $\alpha_2 c \in C_i$ thus $\alpha_1 \alpha_2 c \in C_i$.

(ii) If $\alpha_1, \alpha_2 \in \Gamma'(C_i)$ thus for all $b, c \in C_i$ we have $b, c \in D_j$ if and only if $\alpha_1 b, \alpha_1 c \in D_j$ and we have $b, c \in D_j$ if and only if $\alpha_2 b, \alpha_2 c \in D_j$. Thus $b, c \in D_j$ if and only if $\alpha_2 b, \alpha_2 c \in D_j$ which happens if and only if $\alpha_1 \alpha_2 b, \alpha_1 \alpha_2 c \in D_j$.

(iii) If $\alpha_1, \alpha_2 \in \Gamma'(C_i)$ thus for all $b, c \in C_i$ we have $c \in S(b)$ if and only if $\alpha_1 c \in S(\alpha_1 b)$ and we have $c \in S(b)$ if and only if $\alpha_2 c \in S(\alpha_2 b)$. Thus $c \in S(b)$ if and only if $\alpha_2 c \in S(\alpha_2 b)$ which happens if and only if $\alpha_1 \alpha_2 c \in S(\alpha_1 \alpha_2 b)$.

Thus $\Gamma'(C_i)$ is closed and thus $\Gamma'(C_i)$ is a group. $\square$

We also require a slighter more rigorous concept of an individual components automorphism, for our decomposition of the automorphism group.

Definition 4.12. For a code C with a perspective (θ, δ) consisting of t components $C_1, C_2, \ldots, C_t$, we define $\hat{\Gamma}(C_i)$, the *restricted automorphism group of the focused splitting for the component* C_i as follows, $\alpha \in \hat{\Gamma}(C_i)$ if and only if $\alpha \in \Gamma'(C_i)$ and for all $c \in C \setminus C_i$ we have $\alpha c = c$.

That is we wish $\hat{\Gamma}(C_i)$ to be the identity group when restricted away from C_i.

Lemma 4.13. $\hat{\Gamma}(C_i)$ *is a group.*

Proof By definition $\hat{\Gamma}(C_i) \subseteq \Gamma'(C_i)$ so to prove that $\hat{\Gamma}(C_i)$ is a group we must show that it is closed and has inverses. For $\alpha, \beta \in \hat{\Gamma}(C_i)$ we know $\alpha\beta \in \Gamma'(C_i)$, and so for all $c \in C \setminus C_i$ we get

$$\begin{aligned} (\alpha\beta)c &= \alpha(\beta c) \\ &= \alpha c \\ &= c \end{aligned}$$

as $\alpha c = c$ for all $c \in C \setminus C_i$ and for all $\alpha \in \hat{\Gamma}(C_i)$, thus $(\alpha\beta) \in \hat{\Gamma}(C_i)$. For $\alpha \in \hat{\Gamma}(C_i)$ we know that $\alpha^{-1} \in \Gamma'(C_i)$ and for $c \in C \setminus C_i$ we have

$$\begin{aligned} \alpha^{-1}c &= \alpha^{-1}(\alpha c) \\ &= (\alpha^{-1}\alpha)c \\ &= c \end{aligned}$$

thus $\alpha^{-1} \in \hat{\Gamma}(C_i)$ and thus $\hat{\Gamma}(C_i)$ is a group. $\square$

Before we can prove a theorem on the structure of the automorphism group we must first prove a couple of useful lemmata.

Lemma 4.14. *The intersection of all the automorphism groups of the focused splitting when restricted to a particular components is a subgroup of the automorphisms of the focused splitting. That is*

$$\bigcap_{i=1}^{t} \Gamma'(C_i) \leq \Gamma(C).$$

Proof As each $\Gamma'(C_i)$ is a subgroup of S_M, then clearly $\bigcap_{i=1}^{t} \Gamma'(C_i)$ is a subgroup of S_M, so to prove the lemma we need to show that every element of $\bigcap_{i=1}^{t} \Gamma'(C_i)$ is also an element of $\Gamma(C)$. If $\alpha \in \bigcap_{i=1}^{t} \Gamma'(C_i)$ then:

(i) $c \in C_i$ if and only if $\alpha c \in C_i$;

(ii) for all $b, c \in C_i$ then $b, c \in D_j$ if and only if $\alpha b, \alpha c \in D_j$ for some j;

(iii) for all $b, c \in C_i$ then $c \in S(b)$ if and only if $\alpha c \in S(\alpha b)$;

are true for all $i \in [1, t]$. As $C = \bigcup_{i=1}^{t} C_i$ then we can conclude that for $\alpha \in \bigcap_{i=1}^{t} \Gamma'(C_i)$

(i) for all $b, c \in C$ then $b, c \in D_j$ if and only if $\alpha b, \alpha c \in D_j$ for some j;

(ii) for all $b, c \in C$ then $c \in S(b)$ if and only if $\alpha c \in S(\alpha b)$;

which are restricted variant of the criteria for α to be an element of $\Gamma(C)$. Thus $\alpha \in \Gamma(C)$ and thus we prove the lemma. □

Lemma 4.15. *The intersection of all the automorphism groups of the focused splitting when restricted to a particular components is isomorphic to the product of all the restricted automorphisms of the focused splitting when restricted to a particular component; that is*

$$\bigcap_{i=1}^{t} \Gamma'(C_i) \cong \prod_{i=1}^{t} \hat{\Gamma}(C_i).$$

Proof Let $\alpha = (\alpha_1, \alpha_2, \ldots \alpha_t) \in \prod_{i=1}^{t} \hat{\Gamma}(C_i)$. We prove the lemma by construction of the isomorphism

$$\pi : \prod_{i=1}^{t} \hat{\Gamma}(C_i) \rightarrow \bigcap_{i=1}^{t} \Gamma'(C_i)$$

as:

$$\pi(\alpha_1, \alpha_2, \ldots, \alpha_t) = \alpha_1 \alpha_2 \ldots \alpha_t.$$

Let $c \in C_i$, then as $c \in C \setminus C_j$ for all $j \neq i$ then $\alpha_j c = c$ for all $j \neq i$ and as $\alpha_i c \in C_i$ thus we can show

$$\begin{aligned}(\alpha_1 \alpha_2 \ldots \alpha_t)c &= (\alpha_1 \alpha_2 \ldots \alpha_{i-1})\alpha_i(\alpha_{i+1} \ldots \alpha_t c) \\ &= (\alpha_1 \alpha_2 \ldots \alpha_{i-1})\alpha_i c \\ &= \alpha_i c.\end{aligned}$$

Thus as $\Gamma'(C_i)$ is defined only in terms of what happens to elements $c \in C_i$ thus $\alpha_1 \alpha_2 \ldots \alpha_t \in \Gamma'(C_i)$ and as this is true for all i then $\pi(\alpha) = \alpha_1 \alpha_2 \ldots \alpha_t \in \bigcap_{i=1}^{t} \Gamma'(C_i)$.

For $\beta \in \bigcap_{i=1}^{t} \Gamma'(C_i)$ we know that $\beta \in \Gamma'(C_i)$ for all i and for all i we choose $\alpha_i \in \hat{\Gamma}(C)$ such that $\alpha_i c = \beta c$ for all $c \in C_i$, and thus $\alpha_i c = c$ for all $c \in C \setminus C_i$. Thus for $c \in C_i$ we can show that

$$\begin{aligned}(\alpha_1 \alpha_2 \ldots \alpha_t)c &= (\alpha_1 \alpha_2 \ldots \alpha_{i-1})\alpha_i(\alpha_{i+1} \ldots \alpha_t c) \\ &= (\alpha_1 \alpha_2 \ldots \alpha_{i-1})\alpha_i c \\ &= \alpha_i c \\ &= \beta c\end{aligned}$$

and as this is true for all i, thus $(\alpha_1 \alpha_2 \ldots \alpha_t)c = \beta c$ for all $c \in C$, thus $(\alpha_1 \alpha_2 \ldots \alpha_t) = \beta$. Thus β can be described as an element in the product of the restricted automorphism groups for the components, that is $\pi^{-1}(\beta) \in \prod_{i=1}^{t} \hat{\Gamma}(C_i)$. Thus π is one to one.

Now we need to prove that π is a homomorphism. Let $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_t)$ and let $\beta = (\beta_1, \beta_2, \ldots, \beta_t)$ both in $\prod_{i=1}^{t} \hat{\Gamma}(C_i)$. Thus we can see that

$$\begin{aligned} \pi(\alpha\beta) &= \pi(\alpha_1\beta_1, \alpha_2\beta_2, \ldots, \alpha_t\beta_t) \\ &= \alpha_1\beta_1\alpha_2\beta_2 \ldots \alpha_t\beta_t \end{aligned}$$

and

$$\begin{aligned} \pi(\alpha)\pi(\beta) &= \pi(\alpha_1, \alpha_2, \ldots, \alpha_t)\pi(\beta_1, \beta_2, \ldots \beta_t) \\ &= \alpha_1\alpha_2 \ldots \alpha_t\beta_1\beta_2 \ldots \beta_t \end{aligned}$$

And thus for $c \in C_i$ we can see that as $(\alpha_i\beta_i c) \in C_i$ we get

$$\begin{aligned} \alpha_1\beta_1\alpha_2\beta_2 \ldots \alpha_t\beta_t c &= (\alpha_1\beta_1\alpha_2\beta_2 \ldots \alpha_{i-1}\beta_{i-1})\alpha_i\beta_i(\alpha_{i+1}\beta_{i+1}\alpha_{i+2}\beta_{i+2} \ldots \alpha_t\beta_t)c \\ &= (\alpha_1\beta_1\alpha_2\beta_2 \ldots \alpha_{i-1}\beta_{i-1})\alpha_i\beta_i c \\ &= \alpha_i\beta_i c \end{aligned}$$

and as $\beta_i c \in C_i$ we get

$$\begin{aligned} \alpha_1\alpha_2 \ldots \alpha_t\beta_1\beta_2 \ldots \beta_t c &= (\alpha_1\alpha_2 \ldots \alpha_t\beta_1\beta_2 \ldots \beta_{i-1})\beta_i(\beta_{i+1} \ldots \beta_t)c \\ &= (\alpha_1\alpha_2 \ldots \alpha_t)(\beta_1\beta_2 \ldots \beta_{i-1})\beta_i c \\ &= (\alpha_1\alpha_2 \ldots \alpha_t)\beta_i c \\ &= (\alpha_1\alpha_2 \ldots \alpha_{i-1})\alpha_i(\alpha_{i+1}\alpha_{i+2} \ldots \alpha_t)\beta_i c \\ &= (\alpha_1\alpha_2 \ldots \alpha_{i-1})\alpha_i\beta_i c \\ &= \alpha_i\beta_i c \end{aligned}$$

and thus for $c \in C_i$ we have $\pi(\alpha\beta)c = \pi(\alpha)\pi(\beta)c$ and as this is true for all i we have $\pi(\alpha\beta) = \pi(\alpha)\pi(\beta)$ and thus π is a homomorphism and moreover an isomorphism.

Thus

$$\bigcap_{i=1}^{t} \Gamma'(C_i) \cong \prod_{i=1}^{t} \hat{\Gamma}(C_i).$$

□

Definition 4.16. For a code C with a focused splitting we define the *automorphism group of the split codes* $\lambda(C)$ as follows; $\alpha \in \lambda(C)$ if and only if

(i) $\alpha \in \gamma(C)$;

(ii) for all $\alpha' \in \lambda(C)$ and for all $c \in C$ we have $\alpha c, \alpha' c \in D_i$ if and only if $\alpha c = \alpha' c$.

That is $\alpha \in \lambda(C)$ takes words to specific words in each split code. Moreover as we know $\lambda(C) \subseteq \gamma(C)$ and $\gamma(C)$ is really just a class of subgroups all isomorphic to each other, we can choose $\lambda(C)$ such that $\alpha C_i = C_i$ for all i, this can be done because if $\alpha D_j = D_L$ and we know that $\alpha(C_i \bigcap D_j) = C_k \bigcap D_L$ and as α takes D_j to D_L uniquely thus there will exist a unique way that forces $C_k = C_i$. Essentially we do this such that we choose $\lambda(C)$ to preserve the components of the code.

Lemma 4.17. *$\lambda(C)$ is a group*

Proof As $\lambda(C) \subseteq \gamma(C)$ then to prove $\lambda(C)$ is a group we need only show closure and inverses exist. To prove closure:

(i) Let $\alpha, \beta \in \lambda(C)$, thus $\alpha, \beta \in \gamma(C)$ thus $\alpha\beta \in \gamma(C)$.

(ii) Let $\alpha, \beta, \alpha' \in \lambda(C)$ and let $\alpha\beta c, \alpha' c \in D_i$ and set $\alpha'' = \alpha^{-1}\alpha'$ and by the fact that $\alpha^{-1} \in \gamma(C)$ we know $\alpha'' c, \beta c \in D_j$ for some j and thus $\alpha'' c = \beta c$ and thus $\alpha\alpha'' c = \alpha\alpha^{-1}\alpha' c = \alpha' c = \alpha\beta c$.

To prove inverses exists:

(i) Let $\alpha \in \lambda(C)$, thus $\alpha \in \gamma(C)$ and thus $\alpha^{-1} \in \gamma(C)$.

(ii) Let $\alpha, \alpha^{-1} \in \lambda(C)$ and let $\alpha^{-1}c, \alpha' c \in D_i$ and thus $\alpha\alpha^{-1}c, \alpha\alpha' c \in D_j$ for some j, which is the same as saying $c, \alpha' c \in D_j$ thus $c = \alpha\alpha' c$ and applying α^{-1} to both sides gives $\alpha^{-1}c = \alpha' c$.

Thus $\lambda(C)$ is a group. □

Definition 4.18. Let C be a code consisting of t components with a focused splitting, then we say *components C_i and C_j are in the same component class* if there exists $\alpha \in \Gamma(C)$ such that $\alpha C_i = C_j$.

As groups are closed we can see that the component classes are closed, since if C_i and C_j are in the same class due to α, and C_j and C_k are due to β, then $\beta\alpha C_i = \beta C_j = C_k$.

Lemma 4.19. *Let C be a code with a focused splitting, and let $\hat{t}$ be the number of component classes in C, number the classes from 1 to $\hat{t}$ and let $\rho(i)$ be the number of components in the i^{th} class. Then*

$$\gamma(C) \cong \lambda(C) \times \left(\prod_{i=1}^{\hat{t}} S_{\rho(i)} \right).$$

That is the automorphism group of the components and split codes can be written as the direct product of automorphism group of the split codes and the product of the symmetric group for the size of each component class.

Proof First we prove that $\alpha \in \gamma(C)$ can be represented by a pair of elements

$$(\beta, \phi) \in \lambda(C) \times (\prod_{i=1}^{\hat{t}} S_{\rho(i)}).$$

For $\alpha \in \gamma(C)$, we know that α takes components to components and split codes to split codes. As α takes split codes to split codes then α permutes the θ split codes and so can be thought of as a permutation of 1 to θ. For $\beta \in \lambda(C)$, we know that β takes split codes to split codes, and more over if $\beta c, \beta' c \in D_i$ we know $\beta c = \beta' c$ for all $\beta' \in \lambda(C)$, that is if β takes a word in D_i to a word in D_j, then it always takes it to the same

word, thus $\beta \in \lambda(C)$ can be thought of as a permutation of the θ split codes and thus a permutation of 1 to θ, moreover for each permutation of 1 to θ there will be precisely one element in $\lambda(C)$ which performs that permutation on the split codes. Note for every $\alpha \in \gamma(C)$ there will exist $\beta \in \lambda(C)$ such that they both admit the same permutation on the split codes. For $\alpha \in \gamma(C)$ choose $\beta \in \lambda(C)$ such that they both admit the same permutation on the split codes. Consider $\beta^{-1}\alpha$, as $\beta, \alpha \in \gamma(C)$ thus $\beta^{-1}\alpha \in \gamma(C)$ and as β and α admit the same permutation on the split codes then $\beta^{-1}\alpha$ will take split codes to themselves, thus $\beta^{-1}\alpha$ will only permute the components. Moreover let $\alpha, \alpha' \in \gamma(C)$ and let β, β' be as defined above, then $\beta^{-1}\alpha$ and $\beta'^{-1}\alpha'$ each take split codes to themselves, so if $\beta^{-1}\alpha c, \beta'^{-1}\alpha' c \in C_i$ we know $\beta^{-1}\alpha c, \beta'^{-1}\alpha' c \in D_j$ for $c \in D_j$ and thus $\beta^{-1}\alpha c, \beta'^{-1}\alpha' c \in (C_i \bigcap D_j)$ and as $\beta^{-1}\alpha, \beta'^{-1}\alpha \in \gamma(C)$ thus $\beta^{-1}\alpha c = \beta'^{-1}\alpha' c$. So elements $\beta^{-1}\alpha$ takes a word in C_i to a specific word in C_j for some i and j. As an element $\beta^{-1}\alpha$ permutes the components, and $\beta^{-1}\alpha \in \gamma(C) \leq \Gamma(C)$ thus if $\beta^{-1}\alpha C_i = C_j$ then C_i and C_j must be in the same component class. So $\beta^{-1}\alpha$ will only permute components in the same component class, and by Definition 4.3 an element of $\Gamma(C)$ only has to preserve the split codes and the societies, and thus any permutation of the components in a component class is admissible. Thus when restricted to a given component class, say component class i, then $\beta^{-1}\alpha = \phi \in S_{\rho(i)}$ where $S_{\rho(i)}$ acts on the components of C in the natural manner, taking component C_i to C_j whenever $\phi(i) = j$. Now as this must be true for each component class, of which there are $\hat{t}$ and as the permutations of each component class will not affect each other, thus we know

$$\beta^{-1}\alpha \cong \phi \in \prod_{i=1}^{\hat{t}} S_{\rho(i)}$$

and thus by left multiplication of β we can show

$$\alpha \in \gamma(C) = \beta\phi \cong (\beta, \phi) \in \lambda(C) \times \left(\prod_{i=1}^{\hat{t}} S_{\rho(i)} \right)$$

Secondly we must prove that a mapping $\pi(\beta, \phi) = \beta\phi$ is a homomorphism. That is we must show that $\pi((\beta_1, \phi_1)(\beta_2, \phi_2)) = \pi(\beta_1, \phi_1)\pi(\beta_2, \phi_2)$ and to prove this we first show that $\beta\phi = \phi\beta$ for $\beta \in \lambda(C)$ and $\phi \in \prod_{i=1}^{\hat{t}} S_{\rho(i)}$.

We know that $\alpha(C_i \bigcap D_j) = C_k \bigcap D_L$ for some k and L, and we know from above that $\alpha = \beta\phi$ for $\beta \in \lambda(C)$ and $\phi \in \prod_{i=1}^{\hat{t}} S_{\rho(i)}$ and moreover as ϕ only permutes the components whilst preserving the split codes and β permutes the split codes whilst preserving the components, thus

$$\begin{aligned} \phi(C_i \bigcap D_j) &= (C_k \bigcap D_j) \\ \beta(C_i \bigcap D_j) &= (C_i \bigcap D_L) \end{aligned}$$

for all i and j and thus

$$\begin{aligned} \beta\phi(C_i \bigcap D_j) &= \beta(C_k \bigcap D_j) \\ &= (C_k \bigcap D_L) \end{aligned}$$

and

$$\begin{aligned} \phi\beta(C_i \bigcap D_j) &= \phi(C_i \bigcap D_L) \\ &= (C_k \bigcap D_L) \end{aligned}$$

thus $\beta\phi = \phi\beta$ for all β and ϕ. Thus

$$\pi((\beta_1, \phi_1)(\beta_2, \phi_2)) = \pi(\beta_1\beta_2, \phi_1\phi_2) = \beta_1\beta_2\phi_1\phi_2 = \beta_1\phi_1\beta_2\phi_2 = \pi(\beta_1, \phi_1)\pi(\beta_2, \phi_2)$$

and thus π is a homomorphism, and noting that every element in $\gamma(C)$ can be described by an element in $\lambda(C) \times (\prod_{i=1}^{\hat{t}} S_{\rho(i)})$ and that every element in $\lambda(C) \times (\prod_{i=1}^{\hat{t}} S_{\rho(i)})$ describes

an element in $\gamma(C)$ we can see that

$$\gamma(C) \cong \lambda(C) \times \left(\prod_{i=1}^{\hat{t}} S_{\rho(i)} \right).$$

□

For use in the following section we briefly recap the concept of semi-direct product.

Definition 4.20. For groups N and H and a homomorphism $\psi : H \to Aut(N)$, we denote the automorphism $\psi(h)$, for $h \in H$, acting on the element $n \in N$ by $\psi_h(n)$ and we define the *semi-direct product* $G = N \rtimes_\psi H = \{(n, h) : n \in N, h \in H\}$ where the multiplication is defined by

$$(n_1, h_1)(n_2, h_2) = (n_1 \psi_{h_1}(n_2), h_1 h_2).$$

Theorem 4.21. *Let C be a code with a perspective (θ, δ) consisting of components $C_1, C_2, \ldots, C_t$ and a focused splitting $D_1, D_2, \ldots, D_\theta$. Let $\psi : (\bigcap_{i=1}^{t} \Gamma'(C_i)) \to Aut(\gamma(C))$ be defined as $\psi_\beta(\gamma) = \beta\gamma\beta^{-1}$, that is we conjugate γ by β, and this is well defined because β and γ both belong to S_M. Then*

$$\Gamma(C) \cong \gamma(C) \rtimes_\psi \left(\bigcap_{i=1}^{t} \Gamma'(C_i) \right).$$

Proof Let $\alpha = (\gamma, \beta) \in \gamma(C) \times (\bigcap_{i=1}^{t} \Gamma'(C_i))$. We shall construct the isomorphism

$$\pi : \gamma(C) \rtimes_\psi (\bigcap_{i=1}^{t} \Gamma'(C_i)) \to \Gamma(C)$$

as

$$\pi(\alpha) = \pi(\gamma, \beta) = \gamma\beta$$

where γ and β are treated as members of S_M.

We start by showing that π is a homomorphism:

$$\begin{aligned}
\pi((\gamma_1,\beta_1)(\gamma_2,\beta_2)) &= \pi(\gamma_1\psi_{\beta_1}(\gamma_2),\beta_1\beta_2)\\
&= \pi(\gamma_1\beta_1\gamma_2\beta_1^{-1},\beta_1\beta_2)\\
&= \gamma_1\beta_1\gamma_2\beta_1^{-1}\beta_1\beta_2\\
&= \gamma_1\beta_1\gamma_2\beta_2\\
&= (\gamma_1\beta_1)(\gamma_2\beta_2)\\
&= \pi(\gamma_1,\beta_1)\pi(\gamma_2,\beta_2)
\end{aligned}$$

and thus as the groups are finite we just have to show that the groups are the same size.

By Definition 4.7 we know that $\gamma(C) \leq \Gamma(C)$ and by Lemma 4.14 we also know that $\bigcap_{i=1}^{t}\Gamma'(C_i) \leq \Gamma(C)$, and thus as $\Gamma(C)$ is closed we know that π maps into $\Gamma(C)$. Thus we can see that

$$\pi(\gamma(C) \rtimes_\psi (\bigcap_{i=1}^{t}\Gamma'(C_i))) \leq \Gamma(C),$$

that is the product of $\gamma(C)$ and $\bigcap_{i=1}^{t}\Gamma'(C_i)$ is isomorphic to a subgroup of $\Gamma(C)$. To complete the proof we must show that an element of $\Gamma(C)$ can be represented as the product of an element from $\gamma(C)$ with an element from $\bigcap_{i=1}^{t}\Gamma'(C_i)$, and thus they are the same size and thus isomorphic.

By Lemma 4.6, $\alpha \in \Gamma(C)$ takes components of C to distinct components of C whilst also taking split codes to split codes. Let $D_{i,j} = D_i \bigcap C_j$ and let w be the number of non-empty such sets. Then α will permute the w sets $D_{i,j}$ and so if we label the sets $D_{i,j}$ from 1 to w then there exists an element $\sigma(\alpha) \in S_w$ which will permute the numbers 1 to w in the same manner. Similarly for each element $\gamma \in \gamma(C)$ we can think of γ as permuting $[1, w]$ and thus there is an element $\sigma(\gamma) \in S_w$ which permutes the numbers 1 to w in the same manner. Note that although there may be many $\alpha \in \Gamma(C)$ such that $\sigma(\alpha)$ are the same, for distinct $\gamma \in \gamma(C)$ then $\sigma(\gamma)$ will be distinct, also note that if α permutes the

components and split codes in a certain way, then that permutation is a symmetry of the components and thus there is an element $\gamma \in \gamma(C)$ which will permute the components in the same way. For a given α we choose γ such that $\sigma(\gamma) = \sigma(\alpha)$ and then we consider the element $\gamma^{-1}\alpha$. As $\gamma(C) \leq \Gamma(C)$ and $\Gamma(C)$ is closed thus $\gamma^{-1}\alpha \in \Gamma(C)$, and as α takes C_i to C_j and γ^{-1} takes C_j to C_i then $\gamma^{-1}\alpha$ will take each component to itself and each split-code to itself. Thus for the element $\gamma^{-1}\alpha$ we will have $c \in C_i$ if and only if $\gamma^{-1}\alpha c \in C_i$, and as $\gamma^{-1}\alpha \in \Gamma(C)$ then for all $b, c \in C_i$ then $b, c \in D_j$ if and only if $\gamma^{-1}\alpha b, \gamma^{-1}\alpha c \in D_j$ and for all $b, c \in C_i$ then $c \in S(b)$ if and only if $\gamma^{-1}\alpha c \in S(\gamma^{-1}\alpha b)$. Thus $\gamma^{-1}\alpha \in \Gamma'(C_i)$, and as this is true for all i thus $\gamma^{-1}\alpha \in \bigcap_{i=1}^{t} \Gamma'(C_i)$. Thus there exists $\beta \in \bigcap_{i=1}^{t} \Gamma'(C_i)$ such that

$$\begin{aligned} \gamma^{-1}\alpha &= \beta \\ \implies \quad \alpha &= \gamma\beta \end{aligned}$$

$$\implies \quad \alpha \in \pi\left(\gamma(C) \rtimes_{\psi} \left(\bigcap_{i=1}^{t} \Gamma'(C_i)\right)\right)$$

Thus we have shown for $\alpha \in \Gamma(C)$ that $\alpha \cong (\gamma, \beta) \in \gamma(C) \rtimes_{\psi} (\bigcap_{i=1}^{t} \Gamma'(C_i))$ and thus we have proved that $\Gamma(C) \cong \gamma(C) \rtimes_{\psi} (\bigcap_{i=1}^{t} \Gamma'(C_i))$. □

Corollary 4.22. *Let C be a code with t components and focused splitting, let $\hat{t}$ be the number of component classes in C, number the classes from 1 to $\hat{t}$ and let $\rho(i)$ be the number of components in the i^{th} class. Then*

$$\Gamma(C) \cong \left(\lambda(C) \times \left(\prod_{i=1}^{\hat{t}} S_{\rho(i)}\right)\right) \rtimes_{\psi} \left(\prod_{i=1}^{t} \hat{\Gamma}(C_i)\right)$$

Proof By Theorem 4.21 we know

$$\Gamma(C) \cong \gamma(C) \rtimes_{\psi} (\bigcap_{i=1}^{t} \Gamma'(C_i))$$

and by Lemma 4.15 we know

$$\bigcap_{i=1}^{t} \Gamma'(C_i) \cong \prod_{i=1}^{t} \hat{\Gamma}(C_i).$$

Thus

$$\Gamma(C) \cong \gamma(C) \rtimes_{\psi} (\prod_{i=1}^{t} \hat{\Gamma}(C_i))$$

and by Lemma 4.19 we get

$$\Gamma(C) \cong \big(\lambda(C) \times (\prod_{i=1}^{\hat{t}} S_{\rho(i)})\big) \rtimes_{\psi} (\prod_{i=1}^{t} \hat{\Gamma}(C_i)).$$

□

4.4 Specific Examples of Automorphism Groups

Now we are going to calculate $\Gamma(C)$ for various values of θ and δ and for some constructions.

Theorem 4.23. *For a code C with a perspective with width of $\theta = 1$ and a focused splitting then $\Gamma(C) = S_M$*

Proof As $\theta = 1$ then C splits into one code D_1 and so $D_1 = C$. For $\alpha \in S_M$ then α will always satisfy; $b, c \in D_i$ if and only if $\alpha b, \alpha c \in D_j$ for some j. Similarly as $\theta = 1$ then $S(c) = c$ for all $c \in C$ and so $c \in S(b)$ if and only if $\alpha c \in S(\alpha b)$. Thus $\alpha \in \Gamma(C)$ for all $\alpha \in S_M$ □

Theorem 4.24. *Let C be a code with a perspective with width of $\theta = 2$, and a focused*

splitting. Let t be the number of components, where $t = t_1 + t_2 + t_3$ and t_1 is the number of components consisting of 2 elements, t_2 is the number of components consisting of 1 element which is in D_1 in the focused splitting, and t_3 is the number of components consisting of 1 element which is in D_2 in the focused splitting. Then

(i) *If $t_2 = t_3$ then $\Gamma(C) \cong \mathbb{Z}_2 \times S_{t_1} \times S_{t_2} \times S_{t_3}$.*

(ii) *If $t_2 \neq t_3$ then $\Gamma(C) \cong S_{t_1} \times S_{t_2} \times S_{t_3}$.*

Proof First note that as $\theta = 2$ then $|D_i \bigcap C_j\| = 1$ for all i and j and thus $\hat{\Gamma}(C_j)$ is the identity group for all j, thus by Corollary 4.22 we get

$$\Gamma(C) \cong \gamma(C) \cong \lambda(C) \times (\prod_{i=1}^{\hat{t}} S_{\rho(i)}).$$

The group $\lambda(C)$ permutes the split codes, and as $\theta = 2$ is thus a subgroup of $\mathbb{Z}_2$, so either $\mathbb{Z}_2$ or the identity group. We can see that $\lambda(C) \cong \mathbb{Z}_2$ if and only if the number of words in the split codes is the same, that is $|D_1| = |D_2|$ and as $|D_1| = t_1 + t_2$ and $|D_2| = t_1 + t_3$ then this happens precisely when $t_2 = t_3$ and otherwise $\lambda(C)$ is the identity group. We can easily see that each of the t_1 components will consist of 2 words and thus as there is exactly one such possibly way this could be constructed then these t_1 components will constitute a component class. Similarly the t_2 components will constitute a component class as will the t_3 components. Thus there will be 3 component classes with t_1, t_2 and t_3 words respectively. Thus $\Gamma(C) \cong \lambda(C) \times S_{t_1} \times S_{t_2} \times S_{t_3}$, and so:

(i) If $t_2 = t_3$ then $\Gamma(C) \cong \lambda(C) \times S_{t_1} \times S_{t_2} \times S_{t_3} \cong \mathbb{Z}_2 \times S_{t_1} \times S_{t_2} \times S_{t_3}$;

(ii) If $t_2 \neq t_3$ then $\Gamma(C) \cong \lambda(C) \times S_{t_1} \times S_{t_2} \times S_{t_3} \cong S_{t_1} \times S_{t_2} \times S_{t_3}$.

Which completes the proof. □

Before we deal with the full case for a perspective with a width of $\theta = 3$ we must calculate the restricted automorphism group of the focused splitting for different components.

Lemma 4.25. *Let C be a code with a perspective with width of $\theta = 3$, and a focused splitting. By Theorem 2.21 we know each component has either an unbalanced perspective or has a balanced perspective and* $0 \mod 3$ *words in each component. Then*

(i) *if C_i has a balanced perspective then $\hat{\Gamma}(C_i) \cong \mathbb{Z}_{\frac{M'}{3}}$ where $M' = |C_i|$;*

(ii) *if C_i has an unbalanced perspective then $\hat{\Gamma}(C_i) \cong \mathbb{Z}_1$.*

Proof If C_i has a balanced perspective then we know we can label the words of C_i from 0 to $M' - 1$ such that a word j is neighbours with words $j - 1$ and $j + 1$ and such that 0 and $M' - 1$ are neighbours and thus we can describe the focused splitting by what that word is $\mod 3$. Thus for an automorphism $\alpha \in \hat{\Gamma}(C_i)$ we require the focused splitting to be retained that is we know that $j \mod 3 = (\alpha j) \mod 3$ for all j and as we require the societies to be retained that is we know that $(\alpha j) + 1 = \alpha(j + 1)$ and $(\alpha j) - 1 = \alpha(j - 1)$ for all j. We claim that thus α is equivalent to adding a multiple of 3 to the labelling of each word. Fixing a specific j we know that $j \mod 3 = (\alpha j) \mod 3$ and thus we can see that $\alpha j = j + 3t \mod M'$ where t is between 0 and $\frac{M'}{3} - 1$, now as $(\alpha j) + 1 = \alpha(j + 1)$ we can see that $\alpha(j + 1) = (\alpha j) + 1 = j + 1 + 3t \mod M'$ and be repeated application of this we can see that $\alpha j = j + 3t \mod M'$ for all j and thus α is equivalent to adding 3 times a number between 0 and $\frac{M'}{3} - 1$ and thus as addition is associative and commutative we can see that $\hat{\Gamma}(C_i) \cong \mathbb{Z}_{\frac{M'}{3}}$.

If C_i has an unbalanced perspective then we know that there is either only 1 word in the component or that there exists precisely 2 words with exactly 2 neighbours, and all the other words will have 3 neighbours. In the first case it is clear that $\hat{\Gamma}(C_i) \cong \mathbb{Z}_1$. In the second case we will be employing the same labelling as above and thus the words with exactly 2 neighbours will be labelled 0 and $M' - 1$, and the conditions $j \mod 3 = (\alpha j) \mod 3$ for all j, $(\alpha j) + 1 = \alpha(j + 1)$ and $(\alpha j) - 1 = \alpha(j - 1)$ for all j will still all hold. We start by noticing that any automorphism $\alpha \in \hat{\Gamma}(C_i)$ will have to take the set $\{0, M' - 1\}$ to itself, as these are the only words with 2 neighbours, we shall show that this can only

happen by taking each word to itself. First note that if 0 and $M'-1$ are in the same split code, that is $0 \mod 3 = M'-1 \mod 3$ then their respective neighbours will be in distinct split codes, that is $1 \mod 3 \neq M'-2 \mod 3$ and thus if α is to preserve membership of split codes and membership of societies then α will have to take 0 to 0 and $M'-1$ to itself. If 0 and $M'-1$ are in different split codes then α must take 0 to itself and $M'-1$ to itself. By conclusion if 0 goes to 0 then 1 goes to 1 and so on, as C_i is finite each word is taken to itself and thus $\hat{\Gamma}(C_i) \cong \mathbb{Z}_1$. □

Thus from this information and by Corollary 4.22 we can calculate $\Gamma(C)$ for a code C with a perspective with width of $\theta = 3$ and a focused splitting, by calculating $\lambda(C)$ and by observing how many different components there are and how many component classes there are. Note that two components with balanced perspectives will be in the same component class if and only if they contain the same number of words, and two components with unbalanced perspectives will be in the same component class if and only if they contain the same number of words and the split codes which the words with only 2 neighbours are in for one component are the same as the split codes which the words with only 2 neighbours are in for the other component. We shall now show a specific case for a code with a perspective with width of $\theta = 3$.

Theorem 4.26. *Let C be a code with a balanced perspective with width of $\theta = 3$ and a focused splitting and consisting of exactly one component. Then $\Gamma(C) \cong D_M$ where $M = |C|$.*

Proof By Lemma 4.25 we know that $\hat{\Gamma}(C) \cong \mathbb{Z}_{\frac{M}{3}}$. By Corollary 4.22 we know that

$$\Gamma(C) \cong \left(\lambda(C) \times (\prod_{i=1}^{\hat{t}} S_{\rho(i)})\right) \rtimes_{\psi} (\prod_{i=1}^{t} \hat{\Gamma}(C_i)).$$

Now as there is only one component, thus one component class consisting of one compo-

nent and we know $\hat{\Gamma}(C)$ this reduces to

$$\Gamma(C) \cong \lambda(C) \rtimes_\psi \mathbb{Z}_{\frac{M}{3}}.$$

As C has a balanced perspective thus each split code will consist of the same number of words and thus $\lambda(C)$ will be able to permute the split codes between the 3 of them in any permutation, thus $\lambda(C) \cong S_3$. Thus

$$\Gamma(C) \cong \lambda(C) \rtimes_\psi \mathbb{Z}_{\frac{M}{3}} \cong S_3 \rtimes_\psi \mathbb{Z}_{\frac{M}{3}}.$$

We know show that $S_3 \rtimes_\psi \mathbb{Z}_{\frac{M}{3}} \cong D_M$. Let $M' = \frac{M}{3}$ which we know to be an integer and recall that

$$\begin{aligned} S_3 &= \{\alpha^i\beta_j : \alpha^2 = \beta^3 = 1, \alpha\beta = \beta^{-1}\alpha\} \\ Z_{M'} &= \{\gamma^i : \gamma^{M'} = 1\} \\ D_{3M'} &= \{\sigma^i\omega^j : \sigma^2 = \omega^{3M'} = 1, \sigma\omega = \omega^{-1}\sigma\} \end{aligned}$$

And we define the homomorphism $\pi : S_3 \rtimes_\psi \mathbb{Z}_{M'} \to D_{3M'}$ as $\pi(\alpha^i\beta^j, \gamma^k) = \sigma^i\omega^{jM'+k}$. As such we can consider $\beta = \omega^n$ and $\gamma = \omega$ and thus we can give meaning to the conjugation of ψ and we can also equate $\alpha = \sigma$. Thus we can redefine the groups as

$$\begin{aligned} S_3 &= \{\sigma^i\omega^{jM'} : \sigma^2 = \omega^{3M'} = 1, \sigma\omega = \omega^{-1}\sigma\} \\ Z_{M'} &= \{\omega^{3i} : \omega^{3M'} = 1\} \\ D_{3M'} &= \{\sigma^i\omega^j : \sigma^2 = \omega^{3M'} = 1, \sigma\omega = \omega^{-1}\sigma\} \end{aligned}$$

And thus the homomorphism can be rephrased as $\pi(\sigma^i\omega^j, \omega^k) = \sigma^i\omega^{j+k}$. Now we check the homomorphism property. Let $(\sigma^{i_1}\omega^{j_1M'}, \omega^{3k_1})$ and $(\sigma^{i_2}\omega^{j_2M'}, \omega^{3k_2})$ be two elements in

$S_3 \rtimes_\psi \mathbb{Z}_{M'}$, and we first deal with the case where $i_2 = 1$

$$\begin{aligned}
\pi((\sigma^{i_1}\omega^{j_1 M'}, \omega^{3k_1})(\sigma^{i_2}\omega^{j_2 M'}, \omega^{3k_2}) &= \pi(\sigma^{i_1}\omega^{j_1 M'}\omega^{3k_1}\sigma^{i_2}\omega^{j_2 M'}\omega^{-3k_1}, \omega^{3k_1}\omega^{3k_2}) \\
&= \pi(\sigma^{i_1}\sigma^{i_2}\omega^{-j_1 M'}\omega^{-3k_1}\omega^{j_2 M'}\omega^{-3k_1}, \omega^{3k_1}\omega^{3k_2}) \\
&= \pi(\sigma^{i_1+i_2}\omega^{M'(-j_1+j_2)+3(-k_1-k_1)}, \omega^{3(k_1+k_2)}) \\
&= \sigma^{i_1+i_2}\omega^{M'(-j_1+j_2)+3(-k_1-k_1)}\omega^{3(k_1+k_2)} \\
&= \sigma^{i_1}\sigma^{i_2}\omega^{-j_1 M'}\omega^{j_2 M'}\omega^{-3k_1}\omega^{-3k_1}\omega^{3k_1}\omega^{3k_2} \\
&= \sigma^{i_1}\sigma^{i_2}\omega^{-j_1 M'}\omega^{-3k_1}\omega^{j_2 M'}(\omega^{-3k_1}\omega^{3k_1})\omega^{3k_2} \\
&= \sigma^{i_1}\sigma^{i_2}\omega^{-j_1 M'}\omega^{-3k_1}\omega^{j_2 M'}\omega^{3k_2} \\
&= \sigma^{i_1}\omega^{j_1 M'}\omega^{3k_1}\sigma^{i_2}\omega^{j_2 M'}\omega^{3k_2} \\
&= \pi(\sigma^{i_1}\omega^{j_1 M'}, \omega^{3k_1})\pi(\sigma^{i_2}\omega^{j_2 M'}, \omega^{3k_2})
\end{aligned}$$

and the case where $i_2 = 0$

$$\begin{aligned}
\pi((\sigma^{i_1}\omega^{j_1 M'}, \omega^{3k_1})(\omega^{j_2 M'}, \omega^{3k_2}) &= \pi(\sigma^{i_1}\omega^{j_1 M'}\omega^{3k_1}\omega^{j_2 M'}\omega^{-3k_1}, \omega^{3k_1}\omega^{3k_2}) \\
&= \pi(\sigma^{i_1}\omega^{j_1 M'+3k_1+j_2 M'-3k_1}, \omega^{3k_1+3k_2}) \\
&= \sigma^{i_1}\omega^{j_1 M'+3k_1+j_2 M'-3k_1}\omega^{3k_1+3k_2}) \\
&= \sigma^{i_1}\omega^{j_1 M'}\omega^{3k_1}\omega^{j_2 M'}\omega^{-3k_1}\omega^{3k_1}\omega 3k_2 \\
&= \sigma^{i_1}\omega^{j_1 M'}\omega^{3k_1}\omega^{j_2 M'}\omega 3k_2 \\
&= \pi(\sigma^{i_1}\omega^{j_1 M'}, \omega^{3k_1})\pi(\omega^{j_2 M'}, \omega 3k_2)
\end{aligned}$$

and this shows that π is a homomorphism, and thus by comparing the orders of the groups, $|S_3 \rtimes_\psi Z_{M'}| = 6 \times M' = 6M' = |D_{3M'}|$ we can see that the groups must be isomorphic. □

Theorem 4.27. *Let C be a code with a perspective with width of $\theta = M$, and a focused splitting. Then $\Gamma(C) \cong S_M$.*

Proof As $\theta = M = |C|$ then for all $b, c \in C$ we have $b \in S(c)$ and so there is only 1 component, thus $\hat{t} = 1$ and $\rho(1) = 1$. As C splits into M codes $D_1, D_2, \ldots D_M$ and $\mid D_i \mid = 1$ for all i, then $\hat{\Gamma}(C_1)$ is the identity group. Thus

$$\Gamma(C) \cong \left(\lambda(C) \times (\prod_{i=1}^{\hat{t}} S_{\rho(i)})\right) \rtimes_\psi (\prod_{i=1}^{t} \hat{\Gamma}(C_i)) \cong \left(\lambda(C) \times (\prod_{i=1}^{1} S_1)\right) \rtimes_\psi (\prod_{i=1}^{1} S_1) \cong \lambda(C)$$

. We know $\lambda(C)$ permutes the split codes, and as each split codes consists of exactly 1 element then every permutation of the split codes is admissible, and as there are M split codes thus $\lambda(C) \cong S_M$. Thus $\Gamma(C) \cong S_M$. □

Lemma 4.28. *Let C be a linear code consisting of components $C_1, C_2, \ldots, C_t$, and let $0^n \in C_1$, then $C_i = C_1 + c$ for some $c \in C$.*

Proof As C is linear then the sum of two elements of C is itself in C. Let $b, c \in C_i$ and as C_i is a component then there exists a finite number of words $s_1, s_2, \ldots s_r \in C_i$ such that $b \in S(s_1)$, $s_j \in S(s_{j+1})$ for all $i \in [1, r-1]$ and $s_r \in S(c)$. Now consider the set $C_i + d$ where $d \in C \setminus C_i$, as $b, c, s_j \in C_i$ thus we will have $b + d, c + d, s_j + d \in C_i + d$ and thus $C_i + d$ will be completely contained within a component of C say C_i'. Let $d' = -d$ that is the negative value of each of the non-zero co-ordinates of d, thus $d + d' = 0^n$ and consider $C_i' + d'$, as shown above $C_i' + d'$ will be contained within a component of C and as $b + d + d' \in C_i' + d'$ and $b + d + d' = b$ thus $C_i' + d' \subseteq C_i$. Thus $|C_i| \leq |C_i'|$ and $|C_i'| \leq |C_i|$ thus all components are the same size and $C_i + d = C_i'$ and thus all components will have the same structure. By using C_1 as our initial component we can see that each component can be formed as the sum of C_1 and some other element by taking $d \in C_i$ and thus $C_1 + d$ must contain the element $0^n + d = d$ and thus $C_1 + d = C_i$ for any component C_i. □

Theorem 4.29. *Let C be a linear code with a perspective of (θ, δ) and a focused splitting $D_1, D_2, \ldots, D_\theta$ and moreover let the focused splitting be linear as in Theorem 2.11 , then*

$$\Gamma(C) \cong \left(S_\theta \times S_t\right) \rtimes_\psi \prod_{i=1}^{t} \hat{\Gamma}(C_1).$$

Proof By Theorem 2.11 we know that all the split codes can be described as the sum of the split code containing 0^n and a word $e_i \in S(0) \setminus \{0\}$. Thus a permutation of $S(0)$ would give a permutation of the split codes, thus we can see that $\lambda(C) \cong S_\theta$.

By Theorem 4.28 each component will have the same structure as C_1, and thus

$$\hat{\Gamma}(C_i) \cong \hat{\Gamma}(C_1)$$

for all i. Moreover as each component has the same structure thus there is only one component class and we get that $\hat{t} = 1$ and $\rho(1) = t$, thus $\prod_{i=1}^{\hat{t}} S_{\rho(i)} = S_t$.

By Corollary 4.22 we know that

$$\begin{aligned} \Gamma(C) &\cong \left(\lambda(C) \times \left(\prod_{i=1}^{\hat{t}} S_{\rho(i)}\right)\right) \rtimes_\psi \left(\prod_{i=1}^{t} \hat{\Gamma}(C_i)\right) \\ &\cong \left(S_\theta \times S_t\right) \rtimes_\psi \prod_{i=1}^{t} \hat{\Gamma}(C_1) \end{aligned}$$

□

4.5 Concluding

As we have seen the automorphism groups of focused splittings are imbued with a considerable amount of structure inherited from the focused splitting and the original code, depending on the structure of the components of the original code as well as the relationship of the split codes. We have shown how the automorphism group can be expressed as the direct product and semi-direct product of other groups related to the focused splittings, and thus have a way of calculating the automorphism group of a focused splitting via first calculating several other related groups of the focused splitting.

We have investigated the automorphism groups of focused splittings because it would be amiss of us not to, many combinatorial objects can admit very interesting groups [9] and

as such it is necessary to investigate the automorphism groups of focused splittings. As it turns out the automorphism groups of focused splittings are generally the direct product and semi-direct product of many other groups and although this can be very interesting in itself, we are not considering it to be especially so in this case. Further investigation into the automorphism of a code and the focused splitting, that is permutations of co-ordinates and permutations of the underlying alphabet which preserve both the code words and the structure of the focused splitting would be valuable and potentially interested, especially as certain codes such as the Golay code have considerably interesting automorphism groups, but is alas beyond the scope of this chapter.

We shall now be moving our investigation away from the structure of focused splittings and the structure of the automorphism groups of focused splittings, albeit that there are still questions to be asked within these areas, and shall be moving on to look at the potential for further error correction utilising focused splittings. Returning, as it were, to the crux of the original problem.

Every act of creation is first an act of destruction.

Pablo Picasso

Chapter 5

A Construction for Increased Error Correction

5.1 Overview

In the following Chapter we shall briefly be discussing methods of error correction and encoding information which are different from the normally used block codes. These shall include *convolutional codes*[20], as well as *unequal error correction codes* [27][7] and *low density parity check codes* [25], and we shall look at how these achieve this, we shall do this in a rather informal manner, we are not attempting to explain these ideas in a thorough setting but rather to try and gleam an idea of any overarching concepts used. We shall then advance those ideas with our own construction for a form of block codes using the ideas of focused splittings and the structure thereof.

5.2 Convolutional Codes

In this section we shall briefly outline *convolutional codes*, *unequal error correction codes* and *low density parity check codes*, trying to highlight the structure that makes them successful rather than to impart a usable understanding in them. The aim here is to see

how structure allows for increased error correction, and whether such principles can be employed with focused splittings. Convolutional codes [20] are well known to be of high practical use, whilst also being relatively simple to implement. Technical details can be found in [20], we shall be briefly describing their behaviour here. Often described in terms of circuits which enable the encoding, we shall omit this technicality because it brings in to play concepts which are unnecessary for our purposes. Briefly an encoder takes an input sequence, or number of input sequences, and constructs a number of output sequences from combinations of the input bits.

Definition 5.1. An (n, k) *convolutional encoder* is a machine which takes k input sequences $x_1, \ldots, x_k$ and outputs n output sequences $y_1, \ldots, y_k$, where

$$x_1 = (\ldots, x_1 D^{-1}, x_1, x_1 D, x_1 D^2, \ldots)$$

and

$$y_1 = (\ldots y_1 D^{-1}, y_1, y_1 D, \ldots)$$

and D is a variable representing a delay in time, that is $x_1 D$ was the symbol of the message sent one time step before x_1. The input output relations can be represented as

$$y_j(D) = \sum_{1}^{k} x_i(D) g_{i,j}(D)$$

where $g_{i,j}(D)$ are polynomials in D.

Definition 5.2. The *constraint* length for input i is defined as $c_i = \max_{1 \leq j \leq n}(deg(g_{i,j}(D)))$ and then *constraint* of the encoder is defined as

$$c = \sum_{1}^{k} c_i$$

that is the sum of the constraint length for each input sequence.

The constraint of an encoder is important in the construction of circuits that implement the encoders in real life scenarios. Another view point for convolutional codes is of a machine having a state such that each k symbols of input message is encoded to n symbols, and the way this is done depends on the state, these k symbols then change the state of the machine. From this point of view a convolutional code can be viewed as a trellis, or bipartite graph, where the states are the nodes and the edges represented the combination of k symbols to be transmitted.

Given that every message starts with the machine in some initial state, or that the previous symbols were blank, then a decoding algorithm works by following what the state should be compared with the messages that it gets, if at some point a message is received that would not have been sent from the state the machine should have been in, then the decoder has to decide whether there was a mistake in the received message, or in the previous message, and thus in the state of the machine, this is generally done by the Viterbi algorithm [37] by playing out each possibility and seeing which leads to more acceptable scenarios. Thus we can see that the reason that convolutional codes increase error correction is because if we know a certain number of words have been received correctly, then we know the state of the machine, and thus we know the next word will be one of a specific set, moreover words following the erroneous words will show what state the machine is after the word was sent, these extra bits of information can be used to correct the word. The key idea being that information about the structure of the code allows for an increase in error correction.

Unequal error correction involves giving different digits of the information different weighted protection, therefore we have to define what we mean by this. We shall be referring to the information word in $(\mathbb{F}_q)^k$ which gives rise to the codeword from $(\mathbb{F}_q)^n$.

Definition 5.3. For a code C with a decoding method, we say the i-th digit of the information word has *protection level* e_i if for $e \leq e_i$ errors in the codeword, the i-th digit in the information word is correctly decoded, even though the entire codeword may not

be.

Definition 5.4. For a code C with a decoding method, we say the i-th digit of the codeword has *protection level* e_i if for $e \leq e_i$ errors in the codeword, the i-th digit in the codeword is correctly decoded, even though the entire codeword may not be.

A code has *unequal error correction* if different digits of the codewords have different protection levels, this can be achieved by careful manipulation of the generator matrix as seen in [7] . This exhibits how careful consideration of the process of decoding can be used to manipulate the output, in the case of unequal error correction this is shown by virtue of certain portions of information having a higher protection level than other portions of information despite the nature of encoding usually protecting all information equally.

Low density parity check codes work in a similar way, the idea is that the generator matrix is 'sparse' and as such decoding can be performed efficiently. The basic idea is that for each column in the parity check matrix any error that becomes obvious when calculating the syndrome has only got a few potential locations due to the 'sparse' nature of the matrix, and so there are only a few options for the location of the errors. We have not paid much attention to the problem of efficient decoding of a code, but it can be of huge importance if a large code is to be implemented, see [25], [3], [4]. We now move on to look at focused splittings as a structure on a code and how they may be utilised in a method of error correction.

5.3 A Focused Splitting Based Construction

In this section we look at how knowing that a given code has a focused splitting could potentially be used to create a code with better error correction. For block codes we hope to increase error correction by use of a check-word which shall be constructed from a focused splitting of the code.

Given a code $C = [n, k, d,]_q$ which has a perspective of (θ, δ) and a focused splitting $D_1, D_2, \ldots, D_\theta$, then for a word $c \in C$, knowing i such that $c \in D_i$ allows us to improve the number of errors c can withstand once transmitted. Without knowing i then c can be corrected to $\lfloor \frac{d-1}{2} \rfloor$ errors as per normal, but with the knowledge of i then c can be thought of as an element of D_i which has a minimum distance $d' \geq \delta$, and thus c can be corrected to $\lfloor \frac{d'-1}{2} \rfloor \geq \lfloor \frac{\delta-1}{2} \rfloor$ errors. We shall use $\hat{C}$ to refer to a code that transmits the information of i but with an minimum distance of d'', the length of $\hat{C}$ shall be referred to as n'.

This process in itself only decreases the rate of information while increasing the number of errors correctable and is thus equivalent to adding redundancy. In fact we can see that if we constructed a code $D = \{c\hat{c} : c \in D_1, \hat{c} \in \hat{C}\}$, where $\hat{c}$ is independent of c, and is used to convey more information, then the amount of information D would carry is $\frac{M}{\theta} \times \theta = M$, while if we consider $\tilde{D} = \{c\hat{c} : c \in D_{\hat{c}} \subset C, \hat{c} \in \hat{C}\}$ where the second piece of information is used to correct the first to $\lfloor \frac{d'-1}{2} \rfloor$ errors then the amount of information sent is M. Both codes can withstand $\lfloor \frac{d'-1}{2} \rfloor$ errors for the first n co-ordinates and $\lfloor \frac{d''-1}{2} \rfloor$ for the following n', and thus this construction does not give us any particular edge over just taking 2 codes in order, as we do with D_1 and $\hat{C}$ in the construction of D above.

Where this construction comes into its own is if we transmit several words which each have a respective check word from $\hat{C}$, but rather than transmit all of the check words, we could transmit a related word from another code so that if we knew 'some' of the words we could reconstruct the check word for the other words we transmitted. That is if we are only expecting to need to use the check word for 'some' of the words we transmit. We do that by treating the information of $i \in \hat{C}$ as a new alphabet and constructing a code around this.

Definition 5.5. Given a positive integer T a code C which has a partition into ϕ codes $D_1, D_2, \ldots, D_\phi$ and let $W = (c_1, c_2, \ldots, c_T) \in C^T$ then the *shadow* of W is defined as $\text{Shad}(W) = (j_1, j_2, \ldots, j_T)$ such that $c_i \in D_{j_i}$ for all i. We shall also write $\text{Shad}(c_i)$ to

refer to the shadow of a single word c_i where necessary.

The setting of the following construction is that we wish to transmit T words from C and due to the channel we are expecting, or we are preparing for up to H of them to have more than $\lfloor \frac{d-1}{2} \rfloor$ errors. As there is obviously a maximum amount of errors which we can have an acceptance of, we are going to hope that each of these H words has less than $\lfloor \frac{d'-1}{2} \rfloor$ errors. We choose a code C and it's associated partition such that the codes D_i all have a minimum distance of at least d', we also note that a focused splitting on C would guarantee the existence of such a partition where a perspective θ, δ and a focused splitting $D_1, D_2, \ldots, D_\theta$, with each code D_i having a minimum distance $d' > \delta$.

Definition 5.6. Given a code C which has a partition into ϕ codes $D_1, D_2, \ldots, D_\phi$ where each code D_i has a minimum distance of at least d' and a word $W = (c_1, c_2, \ldots, c_T) \in C^T$. Let P be a perfect code $P = [T, \hat{k}, \hat{d}]_\phi$, assuming one exists, with $\hat{d} \geq 2H + 1$, and thus by Theorem 2.13 the perfect code P will induce a focused splitting over ϕ^T and so $(\mathbb{F}_\phi)^T = P_1 \sqcup P_2 \sqcup \ldots P_{M'}$. Let a code $D = (n', M', d'')_q$, and arbitrarily label the words of D as $1, 2, 3, \ldots, M'$.

Given $W = (c_1, c_2, \ldots, c_T) \in C^T$ then $\text{Shad}(W) \in \phi^T$ and then there exists $s \in [1, M']$ such that $\text{Shad}(W) \in P_s$. Let

$$\tilde{C} = \{Ws : W = (c_1, c_2, \ldots, c_T) \in C^T, s \in D, \text{ such that } \text{Shad}(W) \in P_s\}.$$

Theorem 5.7. *Constructing $\tilde{C}$ as in Definition 5.6 and assuming that:*

(i) *only up to H of the words in W have more $\lfloor \frac{d-1}{2} \rfloor$ errors;*

(ii) *that these each have less than $\lfloor \frac{d'-1}{2} \rfloor$ errors;*

(iii) *that there are no more than $\lfloor \frac{d''-1}{2} \rfloor$ errors in s.*

Then we can correctly decode $\tilde{C}$.

Proof As there are no more than $\lfloor \frac{d''-1}{2} \rfloor$ errors in d then we can decode $s \in D$ correctly. Next we decode each $c_i \in W$ to its nearest neighbour in C giving us $\bar{W}$. As we know s we know that $\text{Shad}(W) \in P_s$. We've assumed only up to H errors greater than $\lfloor \frac{d-1}{2} \rfloor$ and so there are only up to H errors in $\bar{W}$ compared to W. As $P = [T, \hat{k}, \hat{d}]_\phi$, with $\hat{d} \geq 2H + 1$ we can correct up to H errors in $\text{Shad}(\bar{W})$ when considered as a word in P_s, thus giving us $\text{Shad}(W)$.

Now we know $\text{Shad}(W) = (j_1, j_2, \ldots, j_T)$ and so for each $\bar{c_i} \in \bar{W}$ such that $\bar{c_i} \notin D_{j_i}$, of which there are at most H of, we can take the originally received version of c_i and correct it to its nearest neighbour in D_{j_i}. Thus being able to correct up to $\lfloor \frac{d'-1}{2} \rfloor$ errors on each of those H words in W that had more than $\lfloor \frac{d-1}{2} \rfloor$ errors.

The other $T - H$ words in W were assumed to have less than $\lfloor \frac{d-1}{2} \rfloor$ errors and so were decoded correctly in $\bar{W}$.

Thus every word in $\tilde{C}$ can be decoded correctly. □

We note that the existence of a focused splitting on C gives us the partition required in the above Theorem and it is this interpretation which we shall be investigating in the following chapter. Other codes with partitions into smaller codes with higher minimum distance could also be applied to this theorem such as families of Reed-Muller codes. Next we discuss the choice and existence of the code P for the above theorem.

If we recall back to Theorem 1.37 we know that perfect codes are either one of the trivial codes, the Golay codes or the Hamming codes. The trivial perfect codes come in three types. The first are of the form $(n, 1, n)_q$, that is just 1 word, these are of no use in our construction because the focused splitting induced by this code is the splitting of the code into distinct code words, and thus the information telling us which focused splitting the shadow of the sent word would belong to would just be the word itself. The second form of trivial perfect codes are the complete space, this form of perfect code induces a trivial focused splitting, that is a splitting into one code, and thus the value of H in the above construction will be 0. The third form of trivial perfect codes are of the form

$(n, 2, n)_2$, where n is odd, that is 2 words equally distanced. This will only be of use when the width of the perspective of the code is 2, and will allow extra correction on $\lfloor \frac{n-1}{2} \rfloor$ code words and will require $|D| = \frac{2^n}{2} = 2^{n-1}$ and thus have dimension $n-1$. The Golay codes are either binary or ternary. The binary Golay code is a $[23, 12, 7]_2$ code, this will only be of use when the width of the perspective of the code is 2, will allow extra correction on 3 code words and will require $|D| = \frac{2^{23}}{2^{12}} = 2^{11}$. The ternary Golay code is a $[11, 6, 5]_3$ code, this will only be of use when the width of the perspective of the code is 3, will allow extra correction on 2 code words and will require $|D| = \frac{3^{11}}{3^6} = 3^5$. The Hamming codes are $[n, n-r, 3]_q$ codes where $n = \frac{q^r-1}{q-1}$ and r is an integer > 0. This can be used for any value of the width of the perspective of the code, will allow extra correction on 1 code word and will require $|D| = \frac{\theta^n}{\theta^{n-r}} = \theta^r$. Thus we find that the Hamming construction is the most applicable for our requirements.

5.4 Concluding

Although the concept and patterns around focused splittings developed are elegant in their own right, it is pleasing to be able to take the structure of these focused splittings and to apply this structure to the problem of error correction. We shall investigate the success of various classes or constructions of focused splittings with this method of code construction in the next Chapter.

Have no fear of perfection, you'll never reach it.

Salvador Dali

Chapter 6

Testing

6.1 Overview

In this Chapter we shall be constructing a number of examples to illustrate the construction given in Definition 5.6. We shall do this by comparing several different constructions for the existence of focused splittings given in Chapter 2. We shall be looking at the general form of each construction, an example of each construction, and we shall compare the error correcting capabilities of these constructions to known codes of the same length and dimension.

6.2 Error models and Simulation

In this chapter we shall be simulating a range of codes under both of our error models, random error and burst error, for a range of probabilities in each case.

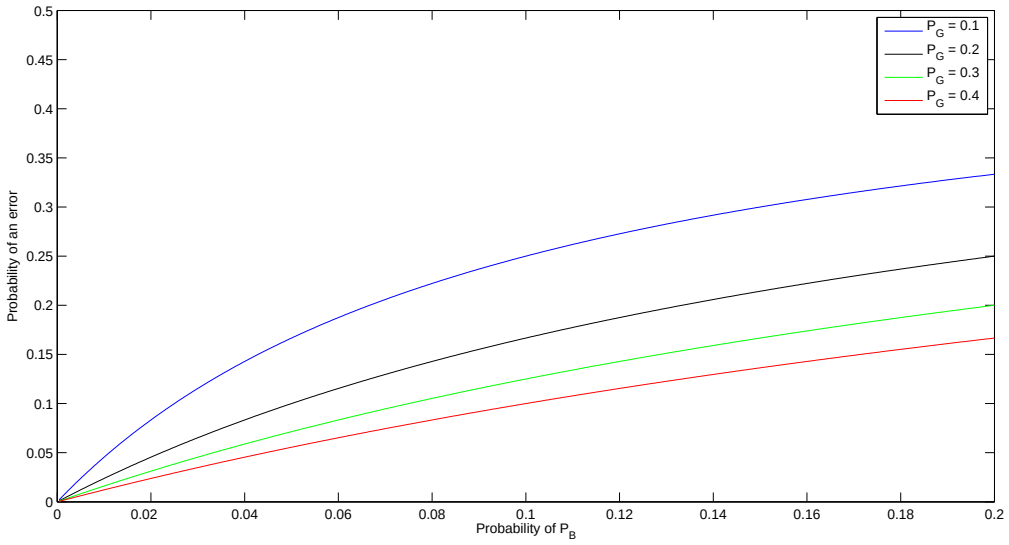
For burst errors we are utilising the Gilbert-Elliot model for burst errors [15], [26] [12] as discussed in Section 1.3 with the 'Good' error rate as 0, the 'Bad' error rate as 0.5 and with a varying state transition rate P_B and with multiple values of P_G. In each simulation we fix the transition rate P_G so that we can vary the overall error rate by varying P_B alone.

We know from Section 1.3 that the overall probability $\mathfrak{P}$ can be expressed as

$$\mathfrak{P} = 0.5(\frac{P_B}{P_B + P_G})$$

then for a fixed P_G we allow $\mathfrak{P} \in [0, \frac{1}{1+P_G}]$. In our series of simulations where we vary P_B from 0 to 0.2 and P_G is fixed at 0.1, 0.2, 0.3 and 0.4 respectively then we see that $\mathfrak{P}$ varies from 0 to $\frac{1}{3}$, $\frac{1}{4}$, $\frac{1}{5}$ and $\frac{1}{6}$, respectively. We can see in Figure 6.1 this relationship between $\mathfrak{P}$, the probability of an error and P_B for various values of P_G.

Figure 6.1: The relationship between $\mathfrak{P}$, the probability of an error and P_B.



We can see in Figure 6.1 that $\mathfrak{P}$ is strictly increasing with P_B and that for higher values of P_G we subsequently get $\mathfrak{P}$ at a lower value than for lower values of P_G. We also recall from Section 1.3 that a lower value of P_G leads to a higher concentration of bursts, a "burstier" channel.

In the following sections of this Chapter we shall be comparing our constructions against the best linear codes of the same length and dimension under the model of burst errors and also under a random error channel. We shall be simulating the running of the codes in Matlab. We simulate the codes by creating a noise vector, that is the distribution

of non-errors and errors through the codeword and then comparing how each of the codes would react under the noise vectors, for the normal block codes we thus compare the number of errors in the noise vector to the number which the code can withstand, and for our constructions we take into consideration the distribution of the errors, and thus count up the number of errors in various regions of the codeword, we then record whether each code successfully corrected the errors or not. For each probability we repeat the simulation $25,000$ times and average the results, thus to get a good idea of when each code succeeds and fails and to minimise uncharacteristic random noise. We do this over 500 probability steps of P_B, in most cases from 0 to 0.2, although for some codes we then take a closer look at take 500 probability steps from 0 to 0.02, for each step we take the average of the $25,000$ simulations and plot that average againstP_B, this is what gives the graphs we compute. For the random error channel we compute the noise vector by using a random number generator and seeing if an error occurs at each co-ordinate. For the burst error channel we first computer the state of the system at each co-ordinate, the two states are G or B, we take the previous co-ordinates state and roll a die to see if the state changes for the next co-ordinate, whilst in state G the probability of going to state B is P_B and the probability of staying in state G is $1 - P_B$, whilst in state B the probability of staying in state B is $1 - P_G$ and the probability of going to state G is P_G, and we start our simulation in state G at a non-realised co-ordinate 0. Having computed the state at each co-ordinate we apply the error probability for that state, in our model we have that in the state G there is 0 probability of an error occurring and in state B there is 0.5 probability of an error occurring, and thus we use a random number generator to simulate the specific probabilities and the distribution of events and we use these to calculate the distribution of errors in this channel.

6.3 Linear Binary Codes with n odd and $1^n \in C$

In this section we shall be looking at the construction of focused splittings given in Theorem 2.25 and how these focused splittings behave when applied to the construction given in Definition 5.6. Let $C = [n, k, d]_2$ be a code, with $1^n \in C$ and n odd. Thus by Theorem 2.25 there exists a focused splitting on C with $\theta = 2^{k-1}$. For a positive integer m, it is known [35] that there exists a perfect code with length

$$T = \frac{q^m - 1}{q - 1} = \frac{\theta^m - 1}{\theta - 1} = \frac{2^{(k-1)m} - 1}{2^{k-1} - 1}$$

and dimension

$$\hat{k} = T - m$$

and minimum distance 3. That is there exists a perfect code $P = [T, \hat{k}, 3]_\theta$.

Thus by Theorem 2.13 we know P induces a partition on θ^T. $(\mathbb{F}_\theta)^T = P_1 \sqcup P_2 \sqcup \ldots, P_{M'}$ where

$$M' = \frac{\theta^T}{\theta^{\hat{k}}} = \frac{\theta^T}{\theta^{T-m}} = \theta^m = 2^{(k-1)m}$$

and thus we need a code $D = [n', (k-1)m, d'']_2$. As m copies of C would give a code with parameters $[nm, mk, d]_2$ we can see that $n' \leq nm$, and thus the code $\tilde{C} = C^T D$ has a total length $\leq n \times (T + m) = n \times (\frac{2^{(k-1)m}-1}{2^{k-1}-1} + m)$.

We now consider a more concrete example, with the Hamming code $C = [7, 4, 3,]_2$. We can calculate that C has weight enumerator $1 + 7x^3 + 7x^4 + x^7$, and thus setting $\delta = 4$ we get $\theta = 8$. As $1^n \in C$, which can be seen from the weight enumerator, by Theorem 2.25 we know that C has a focused splitting into $D_1, D_2, \ldots, D_\theta$ and $|D_i| = 2$. As D_i has minimum distance of 7 we can see that D_i can correct up to 3 errors.

As $\theta = 8$ we require a perfect code over $\mathbb{F}_8$, which is given by $P = [9, 7, 3]_8$, by the Hamming Code construction [35]. As P is perfect, then P induces a splitting over $(\mathbb{F}_8)^9$, and so there are $1 + \binom{9}{1}(q-1) = 1 + 9 \times 7 = 64$ split codes over $(\mathbb{F}_8)^9$ and we label them

$P_1, P_2, \ldots, P_{64}$. Thus we require $|D| = 64$ and thus over $\mathbb{F}_2$ this means D has dimension 6. We set $D = [14, 6, 5]_2$, which is shown to exist [8] and label the words $1, 2, \ldots, 64$.

We transmit $W \in C^9$, that is 9 consecutive words from C and then one from D which represents i such that $\text{Shad}(W) \in P_i$. Thus we can correct 1 error in $\text{Shad}(\bar{W})$, where $\bar{W}$ is the received form form of W once every word has been corrected to it's nearest neighbour in C, and thus can correct 1 of the words in W as though it was in D_i and thus correct up to 3 errors.

Our construction can correct up to 1 error in 8 of the 9 codewords from C and up to 3 errors in any specific $9th$ codeword as well as up to 2 errors in D. Thus our construction can correct all occurrences of 1 or 2 errors, but it can also correct up to a maximum of 13 if the errors are arranged favourably.

We can see we can correct 3 errors if either all the errors are in separate codewords, or only 2 errors are in the same codeword and only up to 2 errors in D, thus calculating the fraction of triple errors possible to correct we get

$$\frac{\binom{9}{3}7^3 + \binom{9}{1}7\binom{8}{1}\binom{7}{2} + \binom{9}{2}7^2\binom{14}{1} + \binom{9}{1}\binom{7}{2}\binom{14}{1} + \binom{9}{1}7\binom{14}{2} + \binom{9}{1}\binom{7}{3}}{\binom{77}{3}}$$

this is as there are either 3 errors in separate codewords, 2 errors in one codeword and 1 in another, 2 errors in separate codewords and 1 error in the D, 2 errors in one codeword and 1 error in D, 1 error in a codeword and 2 errors in D or all 3 errors in one codeword. This thus simplifies to showing that our construction can correct

$$\frac{28812 + 10584 + 24696 + 2646 + 5733 + 315}{73150} = \frac{72786}{73150} = \frac{5199}{5225} \approx 0.9950$$

of triple errors. Similarly we break down the calculation for a higher number of errors by considering them by type. There are either 0, 1 or 2 errors in D and therefore we consider the combinations where these contribute to the total number of errors separately, as shown by the breakdown of rows in the following calculations where the first row will contain

0 errors in D, the second row will contain 1 error in D and the final row will contain 2 errors in D. We also break the calculations down by whether one of the codewords from C contains 2 or 3 errors, in each row the first calculation contains no codewords containing 2 or 3 calculations, the second calculation contains a codeword containing 2 errors and the third calculation contains a codeword containing 3 errors.

Thus calculating the fraction of quadruple errors possible to correct we get

$$\frac{1}{\binom{77}{4}} \left(\begin{array}{l} \binom{9}{4}7^4 + \binom{9}{2}7^2\binom{7}{1}\binom{7}{2} + \binom{9}{1}7\binom{8}{1}\binom{7}{3} + \\ \binom{9}{3}7^3\binom{14}{1} + \binom{9}{1}7\binom{8}{1}\binom{7}{2}\binom{14}{1} + \binom{9}{1}\binom{7}{3}\binom{14}{1} + \\ \binom{9}{2}7^2\binom{14}{2} + \binom{9}{1}\binom{7}{2}\binom{14}{2} \end{array} \right)$$

Which it can be shown simplifies to

$$\begin{aligned} \frac{1}{1353275} & \left(\begin{array}{l} 302526 + 259308 + 17640 + \\ 403368 + 148176 + 4410 + \\ 160524 + 17199 \end{array} \right) \\ &= \frac{1313151}{1353275} \\ &= \frac{187593}{193325} \\ &\approx 0.9704 \end{aligned}$$

of quadruple errors.

Thus calculating the fraction of quintuple errors possible to correct we get

$$\frac{1}{\binom{77}{5}} \left(\begin{array}{l} \binom{9}{5}7^5 + \binom{9}{3}7^3\binom{6}{1}\binom{7}{2} + \binom{9}{2}7^2\binom{7}{1}\binom{7}{3} + \\ \binom{9}{4}7^4\binom{14}{1} + \binom{9}{2}7^2\binom{7}{1}\binom{7}{2}\binom{14}{1} + \binom{9}{1}7\binom{8}{1}\binom{7}{3}\binom{14}{1} + \end{array} \right.$$

$$\binom{9}{3}7^3\binom{14}{2} + \binom{9}{1}7\binom{8}{1}\binom{7}{2}\binom{14}{2} + \binom{9}{1}\binom{7}{3}\binom{14}{2}\Bigg)$$

Which it can be shown simplifies to

$$\begin{aligned}
\frac{1}{19757815} \quad \Bigg(\quad & 2117682 + 3630312 + 432180 + \\
& 4235364 + 3630312 + 246960 + \\
& 2621892 + 963144 + 28665 \Bigg) \\
= \quad & \frac{17906511}{19757815} \\
\approx \quad & 0.9063
\end{aligned}$$

of quintuple errors.

Thus calculating the fraction of sextuple errors possible to correct we get

$$\begin{aligned}
\frac{1}{\binom{77}{6}} \quad \Bigg(\quad & \binom{9}{6}7^6 + \binom{9}{4}7^4\binom{5}{1}\binom{7}{2} + \binom{9}{3}7^3\binom{6}{1}\binom{7}{3} + \\
& \binom{9}{5}7^5\binom{14}{1} + \binom{9}{3}7^3\binom{6}{1}\binom{7}{2}\binom{14}{1} + \binom{9}{2}7^2\binom{7}{1}\binom{7}{3}\binom{14}{1} + \\
& \binom{9}{4}7^4\binom{14}{2} + \binom{9}{2}7^2\binom{7}{1}\binom{7}{2}\binom{14}{2} + \binom{9}{1}7\binom{8}{1}\binom{7}{3}\binom{14}{2}\Bigg)
\end{aligned}$$

Which it can be shown simplifies to

$$\begin{aligned}
\frac{1}{237093780} \quad \Bigg(\quad & 9882516 + 31765230 + 4268880 + \\
& 29647548 + 50824368 + 6050520 + \\
& 2752986 + 23597028 + 1605240 \Bigg) \\
= \quad & \frac{160394316}{237093780} \\
\approx \quad & 0.6765
\end{aligned}$$

of sextuple errors.

Thus calculating the fraction of septuple errors possible to correct we get

$$\frac{1}{\binom{77}{7}} \left(\begin{array}{l} \binom{9}{7}7^7 + \binom{9}{5}7^5\binom{4}{1}\binom{7}{2} + \binom{9}{4}7^4\binom{5}{1}\binom{7}{3} + \\ \binom{9}{6}7^6\binom{14}{1} + \binom{9}{4}7^4\binom{5}{1}\binom{7}{2}\binom{14}{1} + \binom{9}{3}7^3\binom{6}{1}\binom{7}{3}\binom{14}{1} + \\ \binom{9}{5}7^5\binom{14}{2} + \binom{9}{3}7^3\binom{6}{1}\binom{7}{2}\binom{14}{2} + \binom{9}{2}7^2\binom{7}{1}\binom{7}{3}\binom{14}{2} \end{array} \right)$$

Which it can be shown simplifies to

$$\begin{aligned} \frac{1}{2404808340} & \left(\begin{array}{l} 29647548 + 177885288 + 52942050 + \\ 138355224 + 444713220 + 84707280 + \\ 19270906 + 330358392 + 39328380 \end{array} \right) \\ &= \frac{1317208288}{2404808340} \\ &\approx 0.5477 \end{aligned}$$

of septuple errors.

Thus calculating the fraction of octuple errors possible to correct we get

$$\frac{1}{\binom{77}{8}} \left(\begin{array}{l} \binom{9}{8}7^8 + \binom{9}{6}7^6\binom{3}{1}\binom{7}{2} + \binom{9}{5}7^5\binom{4}{1}\binom{7}{3} + \\ \binom{9}{7}7^7\binom{14}{1} + \binom{9}{5}7^5\binom{4}{1}\binom{7}{2}\binom{14}{1} + \binom{9}{4}7^4\binom{5}{1}\binom{7}{3}\binom{14}{1} + \\ \binom{9}{6}7^6\binom{14}{2} + \binom{9}{4}7^4\binom{5}{1}\binom{7}{2}\binom{14}{2} + \binom{9}{3}7^3\binom{6}{1}\binom{7}{3}\binom{14}{2} \end{array} \right)$$

Which it can be shown simplifies to

$$\frac{1}{21042072975} \left(\begin{array}{l} 51883209 + 622598508 + 296475480 + \end{array} \right.$$

$$
\begin{aligned}
& 415065672 + 2490394032 + 741188700 + \\
& 899308956 + 2890635930 + 550597320 \Big) \\
= \; & \frac{8958147807}{21042072975} \\
\approx \; & 0.4257
\end{aligned}
$$

of octuple errors.

Thus calculating the fraction of nonuple errors possible to correct we get

$$
\begin{aligned}
\frac{1}{\binom{77}{9}} \Bigg(& \binom{9}{9}7^9 + \binom{9}{7}7^7\binom{2}{1}\binom{7}{2} + \binom{9}{6}7^6\binom{3}{1}\binom{7}{3} + \\
& \binom{9}{8}7^8\binom{14}{1} + \binom{9}{6}7^6\binom{3}{1}\binom{7}{2}\binom{14}{1} + \binom{9}{5}7^5\binom{4}{1}\binom{7}{3}\binom{14}{1} + \\
& \binom{9}{7}7^7\binom{14}{2} + \binom{9}{5}7^5\binom{4}{1}\binom{7}{2}\binom{14}{2} + \binom{9}{4}7^4\binom{5}{1}\binom{7}{3}\binom{14}{2} \Bigg)
\end{aligned}
$$

Which it can be shown simplifies to

$$
\begin{aligned}
\frac{1}{161322559475} \Bigg(& 40353607 + 1245197016 + 1037664180 + \\
& 726364926 + 8716379112 + 4150656720 + \\
& 2697926868 + 16187561208 + 4817726550 \Big) \\
= \; & \frac{39619830187}{161322559475} \\
\approx \; & 0.2456
\end{aligned}
$$

of nonuple errors.

Thus calculating the fraction of decuple errors possible to correct we get

$$
\begin{aligned}
\frac{1}{\binom{77}{10}} \Bigg(& \binom{9}{8}7^8\binom{1}{1}\binom{7}{2} + \binom{9}{7}7^7\binom{2}{1}\binom{7}{3} + \\
& \binom{9}{9}7^9\binom{14}{1} + \binom{9}{7}7^7\binom{2}{1}\binom{7}{2}\binom{14}{1} + \binom{9}{6}7^6\binom{3}{1}\binom{7}{3}\binom{14}{1} +
\end{aligned}
$$

$$\binom{9}{8}7^8\binom{14}{2} + \binom{9}{6}7^6\binom{3}{1}\binom{7}{2}\binom{14}{2} + \binom{9}{5}7^5\binom{4}{1}\binom{7}{3}\binom{14}{2}\Bigg)$$

Which it can be shown simplifies to

$$\begin{aligned}\frac{1}{1096993404430}\ &\Bigg(\ 1089547389 + 2075328360 + \\ &\quad 564950498 + 17432758224 + 14527298520 + \\ &\quad 4721372019 + 56656464228 + 26979268680\Bigg) \\ &= \frac{124046987918}{1096993404430} \\ &\approx 0.1131\end{aligned}$$

of decuple errors.

Thus calculating the fraction of hendecuple errors possible to correct we get

$$\begin{aligned}\frac{1}{\binom{77}{11}}\ &\Bigg(\ \binom{9}{8}7^8\binom{1}{1}\binom{7}{3} + \\ &\quad \binom{9}{8}7^8\binom{1}{1}\binom{7}{2}\binom{14}{1} + \binom{9}{7}7^7\binom{2}{1}\binom{7}{3}\binom{14}{1} + \\ &\quad \binom{9}{9}7^9\binom{14}{2} + \binom{9}{7}7^7\binom{2}{1}\binom{7}{2}\binom{14}{2} + \binom{9}{6}7^6\binom{3}{1}\binom{7}{3}\binom{14}{2}\Bigg)\end{aligned}$$

Which it can be shown simplifies to

$$\begin{aligned}\frac{1}{6681687099710}\ &\Bigg(\ 1815912315 + \\ &\quad 15253663446 + 29054597040 + \\ &\quad 3672178237 + 113312928456 + 94427440380\Bigg) \\ &= \frac{257536719874}{6681687099710} \\ &\approx 0.0385\end{aligned}$$

of hendecuple errors.

Thus calculating the fraction of duodecuple errors possible to correct we get

$$\frac{1}{\binom{77}{12}} \left(\begin{array}{l} \binom{9}{8}7^8\binom{1}{1}\binom{7}{3}\binom{14}{1} \\ \binom{9}{8}7^8\binom{1}{1}\binom{7}{2}\binom{14}{2} + \binom{9}{7}7^7\binom{2}{1}\binom{7}{3}\binom{14}{2} \end{array} \right)$$

Which it can be shown simplifies to

$$\begin{aligned} \frac{1}{36749279048405} & \left(\begin{array}{l} 25422772410 + \\ \qquad 99148812399 + 188854880760 \end{array} \right) \\ &= \frac{313426465569}{36749279048405} \\ &\approx 0.0085 \end{aligned}$$

of duodecuple errors.

Thus calculating the fraction of triodecuple errors possible to correct we get

$$\frac{1}{\binom{77}{13}} \left(\binom{9}{8}7^8\binom{1}{1}\binom{7}{3}\binom{14}{2} \right)$$

Which it can be shown simplifies to

$$\frac{165248020665}{183746395242025} \approx 0.0009$$

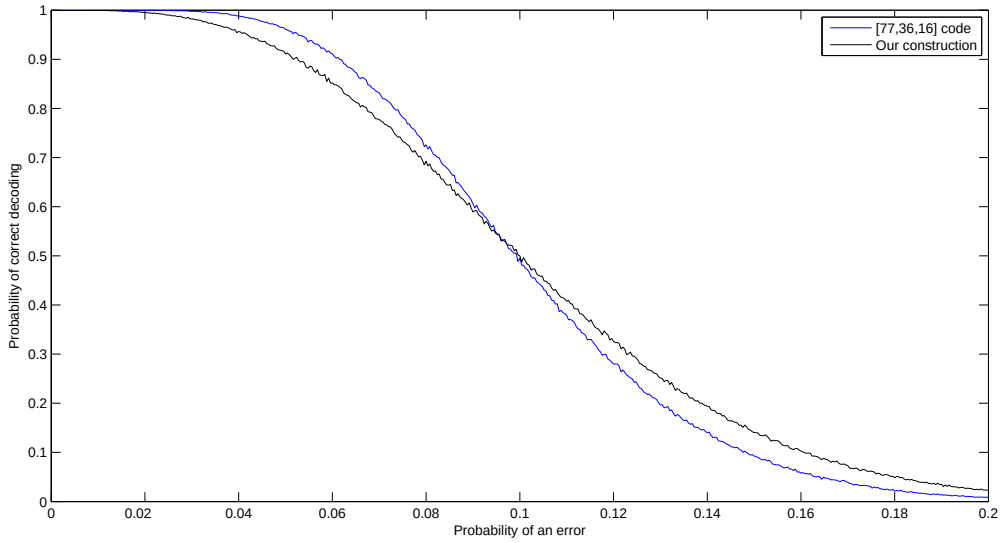
of triodecuple errors.

Collecting this together the fraction of errors correction for 1, 2, 3, etc. errors is approximately 1, 1, 0.9950 0.9704, 0.9063, 0.6765, 0.5477, 0.4257, 0.2456, 0.1131, 0.0385, 0.0085, 0.0009.

Now as these 77 characters carries $9 \times 4 = 36$ bits of information, looking in the

literature [8] we can find a $[77, 36, 16]_2$ code, which would thus correct 7 errors. We now compare our $[77, 36]$ construction to the $[77, 36, 16]_2$ code under a random error channel and obtain Figure 6.2.

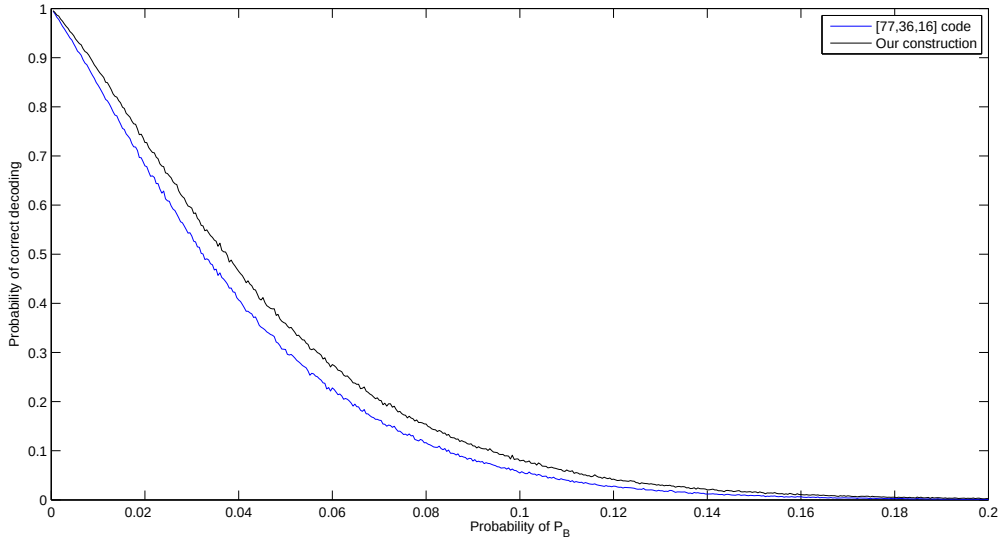
Figure 6.2: Comparison of $[77, 36]_2$ codes for a random error channel up to an error probability of 0.2



We can see in Figure 6.2 that our construction performs better than the $[77, 36, 16]_2$ code for error rates above approximately 0.09, although this gives a relatively low probability of correct decoding and is ultimately impractical. We now simulate the codes under a burst error channel model, but first we need to interlace our code. We do this as our construction for $\tilde{C}$ requires errors to be 'far apart' from each other for a successful decoding, subsequently we rearrange the ordering of the co-ordinates such that 'far apart' errors are actually close together and thus bursty error propagation will lead to errors that are 'far apart' and thus can be corrected by $\tilde{C}$. We have $\tilde{C} = C^9 D$ where C has length 7 and D has length 14, and as such we interlace them such that we get 4 co-ordinates from C^9 and then 1 co-ordinate from D, then 5 co-ordinates from C^9 and then 1 co-ordinate from D, with the co-ordinates from C^9 being from subsequent copies of C, so that two co-ordinates from the same word appear far apart. For each simulation,

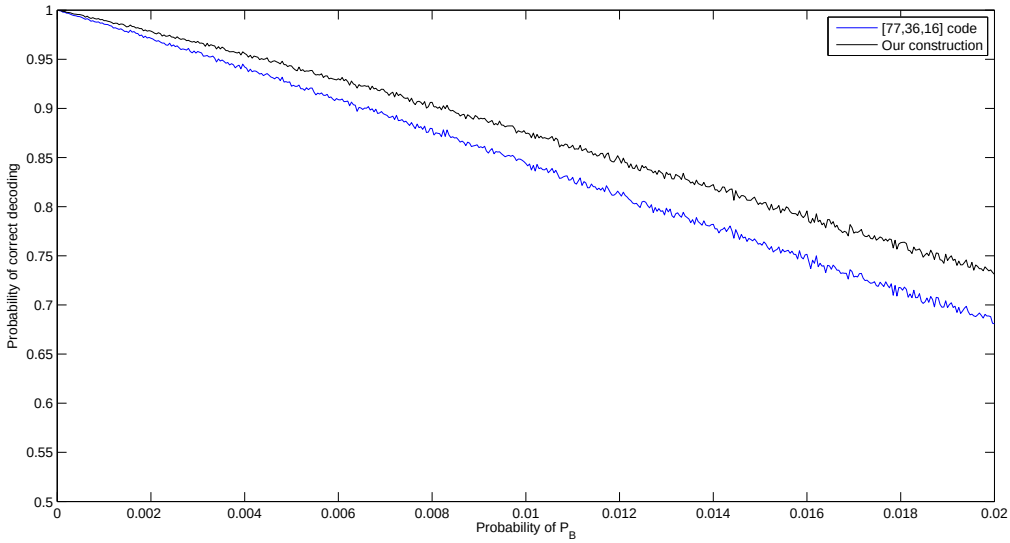
Figure 6.3 through to Figure 6.10 we fixed P_G and we vary P_B from 0 to 0.2 or from 0 to 0.02, once each for each value of P_G in each case with 500 probability steps.

Figure 6.3: Comparison of $[77, 36]_2$ codes for a burst error channel with $P_G = 0.1$ with $P_B \leq 0.2$



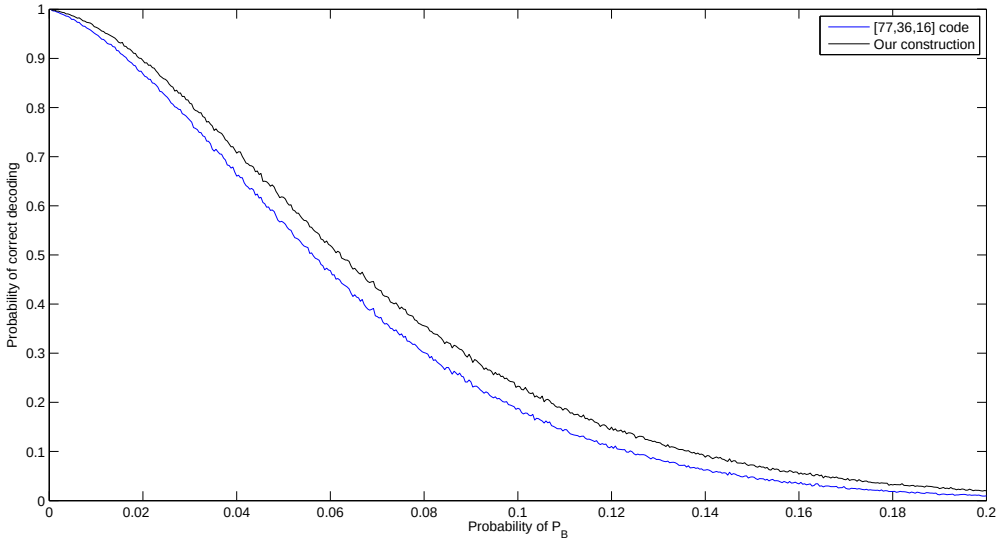
We repeat this simulation for P_B up to 0.02 over 500 probability steps to get a closer look at the lower end of the probability, where the codes are closer.

Figure 6.4: Comparison of $[77, 36]_2$ codes for a burst error channel with $P_G = 0.1$ with $P_B \leq 0.02$



As we can see from Figure 6.3 and Figure 6.4 that even for a very low error probability our construction has a higher probability of a correct decoding than the $[77, 36, 16]_2$ code under the burst error channel with $P_G = 0.1$, where as for the random error channel it did not, this is as due to the bursty nature of the error channel, the errors are more likely to be collected together and are thus more commonly 'favourable' for our construction.

Figure 6.5: Comparison of $[77, 36]_2$ codes for a burst error channel with $P_G = 0.2$ with $P_B \leq 0.2$



We repeat this simulation for P_B up to 0.02 over 500 probability steps to get a closer look at the lower end of the probability, where the codes are closer.

Figure 6.6: Comparison of $[77, 36]_2$ codes for a burst error channel with $P_G = 0.2$ with $P_B \leq 0.02$

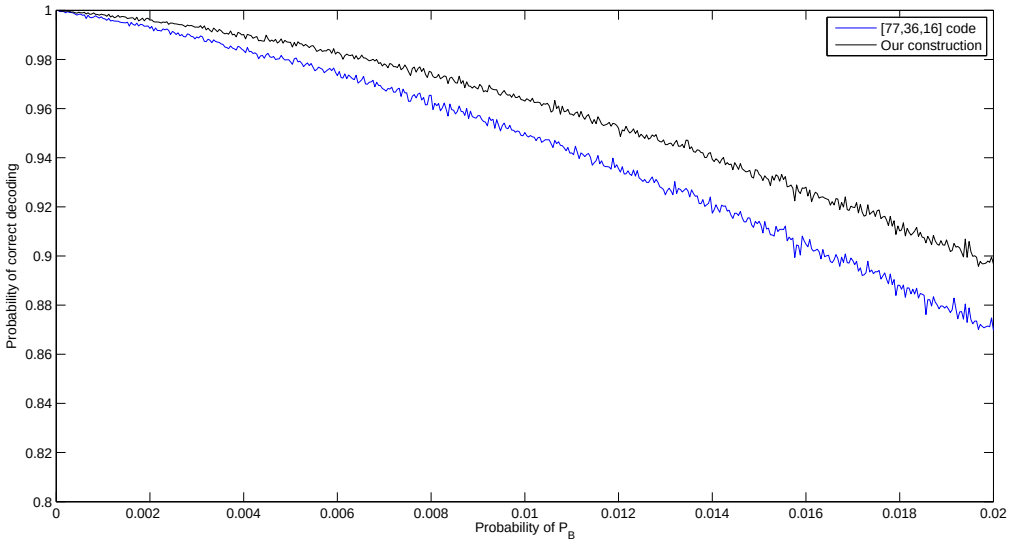
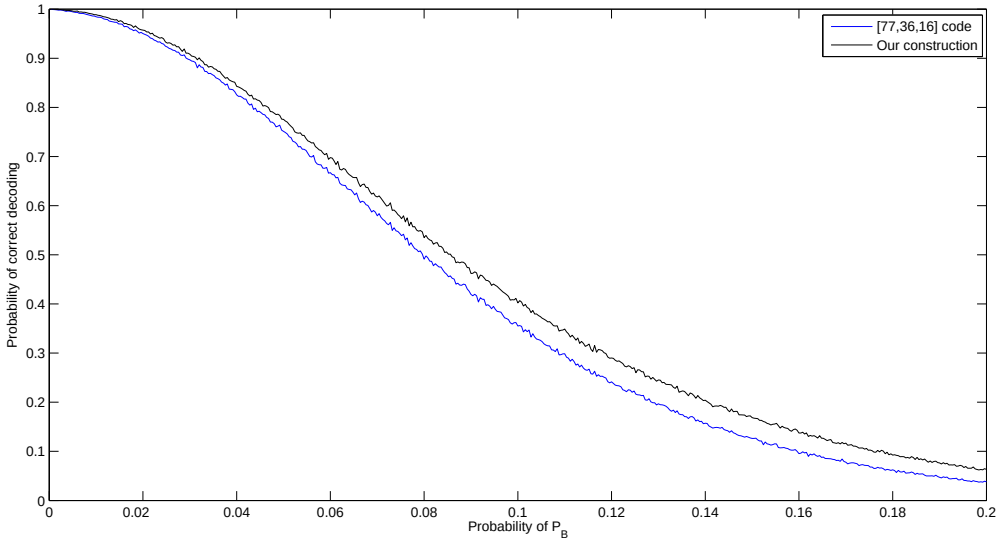
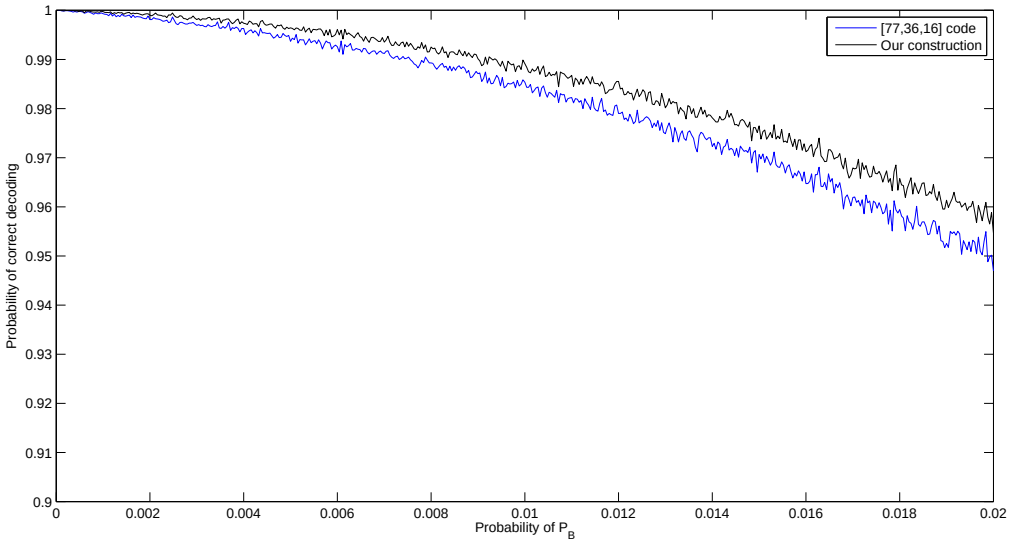


Figure 6.7: Comparison of $[77, 36]_2$ codes for a burst error channel with $P_G = 0.3$ with $P_B \leq 0.2$



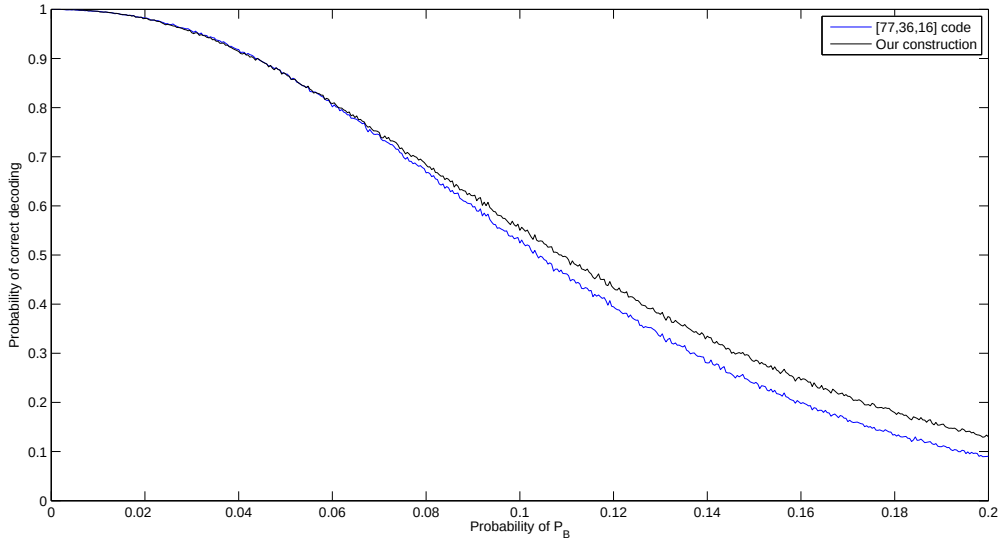
We repeat this simulation for P_B up to 0.02 over 500 probability steps to get a closer look at the lower end of the probability, where the codes are closer.

Figure 6.8: Comparison of $[77, 36]_2$ codes for a burst error channel with $P_G = 0.3$ with $P_B \leq 0.02$



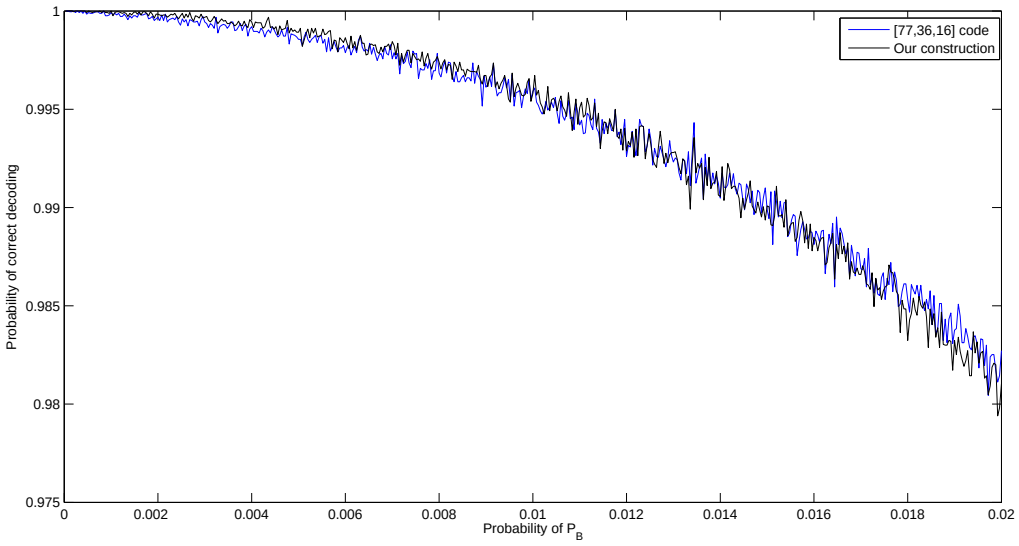
As we can see in Figure 6.5, Figure 6.6, Figure 6.7 and Figure 6.8 that for very low error probability our construction has a higher probability of a correct decoding than the $[77, 36, 16]_2$ code under the burst error channel with $P_G = 0.2$ and $P_G = 0.3$, although the improvement is less than it was for $P_G = 0.1$.

Figure 6.9: Comparison of $[77, 36]_2$ codes for a burst error channel with $P_G = 0.4$ with $P_B \leq 0.2$



We repeat this simulation for P_B up to 0.01 over 500 probability steps to get a closer look at the lower end of the probability, where the codes are closer.

Figure 6.10: Comparison of $[77, 36]_2$ codes for a burst error channel with $P_G = 0.4$ with $P_B \leq 0.02$



As we can see in Figure 6.9 and Figure 6.10 our construction and the $[77, 36, 16]_2$ code have fairly equally probability of a correct decoding for lower values of P_B and where our construction has a higher probability of a correct decoding is where the value of P_B, and thus the overall probability of an error, is higher.

We note that when the channel is 'burstier', i.e. $P_G = 0.1$ our construction performs substantially better than the $[77, 36, 16]_2$ code.

6.4 Perfect Codes as Focused Splittings of Complete Spaces

We now move on to examine the case where the initial code is the complete space $(\mathbb{F}_q)^n$ and to look at the construction when this is the case.

We know by Theorem 2.13 that a perfect code $P \subseteq (\mathbb{F}_q)^n$ will induce a focused splitting

on $(\mathbb{F}_q)^n$ and that the split codes D_i will all be translations of P, that is

$$D_i = \{p + c_i : p \in P\}$$

for some $c_i \in S(0)$. We now use this focused splitting on the complete space as the basis for our construction.

It is known [35] that perfect codes are either trivial, repetition codes, Hamming codes or Golay codes. Repetition codes have parameters $[n, q, n]_q$. Hamming codes have parameters $[\frac{q^m-1}{q-1}, \frac{q^m-1}{q-1} - m, 3]_q$. The binary Golay code has parameters $[23, 12, 7]_2$. The ternary Golay code has parameters $[11, 6, 5]_3$.

For our first example we use the whole space $(\mathbb{F}_2)^3 = [3, 3, 1]_2$ which contains the perfect code $P = [3, 2, 3]_2$ which is both an example of a repetition code and a binary Hamming code with a redundancy of 2. Thus we find that $\theta = 1 + 3(2 - 1) = 4$ and thus for our construction we require the existence of a perfect code P' over $(\mathbb{F}_\theta)^T$ for some T of our choosing. Using the Hamming construction [17] we see that $P' = [\frac{\theta^m-1}{\theta-1}, \frac{\theta^m-1}{\theta-1} - m, 3]_\theta$ exists and so substituting $\theta = 4$ and setting $m = 2$ we get $P' = [5, 3, 3]_4$ and thus we see $|D| = \frac{\theta^5}{\theta^3} = 4^2 = 2^4$ and thus D will require dimension 4, and thus we set $D = [7, 4, 3]_2$ code. Thus with $T = 5$ our construction gives

$$\tilde{C} = \{Wd : W \in C^5, d \in D, Shad(W) \in P_d\}$$

which gives a $[5T + 7, 5T] = [22, 15]_2$ code which can protect against 1 error in the first 15 co-ordinates and 1 error in the final 7.

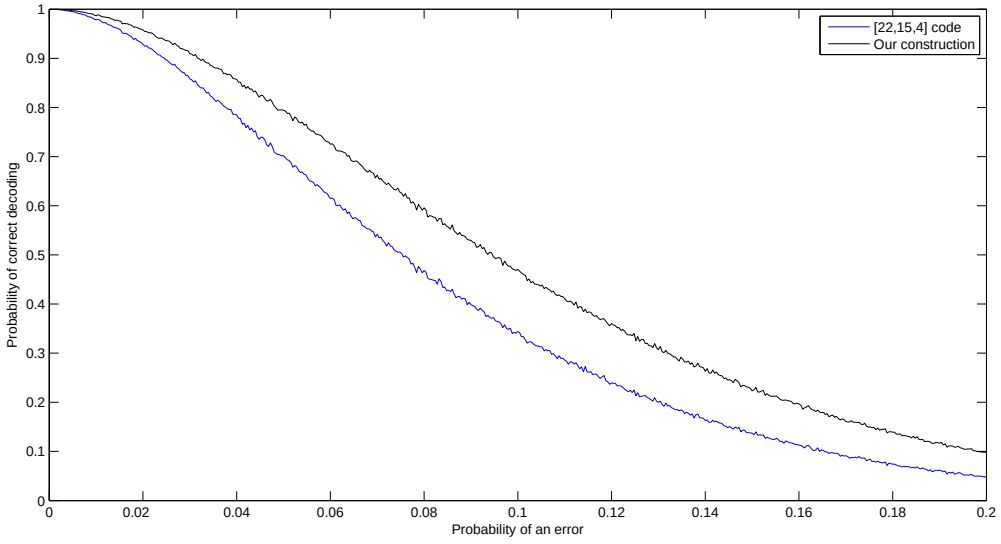
Looking in the literature [8] we see there exists a $[22, 15, 4]_2$ code and no $[22, 15, 5]_2$ code and thus we consider the $[22, 15, 4]_2$ code as an equal length and dimension code and then we compare this to our construction. We know that our construction $\tilde{C}$ can protect against all occurrences of 1 error, and against 2 errors only if 1 error appears in a specific

15 co-ordinates and 1 error in the other 7 co-ordinates, and thus only for

$$\frac{\binom{15}{1}\binom{7}{1}}{\binom{22}{2}} = \frac{15 \times 7}{231} = \frac{105}{231} = \frac{5}{11} \approx 0.4545$$

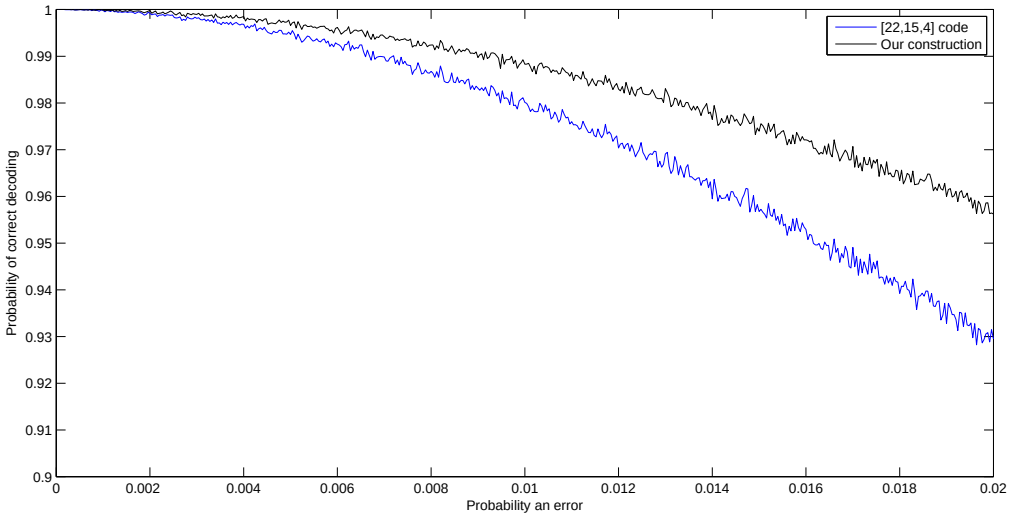
of double errors. We now simulate these codes to see the actual effect of this difference in error correction. For a normal random error model we range our errors from 0 to 0.2 giving us Figure 6.11.

Figure 6.11: Comparison of $[22, 15]_2$ codes for a random error channel up to an error probability of 0.2



Looking at Figure 6.11 we can see that our construction has a higher likelihood of correct decoding than the $[22, 15, 4]_2$ code under a random error channel for higher values of p. We repeated our simulation for the range 0 to 0.02, again with 500 probability steps, to give a better analysis of the difference between the $[22, 15, 4]_2$ code and our construction, giving us Figure 6.12.

Figure 6.12: Comparison of $[22, 15]_2$ codes for a random error channel up to an error probability of 0.02



As can be seen from Figure 6.12 our construction performs better than the $[22, 15, 4]_2$ code for this lower range of probabilities as well.

To simulate the codes under a burst error channel model we need to first interlace our code. We have $\tilde{C} = C^5 D$ where C has length 3 and D has length 7, and as such we interlace them such that we get 2 co-ordinates from C^5 and then 1 co-ordinate from D, with 1 co-ordinate from C^5 at the end. For each figure we fixed P_G and we vary P_B from 0 to 0.2 with 500 probability steps.

Figure 6.13: Comparison of $[22, 15]_2$ codes for a burst error channel with $P_G = 0.1$ with $P_B \leq 0.2$

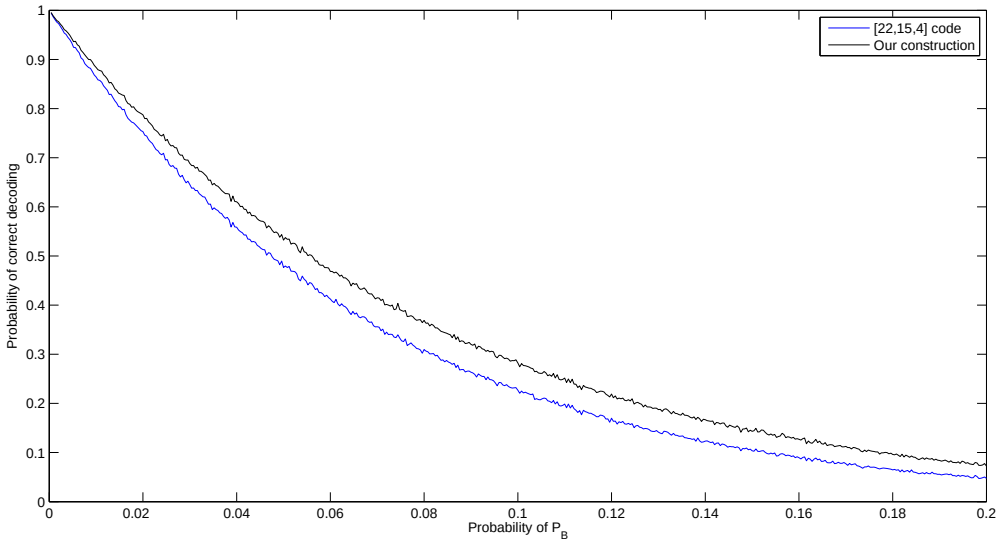


Figure 6.14: Comparison of $[22, 15]_2$ codes for a burst error channel with $P_G = 0.2$ with $P_B \leq 0.2$

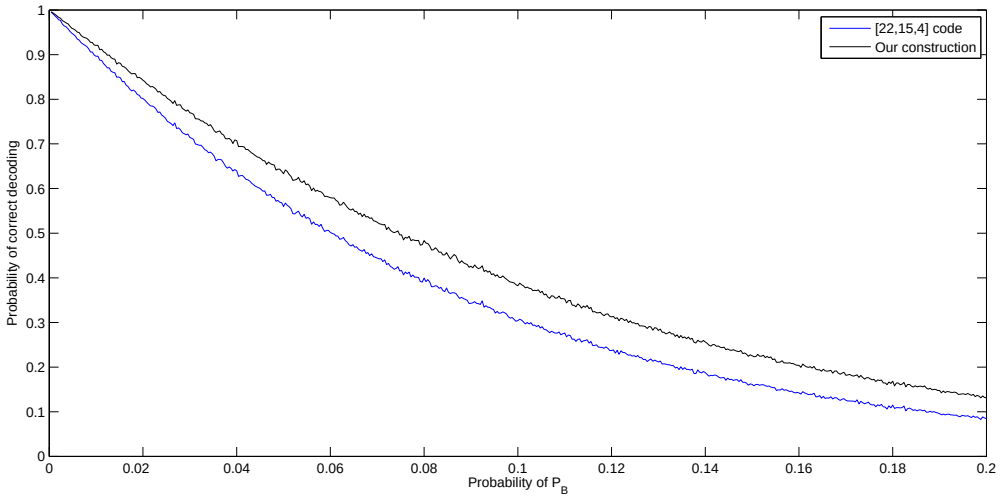


Figure 6.15: Comparison of $[22, 15]_2$ codes for a burst error channel with $P_G = 0.3$ with $P_B \leq 0.2$

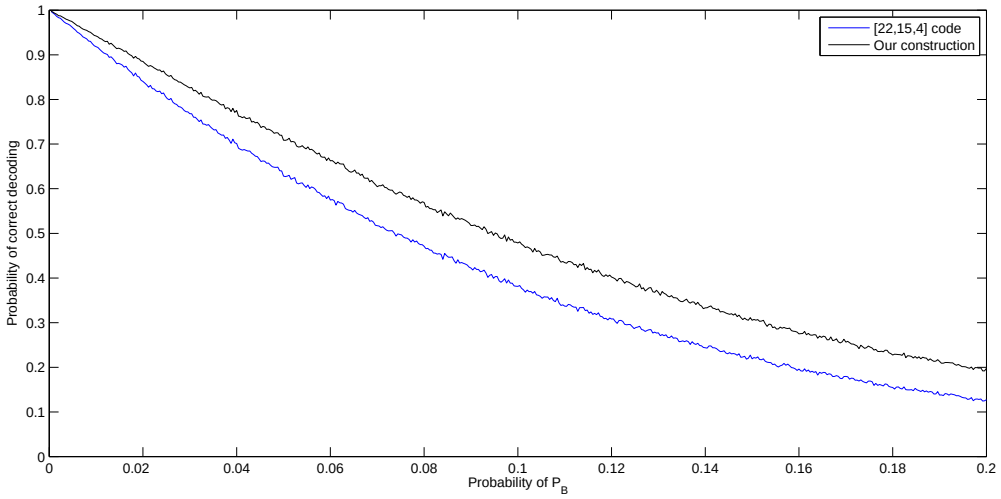
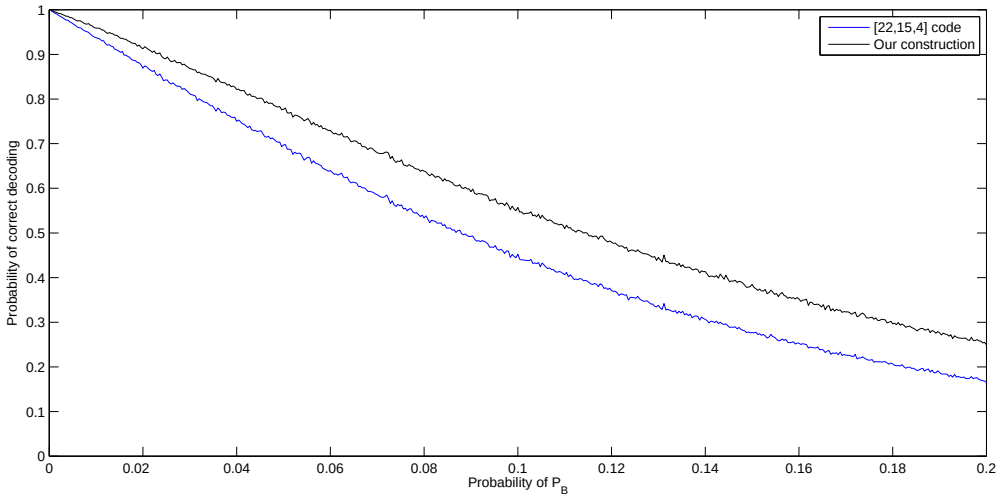


Figure 6.16: Comparison of $[22, 15]_2$ codes for a burst error channel with $P_G = 0.4$ with $P_B \leq 0.2$



In Figures 6.13 6.14 6.15 and 6.16 we can see that our construction has a higher likelihood of a correct decoding than the $[22, 15, 4]_2$ code, moreover this improvement is consistent across the range of P_G values.

For our second example we use the whole space $[7, 7, 1]_2$ which contains the perfect code $P = [7, 4, 3]_2$ which is a binary Hamming code with a redundancy of 3. Thus we

find that $\theta = 1 + 7(2-1) = 8$ and thus for our construction we require the existence of a perfect code P' over $(\mathbb{F}_\theta)^T$ for some T of our choosing. Using the Hamming construction [17] we see that $P' = [\frac{\theta^m-1}{\theta-1}, \frac{\theta^m-1}{\theta-1} - m, 3]_\theta$ exists and so substituting $\theta = 8$ and setting $m = 2$ we get $P' = [9, 7, 3]_8$ and thus we see $|D| = \frac{\theta^9}{\theta^7} = 8^2 = 2^6$ and thus D will require dimension 6, and thus we can set $D = [10, 6, 3]_2$ code or we can set $D = [14, 6, 5]_2$ code. Thus with $T = 5$ our construction gives

$$\tilde{C} = \{Wd : W \in C^9, Shad(W) \in P_d\}$$

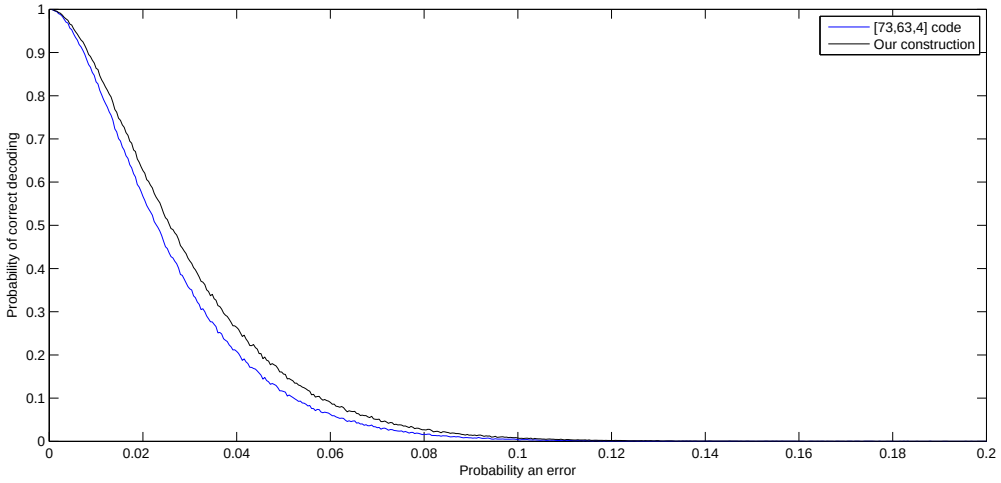
which gives a $[9T + n', 9T]_2$ code which depending on our value of D is either a $[73, 63]_2$ code or a $[77, 63]_2$ code, in the first case it can protect against 1 error in the first 63 co-ordinates and 1 error in the final 10 co-ordinates, in the second case it can protect against 1 error in the first 63 co-ordinates and 2 errors in the final 14 co-ordinates.

Looking in the literature [8] we see there exists a $[73, 63, 4]_2$ code and no $[73, 63, 5]_2$ code and there exists a $[77, 63, 6]_2$ code and no $[77, 63, 7]_2$ code. Comparing our $[73, 63]_2$ construction with the $[73, 63, 4]_2$ code we see that the $[73, 63, 4]_2$ code can protect against 1 error whilst our construction can protect against all occurrences of 1 error and 2 errors if the first error appears in a specific 63 co-ordinates and the second error appears in the other 10, thus our construction can correct

$$\frac{\binom{63}{1}\binom{10}{1}}{\binom{73}{2}} = \frac{630}{2628} = \frac{35}{146} \approx 0.2397$$

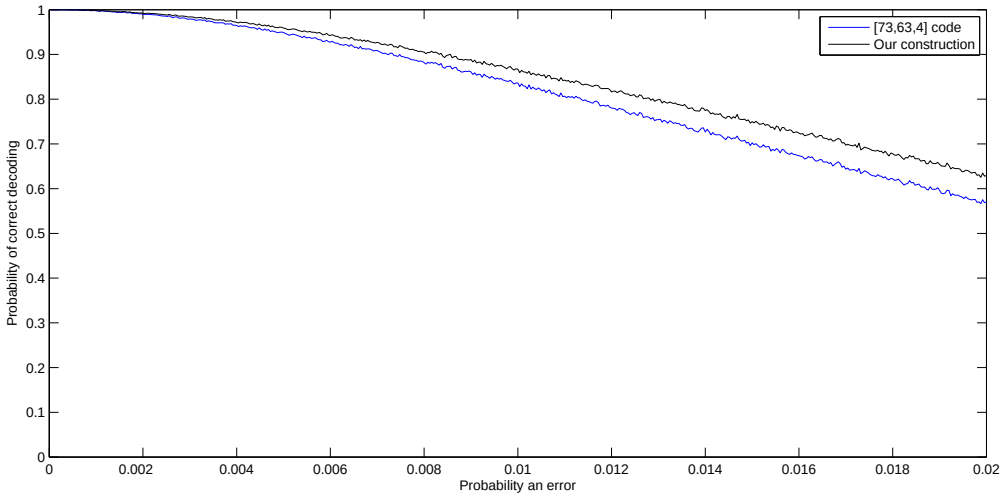
double errors. We now simulate these codes to see the actual effect of this difference in error correction. For a normal random error model we range our errors from 0 to 0.2 giving us Figure 6.17.

Figure 6.17: Comparison of $[73, 63]_2$ codes for a random error channel up to an error probability of 0.2



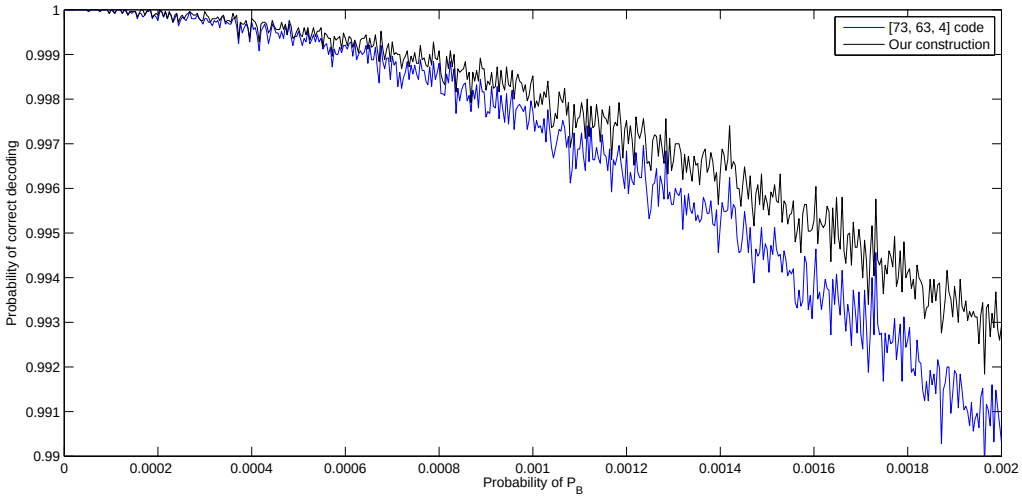
Looking at Figure 6.17 we can see that our construction has a higher likelihood of correct decoding than the $[73, 63, 4]_2$ code under a random error channel for higher values of p. We repeated our simulation for the range 0 to 0.02, again with 500 probability steps, to give a better analysis of the difference between the $[73, 63, 4]_2$ code and our construction, giving us Figure 6.18.

Figure 6.18: Comparison of $[73, 63]_2$ codes for a random error channel up to an error probability of 0.02



To further examine this case at low probabilities we repeated the simulation again for the range 0 to 0.002, again with 500 probability steps, to give a better analysis of the difference between the $[73, 63, 4]_2$ code and our construction, giving us Figure 6.19.

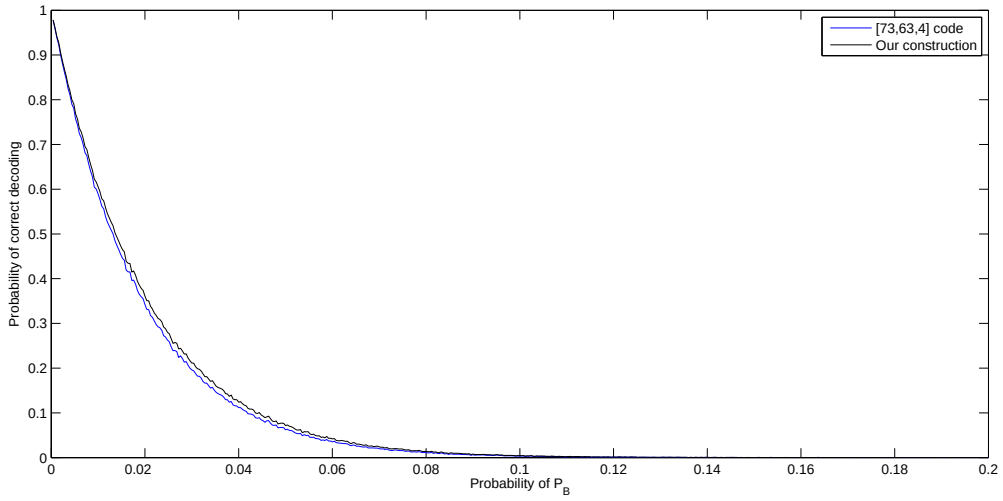
Figure 6.19: Comparison of $[73, 63]_2$ codes for a random error channel up to an error probability of 0.002



We can see in Figure 6.19 and Figures 6.18 and 6.17 that our construction has a higher likelihood of a correct decoding for P_B probability values above 0.001 and that between 0 and 0.001 our construction matches the $[73, 63, 4]_2$ code.

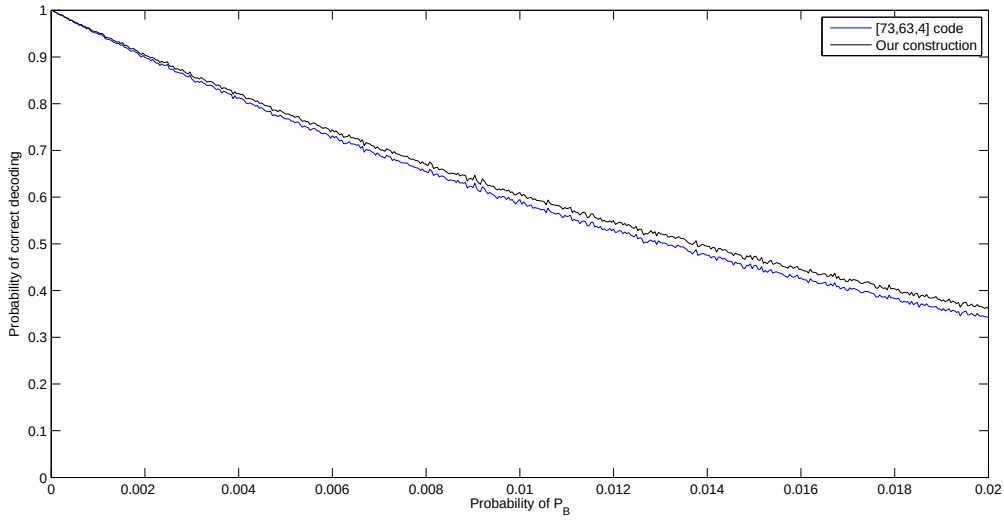
To simulate the codes under a burst error channel model we need to first interlace our code. We have $\tilde{C} = C^9 D$ where C has length 7 and D has length 10, and as such we interlace them such that we get 6 co-ordinates from C^9 and then 1 co-ordinate from D, with 3 co-ordinate from C^9 at the end, this way the co-ordinates for D are as far as they can be from each other. For each figure we fixed P_G and we vary P_B from 0 to 0.2 with 500 probability steps.

Figure 6.20: Comparison of $[73, 63]_2$ codes for a burst error channel with $P_G = 0.1$ with $P_B \leq 0.2$



We repeated this simulation for the probability range 0 to 0.02 to get a clearer picture of what is happening at these probability rates.

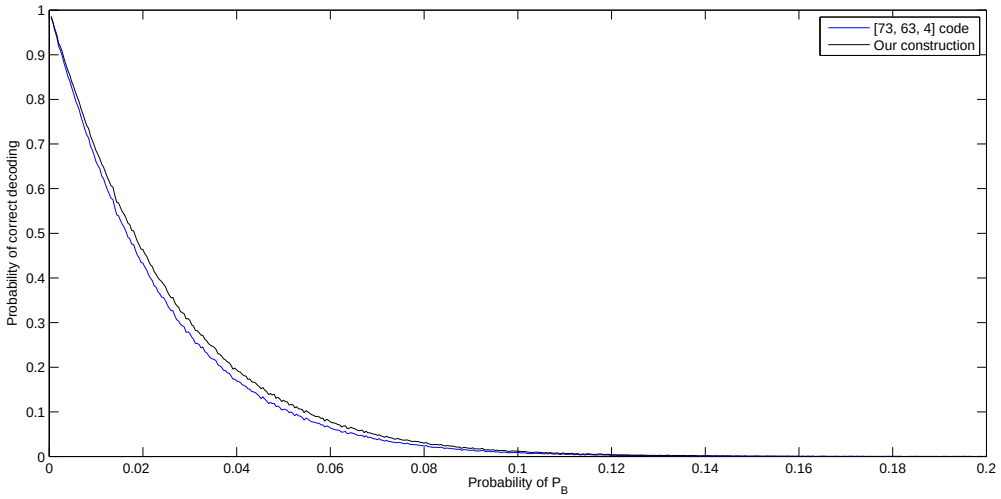
Figure 6.21: Comparison of $[73, 63]_2$ codes for a burst error channel with $P_G = 0.1$ with $P_B \leq 0.02$



As can be seen in Figure 6.20 and Figure 6.21 that our construction and the $[73, 63, 4]$ code have very close likelihood of a correct decoding for a burst error channel with $P_G = 0.1$, but that our construction does have a slightly higher likelihood of a correct decoding

even for very low values of P_B.

Figure 6.22: Comparison of $[73, 63]_2$ codes for a burst error channel with $P_G = 0.2$ with $P_B \leq 0.2$



We repeated this simulation for the probability range 0 to 0.02 to get a clearer picture of what is happening at these probability rates.

Figure 6.23: Comparison of $[73, 63]_2$ codes for a burst error channel with $P_G = 0.2$ with $P_B \leq 0.02$

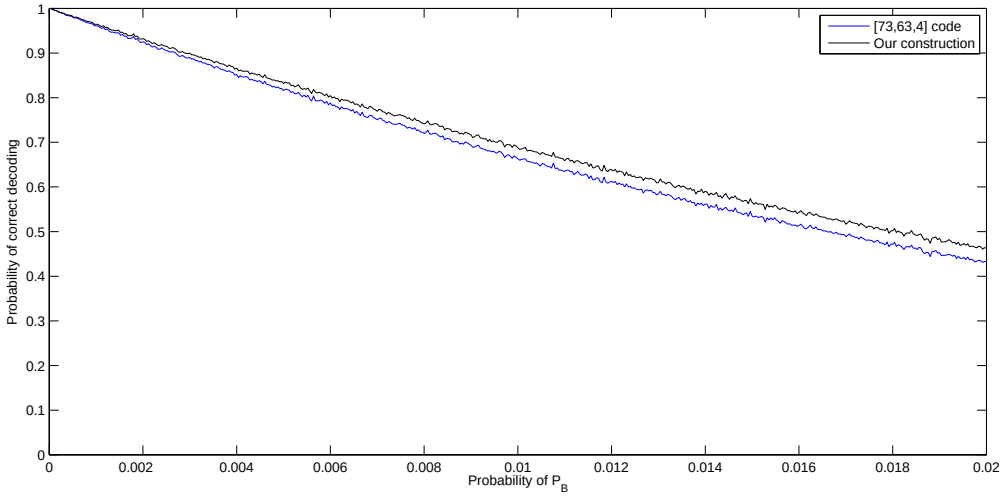
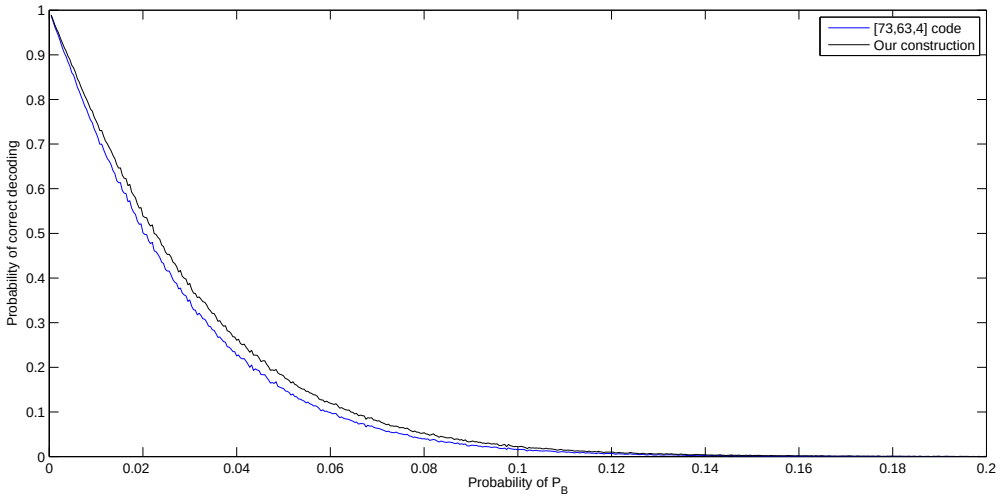
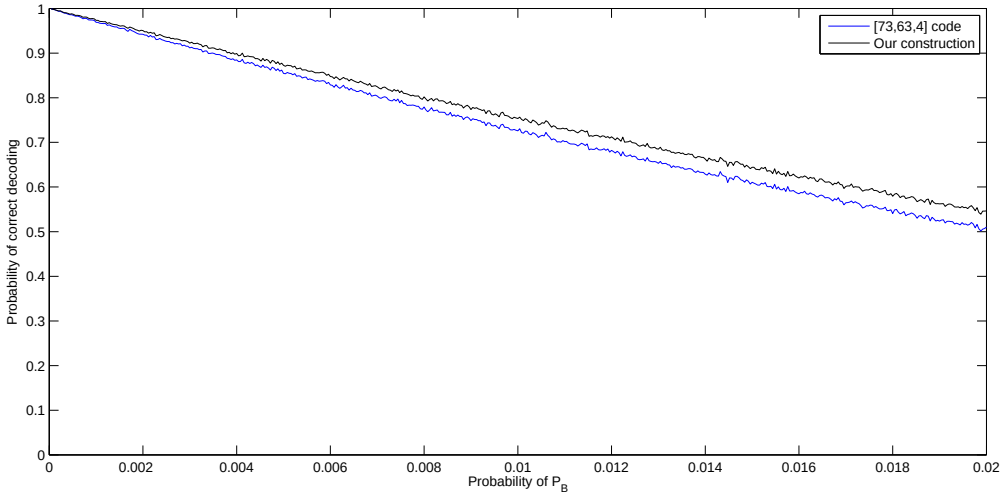


Figure 6.24: Comparison of $[73, 63]_2$ codes for a burst error channel with $P_G = 0.3$ with $P_B \leq 0.2$



We repeated this simulation for the probability range 0 to 0.02 to get a clearer picture of what is happening at these probability rates.

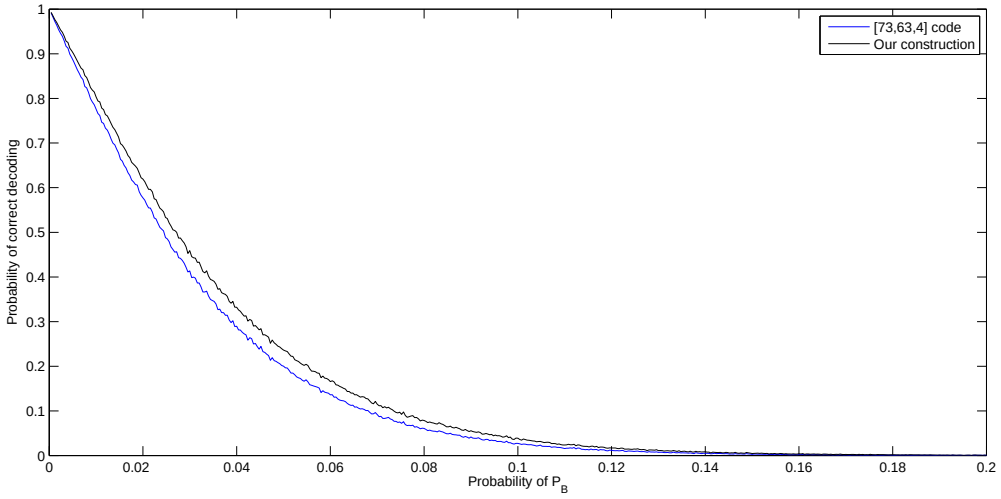
Figure 6.25: Comparison of $[73, 63]_2$ codes for a burst error channel with $P_G = 0.3$ with $P_B \leq 0.02$



As can be seen in Figure 6.22, Figure 6.23, Figure 6.24 and Figure 6.25 that our construction and the $[73, 63, 4]$ code have very close likelihood of a correct decoding for a burst error channel with $P_G = 0.2$ and $P_G = 0.3$, but that our construction does have a

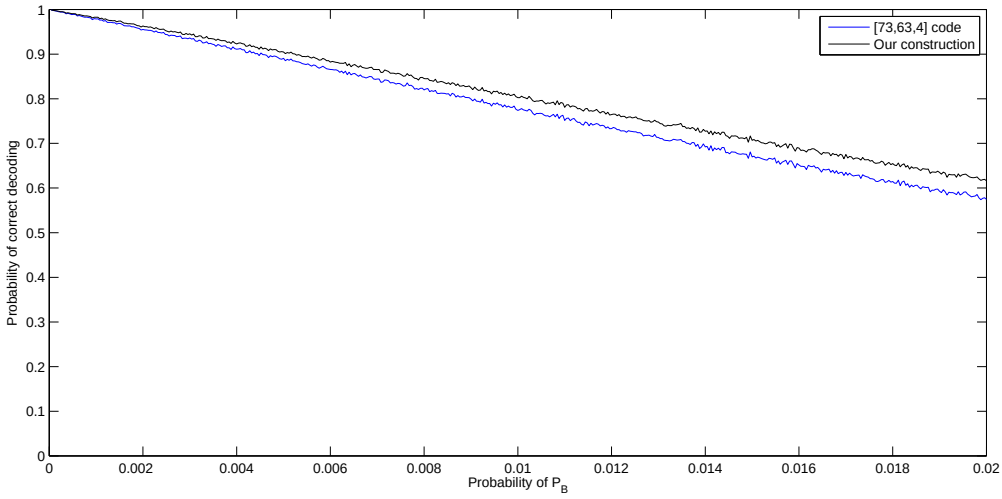
slightly higher likelihood of a correct decoding even for very low values of P_B.

Figure 6.26: Comparison of $[73, 63]_2$ codes for a burst error channel with $P_G = 0.4$ with $P_B \leq 0.2$



We repeated this simulation for the probability range 0 to 0.02 to get a clearer picture of what is happening at these probability rates.

Figure 6.27: Comparison of $[73, 63]_2$ codes for a burst error channel with $P_G = 0.4$ with $P_B \leq 0.02$



As can be seen in Figure 6.26 and Figure 6.27, our construction and the $[73, 63, 4]$ code have very close likelihood of a correct decoding for a burst error channel with $P_G = 0.4$,

but our construction does have a slightly higher likelihood of a correct decoding even for very low values of P_B.

Now comparing our $[77, 63]_2$ construction with the $[77, 63, 6]_2$ code we see that the $[77, 63, 6]_2$ can correct all occurrences of up to 2 errors, whilst our $[77, 63]_2$ construction can deal with all occurrences of 1 error, 2 errors if they either both occur in the final 14 co-ordinates or if 1 error occurs in the first 63 and 1 in the final 14 and it can correct 3 errors if 1 appears in the first 63 and 2 appear in the final 14. Thus it can correct

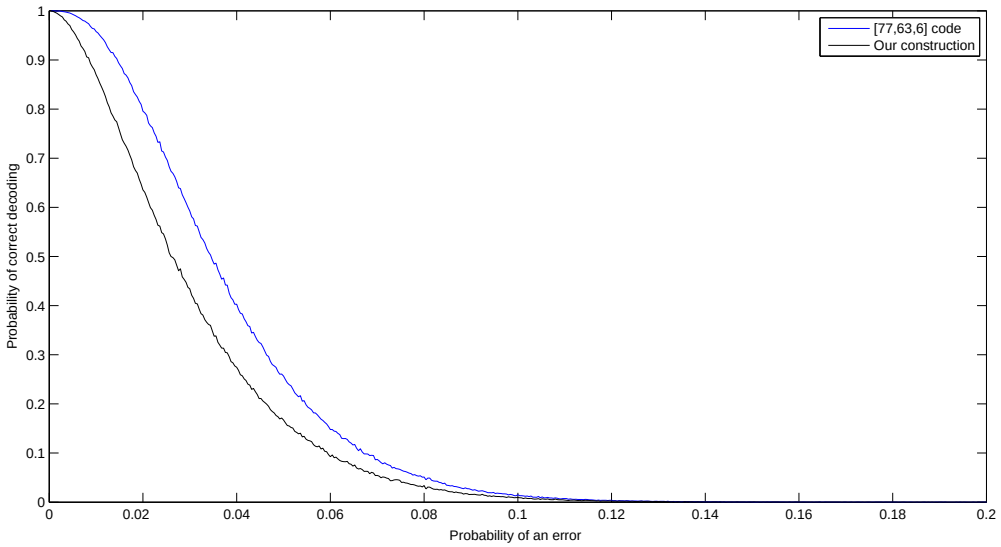
$$\frac{\binom{14}{2}}{\binom{77}{2}} + \frac{\binom{63}{1}\binom{14}{1}}{\binom{77}{2}} = \frac{91}{2926} + \frac{882}{2926} = \frac{139}{418} \approx 0.3325$$

of double errors, and only

$$\frac{\binom{63}{1}\binom{14}{2}}{\binom{77}{3}} = \frac{819}{10450} \approx 0.0784$$

of triple errors. We now simulate these codes to see the actual effect of this difference in error correction. For a normal random error model we range our errors from 0 to 0.2 giving us the following graph.

Figure 6.28: Comparison of $[77, 63]_2$ codes for a random error channel up to an error probability of 0.2



As we can clearly see in Figure 6.28 our $[77, 63]_2$ construction does not compare favourably to the $[77, 63, 6]_2$ code and this is because although the $[77, 63]_2$ construction can correct up to 3 errors in favourable arrangements, it does so at the loss of being able to correct all occurrences of 2 errors, and subsequently has a lower probability of correct decoding.

To simulate the codes under a burst error channel model we need to first interlace our code. We have $\tilde{C} = C^9 D$ where C has length 7 and D has length 14, and as such we interlace them such that we get 8 co-ordinates from C^9 and then 1 co-ordinate from D, then 1 from C then 1 from D, this way for 9 co-ordinates from C we have 2 from D, and thus this is repeated 7 times, also we see that this means that certain bursts of 3 errors will hit 2 co-ordinates of D and only 1 from C. For each figure we fixed P_G and we vary P_B from 0 to 0.2 with 500 probability steps.

Figure 6.29: Comparison of $[77, 63]_2$ codes for a burst error channel with $P_G = 0.1$ with $P_B \leq 0.2$

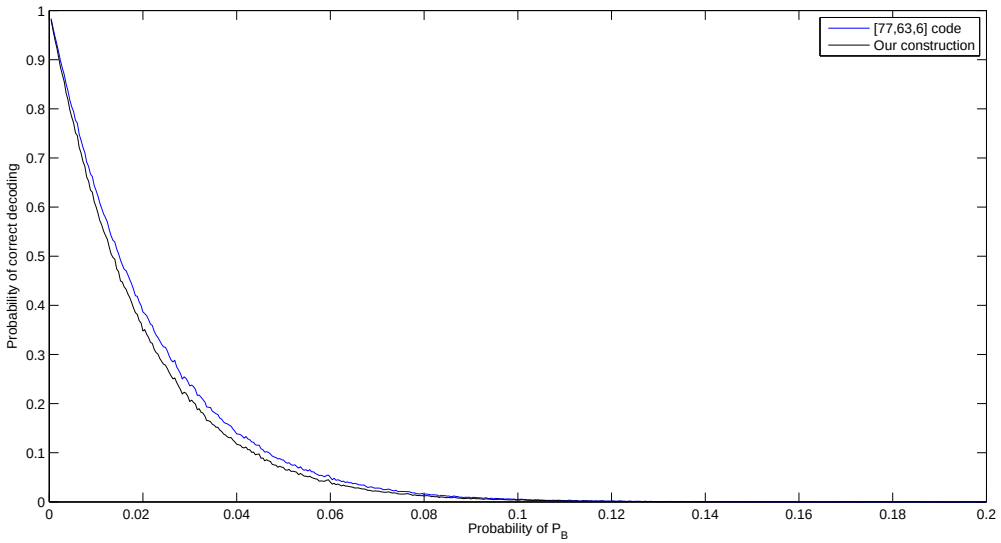


Figure 6.30: Comparison of $[77, 63]_2$ codes for a burst error channel with $P_G = 0.2$ with $P_B \leq 0.2$

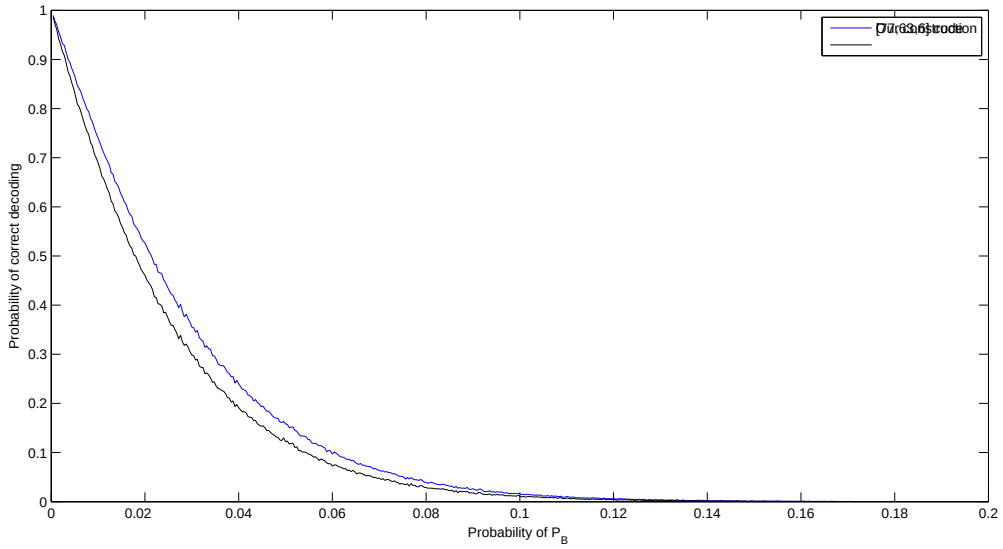


Figure 6.31: Comparison of $[77, 63]_2$ codes for a burst error channel with $P_G = 0.3$ with $P_B \leq 0.2$

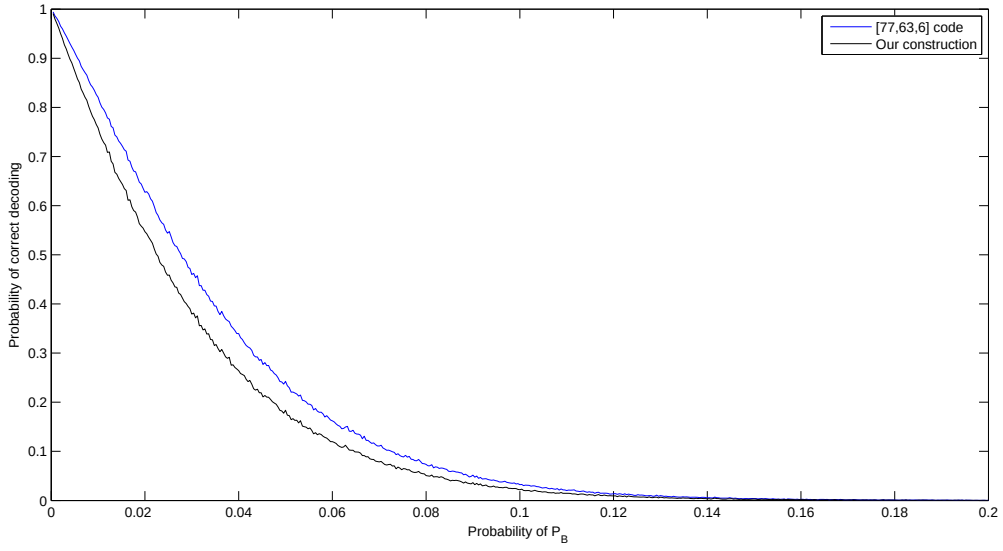
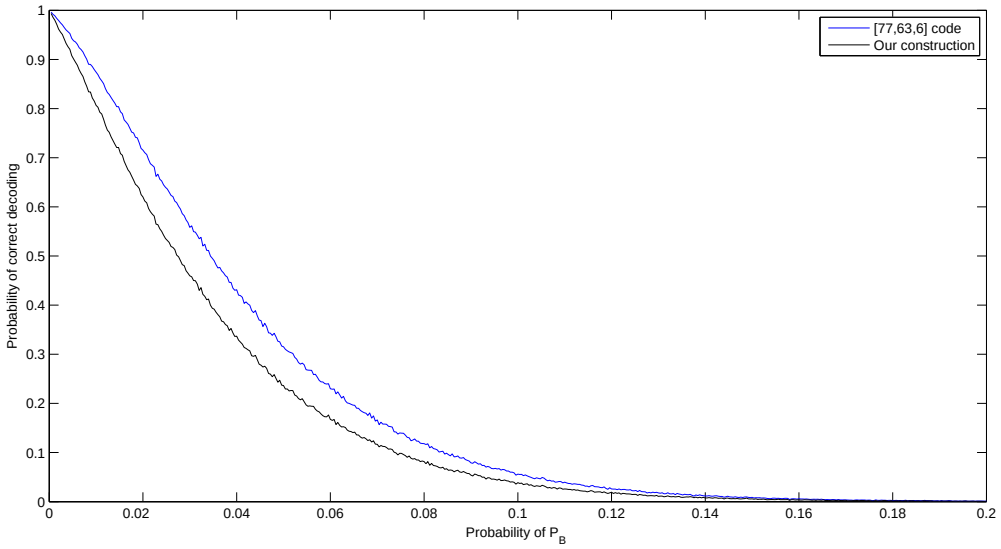


Figure 6.32: Comparison of $[77, 63]_2$ codes for a burst error channel with $P_G = 0.4$ with $P_B \leq 0.2$



As we can see from Figure 6.29, Figure 6.30, Figure 6.31 and Figure 6.32 our $[77, 63]_2$ construction does not compare favourably with the $[77, 63, 6]_2$ code, having a lower likelihood of a correct decoding for $P_G = 0.1$, $P_G = 0.2$, $P_G = 0.3$ and $P_G = 0.4$. We do however note that for lower values of P_G, the difference between the likelihood of a correct decoding for our construction and the $[77, 36, 6]_2$ code is smaller than it is for higher values of P_G, and thus we note that the "burstier" the error channel the more efficient our construction.

Now we briefly discuss the Golay codes, although we will not be able to test them for reasons which will become clear. The binary Golay code is a $[23, 12, 7]_2$ code, and thus this will induce a focused splitting over $(\mathbb{F}_2)^{23}$ with a perspective width of

$$\theta = 1 + \binom{23}{1} + \binom{23}{2} + \binom{23}{3} = 2048 = 2^{11}$$

and thus a perfect code P for our construction would have parameters

$$\left[\frac{\theta^m-1}{\theta-1}, \frac{\theta^m-1}{\theta-1}-m, 3\right]_\theta = \left[\frac{(2^{11})^2-1}{(2^{11})-1}, \frac{(2^{11})^2-1}{(2^{11})-1}-2, 3\right]_{(2^{11})} = [2049, 2047, 3]_{2048}$$

where we use $m = 2$ to get our value of T as short as possible. Subsequently $|D| = \theta^2 = 2^{22}$ and thus D has dimension 22, hence we could codes such as $[32, 22, 5]_2$ for D giving $\tilde{C}$ as a

$$[2049 \times 23 + 32, 2049 \times 23] = [47159, 47127]$$

code. We notice that $\tilde{C}$ would only be able to correct a maximum of 5 errors, that is if 3 errors occurred in a single block of 23 and if 2 errors occurred in the final 32 co-ordinates. We also note that this construction is in-feasible to be tested, as we do not have any information about codes of this size, and as the size is unwieldy.

The tertiary Golay code is a $[11, 6, 5]_3$ code, and thus will induce a focused splitting over $(\mathbb{F}_3)^{11}$ with a perspective width of

$$\theta = 1 + \binom{11}{1}(3-1) + \binom{11}{2}(3-1)^2 = 1 + 22 + 220 = 243 = 3^5$$

and thus a perfect code P for our construction would have parameters

$$\left[\frac{\theta^m-1}{\theta-1}, \frac{\theta^m-1}{\theta-1}-m, 3\right]_\theta = \left[\frac{(3^5)^m-1}{(3^5)-1}, \frac{(3^5)^m-1}{(3^5)-1}-m, 3\right]_{(3^5)} = [244, 242, 3]_{243}$$

where we use $m = 2$ to get our value of T as short as possible, with $T = 244$. Subsequently $|D| = \theta^2 = 3^{10}$ and thus D has dimension 10, hence we can see in [1] that there will at least exist a $[17, 10, 5]_3$ code which we could use for D giving $\tilde{C}$ as a

$$[244 \times 11 + 17, 244 \times 11] = [2701, 2684]$$

construction. We notice that $\tilde{C}$ would only be able to correct a maximum of 4 errors, that

is if 2 occurred in a specific block of 11 and if 2 occurred in D. We also note that this construction is in-feasible to be tested, as we do not have any information about codes of this size.

6.5 Linearly Independent Focused Splittings

We shall begin by working on general example for the class of focused splittings given in Theorem 2.26. Let $C = [n, k, d]_2$ with $S(0) \setminus \{0\}$ being linearly independent, then by Theorem 2.26 there exists a focused splitting on C if and only if there exists a perfect single-error correcting code on $(\mathbb{F}_2)^{\theta-1}$.

It is known [35] that for a positive integer m_1 that $\exists$ a perfect code on $(\mathbb{F}_2)^{\theta-1}$ when

$$\theta - 1 = \frac{2^{m_1} - 1}{2 - 1} = 2^{m_1} - 1 \Longleftrightarrow \theta = 2^{m_1}$$

So we have a focused splitting whenever $\theta = 2^{m_1}$.

Let $P = [T, \hat{k}, \hat{d}]_\theta$ be a perfect code. We know that for a positive integer m_2 we get

$$T = \frac{q^{m_2} - 1}{q - 1} = \frac{2^{m_1 m_2} - 1}{2^{m_1} - 1}$$

$$\hat{k} = T - m_2$$

$$\hat{d} = 3$$

and by Theorem 2.13 P induces a focused splitting on θ^T and thus,

$$(\mathbb{F}_\theta)^T = P_1 \sqcup P_2 \sqcup \ldots, P_{M'}$$

with $M' = \frac{\theta^T}{\theta^{T-m_2}} = \theta^{m_2} = 2^{m_1 m_2}$ and so the code D would have dimension $m_1 m_2$. As we can choose m_2, and we generally shall wish T to be as small as possible comparatively then we can choose $m_2 = 2$, given that the code C must have a perspective of $\theta = 2^{m_1}$,

then D will have dimension $2m_1$, and thus $C^T D$ will have length a little over $n \times T$.

We consider a code $C = [12, 5, 4]_2$ code with the following generator matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

which thus has a weight enumerator

$$1 + 3x^4 + 8x^5 + 8x^6 + 3x^8 + x^{12}$$

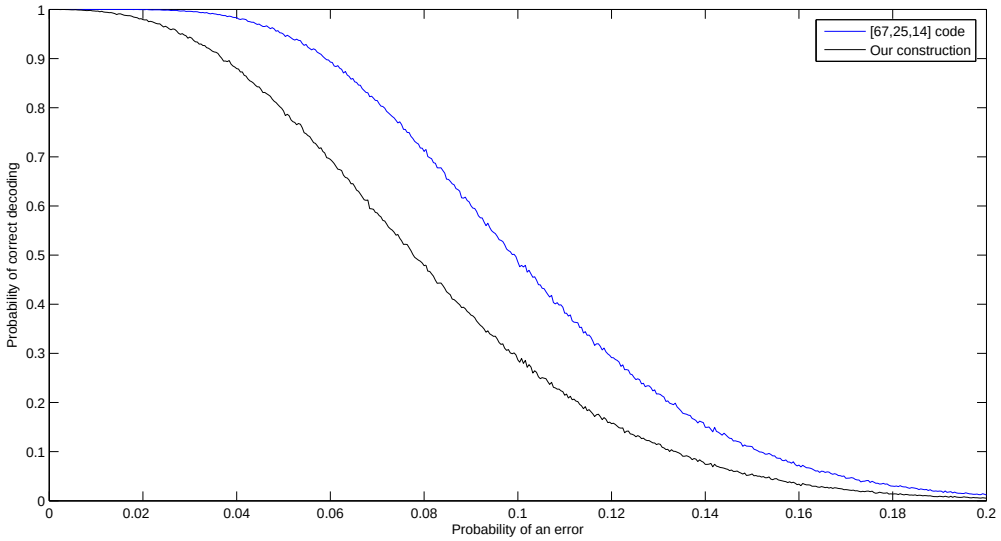
and so we can see that if we set $\delta = 5$ then $\theta = 4$ and then we can see that

$$S(0) \setminus \{0\} = \left\{ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \right\}$$

and it is easy to see that this set is linearly independent. Thus by Theorem 2.26 we can see that C has a focused splitting, since there exists perfect code on $(\mathbb{F}_2)^{\theta-1}$. We can thus see that the split codes will have a minimum distance of 5. As $\theta = 4$ we can construct a perfect code $P = [5, 3, 3]_4$ and thus we can calculate $|D| = \theta^{5-3} = 16 = 2^4$, thus we can use $D = [7, 4, 3]_2$ or $[11, 4, 5]_2$. These give us respectively a $[12 \times 5 + 7, 5 \times 5]_2 = [67, 25]_2$ construction and a $[12 \times 5 + 11, 5 \times 5]_2 = [71, 25]_2$ construction. We shall compare these to the known to exist codes [19]of $[67, 25, 14]_2$ and $[71, 25, 16]_2$. As the $[71, 25, 16]_2$ code corrects one more error than the $[67, 25, 14]_2$ code and our $[71, 25]$ construction will only correct one more error than our $[67, 25]$ construction, and only if that extra error is in D then any improvement of our construction over the block codes will be more prominent

in the comparison of the $[67, 25]$ codes, thus we shall only compare these.

Figure 6.33: Comparison of $[67, 25]_2$ codes for a random error channel up to an error probability of 0.2



As we can see in Figure 6.33 that our construction has a lower likelihood of a correct decoding than the $[67, 25, 14]$ code.

Figure 6.34: Comparison of $[67, 25]_2$ codes for a burst error channel with $P_G = 0.1$ with $P_B \leq 0.2$

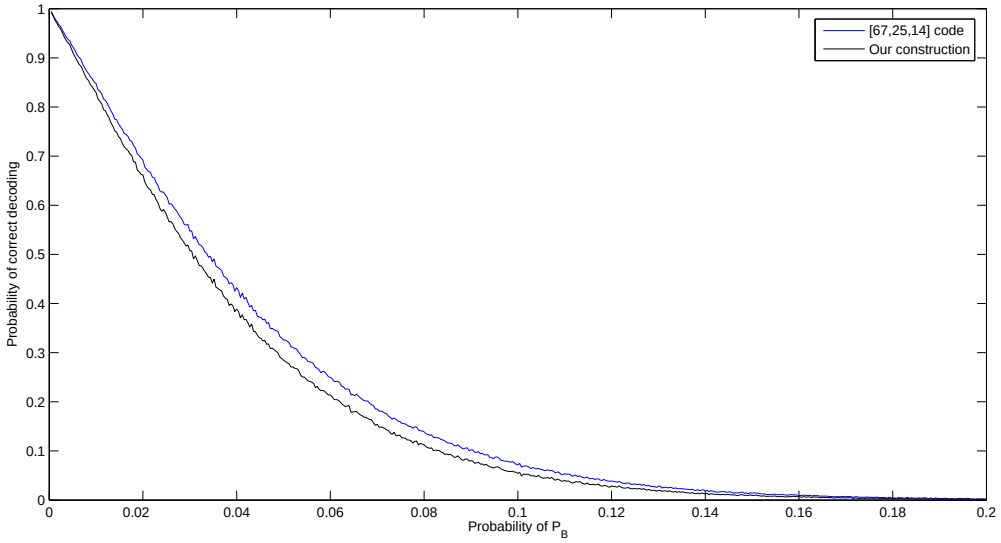


Figure 6.35: Comparison of $[67, 25]_2$ codes for a burst error channel with $P_G = 0.2$ with $P_B \leq 0.2$

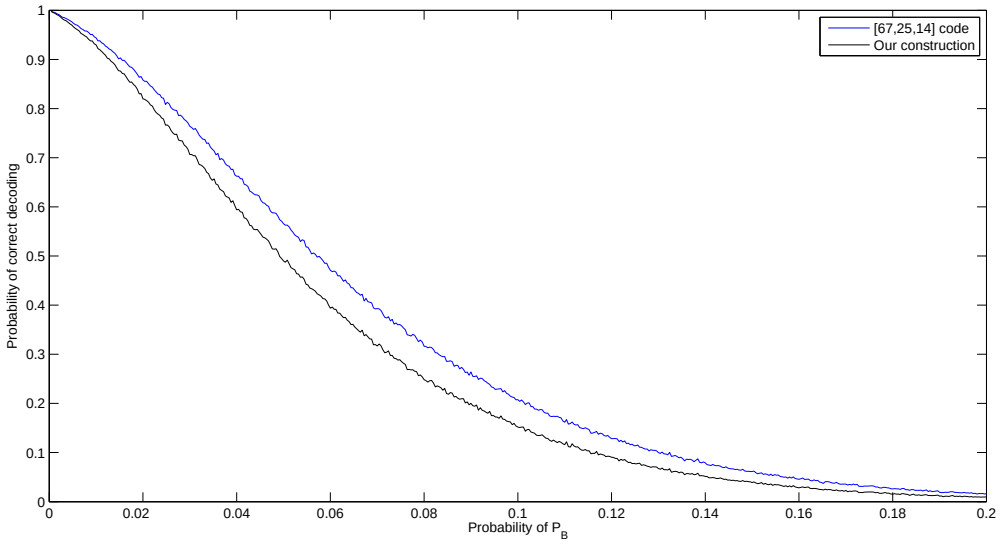


Figure 6.36: Comparison of $[67, 25]_2$ codes for a burst error channel with $P_G = 0.3$ with $P_B \leq 0.2$

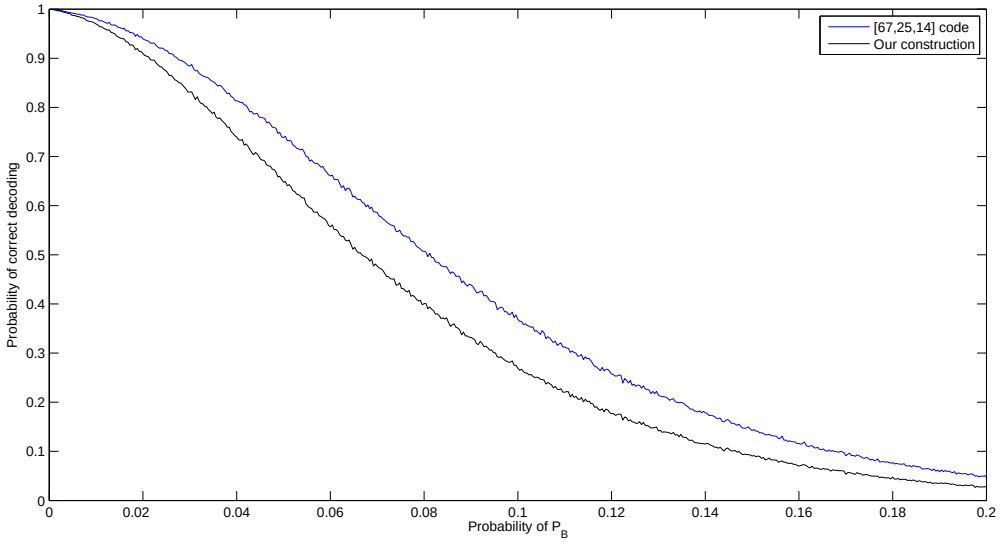
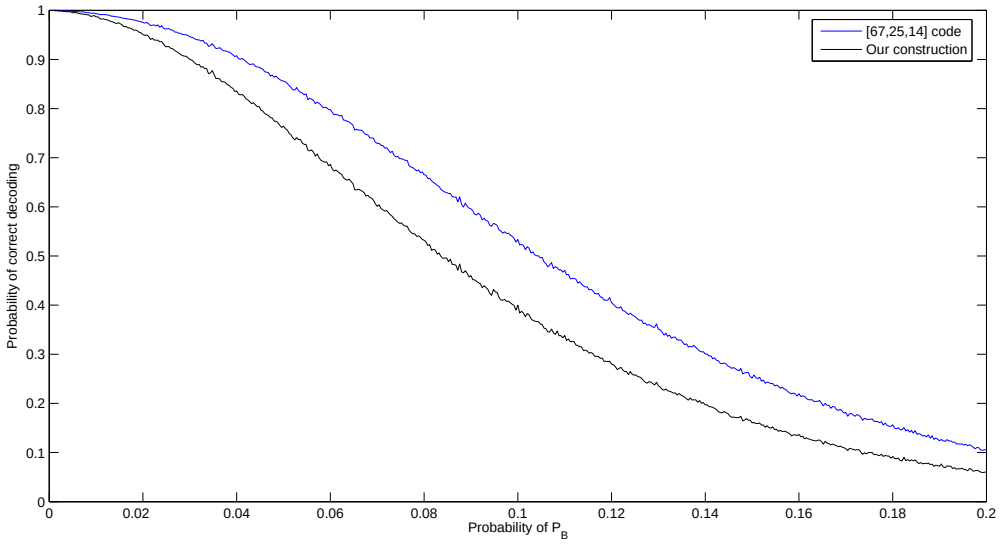


Figure 6.37: Comparison of $[67, 25]_2$ codes for a burst error channel with $P_G = 0.4$ with $P_B \leq 0.2$



As we can see in Figure 6.34, Figure 6.35, Figure 6.36 and Figure 6.37 our construction has a lower likelihood of a correct decoding than the $[67, 25, 14]$ code for all ranges of P_G that we explored. We do however note that the lower values of P_G, and thus the "burstier" channel, have a lesser difference in the likelihood of a correct decoding between the codes.

6.6 Concluding

We have seen that our construction can give codes which perform better than the best known codes of the same length and dimension, but that this also seems to be more prevalent for lower values of length. Our $[77, 36]$ construction performed better than the $[77, 36, 16]_2$ code under burst error channels, but not under a random error channel, and thus there is a potential for this construction to work well under burst error channels because of the sectioned nature of how we allow errors to be distributed if we are to correct them. Our $[22, 15]$ construction and our $[73, 63]$ construction both performed better than the relevant codes of the same length and dimension, and this shows that the focused splittings of whole spaces seemed to be good for the construction, this could be because

of the relatively short length of the codes involved as well as the relatively high value of δ comparatively, and thus a larger number of extra errors which could be corrected if favourably arranged. Some other constructions were less successful compared to the best known codes of the same length and dimension, but this is only to be expected.

We feel that this shows a potential and practical use of focused splittings in an error correcting capacity and we feel that this is a bonus to sit alongside the elegance of the theory.

Bibliography

[1] P.R.J. Östergård A.E. Brouwer, H.O. Hämäläinen and N.J.A. Sloane. Bounds on mixed binary/ternary codes. *IEEE Trans. Inform. Theory*, 44:140–161, 1998.

[2] E. F. Assmus and J. D. Key. *Designs and Their Codes*. Cambridge University Press, 1992.

[3] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate (corresp.). *Information Theory, IEEE Transactions on*, 20:284 – 287, 1974.

[4] C. Berrou, A. Glavieux, and P. Thitimajshima. Near shannon limit error-correcting coding and decoding: Turbo-codes. 1. In *Communications, 1993. ICC 93. Geneva. Technical Program, Conference Record, IEEE International Conference on*, pages 1064 – 1070, 1993.

[5] J. Bierbrauer. Authentication via algebraic-geometric codes. *Rend. Circ. mat. Palermo (2) Suppl.*, 51:139–152, 1998.

[6] I. Blake, C. Heegard, T. Høholdt, and V. Wei. Algebraic-geometry codes. *IEEE Trans. Inform. Theory*, 44:2596 – 2618, 1998.

[7] I.M Boyarinov and G.L. Katsman. Linear unequal error protection codes. *IEEE Trans. Inform. Theory*, 27:168–175, 1981.

[8] A.E. Brouwer and T. Verhoeff. An updated table of minimum-distance bounds for binary linear codes. *IEEE Trans. Inform. Theory*, 39:662 – 677, 1993.

[9] J. H. Conway and N. J. A. Sloane. *Sphere packings, lattices and groups*, volume 290 of *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*. Springer-Verlag, third edition, 1999. With additional contributions by E. Bannai, R. E. Borcherds, J. Leech, S. P. Norton, A. M. Odlyzko, R. A. Parker, L. Queen and B. B. Venkov.

[10] S. Cook. The complexity of theorem-proving procedures. *Proceedings of the Third Annual ACM Symposium on Theory of Computing.*, pages 151–158, 1971.

[11] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to algorithms*. MIT Press, third edition, 2009.

[12] E.O. Elliot. Estimates of error rates for codes on burst-noise channels. *Bell System Technical Journal*, 42:1977 – 1997, 1963.

[13] T. Etzion and A. Vardy. Perfect binary codes: Constructions, properties, and enumeration. *IEEE Trans. Inform. Theory*, 40:754 – 763, 1994.

[14] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completness*. W. H. Freeman, 1979.

[15] E.N. Gilbert. Capacity of a burst-noise channel. *Bell System Technical Journal*, 39:1253 –1265, 1960.

[16] G.R. Grimmett and D.R. Stirzaker. *Probability and random processes (Third edition)*. Oxford University Press, 2001.

[17] R. Hill. *A First Course in Coding Theory*. Oxford Applied Mathematics and Computing Science Series, 2002.

[18] J. W. P. Hirschfeld. *Projective geometries over finite fields.* The Clarendon Press Oxford University Press, second edition, 1998.

[19] David B. Jaffe. Information about binary linear codes.

[20] G.D. Forney Jr. Convolutional codes I: Algebraic structure. *IEEE Trans. Inform. Theory*, Vol. IT-16:720–738, 1970.

[21] H.W. Lenstra Jr. Two theorems on perfect codes. *Discrete Math.*, 3:125–132, 1972.

[22] R.M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, 1972.

[23] S. C. Kleene. Recursive predicates and quantifiers. *Transactions of the American Mathematical Society*, 53:41–73, 1943.

[24] W. Ledermann. *Introduction To Group Theory.* Longman Scientific and Technical, 1991.

[25] D.J.C. MacKay. Good error-correcting codes based on very sparse matrices. *IEEE Trans. Inform. Theory*, 45:399 – 431, 1999.

[26] M. Mushkin and I. Bar-David. Capacity and coding for the Gilbert-Elliot channels. *IEEE Trans. Inform. Theory*, 35:1277 – 1290, 1989.

[27] G.R. Redinbo and W.Y. Cheung. The design and implementation of unequal error-correcting coding systems. *IEEE Trans. Comm.*, 30, 1982.

[28] J. Roos. An algebraic study of group and nongroup error-correcting codes. *Inform Control.*, 8:195 – 214, 1965.

[29] A. Sánchez-Arroyo. Determining the total colouring number is NP-hard. *Discrete Math.*, 78:315 – 319, 1989.

[30] A. Sárközy and G.N. Sárközy. On the size of partial block designs with large blocks. *Discrete Math.*, 305:264–275, 2005.

[31] Y. Song and Z. Li. Secret sharing with a class of minimal linear codes. http://arxiv.org/pdf/1202.4058.

[32] M.A. Tsfasman and S.G. Vlăduţ. Geometric approach to higher weights. *IEEE Trans. Inform. Theory*, 41:1564–1588, 1995.

[33] J.H. van Lint. On the nonexistence of perfect 2- and 3-Hamming-error-correcting code over GF(q). *Inform. Control.*, 16:396–401, 1970.

[34] J.H. van Lint. Nonexistence theorems for perfect error-correcting codes. *Computers in Algebra and Number Theory, in: SIAM-AMS Proceedings*, 4:89–95, 1971.

[35] J.H. van Lint. A survey of perfect codes. *Rocky Mountain J Math.*, 5:199 – 224, 1975.

[36] J.H. van Lint. *Introduction To Coding Theory*. Springer, 1982.

[37] A.J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans. Inform. Theory*, 13:260 – 269, 1967.