2018

The Equivocation of Codes

Schofield, Mark

The Equivocation of Codes

by

MARK SCHOFIELD

Ph.D.

July 2017

The Equivocation of Codes

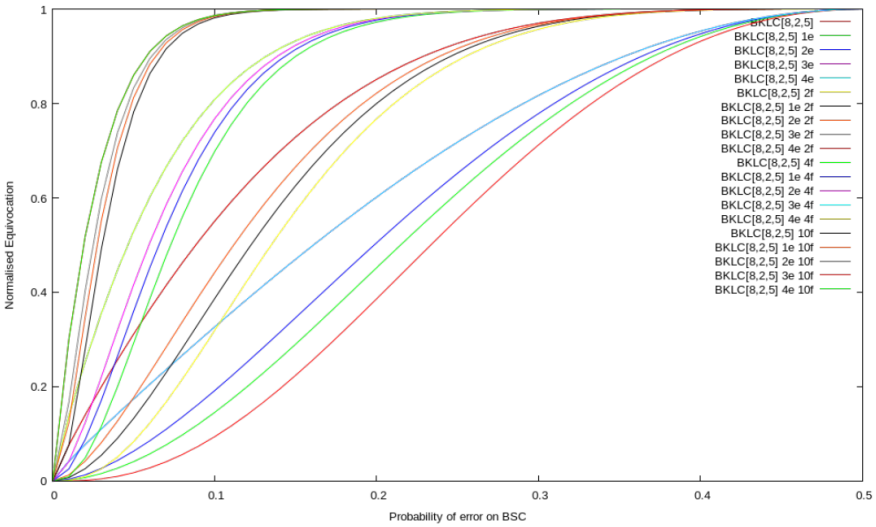
by

MARK SCHOFIELD

A thesis submitted to Plymouth University

in partial fulfilment for the degree of

DOCTOR OF PHILOSOPHY



School of Computing, Electronics and Mathematics

Faculty of Science and Technology

Plymouth University, UK

July 2017

Acknowledgements

FIRST and foremost I must thank my three supervisors, Assoc. Prof Zaki Ahmed, Prof. Martin Tomlinson and Prof. Ingo Stengel. Without doubt, I could not have done this without them. Watching me start and gradually develop must have been like a parent watching their toddler start to crawl - providing a steadying hand where needed, whilst wondering when on earth will they ever learn to walk and perhaps one day run. It has been a long but deeply enjoyable and satisfying walk, with occasional too-brief bouts of jogging. I'm not sure that I have ever run, but thank you all for your unwavering support along the way. Zaki - your drive and passion for your field are inspirational. May all your ventures be successful and your coffee beans always fresh. Martin, you always make me think of the android Marvin in the Hitch-hikers Guide to the Galaxy. Not in any way because of paranoia but because you truly have a brain the size of a planet. Your insight, knowledge and experience are unparalleled - my toddler will forever be in awe of your Olympian. Ingo, thank you for being a voice of calm and reason - always willing to give your time and keep me grounded. I will miss our chats.

To my wife Sue, you are everything to me: best friend, supporter, motivator, advisor. You set standards in every way, every day. I know of no-one whose moral-compass is more truly aligned - if ever I need to know what is the right thing to do, I ask myself what you would do and I have my answer.

As my studies progressed, it became ever more apparent just how truly great are the minds of those whose shoulders I have stood upon. The clarity of insight required just to understand some of the concepts in this field is great. To have seen so much of the field at its point of inception as those like Claude Shannon did, will remain for me as out-of-reach and unimaginable as trying to comprehend and quantify the limitless size of the night sky. But that won't stop me from always trying.

Author's declaration

At no time during the registration for the degree of Doctor of Philosophy has the author been registered for any other University award. Work submitted for this research degree at Plymouth University has not formed part of any other degree either at Plymouth University or at another establishment. Two papers were accepted for publication and presentation at conferences.

Word count for the main body of this thesis: **47216**

Signed: _____

Date: _____

Publications:

Schofield, M. et al. *Using Parallel Processing to Calculate and Improve Code Equivocation* Proceedings of the 16th IEEE International Conference on Communication Technology 2015 pp510-514.

Schofield, M. et al. *Intentional Erasures and Equivocation on the Binary Symmetric Channel* International Computer Symposium 2016 2016, pp233-235.

Abstract

EQUIVOCATION was introduced by Shannon in the late 1940's in seminal papers that kick-started the whole field of information theory. Much ground has been covered on equivocation's counterpart, channel capacity and in particular, its bounds. However, less work has been carried out on the evaluation of the equivocation of a code transmitted across a channel.

The aim of the work covered in this thesis was to use a probabilistic approach to investigate and compare the equivocation of various codes across a range of channels. The probability and entropy of each output, given each input, can be used to calculate the equivocation. This gives a measure of the ambiguity and secrecy of a code when transmitted across a channel. The calculations increase exponentially in magnitude as both the message length and code length increase. In addition, the impact of factors such as erasures and deletions also serve to significantly complicate the process.

In order to improve the calculation times offered by a conventional, linearly-programmed approach, an alternative strategy involving parallel processing with a CUDA-enabled (Compute Unified Device Architecture) graphical processor was employed. This enabled results to be obtained for codes of greater length than was possible with linear programming. However, the practical implementation of a CUDA driven, parallel processed solution gave rise to significant issues with both the software implementation and subsequent platform stability.

By normalising equivocation results, it was possible to compare different codes under different conditions, making it possible to identify and select codes that gave a marked difference in the equivocation encountered by a legitimate receiver and an eavesdropper.

The introduction of code expansion provided a novel method for enhancing equivocation differences still further. The work on parallel processing to calculate equivocation and the use of code expansion was published in the following conference:
Schofield, M., Ahmed, M. & Tomlinson, M. (2015), Using parallel processing to calculate and improve equivocation, *in* 'IEEE Conference Publications - IEEE 16th International Conference on Communication Technology'. In addition to the novel use of a CUDA-enabled graphics process to calculated equivocation, equivocation calculations were also performed for expanded versions of the codes. Code expansion was shown to yield a dramatic increase in the achievable equivocation levels.

Once methods had been developed with the Binary Symmetric Channel (BSC), they were extended to include work with intentional erasures on the BSC, intentional deletions on the BSC and work on the Binary Erasure Channel (BEC). The work on equivocation on the BSC with intentional erasures was published in:
Schofield, M. et al, (2016), Intentional erasures and equivocation on the binary symmetric channel, *in* 'IEEE Conference Publications - International Computer Symposium', IEEE, pp 233-235. The work on the BEC produced a novel outcome

due to the erasure correction process employed. As the probability of an erasure occurring increases, the set of likely decoded outcomes diminishes. This directly impacts the output entropy of the system by decreasing it, thereby also affecting the equivocation value of the system. This aspect was something that had not been encountered previously.

The work also extended to the consideration of intentional deletions on the BSC and the Binary Deletion Channel (BDC) itself. Although the methods used struggled to cope with the additional complexity brought by deletions, the use of Varshamov-Tenengolts codes on the BSC with intentional deletions showed that family of codes to be well-suited to the channel arrangement as well as having the capability to be extended to enable the correction of multiple deletions.

Contents

List of Figures

Chapter 1

Introduction

1.1 Contribution to Knowledge

The equivocation of a code gives a measure of the ambiguity and secrecy of a code when transmitted across a channel.The work covered in this thesis took a probabilistic approach to investigating and comparing the equivocation of various codes across a range of channels. The work used novel approaches to perform the calculations and generated some interesting results that enabled the direct comparison of a range of different codes.

The key points of novel work and contributions to knowledge are summarised below:

- Equivocation calculations were performed on a general purpose computer but used a CUDA-(Compute Unified Device Architecture) enabled graphics processor to perform the calculations in parallel, providing sigificant calculation run-time improvements over their linear counteparts (in some cases, up to 35 times faster).
- The calculations performed enabled a direct comparison of the equivocation levels of a range of codes and, in particular, perfect and best known linear codes. This permitted the identification of codes that give practical and useful outcomes such as a large differential between a low equivocation level for a legitimate user and a high equivocation level for an eavesdropper on a wiretap channel.
- The use of code expansion on a channel can dramatically increase the equiovcation level of the code. Code choice and level of expansion can then be tailored to increase the differential between the equivocation for the legitmate receiver and

an eavesdropper. The novel work on parallel processing to calculate equivocation and the use of code expansion was published by Schofield et al. (2015).

- The calculation methods used for the Binary Symmetric Channel (BSC) were extended to modifications of the BSC and to other channels. Channels investigated included the BSC with intentional erasures (IE+BSC), the BSC with intentional deletions (ID+BSC), Binary Erasure Channel (BEC) and Binary Deletion Channel (BDC). The work on parallel processing to calculate equivocation and the use of code expansion was published by Schofield et al. (2016).

- The erasure correction method used on the BEC gave rise to an interesting situation where a diminishing set of output options triggered a decreasing output entropy, something that had not been encountered previously.

- Varshamov-Tenengolts codes $VT_a(n)$ consist of all binary vectors $(x_1,....,x_n)$ satisfying $\sum_{i=1}^{n} ix_i \equiv a(\text{mod}(n+1))$. It was shown that for $VT_0(n)$ codes with $n = 2^m - 1$, $m \geq 2$, the code will have the same number of codewords as the Hamming$(m,2)$ code (where $m = n-k$), described by the properties Ham$[2^m-1,2^m-m-1,3]$ or Ham$[n,k,3]$, i.e. $|VT_0(n)| = |\text{Ham}[n,k,3]|$.

- The use of Varshamov-Tenengolts codes on the BSC with intentional deletions showed that family of codes to be well-suited to the channel arrangement as well as having the capability to be extended to enable the correction of multiple deletions.

1.2 Basic Principles

Information Theory is defined (Britannica.com 2017) as:

> a mathematical representation of the conditions and parameters affecting the transmission and processing of information

and covering applications including:

- Data compression
- Error-correcting and error-detecting codes
- Cryptology
- Linguistics
- Algorithmic information theory
- Physiology
- Physics

This thesis focuses primarily on understanding and developing an aspect of error-correcting coding called equivocation, which gives a measure of the ambiguity of a signal. By calculating the equivocation of different codes under varying channel arrangements, it is possible to identify codes that provide greater levels of secrecy for the users.

To communicate information from one location (the *source*) to another (the *sink*), the information can be transmitted via a channel to form a signalling system. If the information to be sent across the system must possess certain characteristics, such as secrecy or an ability for errors to be detected or corrected, then it may be necessary to encode the data prior to transmission and then decode it upon receipt. A typical signalling system can therefore be represented as shown in Figure 1.1 (Hamming 1980, p. 4).

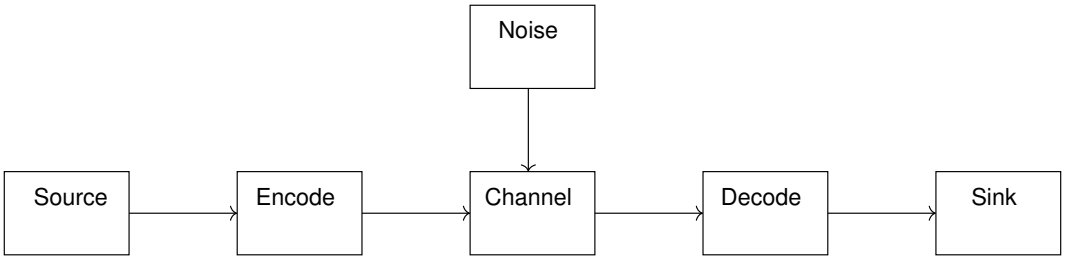


Figure 1.1: A Typical Signalling System

In practice, any channel used for transmitting the data is unlikely to be perfect and is likely to introduce an element of noise to the transmitted data. The noise introduced to the transmission channel can affect whether the encoded data is received correctly so that it can be correctly decoded. Dependent on requirements for the data, the errors introduced by a noisy channel may need to be either detected or corrected.

In this context, a code is a specified system of symbols that are purposefully used to represent other symbols. In many systems, particularly those based on electronic components with binary states, the simplest and most efficient choice of symbols is to use the digits 0 and 1 to represent the data.

Many codes exist that enable the detection and correction of data, each with different properties and characteristics. There are correspondingly many ways of comparing the codes and their relative capabilities. Transmission channels can involve different sources and types of noise that may introduce errors that are random, regular or that come in bursts. Codes that work well in certain situations may be of little or no value in others. For example, an error-correcting code that gives good results on the Binary Symmetric Channel may be unable to correct deletions and therefore may be ineffective at dealing with deletions on the Binary Deletion Channel or on a modified Binary Symmetric Channel with intentional deletions.

In Chapter 2 a range of different channels are introduced and discussed, including the general Information Channel, the Binary Symmetric Channel, Binary Erasure Channel, Binary Deletion Channel, Binary Symmetric Erasure Channel and the WireTap Channel. The equivocation of codes transmitted via the these channels and modifications

of them are calculated in order to compare the levels of secrecy offered by different codes. Equivocation and capacity are two closely related measures that are linked to the amount of information contained by a code and its entropy. Considerable work has been done to establish upper boundaries for the capacity of different channel arrangements. For example, work on the capacity bound on the Binary Deletion Channel has been completed by Dalai (2011), Mitzenmacher (2006), Kalai et al. (2010), Kanoria & Montanari (2010), Kirsch & Drinea (2007), Fertonani & Duman (2010). However much less work has been undertaken on the practical calculation of equivocation for different codes with different channel arrangements. This thesis aims to address that issue. Equivocation gives a measure of the level of ambiguity of a code across a channel and hence its secrecy. By looking at equivocation levels for different error probabilities, it is possible to compare the equivocation for a legitimate receiver with a lower error probability against an illegitimate receiver with a higher error probability.

Calculation of equivocation is a numerically intensive process. To assist this process, software was written to perform the many calculations in parallel, enabling a reduction of time taken to perform the calculation. This was achieved by using the parallel processing capabilities of a graphics card, specifically an Nvidia-made card that was built around the Compute Unified Device Architecture (CUDA) (Nvidia 2015). This enables use of the Graphical Processing Unit (GPU) for more general purpose uses. CUDA is a parallel computing platform and programming model created by Nvidia. It is implemented by computers with CUDA-capable Nvidia GPUs, with the main CPU acting as the 'host' for the linear component of a program which then delegates responsibility for running parallelised sections of code to the GPU 'device' (Sanders & Kandbrot 2011). The task of efficient delegation and management of resources is largely done by the CUDA architecture and programming model. Graphics cards have seen dramatic increases in use across several fields, especially that of crypto-coin mining where recent surges in have led to shortages, price spikes and even rationing of device sales (BBC 2018). Other parallel methods could have been used such as the use of multi-core processors or the use of multiple, linked processors but CUDA was used as a novel

method in this field.

The parallelisation of the calculation by using GPUs can deliver significant performance benefits, either allowing equivocation values to be found more quickly or for longer codes.

Once the equivocation values for different codes have been found and normalised, they can be compared. This enables codes with higher levels of normalised equivocation to be identified and improvements to existing codes to be implemented.

This thesis is presented in 8 chapters.

- The remainder of Chapter 1 is given to introducing a few necessary preliminaries about fields and matrices. Whilst the use of tools such as elementary row operations may not necessarily be explicit or apparent in subsequent chapters, their use is common within the software implementation. The information systems discussed in this thesis will all be based on the transmission of binary symbols, represented by 0 and 1. To effectively operate on pairs of binary symbols, a very brief overview of the underlying algebraic structure of fields will be given. The manipulation of multi-element, extended fields through the use of field polynomials and matrices is also commonplace in this work. Examples of this include tasks such as multiplying the binary elements of an k-bit message by a $k \times n$ generator matrix to produce a codeword or the multiplication of a codeword by the transpose of a parity check matrix to produce a syndrome for the codeword.
- Chapter 2 looks briefly at some types of channel that were investigated during the research for this thesis or that are expected to be of use in subsequent work.
- Chapter 3 looks at some codes and their properties, including the relationship between codes and the geometries associated with sphere-packing.
- Chapter 4 develops the ideas behind equivocation, including information and entropy.

- Chapter 5 gives details of a novel approach involving the practical implementation of a method to calculate equivocation using parallel processing. The new work in Chapter 5 was published in Schofield et al. (2015).
- Chapter 6 extends the work in Chapter 5 to give a new examination of the equivocation of codes containing erasures on both the Binary Symmetric and Binary Erasure Channels.The new work in Chapter 6 was published in Schofield et al. (2016)
- Chapter 7 extends the work further to give a novel consideration of the equivocation of a Binary Symmetric Channel containing intentional deletions. A relationship between the number of codewords in a Hamming code and a Varshamov-Tenengolts code is established and Varshamov-Tenengolts codes are extended to enable the correction of multiple deletions.
- Chapter 8 concludes the thesis, highlighting and suggesting some areas to investigate in future research.

1.3 Information Theoretic Secrecy

In his seminal paper, Shannon (1949) discusses three general types of secrecy system:

1. Concealment systems - the existence of the message is hidden from the eavesdropper
2. Privacy systems - special equipment is required to recover the message
3. Secrecy systems - the meaning of the message is concealed by cipher, code etc.

In a secrecy system, the existence of the message is not hidden and any eavesdropper is assumed to have the equipment needed to intercept and record the transmitted signal. This work in this thesis is based upon the use of codes both as a secrecy system but also for the detection and correction of errors, erasures or deletions.

Shannon defined perfect secrecy as:

> requiring of a system that after a cryptogram is intercepted by the enemy, the *a posteriori* probabilities of this cryptogram representing various messages be identically the same as the *a priori* probabilities of the same messages before the interception.

In other words, with perfect secrecy, the enemy or eavesdropper is no better off after intercepting any amount of material than before. For example a one-time pad, where each character of the message is paired using modular addition with a random secret key that is shared by the sender and receiver. Even if part of the message was known to be "attack at d - - -", it wouldn't be possible to decode the final 3 characters unless the key for each of the letters was known.The attack could be at any of dawn, dusk, dark etc. The one-time pad also offers *information theoretic security* i.e. its robustness comes purely from the principals of information theory and is unable to be broken even with unlimited computing power.

1.4 Fields

1.4.1 Introduction

Whilst data may take many forms and may be represented by many different symbols, this thesis deals solely with binary data, represented by the two symbols 0 and 1. To operate with and manipulate binary data throughout this work, it is useful to touch very briefly upon some of the theory of both algebraic number fields and matrices.

Fields are algebraic structures consisting of a set with two operations, usually called addition (+) and multiplication ($\times$, or omitted) that satisfy certain axioms (Hill 1986). Subtraction is by adding with the additive inverse. The additive inverse of an element a is another element b such that the direct sum $a \oplus b$ is zero. The multiplicative inverse of an element a is another element b such that the tensor product $a \otimes b$ is one. Division is by multiplying with the multiplicative inverse.

The finite field $\mathbb{F}_i$ has i elements $0, 1, ...(i-1)$ and is also called the ring of integers, modulo i or the **G**alois **F**ield of i, **GF**(i). The binary field **GF**(2) consisting of the two

elements 0 and 1 is the smallest finite field. In **GF**(2), for addition, $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$ and $1 \oplus 1 = 0$ and for multiplication, $0 \otimes 0 = 0$, $0 \otimes 1 = 0$, $1 \otimes 0 = 0$ and $1 \otimes 1 = 1$. Galois Fields exist for prime numbers only and can be extended to powers of primes. A primitive element in a base field is an element of the Galois Field whose powers result in all the non-zero elements of the Galois Field. E.g. **GF**(5) has primitive elements 2 and 3 because:

$$2^0 = 1,\ 2^1 = 2,\ 2^2 = 4,\ 2^3 = 3\ (\text{and } 2^4 = 1)$$

and

$$3^0 = 1,\ 3^1 = 3,\ 3^2 = 4,\ 3^3 = 2\ (\text{and } 3^4 = 1)$$

But 4 is not a primitive element, since:

$$4^0 = 1, 4^1 = 4, 4^2 = 1, 4^3 = 4, 4^4 = 1$$

and therefore not all non-zero elements of the field have been generated. This is supported by the modulo-5 multiplication and addition tables in Table 1.1 and Table 1.2.

$\times$	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Table 1.1: Multiplication modulo 5

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

Table 1.2: Addition modulo 5

1.4.2 Extended Fields

Extended fields can be seen as a vector of a base field. E.g. **GF**(2^3) (which can also be represented by the notation $\mathbb{F}_2^3$) is an extended field of **GF**(2) with elements {0, 0, 0}, {0, 0, 1}, {0, 1, 0}, {0, 1, 1}, {1, 0, 0}, {1, 0, 1}, {1, 1, 0} and {1, 1, 1}. The zero element of the extended field is {0, 0, 0} and a primitive element is α. All other elements are obtained from α.

1.4.3 Field Polynomials

A primitive element α in a **GF**(p^q) Galois Field has the property that the polynomial $\alpha^{p^q-1}-1=0$, where p is the number of elements in the base field and must be a prime number and q is the number of elements in the vector of the extended field. This polynomial will be zero if any of the factors in the equation are zero. Factorising it should give factors that are both irreducible and primitive. For example (Sweeney 2002), in **GF**(2^3), i.e. when $p=2$ and $q=3$:

$$\alpha^{2^3-1}-1=\alpha^7-1=0 \tag{1.1}$$

but in modulo 2,

$$\alpha^7-1=(\alpha+1)(\alpha^3+\alpha+1)(\alpha^3+\alpha^2+1) \tag{1.2}$$

therefore

$$(\alpha+1)(\alpha^3+\alpha+1)(\alpha^3+\alpha^2+1)=0 \tag{1.3}$$

and thus at least one of these factors in parentheses must also be equal to zero. So, if $\alpha^3+\alpha+1=0$, then

$$\implies \alpha^3=-\alpha-1=\alpha+1$$
$$\implies \alpha^4=\alpha\alpha^3=\alpha(\alpha+1)=\alpha^2+\alpha$$

$$\implies \alpha^5 = \alpha\alpha^4 = \alpha(\alpha^2+\alpha) = \alpha^3+\alpha^2 = \alpha^2+\alpha+1$$
$$\implies \alpha^6 = \alpha\alpha^5 = \alpha(\alpha^2+\alpha+1) = \alpha^3+\alpha^2+\alpha = \alpha+1+\alpha^2+\alpha = \alpha^2+1$$
$$\implies \alpha^7 = \alpha\alpha^6 = \alpha(\alpha^2+1) = \alpha^3+\alpha = \alpha+1+\alpha = 1$$

or $\alpha^7 - 1 = 0$ as required

This enables all elements to be represented as polynomials, shown in Table 1.3. It also confirms that $\alpha^{2^3-1} - 1 = \alpha^7 - 1 = 0$.

Element	Polynomial	Vector	Value
0	0	{0, 0, 0}	0
α^0	$x^0 = 1$	{0, 0, 1}	1
α^1	$x^1 = x$	{0, 1, 0}	2
α^2	x^2	{1, 0, 0}	4
α^3	$x+1$	{0, 1, 1}	3
α^4	x^2+x	{1, 1, 0}	6
α^5	x^2+x+1	{1, 1, 1}	7
α^6	x^2+1	{1, 0, 1}	5

Table 1.3: Polynomials in **GF**(2^3)

1.5 Matrices

1.5.1 Matrices

The use of matrix multiplication is commonplace throughout this work. Messages and codewords can be represented by $1 \times n$ matrices (row vectors of length n) and the full set of codewords that comprise a linear code can be generated from messages by the multiplication of the message by a $k \times n$ matrix called a generator matrix. Therefore the ability to efficiently manipulate and, in particular, multiply matrices is a key capability in this work.

An n by m matrix **A** is an ordered set of $n \times m$ elements in a rectangular array of n rows and m columns:

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix}$$

1.5.2 Adding Matrices

To add two $(n \times m)$ matrices **A** and **B**, add corresponding elements of each matrix.

$$\begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix} + \begin{bmatrix} b_{11} & \cdots & b_{1m} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nm} \end{bmatrix} = \begin{bmatrix} a_{11}+b_{11} & \cdots & a_{1m}+b_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1}+b_{n1} & \cdots & a_{nm}+b_{nm} \end{bmatrix}$$

1.5.3 Multiplying Matrices

The product of an $(n \times m)$ matrix **A** and an $(m \times p)$ matrix **B** has order $(n \times p)$.

$$\begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix} \begin{bmatrix} b_{11} & \cdots & b_{1p} \\ \vdots & \ddots & \vdots \\ b_{m1} & \cdots & b_{mp} \end{bmatrix} = \begin{bmatrix} \Sigma_{i=1}^{m} a_{1i}b_{i1} & \cdots & \Sigma_{i=1}^{m} a_{1i}b_{ip} \\ \vdots & \ddots & \vdots \\ \Sigma_{i=1}^{m} a_{ni}b_{i1} & \cdots & \Sigma_{i=1}^{m} a_{ni}b_{ip} \end{bmatrix}$$

1.5.4 Matrix Metrics

A *basis* B of a vector space V over a field $\mathbb{F}$ is a linearly independent subset of V that spans V ie the basis set of linearly independent vectors that, in a linear combination, can represent every vector in a given vector space (Moon 2005). The *row space* of an $n \times m$ matrix **M** is the set of all linear combinations of row vectors of **M**:

$$c_1 r_1 + c_2 r_2 + \ldots$$

where c_i are scalars and r_i are row vectors of **M**. The row vectors form a subspace of the vector space of m-tuples. The *row rank* is the dimension of the row space. For example the row space of the binary matrix:

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

includes the vectors: $\begin{pmatrix}1 & 1 & 0 & 1\end{pmatrix}$, $\begin{pmatrix}0 & 1 & 1 & 0\end{pmatrix}$, $\begin{pmatrix}0 & 0 & 0 & 1\end{pmatrix}$, $\begin{pmatrix}1 & 0 & 1 & 1\end{pmatrix}$, $\begin{pmatrix}1 & 1 & 0 & 0\end{pmatrix}$ and $\begin{pmatrix}0 & 1 & 1 & 1\end{pmatrix}$. The *column space* is the set of all linear combinations of column vectors of the matrix and the *column rank* is the dimension of the column space. The *matrix rank* can be found from either the row rank or the column rank of the matrix.

1.5.5 Elementary Matrices

Elementary row operations for matrices include the:

- Interchange of any two rows
- Multiplication of any row by a non-zero field element
- Addition of any multiple of one row to another

A matrix is *non-singular* if all the rows of an $n \times m$ matrix are linearly independent.

An *identity* matrix, **I** consists of 1's on the leading diagonal and 0's elsewhere. e.g.

$$\mathbf{I}_4 = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

The *transpose* of an $n \times m$ matrix $[a_{ij}]$ is the $m \times n$ matrix $[a_{ij}]^T = [a_{ji}]$

An *elementary* matrix is a matrix which differs from the identity matrix by one single elementary row operation. Left multiplication (pre-multiplication) by an elementary matrix represents elementary row operations, while right multiplication (post-multiplication) represents elementary column operations. The elementary matrix:

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

has interchanged rows 2 and 3 of the identity matrix, while

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

has replaced row 3 with the sum of rows 3 and 4.

Chapter 2

Channels

Chapter 2 looks briefly at several different types of channel used within the thesis, including:

- The Information Channel
- The Binary Symmetric Channel (BSC)
- The Binary Erasure Channel (BEC)
- The Binary Deletion Channel (BDC)
- The Binary Symmetric Erasure Channel (BSEC)
- The Wire-Tap Channel (WTC)

Building on the basic theoretical model of the Information Channel, the Binary Symmetric, Erasure and Deletion Channels each affect the transmission of binary data differently. The BSEC is included for completeness in the chapter, but no equivocation values were calculated for it. The addition of a wire-tap channel to each scenario enables the comparison of the channel properties for a legitimate receiver on the main channel and an illegitmate recipient listening via the compounded properties of the main channel and the eavesdropper channel. The WTC is a cascaded channel, where the legitimate receipient receives the data after the first cascade and an eavesdropper receives it after the second cascade. In addition, several modifications to standard channels are considered, in particular on the BSC which is considered with the compounding factors of both intentional erasures in Chapter 6 and intentional deletions in

Chapter 7. All the channels investigated are taken to be *memoryless* i.e. each symbol is independent of what preceded it.

2.1 The Information Channel

The Information Channel represented in Figure 2.1 (Roman 1997) is a statistical model of the medium through which the signal passes (or is stored).

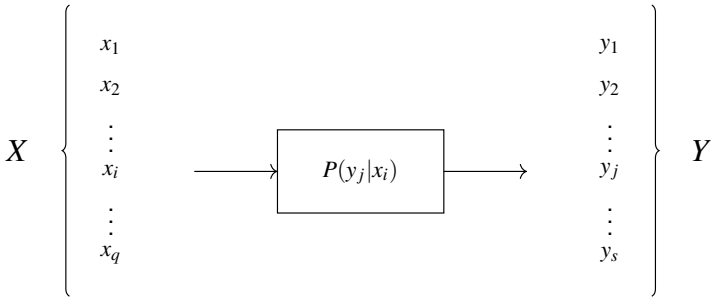


Figure 2.1: The Information Channel (Roman 1997)

A channel is described by a set of conditional probabilities $(P(y_j|x_i))$ $(1 \leq i \leq q,\ \ 1 \leq j \leq s)$ which are the probabilities that an input x_i from an alphabet of q letters will appear as some output y_j from an alphabet of s letters. q and s need not be the same. The channel is completely described by the matrix, **P** of conditional probabilities:

$$\mathbf{P} = (P(y_j|x_i))$$

$$\mathbf{P} = \begin{pmatrix} P(y_1|x_1) & P(y_2|x_1) & \ldots & P(y_j|x_1) & \ldots & P(y_s|x_1) \\ P(y_1|x_2) & P(y_2|x_2) & \ldots & P(y_j|x_2) & \ldots & P(y_s|x_2) \\ \vdots & \vdots & & \vdots & & \vdots \\ P(y_1|x_i) & P(y_2|x_i) & \ldots & P(y_j|x_i) & \ldots & P(y_s|x_i) \\ \vdots & \vdots & & \vdots & & \vdots \\ P(y_1|x_q) & P(y_2|x_q) & \ldots & P(y_j|x_q) & \ldots & P(y_s|x_q) \end{pmatrix}$$

- The i'th row corresponds to the i'th input symbol, x_i
- The j'th column corresponds to the j'th output symbol, y_j

- The sum of the elements in any row i $(1 \leq i \leq q)$ is always 1:

$$\sum_{j=1}^{s} P(y_j|x_i) = 1 \tag{2.1}$$

  For each input x_i we are certain that something will come out.

- If $P(x_i)$ is the probability of the symbol input x_i then:

$$\sum_{i=1}^{q} \sum_{j=1}^{s} P(y_j|x_i)P(x_i) = 1 \tag{2.2}$$

  When something is put into the system, we are certain that something comes out.

This supposes that the channel is *stationary* i.e. the probabilities do not change with time and the errors that occur are independent of each other.

Conditional probability rules give us that either:

$$P(x_i, y_j) = P(x_i)P(y_j|x_i) \tag{2.3}$$

or

$$P(x_i, y_j) = P(y_j)P(x_i|y_j) \tag{2.4}$$

By Bayes' Theorem,

$$P(x_i|y_j) = \frac{P(y_j|x_i)P(x_i)}{P(y_j)} \tag{2.5}$$

or

$$P(y_j|x_i) = \frac{P(x_i|y_j)P(y_j)}{P(x_i)} \tag{2.6}$$

$P(y_j|x_i)$ are known as the forward conditional probabilities, since they start at the front with the x_i given and express the probabilities of occurrence of the y_j. $P(x_i|y_j)$ are known as the backward conditional probabilities: given the output, what symbol caused it?

2.2 The Binary Symmetric Channel

The Binary Symmetric Channel (BSC) represented in Figure 2.2 (Hamming 1980, p. 133) is an idealised channel that relies on the 'cross-over' probability p_e of a different symbol being received than was transmitted for either transmitted symbol. Messages M are encoded as codewords X and transmitted across a noisy channel (Figure 2.3) that has a cross-over probability of p_e. The channel is described as symmetric because the cross-over error probability is the same for each of the input symbols. Estimates Y of the encoded message are received and decoded to estimates of the original message, M_{est}.

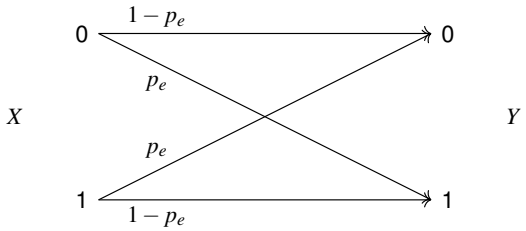


Figure 2.2: The Binary Symmetric Channel

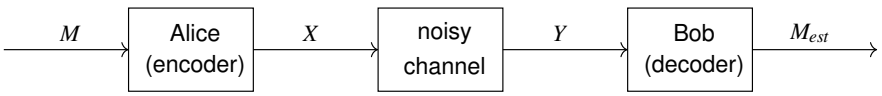


Figure 2.3: Code Transmission Across a Noisy Channel

Considering the probabilities with which the input symbols are chosen,

$$P(X=1) = 1 - P(X=0)$$

$$\implies \quad P(Y=0) = P(X=0)(1-p_e) + (1-P(X=0))p_e$$

$$\text{and} \quad P(Y=1) = P(X=0)p_e + (1-P(X=0))(1-p_e)$$

If input symbols are equally likely, $P(X=0) = P(X=1) = \frac{1}{2}$

$$\implies \quad P(Y=0) = \tfrac{1}{2}(1-p_e) + (1-\tfrac{1}{2})p_e = \tfrac{1}{2} - \tfrac{p_e}{2} + \tfrac{p_e}{2} \qquad = \tfrac{1}{2}$$

$$\text{and} \quad P(Y=1) = \tfrac{p_e}{2} + (1-\tfrac{1}{2})(1-p_e) = \tfrac{p_e}{2} + \tfrac{1}{2} - \tfrac{p_e}{2} \qquad = \tfrac{1}{2}$$

i.e. If the input symbols are equally likely, then so too are the output symbols.

If there is no repetition and the order doesn't matter, the number of r-combinations of a set of n symbols in a codeword is denoted by the combinatorial coefficient

$$C_r^n = \frac{n!}{(n-r)!r!} = \binom{n}{r} \tag{2.7}$$

Using this, if the BSC is assumed and a particular binary code word is transmitted:

- The probability that no error will occur is $(1-p_e)^n$
- The probability that one error will occur in a specified position is $p_e(1-p_e)^{n-1}$
- The probability of a particular received word that differs from the transmitted word in i specified positions is $p_e^i(1-p_e)^{n-i}$.
- The probability of exactly 1 error in any position is

$$n.p_e(1-p_e)^{n-1}$$

- The probability of exactly 2 errors in arbitrary positions is:

$$\frac{n(n-1)}{2}.(1-p_e)^{n-2}p_e^2$$

- The probability of exactly i errors in any positions is:

$$\binom{n}{i}(1-p_e)^{n-i}p_e^i$$

2.3 The Binary Erasure Channel (BEC)

The Binary Erasure Channel (BEC) in Figure 2.4, adapted from (Moser & Chen 2012, p. 135) where p_s is the probability of an erasure occurring, is an idealised channel in which data may be transmitted containing *erasures*. These are symbols that are known to be erroneous in known locations and that are just in need of correction, making the correction of erasures easier than the correction of errors.

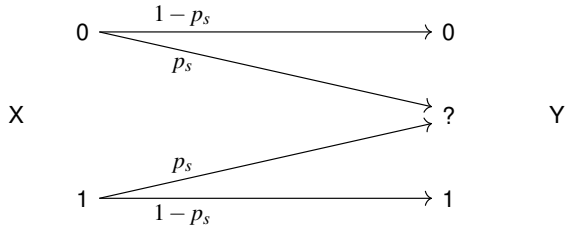


Figure 2.4: The Binary Erasure Channel

Again, considering the probabilities with which the input symbols are chosen,

$$P(X=1) = 1 - P(X=0)$$

$$\implies \quad P(Y=0) = P(X=0)(1-p_s)$$

and
$$\begin{aligned} P(Y=?) &= P(X=0)p_s + P(X=1)p_s \\ &= P(X=0)p_s + (1-P(X=0))p_s \\ &= p_s \end{aligned}$$

and $$P(Y=1) = P(X=1)(1-p_s)$$

If input symbols are equally likely, $P(X=0) = P(X=1) = \frac{1}{2}$

$$\implies \quad P(Y=0) = \tfrac{1}{2}(1-p_s)$$

and $$P(Y=?) = p_s$$

and $$P(Y=1) = \tfrac{1}{2}(1-p_s) = P(Y=0)$$

As for the BSC, the probability of exactly *i* erasures in any positions is:

$$\binom{n}{i}(1-p_s)^{n-i}p_s^i$$

2.4 The Binary Deletion Channel (BDC)

In some respects, the Binary Deletion Channel (BDC) in Figure 2.5 is an extension of the BEC. Erasures involve the value of bits being unknown but in known locations. Deletions still involve bits of ambiguous value however the location of the missing binary digits (bits) is now also unknown. The BEC must consider the possible values of an erased bit in a single position while the BDC must effectively consider the possible values of an erased bit in every location. This brings an extra layer of complexity to the problem and increases calculation times.

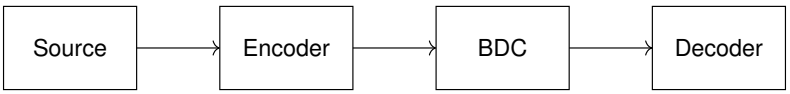


Figure 2.5: Code transmission across a Binary Deletion Channel

Referring to the BDC, Mitzenmacher (2009) notes that:

> Currently, we have no closed-form expression for the capacity, nor do we have an efficient algorithmic means to numerically compute this capacity.

If δ deletions of transmitted symbols $x_i \in X$ with deletion probability p_d are introduced then the positions of the received bits y_i may differ from the transmitted positions, such that $\{x_0x_1....x_{n-1}\} \mapsto \{y_0y_1....y_{n-\delta-1}\}$.

Ullman (1967) defines a *deletion error* at position j as an operator which takes the vector x into the vector y, where $y_i = x_i$ for $i < j$, $y_i = x_{i+1}$ for $n > i > j$ and y_n is either fixed at 0 or fixed at 1. He pictures a deletion as causing a shift left of all the bits to its right as in Figure 2.6.

The deletion of different bits may yield the same outcome. As an example, consider the

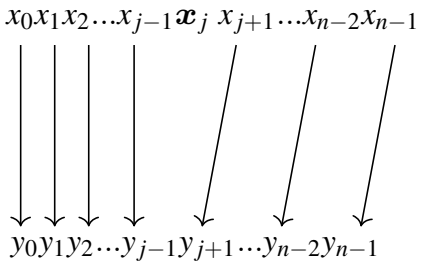


Figure 2.6: The Binary Deletion Channel

8-bit codeword $\begin{pmatrix}1_1 & 0_2 & 0_3 & 1_4 & 0_5 & 1_6 & 1_7 & 1_8\end{pmatrix}$ being transmitted across the BDC and a single deletion occurring. If the bit deleted was in position 6, 7 or 8, the outcome $\begin{pmatrix}1 & 0 & 0 & 1 & 0 & 1 & 1\end{pmatrix}$ will be the same for each position. Similarly, a deleted bit in position 2 or 3 both give the same outcome $\begin{pmatrix}1 & 0 & 1 & 0 & 1 & 1 & 1\end{pmatrix}$. With n possible deletion positions, the 7-bit word received could be any of those shown in Table 2.1. The BDC itself is not directly studied here but Chapter 7 looks at the impact on equivocation of intentional deletions on the BSC.

Received word	Probability
$(1\ 0\ 0\ 1\ 0\ 1\ 1)$	$3/8 = 0.375$
$(1\ 0\ 0\ 1\ 1\ 1\ 1)$	$1/8 = 0.125$
$(1\ 0\ 0\ 0\ 1\ 1\ 1)$	$1/8 = 0.1255$
$(1\ 0\ 1\ 0\ 1\ 1\ 1)$	$2/8 = 0.25$
$(0\ 0\ 1\ 0\ 1\ 1\ 1)$	$1/8 = 0.125$

Table 2.1: Received word options for a single deletion of transmitted codeword 10010111

2.5 The Binary Symmetric Erasure Channel (BSEC)

A further channel discussed by Michelson & Levesque (1985) but not examined in detail here is the Binary Symmetric Erasure Channel (BSEC). This is a binary input, ternary output channel that includes a symmetric cross-over probability of p_e and a symmetric erasure of probability p_s from either input symbol to an output symbol whose state is ambiguous. If the outputs are judged to be unreliable due to factors such as a weak received signal, those outputs are erased as they leave the demodulator. This channel model is depicted in Figure 2.7.

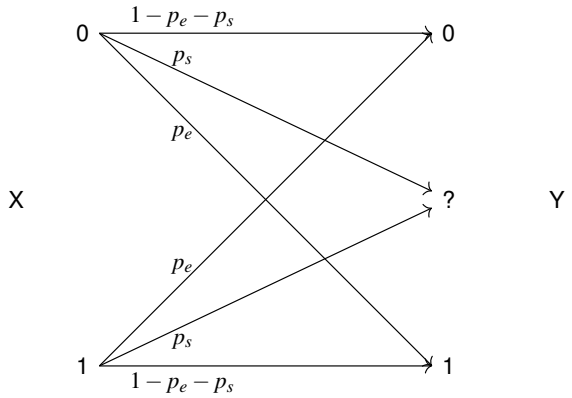


Figure 2.7: The Binary Symmetric Erasure Channel

Once more considering the probabilities with which the input symbols are chosen,

$$P(X=1) = 1 - P(X=0)$$

$$\implies \quad P(Y=0) = P(X=0)(1-p_e-p_s) + P(X=1)p_e$$
$$= P(X=0)(1-p_e-p_s) + (1-P(X=0))p_e$$

and $$P(Y=?) = P(X=0)p_s + P(X=1)p_s$$
$$= P(X=0)p_s + (1-P(X=0))p_s$$
$$= p_s$$

and $$P(Y=1) = P(X=0)p_e + P(X=1)(1-p_e-p_s)$$
$$= P(X=0)p_e + (1-P(X=0))(1-p_e-p_s)$$

If input symbols are equally likely, $P(X=0) = P(X=1) = \frac{1}{2}$

$$\implies \quad P(Y=0) = \tfrac{1}{2}(1-p_e-p_s) + \tfrac{1}{2}p_e$$
$$= \tfrac{1}{2}(1-p_s)$$

and $$P(Y=?) = p_s$$

and $$P(Y=1) = \tfrac{1}{2}p_e + \tfrac{1}{2}(1-p_e-p_s)$$
$$= \tfrac{1}{2}(1-p_s) = P(Y=0)$$

2.6 The Wire-Tap Channel (WTC)

Wyner (1975) introduced the concept of the “Wire-Tap Channel" shown in Figure 2.8.

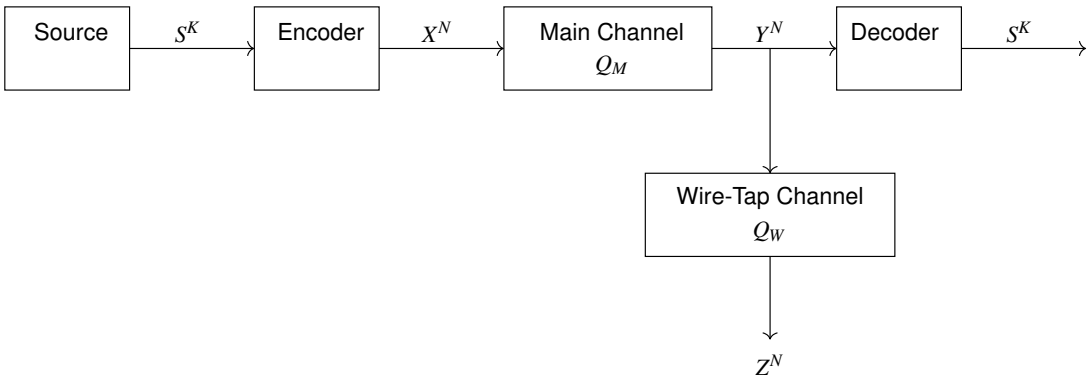


Figure 2.8: Code Transmission Across a Noisy Channel

His results showed that one could obtain perfect secrecy as described in Section 1.3 when a receiver enjoys a better channel than the wire-tapping opponent does.

The source is identified by the sequence $\{S_k\}_1^\infty$ where the S_k are independent, identically distributed random variables that take values in the finite set **S**. The *Main Channel* is a discrete memoryless channel with finite input alphabet X, finite output alphabet Y and probability of a cross-over error, where 0 becomes 1 and vice versa, of $Q_M(y|x),\ x \in X,\ y \in Y$. Since the channel is memoryless, the probability of a cross-over error for N vectors is:

$$Q_M^{(N)}(y|x) = \Pi_{n=1}^N Q_M(y_n|x_n) \tag{2.8}$$

The *Wire-Tap Channel* is also a discrete memoryless channel with finite input alphabet Y, finite output alphabet δ and probability of cross-over error $Q_W(z|y),\ y \in Y,\ z \in Z$. The cascade of the main channel and the wire-tap channel is another memoryless channel with probability of cross-over error:

$$Q_{MW}^{(N)}(z|x) = \Pi_{y \in Y} Q_W(z|y) Q_M(y|x) \tag{2.9}$$

So if the main channel has an error probability of 0.01 and the wire-tap channel had

an error probability of 0.1, the legitimate receiver will receive correct symbols with an average probability of 0.99, while the eavesdropper will only receive correct symbols with an average probability of $0.99 \times 0.9 = 0.891$.

There is a trade-off between the transmission rate R and the minimum number of bits that differ between any two codewords of the code (the Hamming distance d), assuming essentially perfect ('error-free') transmission. Wyner's results implied that there exists a channel secrecy capacity (discussed in Section 4.7), $C_S > 0$, such that reliable transmission at rates up to C_S is possible in levels approaching perfect secrecy. However, we are less concerned here with the assumption of error-free transmission and focus more on the comparison of equivocation values between the legitimate recipient and the eavesdropper. Hence more emphasis is placed in the number of transmitted bits and the length of code for which equivocation can be calculated using the methods discussed than the rate at which the data is transmitted.

The concept of the Wire-Tap Channel will be used throughout this thesis to highlight the difference in the equivocation levels for a legitimate recipient and for an eavesdropper.

2.7 Conclusion

This chapter has introduced a few of the many theoretical models of channels for data transmission. The simple BSC works on a cross-over error probability where the introduction of an error results in a transmitted 0 being received as 1 and vice versa. The BEC, with a given erasure probability, results in a bit of unknown value but in a known position, while the BDC yields a bit of unknown value in an unknown location. The BDC, about which relatively little is known, causes a substantial increase in the complexity of finding a solution. As its name suggests, the BSEC is a hybrid of the BSC and the BEC, however BSEC will not be examined in any further detail here. The WTC extends each scenario to include differing channel conditions for each of a legitimate recipient and an illegitimate eavesdropper.

Chapter 3

Codes

3.1 Introduction

This chapter develops some of the concepts and tools that will either be used in subsequent chapters, in the calculation of code equivocation, or that contribute to the general code picture. These include:

- Terminology
- Code distance and weight
- Classes and types of code including linear, error correcting, simple, dual, turbo, interleaved, punctured, cyclic, best known linear, perfect, Hamming and Golay codes
- Generator and parity check matrices
- Codes and their relationship to sphere packing problems

3.2 Coding Terminology

The terminology below will be used when referring to the characteristics of a code:

- q Number of distinct symbols employed on the channel
  (e.g. for binary codes, $q = 2$)
- n Number of symbols in the codeword

- k Number of symbols in the message.
  Also known as the *information dimension* of the code C
- m Number of parity check symbols added to the message to give the codeword ($k+m=n$)
- M Number of possible messages in a q-ary code of message length k
- $\lfloor$ x $\rfloor$ The floor of x. The largest integer less than or equal to x.
- $\lceil$ x $\rceil$ The ceiling of x. The smallest integer greater than or equal to x.
- Redundancy r is the proportion of a code that is sent in addition to the actual message. $r = \frac{n-k}{n} = \frac{m}{n}$
- Code Rate $R = k/n$ is the proportion of information in the transmitted codeword. The fraction should be given in its simplest form. More generally, $R = \frac{1}{n}\log_2 M = \frac{k}{n}\log_2 q$ bits per symbol (where $M = q^k$ for a linear code)
- The Bit Error Rate (BER) is a measure of how badly a signal is affected by errors. It is given by how many errors exist for a given number of bits transmitted.

Encoding is the process where the k message bits are converted to n bits of the codeword (Hill 1986). Decoding is the reverse process, where the k bits of the message are retrieved from the n bits of the codeword. A code is often described by parameters giving the number of symbols in the codeword and the number of symbols in the message in the form $[n,k]$. For example, a $[6,4]$ code is a code in which the process of encoding adds 2 parity check bits to a 4-bit message, giving a 6-bit codeword.

Two q-ary codes are *equivalent* if one can be obtained from the other by means of:

- A permutation of the positions of the code or
- A permutation of the symbols appearing in one particular position.

3.3 Linear Codes

All the codes studied will be drawn from the $\mathbb{F}_2$ Galois Field i.e. they will be binary codes represented by the symbols 0 and 1.

Linear codes are codes for which any linear combination of codewords is also a codeword (Ryan & Lin 2009). Linear codes can be defined with symbols chosen from a set of arbitrary size, but most significant results have been derived from assuming that the code symbols are elements of a finite field.

In general, if we let $\mathbb{F}_q^n$ denote an n-dimensional vector space over a finite field of q symbols $\mathbb{F}_\mathrm{q}$, for example:

$$\underbrace{[0,1,1,0,\cdots,0]}_{(n-elements)}$$

then an $[n,k,d]_q$ linear code C is a k-dimensional subset of $\mathbb{F}_q^n$. The linear code C has q^k codewords. d is the minimum Hamming distance of the code. Each vector in the k-dimensional subset of $\mathbb{F}_q^n$, which has a length of n symbols, is called a codeword.

For a code to be linear the following rule applies: If c_1 and c_2 are codewords and α_1 and α_2 are field elements, then $c_3 = \alpha_1 c_1 + \alpha_2 c_2$ is also a codeword.
e.g. If a code is binary and linear and c_1 and c_2 are codewords, where $c_1 = 0101$, $c_2 = 0011$ then $c_3 = c_1 + c_2 = 0101 + 0011 = 0110$ is also a codeword.

Linear codes have several advantages over arbitrary codes: (Baylis 1998)

1. Evaluation of the distance of a code is easier
2. Encoding is fast and requires little storage
3. It is much easier to determine which errors are detectable / correctable
4. The probability of correct decoding is much easier to calculate
5. Very slick decoding techniques exist for linear codes

3.4 Hamming Distance

The *Hamming distance* of two codewords is the number of positions in which two valid codewords differ. The Hamming distance of a code is the minimum Hamming distance between any two codewords. In Table 3.1, the minimum distance between any two codewords (of the Hamming $[7,4]$ code) is 3. That is, the Hamming distance of the code is 3, so the code can be written as the Hamming $[7,4,3]$ code.

Message	Codeword
0000	0000000
0001	0001111
0010	0010110
0011	0011001
0100	0100101
0101	0101010
0110	0110011
0111	0111001
1000	1000011
1001	1001100
1010	1010101
1011	1011010
1100	1100110
1101	1101001
1110	1110000
1111	1111111

Table 3.1: Hamming $[7,4,3]$ Code - Messages and their Codewords

If a code is used only for error detection, for the code to be able to detect all patterns of t or fewer errors, it is necessary and sufficient to have $t \leq d-1$ (or $t < d$), where d is the minimum Hamming distance between codewords (Sweeney 2002, p.26).

It is possible to correct all patterns of e or fewer errors if and only if $2e+1 \leq d$.

In Figure 3.1, a filled circle represents a valid codeword and an empty circles represents an invalid codeword that contains at least one error. A code that has a Hamming distance of 3 is able to detect 1 or 2 errors, but can only correct a single error by choosing the closest valid codeword to the sequence received.

In Figure 3.2, a code that has a Hamming distance of 5 is able to detect up to 4 errors

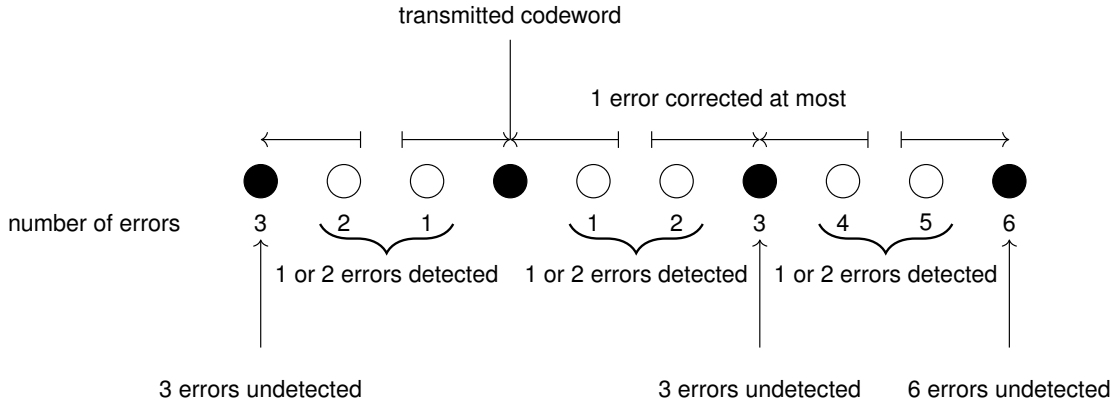


Figure 3.1: Codes with Hamming distance=3

and correct up to 2.

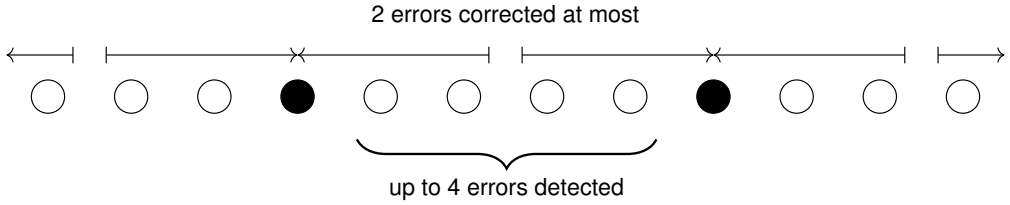


Figure 3.2: Codes with Hamming distance=5

In Figure 3.3, a code that has a Hamming distance of 6 is able to detect up to 5 errors and correct up to 2. A received sequence that is an equal distance of 3 from two valid codewords could not be reliably corrected.

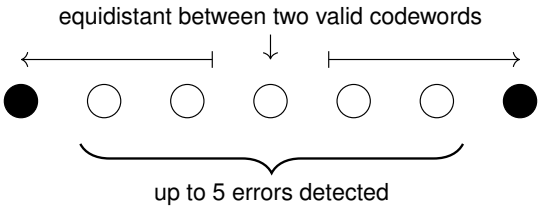


Figure 3.3: Codes with Hamming distance=6

The *relative distance* of a code C is a measure of the minimum distance of the code as a fraction of the code length:

$$\delta(C) = d/n \tag{3.1}$$

Code C' can be said to be a better code than code C if $\delta(C') \geq \delta(C)$.

3.5 Weight

Let $c \in \mathbb{F}^n$. Then the *weight* of c, denoted $w(c)$, is the number of non-zero coordinates in c. The *weight* of a *code* is the minimum weight of any of its codewords. By inspection of Table 3.1, it can be seen that the minimum weight of any non-zero codeword of the Hamming $[7,4]$ code is 3 (e.g. 0010101 has a weight of 3). If C is a linear code then $d(C) = w(C)$. A *constant weight* code is one in which all codewords have equal weight.

The *weight distribution* of a code is the set of numbers $\{A_i(c)\}$ where $A_i(c)$ denote the number of codewords at Hamming distance *i* from a codeword $c \in C$. $A_0(c) = 1$, $A_i(c) \geq 0$ and $\Sigma_i A_i(c) = M$. For Ham$[7,4,3]$, the weight distribution is:

i	0	3	4	7
A_i	1	7	7	1

and $\Sigma_i A_i = 16$.

3.6 Error Correcting Codes

Codes that build in the ability to correct errors are known as forward error correction codes.

When a two way channel is used, an error-detecting code can be used to initiate backward error correction. When an error is detected at one terminal, a request for a repeat transmission can be given, and thus errors can effectively be corrected. Error detection is by its nature a much simpler task than error correction and requires much simpler decoding equipment. Error detection with retransmission is adaptive, in that the transmission of redundant information is increased when errors occur, limiting the efficiency of a simple error detection system. This makes it possible to get a better performance than is theoretically possible on a one-way channel. Whilst there are true examples of one-way channels in which error probabilities can be reduced using error correcting codes, this is not true for error detection and retransmission. Furthermore, systems

can use a combination of error correction and detection with feedback.

3.7 The Error Correcting Code Problem

Three key aspects of general coding problems are:

1. Finding codes that have the required error correcting ability (this usually makes them long)
2. Finding a practical method of encoding the messages
3. Finding a practical method of making the decision at the receiver i.e. a method of error correction.

The *Error-Correcting Code Problem* can be stated (Conway & Sloane 1999) as:

> Given a q-ary alphabet, a length n and a minimum distance d, find a code with these parameters and the maximal possible number of codewords, $A_q(n,d)$

e.g. If $A_2(5,3) = 4$ then the maximal possible number of codewords in a $(5,3)$ binary code is 4.
Many upper and lower bounds have been found for $A(n,d)$; a summary of these, collated by Conway and Sloane, is shown in Table 3.2.

For example, for a code of length $n = 10$ with minimum distance 4, the maximal possible number of codewords is 40. It is worth noting that many code length and minimum difference combinations exist for which the maximal number of codewords is only imprecisely bounded, even for quite short length codes.

3.8 Dual Code (or orthogonal vector space)

- Given a linear code $C \in \mathbb{F}^n$, then the dual code of C, $C^{\perp} = \{v \in \mathbb{F}^n | v.c = 0$ for every $c \in$ C$\}$
- If C is a linear code in $\mathbb{F}^n$ with generator matrix **G**, then $\underline{v} \in$ C$^{\perp}$ if and only if v**G** $= 0$.

n	$d=4$	$d=6$	$d=8$	$d=10$
6	4	2	1	1
7	8	2	1	1
8	16	2	2	1
9	20	4	2	1
10	40	6	2	2
11	72	12	2	2
12	144	24	4	2
13	256	32	4	2
14	512	64	8	2
15	1024	128	16	4
16	2048	256	32	4
17	2720-3276	256-340	36-37	6
18	5312-6552	512-680	64-72	10
19	10496-13104	1024-1288	128-144	20
20	20480-26208	2048-2372	256-279	40
21	36864-43690	2560-4096	512	42-48
22	73728-87380	4096-6942	1024	50-88
23	147456-173784	8192-13774	2048	76-150
24	294912-344636	16384-24106	4096	128-280

Table 3.2: Maximum Possible Number of Codewords for Codes of Length n (Conway & Sloane 1999)

- dim(C) + dim($C^{\perp}$) = n. Thus if C is an $[n,k]$ code then $C^{\perp}$ is an $[n,n-k]$ code.
- $(\mathrm{C}^{\perp})^{\perp} = \mathrm{C}$.
- A code is self-dual if $C = C^{\perp}$

3.9 Simple codes

- The *Zero* code $[n,0,n]$ of length n contains just the codeword 00....0.
- The *Universe* code $[n,n,1]$ $\mathbb{F}_{\mathrm{q}}^{\mathrm{n}}$ is the dual of the Zero code and will contain q^n codewords, with a minimum distance for the code of 1.
- *Triplication* codes - Every message is repeated 3 times and the receiver takes a majority vote. The code only corrects single errors and is very inefficient.
- The *Repetition* code $[n,1,n]$ contains all codewords, $aa....a$, $a \in \mathbb{F}_q$.
- *Rectangular* codes - The information is arranged in an $(m-1)\times(n-1)$ rectangle.

A parity bit is added to each row and to each column, making an $m \times n$ rectangle. Rectangular codes are also known as product codes. An example of a rectangular code is shown in Table 3.3.

1	0	0	1	1	0	0	1
0	1	1	0	0	1	0	1
1	1	0	1	0	0	1	0
0	0	1	1	0	1	1	0
1	0	0	0	0	1	0	0
1	0	0	1	1	1	0	0

Table 3.3: A Rectangular Code with Word and Block Parity Check Bits

- The $[n, n-1, 2]$ zero-sum code contains all vectors such that $\Sigma c_i = 0$. It is the dual of the $[n, 1, n]$ repetition code. When $q = 2$ this is called the even weight code, since it consists of all binary vectors containing an even number of 1's.

  For example, the $[5, 4, 2]$ zero-sum code contains the codewords shown in Table 3.4.

00000	00011	00101	00110
01001	01010	01100	10001
10010	10100	11000	01111
10111	11011	11101	11110

Table 3.4: Codewords for a $[5, 4, 2]$ Zero-sum Code

3.10 Turbo Codes

Turbo codes are high-performing error correction codes that approach Shannon's Theorem for the Channel Capacity (Shannon 1949, p. 47) in Equation 3.2. This gives the maximum rate at which information can be transmitted over a communication channel given a specific noise level.

$$C_a = B \log_2(1 + \frac{S}{N}) \tag{3.2}$$

where C_a is the capacity of the channel in bits per second, B is the bandwidth of the channel in Hertz, S is the average received signal power over the bandwidth (measured

in Watts) and N is the average noise over the bandwidth (measured in Watts). Channel capacity will be discussed further in Chapter 4.

Turbo codes are so named because, in a similar manner to mechanical turbos feeding back power to the engine system, turbo codes work on an iterative process, feeding back the decoded output as a joint input with the original data. This enables further decoding and thereby reduces the number of errors with each iteration.

Turbo codes use extrinsic information in a recursive or iterative manner. This information is shared between different component decoders. The decoder of the overall code is computationally complex but the individual or component decoders are not. The component decoding algorithms are exact, but the overall decoding method is approximate.

3.11 Interleaving

Interleaving enables a burst of errors to be spread throughout the message by rearranging the order in which the code digits are transmitted, which then allows single errors to be corrected (two consecutive errors if they are in adjacent codewords). Spreading out a burst of errors increases the likelihood of being able to correct the errors. The process of interleaving/de-interleaving the information increases transmission delays (latency), processing and storage.

3.12 Puncturing Codes

Let C be an $[n,k,d]$ code over $\mathbb{F}_q$. We can puncture C by deleting the same coordinate i in each codeword (Jones & Jones 2002, p. 104). The resulting code is still linear; its length is $n-1$ and is often denoted by C^*. If **G** is a generator matrix for C, then a generator matrix C^* is obtained from **G** by deleting column i (and omitting a zero or duplicate row that might occur).

If $d > 1$, C^* is an $[n-1,k,d^*]$ code where $d^* = d-1$ if C^* has a minimum weight codeword with a non-zero i'th coordinate and $d^* = d$ otherwise. When d = 1, C^* is an $[n-1,k,1]$ code if C has no codeword of weight 1 whose non-zero entry is in coordinate

i; otherwise, if $k > 1$, C^* is an $[n-1, k-1, d^*]$ code with $d^* \geq 1$.

3.13 Cyclic Codes

A code is cyclic if whenever $c_0 c_1 \cdots c_{n-1}$ is a codeword, so is $c_{n-1} c_0 c_1 \cdots c_{n-2}$ (Morelos-Zaragoza 2002) i.e. A linear block code is cyclic if and only if every cyclic shift of a codeword is another codeword.

To understand cyclic codes, it is best to represent the codewords as polynomials. Thus a codeword $C_1 = \{c_0, c_1, \cdots, c_{n-1}\}$ can be represented by the polynomial $c_0\alpha^0, c_1\alpha^1 \cdots c_{n-1}\alpha^{n-1}$.

Cyclic shifts then are a multiplication of a codeword with the variable α and the result of the multiplication modulo $\alpha^n - 1$. All the codewords can be obtained from one generator polynomial $g(\alpha)$ as long as the information is also represented as a polynomial. Thus encoding is done by polynomial multiplication or convolution using shift registers. The degree of the information polynomial is $k-1$ and the degree of the codeword is $n-1$ which means that the degree of the generator must be $n-k$. Similarly the parity check polynomial $h(\alpha)$ has the property that:

$$g(\alpha)h(\alpha) = \alpha^n - 1 \tag{3.3}$$

Consider length 7 binary cyclic codes. With binary codes, we have the following factorisation into primitive (irreducible) polynomials:

$$\alpha^7 + 1 = (\alpha + 1)(\alpha^3 + \alpha + 1)(\alpha^3 + \alpha^2 + 1) \tag{3.4}$$

Therefore possible generator polynomials are shown in Table 3.5.

Taking the primitive polynomial $\alpha^3 + \alpha + 1$ and setting to zero, we get $\alpha^3 = \alpha + 1$ which can be used to generate the other polynomials.

$\alpha^0 = 1$

$\alpha^1 = \alpha$

Factors	$g(\alpha)$	(n,k)	Comment
1	1	(7,7)	Universe code - all code-words
$\alpha+1$	$\alpha+1$	(7,6)	Single parity check code
$\alpha^3+\alpha+1$	$\alpha^3+\alpha+1$	(7,4)	Hamming code (a)
$\alpha^3+\alpha^2+1$	$\alpha^3+\alpha^2+1$	(7,4)	Hamming code (b)
$(\alpha+1)(\alpha^3+\alpha+1)$	$\alpha^4+\alpha^3+\alpha^2+1$	(7,3)	Dual of Hamming code (a)
$(\alpha+1)(\alpha^3+\alpha^2+1)$	$\alpha^4+\alpha^3+\alpha+1$	(7,3)	Dual of Hamming code (b)
$(\alpha^3+\alpha^2+1)(\alpha^3+\alpha+1)$	$\alpha^6+\alpha^5+\alpha^4+\alpha^3+\alpha^2+\alpha+1$	(7,1)	Repetition code. Dual of the single parity check code
$(\alpha+1)(\alpha^3+\alpha^2+1)(\alpha^3+\alpha+1)$	(α^7+1)	(7,0)	Zero code

Table 3.5: Generators

$\alpha^2 = \alpha \times \alpha = \alpha^2$

$\alpha^3 = \alpha + 1$

$\alpha^4 = \alpha^3 \times \alpha = (\alpha + 1)\alpha = \alpha^2 + \alpha$

$\alpha^5 = \alpha^4 \times \alpha = (\alpha^2 + 1)\alpha = \alpha^3 + \alpha^2 = \alpha^2 + \alpha + 1$

$\alpha^6 = \alpha^5 \times \alpha = (\alpha^2 + \alpha + 1) \times \alpha = \alpha^3 + \alpha^2 + \alpha = \alpha + 1 + \alpha^2 + \alpha = \alpha^2 + 1$

$\alpha^7 = \alpha^6 \times \alpha = (\alpha^2 + 1) \times \alpha = \alpha^3 + \alpha = \alpha + 1 + \alpha = 1$

These polynomials and their related vectors are shown in Table 3.6.

Element	Polynomial	Vector
0	0	{0, 0, 0}
α^0	1	{0, 0, 1}
α^1	α	{0, 1, 0}
α^2	α^2	{1, 0, 0}
α^3	$\alpha + 1$	{0, 1, 1}
α^4	$\alpha^2 + \alpha$	{1, 1, 0}
α^5	$\alpha^2 + \alpha + 1$	{1, 1, 1}
α^6	$\alpha^2 + 1$	{1, 0, 1}
α^7	1	{0, 0, 1}

Table 3.6: Polynomials in GF(2^3)

3.14 Generator Matrices

A generator matrix is a matrix whose rows form the basis for a linear code. The codewords are all of the linear combinations of the rows of the matrix i.e. the linear code is the row space of its generator matrix.

For a block code with $q = 2$ and $n = 5$, the set of vectors (0 0 0 0 0), (1 0 0 1 1), (0 1 0 1 0), (1 1 0 0 1), (0 0 1 0 1), (1 0 1 1 0), (0 1 1 1 1) and (1 1 1 0 0) form a vector space V_1 and hence a linear or group, binary code. The minimum weight is 2 and hence the minimum distance is 2.

Any set of basis vectors for a linear block code V can be considered as rows of a matrix **G**, called a Generator Matrix of V. The code V_1 is the row space of either of the following matrices:

$$\mathbf{G}_1 = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

or

$$\mathbf{G}_2 = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

If **G** is a matrix, it generates the codewords of a linear code C by $w = s\mathbf{G}$, where w is a codeword of the linear code C and s is any vector. A generator matrix for an $[n,k,d]$ q-code is a $k \times n$ matrix, where n is the length of a codeword, k is the number of information bits (the dimension of C as a vector subspace), d is the minimum distance of the code and q is the size of the finite field, that is, the number of symbols in the alphabet.

The standard form for a generator matrix is:

$$\mathbf{G} = [\mathbf{I}_k | \mathbf{A}]$$

where I_k is the $k \times k$ identity matrix and A is a $k \times m$ matrix. m is the number of redundant bits in each codeword, $m = n - k$. For any set of k independent columns of a generator matrix **G** the corresponding set of coordinates form an information set for C. The remaining $m = n - k$ coordinates form a redundancy set.

Whilst there can be many generator matrices for a given code, if the first k coordinates

form an information set, the code has a unique generator matrix of the form $\mathbf{G} = [\mathbf{I}_k|\mathbf{A}]$.

3.15 Parity Check Matrices

Because a linear code is a subspace of a vector space, it is the kernel of some linear transformation. In particular there is an $(n-k) \times n$ matrix **H**, called a parity check matrix for the $[n,k]$ code *C*, defined by :

$$C = \{x \in \mathbb{F}_q^n | \mathbf{H}x^T = 0\} \tag{3.5}$$

The matrix **H** is called a parity-check matrix for a linear code C if the columns of **H** form a basis for the dual code $C^\perp$. As with the generator matrix for an $[n,k]$ code C, the rows of the $(n-k) \times n$ parity check matrix **H** are independent and **H** is the generator matrix of the *dual* code or *orthogonal* of C.

The rows of **H** will also be independent. In general, there are also several possible parity check matrices for C. If $\mathbf{G} = [\mathbf{I}_k|\mathbf{A}]$ is a generator matrix for the $[n,k]$ code C in standard form, then $\mathbf{H} = [-\mathbf{A}^T|\mathbf{I}_{n-k}]$ is a parity check matrix for C.

Taking the $[7,4,3]$ Hamming Code with the generator matrix below as an example,

$$\mathbf{G} = [\mathbf{I}_4|\mathbf{A}] = \left(\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}\right) \tag{3.6}$$

a parity check matrix would be:

$$\mathbf{H} = [\mathbf{A}^T|\mathbf{I}_3] = \left(\begin{array}{cccc|ccc} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right) \tag{3.7}$$

A linear code can be presented with either a generator matrix or a parity check matrix.

A simple parity check for a binary message is achieved by counting the number of 1's in the message and appending a final binary digit, so that the entire message is designed to have either an odd or even number of 1's in it. This only enables an odd number of errors to be detected. The rows of a parity check matrix are parity checks on the codewords of a code. They show how linear combinations of certain digits of each codeword equal zero. For example, the parity check matrix:

$$\begin{bmatrix} 0_1 & 0_2 & 1_3 & 1_4 \\ 1_1 & 1_2 & 0_3 & 0_4 \end{bmatrix}$$

shows that there are two parity checks. The first row specifies that for each codeword, digits 3 and 4 should sum to zero, whilst the second row specifies that for each codeword, digits 1 and 2 should sum to zero. The parity check matrix for a given code can be derived from its generator matrix and vice versa.

3.16 Best Known Linear Codes

Many of the codes used here as examples in the calculation of equivocation are Best Known Linear Codes (BKLCs), i.e. they have the highest minimum weight among all $[n,k]$ codes. Suitable codes were identified from code tables of BKLCs compiled by Grassl (2015), while the generator and parity check matrices for these codes were obtained using the online calculating software *Magma* provided by the University of Sydney (2015). For example, to obtain the generator and parity check matrices for the BKLC $[15,11,3]$ (i.e. Hamming) code, the instructions were submitted to the program as shown in Figure 3.4 and Figure 3.5.

```
C:=BKLC(GF(2),15,11);
C;
```

Figure 3.4: Magma code for obtaining BKLC generator matrix

These inputs yielded the Magma outputs shown in Figure 3.6 and Figure 3.7 respectively. The Magma output is displayed line by line as a sequence of row vectors that,

```
C:=BKLC(GF(2),15,11);
P:=ParityCheckMatrix(C);
P;
```

Figure 3.5: Magma code for obtaining BKLC parity check matrix

when combined, form the overall matrix.

```
[15,11,3] Linear Code over GF(2)
Generator matrix
[1 0 0 0 0 0 0 0 0 0 0 0 0 1 1]
[0 1 0 0 0 0 0 0 0 0 0 0 1 0 1]
[0 0 1 0 0 0 0 0 0 0 0 0 1 1 0]
[0 0 0 1 0 0 0 0 0 0 1 0 0 0 1]
[0 0 0 0 1 0 0 0 0 0 1 0 0 1 0]
[0 0 0 0 0 1 0 0 0 0 1 0 1 0 0]
[0 0 0 0 0 0 1 0 0 0 1 0 1 1 0]
[0 0 0 0 0 0 0 1 0 0 1 0 1 0 1]
[0 0 0 0 0 0 0 0 1 0 1 0 0 1 1]
[0 0 0 0 0 0 0 0 0 1 1 0 0 1 1]
[0 0 0 0 0 0 0 0 0 0 0 1 1 1 1]
```

Figure 3.6: Magma generator matrix output

```
[1 0 1 0 1 0 1 0 1 0 1 0 1 0 1]
[0 1 1 0 0 1 1 0 0 1 1 0 0 1 1]
[0 0 0 1 1 1 1 0 0 0 0 1 1 1 1]
[0 0 0 0 0 0 0 1 1 1 1 1 1 1 1]
```

Figure 3.7: Magma parity check matrix output

3.17 Perfect Codes

A code $C \subset \mathbb{F}_q^n$ with minimum distance $2e+1$ is defined (van Lint 1999, p.34) as *Perfect* if every $x \in \mathbb{F}_q^n$ has distance $\leq e$ to exactly one codeword. A minimum distance of $2e+1$ enables the code to correct e errors. Hamming codes and the Golay code are the only non-trivial examples of perfect codes.

A code C of length n and odd distance $d = 2t+1$ (where t is the number of errors to be corrected) is perfect if C attains the Hamming bound (van Lint 1999):

$$|C| \leq \frac{2^n}{\binom{n}{0} + \binom{n}{1} + \ldots + \binom{n}{t}} \tag{3.8}$$

3.18 Hamming Codes

For any integer m, it is possible to construct a perfect, single error-correcting, binary Hamming code Ham$(m,2)$ with Hamming distance 3 (Hamming 1950). Such a code uses m parity digits to correct any single error in a codeword of size n digits, where $n = 2^m - 1$. The message length is k, where $k = n - m$. The Hamming code Ham$(m,2)$ can be described by the parameters Ham$[n,k,d]$. The generator matrix will have dimensions $k \times (2^m - 1)$ and the parity check matrix has dimensions $m \times (2^m - 1)$.

Consider the binary Hamming code with message length $m = 4$. Since $n = (2^m - 1)$, $n = (2^4 - 1) = 15$ and $k = 15 - 11 = 4$. Ham$(4,2)$ is a **GF**2-code with parameters $[15,11,3]$.

Hamming codes are cyclic codes, as can be seen by considering the Hamming $[15,11,3]$ code. Looking at the **GF**(2^4) field, $\alpha^{2^4-1} - 1 = \alpha^{15} - 1$ This can be factorised as:

$$\alpha^{15} - 1 = (\alpha - 1)(\alpha^4 + \alpha + 1)(\alpha^4 + \alpha^3 + 1)(\alpha^2 + \alpha + 1)(\alpha^4 + \alpha^3 + \alpha^2 + \alpha + 1)$$

Therefore since $(\alpha^4 + \alpha + 1)$ is a primitive polynomial factor and $\alpha^4 + \alpha + 1 = 0$, we can use the polynomial $\alpha^4 = \alpha + 1$ to generate the other polynomials.

$\alpha^0 = 1$

$\alpha^1 = \alpha$

$\alpha^2 = \alpha \times \alpha = \alpha^2$

$\alpha^3 = \alpha \times \alpha \times \alpha = \alpha^3$

$\alpha^4 = \alpha + 1$

$\alpha^5 = \alpha^4 \times \alpha = (\alpha + 1)\alpha = \alpha^2 + \alpha$

$\alpha^6 = \alpha^5 \times \alpha = (\alpha^2 + \alpha) \times \alpha = \alpha^3 + \alpha^2$

$\alpha^7 = \alpha^6 \times \alpha = (\alpha^3 + \alpha^2) \times \alpha = \alpha^4 + \alpha^3 = \alpha^3 + \alpha + 1$

$\alpha^8 = \alpha^7 \times \alpha = (\alpha^3 + \alpha + 1) \times \alpha = \alpha^4 + \alpha^2 + \alpha = \alpha^2 + \alpha + \alpha + 1 = \alpha^2 + 1$

$\alpha^9 = \alpha^8 \times \alpha = (\alpha^2 + 1) \times \alpha = \alpha^3 + \alpha$

$\alpha^{10} = \alpha^9 \times \alpha = (\alpha^3 + \alpha) \times \alpha = \alpha^4 + \alpha^2 = \alpha^2 + \alpha + 1$

$\alpha^{11} = \alpha^{10} \times \alpha = (\alpha^2 + \alpha + 1) \times \alpha = \alpha^3 + \alpha^2 + \alpha$

$\alpha^{12} = \alpha^{11} \times \alpha = (\alpha^3 + \alpha^2 + \alpha) \times \alpha = \alpha^4 + \alpha^3 + \alpha^2 = \alpha^3 + \alpha^2 + \alpha + 1$

$\alpha^{13} = \alpha^{12} \times \alpha = (\alpha^3 + \alpha^2 + \alpha + 1) \times \alpha = \alpha^4 + \alpha^3 + \alpha^2 + \alpha = \alpha^3 + \alpha^2 + \alpha + \alpha + 1 = \alpha^3 + \alpha^2 + 1$

$\alpha^{14} = \alpha^{13} \times \alpha = (\alpha^3 + \alpha^2 + 1) \times \alpha = \alpha^4 + \alpha^3 + \alpha = \alpha^3 + \alpha + \alpha + 1 = \alpha^3 + 1$

$\alpha^{15} = \alpha^{14} \times \alpha = (\alpha^3 + 1) \times \alpha = \alpha^4 + \alpha = \alpha + 1 + \alpha = 1$

These polynomials and their contribution to the matrix are shown in Table 3.7.

Element	Polynomial	Vector
0	0	{0, 0, 0, 0}
α^0	$\alpha^0 = 1$	{0, 0, 0, 1}
α^1	$\alpha^1 = \alpha$	{0, 0, 1, 0}
α^2	α^2	{0, 1, 0, 0}
α^3	α^3	{1, 0, 0, 0}
α^4	$\alpha^4 = \alpha + 1$	{0, 0, 1, 1}
α^5	$\alpha^5 = (\alpha + 1)\alpha = \alpha^2 + \alpha$	{0, 1, 1, 0}
α^6	$\alpha^6 = \alpha^3 + \alpha^2$	{1, 1, 0, 0}
α^7	$\alpha^7 = \alpha^3 + \alpha + 1$	{1, 0, 1, 1}
α^8	$\alpha^8 = \alpha^2 + 1$	{0, 1, 0, 1}
α^9	$\alpha^9 = \alpha^3 + \alpha$	{1, 0, 1, 0}
α^{10}	$\alpha^{10} = \alpha^2 + \alpha + 1$	{0, 1, 1, 1}
α^{11}	$\alpha^{11} = \alpha^3 + \alpha^2 + \alpha$	{1, 1, 1, 0}
α^{12}	$\alpha^{12} = \alpha^3 + \alpha^2 + \alpha + 1$	{1, 1, 1, 1}
α^{13}	$\alpha^{13} = \alpha^3 + \alpha^2 + 1$	{1, 1, 0, 1}
α^{14}	$\alpha^{14} = \alpha^3 + 1$	{1, 0, 0, 1}
α^{15}	$\alpha^{15} = 1$	{0, 0, 0, 1}

Table 3.7: Polynomials in GF(2^4)

Which yields a parity check matrix of:

$$\mathbf{H} = (\alpha^{14}, \alpha^{13}, \alpha^{12}, \alpha^{11}, \alpha^{10}, \alpha^{9}, \alpha^{8}, \alpha^{7}, \alpha^{6}, \alpha^{5}, \alpha^{4}, \alpha^{3}, \alpha^{2}, \alpha^{1}, 1)$$

$$\left(\begin{array}{cccccccccc|cccc} 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right)$$

3.18.1 Encoding Using Hamming Codes

To encode a message $m = \begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix}$ using the Hamming $[7,4,3]$ code, multiply the message by the generator matrix given in Equation 3.6 to obtain the codeword x, so that $x = m \times \mathbf{G}$:

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

Since the first k (=4) columns of the generator matrix consist of the identity matrix, the original message is represented by the first k (=4) bits of the codeword.The Hamming $[7,4,3]$ codewords corresponding to each possible 4-bit message are shown in Table 3.1. Once the message has been encoded, the codeword can be transmitted via the channel.

3.18.2 Decoding Hamming Codes

If an error occurs during transmission (via a binary symmetric channel), the transmitted codeword x will be received as y. If the the codeword $x = \begin{pmatrix} 0_1 & 1_2 & \mathbf{0}_3 & 0_4 & 1_5 & 0_6 & 1_7 \end{pmatrix}$ incurs an error in bit 3 during transmission, where bit 1 is the left-most bit and bit 7 is the right-most bit, then the codeword would be received as $r = \begin{pmatrix} 0_1 & 1_2 & \mathbf{1}_3 & 0_4 & 1_5 & 0_6 & 1_7 \end{pmatrix}$.

The codeword can be corrected by calculating the *syndrome* associated with the received codeword. The syndrome is the set of parity check results obtained by multiplying the received codeword by the parity check matrix. The parity check matrix could be used in its standard form, however for Hamming codes, the syndrome can represent a binary number by re-ordering the columns of the parity check matrix appropriately. This gives not only the result of the checks but also the position of the error. This is because as a *Perfect* code, exactly one syndrome exists for every possible codeword. A syndrome of 0 indicates that no error has been detected.

The parity check matrix used to decode the codeword is therefore:

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

Multiplying the received codeword by the transpose of the parity check matrix, so that $s = r \times \mathbf{H}^T$, yields a syndrome of:

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

$\begin{pmatrix} 0 & 1 & 1 \end{pmatrix}$ is the binary equivalent of 3 and therefore the syndrome indicates that a transmission error has occurred in the third bit of the codeword. The codeword can be corrected to give an estimate of the transmitted codeword $y = \begin{pmatrix} 0 & 1 & \mathbf{0} & 0 & 1 & 0 & 1 \end{pmatrix}$. Since the message is contained by the first 4 bits of the codeword, an estimate of the

original message is decoded to be $m_{est} = \begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix}$. The received codeword has been correctly decoded.

Codes with a minimum Hamming distance of n can correct up to $(n-1)/2$ errors if n is odd or up to $\frac{n}{2}-1$ errors if n is even; else it can detect up to $n-1$ errors without correcting them. With its minimum distance of 3, the Hamming $[7,4,3]$ code only has the capability to correct single bit errors. If two errors occurred during transmission of the codeword $x = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}$ then the codeword would probably not be decoded correctly. If the two errors occurred at position 1 and position 6, then the received codeword would be $r = \begin{pmatrix} \mathbf{1} & 1 & 0 & 0 & 1 & \mathbf{1} & 1 \end{pmatrix}$.

Multiplying the received codeword by the transpose of the parity check matrix, so that $s = r \times \mathbf{H}^T$, yields a syndrome of:

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

This syndrome of $\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$ suggests that the codeword has been received with a single error in position 7. The codeword would be corrected to $y = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$, the first four bits of which give an estimate of the original message as $m_{est} = \begin{pmatrix} 1 & 1 & 0 & 0 \end{pmatrix}$. The message has been incorrectly decoded due to the limited ability of the code to correct at most one error.

3.19 Golay Codes

Golay (1949) noticed that: $C_0^{23} + C_1^{23} + C_2^{23} + C_3^{23} = 2^{11}$. This equality shows the possible

existence of a perfect binary $[23, 12, 7]$ code, that achieves the Hamming bound and is capable of correcting all possible patterns of at most 3 errors in 23 bit positions.

There are two closely related binary Golay codes. The **perfect binary Golay code** encodes 12 bits of message data as a 23-bit codeword in such a way that any 3-bit errors can be corrected or any 6-bit errors can be detected.

The binary Golay code is the $[23, 12, 7]$ cyclic code generated by the polynomial:

$$\alpha^{11} + \alpha^{10} + \alpha^6 + \alpha^5 + \alpha^4 + \alpha^2 + 1$$

The weight distribution of the $2^{12} = 4096$ codewords is:

i	0	7	8	11	12	15	16	23
A_i	1	253	506	1288	1288	506	253	1

Both the code and its dual are proper i.e. the probability $P(\varepsilon)$ of an undetected error for the block code is monotonically increasing in ε for $0 \leq \varepsilon \leq \frac{1}{2}$, where ε is the probability of a symbol error (Leung-Yan-Cheong et al. 1979).

The $[24, 12, 8]$ **extended binary Golay code** is obtained from the perfect binary Golay code by adding a parity bit.

A generator matrix for the extended binary Golay code in standard $[I|A]$ form is:

$$\left(\begin{array}{cccccccccccc|cccccccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1
\end{array}\right)$$

The extended binary Golay code was used by the Voyager spacecraft programme in the 1980's for transmitting images of Saturn and Jupiter back to Earth.

There are also two closely related ternary Golay codes. The **ternary Golay code** is an $[11,6,5]_3$ linear code over a ternary alphabet; the relative distance of the code is as large as it possibly can be for a ternary code, and hence, the ternary Golay code is a perfect code. The **extended ternary Golay code** is a $[12,6,6]_3$ linear code obtained by adding a zero-sum check digit to the $[11,6,5]_3$ code.

The ternary Golay code is the $[11,6,5]$ CRC code (van Lint 1999) generated by the polynomial:

$$\alpha^5 + \alpha^4 - \alpha^3 + \alpha^2 - 1$$

It has a parity check matrix of:

$$\left(\begin{array}{cccccc|cccccc} 1 & 1 & 1 & 2 & 2 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 2 & 1 & 0 & 2 & 0 & 1 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 1 & 2 & 0 & 0 & 1 & 0 & 0 \\ 1 & 2 & 0 & 1 & 2 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 2 & 2 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{array}\right)$$

Its weight distribution is:

i	0	5	6	8	9	11
A_i	1	132	132	330	110	24

Both the code and its dual are proper.

3.20 Codes and Sphere Packing

3.20.1 Geometric Sphere Packing

In a lattice packing, if the lattice has its origin as a centre and there are spheres with centres u and v, then there are also spheres with centres $u+v$ and $u-v$. i.e. the set of centres form an additive group (Conway & Sloane 1999). In n-dimensional space, if we can find n centres $v_1, v_2, \cdots, v_n$, such that the set of all centres consists of the sums $\sum k_i v_i$, where the k_i are integers, then the vectors $v_1, v_2, \cdots, v_n$ form a basis for the lattice.

In the lattice in Figure 3.8, the n-dimensional space contains the following features:

- a is a lattice point
- b is a deep hole
- ρ is the packing radius of the lattice
- R is the covering radius of the lattice

If the sphere radius was R then the whole space would be covered (in this lattice, $R = 2\rho/\sqrt{3}$).

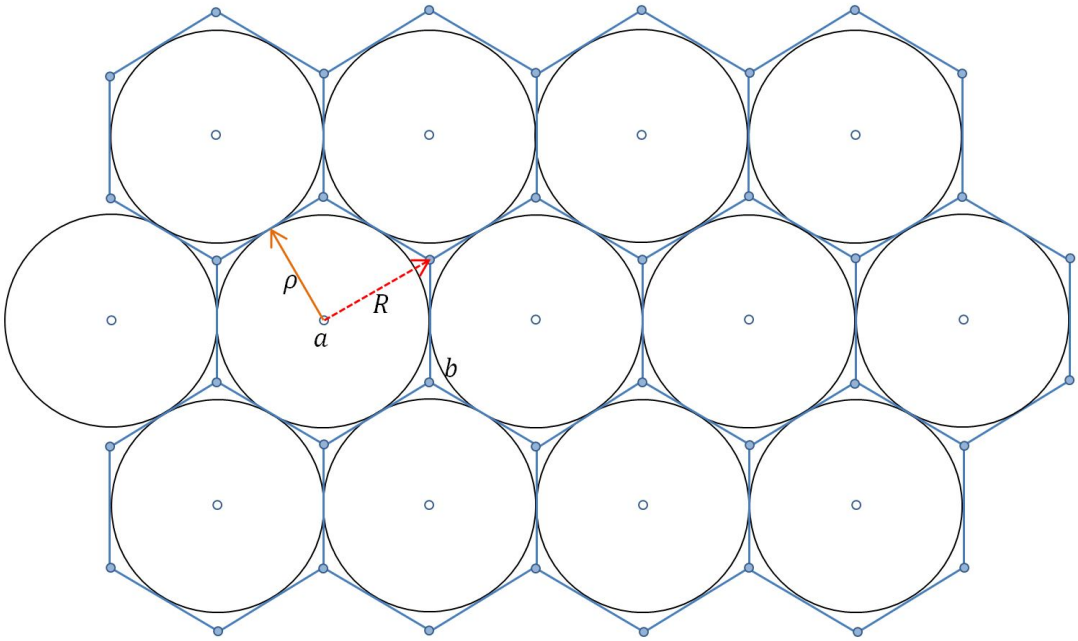


Figure 3.8: A Hexagonal Lattice (Conway & Sloane 1999)

A *fundamental region* for a lattice is a building block which when repeated many times fills the whole space with just one lattice point in each copy. The lattice in Figure 3.8 shows a hexagonal fundamental region.

The *packing density* is a measure of what fraction of the total space is taken up by the spheres.

$$\Delta = \text{Proportion of the space that is occupied by spheres}$$

$$= \frac{\text{Volume of one sphere}}{\text{Volume of fundamental region}} \tag{3.9}$$

3.20.2 Sphere Packing and Error Correcting Codes

Remembering that an $[n,k,d]$ code C over $\mathbb{F}_q$ can correct $t = \lfloor (d-1)/2 \rfloor$ errors, if t or fewer errors are made then the received vector can be uniquely decoded. If the number of errors is more than t but no more than the covering radius R, sometimes these errors can still be uniquely decoded.

3.20.3 Example of Sphere Packing and Error Correcting Codes

An example of the relationship between error-correcting codes and sphere packing can be seen with the Hamming $[7,4,3]$ code (Moser & Chen 2012). The code has $2^4 = 16$ valid codewords out of the $2^7 = 128$ possible code vectors. Each distinct pair of codewords of the Hamming $[7,4,3]$ code is separated by a Hamming distance of at least 3.

Geometrically, we can think of each valid Hamming codeword as a point in n-dimensional space with distance at least 3 from any other codeword. If each codeword is considered as being the centre of a sphere of radius $r = 1$, then the spheres will contain all the code vectors that have a Hamming distance of 1 from the valid codeword. In other words, the code vectors differ from the valid codeword in exactly 1 bit, or they contain a single error. No code vector can lie within two spheres as the spheres are too well separated.

For each valid codeword in the Hamming $[7,4,3]$ code, there are 7 code vectors that differ by 1 bit so, along with the codeword itself, there are 8 code vectors within each of the 16 spheres. Therefore every code vector containing a single bit error will lie closer to one particular valid codeword than to any other. Thus, correcting a single bit error is always possible for the Hamming $[7,4,3]$ code. Since every possible code vector is included within the 16 non-overlapping spheres, the Hamming $[7,4,3]$ code is a perfect code with the tightest possible packing of radius-1 spheres in the 7-dimensional binary space.

3.20.4 The Sphere Packing Problem

The Sphere Packing Problem (Conway & Sloane 1999, p. 1) seeks to investigate how densely a large number of identical spheres can be packed together. Finding the maximal number of non-overlapping radius-t spheres that can be packed into an n-dimensional binary space is the geometric equivalent of finding the maximal number of codewords that a t-error correcting code can have.

3.20.5 The Hamming Bound

For any non-negative integer R and codeword $\underline{u} \in \mathbb{F}^n$, then $S_R(\underline{u})$ denotes the sphere of radius R and centred on $\underline{u}$ (Hill 1986, p. 18), where:

$$S_R(\underline{u}) = \{\underline{v} \in \mathbb{F}^n | d(\underline{u}, \underline{v}) \leq R\} \tag{3.10}$$

The sphere is the set of code vectors for which the Hamming distance between the code vector and the original codeword is less than the sphere radius.

Working in n-dimensional space, each vertex of an n-dimensional cube is represented by a codeword of n 0's and 1's. The vector space consists only of the 2^n vertices - there is nothing else in the space of all possible messages except the 2^n vertices. The surface of a sphere of radius 1 about the point (0, 0,..., 0) is the set of all vertices in the space which are one unit away i.e. all vertices which have a single 1 in their coordinate representation. There are $\binom{n}{1}$ such points.

As with the Hamming $[7,4,3]$ code, the *volume* of a sphere of radius 1 is the centre point itself plus the n points with just one coordinate changed; a volume of $1+n$.

The total volume of the n-dimensional space is 2^n, the total number of possible points. Since the spheres do not overlap, the maximum number of message points k must satisfy (Hamming 1980, p. 46)

$$\frac{\text{total volume}}{\text{volume of a sphere}} \geq \text{maximum number of spheres} \tag{3.11}$$

So

$$\frac{2^n}{n+1} \geq 2^k \tag{3.12}$$

Or, (for a sphere of radius 1) since $n = m+k$

$$n+1 \leq 2^m \tag{3.13}$$

Extending to a sphere of radius R and a field $\mathbb{F}_q$, for $R < n$, $S_R(\underline{u})$ contains exactly:

$$|S_R(\underline{u})| = \begin{pmatrix} n \\ 0 \end{pmatrix} + \begin{pmatrix} n \\ 1 \end{pmatrix} (q-1) + \cdots + \begin{pmatrix} n \\ r \end{pmatrix} (q-1)^R = \sum_{i=0}^{r} \begin{pmatrix} n \\ i \end{pmatrix} (q-1)^i \quad (3.14)$$

points of $\mathbb{F}^n$.

Summing across all codewords within the code, if there is a q-ary (n, M, d)-code and $t = \lfloor (d-1)/2 \rfloor$ (t is the number of errors transmitted in the codeword) then the following inequality is satisfied:

$$M \left\{ \sum_{i=0}^{t} \begin{pmatrix} n \\ i \end{pmatrix} (q-1)^i \right\} \leq q^n \quad (3.15)$$

Or:

$$M \leq \frac{q^n}{\sum_{i=0}^{t} \begin{pmatrix} n \\ i \end{pmatrix} (q-1)^i} \quad (3.16)$$

In addition to earlier definitions relating to codewords and codeword distances, a *Perfect* code can be defined (Hoffman 1991) as a code that satisfies this equation with equality. For a binary code, this simplifies to:

$$M \leq \frac{2^n}{\sum_{i=0}^{t} \begin{pmatrix} n \\ i \end{pmatrix}} \quad (3.17)$$

Thus, the number of codewords in a code is limited by the number of distinct symbols employed by the channel (q), the number of symbols in the codeword (n) and the

number of errors that the code is to be able to correct (t). This can be written as

$$M|S_t| \leq q^n \tag{3.18}$$

For example with a $[5,3,2]$ binary code (q=2, n=5):

$$M\left(1+\binom{n}{1}\right) \leq 2^5 \qquad \text{i.e. } 6M \leq 32, \qquad \text{so } M \leq 5$$

For a $[23,12]$ code, there would be $2^{23} = 8388608$ possible code vectors. Of these, $2^{12} = 4096$ would be valid codewords, corresponding to 4096 possible messages. For each valid codeword (of length 23), there will be 1 code with with a Hamming distance of 0 (i.e. the codeword itself), $\binom{23}{1} = 23$ code vectors that differ by 1 bit, $\binom{23}{2} = 253$ code vectors that differ by 2 bits and $\binom{23}{3} = 1771$ code vectors that differ by 3 bits. Overall there will be $1+23+253+1771 = 2048$ code vectors that have a Hamming distance of 3 or less to each valid codeword. This is true for each codeword, therefore there will be $4096 \times 2048 = 8388608$ codewords in total. In other words and as noted earlier, every possible code vector of a $[23,12]$ code has a Hamming distance of 3 or less.

Since the Golay $[23,12,7]$ code has a minimum distance of 7, it has the ability to correct all code vectors that contain $t = \lfloor (d-1)/2 \rfloor = \lfloor (7-1)/2 \rfloor = 3$ or fewer errors. Hence the Golay code can correct every possible code vector. Applying $k = 12, n = 23$ and $t = 3$ to the Hamming bound,

$$\begin{aligned} 2^{12} &\leq \frac{2^{23}}{\binom{23}{0}+\binom{23}{1}+\binom{23}{2}+\binom{23}{3}} \\ &\leq \frac{2^{23}}{1+23+253+1771} \\ &\leq \frac{8388608}{2048} \\ &\leq 4096 \end{aligned} \tag{3.19}$$

The equation holds with equality and therefore by Equation 3.16, the Golay code is a perfect code.

If a code C has the property that there is an integer t such that the t-spheres around the codewords are disjoint and cover the whole of $\mathbb{F}^n$ then the code achieves equality in Equation 3.18. Tthe code is again perfect and possesses perfect packing.

A code with odd distance $d = 2t+1$ is perfect if and only if there is equality in the sphere packing bound i.e. $M|S_t(\underline{0})| \leq q^n$. Conversely for any perfect code, the distance d must be odd and $d = 2t+1$.

The difference between the two sides of the sphere packing bound is equal to the number of points that are not covered by the spheres.

If a code has minimal distance d, the 'Hamming Spheres' of radius $\rho = \frac{1}{2(d-1)}$ around the codewords are disjoint i.e. they have no element in common, (so ρ is the packing radius of the code) and therefore the code can correct ρ errors.

3.20.5.1 Perfect Codes

Tietäväinen (1973) showed that the only parameters satisfying the Hamming bound with equality are:

$$\begin{cases} n = 2^u - 1, & k = 2^u - u - 1, & t = 1 & \text{for any positive integer } u \\ n = 23, & k = 12, & t = 3 & \\ n = 2u+1, & k = 1, & t = u & \text{for any positive integer } u \end{cases}$$

The first case is a general Hamming code of order u. The second case has been shown (Pless 1968) to only hold for the Golay code. The third case is the $(2u+1)$-times repetition code i.e. repeating the message $(2u+1)$ times.

3.20.6 Other Geometric Properties of Lattices

Around each lattice point a_i is its *Voronoi cell*, $V(a_i)$, consisting of those points $\mathbb{R}_n$ that are at least as close to a_i as to any other P_j. Voronoi cells are also known as nearest neighbour cells. For a hexagonal lattice, the Voronoi cell is the hexagon (Conway &

Sloane 1999, p. 6).

The *kissing number* (τ) of a sphere packing in any dimension is the number of spheres that touch one sphere (Conway & Sloane 1999, p. 21). For a lattice packing, τ is the same for every sphere, but for an arbitrary packing, τ may vary from one sphere to another. The kissing number is also known as the Newton number, contact number, coordination number or ligancy. Values for the kissing number in the first 10 dimensions are shown in Table 3.8.

Dimension (n)	τ_n
1	2
2	6
3	12
4	24-25
5	40-46
6	72-82
7	126-140
8	240
9	306-380
10	500-595
⋮	⋮

Table 3.8: Kissing Number Values in Different Dimensions (Conway & Sloane 1999)

The *covering density* (or sparsity of the covering or *thickness*) (Conway & Sloane 1999, p. 31) is:

$$\Theta = \text{Average number of spheres that contain a point of the space}$$

The *covering problem* asks for the thinnest covering of n-dimensional space by spheres. For a plane, no other arrangement of circles (e.g. square lattice) can cover the plane more efficiently than the hexagonal lattice arrangement. However, as for packings, the optimal coverings are not known in higher dimensions.

A binary analogue to the covering problem is to find the smallest number of overlapping Hamming spheres that will cover $\mathbb{F}_q^n$. Equivalently, let the covering radius of a code C

be:

$$\max_x \min_c d(x,c)(x \in \mathbb{F}_2^n, c \in C)$$

Then the coding equivalent of the covering problem is, for a given code length and covering radius, to find the smallest possible number of codewords such that every possible codeword is within a fixed distance of a valid codeword.

3.21 Complexity Theory

Many of the algorithms in this work are highly intensive, requiring many calculations. Complexity theory gives some indication of the relative difficulty of different calculations. A deterministic algorithm, model or process is one whose resulting behaviour is entirely determined by its initial state and inputs, and which is not random or stochastic. A Turing machine is a hypothetical machine that can simulate any computer algorithm, no matter how complex. Most of the processes considered here are deterministic. Two important and relevant classes of complexity are:

- PTIME - Contains all decision problems that can be solved by a deterministic Turing Machine in Polynomial Time i.e. Using 'big O' notation to classify such problems in terms of how their run time or space requirements grow as the input size of n grows, such problems can be solved in $O(p(n))$ time where $p(n)$ is a polynomial of n. Cobham's thesis (Cobham 1965) holds that P is the class of computational problems that are efficiently solvable or tractable.

- EXPTIME - is the set of all decision problems that can be solved in exponential times ($O(2^{p(n)})$ time) by a deterministic Turing machine. EXPTIME problems are intractable in that no efficient algorithms exists for solving them, only a brute force approach.

Consider the functions $y = x^2$ and $y = 2^x$ shown in Figure 3.9. Visually, provided $x \gtrsim 3$, the exponential curve increases at a faster rate than the polynomial curve, crudely symbolising the difference in complexity between PTIME problems and EXPTIME prob-

lems. For $y = x^2$, $\frac{dy}{dx} = 2x$ and for $y = 2^x$, $\frac{dy}{dx} = 2^x \ln 2$. The gradient of the polynomial increases at a constant rate of $\frac{d^2y}{dx^2} = 2$, whereas the gradient of the exponential curve is itself continuing to increase exponentially at a rate of $\frac{d^2y}{dx^2} = 2^x(\ln 2)^2$. Whilst the actual value of the exponential function will be greater than that of the polynomial function if $x > 4$, above approximately $x = 3.21$ the gradient of the exponential curve will always be greater than that of the polynomial function. This represents the more rapidly increasing complexity of an EXPTIME problem over a PTIME problem.

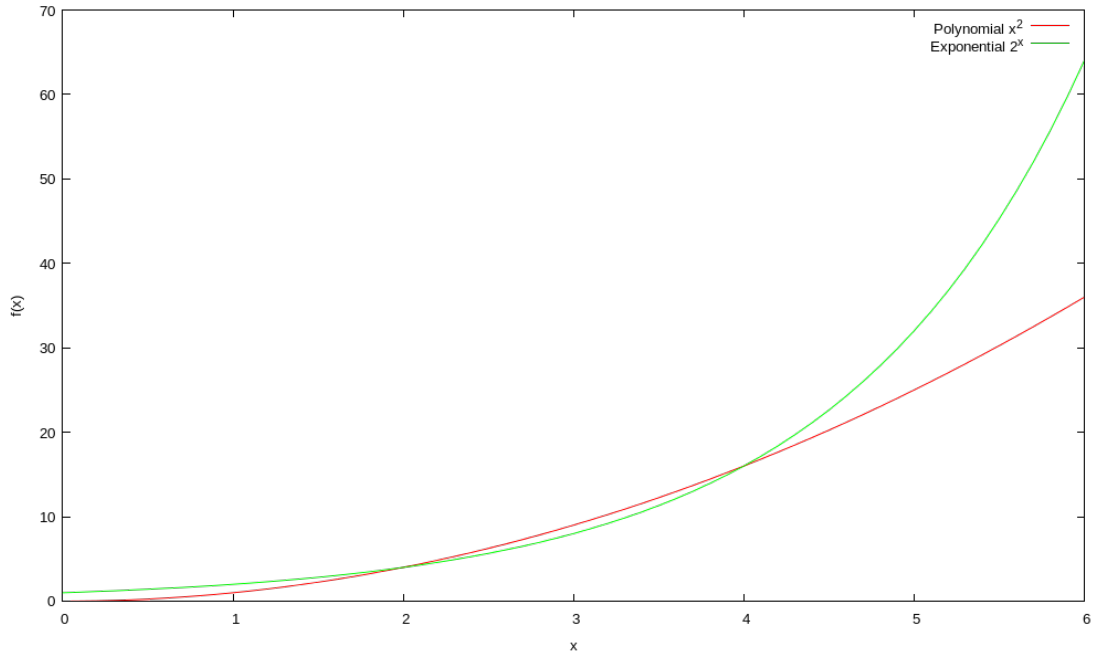


Figure 3.9: Polynomial rate versus Exponential rate

As an example, consider the determination of whether a number is prime or not. The number of steps in the calculation increases relatively slowly in comparison with the number of digits n in the number. Agrawal et al. (2004) showed that the calculation of whether a number is prime is a PTIME problem. Whilst not increasing at a constant rate, in general the number of steps increases according to something approaching the polynomial n^2 i.e. in polynomial time. Board games such as the generalised version of Go (with Japanese ko rules) are considered (Robson 1983) to be an EXPTIME problem because the number of moves available increases exponentially with the size of the board.

The calculations of equivocation for the different channel arrangements encountered in this thesis involve EXPTIME problems. For example, the calculation of equivocation for the Binary Symmetric Channel in Chapter 5 for an $[n,k,d]$ binary linear code must consider 2^k possible messages of length k and corresponding codewords of length n. For each of these, there are 2^n possible errors that could occur during transmission. Therefore to calculate the probabilities of all possible output messages for all input messages, there are $2^k \times 2^n = 2^{n+k}$ possible calculations to perform. The number of calculations increases exponentially with both the code length n and the message length k, rendering it an EXPTIME problem. Few shortcuts to reduce the calculation time were identified, thereby requiring a brute force approach to the calculation. This makes the calculations extremely labour intensive and formed a time-bounded constraint to the length of codes for which results could be evaluated. Such limitations formed a key constraint on the length of codes able to be analysed.

3.22 Conclusion

The fields of coding, sphere packing and complexity theory are each very large and full justice cannot be given to them in this chapter. Instead the aim of this chapter has been to give an overview of the most salient and relevant points in order to help place the main body of the thesis within the wider field. Of the code types discussed, perfect codes and BKLCs will be given the most attention in subsequent chapters. This is primarily to restrict the code choice to a manageable set of relatively straightforward codes that enable slightly more simple calculations to be completed by a single-purpose program in an acceptable time-frame, e.g. less than 2 days. This approach could potentially be extended in future work to other codes such as Hadamard codes, other Reed-Muller codes or Low Density Parity Check Codes (LDPC), although the size of LDPC codes would most likely prove problematic. The use of the approach with an iterative process such as that used for turbo codes would also be likely to be problematic in terms of the resulting calculation complexity; a factor which also affects the method's effectiveness at coping with deletions in Chapter 7.

Chapter 4

Channel Metrics

4.1 Introduction

Chapter 4 looks at some of the tools and measures that will be used to evaluate the information characteristics of a variety of codes when transmitted across different channels. This includes the definition and discussion of:

- Information
- Entropy
- Equivocation
- Mutual Information
- Capacity
- Information leakage

4.2 Information

The amount of information conveyed by a symbol was introduced by Shannon (1949) and developed further by Woodward & Davies (1952). Information is closely related to the amount of uncertainty or the amount of surprise in that event occurring and in an event x_i with probability p_i, it can be defined (Hamming 1980, p. 102) as:

$$I(x_i) = \log_2 \frac{1}{p_i} \tag{4.1}$$

When using base 2 logarithms, the unit of information is the 'bit of information'. If surprise is additive and the probabilities of two independent choices are multiplied together to get the probability of the compound event, then:

$$I(x_1) + I(x_2) = \log_2 \frac{1}{p_1 p_2} = I(x_1, x_2) \tag{4.2}$$

On average, for each symbol x_i, we get $p_i I(x_i)$ bits of information. Over the whole alphabet of q symbols x_i we will get an average amount of information (Hamming 1980, p. 104) of:

$$\sum_{i=1}^{q} p_i \log_2 \left(\frac{1}{p_i} \right) \tag{4.3}$$

4.3 Entropy

The average amount of information per symbol x_i of an information source X is known as the *entropy* of the information source, or equivalently, the uncertainty associated with the source. $H(X) = 0$ when the source is certain and $H(X)$ is maximal when all the x_i are equally likely.

The entropy is given by (Jones & Jones 2002):

$$H_r(X) = \sum_{i=0}^{q} p(x_i) \log_r \frac{1}{p(x_i)} \tag{4.4}$$

where r is the radix or root of the code. For a binary code, $r = 2$ and the entropy is:

$$H(X) = \sum_{i=0}^{q} p(x_i) \log_2 \frac{1}{p(x_i)} \tag{4.5}$$

The entropy function has the following properties:

- $H(X) \geq 0$

- $H(X) \leq \log_2 q$ where q is the number of input symbols
- $H(X) = \log_2 q$ when all the source symbols are equally likely

Note that 'the entropy of a source' has no meaning unless a model of the source is included e.g. for a Pseudo Random Number Generator, the numbers generated by a source would come very much as a surprise unless the formula used for generating them is known.

e.g. for a channel with $q = 4$ symbols (e.g. 00, 01, 10 & 11) occurring with equal probability, the entropy will take a maximal value of 2 ($= log_2 4$):

$$\begin{aligned} H(X) &= \sum_{i=1}^{4} p(x_i) \log \left(\frac{1}{p(x_i)} \right) \\ &= 0.25 \log_2 \left(\frac{1}{0.25} \right) + 0.25 \log_2 \left(\frac{1}{0.25} \right) + 0.25 \log_2 \left(\frac{1}{0.25} \right) + 0.25 \log_2 \left(\frac{1}{0.25} \right) \\ &= 0.5 + 0.5 + 0.5 + 0.5 \\ &= 2 \end{aligned}$$

However if the 4 symbols occur with the probabilities 0.4, 0.3, 0.2, and 0.1, the entropy will be 1.846:

$$\begin{aligned} H(X) &= 0.4 \log_2 \left(\frac{1}{0.4} \right) + 0.3 \log_2 \left(\frac{1}{0.3} \right) + 0.2 \log_2 \left(\frac{1}{0.2} \right) + 0.1 \log_2 \left(\frac{1}{0.1} \right) \\ &= 0.52877 + 0.52109 + 0.46439 + 0.33219 \\ &= 1.84644 \end{aligned}$$

Similarly for the received symbols (where s is the number of output symbols):

$$I(y_j) = p(y_j) \log_2 \frac{1}{p(y_j)} \tag{4.6}$$

$$H(Y) = \sum_{j=1}^{s} p(y_j) \log_2 \frac{1}{p(y_j)} \tag{4.7}$$

4.3.1 Binary Entropy Function

Supposing that X is a binary random variable such that:

$$X = \begin{cases} 1 & \text{with probability} \quad p_e \\ 0 & \text{with probability} \quad 1 - p_e \end{cases} \tag{4.8}$$

Then the entropy of X is in Equation 4.9 and plotted in Figure 4.1. Clearly there is little practical use in a channel with $p_e > 0.5$, since it has become more likely that an error will occur than not. When $p_e = 0.5$, the received bit has become random.

$$H(X) = \sum_{i=1}^{2} p_i \log \frac{1}{p_i} = -p_e \log_2 p_e - (1 - p_e) \log_2 (1 - p_e) \tag{4.9}$$

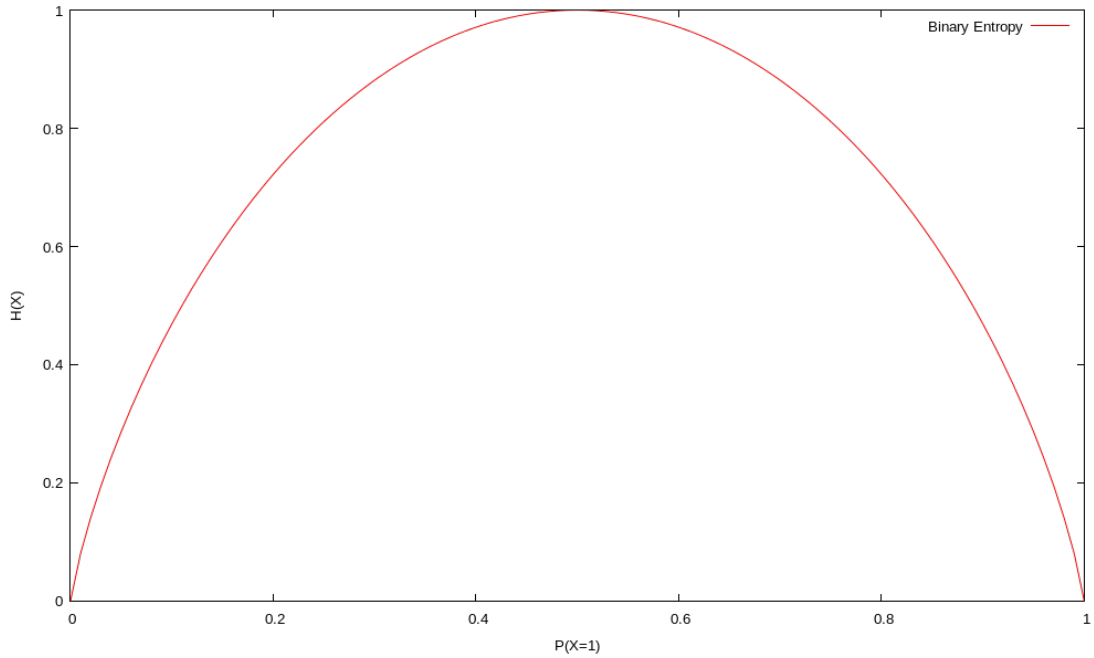


Figure 4.1: The Binary Entropy Function

This can also be written as $H(p_e)$ since the function depends solely on p_e

4.4 System Entropies

The relationships between input and output entropies, joint entropy, conditional entropies and mutual information are shown in Figure 4.2.

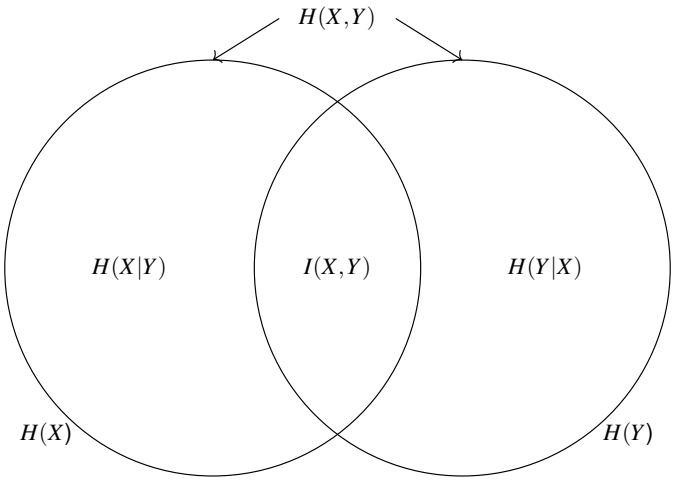


Figure 4.2: Relationship between entropy and mutual information

The joint entropy of a binary system can be given as

$$H(X,Y) = \sum_{x\in X}\sum_{y\in Y} P(x_i,y_j)\log_2\left(\frac{1}{P(x_i,y_j)}\right) \tag{4.10}$$

while the conditional entropy is

$$\begin{aligned} H(Y|X) &= \sum_{x\in X} P(x_i)H(Y|x_i) \\ &= \sum_{x\in X} P(x_i)\sum_{y\in Y} P(y_j|x_i)\log_2\left(\frac{1}{P(y_j|x_i)}\right) \\ &= \sum_{x\in X}\sum_{y\in Y} P(x_i,y_j)\log_2\left(\frac{1}{P(y_j|x_i)}\right) \end{aligned} \tag{4.11}$$

If input and output symbols are dependent, in the same way to that for probabilities $P(X)$ and $P(Y)$, where $P(Y|X) = P(X,Y) - P(X)$,

$$H(X|Y) = H(X,Y) - H(Y) \tag{4.12}$$

or

$$H(Y|X) = H(X,Y) - H(X) \tag{4.13}$$

The conditional entropy is the difference between the joint entropy and the source or output entropy. The conditional entropy $H(Y|X)$ represents the information loss in the channel going from input to output. It is how much must be added to the source entropy to get the joint entropy.

It is worth noting that in situations where $H(X) = H(Y)$, then $H(X|Y) = H(Y|X)$.

If the input X and output Y are statistically independent (i.e. what comes out doesn't depend on what goes in), then:

$$H(X,Y) = H(X) + H(Y) \tag{4.14}$$

4.5 Equivocation

In spoken English, one definition (Dictionary.com 2017) of the verb 'to equivocate' is 'using ambiguous language so as to conceal the truth or avoid committing oneself.' Hence when someone speaks clearly and unambiguously, they are said to be 'unequivocal'.

Similarly in information theory, equivocation was described by Shannon (1948, 1949) as a measure of the average amount of uncertainty in a received signal. It was defined as the conditional entropy of the system, since that represents the information loss of the channel going from input to output.

Shannon also demonstrated that the conditional entropy $H(X|Y)$ of the transmitted signal when the received signal is known is a natural measure of the uncertainty of what was actually transmitted, knowing only the perturbed version given by the received signal. Thus the equivocation of a channel is an appropriate mechanism for measuring the level of secrecy involved in the use of that channel. Shannon defined the *Secrecy* of the system as the conditional entropy, $H(X|Y) = H(X,Y) - H(Y)$.

Shannon's equivocation remains an important analysis tool for information security. Whilst other metrics for assessing secrecy have been proposed such as the security gap (Klinc et al. 2009) and a value function (Cuff 2010), equivocation continues to be recognised as an established metric (Klinc et al. 2009) and will be used in this work.

The probabilities of input codewords (x_i) and output codewords (y_j) can be used to calculate a set of entropies. In turn, these entropies can be used to calculate the equivocation of the code. For each given input message, the conditional probabilities $P(y_j|x_i)$ of each decoded message can be calculated. From the conditional probabilities, the overall joint probability $P(X,Y)$ can be calculated.

To improve communication security, the best codes can be the codes which have the highest value of the information secrecy (the equivocation rate), for a given code length and code rate, and are well-packed schemes. Such codes are known as the *best binary equivocation codes* and have been studied in more detail by Zhang et al. (2014).

4.5.1 Normalised Equivocation

Once the equivocation for a code has been calculated, it is useful to be able to make meaningful comparisons between the equivocation levels of different codes. To achieve this, it must be possible to compare like-for-like measures. This is done by calculating the *normalised* equivocation values by dividing the equivocation by the message length to give the normalised equivocation $\overline{H(X|Y)}$ as:

$$\overline{H(X|Y)} = \frac{H(X|Y)}{k} \tag{4.15}$$

This has the effect of scaling the equivocation value to lie between 0 and 1, to give the mean equivocation per bit of transmitted data. Since the equivocation represents the information loss of the channel, the maximum possible amount of information that could be transmitted (and/or lost) by the channel is k. Therefore dividing the equivocation values by k will produce a normalised equivocation value for an $[n,k,d]$ code as required. For example while a Hamming$[31,26,3]$ code could have an equivocation

value in the range $0 \leq H(X|Y) \leq 26$ and the Golay$[23,12,7]$ code could have an equiovcation in the range $0 \leq H(X|Y) \leq 12$, the normalised equivocation of both codes would be in the range from 0 to 1, enabling a more direct comparison of the relative merits of each codes for a specific error probability.

4.6 Mutual Information

The *a priori probability* $P(x_i)$ is the probability of the input symbol x_i prior to reception (Hamming 1980, p. 138). The *a posteriori probability* $P(x_i|y_j)$ of the input symbol x_i is the conditional probability that x_i was sent given that y_j was received.

The change in probability measures how much the receiver learned from the reception of the symbol y_j. In an ideal channel with no noise, the *a posteriori* probability is 1, since we are certain from the received y_j exactly what was sent.

Mutual Information $I(x_i,y_j)$ is the difference between the information uncertainty before (the *a priori* probabilities) and after reception of a y_j (the *a posteriori* probabilities). It is the gain in information due to the receipt of y_j.

$$I(x_i,y_j) = \log_2 \frac{1}{P(x_i)} - \log_2 \frac{1}{P(x_i|y_j)} = \log_2 \frac{P(x_i|y_j)}{P(x_i)} \tag{4.16}$$

Similarly:

$$I(y_j,x_i) = \log_2 \frac{P(y_j|x_i)}{P(y_j)} \tag{4.17}$$

- $I(x_i,y_j) \geq 0$
- $I(x_i,y_j) = 0$ if and only if x_i and y_j are independent
- $I(x_i,y_j) = I(y_j,x_i)$

The average mutual information is:

$$I(X,Y) = \begin{cases} H(X)+H(Y)-H(X,Y) \\ H(X)-H(X|Y) \\ H(Y)-H(Y|X) \end{cases} \tag{4.18}$$

4.7 Channel Capacity

The *Channel Capacity* is the maximum amount of information that can be conveyed over all possible assignments of the $P(x)$ or the maximum possible error-free information transmission rate across the channel.

$$C_a = \max_{P(x)}\{I(X,Y)\} \tag{4.19}$$

4.7.1 Capacity of the Binary Symmetric Channel

On the BSC and considering all possible input and output options,

$$\begin{aligned} I(X,Y) = &-P(x=0,y=0)\cdot\log_2(P(x=0,y=0)) - P(x=0,y=1)\cdot\log_2(P(x=0,y=1)) \\ &- P(x=1,y=0)\cdot\log_2(P(x=1,y=0)) - P(x=1,y=1)\cdot\log_2(P(x=1,y=1)) \\ \\ = &-P(x=0)\cdot P(y=0|x=0)\cdot\log_2(P(x=0)\cdot P(y=0|x=0)) \\ &- P(x=0)\cdot P(y=1|x=0)\cdot\log_2(P(x=0)\cdot P(y=1|x=0)) \\ &- P(x=1)\cdot P(y=0|x=1)\cdot\log_2(P(x=1)\cdot P(y=0|x=1)) \\ &- P(x=1)\cdot P(y=1|x=1)\cdot\log_2(P(x=1)\cdot P(y=1|x=1)) \end{aligned} \tag{4.20}$$

Referring to Figure 2.2, on the BSC with error probability p_e, this will achieve its maxi-

mum value when X (and Y) is uniform, i.e. when $P(x_0) = P(x_1) = \frac{1}{2}$,

$$\begin{aligned} C_a = \max_{P(x)}\{I(X,Y)\} &= -\frac{1}{2}(1-p_e)\log_2\left(\frac{1-p_e}{2}\right) - \frac{1}{2}p_e\log_2\left(\frac{p_e}{2}\right) \\ &\quad -\frac{1}{2}p_e\log_2\left(\frac{p_e}{2}\right) - \frac{1}{2}(1-p_e)\log_2\left(\frac{1-p_e}{2}\right) \\ &= -p_e\log_2\left(\frac{p_e}{2}\right) - (1-p_e)\log_2\left(\frac{1-p_e}{2}\right) \\ &= -p_e\log_2(p_e) - (1-p_e)\log_2(1-p_e) + p_e\log_2 2 + (1-p_e)\log_2 2 \\ &= -p_e\log_2(p_e) - (1-p_e)\log_2(1-p_e) + \log_2 2 \\ &= 1 - p_e\log_2(p_e) - (1-p_e)\log_2(1-p_e) \\ &= 1 - H(X) \end{aligned} \tag{4.21}$$

4.7.2 Capacity of the Binary Erasure Channel

On a BEC with probability of erasure p_s, if n bits are transmitted then $(1-p_s)n$ bits are received correctly on average. For large n, it is very likely that the actual number of (correctly) received bits will be close to this average. Information can be transmitted reliably at a rate of at most $1-p_s$ bits per channel use i.e. for the BEC, $C_a = 1-p_s$.

4.7.3 Capacity of the Binary Deletion Channel

Mitzenmacher (2006) notes that:

> Currently, we have no closed-form expression for the capacity, nor do we have an efficient algorithmic means to numerically compute this capacity.

4.7.4 Capacity of other Channels

As the equivocation of codes transmitted across the BSEC channel is not directly examined here and the focus is on comparing code equivocations rather than capacity, the capacity of the other channels previously mentioned in Chapter 2, namely the BSEC

and WTC will not be discussed further here.

4.8 Information Leakage

Smith (2011) discusses how, for an eavesdropper, comparing the uncertainty about a source X before and after seeing the value of the output Y can indicate the information leakage available to the eavesdropper:

$$\text{leakage} = \text{initial uncertainty} - \text{remaining uncertainty} \tag{4.22}$$

which leads to defining leakage as *mutual information*

$$\text{leakage} = H(X) - H(X|Y) = I(X;Y) \tag{4.23}$$

4.9 Conclusion

This chapter has introduced the notion of information and extended it to consider the conditional entropy or equivocation of a code being transmitted across a channel in order to gain a measure of the level of ambiguity and inherent secrecy of the arrangement. Whilst equivocation and channel capacity are closely related, the main focus will be on calculating the normalised equivocation. Equivocation will be used extensively over the next 3 chapters to compare the qualities of different codes when transmitted across BSC, BEC and modifications of these.

Chapter 5

Calculation of Equivocation through Parallel Processing

5.1 Introduction

Chapter 5 details the methods used to calculate the equivocation of various codes when transmitted across the BSC. Subsequent chapters will extend the programming techniques used in this chapter to examine modifications of the BSC that incorporate intentional erasures and deletions as well as looking at the BEC. With adaptation, the techniques could potentially be transferred to the other arrangements well. Future work could investigate extending onwards to other channels such as the BSEC, BDC, Additive White Gaussian Noise (AWGN) Channel or Fading channels. The BDC would create technical challenges due to the difficulty of identifying the locations of deleted bits and the complexity of ensuing calculations, whilst the AWGN or Fading channels would potentially require a substantially revised approach to the calculation of conditional probabilities and equivocation values. Two slightly different routes through the calculations are taken in this chapter in order to reduce the time taken to perform the calculations. Once the initial process had been established, the process was adapted to enable parallel processing to be used. The use of a parallel processing method permitted substantial increases in efficiency and the calculation of equivocation for longer codes. Code expansion will also be introduced as a simple method for increasing the equivocation of a code, although the method brings with it a significant increase in the number of bits that need to be transmitted and received and a significant time penalty.

The key outcomes of Chapter 5 were published by Schofield et al. (2015).

5.2 Equivocation Calculations

Full calculation of the equivocation for a code can be very intensive. For an $[n,k,d]$ binary linear code, there are 2^k possible messages of length k and corresponding codewords of length n. For each of these, there are 2^n possible errors that could occur during transmission. Therefore to calculate the probabilities of all possible output messages for all possible input messages, there are $2^k \times 2^n = 2^{n+k}$ possible calculations to perform, which renders it an EXPTIME problem. So even for the very simple Hamming $[7,4,3]$ code, with only 16 possible messages, there are $2^{4+7} = 2^{11} = 2048$ calculations. For the Golay $[23,12,7]$ code, there are $2^{35} = 3.436 \times 10^{10}$ calculations: over 34 billion. This increases exponentially as the code lengths increase.

In order to perform these calculations for any significant length of code, a software solution was implemented. The solution uses some symmetrical properties of codes and parallel processing to achieve a more efficient calculation.

The development of a software solution using parallel processing for calculating the equivocation of a code was done in three stages:

1. Develop the calculation on a spreadsheet for a very short code as a proof of concept.
2. Develop the calculation in C++ using linear programming.
3. Develop the calculation in C++ / CUDA using parallel processing.

5.2.1 Calculating Equivocation - Method 1

Two main linear methods for calculating equivocation were implemented. Each yields the same net calculation via slightly different pathways. A summary of the encoding / correction / decoding process is shown in Figure 5.1. The original message m is encoded as the codeword x before transmission across the BSC. Upon receipt, the codeword r may have been affected by an error vector v and must be corrected by

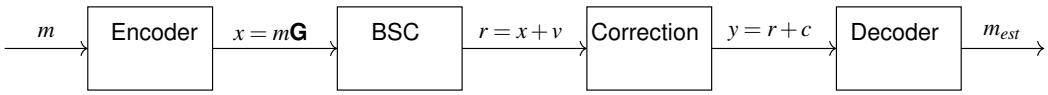


Figure 5.1: BSC encoding / decoding process

the addition of a vector c to obtain the codeword y. Finally, the corrected codeword is decoded to the message estimate m_{est}. If the generator matrix was in standard form, then the decoding process is just a case of taking the first k bits of the received, corrected codeword. Compared to the earlier description of the BSC in Figure 2.3, the processes carried out on the received codeword r to obtain the message estimate have been separated into two clear components; the attempt to correct any error and the decoding of the corrected codeword to obtain the message. This is to enable a clearer description of the two processes. hannon

The first algorithm which was used to calculate the equivocation of a code during early iterations of developing a software solution to the calculation is shown below:

1. Take an input message, m_i and its codeword, $x_i = m_i\mathbf{G}$.
2. Add each of the 2^n possible error vectors e_j to find each of the 2^n possible received codewords, $r_j = x_i + v_j$ and their respective syndromes, $s_j = r_j\mathbf{H}^T$.
3. Using the syndromes as a guide, calculate the correction c_j to be applied to each received vector.
4. Calculate the corrected version, $y_j = r_j + c_j$ of each of the possible 2^n received codewords and decode to find the message estimate, m_{est}.
5. Find the probability, $P(y_j|x_i)$ of receiving that message (or codeword), given the input message.
6. Use these probabilities to find the equivocation, $H(Y|X)$.

A working spreadsheet example was produced for both the Hamming $[7,4,3]$ Code and the Extended $[8,4,4]$ version of the code. Even for these small codes, with either $2^7 = 128$ or $2^8 = 256$ possible errors, the full calculation is becoming difficult to reproduce in document format.

For the Extended Hamming $[8,4,4]$ code, there are $2^4 = 16$ different possible input messages. Once encoded and transmitted as an 8-bit codeword, there are $2^8 = 256$ different possible combinations of errors that could be introduced during transmission across the channel:

$$v_0 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$v_1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\vdots$$

$$v_{254} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

$$v_{255} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

When transmitting the codeword via a binary symmetric channel, the probability of an error occurring in any single bit takes a constant value of p_e. Thus the probability of an 8-bit codeword being transmitted and there being no errors introduced is: $(1-p_e)^8$. So if $p_e = 0.01$, the probability of the codeword being received without any errors is $(1-0.01)^8 = 0.99^8 = 0.9227$ (4 $s.f.$).The probability of an error being transmitted only in a single specified position is $0.01 \times (1-0.01)^7 = 0.01 \times 0.99^7 = 0.00932$ (5 $s.f.$). But since a single error is equally likely to occur in any position, the probability of a single error being transmitted in any position is: $\binom{8}{1} 0.01 \times (1-0.01)^7 = 0.07456$ (5$s.f.$). Similarly, the probability of two errors occurring at any location is $\binom{8}{2} 0.01^2 \times (1-0.01)^6 = 0.00264$ (5 $s.f.$). More generally, for an n-bit codeword, the probability of there being exactly ε errors transmitted is

$$P(\varepsilon) = \binom{n}{\varepsilon} p_e^{\varepsilon} (1-p_e)^{n-\varepsilon} \tag{5.1}$$

Once the probability of errors occurring is known, this can be used to find the entropy and the equivocation of the code.

For the Extended Hamming $[8,4,4]$ code, in standard format, the generator and parity check matrices are shown in Equation 5.2 and Equation 5.3.

$$\mathbf{G} = \left(\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{array}\right) \tag{5.2}$$

$$\mathbf{H} = \left(\begin{array}{cccc|cccc} 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right) \tag{5.3}$$

If the message $m_0 = \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix}$ is encoded as the input codeword $x_0 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$, then there are $2^8 = 256$ possible error vectors v_j, where $j = 0,1,...,(2^8-1)$ that could be introduced during transmission with corresponding vectors $r_j = x_0 + v_j$ that could be received. However only $2^4 = 16$ of the vectors will yield valid codewords. The probability of all of these outcomes can be calculated and recorded and is best done by considering the error vectors in increasing weight order, as in Table 5.1. When multiplied by the transpose of the parity check matrix, each received codeword will give one of the $2^4 = 16$ different possible syndromes, $s = y\mathbf{H}^T$.

The error vector with the lowest weight that yields each syndrome will be correctly decoded. This error vector c is then used as the correction that is 'subtracted' (or added in base 2) from the received vector each time that syndrome is obtained, giving the corrected version of the received vector $y = r + c$. An estimate of the original message, m_{est} can then be obtained from the 'corrected' codeword y.

However, if more than one error occurred during transmission, some of the corrections that are applied to the received codeword will be wrong, because the code can only correct 1 error. Hence in some cases the wrong estimate of the original message will be

obtained. For the 256 possible output messages, 16 will have been correctly 'corrected' to the message $m_0 = \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix}$ and, for the remaining 240 possible outputs, 16 will have been incorrectly 'corrected' to each of the 15 other possible messages, from $m_1 = \begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix}$ to $m_{15} = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}$.

For example,

- the error vector $\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$ with weight 1 is the first vector that yields a syndrome of $\begin{pmatrix} 0 & 1 & 0 & 1 \end{pmatrix}$.
- the error vector $\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}$ with weight 2 also yields a syndrome of $\begin{pmatrix} 0 & 1 & 0 & 1 \end{pmatrix}$.
- therefore $\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}$ is corrected by subtracting (or adding) $\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$ to give a 'corrected' codeword of $\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$
- this gives an incorrect message estimate of $m_{est} = m_1 = \begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix}$

Message	Transmitted Codeword			
0 0 0 0	0 0 0 0 0 0 0			
Received Codeword	Probability $P(r_j \mid x_0)$	Syndrome	Corrected Codeword Estimate y_j	Message Estimate
00000000	0.922744694428	0000	00000000	0000
00000001	0.009320653479	0001	00000000	0000
00000010	0.009320653479	0010	00000000	0000
00000100	0.009320653479	0100	00000000	0000
00001000	0.009320653479	1000	00000000	0000
00010000	0.009320653479	0101	00000000	0000

Message	Transmitted Codeword			
0 0 0 0	0 0 0 0 0 0 0			
Received Codeword	Probability $P(r_j\|x_0)$	Syndrome	Corrected Codeword Estimate y_j	Message Estimate
00100000	0.009320653479	0110	00000000	0000
01000000	0.009320653479	1001	00000000	0000
10000000	0.009320653479	1010	00000000	0000
00000011	0.000094148015	0011	00000000	0000
00000101	0.000094148015	0101	00010101	0001
00000110	0.000094148015	0110	00100110	0010
00001001	0.000094148015	1001	01001001	0100
00001010	0.000094148015	1010	10001010	1000
00001100	0.000094148015	1100	00000000	0000
00010001	0.000094148015	0100	00010101	0001
00010010	0.000094148015	0111	00000000	0000
⋮	⋮	⋮	⋮	⋮
10111111	9.90E-15	0110	10011111	1001
11011111	9.90E-15	1001	10011111	1001
11101111	9.90E-15	1010	01101111	0110
11110111	9.90E-15	0111	11100101	1110
11111011	9.90E-15	1011	10111001	1011
11111101	9.90E-15	1101	11100101	1110
11111110	9.90E-15	1110	11010110	1101
11111111	1.00E-16	1111	10011111	1001

Table 5.1: Hamming $[8,4,4]$ Code - Error Dependent Syndrome Decoding

Once all possible message estimates have been established, the probability of getting

each output codeword given the input codeword, $P(y_j|x_i)$ can be calculated by summing the individual probabilities. So:

$$P(y_j|x_0) = \sum_{y_j=x_0} P(r_j|x_0) \tag{5.4}$$

$\Longrightarrow$

$$\begin{array}{lcll} P(y_0|x_0) & = & 0.9227 + 8\times 0.00932 + 7\times 9.41\times 10^{-5} & = 0.997969 \\ P(y_1|x_0) & = & 3\times 9.41\times 10^{-5} + 5\times 9.51\times 10^{-7} + 5\times 9.61\times 10^{-9} & \\ & & +3\times 9.70\times 10^{-11} & = 0.000287 \\ \vdots & & & \vdots \\ P(y_{15}|x_0) & = & 9.41\times 10^{-5} + 4\times 9.51\times 10^{-7} + 5\times 9.61\times 10^{-9} & \\ & & +4\times 9.70\times 10^{-11} + 2\times 9.80\times 10^{-13} & = 9.80\times 10^{-5} \end{array}$$

Similarly, the other possible message estimates generate Table 5.2.

Message Estimate Probability
$P(y_0\|x_0) = 0.997969$
$P(y_1\|x_0) = 0.000287$
$P(y_2\|x_0) = 0.000289$
$P(y_3\|x_0) = 0.000192$
$P(y_4\|x_0) = 0.000289$
$P(y_5\|x_0) = 0.000192$
$P(y_6\|x_0) = 4.86\times 10^{-8}$
$P(y_7\|x_0) = 4.80\times 10^{-6}$
$P(y_8\|x_0) = 0.000289$
$P(y_9\|x_0) = 3.9\times 10^{-8}$
$P(y_{10}\|x_0) = 0.000192$
$P(y_{11}\|x_0) = 1.95\times 10^{-6}$
$P(y_{12}\|x_0) = 0.000192$
$P(y_{13}\|x_0) = 1.95\times 10^{-6}$
$P(y_{14}\|x_0) = 1\times 10^{-6}$
$P(y_{15}\|x_0) = 9.80\times 10^{-5}$

Table 5.2: Extended Hamming $[8,4,4]$ Code - Conditional Probabilities

Since the output is dependent on the input, $P(x_i,y_j) = P(x_i)P(y_j|x_i)$, so to calculate values of $P(x_i,y_j)$, the probabilities must be considered as possible outcomes of the whole data set and should be multiplied by $P(x_i) = \frac{1}{2^k}$. Since $P(x_i) = 1/2^4 = 1/16$, we obtain the probabilities in Table 5.3.

Message Estimate Probability
$P(x_0, y_0) = 0.06237$
$P(x_0, y_1) = 1.80 \times 10^{-5}$
$P(x_0, y_2) = 1.81 \times 10^{-5}$
$P(x_0, y_3) = 1.20 \times 10^{-5}$
$P(x_0, y_4) = 1.81 \times 10^{-5}$
$P(x_0, y_5) = 1.20 \times 10^{-5}$
$P(x_0, y_6) = 3.04 \times 10^{-9}$
$P(x_0, y_7) = 3.00 \times 10^{-7}$
$P(x_0, y_8) = 1.81 \times 10^{-5}$
$P(x_0, y_9) = 2.44 \times 10^{-9}$
$P(x_0, y_{10}) = 1.20 \times 10^{-5}$
$P(x_0, y_{11}) = 1.22 \times 10^{-7}$
$P(x_0, y_{12}) = 1.20 \times 10^{-5}$
$P(x_0, y_{13}) = 1.22 \times 10^{-7}$
$P(x_0, y_{14}) = 6.25 \times 10^{-8}$
$P(x_0, y_{15}) = 6.13 \times 10^{-6}$

Table 5.3: Hamming $[8,4,4]$ Code - Joint Probabilities

From Equation 4.10, the joint entropy for the code is the summation of the individual entropies, giving $H(x_0, Y)$ as

$$
\begin{aligned}
H(x_0, Y) = \quad & 0.06237 \log_2\left(\tfrac{1}{0.06237}\right) \\
& + 1.80 \times 10^{-5} \log_2\left(\tfrac{1}{1.80 \times 10^{-5}}\right) + 1.81 \times 10^{-5} \log_2\left(\tfrac{1}{1.81 \times 10^{-5}}\right) \\
& + 1.20 \times 10^{-5} \log_2\left(\tfrac{1}{1.20 \times 10^{-5}}\right) + 1.81 \times 10^{-5} \log_2\left(\tfrac{1}{1.81 \times 10^{-5}}\right) \\
& + 1.20 \times 10^{-5} \log_2\left(\tfrac{1}{1.20 \times 10^{-5}}\right) + 3.04 \times 10^{-9} \log_2\left(\tfrac{1}{3.04 \times 10^{-9}}\right) \\
& + 3.00 \times 10^{-7} \log_2\left(\tfrac{1}{3.00 \times 10^{-7}}\right) + 1.81 \times 10^{-5} \log_2\left(\tfrac{1}{1.81 \times 10^{-5}}\right) \\
& + 2.44 \times 10^{-9} \log_2\left(\tfrac{1}{2.44 \times 10^{-9}}\right) + 1.20 \times 10^{-5} \log_2\left(\tfrac{1}{1.20 \times 10^{-5}}\right) \\
& + 1.22 \times 10^{-7} \log_2\left(\tfrac{1}{1.22 \times 10^{-7}}\right) + 1.20 \times 10^{-5} \log_2\left(\tfrac{1}{1.20 \times 10^{-5}}\right) \\
& + 1.80 \times 10^{-7} \log_2\left(\tfrac{1}{1.80 \times 10^{-7}}\right) + 1.80 \times 10^{-8} \log_2\left(\tfrac{1}{1.80 \times 10^{-8}}\right) \\
& + 6.13 \times 10^{-6} \log_2\left(\tfrac{1}{6.13 \times 10^{-6}}\right) \\
& = 0.25717
\end{aligned}
$$

So far, the probabilities have been calculated given that the input message was $m_0 =$

$\left(0000\right)$. However these probabilities must be considered in light of all possible 2^k messages, of which m_0 is just one. No matter what message is input, the errors will affect the transmission process with the same probabilities as for m_0, giving a symmetry to the probability calculations for each message and associated transmitted codeword. Therefore the sum of the entropies of the possible inputs and outputs is as per Equation 5.5.

$$H(X,Y) = \sum_{x_i \in X} H(x_i,Y) = 2^k \times H(x_0,Y) = 16 \times 0.25717 = 4.02748 \tag{5.5}$$

If the 2^k input messages are equally likely, $P(x_i) = \frac{1}{2^k}$, the entropy of the source for the Extended Hamming $[8,4,4]$ Code is

$$H(X) = \sum_{x_i \in X} P(x_i) \log_2(\frac{1}{P(x_i)}) = 2^4 \times \frac{1}{2^4} \times \log_2 \frac{1}{1/2^4} = \log_2 2^4 = 4 \tag{5.6}$$

Therefore from Equation 4.13, the equivocation is

$$H(Y|X) = H(X,Y) - H(X) = 4.02748 - 4 = 0.02748 \tag{5.7}$$

This process can be repeated for any p_e, $0 \leq p_e \leq 0.5$.

5.2.2 Calculating Equivocation - Method 2

Once the first method had been developed and a sound understanding of the process of calculating equivocation had been gained, it became important to develop an equivalent method that gave the same result but with maximum efficiency and that lent itself to parallel processing.

For one message (m_0) and its codeword (x_0):

1. Add successively weighted error vectors.

2. Note for which errors each syndrome is obtained first. These are the errors that

can be correctly corrected.

3. Operating on batches of codewords at a time, add the 2^{n-k} error vectors to each of the 2^k codewords in turn

4. Record the weights and probabilities $P(r_j|x_0)$, $(0 \leq a \leq 2^{n-k})$ of the resulting received codewords.

5. Use these 2^n probabilities to find the equivocation.

Again using the Extended Hamming $[8,4,4]$ code as a worked example, by working through the set of possible error vectors in order until each possible syndrome has been obtained for the first time, Table 5.4 is produced. The first occurrence of each syndrome is highlighted in bold.

Message	Encoded Codeword	
$m_0 = 0000$	$x_0 = 0000000$	
Received Codeword r_j	Probability $P(r_j\|x_0)$	Syndrome
0 0 0 0 0 0 0 0	0.922744694428	**0 0 0 0**
0 0 0 0 0 0 0 1	0.009320653479	**0 0 0 1**
0 0 0 0 0 0 1 0	0.009320653479	**0 0 1 0**
0 0 0 0 0 1 0 0	0.009320653479	**0 1 0 0**
0 0 0 0 1 0 0 0	0.009320653479	**1 0 0 0**
0 0 0 1 0 0 0 0	0.009320653479	**0 1 0 1**
0 0 1 0 0 0 0 0	0.009320653479	**0 1 1 0**
0 1 0 0 0 0 0 0	0.009320653479	**1 0 0 1**
1 0 0 0 0 0 0 0	0.009320653479	**1 0 1 0**
0 0 0 0 0 0 1 1	0.000094148015	**0 0 1 1**
00000101	0.000094148015	0101
00000110	0.000094148015	0110

Message	Encoded Codeword	
$m_0 = 0000$	$x_0 = 0000000$	
Received Codeword r_j	Probability $P(r_j\|x_0)$	Syndrome
00001001	0.000094148015	1001
00001010	0.000094148015	1010
0 0 0 0 1 1 0 0	0.000094148015	**1 1 0 0**
00010001	0.000094148015	0100
0 0 0 1 0 0 1 0	0.000094148015	**0 1 1 1**
00010100	0.000094148015	0001
0 0 0 1 1 0 0 0	0.000094148015	**1 1 0 1**
00100001	0.000094148015	0111
00100010	0.000094148015	0100
00100100	0.000094148015	0010
0 0 1 0 1 0 0 0	0.000094148015	**1 1 1 0**
00110000	0.000094148015	0011
01000001	0.000094148015	1000
0 1 0 0 0 0 1 0	0.000094148015	**1 0 1 1**
01000100	0.000094148015	1101
01001000	0.000094148015	0001
01010000	0.000094148015	1100
0 1 1 0 0 0 0 0	0.000094148015	**1 1 1 1**

Table 5.4: Hamming $[7,4,3]$ Code - Error Dependent Syndrome Decoding

As before with Method 1, $P(y_j|x_0)$ can be calculated by summing the individual probabilities, so:

$$P(y_0|x_0) = 0.9227 + 8 \times 0.00932 + 7 \times 0.0000941 = 0.997969 \tag{5.8}$$

This value is used as before to contribute towards the calculation of the entropy values and the equivocation of the code.

Up until this point, the same algorithm has been used as for method 1. However now that all the lowest weight error vectors that generate unique syndromes have been found, the remainder of the process can be broken into more manageable, independent batches. The process is therefore able to be calculated using parallel processing.

To calculate $P(y_1|x_0)$, only the codewords identified in the previous step as yielding the first occurrence of each syndrome need to be used. These are added to the codeword generated by the message $m_1 = \begin{pmatrix} 0001 \end{pmatrix}$, namely $x_1 = \begin{pmatrix} 0010101 \end{pmatrix}$. This gives the received codewords and associated probabilities shown in Table 5.5.

Message	Encoded Codeword	
$m_1 = 0001$	$x_1 = 0010101$	
Received Codeword r_j	Probability $P(r_j\|x_1)$	Syndrome
00010101	9.51×10^{-7}	0000
00010100	9.41×10^{-5}	0001
00010111	9.61×10^{-9}	0010
00010001	9.41×10^{-5}	0100
00011101	9.61×10^{-9}	1000
00000101	9.41×10^{-5}	0101
00110101	9.61×10^{-9}	0110
01010101	9.61×10^{-9}	1001
10010101	9.61×10^{-9}	1010
00010110	9.51×10^{-7}	0011
00011001	9.51×10^{-7}	1100
00000111	9.51×10^{-7}	0111
00001101	9.51×10^{-7}	1101
00111101	9.70×10^{-11}	1110
01010111	9.70×10^{-11}	1011
01110101	9.70×10^{-11}	1111

Table 5.5: Hamming $[8,4,4]$ Code - Error Dependent Syndrome Decoding

Summing these probabilities as before gives Equation 5.9.

$$\begin{aligned} P(y_1|x_0) &= 3 \times 9.41 \times 10^{-5} \\ &+ 5 \times 9.51 \times 10^{-7} \\ &+ 5 \times 9.61 \times 10^{-9} \\ &+ 3 \times 9.70 \times 10^{-11} \\ &= 2.87 \times 10^{-4} \end{aligned} \tag{5.9}$$

The equivalent of Table 5.2 can be constructed by repeating this process for every possible message and its associated codeword. From this point onwards, the process for calculating the entropies and the equivocation of the code is the same as for Method 1.

Once the code equivocation has been calculated for a single probability of error on the BSC, the calculation can be repeated for multiple other probabilities. By doing this, a table and graph of normalised equivocation values (or equivocation per information bit) can be constructed as in Table 5.6 and Figure 5.2 for the Extended Hamming $[8,4,4]$ Code.

Probability of Error on BSC	Equivocation	Normalised Equivocation
0.01	0.0275	0.0069
0.05	0.4064	0.1016
0.10	1.1132	0.2783
0.15	1.8442	0.4610
0.20	2.4939	0.6235
0.25	3.0222	0.7556
0.30	3.4215	0.8554
0.35	4.7013	0.9253
0.40	3.8784	0.9696
0.45	3.9721	0.9930
0.50	4	1

Table 5.6: Extended Hamming $[8,4,4]$ Code - Equivocation and Normalised Equivocation values

Whilst Figure 5.2 shows normalised equivocation values for a single code, subsequent results will enable the comparison of normalised equivocation levels for different codes,

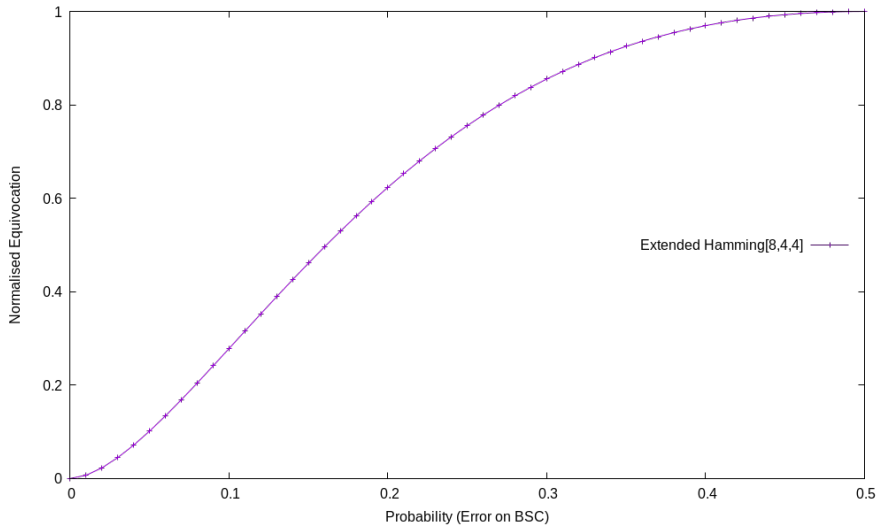


Figure 5.2: Equivocation for Extended Hamming $[8, 4, 4]$ Code for different BSC error probabilities

modifications of those codes and for different channel arrangements. Thus it becomes possible to identify codes and code modifications that yield improved equivocation levels and thereby offer greater levels of secrecy to legitimate users against illegitimate users on an eavesdropper channel of greater noise level.

5.3 Program Implementation

During the development of an efficient program to calculate the equivocation of a code, early iterations employed linear programming to work through a program sequentially. This created a need to repeatedly call sections of code many times. Shoup's Number Theory Library "NTL" (Shoup 2015) was used extensively during this phase, primarily for vector and matrix manipulation over the **GF**2 field.
NTL is described as:

> "a high-performance, portable C++ library providing data structures and algorithms for manipulating signed arbitrary length integers and for vectors, matrices, and polynomials over the integers and over finite fields."

In this respect, NTL worked well during the earlier stages of development, however as the development of the program continued and more parallel programming tech-

niques were introduced, the challenges of porting the library from a linear to a parallel methodology meant that it became of less use.

5.3.1 Useful Algorithms

Generation of binary numbers in numerical order

In order to step from one binary number to the next in numerical order, the algorithm below was used intially:

1. Start from the right-hand, least significant bit (lsb) of the current binary number, bit n
2. Add '1' to the lsb
3. If the result of the sum is a '0', add '1' to the next most significant bit. Repeat until a value of '1' results from the sum.

e.g. To generate the next binary number in numerical order after
$\begin{pmatrix} 1_1 & 0_2 & 1_3 & 0_4 & 0_5 & 1_6 & 1_7 & 1_8 \end{pmatrix}$

- Add '1' to the lsb '1' (bit 8), resulting in a lsb of '0'
- Add '1' to bit 7, becoming '0'
- Add '1' to bit 6, becoming '0'
- Add '1' to bit 5, becoming '1' - STOP
- Bits 4,3,2,1 unchanged
- This gives the next binary number in numerical order of
  $\begin{pmatrix} 1_1 & 0_2 & 1_3 & 0_4 & 1_5 & 0_6 & 0_7 & 0_8 \end{pmatrix}$

In later work, however, as the use of bitwise manipulation of numbers became the preferred method, this process became redundant as only a trivial increment of the variable was required. However, it did act as a useful stepping stone to the process of generating the binary numbers in weight order.

Generation of binary numbers in weight order

In order to step from one binary number to the next in weight order, the algorithm below was used:

1. Start from the right-hand, lsb of the current binary number
2. Working from lsb to most significant bit (msb) (i.e. from right to left), locate the first '1' bit that can be shifted left (i.e. the first '1' bit that has a '0' it the left of it)
3. Shift this bit left
4. Shift any bits to the right of this bit as far to the right as possible
5. If no bits exist that can be shifted left, then no further binary numbers of the current weight exist. Shift all i set bits back to the right-most positions and set the next MSB to 1.

e.g. To generate the next binary number in weight order after 184_{10}, $\begin{pmatrix} 1_1 & 0_2 & 1_3 & 1_4 & 1_5 & 0_6 & 0_7 & 0_8 \end{pmatrix}$,

- Starting from the right, bit '3' is the first '1' bit that has a '0' to the left of it and is therefore able to be shifted to the left
- Bit 3 is shifted left to position 2
- The remaining '1' bits to the right of this, in positions 4 and 5, must all be shifted as far right as possible i.e. to positions 7 and 8
- This gives the next binary number in weight order of $\begin{pmatrix} 1_1 & 1_2 & 0_3 & 0_4 & 0_5 & 0_6 & 1_7 & 1_8 \end{pmatrix} = 195_{10}$

e.g.2 From step 5, $\begin{pmatrix} 1_1 & 1_2 & 1_3 & 0_4 & 0_5 & 0_6 & 9_7 & 9_8 \end{pmatrix} = 224_{10}$ would automatically become $\begin{pmatrix} 0_1 & 0_2 & 0_3 & 0_4 & 1_5 & 1_6 & 1_7 & 1_8 \end{pmatrix} = 15_{10}$

When considering 5-bit numbers (for brevity), the process above will yield a set of binary numbers in weight order as shown in Appendix A.

5.3.2 Parallelisation of Calculations

The high volume of calculations needed to evaluate the equivocation of a code of any significant length means that on a standard laptop or desktop computer, the length of time needed to run a calculation is the dominant limiting factor (although memory can also be a limiting factor). When performing the calculation via a linear process, the implementation could calculate the equivocation for codes of length $n < 32$, but little more. To overcome this, stages of the calculation needed to be performed in parallel i.e. a single process that can be called many times concurrently. To parallelise the calculation, a computer capable of running Nvidia's CUDA architecture and programming model was used.

Graphics Processing Units evolved to take some of the load for managing the graphical output requirements of a computer away from the CPU. Over time, GPUs have become increasingly capable devices for performing relatively simple, repetitive operations in parallel and at high speed, extending their remit beyond graphics and evolving into General Purpose GPU (GPGPUs).

CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model created by Nvidia (Nvidia 2015). It is implemented by computers with CUDA-capable Nvidia GPUs, with the main CPU acting as the 'host' for the linear component of a program which then delegates responsibility for running parallelised sections of code to the GPU 'device' (Sanders & Kandbrot 2011). The task of efficient delegation and management of resources is largely done by the CUDA architecture and programming model.

In this case, a laptop with an i7 processor, 12GB of RAM and the CUDA enabled GeForce GTX470M GPU with 288 cores and 2GB of RAM was used. Different GPUs and versions have different capabilities. Whilst the GTX740M GPU is designed for a laptop device, it was still able to offer a CUDA compute capability of 3.5, enabling it to comfortably access a sufficiently capable version of the CUDA programming model.

The laptop was configured to run Ubuntu 14.04 LTS. Within this platform, Eclipse was used as an Integrated Development Environment for writing the code, along with the Nvidia CUDA Compiler NVCC and Nsight for Eclipse, an add-on for Eclipse to enable development of the CUDA specific code.

The main program is run on the CPU. A section of code within the main program, similar to a function and called a kernel, can be written for the GPU device to execute. Each instance of the kernel is called a thread. The CUDA architecture enables multiple threads to be run concurrently. These threads can be grouped together into blocks and the blocks can be grouped as warps. In order to pass references to variables between the host and the device, pointers were used.

Numerous limitations on memory capacity and processing speed, caused by the architecture and capability of the CPU and GPU, enforced compromises and constraints on block sizes and the grouping together of blocks into batches. For example with a setup with compute capability 3.5, a maximum of $2^{10} = 1024$ threads are permitted per block. Similarly, a maximum of $2^{31} - 1$ blocks are permitted. This implies a maximum of $2^{41} - 2^{10}$ threads per kernel call. In practice, however, the 2GB RAM of the GPU and the dynamic limitations of how much contiguous memory can be allocated to a single variable pointer meant that significantly smaller batches of blocks could be used at any time to prevent memory overflow.

One of the greatest sources of problems encountered during this work was secondary issues arising from using the GPU to parallel process the calculations, despite it providing more time-efficient calculations. The parallel processing load detracted significantly from the GPUs ability to simultaneously perform its more normal duties. This caused intolerable delays in user input and screen refresh rates, often causing the computer to crash. In addition, the significant time taken learning how to program in the CUDA environment created a large time overhead.

5.3.3 Implementation Iterations

The development of the software to calculate code equivocation was undertaken as a series of iterations, with each iteration evolving from and building on the previous one. Five iterations were produced that adopted a linear approach to programming before swapping to a parallel approach and producing fourteen further iterations. During the linear phase, NTL was used extensively, however this caused issued when trying to integrate the NTL functions with the CUDA architecture. The key differences between the parallel iterations are summarised in Table 5.7.

Iteration Number	Key developments
1	First conversion of a linear solution to a parallel solution. NTL functions used previously not working. Variables used with a kernel function must be declared locally or copied into CUDA memory space. Not considered feasible to do this with all background NTL variables.
2	Improved CUDA memory allocation and use of variable pointers.
3	Conversion of NTL **GF**2 format vectors to integer format, so that pointers to vector variables can be sent to the CUDA kernel. Enables error vectors to be dealt with in parallel.
4	Improved memory allocation and de-allocation. Codewords and error vectors treated as integers throughout, rather than being converted from **GF**2 vectors. More efficient rolling calculation of $H(X,Y)$.
5	Minor changes - housekeeping / tidying up
6	Minor changes only.
7	Creation of a second parallel processing task by a distinct kernel. The first kernel calculates the weights of the received vectors. The new kernel uses the vector weights to calculate the $P(X,Y)$ joint probability for each error probability p_e, a task previously done by a linear loop.

Iteration Number	Key developments
8	Improved efficiency of second kernel, calculating combined probabilities. Incorporation of mutual entropy calculation into second kernel, changing the sub-task from a linear one to a parallel one.
9	Attempt to calculate message encoding into a codeword as a parallel task within the first kernel. Discontinued since this could be done more efficiently by NTL in the linear part of the program.
10	Calculation is limited by hardware and architecture constraints. With a CUDA-imposed maximum of 1024 threads per block and a memory limit on the amount of block data that the GPU can store at any time, a compromise between linear and parallel processing must be introduced. The maximum number of blocks processed at any time is limited to a batch size of 1024. Linearly processed loops control the parallel process.
11	Interim version, aimed at improving integration of codeword production memory control into batch process.
12	Minor changes - housekeeping / tidying up.
13	Further optimisation of batch size control. Separation of task into a standalone function. This version is used for the majority of code equivocation and program run-time calculations presented in the report.
14	Interval between error probabilities decreased from 0.05 to 0.01. Equivocation values now being calculated for 51 values rather than 11, increasing calculation run times by a factor of 5. Yielding much smoother output graphs with fewer discontinuities, this version is used for the generation of most of the output graphs in the report.

Table 5.7: Key differences between software iterations

5.3.4 CUDA Coding

In the main part of a program, a CUDA kernel (Sanders & Kandbrot 2011) called `getWts` can be called by the CPU host by the code:

```
getWts«<*blocks,*threads»>(d_Ctx, d_EV, d_powK, d_powNK, d_n, d_wts);
```

The triple angled brackets indicate that the function is a kernel to be run on the GPU device and define the number of blocks and threads to be created by each instance of the kernel. Parameters in the brackets are pointer variables to be used by the kernel. As `malloc` can be used to allocate CPU memory for use by pointers, so `cudaMalloc` can be used in a similar fashion to allocate GPU memory for use by pointers within the device. Prior to calling a kernel, any required host variables must be copied to device memory, using the `cudaMemCpy` function. The code for the kernel to be run in parallel by the device is indicated by the `__global__` function.

```
__global__ void getWts(int* Ctx, int* EV, long*
powK, long* powNK, int* n, int* wts)
{
     long id = blockIdx.x * blockDim.x
     + threadIdx.x;
     long wt = 0;
     for (long i = 0; i < *powNK; i++ )
     {
          wt = 0;
          for (int j = 0; j < *n; j++)
          {
               if ( ( ( *(Ctx + id*(*n) + j) +
               *(EV + i*(*n) + j ) ) % 2 ) == 1)
               {
                    wt++;
               }
```

```
            }
            (*(wts + id * (*n+1) + wt ))++;
        }
}
```

The values held by variables such as `blockDim.x`, `blockIdx.x` and `threadIdx.x` identify the number of threads per block and the identity of the block or thread being executed. This enables each individual thread to access the specific data that it requires.

The central part of the function is used to add each bit of the error vector (pointed to by the variable '`EV`') to each bit of the transmitted codeword (pointed to by the variable '`Ctx`') and then to find the weight of the resultant vector. A record of the weight of each received vector is then stored in memory space allocated to the GPU device and pointed to by the variable '`wts`'. The nested '`for`' loops enable this to be carried out for each bit of every error vector. The parallelisation of the program enables the process to be done in batches for every possible codeword, thereby considering every possible combination of codeword and error vector. A second kernel then uses batch processing to calculate the summation of the probabilities of the received codeword weights, enabling the entropies and the equivocation to be calculated.

5.4 Results

A graph showing the normalised equivocation values of some perfect and extended perfect codes is shown in Figure 5.3 and reproduced in $\log - \log$ form in Figure 5.4.

Of the Hamming codes examined, the longer codes exhibited higher levels of normalised equivocation than the shorter ones. Hamming $[31,26,3]$ has higher normalised equivocation values than Hamming $[15,11,3]$, which in turn, has higher values than Hamming $[7,4,3]$. For $p_e \lesssim 0.17$, Hamming codes also have higher normalised equivocation values than the Golay $[23,12,7]$ code. However this is offset by the Golay code benefitting from its ability to correct up to 3 errors, whilst the Hamming codes can only

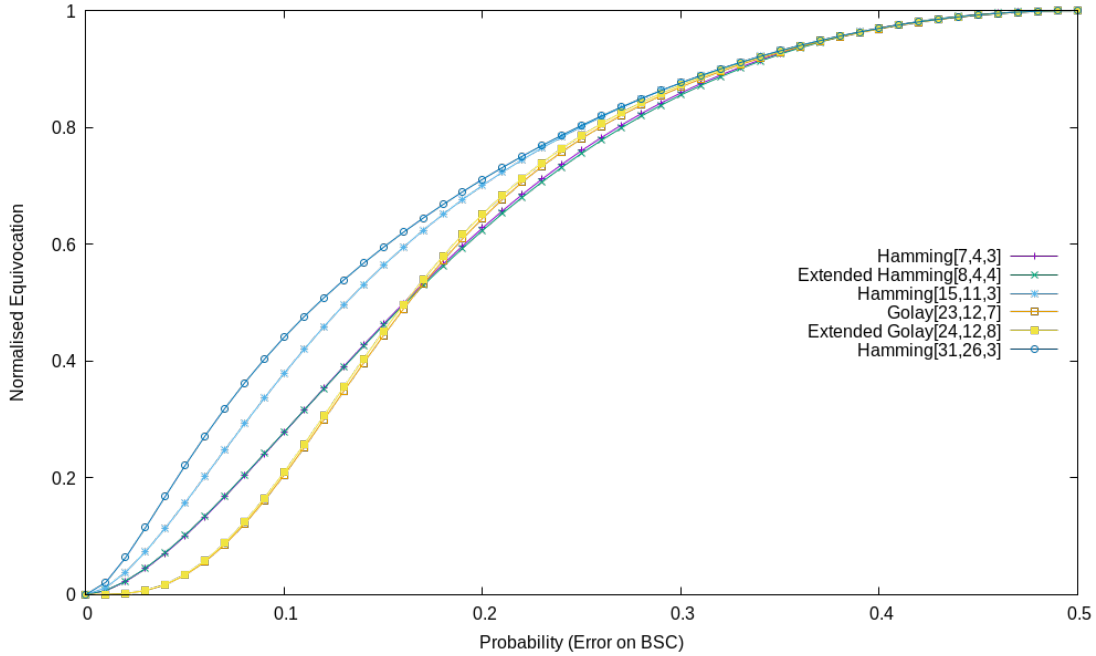


Figure 5.3: Normalized equivocation of some perfect codes and their extensions

correct a single error.

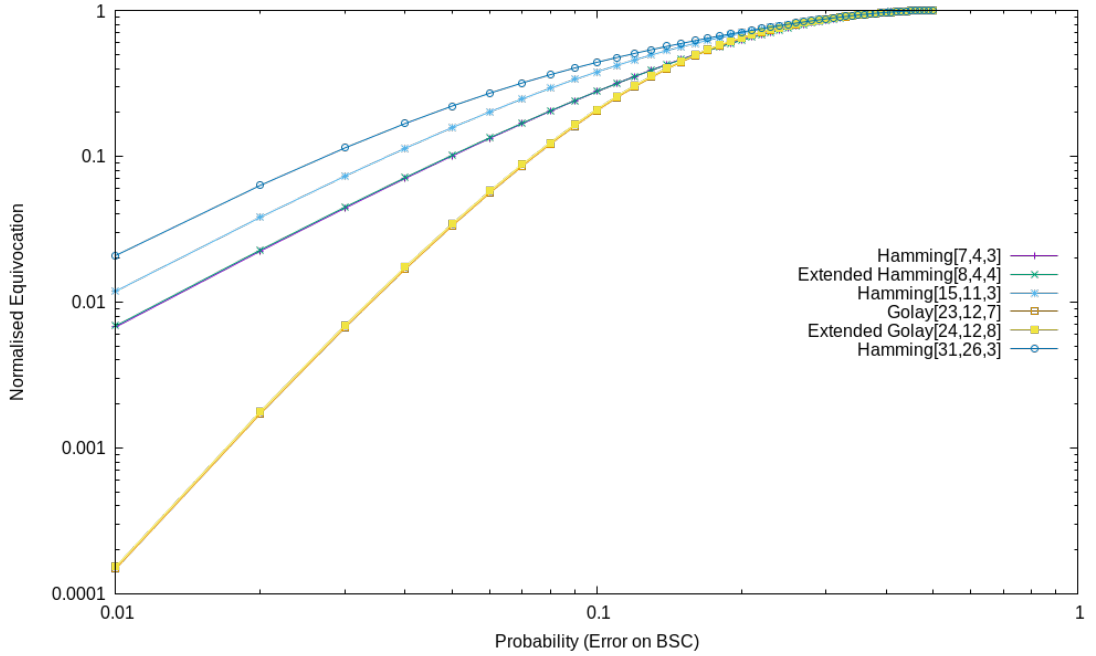


Figure 5.4: $log - log$ graph for normalised equivocation of perfect codes

A comparison of the normalised equivocation of the perfect Golay $[23, 12, 7]$ code and

the two random $[23, 12]$ codes, code A and code B, whose standard form generator matrices, $\mathbf{G}_A$ and $\mathbf{G}_B$ are given below was performed and the results are shown in Figure 5.5.. The first k columns of each matrix form a $k \times k$ identity matrix, while the remaining $n-k$ columns are populated with randomly distributed bits.

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

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

This shows that random codes can possess higher levels of equivocation (and therefore secrecy) than a perfect code, although their error correcting capabilities may be significantly less.

The normalised equivocation values of some longer Best Known Linear Codes are shown in Figure 5.6 and show comparatively little difference between the codes as code length increases.

A $\log - \log$ graph, concentrating on the range $0.01 \leq p_e \leq 0.1$ is shown in Figure 5.7 to highlight the similarity between the results.

The BKLCs used in Figure 5.6 were originally chosen to demonstrate the different lengths of time that the parallelised calculation would take as code lengths increased (discussed in subsection 5.4.1). They were not known to possess similar properties to each other, however as the code length increases, the gradients of their respective equivocation curves become increasingly similar to one another. The expanded log scale graph of Figure 5.7 highlights these similarities.

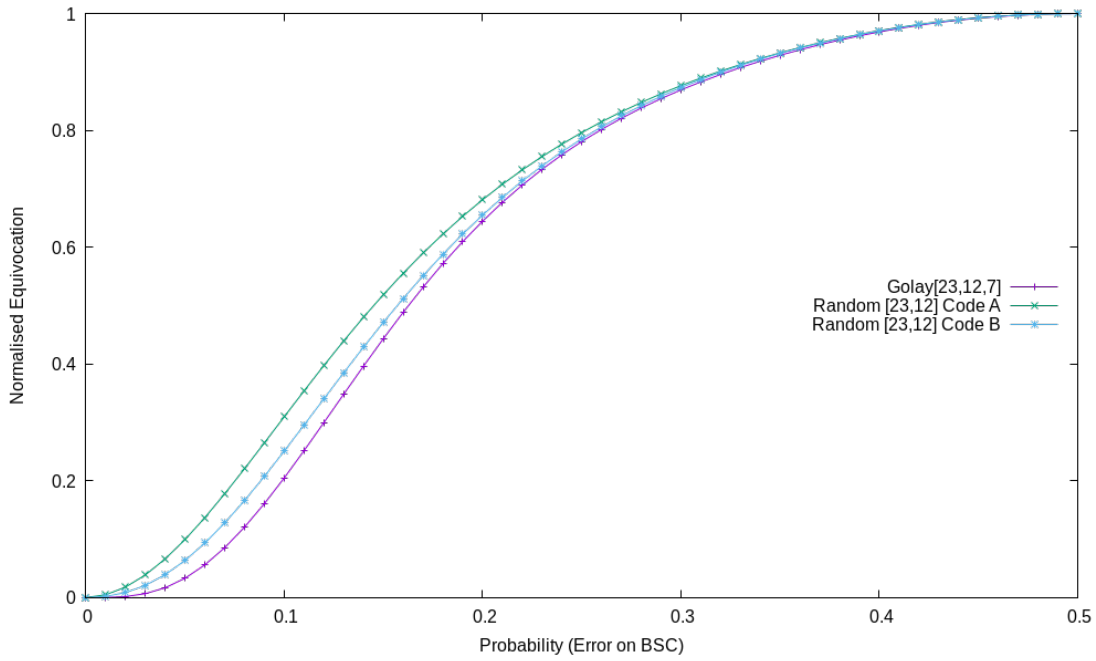


Figure 5.5: Normalized equivocation of Golay $[23, 12, 7]$ and two random $[23, 12]$ codes

For all codes examined, the normalised equivocation is an increasing function when expressed as a function of the probability of error p_e on a binary symmetric channel. However the rate of increase is often at its lowest in the ranges $0 \leq p_e \lesssim 0.05$ and $0.4 \lesssim p_e \lesssim 0.5$, while the rate of increase is often at its highest in the range $0.05 \lesssim p_e \lesssim 0.2$. Given that it may be common for a legitimate receiver to receive the signal through a channel with a low probability of transmission error and for an illegitimate receiver to receive the signal with a markedly higher probability of transmission error, this could be of use when developing codes that are designed to maximise the differences in ambiguity levels between a legitimate and illegitimate receiver.

Such an approach enables a move away from theoretical situations, where both the legitimate recipient and the eavesdropper have a perfect channel, towards situations where both channels may involve a level of signal degradation. By accepting and managing a level of degradation for the legitimate recipient, more coding schemes could be made available that provide a significantly higher level of equivocation for an eavesdropper than for the legitimate receiver. The nature of the channels becomes a

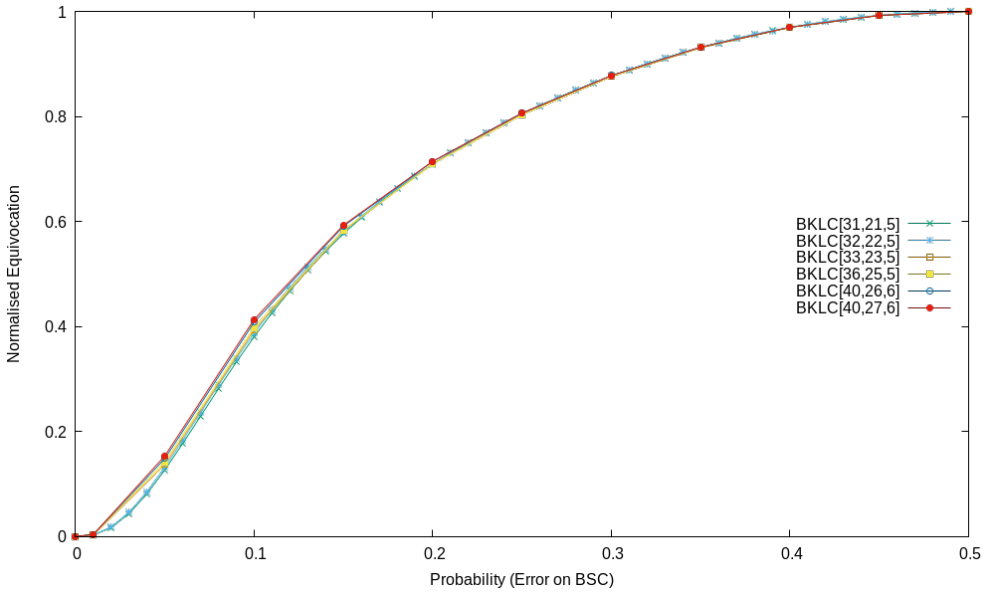


Figure 5.6: Normalized equivocation of longer Best Known Linear Codes

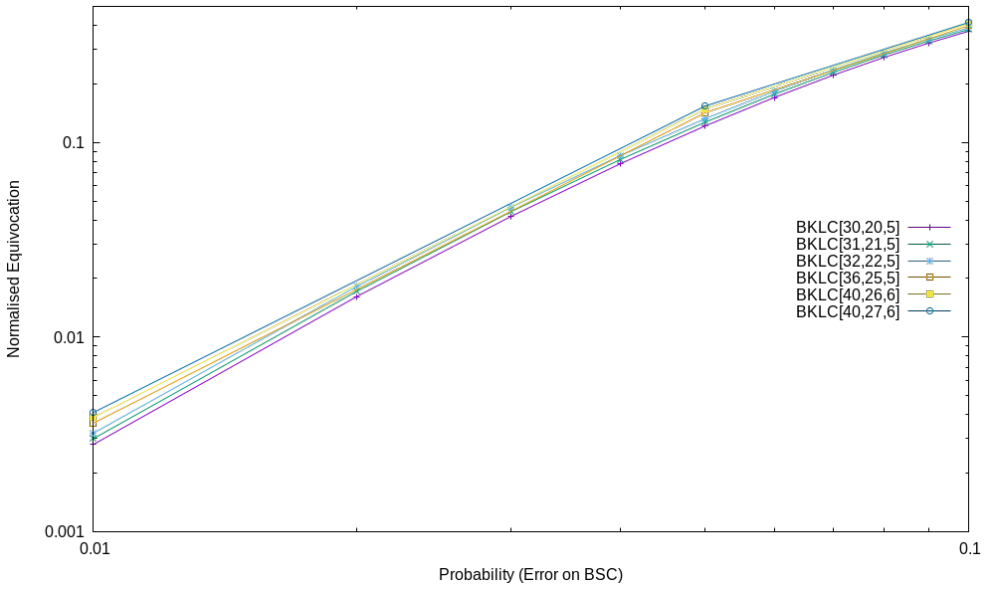


Figure 5.7: $\log - \log$ graph for normalised equivocation of BKLC

means of providing security.

For example, if when transmitting data using the Golay $[23, 12, 7]$ code on the BSC, if the legitimate receiver has a p_e of 0.01 whilst the eavesdropper has a net error probability of 0.1, the legitimate receiver will face a normalised equivocation level of 0.000148. The eavesdropper on the other hand has an equivocation value of 0.205, some 1385

times greater.

5.4.1 Calculation Times

The times taken to calculate equivocation values for some Best Known Linear Codes are shown in Table 5.8.

Code	Linear time (s)	Parallel time (s)	Linear : Parallel ratio
BKLC $[30,20,5]$	3466 ($\approx$ 58min)	105.99	33 : 1
BKLC $[31,21,5]$	6880 (1hr 54m)	238.6	29 : 1
BKLC $[32,22,5]$	13702 (3.8hr)	444	31 : 1
BKLC $[33,21,6]$	27048 (7.5hr)	780	35 : 1
BKLC $[33,23,5]$	27634 (7.7hr)	1021	27 : 1
BKLC $[36,25,5]$	not reasonably calculable	6760	-
BKLC $[36,26,4]$	not reasonably calculable	8878 (2.5 hr)	-
BKLC $[40,27,6]$	not reasonably calculable	91875 (25.5 hr)	-

Table 5.8: Time to calculate equivocation for longer BKLCs

The times taken to calculate equivocation values for some perfect codes and their extensions are shown in Table 5.9.

Code	Linear time (s)	Parallel time (s)	Linear : Parallel ratio
Hamming $[7,4,3]$	0.0174	0.068	1 : 4
Extended Hamming $[8,4,4]$	0.021	0.084	1 : 4
Hamming $[15,11,3]$	0.991	0.147	7 : 1
Golay $[23,12,7]$	67.996	4.48	15 : 1
Extended Golay $[24,12,8]$	238.6	17.8	13 : 1
Hamming $[31,26,3]$	33961	3809	9 : 1

Table 5.9: Time to calculate equivocation for some perfect codes and their extensions

Both Table 5.8 and Table 5.9 show a significant increase in the calculation times as the code length increases. Once code lengths approaching $n = 36$ or higher are considered, the linear calculation on a standard laptop takes sufficiently long as to cease to be reasonably calculable. In this work, an unreasonable calculation was considered to be one that took in excess of 2 days to complete. This was due to the requirement to run a privately owned laptop at maximum processor capability for extended periods of time. Whilst the linear method is actually quicker for very short length codes, the

parallel method soon becomes the preferred solution, producing a result more quickly for all code lengths where $n > 10$. The results for longer length codes show that significant gains can be made, with calculations being performed by the parallel processing method up to 35 times quicker than by the linear method in the instance of the BKLC $[33, 21, 6]$ code.

When considering the different ratios between the calculation times of the parallal processed and linear methods, a possible trend is visible but it is hard to draw any rigorous conclusions from these. Table 5.9 show a general improvement in the linear:parallel ratio from 1:4 for Hamming$[7, 4, 3]$ up to 15:1 for Golay$[23, 12, 7]$, however the addition of an extra bit to the Golay code to yield the extended Golay$[24, 12, 8]$ code does not continue that trend. The linear calculation time increases by a factor of 3.5 from 68s to 239s, whereas the parallel time increases by a factor of 4 from 4.5s to 17.8s, giving a slightly decreased ratio of 13:1 for the extended code. The additional bit has had a slightly greater impact on the software implementation underlying the parallel method than on the linear method. Similarly in Table 5.8, many of the ratios are in the order of 30:1 but, for the codes examined, there is no consistent or predictable pattern. Of all the codes examined, those listed in Table 5.8 gave the greatest ratio of improvement in calculation run-time from linear to parallel processing. However, the codes selected for examination in these 2 tables vary in both their code and message length, making it difficult to confirm the inter-related impact of either unless the factors are isolated from each other.

The calculation time is dependant upon many factors, including both the message length k, the codeword length n, the number of threads per block, blocks per batch and the structure of the calculation program. To highlight the effect of increasing k and n on calculation times, independent runs of calculations for increasing lengths of k and n were performed. The times taken to calculate equivocation values for some Best Known Linear Codes of message length $k = 15$ and increasing values of n are shown in Table 5.10. The times listed are for calculations that were performed for the

probabilities $0 \leq p_e \leq 0.5$ in 0.05 increments, with 0.01 as an additional value. Many of the calculations were subsequently re-performed for probabilities $0 \leq p_e \leq 0.5$ in increments of 0.01 in order to give more accurate and smoother output graphs.

Since these results were published, it has been observed that the calculation times exhibit greater variability on subsequent re-calculations than originally noted, so all times are now only given to 2s.f. rather than 3s.f. . Even so, there is still occasionally some variation around these values, although the values given form a fair median value. This variation is presumed to be due to the computer being in different states on each re-calculation, due to factors such as different background programs being in operation at the time.

Code	Parallel time (s)
BKLC $[20,15,3]$	1.4
BKLC $[21,15,4]$	1.5
BKLC $[22,15,4]$	1.6
BKLC $[23,15,4]$	2.3
BKLC $[24,15,4]$	5.8
BKLC $[25,15,5]$	4.1
BKLC $[26,15,6]$	12
BKLC $[27,15,6]$	66
BKLC $[28,15,6]$	430
BKLC $[29,15,7]$	430

Table 5.10: Calculation times for BKLCs with message length $k = 15$

A graph of these times is shown in Figure 5.8. The graph shows a general trend that as the code length n increases, the calculation time increases exponentially, as would be expected. However several anomalies exist. The calculation time for $n = 25$ is less than that for $n = 24$ while the calculation times for $n = 28$ and $n = 29$ are almost identical. This is not caused by the main body of the calculation where the $P(y_j|x_i)$ are calculated. Instead the irregularities stem from the initial process of setting up the error correction process and the identification of which codeword is the first one in weight order to yield each syndrome. This can take significantly different times for each code, depending on the structure of the code's generator (and parity check) matrix. There is also the potential for different parity check matrices of the same code to cause different

calculation run times, although this was not investigated in any detail here. Again, this would be due to differences in the order in which codewords are identified that are the first to yield each syndrome. It would appear that for the BKLC $[24,15,4]$ and BKLC $[28,15,6]$ codes the first codewords to yield each syndrome by weight order occur significantly later on average than for BKLC $[25,15,5]$ and BKLC $[29,15,7]$ codes respectively.

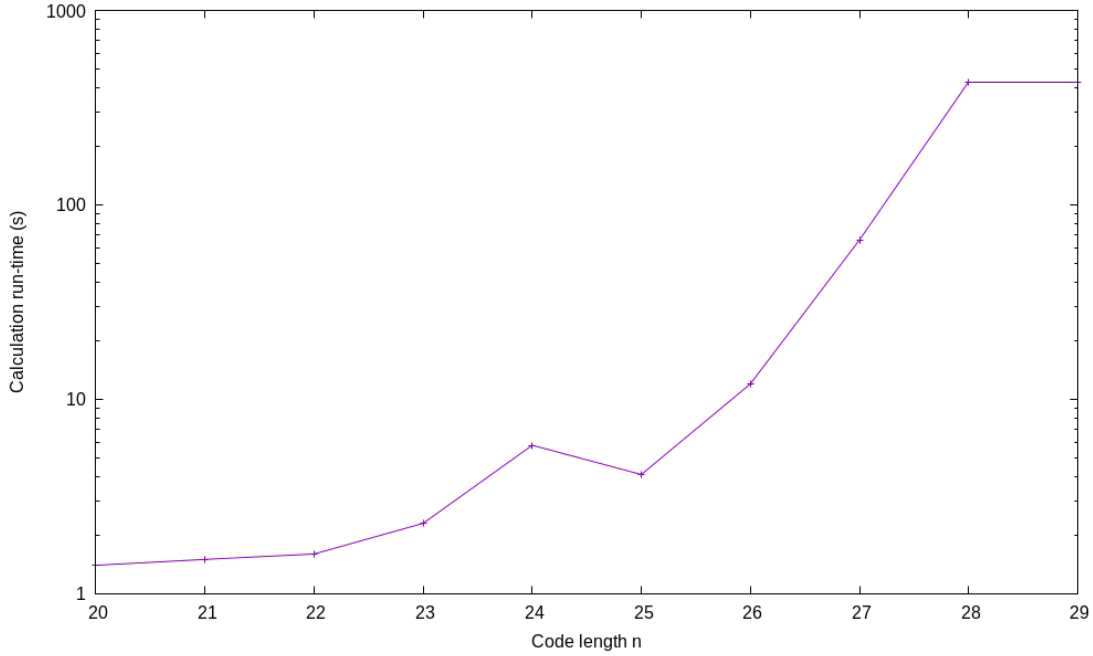


Figure 5.8: Time to calculate equivocation for codes of message length $k = 15$ for different code lengths (n)

The times taken to calculate equivocation values for some Best Known Linear Codes of code length n=30 and increasing values of k are shown in Table 5.11.

A graph of these times is shown in Figure 5.9.

It can be seen that for a fixed codeword length, there is an optimal message length k that gives a minimum calculation time. This occurs around code length $n = 19$ and is due to the calculation method used in the software. There are 2^{10} threads in each block and 2^{10} blocks per instance of the kernel of the parallel component of the program. Below a message length of $k = 20$, the parallel component of the program has not yet achieved maximum efficiency and above $k = 20$, the linear part of the program is per-

Code	Parallel time (s)
BKLC $[30, 16, 7]$	552
BKLC $[30, 17, 6]$	562
BKLC $[30, 18, 6]$	212
BKLC $[30, 19, 5]$	93
BKLC $[30, 20, 5]$	107
BKLC $[30, 21, 4]$	171
BKLC $[30, 22, 4]$	279
BKLC $[30, 23, 4]$	500
BKLC $[30, 24, 4]$	955
BKLC $[30, 25, 3]$	3631

Table 5.11: Calculation times for BKLCs with code length $n = 30$

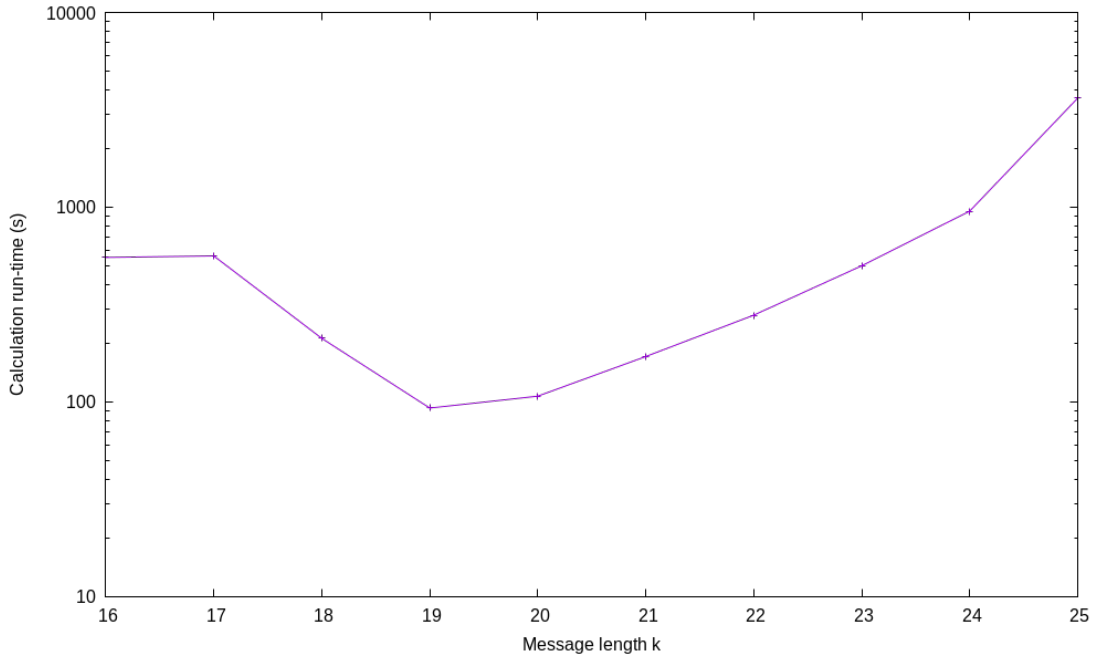


Figure 5.9: Time to calculate equivocation for codes of length $n = 30$ for different message lengths(k)

forming an increasing proportion of the workload. For the program arrangement used, operating with a message length of $k = 19$ to 20 enabled a “sweet spot" of efficiency to be achieved. This also supports the observation made from Table 5.8 that many of the best improvements in calculation time from linear to parallel processing were achieved with message lengths around $k = 20$.

5.4.2 Run-Time Factors

There are, however, many inter-related factors at play that influence the efficiency of the calculation and the time taken for the calculation to run. These include but are not limited to:

- Code length - This is one of the most fundamental factors affecting the length of time that a calculation will take. At its most basic, a single-bit increase in the length of a codeword will approximately double the number of calculations to be performed and the time taken. However the presence of numerous other factors means that this is unlikely to directly bring an exact doubling of the calculation run-time.

- Message length - At very low code lengths (e.g. below $k = 15$), the CPU performs much of the calculation. As the code length increases, an increasing proportion of the workload is able to be taken by the GPU. Once the “hand-off" of tasking by the CPU to the parallel-processing GPU has reached the greatest rate that the GPU and CUDA architecture can manage, each subsequent single-bit increase in message length will approximately double the calculation time if all other factors remain the same.

- Code structure - One of the first tasks of the calculation is to identify those error vectors that yield the first instance of each syndrome. The code structure indicated by the generator matrix will directly affect which error vectors give each syndrome, how long it will take to locate them and thereby, the calculation run-time.

- Available RAM - limitations of the parallel processing performed by the GPU were counter-acted by holding significant amounts of data in memory. For example, in the intiial stage of the calculations, a record of each error vector that gave an initial instance of a syndrome was recorded. This record itself rapidly becomes bounded by memory limitations, both the overal amount of available memory and

the way in which contiguous space is allocated. Increasing the available memory would increase the amount of data held in RAM and reduce the processing load.

- Number of processors - It is reasonable to anticipate that an increase in the number of processors used will bring a proportionate increase in processing capability and a decrease in calculation times. However, as each additional processor will bring time overheads, both through its own internal operation and in it's interactions with other processors, the actual impact on calculation run times is likely to be more complex.

- CPU capability - Since the CPU runs the main, linear component of the program, a more capable CPU (with higher clock speed, bus speed, cache size, number of cores, etc.) would enable the linear calculations to be carried out more quickly.

- Number of GPUs - the software was written to be run on a standard laptop with a single CPU and single GPU. Further work on this topic could extend calculations further by employing a suite of processors with parallel computing capability. Adding further GPUs will decrease calculation run-times with each new set of CUDA enabled cores brought to bear on the calculation.

- GPU capability - the Nvidia GeForce GTX470M GPU used for the calculations had 288 cores, such that 288 threads could be run simultaneously, at a processor clock speed of 1.1GHz. A more capable GPU would have more cores, operating at higher speeds. For example, the GeForce TITAN V has 5120 cores operating at 1.455GHz. Numerous other differences between different CUDA-enabled GPUs will also affect calculation times significantly.

- CUDA compute capability - the GPU used had a compute capability of 3.5, enabling 1024 threads per block, but running subject to the capability of the GPU cores. The GeForce TITAN X has a compute capability of 6.1, giving a range of improvements and advantages such as an increased number of 32-bit registers available per block and an increased amount of shared memory per multiproces-

sor.

- Number of threads per block - Most compute capabilities permit blocks to be created in up to 3-dimensions with up to 1024 threads in x and y dimensions and 64 threads in the z dimension. In the software solution developed, only the x dimension was used. Further development of the software could potentially reduce run-times further through the use of y and z dimensions for running multi-dimensional blocks of threads.

- Number of blocks per warp - At the next level, blocks can be treated in groups of up to 64, called *warps*. Warps were not used in the software solution but do potentially form another mechanism for reducing calculation run-times.

- Programming model: It has been shown that a hybrid programming model, where the core program is linear and repetitive tasks are performed in parallel can offer signficant improvements in efficiency over a purely linear program. A different mechanism for achieving this could come through the multi-core CPUs that many standard PCs now offer. However, it was decided to pursue the CUDA architecture approach because:

  - CUDA-enabled GPUs generally have many more cores than a multi-core CPU, enabling a greater level of parallelisation for the completion of highly repetitive but relatively simple calculations.
  - Use of the CUDA architecture offered a novel approach to performing an equivocation calculation.

- Program efficiency - Finally, one of the greatest factors affecting the calculation run-time was the programming ability of the author, who started the research period with a relatively low level of programming skill. Throughout the course of the PhD, many significant programming lessons were learnt and improvements made. It is inevitable that further significant improvements would be possible, especially in light of the observations just made above.

5.5 Improving Code Equivocation by Expansion

Once a procedure for efficiently calculating the equivocation of a code has been implemented, it encourages the comparison of codes and the location and design of codes with improved levels of equivocation. As an example, consider a modification of the simple Hamming $[7,4,3]$ code. A single bit of data could be replaced by a sequence of data bits, say 4 bits long. The first three bits are randomly generated whilst the fourth bit is chosen to give an overall message parity equivalent to the data that is to be transmitted. So the data bit 0 could be represented as (0011), each component of which is transmitted over the course of 4 successive messages:

$$(\overbrace{001}^{RandomBits}\ \overbrace{1}^{ParityBit})$$

Where previously the probability of an error occurring while a single bit was transmitted might have been quite low, for example $p_e = 0.01$, now that the representation of the data bit 0 is transmitted across more messages, the probability of an error becomes compounded and potentially much increased. If an f-fold expansion of a code is taken to be one in which a data bit is represented by $f-1$ random bits and 1 parity check bit, then the received data bit would be expressed in terms of the parity of the received data bits:

$$Parity = \sum_{i=1}^{f} r_i \tag{5.10}$$

where r_i is the i'th received bit of the code expansion. The received data is the sum of the transmitted data and any associated errors that occur during transmission:

$$r_i = t_i + v_i \tag{5.11}$$

where t_i is the i'th bit of the expansion and v_i is i'th bit of the associated error vector. Therefore:

$$Parity = \sum_{i=1}^{f} t_i + \sum_{i=1}^{f} v_i \tag{5.12}$$

But by design, $\sum_{i=1}^{f} t_i = 0$ and therefore $Parity = \sum_{i=1}^{f} v_i$, so if the received message parity is non-zero, then a decoder error must have occurred.

When considering a 4-fold code expansion, the possible error combinations that could occur range from $\begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix}$ to $\begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}$. Given that we only need to consider the combinations that give an odd parity, then the probability of the 4-fold expansion being decoded erroneously is:

$$\binom{4}{1}(1-p_e)^3 p_e + \binom{4}{3}(1-p_e)p_e^3$$

In general for an f-fold expansion, the probability of an incorrect decoding would be:

$$\sum_{i=0}^{i\leq f/2} \binom{f}{2i+1}(1-p_e)^{f-(2i+1)} p_e^{2i+1}$$

A 4-fold expansion of the Hamming code gives effective channel probability errors as shown in Table 5.12.

Single bit probability of error	4-fold expansion probability of error
0	0
0.01	0.038816
0.05	0.17195
0.10	0.2952
0.15	0.37995
0.20	0.4352
0.25	0.46875
0.30	0.4872
0.35	0.49595
0.40	0.4992
0.45	0.49995
0.50	0.5

Table 5.12: Compounded channel probability errors for a 4-fold code expansion

The effective channel probabilities for 2-, 4-, and 10-fold code expansion are shown in Figure 5.10.

These effective channel probability errors yield normalised equivocation values for a 4-fold expansion of the Hamming $[7,4,3]$ code shown in Table 5.13 and plotted in Fig-

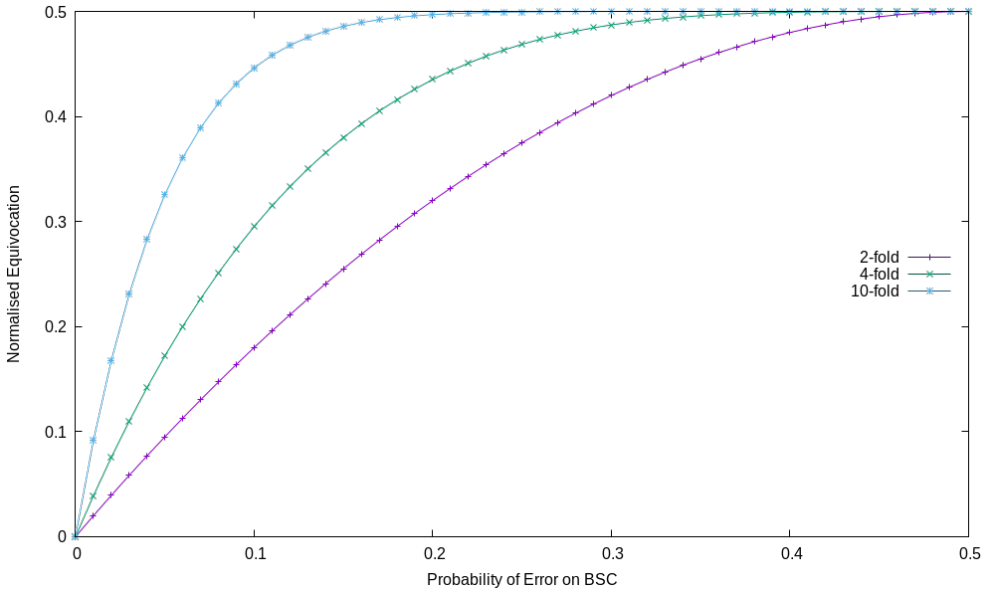


Figure 5.10: Effective Channel Probability Errors 2x, 4x and 10x code expansion

ure 5.11. As previously, normalised equivocation values were calculated by dividing the equivocation by the original message length. The graph also includes 2-, 3- and 10-fold expansions of the Hamming $[7,4,3]$ code. The values for each expansion were calculated in a similar manner to those of the 4-fold expansion.The figure shows that the equivocation values of the code expansions are significantly higher than those of a simple Hamming code.

Single bit probability of error	Normalised Equivocation
0	0
0.01	0.067291
0.05	0.53895
0.1	0.85093
0.15	0.95522
0.2	0.98774
0.25	0.99721
0.3	0.99954
0.35	0.99995
0.4	0.99998
0.45	1
0.5	1

Table 5.13: Normalised equivocation for a 4-fold expansion of the Hamming $[7,4,3]$ code

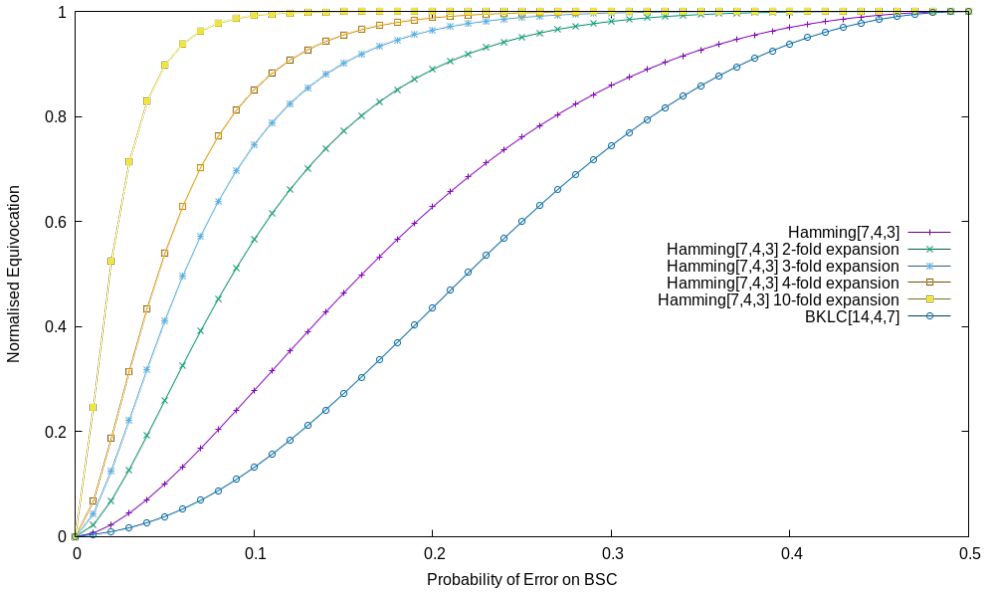


Figure 5.11: Comparison of BKLC $[14,4,7]$ and Hamming code with 2x, 3x, 4x and 10x code expansion

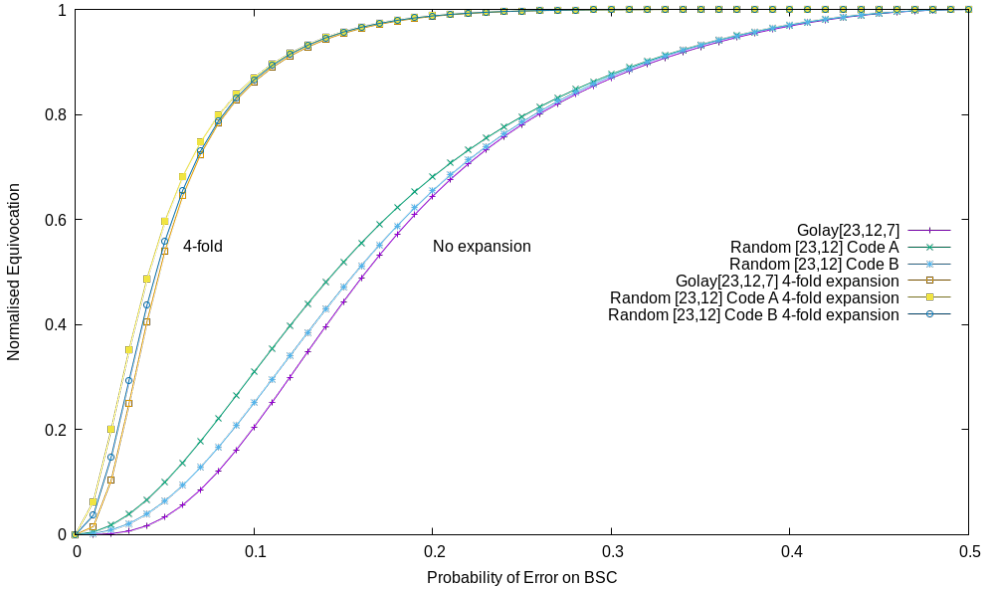


Figure 5.12: Comparison of Golay $[23,12,7]$ and 2 random $[23,12]$ codes with 4-fold expansion

For the 10-fold expansion, a 0.01 probability of transmission error yields a normalised equivocation level of 0.23. However an error probability of just 0.05 now yields a normalised equivocation value of 0.897, compared to 0.101 for the unexpanded Hamming code. If the intended recipient has a channel probability error of 0.01, they still have a

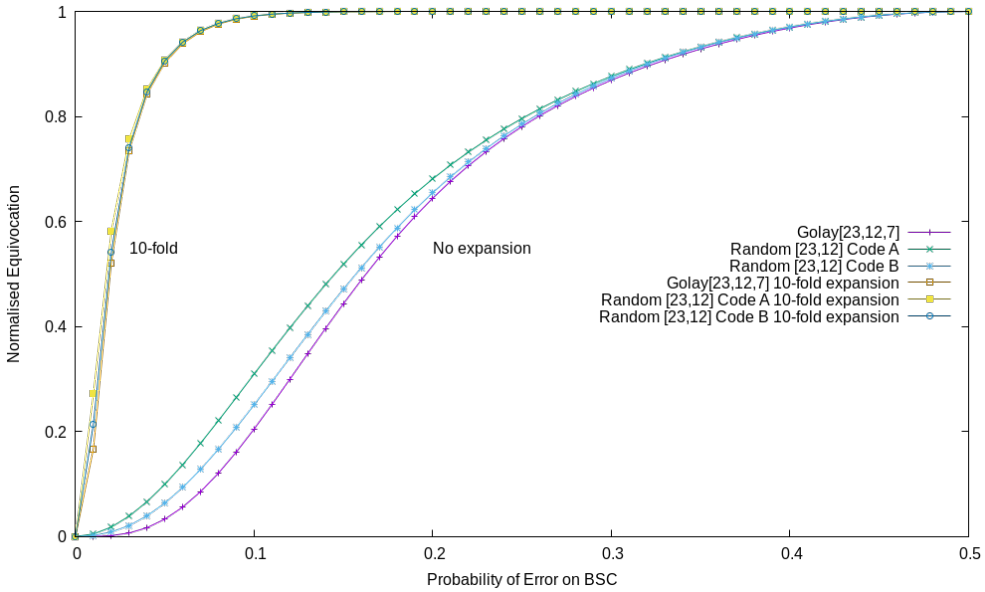


Figure 5.13: Comparison of Golay $[23,12,7]$ and 2 random $[23,12]$ codes with 10-fold expansion

good chance of recovering the data, whereas an illegitimate eavesdropper with a channel error probability of 0.05 will now need to overcome a much higher level of ambiguity in order to recover the data. The secrecy of the data transmission has been very significantly improved by the expansion of a simple Hamming code. However this improved secrapacity ecy comes at a cost; only one bit of data is transmitted for every 4 or 10 bits of data carried by the Hamming code, reducing the net data rate significantly.

It is worth noting that while the Hamming $[7,4,3]$ code with 2-fold expansion and BKLC $[14,4,7]$ code both effectively take 14 bits to transmit 4 bits worth of data, their equivocation graphs are very different. The 2-fold expansion of the Hamming $[7,4,3]$ code offers a significant improvement in equivocation over the basic Hamming $[7,4,3]$ code, whereas the BKLC $[14,4,7]$ code shows a significant worsening in performance. For a BSC error probability of $p_e = 0.05$, the 2-fold expanded Hamming code gives a normalised equivocation of 0.259, while the non-expanded Hamming code gave a value of 0.101 and BKLC$[14,4,7]$ only gives a value of 0.038. This suggests that simply using longer and longer codes may not be as effective at improving equivocation as using code expansions.

Additionally, the 4-fold and 10-fold expansions for the Golay $[23, 12]$ code and the 2 random $[23, 12]$ codes previously discussed are shown in Figure 5.12 and Figure 5.13 respectively. Code expansion of a random code can give a higher normalised equivocation value that for a non-expanded but more structured code. For example, with a BSC error probability $p_e = 0.05$, a 4-fold expansion of the random $[23, 12]$ code A gives a normalised equivocation of 0.597, compared to a non-expanded Golay$[23, 12, 7]$ value of 0.033. As with the Hamming codes, these figures show that there is a very marked increase in normalised equivocation levels when code expansion is used compared to the original codes.

5.6 Chapter Conclusions

This chapter has developed several points with regard to both the use of parallel processing for calculating equivocation and the comparison of equivocation values for different codes and their variants.

- In subsection 5.3.2, it was seen that parallel processing can be effectively implemented on a general purpose computer using the Nvidia CUDA architecture. Developing the code to implement the parallel processed component of the program was both time consuming and challenging but enabled the generation of results for longer codes.
- Parallel processing can provide significant improvements in calculation times for intensive calculations. The best improvements obtained gave calculation results up to 35 times more quickly with parallel processing than with linear processing.
- Parallel processing code must be carefully implemented in order to optimise its efficiency for the actual task. In particular careful memory management is necessary to enable the GPU to continue operate its core duties effectively whilst simultaneously carrying out the parallel processed component of the calculation.
- Section 5.4 shows that normalised equivocation values can be used to compare the relative secrecies of codes, enabling codes with higher inherent levels of se-

crecy to be selected if required, although the other properties of the codes, such as the error correcting capability must also be borne in mind. For example, from Figure 5.3, when operating with a BSC error probability of 0.05, Golay$[23,12,7]$ has a normalised equivocation of 0.033, whereas Hamming$[31,26,3]$ has a value of 0.221, nearly 7 times higher. The Hamming code offers a higher level of secrecy for that error probability than the Golay$[23,12,7]$ code but is only able to correct 1 error rather than 3.

- For the codes examined from the Hamming code family, longer codes had higher normalised equivocation values than the shorter codes. For example, with $p_e = 0.05$, the normalised equivocation of Hamming$[7,4,3]$ is 0.101 compared to 0.157 for Hamming$[15,11,3]$ and 0.221 for Hamming$[31,26,3]$. However, the complexity of the calculations to find equivocation values for longer codes meant that it wasn't possible to verify this for longer Hamming codes such as Hamming$[63,57,3]$ or Hamming$[127,120,3]$. Further work is required to establish if the pattern holds for either longer Hamming codes or for other families of code.

- Largest differences in code equivocation values were often found in the more central range of the error probabilities for the codes investigated (especially for expanded codes). This can be seen by identifying the steepest part of the curve in any of the normalised equivocation graphs from Figure 5.3 to Figure 5.13.

- For the codes examined, the differences in normalised equivocation values between codes decrease as code lengths increase. For example, in Figure 5.6 with longer length BKLCs, the similarity in value of all 7 functions can be seen quite clearly, whereas the differences between the shorter codes in Figure 5.3 are significantly more marked. This may be partly due to structural similarities that must exist in order to yield the best linear codes for each code length. As the code lengths increase, it may be that the relative differences between the codes that give the greatest minimum distance for each code length decrease, however this was not formally confirmed.

- In Section 5.5, it was shown that by expanding a simple code through the use of random data and parity check bits, much greater levels of secrecy can be achieved in some cases. For example, with a BSC error probability $p_e = 0.05$, a 10-fold expansion of the Hamming$[7,4,3]$ code gives a normalised equivocation level of 0.897, compared to 0.101 for the non-expanded code. However, there is a trade-off between achieving the desired level of secrecy by expanding a simple code and the significant increase in the number of bits of data that need to be transmitted in order to achieve it. A 10-fold code expansion of the Hamming$[7,4,3]$ code may enable very high levels of equivocation but would require the transmission of 70 bits of data to send a 4-bit message.

- Rather than designing codes with improved error-correcting capabilities, one of the aims of the work was to identify codes that either a) give a higher equivocation value and level of secrecy than other similar or related codes or b) enable a large differential in equivocation level to be established between the legitimate recipient and the eavesdropper. It was possible to use random codes to achieve higher equivocation values than for more structured but non-expanded codes. For example, with a BSC error probability $p_e = 0.05$, a 4-fold expansion of a random $[23,12]$ code gives a normalised equivocation of 0.597, compared to a non-expanded Golay$[23,12,7]$ value of 0.033. However, although the inherent randomness of the codes increased equivocation levels, this is of little benefit given that the codes offer no error correcting capability.

- The expansion of a code can offer higher normalised equivocation values than a code with a similar net code rate. For example, consider the Hamming$[7,4,3]$ code with 2-fold expansion and the BKLC$[14,4,7]$ code. Both require 14 bits to transmit 4 bits worth of message data, however for $p_e = 0.05$, the 2-fold expanded Hamming code gives a normalised equivocation of 0.259, while the non-expanded Hamming code gave a value of 0.101 and the BKLC$[14,4,7]$ only gives a value of 0.038. Code expansion has a compounding influence on equivocation.

Chapter 6

Equivocation of Erasures

6.1 Introduction

The key outcomes of Chapter 6 were published by Schofield et al. (2016).

Having established a parallel processing method to calculate the equivocation of a code transmitted via a BSC, this approach can be extended to examine the impact of erasures and deletions. In this chapter, two different situations involving erasures are considered:

- Equivocation of codes, including intentional erasures on a BSC
- Equivocation of codes on the Binary Erasure Channel (BEC)

6.2 Equivocation of a BSC with Intentional Erasures (IE+BSC)

It has already been seen that if transmitting across a BSC while an eavesdropper listens via a Wire-Tap Channel, the difference in signal quality can be exploited to improve the inherent transmission secrecy. However if this exploitation involves the 'managed' use of erasures, then the secrecy of the code transmitted across the channel can be improved whilst simultaneously reducing the volume of data that is transmitted. This is in contrast to the increased volume of data incurred by the code expansion methods used in Chapter 5.

In his use of punctured LDPC codes in order to modify the security gap, Klinc et al. (2011) acknowledged that it can be difficult to measure or analyse equivocation. Work has also been done to establish upper and lower bounds for equivocation, including

the use of known and unknown puncturing patterns Almeida (2013). Other techniques have been applied to specific circumstances to calculate equivocation, such as the analysis of the generator matrix of the eavesdropper's code (Wickramasooriya et al. 2013) and the use of probability mass functions for syndrome coding (Al-Hassan et al. 2014). However, calculations such as Al-Hassan's remain computationally demanding, requiring the generation of many error patterns and syndromes.

This section looks at the deliberate introduction of erasures whilst transmitting data across a BSC (IE+BSC) within a Wire-Tap environment. The software solution used in Chapter 5 has been enhanced to permit the calculation of normalised equivocation values for the IE+BSC.

Since an error-correcting code can correct at most $\frac{d-1}{2}$ errors or $d-1$ erasures (Hoffman 1991), in this work, the number of erasures deliberately introduced will always be constrained to be less than or equal to the correction capability of the code.

If deliberate erasures are introduced, then the value of a bit in a specified location is unknown as shown in Figure 6.1 and must be treated as random.

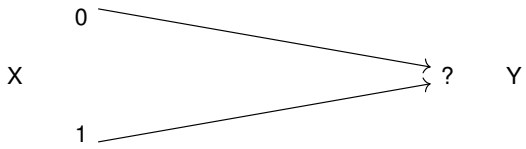


Figure 6.1: Deliberate Erasures

As in Chapter 5, the processes of correcting the received codeword and decoding it are considered separately, as shown in Figure 6.2. The message m is encoded as the codeword x before bits are intentionally erased to give the transmission vector t. Transmission across the BSC and potential introduction of a BSC error v yields a received vector r. This vector r is then corrected by the addition of a vector c to give the corrected, received codeword y. Finally y is decoded to give the message estimate m_{est}.

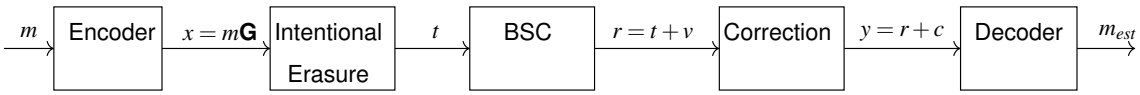


Figure 6.2: IE+BSC encoding / decoding process

6.2.1 Calculations

A message of length k is encoded as a codeword of length n. If a bits, in known locations (for example the right-hand a bits of the codeword), are deliberately erased ahead of transmission, then $n-a$ bits of known value and a erased bits of unknown value (but known location) would be received. The data is transmitted across a BSC and therefore the known bits will still have been susceptible to a cross-over error of probability p_e.

For an $[n,k,d]$ code containing a erasures in known locations, it is necessary to consider all possible values that the erased bits could have held and also all possible BSC errors in non-erased locations. Whilst this adds complexity to the scenario compared to that for calculating the equivocation for codes on the BSC without erasures, it is significantly less complex than when considering deletions where the erasure locations are unknown. This aspect will form the fundamental difference between the approaches for erasures and deletions.

There would be 2^{n-a} possible error vectors that could affect the $n-a$ received bits. For each of those possible error vectors, it would be necessary to consider 2^a possibilities of what the erased bits could have been. If the probability of a symmetric crossover error on the BSC is p_e, the probability of there being ε errors in a received codeword that is known to contain a erasures in known positions is thus:

$$P(\varepsilon,a) = \binom{n-a}{\varepsilon}(1-p_e)^{(n-a)-\varepsilon}p_e^{\varepsilon} \tag{6.1}$$

Therefore if a BKLC $[8,2,5]$ code was transmitted with two bits erased in positions 7 and

8, $\begin{pmatrix} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & ?_7 & ?_8 \end{pmatrix}$, the probability of receiving a codeword of length 8 bits containing a single error with a BSC error probability $p_e = 0.01$ is:

$$P(1,2) = \binom{6}{1} \times 0.99^5 \times 0.01 = 0.0571$$

When calculating the equivocation of the code across the channel, the probability and thence the conditional entropy of each possible output, given each possible input, must be found. Any erased bits could have been either a 0 or a 1 with equal probability, so the received codeword could have been any one of $\frac{1}{2^a}$ possible options before the a bits were erased.

Thus the probability of a received message having contained ε errors in specific locations and the a erasures having come from a particular combination of 0's and 1's will be as shown in Equation 6.2:

$$P(\varepsilon, a) = (1 - p_e)^{(n-a)-\varepsilon} p_e^{\varepsilon} \times \frac{1}{2^a} \tag{6.2}$$

So the probability of receiving an 8-bit codeword on a BSC with error probability $p_e = 0.01$, with errors in positions 1 and 4 with 2 erased bits in positions 7 and 8 (i.e. the vector $\begin{pmatrix} 0_1 & 1_2 & 0_3 & 0_4 & 1_5 & 0_6 & ?_7 & ?_8 \end{pmatrix}$ will be:

$$0.99^4 \times 0.01^2 \times \frac{1}{2^2} = 2.401 \times 10^{-5}$$

Equivocation calculations allowing for erasures were initially performed by the completion of a spreadsheet of calculations for a short code (in this case, the BKLC $[8,2,5]$ code) before being incorporated into the existing C++ program used previously for calculating the code equivocation.

The BKLC $[8,2,5]$ code can have Generator Matrix:

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

and Parity Check Matrix:

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

The 4 possible messages $\begin{pmatrix} 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 \end{pmatrix}$ and $\begin{pmatrix} 1 & 1 \end{pmatrix}$ yield valid codewords of:

$$\begin{matrix} (0 & 0 & 0 & 0 & 0 & 0 & 0 & 0) \\ (0 & 0 & 0 & 1 & 1 & 1 & 1 & 1) \\ (1 & 1 & 1 & 0 & 0 & 1 & 1 & 1) \\ (1 & 1 & 1 & 1 & 1 & 0 & 0 & 0). \end{matrix}$$

The first section of the spreadsheet, showing the calculation of probabilities for each possible error vector and erasure combination for the zero codeword $\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$, with known erasures in positions 7 and 8 and $p_e = 0.01$, is shown in column two of Table 6.1 with possible erasure combinations shown in grey. The error vectors are listed in weight order, containing ε errors where $0 \leq \varepsilon \leq n-a$.

Once the probabilities have been calculated for each possible combination of BSC error and erased bits, these probabilities can be used to find the equivocation.

This is achieved by:

1. Multiplying the received codeword by the parity check matrix to obtain the syndrome.

2. Using the syndrome to correct the received codeword to the most likely valid codeword (columns 3 to 5 of Table 6.1).

3. Summing probabilities to find the total probability of each codeword y_j being received, given a particular message x_i as input. This gives $P(y_j|x_i)$.

4. $P(x_i,y_j) = P(x_i)P(y_j|x_i)$

5. $H(x_i,y_j) = P(x_i,y_j)\log_2 \frac{1}{P(x_i,y_j)}$

6. $H(x_i,Y) = \sum_{y\in Y} H(x_i,y_j) = \sum_{y\in Y} P(x_i,y_j)\log_2 \frac{1}{P(x_i,y_j)})$

7. $H(X,Y) = \sum_{x\in X} H(x_i,Y) = \sum_{x\in X}\sum_{y\in Y} P(x_i,y_j)\log_2 \frac{1}{P(x_i,y_j)})$

The probabilities for the output codeword $y_0 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$ given the input codeword $x_0 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$ are shown in bold in Table 6.1. The probability sums for each codeword, $x_0,...,x_3$ can be seen in Table 6.2.

Table 6.2 shows that the contribution to the joint equivocation from one codeword is 0.50197. The contribution to the joint entropy from each input codeword will be the same and therefore the joint entropy $H(X,Y) = 0.50197 \times 4 = 2.007886$.

Figure 6.3 shows the conditional probabilities for BKLC $[8,2,5]$ with 2 erasures and $p_e = 0.01$. The output probabilities can be found from these, as shown in Table 6.3. This demonstrates that if the input codewords are equally likely then so too are the output codewords i.e. $P(y_j) = 0.25$.

If $P(y_i) = 0.25$, then

$$H(Y) = \sum_{y\in Y} P(y_j)\log(\frac{1}{P(y_j)}) = 4\times 0.25\log_2(\frac{1}{0.25}) = 4\times 0.25\times 2 = 2$$

Therefore when $p_e = 0.01$,

$$H(X|Y) = H(X,Y) - H(Y) = 2.007886 - 2 = 7.886\times 10^{-3}$$

Codeword: 0 0 0 0 0 0 0 0				
Received Codeword +erased bits	$P(\varepsilon, a)$	Syndrome	Correction	Corrected codeword
0 0 0 0 0 0 ? ?				
0 0 0 0 0 0 0 0	**0.23537**	0 0 0 0 0 0	0 0 0 0 0 0 0 0	**0 0 0 0 0 0 0 0**
0 0 0 0 0 0 0 1	**0.23537**	1 1 1 0 1 1	0 0 0 0 0 0 0 1	**0 0 0 0 0 0 0 0**
0 0 0 0 0 0 1 0	**0.23537**	0 0 0 0 0 1	0 0 0 0 0 0 1 0	**0 0 0 0 0 0 0 0**
0 0 0 0 0 0 1 1	**0.23537**	1 1 1 0 1 0	0 0 0 0 0 0 1 1	**0 0 0 0 0 0 0 0**
	$P(0,2) = 0.94148$			
0 0 0 0 0 1 ? ?				
0 0 0 0 0 1 0 0	**0.002377**	0 0 0 0 1 0	0 0 0 0 0 1 0 0	**0 0 0 0 0 0 0 0**
0 0 0 0 0 1 0 1	**0.002377**	1 1 1 0 0 1	0 0 0 0 0 1 0 1	**0 0 0 0 0 0 0 0**
0 0 0 0 0 1 1 0	**0.002377**	0 0 0 0 1 1	0 0 0 0 0 1 1 0	**0 0 0 0 0 0 0 0**
0 0 0 0 0 1 1 1	**0.002377**	1 1 1 0 0 0	0 0 0 0 0 1 1 1	**0 0 0 0 0 0 0 0**
⋮		⋮	⋮	⋮
1 0 0 0 0 0 ? ?				
1 0 0 0 0 0 0 0	**0.002377**	1 0 0 0 0 0	1 0 0 0 0 0 0 0	**0 0 0 0 0 0 0 0**
1 0 0 0 0 0 0 1	**0.002377**	0 1 1 0 1 1	1 0 0 0 0 0 0 1	**0 0 0 0 0 0 0 0**
1 0 0 0 0 0 1 0	**0.002377**	1 0 0 0 0 1	1 0 0 0 0 0 1 0	**0 0 0 0 0 0 0 0**
1 0 0 0 0 0 1 1	**0.002377**	0 1 1 0 1 0	1 0 0 0 0 0 1 1	**0 0 0 0 0 0 0 0**
	$P(1,2) = 0.05706$			
⋮		⋮	⋮	⋮
0 0 0 0 1 1 ? ?				
0 0 0 0 1 1 0 0	2.4×10^{-5}	1 1 1 1 1 0	0 0 0 1 0 0 1 1	0 0 0 1 1 1 1 1
0 0 0 0 1 1 0 1	2.4×10^{-5}	0 0 0 1 0 1	0 0 0 1 0 0 1 0	0 0 0 1 1 1 1 1
0 0 0 0 1 1 1 0	2.4×10^{-5}	1 1 0 0 0 1	0 0 0 1 0 0 0 1	0 0 0 1 1 1 1 1
0 0 0 0 1 1 1 1	2.4×10^{-5}	0 0 0 1 0 0	0 0 0 1 0 0 0 0	0 0 0 1 1 1 1 1
⋮		⋮	⋮	⋮
1 1 0 0 0 0 ? ?				
1 1 0 0 0 0 0 0	2.4×10^{-5}	1 1 0 0 0 0	0 0 1 0 0 1 1 1	1 1 1 0 0 1 1 1
1 1 0 0 0 0 0 1	2.4×10^{-5}	0 0 1 0 1 1	0 0 1 0 0 1 1 0	1 1 1 0 0 1 1 1
1 1 0 0 0 0 1 0	2.4×10^{-5}	1 1 0 0 0 1	0 0 1 0 0 1 0 1	1 1 1 0 0 1 1 1
1 1 0 0 0 0 1 1	2.4×10^{-5}	0 0 1 0 1 0	0 0 1 0 0 1 0 0	1 1 1 0 0 1 1 1
	$P(2,2) = 1.441 \times 10^{-3}$			
⋮		⋮	⋮	⋮
1 1 1 1 1 1 ? ?				
1 1 1 1 1 1 0 0	2.5×10^{-13}	0 0 0 0 1 0	0 0 0 0 0 1 0 0	1 1 1 1 1 0 0 0
1 1 1 1 1 1 0 1	2.5×10^{-13}	1 1 1 0 1 0	0 0 0 0 0 1 0 1	1 1 1 1 1 0 0 0
1 1 1 1 1 1 1 0	2.5×10^{-13}	0 0 0 0 1 1	0 0 0 0 0 1 1 0	1 1 1 1 1 0 0 0
1 1 1 1 1 1 1 1	2.5×10^{-13}	1 1 1 0 0 0	0 0 0 1 1 0 0 0	1 1 1 0 0 1 1 1
	$P(6,2) = 1 \times 10^{-12}$			

Table 6.1: Error and erasure correction process

Since the BKLC $[8,2,5]$ code encodes 2 bit messages as a codeword of length 8, the

Output Codeword	$P(y_j\|x_0)$	$P(x_0,y_j)$	$H(x_0,y_j)$
0 0 0 0 0 0 0 0	0.99940409	0.24985102	0.49991691
0 0 0 1 1 1 1 1	0.00029694	7.4235E-05	0.00101832
1 1 1 0 0 1 1 1	0.00029213	7.3032E-5	0.00100354
1 1 1 1 1 0 0 0	6.8414E-06	1.1703E-06	3.2766E-05
TOTAL	1		0.50197154

Table 6.2: Calculation of joint entropy from probabilities for codeword $x_0 = 00000000$

Output codeword	$P(y_j)$	
00000000	$(0.99940+0.000297+0.00029+6.8414E-06)\times 0.25$	= 0.25
00011111	$(0.000297+0.99940+6.8414E-06+0.00029)\times 0.25$	= 0.25
11100111	$(0.00029+6.8414E-06+0.99940+0.000297)\times 0.25$	= 0.25
11111000	$(6.8414E-06+0.00029+0.000297+0.99940)\times 0.25$	= 0.25

Table 6.3: Combined output probabilities and entropies BKLC $[8,2,5]$ with p_e = 0.01

normalised equivocation will be

$$7.886\times 10^{-3}\div 2 = 3.943\times 10^{-3}$$

In all results for codes of differing lengths, the equivocation values stated have been normalised to between 0 and 1, depending on the message length of the code, to give the equivocation per data bit.

6.2.2 Results

As an imperfect code, the BKLC $[8,2,5]$ is only capable of correcting at most 4 erasures $(d-1)$, where d is the minimum distance of the code. Alternatively it can correct at most 2 errors or other combinations of errors and erasures, subject to the constraint that $2\varepsilon + a < d$ (Sweeney 2002, p.164). Hence the BKLC $[8,2,5]$ code can correct the combinations of errors and erasures in Table 6.4

Errors	Erasures
0	0, 1, 2, 3, 4
1	0, 1, 2
2	0

Table 6.4: Correctable Combinations of Errors and Erasures for the BKLC $[8,2,5]$ Code

$x_0 = 00000000$, $P(x_0) = 0.25$	$\boldsymbol{P(y_0\|x_0) = 0.999404089}$
	$P(y_1\|x_0) = 0.000296941$
	$P(y_2\|x_0) = 0.000292129$
	$P(y_3\|x_0) = 6.8414E-06$
$x_1 = 00011111$, $P(x_1) = 0.25$	$\boldsymbol{P(y_0\|x_1) = 0.000296941}$
	$P(y_1\|x_1) = 0.999404089$
	$P(y_2\|x_1) = 6.8414E-06$
	$P(y_3\|x_1) = 0.000292129$
$x_2 = 11100111$, $P(x_2) = 0.25$	$\boldsymbol{P(y_0\|x_2) = 0.000292129}$
	$P(y_1\|x_2) = 6.8414E-06$
	$P(y_2\|x_2) = 0.999404089$
	$P(y_3\|x_2) = 0.000296941$
$x_3 = 11111000$, $P(x_3) = 0.25$	$\boldsymbol{P(y_0\|x_3) = 6.8414E-06}$
	$P(y_1\|x_3) = 0.000292129$
	$P(y_2\|x_3) = 0.000296941$
	$P(y_3\|x_3) = 0.999404089$

Figure 6.3: Input and output probabilities for BKLC $[8,2,5]$ on IE+BSC with 2 erasures and $p_e = 0.8$

For the Golay $[23,12,7]$, this increases as shown in Table 6.5.

For the BKLC $[8,2,5]$ shown in Figure 6.4, all curves are monotonic (i.e. always increas-

Errors	Erasures
0	0, 1, 2, 3, 4, 5, 6
1	0, 1, 2, 3, 4
2	0, 2, 4
3	0

Table 6.5: Correctable Combinations of Errors and Erasures for the Golay $[23, 12, 7]$ Code

ing), therefore increasing the number of deliberately erased bits that are transmitted increases the equivocation of the code. One important observation is that the curves for both 3 and 4 erasures of the BKLC $[8, 2, 5]$ code are co-linear i.e. transmitting the code with either 3 or 4 erasures produces the same values of normalised equivocation. This is because the increase from 3 to 4 erasures does not change the ability of the code to use the 2^6 available syndrome patterns for the detection or correction of either errors or erasures.

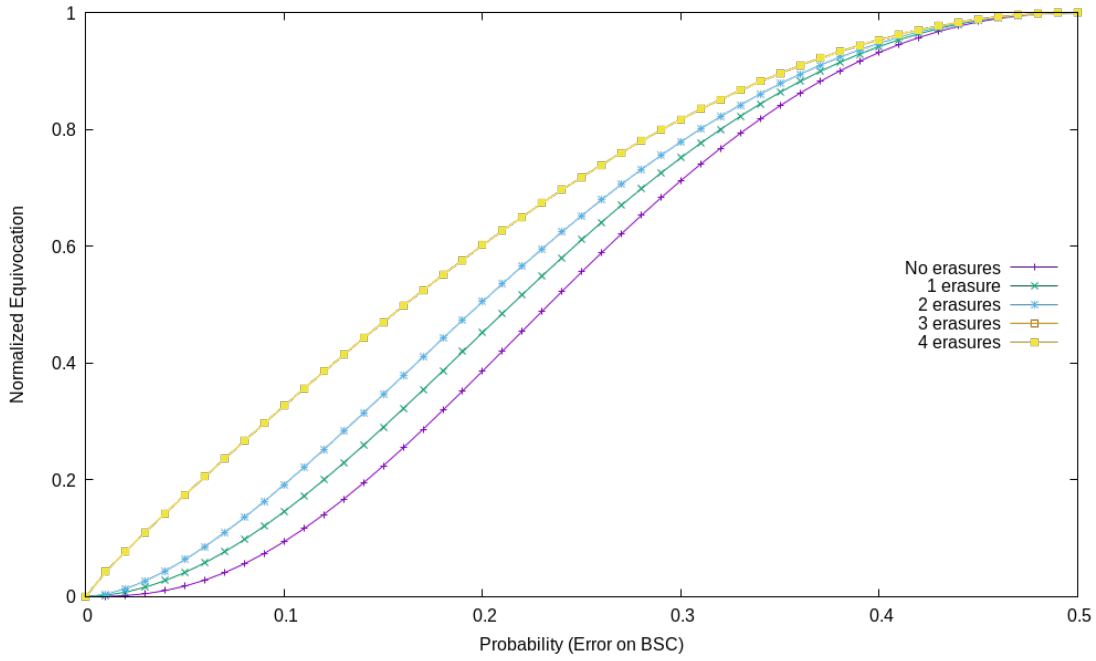


Figure 6.4: Equivocation of BKLC $[8, 2, 5]$ code with up to 4 erasures

Similarly, for the Golay $[23, 12, 7]$ code in Figure 6.5 with the ability to correct up to 6 erasures, 6 monotonic lines are obtained, with the equivocation increasing with each additional erasure.

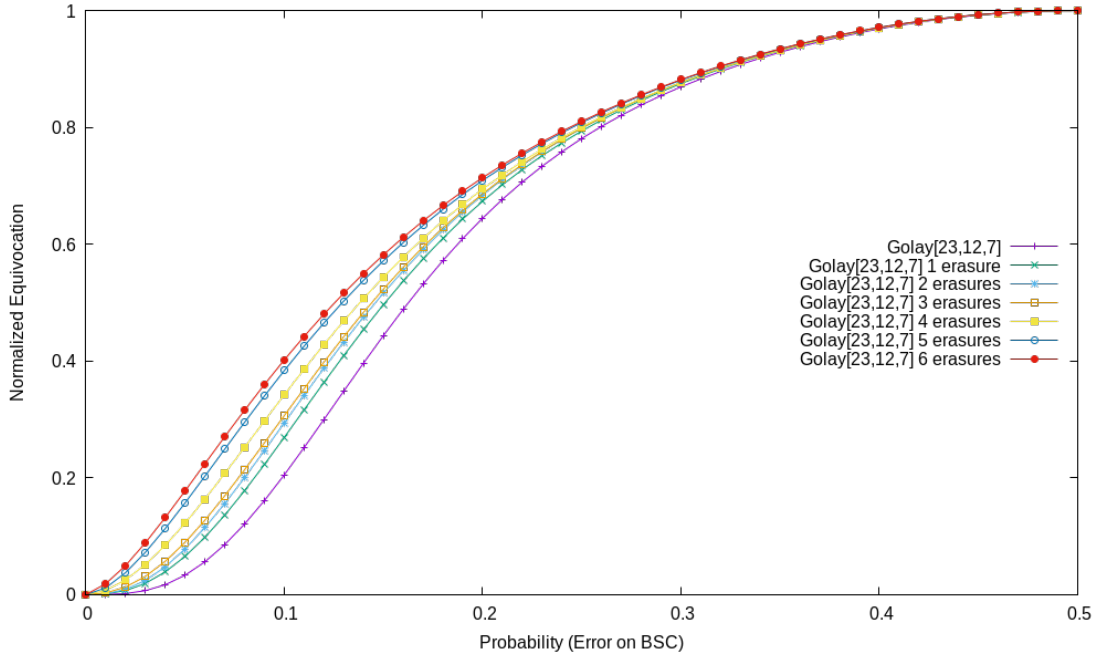


Figure 6.5: Equivocation of Golay $[23, 12, 7]$ code with erasures

When code expansion is brought in to the arrangement as well, as discussed in Section 5.5, the 4 distinct groups of curves on each of Figure 6.6 and Figure 6.7 reflect the different code expansions (none, 2-fold, 4-fold and 10-fold). It is worth noting that each graph shows multiple crossover points when comparing equivocation curves for scenarios with different numbers of erasures and different levels of code expansion.

For example in Figure 6.6, when comparing BKLC $[8, 2, 5]$ with 4 erasures and no code expansion and the same code with no erasures and 2-fold code expansion, there is a crossover point between 0.1 and 0.11. When $p_e \leq 0.1$, the scenarios with 4 erasures and no code expansion gives higher equivocation values. Conversely if $p_e \geq 0.11$, no erasures and 2-fold code expansion give higher equivocation values. Therefore it can be seen that one arrangement gives higher equivocation at lower error probabilities, while the other may be more useful at higher error probabilities. Similarly, comparisons can be drawn between the differing gradients of the curves in different parts of the graph. Between approximately 0.04 and 0.25, 4 erasures and no code expansion gives a steeper graph than no erasures and 2-fold code expansion. Thus it may be pos-

sible to achieve a greater differential between the equivocation level for the intended recipient and the illegitimate eavesdropper. If the likely error probability of the channel can be established, estimated or measured then a choice could be made about which arrangement should be selected in order to maximise the required characteristics. Similar comparisons can be made and implications drawn for any of the crossover points in either Figure 6.6 or Figure 6.7.

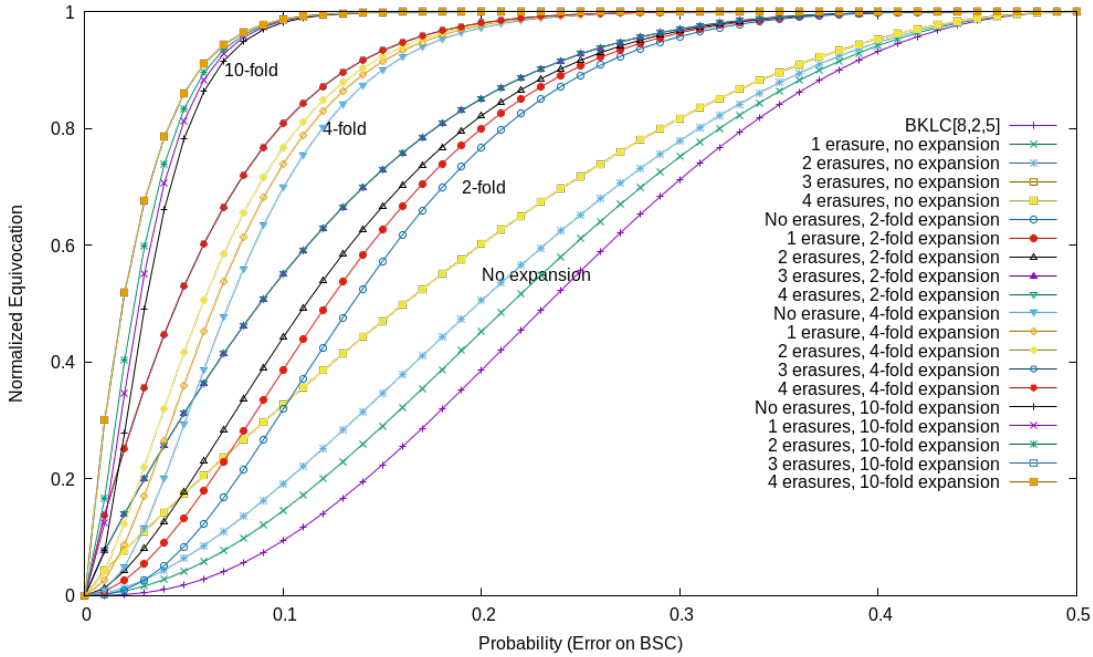


Figure 6.6: Equivocation of BKLC $[8,2,5]$ code with erasures and code expansion

Figure 6.8 highlights some differences between the use of code expansion and the use of erasures, in this case for the Golay $[23,12,7]$ code. For all error probabilities above $p_e = 0.05$, 2-fold code expansion offers greater increases in equivocation than the deliberate introduction of erasures. However below $p_e = 0.05$, the use of multiple erasures gives a greater increase in equivocation. Therefore at close to $p_e = 0.05$ for the Golay code, there is a changeover point above which a 2-fold code expansion gives higher equivocation levels and below which the use of 6 erasures gives higher equivocation. A similar changeover occurs for the BKLC $[31,21,5]$ in Figure 6.9 although the changeover occurs at p_e between 0.01 and 0.02 and is less obvious. Either modification to the code (code expansion or intentional erasures) carries a penalty - the use of

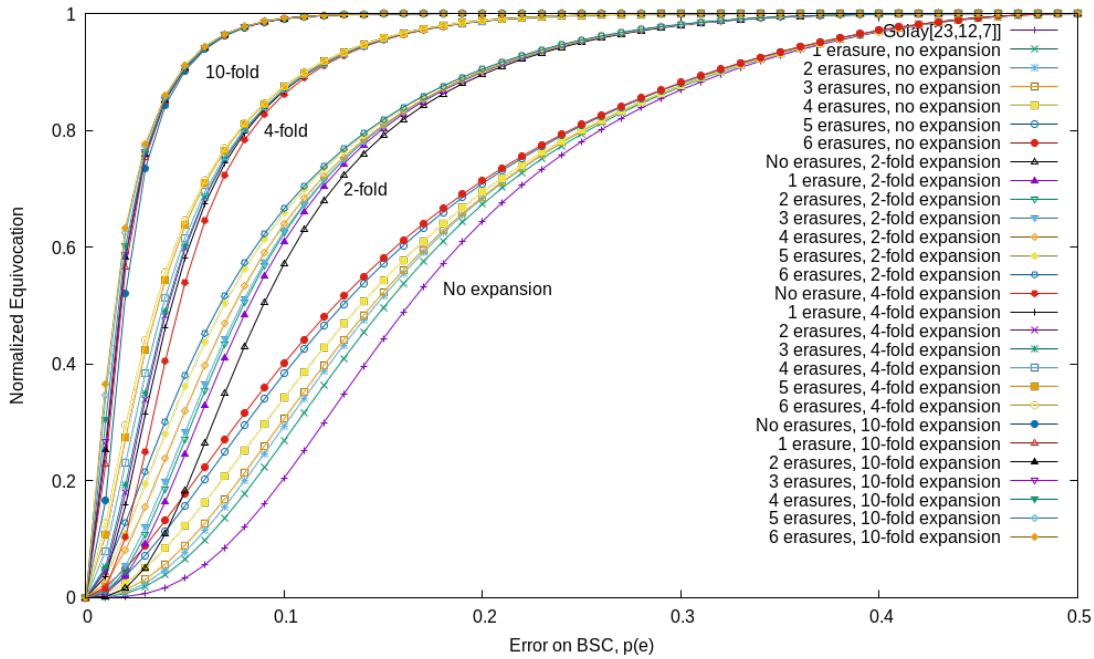


Figure 6.7: Equivocation of Golay $[23, 12, 7]$ code with erasures and code expansion

code expansion increases the number of transmitted bits dramatically whereas the deliberate introduction of erasures decreases the error detecting and correcting capability of the code.

The deliberate introduction of erasures to a transmission system can, if chosen carefully, lead to a greater increase in equivocation for an illegitimate eavesdropper than for the legitimate receiver. For example with the Golay $[23, 12, 7]$ code, a BSC error probability p_e of 0.01 for the legitimate receiver and 0.10 for the eavesdropper will produce the normalised equivocation levels shown in Table 6.6 for scenarios with no erasures and with 2 erasures.

	Legitimate receiver	Eavesdropper
BSC error probability	0.01	0.10
Equivocation (with no erasures)	0.00015	0.20496
Equivocation (with 2 erasures)	0.00147	0.29394
Increase in equivocation	0.00132	0.08898

Table 6.6: Equivocation of Golay $[23, 12, 7]$ code on BSC with and without 2 erasures

In this case, the deliberate introduction of 2 erasures has increased the normalised

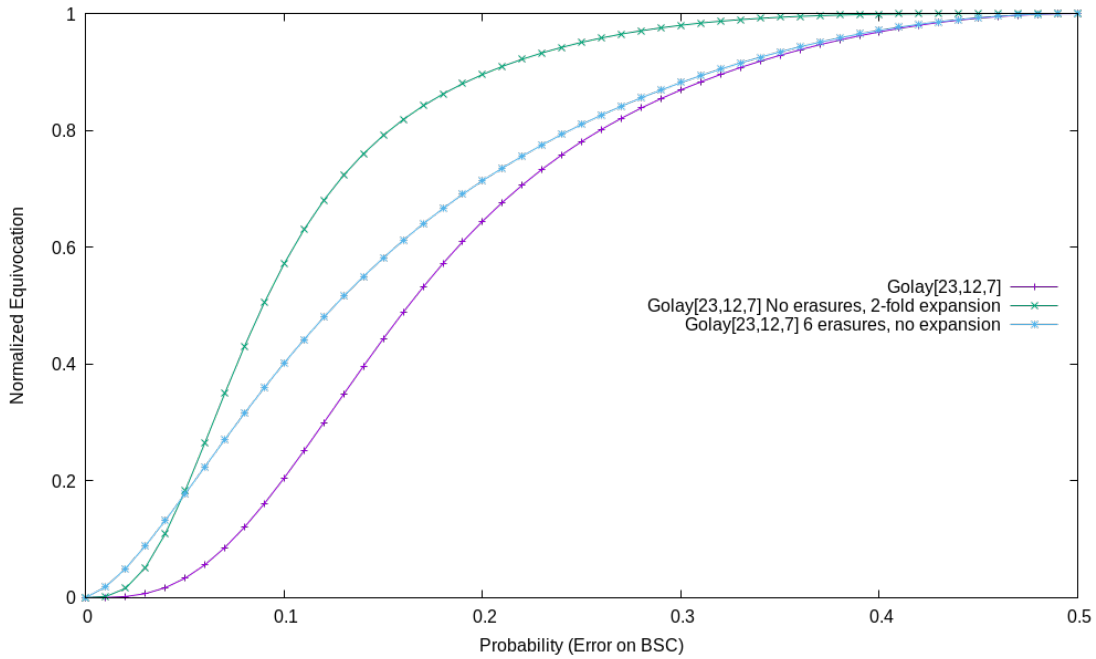


Figure 6.8: Equivocation of Golay $[23, 12, 7]$ code with 6 erasures versus 2-fold code expansion

equivocation for the eavesdropper by a greater amount (0.089) than for the legitimate receiver (0.0013), a factor of 67.

However, this does not always hold and is dependent on the scenarios being compared and the range of error probability being considered. Consider the Golay $[23, 12, 7]$ code with an error probability $p_e = 0.05$ for the legitimate receiver and overall error probability 0.20 for the eavesdropper for scenarios with no erasures and 6 erasures, as shown in Table 6.7.

	Legitimate receiver	Eavesdropper
BSC error probability	0.05	0.20
Equivocation (with no erasures)	0.03317	0.64453
Equivocation (with 6 erasures)	0.17759	0.71385
Increase	0.14442	0.06932

Table 6.7: Equivocation of Golay $[23, 12, 7]$ code on BSC with and without 6 erasures

In this case, the equivocation has increased for the legitimate receiver by a greater amount (0.144) than it has for the eavesdropper (0.069).

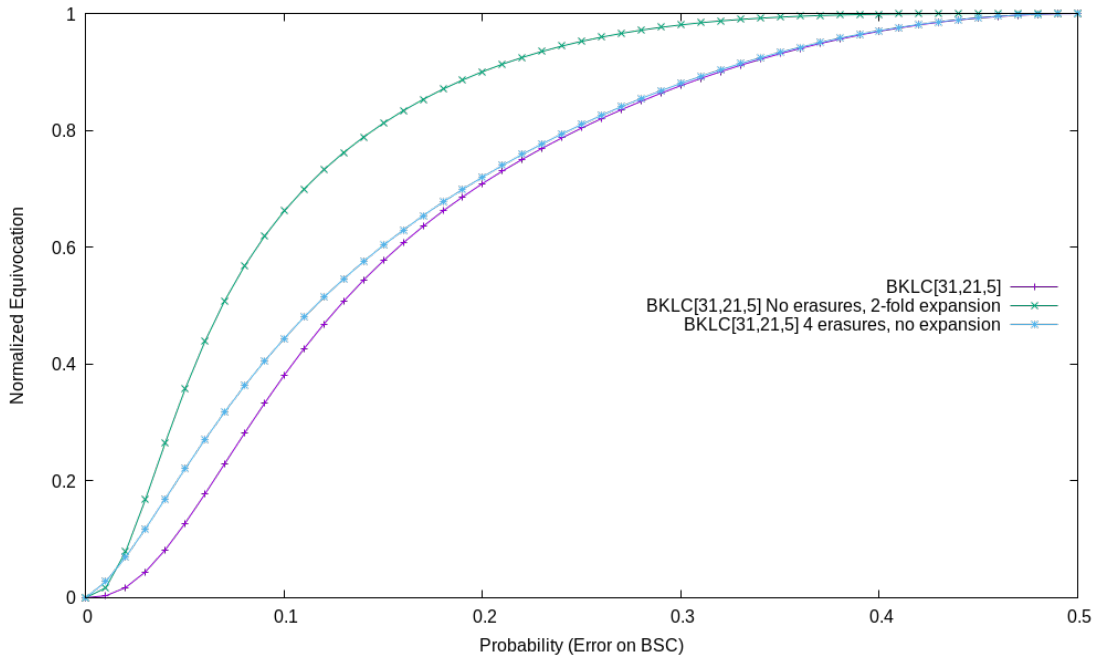


Figure 6.9: Equivocation of BKLC $[31,21,5]$ with 4 erasures versus 2-fold code expansion

6.2.3 Conclusion

- The intentional introduction of erasures to the BSC increases the equivocation of the code, but not generally by as much as the use of code expansion. For example, below approximately 0.05, introducing 6 erasures gives higher equivocation levels than 2-fold code expansion but above 0.05, 2-fold code expansion always gives a higher equivocation value. However 10-fold code expansion would give higher equivocation levels and 6 erasures for every error probability examined (6 erasures would only give higher values for very small error probabilities ie well below 0.01)

- An increase in the number of erasures on the IE+BSC may not always lead to an increase in the equivocation. For example, either 3 or 4 intentional erasures of the BKLC$[8,2,5]$ code on a BSC will yield the same equivocation levels in each case, because the additional erasure doesn't change the ability of the code to use the available syndrome patterns for the detection and correction of either errors

or erasures.

- As with the BSC, code expansion can be used on the IE+BSC to increase equivocation.
- Combinations of erasures and code expansion can be used to maximise the difference in equivocation for the legitimate receiver and an eavesdropper for a particular error probability. For example, from Figure 6.7, the Golay$[23,12,7]$ code on the BSC with 3 erasures and 4-fold expansion could give a legitimate recipient with an error probability of 0.01 a normalised equivocation value of 0.054, whereas an eavesdropper with a WTC error probability of 0.05 would be liable to a normalised equivocation of 0.602, over 11 times greater.

6.3 Equivocation on the Binary Erasure Channel

The previous section had to address two factors that had a simultaneous impact on the equivocation of the channel, namely the intentional use of erasures and the likelihood of an error from the BSC itself.

This section looks at the transmission of data across a Binary Erasure Channel as shown in Figure 6.10. The message m is encoded as the codeword x ahead of transmission across the BEC. On receipt, x may have had a bits erased to yield the received vector r. This is then corrected to give y before being decoded to the message estimate m_{est}. The erasures are no longer 'intentional' or 'controlled' and can occur in any position. The erasures can occur in any single position with a probability of p_s.

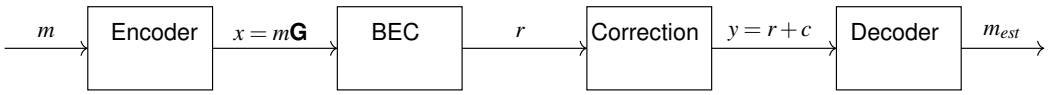


Figure 6.10: BEC encoding / decoding process

The probability of a erasures occurring in any word of length n is:

$$P(a) = \binom{n}{a}(1-p_s)^{n-a}p_s^a \tag{6.3}$$

6.3.1 Calculation

As previously, a spreadsheet was first drawn up as a 'proof of concept' prior to the calculation algorithm being coded in C++.

The equivocation is calculated by initially considering every possible erasure pattern for each transmitted codeword, as in Table 6.8.

					Option probability
Messages	0 0	0 1	1 0	1 1	
Codewords	00000000	00011111	11100111	11111000	
0 Erasures	00000000	00011111	11100111	11111000	$P(a=0)=0.43046721$
1 Erasure	0000000?	0001111?	1110011?	1111100?	0.04782969
	⋮	⋮	⋮	⋮	⋮
	?0000000	?0011111	?1100111	?1111000	0.04782969
					$P(a=1)=0.38263752$
2 Erasures	000000??	000111??	111001??	111110??	0.00531441
	00000?0?	00011?1?	11100?1?	11111?0?	⋮
	⋮	⋮	⋮	⋮	⋮
	??000000	??011111	??100111	??111000	0.00531441
					$P(a=2)=0.14880348$
			⋮		
7 Erasures	0???????	0???????	1???????	1???????	$9E-08$
	⋮	⋮	⋮	⋮	⋮
	???????0	???????1	???????1	???????1	$9E-08$
					$P(a=7)=7.2E-08$
8 Erasures	????????	????????	????????	????????	$P(a=8)=1E-08$

Table 6.8: BEC Erasure probabilities for BKLC $[8,2,5]$ with $p_s = 0.1$

For each erasure option, the origin of the option must be considered. For example, with bits 6 and 8 of the received codeword $\begin{pmatrix} 0_1 & 0_2 & 0_3 & 1_4 & 1_5 & ?_6 & 1_7 & ?_8 \end{pmatrix}$ erased, $2^2 = 4$ options must be taken into account. To decide which option is the most likely, calculate the syndrome. To be a valid codeword decode, the syndrome must be zero.

Table 6.9 shows that for the received codeword $\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & ? & 1 & ? \end{pmatrix}$, $\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$ is the only option that yields a syndrome of zero and is the most likely to be correct. Therefore this option would be chosen as the decode for the

Received codeword	Syndrome
00011?1?	
00011010	111001
00011011	000010
00011110	111011
00011111	**000000**

Table 6.9: BEC Erasure options for bits 6 and 8 for BKLC $[8,2,5]$

received, erased codeword.

This process works well when the errors introduced are well within the erasure correcting capabilities of the codeword. However as more erasures occur, more than one zero-syndrome may occur, with no way of distinguishing which yields the correct decode.

So for the received codeword $\begin{pmatrix} 0 & 0 & ? & ? & ? & ? & ? & ? \end{pmatrix}$ with 6 erasures, two of the $2^6 = 64$ possible options are valid codewords, (00000000 and 00011111) and give a zero-syndrome as shown in Table 6.10.

Received codeword	Syndrome
00??????	
00000000	**000000**
00000001	111011
00000010	000001
00000100	000010
⋮	⋮
00000011	111010
⋮	⋮
11000000	110000
⋮	⋮
11110000	111100
00011111	**000000**
00101111	001100
⋮	⋮
11111011	000100
11111101	111001
11111110	000011
11111111	111000

Table 6.10: Syndromes for BKLC $[8,2,5]$

When looking at the Hamming $[7,4,3]$ code, with its limited error and erasure correcting capability, as a perfect code it can correct up to 2 erasures on the BEC. However for 4 erasures there are 2 possible correct decode options, 4 options for 5 erasures, 8 options for 6 erasures and all 16 possible options for all 7 bits being erased.

This leads to the question of how to decide which valid decode is the correct option. Since all the options that give a valid decode are equally likely, there are 2 apparent decoding strategies.

- Choose a valid decode option randomly.
- Choose the first valid decode option i.e. the first option that gives a zero-syndrome.

Calculation of the equivocation using the first strategy will give a slightly different value each time since different decode options are randomly selected. The values will differ because it is only the received vectors with high numbers of erasures that are affected, beyond the correction capability of the code and therefore with low probabilities of occurrence.

The second strategy is one specific case of the first strategy and its use would provide consistent, predictable results. Since the first valid decode option is just as likely as any other option to be the correct codeword and this strategy yields some interesting results, this is the strategy pursued.

For the BKLC $[8,2,5]$ code with a BEC erasure rate of $p_s = 0.01$, the message $\begin{pmatrix} 0 & 1 \end{pmatrix}$ will be transmitted as the codeword $\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$. The full list of all possible received vectors and their decodes, probabilities and contributions to the joint entropy are listed at Appendix B. Table 6.11 shows that for the transmitted codeword $\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$ there are 8 possible erasure combinations that would result in an incorrect decode using the strategy described. In this case, all would yield the zero codeword as in Table 6.11

When summing the contributions to the joint entropy for each of the transmitted code-

Erasure pattern	Probability	Decode
000?????	9.7E-11	00000000
00??????	9.8E-13	00000000
0?0?????	9.8E-13	00000000
?00?????	9.8E-13	00000000
0???????	9.9E-15	00000000
?0??????	9.9E-15	00000000
??0?????	9.9E-15	00000000
????????	1E-16	00000000

Table 6.11: Incorrect Decodes for BKLC $[8,2,5]$ with $p_s = 0.01$

words, Table 6.12 is obtained. It can be seen that for the BKLC $[8,2,5]$ with a BEC erasure probability of $p_s = 0.01$, the joint equivocation H(X,Y) is close to 2.

Trasnmitted Codeword	Decode	$P(y_j\|x_i)$	$P(x_i,y_j)$	$H(x_i,y_j)$
$x_0 = 00000000$	00000000	0.25	0.0625	0.5
	00011111	0	0	0
	11100111	0	0	0
	11111000	0	0	0
$x_1 = 00011111$	00000000	2.5E-11	6.25E-12	8.805E-10
	00011111	0.25	0.0625	0.5
	11100111	0	0	0
	11111000	0	0	0
$x_2 = 11100111$	00000000	2.5E-13	6.25E-14	1.047E-11
	00011111	2.5E-11	6.25E-12	8.805E-10
	11100111	0.25	0.0625	0.5
	11111000	0	0	0
$x_3 = 11111000$	00000000	2.5E-11	6.25E-13	8.805E-10
	00011111	2.5E-13	6.25E-14	1.047E-11
	11100111	2.5E-11	6.25E-013	8.805E-10
	11111000	0.25	0.0625	0.5
			Joint Entropy $H(X,Y)$	2.0000000035

Table 6.12: Joint Entropy contributions for BKLC $[8,2,5]$ with $p(s)$ = 0.01

6.3.2 Calculating $H(Y)$

In Chapter 4, it was noted that $H(X|Y) = H(X,Y) - H(Y)$ and that $H(Y|X) = H(X,Y) - H(X)$. Up until this point, previous channel arrangements have meant that $H(X)$ and $H(Y)$ have been equal and therefore so have $H(X|Y)$ and $H(Y|X)$. However, due to the decoding method used with transmission across the BEC, this is no longer the case. In order to calculate $H(X|Y)$ as a measure of the secrecy of the code across the channel,

it is important to calculate the value of $H(Y)$ as it changes with p_s.

As the number of erasures increases beyond the ability of the code to correct them, the probability of incorrectly decoding the received codeword increases. Since the decoding strategy in the event of multiple valid decode options is to select the first option, there will be an increasing probability of selecting the first codeword that gives a zero-syndrome, usually the zero codeword. In extremis, if all bits are erased, then the zero codeword will always be selected.

Table 6.12 gives the conditional and joint probabilities obtained for an erasure probability of $p_s = 0.01$, while Figure 6.11 shows just the conditional probabilities for $p_s = 0.8$ with y_0 probabilities emphasised. The output probabilities can be obtained from these, shown for $p_s = 0.01$ in Table 6.13 and for $p_s = 0.8$ in Table 6.14. Both tables (but in particular Table 6.14) show that the probabilities of obtaining each output codeword are no longer uniform as was the case for both the BSC and the IE+BSC. This comes as a direct result of the decoding method being used and the decreasing likelihood of choosing codewords other than the zero codeword as p_s approaches 1.

Output codeword	$P(y)$		$H(y)$
00000000	0.25+2.5E-11 + 2.5E-13 + 2.5E-11	= 0.2500000001	0.5
00011111	0 + 0.25 + 2.5E-11 + 2.5E-13	= 0.25	0.5
11100111	0 + 0 + 0.25 + 2.5E-11	= 0.25	0.5
11111000	0 + 0 + 0 + 0.2499999999	= 0.2499999999	0.5
		H(Y)	2

Table 6.13: Output probabilties and entropies BKLC $[8,2,5]$ with p_s = 0.01

Output codeword	$P(y)$		$H(y)$
00000000	0.25 + 0.0819 + 0.0655 + 0.0819	= 0.4794	0.5085
00011111	0 + 0.1681 + 0.0400 + 0.0236	= 0.2316	0.4888
11100111	0 + 0 + 0.1445 + 0.0400	= 0.1845	0.4498
11111000	0 + 0 + 0 + 0.1045	= 0.1045	0.3405
		H(Y)	1.7876

Table 6.14: Output probabilties and entropies BKLC $[8,2,5]$ with p_s = 0.8

For a much higher p_s, the likelihood of having multiple erasures also increases according to Equation 6.3. If many erasures become likely, then the probability of the code

$x_0 = 00000000$, $P(x_0) = 0.25$	$\boldsymbol{P(y_0\|x_0) = 1}$
	$P(y_1\|x_0) = 0$
	$P(y_2\|x_0) = 0$
	$P(y_3\|x_0) = 0$
$x_1 = 00011111$, $P(x_1) = 0.25$	$\boldsymbol{P(y_0\|x_1) = 0.32768}$
	$P(y_1\|x_1) = 0.67232$
	$P(y_2\|x_1) = 0$
	$P(y_3\|x_1) = 0$
$x_2 = 11100111$, $P(x_2) = 0.25$	$\boldsymbol{P(y_0\|x_2) = 0.262144}$
	$P(y_1\|x_2) = 0.15990784$
	$P(y_2\|x_2) = 0.57794816$
	$P(y_3\|x_2) = 0$
$x_3 = 11111000$, $P(x_3) = 0.25$	$\boldsymbol{P(y_0\|x_3) = 0.32768}$
	$P(y_1\|x_3) = 0.09437184$
	$P(y_2\|x_3) = 0.15990784$
	$P(y_3\|x_3) = 0.41804032$

Figure 6.11: Input and output probabilities for BKLC $[8,2,5]$ on BE C with 2 erasures and $p_e = 0.8$

being unable to correct the erasures will also increase. As this happens, it becomes more likely that the decoding process will resort to selecting the zero codeword as the decode. As previously noted, in extremis when all bits are erased, it becomes a certainty that the zero codeword will be selected as the decode. Hence as $p_s \to 1$, the output entropy $H(Y) \to 0$. When considering the relationship between $H(X)$ and $H(Y)$ as shown in Figure 4.2, as p_s increases $H(Y)$ decreases, as shown in Figure 6.12.

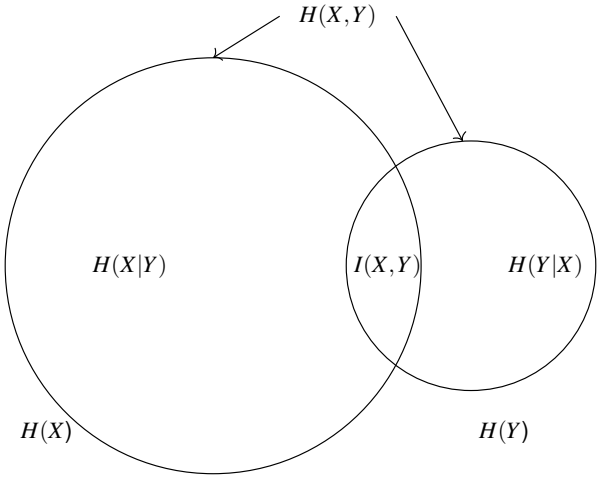


Figure 6.12: Changing Relationship between Input and Output Entropies on BEC as p_s increases and $H(Y)$ decreases

Once both $H(X,Y)$ and $H(Y)$ have been calculated, the equivocation can be found from $H(X|Y) = H(X,Y) - H(Y)$.

6.3.3 Results

When dealing with the BSC, it was only necessary to consider symmetric crossover errors of probability $0 \leq p_e \leq 0.5$. Once p_e reaches 0.5, the channel has become fully randomised, with the output having ceased to be dependent on the input. There is no point proceeding with error probabilities greater than 0.5 because an error has become more likely than not. However for the BEC, the risk of an erasure occurring must be considered up until it becomes a certainty i.e. the full range $0 \leq p_s \leq 1$ must be considered. Figure 6.13 show the normalised equivocation for 4 codes of different lengths and Figure 6.14 shows the corresponding normalised values of $H(Y)$ for each code as p_s changes, demonstrating how the output entropies decrease rapidly as p_s

approaches 1. Figure 6.15 shows the normalised equivocation for a range of Best Known Linear Codes of length $n = 15$. Both Figure 6.13 and Figure 6.15 illustrate the dramatic differences in normalised equivocation between different length codes.

The ability of a code to correct erasures is limited by the d_{min} of the code. The figures suggest that of the BKLCs examined, those codes with low d_{min} values and low erasure correcting capabilities will yield higher equivocation levels. These differences in equivocation level are at their most prominent for mid-range probabilities. For example if $p_s = 0.5$, the normalised equivocation of the BKLC $[15, 4, 8]$ code is 0.0597882 while for the BKLC $[15, 12, 2]$ code it is 0.689081.

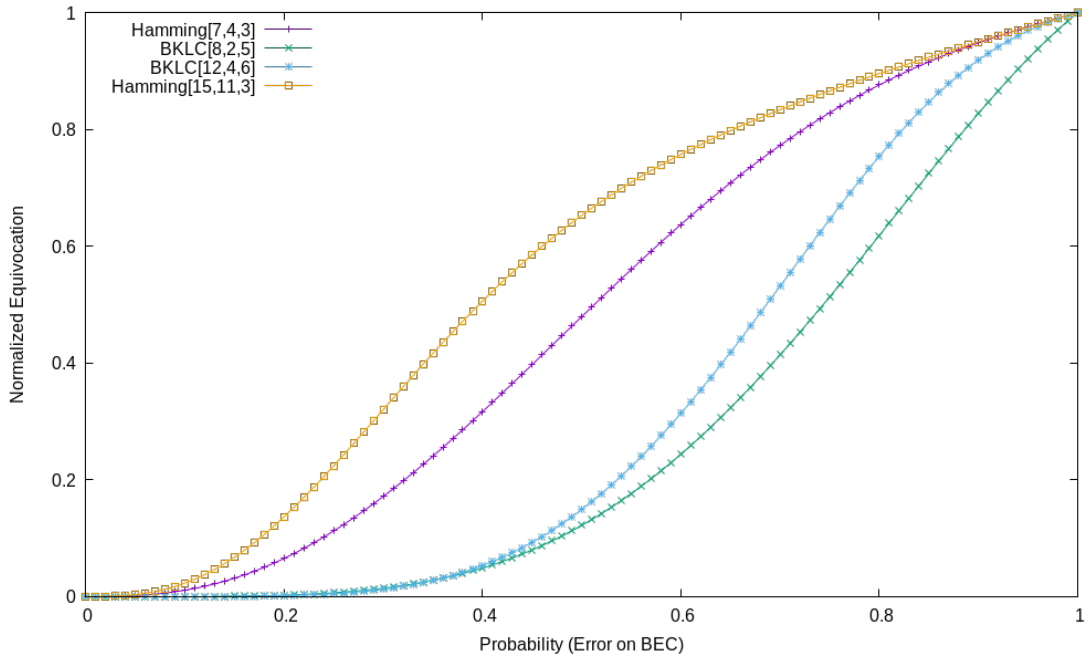


Figure 6.13: Equivocation of codes on the BEC

6.3.4 Conclusions

The work on equivocation on the BEC demonstrates that the methods previously used for the BSC can be extended to include other channels. Whilst some worthwhile results were obtained for a few codes, programming limitations prevented getting results for any codes of length greater than $n = 15$. Despite that, the results showed that there can be some very large differences in normalised equivocation values, even for closely

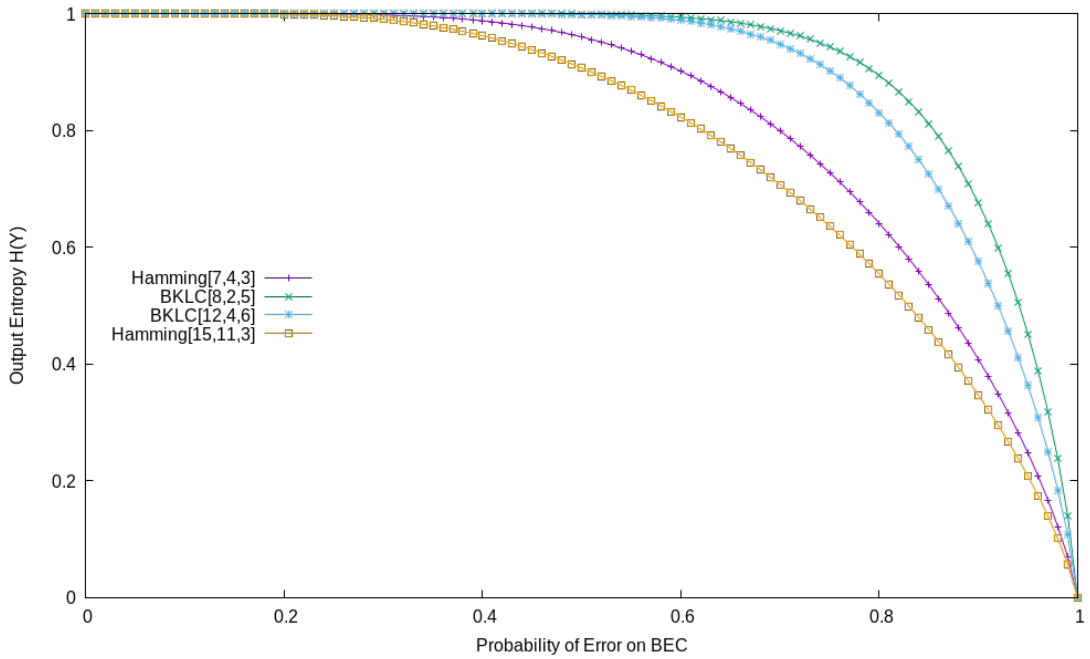


Figure 6.14: Output Entropy $H(Y)$ of codes on the BEC

related codes.

The most worthwhile point developed was the observation of a system, whose correction and decoding mechanism reduces the set of possible outcomes and hence the output entropy as the erasure probabilty p_s increases.

6.4 Chapter Conclusions

In addition to the expansion of codes previously discussed, erasures can be used as an alternative mechanism for increasing the equivocation of a code over a BSC and thereby its average ambiguity and secrecy.

- Increases in equivocation from the introduction of intentional erasures on the BSC can be significant, but not as great as those increases shown by the use of code expansions. For example, from Figure 6.7 in subsection 6.2.2, for the Golay$[23, 12, 7]$ code with $p_e = 0.05$, 6 erasures give a normalised equivocation of 0.178 while 4-fold expansion gives a much higher value of 0.540.

- Some increases in the number of bits erased on the IE+BSC do not necessarily

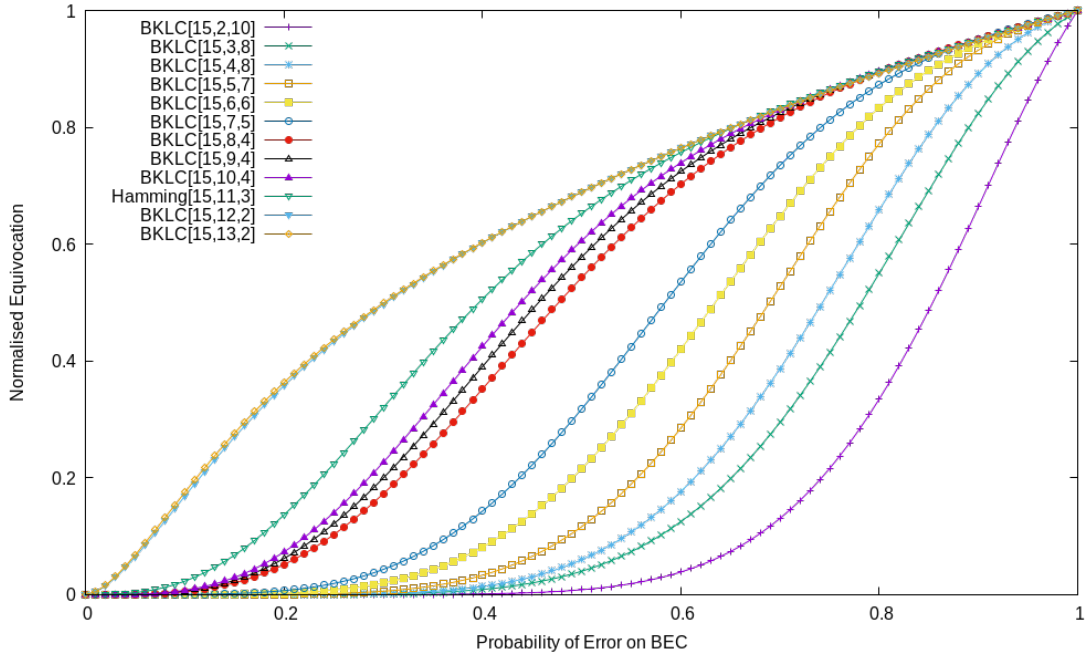


Figure 6.15: Equivocation of BKLCs of length 15 and different k on BEC

lead to an increase in equivocation. This was seen with BKLC$[8,2,5]$, where the increase from 3 to 4 erasures did not change the equivocation. This was because the increase in erasures didn't change the ability of the code to use the available syndrome patterns for the detection or correction of either errors or erasures.

- The controlled introduction of erasures to a BSC can lead to a greater increase in equivocation for an eavesdropper than it does for the legitimate receiver. The introduction of 2 erasures to the Golay$[23,12,7]$ code on the IE+BSC gave an increase in normalised equivocation of 0.00132 for the legitimate receiver with $p_e = 0.05$ but a much larger increase of 0.08898 for an eavesdropper with $p_e = 0.1$.

- A comparison between code expansions and erasures can show situations where, for a particular error probability, there is a changeover point in which method produces the higher equivocation values. It was seen that for Golay $[23,12,7]$ with $p_e < 0.05$, 6 erasures gave a higher equivocation value, whereas for $p_e > 0.05$, 2-fold code expansion gave higher values.

- On the BEC in Section 6.3, even codes with similar properties or similar lengths

can have wildly different equivocation characteristics. Figure 6.13 shows that with $p_s = 0.2$, BKLC$[15, 10, 4]$ had a normalised equivocation of just 0.0733, whereas BKLC$[15, 12, 2]$ had a value of 0.3581.

- The correction and decoding mechansim used for the BEC in subsection 6.3.2 caused a reduction in the possible outcomes and a decreasing output entropy as the probability of an erasure increased. As $p_s \to 1$, $H(Y) \to 0$. In extremis, when all bits are erased, it becomes a certainty that the zero codeword will be selected as the decode.

Chapter 7

Equivocation of Deletions

7.1 Introduction

Golomb et al. (1963) observed that:

> the single insertion and deletion channel, even in the absence of other noise, would make a fascinating study.

Subsequent work by Dobrushin (1967) and Stambler (1970) discussed coding theorems for channels subject to random deletions and insertions, while Graham (2015) introduces an alternative model for the Binary Deletion Channel (BDC) and Davey & Mackay (2001) discussed reliable communication of channels with insertions, deletions and substitutions. No closed-form expression for the capacity of the BDC exists at present, however extensive work has been undertaken on providing tighter bounds for the BDC by Dalai (2011), Mitzenmacher (2006), Kalai et al. (2010), Kanoria & Montanari (2010, 2013), Kirsch & Drinea (2007), Fertonani & Duman (2010).

This chapter looks at a method of evaluating the equivocation of a code when symbols are intentionally deleted ahead of transmission across a Binary Symmetric Channel (ID + BSC), shown in Figure 7.1. The system includes a simple state switch that controls whether a deletion is introduced into the system or not. When the switch is in position '0', the system acts as a simple BSC with wiretap, the results of which have already been discussed in Section 5.4. When the switch is in position '1', the system acts as an ID+BSC with Wire-Tap.

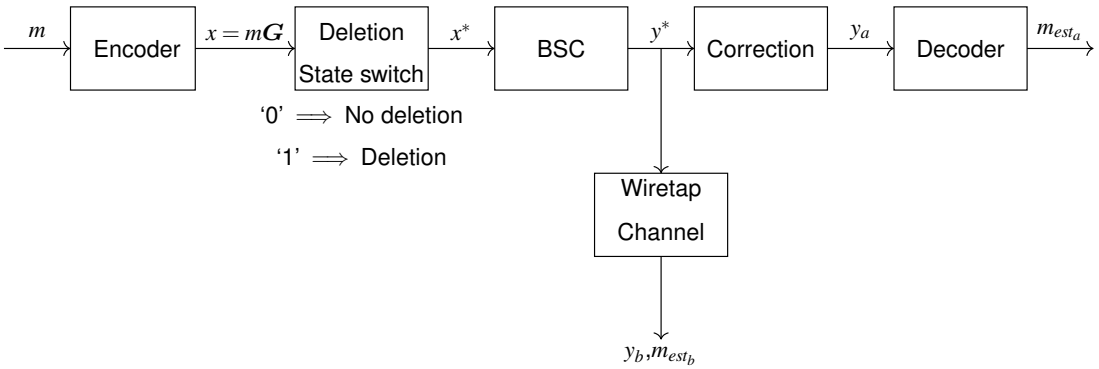


Figure 7.1: Code transmission across a BSC with intentional deletions (ID+BSC), subject to wiretap

7.2 Equivocation of a Binary Symmetric Channel with Intentional Deletions (ID+BSC)

If δ deletions of transmitted symbols $x_i \in X$ are introduced for the ID+BSC then as with the BDC, the positions of the received bits y_i^* may differ from the transmitted positions.

$$\{x_0 x_1 .... x_{n-1}\} \mapsto \{y_0^* y_1^* .... y_{n-\delta-1}^*\}$$

The $n-\delta$ non-deleted bits will also be susceptible to a cross-over probability of p_e on the BSC.

However, when considering multiple deletions, the number of permutations that must be considered rapidly extends beyond the practical capabilities of a 'standard' PC, even if the parallel processing techniques previously described were implemented on the GPU. For δ deletions in a codeword of length n, there are $\binom{n}{\delta}$ possible arrangements of locations in which the bits could have been deleted. If any number of deletions were permitted, up to a value δ, $\sum_{i=0}^{\delta}\binom{n}{i}$ cumulative arrangements must be considered. Even for a relatively short code such as the Golay $[23,12,7]$ code, considering 2 deletions rather than 1 deletion gives 11 times more arrangements of deletion positions

to consider since

$$\frac{\binom{23}{2}}{\binom{23}{1}} = \frac{{}^{23!}/_{21!2!}}{{}^{23!}/_{22!1!}} = \frac{{}^{(23\times22)}/_{(2\times1)}}{{}^{23}/_{1}} = {}^{22}/_{2} = 11$$

Considering up to and including 3 deletions would give 89 times more arrangements of deletion positions to consider:

$$\frac{\binom{23}{1}+\binom{23}{2}+\binom{23}{3}}{\binom{23}{1}} = \frac{23+253+1771}{23} = \frac{2047}{23} = 89$$

The impact of the BSC must still be considered for every possible arrangement. Therefore **this chapter will only consider a single intentional deletion** ahead of transmission via the BSC.

In Figure 7.1, a message m is encoded as the codeword x of length n and subjected to a single-bit deletion, yielding x^*. before being transmitted across the BSC. The received punctured codeword y^* must then be corrected back to a valid codeword y of length n before finally being decoded to an estimate of the original message, m_{est}.

Punctured $(n-1)$-bit codewords x^* will be referred to as *codeword stubs*. Every codeword will generate its own set of n codeword stubs, some of which may be similar to each other or to stubs from other codewords. For every position in which the deletion could have occurred, all 2^{n-1} possible BSC errors must be considered.

The $2^4 = 16$ possible sets of codeword stubs for the Hamming $[7,4,3]$ code are shown at Appendix C.

7.2.1 Calculation of Equivocation

As previously, a proof-of-concept spreadsheet was drawn up in Microsoft Excel prior to coding up the algorithm. Even for a code as simple as the Hamming $[7,4,3]$ code, such a spreadsheet consists of many hundreds of rows and columns. It was assumed that messages selected are independent and identically distributed, such that for a linear $[n,k,d]$ code, $P(x_i) = \frac{1}{2^k}$.

There are two main phases to the process of calculating the equivocation of an ID+BSC

channel:

- Determine how to decode each received punctured codeword
- Evaluate the entropic contribution for every possible input/output combination, allowing for both intentional deletions and BSC errors.

Maximum Likelihood Decoding (MLD) compares received sequences with valid codewords, taking into account the confidence in the received symbols and selecting the codeword closest to the received sequence (Sweeney 2002, van Lint 1999, Togneri & deSilva 2002). To decide how to decode the codeword stubs, a MLD approach is used:

- For all 2^{n-1} received codeword stubs, list the $2n$ possible n-bit rebuild options $r_{u,v}$ that the stub could have come from (u indicates the stub number $0 \leq u \leq 2^{n-1}-1$, v indicates the rebuild number $0 \leq v \leq 2n-1$).
- Find the Hamming distance $d(r_{u,v}, x_i)$ of each rebuild option from all 2^k valid codewords x_i. However only rebuild options with the reconstructed bit equal to the equivalent bit of the valid codeword need be considered, halving the number of distances that must be found. This requires the evaluation of $2^{n-1} \times n \times 2^k = n \times 2^{n+k-1}$ distances, a number that increases exponentially with both n and k.
- Given $d(r_{u,v}, x_i)$, calculate the probability, $P(x_i|r_{u,v})$ of the rebuild option having come from each valid codeword.
- The output codeword $\hat{y}_j$ with the highest probability $P_{max}(x_i|y_j)$ of being the source of the stub is selected.
- $P(\hat{y}_i)$ can be found from the mean probability of the rebuild having come from each codeword. Each transmitted codeword has n possibilities for each bit deleted, with probability $\frac{1}{n}$. The decoder has no information about which bit is deleted so has to sum over all n deletion possibilities to find the most likely codeword, given y_j.

- If two codewords are equally likely then one is chosen randomly.

For example, with the Hamming $[7,4,3]$ code and a BSC $p_e = 0.01$, the codeword stub $\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$ has 14 possible rebuild options with distances to valid codewords as shown in Table D.1 of Appendix D. Once the distances are known, the probabilities in Table D.2 can be calculated. This shows that codewords c_1 and c_8 have equal maximum likelihood of being the correct decode, so one of these is chosen, potentially rebuilding $\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$ as $\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$.

The probability of each received vector, $P(y_j)$ can be found from the mean of these probabilities for each received vector. For 0.01 with the Hamming $[7,4,3]$ code, the codeword stub $\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$ gives $P(y_7) = 0.01768$. A full list of decodes and output probabilities $P(y_j)$ for the Hamming $[7,4,3]$ code with one intentional deletion and when $p_e = 0.01$ is in Appendix E.

To evaluate the contribution to the entropy of every possible combination of input and output codeword:

- Consider every possible input message and corresponding codeword
- For every one of 2^k codewords, consider all n codeword stubs
- For every one of n codeword stubs, consider all 2^{n-1} possible BSC errors
- For every codeword stub with 2^{n-1} possible errors, consider its distance to a valid codeword and calculate $P(y_j|x_i)$. This gives $2^k \times n \times 2^{n-1} = 2^{n+k-1} \times n$ distances to consider i.e. it again rises exponentially with both n and k. Even for the short Hamming $[7,4,3]$ code, this requires the evaluation of 2^{10} distances.

By Bayes' Theorem,

$$P(x_i|y_j) = \frac{P(y_j|x_i) \cdot P(x_i)}{P(y_j)} \tag{7.1}$$

and since:

- $P(y_j|x_i)$ is known
- $P(y_j)$ is known
- $P(x_i) = \frac{1}{2^k}$ for an $[n,k,d]$ linear code with equally likely input codewords
- $H(x_i|y_j) = P(x_i|y_j) \cdot \log_2(\frac{1}{P(x_i|y_j)})$

we can calculate $H(x_i|y_j)$. By summing across all input codewords, we obtain Equation 7.2

$$H(X|y_j) = \sum_{x \in X} P(x_i|y_j) \log_2 \left(\frac{1}{P(x_i|y_j)} \right) \tag{7.2}$$

The contributions to the conditional entropy of codeword y_7 are shown in Table 7.1. Subsequent summing over all received codewords gives Equation 7.3

$$\begin{aligned} H(X|Y) &= \sum_{y \in Y} P(y_j) H(X|y_j) \\ &= \sum_{x \in X} \sum_{y \in Y} P(y_j) P(x_i|y_j) \log_2 \left(\frac{1}{P(x_i|y_j)} \right) \\ &= \sum_{x \in X} \sum_{y \in Y} P(x_i, y_j) \log_2 \left(\frac{1}{P(x_i|y_j)} \right) \end{aligned} \tag{7.3}$$

For Hamming $[7,4,3]$ with $p_e = 0.01$, $H(X|Y) = 0.1681$. The calculation can be repeated for a range of probabilities and codes to give the results in subsection 7.2.5.

7.2.2 Code Choice

Initital calculations were performed with simple codes such as Hamming codes or Best Known Linear Codes, taken from existing code tables (Grassl 2015). However, whilst they may be efficient in terms of offering a good minimum Hamming distance relative to the code length, such codes may not be effective codes with regards to how they cope with deletions.

Take, for example, the two 7-bit codewords $x_1 = \begin{pmatrix} 0_1 & 1_2 & 0_3 & 0_4 & 1_5 & 0_6 & 1_7 \end{pmatrix}$ and $x_2 = \begin{pmatrix} 1_1 & 0_2 & 0_3 & 1_4 & 0_5 & 1_6 & 1_7 \end{pmatrix}$. They differ in positions 1,2,4,5 and 6 i.e. they have a Hamming distance of 5. However if bit 1 of x_1 is deleted and either bit 6 or 7

x_i	$P(y_7)$	$P(y_7\|x_i)$	$P(x_i\|y_7)$	$H(x_i\|y_7)$
$x_0 = 0000$	0.0176839	9.70299e-07	3.42932e-06	1.1009e-06
$x_1 = 0001$	0.0176839	0.138614	0.489903	0.00891835
$x_2 = 0010$	0.0176839	0.00141373	0.00499653	0.000675483
$x_3 = 0011$	0.0176839	0.00141373	0.00499653	0.000675483
$x_4 = 0100$	0.0176839	0.00141373	0.00499653	0.000675483
$x_5 = 0101$	0.0176839	0.00141373	0.00499653	0.000675483
$x_6 = 0110$	0.0176839	2.78642e-05	9.84805e-05	2.31792e-05
$x_7 = 0111$	0.0176839	4.243e-09	1.4996e-08	6.89245e-09
$x_8 = 1000$	0.0176839	0.138614	0.489903	0.00891835
$x_9 = 1001$	0.0176839	2.78642e-05	9.84805e-05	2.31792e-05
$x_{10} = 1010$	0.0176839	2.82843e-07	9.99651e-07	3.52353e-07
$x_{11} = 1011$	0.0176839	2.82843e-07	9.99651e-07	3.52353e-07
$x_{12} = 1100$	0.0176839	2.82843e-07	9.99651e-07	3.52353e-07
$x_{13} = 1101$	0.0176839	2.82843e-07	9.99651e-07	3.52353e-07
$x_{14} = 1110$	0.0176839	4.243e-09	1.4996e-08	6.89245e-09
$x_{15} = 1111$	0.0176839	9.70299e-07	3.42932e-06	1.1009e-06
			Sum $H(X\|y_7)$	0.02058863

Table 7.1: Entropy contributions for y_7

of x_2 is deleted ahead of transmission, the vector $\begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}$ is transmitted in both cases. Thus, even if there is no risk of error from the BSC $(p_e = 0)$, upon receipt it isn't possible to determine which codeword the received stub originated from. This will contribute towards the code having a non-zero equivocation even when $p_e = 0$.

In order to mitigate against this, codes can be designed for use with the deletion channel. Such codes will need to consider the number of ambiguous codeword stubs (i.e. they could have originated from more than one valid codeword) that exist once one or more deletions have been intentionally made.

However to consider every possible codeword stub interaction with all others in order to optimise the code for deletions is not trivial. An $[n,k,d]$ code will have 2^k codewords, each with n possible stubs. There will be $(2^k \times n) \times (2^{k-1} \times n) = 2^{2k-1} \times n^2$ interactions between possible pairs of stubs. For a $[7,2,d]$ code, this would give $2^3 \times 7^2 = 392$ interactions, but for the Hamming $[15,11,3]$ code it rises to $2^{21} \times 15^2 = 4.82 \times 10^8$.

7.2.3 Varshamov-Tenengolts Codes

One useful set of codes when dealing with deletions highlighted by Sloane (2002) are the Varshamov & Tenengolts (1965) codes. These codes were proven to be useful for correcting single deletions by Levenshtein (1965*a*) and Levenshtein (1965*b*).

For $0 \leq a \leq n$, the Varshamov-Tenengolts code $VT_a(n)$ consists of all binary vectors $(x_1,...,x_n)$ satisfying

$$\sum_{i=1}^{n} ix_i \equiv a(\text{mod}(n+1)) \tag{7.4}$$

Alternatively,

$$x_1 + 2x_2 + ix_i, ...nx_n = m(n+1), \qquad m \in \mathbb{Z}^+$$

Codes with $a = 0$ contain the most codewords, giving the first few such codes as:

Code	Codewords
$VT_0(1)$	$\{0\}$
$VT_0(2)$	$\{00, 11\}$
$VT_0(3)$	$\{000, 101\}$
$VT_0(4)$	$\{0000, 1001, 0110, 1111\}$
$VT_0(5)$	$\{00000, 10001, 01010, 11011, 11100, 00111\}$
$VT_0(6)$	$\{000000, 001011, 001100, 010010, 011110, 100001, 101101, 110011, 110100, 111111\}$
$\vdots$	$\vdots$

Table 7.2: Varshamov-Tenengolts codes

A binary word of length n is a valid codeword of a Varshamov-Tenengolts $VT_0(n)$ code if the sum of the positions of the bits with value 1 is equal to 0 mod $n+1$. So for the $VT_0(8)$ code, $\begin{pmatrix} 1_1 & 1_2 & 0_3 & 0_4 & 0_5 & 0_6 & 1_7 & 1_8 \end{pmatrix}$ is a codeword because bits 1,2,7 and 8 are set and $(1+2+7+8)$ mod $9 = 0$.

The key characteristic of $VT_0(n)$ codes that is of use in this situation is their resistance to deletions. Considering the $VT_0(5)$ code, deletion of a single bit of each codeword gives codeword stubs of length 4 as shown in Table 7.3.

Each codeword stub is unique to a single codeword without ambiguity, making it possible to determine its origin. All that remains is to allow for the impact of possible errors from transmission across the BSC.

Codeword	Single deletion codeword stubs
00000	0000
10001	1000, 1001, 0001
01010	0101, 0100, 0110, 0010, 1010
11011	1101, 1111, 1011
11100	1110, 1100
00111	0011, 0111

Table 7.3: Single deletion codeword stubs for $VT_0(5)$

It is worth noting Sloane's observation (Sloane 2002) that few $VT_0(n)$ codes are linear. Indeed $VT_0(n)$ codes are linear for $n \leq 4$ but are never linear again, since for $n \geq 5$, $VT_0(n)$ contains the vectors 1 0 . . . 0 0 1 and 1 1 . . . 1 0 0 but not their sum.

To identify $VT_0(n)$ codewords, the computer code in Figure 7.2 was used.

```
powN = 1 « n;
for( int x = 0; x < powN; x++ ){              // Considers every n-bit vector
     sum=0;
     for( int i = 0; i < n; i++ ){            // Considers each bit of the vector
          val = ( x & ( 1 « i) ) » i;
          if( val == 1 ){
                sum += ( n - i );} }          // Sum of positions with bit = 1
     if( sum % ( n + 1 ) == 0 ) {             // Checks if codeword is valid
          c[ nVT ] = x;                       // Adds to list of valid codewords
          nVT++;} }                           // Tally of valid codewords
```

Figure 7.2: Code for generating Varshamov-Tenengolts codes

The number of codewords in a particular VT code (the *cardinality* of the code) can be calculated from Sloane's Theorem:

$$|VT_a(n)| = \frac{1}{2(n+1)} \sum_{\text{odd } d|(n+1)} \phi(d) \frac{\mu(\frac{d}{(d,a)})}{\phi(\frac{d}{(d,a)})} 2^{(n+1)/d} \tag{7.5}$$

where d are the odd divisors of n, ϕ is the Euler totient function, $\mu(n)$ is the Möbius function and $(d,a) = gcd(d,a)$. For $a = 0$, this gives the sequence in Table 7.4

Varshamov-Tenengolts Code	Cardinality
$VT_0(1)$	1
$VT_0(2)$	2
$VT_0(3)$	2
$VT_0(4)$	4
$VT_0(5)$	6
$VT_0(6)$	10
$VT_0(7)$	16
$VT_0(8)$	30
$VT_0(9)$	52
$VT_0(10)$	94
$VT_0(11)$	172
$VT_0(12)$	316
$VT_0(13)$	586
$VT_0(14)$	1096
$VT_0(15)$	2048
$\vdots$	$\vdots$

Table 7.4: Cardinality of $VT_0(n)$ codes

A code with cardinality $|VT_0(n)|$ will contain x bits of information, where $2^x = |VT_0(n)|$ or $x = \frac{\log(|VT_0(n)|)}{\log 2}$. So for $VT_0(5)$ with 6 codewords, the equivocation must be divided by $\frac{\log 6}{\log 2}$ to normalise it.

If $a = 0$, then $gcd(d,a) = d$ and

$$\begin{aligned}
|VT_0(n)| &= \frac{1}{2(n+1)} \sum_{\text{odd } d|(n+1)} \phi(d) \frac{\mu(\frac{d}{d})}{\phi(\frac{d}{d})} 2^{(n+1)/d} \\
&= \frac{1}{2(n+1)} \sum_{\text{odd } d|(n+1)} \phi(d) \frac{\mu(1)}{\phi(1)} 2^{(n+1)/d} \\
&= \frac{1}{2(n+1)} \sum_{\text{odd } d|(n+1)} \phi(d) 2^{(n+1)/d}
\end{aligned}$$

However, if the special case where $n = 2^m - 1$, $m \in \mathbb{Z}^+$ is considered,

$$\begin{aligned}
|VT_0(2^m-1)| &= \frac{1}{2(2^m-1+1)} \sum_{\text{odd } d|(2^m-1+1)} \phi(d) 2^{(2^m-1+1)/d} \\
&= \frac{1}{2^{m+1}} \sum_{\text{odd } d|(2^m)} \phi(d) 2^{2^m/d}
\end{aligned}$$

but the only odd divisor of 2^m is 1, so

$$\begin{aligned}|VT_0(2^m-1)| &= \frac{1}{2^{m+1}}\phi(1)2^{(2^m)} \\ &= \frac{1}{2^{m+1}}2^{2^m} \\ &= 2^{2^m-m-1}\end{aligned}$$

Therefore, for $1 \leq m \leq 6$, the $VT_0(n)$ codes will have the cardinalities shown in Table 7.5

m	n	$\lvert VT_0(n)\rvert$
1	0	2^0
2	3	2^1
3	7	2^4
4	15	2^{11}
5	31	2^{26}
6	63	2^{57}

Table 7.5: Cardinality of $VT_0(n)$ codes with $n = 2^m - 1$

It can be seen that for $VT_0(n)$ codes with $n = 2^m - 1$, $m \geq 2$, the code will have the same number of codewords as the Hamming $(m,2)$ code (where $m = n-k$), described by the properties Ham$[2^m-1, 2^m-m-1, 3]$ or Ham$[n,k,3]$, i.e.:

$$|VT_0(n)| = |\text{Ham}[n,k,3]| \tag{7.6}$$

It is believed that this is the first time that this relationship has been described.

7.2.4 Expanded Varshamov-Tenengolts Codes

Consider the combination of a Varshamov-Tenengolts code with a repetition code where the f-fold expansion of any codeword repeats each original bit f times. Describing each bit of an expanded codeword with respect to its source bit and its repetition number, we have Figure 7.3.

If $f = 3$, the $VT_0(6)$ codeword $\left(101101\right)$, with $\sum_{i=1}^{6} ix_i = 1+3+4+6 = 14 = 0 \pmod 7$, becomes the $VT_0^3(6)$ codeword $\left(111000111111000111\right)$. The bit originally in position 4, x_4 would be repeated in positions 10, 11 and 12 in the expanded codeword and would

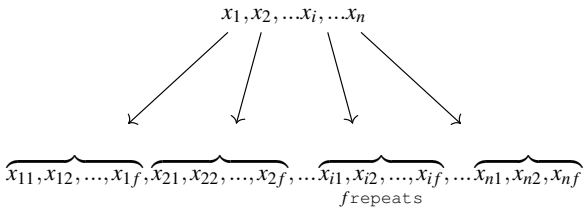


Figure 7.3: Expansion of a Varshamov-Tenengolts codeword

give a contribution of $10x_4 + 11x_4 + 12x_4$ to the sum of products. For any original bit x_i, the triplicated bits will now reside in positions $3i-2, 3i-1$ and $3i$. In general, for f repeats, the repeated bits will be in positions $f(i-1)i+1, f(i-1)i+2, \ldots f(i-1)i+f$. Overall, the sum of the products would become:

$$\begin{aligned}
\textstyle\sum_{i=1}^{n}\sum_{j=1}^{f} ix_{ij} \quad = \quad & (x_1 + 2x_1 + \ldots + fx_1) + \\
& ((f+1)x_2 + (f+2)x_2 + \ldots + 2fx_2) + \\
& \vdots \\
& (f(i-1)+1)x_i) + (f(i-1)+2)x_i + \ldots + (f(i-1)+f)x_i + \\
& \vdots \\
& (f(n-1)+1)x_n + (f(n-1)+2)x_n + \ldots + nfx_n
\end{aligned}$$

So the f terms that originated from the i'th bit yield a contribution of:

$$\begin{aligned}
& ((f(i-1)+1) + (f(i-1)+2) + \ldots (f(i-1)+f))x_i \\
& = (ff(i-1) + \Sigma_{i=1}^{f} i)x_i \\
& = ff(i-1) + \tfrac{f}{2}(f+1))x_i \\
& = f(fi - \tfrac{1}{2}(f-1))x_i
\end{aligned}$$

Which will always give an integer value of x_i since $f(f-1)$ is always even. For example, with triplication, the contribution from the expansion of the original fourth term of a $VT_0^3(6)$ codeword would be $3(3\times 4 - \frac{1}{2}(3-1))x_4 = 3(12-1)x_4 = 33x_4$ as previously noted.

Over all terms for an expanded VT code with i original bits, x_i , and f-fold expansion, this gives a sum of

$$\sum_{i=1}^{n}\sum_{j=1}^{f} ix_{ij} = \sum_{i=1}^{n} f(fi - \frac{1}{2}(f-1))x_i \tag{7.7}$$

So for the $VT_0^3(6)$ codeword, $\left(101101\right)$, we get

However $114 \neq 0 \pmod 7$, so expanded VT codewords do not retain this property of the

$$\begin{aligned}\sum_{i=1}^{n}\sum_{j=1}^{f} ix_{ij} &= 3(3\times 1-\tfrac{1}{2}(3-1))\cdots 1)+3(3\times 2-\tfrac{1}{2}(3-1))\cdots 0)+\\ &\quad 3(3\times 3-\tfrac{1}{2}(3-1))\cdots 1)+3(3\times 4-\tfrac{1}{2}(3-1))\cdots 1)+\\ &\quad 3(3\times 5-\tfrac{1}{2}(3-1))\cdots 0)+3(3\times 6-\tfrac{1}{2}(3-1))\cdots 1)\\ &= 6+0+24+33+0+51\\ &= 114\end{aligned}$$

primary VT codes. Despite this, an f-fold expansion of a $VT_0(n)$ code is resilient to multiple deletions, inheriting a combination of the VT code's ability to tolerate deletions and a repetition code's ability to increase its resistance to defects.

Landjev & Haralambiev (2007) proved that if C is a t-deletion correcting code of length n, then the expanded code:

$$C^f = \{(\underbrace{x_1\ldots x_1}_{f},\underbrace{x_2\ldots x_2}_{f},\ldots\underbrace{x_n\ldots x_n}_{f})|(x_1,x_2,\ldots x_n)\in C\} \tag{7.8}$$

is a code of length fn correcting $ft+f-1$ deletions.

So single-deletion correcting VT codes that have undergone f-fold expansion are capable of correcting $2f-1$ deletions e.g. a $VT_0^2(n)$ code can correct 3 deletions. For example, the $VT_0(4)$ code has the codewords (0 0 0 0), (1 0 0 1), (0 1 1 0), and (1 1 1 1). With 2-fold expansion, the $VT_0^2(4)$ codewords become those in Table 7.6.

Binary	Denary
(0 0 0 0 0 0 0 0)	0
(1 1 0 0 0 0 1 1)	195
(0 0 1 1 1 1 0 0)	60
(1 1 1 1 1 1 1 1)	255

Table 7.6: 2-fold expansion of Varshamov-Tenengolts $VT_0(4)$

When single, double and triple deletions are applied to each of these codewords, the 7, 6 and 5-bit codeword stubs (in denary) in Table 7.7, Table 7.8 and Table 7.9 are obtained respectively:

In each of the tables, each stub is unique to a single codeword. There are no ambiguities about the origin of the codeword stub. As discussed in subsection 7.2.2, if there is no ambiguity in the source of the codeword stub, then it possible to correctly

Codeword	Single deletion codeword stubs
0	0
195	97,99,67
60	30, 28, 60
255	127

Table 7.7: Single deletion codeword stubs for $VT_0^2(4)$

Codeword	Twin deletion codeword stubs
0	0
195	48, 49, 33, 51, 35, 3
60	15, 14, 30, 12, 28, 60
255	63

Table 7.8: Twin deletion codeword stubs for $VT_0^2(4)$

Codeword	Triple deletion codeword stubs
0	0
195	24, 16, 25, 17, 1, 27, 19, 3
60	7, 15, 6, 14, 30, 4, 12, 28
255	31

Table 7.9: Triple deletion codeword stubs for $VT_0^2(4)$

'correct' the received stub to the transmitted codeword, enabling an equivocation of zero, provided $p_e = 0$. Thus the expansion of Varshamov-Tenengolts codes enables the construction of codes that are resistant to multiple deletions. It's worth noting that whilst individual codeword lengths may be quite long, the number of codewords can be optimized for the intended use.

To construct a 'simple' multi-deletion tolerant code, the algorithm below could be used:

- Identify the number of information bits b to be transmitted
- Identify the level of deletion correction required e.g. to correct 5 deletions requires $2f-1=5$, or $f=3$.
- Select a suitable Varshamov-Tenengolts code where $|VT_0(n)| > 2^b$
- Generate the f-fold expanded VT code, $VT_0^f(n)$ from the original VT code

For example, a code that could transmit 11 information bits and be resistant to 2 dele-

tions would require a VT code of length 15 to be expanded 2-fold, giving 2048 codewords each of length 30. Unfortunately, it was considered impractical to write software to calculate the equivocation of these expanded VT codes as even this simple $VT_0^2(15)$ code would be moving beyond the processing capabilities available to the author.

7.2.5 Results

As with other scenarios investigated, the program was written to calculate the equivocation of the code on the ID+BSC for a range of probabilities $0 \leq p_e \leq 0.5$ with step size 0.01. The equivocation was found for a range of codes, up to the time-bounded practical limits of the algorithm. For the BKLC $[19,10,5]$, a run-time of approximately 2 days was required. Results for some codes of length less than 16 are shown in Figure 7.4 and for codes of length $16 \leq n \leq 19$ in Figure 7.5.

Many of the curves follow a similar pattern. One of the largest differences lies between those codes whose equivocation is zero when $p_e = 0$ and those with non-zero values.

If a code has a non-zero equivocation value when $p_e = 0$, then it is unable to cope with a single deletion even before the possibility of additional errors from the BSC is considered. Codes that cannot cope with a single deletion are of less interest and use than those that can tolerate a deletion. In all cases examined, the only crossover points occurred when comparing against a code with a non-zero equivocation. No crossover points between different codes of interest were identified.

Of the codes evaluated initially, only BKLC $[12,4,6]$, BKLC $[15,5,7]$, BKLC $[16,5,8]$ and the manually constructed code $[17,9]$ code had normalised equivocation values of less than the nominal value 0.01 when $p_e = 0$, as shown in Table 7.10. The alternative $[17,9]$ code had been designed as an attempt to reduce the impact of ambiguous codeword stubs, whilst enabling comparison with BKLC $[17,9,5]$.

7.2.6 Comparison of Equivocation for Erasures and Deletions

Since a deletion might be considered as an erasure that can occur in any position, it might appear tempting to compare their respective graphs of equivocation with era-

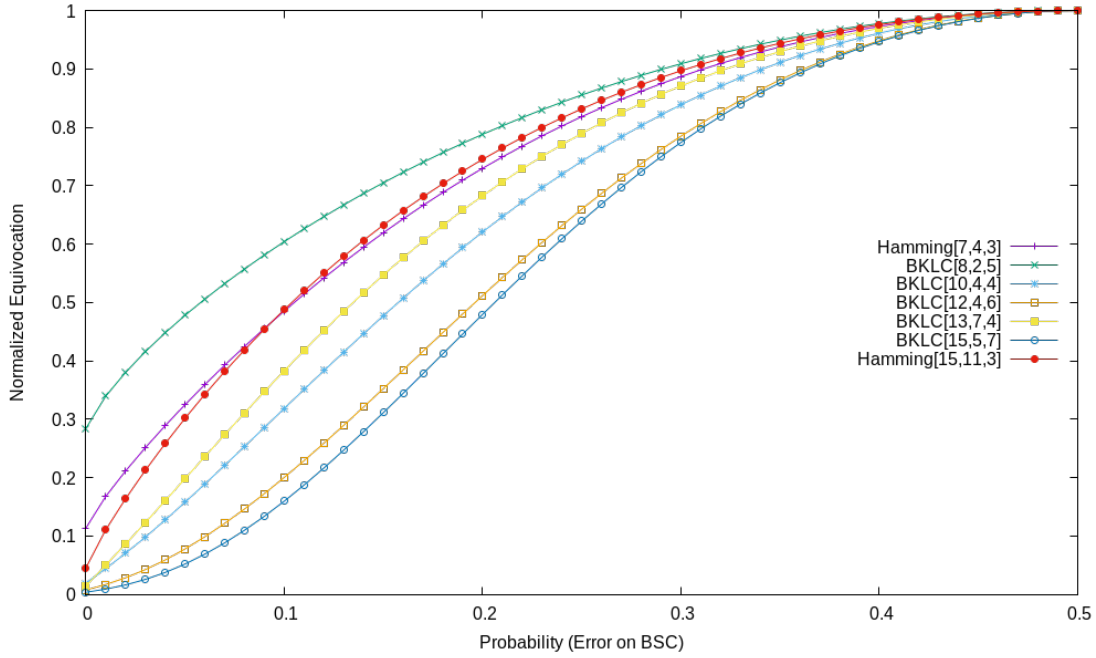


Figure 7.4: Equivocation of different codes of length 15 or less on the ID+BSC

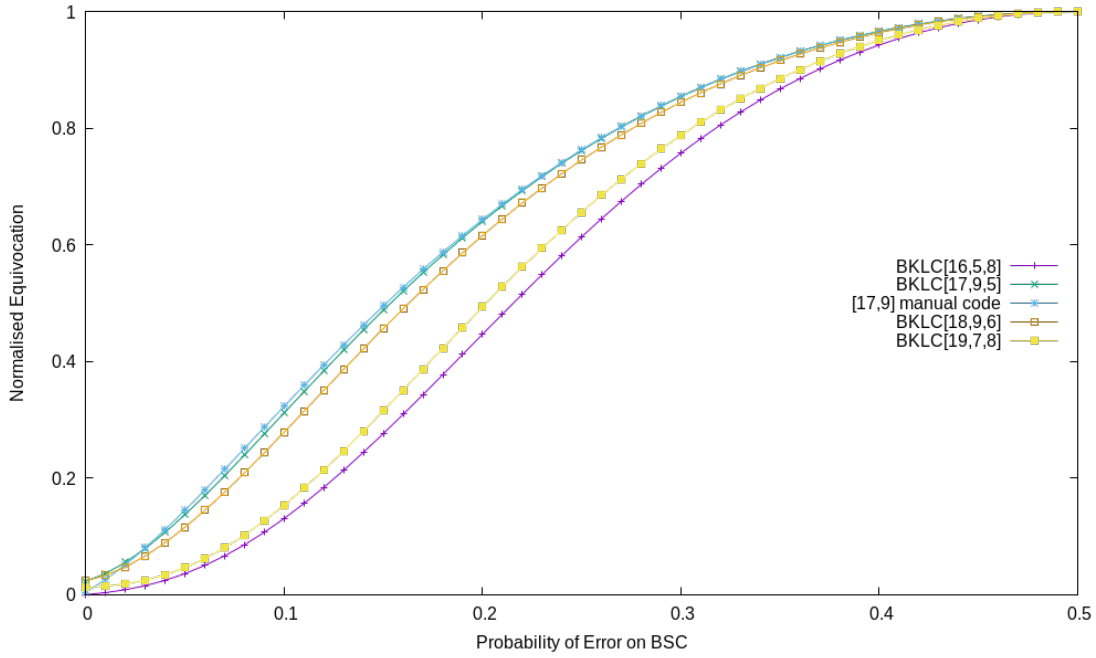


Figure 7.5: Equivocation of different codes of length $16 \leq n \leq 19$ on the ID+BSC

sures and deletions. For example, the equivocation of erasures on the BSC for BKLC $[8,2,5]$ in Figure 6.4 could perhaps be compared to the equivocation of a deletion on the BSC for the same code, shown in Figure 7.4. Such a comparison is shown in Fig-

Code	Normalized equivocation $H_N(X\|Y)$
Hamming $[7,4,3]$	0.1139
BKLC $[8,2,5]$	0.2845
BKLC $[10,4,4]$	0.0195
BKLC $[12,4,6]$	0.0078
BKLC $[13,7,4]$	0.0146
BKLC $[15,7,7]$	0.0039
Hamming $[15,11,3]$	0.0458
BKLC $[16,5,8]$	0.0000
BKLC $[17,9,5]$	0.0228
BKLC $[18,9,6]$	0.0239
BKLC $[19,7,8]$	0.0124

Table 7.10: Normalized equivocation values for ID+BSC when $p_e = 0$

ure 7.6. Consideration could be given as to whether the equivocation for a deletion might form some type of upper bound for increasing numbers of erasures. However it is not appropriate to make such a comparison. The underlying mechanisms behind how intentional erasures and intentional deletions on the BSC are handled and how the equivocation is calculated are significantly different because of the unknown location of the deletions. With deletions, decisions must be made about how to decode each punctured codeword, even though there may be ambiguities in that decision process when working with codes not specifically designed for handling deletions. Due to the different error correction and deletion correction properties of the code, the equivocation curve for a BKLC with single erasure starts at zero but the curve for a deletion may not. With deletions, entropic contributions for each input/output combination are calculated based on a maximum likelihood decoding that compares received sequences with valid codewords and must take into account the level of confidence in the received symbols. Furthermore, even if some form of relationship between i erasures and i deletions could be established, the practical challenges of calculating the equivocation for multiple deletions made it impractical to investigate during this research.

7.2.7 Equivocation of Varshamov-Tenengolts Codes

When equivocation values are calculated for Varshamov-Tenengolts codes on the ID+BSC, shown in Figure 7.7, a significantly different picture emerges than that for BKLCs. By

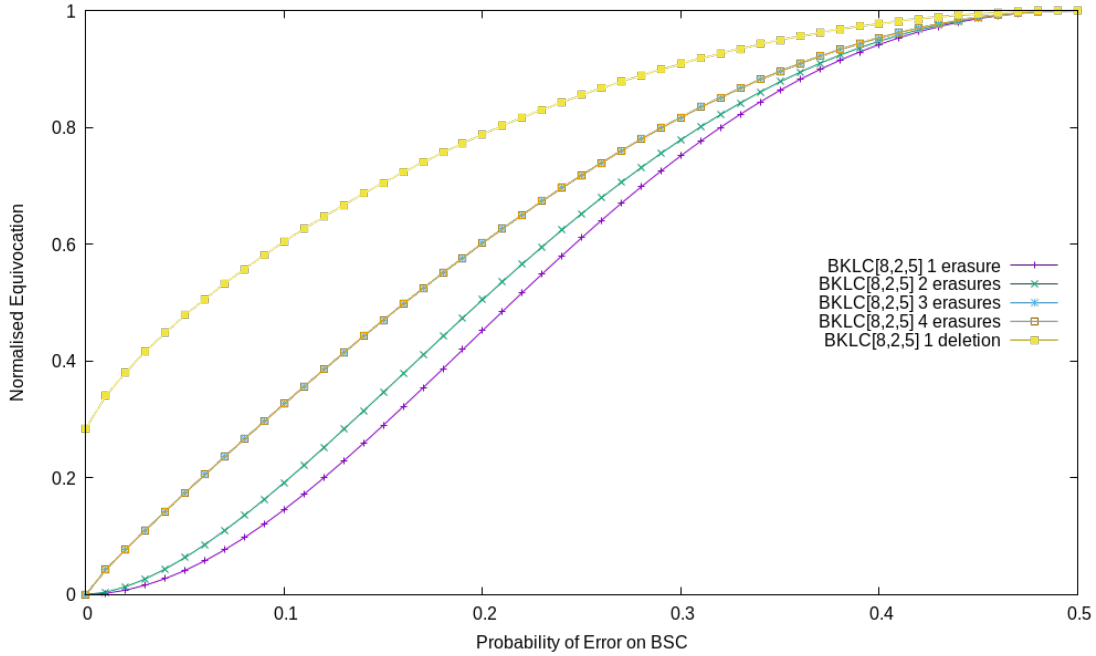


Figure 7.6: Comparison of a single deletion and multiple erasures for BKLC$[8,2,5]$

design, all VT codes offer an equivocation of zero when $p_e = 0$, however the graphs for the more 'purpose-built' VT codes also exhibit far less variation in their shape, with a gradient that decreases as p_e increases. For several BKLCs such as BKLC $[16,5,8]$, the gradient initially increases before reaching a point of inflexion and then decreasing. All graphs are monotonic i.e. if $p_{e_1} < p_{e_2}$ then $H_{p_{e_1}}(X|Y) < H_{p_{e_2}}(X|Y)$.

For the $VT_0(12)$ code, if the intended recipient had a BSC error probability p_e of 0.01, they would be liable for a normalised equivocation value of 0.084 whereas if the eavesdropper had a $p_e = 0.1$, they would be liable to a value nearly six times greater of 0.485.

A simple comparison in Figure 7.8 of the $VT_0(17)$ code with the BKLC $[17,9,5]$ and a code of length $n = 17$, manually constructed to reduce the risk of ambiguities between stubs when a bit is deleted, shows that the Varshamov-Tenengolts code provides an equivocation of zero when $p_e = 0$ but it also provides significantly higher equivocation values for all other BSC error probabilities.

When code expansion is introduced on top of intentional deletions on the BSC for Varshamov-Tenengolts codes, the results in Figure 7.9 are obtained.

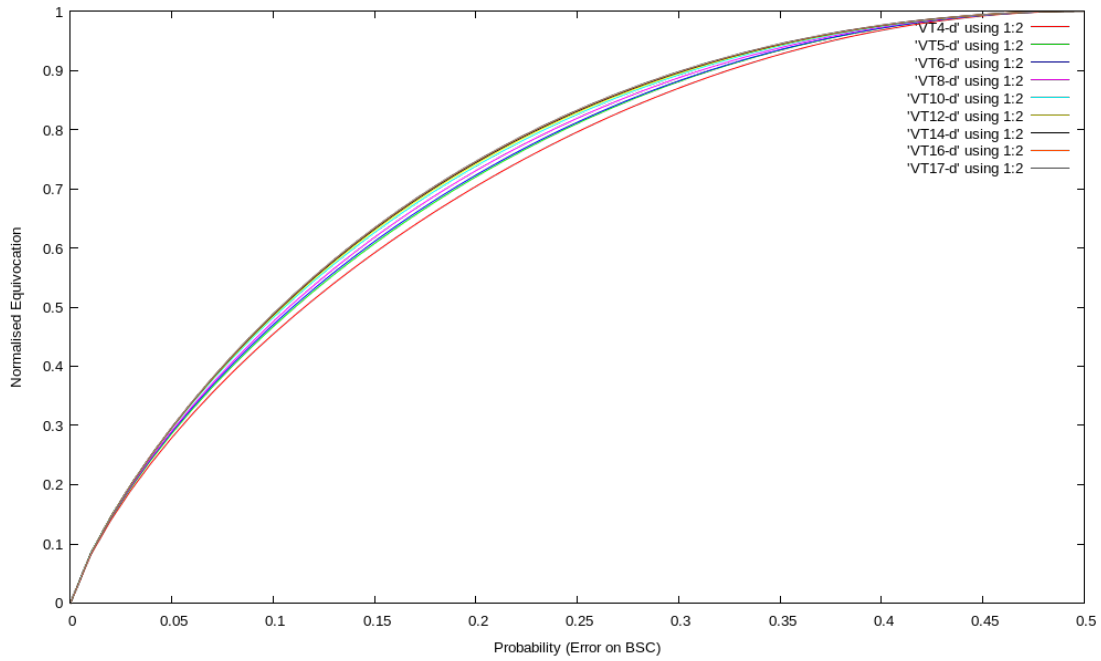


Figure 7.7: Equivocation of Varshamov-Tenengolts codes on ID+BSC

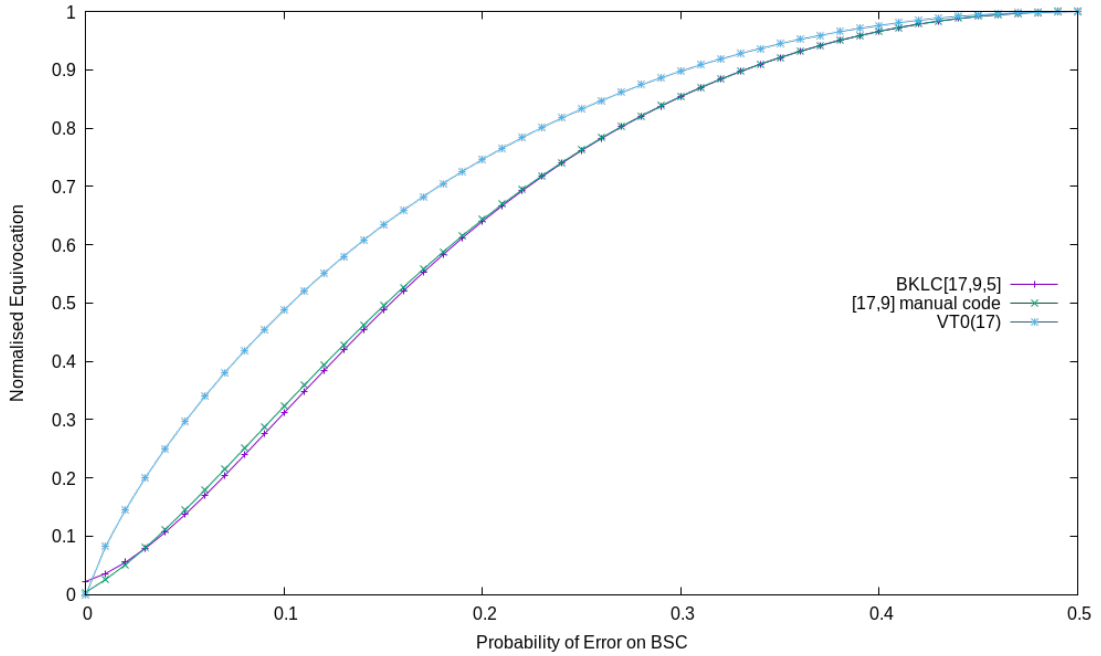


Figure 7.8: Comparison for Varshamov-Tenengolts, manually constructed and BKLCs of length 17 on ID+BSC

The temptation to compare the BSC+ID equivocation of BKLCs of length n and $VT_0(n)$ codes is not necessarily a constructive one, as the BKLC code may not provide re-

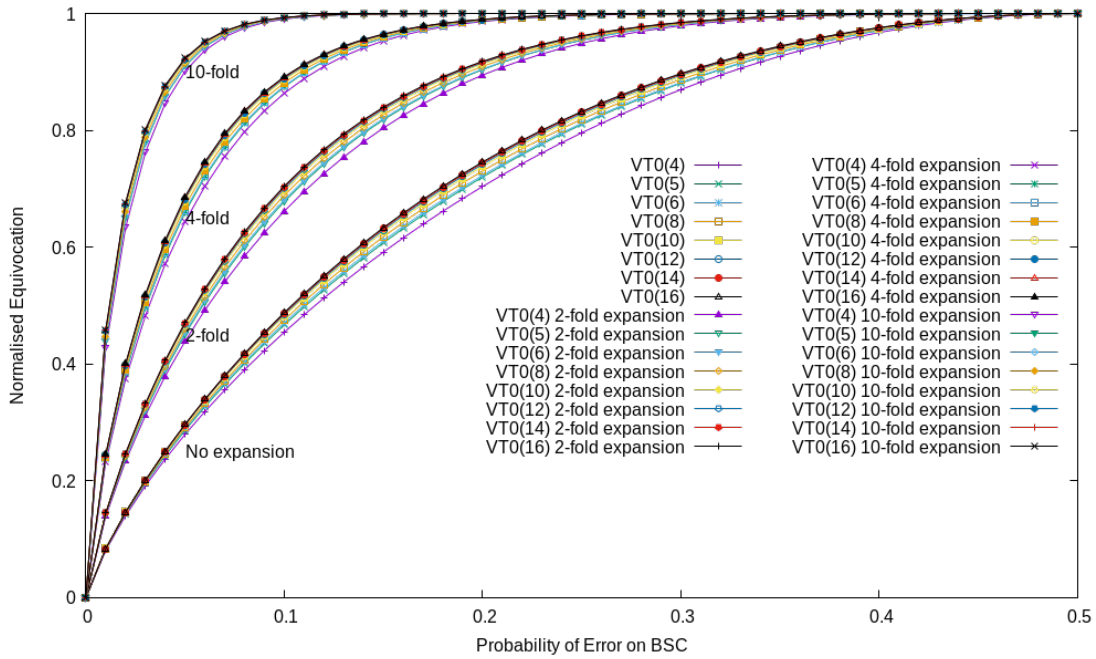


Figure 7.9: Equivocation of Varshamov-Tenengolts codes on ID+BSC with 2, 4 and 10-fold code expansion

silience against a single deletion. However it is worth comparing a VT_0 code that can carry k bits of information with a BKLC with message length k that offers resistance to a deletion when $p_e = 0$. For example, consider $VT_0(7)$ which has 16 codewords, can transmit 4 information bits and is resilient to 1 deletion. This can be compared to BKLCs with $k = 4$ (and which therefore also can transmit 4 information bits) of increasing codeword length n.

A graph of the equivocation of BKLCs on the ID+BSC as n increases is shown in Figure 7.10. Closer examination of the equivocation values when $p_e = 0$ for each code gives us Table 7.11.

Unlike the Varshamov Tenengolts codes (and specifically the $VT_0(7)$ code), whose codewords always yield unambiguous codeword stubs and an equivocation of zero when $p_e = 0$, BKLCs with $k = 4$ (and therefore 16 codewords) do not achieve an equivocation of zero until a code length of at least $n = 14$ is reached. A comparison of the $VT_0(7)$ codes with BKLCs that have $k = 4$ and an equivocation of zero when $p_e = 0$ is in Figure 7.11.

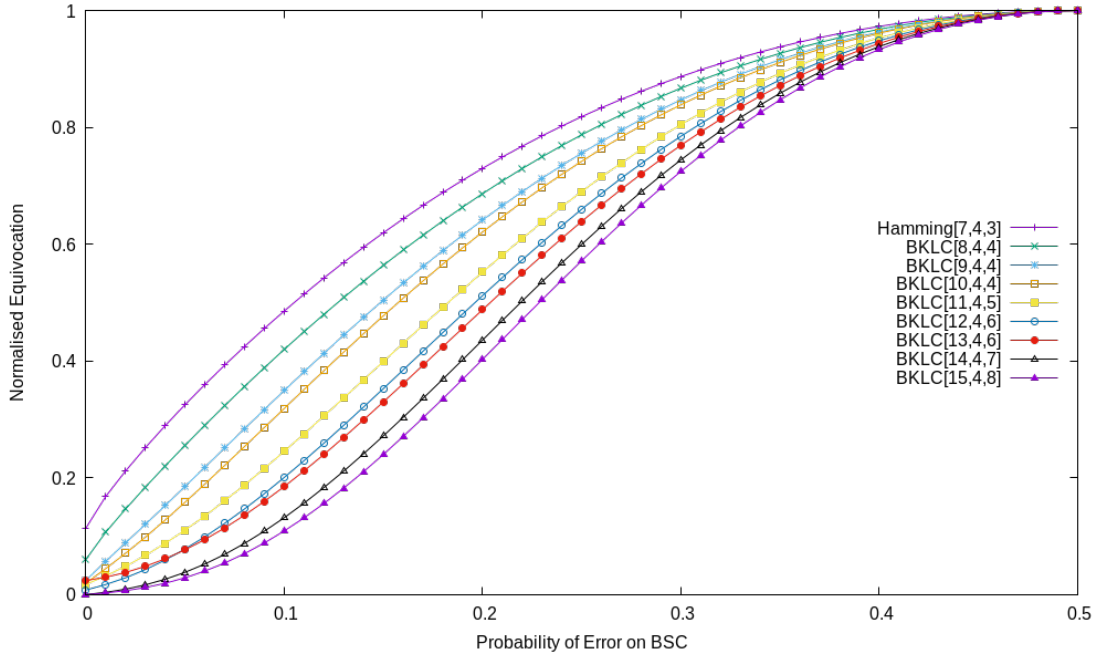


Figure 7.10: Equivocation of BKLCs with $k = 4$ on ID+BSC

Code	Normalized equivocation $H_N(X\|Y)$
Hamming $[7,4,3]$	0.113883
BKLC $[8,4,4]$	0.060068
BKLC $[9,4,4]$	0.024306
BKLC $[10,4,4]$	0.019516
BKLC $[11,4,5]$	0.017421
BKLC $[12,4,6]$	0.007813
BKLC $[13,4,6]$	0.024038
BKLC $[14,4,7]$	0.000000
BKLC $[15,4,8]$	0.000000
BKLC $[16,4,8]$	0.000000

Table 7.11: Normalized equivocation values for ID+BSC for BKLCs, $k = 4$ when $p_e = 0$

The $VT_0(7)$ code achieves its designed aim of providing effective tolerance of deletions far more efficiently, with codewords half the length needed for a BKLC. In addition, $VT_0(7)$ also yields higher equivocation values than any of the BKLCs that manage to offer an equivocation of zero when $p_e = 0$.

7.2.8 Chapter Conclusions

- In subsection 7.2.2, it was seen that Best Known Linear Codes that are selected to maximise the minimum distance for any given value of n and k are not nec-

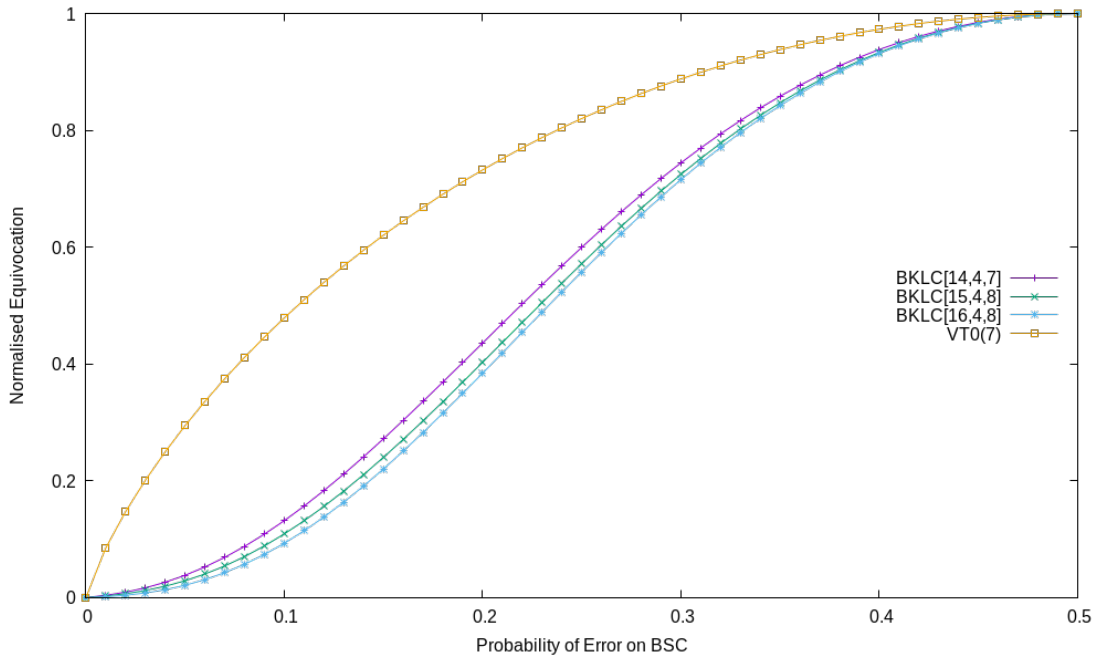


Figure 7.11: Comparison of $VT_0(7)$ with BKLCs with $k = 4$ on ID+BSC

essarily good codes for using when deletions are involved. Neither the "perfect" Hamming$[7,4,3]$ nor Hamming$[15,11,3]$ cannot reliably correct even a single deletion on ID+BSC to give an equivocation of 0 when $p_e = 0$.

- Codes that cannot correct a single intentional deletion always retain some ambiguity in their decoding method and do not give an equivocation of zero when p_e=0. They are therefore are of little practical use on the ID+BSC.

- subsection 7.2.3 showed that Varshamov-Tenengolts $VT_0(n)$ codes provide a set of codes that eliminate ambiguities between codewords stubs when a single deletion in involved. $VT_0(n)$ codes offered higher levels of equivocation than the reliably correcting BKLCs examined on the ID+BSC. For example, for $p_e = 0.1$, the 3 BKLC codes in Figure 7.11 (in which all codes convey 4 bits of information) had normalized equivocation values of 0.093, 0.109 and 0.132, compared to $VT_0(7)$ which had a much higher value of 0.479.

- Varshamov-Tenengolts codes can generally offer the ability to correct a deletion for a higher code rate ($R = k/n$) and lower redundancy than a BKLC that carries

the same amount of information. Again for the codes in Figure 7.11, the BKLCs had code rates of 0.286 (4/14), 0.2667 (4/15) and 0.25 (4/16) whereas $VT_0(7)$ had the much more efficient code rate of 0.571 (4/7).

- In subsection 7.2.4, it was shown that codes to correct any number of deletions can be designed, based on Varshamov-Tenengolts code, despite the modified codes no longer satisfying $\sum_{i=1}^{n} ix_i \equiv a(\text{mod}(n+1))$. For example, the 2-fold expanded Varshamov-Tenengolts code, $VT_0^2(4)$, is resistant to 3 deletions.

- If $m \geq 2$, $n = 2^m - 1$, $k = n - m$, then $|VT_0(n)| = |\text{Ham}[n,k,3]|$

Chapter 8

Conclusion

This doctoral work has enabled an investigation of the equivocation properties of many different types of code and transmission channel. It was partially achieved through the use of a novel mechanism for calculating the equivocation - parallel processing via the CUDA architecture. Key points developed within the work include:

- In subsection 5.3.2, it was seen that parallel processing can be effectively implemented on a general purpose computer using the Nvidia CUDA architecture. Developing the code to implement the parallel processed component of the program was both time consuming and challenging but enabled the generation of results for longer codes. ı
- Parallel processing can provide significant improvements in calculation times for intensive calculations. The best improvements obtained gave calculation results up to 35 times more quickly with parallel processing than with linear processing.
- Parallel processing code must be carefully implemented in order to optimise its efficiency for the actual task. In particular careful memory management is necessary to enable the GPU to continue operate its core duties effectively whilst simultaneously carrying out the parallel processed component of the calculation.
- Section 5.4 shows that normalised equivocation values can be used to compare the relative secrecies of codes, enabling codes with higher inherent levels of secrecy to be selected if required, although the other properties of the codes, such as the error correcting capability must also be borne in mind. For example, from

Figure 5.3, when operating with a BSC error probability of 0.05, Golay$[23,12,7]$ has a normalised equivocation of 0.033, whereas Hamming$[31,26,3]$ has a value of 0.221, nearly 7 times higher. Based on the codes' normalised equivocation values for that error probability, the Hamming code offers a higher level of secrecy than the Golay$[23,12,7]$ code despite having a lesser error correcting capability.

- For the codes examined from the Hamming code family, longer codes tended to have higher normalised equivocation values than the shorter codes. For example, with $p_e = 0.05$, the normalised equivocation of Hamming$[7,4,3]$ is 0.101 compared to 0.157 for Hamming$[15,11,3]$ and 0.221 for Hamming$[31,26,3]$. However, the complexity of the calculations to find equivocation values for longer codes meant that it wasn't possible to verify this for longer Hamming codes such as Hamming$[63,57,3]$ or Hamming$[127,120,3]$. Further work is required to establish if the pattern holds for either longer Hamming codes or for other families of code.

- Largest differences in code equivocation values were often found in the more central range of the error probabilities, for the codes investigated (especially for expanded codes). This can be seen by identifying the steepest part of the curve in any of the normalised equivocation graphs from Figure 5.3 to Figure 5.13.

- For the codes examined, differences in normalised equivocation values between codes decrease as code lengths increase. For example, in Figure 5.6 with longer length BKLCs, the similarity in value of all 7 functions can be seen quite clearly, whereas the differences between the shorter codes in Figure 5.3 are signficantly more marked.

- In Section 5.5, it was shown that by expanding a simple code through the use of random data and parity check bits, much greater levels of secrecy can be achieved in some cases. For example, with a BSC error probability $p_e = 0.05$, a 10-fold expansion of the Hamming$[7,4,3]$ code gives a normalised equivocation level of 0.897, compared to 0.101 for the non-expanded code. However, there is a trade-off between achieving the desired level of secrecy by expanding a sim-

ple code and the significant increase in the number of bits of data that need to be transmitted in order to achieve it. A 10-fold code expansion of the Hamming$[7,4,3]$ code may enable very high levels of equivocation but would require the transmission of 70 bits of data to send a 4-bit message.

- One of the aims of the work was not to design codes with improved error-correcting capabilities but to identify codes that either a) give a higher equivocation value and level of secrecy than other similar or related codes or b) enable a large differential in equivocation level to be established between the legitimate recipient and the eavesdropper. By expanding random codes, higher equivocation values can be achieved than for more structured but non-expanded codes. For example, with a BSC error probability $p_e = 0.05$, a 4-fold expansion of a random $[23,12]$ code gives a normalised equivocation of 0.597, compared to a non-expanded Golay$[23,12,7]$ value of 0.033. However, although the inherent randomness of the codes increased equivocation levels, this is of little benefit given that the codes offer no error correcting capability.

- The expansion of a code can offer higher normalised equivocation values than a code with a similar net code rate. For example, consider the Hamming$[7,4,3]$ code with 2-fold expansion and the BKLC$[14,4,7]$ code. Both effectively require 14 bits to transmit 4 bits worth of message data, however for a BSC error probability of $p_e = 0.05$, the 2-fold expanded Hamming code gives a normalised equivocation of 0.259, while the non-expanded Hamming code gave a value of 0.101 and the BKLC$[14,4,7]$ only gives a value of 0.038.

- Increases in equivocation from the introduction of intentional erasures on the BSC are significant, but not as great as those increases shown by the use of code expansions. For example, from Figure 6.7 in subsection 6.2.2, for the Golay$[23,12,7]$ code with $p_e = 0.05$, 6 erasures give a normalised equivocation of 0.178 while 4-fold expansion gives a much higher value of 0.540.

- Some increases in the number of bits erased on the IE+BSC do not necessarily

lead to an increase in equivocation. This was seen with BKLC$[8,2,5]$, where the increase from 3 to 4 erasures did not change the equivocation. This was because the increase in erasures didn't change the ability of the code to use the available syndrome patterns for the detection or correction of either errors or erasures.

- The controlled introduction of erasures to a BSC can potentially lead to a greater increase in equivocation for an eavesdropper than it does for the legitimate receiver. The introduction of 2 erasures to the Golay$[23,12,7]$ code on the IE+BSC gave an increase in normalised equivocation of 0.00132 for the legitimate receiver with $p_e = 0.05$ but a much larger increase of 0.08898 for an eavesdropper with $p_e = 0.1$.

- A comparison between code expansions and erasures can show situations where, for a particular error probability, there is a changeover point in which method produces the higher equivocation values. It was seen that for Golay $[23,12,7]$ with $p_e < 0.05$, 6 erasures gave a higher equivocation value, whereas for $p_e > 0.05$, 2-fold code expansion gave higher values.

- On the BEC in Section 6.3, even codes with similar properties or similar lengths can have wildly different equivocation characteristics. Figure 6.13 shows that with $p_s = 0.2$, BKLC$[15,10,4]$ had a normalised equivocation of just 0.0733, whereas BKLC$[15,12,2]$ had a value of 0.3581.

- The correction and decoding mechansim used for the BEC in subsection 6.3.2 caused a reduction in the possible outcomes and a decreasing output entropy as the probability of an erasure increased. As $p_s \to 1$, $H(Y) \to 0$. In extremis, when all bits are erased, it becomes a certainty that the zero codeword will be selected as the decode.

- In subsection 7.2.2, it was seen that Best Known Linear Codes that are selected to maximise the minimum distance for any given value of n and k are not necessarily good codes for using when deletions are involved. Neither the "perfect"

Hamming$[7,4,3]$ nor Hamming$[15,11,3]$ cannot reliably correct even a single deletion on ID+BSC to give an equivocation of 0 when $p_e = 0$.

- Codes that cannot correct a single intentional deletion always retain some ambiguity in their decoding method and do not give an equivocation of zero when p_e=0. They are therefore are of little practical use on the ID+BSC.

- subsection 7.2.3 showed that Varshamov-Tenengolts $VT_0(n)$ codes provide a set of codes that eliminate ambiguities between codewords stubs when a single deletion in involved. $VT_0(n)$ codes offered higher levels of equivocation than the reliably correcting BKLCs examined on the ID+BSC. For example, for $p_e = 0.1$, the 3 BKLC codes in Figure 7.11 (in which all codes convey 4 bits of information) had normalized equivocation values of 0.093, 0.109 and 0.132, compared to $VT_0(7)$ which had a much higher value of 0.479.

- Varshamov-Tenengolts codes can generally offer the ability to correct a deletion for a higher code rate ($R = k/n$) and lower redundancy than a BKLC that carries the same amount of information. Again for the codes in Figure 7.11, the BKLCs had code rates of 0.286 (4/14), 0.2667 (4/15) and 0.25 (4/16) whereas $VT_0(7)$ had the much more efficient code rate of 0.571 (4/7).

- In subsection 7.2.4, it was shown that codes to correct any number of deletions can be designed, based on Varshamov-Tenengolts code, despite the modified codes no longer satisfying
$\sum_{i=1}^{n} ix_i \equiv a(\text{mod}(n+1))$. For example, the 2-fold expanded Varshamov-Tenengolts code, $VT_0^2(4)$, is resistant to 3 deletions.

- If $m \geq 2$, $n = 2^m - 1$, $k = n - m$, then $|VT_0(n)| = |\text{Ham}[n,k,3]|$.

8.1 Discussion

This programme of Ph.D. study was started immediately following successful completion of an Open University degree in computing and with a historical background of a

pass degree in mathematics some 20 years previously. The key skills required during the period of research included:

- the development of an understanding of the broader aspects of coding theory
- a specific understanding of equivocation
- sufficient ability to program software solutions for calculating equivocation
- the perseverance to work through the many issues encountered along the way.

The material covered during the OU computing degree formed a broad introduction to some aspects of coding theory such as the error-correcting capabilities of Hamming codes, however this was in very little detail, except to have been sufficient to encourage the author to make further steps into the field. Before the PhD. was started, the author had no knowledge of equivocation.

Similarly, some work on C++ and object-oriented programming in Java had been completed during the OU degree programme. However as the doctorate work progressed, it became increasingly apparent how much there was to learn in order to construct programs that provided solutions to the necessary calculations in anything approaching a time-effective manner.

Consequently, a huge amount of time and effort has been expended on bringing these skills to a higher standard. It is particularly interesting to look back through the various iterations of the programs produced to see how the programming methods have changed and improved en route.

As examples,

- **NTL library**. At the outset, significant time was spent learning how to work with Shoup's C++ Number Theory Library (NTL). The aim of this was to enable operation with very large numbers through mechanisms such as the **ZZ** class for arbitrary length integers, **GF2** for integers mod 2 and the **GF2X** class for poly-

nomicals over **GF**2. The manipulation of these classes and passing of values between them took a long time to become used to.

Unfortunately, as the research progressed and became more focussed towards the calculation of equivocation on different channel arrangements, it became apparent that the ability to handle large, long numbers was far less important than the ability to handle smaller numbers very quickly. Eventually use of the NTL library was dropped in favour of simpler mechanisms that were felt to handle smaller codes more efficiently for the task. As an example in the first, linear version of the program to calculate equivocation on the BSC, the function below was used to generate the next binary number:

```
vec_GF2 getNextBin(vec_GF2 vecA)
{
    carry = 1;
    posn = k -1;
    vecA[posn] += 1;
    while ( (carry == 1) && (posn > 0) )
    {
        if (vecA[posn] == 0)
        {
            carry = 1;
        }
        else
            carry = 0;
        vecA[posn-1] += carry;
        posn-;
    }
    return vecA;
}
```

However, once the change away from using NTL had been made in favour of using bitwise manipulation of numbers stored as integers, this reduced back to the triviality of a numerical increment:

```
num++;
```

- **Bit-shifting**. For a large proportion of the study period, n-bit binary numbers were first handled using the `vector` and then using pointers with either the `int` or `long` number types to hold each single bit. Eventually this approach was dropped and the whole n-bit number was stored as a single `int` or `long`. This was then manipulated using left- and right- bit-shifting to multiply and divide by powers of 2 and the logical operations & (AND), | (OR) and ˆ(XOR) to enable functions such as matrix multiplication. Not only did this significantly reduce the amount of memory needed to store the variables, but it also significantly increased the speed of calculation and hence the length of codes for which the equivocation could be found. An example of this approach to coding has already been seen in Figure 7.2 on page 151.

- **Balance between speed (calculation time) and memory**. The calculations done during the research were very intensive, increasing exponentially with the length of both the message and the codeword. This constantly pushed at the boundaries of what a 'standard' computer was able to achieve, both in terms of the processing speed and capability and the memory requirements. For a calculation to run quickly, there was often a need to store volumes of data beyond the capacity of the computer, so compromises had to be found for each calculation.

- **CUDA architecture**. The use of Nvidia's CUDA architecture was viewed as a viable mechanism for reducing calculation times. By employing the graphics card for parallel processing the most repetitive components of calculations, significant time savings were able to be obtained, fulfilling the primary purpose of CUDA. This was seen in Table 5.8 where parallel processed times were up to 35 times

quicker than those for linear processing.

However implementing the software within the CUDA architecture came at a very great time cost to the research programme. Where a working linear version of a program could be written increasingly quickly, converting this across to work as a parallel processed equivalent with CUDA took considerable time and effort. The first working, parallel-processed version of the program to calculate equivocation on the BSC took several months to complete. CUDA kernels have their own instruction set and do not easily permit the use of external libraries. The use of NTL within the CUDA environment was never able to be resolved and was another major reason for discontinuing the use of NTL. These time penalties in learning such programming techniques and pitfalls had a massive, dominating impact on the overall amount of visible outputs from this research project.

- **GPU limitations**. The CUDA architecture is specifically designed to enable the use of a Nvidia GPU within a standard computer to perform parallel calculations. However in practice, this was found to present significant problems.

  - **Usability**. When running an intensive CUDA program, most of the unit's processing capability is given over to the calculation. This leads the display(s) to react and refresh much more slowly and all responses to inputs such as mouse movement and keyboard input are severely delayed. This usually rendered the computer unusable for any other tasks whilst the CUDA program was running. If the program was on an extended run (e.g. the calculation of equivocation on the ID+BSC for BKLC $[19, 7, 8]$ for 50 probabilities in 0.01 intervals took in excess of 36 hours), then no other work could be done on the computer during that time.

  - **Stability**. The extensive but necessary use of memory to store mid-calculation values and the intensive use of the GPU led to severe stability issues for the computer. The computer would often 'hang' and become completely unusable. Any changes to the physical set-up of the computer e.g. (un)plugging

a second monitor would usually trigger an instability. Re-starting would not necessarily clear the issues and caused immense frustration and loss of study time. When the computer was working correctly, it was generally left powered on and running for days or weeks at a time to minimize any risks from changes. Whilst it was believed that most of the issues emanated from the atypical use of the GPU and heavy use of memory, no satisfactory memory management solutions were found. Once the majority of the parallel processing phase of the work had been completed, the (Ubuntu 14.04 LTS) computer gradually re-acquired good stability.

8.2 Further areas of Study

In summary, the field has proven an interesting one to study as a Ph.D student and one that would bear significant further study. There was relatively little research to draw on regarding the actual evaluation of equivocation for different channels, especially when extending to the Binary Erasure and Binary Deletion Channels. This doctoral work has opened up numerous possible avenues for further research.

1. Investigation of other decoding strategies that yield a decreasing output entropy $H(Y)$ as encountered in Section 6.3.

2. Extension of the work in Chapter 7 to include multiple deletions.

3. Further examination of the equivocation properties of Varshamov-Tenengolts codes. Whilst VT codes work very well for single-deletion situations, processing limitations prevented the actual calculation of equivocation values for multi-deletion resistant codes. It would be interesting to pursue this further.

4. Calculation of equivocation on other channels, including the Binary Deletion Channel (BDC) and Binary Symmetric Erasure Channel (BSEC). These two channels would form a very natural extension to the work already covered and have already been discussed to some extent. It is anticipated that calculations for the BSEC would require little additional modification of existing software and would

yield results for similar length codes to those already obtained. However, although addressing the BDC would require few modifications to existing software solutions, the additional complexity that deletions bring in comparison to erasures would only make it possible to obtain results for quite short codes. In addition to the channels already discussed, other channels might be considered such as the Additive White Gaussian Noise (AWGN) channel or Fading channels. The AWGN channel is a model that mimics the impact of noise due to natural processes by adding further noise to the intrinsic noise of an existing information system. Gaussian noise is normally distributied with respect to time and White noise possesses a uniform power level across the frequency band of the system. The AWGN channel models satellite and other space-based communication links well but less so for terrestrial environments, due to other factors such as interference and multipath propagation. Fading channels model the degradation of signal quality over large distances, even without significant AWGN. Fading is more likely to be due to terrestrial influences such as multipath propagation, terrain/geography, weather phenomena etc. It is considered that the existing methods developed during this thesis would not transfer well to either the AWGN channel or Fading channels without a substantially revised approach to the calculation of conditional probabilities and equivocation values.

5. Development of further techniques to reduce calculation run-times and thereby enable calculation of equivocation for longer codes The software solutions used in this work have been shown to be effective at directly calculating equivocation values that enable comparisons to be made between different codes, but only for codes of length $n \leq 40$. Whilst further channels arrangements could be considered, the primary way of extending the calculation to longer codes would be via more efficient programming and by bringing greater processing capability to bear on the problem. Those increases might come from the methods suggested in subsection 5.4.2, however it is likely that they would only bring a few extra bits of code length, since each extra bit of code length will approximately double the

necessary processing.

6. Calculation of equivocation for other types of code. The codes examinded in this thesis were mostly either perfect codes or best known linear codes, although the work was extended to Varshamov-Tenengolts codes in order to better cope with intentional deletions on the BSC. Extensions that might be considered would be to look at Hadamard codes, Reed-Muller codes or Low Density Parity Check (LDPC) codes. As relatively straightforward binary linear codes with definable generator matrices and that are used for error detection and correction, both Hadamard and Reed-Muller codes would make good candidates for further investigation using the methods employed in this work. However, LDPC codes usually have very much larger parity check matrices (PCMs), in the order of hundreds, if not thousands, of both columns and rows. It is considered unlikely that the methods used here would be able to be extended to cope with PCMs on that scale. Similarly, Turbo Codes, which are high-performing codes that approach the channel capacity, rely on an iterative feedback process to correct errors during decoding. The additional complexity brought by the compounding of calculations caused by the feedback process would again probably render the methods used as inadequate for calculating the equivocation of turbo codes.

Appendix A

5-Bit Error Vectors in Weight Order

0	0	0	0	0
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	1	0	0	0
1	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
0	1	1	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0
1	1	1	1	1

Appendix B

BKLC$[8,2,5]$ Decode Probabilities and Joint Entropy Contributions

Transmitted codeword $x_1 = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$, $p_s = 0.01$

Received codeword	Decode	$P(y\|x_1)$	$P(x_1,y)$	$H(x_1,y)$
No erasures				
00011111	00011111	0.9227	0.2307	0.4881
1 erasure				
0001111?	00011111	0.0093	0.0023	0.0203
000111?1	00011111	0.0093	0.0023	0.0203
00011?11	00011111	0.0093	0.0023	0.0203
0001?111	00011111	0.0093	0.0023	0.0203
000?1111	00011111	0.0093	0.0023	0.0203
00?11111	00011111	0.0093	0.0023	0.0203
0?011111	00011111	0.0093	0.0023	0.0203
?0011111	00011111	0.0093	0.0023	0.0203
2 erasures				
000111??	00011111	9.4148E-05	2.3537E-05	3.6187E-04
00011?1?	00011111	9.4148E-05	2.3537E-05	3.6187E-04
00011??1	00011111	9.4148E-05	2.3537E-05	3.6187E-04
0001?11?	00011111	9.4148E-05	2.3537E-05	3.6187E-04

Received codeword	Decode	$P(y\|x_1)$	$P(x_1,y)$	$H(x_1,y)$
0001?1?1	00011111	9.4148E-05	2.3537E-05	3.6187E-04
0001??11	00011111	9.4148E-05	2.3537E-05	3.6187E-04
000?111?	00011111	9.4148E-05	2.3537E-05	3.6187E-04
000?11?1	00011111	9.4148E-05	2.3537E-05	3.6187E-04
000?1?11	00011111	9.4148E-05	2.3537E-05	3.6187E-04
000??111	00011111	9.4148E-05	2.3537E-05	3.6187E-04
00?1111?	00011111	9.4148E-05	2.3537E-05	3.6187E-04
00?111?1	00011111	9.4148E-05	2.3537E-05	3.6187E-04
00?11?11	00011111	9.4148E-05	2.3537E-05	3.6187E-04
00?1?111	00011111	9.4148E-05	2.3537E-05	3.6187E-04
00??1111	00011111	9.4148E-05	2.3537E-05	3.6187E-04
0?01111?	00011111	9.4148E-05	2.3537E-05	3.6187E-04
0?0111?1	00011111	9.4148E-05	2.3537E-05	3.6187E-04
0?011?11	00011111	9.4148E-05	2.3537E-05	3.6187E-04
0?01?111	00011111	9.4148E-05	2.3537E-05	3.6187E-04
0?0?1111	00011111	9.4148E-05	2.3537E-05	3.6187E-04
0??11111	00011111	9.4148E-05	2.3537E-05	3.6187E-04
?001111?	00011111	9.4148E-05	2.3537E-05	3.6187E-04
?0011?1	00011111	9.4148E-05	2.3537E-05	3.6187E-04
?0011?11	00011111	9.4148E-05	2.3537E-05	3.6187E-04
?001?111	00011111	9.4148E-05	2.3537E-05	3.6187E-04
?00?1111	00011111	9.4148E-05	2.3537E-05	3.6187E-04
?0?11111	00011111	9.4148E-05	2.3537E-05	3.6187E-04
??011111	00011111	9.4148E-05	2.3537E-05	3.6187E-04
3 erasures				
00011???	00011111	9.5099E-07	2.3774E-07	5.2314E-006
0001?1??	00011111	9.5099E-07	2.3774E-07	5.2314E-006

Received codeword	Decode	$P(y\|x_1)$	$P(x_1,y)$	$H(x_1,y)$
0001??1?	00011111	9.5099E-07	2.3774E-07	5.2314E-006
0001???1	00011111	9.5099E-07	2.3774E-07	5.2314E-006
000?11??	00011111	9.5099E-07	2.3774E-07	5.2314E-006
000?1?1?	00011111	9.5099E-07	2.3774E-07	5.2314E-006
000?1??1	00011111	9.5099E-07	2.3774E-07	5.2314E-006
000??11?	00011111	9.5099E-07	2.3774E-07	5.2314E-006
000??1?1	00011111	9.5099E-07	2.3774E-07	5.2314E-006
000???11	00011111	9.5099E-07	2.3774E-07	5.2314E-006
00?111??	00011111	9.5099E-07	2.3774E-07	5.2314E-006
00?11?1?	00011111	9.5099E-07	2.3774E-07	5.2314E-006
00?11??1	00011111	9.5099E-07	2.3774E-07	5.2314E-006
00?1?11?	00011111	9.5099E-07	2.3774E-07	5.2314E-006
00?1?1?1	00011111	9.5099E-07	2.3774E-07	5.2314E-006
00?1??11	00011111	9.5099E-07	2.3774E-07	5.2314E-006
00??111?	00011111	9.5099E-07	2.3774E-07	5.2314E-006
00??11?1	00011111	9.5099E-07	2.3774E-07	5.2314E-006
00??1?11	00011111	9.5099E-07	2.3774E-07	5.2314E-006
00???111	00011111	9.5099E-07	2.3774E-07	5.2314E-006
0?0111??	00011111	9.5099E-07	2.3774E-07	5.2314E-006
0?011?1?	00011111	9.5099E-07	2.3774E-07	5.2314E-006
0?011??1	00011111	9.5099E-07	2.3774E-07	5.2314E-006
0?01?11?	00011111	9.5099E-07	2.3774E-07	5.2314E-006
0?01?1?1	00011111	9.5099E-07	2.3774E-07	5.2314E-006
0?01??11	00011111	9.5099E-07	2.3774E-07	5.2314E-006
0?0?111?	00011111	9.5099E-07	2.3774E-07	5.2314E-006
0?0?11?1	00011111	9.5099E-07	2.3774E-07	5.2314E-006
0?0?1?11	00011111	9.5099E-07	2.3774E-07	5.2314E-006

Received codeword	Decode	$P(y\|x_1)$	$P(x_1,y)$	$H(x_1,y)$
0?0??111	00011111	9.5099E-07	2.3774E-07	5.2314E-006
0??1111?	00011111	9.5099E-07	2.3774E-07	5.2314E-006
0??111?1	00011111	9.5099E-07	2.3774E-07	5.2314E-006
0??11?11	00011111	9.5099E-07	2.3774E-07	5.2314E-006
0??1?111	00011111	9.5099E-07	2.3774E-07	5.2314E-006
0???1111	00011111	9.5099E-07	2.3774E-07	5.2314E-006
?0011???	00011111	9.5099E-07	2.3774E-07	5.2314E-006
?0011?1?	00011111	9.5099E-07	2.3774E-07	5.2314E-006
?0011??1	00011111	9.5099E-07	2.3774E-07	5.2314E-006
?001?11?	00011111	9.5099E-07	2.3774E-07	5.2314E-006
?001?1?1	00011111	9.5099E-07	2.3774E-07	5.2314E-006
?001??11	00011111	9.5099E-07	2.3774E-07	5.2314E-006
?00?111?	00011111	9.5099E-07	2.3774E-07	5.2314E-006
?00?11?1	00011111	9.5099E-07	2.3774E-07	5.2314E-006
?00?1?11	00011111	9.5099E-07	2.3774E-07	5.2314E-006
?00??111	00011111	9.5099E-07	2.3774E-07	5.2314E-006
?0?1111?	00011111	9.5099E-07	2.3774E-07	5.2314E-006
?0?111?1	00011111	9.5099E-07	2.3774E-07	5.2314E-006
?0?11?11	00011111	9.5099E-07	2.3774E-07	5.2314E-006
?0?1?111	00011111	9.5099E-07	2.3774E-07	5.2314E-006
?0??1111	00011111	9.5099E-07	2.3774E-07	5.2314E-006
??01111?	00011111	9.5099E-07	2.3774E-07	5.2314E-006
??0111?1	00011111	9.5099E-07	2.3774E-07	5.2314E-006
??011?11	00011111	9.5099E-07	2.3774E-07	5.2314E-006
??01?111	00011111	9.5099E-07	2.3774E-07	5.2314E-006
??0?1111	00011111	9.5099E-07	2.3774E-07	5.2314E-006
???11111	00011111	9.5099E-07	2.3774E-07	5.2314E-006

Received codeword	Decode	$P(y\|x_1)$	$P(x_1,y)$	$H(x_1,y)$
4 erasures				
0001????	00011111	9.6060E-009	2.4015E-009	6.8763E-008
000?1???	00011111	9.6060E-009	2.4015E-009	6.8763E-008
000??1??	00011111	9.6060E-009	2.4015E-009	6.8763E-008
000???1?	00011111	9.6060E-009	2.4015E-009	6.8763E-008
000????1	00011111	9.6060E-009	2.4015E-009	6.8763E-008
00?11???	00011111	9.6060E-009	2.4015E-009	6.8763E-008
00?1?1??	00011111	9.6060E-009	2.4015E-009	6.8763E-008
00?1??1?	00011111	9.6060E-009	2.4015E-009	6.8763E-008
00?1???1	00011111	9.6060E-009	2.4015E-009	6.8763E-008
00??11??	00011111	9.6060E-009	2.4015E-009	6.8763E-008
00??1?1?	00011111	9.6060E-009	2.4015E-009	6.8763E-008
00??1??1	00011111	9.6060E-009	2.4015E-009	6.8763E-008
00???11?	00011111	9.6060E-009	2.4015E-009	6.8763E-008
00???1?1	00011111	9.6060E-009	2.4015E-009	6.8763E-008
00????11	00011111	9.6060E-009	2.4015E-009	6.8763E-008
0?011???	00011111	9.6060E-009	2.4015E-009	6.8763E-008
0?01?1??	00011111	9.6060E-009	2.4015E-009	6.8763E-008
0?01??1?	00011111	9.6060E-009	2.4015E-009	6.8763E-008
0?01???1	00011111	9.6060E-009	2.4015E-009	6.8763E-008
0?0?11??	00011111	9.6060E-009	2.4015E-009	6.8763E-008
0?0?1?1?	00011111	9.6060E-009	2.4015E-009	6.8763E-008
0?0?1??1	00011111	9.6060E-009	2.4015E-009	6.8763E-008
0?0??11?	00011111	9.6060E-009	2.4015E-009	6.8763E-008
0?0??1?1	00011111	9.6060E-009	2.4015E-009	6.8763E-008
0?0???11	00011111	9.6060E-009	2.4015E-009	6.8763E-008
000111??	00011111	9.6060E-009	2.4015E-009	6.8763E-008

Received codeword	Decode	$P(y\|x_1)$	$P(x_1,y)$	$H(x_1,y)$
0??11?1?	00011111	9.6060E-009	2.4015E-009	6.8763E-008
0??11??1	00011111	9.6060E-009	2.4015E-009	6.8763E-008
0??1?11?	00011111	9.6060E-009	2.4015E-009	6.8763E-008
0??1?1?1	00011111	9.6060E-009	2.4015E-009	6.8763E-008
0??1??11	00011111	9.6060E-009	2.4015E-009	6.8763E-008
0???111?	00011111	9.6060E-009	2.4015E-009	6.8763E-008
0???11?1	00011111	9.6060E-009	2.4015E-009	6.8763E-008
0???1?11	00011111	9.6060E-009	2.4015E-009	6.8763E-008
0????111	00011111	9.6060E-009	2.4015E-009	6.8763E-008
?0011???	00011111	9.6060E-009	2.4015E-009	6.8763E-008
?001?1??	00011111	9.6060E-009	2.4015E-009	6.8763E-008
?001??1?	00011111	9.6060E-009	2.4015E-009	6.8763E-008
?001???1	00011111	9.6060E-009	2.4015E-009	6.8763E-008
?00?11??	00011111	9.6060E-009	2.4015E-009	6.8763E-008
?00?1?1?	00011111	9.6060E-009	2.4015E-009	6.8763E-008
?00?1??1	00011111	9.6060E-009	2.4015E-009	6.8763E-008
?00??11?	00011111	9.6060E-009	2.4015E-009	6.8763E-008
?00??1?1	00011111	9.6060E-009	2.4015E-009	6.8763E-008
?00???11	00011111	9.6060E-009	2.4015E-009	6.8763E-008
?0?111??	00011111	9.6060E-009	2.4015E-009	6.8763E-008
?0?11?1?	00011111	9.6060E-009	2.4015E-009	6.8763E-008
?0?11??1	00011111	9.6060E-009	2.4015E-009	6.8763E-008
?0?1?11?	00011111	9.6060E-009	2.4015E-009	6.8763E-008
?0?1?1?1	00011111	9.6060E-009	2.4015E-009	6.8763E-008
?0?1??11	00011111	9.6060E-009	2.4015E-009	6.8763E-008
?0??111?	00011111	9.6060E-009	2.4015E-009	6.8763E-008
?0??11?1	00011111	9.6060E-009	2.4015E-009	6.8763E-008

Received codeword	Decode	$P(y\|x_1)$	$P(x_1,y)$	$H(x_1,y)$
?0??1?11	00011111	9.6060E-009	2.4015E-009	6.8763E-008
?0???111	00011111	9.6060E-009	2.4015E-009	6.8763E-008
??0111??	00011111	9.6060E-009	2.4015E-009	6.8763E-008
??011?1?	00011111	9.6060E-009	2.4015E-009	6.8763E-008
??011??1	00011111	9.6060E-009	2.4015E-009	6.8763E-008
??01?11?	00011111	9.6060E-009	2.4015E-009	6.8763E-008
??01?1?1	00011111	9.6060E-009	2.4015E-009	6.8763E-008
??01??11	00011111	9.6060E-009	2.4015E-009	6.8763E-008
??0?111?	00011111	9.6060E-009	2.4015E-009	6.8763E-008
??0?11?1	00011111	9.6060E-009	2.4015E-009	6.8763E-008
??0?1?11	00011111	9.6060E-009	2.4015E-009	6.8763E-008
??0??111	00011111	9.6060E-009	2.4015E-009	6.8763E-008
???1111?	00011111	9.6060E-009	2.4015E-009	6.8763E-008
???111?1	00011111	9.6060E-009	2.4015E-009	6.8763E-008
????11?1	00011111	9.6060E-009	2.4015E-009	6.8763E-008
???1?111	00011111	9.6060E-009	2.4015E-009	6.8763E-008
????1111	00011111	9.6060E-009	2.4015E-009	6.8763E-008
5 erasures				
000?????	00000000	9.7030E-011	2.4257E-011	8.5539E-010
00?1????	00011111	9.7030E-011	2.4257E-011	8.5539E-010
00??1???	00011111	9.7030E-011	2.4257E-011	8.5539E-010
00???1??	00011111	9.7030E-011	2.4257E-011	8.5539E-010
00????1?	00011111	9.7030E-011	2.4257E-011	8.5539E-010
00?????1	00011111	9.7030E-011	2.4257E-011	8.5539E-010
0?01????	00011111	9.7030E-011	2.4257E-011	8.5539E-010
0?0?1???	00011111	9.7030E-011	2.4257E-011	8.5539E-010
0?0??1??	00011111	9.7030E-011	2.4257E-011	8.5539E-010

Received codeword	Decode	$P(y\|x_1)$	$P(x_1,y)$	$H(x_1,y)$
0?0???1?	00011111	9.7030E-011	2.4257E-011	8.5539E-010
0?0????1	00011111	9.7030E-011	2.4257E-011	8.5539E-010
0??11???	00011111	9.7030E-011	2.4257E-011	8.5539E-010
0??1?1??	00011111	9.7030E-011	2.4257E-011	8.5539E-010
0??1??1?	00011111	9.7030E-011	2.4257E-011	8.5539E-010
0??1???1	00011111	9.7030E-011	2.4257E-011	8.5539E-010
0???11??	00011111	9.7030E-011	2.4257E-011	8.5539E-010
0???1?1?	00011111	9.7030E-011	2.4257E-011	8.5539E-010
0???1??1	00011111	9.7030E-011	2.4257E-011	8.5539E-010
0????11?	00011111	9.7030E-011	2.4257E-011	8.5539E-010
0????1?1	00011111	9.7030E-011	2.4257E-011	8.5539E-010
0?????11	00011111	9.7030E-011	2.4257E-011	8.5539E-010
?001????	00011111	9.7030E-011	2.4257E-011	8.5539E-010
?00?1???	00011111	9.7030E-011	2.4257E-011	8.5539E-010
?00??1??	00011111	9.7030E-011	2.4257E-011	8.5539E-010
?00???1?	00011111	9.7030E-011	2.4257E-011	8.5539E-010
?00????1	00011111	9.7030E-011	2.4257E-011	8.5539E-010
?0?11???	00011111	9.7030E-011	2.4257E-011	8.5539E-010
?0?1?1??	00011111	9.7030E-011	2.4257E-011	8.5539E-010
?0?1??1?	00011111	9.7030E-011	2.4257E-011	8.5539E-010
?0?1???1	00011111	9.7030E-011	2.4257E-011	8.5539E-010
?0??11??	00011111	9.7030E-011	2.4257E-011	8.5539E-010
?0???11?	00011111	9.7030E-011	2.4257E-011	8.5539E-010
?0????1?	00011111	9.7030E-011	2.4257E-011	8.5539E-010
?0?????1	00011111	9.7030E-011	2.4257E-011	8.5539E-010
??011???	00011111	9.7030E-011	2.4257E-011	8.5539E-010
??01?1??	00011111	9.7030E-011	2.4257E-011	8.5539E-010

Received codeword	Decode	$P(y\|x_1)$	$P(x_1,y)$	$H(x_1,y)$
??01??1?	00011111	9.7030E-011	2.4257E-011	8.5539E-010
??01???1	00011111	9.7030E-011	2.4257E-011	8.5539E-010
??0?11??	00011111	9.7030E-011	2.4257E-011	8.5539E-010
??0?1?1?	00011111	9.7030E-011	2.4257E-011	8.5539E-010
??0?1??1	00011111	9.7030E-011	2.4257E-011	8.5539E-010
??0??11?	00011111	9.7030E-011	2.4257E-011	8.5539E-010
??0??1?1	00011111	9.7030E-011	2.4257E-011	8.5539E-010
??0???11	00011111	9.7030E-011	2.4257E-011	8.5539E-010
???11???	00011111	9.7030E-011	2.4257E-011	8.5539E-010
???11?1?	00011111	9.7030E-011	2.4257E-011	8.5539E-010
???11??1	00011111	9.7030E-011	2.4257E-011	8.5539E-010
???1?11?	00011111	9.7030E-011	2.4257E-011	8.5539E-010
???1?1?1	00011111	9.7030E-011	2.4257E-011	8.5539E-010
???1??11	00011111	9.7030E-011	2.4257E-011	8.5539E-010
????111?	00011111	9.7030E-011	2.4257E-011	8.5539E-010
????11?1	00011111	9.7030E-011	2.4257E-011	8.5539E-010
????1?11	00011111	9.7030E-011	2.4257E-011	8.5539E-010
?????111	00011111	9.7030E-011	2.4257E-011	8.5539E-010
6 erasures				
00??????	00000000	9.801E-013	2.4503E-013	1.0265E-011
0?0?????	00000000	9.801E-013	2.4503E-013	1.0265E-011
0??1????	00011111	9.801E-013	2.4503E-013	1.0265E-011
0???1???	00011111	9.801E-013	2.4503E-013	1.0265E-011
0????1??	00011111	9.801E-013	2.4503E-013	1.0265E-011
0?????1?	00011111	9.801E-013	2.4503E-013	1.0265E-011
0??????1	00011111	9.801E-013	2.4503E-013	1.0265E-011
?00?????	00000000	9.801E-013	2.4503E-013	1.0265E-011

Received codeword	Decode	$P(y\|x_1)$	$P(x_1,y)$	$H(x_1,y)$
?0?1????	00011111	9.801E-013	2.4503E-013	1.0265E-011
?0??1???	00011111	9.801E-013	2.4503E-013	1.0265E-011
?0???1??	00011111	9.801E-013	2.4503E-013	1.0265E-011
?0????1?	00011111	9.801E-013	2.4503E-013	1.0265E-011
?0?????1	00011111	9.801E-013	2.4503E-013	1.0265E-011
??01????	00011111	9.801E-013	2.4503E-013	1.0265E-011
??0?1???	00011111	9.801E-013	2.4503E-013	1.0265E-011
??0??1??	00011111	9.801E-013	2.4503E-013	1.0265E-011
??0???1?	00011111	9.801E-013	2.4503E-013	1.0265E-011
??0????1	00011111	9.801E-013	2.4503E-013	1.0265E-011
???11???	00011111	9.801E-013	2.4503E-013	1.0265E-011
???1?1??	00011111	9.801E-013	2.4503E-013	1.0265E-011
???1??1?	00011111	9.801E-013	2.4503E-013	1.0265E-011
???1???1	00011111	9.801E-013	2.4503E-013	1.0265E-011
????11??	00011111	9.801E-013	2.4503E-013	1.0265E-011
????1?1?	00011111	9.801E-013	2.4503E-013	1.0265E-011
????1??1	00011111	9.801E-013	2.4503E-013	1.0265E-011
?????11?	00011111	9.801E-013	2.4503E-013	1.0265E-011
?????1?1	00011111	9.801E-013	2.4503E-013	1.0265E-011
??????11	00011111	9.801E-013	2.4503E-013	1.0265E-011
7 erasures				
0???????	00000000	9.9E-015	2.475E-015	1.2009E-013
?0??????	00000000	9.9E-015	2.475E-015	1.2009E-013
??0?????	00000000	9.9E-015	2.475E-015	1.2009E-013
???1????	00011111	9.9E-015	2.475E-015	1.2009E-013
????1???	00011111	9.9E-015	2.475E-015	1.2009E-013
?????1??	00011111	9.9E-015	2.475E-015	1.2009E-013

Received codeword	Decode	$P(y\|x_1)$	$P(x_1,y)$	$H(x_1,y)$
??????1?	00011111	9.9E-015	2.475E-015	1.2009E-013
???????1	00011111	9.9E-015	2.475E-015	1.2009E-013
8 erasures				
????????	00000	1E-016	2.5E-017	1.3787E-015

Table B.1: BEC Received vector decodes, probabilities and joint entropy contributions for BKLC$[8,2,5]$ with P(A) = 0.01

Appendix C

6-bit Codeword Stubs for Ham$[7,4,3]$

Codeword $C_0 = 0\ 0\ 0\ 0\ 0\ 0\ 0$
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0

Codeword $C_2 = 0\ 0\ 1\ 0\ 0\ 1\ 1$
0 0 1 0 0 1
0 0 1 0 0 1
0 0 1 0 1 1
0 0 1 0 1 1
0 0 0 0 1 1
0 1 0 0 1 1
0 1 0 0 1 1

Codeword $C_1 = 0\ 0\ 0\ 1\ 1\ 1\ 0$
0 0 0 1 1 1
0 0 0 1 1 0
0 0 0 1 1 0
0 0 0 1 1 0
0 0 1 1 1 0
0 0 1 1 1 0
0 0 1 1 1 0

Codeword $C_3 = 0\ 0\ 1\ 1\ 1\ 0\ 1$
0 0 1 1 1 0
0 0 1 1 1 1
0 0 1 1 0 1
0 0 1 1 0 1
0 0 1 1 0 1
0 1 1 1 0 1
0 1 1 1 0 1

Codeword $C_4 = 0\ 1\ 0\ 0\ 1\ 0\ 1$
$0\ 1\ 0\ 0\ 1\ 0$
$0\ 1\ 0\ 0\ 1\ 1$
$0\ 1\ 0\ 0\ 0\ 1$
$0\ 1\ 0\ 1\ 0\ 1$
$0\ 1\ 0\ 1\ 0\ 1$
$0\ 0\ 0\ 1\ 0\ 1$
$1\ 0\ 0\ 1\ 0\ 1$

Codeword $C_7 = 0\ 1\ 1\ 1\ 0\ 0\ 0$
$0\ 1\ 1\ 1\ 0\ 0$
$0\ 1\ 1\ 1\ 0\ 0$
$0\ 1\ 1\ 1\ 0\ 0$
$0\ 1\ 1\ 0\ 0\ 0$
$0\ 1\ 1\ 0\ 0\ 0$
$0\ 1\ 1\ 0\ 0\ 0$
$1\ 1\ 1\ 0\ 0\ 0$

Codeword $C_5 = 0\ 1\ 0\ 1\ 0\ 1\ 1$
$0\ 1\ 0\ 1\ 0\ 1$
$0\ 1\ 0\ 1\ 0\ 1$
$0\ 1\ 0\ 1\ 1\ 1$
$0\ 1\ 0\ 0\ 1\ 1$
$0\ 1\ 1\ 0\ 1\ 1$
$0\ 0\ 1\ 0\ 1\ 1$
$1\ 0\ 1\ 0\ 1\ 1$

Codeword $C_8 = 1\ 0\ 0\ 0\ 1\ 1\ 1$
$1\ 0\ 0\ 0\ 1\ 1$
$1\ 0\ 0\ 0\ 1\ 1$
$1\ 0\ 0\ 0\ 1\ 1$
$1\ 0\ 0\ 1\ 1\ 1$
$1\ 0\ 0\ 1\ 1\ 1$
$1\ 0\ 0\ 1\ 1\ 1$
$0\ 0\ 0\ 1\ 1\ 1$

Codeword $C_6 = 0\ 1\ 1\ 0\ 1\ 1\ 0$
$0\ 1\ 1\ 0\ 1\ 1$
$0\ 1\ 1\ 0\ 1\ 0$
$0\ 1\ 1\ 0\ 1\ 0$
$0\ 1\ 1\ 1\ 1\ 0$
$0\ 1\ 0\ 1\ 1\ 0$
$0\ 1\ 0\ 1\ 1\ 0$
$1\ 1\ 0\ 1\ 1\ 0$

Codeword $C_9 = 1\ 0\ 0\ 1\ 0\ 0\ 1$
$1\ 0\ 0\ 1\ 0\ 0$
$1\ 0\ 0\ 1\ 0\ 1$
$1\ 0\ 0\ 1\ 0\ 1$
$1\ 0\ 0\ 0\ 0\ 1$
$1\ 0\ 1\ 0\ 0\ 1$
$1\ 0\ 1\ 0\ 0\ 1$
$0\ 0\ 1\ 0\ 0\ 1$

Codeword $C_{10} = 1$	0	1	0	1	0	0
1	0	1	0	1	0	
1	0	1	0	1	0	
1	0	1	0	0	0	
1	0	1	1	0	0	
1	0	0	1	0	0	
1	1	0	1	0	0	
0	1	0	1	0	0	

Codeword $C_{11} = 1$	0	1	1	0	1	0
1	0	1	1	0	1	
1	0	1	1	0	0	
1	0	1	1	1	0	
1	0	1	0	1	0	
1	0	1	0	1	0	
1	1	1	0	1	0	
0	1	1	0	1	0	

Codeword $C_{12} = 1$	1	0	0	0	1	0
1	1	0	0	0	1	
1	1	0	0	0	0	
1	1	0	0	0	1	
1	1	0	0	0	1	
1	1	0	0	0	1	
1	0	0	0	1	0	
1	0	0	0	1	0	

Codeword $C_{13} = 1$	1	0	1	1	0	0
1	1	0	1	1	0	
1	1	0	1	1	0	
1	1	0	1	0	0	
1	1	0	1	0	0	
1	1	1	1	0	0	
1	0	1	1	0	0	
1	0	1	1	0	0	

Codeword $C_{14} = 1$	1	1	0	0	0	1
1	1	1	0	0	0	
1	1	1	0	0	1	
1	1	1	0	0	1	
1	1	1	0	0	1	
1	1	0	0	0	1	
1	1	0	0	0	1	
1	1	0	0	0	1	

Codeword $C_{15} = 1$	1	1	1	1	1	1
1	1	1	1	1	1	
1	1	1	1	1	1	
1	1	1	1	1	1	
1	1	1	1	1	1	
1	1	1	1	1	1	
1	1	1	1	1	1	
1	1	1	1	1	1	

Appendix D

Codeword stub rebuilds for Hamming$[7,4,3]$

Received codeword stub: $\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$

	Valid codewords															
Stub rebuilds	0000000	0001110	0010011	0011101	0100101	0101011	0110110	0111000	1000111	1001001	1010100	1011010	1100010	1101100	1110001	1111111
0001110	3	0					3	4			4	3	4	3		
0001111			3	2	3	2			2	3				6	3	
0001101	3			1	2			4		2	4			3	5	
0001111		1	3			2	4		2			4	5			3
0001011	3		2			1		4		2		3	4		5	
0001111		1		2	3		4		2		5			4		3
0000111	3		2		2		3		1		4		4		5	
0001111		1		2		2		5		3		4		4		3
0000111	3	2			2	3			1	4			4	5		
0010111			1	2			2	5			3	4			4	3
0000111	3	2	2	3					1	4	4	5				
0100111					1	2	2	5					3	4	4	3
0000111	3	2	2	3	2	3	3	6								
1000111									0	3	3	4	3	4	4	3

Table D.1: Codeword stub rebuild distances to valid codewords for Hamming$[7,4,3]$

Stub rebuilds	Valid codewords																
	0000000	0001110	0010011	0011101	0100101	0101011	0110110	0111000	1000111	1001001	1010100	1011010	1100010	1101100	1110001	1111111
0001110	1.4E-07	0.1345					1.4E-07	1.4E-09			1.4E-09	1.4E-07	1.4E-09	1.4E-07		
0001111			1.4E-07	1.4E-05	1.4E-07	1.4E-05			1.4E-05	1.4E-07				1.4E-13	1.4E-07	
0001101	1.4E-07			0.00136	1.4E-05			1.4E-09		1.4E-05	1.4E-09			1.4E-07	1.4E-11	
0001111		0.00136	1.4E-07			1.4E-05	1.4E-09		1.4E-05			1.4E-09	1.4E-11			1.4E-07
0001011	1.4E-07		1.4E-05			0.00136		1.4E-09		1.4E-05		1.4E-07	1.4E-09		1.4E-11	
0001111		0.00136		1.4E-05	1.4E-07		1.4E-09		1.4E-05		1.4E-11			1.4E-09		1.4E-07
0000111	1.4E-07		1.4E-05		1.4E-05		1.4E-07		0.00136		1.4E-09		1.4E-09		1.4E-11	
0001111		0.00136		1.4E-05		1.4E-05		1.4E-11		1.4E-07		1.4E-09		1.4E-09		1.4E-07
0000111	1.4E-07	1.4E-05			1.4E-05	1.4E-07			0.00136	1.4E-09			1.4E-09	1.4E-11		
0010111			0.00136	1.4E-05			1.4E-05	1.4E-11			1.4E-07	1.4E-09			1.4E-09	1.4E-07
0000111	1.4E-07	1.4E-05	1.4E-05	1.4E-07					0.00136	1.4E-09	1.4E-09	1.4E-11				
0100111					0.00136	1.4E-05	1.4E-05	1.4E-11					1.4E-07	1.4E-09	1.4E-09	1.4E-07
0000111	1.4E-07	1.4E-05	1.4E-05	1.4E-07	1.4E-05	1.4E-07	1.4E-07	1.4E-13								
1000111									0.1345	1.4E-07	1.4E-07	1.4E-09	1.4E-07	1.4E-09	4	3
TOTAL	9.7E-07	**0.1386**	0.0014	0.0014	0.0014	0.0014	2.8E-05	4.2E-09	**0.1386**	2.8E-05	2.8E-07	2.8E-07	2.8E-07	2.8E-07	4.2E-09	9.7E-09
															$p(y_7)$ for 000111	0.01768

Table D.2: Codeword stub rebuild probabilities for Hamming$[7,4,3]$

Appendix E

Codeword stub decodes and probabilities for Hamming $[7,4,3]$

Codeword stub	Decode	Output probabilities $p(y_j)$
0 0 0 0 0 0	0 0 0 0 0 0 0	0.05886
0 0 0 0 0 1	0 0 0 0 0 0 0	0.00121
0 0 0 0 1 0	0 0 0 0 0 0 0	0.00121
0 0 0 0 1 1	0 0 1 0 0 1 1	0.00945
0 0 0 1 0 0	0 0 0 0 0 0 0	0.00121
0 0 0 1 0 1	0 1 0 0 1 0 1	0.00945
0 0 0 1 1 0	0 0 0 1 1 1 0	0.02592
0 0 0 1 1 1	0 0 0 1 1 1 0	0.01768
0 0 1 0 0 0	0 0 0 0 0 0 0	0.00121
0 0 1 0 0 1	0 0 1 0 0 1 1	0.02592
0 0 1 0 1 0	0 0 0 1 1 1 0	0.00121
0 0 1 0 1 1	0 0 1 0 0 1 1	0.00259
0 0 1 1 0 0	0 0 1 1 1 0 1	0.00121
0 0 1 1 0 1	0 0 1 1 1 0 1	0.02592
0 0 1 1 1 0	0 0 0 1 1 1 0	0.03415
0 0 1 1 1 1	0 0 1 1 1 0 1	0.00945

Codeword stub	Decode	Output probabilities $p(y_j)$
0 1 0 0 0 0	0 0 0 0 0 0 0	0.00121
0 1 0 0 0 1	0 1 0 0 1 0 1	0.00945
0 1 0 0 1 0	0 1 0 0 1 0 1	0.00945
0 1 0 0 1 1	0 0 1 0 0 1 1	0.03415
0 1 0 1 0 0	1 0 1 0 1 0 0	0.00945
0 1 0 1 0 1	0 1 0 0 1 0 1	0.03415
0 1 0 1 1 0	0 0 0 0 0 0 0	0.01768
0 1 0 1 1 1	0 1 0 1 0 1 1	0.00945
0 1 1 0 0 0	0 1 1 1 0 0 0	0.02592
0 1 1 0 0 1	1 1 1 0 0 0 1	0.00121
0 1 1 0 1 0	0 1 1 0 1 1 0	0.02592
0 1 1 0 1 1	0 1 0 1 0 1 1	0.01768
0 1 1 1 0 0	0 1 1 1 0 0 0	0.02592
0 1 1 1 0 1	0 0 1 1 1 0 1	0.01768
1 0 1 1 1 0	0 1 1 0 1 1 0	0.00945
1 0 1 1 1 1	1 1 1 1 1 1 1	0.00121
1 0 0 0 0 0	0 0 0 0 0 0 0	0.00121
1 0 0 0 0 1	1 0 0 1 0 0 1	0.00945
1 0 0 0 1 0	1 1 0 0 0 1 0	0.01768
1 0 0 0 1 1	1 0 0 0 1 1 1	0.02592
1 0 0 1 0 0	1 0 1 0 1 0 0	0.01768
1 0 0 1 0 1	1 0 0 1 0 0 1	0.02592
1 0 0 1 1 0	0 0 0 1 1 1 0	0.00121
1 0 0 1 1 1	1 0 0 0 1 1 1	0.02592
1 0 1 0 0 0	1 0 1 0 1 0 0	0.00945

Codeword stub	Decode	Output probabilities $p(y_j)$
1 0 1 0 0 1	1 0 0 1 0 0 1	0.01768
1 0 1 0 1 0	1 0 1 1 0 1 0	0.03415
1 0 1 0 1 1	0 1 0 1 0 1 1	0.00945
1 0 1 1 0 0	1 1 0 1 1 0 0	0.03415
1 0 1 1 0 1	1 0 1 1 0 1 0	0.00945
1 0 1 1 1 0	1 0 1 1 0 1 0	0.00945
1 0 1 1 1 1	1 1 1 1 1 1 1	0.00121
1 1 0 0 0 0	1 1 0 0 0 1 0	0.00945
1 1 0 0 0 1	1 1 1 0 0 0 1	0.03415
1 1 0 0 1 0	1 1 0 0 0 1 0	0.02592
1 1 0 0 1 1	1 1 0 0 0 1 0	0.00121
1 1 0 1 0 0	1 1 0 1 1 0 0	0.02592
1 1 0 1 0 1	1 1 1 0 0 0 1	0.00121
1 1 0 1 1 0	1 1 0 1 1 0 0	0.02592
1 1 0 1 1 1	1 1 1 1 1 1 1	0.00121
1 1 1 0 0 0	0 1 1 1 0 0 0	0.01768
1 1 1 0 0 1	1 1 1 0 0 0 1	0.02592
1 1 1 0 1 0	1 0 1 1 0 1 0	0.00945
1 1 1 0 1 1	1 1 1 1 1 1 1	0.00121
1 1 1 1 0 0	1 1 0 1 1 0 0	0.00945
1 1 1 1 0 1	1 1 1 1 1 1 1	0.00121
1 1 1 1 1 0	1 1 1 1 1 1 1	0.00121
1 1 1 1 1 1	1 1 1 1 1 1 1	0.05886

Table E.1: Received stub decodes and probabilities for Hamming$[7,4,3]$ with $p_e = 0.01$

List of references

Agrawal, M., Kayal, N. & Saxena, N. (2004), 'Primes is in p', *Annals of Mathematics* **160**(2), 781–793.

Al-Hassan, S., Ahmed, M. & Tomlinson, M. (2014), New best equivocation codes for syndrome coding, *in* 'IEEE Conference Publications - International Conference on Information and Communication Technology Convergence', pp. 669–674.

Almeida, J. Barros, J. (2013), Random puncturing for secrecy, *in* 'IEEE Conference Publications - Asilomar Conference on Signals, Systems and Computers'.

Baylis, J. (1998), *Error-correcting Codes*, Chapman & Hall, London, UK. 005.72 BAY.

BBC (2018), 'Crypto-cash boom prompts graphics card rationing'.
URL: *http://www.bbc.co.uk/news/technology-42774882*

Britannica.com (2017), 'Information theory'.

Cobham, A. (1965), The intrinsic computational difficulty of functions, *in* Y. Bar-Hillel, ed., 'Logic, methodology and philosophy of science, Proceedings of the 1964 international congress (Studies in Logic and the Foundations of mathematics)', North-Holland Publishing, pp. 24–30.

Conway, J. & Sloane, N. (1999), *Sphere Packings, Lattices and Groups*, Springer, NY.

Cuff, P. (2010), Using a secret key to foil an eavesdropper, *in* 'IEEE Conference Publications - 48th Annual Allerton Conference on Communication, Control and Computing'.

Dalai, M. (2011), A new bound for the capacity of the deletion channel with high deletion probabilities, *in* 'IEEE International Symposium on Information Theory', pp. 499–502.

Davey, M. & Mackay, D. (2001), 'Reliable communication over channels with insertions, deletions and substitutions', *IEEE Transactions on Information Theory* **47**(2), 687–698.

Dictionary.com (2017), 'Dictionary.com'.
URL: *www.dictionary.com/browse/equivocate*

Dobrushin, R. (1967), 'Shannon's theorems for channels with synchroization errors', *Problems on Information Transmission* **3**, 11–26.

Fertonani, D. & Duman, T. (2010), 'Novel bounds on the capacity of the binary deletion channel', *IEEE Transactions on Information Theory* **56**(6), 2753–2765.

Golay, M. (1949), Notes on digital coding, *in* 'Proc. IRE'.

Golomb, S., Davey, J., Reed, I., Van Trees, H. & Stiffler, J. (1963), 'Synchronization', *IEEE Transactions on Communications Systems* **11**(4), 481–491.

Graham, B. (2015), 'A binary deletion channel with a fixed number of deletions', *Combinatorics, Probability and Computing* **24**(3), 486–489.

Grassl, M. (2015), 'Code tables: Bounds on the parameters of various types of codes', Internet.
URL: *http://www.codetables.de/*

Hamming, R. (1950), 'Error detecting and correcting codes', *Bell System Technical Journal* **24**(2).
URL: *http://www.lee.eng.uerj.br/ gil/redesII/hamming.pdf*

Hamming, R. (1980), *Coding and Information Theory*, Prentice Hall, Inc.Englewood Cliffs, NJ.

Hill, R. (1986), *A First Course in Coding Theory*, Oxford University Press, NY.

Hoffman, D. (1991), *Coding Theory - The Essentials*, Marcel Dekker, NY.

Jones, G. & Jones, J. (2002), *Information and Coding Theory*, Springer, London. 003.54 JON.

Kalai, A., Mitzenmacher, M. & Sudan, M. (2010), Tight asymptotic bounds for the deletion channel with small deletion probabilities, *in* 'IEEE International Symposium on Information Theory', pp. 997–1001.

Kanoria, Y. & Montanari, A. (2010), On the deletion channel with small deletion probability, *in* 'IEEE International Symposium on Information Theory', pp. 1002–1006.

Kanoria, Y. & Montanari, A. (2013), 'Optimal coding for the binary deletion channel with small deletion probability', *IEEE Transactions on Information Theory* **9**(1).

Kirsch, A. & Drinea, E. (2007), Directly lower bounding the information capacity for channels with i.i.d. deletions and duplications, *in* 'IEEE International Symposium on Information Theory', pp. 1731–1735.

Klinc, D., Jeongseok, H., Mclaughlin, S., Barros, J. & Byung-Jae, K. (2009), Ldpc for physical layer security, *in* 'IEEE Conference Publications - IEEE Global Telecommunications Conference'.

Klinc, D., Jeongseok, H., Mclaughlin, S., Barros, J. & Byung-Jae, K. (2011), 'Ldpc codes for the gaussian wiretap channel', *IEEE Transactions on Information Forensics and Security* **6**(3), 532–540.

Landjev, I. & Haralambiev, K. (2007), 'On multiple deletion codes', *Serdica Journal of Computing* **1**(1), 13–26.

Leung-Yan-Cheong, S., Barnes, E. & Friedman, D. (1979), 'On some properties of the undetected error probability of linear codes', *IEEE Transactions on Information Theory* **IT-25**(1), 110–112.

Levenshtein, V. (1965*a*), 'Binary codes capable of correcting deletions, insertions and reversals (in russian)', *Problemy Peredachi Informatsii* **163**(4), 845–848. English translation in Soviet Physics Dokl. 10 (No. 8, 1966), 707-710.

Levenshtein, V. (1965*b*), 'Binary codes capable of correcting spurious insertions and deletions of ones (in russian)', *Problemy Peredachi Informatsii* **1**(1), 12–25. English translation in Problems of Information Transmission 1 (No.1, 1965), 8-17.

Michelson, A. & Levesque, A. (1985), *Error Control Techniques for DIgital Communication*, J. Wiley & Sons.

Mitzenmacher, M. (2006), A simple lower bound for the capacity of the deletion channel, *in* 'IEEE Transactions on Information Theory', Vol. 52, pp. 4657–4660.

Mitzenmacher, M. (2009), 'A survey of results for deletion channel and related synchronization channels', *Probability Surveys* **6**.

Moon, T. (2005), *Error Correction Coding*, Wiley & Sons, NJ. 621.382 MOO.

Morelos-Zaragoza, R. (2002), *The Art of Error Correcting Coding*, John Wiley & Sons, Chichester, UK. 005.72 MOR.

Moser, S. & Chen, P. (2012), *A Student's Guide to Coding and Information Theory*, Cambridge University Press, Cambridge, UK.

Nvidia (2015), 'Nvidia cuda - about cuda'.
URL: *https://developer.nvidia.com/about-cuda*

Pless, V. (1968), 'On the uniqueness of the golay codes', *Journal on Combination Theory* **5**, 215–228.

Robson, J. (1983), The complexity of go, *in* 'Proceedings of the IFIP 9th World Computer Congress on Information Processing', pp. 413–417.

Roman, S. (1997), *Introduction to Coding and Information Theory*, Springer, NY. 003.54 ROM.

Ryan, W. & Lin, S. (2009), *Channel Codes: Classicial and Modern*, Cambridge University Press, Cambridge, UK.

Sanders, J. & Kandbrot, E. (2011), *CUDA by Example - An Introduction to General-Purpose CPU Programming*, Nvidia.

Schofield, M., Ahmed, M. & Tomlinson, M. (2015), Using parallel processing to calculate and improve code equivocation, *in* 'IEEE Conference Publications - IEEE 16th International Conference on Communication Technology'.

Schofield, M., Ahmed, M., Tomlinson, M. & Stengel, I. (2016), Intentional erasures and equivocation on the binary symmetric channel, *in* 'IEEE Conference Publications - International Computer Symposium', IEEE, pp. 233–235.

Shannon, C. (1948), 'A mathematical theory of communication', *Bell System Technical Journal* **27**(3), 379–423.

Shannon, C. (1949), 'Communication theory of secrecy systems', *Bell System Technical Journal* **28**(4), 656–715.

Shoup, V. (2015), 'Ntl: A library for doing number theory'.
URL: *http://www.shoup.net/ntl/*

Sloane (2002), 'On single-deletion-correcting codes', *Codes and Designs* pp. pp273–291.

Smith, G. (2011), Quantifying information flow using min-entropy, *in* '8th International Conference on Quantitative Evaluation of Systems'.

Stambler, S. (1970), 'Memoryless channels with synchronization errors - general case', *Problems on Information Transmission* **6**(3), 43–49.

Sweeney, P. (2002), *Error Control Coding - from Theory to Practice*, Wiley & Sons, Chichester, UK. 005.72 SWE.

Tietäväinen, A. (1973), 'On the nonexistence of perfect codes over finite fields', *SIAM Journal on Applied Mathematics* **24**(1), 88–96.

Togneri, R. & deSilva, J. (2002), *Fundamentals of Information Theory and Coding Design*, Chapman & Hall.

Ullman, J. (1967), 'On the capabilities of codes to correct synchronization errors', *IEEE Transactions on Information Theory* **IT-13**(1), 95–105.

University of Sydney (2015), 'Magma computation algebra'.
URL: *http://magma.maths.usyd.edu.au/magma/*

van Lint, J. (1999), *Introduction to Coding Theory*, 3rd edn, Springer, Berlin. 003.54 LIN.

Varshamov, R. & Tenengolts, G. (1965), 'Codes wihich correct single asymmetric errors (in russian))', *Avtomatika i Telemekhanika* **26**(2), 288–292. English translation in Automation and Remote Control vol26, no.2 1965, 286-290.

Wickramasooriya, A., Land, I. & Subramanian, R. (2013), Comparison of coding strategies for the block fading earasure wiretap channel, *in* 'Australian Communications Theory Workshop', pp. 158–163.

Woodward, P. & Davies, I. (1952), 'Information theory and inverse probability in telecommunication"', *Proceedings of the IEE* **99**(58), pp37–44.

Wyner, A. (1975), 'The wire-tap channel', *Bell System Technical Journal* **54**(8), 1355–1387.

Zhang, K., Tomlinson, M., Ahmed, M., Ambroze, M. & Rodrigues, M. (2014), Best binary equivocation code construction for syndrome coding, *in* 'IET Communications', Vol. 8, IET, pp. 1696–1704.

Bound copies of published papers

Using Parallel Processing To Calculate and Improve Code Equivocation

Mark Schofield, Mohammed Zaki Ahmed, Ingo Stengel and Martin Tomlinson
School of School of Computing and Mathematics
Plymouth University, Plymouth, United Kingdom
Email: mark.schofield, M.Ahmed, I.Stengel, M.Tomlinson @plymouth.ac.uk

***Abstract*—Equivocation gives a measure of the average level of ambiguity of a received signal and the level of security that a code can offer. The development of a software solution using GPU parallel processing to calculate the equivocation more efficiently enables values for longer code lengths to be calculated. Equivocation values for various codes and their expansions are compared and inferences drawn about their relative security.**

I. Introduction

One metric of a code's secrecy is its equivocation. By comparing the equivocation of different codes and modifications of those codes, codes with higher levels of equivocation can be identified. Since such a process can be very computationally intensive, a method for calculating the equivocation using parallel processing was developed. Whilst others [1],[2] have used equivocation calculations to compare and construct codes and improve code secrecy, especially involving syndrome coding, none have previously done so using parallel processing.

II. Codes and Equivocation

A *binary*, *linear* code is constructed from symbols belonging to the binary field $\mathbb{F}_2$ (also known as the Galois Field of 2, **GF**2) [3] in which any linear combination of codewords is also a codeword.

Equivocation (or the conditional entropy) describes the average ambiguity of a received signal [4]. It represents the information loss of the channel going from input to output. The probabilities of input messages (X) and output messages (Y) can be used to calculate a set of entropies. In turn, these entropies can be used to calculate the equivocation of the code. For each given input message, the conditional probabilities $\Pr(y \mid x)$ of each decoded message can be calculated. From the conditional probabilities, the joint probabilities $\Pr(X, Y)$ can be calculated. The entropy of an information source is given by equation 1:

$$H_r(X) = -\sum_i \Pr(x_i) \log(\Pr(x_i)) \tag{1}$$

Similarly the joint entropy of an information source and output is given by equation 2:

$$H_r(X,Y) = -\sum_{i,j} \Pr(x_i, y_j) \log(\Pr(x_i, y_j)) \tag{2}$$

The joint entropy is the sum of the source entropy and the conditional entropy. Hence the equivocation (or conditional entropy) can be found from equation 3 :

$$H_r(Y|X) = H_r(X,Y) - H_r(X) \tag{3}$$

The equivocation for an (n, k, d) code where *n*=codeword length, *k*=message length and *d*=minimum distance of the code, can be calculated using the method:
For one message and its codeword,

1) Add successively weighted error vectors.
2) Note for which errors each syndrome is obtained first. These are the errors that can be correctly corrected.
3) Operating on batches of codewords at a time, add the 2^{n-k} error vectors to each of the 2^k codewords in turn
4) Record the weights and probabilities of the resulting received codewords.
5) Use these 2^n probabilities to find the equivocation.

III. Implementation and Parallelisation of Calculations

Early iterations of an efficient program to calculate the equivocation of a code employed linear programming. This created a need to call sections of code many times. Shoup's Number Theory Library "NTL" [5] was used extensively during this phase, primarily for vector and matrix manipulation over the GF2 field.

The high volume of calculations needed to evaluate the equivocation of a code of any significant length means that on a standard laptop or desktop computer, the length of time needed to run a calculation is the dominant limiting factor. When performing the calculation via a linear process, the implementation could calculate the equivocation for codes of length $n < 32$ but little more. To overcome this, stages of the calculation needed to be performed in parallel.

To parallelise the calculation, a computer capable of running Nvidia's Cuda architecture and programming model was used. CUDA (Compute Unified Device Architecture) [6] is a parallel computing platform and programming model created by Nvidia. It is implemented by computers with Cuda-capable Nvidia GPUs, with the main CPU acting as the 'host' for the linear component of a program which then delegates responsibility for running parallelised sections of code to the GPU 'device'. Different combinations of GPUs and Cuda versions have different capabilities. To write and compile the Cuda specific code, Nsight for Eclipse was used as an enhanced Integrated Development Environment, along with the Nvidia Cuda Compiler NVCC.

Once compiled, the linear part of the program is run on the host CPU. The part of the program to be executed in parallel by the GPU device is similar in structure to a function and is called a

kernel. Each instance of the kernel is called a *thread*. The Cuda architecture enables multiple threads to be run concurrently, grouped into blocks.
Limitations on memory capacity and processing speed enforced compromises and constraints on block sizes and the grouping together of blocks into batches. A set-up with compute capability 3.5 will restrict the maximum number of threads per block to 2^{10}. Similarly, a maximum of $(2^{31}-1)$ blocks are permitted. This implies a maximum of $(2^{41}-2^{10})$ threads per kernel call. In practice however, the dynamic limitations of how much contiguous memory can be allocated to a single variable pointer meant that significantly smaller batches of blocks had to be used in order to prevent memory overflow.

IV. Cuda Coding

A Cuda kernel [7] called `getWts` can be called by the CPU host by the following code:

```
getWts<<<*blocks,*threads>>>(d_Ctx,
d_EV, d_powK, d_powNK, d_n, d_wts);
```

The triple angled brackets indicate that the function is a kernel to be run on the GPU device and define the number of blocks and threads to be created by each instance of the kernel. Parameters in the brackets are pointer variables to be used by the kernel. The code for the kernel to be run in parallel by the device is indicated by the `__global__` function.

```
__global__ void getWts(int* Ctx, int* EV, long*
powK, long* powNK, int* n, int* wts)
{
      long id = blockIdx.x * blockDim.x
      + threadIdx.x;
      long wt = 0;
      for (long i = 0; i < *powNK; i++ )
      {
            ....
      }
}
```

The values held by variables such as `blockDim.x`, `blockIdx.x` and `threadIdx.x` identify the number of threads per block and the identity of the block or thread being executed. This enables each individual thread to access the specific data that it requires.

V. Results

The use of parallel processing enabled calculations to be performed for codewords of length up to 40, an improvement of 8 bits over the linear method. The calculation time is dependent upon many factors, including the message length k, the codeword length n, the number of threads per block and blocks per batch and the structure of the calculation program. Independent runs of calculations for increasing lengths of k and n were performed. A graph of the times taken to calculate equivocation values for some Best Known Linear Codes (BKLCs) of message length k=15 and increasing values of n is shown in figure 1. The BKLCs used were generated using Magma software from the University of Sydney [8]. The calculations that were performed for the probabilities 0 to 0.5 in 0.05 increments, with 0.01 as an additional value. Many of the calculations were subsequently re-performed for probabilities 0 to 0.5 in 0.01 increments in order to give more accurate and smoother output graphs.
A graph of the times taken to calculate equivocation values for

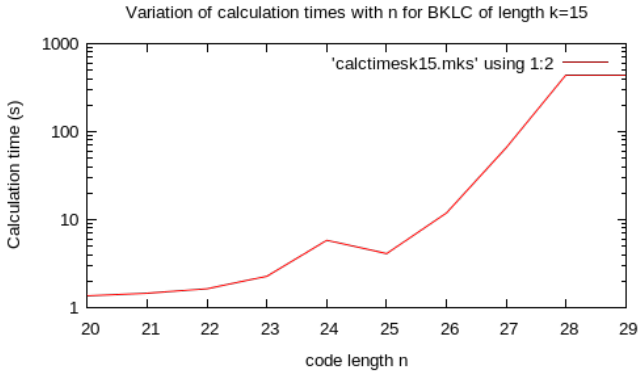


Fig. 1. Time to calculate equivocation for codes of message length k=15 for different code lengths(n)

some BKLCs of code length n=30 and increasing values of k are shown in figure 2. It can be seen that for a fixed codeword length, there is an optimal message length k that gives a minimum calculation time. This occurs around k=20 and is due to the calculation method used in the software. There are 2^{10} threads in each block and 2^{10} blocks per instance of the kernel of the parallel component of the program. Below a message length of k=20, the parallel component of the program has not yet achieved maximum efficiency and above k=20, the linear part of the program is performing an increasing proportion of the workload. The times taken to calculate equivocation values

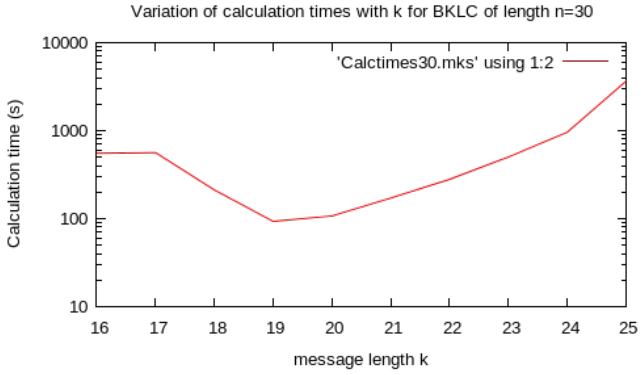


Fig. 2. Time to calculate equivocation for codes of length n=30 for different message lengths(k)

for some perfect codes,their extensions and some longer codes are shown in table I. Whilst the linear method is actually quicker for very short length codes, the parallel method soon becomes the preferred method, producing a result more quickly for all code lengths where $n > 10$. The results for longer length codes show that significant gains can be made, with calculations being performed by the parallel processing method up to 35 times quicker than by the linear method in the instance of the BKLC(33,21,6) code.
A graph showing the normalised equivocation values of different perfect and extended perfect codes is shown in figure 3.

Code	Linear time (s)	Parallel time (s)	Linear to parallel ratio
Hamming(7, 4, 3)	0.0174	0.068	1 : 4
Ext. Hamming (8, 4, 4)	0.021	0.084	1 : 4
Hamming(15, 11, 3)	0.991	0.147	7 : 1
Golay(23, 12, 7)	67.996	4.48	15 : 1
Ext. Golay(24, 12, 8)	238.6	17.8	13 : 1
Hamming(31, 26, 3)	33961	3809	9 : 1
BKLC(30, 20, 5)	3466	105.99	33 : 1
BKLC(31, 21, 5)	6880	238.6	29 : 1
BKLC(32, 22, 5)	13702	444	31 : 1
BKLC(33, 21, 6)	27048	780	35 : 1
BKLC(33, 23, 5)	27634	1021	27 : 1
BKLC(36, 25, 5)	not reasonably calculable	6760	-
BKLC(36, 26, 4)	not reasonably calculable	8878	-
BKLC(40, 27, 6)	not reasonably calculable	91875	-

TABLE I. EQUIVOCATION CALCULATION TIMES FOR SOME PERFECT CODES, THEIR EXTENSIONS AND LONGER BKLCS

Longer perfect codes generally exhibit higher levels of equivocation than shorter ones. Hamming (31,26,3) has higher normalised equivocation values than Hamming (15,11,3), which in turn, has higher values than Hamming(7,4,3). Hamming codes also have higher equivocation values than the Golay(23,12,7) code at lower probabilities ($p(e) < 0.15$). However this is offset by the Golay code benefitting from its ability to correct up to 3 errors whilst the Hamming codes can only correct a single error [9].

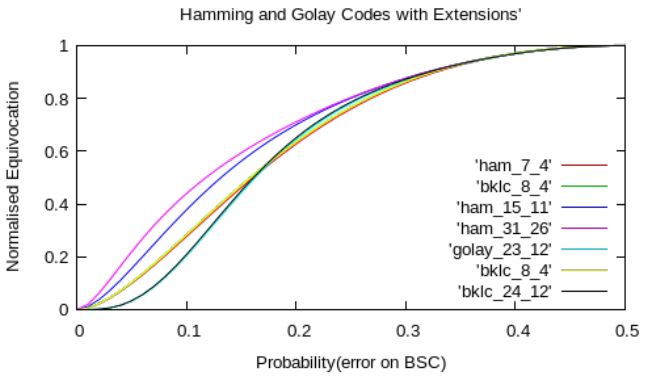


Fig. 3. Normalized equivocation of some perfect codes and their extensions

The normalised equivocation values of some longer Best Known Linear Codes (BKLC) in figure 4 show comparatively little difference between the codes as code length increases. The BKLCs used in figures 4 were chosen to demonstrate the different lengths of time that the parallelised calculation would take as code lengths increased. They were not known to possess similar properties to each other, however as the code length increases, the gradients of their respective equivocation curves become increasingly similar to each other.

For all codes examined, the normalised equivocation is an increasing function when expressed as a function of the probability of error on a binary symmetric channel. However the rate of increase is often at its lowest in the range $0 \leq p(e) \leq 0.05$, while the rate of increase is often at its highest in the range $0.05 \leq p(e) \leq 0.2$. Given that it may be common for a legitimate receiver to receive the signal through a channel with a low probability of transmission error and for an illegitimate receiver to receive the signal with a markedly higher probability of transmission error, this could

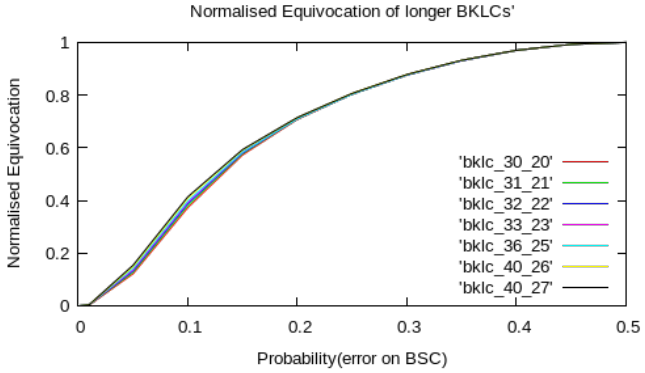


Fig. 4. Normalized equivocation of longer Best Known Linear Codes

be of use when developing codes that are designed to maximise the differences in ambiguity levels between a legitimate and illegitimate receiver.

Such an approach enables a move away from theoretical situations where both the legitimate recipient and the eavesdropper have a perfect channel towards situations where both channels may involve a level of signal degradation. By accepting and managing a level of degradation for the legitimate recipient, more coding schemes could be made available that provide a significantly higher level of equivocation for an eavesdropper than for the legitimate receiver. The nature of the channels becomes a means of providing security.

VI. EXPANDING CODES

Once a procedure for efficiently calculating the equivocation of a code has been implemented, it also encourages the ability to compare codes and to locate and design codes with improved levels of equivocation. As an example, consider a modification of the simple Hamming(7,4,3) code. A single bit of data could be replaced by a sequence of data bits, say, 4 bits long. The first three bits are randomly generated whilst the fourth bit is chosen to give a message parity equivalent to the data that is to be transmitted. So the data bit 0 could be represented as (0011), each component of which is transmitted as the first bit of 4 successively transmitted messages:

$$(\overbrace{001}^{RandomBits}\ \overbrace{1}^{ParityBit})$$

Where previously the probability of an error occurring as a single bit was transmitted might have been quite low, for example 0.01, now that the representation of the data bit 0 is transmitted across more messages, the probability of an error becomes compounded and potentially much increased. If an *x*-fold expansion of a code is taken to be one in which a data bit is represented by *x-1* random bits and 1 parity check bit, then the received data bit would be expressed in terms of the parity of the received data bits:

$$Parity = \sum_{i=0}^{x} r_i \tag{4}$$

where r_i is the *i*'th received bit of the code expansion. The received data is the sum of the transmitted data and any

associated errors that occur during transmission:

$$r_i = t_i + e_i \tag{5}$$

where t_i is the i'th bit of the expansion and e_i is the associated error. Therefore:

$$Parity = \sum_{i=0}^{x} t_i + \sum_{i=0}^{x} e_i \tag{6}$$

But by design, $\sum_{i=0}^{x} t_i = 0$ and therefore $Parity = \sum_{i=0}^{x} e_i$,so if the received message parity equals 1, then a decoder error must have occurred.

When considering a 4-fold code expansion, the possible error combinations that could occur range from 0000 to 1111. Given that we only need to consider the combinations that give an odd parity, then the probability of the 4-fold expansion being decoded erroneously is:

$$\binom{4}{1}(1-p(e))^3 p(e) + \binom{4}{3}(1-p(e))p(e)^3$$

In general for an x-fold expansion, the probability of an incorrect decoding would be:

$$\sum_{i=0}^{i \le x/2} \binom{x}{2i+1}(1-p(e))^{x-(2i+1)} p(e)^{2i+1}$$

A 4-fold expansion of the Hamming code gives effective channel probability errors as shown in table II. These effective channel probability errors yield normalized equivocation values for a 4-fold expansion of the Hamming(7, 4, 3) code plotted in figure 5, along with 2, 3 and 10-fold expansions of the code. The figure shows that the equivocation values of code expansions can be significantly higher than that of a simple Hamming code. For the 10-fold expansion, a 0.01 probability of transmission error yields a normalised equivocation level of 0.23, however an error probability of just 0.05 now yields a normalised equivocation value of 0.897, compared to 0.1005 for the unexpanded Hamming code. If the intended recipient has a channel probability error of 0.01, they still have a good chance of recovering the data, whereas an illegitimate eavesdropper with a channel error probability of 0.05 will now need to overcome a much higher level of ambiguity in order to recover the data. The secrecy of the data transmission has been very significantly improved by the expansion of a simple Hamming code. However this improved secrecy comes at a cost; only one bit of data is transmitted for every 4 or 10 bits of data carried by the Hamming code, reducing the data rate significantly.

Single bit probability of error	4-fold expansion probability of error
0	0
0.01	0.038816
0.05	0.17195
0.1	0.2952
0.15	0.37995
0.2	0.4352
0.25	0.46875
0.3	0.4872
0.35	0.49595
0.4	0.4992
0.45	49995
0.5	0.5

TABLE II. COMPOUNDED CHANNEL PROBABILITY ERRORS FOR A 4-FOLD CODE EXPANSION

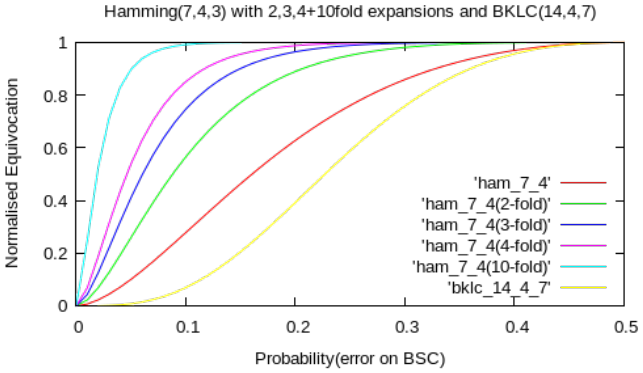


Fig. 5. Comparison of Hamming code with 2x, 3x, 4x and 10x code expansion and BKLC(14,4)

While the Hamming(7,4,3) code with 2-fold expansion and BKLC(14,4,7) code both effectively take 14 bits to transmit 4 bits worth of data (and therefore have the same effective code rate), their equivocation graphs are very different. The 2-fold expansion of the Hamming (7,4,3) code offers a significant improvement in equivocation over the basic Hamming (7,4,3) code, whereas the BKLC(14,4,7) code shows a significant worsening in performance. This suggests that using code expansions is more effective at improving equivocation than simply using longer and longer codes.

A comparison of the equivocation of the perfect Golay(23,12,7) code and two random (23,12) codes is shown in figure 6, along with their 4-fold expansions. This shows that:

- Random codes can possess higher levels of equivocation (and therefore secrecy) than a perfect code, although their error correcting capabilities may not be as good.
- Normalised equivocation rates of 4-fold expansions of the Golay and random (23,12) codes are significantly higher than those of the original codes.

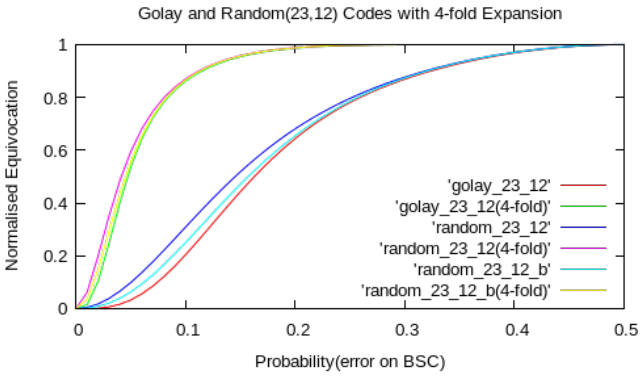


Fig. 6. Comparison of Golay(23, 12, 7) and 2 random (23, 12) codes with 4-fold expansion

VII. CONCLUSION

This paper has developed several points with regard to both the use of parallel processing for calculating equivocation and

the comparison of equivocation values for different codes and their variants.

- Parallel processing significantly improve equivocation calculation times.
- Longer codes of the same family will tend to have higher relative equivocation values.
- Largest differences in code equivocation values are often found in the $0.01 < p(e) < 0.2$ range, especially for expanded codes.
- Differences in normalised equivocation values between codes appear to decrease as code lengths increase.
- By expanding a code through using random data and parity check bits, greater secrecy can be achieved.
- By expanding random codes, higher equivocation values can be achieved than for non-expanded codes.
- Code expansions can offer higher normalised equivocation values than a code with a similar code rate.

REFERENCES

[1] K. Zhang, *et al.*, *Best binary equivocation code construction for syndrome coding* IET Commum., Vol.8(10), pp1969-1704, February, 2014.

[2] S. Al-Hassan, M.Z. Ahmed, M. Tomlinson,*Secrecy coding for the wiretap channel using best known linear codes* in *2013 Global Information Infrastructure Symposium* pp.1-6

[3] R. Hill, *A First Course in Coding Theory* Oxford, UK: OUP, 1986.

[4] C. Shannon, *A Mathematical Theory of Communication* The Bell System Technical Journal, Vol. 27, pp. 379-423, 623-656, July, October, 1948.

[5] V. Shoup. *NTL: A Library for Doing Number Theory.* (2014) [Online]. Available: http://www.shoup.net/ntl/, Accessed on: July 16, 2015.

[6] Nvidia, Santa Clara, CA, USA *CUDA Toolkit - Programming Guide.* (2015) [Online]. Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#compute-capabilities, Accessed on: July 17, 2015.

[7] J. Sanders, E. Kandrot, *Cuda by Example - An Introduction to General - Purpose GPU Programming* Upper Saddle River, NJ: Addison-Wesley, 2011.

[8] University of Sydney, Australia *Magma Computation Algebra System* (2015) [Online]. Available: http://magma.maths.usyd.edu.au/magma/, Accessed on: July 18, 2015.

[9] R. Hamming, *Coding and Information Theory* Englewood Cliffs, NJ, USA: Prentice Hall Inc., 1980.

Intentional Erasures and Equivocation on the Binary Symmetric Channel

Mark Schofield, Mohammed Zaki Ahmed, Ingo Stengel and Martin Tomlinson
School of Computing, Electronics and Mathematics
Plymouth University, Plymouth, United Kingdom
Email: mark.schofield, M.Ahmed, M.Tomlinson @plymouth.ac.uk, Ingo.Stengel@hs-karlsruhe.de

***Abstract*—Secure data transfer is often achieved by encryption. However, if transmitting across a Binary Symmetric Channel while an eavesdropper listens via a Wiretap Channel, the difference in signal quality can be exploited to improve the inherent transmission secrecy. If this exploitation involves the managed use of erasures, then the secrecy of a code transmitted across the channel can be improved whilst simultaneously reducing the volume of data that is transmitted.**

***Keywords*-Ambiguity, binary symmetric channel (BSC), code expansion, equivocation, erasure, graphical processing unit (GPU), parallel processing, security, wiretap.**

I. Introduction

Shannon's seminal papers [1], [2] introduced equivocation (conditional entropy) as a measure of the secrecy of the information contained within a received signal. Other metrics for assessing secrecy have been proposed such as the value function [3] and the security gap [4], however Klinc's paper also noted that equivocation continues to be recognised as an established metric. As such, equivocation will be used in this work.

It is acknowledged [5] that it can be difficult to measure or analyse equivocation. Work has been done to establish upper and lower bounds for equivocation such as Almeida's use of known and unknown puncturing patterns [6], while other techniques have been applied to specific circumstances to calculate equivocation. These include Wickramsooriya's analysis of the generator matrix of the eavesdropper's code [7], [8] and Al-Hassan's use of probability mass functions for syndrome coding [9].

Zhang's work [10], [11] constructs best binary equivocation codes for syndrome coding in the Binary Symmetric Channel (BSC) [12], however the direct evaluation of the equivocation of a code suitable for a channel having erasures has received little attention. The consideration of all possible input messages to a transmission system and all possible decoded output messages is highly numerically intensive. An $[n, k, d]$ code (n = code length, k = message length, d = min. distance) must consider 2^n error vectors for each of the 2^k messages. This was addressed by the author in [13] when GPU parallel processing techniques were used to calculate the equivocation values for codes and their expansions more efficiently. The use of GPU-based parallel processing permitted calculation times up to 35 times quicker than with traditional linear techniques.

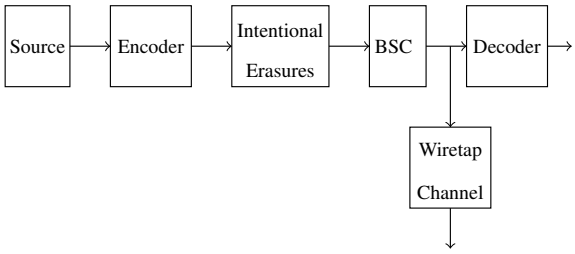


Fig. 1. Code Transmission across a BSC with erasures, subject to wiretap

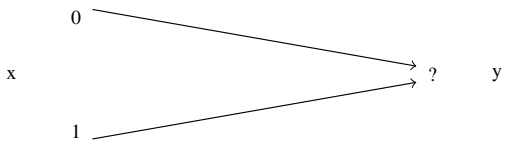


Fig. 2. Deliberate Erasures

This paper extends that work to consider the normalised equivocation of codes that include intentionally erased bits.

An erasure of a data bit is where the value of the data bit is not known but the location of the bit is known. The Binary Erasure Channel (BEC) is an idealised channel that uses erasures [14]. Equivocation of BECs in a wiretap environment [15] has been examined [16], [17], [18], however here we look at the intentional erasure of one or more bits of data ahead of transmission across a BSC (IE+BSC) within a wiretap environment, shown in Fig. 1. If erasures of a transmitted symbol $x \in X$ are introduced for the IE+BSC then the value of a received symbol $y \in Y$ in a specified location is unknown as in Fig. 2 and has a uniform distribution. The non-erased bits will be susceptible to a cross-over probability of p on the BSC.

II. Calculation of Equivocation

If a message of length k is encoded as codeword of length n and s bits in known locations are then intentionally erased ahead of transmission, the probability of a received message having contained e errors in specific locations and the s erasures having come from a particular combination of 0's

and 1's is:

$$\mathrm{p}(e,s) = (1-p)^{(n-s)-e}p^e \times \frac{1}{2^s} \tag{1}$$

The joint entropy of a set of source symbols X and received symbols Y is:

$$H(X,Y) = \sum_{x\in X}\sum_{y\in Y} p(x,y)\log\frac{1}{p(x,y)} \tag{2}$$

Shannon defined secrecy as:

$$H(X|Y) = H(X,Y) - H(Y) \tag{3}$$

This is evaluated as:

$$H(X|Y) = \sum_{x\in X}\sum_{y\in Y} p(x,y)\log\frac{1}{p(x,y)} - \sum_{y\in Y} p(y)\log\frac{1}{p(y)} \tag{4}$$

The evaluation of Eqn. 4 is numerically intensive but evaluation for short codes is possible with modern hardware. The normalised equivocation is the equivocation per transmitted bit, $\frac{H(X|Y)}{k}$.

III. Software Implementation

The software implementation of the equivocation calculation was achieved through the use of the GPU parallel processing capabilities afforded by the Nvidia CUDA architecture [19], where the linear component of the program is run on the CPU and the repetitive codeword and error vector additions are performed in parallel on a CUDA-enabled Nvidia graphics processor [20]. This multi-threaded approach offers significant efficiency (and therefore time) savings over standard linear methods. The calculations were run for a variety of binary linear codes, including perfect Hamming and Golay codes and best known linear codes (BKLCs) identified in [21]. Since combinations of e errors and s erasures can be correctly decoded provided that $2e + s < d$ [22], a maximum of $d-1$ erasures can be corrected. Generator and parity check matrices for codes were generated using online Magma software [23].

IV. Results

It can be seen from the BKLC$[8,2,5]$ in Fig. 3 and the Golay$[23,12,7]$ code in Fig. 4 that increasing the number of deliberately erased bits that are transmitted increases the equivocation of the code. The curves for both 3 and 4 erasures of the BKLC$[8,2,5]$ code are co-linear i.e. transmitting the code with either 3 or 4 erasures produces the same values of normalised equivocation. This is because the increase from 3 to 4 erasures does not change the ability of the code to use the 2^6 available syndrome patterns for the detection or correction of either errors or erasures.

The deliberate introduction of erasures to a transmission system can, if chosen carefully, lead to a greater increase in equivocation for an illegitimate eavesdropper than for the legitimate receiver. For example with the Golay$[23,12,7]$ code, a BSC error probability of 0.01 for the legitimate receiver and 0.10 for the eavesdropper will produce the normalised

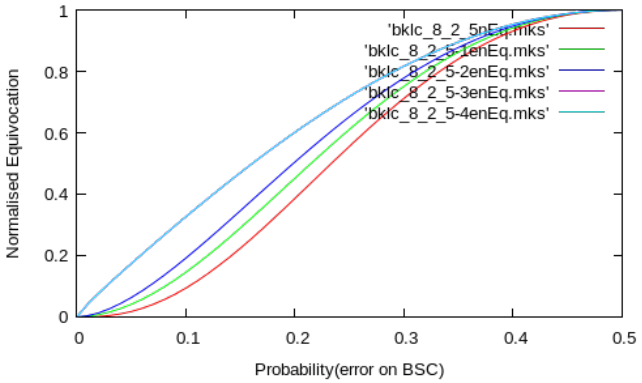


Fig. 3. Equivocation of BKLC$[8,2,5]$ code with up to 4 erasures

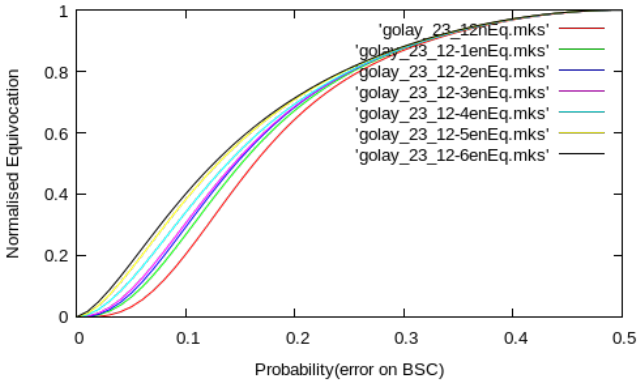


Fig. 4. Equivocation of Golay$[23,12,7]$ code with up to 6 erasures

	Legitimate receiver	Eavesdropper
BSC error probability	0.01	0.10
Equivocation (with no erasures)	0.00015	0.20496
Equivocation (with 2 erasures)	0.00147	0.29394
Increase in equivocation	0.00132	0.08898

TABLE I
Equivocation of Golay$[23,12,7]$ code on BSC with and without 2 erasures

equivocation levels in Table I for scenarios with no erasures and with 2 erasures.

In this case, the deliberate introduction of 2 erasures has increased the normalised equivocation for the eavesdropper by a greater amount (0.089) than for the legitimate receiver (0.0013).

However, this does not always hold. Consider the Golay$[23,12,7]$ code with an error probability of 0.05 for the legitimate receiver and 0.20 for the eavesdropper for scenarios with no erasures and 6 erasures, shown in Table II.

In this case, the equivocation has increased for the legitimate receiver by a great amount (0.144) than it has for the eavesdropper (0.069).

Fig. 5 highlights some differences between the use of code expansion and the use of erasures, in this case for the Golay$[23,12,7]$ code. n-fold code expansion was achieved by

	Legitimate receiver	Eavesdropper
BSC error probability	0.05	0.20
Equivocation (with no erasures)	0.03317	0.64453
Equivocation (with 2 erasures)	0.17759	0.71385
Increase	0.14442	0.06932

TABLE II
EQUIVOCATION OF GOLAY$[23, 12, 7]$ CODE ON BSC WITH AND WITHOUT 6 ERASURES

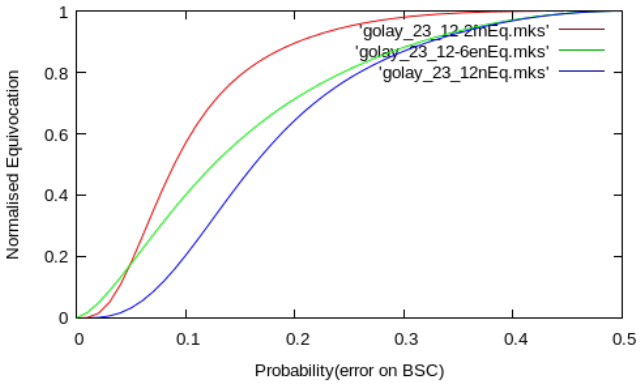


Fig. 5. Equivocation of Golay$[23, 12, 7]$ code with 6 erasures versus 2-fold code expansion

replacing a single data bit with a sequence of $n-1$ random bits and an n'th bit that gives a message parity equivalent to the data to be transmitted. For all error probabilities above 0.05, code expansion offers greater increases in equivocation than the deliberate introduction of erasures. However below 0.05, the use of multiple erasures gives a greater increase in equivocation. Therefore at close to 0.05 for the Golay code, there is a changeover point above which a 2-fold code expansion gives higher equivocation levels and below which the use of 6 erasures gives higher equivocation. Either adaptation to the code carries a penalty - the use of code expansion increases the number of transmitted bit dramatically whereas the deliberate introduction of erasures decreases the error detecting and correcting capability of the code.

V. CONCLUSION

In addition to the expansion of codes previously discussed, erasures can be used as an alternative mechanism for increasing the equivocation of a code over a Binary Symmetric Channel and thereby its average ambiguity and secrecy.

- Increases in equivocation in the erasure examples given are significant but not as great as those increases shown by the use of code expansions
- Some increases in the number of bits erased do not necessarily lead to an increase in equivocation
- The introduction of erasures to a channel can potentially lead to a greater increase in equivocation for an eavesdropper than it does for the legitimate receiver.
- A comparison between code expansions and erasures can show situations where, for a particular error probability, there is a changeover point in which method produces the higher equivocation values e.g. at around $p(e) = 0.05$ for the Golay$[23, 12, 7]$ code.

REFERENCES

[1] C. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, July 1948.

[2] ——, "Communication theory of secrecy systems," *Bell System Technical Journal*, vol. 28, no. 4, pp. 656–715, 1949.

[3] P. Cuff, "Using a secret key to foil an eavesdropper," in *IEEE Conference Publications - 48th Annual Allerton Conference on Communication, Control and Computing*, 2010.

[4] D. Klinc, H. Jeongseok, S. Mclaughlin, J. Barros, and K. Byung-Jae, "Ldpc for physical layer security," in *IEEE Conference Publications - IEEE Global Telecommunications Conference*, 2009.

[5] ——, "Ldpc codes for the gaussian wiretap channel," *IEEE Transactions on Information Forensics and Security*, vol. 6, no. 3, pp. 532–540, 2011.

[6] J. Almeida, J. Barros, "Random puncturing for secrecy," in *IEEE Conference Publications - Asilomar Conference on Signals, Systems and Computers*, 2013.

[7] A. Wickramasooriya, I. Land, and R. Subramanian, "Comparison of coding strategies for the block fading erasure wiretap channel," in *Australian Communications Theory Workshop*, 2013, pp. 158–163.

[8] ——, "Comparison of equivocation rate of finite length codes for the wiretap channel," in *VDE Conference Publications - Proceedings of 2013 9th International ITG Conference on Systems, Communication and Coding*, 2013.

[9] S. Al-Hassan, M. Ahmed, and M. Tomlinson, "New best equivocation codes for syndrome coding," in *IEEE Conference Publications - International Conference on Information and Communication Technology Convergence*, 2014, pp. 669–674.

[10] K. Zhang, M. Tomlinson, and M. Ahmed, "The average equivocation of random linear binary codes in syndrome coding," in *21st International Conference on Telecommunications*, May 2014, pp. 47–51.

[11] K. Zhang, M. Tomlinson, M. Ahmed, M. Ambroze, and M. Rodrigues, "Best binary equivocation code construction for syndrome coding," in *IET Communications*, vol. 8, no. 10. IET, 2014, pp. 1696–1704.

[12] G. Jones and J. Jones, *Information and Coding Theory*. Springer, London, 2002, 003.54 JON.

[13] M. Schofield, M. Ahmed, and M. Tomlinson, "Using parallel processing to calculate and improve code equivocation," in *IEEE Conference Publications - IEEE 16th International Conference on Communication Technology*, 2015.

[14] S. Moser and P. Chen, *A Student's Guide to Coding and Information Theory*. Cambridge University Press, Cambridge, UK, 2012.

[15] A. Wyner, "The wire-tap channel," *Bell System Technical Journal*, vol. 54, no. 8, pp. 1355–1387, October 1975. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1002/j.1538-7305.1975.tb02040.x/abstract

[16] M. Andersson, V. Rathi, R. Thobaben, J. Kliewer, and M. Skoglund, "Equivocation of eve using two edge type ldpc codes for the binary erasure channel," in *IEEE Conference Record of the Forty Fourth Asilomar Conference on Signals, Systems and Computers*, November 2010.

[17] V. Rathi, R. Urbanke, M. Andersson, and M. Skoglund, "Rate-equivocation optimal spatially coupled ldpc codes for the bec wiretap channel," in *IEEE International Symposium on Information Theory Proceedings (ISI)*. IEEE, August 2011, pp. 2393–2397.

[18] V. Rathi, M. Andersson, R. Thobaben, J. Kliewer, and M. Skoglund, "Performance analysis and design of two edge-type ldpc codes for the bec wiretap channel," *IEEE Transactions on Information Theory*, vol. 59, no. 2, pp. 1048–1064, 2013.

[19] Nvidia. (2015) Nvidia CUDA - About CUDA. NVIDIA. [Online]. Available: https://developer.nvidia.com/about-cuda

[20] J. Sanders and E. Kandbrot, *CUDA by Example - An Introduction to General-Purpose CPU Programming*. Nvidia, 2011.

[21] M. Grassl. (2015, Jan) Code tables: Bounds on the parameters of various types of codes. Internet. [Online]. Available: http://www.codetables.de/

[22] P. Sweeney, *Error Control Coding - from Theory to Practice*. Wiley & Sons, Chichester, UK, 2002, 005.72 SWE.

[23] University of Sydney. (2015) Magma computation algebra. [Online]. Available: http://magma.maths.usyd.edu.au/magma/