CHALMERS

Upper bounds for block codes

Bojja Neelima

Rahman Syed Mustafizur

Master's Thesis
International Master's Program in
Digital Communication Systems and Technology.
Department of Signals and Systems,
Chalmers University of Technology
Göteborg, Sweden, 2005

Report Number
EX037/2005

Abstract

The main goal of this dissertation is to proffer an extensive table of upper bounds on $A(n,d)$ with the help of the values on $A(n,d,w)$. $A(n,d)$ denotes the maximal number of binary code words of length n and Hamming distance of at least d with no restriction on weight. $A(n,d,w)$ designates the maximal possible number of code words of length n, Hamming distance d apart and weight w of all codewords (number of nonzero components) in a constant weight binary code.

An (n,d,w) constant weight binary code is an (n, d) binary code in which all code words have same number w of *ones* and $(n-w)$ *zeros*.

The maximum possible sizes of binary codes $A(n,d)$ and constant weight binary codes $A(n,d,w)$ have been computed so far for all lengths $n \leq 28$ and $d \leq 14$. This paper presents a detailed survey of the computation of known upper bounds for the size of block codes and it also evolved around improving the previous research and presenting the improvements. Most improvements occur for $A(n,12,w)$, for $n=27$ and $n=28$. In addition, this paper also presents extended tables of the existing tables of upper bounds of block codes $A(n,d)$ for $n \leq 40$ and $d \leq 20$. It was achieved by making use of extended tables of upper bounds of constant weight binary codes $A(n,d,w)$ for $n \leq 40$, $d \leq 20$ and $w \leq 20$.

To obtain these results, we have implemented few theorems in the C programming language and these theorems are taken from [1].Also, the GLPK software has been used in solving the linear programming part, which facilitates solving problems with a large number of constraints. Further more a detailed survey on different types of codes is presented in this paper.

Acknowledgements

We would like to express our sincere thanks to Erik Agrell, our supervisor and examiner, who offered us the opportunity to participate in this interesting research project. This thesis work can not be prepared without his gracious supervision. All information and comments we have taken from him were very helpful to construct the work. His constant encouragement and great help has helped us to wind up this thesis.

We would also thank our friends especially Mr. Amin for his help and support.

Last but not the least; we would like to give special thanks to the parents, for their encouragement.

Author names and references are stated in alphabetical order.

Neelima Bojja
Syed Mustafizur Rahman.

CONTENTS

Chapter 1

1. Introduction

The ultimate aim of communication systems is to transmit information from the information source to the destination without any errors like noise, attenuation, bandwidth limitations, interference etc., which are introduced in the channel. One of the ways of detecting and minimizing these errors over a noisy communication channel is by channel coding.

The intention of reducing these errors has resulted in the invention of different types of codes like linear codes, block codes, convolutional codes, Hadamard codes, etc., each designed to meet specific transmission requirements in specific channels [29]. Block codes are one of the types of codes that can reduce the errors to a considerable extent. The quest running for decades in communication theory for finding the maximum possible size of block codes with given parameters has been quenched by computing upper and lower bounds for block codes. These upper and lower bounds are computed because the maximum possible size of block codes is in general not known.

An *(n,d)* is a binary code and it is a set of code words of length n, distance d. Let *A(n,d)* denote the maximum number of codewords in any linear or nonlinear binary code of length n and minimum distance d between code words.

An *(n,d,w)* constant weight binary code is a set of binary code words *(n,d)* in which all code words have same number *w* of *ones* and *(n-w) zeros*, where *n* is the length of the binary code, *d* is the Hamming distance and *w* is the weight of all code words(number of nonzero components). Let *A(n,d,w)* denote the maximum possible number of code words in a constant weight binary code.

Much research has been carried out in computing upper bounds for the size of block codes for all lengths $n \leq 28$ and minimum distances $d \leq 14$ [3]. Since 2001, there has not been much improvement in this research. This thesis is focused on reproducing the values of the

upper bounds on block codes $A(n,d)$ in table I in [1],and to extend this table to $n \leq 40$ and $d \leq 20$ with the help of the extended tables of constant weight codes $A(n,d,w)$ for $n \leq 40$, $d \leq 20$ and $w \leq 20$.

This thesis takes a structured approach to compute the known values of the upper bounds, starting with the application of few theorems that were derived by different authors [1] and working gradually to reach the goal. Initially theorems 1- 4 in [1] are implemented in the C programming language to reproduce table I in [1] and later this table has been improved and extended to $n \leq 40$ and $d \leq 20$.

The thesis is organized as follows:

Chapter 1 presents a sneak preview and a broad brush overview of the whole thesis work.

In Chapter 2, the basic idea of information theory, definition of the information and an expression for the measurement of the information content, definition of the entropy are provided under the section 2.1. Also, an analysis and elementary design of the elements of the overall communication system, with an explanation to some of the basic terms like Shannon's law, explained in the subsection 2.1.4.1 and SNR (Signal to- Noise ratio), explained in the subsection 2.1.5 are provided in this chapter. Few more concepts like coding, its necessity in communication systems, types of coding are also briefly discussed.

Chapter 3 provides a detailed coverage of block codes and classification there of. Some important terms such as Hamming distance and Hamming weight are explained in the sections 3.2 and 3.3 respectively. Different types of codes such as constant weight codes, linear codes, Hamming codes (an example of block codes), cyclic codes, BCH codes, Reed Solomon codes are briefly explained in this chapter. The significance of bounds on block codes is also elucidated in this chapter.

Chapter 4 gives a complete survey of the previous research results accomplished in the generation of upper bounds for block codes in the last decades.

Chapter 5 includes a mathematical treatment of linear programming. A few linear programming software packages are investigated in section5.2.In addition, this chapter also confers the experiences from using a few linear programming software packages. Also, the GLPK linear programming software package is described in section 5.3.

Chapter 6 is devoted to the computation of the known upper bounds for block codes, $A(n,d)$, for $n \leq 28$ and $d \leq 14$ using theorems 1-4 in [3], by utilizing bounds on constant weight codes $A(n,d,w)$, for $n \leq 28$ and $d \leq 14$ in [4].The above computation is explained in detail under the section 6.1.An overview on the analysis of upper bounds on the constant weight codes is presented in section 6.2 and the improvements upon the known upper bounds on $A(n,d,w)$ for $n \leq 28$ and $d \leq 14$ in [3] are discussed in detail in section 6.3.Also section 6.4 thoroughly discusses the improved values of known upper bounds for block codes, $A(n,d)$ for $n \leq 28$ and $d \leq 14$.This chapter also covers a discussion some problems that arose during the computation.

In Chapter 7, section 7.1 covers a discussion of the extended tables for constant weight codes, $A(n,d,w)$,from $n \leq 40$, $d \leq 20$ and $w \leq 20$. Subsection 7.1.1 discusses few problems that arose during the extension of tables of upper bounds for constant weight codes, $A(n,d,w)$,for $n \leq 40$, $d \leq 20$, $w \leq 20$. Also, Section 7.2 focuses on main part of this dissertation i.e. computation of upper bounds for block codes, $A(n,d)$, for $n \leq 40$, $d \leq 20$.The subsection 7.2.1 discusses some problems that arose during the computation of upper bounds for block codes, $A(n,d)$, for $n \leq 40$, $d \leq 20$.

Finally, chapter 8 is the epilogue of this dissertation, which includes conclusions in section 8.1 and ends with future works in the section 8.2.

Chapter 2

2. Basic idea of Information Theory

This chapter is completely devoted to some of the important terms in communication systems. It gives a brief introduction to information theory, provides definitions of information, entropy. An expression for the measurement of information content is also presented in this chapter. An analysis on the transfer of messages through the channels is discussed in the section 2.1.4.Definition of Signal to noise ratio *(SNR)* is provided in the section 2.1.4.1. This chapter provides an overview of coding, its necessity in communication system and also different types of coding are explained in this chapter.

2.1 Information Theory

The idea of quantitative measure of the information has been around for a while, to explain the aspects of information and communication. Claude Elwood Shannon in 1948 formulated principally a new field what is now called Information theory. Since then he went on to invent new disciplines of information theory and revolutionize the field of communications.

2.1.1 Information -Definition

Shannon defined the 'Information' as the symbols that contain unpredictable news i.e. he proposed the idea that information is based on uncertainty. In information theory, the term *information* is used in a special sense, it is measure of the freedom of choice with which a message is selected from the set of all possible messages thus it is distinct from the meaning, since it is entirely possible for a string of nonsense words and a meaningful sentence to be equivalent with respect to information content [37].

2.1.2 Measurement of the Information content

Shannon defined a quantity called *self-information.* The mathematical expression for the information content is [37]. Information theory numerically measures the quantity of the information in a message in bits.

$$i(A) = -\log_b P(A) \qquad (2\text{-}1)$$

Where:

A is an event, which is a set of outcomes of a random experiment,
P (A) denotes the probability of the occurrence of the event *A* and
i(A) is the self information associated with *A*.

The above concept explains that if the probability of an event is low (more uncertainty) in the communication channel, the amount of the self- information associated with it high information associated with it is high and vice versa.

2.1.2.1 Entropy-Definition

Shannon showed that if the experiment is a source that puts out the symbols A_i from a set A, then the *entropy* is a measure of the average number of binary symbols needed to code the output of the source [37].

Average self information:

$$H = -\sum P(A_i)\log_b P(A_i) \qquad (2\text{-}2)$$

2.1.3 Code-Definition

Definition: The set of binary sequences is called a Code [42].Codes is invented to correct errors on noisy communication channels. Suppose there a lot of important messages *(0's and 1's)* to be sent over the communication channel. There can be a possibility that the receiver will receive 1, when 0 is sent and receives 0 when 1 is sent. Say on an average 1 out of every 100 symbols will be in error that is for each bit there is a probability *P=0.01* that the channel will make a mistake and this channel is called *binary symmetric channel.* For in order to give some protection against errors on the communications channel, these messages are encoded into a code word i.e. A block of *k* message symbols $u = u_1 u_2 \ldots . u_k$ ($u_i = 0$ or 1) will be encoded into a code word $x = x_1 x_2 \ldots . x_n$ ($x_i = 0$ or 1) where $n \geq k$ these code words form a *code.* The figure 3.2 in section 3.1.1.1 clearly illustrates this [29].

2.1.4 Analysis of the transfer of messages through the channels

To get a better understanding of Information theory Shannon explained some of the details: Any communication system involves three steps 1.coding the message at its source 2.transmitting the message through a communication channel and 3.decoding the message at its destination. In the first step the message has to be in put into some kind symbolic representations that is in the form of codes and this information is transmitted via the channel where at the destination this message is decoded to extract the original data. The information may be corrupted by the noise during this process. Information theory further shows that this noise creates uncertainty as to the correspondence between the transmitted and the received signals. His theory replaces each element in this model with a mathematical model that describes the elements behavior within the system.

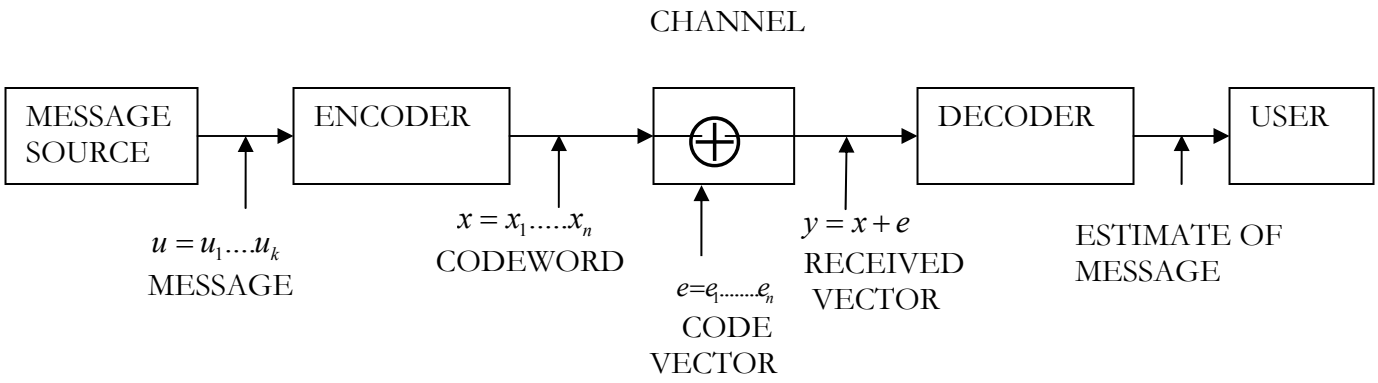

Figure 2.1 Block diagram of overall communication system.

An important theorem of information theory states that if a source with a given entropy feeds information to a channel with a given capacity, and if the source entropy is less than the channel capacity, a code exists for which the frequency of errors may be reduced as low as desired. If the channel capacity is less than the source entropy, no such code exists[35].

The ultimate limitation of a channel to efficiently and reliably transmit information without error is characterized by the channel capacity defined in the section 2.1.4.2.The task of providing high channel efficiency is the goal of coding techniques. The failure to meet perfect performance is measured by the bit-error-rate (BER). Typically BERs are of the order 10^{-6}. Shannon used mathematics to define the capacity of any communication channel to optimize the signal to noise ratio in his law. Before stating Shannon's law the signal to noise ratio is defined in the section 2.1.4.1.

2.1.4.1 Signal to- noise ratio

Definition: It is a measure of the received average- signal power to the noise power written as S/N or SNR. It is measured in decibels [35-36].There is a distinction between communication channel designed bit rate of so many bits and its actual information capacity. Information theory says that one need not lower the transmission rate to anything below the channel capacity to achieve smaller error probabilities [35].

$$\frac{S}{N} = \frac{(E_b R)}{(N_o W)} \qquad (2\text{-}3)$$

Where:

$\frac{E_b}{N_0}$ is the ratio of the bit energy to the noise power spectral density, in decibels.

$\frac{R}{W}$ is the spectral efficiency.

R is the code rate.

W is the bandwidth

2.1.4.2 Shannon's law

Capacity –Definition: This statement defines the theoretical maximum rate at which error-free digits can be transmitted over a bandwidth–limited channel in the presence of noise. The Shannon-Hartley formula for channel capacity is given by [35].

$$C = W \log_2\left(1 + \frac{S}{N}\right) \qquad (2\text{-}4)$$

Where:

C is the channel capacity in bits per second,

W is the bandwidth in Hertz (cycles/sec) and

$\frac{S}{N} = \frac{E_b R}{N_0 W}$ is the signal to noise ratio.

Explanation

When the information rate R, equals the channel capacity C, the curve separates the region of reliable communications from that region where reliable communications is not possible. The expression clearly states that reliable transmission (transmission with error probability less any given value) is possible even over noisy channels as long as the transmission rate is less than channel capacity [36]. The noisy channel coding theorem stated in the subsection 2.1.4.4, gives the capacity of a general discrete – memory less channel. The graph illustrates the relation between the capacity C and the signal to Noise ratio SNR[36].

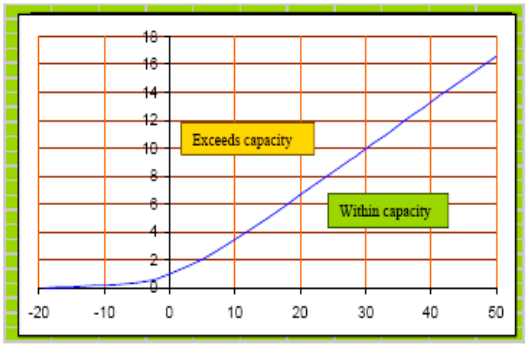

Figure 2.2 Graph showing the capacity against Signal to Noise ratio.

2.1.4.3 Mutual Information

The mutual information between two discrete random variables X and Y is denoted by *I(X;Y)* and defined by [35].

$$I(X;Y) = H(X) - H(X/Y) \qquad (2\text{-}5)$$

Where:

$H\left(\frac{X}{Y}\right)$ denotes the entropy (or uncertainty) of the random variable X after random variable Y is known.

$H(X) - H\left(\frac{X}{Y}\right)$ is the amount of the information provided by the random variable Y about the random variable X.

2.1.4.4 Noisy channel coding theorem

The maximum rate at which one can communicate over a discrete memory less channel and still make the error probability approach 0 as the code block length increases is called channel capacity. The capacity of a discrete memory less channel is given by [35].

$$C = \max_{P(x)} I(X;Y) \qquad (2\text{-}6)$$

Where:

$I(X;Y)$ is the mutual information between the channel input X and the output Y.

Explanation

If the transmission rate *R* is less than *C*, then for any $\delta > 0$ there exists a code with block length *n* large enough whose error probability is less than δ.If *R*>*C*, the error probability of any code with any block length is bounded away from 0.This theorem gives a fundamental limit on the possibility of reliable communication over a noisy channel [35].

2.2 Coding

This section presents definition of coding and its necessity in communications systems. Also types of coding are types are included in section 2.2.3.

2.2.1 Definition

The "Coding" is perceived as the operation of the identification of the symbols or the bit groups of one code to the symbols or the bit groups of other code. Information transmission through communication channels becomes considerably more difficult because of the disturbances and noises in the channel. It is an effective means by which errors can be detected and corrected in a communications channel [37].

2.2.2 Necessity of Coding

The necessity of coding arises first of all from need to adapt the form of message to the given communication channel intended for transformation. The main purpose of coding in communication systems, the altering of the characteristics of a signal is to make the signal more suitable for an intended application. One of the important applications of coding is to increase the overall bit error probability. As per Shannon's channel capacity theorem the probability of the error $P_b(E)$ can approach zero, provided the information rate is less than the channel capacity. For in order to make the probability of the error tend to zero "coding" is necessary[35].The only way to transmit messages reliably over a noisy communication channel at a positive rate without an exponential increase in the bandwidth is by coding.

2.2.3 Types of Coding

The term coding is applied to many operations within communications systems. Shannon's information theory deals with few of the types of coding namely, source coding (data compression), and channel coding (error protection) [9].

- **Source coding**

A form of Coding where an analog or digital source is altered in some way to make it best suited for transmission purposes.

- Reduces "size" of data.
- Analog- Encode analog source data into a binary format.
- Digital – Reduce the "size" of digital source data.

- **Channel coding**

It involves the addition of some extra bits to the transmitted data stream resulting in longer coded vector of symbols in order to provide a means of correcting transmission errors. Mostly involves operations on binary data.

- Increases "size" of data.
- Digital – add redundancy to identify and correct errors.
- Analog – represent digital values by analog signals.

2.2.4 Channel Coding

Channel Coding is most often applied to communications links in reducing the information rate and improve the reliability of the information being transferred. In information theory and coding there are two ways of detecting and correcting the errors namely-

2.2.4.1 Error Detection Coding

An error detection technique involves recognizing that part of the received information in error and if admissible it requests a repeat transmission or may simply inform the receiver that the transmission was corrupted (ARQ) Automatic Repeat Request system. There are three types of ARQ operations namely:

- **Stop and Wait ARQ**: One of the ARQ methods where the transmitter waits for an acknowledgement of correct reception known as (ACK). If the received message is in error it returns a NAK (negative acknowledgement).

- **Go Back n ARQ**: The method in which the transmitter continues to transmit messages in a sequence until a NAK is received. It identifies the message in error and the transmitter back tracks this message starting to retransmit all messages in the sequence from which the error occurred. No ACK'S are used and has less signaling overhead.

- **Selective ARQ**: by making the protocol slightly complex, and by providing buffer in the receiver and at the transmitter, it is desirable for the receiver to inform the transmitter of the specific message that is in error, making the necessity of the transmitter to send that particular error message.

2.2.4.2 Forward Error Correction Coding

It is a method of providing reliable digital data transmission and storage when the communication medium used has low Signal to Noise Ratio. In forward error correction coding technique, instead of transmitting the digital data in a raw bit form the data is encoded with extra bits at the source and these extra bits transform the data into valid code word in

the coding scheme. Hence the longer code word is transmitted and the receiver can decode it, to retrieve the desired information. The space of valid code words is smaller than the space of possible bit strings of that length therefore the destination can recognize invalid code words. This whole process takes place without recourse for retransmission hence named as *Forward Error Correction* also known as *error correction and error detection coding*. There are two types of channel codes namely [18&35].

1. Block Coding

In a block code the information sequence is broken into blocks of length *k* and each block is mapped into channel inputs of length *n*.This mapping is independent from the previous blocks i.e. there exists no memory from one block to another block[35] .Its is explained in detail in chapter 3.

2. Convolutional Coding

Convolution codes extend the concept of a block code to allow memory from block to block. Each encoded bit is therefore a linear combination of information symbols in the current block and a selected number of preceding blocks. In convolutional codes, there exists a shift register of length L. The information bits enter the shift register k_0 bits at a time and then n_0 bits which are linear combinations of various shift register bits are transmitted over the channel. These n_0 bits depend not only on the recent k_0 bits that just entered the shift register, but also on the $(L-1)k_0$ previous contents of the shift register that constitute its state. The quantity $m_c = L$ is defined as the constraint length of the convolutional code. The numbers of states of the convolutional code are equal to $2^{(L-1)k_0}$ [35].

2.2.5 Coding efficiency

The efficiency of a code is a measure of how well errors can be detected versus bit overhead required to implement the code. Certain code types are better at detecting errors than correcting them, and these are thus well suited to ARQ schemes explained in subsection 2.2.4.1 where we need to know an error has occurred, and our corrective action is to request a retransmission. Equally there are codes that are best suited for correcting and these will be used where no retransmission was possible. Example of this type of code is missile control systems [6].

Chapter 3

3. Block Coding

Block coding is a special case of error-control coding defined in section 2.2.4.2. This chapter includes basic idea of block codes, constant weight codes- definition, linear codes, Hamming codes, Cyclic codes, Bose–Chaudhuri-Hocquenghem (BCH) codes and Reed Solomon codes. It also defines some of the terms like Hamming distance, Hamming weight in sections 3.2 and 3.3 respectively. Also a brief idea of bounds on block codes is provided in section 3.10.

3.1 Linear Block Codes

Block codes operate on relatively large (typically, up to a couple of hundred bytes) message blocks, each of k information bits. The encoder transforms each message word independently into code word thus corresponding to 2^k different possible messages i.e. there are 2^k different possible code words at the encoder output .This set of 2^k code words of length n is called (n, k) block code.

The increase in block length means that the useful data rate (the information transfer rate) is reduced by a factor k/n called code rate. The additional bits are carefully chosen such that they help to differentiate one pattern of k bits in a block from a different pattern of k input bits. The redundancy and code rate of the block code are given below [6, 18 & 35].

$$\text{Code rate} = \frac{k}{n} \tag{3-1}$$

$$\text{Redundancy} = 1 - \frac{k}{n} \tag{3-2}$$

Our ability to detect errors depends on the rate. A low rate has a high detection probability, but a high redundancy.The class of block coding techniques includes categories shown in the diagram below.

Figure 3.1 Block diagram explaining block codes

3.1.1. Linear block codes - Definition

A block code is linear, if any linear combination of two code words is also a codeword. In the binary case that if c_i and c_j are code words then $c_i \oplus c_j$ is also a code word, where $\oplus$ denotes bit –wise modulo-2 addition [35].

3.1.1.1. Linear codes -Explanation

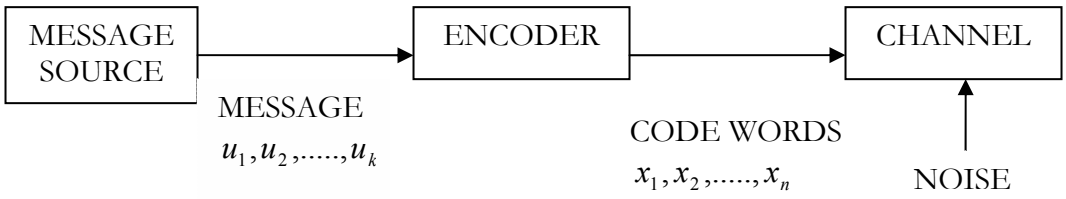

Figure 3.2 Block diagram explaining Linear codes

A block of messages bits $u = u_1, u_2, ..., u_k$ will be encoded a codeword $x = x_1, x_2, ..., x_n$ which form a code where $n \geq k$. A linear code is described by this method of encoding. The first part of the codeword consists of the message itself followed by *(n-k)* check symbols $x_{k+1}, ..., x_n$. the check symbols are chosen in such a way that the codewords satisfy

$$H \begin{pmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{pmatrix} = Hx^{tr} = 0 \qquad (3\text{-}3)$$

Where H is parity check matrix of the code of length $(n-k)\times n$ and is given by $H=[A\,|\,I_{n-k}]$ and A is some fixed $(n-k)\times k$ matrix of 0's and 1's and I_{n-k} is a unit matrix of length $(n-k)\times(n-k)$. '0' is a column matrix of length $(n-k)\times 1$. The Equation (3-3) is to be performed *modulo2*. The above definition clearly explains that a linear block code is a *k*-dimensional subspace of an *n*-dimensional space.

3.1.1.2 Classification of Linear block codes

The following tree diagram illustrates some examples of linear block codes and their relations[35].

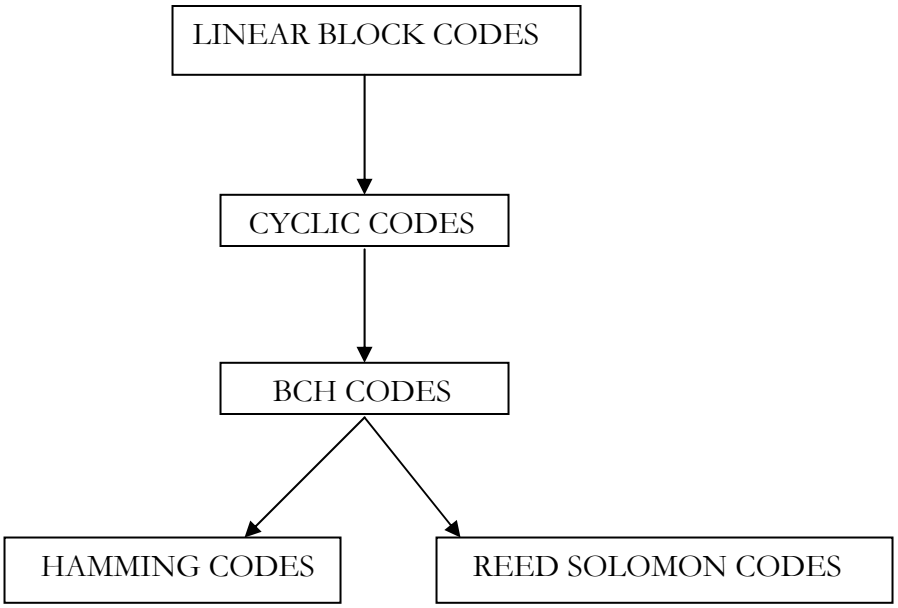

Figure 3.3 Tree diagram of classification of Block codes

3. 2 Hamming distance

This section defines Hamming distance and minimum Hamming distance.

3.2.1 Hamming distance-Definition

It is defined as the number of bit positions in which the corresponding bits of two binary words of the same length are different .i.e. The Hamming distance between 1011101 and 1001001 is two. It is denoted as $d(c_i,c_j)$ where c_i and c_j are code words. The greater the Hamming distance, the more dissimilar the code words and the better the chance of detecting or correcting errors [35].

3.2.2 Minimum Hamming distance

The minimum Hamming distance of a code is the minimum Hamming distance between all pairs of code words in a code [35].The errors detected and corrected by a block code are given below [6].

$$\text{Errors detected} = (d-1) \tag{3-4}$$

$$\text{Errors corrected} = \frac{(d-1)}{2} \tag{3-5}$$

Where:

d is the minimum Hamming distance.

3.3 Hamming weight

This section defines Hamming weight and minimum Hamming weight.

3.3.1 Hamming weight

The Hamming weight is defined as the number of "1" bits (i.e. *non zero* bits) of a binary code word in a bit sequence denoted by *w* (c_i) where c_i is the code word. For example the hamming weight of 1010 is 2 [35].

3.3.2 Minimum Hamming weight

The minimum Hamming weight of a code is the minimum of the weights of the code words except the all zero code words [35].

3.4 Constant weight Codes-Definition

Constant weight binary codes are an important class of error correcting codes .An *(n,d,w)* constant weight binary code is an *(n,d)* binary code in which all the code words has same number *w* of ones. Where *(n,d)* is a binary code and it is a set of code words of length *n*, distance *d* [29].

3.5 Cyclic Codes

Cyclic codes are a subclass of linear block codes for which easily implementable encoders and decoders exist. This section defines cyclic codes and also discusses the properties of cyclic codes.

3.5.1 Cyclic codes -Definition

A Cyclic code is a linear code with an extra condition that if C is a code word, a cyclic shift of it is also a codeword [29]. A cyclic shift of the codeword $C = (c_1, c_2, ..., c_{n-1}, c_n)$ is defined to be $C^{(1)} = (c_2, c_3, ..., c_{n-1}, c_n, c_1)$. A cyclic code can easily be created by

performing right cyclic shifts on an initial code word, developing a new code word with each shift, which continues until original code word is obtained i.e. simply all the elements of a code word are shifted one space to the right and the element on the end moves to the beginning of the codeword [35&28].

3.5.2 Properties of Cyclic codes

The structure of these cyclic codes was better understood and studied by mapping them to polynomials. The code word polynomial corresponding to $C = (c_1, c_2, ..., c_{n-1}, c_n)$ is defined by:

$$C(p) = \sum_{i=1}^{n} c_i p^{n-i} = c_1 p^{n-1} + c_2 p^{n-2} + ... + c_{n-1} p + c_n \quad (3\text{-}6)$$

The code word polynomial corresponding to $C^{(1)} = (c_2, c_3, ..., c_{n-1}, c_n, c_1)$

$$C^{(1)}(p) = c_2 p^{n-1} + c_3 p^{n-2} + ... + c_{n-1} p^2 + c_n p + c_1 \quad (3\text{-}7)$$

The following theorem clearly explains the structure of cyclic codes

Theorem: In any *(n,k)* cyclic code all code word polynomials are multiples of a polynomial of degree *(n-k)* of the form

$$g(p) = p^{n-k} + g_2 p^{n-k-1} + ... + g_{n-k} p + 1 \quad (3\text{-}8)$$

called the generator polynomial, where $g(p)$ divides $p^n + 1$. Further more for any information sequence $x = (x_1, x_2, ..., x_{k-1}, x_k)$ we have the information sequence polynomial $X(p)$ defined by $X(p) = x_1 p^{k-1} + x_2 p^{k-2} + ... + x_{k-1} p + x_k$ and the code word polynomial corresponding to x is given by $C(p) = X(p) g(p)$. The fact that any code word polynomial is the product of the generator polynomial and the information sequence polynomial, implies that $C = (c_1, c_2, ..., c_{n-1}, c_n)$ is the discrete convolution of the two sequences $x = (x_1, x_2, ..., x_{k-1}, x_k)$ and $g = (1, g_2, ..., g_{n-k}, 1)$. Further information of generating the generator matrix and encoding of cyclic codes can be found in [35].

3.6 BCH codes

They are a general family of the block codes called Bose, Chaudhuri, and Hocquenghem codes which can correct any number of errors (*t* errors) if the code word used is long enough. They are a subclass of cyclic codes which are versatile in design. BCH code can be defined by the following parameters for any *m* and *t* [29 & 35].

$$n = 2^m - 1 \quad (3\text{-}9)$$

$$n - k \le mt \quad (3\text{-}10)$$

$$d = 2t + 1 \qquad (3\text{-}11)$$

Where:

t is the number of errors that can be corrected by BCH code.

3.7 Hamming Codes

This section briefly explains the Hamming codes and also discusses the parameters used to define them.

3.7.1 Introduction to Hamming codes

One example of a cyclic code are the binary Hamming codes.. They were discovered independently in 1949 by Marcel Golay and in 1950 by Richard Hamming [28].

3.7.2 Hamming codes -Definition

A Binary Hamming code is constructed by a parity check matrix ***H***, in which the columns of this matrix consist of all nonzero binary vectors of length *r* (each vector used only once).The parameters of these particular codes are $n = 2^r - 1 (r \geq 2)$ (size of each code word), $k = 2^r - 1 - r$ (for determining the size of code), and d=3(minimum distance) for all Hamming codes. This makes binary Hamming codes [$n = 2^r - 1, k = 2^r - 1 - r$, *d=3*] as *perfect* single error correcting codes, and equivalent to cyclic codes. They can:

- Detect all single- and double-bit errors
- Correct all single-bit errors.

In order to decode Hamming codes, a simple *syndrome decoding method* is used, where the codeword received is multiplied by the Hamming Matrix and the result is the binary representation of where the error occurred [6, 35 & 28].

Rate of these codes is [35]:

$$R_C = \frac{2^r - r - 1}{2^r - 1} \qquad (3\text{-}12)$$

Where:

n is the total number of bits in the block.

k is the number of information bits in the block.

r is the number of check bits in the block, where $r = n - k$.

3.8 Reed-Solomon codes

This section provides an introduction to Reed Solomon codes. It also discusses the properties of Reed Solomon codes along with few of its applications

3.8.1 Introduction

Reed -Solomon (RS) codes are block-based error correcting codes with a wide range of applications in digital communications. The Reed-Solomon encoder takes a block of digital data and adds extra "redundant" bits. Errors occur during transmission or storage for a number of reasons (for example noise, scratches on a CD, etc). The Reed-Solomon decoder processes each block and attempts to correct errors and recover the original data. The number and type of errors that can be corrected depends on the characteristics of the Reed-Solomon code.

3.8.2 Properties of Reed -Solomon codes

Reed –Solomon codes operate at the symbol level rather than the bit level i.e. the incoming data stream is first packaged into small blocks, and these blocks are then treated as new set of *k* symbols of *s* bits each to be packaged into a super-coded block of n symbols, the result is that the decoder is able to detect and correct complete error blocks up to *t* symbols. It is thus possible for a whole symbols to be corrupted leading to a burst of errors, even still the receiver will be able to reinstate the correct information. Thus the Reed Solomon Code is specified as *RS(n ,k)* [6&35].

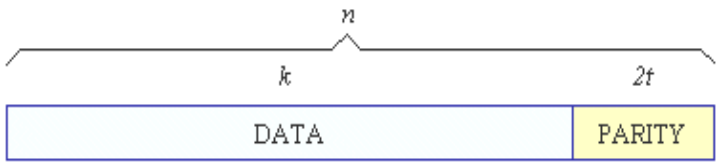

Figure 3.3 Block diagram of the typical Reed Solomon code.

A Reed Solomon is clearly defined by the following parameters-

$$n = 2^s - 1 \qquad (3\text{-}13)$$

$$d = 2t + 1 \qquad (3\text{-}14)$$

Where:

$2t = n-k$

n is the maximum code word length of a Reed Solomon code.

s is the symbol size in bits.

t defines the number of errors the code is capable of correcting.

3.8.3 Applications

- RS codes are often to correct errors in mobile radio systems where burst errors are common.
- Also used as a part of the error correcting mechanism in many storage devices (including tape, Compact Disk, DVD, barcodes, etc) and CD players
- They are also helpful in correcting the burst of errors caused by the inevitable scratches on the disk surface [6].

3.9 Brief idea of bounds on block codes

Say

A(n,d,w) = Maximum number of code words in any binary code of length *n*, constant weight *w* and minimum distance *d*.

A(n,d) = Maximum number of code words in any linear or nonlinear binary code of length *n* and minimum distance *d* between code words.

The quest of finding the largest possible size of *A(n,d)* in coding theory was quenched in the other way by finding lower and upper bounds on *A(n,d)* because the maximum size in general is not known. The upper bounds on *A(n,d)* were found having assumed that bounds on *A(n,d,w)* are known[4].

Chapter 4

4. Previous Works

In order to understand, analyze and improve an idea, the best way is to have basic knowledge of the background research and also grasping some of the ideas from them is very important. This chapter is reserved for previous research carried out in computing upper bounds for block codes and also discusses previous research on bounds. The question of finding the largest possible size $A(n,d)$ of an (n,d) binary code remained unanswered as it is impossible to find it. Though this is not solved, various lower and upper bounds were developed. Computation of upper bounds involved many analytical methods like linear programming and algebra. Upper bounds on $A(n,d)$ were developed with the help of upper bounds for constant weight codes, $A(n,d,w)$.

A number of attempts have been made and the upper bounds for error correcting and error detecting codes were found in [20] April 1950 by Hamming. Later in Dec 1959 [38] upper bounds for error detecting and error correcting codes of finite length were obtained by exploiting the geometric model of coding. Then in April 1962 [22] Johnson found a new upper bound for error –correcting codes which improves most values on Wax's bounds.

In the course of introducing this new upper bounds Johnson introduced two new bounds (A and B) on the maximum number of code words in constant weight minimum distance binary block code which had been obtained independently during the study of error correcting codes by Freiman. Johnson described an upper bound for $A(n,d,w)$ with added refinement in [23&26].

Then later in July 1964 [17] Freiman discussed some upper bounds on the size of constant weight codes with minimum distance d=2u, given by Johnson in [22] and derived one of its own, based on packing considerations. The first table of upper bounds on $A(n,d)$ and on $A(n,d,w)$,for, $n \leq 24$,$d \leq 10$ and $w \leq 14$ was found in 1977 [29, pg 684-691]. The improved version of these tables with some new values was presented in [7].Another update appeared in Honkala's Licentiate thesis [21, sec 6].

Some new upper bounds for constant weight codes (12 new upper bounds) appeared in [15] in July 1980.These bounds were obtained, using the updated tables of Graham and Sloane [19] as initial values in Johnson's formula [26].These updated tables in [19] are a revised version of the lower bounds in [7].Revised tables with improved lower bounds were further published in May 1980 [27 & 11].

The best known lower bounds on $A(n,d,w)$ for, $n \leq 28$, $d \leq 18$ and $w \leq 18$ were published in 1990 [12]. In [12] upper bounds are given only for those parameters where these bounds are known to coincide with lower bound Then in Jun 2000 new upper bounds on the size of constant weight binary codes, derived from bounds of spherical codes were presented in [2], in addition to this, Johnson bound given in [22] and the linear programming bound for constant weight codes were improved. Further in Nov 2000, improvements on the best known upper bounds on $A(n,d,w)$ for $n \leq 24$, $d \leq 12$ and $w \leq 14$ were made and also extended tables up to $n \leq 28$ and $d \leq 14$ were presented[1].

A Table of upper bounds on $A(n,d)$ for the range $n \leq 24$ and $d \leq 10$ was published in [16 p.248].A wider range of parameters was included [12].Updates to the combination of the upper bounds in [16] and [12] are published in [3].The table in [3] includes 4 new bounds and extends the range of the parameters compared to the previous tables on $A(n,d)$.

A new upper bound which is at least as good as Johnson bound for all values of n and d is given in April 2002. It is found that for small values of n and d the best known method to obtain upper bounds on $A(n,d)$ is linear programming and a new set of inequalities for linear programming. The improved upper bounds on $A(n,d)$ for $n \leq 28$ generated with these new inequalities were published in [32].An updated version of the tables of upper bounds for small general binary codes after the published tables in [7] were made available online [10].A new upper bound on the maximum size $A(n,d)$ was presented in [38]. This bound is based on block-diagonalsing the Terwilliger algebra of the Hamming scheme. It was shown that the bound strengthens the Delsarte bound and also the improved upper bounds on $A(n,d)$ for $n \leq 28$ and $d \leq 10$ were presented in [38].

Chapter 5

5. Linear Programming

This chapter begins with an overview that is intended to introduce linear programming in the section 5.1.The results of the survey on the linear programming software packages are discussed in the section 5.2.The very next part of this section presents the list of very few freely available linear programming software packages. The last part of this section explains one of the experiences faced in an attempt to use one of these software packages. The GLPK linear programming software package with supported reasons for choosing it are demonstrated in the section 5.3.

5.1 Linear programming

The basic purpose of linear programming is to optimize a linear objective function of continuous real variables, subject to linear constraints. The problem was first solved in 1940s when Dantzig developed the simplex method to solve planning problems for US Air force. Later it was applied to problems ranging from engineering and economics. Later other methods like ellipsoid and interior point method were developed. Simplex method and linear programming methods are still used in practice [31].

Primal problem of linear programming

Any minimization problem can be formulated as a maximization problem and vice versa by negating the sign of the objective function

Max	c^T x $\rightarrow$ (Objective function)	(5-1)
Subj	$Ax = b$	(5-2)
	$x_i \geq 0$	(5-3)

Where:

$c \in \Re^{m\times 1}$, $A \in \Re^{m\times n}$ is non singular and

$b \in \Re^{m\times 1}$ and $x \in \Re^{n\times 1}$ is variable.

The objective function attains optimal value at some optimal solution. In addition any point that satisfies the constraints is called feasible. Further information on linear programming can be found in [29].

5.2 Survey on linear programming software packages

An online survey made on the free accessible linear programming software packages has resulted in many products, which aid in maximizing and minimizing linear constraints subject to linear equalities and inequalities in continuous decision variables. Many of these products are also intended to solve large scale linear programming problems [31].

5.2.1 List of very few Linear Programming software packages

There are large numbers of software packages. Very few of the linear programming Software packages Linear Programming, Simplex /Interior Point method [31]:

- CPLEX (purpose: Linear programming , simplex/Interior Point method)
- LIPSOL (Matlab sparse LP by interior points)
- PCx, sparse interior-point code linear programming
- CLP, Simplex based linear programming solver from COIN-OR.
- Linprog, low dimensional linear programming in C(Seidel's algorithm, by Mike Hohmeyer).
- GLPK, GNU Linear programming Kit (ANSI C package for large scale linear and mixed integer linear programming) etc.

5.2.2 Experience faced while using few of the software packages

An attempt in using the PCx software package went vain as it does not serve the purpose of this thesis even though there are C libraries available in this package. The reason is that it doesn't work for higher number of constraints or variables and its limitation is up to 25 variables and worked very well for up to 25 variables. Part of our time is not spared in at least attempting to increase the limit of this software package as it is discovered to be very difficult and shifted in trying for another free available linear programming software package that meets the requirements of this thesis.

Next trail in using the GLPK (GNU linear programming kit) worked well even for larger number of variables and has a set of routines written in C language which is rudiment in this dissertation to solve the linear programming part.

5.3 GLPK (GNU Linear programming Kit)

The GLPK is widely available general purpose software package. It is destined for solving large scale linear programming (LP), mixed integer programming (MIP) and other related problems by means of revised simplex method with minimum effort .This package available with the documentation is currently developed, maintained and made available online for free use by Andrew Makhorin. It is a productive contributor for the people all over the world. It supports the GNU programming languages which are a subset of AMPL language. The impetus for choosing only this particular package in spite of not using all other packages even though some are very fast than this (for example CPLEX is 10-100 times faster) is that it meets some of our requirements like, it is a set of routines written in ANSI C programming language and organized in the form of callable library (it meets the requirement of this work as all the simulations of the thesis are actualized in C programming language) [31].

In addition to this GLPK software package also helps to find an optimum solution for large scale linear programming problems using simplex method, with up to several 100,000 constraints, variables with less effort. It is even faster than PCx software package and robust too. It is capacious, versatile, efficient and numerically more stable enough to characterize the sensitivity of optimum with respect to the changes in the data [30].For further information about the simplex method and how does this software package converts the linear programming problem into its standard form depending on the types of the variable and bounds of variable refer to the documentation of GLPK [31].

Understanding of the concepts of types of the variables and bounds of variables plays a vital role in getting appropriate results as per the need. Many errors arose at the time of this work when GLPK is implemented because of illusive understanding of above concepts. Rectifications of these bugs are very clearly discussed in the section 6.3.

Chapter 6

6. Upper Bounds

Having defined some basic concepts like block codes, convolutional codes, types of block codes, the main discussion of reproducing upper bounds on block codes can be started. This chapter includes computation of known upper bounds for block codes, *A(n,d)* for $n \leq 28$, $d \leq 14$. It also includes a detailed discussion of the improvements on known upper bounds for block codes which occurred due to the improvements on constant weight codes, *A(n,d,w)* for $n \leq 28$, $d \leq 14$ and $w \leq 14$ followed by the discussion of problems that arise when the theorems are applied in C programming.

6.1 Computation of the known upper bounds on block codes

This section is a practical treatment of computing known upper bounds for block codes *A (n, d)* for $n \leq 28$ and $d \leq 14$ .Sections 6.1.1 and 6.1.2 include reproducing upper bounds on block codes using theorems 1- 2 and 3- 4 in [3] respectively. These bounds are computed using known upper bounds on constant weight codes *A(n,d,w)* for $n \leq 28$, $d \leq 14$ and $w \leq 14$ in [4] . Motivated by the published bounds for block codes in [3] these bounds for *A(n,d)* are extended from $n \leq 40$ and $d \leq 20$ which are discussed in chapter 7. In order to reach this goal, the upper bounds for constant weight codes are to be extended to $n \leq 40$, $d \leq 20, w \leq 20$ which are discussed in section 7.1. Most improvements occurred when extending the tables of constant weight codes which are thoroughly discussed in section 6.2. These improvements on bounds for constant weight codes, *A(n,d,w)*, for $w=10$ & $w=11$ at $n=28$ and $d=14$ lead to the improvements on block codes *A(n,d)* for $n \leq 28$ and $d \leq 14$ which are discussed in section 6.3.

In other words, it can be said that the better bounds on *A(n,d,w)* lead to new bounds on *A(n,d)*. The interrelationship between *A(n,d,w)* and *A(n,d)* is clearly explained by Elias [8,pp.451-456] and Bassalygo[5].

Bassalygo-Elias inequality:

$$A(n,d) \le \frac{2^n}{\binom{n}{w}} A(n,d,w) \qquad (6\text{-}1)$$

6.1.1 Upper bounds on *A(n,d)*, for n ≤ 28 and d ≤ 14 using theorems 1 & 2

The superscripts on upper bounds produced in [3] indicates the method (theorem) from which they are computed, except the values with the superscript S. Bounds with the superscript S are simply taken from [3]. This section mainly presents a brief explanation of reproducing upper bounds on *A(n,d)* by applying theorems 1-4 [3] in the C programming language. The following bounds are due to Plotkin [34].

Theorem 1

$$A(n,d) \le 2A(n-1,d) \qquad (6\text{-}2)$$

$$A(n,d) \le 2\left\lfloor \frac{d}{2d-n} \right\rfloor \qquad \text{if } n < 2d \qquad (6\text{-}3)$$

$$A(n,d) \le 2n \qquad \text{if } n = 2d \qquad (6\text{-}4)$$

Theorem 2

The improved sphere packing bound by Johnson [12, pg.532] is given below.For every positive integer δ

$$A(n,2\delta) \le 2^{n-1}\left(\binom{n-1}{0} + \ldots\ldots\ldots\ldots + \binom{n-1}{\delta-1} + \frac{\binom{n-1}{\delta} - \binom{2\delta-1}{\delta-1} A(n-1,2\delta,2\delta-1)}{\left\lfloor \frac{(n-1)}{\delta} \right\rfloor} \right)^{-1} \qquad (6\text{-}5)$$

Theorem 2 taken from [3] was implemented in the C programming language to compute upper bounds with superscript 2. The bounds on *A(n,d,w)* required for computation are taken from [4]. As a result of implementation, it was observed that the upper bounds that were computed first did not match the upper bounds in [3]. When rechecked thoroughly, it was noticed that theorem 2 was stated incorrectly in [3]. It is thus rectified and presented correctly in this paper using [29].

One important result of the above theorem is A(24, 4) ≤ 344308, which was known to Johnson in 1971 [25, table I, p.472] but has been overlooked in later tables [12&16] [3].

6.1.2 Upper bounds on *A(n,d)*, for $n \leq 28$ and $d \leq 14$ using theorems 3& 4.

This section provides the definition of the distance distribution of a binary code and the linear programming bound introduced by Delsarte. Theorems 3 & 4 are also stated in this section.

6.1.2.1 Distance distribution of a binary code

Definition: The distance distribution of a binary code C consists of the numbers $A_0, A_1, \ldots\ldots, A_n$ and its sequence is defined as [3]:

$$A_i = \left|\left\{\left(c_1, c_2\right) \in C \times C : d\left(c_1, c_2\right) = i\right\}\right| / \left|C\right| \qquad (6\text{-}6)$$

For $i = 0,1,2,\ldots,n$, where $d\left(c_1, c_2\right)$ is the Hamming distance between code words, c_1 & c_2. Also, if d is odd then

$$A(n, d) = A(n+1, d+1) \qquad (6\text{-}7)$$

Also, $A_i = 0$ for all odd I, and for any (n,d) binary code with even d, there exists another (n,d) binary code with the same number of code words having even weight [3].

6.1.2.2 Upper bounds on *A(n,d)*, for $n \leq 28$, $d \leq 14$ using theorem 3.

Linear programming bound by Delsarte

Delsarte showed that the distance distribution of any code satisfies [3]:

$$\sum_{i=0}^{n} A_i P_k\left(i\right) \geq 0 \qquad (6\text{-}8)$$

Where:

k =0, 1, 2……, n

$P_k\left(x\right)$ is the Krawtchouk polynomial of degree k, given by [3]

$$P_k\left(x\right) = \sum_{j=0}^{k} (-1)^j \binom{x}{j} \binom{n-x}{k-j} \qquad (6\text{-}9)$$

Theorem 3:

For every even positive integer *d*

$$A(n,d) \le 1 + \left\lfloor \max\left(A_d + A_{d+2} + ... + A_{2\lfloor n/2 \rfloor} \right) \right\rfloor \qquad (6\text{-}10)$$

Subject to constraints

$$0 \le A_i \le A(n,d,i), \qquad i = d, d+2, ..., 2\left\lfloor \frac{n}{2} \right\rfloor \qquad (6\text{-}11)$$

$$\sum_{j=d/2}^{\left\lfloor \frac{n}{2} \right\rfloor} A_{2j} P_k(2j) \ge -\binom{n}{k}, \qquad k = 1, 2, ..., \left\lfloor \frac{n}{2} \right\rfloor. \qquad (6\text{-}12)$$

Implementation

The above theorem [3] is implemented in the C programming language to compute upper bounds on $A(n,d)$ for $n \le 28$, $d \le 14$ with superscript 3. At first these bounds were computed with the help of the values of $A(n,d,w)$ for $n \le 28$, $d \le 14$ and $w \le 14$ taken from [4]. Later these bounds on $A(n,d,w)$, for $n \le 28$, $d \le 14$ & $w \le 14$ were also computed when extending the tables for blockcodes. All values of $A(n,d,w)$ for all n up to 28 and all even *d* up to 14 and *w* ranging from $(d/2+1)$ to the integer part of $n/2$ are taken from [4] except a few values of $A(n,d,w)$ for w outside this interval or for odd d. These values are computed by implementing theorem 8 in [1] and the source for this idea was obtained from [3].

The linear programming in the above theorem is solved using GLPK software which is discussed in section 5.3. The first and foremost step taken to solve linear programming is to analyze the constraints for the type of the variable and the bounds of the variable. Depending on their type, the constraints are altered from their original system of equality to the standard form given in the documentation of the GLPK software. These set of constraints are solved using the C programming language which uses GLPK routines.

The variables in the above set of constraints fall in the category of double bounded variables. Many errors arose when theorem 3 was implemented because of misconception of the type of the variable and the bounds of the variable in the documentation of GLPK software. Rectification of these bugs is discussed in detail in section 6.4.

6.1.2.3 Upper bounds on *A(n,d)*, for $n \leq 28$ and $d \leq 14$ using theorem 4

The distance distribution of an (n, d) binary code of odd size M satisfies [1]

$$\sum_{j=d/2}^{\left\lfloor \frac{n}{2} \right\rfloor} A_{2j} P_k(2j) \geq \frac{1-M}{M} \binom{n}{k} \qquad k = 1,2,\ldots\ldots\ldots,\left\lfloor \frac{n}{2} \right\rfloor \tag{6-13}$$

While if $M \equiv 2 \pmod 4$, then for at least one $l \in \{0,\ldots\ldots, n\}$

$$\sum_{j=d/2}^{\left\lfloor \frac{n}{2} \right\rfloor} A_{2j} P_k(2j) \geq \frac{(2-M)\binom{n}{k} + 2P_k(l)}{M} \qquad k = 1,\ldots\ldots\ldots,\left\lfloor \frac{n}{2} \right\rfloor. \tag{6-14}$$

The bounds with the superscript 4 in the table I [3] are computed by implementing theorem 4. This theorem is merely based on verifying the bounds obtained by theorems 1, 2 & 3. The lowest bound obtained by the above theorems is checked to fall in the category of any one of the below inequalities and then is correspondingly verified by inserting that bound. If the resultant value of theorem 4 is not matched with the value obtained previously, then that bound is reduced by 1 and tried again by the above procedure. This procedure is repeated until the obtained value matches the inequalities.

Upper bounds with superscript S

The bounds with the superscript S are explained as bounds for specific parameters which do not follow theorems 1-4. Each of these values is taken from [3]. The source for these bounds is clearly explained in [3]

6.2 Discussion of problems when reproducing upper bounds on A(n,d)

The problems that arose when reproducing upper bounds on *A(n,d)*, for $n \leq 28$ and $d \leq 14$ are discussed in this section.

No errors occurred when theorems 1, 2[3] were implemented, but many errors arose when theorem 3 was implemented, i.e. the generated bounds were not the same as the bounds in [3],they were differing by one or two significant figures compared to the actual bounds. In a step to debug these errors, all of the constraints generated with the help of the krawtchouk polynomial for a specific value of *n* and *d* were printed out and the maximum value of the objective function was calculated by hand. It didn't help to rectify the bugs and this proved that the constraints generated were correct and there was something wrong in the linear programming implementation. The program found a feasible point where the objective function was higher than the true maximum. Careful review of the GLPK documentation proved that the implementation of the linear programming software package was wrong. The

bugs were due to misunderstanding of the concept of "the types of the variables and the bounds of the variables" in the GLPK documentation [30]. At first it was considered that the constraints fell in the category of "variable with upper bound and variable with lower bound," but later it was discovered as an error and it was found that the constraints fell in the category of "double bounded variable."

6.3 Improvements on known upper bounds on A(n,d,w)

The improvements on known upper bounds for constant weight codes, A*(n,d,w)* for $n \leq 40$, $d \leq 20$ and $w \leq 20$ are discussed in this section.

In order to compute the upper bounds of block codes *A(n,d)* for $n \leq 40$, $d \leq 20$, the existing tables on *A(n,d,w)* were extended to $n \leq 40$, $d \leq 20$, $w \leq 20$ by implementing corollary 5 and 6 and theorems 9,10,11,12,13,14,20 [1]. The result of the above implementation improved a couple of bounds on *A(n,d,w)*. Three new updates to bounds in these tables in [4] are presented in this section. These are bounds on *A(n,d,w)* for $n \leq 28$, $d \leq 14$ and $w \leq 14$, namely, $A(27,12,10) \leq 58$, $A(27,12,11) \leq 90$ and $A(28,12,11) \leq 147$. These improved bounds were generated using theorem 9.

6.4 Extended tables of upper bounds on A(n,d,w).

The extended tables of upper bounds on *A(n,d,w)* for $n \leq 40$, $d \leq 20$ & $w \leq 20$ are presented in the appendix. These tables were extended after reproducing the existing tables of constant weight codes, *A(n,d,w),* for $n \leq 40$, $d \leq 20$, $w \leq 20$. This is done in sequence of steps: At first theorems 5,9,20 were applied in C programming language and the lowest value obtained from these three theorems was taken as an initial constant weight bound. Secondly, this bound was taken as a reference and was made use in the other theorems mentioned in the section 6.3.The lowest resultant value of theorems 10,11,12,13,14 was taken as the upper bound for the constant weight code.

6.4.1 Discussion of Problems while extending the tables of upper bounds on A(n,d,w)

The problems that arose when extending the tables of upper bounds on A(n,d,w) for $n \leq 40$, $d \leq 20$, $w \leq 20$ are thoroughly discussed in this section. When the existing tables of constant weight codes were extended, many problems occurred, e.g. the bounds generated on *A(n,d,w)* for $n \leq 28$ and $d \leq 14$ were greater than the known bounds in [4]. In order to debug these errors, the implementation of corollary 5 [1] and theorems 9, 10, 11, 12, 13, 14 and 20[1] was rechecked and there was an error in the implementation of theorem 11[1]. Initially the output of corollary 6[1] is taken, but then later this idea is changed by considering the least values obtained from the implementation of all of the theorems, corollary 6, 32(partially), 33 and 35 [1]. Better bounds for constant weight codes were obtained in the result. Another problem in theorem 9[1] was that there were a lot of recursions (it takes long time to compute the values). After changing the code, time was saved and computation was fast.

Chapter 7

7. Extending the table of upper bounds for block codes

This chapter mainly focuses on extending the block codes on $A(n,d)$, for $n \leq 40$, $d \leq 20$, with the help of the extended tables of constant weight codes, $A(n,d,w)$, for $n \leq 40$, $d \leq 20$, $w \leq 20$ discussed in the previous chapter in section 6.4.1.Section 7.1 discusses the improvements on known upper bounds on $A(n,d)$,for $n \leq 28$ and $d \leq 14$.The problems that arose while computing upper bounds on $A(n,d)$, for $n \leq 40$, $d \leq 20$, $w \leq 20$ are explained in subsection 7.2.1.

7.1 Improvements on known upper bounds on A(n,d)

The discussion of improvements on known upper bounds for block codes, $A(n,d)$ for $n \leq 28$ and $d \leq 14$ is included in this section.

The three new bounds on $A(n,d,w)$, for $n \leq 28$, $d \leq 14$ and $w \leq 14$ namely, $A(27,12,10) \leq 58$, $A(27,12,11) \leq 90$ and $A(28,12,11) \leq 147$ did not improve any bounds on $A(n,d)$, for $n \leq 28$ and $d \leq 14$.

7.2 Computation of upper bounds on A(n,d), for n≤ 40 and d≤ 20.

The bounds on $A(n,d,w)$ for $n \leq 40$, $d \leq 20$ & $w \leq 20$, generated in section 6.4 were used to compute the upper bounds on $A(n,d)$,for $n \leq 40$ *and* $d \leq 20$. These upper bounds were produced by applying theorems 1-4[3] in the C programming language. Initially theorems 1-3[3] were applied in the C programming language and the lowest value computed from these three theorems is taken as an upper bound. Later this lowest value was checked to fall in any one of the inequalities of theorem 4 and it is correspondingly verified by inserting that value. If the resultant value of the theorem 4 is not matched with the bound obtained previously, then that bound is reduced by 1 and the above procedure is repeated until the obtained value matches the inequalities. The tables of upper bounds on $A(n,d)$, for $n \leq 40$, $d \leq 20$ is presented in this section.

TABLE I

A TABLE OF BOUNDS ON A(n,d).

n	*d*								
	4	**6**	**8**	**10**	**12**	**14**	**16**	**18**	**20**
6	4^{1}	2^{1}							
7	8^{1}	2^{1}							
8	16^{1}	2^{1}	2^{1}						
9	20^{4}	4^{1}	2^{1}						
10	40^{1}	6^{1}	2^{1}	2^{1}					
11	72^{S}	12^{1}	2^{1}	2^{1}					
12	144^{S}	24^{1}	4^{1}	2^{1}	2^{1}				
13	256^{3}	32^{S}	4^{1}	2^{1}	2^{1}				
14	512^{3}	64^{3}	8^{1}	2^{1}	2^{1}	2^{1}			
15	1024^{2}	128^{3}	16^{1}	4^{1}	2^{1}	2^{1}			
16	2048^{2}	256^{2}	32^{1}	4^{1}	2^{1}	2^{1}	2^{1}		
17	3276^{3}	340^{S}	37^{S}	6^{1}	2^{1}	2^{1}	2^{1}		
18	6552^{1}	680^{1}	72^{S}	10^{1}	4^{1}	2^{1}	2^{1}	2^{1}	
19	13104^{1}	1288^{4}	144^{4}	20^{1}	4^{1}	2^{1}	2^{1}	2^{1}	
20	26208^{1}	2372^{4}	279^{3}	40^{1}	6^{1}	2^{1}	2^{1}	2^{1}	2^{1}
21	43689^{4}	4096^{S}	512^{S}	48^{S}	8^{1}	4^{1}	2^{1}	2^{1}	2^{1}
22	87378^{1}	6941^{4}	1024^{3}	88^{S}	12^{1}	4^{1}	2^{1}	2^{1}	2^{1}
23	173491^{3}	13774^{4}	2048^{3}	150^{4}	24^{1}	4^{1}	2^{1}	2^{1}	2^{1}
24	344308^{2}	24106^{4}	4096^{2}	280^{3}	48^{1}	6^{1}	4^{1}	2^{1}	2^{1}
25	599185^{4}	48148^{3}	6425^{4}	549^{4}	56^{S}	8^{1}	4^{1}	2^{1}	2^{1}
26	1198370^{1}	86132^{4}	10336^{4}	1029^{4}	98^{S}	14^{1}	4^{1}	2^{1}	2^{1}
27	2396740^{1}	162400^{4}	17804^{3}	1764^{3}	169^{4}	28^{1}	6^{1}	4^{1}	2^{1}
28	4793480^{1}	291269^{4}	32205^{4}	3200^{3}	288^{3}	56^{1}	8^{1}	4^{1}	2^{1}
29	8388608^{3}	581825^{4}	58096^{3}	6361^{4}	572^{3}	72^{3}	10^{1}	4^{1}	2^{1}
30	16777215^{3}	1110093^{4}	110681^{4}	12441^{4}	1124^{3}	128^{4}	16^{1}	6^{1}	4^{1}
31	33554430^{1}	2032919^{3}	185452^{3}	22293^{4}	1821^{4}	194^{4}	32^{1}	6^{1}	4^{1}
32	67108860^{1}	3710516^{3}	351908^{3}	40866^{3}	3081^{4}	314^{4}	64^{1}	8^{1}	4^{1}
33	119304645^{4}	7417221^{4}	621377^{4}	80230^{4}	5820^{3}	628^{3}	81^{4}	12^{1}	4^{1}
34	238609290^{1}	14589012^{4}	1203920^{3}	149100^{3}	11640^{4}	1253^{4}	141^{4}	18^{1}	6^{1}
35	475815004^{3}	26460065^{4}	2122136^{4}	238969^{4}	21273^{4}	2004^{3}	213^{4}	36^{1}	8^{1}
36	948239532^{2}	48457828^{4}	3945972^{4}	450436^{4}	36608^{3}	3176^{3}	352^{3}	72^{1}	10^{1}
37	1717986917^{4}	95392922^{4}	7023248^{3}	755268^{3}	66940^{3}	5256^{3}	704^{3}	92^{3}	12^{1}
38	3435973834^{1}	185174115^{3}	13095832^{4}	1421104^{3}	133819^{3}	10209^{4}	1408^{3}	157^{4}	20^{1}
39	6871947668^{1}	351813133^{4}	24866373^{4}	2584109^{4}	254498^{4}	20117^{4}	2212^{4}	229^{4}	40^{1}
40	13743895336^{1}	625445573^{4}	47234313^{4}	4563008^{4}	458265^{4}	36041^{4}	3508^{4}	349^{4}	80^{1}

7.2.1 Discussion of problems during the computation of upper bounds on A(n,d).

The problems that arose while computing the upper bounds on $A(n,d)$ for $n \leq 40$ and $d \leq 20$ are discussed in this subsection. When computing upper bounds on $A(n,d)$ for higher values of n and for lower value d (say for example $A(40,4)$), it was observed that the value of the computed upper bounds was too high. These higher values of upper bounds created an error of numerical instability in linear programming in theorems 3 and 4 [3].
This problem was solved by calling C routines (*lpx_scale_prob(lp) or lpx_set_in_parm(lp,LPX_K_PRESOL,1)*).These routines help in scaling of the constants in the linear inequalities.

Chapter 8

8. Conclusions

This section presents the conclusions in producing upper bounds on $A(n,d)$, for $n \leq 40$ *and* $d \leq 20$ and also future works.

8.1 Conclusions

The new upper bounds for block codes, $A(n,d)$, for $n \leq 40$ and $d \leq 20$ were computed with the help of the extended tables of upper bounds on $A(n,d,w)$, for $n \leq 40$, $d \leq 20$ and $w \leq 20$.Upper bounds on $A(n,d)$, for $n \leq 28$ and $d \leq 14$ were also re-produced and are presented in this paper. Three new updates on $A(n,d,w)$ for $n \leq 28$, $d \leq 14$ and $w \leq 14$ were computed namely, $A(27,12,10) \leq 58$, $A(27,12,11) \leq 90$ and $A(28,12,11) \leq 147$. The new updates for constant weight codes, $A(n,d,w)$ for $n \leq 28$, $d \leq 14$ and $w \leq 14$ didn't improve any bounds on block codes, $A(n,d)$ for $n \leq 28$ and $d \leq 14$.

8.2 Future works

Bounds on $A(n,d,w)$ may be still improved(i.e a tight bound can be produced) by implementing theorems 36 and 37 [1]. But this may consume a lot of time when implemented for higher values of n. These bounds on $A(n,d,w)$ may also improves bounds *on* $A(n,d)$ but not to the large extent.

APPENDIX - I

A TABLE OF UPPER BOUNDS ON *A(n,4,w)*

n	*w*									
	3	4	5	6	7	8	9	10	11	12
6	4^{5}									
7	7^{5}									
8	8^{5}	14^{5}								
9	12^{9}	18^{9}								
10	13^{9}	30^{9}	36^{9}							
11	17^{9}	35^{9}	66^{9}							
12	20^{9}	51^{9}	84^{9}	132^{9}						
13	26^{9}	65^{9}	132^{9}	182^{9}						
14	28^{9}	91^{9}	182^{9}	308^{9}	364^{9}					
15	35^{9}	105^{9}	271^{13}	455^{9}	660^{9}					
16	37^{9}	140^{9}	336^{9}	722^{9}	1040^{9}	1320^{9}				
17	44^{9}	157^{9}	476^{9}	952^{9}	1753^{9}	2210^{9}				
18	48^{9}	198^{9}	565^{9}	1428^{9}	2448^{9}	3944^{9}	4420^{9}			
19	57^{9}	228^{9}	752^{9}	1789^{9}	3876^{9}	5814^{9}	8326^{9}			
20	60^{9}	285^{9}	912^{9}	2506^{9}	5111^{9}	9690^{9}	12920^{9}	16652^{9}		
21	70^{9}	315^{9}	1197^{9}	3192^{9}	7518^{9}	13416^{9}	22610^{9}	27132^{9}		
22	73^{9}	385^{9}	1386^{9}	4389^{9}	10032^{9}	20674^{9}	32794^{9}	49742^{9}	54264^{9}	
23	83^{9}	419^{9}	1771^{9}	5313^{9}	14421^{9}	28842^{9}	52833^{9}	75426^{9}	104006^{9}	
24	88^{9}	498^{9}	2011^{9}	7084^{9}	18216^{9}	43263^{9}	76912^{9}	126799^{9}	164565^{9}	208012^{9}
25	100^{9}	550^{9}	2490^{9}	8379^{9}	25300^{9}	56925^{9}	120175^{9}	192280^{9}	288179^{9}	342843^{9}
26	104^{9}	650^{9}	2860^{9}	10790^{9}	31122^{9}	82225^{9}	164450^{9}	312455^{9}	454480^{9}	624387^{9}
27	117^{9}	702^{9}	3510^{9}	12870^{9}	41618^{9}	105036^{9}	246675^{9}	444015^{9}	766935^{9}	1022580^{9}
28	121^{9}	819^{9}	3931^{9}	16380^{9}	51480^{9}	145663^{9}	326778^{9}	690690^{9}	1130220^{9}	1789515^{9}
29	134^{9}	877^{9}	4749^{9}	18999^{9}	67860^{9}	186615^{9}	469358^{9}	947656^{9}	1820910^{9}	2731365^{9}
30	140^{9}	1005^{9}	5262^{9}	23745^{9}	81424^{9}	254475^{9}	622050^{9}	1408074^{9}	2584516^{9}	4552275^{9}
31	155^{9}	1085^{9}	6231^{9}	27187^{9}	105156^{9}	315518^{9}	876525^{9}	1928355^{9}	3968208^{9}	6676666^{9}
32	160^{9}	1240^{9}	6944^{9}	33232^{9}	124283^{9}	420624^{9}	1121841^{9}	2804880^{9}	5609760^{9}	10581888^{9}
33	176^{9}	1320^{9}	8184^{9}	38192^{9}	156665^{9}	512667^{9}	1542288^{9}	3702075^{9}	8414640^{9}	15426840^{9}
34	181^{9}	1496^{9}	8976^{9}	46376^{9}	185504^{9}	665826^{9}	1936742^{9}	5243779^{9}	11442777^{9}	23841478^{13}
35	197^{9}	1583^{9}	10472^{9}	52360^{9}	231880^{9}	811580^{9}	2589323^{9}	6778597^{9}	16684751^{9}	33374766^{9}
36	204^{9}	1773^{9}	11397^{9}	62832^{9}	269280^{9}	1043460^{9}	3246320^{9}	9321562^{9}	22184499^{9}	50054253^{9}
37	222^{9}	1887^{9}	13120^{9}	70281^{9}	332112^{9}	1245420^{9}	4289778^{13}	12011384^{9}	31354344^{9}	68402205^{9}
38	228^{9}	2109^{9}	14341^{9}	83093^{9}	381525^{9}	1577532^{9}	5258440^{9}	16301156^{9}	41493872^{9}	99288756^{9}
39	247^{9}	2223^{9}	16449^{9}	93216^{9}	462946^{9}	1859934^{9}	6835972^{9}	20507916^{9}	57795007^{9}	134855084^{9}
40	253^{9}	2470^{9}	17784^{9}	109660^{9}	532662^{9}	2314730^{9}	8266373^{9}	27343888^{9}	74574240^{9}	192650023^{9}

Continue Bounds on *A(n,4,w)*

n	*w*							
	13	14	15	16	17	18	19	20
26	685686^{9}							
27	1296803^{9}							
28	2202480^{9}	2593606^{9}						
29	3991995^{9}	4562280^{9}						
30	6303150^{9}	8554273^{13}	9124560^{9}					
31	10855423^{13}	13956975^{9}	17678830^{9}					
32	16434870^{9}	24812395^{9}	29774880^{9}	35357660^{9}				
33	26861715^{9}	38739336^{9}	54587269^{9}	61410689^{20}				
34	40347120^{9}	65235593^{9}	87809161^{9}	115997946^{9}	122821378^{9}			
35	64188594^{9}	100867800^{9}	152216383^{9}	192082539^{9}	238819300^{9}			
36	92422428^{9}	165056384^{9}	242082720^{9}	342486861^{9}	406763023^{9}	477638600^{9}		
37	142462104^{9}	244259274^{9}	407139080^{9}	559816290^{9}	745412579^{9}	836123991^{9}		
38	199944906^{9}	386682853^{9}	618790160^{9}	966955315^{9}	1251354060^{9}	1573648777^{9}	1672247982^{9}	
39	297866268^{9}	556989381^{9}	1005375417^{9}	1508301015^{9}	2218309252^{9}	2711267130^{9}	3230121173^{9}	
40	414938720^{9}	851046480^{9}	1485305016^{9}	2513438542^{9}	3548943564^{9}	4929576115^{9}	5707930800^{9}	6460242346^{9}

A TABLE OF UPPER BOUNDS ON *A(n,6,w)*

n	*w*								
	4	5	6	7	8	9	10	11	12
8	2^{10}								
9	3^5								
10	5^5	6^5							
11	6^5	11^5							
12	9^5	12^5	22^5						
13	13^5	18^{21}	26^9						
14	14^5	28^{20}	42^{20}	42^{21}					
15	15^5	42^9	70^{20}	78^9					
16	20^9	48^9	112^9	138^9	150^{20}				
17	20^{21}	68^9	136^9	228^T	280^T				
18	22^9	72^9	199^T	349^9	428^{20}	425^{20}			
19	25^{21}	83^9	228^9	520^{20}	718^T	789^{20}			
20	30^9	100^9	276^9	651^9	1107^{14}	1363^{20}	1403^1		
21	31^9	126^9	350^9	828^9	1695^{14}	2359^T	2685^T		
22	37^9	136^9	462^9	1100^9	2277^9	3766^T	4415^T	5064^{20}	
23	40^9	170^9	521^9	1518^9	3162^9	5819^9	7521^{20}	7953^{20}	
24	42^9	192^9	680^9	1786^9	4554^9	8432^9	12186^{14}	14682^9	15906^{20}
25	50^9	210^9	800^9	2428^9	5581^9	12620^{14}	19037^{14}	24630^{20}	30587^9
26	52^9	260^9	910^9	2971^9	7891^9	16122^9	28893^{14}	42080^T	50169^T
27	54^9	280^9	1170^9	3510^9	10027^9	23673^9	43529^9	66079^{20}	84574^{20}
28	63^9	302^9	1306^9	4680^9	12285^9	31195^9	63756^{14}	104231^{20}	142117^{14}
29	65^9	364^9	1459^9	5410^9	16965^9	39585^9	90465^9	164749^{20}	228441^{20}
30	67^9	390^9	1820^9	6252^9	20287^9	56550^9	118755^9	246722^9	359452^{20}
31	76^9	415^9	2015^9	8060^9	24226^9	69877^9	175305^9	334673^9	566481^{20}
32	80^9	486^9	2213^9	9211^9	32240^9	86136^9	223606^9	509977^9	868434^{14}
33	82^9	528^9	2673^9	10432^9	37995^9	118213^9	284248^9	670818^9	1299697^{14}
34	92^9	557^9	2992^9	12983^9	44336^9	143536^9	401924^9	878584^9	1900651^9
35	96^9	644^9	3249^9	14960^9	56800^9	172417^9	502376^9	1278849^9	2562536^9
36	99^9	691^9	3864^9	16709^9	67320^9	227200^9	620701^9	1644139^9	3836547^9
37	111^9	732^9	4261^9	20424^9	77279^9	276760^9	840640^9	2087812^9	5069428^9
38	114^9	842^9	4636^9	23131^9	97014^9	326289^9	1051688^9	2904029^9	6611404^9
39	117^9	889^9	5473^9	25829^9	112763^9	420394^9	1272527^9	3728712^9	9438094^9
40	130^9	936^9	5926^9	31274^9	129145^9	501168^9	1681576^9	4627370^9	12429040^9

Continue Bounds on *A(n,6,w)*

n	*w*							
	13	14	15	16	17	18	19	20
26	61174^9							
27	91080^{20}							
28	164220^{20}	169740^{20}						
29	289604^{20}	322266^{20}						
30	510937^{20}	603028^{20}	583148^{20}					
31	803583^{20}	1014649^{20}	1129849^9					
32	1264543^{20}	1746057^{14}	2015135^{20}	1990688^{20}				
33	1990876^{20}	2862969^{20}	3613792^{20}	3779731^{20}				
34	3133015^{20}	4514363^{20}	6404304^{20}	6844600^{20}	6902118^{20}			
35	4921126^{20}	7112473^{20}	10298390^{20}	12520303^{20}	13398784^{20}			
36	7096253^9	11194209^{20}	16294029^{20}	21493741^{14}	23733599^{20}	25499492^{20}		
37	10919403^9	17586482^{20}	25725823^{20}	36066595^{14}	43687624^{20}	46674812^{20}		
38	14818328^9	26859872^{14}	40541832^{20}	59306598^{20}	76749715^{20}	83423603^{20}	86716641^{20}	
39	19834212^9	40225345^{14}	63755301^{20}	93932811^{20}	130488792^{14}	152095473^{20}	168997893^{20}	
40	29040289^9	56669177^9	99984504^{20}	148368819^{20}	217280568^{20}	270368316^{20}	297181571^{20}	328206021^{14}

A TABLE OF UPPER BOUNDS ON *A(n,8,w)*

n	*w*								
	5	6	7	8	9	10	11	12	13
10	2^{5}								
11	2^{10}								
12	3^{5}	4^{5}							
13	3^{10}	4^{10}							
14	4^{10}	7^{5}	8^{5}						
15	6^{5}	10^{5}	15^{5}						
16	6^{10}	16^{5}	16^{5}	30^{5}					
17	7^{14}	17^{5}	24^{21}	34^{9}					
18	9^{10}	21^{9}	39^{9}	54^{9}	68^{9}				
19	12^{5}	28^{9}	57^{9}	92^{9}	114^{9}				
20	16^{5}	40^{9}	80^{9}	142^{9}	195^{11}	228^{9}			
21	21^{5}	56^{9}	120^{9}	210^{9}	320^{11}	389^{T}			
22	21^{21}	77^{9}	176^{9}	330^{9}	493^{11}	641^{11}	724^{T}		
23	23^{5}	80^{9}	253^{9}	506^{9}	742^{T}	1078^{T}	1309^{T}		
24	24^{5}	92^{9}	274^{9}	759^{9}	1078^{T}	1624^{T}	2188^{11}	2576^{20}	
25	30^{9}	100^{9}	328^{9}	856^{9}	1539^{T}	2446^{T}	3554^{T}	4169^{11}	
26	30^{21}	130^{9}	371^{9}	1066^{9}	2160^{11}	3691^{T}	5315^{20}	6834^{11}	7083^{T}
27	32^{9}	135^{9}	500^{9}	1252^{9}	2914^{11}	5260^{20}	7837^{11}	10547^{11}	11981^{T}
28	33^{9}	149^{9}	540^{9}	1750^{9}	3895^{9}	7367^{T}	11939^{11}	17299^{20}	21736^{T}
29	39^{9}	159^{9}	617^{9}	1957^{9}	5296^{14}	10320^{20}	17736^{20}	25730^{20}	36498^{20}
30	42^{9}	195^{9}	681^{9}	2313^{9}	6523^{9}	14440^{20}	24890^{20}	38878^{20}	59299^{20}
31	43^{9}	217^{9}	863^{9}	2638^{9}	7967^{9}	19478^{14}	34898^{20}	60694^{20}	89152^{20}
32	44^{9}	229^{9}	992^{9}	3452^{9}	9379^{9}	25494^{9}	48872^{20}	86570^{20}	140580^{14}
33	51^{9}	242^{9}	1079^{9}	4092^{9}	12657^{9}	30950^{9}	68324^{20}	121856^{20}	209874^{14}
34	54^{9}	289^{9}	1175^{9}	4585^{9}	15458^{9}	43033^{9}	94204^{14}	171169^{20}	305660^{14}
35	56^{9}	315^{9}	1445^{9}	5140^{9}	17830^{9}	54103^{9}	126393^{14}	239936^{20}	434507^{20}
36	57^{9}	336^{9}	1620^{9}	6502^{9}	20560^{9}	64188^{9}	168010^{14}	335546^{20}	613457^{20}
37	65^{9}	351^{9}	1776^{9}	7492^{9}	26730^{9}	76072^{9}	215905^{9}	467976^{20}	863628^{20}
38	68^{9}	411^{9}	1905^{9}	8436^{9}	31632^{9}	101574^{9}	262794^{9}	650582^{20}	1212483^{20}
39	70^{9}	442^{9}	2289^{9}	9286^{9}	36556^{9}	123364^{9}	360126^{9}	854080^{9}	1697476^{20}
40	72^{9}	466^{9}	2525^{9}	11445^{9}	41271^{9}	146224^{9}	448596^{9}	1200420^{9}	2369340^{20}

Continue Bounds on *A(n,8,w)*

n	*w*						
	14	15	16	17	18	19	20
28	23265^{T}						
29	39570^{20}						
30	68312^{20}	77927^{20}					
31	114471^{20}	128820^{20}					
32	190864^{14}	217646^{20}	236998^{20}				
33	303505^{20}	369898^{20}	404550^{20}				
34	463168^{20}	606753^{20}	690944^{20}	751168^{20}			
35	705369^{14}	1008678^{20}	1241052^{20}	1385964^{20}			
36	1100376^{20}	1571965^{14}	2031818^{20}	2374588^{20}	2637516^{20}		
37	1570752^{20}	2466859^{14}	3392826^{14}	4224238^{20}	4614454^{20}		
38	2232508^{20}	3736123^{14}	5403761^{20}	6811026^{20}	7896061^{20}	8390801^{20}	
39	3161117^{20}	5525459^{14}	8473650^{20}	11103454^{20}	14294593^{20}	16071975^{20}	
40	4460740^{20}	8207579^{20}	12892738^{14}	18458566^{20}	23704627^{20}	28076842^{20}	31583317^{20}

A TABLE OF UPPER BOUNDS ON *A(n,10,w)*

n	*w*						
	6	7	8	9	10	11	12
12	2^{5}						
13	2^{5}						
14	2^{10}	2^{20}					
15	3^{5}	3^{5}					
16	3^{10}	4^{5}	4^{20}				
17	3^{14}	5^{5}	6^{5}				
18	4^{10}	6^{5}	9^{5}	10^{5}			
19	4^{10}	8^{5}	12^{10}	19^{5}			
20	5^{10}	10^{10}	17^{21}	20^{5}	38^{5}		
21	7^{5}	13^{11}	21^{5}	35^{9}	42^{9}		
22	7^{5}	16^{21}	33^{9}	51^{9}	73^{21}	81^{21}	
23	8^{5}	20^{21}	46^{9}	81^{11}	117^{9}	135^{20}	
24	9^{10}	24^{5}	60^{9}	119^{20}	171^{20}	223^{20}	247^{20}
25	10^{10}	32^{9}	75^{9}	158^{20}	262^{20}	383^{7}	444^{7}
26	13^{5}	36^{11}	104^{9}	214^{20}	410^{9}	581^{11}	728^{11}
27	14^{10}	48^{5}	121^{9}	299^{20}	577^{9}	900^{11}	1289^{11}
28	16^{10}	56^{9}	168^{9}	376^{9}	821^{20}	1434^{11}	1981^{20}
29	20^{5}	66^{9}	203^{9}	541^{9}	1090^{9}	2055^{20}	3097^{20}
30	25^{5}	85^{9}	247^{9}	676^{9}	1623^{9}	2945^{20}	5137^{9}
31	31^{5}	109^{9}	329^{9}	850^{9}	2095^{9}	4328^{20}	7597^{20}
32	31^{11}	139^{9}	436^{9}	1169^{9}	2720^{9}	6094^{9}	11541^{9}
33	33^{5}	146^{9}	573^{9}	1598^{9}	3857^{9}	8160^{9}	16758^{9}
34	34^{5}	160^{9}	620^{9}	2164^{9}	5433^{9}	11758^{20}	23120^{9}
35	35^{5}	170^{9}	700^{9}	2411^{9}	6966^{14}	15358^{20}	31273^{20}
36	35^{11}	180^{9}	765^{9}	2800^{9}	8613^{14}	20026^{20}	41106^{20}
37	41^{9}	185^{9}	832^{9}	3145^{9}	10360^{9}	26058^{20}	53923^{20}
38	44^{9}	222^{9}	878^{9}	3512^{9}	11951^{9}	33825^{20}	70599^{20}
39	45^{9}	245^{9}	1082^{9}	3804^{9}	13696^{9}	41430^{14}	92246^{20}
40	46^{9}	257^{9}	1225^{9}	4808^{9}	15216^{9}	49803^{9}	120271^{20}

Continue Bounds on *A(n,10,w)*

n	*w*							
	13	14	15	16	17	18	19	20
26	824^{1}							
27	1460^{11}							
28	2438^{20}	2629^{20}						
29	4418^{9}	5050^{9}						
30	7009^{20}	8888^{20}	9988^{20}					
31	11574^{20}	15409^{20}	17294^{20}					
32	18608^{20}	24679^{20}	29770^{20}	30316^{20}				
33	28033^{20}	39786^{20}	53535^{20}	55685^{20}				
34	39452^{20}	65498^{14}	79867^{20}	93890^{20}	109301^{20}			
35	55392^{20}	97964^{20}	134693^{20}	165013^{20}	176685^{20}			
36	79031^{20}	141080^{20}	208896^{20}	256172^{20}	286906^{20}	288780^{20}		
37	110721^{14}	203513^{14}	323126^{20}	434743^{20}	487339^{20}	532074^{20}		
38	147770^{20}	275485^{14}	471136^{20}	661573^{20}	813629^{20}	889880^{20}	926414^{20}	
39	195111^{20}	396428^{14}	661166^{14}	1011085^{20}	1325690^{20}	1537543^{20}	1607405^{20}	
40	256946^{20}	540123^{20}	984261^{20}	1534849^{14}	2202622^{20}	2656882^{20}	2952378^{20}	3009020^{20}

A TABLE OF UPPER BOUNDS ON *A(n,12,w)*

n	w													
	7	8	9	10	11	12	13	14	15	16	17	18	19	20
14	2^5													
15	2^5													
16	2^5	2^{20}												
17	2^{10}	2^{20}												
18	3^5	3^5	4^5											
19	3^5	3^{10}	4^5											
20	3^{10}	5^5	5^5	6^5										
21	3^{14}	5^5	7^5	7^5										
22	4^5	6^5	8^5	11^5	12^5									
23	4^{10}	6^{10}	10^{10}	16^{10}	23^5									
24	4^{10}	9^5	16^5	24^5	24^5	46^5								
25	5^9	10^5	25^5	38^{20}	42^9	50^9								
26	5^{10}	13^5	26^5	37^7	69^{21}	83^{21}	92^{21}							
27	6^9	15^{10}	39^9	58^9	90^9	140^{20}	156^{20}							
28	8^5	19^{11}	45^{11}	87^7	147^9	199^{20}	245^{20}	265^{20}						
29	8^5	26^9	61^9	129^{20}	197^{20}	298^{20}	443^9	507^9						
30	9^5	30^5	80^{20}	159^{20}	268^{20}	492^9	679^{20}	903^{20}	1003^{20}					
31	9^{14}	31^5	97^{20}	197^{20}	380^{20}	692^9	1033^{20}	1424^{20}	1583^{20}					
32	10^{10}	36^9	110^9	245^{20}	573^9	981^{20}	1656^{20}	2143^{20}	2511^{20}	2652^{20}				
33	11^{10}	41^9	132^9	309^{20}	735^9	1412^{20}	2339^{20}	3235^{20}	4117^{20}	4700^{20}				
34	12^{10}	46^9	154^9	396^{20}	955^9	2082^9	3333^{20}	5119^{20}	7332^9	8748^9	9400^9			
35	15^5	52^9	178^9	519^{20}	1260^9	2777^{20}	4746^{20}	8332^9	11427^{20}	15064^{20}	17638^{20}			
36	16^5	66^9	208^9	640^9	1698^9	3613^{20}	7113^{20}	11943^{20}	18561^{20}	25016^{20}	28861^{20}	30260^{20}		
37	17^{10}	74^9	271^9	769^9	2152^9	4738^{20}	10283^9	17459^{20}	28981^{20}	38462^{20}	47144^{20}	52570^{20}		
38	19^{10}	80^9	312^9	1029^9	2656^9	6293^{20}	13849^9	27643^9	42310^{20}	60794^{20}	82592^{20}	99526^9	105140^9	
39	22^{14}	92^9	346^9	1216^9	3648^9	8540^{20}	18879^9	38579^9	63116^{20}	102979^{20}	137371^{20}	169699^{20}	203636^{20}	
40	25^{10}	110^9	408^9	1384^9	4421^9	12010^{20}	26276^9	52871^{20}	96738^{20}	154724^{20}	215914^{20}	292752^{20}	342835^{20}	355493^{20}

A TABLE OF UPPER BOUNDS ON *A(n,14,w)*

n	w												
	8	9	10	11	12	13	14	15	16	17	18	19	20
17	2^5												
18	2^5	2^5											
19	2^5	2^{10}											
20	2^{10}	2^{10}	2^{10}										
21	3^5	3^5	3^5										
22	3^5	3^{10}	4^5	4^5									
23	3^5	3^{10}	4^{10}	4^{10}									
24	3^{14}	4^{10}	5^{10}	6^5	6^{20}								
25	3^{14}	5^5	6^{10}	7^{10}	8^{10}								
26	4^5	6^5	8^5	10^5	13^5	14^5							
27	4^{10}	6^{10}	9^5	13^{10}	20^{10}	27^5							
28	4^{10}	7^5	11^{10}	21^5	28^5	28^5	54^5						
29	4^{14}	7^{10}	15^5	28^{11}	47^9	50^9	58^9						
30	5^9	10^5	21^5	30^5	70^9	88^9	107^9	116^9					
31	5^{10}	10^{10}	31^5	46^9	77^9	137^{20}	161^{20}	183^{20}					
32	5^{10}	12^{10}	31^{11}	70^9	122^9	183^{20}	233^{20}	277^{20}	295^{20}				
33	6^9	15^5	33^5	93^9	168^{20}	243^{20}	348^{20}	483^{20}	571^9				
34	6^{10}	17^{10}	46^9	102^9	210^{20}	334^{20}	590^9	787^{20}	1011^{20}	1142^9			
35	7^9	21^{10}	59^9	146^9	263^{20}	491^{20}	835^9	1226^{20}	1642^{20}	1792^{20}			
36	9^5	28^5	75^9	187^{20}	336^{20}	728^9	1201^{20}	1940^{20}	2386^{20}	2731^{20}	2848^{20}		
37	9^5	37^5	92^{20}	221^{20}	441^{20}	956^9	1777^{20}	2690^{20}	3456^{20}	4142^{20}	4562^{20}		
38	9^{10}	37^{11}	114^{20}	261^{20}	603^{20}	1267^{20}	2548^{20}	3686^{20}	5082^{20}	6651^{20}	8044^{20}	8633^{20}	
39	10^5	39^5	139^{20}	309^{20}	848^9	1702^{20}	3333^{20}	5144^{20}	7881^{20}	11658^9	14405^{20}	16511^9	
40	10^{14}	40^5	156^9	368^{20}	1030^9	2459^{20}	4383^{20}	7407^{20}	12860^9	17919^{20}	23902^{20}	29164^{20}	30438^{20}

A TABLE OF UPPER BOUNDS ON *A(n,16,w)*

n	*w*											
	9	10	11	12	13	14	15	16	17	18	19	20
18	2^{5}											
19	2^{5}											
20	2^{5}											
21	2^{5}	2^{5}										
22	2^{5}	2^{10}	2^{20}									
23	2^{10}	2^{10}	2^{10}									
24	3^{5}	3^{5}	3^{5}	4^{5}								
25	3^{5}	3^{10}	3^{10}	4^{5}								
26	3^{5}	3^{10}	4^{5}	4^{10}	4^{10}							
27	3^{14}	3^{10}	5^{5}	5^{10}	6^{5}							
28	3^{14}	4^{9}	5^{10}	7^{5}	7^{5}	8^{5}						
29	3^{14}	4^{10}	6^{5}	7^{10}	9^{5}	10^{5}						
30	4^{5}	6^{5}	6^{10}	10^{5}	12^{5}	15^{5}	16^{5}					
31	4^{5}	6^{5}	8^{5}	11^{10}	16^{11}	24^{5}	31^{5}					
32	4^{10}	6^{10}	9^{10}	16^{5}	26^{9}	32^{5}	32^{5}	62^{5}				
33	4^{10}	7^{5}	12^{5}	22^{5}	33^{5}	55^{9}	58^{9}	66^{9}				
34	4^{14}	7^{10}	13^{10}	34^{5}	53^{9}	80^{9}	103^{9}	123^{9}	132^{9}			
35	5^{9}	8^{10}	16^{10}	35^{5}	81^{20}	124^{20}	157^{20}	184^{20}	200^{20}			
36	5^{10}	9^{10}	21^{10}	48^{9}	96^{9}	160^{20}	212^{20}	265^{20}	307^{20}	324^{20}		
37	5^{10}	11^{5}	28^{10}	64^{9}	136^{9}	207^{20}	296^{20}	407^{20}	524^{20}	606^{20}		
38	5^{10}	12^{5}	37^{11}	87^{20}	172^{20}	268^{20}	414^{20}	703^{9}	909^{9}	1106^{9}	1212^{9}	
39	6^{9}	12^{11}	39^{5}	113^{20}	214^{20}	344^{20}	636^{20}	995^{20}	1429^{20}	1867^{20}	2016^{20}	
40	6^{10}	16^{5}	40^{5}	130^{9}	257^{20}	456^{20}	917^{9}	1517^{20}	2254^{20}	2738^{20}	3075^{20}	3197^{20}

A TABLE OF UPPER BOUNDS ON *A(n,18,w)*

n	*w*										
	10	11	12	13	14	15	16	17	18	19	20
21	2^{5}										
22	2^{5}	2^{5}									
23	2^{5}	2^{5}									
24	2^{5}	2^{5}	2^{20}								
25	2^{10}	2^{10}	2^{20}								
26	2^{10}	2^{10}	2^{10}	2^{10}							
27	3^{5}	3^{5}	3^{5}	3^{5}							
28	3^{5}	3^{5}	3^{10}	4^{5}	4^{5}						
29	3^{5}	3^{10}	3^{10}	4^{5}	4^{10}						
30	3^{5}	3^{10}	5^{5}	5^{5}	5^{5}	6^{5}					
31	3^{10}	3^{10}	5^{5}	5^{10}	6^{5}	6^{10}					
32	3^{10}	4^{9}	5^{10}	6^{10}	7^{10}	8^{5}	8^{10}				
33	3^{10}	4^{10}	6^{5}	7^{10}	9^{5}	11^{5}	11^{5}				
34	4^{5}	4^{10}	6^{10}	8^{10}	10^{10}	14^{5}	17^{5}	18^{5}			
35	4^{5}	5^{9}	6^{11}	10^{5}	15^{5}	21^{5}	28^{5}	35^{5}			
36	4^{10}	6^{5}	9^{5}	12^{5}	19^{10}	36^{5}	36^{5}	36^{5}	70^{5}		
37	4^{10}	6^{10}	9^{10}	15^{5}	29^{10}	37^{5}	63^{9}	66^{9}	74^{9}		
38	4^{14}	7^{5}	10^{10}	19^{10}	38^{5}	61^{9}	87^{9}	119^{9}	139^{9}	148^{9}	
39	4^{14}	7^{10}	13^{5}	27^{5}	39^{5}	98^{9}	145^{20}	177^{20}	203^{20}	217^{20}	
40	5^{9}	8^{5}	13^{10}	40^{5}	60^{9}	104^{9}	186^{20}	235^{20}	282^{20}	316^{20}	329^{20}

A TABLE OF UPPER BOUNDS ON *A(n,20,w)*

n	*w*									
	11	12	13	14	15	16	17	18	19	20
22	2^5									
23	2^5									
24	2^5	2^5								
25	2^5	2^5								
26	2^5	2^5	2^5							
27	2^5	2^{10}	2^{10}							
28	2^{10}	2^{10}	2^{10}	2^{10}						
29	2^{10}	2^{10}	2^{10}	2^{10}						
30	3^5	3^5	3^5	3^5	4^5					
31	3^5	3^5	3^{10}	3^{10}	4^5					
32	3^5	3^{10}	3^{10}	4^5	4^5	4^{10}				
33	3^5	3^{10}	3^{10}	4^{10}	4^{10}	4^{10}				
34	3^5	3^{10}	4^9	5^5	5^{10}	6^5	6^5			
35	3^{10}	3^{10}	5^5	5^{10}	7^5	7^5	7^5			
36	3^{10}	4^9	5^5	6^5	7^{10}	9^5	9^5	10^5		
37	3^{10}	4^{10}	6^5	6^{10}	8^{10}	10^5	11^{10}	12^{10}		
38	4^5	4^{10}	6^5	8^5	10^5	12^{10}	16^5	19^5	20^5	
39	4^5	4^{10}	6^{10}	9^5	13^5	17^5	23^{10}	31^{10}	39^5	
40	4^5	5^9	7^{10}	10^{10}	16^5	25^5	39^{11}	40^5	40^5	78^5

9.REFERENCES

[1] E.Agrell, A.Vardy and K.Zeger, "Upper bounds for constant weight codes,"*IEEE Trans.Inform.Theory.*, vol.46, pp.2373-2395, Nov.2000.

[2] E.Agrell, A.Vardy, KZeger, "Constant weight Code Bounds from Spherical Code Bounds," ISIT2000, Sorrento, Italy 25-30, 2000.

[3] E.Agrell, A.Vardy and K.Zeger, "A Table of upper bounds for binary codes," *IEEE Trans. Inform.Theory*, Vol 47, no.7, Nov. 2001.

[4] E.Agrell,A.vardy, and K.Zeger. Tables of binary block codes. Online available www.s2.chalmers.se/~agrell

[5] L.A.Bassalygo, "New upper Bounds for Error correcting codes, Probl.Inform.Trans" vol.1, no.4, pp.32-35, 1968. (Original appearance in Russian in *Probl.Pered.Inform.* vol.1, pp.41-44, Oct.-Dec.1965).

[6] A. Bateman, "Digital communications," Design for the Real World, first edition 1998.

[7] M.R.Best,A.E.Brouwer,F.J.MacWilliams, A.M.Odlyzko, and N.J.A.Sloane, "Bounds for binary codes of length less than 25," *IEEE Trans.Inform.Theory*, vol.IT-24, pp.81-93, Jan.1978.

[8] R.E Blahut, Theory and Practice of Error Control Codes. Reading, MA: Addison-Wesley, 1983.

[9] C. A. Bouman: Digital Color Imaging - April 29, 2005,School of Electrical Engineering, Purdue University,West Lafayette, IN 47907,USA.
http://dynamo.ecn.purdue.edu/~bouman/ee637/notes/pdf/SourceCoding.pdf

[10]A.E.Brouwer,Dept.ofMath.,Techn.University.Eindhoven,Netherlands,aeb@cwi.nl,vo1.3, March23,2004.http//www.win.tue.nl/~aeb/codes/binary-1.html.

[11] A.E.Brouwer,"A Few New Constant Weight Codes,"*IEEE Trans.Inform.Theory.*, vol.26, Pp.366-366, May 1980.

[12] A. E.Brower, J.B.Shearer, N.J.A.Sloane and W.D.Smith."A New table of constant weight codes," IEEE Tran.Inform.Theory, vol.36, pp.1334-1380, Nov.1990.

[13] M.Buratti "Recursive constructions for difference matrices and relative difference families, J.Combin.Des" vol.6, pp.165-182, 1998.

[14]M.J.Colbourn, "Analytic and Computer Techniques for set packings,"M.Sc.Thesis, Department of Computer science, University of Toronto, Toronto, ON, Canada, 1977.

[15] M.J.Colbourn, "Some Upper Bounds for Constant Weight Codes," *IEEE Trans. Inform.Theory.*, vol.IT-26, No.4, July1980.

[16] J.H.Conway and N.J.A.Sloane, Sphere packings, Lattices and Groups. New York, NY: Springer, 3rd ed., 1999.

[17]C.V.Frieman, "Upper bounds for fixed weight codes of specified minimum distance," *IRE Trans.Inform.Theory.*, vol.IT, pp.246-248, July 1964.

[18]J.D.Gibson, The Communications Handbook, Second Edition, CRC Press, NY, USA 1996**.**

[19]R.L.Graham and N.J.A.Sloane, "Lower Bounds for Constant Weight Codes," *IEEE Trans.Inform.Theory*, vol.IT-26, pp.37-43, Jan.1980.

[20]R.W.Hamming, "Error detecting and Error correcting codes," Bell sys.Tech.J, vol.29, pp.147-160; April, 1950.

[21]Honkala, "Bounds for Binary constant weight and coverage codes," Licentiate Thesis, Dept.Math., Univ.Turku, Finland, Mar.1987.

[22]S.M.Johnson, "A New Upper Bounds for Error correcting codes," *IRE Trans.Inform Theory.,* vol.IT-8, pp.203-207, April 1962.
[23]S.M.Johnson, "Improved asymptotic bounds for error–correcting codes," *IEEE Trans. Inform. Theory.,* vol.IT-9, pp.198-205, July 1963.
[24] S.MJohnson "A new upper bound for error-correcting codes," *IRE Trans.Inform. Theory.*, vol.IT-17, pp.446-478, July1971.
[25] S.M.Johnson, "On Upper bounds for unrestricted binary error- correcting codes," *IEEE Trans.Inform. Theory.*, vol.IT-17, pp.466-478, July 1971.
[26] S.M.Johnson, "Upper Bounds for Constant Weight error correcting codes," Discrete Math., vol.3, pp.109-124, 1972.
[27] R.E.Kibler, "Some New Constant Weight Codes," *IEEE Trans.Inform.Theory.,* vol.26, pp.364-365, May 1980.
[28] Lubos Thoma**,** Department of Mathematics**,** *9 GreenhouseRoad, Suite3Kingston, RhodeIsland 02881-0816,USA* .http://www.math.uri.edu/~thoma/teaching/mth391_fall2004/hams.htm.
[29] F.J.MacWilliams and N.J.A.Sloane, "*The Theory of Error-Correcting Codes*," Amsterdam, The Netherlands: North- Holland, 1977.
[30] A. Makhorin, Jan 2001, Free Software Foundation, 51 Franklin St, Fifth Floor, Boston MA 02110-1301 USA http://www.gnu.org/software/glpk/glpk.htm
[31] A. Makhorin, Jan 2001 Free Software Foundation, 51 Franklin St, Fifth Floor, Boston MA 02110-1301 USA http://sourceforge.net/project/shownotes.php?release_id=228974.
[32] B.Mounits, T.Etzion and S. Litsyn, *IEEE Trans. Inform. Theory.*, vol.48.No.4, April 2002.
[33] P.R.J.Östergård, T.Baicheva, and E.Kolev, "Optimal binary one-error correcting codes of length 10 have 72 code words,"IEEE Trans.Inform.Theory, vol.45, pp.1229-1231, May 1999.
[34] M.Plotkin, "Binary codes with specified minimum distance," *IRE Trans .Inform. Thoery.*, vol.IT-6, pp.445-450, Sep.1960.
[35]G.Proakis and M. Salehi "Communication Systems Engineering," Second Edition, 2002.
[36] I.S.Reed and X. Chen, "Error Control Coding for Data networks," First Edition,1999.
[37] K. Sayood, "Introduction to Data Compression," Second Edition.2000
[38]A. Schrijver, "New code upper bounds from the Terwilliger algebra," Apr. 2004. http://homepages.cwi.nl/~lex/files/codes.pdf
[39] C.L.M.Van Pul, "On bounds on codes," Masters Thesis, Dept. Math.Comput. Sci., Eindhoven Univ.Technol., Eindhoven,The Netherlands, Aug.1982.
[40] N.Wax."On Upper Bounds for Error Detecting and Error correcting codes of finite length," *IEEE Trans Inform.Theory.*, vol.IT-5, pp, 168-174; December, 1959.
[41]Doig Scott Building,Craibstone Estate, Bucksburn, Aberdeen, AB21 9YA, Scotland.
http://www.4i2i.com/reed_solomon_codes.htm
[42] Lucent Technologies, 600 Mountain Ave., MurrayHill, NJ, USA.
http://www.lucent.com/minds/infotheory/docs/history.pdf