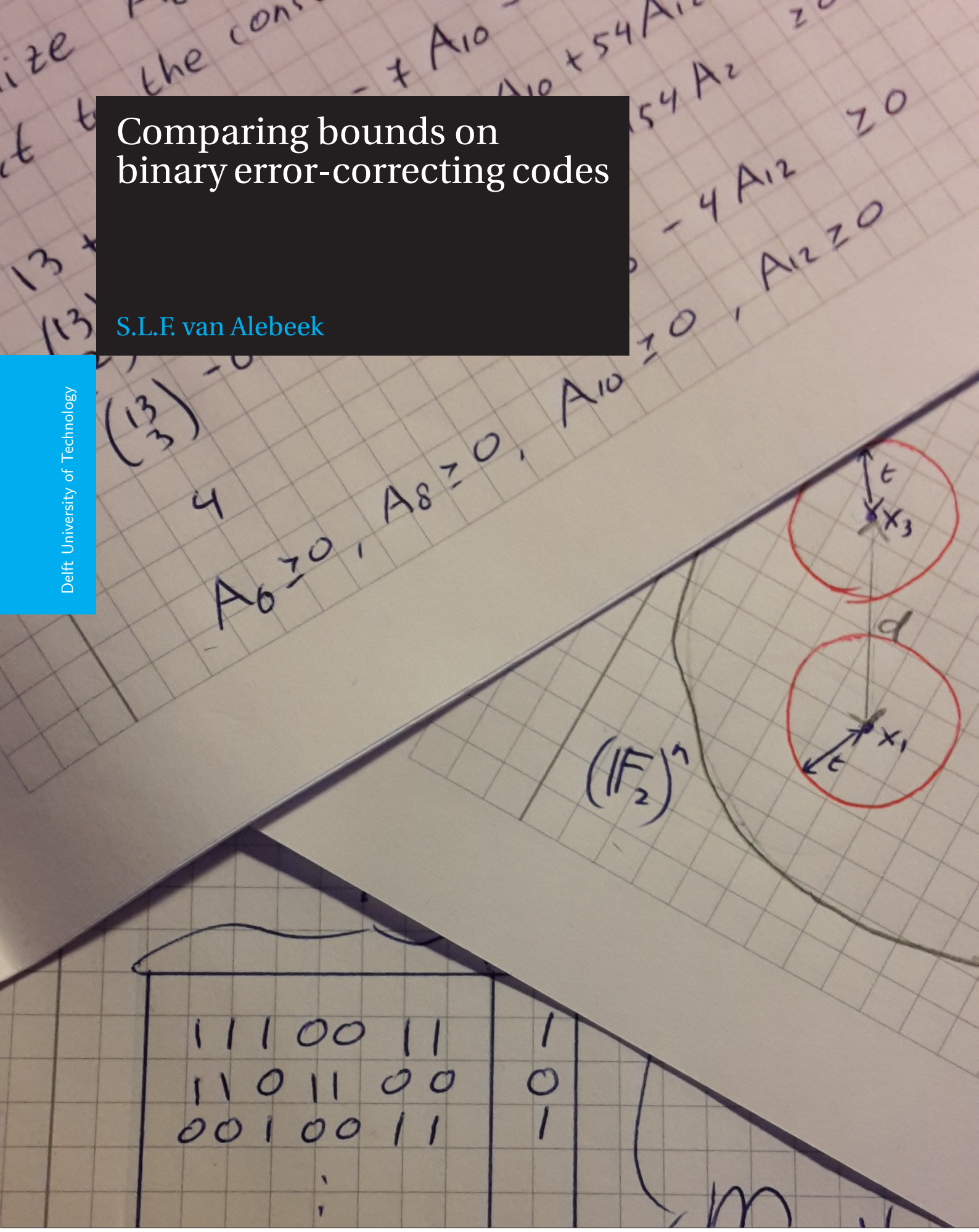
Comparing bounds on binary error-correcting codes

S.L.F. van Alebeek

Delft University of Technology



TU Delft Delft University of Technology

Challenge the future

Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft Institute of Applied Mathematics

Comparing bounds on binary error-correcting codes

(Dutch title: Vergelijken van grenzen voor binaire foutverbeterende codes)

A thesis submitted to the
Delft Institute of Applied Mathematics
in partial fulfillment of the requirements

for the degree

BACHELOR OF SCIENCE
in
APPLIED MATHEMATICS

by

Sanne van Alebeek

Delft, Nederland
Maart, 2017

BSc THESIS APPLIED MATHEMATICS

"Comparing bounds on binary error-correcting codes"

(Dutch title: "Vergelijken van grenzen voor binaire foutverbeterende codes")

Sanne van Alebeek

Delft University of Technology

Supervisor

Dr.ir. J.H. Weber

Thesis committee

Dr. D. C. Gijswijt

Drs. E.M. van Elderen

Dr. D. Kurowicka

March, 2017 Delft

PREFACE

Delft, March 2017

My first encounter with error-correcting codes was already during high school, when I was working on a school project ("profielwerkstuk"). In particular I was intrigued by the special characteristics of a Hamming code. During my studies I also followed the course "Toegepaste algebra, codes en cryptosystemen", where we dove deeper into this subject. One of the lecturers, Joost de Groot, made the suggestion of bounds on error-correcting codes as a subject for a bachelor thesis, where as a result this report now lies in front of you.

I would like to extend my gratitude to my thesis supervisor, Jos Weber, for his patience, guidance and support, and his critical questions and remarks. Also I would like to thank my thesis committee for taking the time to read and critically review my thesis. Next, I would like to extend my gratitude to my loving and caring family for taking the time to read through my thesis and give their welcome suggestions. Finally, I would like to thank Hugo for sharing his expertise with Matlab and most of all for his love and moral support.

SUMMARY

This report deals with binary error-correcting codes. A binary code is a collection words with the same length n that consists only of zeroes and ones. The error-correcting quality of a code is determined by the Hamming distance d of a code, which is the minimum distance between any two codewords in a code. The distance $d(\mathbf{x},\mathbf{y})$ between two codewords $\mathbf{x}$ and $\mathbf{y}$ is defined as the number of positions in which they differ. A classical question in coding theory is: What is the maximum number of codewords in a code with length n and Hamming distance d? This maximum is denoted with $A(n,d)$. Determining $A(n,d)$ is quite difficult and often we have to make due with lower and upper bounds on $A(n,d)$.

In this report we compare six different methods for finding an upper bound on $A(n,d)$. We start by refreshing some basics for error-correcting binary codes, in particular we explain which approach is normally used to find $A(n,d)$. Next, we introduce these six different types of bounds and for each of these we give a proof and a clarifying example. Two of the bounds, namely the linear programming bound and the linear programming bound with extra constraints, we investigate in more detail. The idea behind these bounds is to use the distance distribution $(A_0, A_1, A_2, \ldots, A_n)$ of a code to find an upper bound, where A_i is the average number of codewords at distance i from a codeword. The A_i comply to two sorts of linear conditions. The first is that $A_i \geq 0$, and if there are no codewords at distance i then $A_i = 0$. The second sort are inequalities that originate from Krawtchouck polynomials. Together these conditions form a linear program where upper bounds on $A(n,d)$ can be found with. The difference between these two linear programming bounds is the addition of extra constraints. These restrict A_i to be smaller than or equal to the number of codewords with length n, Hamming distance d and weight i, where the weight of a binary word is the number of ones in that word.

Furthermore, we compare the six types of bounds with each other. The main conclusion from this comparison is that a method provides a better upper bound when it requires more input and involves more complex calculations. This input can for example consist of Krawthcouck polynomials or upper bounds on the number of codewords with length n, Hamming distance d and weight w. The linear programming bound with extra constraints is the best bound for $n = 6,7,\ldots,28$ and $d = 4,6,\ldots,12$, except for $A(24,4)$ and $A(12,4)$, in those cases the Johnson bound gives the best results. At a first glance this does not show in the results for the upper bound for $n = 12$ and $d = 4$, this is caused by the integer rounding of the result of the linear programming bound with extra constraints. Due to this integer rounding the presented results of the upper bound of these two methods is in this case the same. For the case of $n = 24$ and $d = 4$ the linear programming bound gives an upper bound of 344.636 and the Johnson bound of 344.308. This is caused by the fact that, compared to other n, the linear programming bound gives a relatively high value of the upper bound and the Johnson bound gives a relatively low value, a more detailed explanation is given in Chapter 4.

CONTENTS

1

INTRODUCTION AND FUNDAMENTALS

This report discusses binary error-correcting codes, and in particular compares different methods of finding upper bounds on the number of codewords in these codes. The formal introduction to this report can be found in Section 1.1. Subsequently, some fundamentals of error-correcting codes can be found in Section 1.2. We conclude this chapter with some fundamentals about finding bounds on binary error-correcting codes in Section 1.3.

1.1. INTRODUCTION

A binary error-correcting code is a collection words with the same length n that consists only of zeroes and ones. The error-correcting part means that errors induced in the codewords can be corrected by the receiver. By adding checkbits to the end of a codeword one introduces the possibility of error-correction, the fundamentals hereof are explained in Section 1.2. By adding checkbits the length of a codeword increases, meaning that the amount of data to be transmitted increases. On the other hand, by adding checkbits one creates the possibility to increase the distance (the amount of bits two codewords differ) between two codewords in a code. The quality of a code is determined by the Hamming distance d, which is the minimum distance between any two codewords in a code. In practice n and d are often already set, which gives rise to the question: What is the maximum number of codewords in a code with the given n and d? This maximum is denoted by $A(n,d)$. Determining $A(n,d)$ proves to be quite difficult and often we have to revert to lower and upper bounds on $A(n,d)$. This report focuses on comparing methods for finding upper bounds on $A(n,d)$. In total we compare six different methods and try to establish which method gives the lowest (best) upper bound. Any peculiarities stemming from this comparison are explained in more detail.

In Chapter 2 we introduce the different types of methods for finding upper bounds, for each we give a proof and a clarifying example. Next, in Chapter 3 we explain the linear programming bound with and without extra constraints in more detail. Hereafter, we present the comparisons between the different methods in Chapter 4. Finally, we give conclusions and recommendations in Chapter 5.

1.2. FUNDAMENTALS OF ERROR-CORRECTING CODES

Coding theory is studied for the purpose of designing efficient and reliable data transmission and storage methods. Data transmission is the transfer of data over a point-to-point channel. Before we send the data over this channel, we convert it into messages consisting only of zeroes and ones: binary words. These binary words are from the set of the form: $(\mathbb{F}_2)^k = \{a_1 a_2 \dots a_k | a_i \in \mathbb{F}_2, i = 1,2,\dots k\}$. Where $\mathbb{F}_2$ is the finite field with 2 elements: "0" and "1".

Example 1 (Route system)**.** A simple route can be indicated by a row of words from the set M ={north, east, south, west} These are then converted to binary words from $(\mathbb{F}_2)^2$, for example:

$$\text{north} \rightarrow 00$$
$$\text{east} \rightarrow 01$$
$$\text{south} \rightarrow 10$$
$$\text{west} \rightarrow 11$$

When sending a binary message from source to destination, there is always a chance of bits getting lost or changed. To avoid miscommunications, the receiver of the message should (in most cases) be able to find the original message sent from the source, which is only possible if the errors in the received message can be found and corrected. Codes that allow for this type of corrections are called error-correcting codes. A minimum condition for error-detection or error-correction is that the message, when sent through the channel, does not change into a different message, as explained in Example 2. Errors in binary codewords are always bit changing, i.e. from zero to one or vice versa. The bit error probability p is the chance of one bit changing. In a symmetric system the chance of a bit changing from one to zero is equal to the chance of a bit changing from zero to one. In this report only binary symmetric systems are discussed with $0 < p < \frac{1}{2}$, meaning that the chance of a bit changing is always smaller than the chance of a bit remaining it's original value.

Example 2. Suppose someone is using the route system of Example 1. His system tries to send him to the north, so the system sends the message 00.Due to bad reception, the message is transferred incorrect and the second bit changes to 1. The person then receives 01 and travels to the east instead of the north. With only one small error occurring, the person is sent in the wrong direction.

In 1950, Richard Hamming introduced a new coding strategy called "channel coding" and designed the first error-correcting code, namely the "Hamming(7,4)"-code [1]. The words used in channel coding are called codewords and the set of all the codewords is called the code. Channel coding consists of three major phases: encoding, sending/receiving and decoding. Encoding a binary message to a codeword is done by adding bits, so-called checkbits, to the end of the original binary message. The sending of a codeword proceeds through a channel to the receiver, who then decodes the codeword and estimates the original message. An example of channel coding is given in Example 3. When using channel coding it is important that both source and destination are familiar with the code and know all of the codewords.

Example 3 (Channel coding)**.** The messages from Example 1 are encoded. The four directions become four codewords and form code C_1. Remark: the receiver also knows which direction is encoded in which codeword but does not know which codeword was sent.

north: 00 → codeword: 00 000

east: 01 → codeword: 01 011

south: 10 → codeword: 10 101

west: 11 → codeword: 11 110

The route system (source) wants to send the receiver (destination) in the direction east, so it encodes the message 01 to the codeword 01011, which is then sent through the channel, see Figure 1.1. An error occurs, and the destination receives 00 011. When searching in code C_1 it becomes clear to the receiver that 00 011 is not a codeword that belongs to C_1, which means that an error has occurred. Because the probability of a bit changing is less than 50%, it is most likely that the lowest possible number of errors has occurred. Now the receiver has to determine all possible bit changes that could have caused a codeword to change into 00 011. To determine which of the codewords originally was sent, one needs to find the codeword which differs from 00 011 in the least number of places:

codeword: 00 000 → 00 011: 2 bits changed

codeword: 01 011 → 00 011: 1 bit changed

codeword: 10 101 → 00 011: 3 bits changed

codeword: 11 110 → 00 011: 4 bits changed

So in this case it is most likely that the second bit (red in Figure 1.1) in the sent codeword 01 011 has changed. The receiver "fixes" the error and estimates the codeword: 01 011. So he was able to detect and correct the error that occurred during the transfer of the codeword.

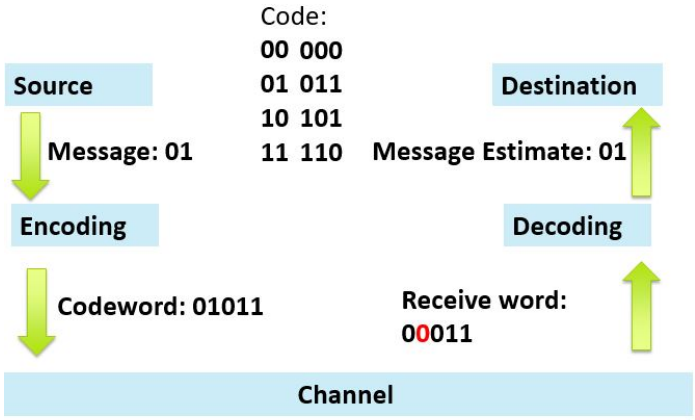


Figure 1.1: Example of error-correction[2]

Often all of the codewords are of the same length n, this is called a block code. In this report we focus on a specific type of code: Binary error-correcting block codes . In this specific type of code, codewords are all of the same length n and consist only of zeroes and ones. In mathematical terms: The codewords in a code C are from a subset of a vector space: $(\mathbb{F}_2)^n = \{a_1 a_2 \dots a_n | a_i \in \mathbb{F}_2, i = 1,2,\dots n\}$.

As seen in Example 3, decoding is actually searching for the codeword that differs from the received word in the least amount of bits. This is also called nearest neighbor decoding. Hence the next definitions:

Definition 1. *Let C_1 be a binary block code. The distance $d(\mathbf{x},\mathbf{y})$ between two codewords $\mathbf{x},\mathbf{y} \in C_1$ is the number of places these two words $\mathbf{x}$ and $\mathbf{y}$ differ.*
In mathematical terms: $d(\mathbf{x},\mathbf{y}) = |\{i | \mathbf{x}_i \neq \mathbf{y}_i\}|$. When $\mathbf{x} = \mathbf{y}$ then $d(\mathbf{x},\mathbf{y}) = 0$.

Definition 2. *The Hamming distance of a code C, denoted with d, is the minimal distance between any $\mathbf{x}$ and $\mathbf{y} \in C$. In mathematical terms: $d = min\{d(\mathbf{x},\mathbf{y}) | \mathbf{x},\mathbf{y} \in C \text{ and } \mathbf{x} \neq \mathbf{y}\}$.*

When receiving a binary word that is indeed a codeword, it is assumed that no errors have occurred. If the received word does not equal a codeword, then certainly something went wrong: an error is detected. A code is called s-error detecting if it detects one up to and including s errors. Sometimes, when a retransmission is possible, error detection is enough. However, in most cases nearest neighbor decoding is used to find the original codeword that was sent from the source. A code is t-error correcting if it corrects one up to and including t errors. The quality of a code is determined by the number of errors it is able to detect or correct.

Theorem 1. *A code C with Hamming distance d can detect up to $d-1$ errors or can correct up to $\frac{d-1}{2}$ errors.[6]*

Definition 3. *A code consisting of M codewords with lenght n and Hamming distance d is called a (n, M, d) code*

Example 4. Code C_1 from Example 3 has $M = 4$ codewords with length $n = 5$ and Hamming distance $d = 3$. C_1 is a (5,4,3) code

When designing codes it is desirable to enable the code to detect and correct as many errors as possible. The number of errors that can be detected depends on the Hamming distance d of a code, a larger d allows for more errors to be detected or corrected, see Theorem 1. A larger Hamming distance is easily achieved by adding more bits at the end of the binary words, this also increases the length n of the code. A larger n means more information needs to be sent through the channel, which is less efficient. This efficiency of a binary code is expressed with the code-rate $R = \frac{^2\log M}{n}$. The code-rate can also be calculated by dividing the length of the message before encoding, by the length of the codeword after encoding. Therefore, the code-rate is a measure to see the ratio between information carrying bits and safety bits in a codeword. For an efficient code the code-rate R should be as close to 1 as possible. The challenge of designing a code lies in finding the balance between the ability to detect and correct errors, expressed by a large hamming distance, and the efficiency of a code, expressed by the code-rate.

Example 5. Code C_1 from Example 3 has code-rate $R = \frac{2}{5}$

1.3. BOUNDS ON THE SIZE OF BINARY CODES

In practice n and d are often fixed, so then the question becomes what the maximum number of codewords is in a code with the respective n and d. The maximum number of codewords in any binary code of length n and Hamming distance d is denoted by $A(n,d)$. In the next theorem we give a few characteristics of $A(n,d)$.

Theorem 2 (Characteristics of $A(n,d)$)**.** *For any binary block code C with codewords of length n and Hamming distance d the following (in)equalities apply:*

1. *If $d > n$ then $A(n,d) = 1$ [3]*
2. *If $d = 1$ then $A(n,d) = 2^n$ [4]*
3. *$A(n-1,2e-1) = A(n,2e)$ [5]*
4. *$A(n,d) \leq 2 * A(n-1,d)$ [4]*

Proof.

1. Two codewords can never differ in more bits than the number of bits they consist of. [3]
2. We count all the vectors in $(\mathbb{F}_2)^n$, this is 2^n [4].
3. We start with a code with Hamming distance $2e-1$ and length $n-1$ with M codewords. We build a new code C_1 by adding a bit to the end of every codeword from C. If the codeword has an even number of ones, we add a zero. If the codeword has an odd number of ones, we add a one. By doing this we create a code with M codewords with length n and an even number of ones. Because the codewords all have an even number of ones the distance between the codewords also has to be even. This then gives us $d = 2e$. We created a $(n,M,2e)$ code. So $A(n-1,2e-1) \leq A(n,2e)$. Conversely, given a $(n,M,2e)$ code, deleting one bit gives a $(n-1,M,2e-1)$ code. So $A(n-1,2e-1) \geq A(n,2e)$.[5]
4. Let C be a $A(n,d)$ code. Divide the code into two sets, codewords beginning with a "1" and codewords beginning with a "0". One set must contain at least half or more of the codewords, we call this set C_1. Now deleting the first bit of every codeword in C_1 leaves a code C_2 with minimum distance d_2 and length $n-1$, containing at least $\frac{A(n,d)}{2}$ codewords. If $d_2 = d$ then this gives us the inequality $A(n,d) \leq 2 * A(n-1,d)$. However if $d_2 \geq d$ we are not finished yet. Search in C_2 for the two closest codewords $\mathbf{x}$ and $\mathbf{y}$. Now find a bit in which $\mathbf{x}$ and $\mathbf{y}$ differ and replace this bit in every codeword in C_2 with zero. By doing this we make d_2 one smaller. Repeat this technique untill $d_2 = d$, then the first case applies.[4, 6]

□

A(n,d) is found by finding a lower bound and upper bound for $A(n,d)$. When the lower and upper bound are equal to each other, $A(n,d)$ is found. In mathematical terms, set $L \leq A(n,d) \leq U$, when $L = U$, then $A(n,d) = L = U$. However, in most cases only a lower and upper bound that are not equal to each other are known, so we only know $L \leq A(n,d) \leq U$. In Table 1.1 the best known bounds for $n = 6,7,\ldots 28$ and $d = 4,6,\ldots 12$ are shown. All cases with a known A(n,d) are shown in blue in Table 1.1. When only a lower and upper bound are known, two values are shown, the first value is the lower bound and the second the upper bound. Table 1.1 is restricted to even d because of part 3 of Theorem 2.

A lower bound is usually found by designing or finding a code that corresponds with a certain d and n. An example of finding a lower bound for $A(6,4)$ is given below. Remark: by part 3 of Theorem 2 it holds that $A(6,4) = A(5,3)$. So in this example a lower bound for A(n,d) is found for the same Hamming distance and length as Example 3.

Example 6. We design a code C_2 to obtain a lower bound for $A(6,4)$ We know: $n = 6$ and $d = 4$. This means that all the codewords of C_2 must have length 6 and $d(\mathbf{x},\mathbf{y}) \geq 4, \forall \mathbf{x},\mathbf{y} \in C_2$. We start by choosing the first codeword $\mathbf{x}_1 = 000\ 000$. Then we find a codeword $\mathbf{x}_2$ that has 4 ones to make sure it's distance from $\mathbf{x}_1$ is at least 4. We choose: 001 111. Now we want to find $\mathbf{x}_3$ that has minimum distance 4 from $\mathbf{x}_1$ and $\mathbf{x}_2$. We choose: 110 011. And for $\mathbf{x}_4$ we choose 111 100. Notice that the minimum distance between all $\mathbf{x}_i$ is 4. So we found a code C_2 that consist of 4 codewords of length 6 and with a minimum distance of 4. This then gives the lower bound for $A(6,4) \geq 4$.

n	d=4	d=6	d=8	d=10	d=12
6	4^{a_1}	2^{a_1}	1^{a_1}	1^{a_1}	1^{a_1}
7	8^{a_1}	2^{a_1}	1^{a_1}	1^{a_1}	1^{a_1}
8	16^{a_1}	2^{a_1}	2^{a_1}	1^{a_1}	1^{a_1}
9	20^{b_1}	4^{a_1}	2^{a_1}	1^{a_1}	1^{a_1}
10	40^{a_1}	6^{a_1}	2^{a_1}	2^{a_1}	1^{a_1}
11	72^{d}	12^{a_1}	2^{a_1}	2^{a_1}	1^{a_1}
12	144^{d}	24^{a_1}	4^{a_1}	2^{a_1}	2^{a_1}
13	256^{a_3}	32^{b_2}	4^{a_1}	2^{a_1}	2^{a_1}
14	512^{a_3}	64^{a_3}	8^{a_1}	2^{a_1}	2^{a_1}
15	1024^{a_2}	128^{a_3}	16^{a_1}	4^{a_1}	2^{a_1}
16	2048^{a_2}	256^{a_2}	32^{a_1}	4^{a_1}	2^{a_1}
17	$2816-3276^{a_3}$	$256-340^{b_3}$	36^{O}	6^{a_1}	2^{a_1}
18	$5632-6552^{a_1}$	$512-673^{G}$	$64-71^{H}$	10^{a_1}	4^{a_1}
19	$10496-13104^{a_1}$	$1024-1237^{G}$	$128-131^{H}$	20^{a_1}	4^{a_1}
20	$20480-26168^{a_1}$	$2048-2279^{G}$	256^{G}	40^{a_1}	6^{a_1}
21	$40960-43688^{M}$	$2560-4096^{b_3}$	512^{b_3}	$42-47^{G}$	8^{a_1}
22	$81920-87333^{M}$	$4096-6941^{b_1}$	1024^{a_3}	$64-84^{G}$	12^{a_1}
23	$163840-172361^{M}$	$8192-13674^{G}$	2048^{a_3}	$80-150^{b_1}$	24^{a_1}
24	$327680-344308^{a_2}$	$16384-24106^{b_1}$	4096^{a_2}	$136-268^{G}$	48^{a_1}
25	$2^{19}-599184^{M}$	$17920-47538^{S}$	$4096-5421^{G}$	$192-466^{G}$	$52-55^{G}$
26	$2^{20}-1198368^{M}$	$32768-84260^{M}$	$4104-9275^{G}$	$384-836^{G}$	$64-96^{G}$
27	$2^{21}-2396736^{M}$	$65536-157285^{M}$	$8192-17099^{G}$	$512-1585^{G}$	$128-169^{b_1}$
28	$2^{22}-4792950^{M}$	$131072-291269^{b_1}$	$16384-32151^{S}$	$1024-2817^{G}$	$178-288^{a_3}$

Table 1.1: Table with the best known bounds for $n = 6, 7, \ldots 28$ and $d = 4, 6, \ldots 12$ [3, 7]. Legend of superscripts:

- a_1: Found with Thm 1 due to M. Plotkin in [8]
- a_2: Found with Thm 2 due to S. M. Johnson in [8]
- a_3: Found with Thm 3 due to P. Delsarte in [8]
- b_1: Found with Thm 5 and Thm 8 in [9]
- b_2: Found in Thm 4 in [9]
- b_3: Found with linear programming in [9]
- d: Found in [10]
- M: Found in [11]
- S: Found in [12]
- O: Found in [13]
- G: Found in [14]
- H: Found in [15]

A linear code is a code for which any two codewords added together also form a codeword. Linear error-correcting codes allow for more efficient encoding and decoding algorithms than non-linear codes. A binary linear code of length n and rank k, the length of the original messages before encoding, is a linear subspace C wtih dimension k of the vector space $(\mathbb{F}_2)^n$. For a binary linear code it follows that the number of codewords equals 2^k [6]. The code from Example 3 is a linear code with $n = 5$ and $k = 2$. Building a linear code with a certain n and d is still very difficult but can be done with more structure than a non-linear code. For this reason a lower bound on $A(n,d)$ is often found by designing a linear code with 2^k codewords. Therefore, many of the lower bounds in Table 1.1 are a power of 2.

This report mainly focuses on the upper bounds on $A(n,d)$ and how they can be found. In the next chapter a few of the most used methods to find an upper bound are explained. For one of these methods another type of maximum is needed: $A(n,d,w)$. $A(n,d,w)$ denotes the maximum number of codewords in any binary code of length n, Hamming distance d and weight w.

Definition 4. *The weight $w(\mathbf{x})$ of a vector $\mathbf{x} \in (\mathbb{F}_2)^n$ is the number of bits unequal to zero. In mathematical terms: $w(\mathbf{x}) = |\{i | x_i \neq 0\}|$*

Theorem 3. *For two vectors $\mathbf{x}, \mathbf{y} \in (\mathbb{F}_2)^n$ it holds that $d(\mathbf{x}, \mathbf{y}) = w(\mathbf{x} + \mathbf{y})$ [6]*

$A(n,d,w)$ is found in the same way as $A(n,d)$, by finding a lower- and upper bound. The tables for $A(n,d,w)$ for $n = 6, 7, \ldots 28$, $d = 4, 6, \ldots 12$ and $w = 2 \ldots 12$ used in this report are in Appendix C.

2

FINDING UPPER BOUNDS ON A(N,D)

When finding an upper bound U for $A(n,d)$, it follows that the maximum number of codewords in a binary code of length n and Hamming distance d can never be more than this upper bound. So it is desirable to find an upper bound on $A(n,d)$ that is as small as possible, because then the upperbound on $A(n,d)$ is as close to $A(n,d)$ as possible. When trying to find an upper bound on $A(n,d)$, there are several methods that can be used. These methods that give an upper bound for $A(n,d)$ are called bounds. In this chapter the following bounds are discussed:

1. Singleton bound
2. Plotkin bound
3. Hamming bound
4. Johnson bound

For every bound we give a detailed description, a proof and a clarifying example. The linear programming bound is treated seperately in Chapter 3.

2.1. THE SINGLETON BOUND

The Singleton bound is a crude upper bound on $A(n,d)$ which in most cases does not give the best upper bound. However, it is very easy to calculate.

Theorem 4 (Singleton bound [5, 16])**.** *Let C be a binary error-correcting code of length n and Hamming distance d, then the maximum number of codewords in C is limited by: $A(n,d) \leq 2^{n-d+1}$.*

Proof. [5, 16, 17] First observe that the total number of binary words of length n is 2^n, since each bit in a codeword may be a one or a zero independently of the remaining bits. Now let C be an arbitrary binary code with Hamming distance d. If we shorten the code by deleting the first $d-1$ bits of each codeword, then all resulting codewords must still be pairwise different. Thus the size of the new shortened code is the same as the original code C. The new codewords have length $n-(d-1) = n-d+1$, thus there can be at most 2^{n-d+1} new codewords. Since C was arbitrary and the same size as the new code, this bound must hold for the largest possible code with this length and Hamming distance. □

Example 7. We calculate the Singleton bound for $A(6,4)$, so $n = 6$ and $d = 4$:

$$A(6,4) \leq 2^{n-d+1} = 2^{6-4+1} = 2^3 = 8$$

Hence with Example 6 we find $4 \leq A(6,4) \leq 8$.

2.2. THE PLOTKIN BOUND

The Plotkin bound only applies to (n, M, d) codes with $n < 2d$. Just as the Singleton bound the Plotkin bound is easy to calculate, the proof however is a bit more difficult. For the special cases $n = 2d$ another Plotkin bound applies. We discuss this "second part" of the Plotkin bound later in this paragraph.

Theorem 5 (Plotkin bound [4, 5])**.** *Let C be a binary error-correcting code of length n and Hamming distance d. If $n < 2d$, then the maximum number of codewords in C is limited by:*

$$A(n,d) \leq 2\left\lfloor \frac{d}{2d-n} \right\rfloor \tag{2.1}$$

Proof. [5]Let $M = A(n,d)$ and let C be a (n, M, d) code. Define

$$S = \sum_{\mathbf{x}\in C} \sum_{\mathbf{y}\in C} d(\mathbf{x},\mathbf{y}) \tag{2.2}$$

Equation (2.1) is found by finding a lower and upper bound for S. We start by finding a lower bound for S, note:

$$d(\mathbf{x},\mathbf{y}) \begin{cases} = 0, \text{ if } \mathbf{x} = \mathbf{y} \\ \geq d, \text{ otherwise} \end{cases}$$

So for S (2.2) it follows that:

$$S \geq M(M-1)d \tag{2.3}$$

Now we find an upper bound for S. Let A be an $M \times n$ matrix whose rows are the cowords that belong to C. Suppose the i^{th} colomn of A contains α_i zeroes and $M - \alpha_i$ ones. Then the i^{th} colomn of A contributes $2\alpha_i(M-\alpha_i)$ to the sum S. So now S becomes:

$$S = \sum_{i=1}^{n} 2\alpha_i(M-\alpha_i) \tag{2.4}$$

We now distinguish two cases: M is even and M is odd. For the case that M is even, Equation (2.4) is maximized if for all i, $\alpha_i = \frac{1}{2}M$. So now we find:

$$S = \sum_{i=1}^{n} 2\alpha_i(M-\alpha_i) \leq \sum_{i=1}^{n} 2 * \frac{1}{2}M(M-\frac{1}{2}M) = \frac{1}{2}nM^2 \tag{2.5}$$

Combining (2.3) and (2.5) gives:

$$M(M-1)d \leq S \leq \frac{1}{2}nM^2 \Rightarrow$$
$$M \leq \frac{2d}{2d-n} \tag{2.6}$$

Because M is even equation (2.6) becomes:

$$M \leq 2\left\lfloor \frac{d}{2d-n} \right\rfloor \tag{2.7}$$

For the case that M is odd, Equation (2.4) is maximized if for all i, $\alpha_i = \frac{M-1}{2}$. So now we find:

$$S = \sum_{i=1}^{n} 2\alpha_i(M-\alpha_i) \leq n\frac{M^2-1}{2} \tag{2.8}$$

Combining (2.3) and (2.8) and using that $\lfloor 2x \rfloor \leq 2\lfloor x \rfloor + 1$, gives:

$$M(M-1)d \leq S \leq n\frac{M^2-1}{2} \Rightarrow$$
$$M \leq \frac{n}{2d-n} = \frac{2d}{2d-n} - 1 \leq 2\left\lfloor \frac{d}{2d-n} \right\rfloor \tag{2.9}$$

Equations (2.7) and (2.9) result in the Plotkin bound. □

Example 8. We calculate the Plotkin bound for $A(6,4)$. First we have to check whether $2d \geq n$, filling in $n = 6$ and $d = 4$ gives us: $2 * 4 \geq 6$. So the Plotkin bound applies to $A(6,4)$. Now we find:

$$A(6,4) \leq 2\left(\frac{d}{2d-n}\right) = 2\left(\frac{4}{2*4-6}\right) = 4$$

Hence with Example 6 we find $4 \leq A(6,4) \leq 4$. This implies: $A(6,4) = 4$.

Now we discuss the special cases $n = 2d$.

Theorem 6 (Plotkin bound "second part" [5])**.** *Let C be a binary error-correcting code of length n and Hamming distance d. If $n = 2d$ then,*

$$A(n,d) \leq 2n$$

Proof. From part 4 in Theorem 2 we know $A(n,d) \leq 2 * A(n-1,d)$. Now set $n = 2d$, with the Plotkin bound from 2.1 we find $A(n,d) = A(2d,d) \leq 2 * A(2d-1,d) \leq 2 * 2 * d = 4d = 2n$ □

2.3. THE HAMMING BOUND

The Hamming bound is somewhat different from the Singleton and Plotkin bound. The Hamming bound does not just count the number of possible codewords in a code C, it originates from a different perspective. The bound starts with all possible vectors with length n, this is 2^n, and then divides this by the volume of a "Hamming sphere". This is where the name sphere-packing bound comes from. The definition of a Hamming sphere and how these spheres are used to find an upper bound on $A(n,d)$ is given in the proof of Theorem 7.

Theorem 7 (Hamming bound [5])**.** *Let C be a binary error-correcting code of length n and Hamming distance d. If d is odd and $d = 2t+1$ then $A(n,d)$ is bound by:*

$$A(n,d) \leq \frac{2^n}{\sum_{k=0}^{t}\binom{n}{k}}$$

If $d = even$ we use part 3 of Theorem 2 to calculate the Hamming bound as shown in Example 9.

Proof. [5] Note that through the definition of d, at most $t = \frac{d-1}{2}$ errors can be corrected by nearest neighbor decoding. For a given codeword $\mathbf{x}_i$ $(i = 1,\ldots,M)$ in a code C, consider the sphere with radius t and center $\mathbf{x}_i$, see Figure 2.1. Every pair of spheres with a codeword $\mathbf{x}_i$ ($i = 1,\ldots,M$) in the center and radius t contains w words. Any two different spheres with centers $\mathbf{x}_i$ and $\mathbf{x}_j$ with $i, j = 1,\ldots,M$ and $i \neq j$ are non-intersecting. These spheres are called "Hamming spheres". The sum $w = \binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{t}$ describes the volume of such a sphere. The first term in the sum is the number of words that differ from $\mathbf{x}$ in zero positions, that is the number of words equal to $\mathbf{x}$. Therefore the first term is equal to 1. The second term in the sum is the number of words that differ from $\mathbf{x}$ in one position. There are $\binom{n}{1} = n$ positions in which a word might differ from $\mathbf{x}$. The other terms are similar, but for each consecutive term there is one more bit that differs from $\mathbf{x}$. The last term ads the number of words that differ from $\mathbf{x}$ in t positions. Note that the volume of the Hamming sphere is the number of words that decode to $\mathbf{x}$. When we multiply the number of spheres (this is equal to the number of codewords) with the volume of such a sphere, we get the total number of vectors in all of the spheres. This number has to be smaller or equal to the number of all possible vectors with length n, this is 2^n. In mathematical terms:

$$M * w = M * \sum_{k=0}^{t}\binom{n}{k} \leq 2^n \quad \Rightarrow \quad M \leq \frac{2^n}{\sum_{k=0}^{t}\binom{n}{k}}$$

Concluding, the maximum number of possible words in a binary alphabet of length n $(= 2^n)$, divided by the sum of the volumes of the Hamming spheres, gives an upper bound on the maximum number of spheres and therefore on the maximum number of codewords. □

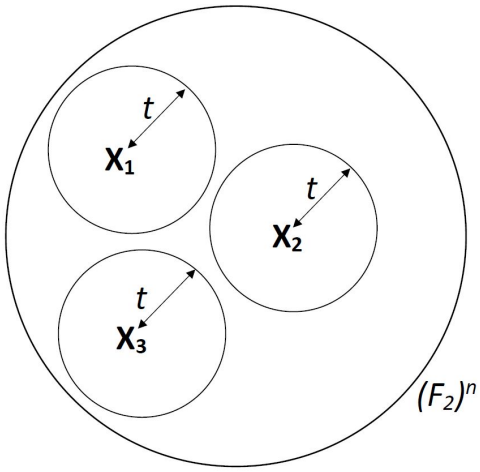


Figure 2.1: Illustration of the Hamming bound.

Example 9. We calculate the Hamming bound for $A(6,4)$. Note: $A(6,4) = A(5,3)$ by part 3 of Theorem 2, so we calculate $A(5,3)$ with odd d and $t = \frac{d-1}{2} = 1$.

$$A(5,3) \leq \frac{2^5}{\sum_{k=0}^{1} \binom{5}{k}} = \frac{32}{\binom{5}{0} + \binom{5}{1}} = \frac{32}{1+5} = 5\frac{1}{3}$$

There is no such thing as $\frac{1}{3}$ codeword, so we round down to the next lower integer. Hence with Example 6 we find $4 \leq A(6,4) = A(5,3) \leq 5$.

2.4. THE JOHNSON BOUND

The Johnson bound is an improvement on the Hamming bound. The Johnson bound also divides the total number of possible binary words of length n , 2^n, by the Hamming spheres from the previous proof. In addition it adds the number of vectors that differ $t+1$ from the center codeword $\mathbf{x}$, making sure they are not counted twice, to the denominator. In turn, this gives a larger denominator and a better upper bound for $A(n,d)$.

Theorem 8 (Johnson bound [5])**.** *Let C be a binary error-correcting code of length n and Hamming distance d. If d is odd and $d = 2t+1$ then $A(n,d)$ is bound by:*

$$A(n,d) \leq \frac{2^n}{\sum_{k=0}^{t}\binom{n}{k} + \frac{\binom{n}{t+1}-\binom{d}{t}A(n,d,d)}{A(n,d,t+1)}}$$

Proof. [5] For a given codeword $\mathbf{x}$ in a code C, consider the sphere with radius t and center $\mathbf{x}$. See Figure 2.2. Every pair of spheres with a different codeword in the center and radius t is non-intersecting and contains w words. The sum $w = \sum_{k=0}^{t}\binom{n}{k}$ gives the volume of such a sphere (just as in the proof of Theorem 7). The idea behind the Johnson bound is dividing the maximum number of possible words in a binary alphabet of length n, $(= 2^n)$, by the volume of the spheres with center $\mathbf{x}_i$ $(i = 1,\ldots,M)$ and radius $t+1$. However, this procedure no longer guarantees that any two spheres with a different codeword in the center and radius $t+1$ are non-intersecting, causing some of the vectors to be counted more than once. Note that the number of vectors that differ from $\mathbf{x}$ in $t+1$ places is $\binom{n}{t+1}$.

The next part focuses on counting how many vectors there are that differ from $\mathbf{x}$ in $t+1$ places without counting them more than once. Assume (without loss of generality) that $\mathbf{x}$ is the zero vector (vector consisting of only zeroes) and $\mathbf{f}$ is a vector with $d(\mathbf{x},\mathbf{f}) = t+1$. Notice that $w(\mathbf{f}) = t+1$.The number of spheres $\mathbf{f}$ could be contained in, is the number of codewords $\mathbf{h}$ at distance d from $\mathbf{x}$, times the number of vectors with weight $t+1$ and distance t from $\mathbf{h}$. The number of codewords $\mathbf{h}$ at distance d from $\mathbf{x}$ is at most $A(n,d,d)$ and the number of vectors with weight $t+1$ and distance t from $\mathbf{h}$ is $\binom{d}{t+1} = \binom{d}{t}$. So the number of vectors at distance $t+1$ from $\mathbf{x}$ without also being contained in the sphere with radius t of another codeword $\mathbf{h}$ is at most $\binom{n}{t+1} - \binom{d}{t}A(n,d,d)$. Take one of these vectors and call it $\mathbf{k}$. Assume (without loss of generality) that $\mathbf{k}$ is the zero vector. $\mathbf{k}$ could still be contained in another sphere with center $\mathbf{u}$, radius $t+1$ and $d(\mathbf{k},\mathbf{u}) = t+1$. The number of these spheres is at most $A(n,d,t+1)$, so dividing $\binom{n}{t+1} - \binom{d}{t}A(n,d,d)$ by $A(n,d,t+1)$ gives the maximum number of vectors that differ from $\mathbf{x}$ in $t+1$ places without being counted more than once. Now the Johnson bound is given by dividing the maximum number of possible words in a binary alphabet of length n, 2^n, by the volume of a Hamming sphere and the number of codewords that differ from $\mathbf{x}$ in $t+1$ places (counted once). □

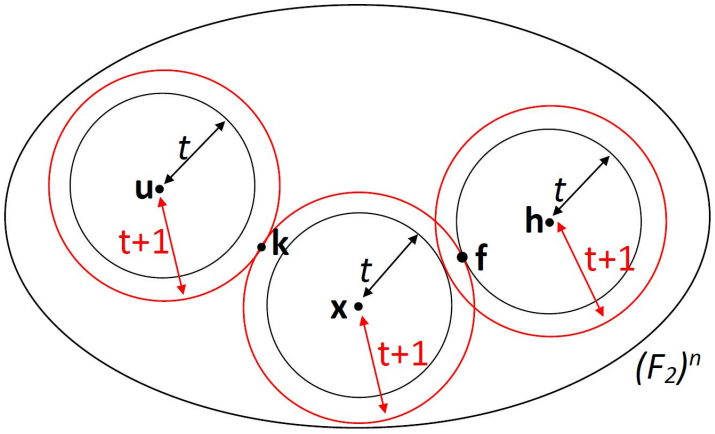


Figure 2.2: Illustration of the Johnson bound.

Example 10. We calculate the Johnson bound for $A(6,4)$.
Note: $A(6,4) = A(5,3)$ by part 3 of Theorem 2, so we calculate $A(5,3)$ with odd d and $t = \frac{d-1}{2} = 1$.

$$A(5,3) \leq \frac{2^n}{\sum_{k=0}^{t} \binom{n}{k} + \frac{\binom{n}{t+1} - \binom{d}{t} A(n,d,d)}{A(n,d,t+1)}} = \frac{2^5}{\binom{5}{0} + \binom{5}{1} + \frac{\binom{5}{2} - \binom{3}{2} * 2}{2}} = 4$$

The values for $A(n,d,w)$ can be found in the tables for $A(n,d,w)$ for $n = 6 \ldots 28$, $d = 4 \ldots 12$ and $w = 2 \ldots 12$ in Appendix C.

3

LINEAR PROGRAMMING BOUND

A widely used method for finding $A(n,d)$ is the linear programming bound, which optimizes a single objective function subject to some linear constraints. This chapter starts with an explanation of the canonical form of a linear programming problem. Hereafter we present the theorem on which the linear programming bound is based and how linear programming is used to find an upper bound on $A(n,d)$.

3.1. LINEAR PROGRAMMING

In linear programming a single objective function is optimized while subject to constraints. The problem is usually expressed in canonical form as:

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}^T\mathbf{x} \\ \text{such that} \quad & \mathbf{H}\mathbf{x} \leq \mathbf{b}, \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

Here $\mathbf{c}^T$ is a transposed n-dimensional vector, $\mathbf{b}$ is a m-dimensional vector, $\mathbf{x}$ is a n-dimensional vector and $\mathbf{H}$ is a $m \times n$-matrix. The vector $\mathbf{x}$ consists of variables that have to be determined. Although it is still not known if it is polynomial in the worst case scenario, the simplex and dual simplex methods are often used for solving linear programming problems because they are efficient in practice. The dual simplex method is the same as first writing the linear program in the dual form and then using the simplex method to find the optimal solution. The dual form of a linear programming problem is formulated by in Table 3.1.

LP formulation	Dual formulation
Max $\mathbf{c}^T\mathbf{x}$ $\mathbf{x} \geq \mathbf{0}$ $\mathbf{H}\mathbf{x} \leq \mathbf{b}$	Min $\mathbf{b}^T\mathbf{y}$ $\mathbf{H}^T\mathbf{y} \geq \mathbf{b}$ $\mathbf{y} \geq 0$

Table 3.1: Primal to Dual transformation of a linear program [18]

Theorem 9 (Duality [18])**.** *Let* $\mathbf{x}$ *and* $\mathbf{y}$ *be feasible solutions to the LP formulation and dual formulation respectively. Then* $\mathbf{x}$ *and* $\mathbf{y}$ *are both optimal if and only if* $\mathbf{c}^T\mathbf{x} = \mathbf{b}^T\mathbf{y}$

Example 11. f Suppose we want to find the optimal solution for the following linear program.

$$\begin{aligned} \text{maximize} \quad & x_1 + x_2 \\ \text{such that} \quad & x_1 + 2x_2 \leq 10, \\ & -x_1 + x_2 \leq 3, \\ & 3x_1 + x_2 \leq 10 \\ & x_1 \geq 0 \\ & x_2 \geq 0 \end{aligned}$$

In Figure 3.1 a graphical representation is given. The dashed line indicates the objective function and the set of feasible solutions is depicted in the grey shaded area. In general, if there exists an optimal solution to the problem, the objective function takes it's maximum (or minimum) in one of the corners. In Figure 3.1 we can see that a solution is found for $\mathbf{x} = (2,4)$, this solution gives the maximum value for the objective function, $x_1 + x_2 = 2 + 4 = 6$.

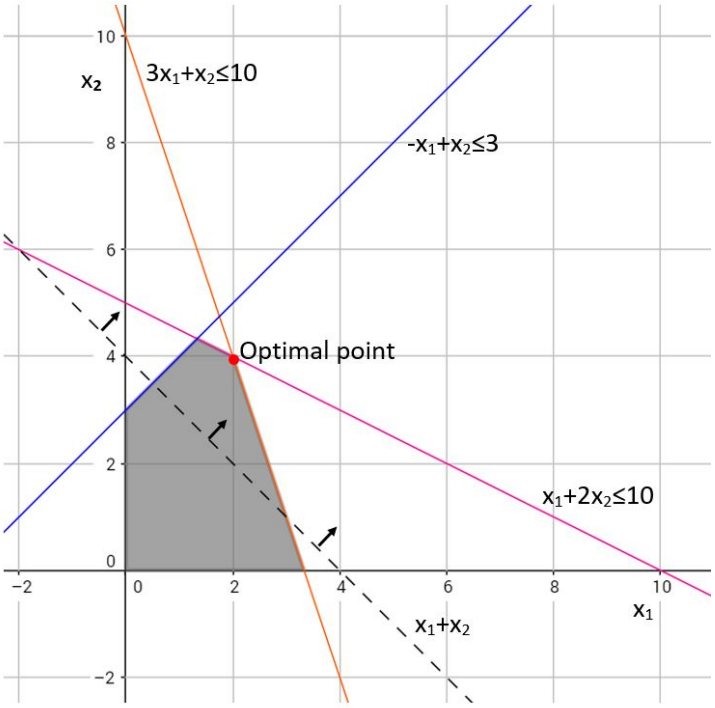


Figure 3.1: Graphical representation of the linear program from Example11

Another possibility to find the optimal solution is to use the dual representation of this linear program. We find the dual linear program by using Table 3.1.

$$\begin{array}{ll} \text{minimize} & 10y_1 + 3y_2 + 10y_3 \\ \text{such that} & y_1 - y_2 + 3y_3 \geq 1, \\ & 2y_1 + y_2 + y_3 \geq 1, \\ & y_1 \geq 0 \\ & y_2 \geq 0 \\ & y_2 \geq 0 \end{array}$$

Now by using the simplex method we find a solution for $y_1 = 0.4$, $y_2 = 0$ and $y_3 = 0.2$, which gives $10y_1 + 3y_2 + 10y_3 = 6$ as the minimum value of the objective function. With Theorem 9 we now find that $\mathbf{x}$ and $\mathbf{y}$ are optimal.

3.2. DELSARTE'S THEOREM

This section explains Delsarte's theorem, where the linear programming bound is based upon. For this theorem two new definitions are needed and these are provided next.

Definition 5. *Let C be an (n, M, d) code. The weight distribution of C with respect to a vector $\mathbf{u}$ is the $(n+1)$-tuple of integers $(A_i(\mathbf{u}), i = 0, \cdots, n)$, where $A_i(\mathbf{u})$ is the number of codewords $\mathbf{v} \in C$ such that $d(\mathbf{u}, \mathbf{v}) = i$.*

Definition 6. *Let C be an (n, M, d) code. The distance distribution of C is the $(n+1)$-tuple of rational numbers $(A_0, A_1, \ldots, A_n)$, defined by $A_i = \frac{1}{M} \sum_{\mathbf{u} \in C} A_i(\mathbf{u})$*

Note that for every (n, M, d) code C $A_0 = 1$, $A_i \geq 0$ and $\sum_i A_i = M \leq A(n, d)$. In Example 12 we give the weight and distance distributions of a code C_2.

Example 12. We build a $(5, 4, 2)$ code C_2 and give the weight and distance distributions:

codeword	$A_0(\mathbf{u})$	$A_1(\mathbf{u})$	$A_2(\mathbf{u})$	$A_3(\mathbf{u})$	$A_4(\mathbf{u})$	$A_5(\mathbf{u})$
$\mathbf{u}_1 = 00\,101$	1	0	0	2	0	1
$\mathbf{u}_2 = 11\,010$	1	0	2	0	0	1
$\mathbf{u}_3 = 10\,011$	1	0	1	1	1	0
$\mathbf{u}_4 = 01\,110$	1	0	1	1	1	0
	$A_0 = 1$	$A_1 = 0$	$A_2 = 1$	$A_3 = 1$	$A_4 = \frac{1}{2}$	$A_5 = \frac{1}{2}$

In Example 13 the weight and distance distributions of linear code C_1 from Example 3 are given.

Example 13. Recall the linear code C_1 from Example 3, we give the weight and distance distributions:

codeword	$A_0(\mathbf{u})$	$A_1(\mathbf{u})$	$A_2(\mathbf{u})$	$A_3(\mathbf{u})$	$A_4(\mathbf{u})$	$A_5(\mathbf{u})$
$\mathbf{u}_1 = 00\,000$	1	0	0	2	1	0
$\mathbf{u}_2 = 01\,011$	1	0	0	2	1	0
$\mathbf{u}_3 = 10\,101$	1	0	0	2	1	0
$\mathbf{u}_4 = 11\,110$	1	0	0	2	1	0
	$A_0 = 1$	$A_1 = 0$	$A_2 = 0$	$A_3 = 2$	$A_4 = 1$	$A_5 = 0$

In general it holds for a linear code that $A_i(\mathbf{u}) = A_i(\mathbf{v})$ for $\mathbf{u}, \mathbf{v} \in C$. So it also holds that $A_i = A_i(\mathbf{u})$ for any $\mathbf{u} \in C$ [9]. Before we can discuss the Linear programming bound we need a new theorem, Delsarte's theorem:

Theorem 10 (Delsarte [9])**.** *Let C be an (n, M, d) code with distance distribution $(A_0, A_1, \cdots, A_n)$ Then the quantities $B_0, B_1, \cdots, B_n$ are nonnegative, where*

$$B_k = M^{-1} \sum_{i=0}^{n} A_i K_k(i), \quad k = 0, 1, \cdots, n \tag{3.1}$$

with K_k a Kwawtchouk polynomial, defined by

$$K_k(t) = \sum_{j=0}^{k} (-1)^j \binom{n-t}{k-j} \binom{t}{j}, \quad k = 0, 1, \cdots, n \tag{3.2}$$

Proof. [9] First consider the dotproduct: $\langle \mathbf{w}, \mathbf{x} \rangle = \sum_{i=0}^{n} \mathbf{w}_i \mathbf{x}_i$ with $w(\mathbf{w}) = i$ and $w(\mathbf{x}) = k$ as in Figure 3.2.

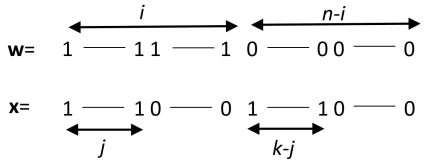


Figure 3.2: composition of $\mathbf{w}$ and $\mathbf{x}$

Then the number of different $\mathbf{x}$ with fixed $\mathbf{w}$ and j is:

$$\binom{i}{j}\binom{n-i}{k-j} \tag{3.3}$$

Note that

$$\langle \mathbf{w}, \mathbf{x} \rangle = \begin{cases} 0, \text{ if } j \text{ is even} \\ 1, \text{ if } j \text{ is odd} \end{cases} \tag{3.4}$$

so combining (3.2), (3.3) and (3.4) we find:

$$\sum_{\substack{\mathbf{x}\in\{0,1\}^n \\ wt(\mathbf{x})=k}} (-1)^{\langle \mathbf{w},\mathbf{x} \rangle} = \sum_{j=0}^{k} (-1)^j \binom{n-i}{k-j}\binom{i}{j} = K_k(i) \tag{3.5}$$

Now we use this to rewrite B_k and find that they are all nonnegative.

$$B_k = M^{-1} \sum_{i=0}^{n} A_i K_k(i)$$

Now we fill in the definition for $A_i = \frac{1}{M}\sum_{\mathbf{u}\in C} A_i(\mathbf{u})$ and we get:

$$= M^{-1} \sum_{i=0}^{n} M^{-1} \sum_{\mathbf{u}\in C} A_i(\mathbf{u}) K_k(i)$$

With the definition of $A_i(\mathbf{u})$ we find:

$$= M^{-2} \sum_{i=0}^{n} \sum_{\mathbf{u}\in C} \sum_{\substack{\mathbf{v}\in C \\ wt(\mathbf{u}-\mathbf{v})=i}} K_k(i)$$

With equation 3.5 we find:

$$= M^{-2} \sum_{i=0}^{n} \sum_{\mathbf{u}\in C} \sum_{\substack{\mathbf{v}\in C \\ wt(\mathbf{u}-\mathbf{v})=i}} \sum_{\substack{\mathbf{x}\in\{0,1\}^n \\ wt(\mathbf{x})=k}} (-1)^{\langle \mathbf{u}-\mathbf{v},\mathbf{x} \rangle}$$

By switching the sums and rewriting the inproduct we find:

$$= M^{-2} \sum_{\substack{\mathbf{x}\in\{0,1\}^n \\ wt(\mathbf{x})=k}} \sum_{\mathbf{u}\in C} \sum_{i=0}^{n} \sum_{\substack{\mathbf{v}\in C \\ wt(\mathbf{u}-\mathbf{v})=i}} (-1)^{\langle \mathbf{u},\mathbf{x} \rangle} (-1)^{\langle \mathbf{v},\mathbf{x} \rangle}$$

By combining the two last sum signs we find:

$$= M^{-2} \sum_{\substack{\mathbf{x}\in\{0,1\}^n \\ wt(\mathbf{x})=k}} \sum_{\mathbf{u}\in C} (-1)^{\langle \mathbf{u},\mathbf{x} \rangle} \sum_{\mathbf{v}\in C} (-1)^{\langle \mathbf{v},\mathbf{x} \rangle}$$

$$= M^{-2} \sum_{\substack{\mathbf{x}\in\{0,1\}^n \\ wt(\mathbf{x})=k}} (b_{\mathbf{x}})^2 \quad \text{, with } b_{\mathbf{x}} = \sum_{\mathbf{u}\in C} (-1)^{\langle \mathbf{u},\mathbf{x} \rangle}$$

Because the square is always nonnegative it follows that $B_k \geq 0$. □

3.3. THE BOUND

Let C be a code with the maximum number of codewords for a given length n and Hamming distance d, then

$$M = A(n,d) = 1 + A_d + A_{d+1} + \cdots + A_n$$

Theorem 11 (Linear programming bound [9])**.** *Suppose $L(n,d)$ is the optimal solution to the following linear programming problem:*

$$\begin{array}{lll} \textit{maximize} & A_d + A_{d+1} + \cdots + A_n & \\ \textit{such that} & A_i \geq 0, & i = d, \cdots, n, \\ & B_k \geq 0, & k = 0, \cdots, n \end{array} \tag{3.6}$$

Where $B_k = M^{-1} \sum_{i=0}^{n} A_i K_k(i)$. Then $A(n,d) \leq 1 + L(n,d)$. This is the simplest form of the linear programming bound for binary codes.

To apply this linear programming bound we need a few new observations:

Observation 1 (Puncturing [19])**.** Suppose C_1 is a (n,M,d) code with even d. Choose two codewords $\mathbf{x}$ and $\mathbf{y}$ with $d(\mathbf{x},\mathbf{y}) = d$. Search for a bit in which $\mathbf{x}$ and $\mathbf{y}$ differ, say the i^{th} bit. Now remove the i^{th} bit from every codeword. The distance between $\mathbf{x}$ and $\mathbf{y}$ now becomes $d-1$ and we obtain a $(n-1,M,d-1)$ code. This is called puncturing a code.

Theorem 12 (Extending [19])**.** *Suppose C_1 is a (n,M,d) code with odd d. We build a new code C_2 by adding a "0" at the end of every codeword with even weight and a "1" at the end of every codeword with odd weight. Then we obtain a $(n+1,M,d+1)$ code C_2. All the codewords in C_2 now have even weight. This is called extending a code.*

Proof. [19] Take $\mathbf{x},\mathbf{y} \in C$ with $d(\mathbf{x},\mathbf{y}) = d$. We first proof the following equation: $d(\mathbf{x},\mathbf{y}) = w(\mathbf{x}) + w(\mathbf{y}) - 2w(\mathbf{x} \cap \mathbf{y})$, where $w(\mathbf{x} \cap \mathbf{y})$ denotes the number of positions both $\mathbf{x}$ and $\mathbf{y}$ have a "1". For $d(\mathbf{x},\mathbf{y})$ we only want to count the ones where $\mathbf{x}$ and $\mathbf{y}$ differ. When adding $w(\mathbf{x})$ and $w(\mathbf{y})$, the total number of ones in $\mathbf{x}$ and $\mathbf{y}$ is counted. This means that the ones that $\mathbf{x}$ and $\mathbf{y}$ have in common are counted twice, namely once in $\mathbf{x}$ and once in $\mathbf{y}$. Thus we need to deduct $2w(\mathbf{x} \cap \mathbf{y})$ from $w(\mathbf{x}) + w(\mathbf{y})$. Now we proceed with the proof of Theorem 12. Because $d(\mathbf{x},\mathbf{y})$ is odd, $w(\mathbf{x}) + w(\mathbf{y}) - 2w(\mathbf{x} \cap \mathbf{y})$ has to be odd as well. Notice that $2w(\mathbf{x} \cap \mathbf{y})$ is always even. Hence we find that $w(\mathbf{x})$ or $w(\mathbf{y})$ has to be odd and the other even. This means that we add a "1" to either $\mathbf{x}$ or $\mathbf{y}$ and a "0" to the other. Therefore the minimum distance becomes $d+1$. □

Theorem 13. *[9] Suppose C_1 is a (n,M,d) code with d even. By first puncturing and then extending this code we find an (n,M,d) code C_2 in which all distances are even.*

Proof. We use the following equation: $d(\mathbf{x},\mathbf{y}) = w(\mathbf{x}) + w(\mathbf{y}) - 2w(\mathbf{x} \cap \mathbf{y})$ (the proof can be found in the proof of Theorem 12), where $w(\mathbf{x} \cap \mathbf{y})$ denotes the number of positions both $\mathbf{x}$ and $\mathbf{y}$ have a "1". We know $w(\mathbf{x})$, $w(\mathbf{y})$ and $2w(\mathbf{x} \cap \mathbf{y})$ are even, hence $d(\mathbf{x},\mathbf{y})$ is even. □

In the following example an upper bound for $A(6,4)$ is found by using the Linear programming bound.

Example 14. Finding an upper bound for $A(6,4)$ with the linear programming bound. Let C be a (6,M,4) code, first observe that by Theorem 13 we can puncture and then extend this code to obtain a (6,M,4) code in which all distances are even. If $(A_0, A_1, \ldots, A_n)$ is the distance distribution of C, then $A_0 = 1$ and the remaining A_i's are zero except (possibly) for A_4 and A_6. The inequalities $B_k \geq 0$ from 3.6 become:

$$\begin{aligned} B_0 &= M^{-1}(A_0K_0(0) + A_4K_0(4) + A_6K_0(6)) = M^{-1}(1 + A_4 + A_6) \geq 0 \\ B_1 &= M^{-1}(A_0K_1(0) + A_4K_1(4) + A_6K_1(6)) = M^{-1}(6 - 2A_4 - 6A_6) \geq 0 \\ B_2 &= M^{-1}(A_0K_2(0) + A_4K_2(4) + A_6K_2(6)) = M^{-1}(15 - 1A_4 + 15A_6) \geq 0 \\ B_3 &= M^{-1}(A_0K_3(0) + A_4K_3(4) + A_6K_3(6)) = M^{-1}(20 + 4A_4 - 20A_6) \geq 0 \end{aligned} \tag{3.7}$$

There is a certain symmetry in the Krawtchouck polynomials that cause: $B_0 = B_6$, $B_1 = B_5$ and $B_2 = B_4$. Because $M \geq 0$, the inequalities from 3.7 now become:

$$\begin{aligned} 1 + A_4 + A_6 &\geq 0 \\ 6 - 2A_4 - 6A_6 &\geq 0 \\ 15 - 1A_4 + 15A_6 &\geq 0 \\ 20 + 4A_4 - 20A_6 &\geq 0 \end{aligned} \tag{3.8}$$

When we add the objective function "maximize $A_4 + A_6$" and write the inequalities in the form of a linear program from Section 3.1 with

$$\mathbf{H} = \begin{pmatrix} -1 & -1 \\ 2 & 6 \\ 1 & -15 \\ -4 & 20 \end{pmatrix}, \mathbf{x} = \begin{pmatrix} A_4 \\ A_6 \end{pmatrix} \text{ and } \mathbf{b} = \begin{pmatrix} 1 \\ 6 \\ 15 \\ 20 \end{pmatrix}$$

we find:

$$\begin{aligned} \text{maximize} \quad & A_4 + A6 \\ \text{such that} \quad & -A_4 - A_6 \leq 1, \\ & 2A_4 + 6A_6 \leq 6 \\ & A_4 - 15A_6 \leq 15 \\ & -4A_4 + 20A_6 \leq 20 \\ & A_4 \geq 0 \\ & A_6 \geq 0 \end{aligned}$$

Now applying the dual simplex method we find that $A_4 = 3$ and $A_6 = 0$ and the optimal solution $A_4 + A_6 = 3$. Consequently, this gives $A(6,4) \leq 1 + A_4 + A_6 = 4$.

3.4. ADDITIONAL CONSTRAINTS

In the linear programming problem of (3.6) it is possible to add extra constraints. Often these constraints lead to a lower optimal solution for the linear programming problem. This lower $L(n,d)$ leads to a lower upper bound for $A(n,d) \leq 1 + L(n,d)$. So adding extra constraints can result in a better upper bound. For these extra constraints the known bounds on $A(n,d,w)$ are used (see Appendix C).

Theorem 14. *[9] Let C be a (n,M,d) code. The number of codewords $\mathbf{v} \in C$ such that $d(\mathbf{u},\mathbf{v}) = i$ $(i = d,\ldots,n)$ is less or equal to the number of codewords $\mathbf{v} \in C$ such that $w(\mathbf{v}) = i$,*

$$A_i(\mathbf{u}) \leq A(n,d,i) \tag{3.9}$$

Because $A_i(\mathbf{u}) \leq A(n,d,i)$ for all $\mathbf{u} \in C$ it follows that $A_i \leq A(n,d,i)$

Proof. Let $\mathbf{u}$ be a codeword in a (n,M,d) code C. Note that

$$A_i(\mathbf{u}) = |\{\mathbf{v} \in C \text{ such that } d(\mathbf{u},\mathbf{v}) = i\}| \tag{3.10}$$

Now examine the $\mathbf{v}$ from 3.10: Those $\mathbf{v}$ have distance i from $\mathbf{u}$. Then, if $\mathbf{u}$ is transferred to the codeword containing only zeroes (without loss of generality), the $\mathbf{v}$ all have weight i. The number of codewords $\mathbf{v}$ that have weight i is $A(n,d,i)$. So now we find $A_i(\mathbf{u}) \leq A(n,d,i)$. □

In Example 15 Theorem 14 is used to find an upper bound for $A(6,4)$ from Example 14.

Example 15. We add the extra constraints $A_i(\mathbf{u}) \leq A(n,d,i)$ to the linear program from Example 14.

$$\begin{aligned} A_4(\mathbf{u}) &\leq A(6,4,4) = 3, \\ A_6(\mathbf{u}) &\leq A(6,4,6) = 1 \end{aligned}$$

Now the linear program from Example 14 becomes:

$$\begin{array}{ll} \text{maximize} & A_4 + A_6 \\ \text{such that} & -A_4 - A_6 \leq 1, \\ & 2A_4 + 6A_6 \leq 6 \\ & A_4 - 15A_6 \leq 15 \\ & -4A_4 + 20A_6 \leq 20 \\ & A_4 \leq 3, \\ & A_6 \leq 1 \\ & A_4 \geq 0 \\ & A_6 \geq 0 \end{array}$$

with

$$\mathbf{H} = \begin{pmatrix} -1 & -1 \\ 2 & 6 \\ 1 & -15 \\ -4 & 20 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}, \mathbf{x} = \begin{pmatrix} A_4 \\ A_6 \end{pmatrix} \text{ and and } \mathbf{b} = \begin{pmatrix} 1 \\ 6 \\ 15 \\ 20 \\ 3 \\ 1 \end{pmatrix}$$

By applying the dual simplex method we find that $A_4 = 3$ and $A_6 = 0$ and the maximum value for the objective function, $A_4 + A_6 = 3$. In turn, this gives $A(6,4) \leq 1 + A_4 + A_6 = 4$. In this case we don't find a better upper bound for $A(6,4)$. From Example 6 we already know that $A(6,4) \geq 4$, therefore finding a better upper bound is impossible.

4

COMPARING BOUNDS

In this chapter we compare the six bounds introduced in Chapter 2 and 3. For this purpose we introduce a new notation: $U(n,d)$, this denotes the result for n and d given by the bound corresponding to U. We then have: $Plot(n,d)$, $Sing(n,d)$, $Ham(n,d)$, $John(n,d)$, $Lin(n,d)$ and $LinExtra(n,d)$. We also use $A^*(n,d)$ to denote the best known upper bound from Figure 1.1. In Section 4.1 we explain how we calculated all of the results for the bounds using Matlab. Next, in Section 4.2 we compare the Plotkin and Singleton bound, in Section 4.3 we compare the Hamming and Johnson bound and in Section 4.4 we compare the linear programming bound to the linear programming bound with extra constraints. Also, in Section 4.4 we zoom in on the linear programming bound without and with extra constraints for $d = 4$. To conclude this chapter, we compare the linear programming bound with extra constraints to the Johnson bound and discuss a special case for $n = 24$ and $d = 4$ in Section 4.5.

4.1. MATLAB IMPLEMENTATION

To obtain all the results for the six different bounds for $n = 6,7,\ldots,28$ and $d = 4,6,\ldots,12$, we implemented these in Matlab. The Matlab codes can be found in Appendix A. To scale and compare the results we use a variable similar to the code rate, introduced in Chapter 1. This variable is defined as the logarithm of the results for $U(n,d)$ devided by n, the use hereof also allows for a better graphical representation.

For the linear programming bound with and without extra constraints we used the Matlab function "linprog(**f**,**A**,**b**,Aeq,beq,lb,ub,options)". This function finds the optimal solution by means of iteration and has an "Function Tolerance" of $1 \cdot 10^{-8}$. This is a stopping criterium for the function "linprog", and means that the iterations end when the step size in the objective function is less then $1 \cdot 10^{-8}$. Due to this tolerance the results found for $Lin(n,d)$ and $LinExtra(n,d)$ have an uncertainty of up to a $1 \cdot 10^{-8}$, therefore we round the results to six decimals.

4.2. PLOTKIN AND SINGLETON BOUND

First we compare two bounds that are straightforward to calculate, the Plotkin bound and Singleton bound. For all $n = 6,7,\ldots,28$ and $d = 4,6,\ldots,12$, we compute $Plot(n,d)$ and $Sing(n,d)$ using Matlab as discussed in Section 4.1. For $n = 2d$ we use the "second" part of the Plotkin bound. The results for all $d = 4,6,\ldots,12$ can be found in Appendix B and in Figure 4.1 a graphical representation is given. The bounds on $A(n,d)$ for $d = 8$ can also be found in Table 4.1. When a result is equal to $A^*(n,d)$ it is denoted in blue.

The Plotkin bound is unmistakably better than the Singleton bound for every $n = 6,7,\ldots,2d$ and $d = 4,6,\ldots,12$. The only downside of the Plotkin bound is that it gives no results for $n > 2d$. This is why there are no values in Table 4.1 for $n > 16$. However, when the Plotkin bound does apply, $Plot(n,d)$ is always (for $n = 6,7,\ldots,2d$ and $d = 4,6,\ldots,12$) equal to $A^*(n,d)$.

Upon examination of the Singleton bound we find that, by augmenting n with 1 for $n \geq d$, $Sing(n+1,d)$ becomes two times larger then $Sing(n,d)$. This explains the rapid increase of the results for $Sing(n,d)$. Overall the Singleton bound is not useless, but when comparing $A^*(n,d)$ (shown in Appendix B) with $Sing(n,d)$, it is safe to say that it is of no use when searching for $A(n,d)$. In the cases that $Sing(n,d) = A^*(n,d)$, then the Plotkin bound gives the same result.

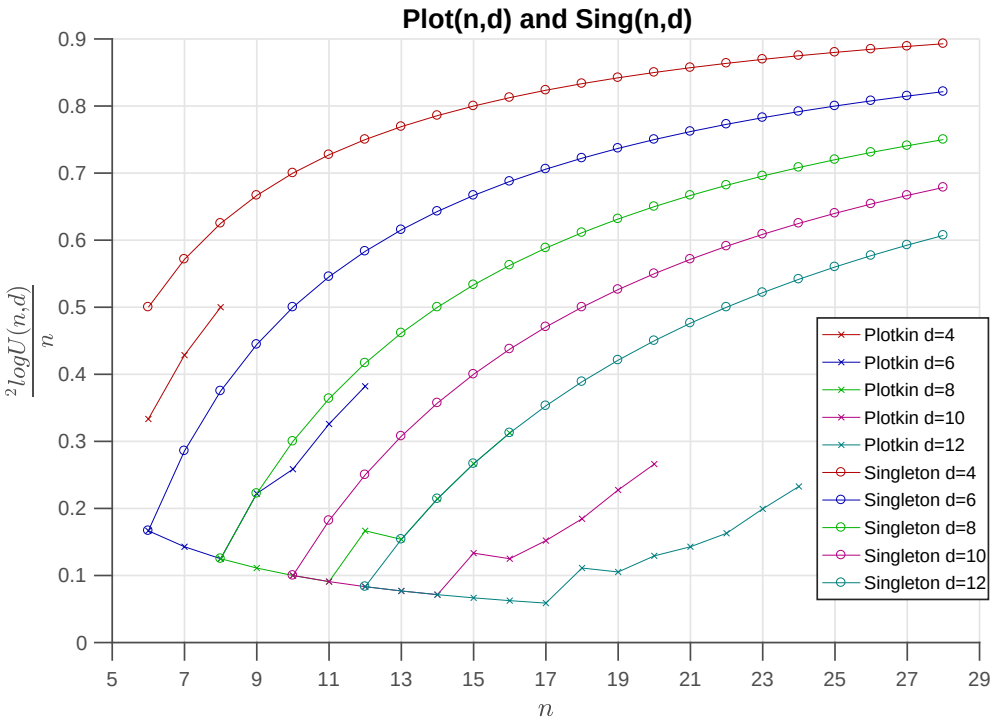


Figure 4.1: Upper bounds on A(n,d) found with the Plotkin and Singleton bound.

n	$A^*(n,8)$	$Plot(n,8)$	$Sing(n,8)$
6	1	1	1
7	1	1	1
8	2	2	2
9	2	2	4
10	2	2	8
11	2	2	16
12	4	4	32
13	4	4	64
14	8	8	128
15	16	16	256
16	32	32	512
17	36		1.024
18	71		2.048
19	131		4.096
20	256		8.192
21	512		16.384
22	1024		32.768
23	2048		65.536
24	4096		131.072
25	5421		262.144
26	9275		524.288
27	17099		1.048.576
28	32151		2.097.152

Table 4.1: Upper bounds on A(n,8) found with the Plotkin and Singleton bound

4.3. HAMMING AND JOHNSON BOUND

In this paragraph we compare the Hamming bound to the Johnson bound. The Hamming bound is fairly straightforward to calculate and it already has much better results than the Singleton bound, see Appendix B. For all $n = 6,7,\ldots,28$ and $d = 4,6,\ldots,12$, we calculate $Ham(n,d)$ and $John(n,d)$ using Matlab as discussed in Section 4.1. The results for all $d = 4,6,\ldots,12$ are calculated using $A(n-1,2e-1) = A(n,2e)$ from part 3 of Theorem 2 and can be found in Appendix B, in Figure 4.2 a graphical representation is given. The bounds on $A(n,d)$ for $d = 6$ can also be found in Table 4.2, when a result is equal to $A^*(n,d)$ it is denoted in blue.

When closely observing the results in Figure 4.2 and in Appendix B, it is clear that the Johnson bound gives a better upper bound for $A(n,d)$ than the Hamming bound (for $d = 4,6,\ldots,12$ and $n = 6,7,\ldots,28$). The Johnson bound improves the Hamming bound (when $Ham(n,d) > A^*(n,d)$ and for $d = 4,6,\ldots,12$, $n = 6,7,\ldots,28$) in %91,3 of the cases. $Jon(n,d)$ is equal to $A^*(n,d)$ in %28,7 of the cases (for $d = 4,6,\ldots,12$, $n = 6,7,\ldots,28$).

From the definitions of $Ham(n,d)$ and $John(n,d)$ it is clear that $John(n,d) \leq Ham(n,d)$. A non-negative value is added to the denominator of the Hamming bound to obtain the denominator of the Johnson bound. Therefore, the denominator in $John(n,d)$ is equal to or larger than the denominator in $Ham(n,d)$. Because the numerators of the Hamming and Johnson bound are equal, it is now clear that $John(n,d) \leq Ham(n,d)$. A downside of the Johnson bound is the dependence on the tables for $A(n,d,w)$, this is quite a challenge to implement in Matlab. However, it often provides good values for the upper bound and the values are often close to $A^*(n,d)$.

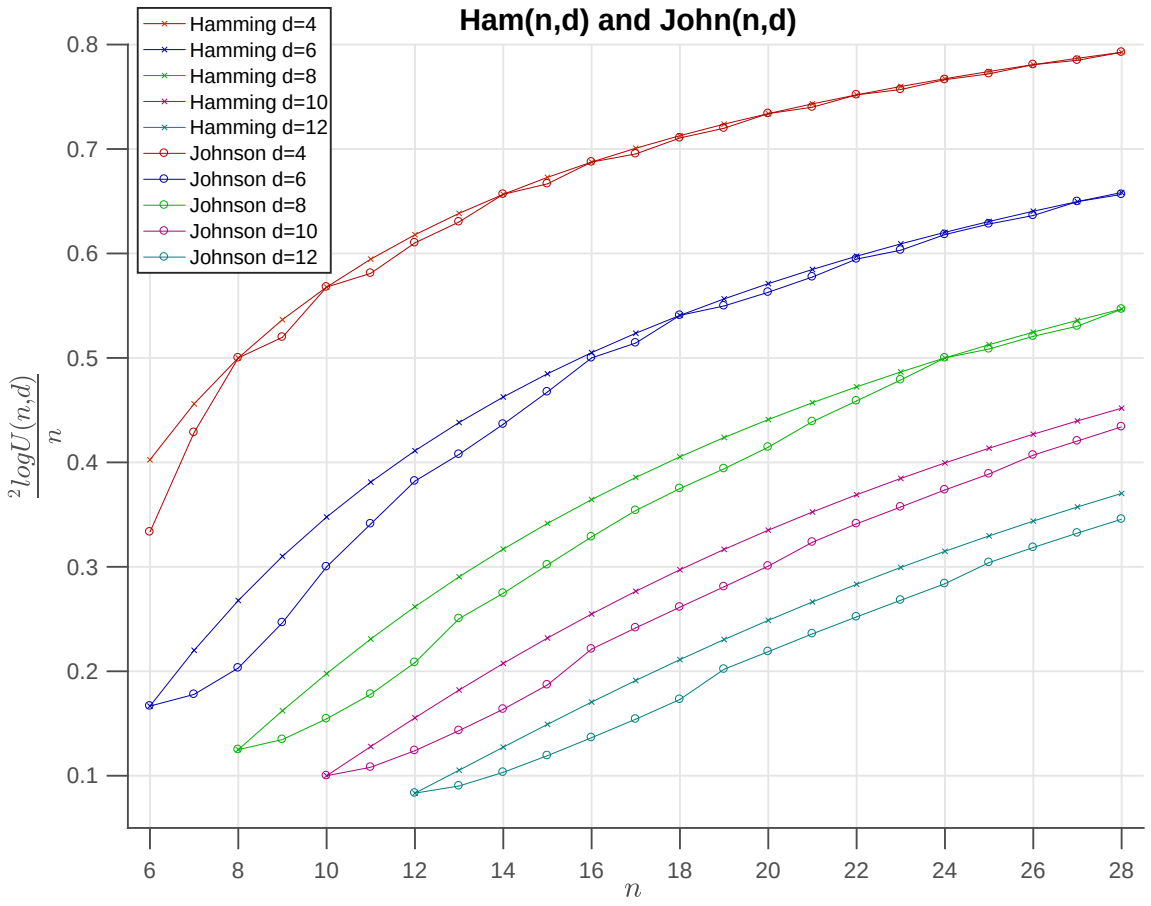


Figure 4.2: Upper bounds on A(n,d) found with the Hamming and Johnson bound.

n	$A^*(n,6)$	$Ham(n,6)$	$John(n,6)$
6	2	2	2
7	2	2	2
8	2	4	3
9	4	6	4
10	6	11	8
11	12	18	13
12	24	30	24
13	32	51	39
14	64	89	69
15	128	154	129
16	256	270	256
17	340	478	428
18	673	851	851
19	1.237	1.524	1.394
20	2.279	2.744	2.448
21	4.096	4.969	4.474
22	6.941	9.039	8.665
23	13.674	16.513	14.994
24	24.106	30.283	29.214
25	47.538	55.738	53.430
26	84.260	102.927	95.596
27	157.285	190.650	190.650
28	291.269	354.136	341.617

Table 4.2: Upper bounds on A(n,6) found with the Hamming and Johnson bound

4.4. LINEAR PROGRAMMING BOUND WITH AND WITHOUT EXTRA CONSTRAINTS

In this paragraph we compare the linear programming bound to the linear programming bound with extra constraints. For all $n = 6,7,\ldots,28$ and $d = 4,6,\ldots,12$, we calculate $Lin(n,d)$ and $LinExtra(n,d)$ using Matlab as discussed in Section 4.1. The results for all $d = 4,\ldots,12$ can be found in Appendix B and in Figure 4.3 a graphical representation is given. The bounds on $A(n,d)$ for $d = 4$ can also be found in Table 4.3. When a result is equal to $A^*(n,d)$ it is denoted in blue.

When closely observing the results in Figure 4.3 and in Appendix B, it is clear that the linear programming bound with extra constraints gives an equal or better upper bound for $A(n,d)$ than the linear programming bound (for $d = 4,6,\ldots,12$ and $n = 6,7,\ldots,28$). The linear programming bound with extra constraints improves the linear programming bound (when $Lin(n,d) > A^*(n,d)$ and for $d = 4,6,\ldots,12$, $n = 6,7,\ldots,28$) in %63,2 of the cases. $LinExtra(n,d)$ is equal to $A^*(n,d)$ in %58,3 of the cases (for $d = 4,6,\ldots,12$, $n = 6,7,\ldots,28$).

The linear programming bound is more complex to calculate than the previously mentioned bounds and gives a reasonably good upper bound on $A(n,d)$. In Appendix B it is easy to see that the linear programming bound already has better results than the Hamming bound. From the definitions of $Lin(n,d)$ and $LinExtra(n,d)$ it is clear that $LinExtra(n,d) \leq Lin(n,d)$. In $Lin(n,d)$ extra constraints are added to obtain $LinExtra(n,d)$. These extra constraints are of the form: $A_i \leq A(n,d,i)$ $(i = d,\ldots,n)$. A lower A_i results in a lower objective function: maximize $A_d + A_{d+1} + \cdots + A_n$. Consequently the extra constraints can only result in a lower upper bound on $A(n,d)$. A downside of the linear programming bound with extra constraints is the dependence on the tables for $A(n,d,w)$, this is quite a challenge to implement in Matlab. However, it provides good values for the upper bound, which are close to and often equal to $A^*(n,d)$.

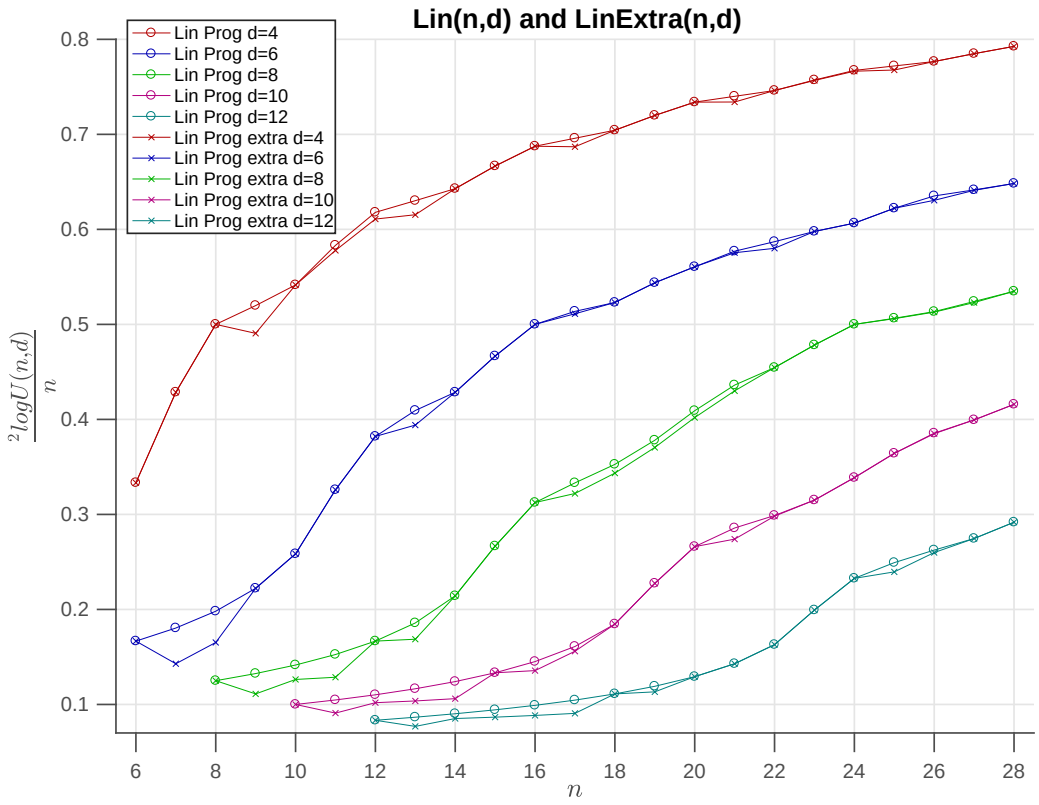


Figure 4.3: Upper bounds on A(n,d) found with the linear programming bound and the linear programming bound with extra constraints.

n	$A^*(n,4)$	$Lin(n,4)$	$LinExtraA(n,4)$
6	4	4	4
7	8	8	8
8	16	16	16
9	20	25	21
10	40	42	42
11	72	85	81
12	144	170	160
13	256	292	256
14	512	512	512
15	1.024	1.024	1.024
16	2.048	2.048	2.048
17	3.276	3.640	3.276
18	6.552	6.553	6.553
19	13.104	13.107	13.107
20	26.168	26.214	26.214
21	43.688	47.662	43.690
22	87.333	87.381	87.381
23	172.361	174.762	173.491
24	344.308	349.525	344.636
25	599.184	645.277	599.186
26	1.198.368	1.198.372	1.198.372
27	2.396.736	2.396.745	2.396.745
28	4.792.950	4.793.490	4.793.490

Table 4.3: Upper bounds on A(n,4) found with the linear programming bound and the linear programming bound with extra constraints.

4.4.1. LINEAR PROGRAMMING BOUND WITH $d = 4$

We now take a closer look at the linear programming bound for $d = 4$. In Figure 4.3 we see that $Lin(n,d)$ is relatively (compared to the other n) good for $n = 10,14,18,22,26$ and bad for $n = 12,16,20,24,28$. In this section we explain these remarkable traits.

If we denote the linear programming bound in the form of Example 14 and 3.1, we are actually rewriting the inequalities $B_k \geq 0$ into $\mathbf{Hx} \leq \mathbf{b}$:

$$\begin{array}{ll} \text{maximize} & A_d + A_{d+2} + \cdots + A_n \ (A_{n-1} \text{ for odd } n) \\ \text{such that} & \mathbf{Hx} \leq \mathbf{b}, \\ & A_i \geq \mathbf{0} \end{array}$$

The elements of H are shown below, with

$$Kraw(k,t,n) = -K_k(t) = -\sum_{j=0}^{k} (-1)^j \binom{n-t}{k-j}\binom{t}{j}, \quad k = 0,1,\ldots,n/2 \quad t = d,\ldots,n$$

For n is even

$$\mathbf{H} = \begin{pmatrix} Kraw(0,d,n) & Kraw(0,d+2,n) & \cdots & Kraw(0,n,n) \\ Kraw(1,d,n) & Kraw(1,d+2,n) & \cdots & Kraw(1,n,n) \\ \vdots & \vdots & \ddots & \vdots \\ Kraw(n/2,d,n) & Kraw(n/2,d+2,n) & \cdots & Kraw(n/2,n,n) \end{pmatrix}$$

For n is odd

$$\mathbf{H} = \begin{pmatrix} Kraw(0,d,n) & Kraw(0,d+2,n) & \cdots & Kraw(0,n-1,n) \\ Kraw(1,d,n) & Kraw(1,d+2,n) & \cdots & Kraw(1,n-1,n) \\ \vdots & \vdots & \ddots & \vdots \\ Kraw(\frac{n-1}{2},d,n) & Kraw(\frac{n-1}{2},d+2,n) & \cdots & Kraw(\frac{n-1}{2},n-1,n) \end{pmatrix}$$

From Example 14 we know that these elements are calculated with krawtchouck polynomials. Because krawtchouck polynomials have a certain symmetry for even n, the matrix $\mathbf{H}$ also contains a certain symmetry for even n. With this symmetry we can explain why $Lin(n,d)$ is relatively low for $n = 10,14,18,22,26$ and relatively high for $n = 12,16,20,24,28$ (compared to the other n). In Observation 2 we give some properties of $kraw(k,t,n)$ for $d = 4$.

Observation 2. If n is even and $Kraw(k,t,n) = -K_k(t) = -\sum_{j=0}^{k}(-1)^j\binom{n-t}{k-j}\binom{t}{j}$, with $k = 0,1,\ldots,n/2$ and $t = d,\ldots,n$, then:

1. If $n = 12,16,20,24,28$ and k is odd and $t = n/2$, then $Kraw(k,t,n) = 0$ and $Kraw(k,t-2,n) = -Kraw(k,t+2,n)$
2. If $n = 12,16,20,24,28$ and $k > 2$ is even and $t = n/2$, then if
   - $Kraw(k,t,n) > 0$ then $Kraw(k,t+2,n) = Kraw(k,t-2,n) < 0$
   - $Kraw(k,t,n) < 0$ then $Kraw(k,t+2,n) = Kraw(k,t-2,n) > 0$
3. If $n = 10,14,18,22,26$ and k is odd and $t = n/2$, then $Kraw(k,t-c,n) = -Kraw(k,t+c,n)$ for $c = 1,\ldots,n/2$
4. If $n = 10,14,18,22,26$ and k is even and $t = n/2$, then $Kraw(k,t-c,n) = Kraw(k,t+c,n)$ for $c = 1,\ldots,n/2$ and if
   - $Kraw(k,t-1,n) = Kraw(k,t+1,n) > 0$ then $Kraw(k,t+3,n) = Kraw(k,t-3,n) < 0$
   - $Kraw(k,t-1,n) = Kraw(k,t+1,n) < 0$ then $Kraw(k,t+3,n) = Kraw(k,t-3,n) > 0$

In general the absolute values of the Krawtchouck polynomials are the smallest (compared to other t) for $t = n/2$ and become larger when $|t - \frac{n}{2}|$ gets larger. In fact, this increment (of $|Kraw(k,t,n)|$) increases when $|t - \frac{n}{2}|$ gets larger. We call the column (which corresponds to A_t) from the matrix $\mathbf{H}$, with the values $kraw(k,t,n)$ as coordinates, C_t. With Observation 2 it follows that the matrix $\mathbf{H}$ is symmetric (up to a minus sign) around $C_{n/2}$. In Example 16 we give the matrix $\mathbf{H}$ for $n = 20$. Notice that the symmetry axis is located at C_{10}.

Example 16.

$$\begin{array}{c} \begin{pmatrix} C_4 & C_6 & C_8 & C_{10} & C_{12} & C_{14} & C_{16} & C_{18} & C_{20} \end{pmatrix} = \\ \begin{pmatrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -12 & -8 & -4 & 0 & 4 & 8 & 12 & 16 & 20 \\ -62 & -22 & 2 & 10 & 2 & -22 & -62 & -118 & -190 \\ -172 & -8 & 28 & 0 & -28 & 8 & 172 & 528 & 1140 \\ -237 & 83 & 19 & -45 & 19 & 83 & -237 & -1581 & -4845 \\ 16 & 160 & -80 & 0 & 80 & -160 & -16 & 3264 & 15504 \\ 664 & -8 & -104 & 120 & -104 & -8 & 664 & -4488 & -38760 \\ 1104 & -352 & 112 & 0 & -112 & 352 & -1104 & 3264 & 77520 \\ 494 & -338 & 238 & -210 & 238 & -338 & 494 & 1326 & -125970 \\ -936 & 208 & -56 & 0 & 56 & -208 & 936 & -7072 & 167960 \\ -1716 & 572 & -308 & 252 & -308 & 572 & -1716 & 9724 & -184756 \end{pmatrix} \end{array}$$

For $n = 12, 16, 20, 24, 28$, $\frac{n}{2}$ is even and $C_{n/2}$ is a part of the matrix $\mathbf{H}$. In this case, the Krawtchouck polynomials in $C_{\frac{n}{2}-2}$, $C_{\frac{n}{2}}$ and $C_{\frac{n}{2}+2}$ are (in general) relatively small compared to the other C_t. Because we are maximizing the A_t, while $\mathbf{Hx} \leq \mathbf{b}$, the $A_{\frac{n}{2}-2}$, $A_{\frac{n}{2}}$ and $A_{\frac{n}{2}+2}$ corresponding to the $C_{\frac{n}{2}-2}$, $C_{\frac{n}{2}}$ and $C_{\frac{n}{2}+2}$ can get relatively large compared to the other t. We conclude that three of all the A_t can get relatively large.

However, for $n = 10, 14, 18, 22, 26$, $\frac{n}{2}$ is odd and $C_{n/2}$ is **not** a part of the matrix $\mathbf{H}$. In this case, the Krawtchouck polynomials in $C_{\frac{n}{2}-1}$ and $C_{\frac{n}{2}+1}$ are (in general) relatively small compared to the other C_t. Because we are maximizing the A_t, while $\mathbf{Hx} \leq \mathbf{b}$, the $A_{\frac{n}{2}-1}$ and $A_{\frac{n}{2}+1}$ corresponding to the $C_{\frac{n}{2}-1}$ and $C_{\frac{n}{2}+1}$ can get relatively large compared to the other t. We conclude that only two of all the A_t can get relatively large.

Combining these two observations gives us a reason why $Lin(n,d)$ is relatively good for $n = 10, 14, 18, 22, 26$ and bad for $n = 12, 16, 20, 24, 28$, compared to the other n. We illustrate the preceding observations in Example 17.

Example 17. We give the matrix $\mathbf{H}$ and the solution $\mathbf{x}$ for $n = 20$. Notice that A_8, A_{10} and A_{12} are the three high A_t

$$\mathbf{H} = \begin{pmatrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -12 & -8 & -4 & 0 & 4 & 8 & 12 & 16 & 20 \\ -62 & -22 & 2 & 10 & 2 & -22 & -62 & -118 & -190 \\ -172 & -8 & 28 & 0 & -28 & 8 & 172 & 528 & 1140 \\ -237 & 83 & 19 & -45 & 19 & 83 & -237 & -1581 & -4845 \\ 16 & 160 & -80 & 0 & 80 & -160 & -16 & 3264 & 15504 \\ 664 & -8 & -104 & 120 & -104 & -8 & 664 & -4488 & -38760 \\ 1104 & -352 & 112 & 0 & -112 & 352 & -1104 & 3264 & 77520 \\ 494 & -338 & 238 & -210 & 238 & -338 & 494 & 1326 & -125970 \\ -936 & 208 & -56 & 0 & 56 & -208 & 936 & -7072 & 167960 \\ -1716 & 572 & -308 & 252 & -308 & 572 & -1716 & 9724 & -184756 \end{pmatrix} \quad \mathbf{x} = \begin{pmatrix} 285 \\ 1824 \\ 6498 \\ 8998 \\ 6498 \\ 1824 \\ 285 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} A_4 \\ A_6 \\ A_8 \\ A_{10} \\ A_{12} \\ A_{14} \\ A_{16} \\ A_{18} \\ A_{20} \end{pmatrix}$$

We give the matrix $\mathbf{H}$ and the solution $\mathbf{x}$ for $n = 18$. Notice that A_8 and A_{10} are the two high A_t.

$$\mathbf{H} = \begin{pmatrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -10 & -6 & -2 & 2 & 6 & 10 & 14 & 18 \\ -41 & -9 & 7 & 7 & -9 & -41 & -89 & -153 \\ -80 & 16 & 16 & -16 & -16 & 80 & 336 & 816 \\ -36 & 60 & -20 & -20 & 60 & -36 & -820 & -3060 \\ 168 & 24 & -56 & 56 & -24 & -168 & 1288 & 8568 \\ 364 & -116 & 28 & 28 & -116 & 364 & -1092 & -18564 \\ 208 & -144 & 112 & -112 & 144 & -208 & -208 & 31824 \\ -286 & 66 & -14 & -14 & 66 & -286 & 2002 & -43758 \\ -572 & 220 & -140 & 140 & -220 & 572 & -2860 & 48620 \end{pmatrix} \quad \mathbf{x} = \begin{pmatrix} 180 \\ 874 \\ 2257 \\ 2131 \\ 958 \\ 144 \\ 9 \\ 0 \end{pmatrix} \begin{pmatrix} A_4 \\ A_6 \\ A_8 \\ A_{10} \\ A_{12} \\ A_{14} \\ A_{16} \\ A_{18} \end{pmatrix}$$

4.4.2. LINEAR PROGRAMMING BOUND WITH EXTRA CONSTRAINTS WITH $d = 4$

For the linear programming bound with extra constraints it is remarkable that it is relatively low for $n = 9, 13, 17, 21, 25$ compared to the other n. Before we can explain where this property comes from, we need some characteristics for the equations in the linear programming bound. If we denote the linear programming bound in the form of Example 14 and Section 3.1, we are actually rewriting the inequalities $B_k \geq 0$ into $\mathbf{Hx} \leq \mathbf{b}$, $\mathbf{b} = (b_0, b_1, b_2, \ldots, b_{\frac{n-1}{2}})$.

$$\begin{array}{ll} \text{maximize} & A_d + A_{d+2} + \cdots + A_{n-1} \\ \text{such that} & \mathbf{Hx} \leq \mathbf{b}, \\ & A_i \geq \mathbf{0} \end{array}$$

When we fill in the optimal solution $\mathbf{x} = (A_d, A_{d+2}, \ldots, A_{n-1})$ in the B_k, we find that $B_k = 0$ for all $k = 1, \ldots, \lfloor n/2 - 2 \rfloor$. This is the same as multiplying $\mathbf{H}$ with the optimal solution $\mathbf{x}$, we denote this as $\mathbf{Hx} = \mathbf{s} = (s_0, s_1, s_2, \ldots, s_{\frac{n-1}{2}})$, then we again find that $\mathbf{s}_k = \mathbf{b}_k$ for all $k = 1, \ldots, \lfloor n/2 - 2 \rfloor$. We illustrate this in Example 18.

Example 18. For $n = 13$ (and $d = 4$) the matrix $\mathbf{H}$ and vector $\mathbf{b}$ become:

$$\mathbf{H} = \begin{pmatrix} -1 & -1 & -1 & -1 & -1 \\ -5 & -1 & 3 & 7 & 11 \\ -6 & 6 & 2 & -18 & -54 \\ 10 & 6 & -14 & 14 & 154 \\ 29 & -15 & 5 & 25 & -275 \\ 9 & -15 & 25 & -63 & 297 \\ -36 & 20 & -20 & 36 & -132 \end{pmatrix} \quad \mathbf{b} = \begin{pmatrix} 1 \\ 13 \\ 78 \\ 286 \\ 715 \\ 1287 \\ 1716 \end{pmatrix}$$

The optimal solution $\mathbf{x}$ now becomes: $\begin{pmatrix} 65,00 \\ 104,00 \\ 105,86 \\ 14,86 \\ 1,86 \end{pmatrix}$ When we multiply $\mathbf{H}$ with $\mathbf{x}$ we find:

$$\mathbf{Hx} = \begin{pmatrix} -1 & -1 & -1 & -1 & -1 \\ -5 & -1 & 3 & 7 & 11 \\ -6 & 6 & 2 & -18 & -54 \\ 10 & 6 & -14 & 14 & 154 \\ 29 & -15 & 5 & 25 & -275 \\ 9 & -15 & 25 & -63 & 297 \\ -36 & 20 & -20 & 36 & -132 \end{pmatrix} \begin{pmatrix} 65,00 \\ 104,00 \\ 105,86 \\ 14,86 \\ 1,86 \end{pmatrix} = \begin{pmatrix} -255 \\ 13 \\ 78 \\ 286 \\ 715 \\ 519 \\ -1356 \end{pmatrix} = \mathbf{s} \quad \text{and} \quad \mathbf{b} = \begin{pmatrix} 1 \\ 13 \\ 78 \\ 286 \\ 715 \\ 1287 \\ 1716 \end{pmatrix}$$

Now we see that $\mathbf{s}_k = \mathbf{b}_k$ for all $k = 1, \ldots, 4$

When we add the extra constraints to the linear programming bound, the original A_i, found without the extra constraints, don't necessarily comply to the new constraints. For $n = 9, 13, 17, 21, 25$ it is noticeable that with the linear programming bound we find $A_{n-1} > 1$. However the extra constraints restrict $A_{n-1} \leq A(n, 4, n-1) = 1$. Since we know that $\mathbf{s}_k = \mathbf{b}_k$ for all $k = 1, \ldots, \lfloor n/2 - 2 \rfloor$ (see Example 18), adjusting A_{n-1} to the new constraint and setting $A_{n-1} \leq 1$, will have influence on the other A_i. Because the $Kraw(k, t, n)$ ($k = 0, \ldots, \frac{n-1}{2}$ and $t = d, \ldots, n-1$) are the largest for $t = n-1$, changing A_{n-1} has big influence on the other A_i. When $Kraw(k, n-1, n)$ is negative, setting $A_{n-1} \leq 1$ means other $A_i * Kraw(k, i, n)$ have to lower as well to comply to the $\mathbf{Hx} \leq \mathbf{b}$ inequalities. This then leads to a lower bound for $n = 9, 13, 17, 21, 25$. In Example 19 we illustrate this for $n = 13$.

Example 19. For $n = 13$ the matrix $\mathbf{H}$ and the solution to the linear program without extra constraints is:

$$\mathbf{Hx} = \begin{pmatrix} -1 & -1 & -1 & -1 & -1 \\ -5 & -1 & 3 & 7 & 11 \\ -6 & 6 & 2 & -18 & -54 \\ 10 & 6 & -14 & 14 & 154 \\ 29 & -15 & 5 & 25 & -275 \\ 9 & -15 & 25 & -63 & 297 \\ -36 & 20 & -20 & 36 & -132 \end{pmatrix} \begin{pmatrix} 65,00 \\ 104,00 \\ 105,86 \\ 14,86 \\ 1,86 \end{pmatrix} = \begin{pmatrix} -255 \\ 13 \\ 78 \\ 286 \\ 715 \\ 519 \\ -1356 \end{pmatrix} = \mathbf{s} \quad \text{and} \quad \mathbf{b} = \begin{pmatrix} 1 \\ 13 \\ 78 \\ 286 \\ 715 \\ 1287 \\ 1716 \end{pmatrix}$$

And the matrix **H** and **b** for the linear program with extra constraints are:

$$\mathbf{Hx} = \begin{pmatrix} -1 & -1 & -1 & -1 & -1 \\ -5 & -1 & 3 & 7 & 11 \\ -6 & 6 & 2 & -18 & -54 \\ 10 & 6 & -14 & 14 & 154 \\ 29 & -15 & 5 & 25 & -275 \\ 9 & -15 & 25 & -63 & 297 \\ -36 & 20 & -20 & 36 & -132 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} A_4 \\ A_6 \\ A_8 \\ A_{10} \\ A_{12} \end{pmatrix} \quad \text{and} \quad \mathbf{b} = \begin{pmatrix} 1 \\ 13 \\ 78 \\ 286 \\ 715 \\ 1287 \\ 1716 \\ 65 \\ 182 \\ 132 \\ 26 \\ 1 \end{pmatrix}$$

This gives us that $1 * A_{12} \leq 1$. However, with the linear programming bound without extra constraint we found that $A_{12} = 1,86$. By adjusting $A_{12} \leq 1$ and complying to the other original and extra constraints of the linear programming bound we find the optimal solution: $\mathbf{x} = \begin{pmatrix} 55 \\ 96 \\ 87 \\ 16 \\ 1 \end{pmatrix}$

For other n it is noticeable that the A_{n-1} (or A_n for even n) already comply to the extra constraint of $A_{n-1} \leq A(n,4,n-1) = 1$ (or for even n: $A_n \leq A(n,4,n) = 1$). Some of the other A_i might not comply to the other extra constraints, because the corresponding $Kraw(k,t,n)$ are significantly smaller for these A_i, the adjusting of these A_i has significantly less influence on lowering the bound.

From this section we conclude that compared to the linear programming bound without extra constraints, the linear programming bound with extra constraints improves noticeably for $n = 9,13,17,21,25$, and only improves slightly for all other n.

4.5. JOHNSON AND LINEAR PROGRAMMING BOUND WITH EXTRA CONSTRAINTS

In the above sections the Linear programming bound with extra constraints and the Johnson bound have proven to give the lowest upper bounds on $A(n,d)$. Therefore we compare these two bounds to each other (see Figure 4.4). The linear programming bound with extra constraints presents the lowest upper boud for every $n = 6,7,\ldots,28$ and $d = 4,6,\ldots,12$, except for $n = 24$ and $d = 4$, for these n and d the Jonhson bound gives a lower upper bound (see Table 4.4). When a result is denoted in blue it is the lowest upper bound closest to $A^*(n,4)$.

For the specific case $n = 24$ and $d = 4$ In Section 4.4.1 we found that the linear programming bound is relatively high for $n = 8,12,16,20,24,28$. In Section 4.4.2 we found that the linear programming bound with extra constraints improves the linear programming upper bound by a lot for $n = 9,13,17,21,25$. But it only improves slightly for all the other n. Combining these two properties leads to a relatively high upper bound for $n = 8,12,16,20,24,28$. Remarkably the Johnson bound is always at least as good as the linear programming bound with extra constraints for $n = 8,12,16,20,24,28$. To explain this we take a closer look at the Jonhson bound for $d = 4$. We illustrate the increase of the Johnson bound in Table 4.5 by deviding $John(n,d)$ by $John(n-1,d)$. For odd n the increase in $John(n,4)$ is relatively low, which tells us that the johnson bound is relatively low for odd n. For even n $\frac{John(n,4)}{John(n-1,4)} = 2$ except for $n = 12,18,24$. For these n the Johnson bound increases less than we would expect it to do. So for $n = 12,18,24$ the Johnson bound is relatively better than for other even n. Combining the relatively higher $LinExtra(n,4)$ for $n = 8,12,16,20,24,28$ and the relatively low $John(n,4)$ for $n = 12,18,24$, gives us an explenation why $Jon(24,4) \leq LinExtra(24,4)$. However by this reasoning $Jon(12,4)$ should also be lower than $LinExtra(12,4)$. When we take a closer look at $LinExtra(12,4)$ and $Jon(12,4)$, we find that the exact values are : $LinExtra(12,4) = 160,91$ and $Jon(12,4) = 160$. However, $160,91$ is rounded down to 160, which leads to the equal results.

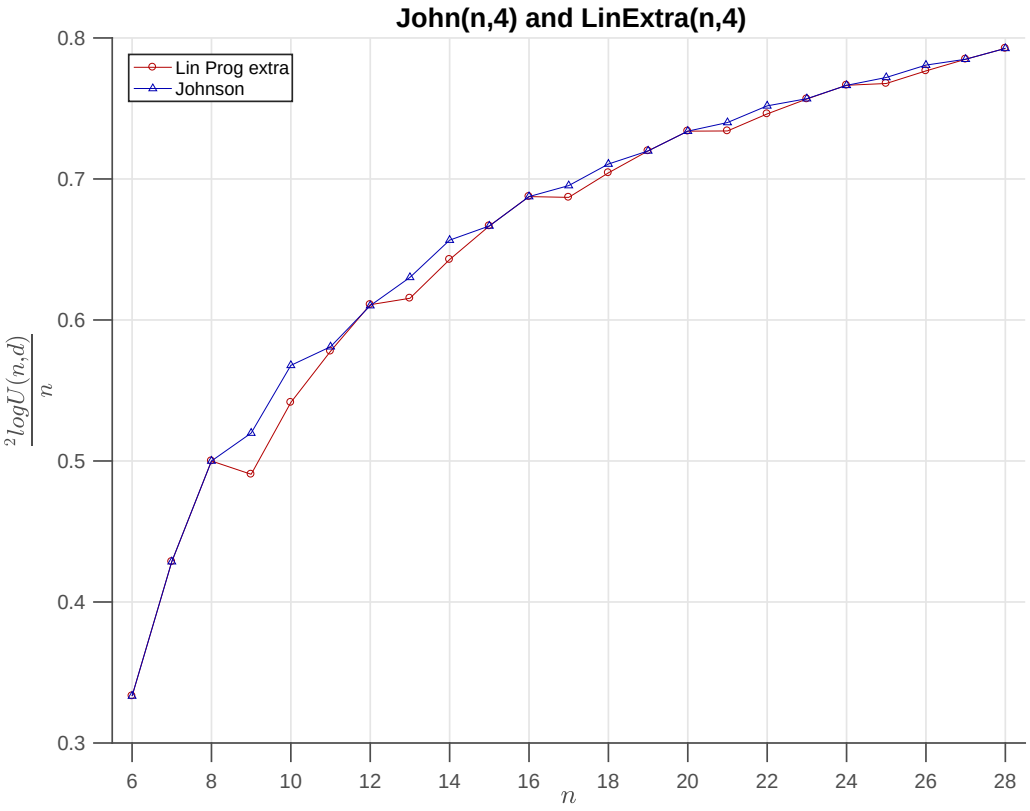


Figure 4.4: Upper bounds on A(n,4) found with the Johnson bound and the linear programming bound with extra constraints.

n	$A^*(n,4)$	$LinExtra(n,4)$	$John(n,4)$
6	4	4	4
7	8	8	8
8	16	16	16
9	20	21	25
10	40	42	51
11	72	81	83
12	144	160	160
13	256	256	292
14	512	512	585
15	1.024	1.024	1.024
16	2.048	2.048	2.048
17	3.276	3.277	3.615
18	6.552	6.553	7.084
19	13.104	13.107	13.107
20	26.168	26.214	26.214
21	43.688	43.690	47.662
22	87.333	87.381	95.325
23	172.361	173.491	174.103
24	344.308	344.636	344.308
25	599.184	599.186	645.277
26	1.198.368	1.198.372	1.290.555
27	2.396.736	2.396.745	2.396.745
28	4.792.950	4.793.490	4.793.490

Table 4.4: Upper bounds on A(n,4) found with the Johnson bound and the linear programming bound with extra constraints.

n	$\frac{John(n,4)}{John(n-1,4)}$
6	-
7	2
8	2
9	1,6
10	2
11	1,64
12	1,91
13	1,83
14	2
15	1,75
16	2
17	1,77
18	1,96
19	1,85
20	2
21	1,82
22	2
23	1,83
24	1,98
25	1,87
26	2
27	1,86
28	2

Table 4.5: Increase of the Johnson bound denoted by $\frac{John(n,4)}{John(n-1,4)}$.

5

CONCLUSION AND RECOMMENDATIONS

5.1. CONCLUSION

First of all, the conclusions presented here are only valid for the n $(= 6,7,\ldots,28)$ and d $(= 4,6,\ldots,12)$ used in this report. Overall, the Plotkin bound proves to give better results than the Singleton bound, however it only gives results for certain combinations of n and d. When the Plotkin bound gives results it is always equal to $A^*(n,d)$, the best known value for the upper bound. The Singleton bound only gives a crude upper bound on $A(n,d)$, nevertheless it is straightforward to calculate.

The Johnson bound gives a better or equal upper bound compared to the Hamming bound, because it is an improvement of the latter. The Hamming bound is straightforward to calculate and already gives significantly improved results compared to the Singleton bound. The Johnson bound depends on the values for $A(n,d,w)$, the implementation of these values in Matlab is time consuming. The results of the Johnson bound are reasonably close to the best known upper bound.

The linear programming bound with extra constraints always gives a better or equal upper bound compared to the linear programming bound without extra constraints, simply because one adds more linear constraints to a possible solution. The linear programming bounds involve complex calculations and in addition the implementation in Matlab of the extra constraints is time consuming, because it again requires the values of $A(n,d,w)$.

The linear programming bound with extra constraints is the best bound for $n = 6,7,\ldots,28$ and $d = 4,6,\ldots,12$, except for $A(24,4)$ and $A(12,4)$, in those cases the Johnson bound gives the best results. At a first glance, this does not show in the results for the upper bound for $n = 12$ and $d = 4$, which is caused by the integer rounding of the result of the linear programming bound with extra constraints. Due to this integer rounding the presented results of the upper bound of these two methods is in this case the same. For the case of $n = 24$ and $d = 4$ the linear programming bound gives an upper bound of 344.636 and the Johnson bound of 344.308. To explain why the Johnson bound is better for $n = 12,24$ and $d = 4$ we have to look at both the Johnson and linear programming bound with extra constraints. For $n = 8,12,16,20,24,28$ the linear programming bound gives a relatively high value for the upper bound and the extra constraints do not improve the results by much. Specifically for $n = 12$ and $n = 24$, the results of the Johnson bound are relatively better than for other even n. The combination of the relative high value of the linear programming bound, for $n = 8,12,16,20,24,28$, and the relative lower value of the Johnson bound, for $n = 12$ and $n = 24$, causes the Johson bound to give a better upper bound than the linear programming bound for $n = 12$ and $n = 24$.

The main conclusion from these comparisons is that in general a method provides a better upper bound when it requires more input and involves more complex calculations. This input can for example consist of Krawthcouck polynomials or upper bounds on the number of codewords with length n, Hamming distance d and weight w.

5.2. RECOMMENDATIONS

In this report we have two recommendations. First, one could add additional constraints to the linear programming bound with extra constraints in order to improve the upper bound for some $A(n,d)$. The combining of $A_i(\mathbf{u}) \leq A(n,d,i)$ could lead to a new linear constraint that is more stringent on the A_i. For example, for $A(13,6)$ we know that: $A_{10}(\mathbf{u}) \leq A(13,6,10) = 4$ and $A_{12}(\mathbf{u}) \leq A(13,6,12) = 1$. If $A_{12}(\mathbf{u}) = 1$ then $A_{10}(\mathbf{u}) = 0$, averaging over $\mathbf{u}$ gives us: $A_{10} + 4A_{12} \leq 4$. In this case the upper bound is improved from 40 (for the linear programming bound with extra constraints) to 32, as can be seen in the article by Best et al. [9]. We expect this could improve the upper bound in more cases and therefore recommend investigating it. Second, there are more methods to calculate upper bounds than those six compared in this report. An example of such a method is semidefinite programming, see [14]. One could perform a similar comparison for other methods of calculating upper bounds on $A(n,d)$.

BIBLIOGRAPHY

[1] R. W. Hamming, *Error detecting and error correcting codes*, The Bell System Technical Journal, Vol 29, Page 147, 1950.

[2] J. H. Weber, *Error-correcting codes*, TU Delft lecture notes, course: et4-030, 2013.

[3] A. E. Brouwer, *Table of general binary codes*, 2016, https://www.win.tue.nl/~aeb/codes/binary-1.html

[4] M. Plotkin, *Binary codes with specified minimum distance*, IEEE Transactions on Information Theory, Vol 6, Page 445, 1960.

[5] F. J. Macwilliams and N. J. A Sloane, *The theory of error-correcting codes*, North-Holland, 2000.

[6] D. R. Hankerson, D. G Hoffman, D. A. Leonard, C. C. Lindner, K. T. Phelps, C. A Rodger and J. R. Wall, *Coding theory and cryptography, the essentials*, Marcel Dekker, 2000.

[7] E. Agrell, *Erik Agrell's tables of binary block codes*, 2015, http://codes.se/bounds/

[8] E. Agrell, A. Vardy and K. Zeger, *A table of upper bounds for binary codes*, IEEE Transactions on Information Theory, Vol 47, Page 3004, 2001.

[9] 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 Transactions on Information Theory, Vol 24, Page 81, 1978.

[10] P. R. Östergard, T. Baicheva and E. Kolev, *Optimal binary one-error-correcting codes of length 10 have 72 codewords* IEEE Transactions on Information Theory, Vol 45, Page 1229, 1999.

[11] B. Mounits, T. Etzion and S. Litsyn, *Improved upper bounds on sizes of codes*, IEEE Transactions on Information Theory, Vol 48, Page 880, 2002.

[12] A. Schrijver, *New code upper bounds from the Terwilliger algebra and semidefinite programming*, IEEE Transactions on Information Theory, Vol 51, Page 2859, 2005.

[13] P. R. J. Östergard, *On the size of optimal three-error-correcting binary codes of length 16*, IEEE Transactions on Information Theory, Vol 57, Page 6824, 2011.

[14] D. C. Gijswijt, H. D. Mittelmann and A. Schrijver, *Semidefinite Code Bounds Based on Quadruple Distances*, IEEE Transactions on Information Theory, Vol 58, Page 2697, 2012.

[15] Hyun Kwang Kim and Phan Thanh Toan, *Improved Semidefinite Programming Bound on Sizes of Codes*, IEEE Transactions on Information Theory, Vol 59, Page 7337, 2013.

[16] R. Singleton, *Maximum distance q-ary codes*, IEEE Transactions on Information Theory, Vol 10, Page 116, 1964.

[17] V. Guruswami, *Introduction to coding theory: Elementary bounds on codes*, 2010, https://errorcorrectingcodes.wordpress.com/2010/01/30/notes-4-elementary-bounds-on-codes/

[18] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: Algorithms and complexity*, Dover publications, 1998.

[19] J. A. M. de Groot, *TU Delft lectue notes, course: " Toegepaste algebra: Codes en Cryptosystemen*, 2017.

A

MATLAB CODES

A.1. MATLAB CODES FOR PLOTKIN, SINGLETON, HAMMING AND JOHNSON BOUNDS

The Plotkin bound:

```
function [s]=plo(n,d)
b=mod(d,2);
    if 2*d>n
        s=2*floor((d/(2*d-n)));
    elseif 2*d==n
        s=4*d;
    else
        s=inf;
    end
s;
end
```

The Singleton bound:

```
function [s]=sing(n,d)
s=2^(n-d+1);
end
```

The Hamming bound:

```
function [s]=Ham(n,d,j)
b=mod(d,2);
if (b==0)
    s=Ham(n-1,d-1,j);
else
    t=(d-1)/2;
    s=(2^n)/symsum((nchoosek(n,j)),j,0,t);
    s=double(s);
end
s;
```

The Johnson bound:

```
function [s]=jon2(n,d,j)
b=mod(d,2);

if (b==0)
```

```
    s=jon2(n-1,d-1,j);
else
    t=(d-1)/2;
    k=d+1;
    r=(k-2)/2;
    g=nchoosek(n,j);
    %Z is the matrix containing A(n,d,w)
    s=(2^n)/((symsum(g,j,0,t)+((nchoosek(n,t+1)-(nchoosek(d,t)*Z(n-3,d+1,r)))/Z(n-3,
        t+2,r))));
end
s=double(s);
end
```

A.2. MATLAB CODES FOR THE LINEAR PROGRAMMING BOUND

Adding 1 (the zero vector) to the linear program bound.

```
function[s]=MaxA(n,d,j)
s=lp(n,d,j)+1;
end
```

Calling all the needed functions for linprog and calculating the optimal solution with linprog.

```
function[s]=lp(n,d,j)
f=ones(1,aantalf(n,d))*-1;
A=mat(n,d,j);
b=vec(n,d);
Aeq=[];
beq=[];
lb=zeros(aantalf(n,d),1);
ub=ones(aantalf(n,d),1)*inf;
options = optimoptions('linprog','Display','off','Algorithm','dual-simplex');
[x,fval]=linprog(f,A,b,Aeq,beq,lb,ub,options);
s=abs(fval);
end
```

The number of A_i in the objective function.

```
function[s]=aantalf(n,d)
b=mod(n,2);
c=mod(d,2);
if(b==0)
    if(c==0)
       aantal=(n-d)/2+1;
    else
        aantal=(n+1-d)/2;
    end
else
    if(c==0)
        aantal=(n+1-d)/2;
    else
        aantal=(n-d)/2;
    end
end
s=aantal;
end
```

Calculating b for the linear program $Ax \leq b$.

```
function[b]=vec(n,d)
```

```
b=mod(n,2);
if b==0
    m=n/2;
else
    m=(n+1)/2-1;
end
for i=0:m
    b(i+1,1)=nchoosek(n,i);
end
b=double(b);
end
```

Calculating A for the linear program $Ax \leq b$.

```
function[X]=mat(n,d,j)
b=mod(n,2);
if b==0
    m=n/2;
else
    m=(n+1)/2-1;
end

for q=0:m
    for i=0:+2:(aantalf(n,d)*2-1)
        B(q+1,i/2+1)=kraw(q,i+d,n,j);
    end
end
B=double(B);
A=-1.*B;
X=A;
end
```

Calculating the Krawtchouck polynomials for A

```
function[s]=kraw(k,t,n,j)
s=symsum(((-1)^j)*nchoosek(n-t,k-j)*nchoosek(t,j),j,0,k);
end
```

A.3. MATLAB CODES FOR THE LINEAR PROGRAMMING BOUND WITH EXTRA CONSTRAINTS

Adding 1 to the linear program solution for the zero vector

```
function[s]=MaxA2(n,d,j)
s=lp2(n,d,j)+1;
end
```

Calling all the needed functions for linprog and calculating the optimal solution with linprog.

```
function[s]=lp2(n,d,j)
f=ones(1,aantalf(n,d))*-1;
A=mat2(n,d,j);
b=vec2(n,d);
Aeq=[];
beq=[];
lb=zeros(aantalf(n,d),1);
ub=ones(aantalf(n,d),1)*inf;
options = optimoptions('linprog','Display','off','Algorithm','dual-simplex');
[x,fval]=linprog(f,A,b,Aeq,beq,lb,ub,options);
```

```
s=abs(fval);
end
```

Calculating b from the linear program $Ax \leq b$ with extra constraints

```
function[F]=vec2(n,d)
b=mod(d,2);
if (b==0)
    k=d;
else
    k=d+1;
end
r=(k-2)/2;

for v=k:2:n
    if v<=14
        x((v-k)/2+1,1)=Z(n-3,v+1,r);
    else
        x((v-k)/2+1,1)=Z(n-3,n-v+1,r);
    end
end
x;
F=cat(1,vec(n,d),x);
end
```

Calculating A from the linear program $Ax \leq b$ with extra constraints

```
function[E]=mat2(n,d,j)
[m,k]=size(mat(n,d,j));
for i=1:k
    v(i)=1;
end
v;
D=diag(v);
E=cat(1,mat(n,d,j),D);
end
```

A.4. MATLAB CODES FOR COMPUTING THE RESULTS OF THE BOUNDS.

The values used for $A(n,d,w)$ are given in matrix Z. After this we calculate the values used for the graphical representations of the results.

```
syms j;
A=[1 1 2 1 1 0 0 0 0 0 0 0 0 0 0 ; 1 1 2 2 1 1 0 0 0 0 0 0 0 0 0 ; 1 1 3 4 3 1 1 0 0
    0 0 0 0 0 0 ; 1 1 3 7 7 3 1 1 0 0 0 0 0 0 0 ; 1 1 4 8 14 8 4 1 1 0 0 0 0 0 0 ; 1
    1 4 12 18 18 12 4 1 1 0 0 0 0 0 ; 1 1 5 13 30 36 30 13 5 1 1 0 0 0 0 ; 1 1 5 17
    35 66 66 35 17 5 1 1 0 0 0 ; 1 1 6 20 51 84 132 84 51 20 6 1 1 0 0; 1 1 6 26 65
   132 182 182 132 65 26 6 1 1 0; 1 1 7 28 91 182 308 364 308 182 91 28 7 1 1 ; 1 1
   7 35 105 271 455 660 660 455 271 105 35 7 1 ; 1 1 8 37 140 336 722 1040 1320 1040
    722 336 140 37 8 ; 1 1 8 44 157 476 952 1753 2210 2210 1753 952 476 157 44 ; 1 1
    9 48 198 565 1428 2448 3944 4420 3944 2448 1428 565 198 ; 1 1 9 57 228 752 1789
   3876 5814 8326 8326 5814 3876 1789 752 ; 1 1 10 60 285 912 2506 5111 9690 12920
   16652 12920 9690 5111 2506 ; 1 1 10 70 315 1197 3192 7518 13416 22610 27132 27132
    22610 13416 7518; 1 1 11 73 385 1386 4389 10032 20674 32794 49742 54264 49742
   32794 20674 ; 1 1 11 83 419 1771 5313 14421 28842 52833 75426 104006 103539 75426
    52833; 1 1 12 88 498 2011 7084 18216 43263 76912 126799 164565 208012 164565
   126799 ; 1 1 12 100 550 2490 8379 25300 56925 120175 192280 288197 342843 342843
   288197 ; 1 1 13 104 650 2860 10790 31122 82225 164450 312455 454480 624387 685686
    624387 ; 1 1 13 117 702 3510 12870 41618 105036 246675 444015 766935 1022580
```

```
    1296803 1296803 ; 1 1 14 121 819 3931 16380 51480 145663 326778 690690 1130220
    1789515 2202480 2593606 ];
B=[1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 ; 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 ; 1 1 1 2 1 1 1 0 0
    0 0 0 0 0 0 ; 1 1 1 2 2 1 1 1 0 0 0 0 0 0 0 ; 1 1 1 2 2 2 1 1 1 0 0 0 0 0 0 ; 1
    1 1 3 3 3 3 1 1 1 1 0 0 0 0 ; 1 1 1 3 5 6 5 3 1 1 1 0 0 0 0 ; 1 1 1 3 6 11 11 6 3
    1 1 1 0 0 0 ; 1 1 1 4 9 12 22 12 9 4 1 1 1 0 0 ; 1 1 1 4 13 18 26 26 18 13 4 1 1
    1 0 ; 1 1 1 4 14 28 42 42 42 28 14 4 1 1 1 ; 1 1 1 5 15 42 70 78 78 70 42 15 5 1
    1 ; 1 1 1 5 20 48 112 138 150 138 112 48 20 5 1 ; 1 1 1 5 20 68 136 228 280 280
    228 136 68 20 5 ; 1 1 1 6 22 72 199 349 428 425 428 349 199 72 22 ; 1 1 1 6 25 83
    228 520 718 789 789 718 520 228 83 ; 1 1 1 6 30 100 276 651 1107 1363 1403 1363
    1107 651 276 ; 1 1 1 7 31 126 350 828 1695 2359 2685 2685 2359 1695 828 ; 1 1 1 7
    37 136 462 1100 2277 3766 4415 5064 4415 3766 2277 ; 1 1 1 7 40 170 521 1518
    3162 5819 7521 7953 7953 7521 5819 ; 1 1 1 8 42 192 680 1786 4554 8432 12186
    14682 15906 14682 12186 ; 1 1 1 8 50 210 800 2428 5581 12620 19037 24630 30587
    30587 24630 ; 1 1 1 8 52 260 910 2971 7891 16122 28893 42080 50169 61174 50169 ;
    1 1 1 9 54 280 1170 3510 10027 23673 43529 66079 84574 91080 91080 ; 1 1 1 9 63
    302 1306 4680 12285 31195 63756 104231 142117 164220 169740];
C=[1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 ; 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 ; 1 1 1 1 1 1 1 0 0
    0 0 0 0 0 0 ; 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 ; 1 1 1 1 2 1 1 1 1 0 0 0 0 0 0 ; 1
    1 1 2 2 1 1 1 1 0 0 0 0 0 ; 1 1 1 1 2 2 2 1 1 1 1 0 0 0 0 ; 1 1 1 1 2 2 2 2 1 1
    1 1 0 0 0 ; 1 1 1 1 3 3 4 3 3 1 1 1 1 0 0 ; 1 1 1 1 3 3 4 4 3 3 1 1 1 1 0 ; 1 1
    1 1 3 4 7 8 7 4 3 1 1 1 1 ; 1 1 1 1 3 6 10 15 15 10 6 3 1 1 1 ; 1 1 1 1 4 6 16 16
    30 16 16 6 4 1 1 ; 1 1 1 1 4 7 17 24 34 34 2417 7 4 1 1 ; 1 1 1 1 4 9 21 33 49
    58 49 33 21 9 4 ; 1 1 1 1 4 12 28 52 78 103 103 78 52 28 12 ; 1 1 1 1 5 16 40 80
    130 173 206 173 130 80 40 ; 1 1 1 1 5 21 56 120 210 302 363 363 302 210 120 ; 1 1
    1 1 5 21 77 176 330 473 634 680 634 473 330 ;1 1 1 1 5 23 80 253 506 707 1025
    1288 1288 1025 707 ; 1 1 1 1 6 24 92 274 759 1041 1551 2142 2576 2142 1551 ; 1 1
    1 1 6 30 100 328 856 1486 2333 3422 4087 4087 3422 ; 1 1 1 1 6 30 130 371 1066
    2108 3496 5225 6741 7080 6741 ; 1 1 1 1 6 32 135 500 1252 2914 4986 7833 10547
    11981 11981 ; 1 1 1 1 7 33 149 540 1750 3895 7016 11939 17011 21152 22710];
D=[1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 ; 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 ; 1 1 1 1 1 1 1 0 0
    0 0 0 0 0 0 ; 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 ; 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 ; 1
    1 1 1 1 1 1 1 1 1 0 0 0 0 0 ; 1 1 1 1 1 2 1 1 1 1 1 0 0 0 0 ; 1 1 1 1 1 2 2 1 1 1
    1 1 0 0 0 ; 1 1 1 1 1 2 2 2 1 1 1 1 1 0 0 ; 1 1 1 1 1 2 2 2 2 1 1 1 1 1 0 ; 1 1
    1 1 1 2 2 2 2 2 1 1 1 1 1 ; 1 1 1 1 1 3 3 3 3 3 3 1 1 1 1 ; 1 1 1 1 1 3 3 4 4 4 3
    3 1 1 1 ; 1 1 1 1 1 3 3 5 6 6 5 3 3 1 1 ; 1 1 1 1 1 3 4 6 9 10 9 6 4 3 1 ; 1 1 1
    1 1 3 4 8 12 19 19 12 8 4 3 ; 1 1 1 1 1 4 5 10 17 20 38 20 17 10 5 ; 1 1 1 1 1 4
    7 13 21 35 42 42 35 21 13 ; 1 1 1 1 1 4 7 16 33 51 72 80 72 51 33 ; 1 1 1 1 1 4
    8 20 46 81 117 135 135 117 81 ; 1 1 1 1 1 4 9 24 60 118 171 223 247 223 171 ; 1 1
    1 1 1 5 10 32 75 158 262 380 434 434 380 ; 1 1 1 1 1 5 13 36 104 214 406 566 702
    754 702 ; 1 1 1 1 1 5 14 48 121 299 571 882 1201 1419 1419 ; 1 1 1 1 1 5 16 56
    168 376 821 1356 1977 2438 2629];
E=[1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 ; 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 ; 1 1 1 1 1 1 1 0 0
    0 0 0 0 0 0 ; 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 ; 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 ; 1
    1 1 1 1 1 1 1 1 1 0 0 0 0 0 ; 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 ; 1 1 1 1 1 1 1 1 1 1
    1 1 0 0 0 ; 1 1 1 1 1 1 1 2 1 1 1 1 1 1 0 0 ; 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 0 ; 1 1
    1 1 1 1 1 2 2 2 1 1 1 1 1 1 ; 1 1 1 1 1 1 1 2 2 2 2 1 1 1 1 1 ; 1 1 1 1 1 1 1 2 2 2 2 2
    1 1 1 1 ; 1 1 1 1 1 1 1 2 2 2 2 2 2 1 1 1 ; 1 1 1 1 1 1 1 3 3 3 4 3 3 3 1 1 ; 1 1 1
    1 1 1 1 3 3 3 4 4 3 3 3 1 ; 1 1 1 1 1 1 1 3 3 5 5 6 5 5 3 3 ; 1 1 1 1 1 1 1 3 3 5 7 7 7
    7 5 3 ; 1 1 1 1 1 1 1 3 4 6 8 11 12 11 8 6 ; 1 1 1 1 1 1 1 3 4 6 10 16 23 23 16 10 ;
    1 1 1 1 1 1 1 4 4 9 16 24 24 46 24 24 ; 1 1 1 1 1 1 1 4 5 10 25 37 42 50 50 42 ; 1 1
    1 1 1 1 1 4 5 13 26 48 66 83 91 83 ; 1 1 1 1 1 1 1 4 6 15 39 64 100 140 156 156 ; 1
    1 1 1 1 1 4 8 19 45 87 149 199 245 265];
Z=cat(3,A,B,C,D,E);

x=[6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28];
```

```
echte4=[4 8 16 20 40 72 144 256 512 1024 2048 3276 6552 13104 26168 43688 87333
    172361 344308 599184 1198368 2396736 4792950];
echte6=[2 2 2 4 6 12 24 32 64 128 256 340 673 1237 2279 4096 6941 13674 24106 47538
    84260 157285 291269];
echte8=[1 1 2 2 2 2 4 4 8 16 32 36 71 131 256 512 1024 2048 4096 5421 9275 17099
    32151];
echte10=[1 1 1 1 2 2 2 2 2 4 4 6 10 20 40 47 84 150 268 466 836 1585 2817];
echte12=[1 1 1 1 1 1 2 2 2 2 2 2 4 4 6 8 12 24 48 55 96 169 288];

for i=1:23
      echt4(i)=(log2(echte4(i)))/(i+5);
      k1(i)=log2(maxA(i+5,4,j))/(i+5);
      k2(i)=log2(MaxA2(i+5,4,j))/(i+5);
      k3(i)=log2(plo(5+i,4))/(i+5);
      k4(i)=log2(sing(5+i,4))/(i+5);
      k5(i)=log2(double((Ham(i+5,4,j))))/(i+5);
      k6(i)=log2(double((jon2(i+5,4,j))))/(i+5);
 end

for i=1:23
    echt6(i)=(log2(echte6(i)))/(i+5);
    m1(i)=log2(maxA(i+5,6,j))/(i+5);
    m2(i)=log2(MaxA2(i+5,6,j))/(i+5);
    m3(i)=log2(plo(5+i,6))/(i+5);
    m4(i)=log2(sing(5+i,6))/(i+5);
    m5(i)=log2(double((Ham(i+5,6,j))))/(i+5);
    m6(i)=log2(double((jon2(i+5,6,j))))/(i+5);
end

 for i=3:23
      echt8(i)=(log2(echte8(i)))/(i+5);
      p1(i)=log2(maxA(i+5,8,j))/(i+5);
      p2(i)=log2(MaxA2(i+5,8,j))/(i+5);
      p3(i)=log2(plo(5+i,8))/(i+5);
      p4(i)=log2(sing(5+i,8))/(i+5);
      p5(i)=log2(double((Ham(i+5,8,j))))/(i+5);
      p6(i)=log2(double((jon2(i+5,8,j))))/(i+5);
 end
 echt8(1:2)=inf;
 p1(1:2)=inf;
 p2(1:2)=inf;
 p3(1:2)=inf;
 p4(1:2)=inf;
 p5(1:2)=inf;
 p6(1:2)=inf;

 for i=5:23
      echt10(i)=(log2(echte10(i)))/(i+5);
      r1(i)=log2(maxA(i+5,10,j))/(i+5);
      r2(i)=log2(MaxA2(i+5,10,j))/(i+5);
      r3(i)=log2(plo(5+i,10))/(i+5);
      r4(i)=log2(sing(5+i,10))/(i+5);
      r5(i)=log2(double((Ham(i+5,10,j))))/(i+5);
      r6(i)=log2(double((jon2(i+5,10,j))))/(i+5);
 end
 echt10(1:4)=inf;
```

```
r1(1:4)=inf;
r2(1:4)=inf;
r3(1:4)=inf;
r4(1:4)=inf;
r5(1:4)=inf;
r6(1:4)=inf;

for i=7:23
    echt12(i)=(log2(echte12(i)))/(i+5);
    s1(i)=log2(maxA(i+5,12,j))/(i+5);
    s2(i)=log2(MaxA2(i+5,12,j))/(i+5);
    s3(i)=log2(plo(5+i,12))/(i+5);
    s4(i)=log2(sing(5+i,12))/(i+5);
    s5(i)=log2(double((Ham(i+5,12,j))))/(i+5);
    s6(i)=log2(double((jon2(i+5,12,j))))/(i+5);

end
echt12(1:6)=inf;
s1(1:6)=inf;
s2(1:6)=inf;
s3(1:6)=inf;
s4(1:6)=inf;
s5(1:6)=inf;
s6(1:6)=inf;
```

B

RESULTS FOR $A(n,d)$

Table B.1: Upper bounds on $A(n,4)$

n	$A^*(n,4)$	$Lin(n,4)$	$LinExtra(n,4)$	$Plot(n,4)$	$Sing(n,4)$	$Ham(n,4)$	$John(n,4)$
6	4	4	4	4	8	5	4
7	8	8	8	8	16	9	8
8	16	16	16	16	32	16	16
9	20	25	21		64	28	25
10	40	42	42		128	51	51
11	72	85	81		256	93	83
12	144	170	160		512	170	160
13	256	292	256		1.024	315	292
14	512	512	512		2.048	585	585
15	1.024	1.024	1.024		4.096	1.092	1.024
16	2.048	2.048	2.048		8.192	2.048	2.048
17	3.276	3.640	3.276		16.384	3.855	3.615
18	6.552	6.553	6.553		32.768	7.281	7.084
19	13.104	13.107	13.107		65.536	13.797	13.107
20	26.168	26.214	26.214		131.072	26.214	26.214
21	43.688	47.662	43.690		262.144	49.932	47.662
22	87.333	87.381	87.381		524.288	95.325	95.325
23	172.361	174.762	173.491		1.048.576	182.361	174.103
24	344.308	349.525	344.636		2.097.152	349.525	344.308
25	599.184	645.277	599.186		4.194.304	671.088	645.277
26	1.198.368	1.198.372	1.198.372		8.388.608	1.290.555	1.290.555
27	2.396.736	2.396.745	2.396.745		16.777.216	2.485.513	2.396.745
28	4.792.950	4.793.490	4.793.490		33.554.432	4.793.490	4.793.490

Table B.2: Upper bounds on $A(n,6)$

n	$A^*(n,6)$	$Lin(n,6)$	$LinExtra(n,6)$	$Plot(n,6)$	$Sing(n,6)$	$Ham(n,6)$	$John(n,6)$
6	2	2	2	2	2	2	2
7	2	2	2	2	4	2	2
8	2	3	2	2	8	4	3
9	4	4	4	4	16	6	4
10	6	6	6	6	32	11	8
11	12	12	12	12	64	18	13
12	24	24	24	24	128	30	24
13	32	40	34		256	51	39
14	64	64	64		512	89	69
15	128	128	128		1.024	154	129
16	256	256	256		2.048	270	256
17	340	425	412		4.096	478	428
18	673	682	682		8.192	851	851
19	1.237	1.289	1.289		16.384	1.524	1.394
20	2.279	2.373	2.373		32.768	2.744	2.448
21	4.096	4.443	4.339		65.536	4.969	4.474
22	6.941	7.723	6.943		131.072	9.039	8.665
23	13.674	13.775	13.775		262.144	16.513	14.994
24	24.106	24.107	24.107		524.288	30.283	29.214
25	47.538	48.148	48.148		1.048.576	55.738	53.430
26	84..260	93.622	86.133		2.097.152	102.927	95.596
27	157.285	163.840	162.401		4.194.304	190.650	190.650
28	291.269	291.271	291.271		8.388.608	354.136	341.617

Table B.3: Upper bounds on $A(n,8)$

n	$A^*(n,8)$	$Lin(n,8)$	$LinExtra(n,8)$	$Plot(n,8)$	$Sing(n,8)$	$Ham(n,8)$	$John(n,8)$
6	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1
8	2	2	2	2	2	2	2
9	2	2	2	2	4	2	2
10	2	2	2	2	8	3	2
11	2	3	2	2	16	5	3
12	4	4	4	4	32	8	5
13	4	5	4	4	64	13	9
14	8	8	8	8	128	21	14
15	16	16	16	16	256	34	23
16	32	32	32	32	512	56	38
17	36	50	44		1.024	94	64
18	71	81	72		2.048	157	107
19	131	145	131		4.096	265	179
20	256	290	262		8.192	451	313
21	512	571	522		16.384	776	595
22	1.024	1.024	1.024		32.768	1.342	1.092
23	2.048	2.048	2.048		65.536	2.337	2.071
24	4.096	4.096	4.096		131.072	4.096	4.096
25	5.421	6.474	6.427		262.144	7.216	6.717
26	9.275	10.435	10.337		524.288	12.777	11.894
27	17.099	18.189	17.804		1.048.576	22.733	20.463
28	32.151	32.206	32.206		2.097.152	40.622	40.520

Table B.4: Upper bounds on $A(n,10)$

n	$A^*(n,10)$	$Lin(n,10)$	$LinExtra(n,10)$	$Plot(n,10)$	$Sing(n,10)$	$Ham(n,10)$	$John(n,10)$
6	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1
10	2	2	2	2	2	2	2
11	2	2	2	2	4	2	2
12	2	2	2	2	8	3	2
13	2	2	2	2	16	5	3
14	2	3	2	2	32	7	4
15	4	4	4	4	64	11	6
16	4	5	4	4	128	16	11
17	6	6	6	6	256	26	17
18	10	10	10	10	512	40	26
19	20	20	20	20	1.024	64	40
20	40	40	40	40	2.048	104	64
21	47	64	53		4.096	169	111
22	84	95	94		8.192	277	181
23	150	151	151		16.384	460	297
24	268	280	280		32.768	769	500
25	466	551	551		65.536	1.295	844
26	836	1.040	1.030		131.072	2.196	1.530
27	1.585	1.765	1.764		262.144	3.748	2.614
28	2.817	3.200	3.200		524.288	6.436	4.555

Table B.5: Upper bounds on $A(n,12)$

n	$A^*(n,12)$	$Lin(n,12)$	$LinExtra(n,12)$	$Plot(n,12)$	$Sing(n,12)$	$Ham(n,12)$	$John(n,12)$
6	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1
11	1	1	1	1	1	1	1
12	2	2	2	2	2	2	2
13	2	2	2	2	4	2	2
14	2	2	2	2	8	3	2
15	2	2	2	2	16	4	3
16	2	3	2	2	32	6	4
17	2	3	2	2	64	9	6
18	4	4	4	4	128	13	8
19	4	4	4	4	256	20	14
20	6	6	6	6	512	31	20
21	8	8	8	8	1.024	48	30
22	12	12	12	12	2.048	75	46
23	24	24	24	24	4.096	118	71
24	48	48	48	48	8.192	188	112
25	55	75	63		16.384	302	194
26	96	113	108		32.768	490	311
27	169	170	170		65.536	801	502
28	288	288	288		131.072	1.321	818

C

TABLES FOR $A(n, d, w)$

Table C.1: Upper bounds on $A(n, 4, w)$ [7, 9].

n \ w	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
4	1	1	2	1	1	0	0	0	0	0	0	0	0	0	0
5	1	1	2	2	1	1	0	0	0	0	0	0	0	0	0
6	1	1	3	4	3	1	1	0	0	0	0	0	0	0	0
7	1	1	3	7	7	3	1	1	0	0	0	0	0	0	0
8	1	1	4	8	14	8	4	1	1	0	0	0	0	0	0
9	1	1	4	12	18	18	12	4	1	1	0	0	0	0	0
10	1	1	5	13	30	36	30	13	5	1	1	0	0	0	0
11	1	1	5	17	35	66	66	35	17	5	1	1	0	0	0
12	1	1	6	20	51	84	132	84	51	20	6	1	1	0	0
13	1	1	6	26	65	132	182	182	132	65	26	6	1	1	0
14	1	1	7	28	91	182	308	364	308	182	91	28	7	1	1
15	1	1	7	35	105	271	455	660	660	455	271	105	35	7	1
16	1	1	8	37	140	336	722	1040	1320	1040	722	336	140	37	8
17	1	1	8	44	157	476	952	1753	2210	2210	1753	952	476	157	44
18	1	1	9	48	198	565	1428	2448	3944	4420	3944	2448	1428	565	198
19	1	1	9	57	228	752	1789	3876	5814	8326	8326	5814	3876	1789	752
20	1	1	10	60	285	912	2506	5111	9690	12920	16652	12920	9690	5111	2506
21	1	1	10	70	315	1197	3192	7518	13416	22610	27132	27132	22610	13416	7518
22	1	1	11	73	385	1386	4389	10032	20674	32794	49742	54264	49742	32794	20674
23	1	1	11	83	419	1771	5313	14421	28842	52833	75426	104006	103539	75426	52833
24	1	1	12	88	498	2011	7084	18216	43263	76912	126799	164565	208012	164565	126799
25	1	1	12	100	550	2490	8379	25300	56925	120175	192280	288197	342843	342843	288197
26	1	1	13	104	650	2860	10790	31122	82225	164450	312455	454480	624387	685686	624387
27	1	1	13	117	702	3510	12870	41618	105036	246675	444015	766935	1022580	1296803	1296803
28	1	1	14	121	819	3931	16380	51480	145663	326778	690690	1130220	1789515	2202480	2593606

Table C.2: Upper bounds on $A(n,6,w)$ [7, 9].

n \ w	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
4	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
5	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
6	1	1	1	2	1	1	1	0	0	0	0	0	0	0	0
7	1	1	1	2	2	1	1	1	0	0	0	0	0	0	0
8	1	1	1	2	2	2	1	1	1	0	0	0	0	0	0
9	1	1	1	3	3	3	3	1	1	1	1	0	0	0	0
10	1	1	1	3	5	6	5	3	1	1	1	0	0	0	0
11	1	1	1	3	6	11	11	6	3	1	1	1	0	0	0
12	1	1	1	4	9	12	22	12	9	4	1	1	1	0	0
13	1	1	1	4	13	18	26	26	18	13	4	1	1	1	0
14	1	1	1	4	14	28	42	42	42	28	14	4	1	1	1
15	1	1	1	5	15	42	70	78	78	70	42	15	5	1	1
16	1	1	1	5	20	48	112	138	150	138	112	48	20	5	1
17	1	1	1	5	20	68	136	228	280	280	228	136	68	20	5
18	1	1	1	6	22	72	199	349	428	425	428	349	199	72	22
19	1	1	1	6	25	83	228	520	718	789	789	718	520	228	83
20	1	1	1	6	30	100	276	651	1107	1363	1403	1363	1107	651	276
21	1	1	1	7	31	126	350	828	1695	2359	2685	2685	2359	1695	828
22	1	1	1	7	37	136	462	1100	2277	3766	4415	5064	4415	3766	2277
23	1	1	1	7	40	170	521	1518	3162	5819	7521	7953	7953	7521	5819
24	1	1	1	8	42	192	680	1786	4554	8432	12186	14682	15906	14682	12186
25	1	1	1	8	50	210	800	2428	5581	12620	19037	24630	30587	30587	24630
26	1	1	1	8	52	260	910	2971	7891	16122	28893	42080	50169	61174	50169
27	1	1	1	9	54	280	1170	3510	10027	23673	43529	66079	84574	91080	91080
28	1	1	1	9	63	302	1306	4680	12285	31195	63756	104231	142117	164220	169740

Table C.3: Upper bounds on $A(n,8,w)$ [7, 9].

n \ w	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
4	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
5	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
6	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
7	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
8	1	1	1	1	2	1	1	1	1	0	0	0	0	0	0
9	1	1	1	1	2	2	1	1	1	1	0	0	0	0	0
10	1	1	1	1	2	2	2	1	1	1	1	0	0	0	0
11	1	1	1	1	2	2	2	2	1	1	1	1	0	0	0
12	1	1	1	1	3	3	4	3	3	1	1	1	1	0	0
13	1	1	1	1	3	3	4	4	3	3	1	1	1	1	0
14	1	1	1	1	3	4	7	8	7	4	3	1	1	1	1
15	1	1	1	1	3	6	10	15	15	10	6	3	1	1	1
16	1	1	1	1	4	6	16	16	30	16	16	6	4	1	1
17	1	1	1	1	4	7	17	24	34	34	2417	7	4	1	1
18	1	1	1	1	4	9	21	33	49	58	49	33	21	9	4
19	1	1	1	1	4	12	28	52	78	103	103	78	52	28	12
20	1	1	1	1	5	16	40	80	130	173	206	173	130	80	40
21	1	1	1	1	5	21	56	120	210	302	363	363	302	210	120
22	1	1	1	1	5	21	77	176	330	473	634	680	634	473	330
23	1	1	1	1	5	23	80	253	506	707	1025	1288	1288	1025	707
24	1	1	1	1	6	24	92	274	759	1041	1551	2142	2576	2142	1551
25	1	1	1	1	6	30	100	328	856	1486	2333	3422	4087	4087	3422
26	1	1	1	1	6	30	130	371	1066	2108	3496	5225	6741	7080	6741
27	1	1	1	1	6	32	135	500	1252	2914	4986	7833	10547	11981	11981
28	1	1	1	1	7	33	149	540	1750	3895	7016	11939	17011	21152	22710

Table C.4: Upper bounds on $A(n, 10, w)$ [7, 9].

n \ w	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
4	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
5	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
6	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
7	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
8	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
9	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
10	1	1	1	1	1	2	1	1	1	1	1	0	0	0	0
11	1	1	1	1	1	2	2	1	1	1	1	1	0	0	0
12	1	1	1	1	1	2	2	2	1	1	1	1	1	0	0
13	1	1	1	1	1	2	2	2	2	1	1	1	1	1	0
14	1	1	1	1	1	2	2	2	2	2	1	1	1	1	1
15	1	1	1	1	1	3	3	3	3	3	3	1	1	1	1
16	1	1	1	1	1	3	3	4	4	4	3	3	1	1	1
17	1	1	1	1	1	3	3	5	6	6	5	3	3	1	1
18	1	1	1	1	1	3	4	6	9	10	9	6	4	3	1
19	1	1	1	1	1	3	4	8	12	19	19	12	8	4	3
20	1	1	1	1	1	4	5	10	17	20	38	20	17	10	5
21	1	1	1	1	1	4	7	13	21	35	42	42	35	21	13
22	1	1	1	1	1	4	7	16	33	51	72	80	72	51	33
23	1	1	1	1	1	4	8	20	46	81	117	135	135	117	81
24	1	1	1	1	1	4	9	24	60	118	171	223	247	223	171
25	1	1	1	1	1	5	10	32	75	158	262	380	434	434	380
26	1	1	1	1	1	5	13	36	104	214	406	566	702	754	702
27	1	1	1	1	1	5	14	48	121	299	571	882	1201	1419	1419
28	1	1	1	1	1	5	16	56	168	376	821	1356	1977	2438	2629

Table C.5: Upper bounds on $A(n, 12, w)$ [7, 9].

n \ w	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
4	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
5	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
6	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
7	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
8	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
9	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
10	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
11	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
12	1	1	1	1	1	1	2	1	1	1	1	1	1	0	0
13	1	1	1	1	1	1	2	2	1	1	1	1	1	1	0
14	1	1	1	1	1	1	2	2	2	1	1	1	1	1	1
15	1	1	1	1	1	1	2	2	2	2	1	1	1	1	1
16	1	1	1	1	1	1	2	2	2	2	2	1	1	1	1
17	1	1	1	1	1	1	2	2	2	2	2	2	1	1	1
18	1	1	1	1	1	1	3	3	3	4	3	3	3	1	1
19	1	1	1	1	1	1	3	3	3	4	4	3	3	3	1
20	1	1	1	1	1	1	3	3	5	5	6	5	5	3	3
21	1	1	1	1	1	1	3	3	5	7	7	7	7	5	3
22	1	1	1	1	1	1	3	4	6	8	11	12	11	8	6
23	1	1	1	1	1	1	3	4	6	10	16	23	23	16	10
24	1	1	1	1	1	1	4	4	9	16	24	24	46	24	24
25	1	1	1	1	1	1	4	5	10	25	37	42	50	50	42
26	1	1	1	1	1	1	4	5	13	26	48	66	83	91	83
27	1	1	1	1	1	1	4	6	15	39	64	100	140	156	156
28	1	1	1	1	1	1	4	8	19	45	87	149	199	245	265