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

Codes for Noisy Channels with unknown offset

(Dutch title: Codes voor kanalen met ruis en onbekende offset)

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

GUUS VAN HEMERT

Delft, Nederland
July 2020

BSc report APPLIED MATHEMATICS

"Codes for Noisy Channels with unknown offset"

(Dutch title: "Codes voor kanalen met ruis en onbekende offset "

GUUS VAN HEMERT

Delft University of Technology

Supervisors

Dr.ir. J. H. Weber
R. Bu, MSc

Committee

Dr. Y. van Gennip Prof.dr.ir. K. Aardal

July, 2020 Delft

Abstract

Storage systems, such as flash memories, suffer, apart from the always present noise, also from offset. The presence of this noise can decrease the performance of a decoder using the Euclidean distance significantly. To negate the effects of offset, a new distance, the modified Euclidean distance, was introduced, which offers immunity to offset. However, the modified Euclidean distance is less resistant to noise, which calls for methods to improve its resistance. The coset of Hamming codes, constant weight codes and Berger codes are discussed and are simulated to investigate their performance with both distances. These codes are compared to each other for the Euclidean distance and the modified Euclidean distance.

Contents

Chapter 1

Introduction

1.1 Problem Statement

In storage and communication systems noise is oftentimes an issue. When we want to transmit data, noise can cause errors within this data, which can lead to wrong data being received. However apart from the always present noise, in storage systems in particular, there might also be an offset which may cause problems as well. Particularly in Flash memories, offset is a well known issue [7]. This offset is a result of charge leakage, which is a major issue within these Flash memories, caused by factors such as temperature and aging. Due to this leakage, the stored voltage level will decrease, which causes the mentioned offset. Since the amount of voltage leakage is not constant, but depends on time, it is not possible to shift all the voltage levels of the data to neutralize the shift caused by the offset. Luckily there are various methods to counteract the offset, however they also come with their own disadvantages. Some of these methods will be discussed in Section 1.3 where we will formalize the problem.

1.2 Introduction to Coding Theory

To be able to formalize the problem, we first need to introduce the basic concepts of coding theory. This will be partially based on [3]. Coding theory is based on adding redundancy to the information to be able to detect and correct errors made during transmission. Typically, we represent information using a sequence of different symbols. In storage systems these symbols represent a certain voltage level. These symbols we can encode by adding redundant symbols to protect our information symbols. All symbols we send, that is the information symbols and the redundant symbols, form a codeword. Gathering multiple codewords we can construct a code, which is the set of all the codewords that we want to be able to send. If done properly we make sure that not all possible sequences of a given length form a codeword, such that if we would receive for example a sequence which is not part of the code, we know that an error was made, since we could not have send this sequence. In general we can choose q symbols to construct our words, so we can choose the symbols from the q-ary alphabet $\{0, 1, \ldots, q-1\}$, this means that we have q different voltage levels, where the 0 is the lowest voltage level and $q-1$ represents the highest voltage

level. When the codewords are constructed from n different symbols chosen from the q-ary alphabet, the codewords are elements from the vector space $\mathbb{F}_q^n$ over the finite field $\mathbb{F}_q$. For example if we would choose $q = 5$ and $n = 5$, some possible codewords could be $\{12341, 03123, 44120, 00000\}$. In this report we will only be looking at the binary case, so that means that our codewords are taken from $\mathbb{F}_2^n$ over the finite field $\mathbb{F}_2$. In the binary case the symbols consist of only 0's and 1's and they are called bits. When a word is received, there is a high probability that the received word is not part of the code due to errors made during transmission. Since we do not know which codeword was sent, we need to choose a codeword from the code that has the highest probability to be the codeword that was actually sent. The selection of this codeword is done by using a distance measure. Upon receipt of a vector, the decoder calculates the distance from the received vector to each codeword in the code and picks the codeword with the smallest distance to the received word. This way the chances are the highest of picking the right codeword.

1.3 Formalization of the Problem

Now that we have given a small introduction into the coding theory, we can formalize our problems with the use of mathematical terms. As we mentioned in Section 1.1 the offset is dependent on time, however it is assumed to be constant within each word, and thus the amount of offset is the same for each bit in the word, but it can differ from word to word. Apart from the offset, we also still have to take the noise in consideration, so if we send the codeword $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ from a code book $C \subseteq \{0, 1\}^n$, then we receive the word

$$\mathbf{r} = \mathbf{x} + \boldsymbol{\nu} + b\mathbf{1},$$

where

- $\mathbf{x} = (x_1, \ldots, x_n)$ is the sent codeword of length n,
- $\boldsymbol{\nu} = (\nu_1, \ldots, \nu_n)$ is the noise vector where ν_i is independently normally distributed with mean 0 and variance σ^2 for all $i = 1, \ldots, n$,
- b is the unknown offset with $b \in \mathbb{R}$,
- $\mathbf{1} = (1, \ldots, 1)$ is the all-1 vector of length n,
- $\mathbf{r} = (r_1, \ldots, r_n)$ is the received word with $r_i \in \mathbb{R}$.

When offset plays no part and there is only noise, a good way to decode is by using the squared Euclidean distance

$$\delta(\mathbf{u}, \mathbf{v}) = \sum_{i=1}^{n} (u_i - v_i)^2. \tag{1.1}$$

Here $\mathbf{u} = (u_1, \ldots, u_n)$ and $\mathbf{v} = (v_1, \ldots, v_n)$ are both vectors of length n. While this distance performs well when there is only noise, when there is also offset, the performance of a decoder using the δ distance becomes worse, especially when the offset is quite large. To reduce the number of errors made

during the decoding process in the presence of offset, we need to adopt another method.

There are various ways to reduce the number of errors caused by the offset, like using reference symbols [4] or using the Pearson distance, based on the Pearson correlation coefficient [5],[9]. In the former method, two reference symbols, equal to the lowest signal level and the highest signal level, are assigned to two positions in each codeword. These positions and the signal levels of the reference symbols are known to the receiver, meaning that the receiver can measure the amplitude of the reference symbols and normalize the amplitudes of the other symbols of the received word. While it is a solution to offset, it does come with a price as we need to add two bits to each word, resulting in a higher redundancy. As for the Pearson codes, while decoding with the Pearson distance offers immunity to offset mismatch, it is more susceptible to noise.

Another distance that can be used is the modified Euclidean distance $\delta^*(\mathbf{u}, \mathbf{v})$ defined in [8] as:

$$\delta^*(\mathbf{u}, \mathbf{v}) = \delta(\mathbf{u} - \bar{\mathbf{u}}\mathbf{1}, \mathbf{v} - \bar{\mathbf{v}}\mathbf{1}), \tag{1.2}$$

where $\mathbf{u} = (u_1, \ldots, u_n)$ and $\mathbf{v} = (v_1, \ldots, v_n)$ are vectors of length n, $\delta(\mathbf{u}, \mathbf{v})$ is the squared Euclidean distance and $\bar{\mathbf{w}} = \frac{1}{n}\sum_{i=1}^{n} w_i$ is the average. For example, the modified Euclidean distance between the two words $\mathbf{u} = (1, 0, 1, 1, 0, 0)$ and $\mathbf{v} = (0, 0, 1, 1, 1, 1)$ is $\delta^*(\mathbf{u}, \mathbf{v}) = \delta(\mathbf{u} - 0.5\mathbf{1}, \mathbf{v} - 0.67\mathbf{1}) \approx 2.83$.

While the use of this distance offers immunity to offset, it poses another problem as it is more sensitive to noise. The reason for this can be found by looking at a possible interpretation of (1.2). This function can be interpreted as a projection onto $S = \{\mathbf{a} \in \mathbb{R}^n : \bar{\mathbf{a}} = 0\}$, since the average of $\mathbf{a} - \bar{\mathbf{a}}\mathbf{1}$ is equal to zero for all vectors $\mathbf{a} \in \mathbb{R}^n$. This causes that all the projected words on S are closer to each other than the original words, resulting in higher noise sensitivity. This is also apparent for the words of the previous example. In that example we saw that $\delta^*(\mathbf{u}, \mathbf{v}) \approx 2.83$. When we also calculate the Euclidean distance we get $\delta(\mathbf{u}, \mathbf{v}) = 3$, which shows that they are indeed closer to each other for the δ^* distance. Luckily there are multiple codes that can be constructed that have a better noise performance while also maintaining the immunity to offset. Examples of these codes can be found in the cosets of linear codes, constant weight codes and unordered codes.

1.4 Organization of the Report

The remaining report will be structured in the following way. In Chapter 2 some preliminaries on coding theory, which is needed for our report, will be provided. Then in Chapter 3, 4 and 5 we will be discussing the use of the Hamming code, the constant weight code and the Berger code respectively and their use in the problem setting along with simulations of these codes. In Chapter 6 the performances and properties of these codes will be compared. Finally, in Chapter 7 the report is concluded.

Chapter 2

Prerequisites

This chapter will provide some extra information on coding theory, which will be used in later parts of the report. Section 2.1 will provide some general definitions and terminology used in coding theory. Following that, Section 2.2 will introduce the concept of linear codes and parity check matrices. Section 2.3 introduces the definition of cosets of codes and lastly, Section 2.4 will give theoretical approximations of the word error rate.

2.1 Definitions and Terminology

In this section we will be introducing some definitions and terms that are common and useful within the coding theory. Last section we finished by talking about using a distance measure to be able to detect and correct errors. The most common distance when talking about binary codes is the Hamming distance. The Hamming distance between two binary words $\mathbf{u}, \mathbf{v}$ of length n is defined as

$$d(\mathbf{u}, \mathbf{v}) = \sum_{i=1}^{n} |u_i - v_i|. \tag{2.1}$$

For example, the Hamming distance between $\mathbf{u} = (1, 1, 0, 0, 1, 1)$ and $\mathbf{v} = (0, 1, 0, 1, 0, 1)$ is $d(\mathbf{u}, \mathbf{v}) = 2$. Note that the Hamming distance is actually the same as the Euclidean distance if both words are elements from $\mathbb{F}_2^n$, however since in our scenario the received vector is no longer an element from $\mathbb{F}_2^n$, but from $\mathbb{R}^n$, the Hamming distance is no longer defined for the received vector. Another common term used for binary words is the weight of a word. The weight of a binary word $\mathbf{u}$ is defined as

$$w(\mathbf{u}) = |\{i : u_i = 1\}|. \tag{2.2}$$

Taking the words $\mathbf{u}$ and $\mathbf{v}$ from the previous example, their weights are respectively $w(\mathbf{u}) = 4$ and $w(\mathbf{v}) = 3$. When discussing different codes, it is often useful to also consider their respective redundancy and code rate. Before we introduce the redundancy and the code rate, we will first define the cardinality $|C|$ of a code C, which is the number of codewords in C

$$|C| = |\{\mathbf{u} : \mathbf{u} \in C\}|. \tag{2.3}$$

As for the redundancy and code rate of a code, the redundancy is defined as

$$r = n - \log_2(|C|) \tag{2.4}$$

and the code rate is defined as

$$c = \frac{\log_2(|C|)}{n}. \tag{2.5}$$

The code rate of a code can be seen as the ratio of the codewords that carry the information, while the redundancy is the amount of space we sacrifice to be able to protect the information when sending a word. For example, when we have 8 information bits in a word of length 12, we have a code rate of 0.67. To make sure that these codewords containing the information have a big enough distance between them, we append 4 bits, so a redundancy of 4, to protect the message during transmission.

2.2 Linear Codes

A common class of codes is the linear code. A linear $[n, k]$-code is a code with the property that for any two codewords $\mathbf{u}, \mathbf{v}$ in a code C, the sum of the codewords $\mathbf{u}+\mathbf{v}$ is again part of the code. Note that this property also implies that the all-0 vector, $\mathbf{0} = (0, \ldots, 0)$ is always an element of a linear code, since for any binary word $\mathbf{c}$ it holds that $\mathbf{c} + \mathbf{c} = \mathbf{0}$. The parameters n and k indicate the length of the codewords and the dimension of the code respectively. The dimension of a code is the number of codewords in the basis of the code from which the entire code can be constructed through linear combinations of the basis vectors. An example of a linear code is the code $C = \{0000, 0001, 0010, 0011\}$. It is a code of dimension 2 with basis vectors $\mathbf{b}_1 = (0, 0, 0, 1)$ and $\mathbf{b}_2 = (0, 0, 1, 0)$.

Linear $[n, k]$-codes have k information bits and thus a code rate of $\frac{k}{n}$ and a redundancy of k. The property that the sum of two codewords is again a codeword offers some algebraic structure to linear codes, which can be found helpful for decoding. When describing a linear code, a common way is by describing it with a generator matrix. A generator matrix G of a linear $[n, k]$-code will be given by a $k \times n$ matrix. Using the generator matrix, the code can be constructed by taking all possible binary words $\mathbf{u}$ of length k and multiplying them with the matrix. Formally this means that C can be written as

$$C = \{\mathbf{u}G, \quad \forall \mathbf{u} \in \mathbb{F}_2^k\}. \tag{2.6}$$

So for any codeword $\mathbf{v}$ of length n in the code, there exists a word $\mathbf{u}$ of length k such that $\mathbf{v} = \mathbf{u}G$. Alongside the generator matrix, we also introduce the parity check matrix. A matrix H is a parity check matrix for a linear code C if the columns of H form a basis for the dual code $C^\perp$. The dual code $C^\perp$ of a code C is the set

$$C^\perp = \{\mathbf{u} \in \mathbb{F}_2^n | \mathbf{u} \cdot \mathbf{v} = 0, \quad \forall \mathbf{v} \in C\}, \tag{2.7}$$

where $\mathbf{x} \cdot \mathbf{y}$ is the inner product between the two vectors. The dual code consists of all the words orthogonal to the words in the original code. A parity check matrix of a linear $[n, k]$-code is a $n \times (n - k)$ matrix. Since the columns of the

parity check matrix are a basis for the dual code, the code can be constructed using the parity check matrix in the following way

$$C = \{\mathbf{v} \in \mathbb{F}_2^n | vH = 0\}. \tag{2.8}$$

2.3 Cosets of Codes

Furthermore we would like to discuss cosets of codes as it is useful for a later chapter in this report. Let C be a linear $[n, k]$-code and $\boldsymbol{\beta}$ be any binary vector of length n, then we define the coset of C determined by $\boldsymbol{\beta}$, denoted by $C_{\boldsymbol{\beta}}$, as

$$C_{\boldsymbol{\beta}} = \{\boldsymbol{\beta} + \mathbf{c} \pmod 2 | \mathbf{c} \in C\}. \tag{2.9}$$

The notation $\boldsymbol{\beta} + \mathbf{c} \pmod 2$ means that for words $\boldsymbol{\beta}$ and $\mathbf{c}$ of length n, we take $\beta_i + c_i \pmod 2$ for each $i = 1, \ldots, n$. If we for example take the linear code from the previous section and the word $\boldsymbol{\beta} = (1, 0, 0, 1)$, then the coset $C_{\boldsymbol{\beta}}$ is $\{1001, 1000, 1011, 1010\}$. This example also makes clear that, even though C is a linear code, the coset does not have to be linear again, however it does have the same number of codewords and length.

2.4 Theoretical Word Error Rate

Lastly we give some theoretical approximations of the word error rate (WER). The WER is the number of words that are decoded improperly as a fraction of the total number of words sent. So for example if we would have a code with a WER of 0.5 and we send 10.000 words, then on average only 5000 of those words would be decoded properly. The WER of a code C when there is no offset and the decoder uses (1.1), can be estimated for small values of σ as follows [8]

$$\text{WER} \approx N_{\delta_{\min}} \times Q\left(\frac{\sqrt{\delta_{min}}}{2\sigma}\right), \tag{2.10}$$

where Q is the well known Q-function

$$Q(z) = \frac{1}{\sqrt{2\pi}} \int_z^\infty e^{-u^2/2} \mathrm{d}u \tag{2.11}$$

and

$$N_{\delta_{\min}} = \frac{1}{|C|} \sum_{\mathbf{u} \in C} |\{\mathbf{v} \in C, \mathbf{v} \neq \mathbf{u} : \delta(\mathbf{u}, \mathbf{v}) = \delta_{\min}\}| \tag{2.12}$$

and

$$\delta_{\min} = \min_{\mathbf{u}, \mathbf{v} \in C, \mathbf{u} \neq \mathbf{v}} \delta(\mathbf{u}, \mathbf{v}). \tag{2.13}$$

In a similar matter, one can also find an approximation of the WER when using (1.2) for small values of σ when there is no offset

$$\text{WER}^* \approx N^*_{\delta_{\min}} \times Q\left(\frac{\sqrt{\delta^*_{min}}}{2\sigma}\right), \tag{2.14}$$

where

$$N^*_{\delta_{\min}} = \frac{1}{|C|} \sum_{\mathbf{u} \in C} |\{\mathbf{v} \in C, \mathbf{v} \neq \mathbf{u} : \delta^*(\mathbf{u}, \mathbf{v}) = \delta^*_{\min}\}| \tag{2.15}$$

and

$$\delta^*_{\min} = \min_{\mathbf{u}, \mathbf{v} \in C, \mathbf{u} \neq \mathbf{v}} \delta^*(\mathbf{u}, \mathbf{v}). \tag{2.16}$$

Chapter 3

Hamming Codes

In this chapter we will discuss the first method to improve the performance of the modified Euclidean distance with respect to noise, namely using the coset of the Hamming codes. Section 3.1 will define Hamming codes and shortened Hamming codes while also choosing a suitable coset to improve the performance. After the shortened Hamming codes have been introduced, we will simulate its performance in Section 3.2. Finally, in Section 3.3 the results of these simulations are discussed.

3.1 (Shortened) Hamming Codes

As shown in [8], using a well chosen coset of a code can result in a better noise performance using δ^* while maintaining the offset immunity. An example of such a coset is the coset of the Hamming code or the shortened Hamming code. A Hamming code $\mathcal{H}_s$, with $s \geq 3$, is a $[2^s-1, 2^s-1-s, 3]$ code where its parity check matrix H_s is a $s \times (2^s-1)$ matrix which has all possible columns of length s except for the all-zero column. From this Hamming code, we can create the shortened Hamming code $\mathcal{H}_s[a]$ by removing a columns from the parity check matrix of $\mathcal{H}_s$, while making sure the rank of the matrix does not change.

This results in a $[2^s-1-a, 2^s-1-s-a, 3]$ code for all $1 \leq a \leq 2^s-s-2$. Since the Hamming code contains both the all-1 vector and the all-0 vector, using the δ^* distance can cause problems, because $\delta^*(\mathbf{0}, \mathbf{1}) = 0$. So a coset has to be chosen in such a way that both $\mathbf{0}$ and $\mathbf{1}$ are not part of the same code. According to Theorem 2 in [8], we can choose a vector $\mathbf{a}$ of weight $\lfloor \frac{d_{min}}{2} \rfloor$, where d_{min} is the minimum Hamming distance of the code, to have a higher δ^*_{min}. As the Hamming code has $d_{\min} = 3$, we choose a binary vector of weight 1.

3.2 Simulation of the Shortened Hamming Code

To investigate the performance of these codes with respect to their cosets, we will do a simulation. In this simulation we will evaluate the performance of the $[12, 8, 3]$ shortened Hamming code $\mathcal{H}_4[3]$ and its coset $\mathcal{H}_{4,\boldsymbol{\beta}}[3]$. In this case the coset is obtained by adding the word $\boldsymbol{\beta}$ of length 12 with weight 1 to each word in the code. In Figure 3.1 we plot the WER against the signal to noise ratio (SNR) for an offset value of $b = 0.02$, where the SNR is defined as $-20\log_{10}(\sigma)$

dB. In Figure 3.2, we again plot the WER against the SNR but with an offset of $b = 0.2$.

3.3 Results of the Simulations

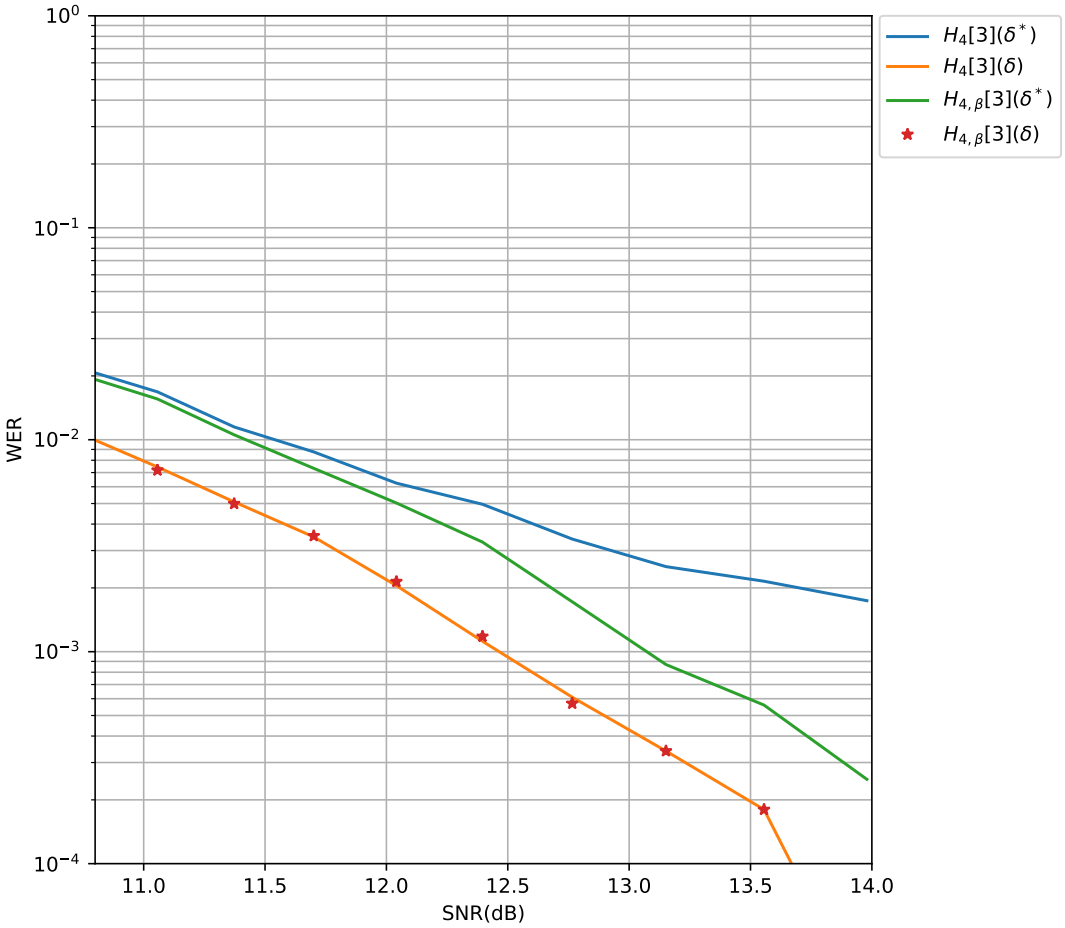

Figure 3.1: Simulation of the $\mathcal{H}_4[3]$ code and its coset $\mathcal{H}_{4,\boldsymbol{\beta}}[3]$ with an offset of $b = 0.02$

From Figure 3.1 we can observe that a decoder using δ has a better performance. Furthermore it is noticeable that when using δ there is almost no difference in using the shortened Hamming code $\mathcal{H}_4[3]$ or its coset $\mathcal{H}_{4,\boldsymbol{\beta}}[3]$ while it does make a big difference for high SNR values when using δ^*. While there is almost no difference in WER between the shortened Hamming code and its coset when using δ^* for low SNR values, as the SNR increases, so does the difference in performance. For example when we have a SNR of 14, the standard deviation σ of the noise is lower than for a SNR of 11. As expected, since there is less noise, the WER is smaller for both $\mathcal{H}_4[3]$ and $\mathcal{H}_{4,\boldsymbol{\beta}}[3]$, but the coset performs considerably better, since it does not contain both $\mathbf{0}$ and $\mathbf{1}$ and thus is more resistant to noise. However when the offset increases to $b = 0.2$, we can see that

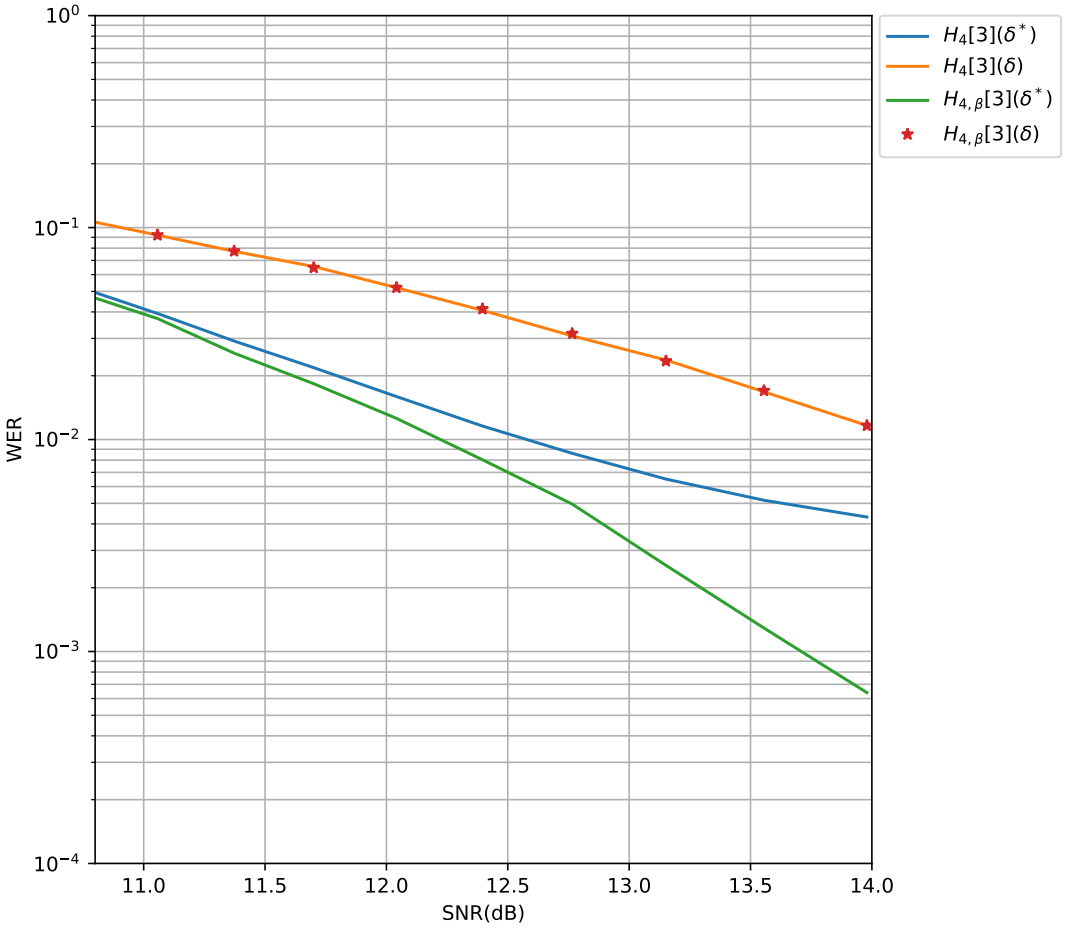

Figure 3.2: Simulation of the $\mathcal{H}_4[3]$ code and its coset $\mathcal{H}_{4,\boldsymbol{\beta}}[3]$ with an offset of $b = 0.2$

δ^* performs the same, which is to be expected, since it is immune to offset, but δ performs considerably worse.

Chapter 4

Constant Weight Codes

This chapter will discuss the second code which is useful to reduce the effect of offset. Section 4.1 will provide the information about what constant weight codes are. Section 4.2 introduces the the constructed code we used for simulations and Section 4.3 will discuss the acquired results of the simulation.

4.1 Constant Weight Codes

Another useful code is the constant weight code. Constant weight codes $\mathcal{C}(n, M, d, w)$ are codes with M codewords of length n, weight w and with a mutual Hamming distance of at least d, $0 \leq w \leq n$ and $2 \leq d \leq n$. As an example of a constant weight code we take the code

$$\begin{aligned}\mathcal{C}(5, 10, 2, 2) = \{&00011, 00101,\\ &00110, 01001,\\ &01010, 01100,\\ &10001, 10010,\\ &10100, 11000\}.\end{aligned}$$

This code is constructed by taking all the codewords of length 5 with weight 2. Constant weight codes have innate offset immunity, as we will see in Section 4.3, which is useful in our setting. A downside of the constant weight codes however, is that they do not have a nice structure, in the sense that the information bits are not separated from the redundant bits.

4.2 Simulation of the Constant Weight Code

Like we did for the shortened Hamming code and its coset, we will also do a simulation of a constant weight code. For this we choose the constant weight code $\mathcal{C}(7, 35, 2, 3)$, which contains all the 35 words of length 7 with weight 3. In Figure 4.1 we again chose for an offset of $b = 0.2$.

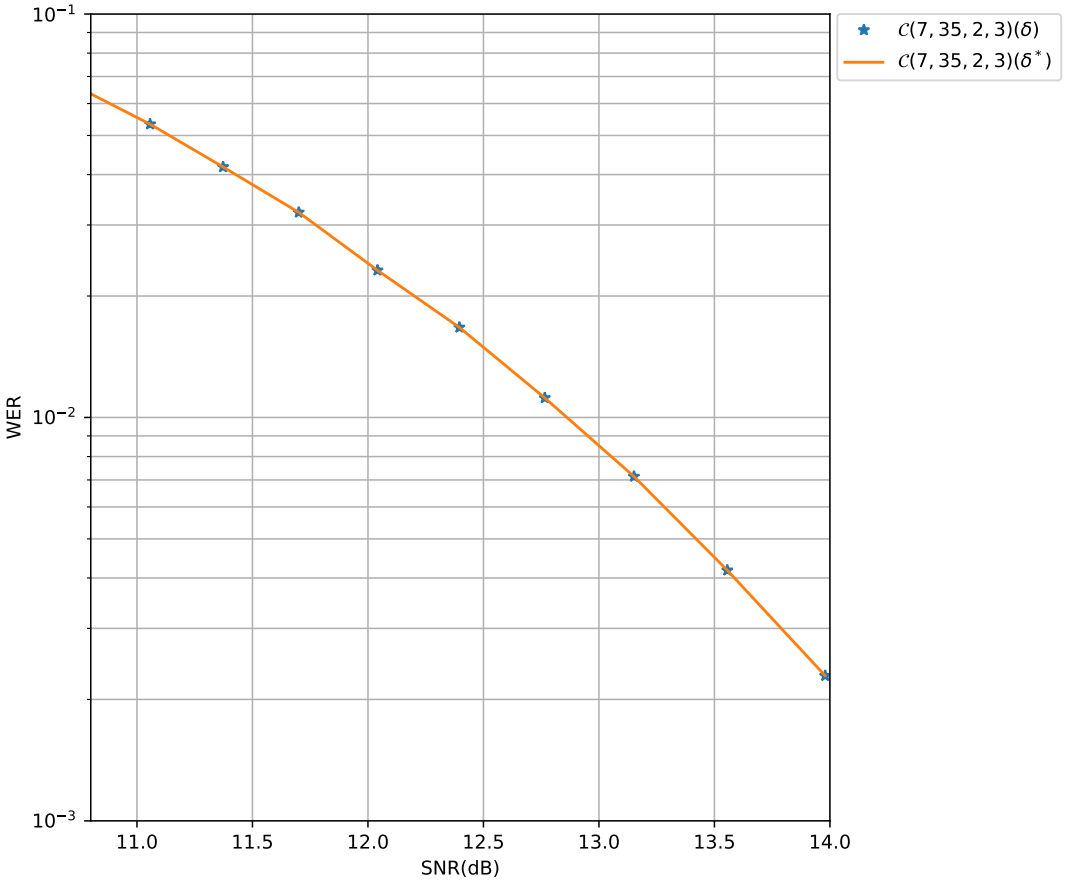

Figure 4.1: Simulation of the constant weight code $\mathcal{C}(7, 35, 2, 3)$ containing all possible codewords of length 7 with weight 3 with an offset of $b = 0.2$

4.3 Results of the Simulations

When evaluating the results of the simulations in figure 4.1 we remark that the WER is the same for both distance measures. The reason for this can be found by taking a closer look at (1.2) and (1.1). Assuming that the sent codeword $\mathbf{x}$ of length n is received as $\mathbf{r} = \mathbf{x} + \boldsymbol{\nu} + b\mathbf{1}$, we can calculate the δ^* distance to an arbitrary codeword $\mathbf{y}$ in the code:

$$\begin{aligned}\delta^*(\mathbf{r}, \mathbf{y}) = \delta^*(\mathbf{x} + \boldsymbol{\nu} + b\mathbf{1}, \mathbf{y}) &= \delta(\mathbf{x} + \boldsymbol{\nu} - (\bar{\mathbf{x}} + \bar{\boldsymbol{\nu}})\mathbf{1}, \mathbf{y} - \bar{\mathbf{y}}\mathbf{1}) \\ &= \sum_{i=1}^{n} \left(x_i + \nu_i - (\bar{\mathbf{x}} + \bar{\boldsymbol{\nu}}) - y_i + \bar{\mathbf{y}}\right)^2 \\ &= \sum_{i=1}^{n} \left(x_i + \nu_i - \bar{\boldsymbol{\nu}} - y_i\right)^2 .\end{aligned}$$

The third equation follows from the fact that all words have equal weight w and the average of each codeword is equal to $\frac{w}{n}$. Further developing this equation

yields

$$\delta^*(\mathbf{r},\mathbf{y}) = \sum_{i=1}^{n}(x_i - y_i)^2 + 2\sum_{i=1}^{n}\nu_i(x_i - y_i) + \sum_{i=1}^{n}\nu_i^2 - 2\bar{\boldsymbol{\nu}}\sum_{i=1}^{n}(x_i - y_i + \nu_i) + n\bar{\boldsymbol{\nu}}^2. \tag{4.1}$$

Doing the same for the δ distance gives:

$$\delta(\mathbf{r},\mathbf{y}) = \sum_{i=1}^{n}(x_i - y_i)^2 + 2\sum_{i=1}^{n}\nu_i(x_i - y_i) + \sum_{i=1}^{n}\nu_i^2 + 2b\sum_{i=1}^{n}(x_i - y_i + \nu_i) + nb^2. \tag{4.2}$$

Using (4.1) and (4.2) we can now look at the difference of these two functions in (4.3). The reason that the terms containing $\sum_{i=1}^{n}(x_i - y_i + \nu_i)$ become $\sum_{i=1}^{n}\nu_i$ lies again in the fact that all words have the same weight. From (4.3) we observe that the difference is constant for each codeword $\mathbf{y}$. Since the difference is constant, the word with minimum distance to $\mathbf{r}$ is the same for both δ^* and δ, which means that the WER will also be the same. Note that we can also conclude from this that constant weight codes are inherently immune to offset.

$$|\delta(\mathbf{r},\mathbf{y}) - \delta^*(\mathbf{r},\mathbf{y})| = \left|2(b + \bar{\boldsymbol{\nu}})\sum_{i=1}^{n}\nu_i + n(b^2 + \bar{\boldsymbol{\nu}}^2)\right|. \tag{4.3}$$

Chapter 5

Berger Codes

This chapter will follow the same structure as the previous two chapters. First, in Section 5.1, the Berger codes are introduced, secondly, we construct a Berger code for the simulations in Section 5.2 and we end this chapter by discussing the results of our simulations in Section 5.3.

5.1 Berger Codes

Berger codes are a type of unordered codes which are constructed by adding a tail of $\lceil \log_2(k) \rceil$ to k information bits, where the tail is the binary representation of the number of zeros in the k information bits. As a matter of fact, in the binary case Berger codes are optimal systematic unordered codes [6]. A code is said to be systematic when the information bits are separated from the redundant bits. Systematic codes have the advantage that the information can directly be retrieved from the codewords.

Binary unordered codes are a type of code where the positions of the ones in each codeword is not a subset of the positions of ones for each other codeword [2]. So for example the words $\mathbf{x} = (0, 1, 0, 1, 1)$ and $\mathbf{y} = (0, 0, 0, 1, 1)$ cannot be contained in the same code since the positions of the ones of $\mathbf{y}$ is a subset of the positions of the ones of $\mathbf{x}$. A property of unordered codes is that they are able to detect all unidirectional errors, which are errors of the type $0 \to 1$ or $1 \to 0$. Denote the Berger code of length n with k information bits and M codewords as $\mathcal{B}(n, k, M)$. An example of a Berger code is the $\mathcal{B}(5, 3, 6)$ code

$$\begin{aligned}\mathcal{B}(5,3,6) = \{&00011, 10101,\\ &11001, 00110,\\ &10010, 11100\},\end{aligned}$$

where we take 3 information bits and append 2 redundant bits representing the number of zeros in the information.

5.2 Simulation of the Berger Code

For this simulation we took the Berger code with words of length 10 which are constructed by taking all possible words of length 7 and adding a tail of 3 bits

to them. The results of the simulations of δ and δ^* are shown in Figure 5.1 with an offset of $b = 0.2$ and Figure 5.2 with an offset of $b = 0.4$.

5.3 Results of the Simulations

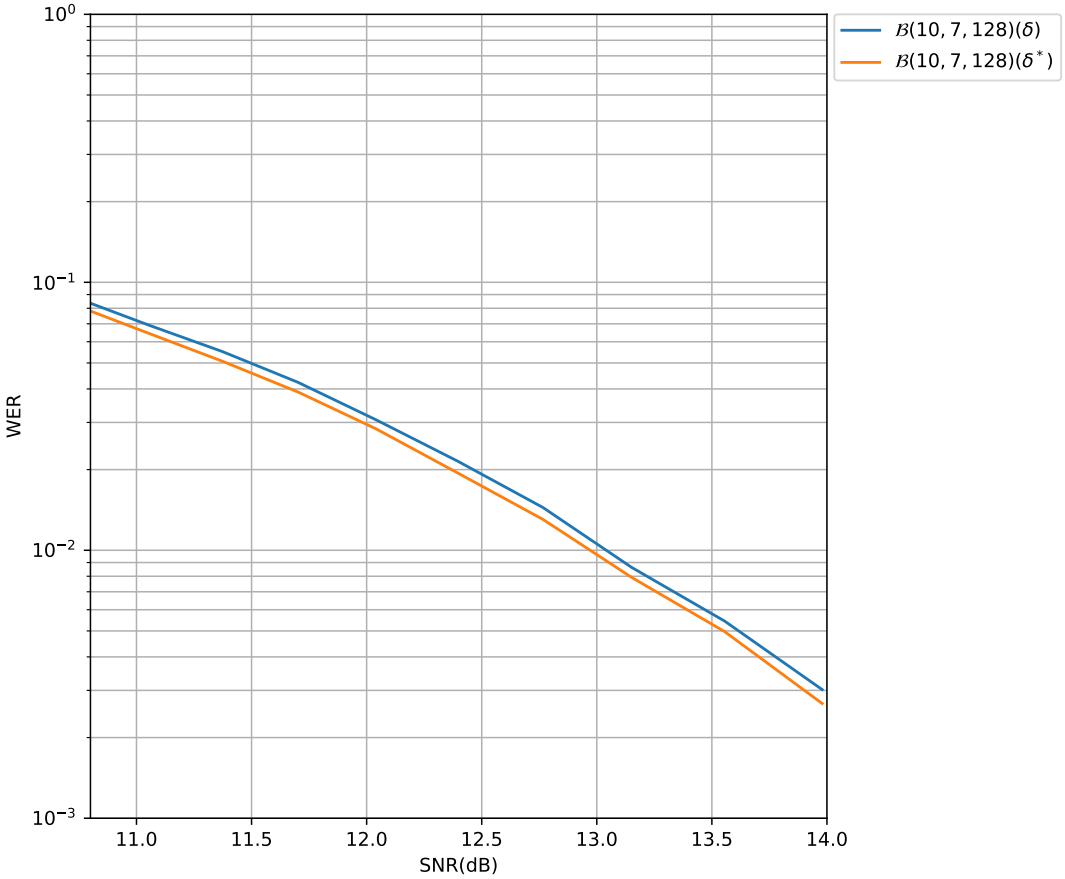

Figure 5.1: Simulation of the Berger code $\mathcal{B}(10, 7, 128)$, constructed by taking all possible codewords of length 7, with an offset of $b = 0.2$

When comparing Figure 5.1 with Figure 5.2 we can clearly see that the offset has a negative impact on δ while it has no effect on δ^*. Furthermore we observe that the Berger code suffers the same amount from noise when using either δ or δ^*. This is especially evident in Figure 5.1 where the curve for δ^* has almost the exact same behavior as the curve for δ. The main difference between the curves is that one curve lies slightly above the other, which is caused by the offset. As a matter of fact $\delta_{\min}$ for this code is the same as $\delta^*_{\min}$, which implies that when estimating the WER using (2.10) and (2.14), we have $Q(\frac{\delta_{min}}{2\sigma}) = Q(\frac{\delta^*_{min}}{2\sigma})$. Furthermore we found out that for this code $N_{\delta_{\min}} = N^*_{\delta^*_{\min}}$. Thus the WER when using δ will approximately be the same as the WER using δ^* when there is no offset, which explains why the curve for δ lies slightly above the curve for δ^* for an offset of $b = 0.2$ as δ has no immunity to offset.

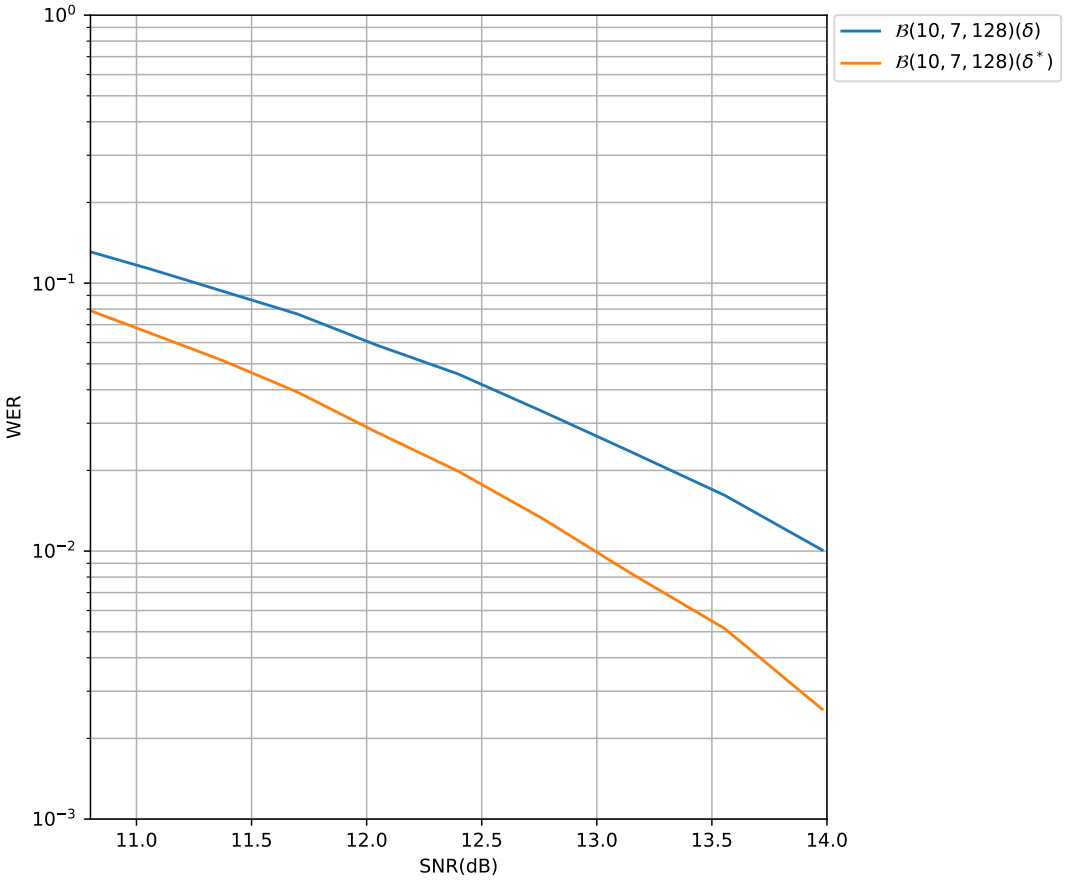

Figure 5.2: Simulation of the Berger code $\mathcal{B}(10, 7, 128)$, constructed by taking all possible codewords of length 7, with an offset of $b = 0.4$

Chapter 6

Comparison of the Codes

This chapter will be composed of two sections. In the first section we compare the results of the codes we constructed in Chapter 3, 4 and 5. In the second section, we construct a new constant weight code and a new Berger code after which we compare these new codes to the coset of the shortened Hamming code.

6.1 A first Comparison

Now that we have results for these three codes, we can start comparing their performances and behavior. But first we calculate the the code rate and redundancy of our codes following the definitions in Section 2.1. For the $\mathcal{H}_{4,\beta}[3]$ code, we have 8 information bits in a word of length 12, so we have a code rate of 0.67. To make sure that these codewords containing the information have a big enough distance between them, we append 4 bits, so a redundancy of 4, to protect the message during transmission. Following the definitions, we also calculate the code rate and redundancy for the $\mathcal{C}(7, 35, 2, 3)$ code and the $\mathcal{B}(10, 7, 128)$ code, which is presented in Table 6.1.

Code	Length (n)	Code rate	Redundancy	δ_{min}	δ^*_{min}
$\mathcal{H}_{4,\beta}[3]$	12	0.67	4	3	2.25
$\mathcal{C}(7, 35, 2, 3)$	7	0.73	1.87	2	2
$\mathcal{B}(10, 7, 128)$	10	0.7	3	2	2

Table 6.1: Length, code rate, redundancy, δ_{min} and δ^*_{min} of the $\mathcal{H}_{4,\beta}[3]$ code, $\mathcal{C}(7, 35, 2, 3)$ code and $\mathcal{B}(10, 7, 128)$ code.

In Figure 6.1 we plotted the WER for the codes we used from the previous sections while decoding with δ. From this figure we can conclude that the constant weight code performs best for all values of the SNR. While the WER's are very close to each other for smaller SNR values, for larger values the Berger code has a considerably lower WER than the coset of the shortened Hamming code. Even though $H_{4,\beta}[3]$ has the highest δ_{min}, namely $\delta_{min} = 3$, compared to the δ_{min}, ($\delta_{min} = 2$), from the Berger code and constant weight code, it

has the worst performance, due to the offset. Like we saw in section 4 and 5, the constant weight code does not suffer from any value of offset, while the Berger code suffers only a small amount for an offset of $b = 0.2$, which explains why these codes have a better performance. As for the comparison between the constant weight code and the Berger code, although they have the same δ_{min}, there are relatively more of pairs of codewords in the Berger code that have a distance of 2 to each other than in the constant weight codes. Using the theoretical expression (2.12) in the WER (2.10), we found that $N_{\delta_{\min}} = 7$ for the Berger code, while it is $N_{\delta_{\min}} = 6$ for the constant weight code.

In Figure 6.2 we compared the performance of the codes while using δ^* with an offset of $b = 0.2$. Note that the WER of these codes is now only dependent on the noise and not the offset, since δ^* does not suffer from offset. So it comes to no surprise that the $\mathcal{H}_{4,\beta}[3]$ code has the best performance as it has the highest δ^*_{min}. Furthermore it is notable that the coset of the shortened Hamming code is the only code which improves a fair amount from using δ^* with an offset of $b = 0.2$, since the constant weight code has the same performance for both distances and the Berger code improves a considerably smaller amount.

To further investigate why the Berger code using the δ distance changes very little from using δ^*, while the coset of the Hamming code changes drastically, we also simulated the codes for different offset values. Figure 6.3 shows these results, where we plotted the WER against the offset b, for a fixed SNR of 12. The graph clearly shows that the constant weight code is indeed immune to offset when using δ, like we showed in Section 4.3. Furthermore we can see that when there is no offset, the coset of the shortened Hamming code clearly performs best, while for larger values the constant weight code is a better option. If we take a closer look at the WER for an offset of $b = 0.2$, we can see that the WER of $\mathcal{H}_{4,\boldsymbol{\beta}}[3]$ has increased a lot with respect to the WER at $b = 0$, while the WER of the Berger code has barely increased compared to its WER at $b = 0$. This also explains why the curve of the Berger code using δ was only slightly higher than that of δ^* in Figure 5.1 in Section 5.3. When the offset increases to $b = 0.4$, there is indeed a more noticeable difference in the Berger code, like we also shaw in Figure 5.1.

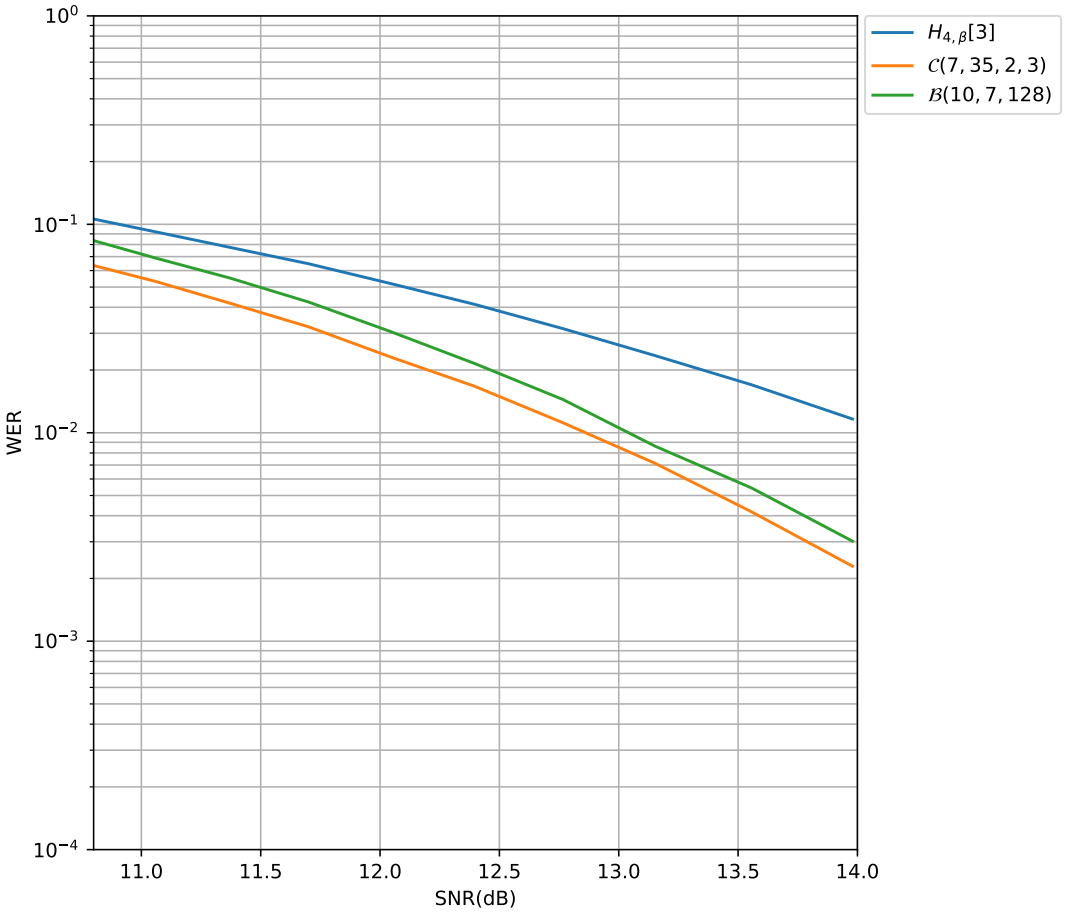

Figure 6.1: Simulation of the $\mathcal{H}_{4,\beta}[3]$ code, $\mathcal{C}(7, 35, 2, 3)$ code and $\mathcal{B}(10, 7, 128)$ code using δ with and offset of $b = 0.2$.

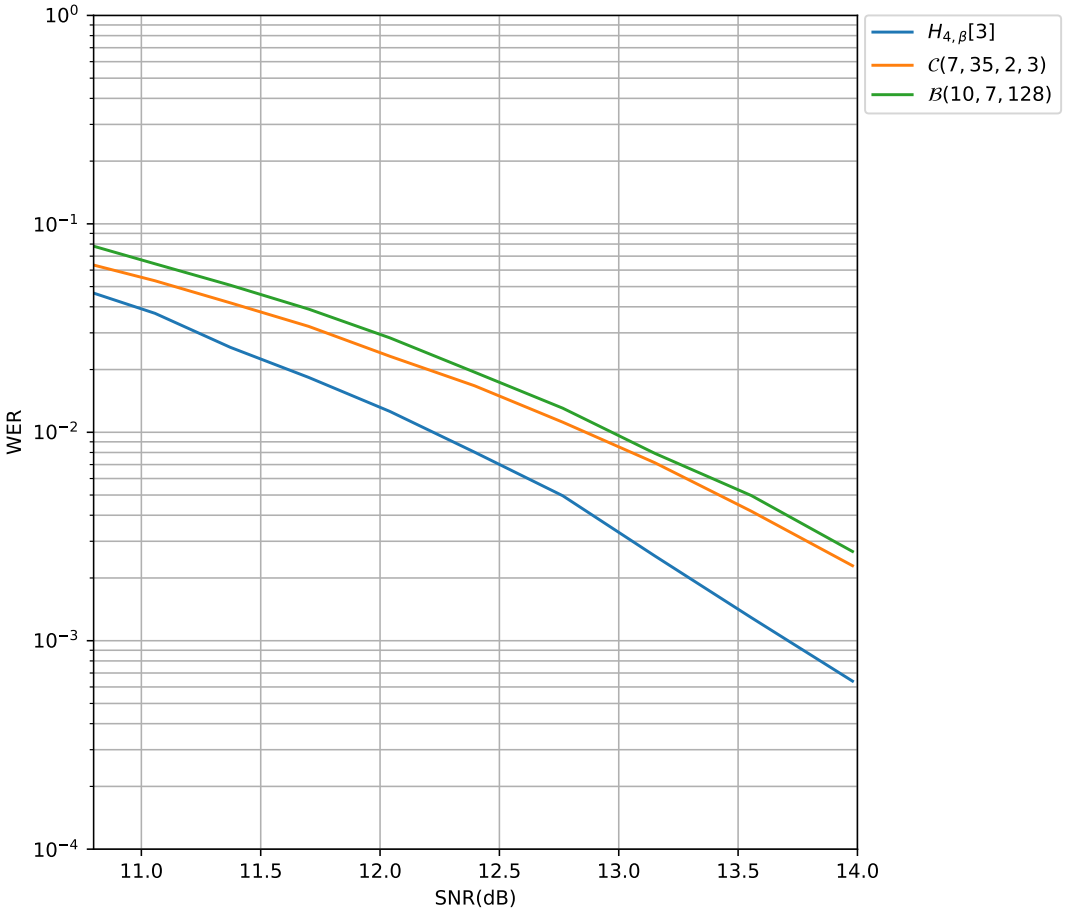

Figure 6.2: Simulation of the $\mathcal{H}_{4,\beta}[3]$ code, $\mathcal{C}(7, 35, 2, 3)$ code and $\mathcal{B}(10, 7, 128)$ code using δ^* with and offset of $b = 0.2$.

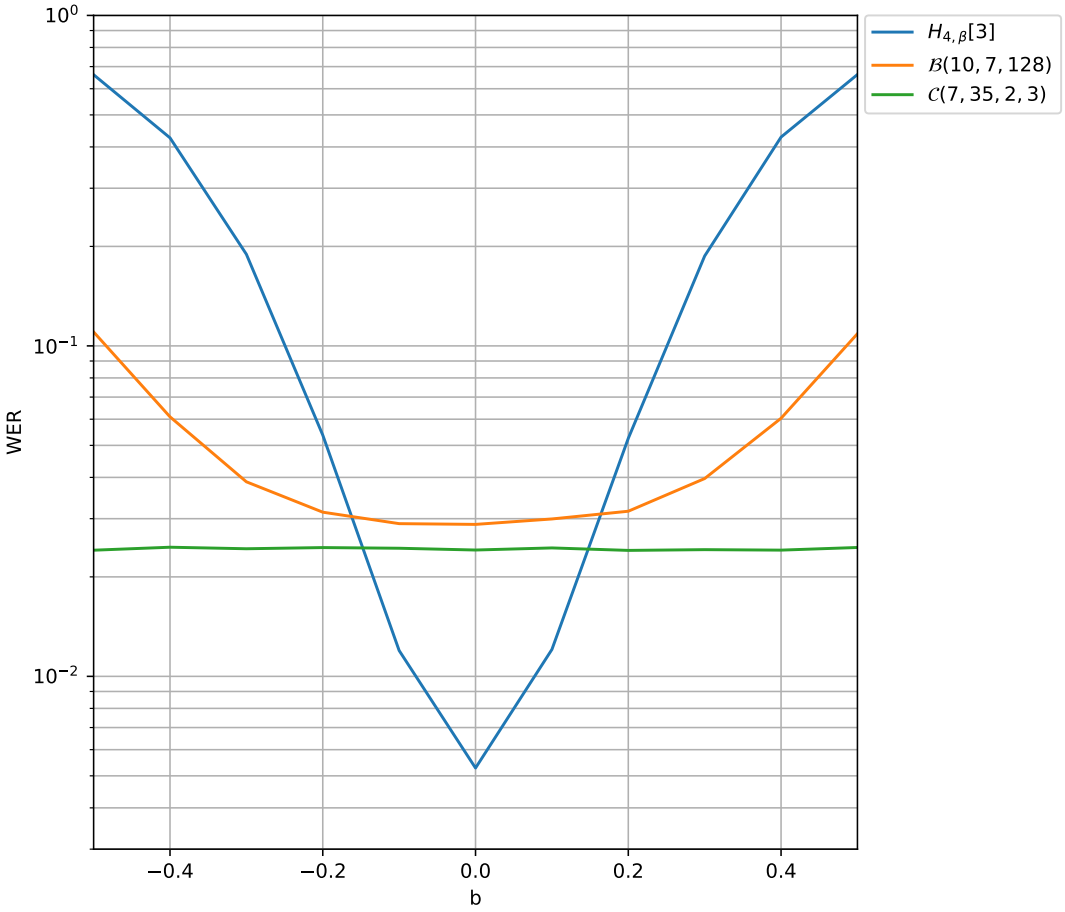

Figure 6.3: Simulation of the $\mathcal{H}_{4,\beta}[3]$ code, $\mathcal{C}(7, 35, 2, 3)$ code and $\mathcal{B}(10, 7, 128)$ code using δ with variable offset and SNR of 12.

6.2 Comparison of the new Codes

These comparisons might offer some insights on the difference in performance of these codes, however there are some unfair comparisons. For example, while the coset of the shortened Hamming code has the best performance when using $\delta^*_{\min}$, it also has the highest redundancy, so it will be more expensive to use this code, compared to the constant weight code. To make a more fair comparison, we constructed another Berger code and another constant weight code to better match the redundancy, code rate and length of the codewords of the $\mathcal{H}_{4,\beta}[3]$ code.

For the new Berger code we chose the $\mathcal{B}(12, 8, 256)$ code constructed by taking all possible words of length 8 and appending a tail of 4 bits to them. This way, it has the exact same redundancy, length and code rate as the shortened Hamming code. As for the constant weight code, we chose the $\mathcal{C}(11, 128, 2, 3)$ constructed by taking all 165 codewords of length 11 and weight 3 and removing 37 codewords. By removing the 37 words, we end up with a redundancy of 4 and a code rate of 0.64. The properties of these new codes are summarized in Table 6.2.

Code	Length (n)	Code rate	Redundancy	δ_{min}	δ^*_{min}
$\mathcal{H}_{4,\beta}[3]$	12	0.67	4	3	2.25
$\mathcal{C}(11, 128, 2, 3)$	11	0.64	4	2	2
$\mathcal{B}(12, 8, 256)$	12	0.67	4	2	2

Table 6.2: Length, code rate, redundancy, δ_{min} and δ^*_{min} of the $\mathcal{H}_{4,\beta}[3]$ code, $\mathcal{C}(11, 128, 2, 3)$ code and $\mathcal{B}(12, 8, 256)$ code.

When comparing the new codes in Figure 6.4 and 6.5 we immediately notice that both the $\mathcal{C}(11, 128, 2, 3)$ code and the $\mathcal{B}(12, 8, 256)$ perform worse than the $\mathcal{C}(7, 35, 2, 3)$ code and the $\mathcal{B}(10, 7, 128)$ code. We also note that now the Berger code has a slightly better performance than the constant weight code, despite still having the same $\delta^*_{\min}$ and $\delta_{\min}$. This again has to do with the fact that the constant weight code has relatively more codewords with the minimum distance to each other than the Berger code. Calculating $N_{\delta_{\min}}$ for both codes again, we now found $N_{\delta_{\min}} \approx 10.26$ for the constant weight code and $N_{\delta_{\min}} = 9$ for the Berger code. From these new codes we can see that it would be best to use the coset of the shortened Hamming code when using δ^* as it has the best performance of these three codes while also having the same code rate as the Berger code and a slightly better code rate than the constant weight code. However, when using δ it would be better to use $\mathcal{B}(12, 8, 256)$ or $\mathcal{C}(7, 35, 2, 3)$. Furthermore, while the Berger code does not have the best performance in Figure 6.5, it is able to detect all unidirectional errors, while $\mathcal{H}_{4,\beta}[3]$ is not.

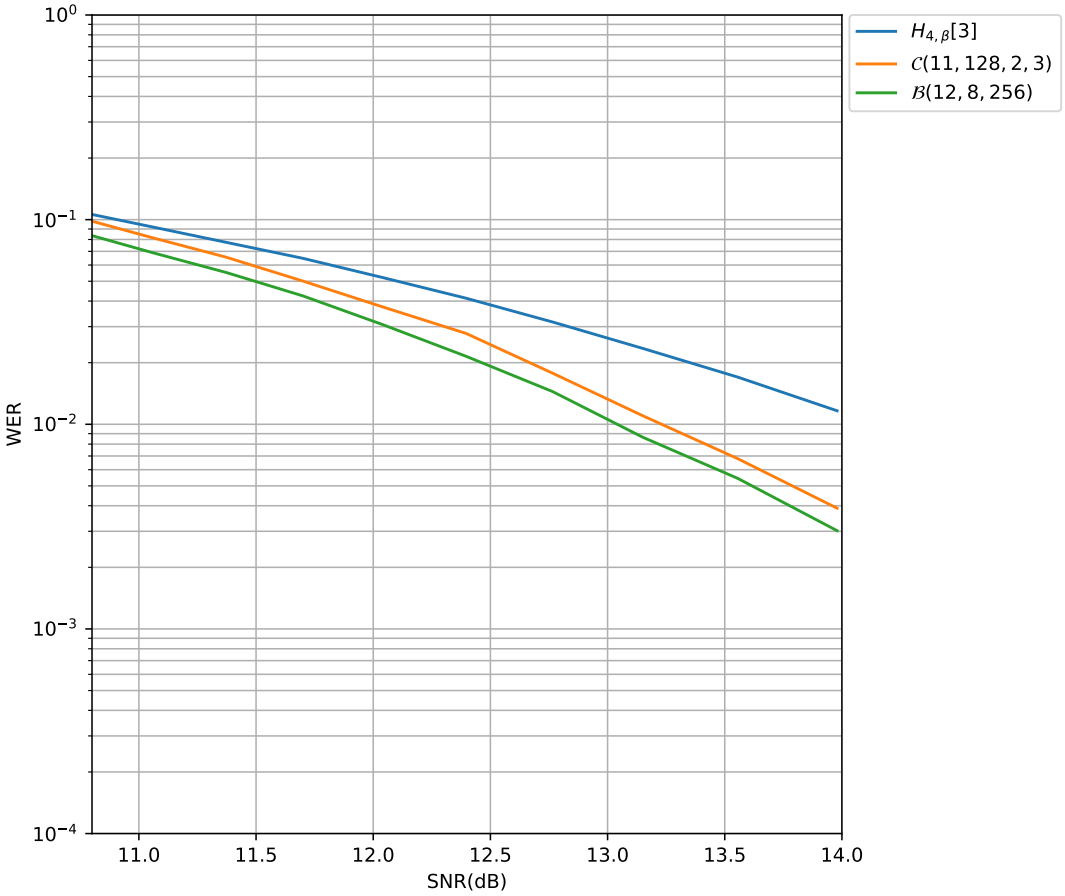

Figure 6.4: Simulation of the $\mathcal{H}_{4,\beta}[3]$, $\mathcal{C}(11, 128, 2, 3)$ and $\mathcal{B}(12, 8, 256)$ using δ with and offset of $b = 0.2$

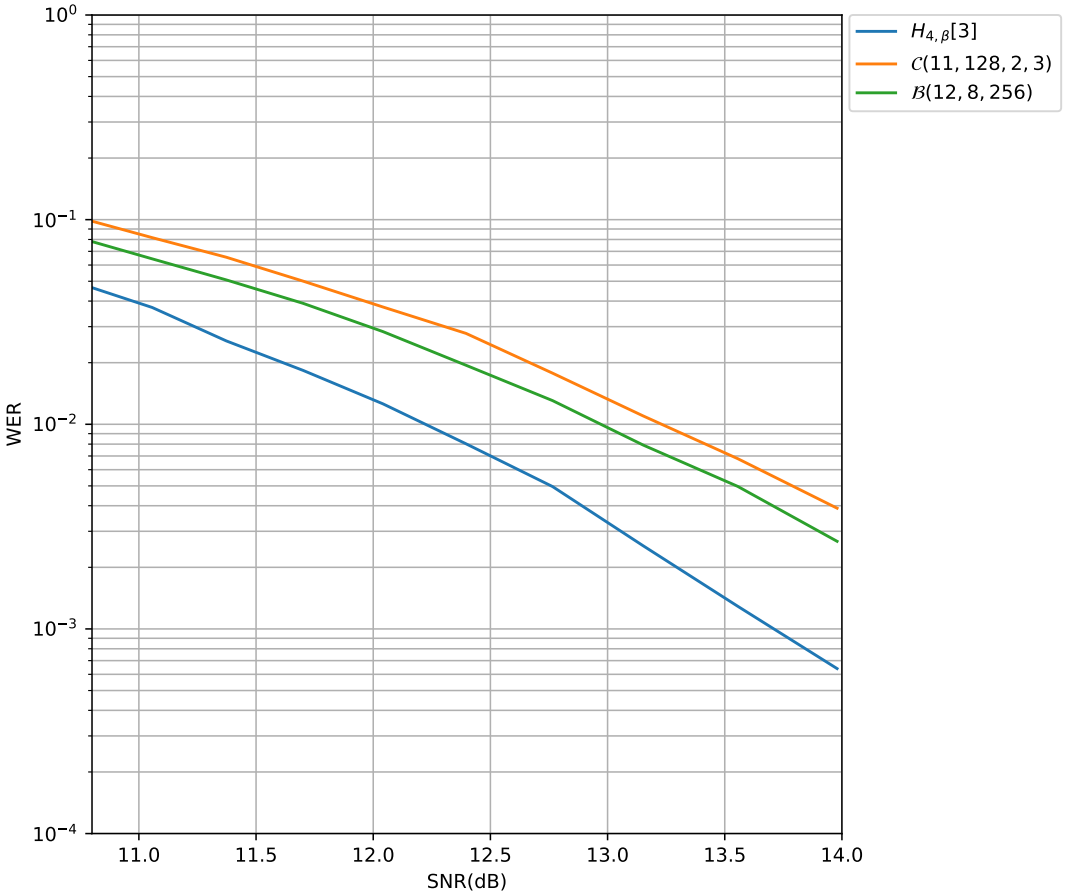

Figure 6.5: Simulation of the $\mathcal{H}_{4,\beta}[3]$, $\mathcal{C}(11, 128, 2, 3)$ and $\mathcal{B}(12, 8, 256)$ using δ^* with and offset of $b = 0.2$

Chapter 7

Conclusions and further Research

7.1 Conclusions

In this report we discussed three different codes to deal with offset and make use of the δ^* distance which has immunity to offset. Firstly, we examined the coset of the shortened Hamming code, which was able to make use of δ^* to gain offset immunity, while also maintaining a good noise performance. In the simulations of $\mathcal{H}_4[3]$ and $\mathcal{H}_{4,\boldsymbol{\beta}}[3]$, we saw that the coset indeed improves the performance of the decoder using δ^* for higher SNR, while the shortened Hamming code itself had a worse WER for a higher offset value of $b = 0.2$. Secondly, we considered the use of constant weight codes as they inherently have offset immunity. When doing simulations for the $\mathcal{C}(7, 35, 2, 3)$ code, we observed that the code has the exact same performance for both distances. This was a result of the fact that $\delta(\mathbf{u}, \mathbf{v}) = \delta^*(\mathbf{u}, \mathbf{v})$ for any pair of words $\mathbf{u}$ and $\mathbf{v}$. The third code we discussed was the Berger code, which also had a higher δ^*_{min} while also being able to detect unidirectional errors. Simulating the $\mathcal{B}(10, 7, 128)$ code, it was clear that for an offset of $b = 0.2$, the performance of the δ^* distance was just slightly better than δ. Furthermore, we saw that when there was no offset, the WER of δ would be the same as the WER of δ^*, which followed from the theoretical approximation. When we increased the offset to $b = 0.4$, there was a clear difference between the use of the Euclidean distance and the modified Euclidean distance. Finally, we compared the $\mathcal{H}_{4,\boldsymbol{\beta}}[3]$ code, the $\mathcal{C}(7, 35, 2, 3)$ code and the $\mathcal{B}(10, 7, 128)$ code. It was clear that the coset of the shortened Hamming code had the best performance when using the modified Euclidean distance, while the constant weight code had the best performance for the Euclidean distance. We argued that, while this might offer some insights, it was not a fair comparison as the redundancy and code rate differed for the codes. To make a better comparison, we introduced a new Berger code, the $\mathcal{B}(12, 8, 256)$ code, and a new constant weight code, the $\mathcal{C}(11, 128, 2, 3)$ code, to better match the code rate and redundancy. Here we observed that the coset of the shortened Hamming code still performed best, as it also has the highest $\delta^*_{\min}$, but it was still worse when using δ, since it is less resistant to an offset of $b = 0.2$.

7.2 Further Research

To conclude our report, we give some suggestions on further research that can be done on this subject. The aim of this project was to compare the different methods proposed in [8] by the means of simulations. In this project we only studied cosets of Hamming codes, a single constant weight code and the Berger codes. In [8] unions of constant weight codes are proposed and unordered codes in general. It might be useful to investigate if taking the union of different constant weight codes offers different results. Furthermore, instead of only taking Berger codes into consideration, one can study the performance of t-EC AUED codes [1], which are codes able to detect t errors in addition to all unidirectional errors. Additionally, cosets of other linear codes like the Reed Solomon code or BCH code may be interesting to study. Lastly, it can be useful to investigate the number of nearest neighbors for the codes as they might offer some insights on the behavior of the performances of the codes.

Bibliography

[1] M. Blaum and H. Van Tilborg, "On t-error correcting/all unidirectional error detecting codes," in *IEEE Transactions on Computers*, vol. 38, no. 11, pp. 1493-1501, Nov. 1989, doi: 10.1109/12.42121.

[2] B. Bose, "On unordered codes," in *IEEE Transactions on Computers*, vol. 40, no. 2, pp. 125-131, Feb. 1991, doi: 10.1109/12.73583.

[3] V. Guruswami, A. Rudra and M. Sudan, "The Fundamental Question," in *Essential Coding Theory*, n.p., 2019 [Online]. Available: https://cse.buffalo.edu/faculty/atri/courses/coding-theory/book/web-coding-book.pdf

[4] K. A. S. Immink, "Coding schemes for multi-level flash memories that are intrinsically resistant against unknown gain and/or offset using reference symbols," in *Electronics Letters*, vol. 50, no. 1, pp. 20-22, 2 January 2014, doi: 10.1049/el.2013.3558.

[5] K. A. S. Immink and J. H. Weber, "Minimum Pearson Distance Detection for Multilevel Channels With Gain and/or Offset Mismatch," in *IEEE Transactions on Information Theory*, vol. 60, no. 10, pp. 5966-5974, Oct. 2014, doi: 10.1109/TIT.2014.2342744.

[6] L. Pezza, L. G. Tallini and B. Bose, "Variable Length Unordered Codes," in *IEEE Transactions on Information Theory*, vol. 58, no. 2, pp. 548-569, Feb. 2012, doi: 10.1109/TIT.2011.2173633.

[7] F. Sala, K. A. Schouhamer Immink and L. Dolecek, "Error Control Schemes for Modern Flash Memories: Solutions for Flash deficiencies," in *IEEE Consumer Electronics Magazine*, vol. 4, no. 1, pp. 66-73, Jan. 2015, doi: 10.1109/MCE.2014.2360965.

[8] J. H. Weber, R. Bu, K. Cai and K. A. S. Immink, "Binary Block Codes for Noisy Channels with Unknown Offset," in *IEEE Transactions on Communications*, doi: 10.1109/TCOMM.2020.2986200.

[9] J. H. Weber, K. A. Schouhamer Immink and S. R. Blackburn, "Pearson Codes," in *IEEE Transactions on Information Theory*, vol. 62, no. 1, pp. 131-135, Jan. 2016, doi: 10.1109/TIT.2015.2490219.