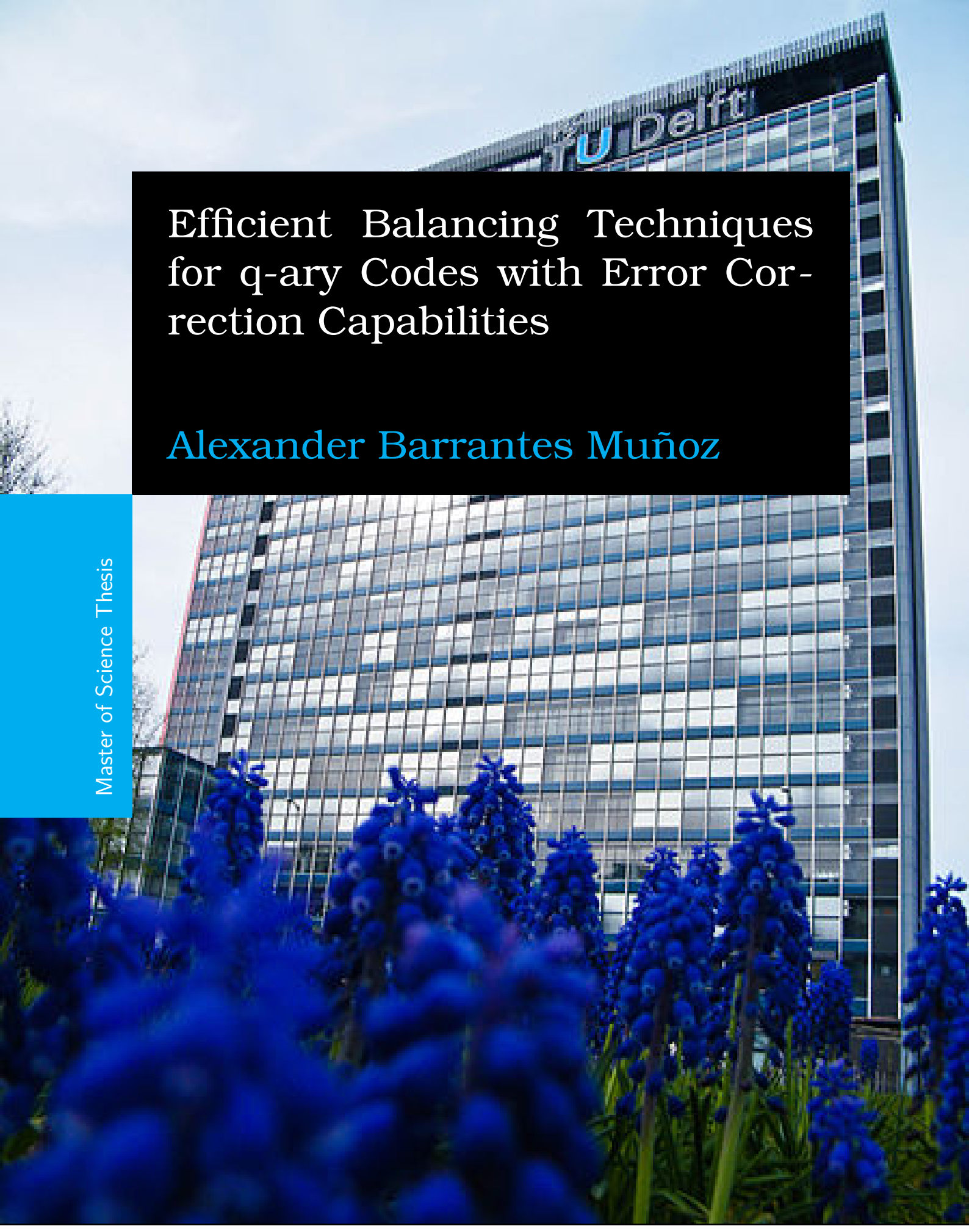

Efficient Balancing Techniques for q-ary Codes with Error Correction Capabilities

Alexander Barrantes Muñoz

Master of Science Thesis

TU Delft Delft University of Technology

Intelligent Systems (INSY), EEMCS.

Efficient Balancing Techniques for q-ary Codes with Error Correction Capabilities

Master of Science Thesis

For the degree of Master of Science in Telecommunications & Sensing Systems at Delft University of Technology

Alexander Barrantes Muñoz

December 9, 2013

Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS)
Delft University of Technology

Delft University of Technology
Department of
Intelligent Systems (insy)

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS) for acceptance a thesis entitled

Efficient Balancing Techniques for q-ary Codes with Error Correction Capabilities

by

Alexander Barrantes Muñoz

in partial fulfillment of the requirements for the degree of

Master of Science Telecommunications & Sensing Systems

Dated: December 9, 2013

Supervisor(s):

dr.ir. Jos H. Weber

Reader(s):

prof.dr.ir. Kees A. Schouhamer Immink

dr.ir. Huijuan Wang

Abstract

This document aims to explore charge balancing methods, with error correction capabilities for q-ary sequences, and to develop new techniques or to extend previous ones. The problem is approached by using analytical, as well as simulation tools.
Block codes over q-ary alphabets can be balanced under various perspectives, namely Symbol-Balanced (SB), Charge-Balanced (CB), and Polarity-Balanced (PB) Codes. A q-ary Code is CB if the sum of its symbols equals zero, for the alphabet forms $\{-(q-1)/2, ..., -2, -1, 0, +1, +2, ..., +(q-1)/2\}$ if q is odd, or $\{-(q-1), ..., -3, -1, +1, +3, ..., +(q-1)\}$ if q is even. The alphabet can also be expressed as $\{0, 1, 2, ..., q-1\}$ for practical purposes, but the sum of CB Code symbols is not longer zero in this case. For instance, a binary CB sequence has the same number of zeros and ones.
Knuth presented an efficient method to design binary CB Codes, derived by mapping any binary sequence of even length, via a reversible algorithm. Knuth's method inverts the first z sequence bits until achieving balance, then such index z is communicated to the decoder, through a short balanced prefix.
The binary Knuth-like balancing scheme with error-correcting capabilities, exhibited by Weber, Immink and Ferreira, is extended to q-ary Codes. This proposal is very broad, since any Error Correcting Codes (ECC) can be applied to the payload. Different error-correction levels over the prefix and the payload are allowed. A method to reduce the iterations, when searching balancing indices is newly presented, as an improvement of the previous algorithm.
q-ary Knuth-based Codes are suitable for applications with large codewords, due to their low memory resources use and complexity. However, their redundancy doubles that of the full balanced set of codewords. Therefore, three previous approaches to improve performance are analyzed for q-ary Codes.
Another q-ary CB Codes with error-correcting capabilities are introduced, namely Concatenated Codes. They are extensions from the binary scheme by van Tilborg and Blaum, in which balanced blocks are considered as symbols over an alphabet, to build ECC over that alphabet. Concatenated Codes restrict the runlength of a bit, and the Digital Sum Variation (DSV) to reasonable bounds (desirable features).
The comparison between binary Concatenated and Knuth-based Codes, with double error correction capabilities is carried on, evaluating their DSV's and Code Rates.

Contents

List of Figures

List of Tables

Preface

This dissertation is the final requirement, for the accomplishment of the Msc. in Telecommunications & Sensing Systems. I always had particular interest in communications, back in my bachelor at the University of Costa Rica. My motivation led me to apply to TU Delft, due to its remarkable reputation, as well as that of the Netherlands, in regards to the high involvement in technology and research.
After being gladly admitted as student, I realized about the existence of ECC, during the first classes of the program. I had special fascination for the courses instructed by dr.ir. Jos H. Weber. I stopped by his office one day, and asked for a related topic to research. He kindly accepted to be my thesis supervisor, and introduced Knuth balancing method to me, subject matter which I did not hesitate to accept.
Besides the potential applicability of q-ary CB Codes, studies about them are relatively recent. Thus, exploring this topic brings high expectations, to contribute with useful findings for future implementations...

Acknowledgements

Allow me to express my gratitude to dr.ir. Jos H. Weber, for his guidance during the development of this dissertation. This work is an achievement from both of us, without a doubt. My supervisor was very modest and patient with me, despite the differences in the education system, with which I am familiar. I hope my ideas partially payoff his huge investment of time, along this challenging process.
I would like to thank my dear godmother Kathleen, for her constant support, advices full of wisdom and willingness to listen to me. Friends are definitely the family members that one voluntarily chooses.
I am thankful to my three best friends in Delft, Veronika, Aaron and Felipe. They helped to overcome my obstacles within this experience. I am glad to be surrounded by such good people, we laughed together uncountable times.
I am graceful to my sister Vanessa, who plays an important role by holding my family together. Her unconditionally love for me is very tangible.
Finally, I would like to thank anyone who helped me make this journey possible.

Delft, University of Technology — Alexander Barrantes Muñoz
December 9, 2013

I dedicate this document to my beloved father Jose Luis, for his support along my master experience, and implication in the success of my life. He began to work at an early age, and such effort has made my biggest dream come true (study abroad). Consequently, he is a big part of this accomplishment. I will never forget his favorite quote, which also describes his lifestyle:

> "Persistence and Perseverance are the keys to success."
>
> — *My Dad*

Chapter 1

Introduction

Due to exponential growth in technology that our era continuously experiences, new devices give birth to more rigorous requirements. There is a constant fight between performance and energy efficiency. Both concepts are continuously evolving and being optimized over previous techniques. When it comes to communications, the goal is to transmit using the least energy possible, with correctable errors or none at all. Consequently, the lower the energy in transmission, the greater the number of errors that take place along the medium. Since power-saving features aim to turn energy resources, into values close to the channel noise floor. This fact is reflected on the overwhelming market competition, among the leading technology companies, when offering their innovations. Consumers develop new needs, and have higher expectations of products every day. For instance, it is common to complain when a smartphone battery lasts only one day, even though one has made plenty use of the phone features, such as GPS and wireless connectivity.
Error control to improve the system performance is required, as solution of the problem above. Error Correcting Codes (ECC) play a fundamental role, pursuing the balance in the coexistence between performance and energy efficiency.
We start by explaining the typical communication taxonomy, in order to provide an overview of ECC. The basic communication block diagram from [1] is shown in the Figure 1-1. The cycle is bidirectional, lasts until the end of transmission, and is initiated by the *Source*, which originates the information, e.g. a person (information: voice), computer server, data base. The *Input Device* is the first interface, encharged of entering the data into the system, e.g. mobile phone. The *Data Reduction*, as the term implies, discards all negligible information for the receiver. The data is converted to another format and/or compressed, within the *Source Coding*. Protection is added to the information via secured algorithms, during the *Encryption* stage, the outcome is meant to be decoded only at the chosen destination. Error control is applied over the data, by *Channel Coding* techniques. The scope of this research is precisely related to this step. The aim of Error Correcting Codes is to reinforce information, with protection against random errors, that occur through the medium.

The analog signal that goes throughout the medium, is generated at the *Modulation* stage. Every output from the encoder is transformed into an analog waveform. There are several

Source		Sink
Input Device		Output Device
Data Reduction		Data Reconstruction
Source Codding		Source Decoding
Encryption		Decryption
Channel Coding		Channel Decoding
Modulation		Demodulation
Channel		

Figure 1-1: Typical Communication Block Diagram.

modulation methods, based on different principles such as signal frequency, amplitude or phase shifts, e.g. BPSK, QPSK, QAM, ASK, FSK, PSK. The modulation technique can be also chosen, according to the number of symbols, required to fully express the message. For instance, the quantization of a signal modulated using PAM, is shown in the Figure 1-2, this process is carried on by Pulse Code Modulation, to encode voice for VoIP calls [2].
Finally, the path taken by the data, during transmission or storage is named *Channel or Medium*. The channel can be physical such as cable, telephone lines and optic-fiber, or it can be the free space. Wireless connectivity takes place through the free space, such as mobile cellular telephony, radio frequency, telemetry, microwave and satellite links. The medium can also be digital storage media, such as core and semiconductor memories, DVD/CD's, hard-disk files, optical memory units [3].
The reverse operations are conducted at the destination side, right side of the Figure 1-1.

1-1 General Definitions and Terminology

Two types of errors can be found: random and burst. The former errors occur randomly and independently. On the contrary, the latter errors are codependent, caused by the memory of the medium, e.g. the scratches on a CD's surface. A well-known technique to combat burst of errors is called interleaving, which will not be explained further since from here on only reference to random errors will be made.
Every single Error Correcting Code is based on the same basic principle: *Redundancy*, which is the additional data added to the raw information [4], denoted by the letter r. The redundant elements are not necessarily correlated to the information directly. The outcome from the encoder is named *codeword*, denoted as $\mathbf{x}$, and it is unique for every input message $\mathbf{u}$. The redundancy causes direct impact over the code rate R, decreasing the code efficiency in exchange for robustness (more symbols are encoded).
There are two types of codes, classified by the way that the information is processed (encoded/decoded). The first kind is *Convolutional Codes*, in which the information is seen as a continuous stream of elements. The data is encoded element-by-element, using shift registers. The current set of elements dependents on the previous input. The most popular decoding algorithm is called Viterbi.
The second kind is *Block Codes*, in which the information is independently encoded, in blocks of n elements (codewords), and processed on a block-by-block basis.
The elements or symbols of sequences, either messages or codewords, can take values from the

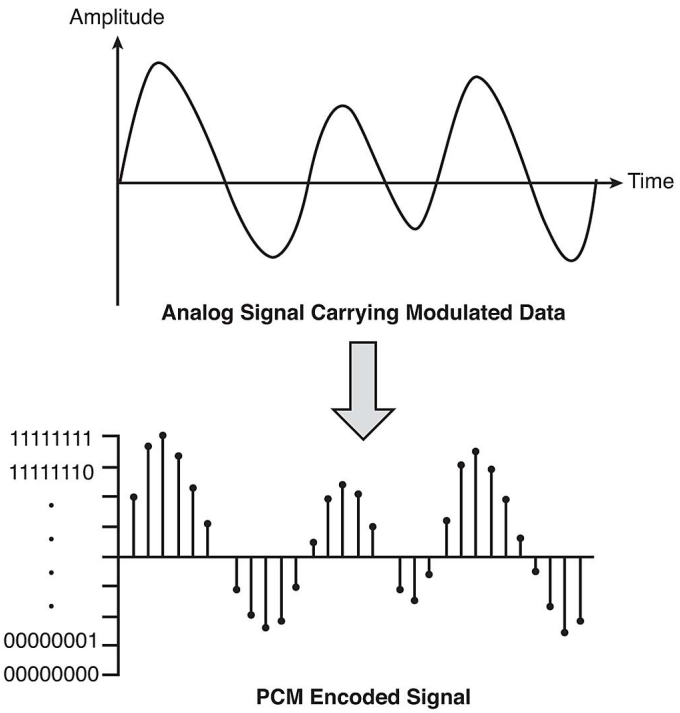

Figure 1-2: An analog signal conversion to an equivalent digital representation, via Pulse Code Modulation.

set $\{0, 1, 2, ..., q-1\}$, established by their alphabet $\mathcal{A}_q$, e.g. $q = 2$ for binary, thus $\{0, 1\} \in \mathcal{A}_2$, these symbols are broadly known as bits. The symbols are real representations of physical voltage levels, with different polarity.

Let the Hamming distance between two sequences $\mathbf{x}_0, \mathbf{x}_1 \in \mathcal{A}_q$ of length n, $d(\mathbf{x}_0, \mathbf{x}_1)$, be the number of unitary coordinates in which $\mathbf{x}_0$ and $\mathbf{x}_1$ differ. This interpretation takes into account the symbol positions and magnitudes, from alphabets larger than binary. For instance, the sequences $\mathbf{x}_0 = 01222$ and $\mathbf{x}_1 = 01100 \in \mathcal{A}_3$, differ in the three last positions by 1, 2 and 2 respectively. Thus, the distance between them is calculated as $d = 1 + 2 + 2 = 5$. The hamming distance is directly correlated to the number of errors t, that the block code can correct, such that $d \geq 2t + 1$.

The code rate is defined by $R = \log_q(M)/n$, whose values are in the range $[0, 1]$, where M is the number of valid codewords. The redundancy can be expressed as $r = n(1 - R)$. The standard notation for q-ary block codes is $(n, k, d)_q$. Where d, n and k are the minimum hamming distance of the code, the codeword and the message lengths respectively. The full set or total number of messages $\in \mathbb{F}_q^k$ is $M = q^k$, as well as the number of codewords from the block code, thus $R = k/n = k/(k + r)$ and $r = n - k$. Shortening techniques can be applied over the block code, if a shorter codeword length is wanted, such that $(n-a, k-a, d)_q$, where $1 \leq a \leq k - 1 \; \forall \; a \in \mathbb{Z}^+$.

Codes can operate under two modes: Forward Error Correction (FEC) and Automatic Repeat Request (ARQ). None error is allowed in the ARQ mode, retransmission of the codeword is requested when it happens. This mode requires less complexity, but it is very vulnerable to noisy channels, thus it takes longer to transmit. On the other hand, ARQ is very reliable,

which makes it suitable for applications with high standards of accuracy, e.g. internet banking. The FEC mode is based on ECC, thus it requires higher encoding/decoding complexity. However, its transmission time frame is predictable, and it is used in many applications.
There are different types of block codes, nevertheless linear codes are mostly used, given the simplicity of their algebraic structure, description, encoding and decoding. Encoders can be easily built using D flip-flops, exclusive-OR, and AND gates in digital circuitry [5].

1-2 Literature Review

Many applications from the digital world could upgrade their current deployments, if there was a way to balance the information sequences being in use. Those sequences can be of different nature, go through distinct channels, and their balancing can achieve diverse goals related to the challenges that face. These information vectors are modeled by q-ary block codes, independently of the purpose, then balanced according to particular interests. Concretely speaking, three classes of balanced codes and one combination of them, have been theoretically identified in [6]: Symbol-Balanced (SB), Charge-Balanced (CB), Polarity-Balanced (PB) and Charged and Polarity-Balanced (CPB) Codes.
In order to illustrate definitions with examples, assume some series of codes with variable length, to belong to an alphabet $\{-3, -2, -1, 0, 1, 2, 3\} \in \mathcal{A}_7$. A code is SB if all its alphabet symbols have the same incidence, e.g. sequence $(3, -1, 0, 2, -2, -3, 1)$. When the summation of the symbols from every codeword results zero, the code is said to be CB, e.g. sequence $(2, -3, 0, 1, 2, -2)$. If a code has as many positive symbols as those negative in each codeword, it is said to be PB. This type lacks of practical applicability, in q-ary codes at this moment, e.g. sequence $(-1, -2, 0, -3, 2, 2, 2)$. By CPB is meant those codes which are both PB and CB, e.g. sequence $(-1, 1, -1, 1)$. In general, balanced codes are found in optical and magnetic storage, wired transmissions, asymmetric/unidirectional error detection/correction and noise reduction in VLSI Systems [7].
In 1986, D. Knuth published an article implementing a binary coding scheme, in which the incidence of zeros and ones within codewords is equal, for efficient encoding and decoding purposes [8]. Knuth only focuses on binary CB codes, nonetheless his assumptions can also be extended to q-ary CB codes. He proves that any sequence can be balanced, by complementing z number of its elements consecutively, where $0 \leq z < n$, regardless the sequence's side from where their inversion starts (right or left). The scheme associates p parity elements, to depict the values of z, forming a balanced codeword. The assignment of the entries of z is carried on by look-up tables, and it can be customized. Such length p should be insignificant in comparison with the length n, to cause minimal impact over the code's efficiency i.e. $p << n$. The index z is extracted from the p-parity elements at the decoder's side. Then it is used to complement the symbols back, leading to the original sequence.
This article has inspired this dissertation and originated some other research works, such as [9], [10] and [11], presenting a balancing scheme with error-correcting capabilities, and several encoding approaches, to optimize the efficiency of Knuth-like codes.
Knuth's method also settled down the foundations for q-ary coding schemes, from some state of the art research works. For instance, balancing techniques with parallel decoding [7], or prefixless balanced codes with ECC [12]. In general, Knuth-based codes are characterized by their simple decoding, low complexity and memory resources usage, ideal for applications

with long codewords. Nevertheless, their redundancy is twice greater than that of the full balanced set, unless additional implementations are combined. Such downside is overcome by other methods, in exchange for complexity, e.g. efficient q-ary immutable codes [13].
An alternative binary code construction is proposed by van Tilborg and Blaum in [14], with prefix-less balanced codewords, composed by the concatenation of balanced blocks, called Concatenated Codes within this text. They introduced two precious properties: the restriction of the runlength of a bit, and the Digital Sum Variation (DSV), to reasonable thresholds.

1-3 Problem Statement

Studies about non-binary charge balancing methods are relatively recent. Therefore, q-ary CB Codes with error correction capabilities are still scarce in literature. Additionally, these codes have potential engineering applicability in the future. For instance, CB Codes are suitable for DC-free applications and PAM-related implementations.
This document aims to explore charge balancing methods, with error correction capabilities for q-ary sequences, and to develop new techniques or to extend previous ones. The topic problem is approached using analytical as well as simulation tools. The data generated from the software calculations are carefully examined. The main goal is to find out tendencies or correlations, by observing the data behavior.

1-4 Scope

Useful contributions in literature for q-ary CB Codes are highly expected, due to the recent studies in this area. This aspect mainly constitutes the motivation of this dissertation.
This research work involves the study of ECC, the Knuth's balancing method over q-ary codes, as well as the redundancy, and binary balanced coding schemes in existence.
The theory of block codes, with focus on q-ary Hamming and binary BCH codes, is firstly consulted. Then, the basics of Knuth's method applied to q-ary are covered. The binary error correction scheme from [7] is studied, and adapted to q-ary codes. Matlab calculations are performed, to find the maximum number of vectors in the balanced set that allows error correction, to depict prefixes (non-optimal). Thus, balanced block codes are built using Hamming encoder, for some prime numbers of q. The entropy expressions in [9] and [10] are adjusted to q-ary codes, from the Transmitted Index, Auxiliary Information Encoding and A Better Variable-Length-Prefix Construction.
The binary Concatenated code constructions from [14] are reviewed. After that, the feasibility of four configurations is examined for several values of q. Code constructions with double error correction capabilities are proposed for each configuration, added by q-ary Hamming encoder. Some tables with typical parameters are generated using Matlab tools.
BCH Codes are employed in binary Knuth-based Codes, to increase correction capabilities over the data up to two errors. Then, they are compared with the Concatenated binary codes. Characteristics such as code rates and DSV from both codes are evaluated. Conclusions are drawn based on the results, derived from the data inputs in some critical cases.

1-5 Limitations

Various limitations are encountered, when performing Matlab calculations. The biggest obstacle of this tool, is the long execution when handling large numbers. Entropies and other computations are only conducted for small values of q, and short sequence lengths. Thus, practical outcomes for real applications are inconclusive or infeasible. For instance, some combinatorial expressions are evidently required, to depict variables such as the transmitted index, and the balancing positions in a full set. The identification of some attributes from q-ary codes is outlined for future research.
Sources with the bounds for q-ary constant weight codes are inexistent, thus Matlab routines are implemented to build the prefix, in q-ary Knuth-based Codes. A reductive technique for the number of iterations when searching the balancing index is presented, as an improvement of the existing algorithm.
Matlab toolboxes only offer block codes over binary finite fields. Block code routines over higher values of q are needed, but their development is out of this thesis scope. Nevertheless, q-ary Hamming encoder/decoder is implemented due to its simple algorithm. Consequently, the comparison between Concatenated and Knuth-based Codes is constrained to binary, and double error correction capabilities.

1-6 Dissertation Outline and Contributions

The content overview of this document is broken down as follows:

- **Chapter 2, Fundamentals of Block Codes**: Basic concepts for the construction of block codes over finite fields are presented, as well as extension fields. The background information for the implementation of q-ary Hamming Codes, and a general understanding of binary BCH Codes is given.
- **Chapter 3, Knuth-based Codes**: The fundamentals of Knuth's balancing method applied to q-ary codes are introduced. A previous binary error correction scheme is extended to q-ary codes. A new method to reduce the iterations when searching balancing indices is proposed. The redundancy of q-ary Knuth-based codes is analyzed for the Transmitted Index, Auxiliary Information Encoding and A Better Variable-Length-Prefix Construction, approaches previously conducted for binary codes.
- **Chapter 4, Concatenated Codes**: The previous binary Concatenated Code construction is extended to q-ary codes. Four similar code constructions are exhibited.
- **Chapter 5, Binary Codes Comparison**: The comparison between Knuth-based and Concatenated Binary codes is carried on. Evaluating characteristics from both codes, such as code rates and DSV.
- **Chapter 6, Conclusions and Recommendations for Future Research**: The summary of the results is drawn, and potential future research is suggested.

The contributions of this research are listed as follows:

- The Matlab implementation of Hamming Code encoder and decoder for prime numbers of q.
- The extension of a q-ary Knuth-based coding scheme with error correction capabilities.
- The introduction of a new method to reduce the iterations when searching balancing indices.
- The identification of characteristics from q-ary Codes useful for the approaches: Transmitted Index, Auxiliary Information Encoding and A Better Variable-Length-Prefix Construction. Such properties are used to extend entropy expressions to q-ary codes.
- The adaptation of four Concatenated Code constructions to q-ary Codes.
- The comparison of the Code Rates and the DSV between Knuth-based and Concatenated Binary Codes.

Chapter 2

Fundamentals of Block Codes

Block codes can be implemented over different alphabets $\mathcal{A}_q$. However, operations such as multiplication and addition have convenient properties when q is a prime power. These features are useful for the development of practical encoding and decoding algorithms. Consequently, some block code constructions are analyzed in these cases, to be used in the next chapters.
Several linear codes are well-known in literature, among the most popular of them are found: Hamming Codes, and Cyclic Codes such as Bose, Ray-Chaudhuri and Hocquenghem (BCH), both defined over GF(q)[1]. The former codes are reviewed along this section for prime numbers of q, given their easy implementation as in [15]. The latter codes are studied for binary alphabet, due to their multiple error correction capabilities. The knowledge needed for an understanding of them is briefly examined as well.

2-1 Basics of Error Correcting Codes (ECC)

Roughly speaking for matrix representation, messages are encoded by the Generator Matrix G of the Linear Code $\mathcal{C}$. Then errors are corrected via the Parity-Check Matrix H. The construction of these matrices depends on the particular block code chosen. The codeword $\mathbf{x} \in \mathcal{C}$ is the product from the matrix multiplication:

$$\mathbf{x} = \mathbf{u} * G \tag{2-1}$$

Where G of $\mathcal{C}$ has dimensions k-by-n. At the decoder side, the Parity-Check Matrix H of $\mathcal{C}^{\perp}$ whose size is r-by-n must meet:

$$\mathcal{C} = \{\mathbf{x} \in \mathbb{F}_q^n | H * \mathbf{x}^T = 0\} \tag{2-2}$$

[1] Galois Field (GF): in honor of the French mathematician Pierre Galois.

The product from the operation $H * \mathbf{x}^T$ is not zero if any error occurs, and it is called *syndrome*. Its formal definition is: "the syndrome is the linear transformation of the error vector introduced in the channel" [4]. Expressing these words in algebraic terms, let $\mathbf{e}$ and $\mathbf{y}$ be the error vector and the received codeword, such that $\mathbf{y} = \mathbf{x} + \mathbf{e}$. Replacing $\mathbf{x}$ with $\mathbf{y}$ in the Expression 2-2, then solving as $H * \mathbf{y}^T = H * (\mathbf{x} + \mathbf{e})^T = \mathbf{0} + H * \mathbf{e}^T = H * \mathbf{e}^T$, the syndrome $\mathbf{s}$ is specified by:

$$\mathbf{s} = H * \mathbf{e}^T \tag{2-3}$$

In some codes, $\mathbf{y}$ is compared with the elements from the codebook of $\mathcal{C}$, whose distances differ by d or less. Then $\mathbf{y}$ is replaced by the closest of them, which turns out impractical for large values of q^n. For this reason, techniques with look-up tables of q^{n-k} entries are more attractive, such as syndrome decoding.
Block codes can be also expressed as polynomials. Let $u(x)$ and $g(x)$ be the information and the generator polynomial of the code (n, k, d) over $GF(q)$, such that $u(x) = u_0 + u_1 x + ... + u_{k-1}x^{k-1}$ and $g(x) = g_0 + g_1 x + ... + g_{n-k}x^{n-k}$ with $u_i, g_i \in GF(q)$. Hence, the code polynomial is obtained as:

$$c(x) = u(x) * g(x) = c_0 + c_1 x + ... + c_{n-1}x^{n-1} \ \ \forall \ \ c_i \in GF(q) \tag{2-4}$$

The term *GF(q)* is further explained, messages and codewords have forms $\mathbf{u} = u_0 u_1 u_2 ... u_{k-1} \in \mathbb{F}_q^k$ and $\mathbf{x} = c_0 c_1 c_2 ... c_{n-1} \in \mathbb{F}_q^n$. Similar definitions as those from matrix representation are derived for the parity-check, syndrome, error and received codeword polynomial, i.e. $h(x), s(x), e(x)$ and $y(x)$.

2-2 Block Code Construction over GF(q) with q prime

Some basic concepts and definitions from [16] are first recalled. Two modulo operations can be applied over the elements from a set: multiplication and addition. Assume a, b, c to belong to the set G' for simplicity, the most common conditions that these operations can meet are:

1. Commutativity under multiplication: $a * b = b * a$
2. Commutativity under addition: $a + b = b + a$.
3. Identity: $a * \iota = a$, where ι is the identity.
4. Inverse: $a * a^{-1} = \iota \ \exists \ a^{-1} \in G'$.
5. Associativity under multiplication: $(a * b) * c = a * (b * c)$.
6. Distributivity: $a * (b + c) = (a * b) + (a * c)$.

Let the group G be a set defined by multiplication, whose elements are either finite or infinite, which meets the properties 3, 4 and 5 (Identity, Inverse and Associativity). A group is called "Abelian", if it also fulfills the condition 1 (Commutativity under multiplication). The multiplication of any two elements generates another element of the group, feature named

closure. The Table 2-1 c) shows the products in $GF(5)$, illustrating such characteristic.
Let the finite field $GF(q)$ or $\mathbb{F}_q$ be a group of elements which meets the conditions $1, 2, 3, 4$ and 6 (Commutativity under non-null multiplication and addition, Identity, Inverse and Distributivity) over the ring $\mathbb{Z}_q$ (modulo q). A finite field has finite number of elements, where $q > 1$ is a prime number or a power of a prime, i.e. $q = p_r^m \; \forall \; m \geq 1$. All the elements from the finite field have an order, which indicates how many times an element is multiplied by itself, to lead 1 in modulo q. The element α with the order equal to $q-1$ is named primitive. Every single non-zero element from a finite field, can be expressed as powers of primitive elements. E.g. given the non-zero alphabet symbols $\{1, 2, 3, 4, 5, 6\} \in \mathcal{A}_7$. The order 3 of the element $2 \in \mathcal{A}_7$, is calculated as $2 * 2 * 2 = (8) mod_7 = 1$. Likewise, the other orders of the elements $\in \mathcal{A}_7$ are displayed in the Table 2-1 a). The elements 3 and 5 are primitives in $GF(7)$, since they have orders of $q - 1 = 7 - 1 = 6$. Thus, if $\alpha = 3$ all the alphabet symbols $\in \mathcal{A}_7$ can be expressed as powers of 3, i.e. $\{3^6, 3^2, 3^1, 3^4, 3^5, 3^3\} \in GF(7)$.

Element in GF(7)	Order
1	1
2	3
3	6
4	3
5	6
6	2

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

*	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Table 2-1: a) Element orders in $GF(7)$, b) Addition over $GF(5)$ and c) Multiplication over $GF(5)$.

Block Codes over $GF(q)$ with $q = p_r$ are built straight-forward, by performing operations in modulo-q, using the symbols $\{0, 1, 2..., q-1\}$. However if $q = p_r^m \; \forall \; m > 1$, the construction of codes over $GF(q)$ is more complex, such fields are referred to as extensions.

2-2-1 q-ary Hamming Codes

As stated before, Hamming Codes are defined over finite fields $GF(q)$. A common notation for these Codes is $Ham(r, q)$, their codewords and data lengths n & k must meet the following requirements:

$$\begin{aligned} & n \leq (q^r - 1)/(q - 1) \\ & r = n - k \geq 2 \\ & d = 3 \end{aligned} \tag{2-5}$$

Where r is the code redundancy, d is the hamming distance and q is a prime number.
Some codeword lengths n's for different values of q and r are listed on the Table 2-2. Moreover, the binary code lengths are modified, to apply balance. Thus, shortening over these codes is applied to achieve this purpose, i.e. $(n-1, k-1, d)$.

In general, a block code is built by adding elements besides the information. Such additional elements pretend to set up the minimum distance d, to secure error correction capabilities. This redundancy is related to the data via algebraic operations.

r	q																	
	2	3	5	7	11	13	17	19	23	29	31	37	41	43	47	53	59	61
2		4	6	8	12	14	18	20	24	30	32	38	42	44	48	54	60	62
3	6	13	31	57	133	183	307	381	553	871	993	1407	1723	1893	2257	2863	3541	3783
4	14	40	156	400	1464	2380												
5	30	121	781	2801														
6	62	364	3906															
7	126	1093																
8	254																	

Table 2-2: Maximum Hamming codeword length n, for different values of r and q.

The mathematical operations involved in the Hamming Codes are defined by the *Parity-Check Equations*. Such operations establish the correlation between information and redundancy, and they are constrained by the linear independence among themselves. Recalling the same encoding principle as in the Equation 2-1. The Systematic Generator Matrix $\mathbf{G}_{r,q} = [\mathbf{I}_k|\mathbf{P}] : k$ x n is composed by two parts: the Identity Matrix of size k, $\mathbf{I}_k$, and the Parity Matrix of size k-by-r, $\mathbf{P}$. The latter matrix contains r parity-check equations, one per parity/redundant element.
For instance, codewords have the form $\mathbf{x} = x_0x_1x_2x_3x_4x_5$ in case $Ham(2,5)$, for the lengths $n = (5^2-1)/(5-1) = 6$ and $k = 6-2 = 4$. Two possible parity-check equations are: $x_4 = 4x_0 \oplus_5 4x_1 \oplus_5 4x_2 \oplus_5 4x_3$ and $x_5 = 4x_0 \oplus_5 3x_1 \oplus_5 2x_2 \oplus_5 x_3$. Hence, the vectors derived from the equation coefficients: $[4,4,4,4]^T$ and $[4,3,2,1]^T$, constitute the columns of $\mathbf{P}$.
Analogously, decoding is carried on as in the Equation 2-2. The Parity-Check Matrix $\mathbf{H}$, has dimensions r-by-n with n l.i.[2] vectors $\in \mathbb{F}_q^r$, regardless their order. Preferably, ones are placed in the first non-zero row entries. This matrix is formed by the transpose of the negated Parity Matrix $-\mathbf{P}^T$, with k vectors $\in \mathbb{F}_q^r$ as columns, and the Identity Matrix of size r, $\mathbf{I}_r$, such that $\mathbf{H}_{r,q} = [-\mathbf{P}^T|\mathbf{I}_r]$. Single error correction is guaranteed, due to fact that any two columns are l.i., thus $d = 3$ at least. The product of the matrix multiplication $\mathbf{G}\mathbf{H}^T = \mathbf{0}_{k,r}$, is the all-zeros matrix.
E.g. the construction example of $Ham(2,3)$ is observed in the Figure 2-1, where $n = (3^2-1)/(3-1) = 4$ and $k = 4-2 = 2$. Thus the parity-check equations are $x_2 = 2x_0 \oplus_3 2x_1$ and $x_3 = 2x_0 \oplus_3 x_1$ (both l.i.), whose coefficients are reflected on the sub-matrix $\mathbf{P}$ from $\mathbf{G}_{2,3}$. The rectangle in the Figure 2-1 encloses all the parity elements, generated by the full set of messages $\in \mathbb{F}_3^2$.

The all-zeros syndrome $\mathbf{s} = \mathbf{0}_r$, when applying the Equation 2-3, represents a codeword without any error. Otherwise the syndrome is factored, and its vectorial multiple is compared with every column of $\mathbf{H}$. The column corresponding to the match determines the error position in the codeword, and the remaining scalar multiple is the error magnitude.
For instance, if the message $\mathbf{u} = 0120120120$ is encoded by $Ham(3,3)$. Then, an error takes place in the seventh position from the codeword, such that $\mathbf{x} = 012012\underline{2}120211$ (see underlined). The syndrome obtained by the Parity-Check Matrix $\mathbf{H}_{3,3}$ from Table 2-1 is $\mathbf{s} = [2,0,2]^T$, factored as $\mathbf{s} = 2 * [1,0,1]^T$. This vectorial multiple equals the seventh column of $\mathbf{H}_{3,3}$, enclosed by a rectangle. Thus the original codeword is recovered through the following operation: $\hat{\mathbf{x}} = \mathbf{y} - \mathbf{e} = 0120122120211 - 0000002000000 = \underline{0120120120}211$. The message $\mathbf{u}$, coincides with the first 10 elements from the codeword $\hat{\mathbf{x}}$.

[2] l.i. is the abbreviation for linearly independent.

$$\mathrm{G}_{2,3} = [\mathrm{I}_k|\mathrm{P}] = \left(\begin{array}{cc|cc} 1 & 0 & 2 & 2 \\ 0 & 1 & 2 & 1 \end{array}\right)$$

$$\mathrm{H}_{2,3} = [-\mathrm{P}^T|\mathrm{I}_r] = \left(\begin{array}{cc|cc} 1 & 1 & 1 & 0 \\ 1 & 2 & 0 & 1 \end{array}\right)$$

$$\mathrm{H}_{3,3} = \left(\begin{array}{cccccc|c|ccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 2 & 2 & 2 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 1 & 2 & 0 & 0 & 1 \end{array}\right)$$

00	00
01	21
02	12
10	22
11	10
12	01
20	11
21	02
22	20

Figure 2-1: Generator Matrix for $Ham(2,3)$, Parity-Check Matrices for $Ham(2,3)$ and $Ham(3,3)$. Codewords of the full set of messagess $\in \mathbb{F}_3^2$.

An alternative decoder for short codeword lengths, increasing error correction capabilities, can be implemented by the coset leader technique. Let $\mathcal{C}$ be a complete set of codewords, such that its coset is defined as: $\mathcal{C} + \mathbf{s} = \{\mathbf{x} + \mathbf{s} \mid \mathbf{x} \in \mathcal{C}\}$, where $\mathbf{s}$, $\mathbf{x}$ are any syndrome and codeword respectively. Every non-null syndrome from those $q^r - 1$ in total is linked to q^k different coset members. The coset leader of a particular syndrome has the minimum weight, but it is not unique. Moreover, the coset leader equals the most likely error $\mathbf{e} = \{\mathbf{x} + \mathbf{s}, \mid \sigma(\mathbf{x} + \mathbf{s}) = min\{\sigma(\mathcal{C} + \mathbf{s})\} \geq 1\}$. Clearly, the use of computer resources turns more arduous when increasing q^k, until reaching unpractical levels.
The q-ary Hamming encoder/decoder can be found in the Appendix A-1. These routines are newly implemented, since Matlab does not offer any packet with Non-Binary Hamming Codes.

2-3 Block Code Construction over Extension Fields

The information symbols experience transformations, when building block codes over extension fields by the usual schemes, i.e. $\mathcal{A}_q \rightarrow \mathcal{A}_{p_r} \ \forall \ q = p_r^m$. Keeping the original alphabet invariable is crucial, due to the fact that its symbols could have real and physical meaning. Nevertheless, alternative block code constructions, which indirectly employ extension fields and with the input alphabet equal to the output, can be implemented.
Extension fields contain symbols as $\{0, 1, \alpha, \alpha^2, ..., \alpha^{p_r^m - 2}\} \in GF(p_r^m)$, where α is the primitive element of order $p_r^m - 1$. The elements from $GF(p_r^m)$ are expressed as m-tuple vectors with elements in $GF(p_r)$. Their construction is explained along the following lines.
Lets start by reviewing the four basic operations, for the two polynomials $f(x) = 3x + 2$ and $g(x) = 3x^2 + x + 1$, both $\in \mathcal{A}_4$:

- Subtraction:
  $f(x) - g(x) = 3x + 2 - (3x^2 + x + 1) = x^2 + 2x + 1$

- Summation:
  $f(x) + g(x) = 3x^2 + 3x + x + 2 + 1 = 3x^2 + 0x + 3 = 3x^2 + 3$

- Multiplication:
  $f(x) * g(x) = (3x+2) * (3x^2+x+1) = (3*3)x^3 + (3*1+2*3)x^2 + (3+2)x + 2 = x^3 + x^2 + x + 2$

- Division: $g(x)/f(x) =$

$$\begin{array}{r|l} 3x^2+x+1 & 3x+2 \\ \hline 3x^2+2x(-) & x+1 \\ 3x+1 & \\ 3x+2(-) & \\ 3 & \end{array} \qquad g(x) = f(x) * q(x) + r(x) = (3x+2) * (x+1) + 3$$

  Where $q(x) = x+1$, $r(x) = 3$ are the quotient and the remainder respectively, thus $g(x)$ is not divisible by $f(x)$ (otherwise $r(x) = 0$).

An important concept is introduced in [1], regarding to the division: *irreducibility*. A polynomial $f(x)$ is irreducible over $GF(p_r^m)$, if it is indivisible by any polynomial of degree between 0 and $p_r^m - 1$, similarly to prime numbers. In other words, irreducible polynomials cannot be factored, and do not have roots in $GF(p_r^m)$. e.g. the polynomial $3x^2 + x + 2$ is the product of $(3x+2)(x+1)$, thus it is not irreducible over $GF(4)$. However, an irreducible polynomial in one finite field can be factored over other finite field.
Let $\varphi(x)$ be the minimal polynomial (smallest degree) over $GF(p_r)$, such that $\varphi(\theta) = 0$, where θ is also an element $\in GF(p_r^m)$. Therefore $\varphi(x)$ can be expressed as:

$$\varphi(x) = \prod_{i=0}^{m-1} x - (\theta^{p_r^i}) mod_{(p_r^m - 1)} \tag{2-6}$$

Where sum and multiplication are conducted by adding term powers in modulo (p_r) and $(p_r^m - 1)$ respectively. For instance, the minimal polynomial $\varphi(x) \in GF(3)$, with the root $\theta = 1 \in GF(9)$, is calculated as: $\varphi(x) = \prod_{i=0}^{1} x - (1^{3^i}) mod_8 = (x - 1^1)(x - 1^3) = x^2 - 2x + 1$.
The concept that encloses the requirements of extension fields is named *primitiveness*. "An irreducible polynomial $f(x)$ of degree $m \in GF(p_r)$ is primitive, if the smallest positive integer n, for which $f(x)$ divides $x^n - 1$ is $n = p_r^m - 1$" [16]. The polynomial $f(x)$ has the primitive α as root, if $f(\alpha) = 0$. In other words, α is a mathematical trick to find solutions, as well as complex numbers are used if roots are not longer depicted by real numbers.
The elements of $GF(p_r^m)$ are expressed as $b_0 + b_1\alpha + b_2\alpha^2 + ... + b_{m-1}\alpha^{m-1}$, where $b_0 b_1 b_2 ... b_{m-1}$ is the m-tuple vector $\in GF(p_r)$. Primitive polynomials of degree m over $GF(p_r^m)$ are not unique.
The steps to build the extension field $GF(9)$ in the Table 2-3 are summarized:

1. Express the extension field as $GF(p_r^m)$: $GF(3^2) = GF(9)$.
2. Choose a primitive polynomial of degree $m = 2 \in GF(3)$: $f(x) = x^2 - 2x - 1$.
3. Solve $f(\alpha) = 0$ for the primitive α: $\alpha^2 = 2\alpha + 1$.
4. Express the elements $\{0, 1, \alpha, \alpha^2, ..., \alpha^7\} \in GF(9)$, in terms of $b_0 + b_1\alpha$.
   Products are calculated by adding powers in $\text{mod}_{(p_r^m - 1)}$, i.e. $\alpha^2 * \alpha^3 = \alpha^{(2+3) mod_8} = \alpha^5$.
   Sums are calculated by adding powers in $\text{mod}_{(p_r)}$, i.e. $\alpha + \alpha^2 = \alpha^{(1+2) mod_3} = \alpha^0 = 1$, or $(10) + (21) = 01$.

5. Form the 2-tuple vectors $\in GF(3)$, using the coefficients $b_0 b_1$.

Elements $\in GF(9)$	Elements expressed as the sum of α^1 and α^0	2-tuple vector $\in GF(3)$
0	0	00
1	1	01
α	α	10
α^2	$2\alpha + 1$	21
α^3	$2\alpha^2 + \alpha = 2(2\alpha + 1) + \alpha = 4\alpha + 2 + \alpha = 5\alpha + 2 = 2\alpha + 2$	22
α^4	$2\alpha^2 + 2\alpha = 2(2\alpha + 1) + 2\alpha = 4\alpha + 2 + 2\alpha = 6\alpha + 2 = 2$	02
α^5	2α	20
α^6	$2\alpha^2 = 2(2\alpha + 1) = 4\alpha + 2 = \alpha + 2$	12
α^7	$\alpha^2 + 2\alpha = 2\alpha + 1 + 2\alpha = 4\alpha + 1 = \alpha + 1$	11

Table 2-3: Extension field construction $GF(9)$, via the primitive polynomial $x^2 - 2x - 1$ [16].

Finally, the original alphabet symbols change from $\{0, 1, 2, 3, 4, 5, 6, 7, 8\} \in GF(9)$ to $\{00, 01, 10, 21, 22, 02, 20, 12, 11\}$ with elements $\in GF(3)$, in this case.

2-3-1 Binary BCH Codes

The binary BCH Codes of length n and design distance d' are Cyclic Codes, generated by the multiplication of distinct minimal polynomials of the primitive elements $\alpha, \alpha^2, \alpha^3, ..., \alpha^{d'-1} \in GF(2^m)$ [15]. The design distance d' is the minimum distance expected, before building the BCH codes. It is guaranteed to be the same or lower than the distance of the block codes d, after their construction. The design distance, codeword and information lengths are user-defined, by adjusting the parameters m and t such that:

$$\begin{aligned} & n = 2^m - 1 && \forall \quad m \geq 2 \\ & k \geq n - mt && \forall \quad 1 < t < n/m \\ & d \geq d' = 2t + 1 \end{aligned} \tag{2-7}$$

The construction of BCH Codes is carried on by polynomial models. The binary BCH codes: $(7, 4, 3)$ and $(7, 1, 7)$ are built step-by-step to illustrate this theory.

1. Choose the parameters $m = 3$, $t = 1$ & 3, such that the code length and the design distances are $n = 2^3 - 1 = 7$, $d'_0 = 2 * 1 + 1 = 3$ & $d'_1 = 2 * 3 + 1 = 7$ respectively.

2. Build the extension field $GF(8)$ as $\{000, 100, 010, 001, 110, 011, 111, 101\}$, via the primitive polynomial $f(x) = x^3 + x + 1$.

3. Calculate the minimal polynomials of $d'_1 - 1 = 6$ primitive elements $\in GF(8)$, using the Equation 2-6, as shown in the Table 2-4.

4. Build the generator polynomial, as the product of distinct minimal polynomials of $d'_0 - 1 = 2$ primitive elements, α, α^2: $g_0(x) = x^3 + x + 1$. The minimum distance $d_0 = 3$ equals the weight of $g_0(x)$. Thus $t = 1$ & $k = 7 - 3 * 1 = 4$ for the BCH Code $(7, 4, 3)$.

5. Build the generator polynomial, as the product of distinct minimal polynomials of $d_1' - 1 = 6$ primitive elements, $\alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6$: $g_1(x) = (x^3 + x + 1)(x^3 + x^2 + 1) = 1 + x + x^2 + x^3 + x^4 + x^5 + x^6$. The minimum distance $d_1 = 7$ equals the weight of $g_1(x)$. Thus $t = 3$ & $k = 7 - 3 * 3 = 1$ for the BCH Code $(7, 1, 7)$.

Elements $\in GF(8)$	Minimal Polynomial $\varphi(x)$
1	$x + 1$
$\alpha, \alpha^2, \alpha^4$	$x^3 + x + 1$
$\alpha^3, \alpha^5, \alpha^6$	$x^3 + x^2 + 1$

Table 2-4: Minimal polynomials $\varphi(x)$ of the primitive elements $\in GF(8)$.

Finally, the messages in polynomial representation can be encoded using the Equation 2-4.

Chapter 3

Knuth-based Codes

The current chapter introduces the fundamentals of Knuth's balancing method applied to q-ary codes. It also adapts the binary error correction scheme, proposed by Weber, Immink and Ferreira in [11] to q-ary codes, supported by the findings about q-ary sequences from [7]. A new optimization method is presented, to reduce the number of iterations in the search of balancing indices. Redundancy is analyzed under three distinct concepts: Transmitted Index, Auxiliary Information Encoding (derived by the balancing position selection) and A Better Variable-Length-Prefix Construction. The procedures from [10] and [9] are first recalled, then they are applied over q-ary codewords.

3-1 q-ary Knuth-based Coding Scheme with Error Correction Capabilities

This section refers to Charge-Balanced Codes hereafter, whose alphabet can have one of three possible forms, as in a), b) or c) below. Subscripts are assigned to q and n to quickly reference their even or odd values. Swart and Weber define the "balancing value" β in [7], of the balanced sequence $\mathbf{x} = x_0 x_1 ... x_{n-1}$ of length n, with symbols from an alphabet $\mathcal{A}_q$, such that:

$$\beta = \sum_{i=0}^{n-1} x_i = \frac{n(q-1)}{2} \tag{3-1}$$

An exception is applied for q_{even} with n_{odd}.

a) $\beta > 0$ for $\mathcal{A}_q$ of the form: $\{0, 1, ..., q-1\} \forall\, q$.

Or $\beta = 0$ for $\mathcal{A}_q$ of the forms:

b) $\{-(q-1)/2, ..., -2, -1, 0, +1, +2, ..., +(q-1)/2\} \forall\, q_{odd}$.

c) $\{-(q-1), ..., -3, -1, +1, +3, ..., +(q-1)\} \forall\, q_{even}$.

This section makes reference only to the first form, since the alphabet mapping from the format "a)" to "b)" or "c)" is done straightforward, using the transformation proposed by Weber, Immink, Siegel and Swart. Let i be a symbol from an alphabet $\mathcal{A}_q$ of any form, thus i can be mapped to any other form, such that $i \rightarrow ai+b$, where $a \neq 0$ and $b \in \mathfrak{R}$. For instance, alphabet conversion from $\{0,1,2,3\}$, of form a), to $\{-3,-1,1,3\}$, of form c), both $\in \mathcal{A}_4$, is done by choosing $a=2$ and $b=-3$, thus $\{2*0-3, 2*1-3, 2*2-3, 2*3-3\} = \{-3,-1,1,3\}$. Similar steps for q-ary codes are adapted, from the scheme with error correction capabilities, for the binary Knuth-based codes proposed in [11]. They are broken down as follows:

1. Encode the message $\mathbf{u} \in \mathbb{F}_q^k$, using a Block Code $\mathcal{C}_1 : (n,k,d_1)$ over $GF(q^m)\ \forall\ m \geq 1$. Then obtain the codeword $\mathbf{x} \in \mathbb{F}_q^n$, such that $\mathbf{x} = \phi(\mathbf{u}) = \mathbf{u} * G_1$, where G_1 is the Generator Matrix from Equation 2-1, and ϕ is the encoding function.

2. Calculate the balancing index β from Equation 3-1. Compare β with the weight $\sigma(\mathbf{y}_i)$, in every attempt to find a balanced codeword, by running an iterative procedure qn times. Store the balancing indexes if both parameters are equal, i.e. $\sigma(\mathbf{y}) = \beta$. Follow the next operations to find out the balancing variables:

   a. Let s and e be indices whose values shift from 0 to $q-1$, and 0 to $n-1$ respectively.

   b. Denote a balancing sequence $\mathbf{b}(s,e)$, with arguments s and e, of the form $b_0 b_1 ... b_{n-1}$ and length n. Calculate every element of $\mathbf{b}(s,e)$, using the following formula:

$$b_i = \begin{cases} s, & \forall \quad i \geq e, \\ s+1, & \forall \quad i < e. \end{cases} \tag{3-2}$$

   In such a way that $\mathbf{b}(s,e) = \overbrace{(s+1)(s+1)...(s+1)}^{e}\overbrace{ss...s}^{n-e}$, is decomposed as $\underbrace{sss...s}_{n} \oplus_q \underbrace{1...1}_{e}\underbrace{0...0}_{n-e}$, where $\oplus_q$ is a sum in modulo q.

   c. Obtain the weight $\sigma(\mathbf{y})$, of the sum $\mathbf{y} = \mathbf{x} \oplus_q \mathbf{b}(s,e)$, where $\sigma(\mathbf{y}) = \sum_{i=0}^{n-1} y_i$.

3. Merge the balancing parameters s & e into one index $z = sn + e$, where $0 \leq z < qn$, as shown in the Table 3-1 a).

4. Perform an inversion[1] over the codeword $\mathbf{x}$, such that $\mathbf{y} = \mathbf{x} \oplus_q \mathbf{b}(z)$[2].

5. Encode the index z into a unique balanced codeword ρ, called *prefix*, via the block code $\mathcal{C}_2$[3]. Where ψ is the encoding function, such that $\rho = \psi(z)$.

Aiming to clarify the steps from 2 to 4, the Table 3-1 b) shows all balancing and balanced sequences of the randomly chosen vector $2132041314 \in \mathcal{A}_5$. These six possible balanced sequences have weight $20 = \beta$ (from the Equation 3-1). For instance, the first balanced sequence is determined as $2132041314 \oplus_5 2222111111 = 4304102420$, with weight calculated as $4+3+0+4+1+0+2+4+2+0 = 20$. Finally, its balancing index is estimated as $z = sn + e = 1*10+4 = 14$, the parameters s & e are easily figured out from the balancing sequence. A Matlab routine for these operations is found in the Appendix A-2.

[1] The term *inversion* or *complementation* refers to binary symbol conversion, $0 \rightarrow 1$ or $1 \rightarrow 0$. This reference is kept to indicate the sum of balancing sequences $\forall\ q > 2$, for simplicity.

[2] Several balancing sequences could be obtained.

[3] The prefix's length p, necessary to represent z must be $p > \log_q(qn)$.

Table 3-1: a) Balancing parameters s and e merged into z. b) Table of all the balancing and the balanced sequences of $2132041314 \in \mathcal{A}_5$.

s/e	0	1	2	$\cdots$	n-1
0	0	1	2	$\cdots$	n-1
1	n	n+1	n+2	$\cdots$	2n-1
2	2n	2n+1	2n+2	$\cdots$	3n-1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
q-1	(q-1)n	(q-1)n+1	(q-1)n+2	$\cdots$	qn-1

z	s	e	b(z)	y
14	1	4	2222111111	4304102420
19	1	9	2222222221	4304213030
29	2	9	3333333332	0410324141
34	3	4	4444333333	1021324142
44	4	4	5555444444	2132430203
49	4	9	5555555554	2132041313

As displayed in the Figure 3-1 a), the encoder's output is the link between the prefix ρ and the balanced codeword $\mathbf{y}$, the latter is named *bulk* or *payload*.
Similarly as at the source's side, the inverse operations are performed for decoding at the destination. Thus the next steps proceed:

1. Separate the prefix from the bulk, then compare the former with the codebook entries from $\mathcal{C}_2$. Choose the closest vector, denoted as $\hat{\rho}$[4].
2. Extract the index $\hat{z}$ from $\hat{\rho}$, $\hat{z} = \psi^{-1}(\hat{\rho})$, then calculate the balancing parameters to build $\mathbf{b}(\hat{z})$ as: $\hat{e} = mod(\hat{z}, n)$ and $\hat{s} = \lfloor \hat{z}/n \rfloor$.
3. Recover the sequence $\hat{\mathbf{x}}$, by subtracting the balancing sequence $\mathbf{b}(\hat{z})$ from $\hat{\mathbf{y}}$: $\hat{\mathbf{x}} = \hat{\mathbf{y}} \ominus_q \mathbf{b}(\hat{z})$.
4. Decode the codeword $\hat{\mathbf{x}}$, according to the block code $\mathcal{C}_1$, $\hat{\mathbf{u}} = \phi^{-1}(\hat{\mathbf{x}})$.

Retaking the example previously described, let the sequence $\hat{\mathbf{y}}$ and the index z be 4304102420 and 14 respectively, after the decoding step 2. The balancing parameters are calculated as: $\hat{e} = mod(14, 10) = 4$ and $\hat{s} = \lfloor 14/10 \rfloor = 1$. Thereafter, the following subtraction is performed: $\hat{\mathbf{x}} = 4304102420 \ominus_5 2222111111 = 2132041314$ (the original vector).
A diagram of this process is depicted in the Figure 3-1 b), and the Matlab routine of steps 2 and 3 is shown in the Appendix A-3.

The hamming distances of $\mathcal{C}_1$ and $\mathcal{C}_2$ from the Figure 3-1 a) are chosen to be $d_1 = 2t + 1$ and $d_2 = 2t + 2$ respectively in [7], for correcting up to t errors. This statement is still valid for q-ary codes. For instance, let two messages be $\mathbf{u}_1 = 10$ and $\mathbf{u}_5 = 21$, both $\in \mathbb{F}_3^2$. Single error correction is guaranteed ($t = 1$), by applying the Hamming Code $\mathcal{C}_1 : (4, 2, 3)_3$ to $\mathbf{u}_1$ and $\mathbf{u}_5$. The codewords $\in \mathbb{F}_3^4$: $\mathbf{x}_1 = 2210$ and $\mathbf{x}_5 = 0221$ are obtained. Then balancing sequences with indices $z_1 = 5$ and $z_5 = 2$ are added to $\mathbf{x}_1$ and $\mathbf{x}_5$, which leads to the same vector $\mathbf{y}_1 = \mathbf{y}_5 = 1021$, as shown in the Table 3-2. Consequently, the minimum distance of the codeword only resides in the prefix in such cases, i.e. $d_2(\rho_1, \rho_5)$.
Let $A_q(p, d, w)$ be the maximum size of q-ary codes of length p, minimum distance d, and constant weight w. The lower bound of a balanced code under the *fixed-length-prefix* scheme

[4]The notation ˆ is used to differentiate from the original values at the encoder, both terms are the same if no error occurs.

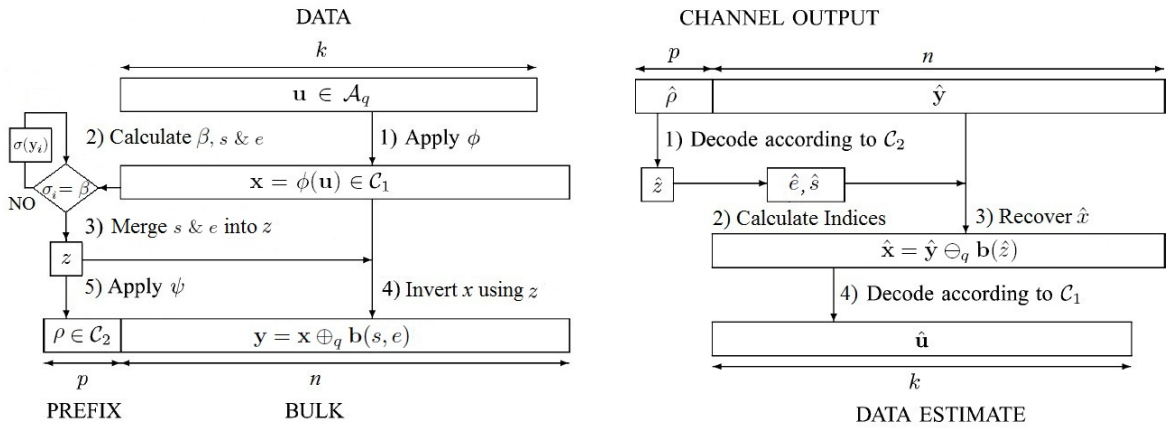

Figure 3-1: a) q-ary Encoding Procedure Diagram. b) q-ary Decoding Procedure Diagram [11].

must be $A_q(p, d_2, \frac{p(q-1)}{2}) \geq qn \ \forall \ q > 2$ (n if $q = 2$). The mapping of the balancing index to the prefix $\rho = \psi(z)$ is carried on by look-up tables. Some experimental cardinalities of balanced codes for different values of p and q, and hamming distance $d_2 \geq 4$, are listed in the Table 3-3 a). These numbers were determined by the Matlab routine from the Appendix A-4, which attempts to find the maximum interception of vectors with $d_2 \geq 4$ in the balanced set[5]. Several binary results are placed together with those from [7], showing high proximity.
Using the values from Table 3-3 a) in the previous case, $\mathcal{C}_2 : A_3(5, 4, 5) = 12 = 3 * 4$ meets the requirements, i.e. $d_2(11021, 02012) = 4$ for $\mathbf{u}_1$ and $\mathbf{u}_5$.
On the other hand, minimum distances of codewords with the same prefix depend on the bulk's segment, for example $d_1(\mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_7) = d_1(1120, 2101, 0112) = 4$ from the Table 3-2. The code rate is calculated as $R = k/(p + n) = 2/(5 + 4) = 0.222$.

u	ρ	y
00	10211	1111
10	11021	1021
20	00122	1120
01	00122	2101
11	01220	2011
21	02012	1021
02	02012	2002
12	00122	0112
22	01220	0022

Table 3-2: q-ary Knuth-based Codes with single error correction capabilities: Full set of messages $\in \mathbb{F}_3^2$ under fixed-length-prefix scheme.

Additionally, the determination of t is vital, since lacking of sufficient error protection over the prefix would lead to devastating results, due to the fact that a wrong complementation creates a completely different codeword. Furthermore, this technique is meant for values of n much

[5]These outcomes are not proven optimal.

greater than those of p, $n >> p$. Thus the probability of a random error hitting the prefix is significantly lower, in comparison with one hitting the bulk, i.e. $Pr(p) = p/(p+n) << Pr(n) = n/(p+n)$, considering this fact $d_1 >> d_2$ in reality.
Block code constructions for some prime numbers of q are shown in the Table 3-3 b), using the experimental outcomes from a), with single error correction capabilities added by Hamming Codes (to both: prefix & bulk).

q	p	$A_q(p, d_2, \frac{p(q-1)}{2})$	
		Max	Exp.
2	4	2	2
2	6	4	4
2	8	14	14
2	10	36	31
2	12	132	132
2	14	325	257
3	2		2
3	3		3
3	4		7
3	5		12
3	6		31
3	7		59
3	8		183
5	2		3
5	3		7
5	4		25
5	5		73
5	6		253
7	2		4
7	3		12
7	4		63
7	5		250
11	2		6
11	3		29
11	4		231
13	2		7
13	3		40
13	4		377
17	2		9
17	3		66
17	4		833
19	2		10
19	3		79

q	n	p	R
2	4	6	0.1
2	6	8	0.214
2	8	8	0.25
2	10	8	0.333
2	16	10	0.423
2	20	10	0.5
2	24	10	0.559
2	30	10	0.625
2	32	12	0.591
3	4	5	0.222
3	6	6	0.25
3	7	6	0.308
3	8	6	0.357
3	10	6	0.438
3	16	7	0.522
3	20	8	0.571
3	24	8	0.625
3	30	8	0.684
3	32	8	0.7
5	4	4	0.25
5	6	5	0.364
5	7	5	0.333
5	8	5	0.385
5	10	5	0.467
5	16	6	0.591
5	20	6	0.654
5	24	6	0.7
5	30	6	0.75
5	32	6	0.737
7	4	4	0.25
7	6	4	0.4
7	7	4	0.455
7	8	4	0.5
7	10	5	0.467
7	16	5	0.619
7	20	5	0.68
7	24	5	0.724
7	30	5	0.771
7	32	5	0.784
11	4	4	0.25
11	6	4	0.4
11	7	4	0.455
11	8	4	0.5
11	10	4	0.571
11	16	4	0.65
11	20	4	0.708
13	4	4	0.25
13	6	4	0.4
13	7	4	0.455
13	8	4	0.5
13	10	4	0.571
13	16	4	0.65
13	20	4	0.708
13	24	4	0.75
17	4	4	0.25
17	6	4	0.4
17	7	4	0.455
17	8	4	0.5
17	10	4	0.571
17	16	4	0.7
17	20	4	0.708
17	24	4	0.75
17	30	4	0.794
17	32	4	0.806
19	4	3	0.286

Table 3-3: a) Cardinalities of balanced codes of length $p \leq 12$, and hamming distance $d_2 \geq 4$, for $q = 2, 3, 5, 7, 11, 13, 17, 19$. b) q-ary Knuth-based codes with single error correction, using fixed-length-prefix scheme, for $q = 2$ to 19 and $n = 4, 6, 7, 8, 10, 16, 20, 24, 30, 32$.

3-1-1 Iteration reduction over the balancing index search

Recalling the step 2 from the encoding procedure, the index z is stored if the sequence's weight is $\sigma(\mathbf{y}) = \beta$, tested qn times by an iterative procedure. Such practice becomes very inefficient when dealing with large values of n. For instance, a cycle must be executed $qn = 32 * 1024 = 32768$ times for every codeword $\in \mathcal{A}_{32}$ of length $n = 1024$, to find out all the balancing sequences. This process demands a very high cost in terms of computer resources. For this reason, a need of finding optimization methods to reduce the number of laps arises. Initial information for the construction of a precise loop-reduction model is scarce: $\sigma(\mathbf{x})$ and β. Moreover, distinct values of z for the same $\mathbf{x}$ are indistinguishable. Thus, only lower bounds can be established with those. Consequently, the next definition is made:
"Let $\beta - \sigma(\mathbf{x})$ be the balance disparity, for any sequence $\mathbf{x}$, before searching any balancing index z". This term indicates how far the weight of $\mathbf{x}$ is from the state of balance.
The main goal of this proposal is to set up a higher start point for the search of z, since it always begins from zero regardless any balance disparity. Hence, this section mainly attempts to find the correlation between $\beta - \sigma(\mathbf{x})$ and z for any sequence. The first found index z is enough to achieve balance and to end the loop. Nevertheless, one might also want to find all the balancing sequences, thus step incrementation among the values of z is analyzed.
A 2-D plane containing all the sequence's weights from the full set $\in \mathcal{A}_q$ of length n is traced out in the Figures 3-2 & 3-3 a) to c). The balancing indices z's[6], associated to those q^n sequences, are drawn on the Y-axis. The balance disparity lies on the X-axis, and it is either positive or negative. The 2-D plane has its origin in the coordinates $(\beta - \sigma(\mathbf{x}), z) = (0, 0)$, point related to vectors with weight $\sigma(\mathbf{x}) = \beta$. A point is located in the I-Quadrant if $\sigma(\mathbf{x}) < \beta$. On the other hand, every point located in the II-Quadrant meets $\sigma(\mathbf{x}) > \beta$. E.g. the plotting is made from the sequence 0000000000 to 2222222222 for $q = 3$ and $n = 10$. Some examples to clarify this procedure are shown in the Appendix B-2.
The points from equal-weighted sequences are either overlapped or vertically aligned. The term $\beta - \sigma(\mathbf{x})$ lower approximates z in the I-Quadrant, thus the abscissas are matched with the lowest ordinates. The initial values of z require an extra adjustment in the II-Quadrant. Every point is aligned on a straight-line, located at least q units apart from any other parallel set of points. Hence, such adjustment is $z \geq \beta - \sigma(\mathbf{x}) + q * j$, where j is the straight-line number. The straight-lines are numbered starting from the origin as $j = 0$. The estimation of the parameter j is safely made by $j = \lceil \frac{\beta - \sigma(\mathbf{x})}{1-q} \rceil$. Summarizing all the previous statements into an expression, for any value of q and n is said:

$$z \geq \beta - \sigma(\mathbf{x}) + q * j \quad \begin{cases} j = 0 & \forall \quad \sigma(\mathbf{x}) \leq \beta, \\ j \geq 1 & \forall \quad \sigma(\mathbf{x}) > \beta. \end{cases} \tag{3-3}$$

Blue lines shaping the lower border in the 2-D plane are shown in the Figures 3-2 & 3-3 a)-c), when applying this method. This technique is always precise for binary case as observed in the Figure 3-2 a). None accuracy, but a good adjustment is achieved for some values of q and n, as shown in the Figure 3-2 b) & d).
Swart and Weber demonstrate in [7] a property of $\sigma(\mathbf{y}_i)$ or $\sigma(z)$ named *random walk*, characteristic from the weight of sequences $\in \mathcal{A}_q$ of length n, to increase and to decrease by steps of one and $q - 1$ respectively. At least one of the them achieves balance $\sigma(z) = \beta$, statement based on Knuth's principles. Recalling the example of the sequence 2132041314, this

[6] All the balancing sequences of $\mathbf{x}$ are considered.

phenomenon is observed in the weight behavior from the Figure 3-3 d) (red dashed-line). As shown before, six sequences achieve balance in this case, located over the blue dashed-line that draws β. A collateral feature of the random walk is useful to speed up the search of z: balancing indices are spaced among themselves by steps of q units, i.e. $z, ..., z + qj \ \forall \ j \geq 1$. However, this statement does not affirm that balancing indices are necessarily consecutive. Such property is seen in the vertical green straight-line, that connects the points from all the balancing indices z's, of randomly chosen vectors $\mathbf{x}$ in the Figures 3-2 & 3-3 a) to c), denoted as $\sigma(z)$'s.
This technique fully guarantees to place all the balancing indices in phase (qj units away). Although, the search of z might not start from lowest point, in the straight-line of the first assertive index, but other qj units below. This method aims to be applicable for any value of q or n. Further improvements for real applications can be made once such variables are defined.

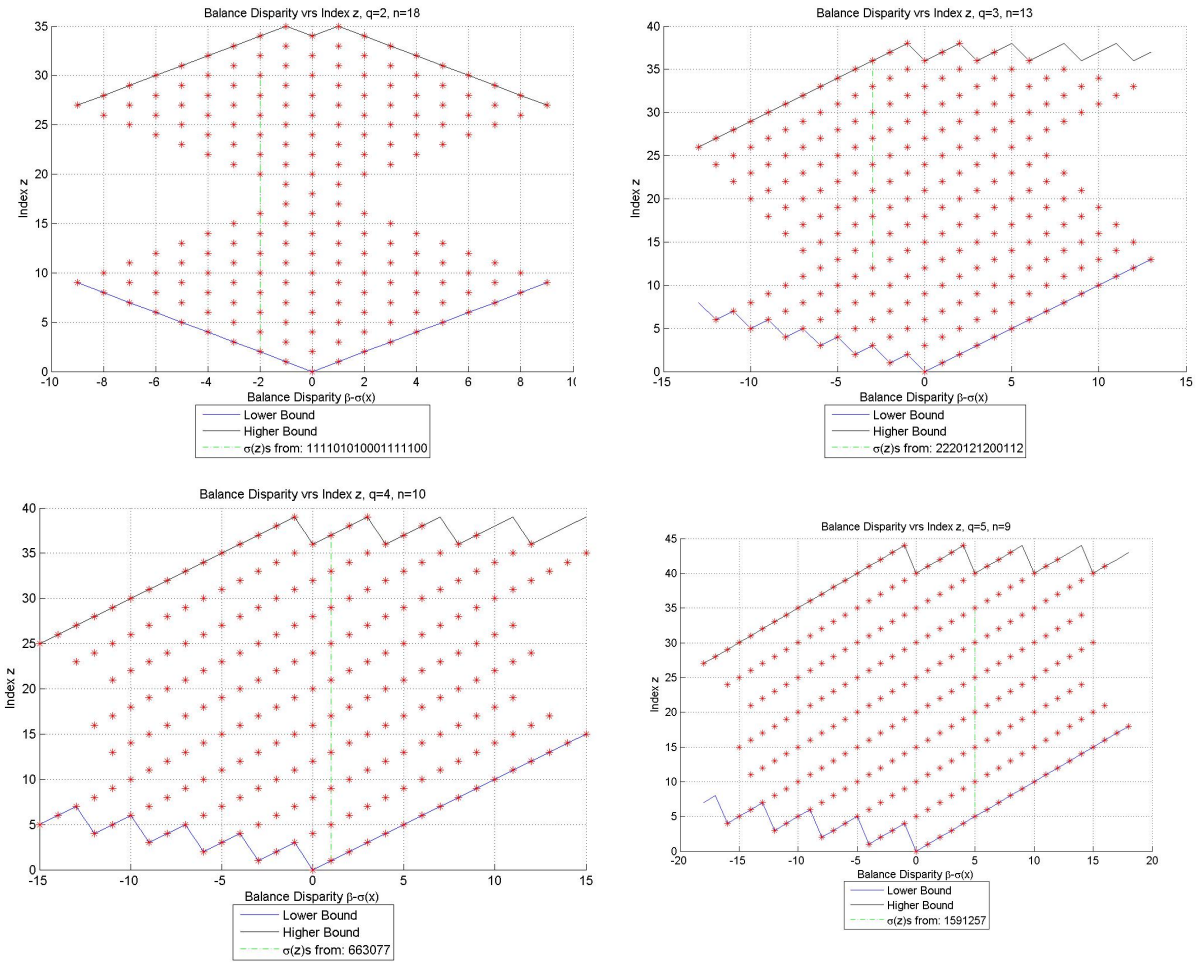

Figure 3-2: 2-D Plane of z versus $\beta - \sigma(\mathbf{x})$ for: a) $q = 2$ & $n = 18$, b) $q = 3$ & $n = 13$, c) $q = 4$ & $n = 10$ and d) $q = 5$ & $n = 9$.

Estimating the operational performance of this technique is difficult, due to the random

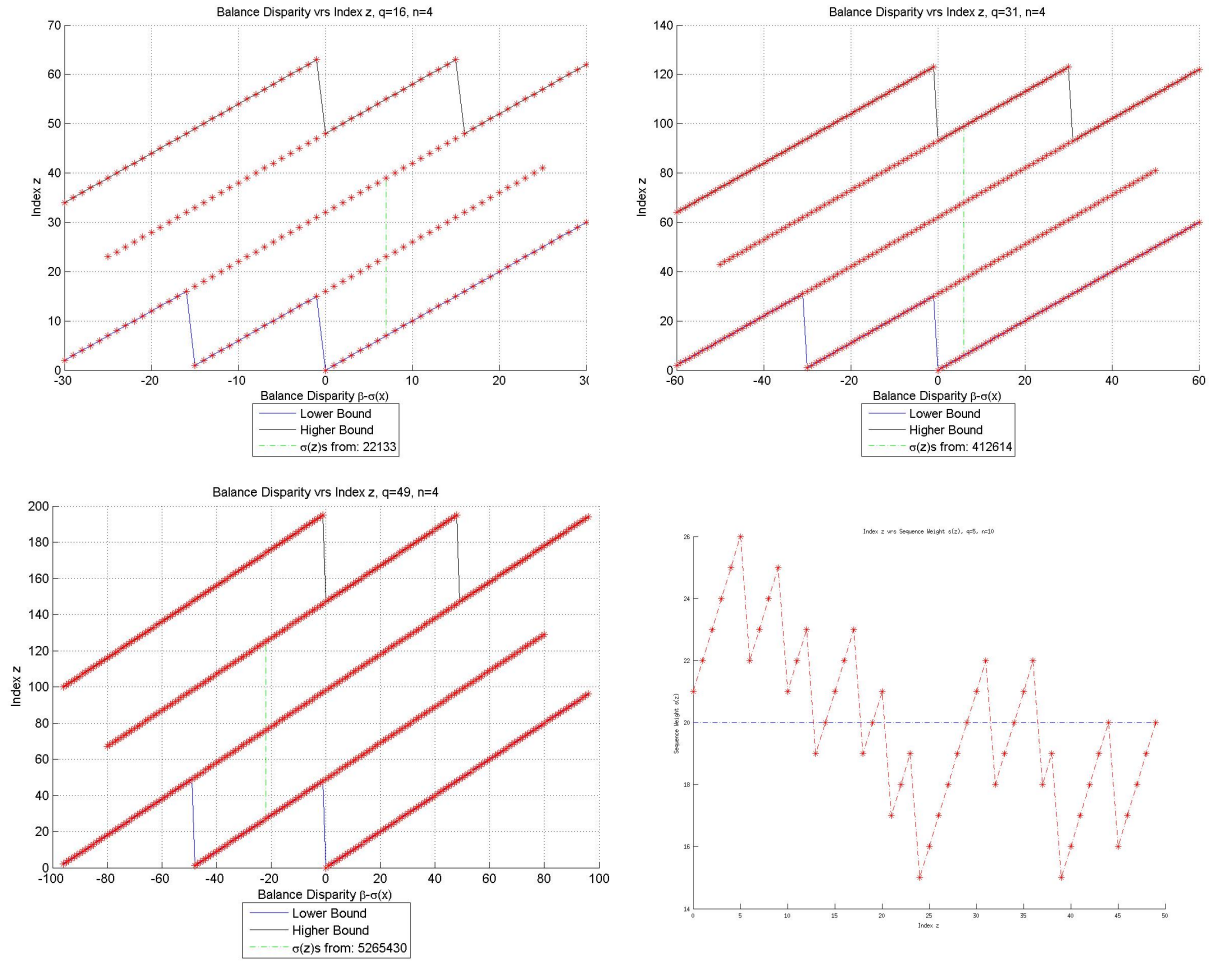

Figure 3-3: 2-D Plane of z versus $\beta - \sigma(\mathbf{x})$ for $n = 4$ & $q =$ a) 16, b) 31 and c) 49. d) Weight $\sigma(z)$ vrs index z of the sequence "2132041314".

occurrence of the information. Nevertheless, an analytical way to measure performance can be carried on, by comparing the average number of cycles of a full set of vectors $\in \mathcal{A}_q$ of length n required to find the balancing indices z's, with and without this implementation. However, some sequences are more probable to occur than others in reality, in fact some of them might never come out.

Several values of q and n are listed on the Figure 3-4 a), chosen small due to the computer limitation handling large numbers (i.e. q^n). Some numbers are not integers, but averaged float values, after the division by the total number of sequences q^n. The total number of cycles per sequence before implementing this algorithm is listed on the third column (qn). The minimum averaged iterations for each q and n are displayed in the forth column, if all the balancing sequences were found in their first search attempt (calculations shown in the Appendix B-1). The fifth column contains the number of iterations performed by this method, for all the balancing sequences. Similarly, the sixth and seventh columns have the number of iterations when only looking for the first balancing sequence, before and after applying this technique respectively.

The iterations of this method approach n when searching all the balancing sequences, as shown in the fifth column, leading to a loop reduction of a factor of q, as well as the search step. The number of laps slightly changes when increasing q if only the first z is needed. However, an increment is expected for larger lengths n than those studied. The Figure 3-4 b) displays a graph for different values of q and fixed n. The red curve describes the cases when all balancing sequences are searched. Likewise, the blue curve is traced from calculations that only contemplate the first balancing sequence. One or two iterations are required to find the first z in average.
The number of balancing sequences, or balancing positions, is denoted by the letter v in [10], where $1 \leq v \leq n/2 \ \forall \ q = 2$, and modified for $q > 2$ as $1 \leq v \leq n$, due to the Matlab calculations from Table B-1. Unlike binary case the occurrence distribution of balancing positions $\forall \ q > 2$ is not well-known[7], some curves are shown in the Figure 3-5 b).
The establishment of a relevant upper bound is impossible, since the value v for a particular vector $\in \mathcal{A}_q$ of length n is unpredictable. For instance, there are more balancing positions v's for sequences of weight close to β, than for those of weight near to 0 or 2β, for $q = 3$ and $n = 10$ in the Figure 3-5 a). Several attempts of finding an upper bound were performed, shown in the upper blue curve in the Figures 3-2 & 3-3 a)-c), leading to negligible improvements in the order of a hundred of an iteration.
This implementation is incorporated in the Matlab routine from the Appendix A-2.

q	n	Number of Iterations				
		All z's Found			First z Found	
		Total (q*n)	Minimum	Reduced	Average	Reduced
2	10	20	4.92	9.52	3.50	1.63
2	14	28	5.87	13.32	4.50	2.02
3	6	18	3.48	5.78	3.96	1.47
3	10	30	4.55	9.60	5.62	1.90
4	8	32	3.95	7.70	6.47	1.77
5	6	30	3.36	5.79	6.62	1.58
7	6	42	3.33	5.81	9.22	1.62
8	4	32	2.69	3.90	7.88	1.35
9	4	36	2.68	3.90	8.85	1.36
10	4	40	2.68	3.90	9.80	1.36
11	4	44	2.68	3.90	11.05	1.37
12	4	48	2.68	3.90	11.16	1.37
13	4	52	2.67	3.90	10.54	1.37
14	4	56	2.67	3.91	10.60	1.37
15	4	60	2.67	3.91	12.18	1.38
16	4	64	2.67	3.91	14.63	1.38
31	3	93	2.25	2.95	29.47	1.31

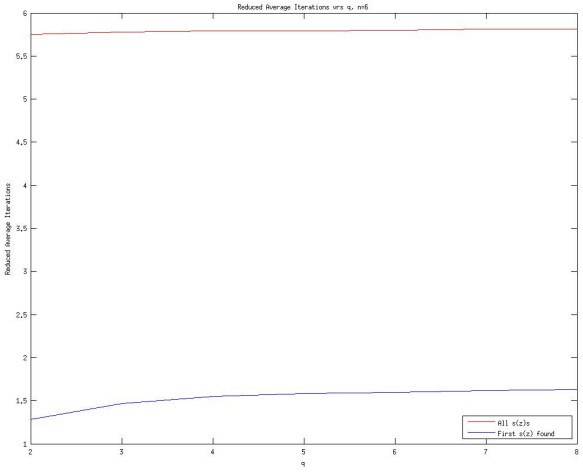

Figure 3-4: a) Number of iterations per sequence $\in \mathcal{A}_q$ of length n in the full set, when searching the balancing index z. b) Graph of q-Factor loop reduction for different q's & fixed length $n = 6$.

3-2 Redundancy of q-ary Knuth-based Codes

In general, this method's redundancy r and code rate R are calculated as $r = n + p - k$ and $R = k/(n + p)$ respectively, but redundancy can be examined in more detail. An important

[7] An expression to describe its behavior has not been found.

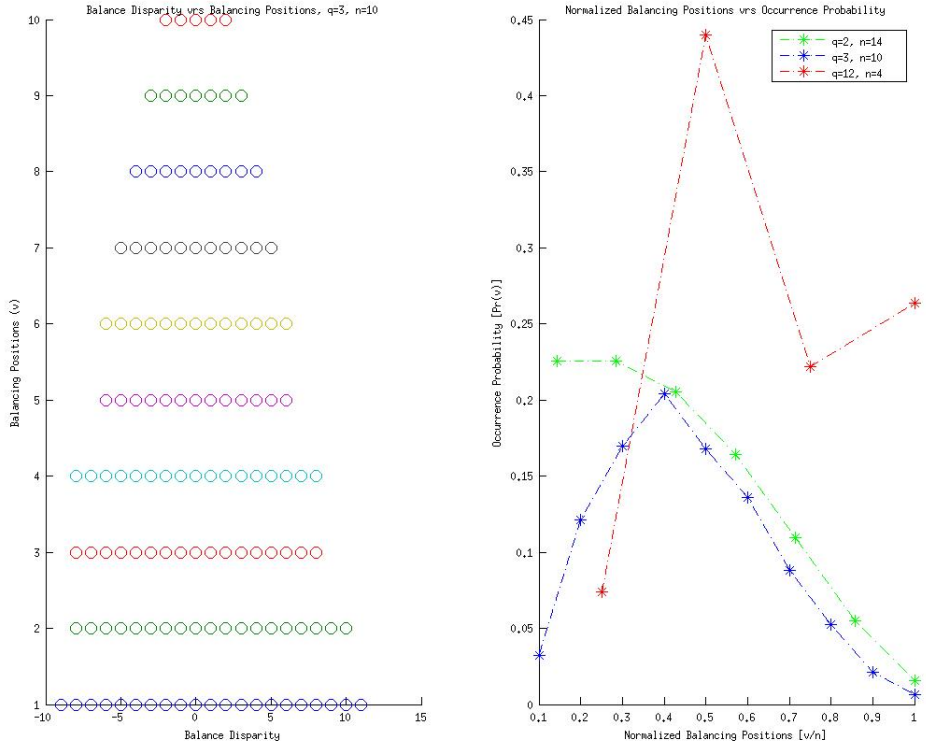

Figure 3-5: a) Graph of Balance disparity versus Balancing positions (v's) for $q = 3$ and $n = 10$. b) Plot of Normalized balancing positions (v/n) versus Occurrence probability [Pr(v)=Occurrences/q^n] for $q = 2, 3, 12$ & $n = 14, 10, 4$.

definition is made in [13] & [7]: The number of sequences $\in \mathcal{A}_q$ of length n and weight equal to β in the full set is denoted as $\mathcal{S}_q^n$ and named cardinality, where it is given by:

$$\mathcal{S}_q^n = q^n \sqrt{\frac{6}{\pi n(q^2-1)}} \left(1 + \mathcal{O}\left(\frac{1}{n}\right)\right) \tag{3-4}$$

This expression is fundamental because only balanced sequences are transmitted or stored. An acceptable approximation is accomplished for large values of n, by making the term $\mathcal{O}\left(\frac{1}{n}\right) \approx 0$. The prefix p should be balanced, so the Equation 3-4 can be applied to it too ($\mathcal{S}_q^p$). Additionally, the prefix length p must equal $\lceil \log_q(qn) \rceil$ at least, to be capable to represent all the values of z. Both conditions are met when the cardinality is $\mathcal{S}_q^p \geq qn$. Several code constructions have been proposed, decreasing redundancy almost by half, i.e. [13]. However, the Knuth's method has easy decoding, less complexity and low memory resources usage (short look-up tables), suitable for applications with long codewords. Therefore, the normalized redundancy for this scheme is:

$$\tfrac{1}{n} \log_q(n) \quad \forall \quad n >> 1. \tag{3-5}$$

This is two times greater than that of the full balanced set, which is a trade-off for simplicity. For this reason, reducing redundancy without loosing features is always under evaluation.

Aiming to do that in the following sections, redundancy of q-ary codewords is reviewed by the approaches: Transmitted Index, Auxiliary Information Encoding and A Better Variable-Length-Prefix Construction. Additionally, computations are addressed under the assumption that all codewords are independent and equiprobable. On the other hand, the prefix is assumed of the length $1+log_q(n)$, but not balanced. Moreover, the *one* from the term is required to represent the symbol index s (not needed in binary).
Only conclusions based on small values are drawn, due to the Matlab limitations when handling large numbers and their long execution, i.e. $q = 3, 4, 5$ & $n \approx 10$.
Calculation logs of the entropies can be found in the Appendix B-3.

3-2-1 Transmitted Index Analysis

Low position indices are more prone to occur, because the encoder begins the inversion from them prioritizing their selection to achieve balance. Additionally, balanced sequences $\mathbf{x}$'s from the studied q & n in the Table B-1 ($e = 0$) have the highest occurrence of them all. Recalling the same premise examined in [10], implementing the variable-length-prefix scheme is an improvement gap. Conveniently, shorter prefix lengths are required to depict low indices. Thus, the prefix's average length must be lower than $log_q(qn)$, due to the unequal index distribution. The prefix is also considered *user data dependent*.
Only the first balancing position index found e is taken into account for this analysis. The balancing symbol index s is irrelevant for the probability estimation. The transmitted index is equal to e within the range $0 \leq e < n$, thus no distinction is made from now. Its maximum value is $n - 1$, since the encoder never inverts the whole sequence. Some distributions of the first balancing position index for different values of q & n can be seen in the Figure 3-6.

The format modification over e is made as $1 \leq e \leq n \ \forall \ q = 2$ in [10], for practical purposes. Such change does not influence the results significantly. Binary sequences fall into the category of Dyck words of length n_{even}, given their symmetrical and bipolar nature. Thus, the combinatorial behavior of their index distribution is described by the n-th Catalan Number. Therefore, the following equation is applied in this particular case:

$$Pr_1(2j) = Pr_1(2j-1) = \frac{n-2j+1}{n2^{n-2}} \begin{pmatrix} 2(j-1) \\ j-1 \end{pmatrix} \begin{pmatrix} 2(n/2-j) \\ n/2-j \end{pmatrix} \quad \forall \quad 1 \leq j \leq n/2. \tag{3-6}$$

Where $Pr_1(2j)$ is the probability for the transmitted index e to equal $2j$.
Unfortunately q-ary codewords follow other combinatorial rules. Nevertheless, some peculiarities about the index distribution remain constant $\forall \ q \geq 2$, and they can be observed in the Matlab calculations from the Table B-1. Sets of q indices have the same probability for codeword lengths of the form $n = qm \ \forall \ m \geq 1$, denoted by $Pr_1(qj - i)$, where $1 \leq j \leq n/q$ and $0 \leq i \leq q - 1$ if adopting the format $1 \leq e \leq n$. Notice that binary sequences also meet this condition. The entropy of the transmitted index for q-ary sequences is computed as:

$$H_e(n) = -\sum_{e=0}^{n-1} Pr_1(e) * \log_q(Pr_1(e)) \tag{3-7}$$

Where $Pr_1(e) = \frac{N(e)}{q^n}$, and $N(e)$ is the number of occurrences of e.
Figures 3-7, 3-8 and 3-9 a) show the entropy $H_e(n)$ for different values of q. The entropy

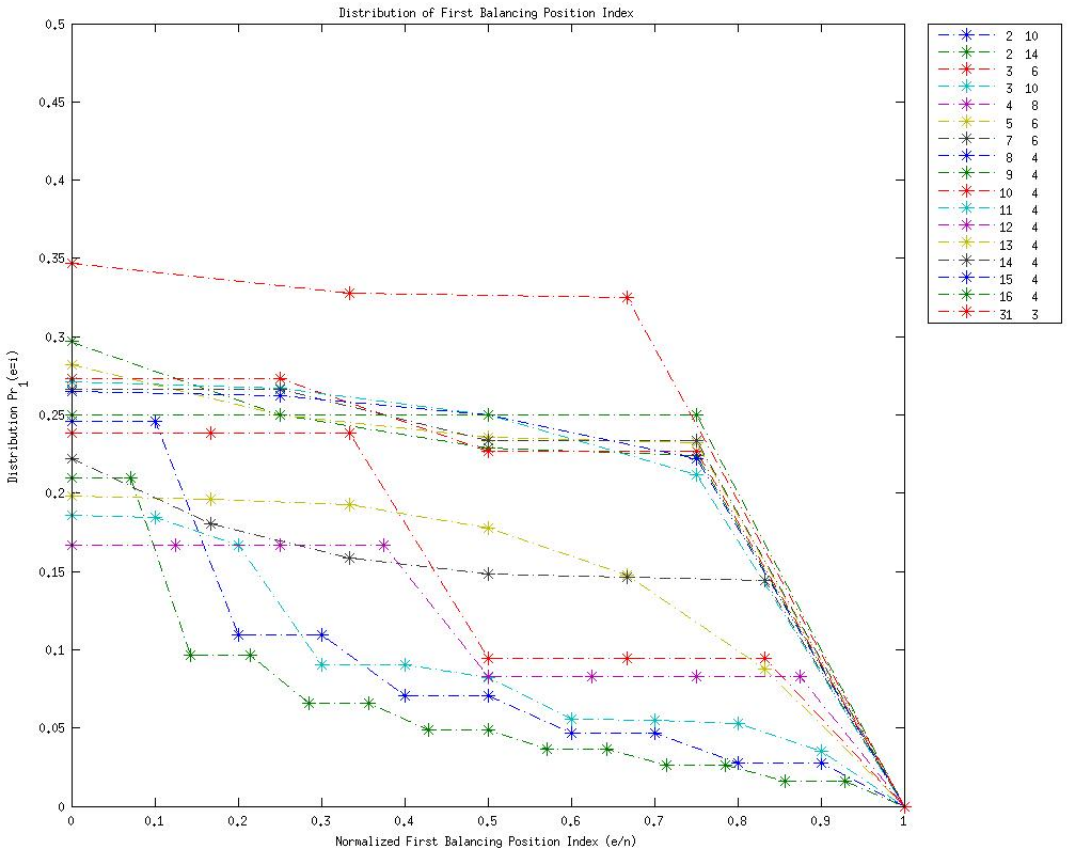

Figure 3-6: Distribution of the first balancing position index: $Pr_1(e)$ vrs e/n for different values of q & n, where $0 \leq e < n$ (based on the data from Table B-1).

$H_e(n)$ is marginally one unit lower than $\log_q(qn)$ for any q. But $H_e(n)$ does not consider the unit necessary to express the balancing symbol s. Thus, the reduction of redundancy is negligible in this scheme for the cases under evaluation.

3-2-2 Review of Auxiliary Information Encoding

This section reviews whether or not the selection of balancing positions is used, to depict additional information to reduce redundancy. As known, more than one balancing position per sequence can achieve balance. The prefix ρ carries on the balancing position by default. The number of balancing positions is variable among codewords, which are defined by the user data. The maximum number of balancing positions is $n \; \forall \; q > 2$, which doubles that of binary codes $(n/2)$, as shown by the Matlab calculations from Table B-1. The average amount of information handed over by the selection of v is expressed as:

$$H_a(n) = \sum_{v=1}^{n} Pr_2(v) \log_q(v) \tag{3-8}$$

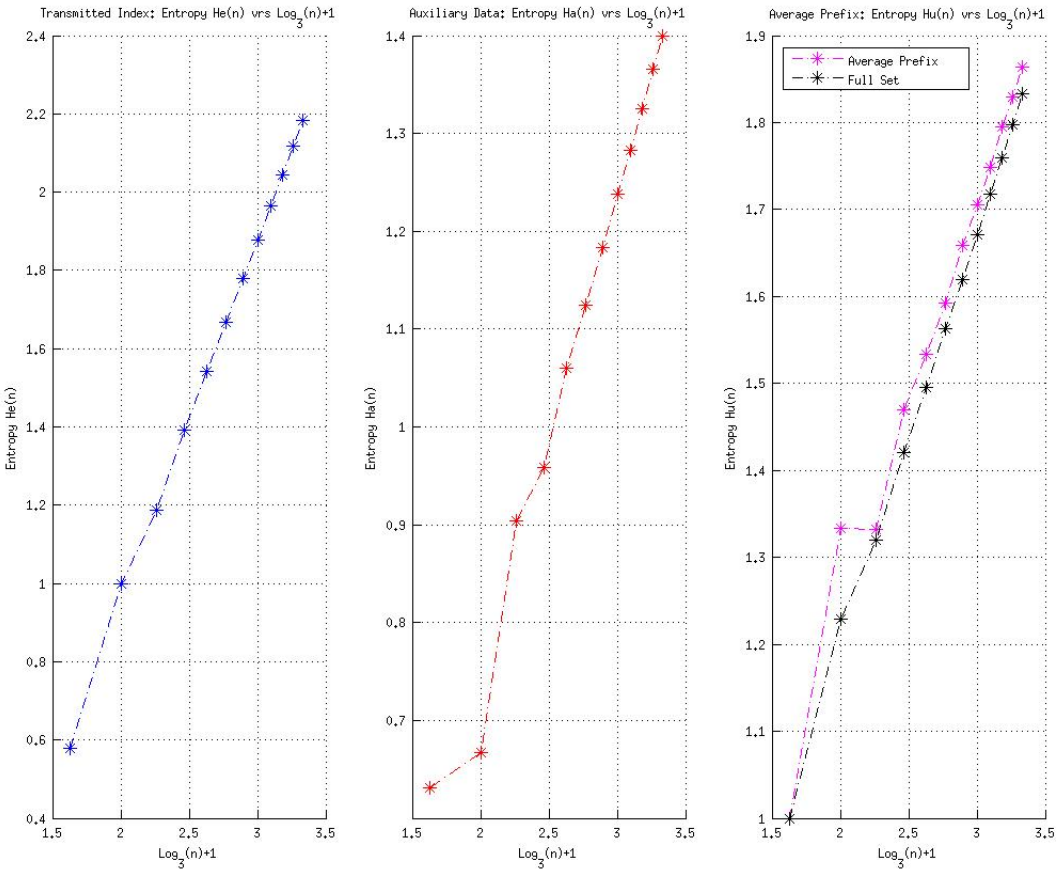

Figure 3-7: Entropy of a) Transmitted Index $H_e(n)$, b) Auxiliary Data $H_a(n)$, and c) Average Prefix $H_u(n)$, versus $\log_3(n)+1$ for $q=3$ ($n_{max}=13$).

Where $Pr_2(v) = N(v)/q^n$ is the occurrence probability of the balancing position v.
The approximation $H_a(n) \approx \frac{1}{2}\log_2(n) - 0.916$ is made in binary, but codewords obey different distributions $\forall\ q > 2$. The Figures 3-7, 3-8 and 3-9 b) show the entropy $H_a(n)$ for different values of q. The entropy $H_a(n)$ is lower than half $\log_q(qn)$ for the studied cases. Thus, the auxiliary data obtained from the balancing position selection compensates the excess of Knuth-based code's redundancy, in comparison with that of the full set of balanced codewords.

3-2-3 A Better Variable-Length-Prefix Construction

This section retakes the strategy explored in [9] and extend it to q-ary sequences, aiming to rectify the previous attempt of implementing a variable-length-prefix scheme to reduce redundancy. Encoding and decoding are addressed more efficiently this time. Recalling some definitions: "The total number of sequences associated to the same balanced codeword in a full set is denoted as $d(\mathbf{y})$". For instance, the balanced codeword $102 \in \mathcal{A}_3$ is related to any of the sequences $\{002, 022, 102\}$, thus $d(102) = 3$.
In general, the minimum value $d_{min}(\mathbf{y}) = q\ \forall\ q \geq 2$. The upper bound is variable for $q > 2$,

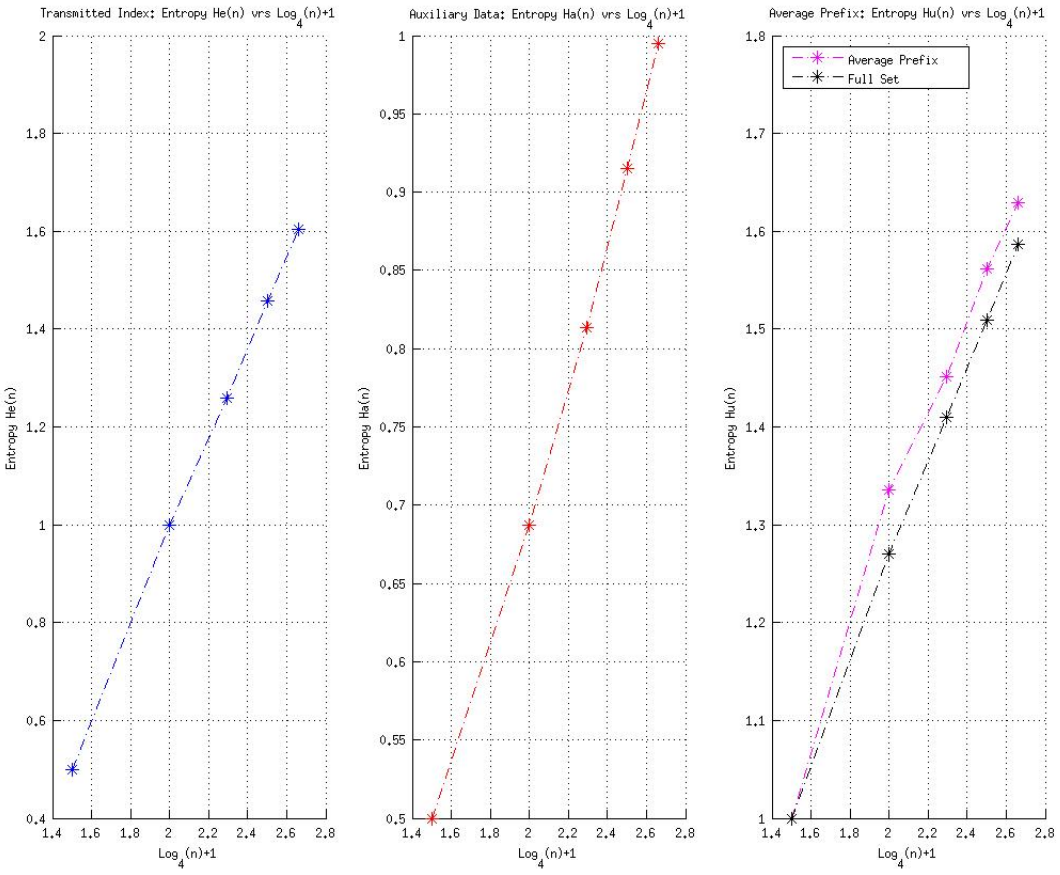

Figure 3-8: Entropy of a) Transmitted Index $H_e(n)$, b) Auxiliary Data $H_a(n)$, and c) Average Prefix $H_u(n)$, versus $\log_4(n)+1$ for $q=4$ ($n_{max}=10$).

thus its maximum value is $d_{max}(\mathbf{y}) = qn$. Both bounds can be seen in the Matlab calculations from Table B-3.
Extending the definition in [9] to q-ary codes: "$P(u,n)$ is the number of balanced codewords $\mathbf{y}$ of length n with $d(\mathbf{y}) = u$, such that $\sum_{u=q}^{qn} P(u,n) = \mathcal{S}_q^n$". Consequently, the amount of information necessary to represent the prefix is given by:

$$H_u(n) = q^{-n} \sum_{u=q}^{qn} uP(u,n) log_q(u) \tag{3-9}$$

The Figures 3-7, 3-8 and 3-9 c) show the expression above for different values of q. The redundancy of the full set of balanced codes $\in \mathcal{A}_q$ of length n, $H_0(n) = n - log_q(\mathcal{S}_q^n)$, is also plotted, where $\mathcal{S}_q^n$ is calculated using the Equation 3-4 with $\mathcal{O}\left(\frac{1}{n}\right) = 0$. Finally, $H_u(n)$ is slightly greater than $H_0(n)$ for any value of q. Thereupon for the cases considered, this scheme almost fully compensates the loss of redundancy, in comparison with that of the full set of balanced codewords.

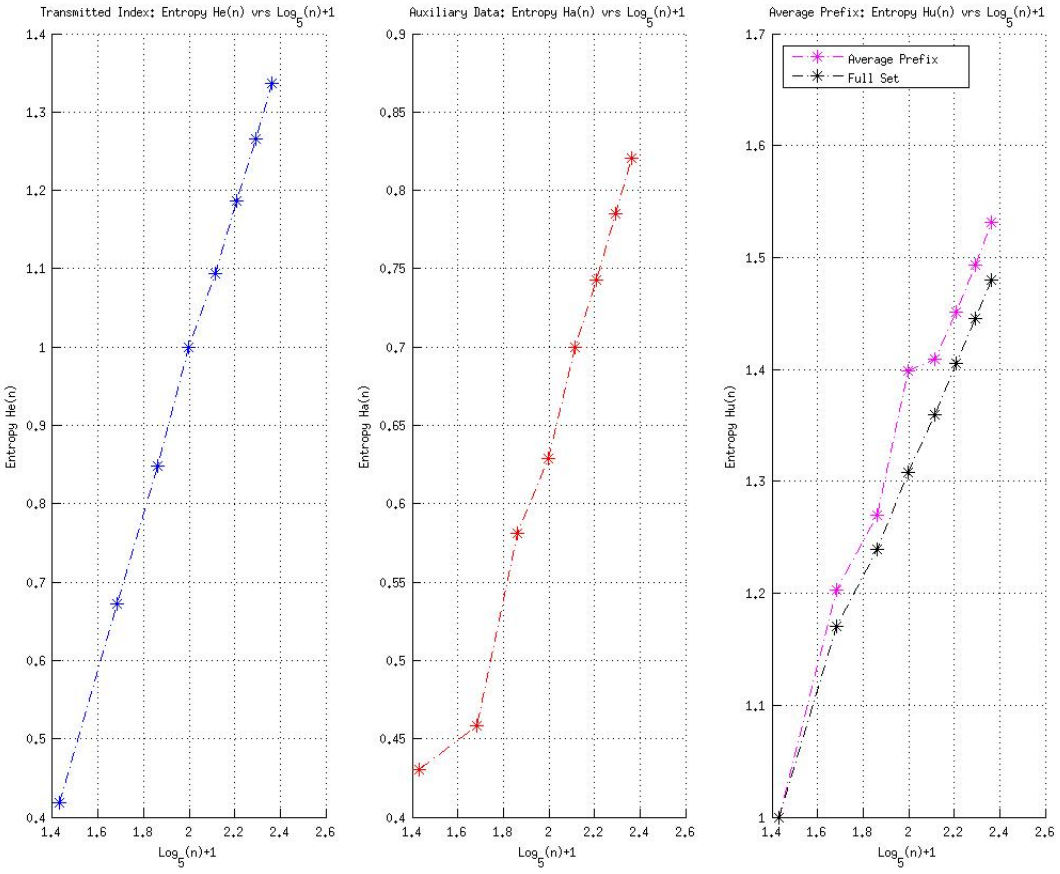

Figure 3-9: Entropy of a) Transmitted Index $H_e(n)$, b) Auxiliary Data $H_a(n)$, and c) Average Prefix $H_u(n)$, versus $\log_5(n) + 1$ for $q = 5$ ($n_{max} = 9$).

Chapter 4

Concatenated Codes

This chapter aims to apply the principles from [14] and extend them to larger alphabets than binary.
Initially, the balanced block length $2l$ equals the summation of l-number of ones and zeros per block, due to the symmetry of balanced binary vectors ($\beta = l$). However, this parameter lacks of meaning for larger values of q. Besides, balanced sequence lengths over an odd alphabet (q_{odd}) are not necessarily even. Thus the block length is denoted as n hereafter, according to the terminology previously used within this text.
The main idea extracted from the proposal by van Tilborg and Blaum in [14] is the construction of prefix-less balanced codewords, conformed by the concatenation of a fixed number N of balanced blocks of length n. Extending the definition to any q & n with cardinality $\mathcal{S}_q^n$:
"$\mathcal{S}_q^n$ blocks are taken as symbols in an alphabet. An error correcting code of length N is built, securing a minimum distance of $t+1$ over that alphabet. Additionally, the distance among those $\mathcal{S}_q^n$ blocks is at least 2 (balanced set feature by default). Hence, the balanced codewords created have a minimum distance of $2t+2$, which is enough to correct up to t errors".
Five different codeword constructions are exhibited in [14], involving similar procedures and principles, i.e. $2.1 - 2.5$. However, only Constructions from 2.2 to 2.5 are referred in the next sections, since construction 2.2 is an improvement over 2.1.
The expression for the balanced set distribution U from [14] is extended to q-ary sequences such that:

$$\mathbf{x'}(i+L) = (q-1) * \mathbf{1} \ominus_q \mathbf{x'}(i) \quad \forall \quad 0 \leq i < L = \lfloor \mathcal{S}_q^n / 2 \rfloor \tag{4-1}$$

An adjustment over L is necessary because the cardinality $\mathcal{S}_q^n$ is always odd for odd values of q. Thus, the remaining vector not included in the first $2L$ codewords is moved at the end of U, as shown in the position 6 from Figure 4-1 a). The same principle prevails: two groups of L members, in which any two vectors with the maximum achievable distance are separated L positions from each other. The last element of U is only considered by Construction 2.3 in the event of $\mathcal{S}_q^n$ being a prime power. All other constructions implicitly discard this element by partially using the balanced set.

Two of the valuable features introduced by binary concatenated codes of length $2l$ are:

U	P
012	0
021	1
102	2
210	3
201	4
120	5
111	6

$GF(9)$	$GF(3)$	U
0	00	08
1	10	17
2	01	26
3	12	35
4	22	80
5	20	71
6	02	62
7	21	53
8	11	44

$GF(4)$	$GF(2)$
0	00
1	10
2	01
3	11

Figure 4-1: a) Balanced Set U, for $q = 3$, $n = 3$ and $L = 3$. b) Extension field over $GF(3^2)$, through the primitive polynomial $x^2 + x + 2$, balanced set U for $q = 9$ and $n = 2$. c) Extension field over $GF(2^2)$, through the primitive polynomial $x^2 + x + 1$.

- Restrict the *Runlength* of a bit to $2l$ at most, term defined in [17] as "the length of time usually expressed in channel bits between consecutive transitions", i.e. the runlengths in the sequence 100111000011111 are 1, 2, 3, 4 and 5.
- Constraint the maximum *Digital Sum Variation (DSV)* to l, measure of the DC component in a code. Where the DSV denoted as z_i of a bipolar sequence $\{x_i\} = \{x_1, ..., x_i, ..., x_n\}$, with $x_i \in \{-1, 1\}$, is defined by:

$$z_i = \sum_{i=1}^{n} x_i = z_{i-1} + x_i \tag{4-2}$$

  Where $z_0 = 0$ and the DSV should be kept under reasonable bounds. e.g. after replacing the logic low & high values $\{0, 1\}$ with $\{-1, +1\}$, the vector $\mathbf{x} = 011000111100$ is depicted as $\mathbf{x} = -1, +1, +1, -1, -1, -1, +1, +1, +1, +1, -1, -1$, thus its accumulated charge (DSV) is $z = -1, 0, 1, 0, -1, -2, -1, 0, 1, 2, 1, 0$.

Constructions from 2.2 to 2.5 in [14] are briefly explained and extended for $q \geq 2$, some Tables are shown in Appendix C ($n \geq 4$ for binary to ensure $d_{min} = 2$).

4-1 Construction 2.2

This configuration only contemplates the first $2L$ elements from the balanced set U, after performing the distribution from 4-1. N elements from $U : \{0, 1, ..., 2L - 1\}$ are selected into the subset $I : \{i_0, i_1, ...i_{N-1}\}$, such that the sum of its symbols results zero over the ring $\mathbb{Z}_L$, i.e. $I : \{i_0 \oplus_L i_1 \oplus_L ... \oplus_L i_{N-1} = 0\}$. The first $N - 1$ elements of I are chosen the same as the information $\in \mathcal{A}_{2L}$. The N^{th} element is a check symbol over $\mathbb{Z}_L$ to guarantee a minimum distance of $t + 1$ in the set. The symbols from I are the positions of the vectors in U of which the final codeword is composed. The overall hamming distance of the code is fixed $d = 2 * 2 = 4$, and it is given by the minimum distance of U and the selection of I. Thus, this configuration is capable to correct a single error.

The code rate is calculated as $R \approx log_q((2L)^{N-1})/(nN)$. The numerator $\lfloor log_q((2L)^{N-1}) \rfloor$ is the number of elements from the input $\mathbf{u} \in \mathcal{A}_q$, required to represent an information vector $\mathbf{u} \in \mathcal{A}_{2L}$ of length $N-1$. The denominator nN is the total length of the codeword.
We review a simple example for $q = 3$, $n = 3$, and $N = 4$. The cardinality for this case is $\mathcal{S}_3^3 = 7$, thus $L = \lfloor \mathcal{S}_3^3/2 \rfloor = \lfloor 7/2 \rfloor = 3$. For instance, if the message is 2222 the alphabet conversion is carried on as $\mathbf{u} = 2222 \in \mathcal{A}_3 \rightarrow 80 \rightarrow \mathbf{u} = 212 \in \mathcal{A}_6$ (80 is the decimal value), leading to the first $N-1 = 4-1 = 3$ elements of $I : \{2, 1, 2\}$. The forth symbol is $i_3 = 1$, since $2+1+2+1 = (6) mod_3 = 0$. This configuration grants the freedom of selecting i_3 equal to 1 or $1 + L = 1 + 3 = 4$, i.e. $I = \{2, 1, 2, 1\}$ or $I = \{2, 1, 2, 4\}$. Choosing $i_3 = 1$, then swapping the elements of I for those of U as displayed in the Figure 4-1 a), the vector $\mathbf{x} = 102\ 021\ 102\ 021$ is formed. If an error takes place in the second position, $\mathbf{y} = 1\underline{1}2\ 021\ 102\ 021$, the decoder will detect an unbalance in the first triplet of elements and correct the error by performing inverse operations. There is a pair of possible solutions, i.e. $I : \{1 \oplus_3 2 \oplus_3 1 \oplus_3 \underline{2} = 1 \oplus_3 2 \oplus_3 1 \oplus_3 \underline{5} = 0\}$, but 112 is closer to 102 than to 120 (positions 2 & 5 in U), thus the decision is made in favor of 102.
The resulting code rate is $R = \lfloor log_3((2 * 3)^{4-1}) \rfloor / (3 * 4) = 0.333$.

4-1-1 Further Improvements

Van Tilborg and Blaum also present a potential improvement to increase error correction capabilities. The balanced set is arranged in groups with variable number of codewords named classes, and maximum distance achievable among the members of each class. The variable L is redefined as the number of classes in U. The parity check operation is applied over the classes to lead zero in $\mathbb{Z}_L$. In other words, a codeword is integrated by vectors, in such a way that the sum of the classes to which they belong results zero over $\mathbb{Z}_L$.

4-2 Construction 2.3

In this configuration N vectors from U are chosen as symbols from an alphabet $\{0, 1, ...Q-1\}$, where $Q = p_r^m$ is the largest prime power such that $\mathcal{S}_q^n \geq Q$. Any block code $\mathcal{C}'$ of the form $(mN, k, t+1)$ over $GF(p_r)$ can be used. Finally, the elements of $\mathcal{C}$ are the positions of the vectors from U. Their mapping is carried on as $\mathcal{C}' \rightarrow \mathcal{C} \in \mathbb{F}_Q^N$ by extension fields if $m > 1$.
The overall minimum distance of the code is $d_{min} = 2(t+1)$, derived from the combination of $\mathcal{C}'$ with the minimum distance in U. The code rate is determined as $R \approx log_q(Q^k)/(nN)$. Similarly, the numerator is the number of data elements $\in \mathcal{A}_q$, representing k symbols $\in \mathcal{A}_Q$.
We take a look at the case $q = 9$, $n = 2$, $N = 4$ and cardinality $\mathcal{S}_9^2 = Q = 9$. For instance, if a message is $\mathbf{u} = 46 \in \mathcal{A}_9 \rightarrow 42 \rightarrow \mathbf{u} = 1120 \in \mathcal{A}_3$, a 2-tuple vector over $GF(3)$ is necessary for the extension field $GF(9) = GF(3^2)$, thus a block code twice the size of N is build. The vector $\mathcal{C}' = 22201120 \in \mathbb{F}_3^8$ is created by applying the Hamming Code $(8, 4, 3)$ over $GF(3)$. The use of the extension fields from Figure 4-1 b) to express $\mathcal{C}'$ as symbols of Q leads to a final output of $\mathcal{C} = 4585 \in \mathbb{F}_9^4$. The vector $\mathbf{x} = 80714471$ is generated by replacing the elements of $\mathcal{C}$ for those of U, as displayed in the Figure 4-1 b).
This code $\mathcal{C}'$ guarantees a minimum distance of 3 by default. The overall minimum distance of the code is $d = 2 * 3 = 6$, correcting up to $t = (6-2)/2 = 2$ errors. The code rate is

calculated as $R = log_9(3^4)/(2*4) = 0.25$.
Notice that this construction does not depend on the distribution of the balanced set U.

4-3 Construction 2.4

This construction is intended for cardinalities which can be expanded as factors of prime powers, such that $\mathcal{S}_q^n = Q_0$ x Q_1 x Q_2 x ... x Q_{j-1}, where $Q_j = p_{r_j}^{m_j}$ is a prime power, and j is the number of factors. Therefore, $j*N$ codewords from U are chosen, building j block codes of length N over $GF(Q_0)$, $GF(Q_1)$, $GF(Q_2)$, ..., $GF(Q_{j-1})$ respectively, likewise Construction 2.3. Originally in Constructions $2.3-2.5$ van Tilborg and Blaum proposed the mapping from bits to any other alphabet. This technique is valid for any q, such that $\log_q(Q_j^{k_j}) \geq 1$, requirement usually defined by the parameter N.
Minimum distances are set up independently by every block code, $d_0, d_1, d_2, ..., d_{j-1}$, but the reference is based on the smallest of them all for simplicity. The overall distance of the code is calculated as $d = 2*min\{d_0, d_1, d_2, ..., d_{j-1}\}$, where the factor 2 comes from the minimum distance of U. The code rate is specified by $R \approx log_q(\prod_{i=0}^{j-1} Q_i^{k_i})/(nN)$, analogously to Construction 2.3.
For instance, the cardinality in case $q = 2$ and $n = 10$ is expanded as $\mathcal{S}_2^{10} = 252 = 4$ x 7 x 9. Blocks of length $N = 4$ are made via extension fields, by applying the Hamming Codes: $(8,4,3)$ over $GF(2)$, $(8,4,3)$ over $GF(3)$ and $(4,2,3)$ over $GF(7)$. If the message is $\mathbf{u} = 111011111100001$ the following codes are generated: $\mathcal{C}_0' = 10011110 \in \mathbb{F}_2^8$, $\mathcal{C}_1' = 00022100 \in \mathbb{F}_3^8$ and $\mathcal{C}_2 = 6501 \in \mathbb{F}_7^4$. Applying transformations over $\mathcal{C}_0'$ and $\mathcal{C}_1'$ by the extension fields from Figures 4-1 b) & c), we obtain $\mathcal{C}_0 = 1231 \in \mathbb{F}_4^4$ and $\mathcal{C}_1 = 0670 \in \mathbb{F}_9^4$. Finally, the code $\mathcal{C}$ of which the elements indicate the positions of the vectors in U is formed as $\mathcal{C} = \mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2 = 123106706501$.
The code rate and minimum distance are $R = \lfloor log_2(4^2 * 7^2 * 9^2) \rfloor/(10*4) = 0.375$, and $d = 2*min\{3,3,3\} = 6$ respectively. Thus, $t = (6-2)/2 = 2$ errors can be corrected.
Sorting out U to increase minimum distance is irrelevant for this configuration, unless $max\{Q_j\}$ codewords with greater distance among them are arranged.

4-4 Construction 2.5

This configuration was initially contemplated for binary alphabet and its even cardinalities, however it can be extended to even values of q. This construction equals 2.4 in essence with an additional implementation.
Once the cardinality is expanded as $\mathcal{S}_q^n = Q_0$ x Q_1 x Q_2 x ... x Q_{j-1}, where $Q_0 = 2^{m_0}$, if:

- $m_0 = 1$, a parity check code of the form $(N, N-1, 2)$ is build.
- $m_0 > 1$, two binary codes of the form $(N, N, 1)$ are build.

The distance of the binary codes is optimal, since only two vectors from U are required. Thus, any pair of codewords $\mathbf{x}$'$(i+L)$ and $\mathbf{x}$'(i) with maximum distance $d = n(q-1)$ is chosen. The overall distance of the code is calculated as $d = min\{n(q-1)*d_0, 2*d_1, 2*d_2, ..., 2*d_{j-1}\}$,

where d_0 is either 2 if $m_0 = 1$, or 1 otherwise.
For instance, the cardinality is expanded as $\mathcal{S}_2^4 = 6 = 2$ x 3 in case $q = 2$, $n = 4$. If blocks of length $N = 5$ are set up, the following codes are built: binary parity check $(5, 4, 2)$ and $Ham(3, 3) : (5, 2, 3)$. If message is $\mathbf{u} = 1011101$ the codes: $\mathcal{C}_0' = 10111 \in \mathbb{F}_2^5$ and $\mathcal{C}_1 = 00112 \in \mathbb{F}_3^5$ are created. The binary code $\mathcal{C}_0'$ is modified as $\mathcal{C}_0 = L * \mathcal{C}_0' = 30333$ to ensure the selection of the vectors $\mathbf{x}$'$(0 + 3)$ and $\mathbf{x}$'(0) in U $(L = 3)$.
The code rate and distance are $R = log_q(\prod Q^k)/(nN) = log_2(2^4 * 3^2)/(4 * 5) = 0.35$ and $d = min\{2 * 4, 2 * 3\} = 6$ respectively. Thus, $t = (6 - 2)/2 = 2$ errors can be corrected.
Another example is for $q = 4$ & $n = 4$, with cardinality expressed as $\mathcal{S}_4^4 = 44 = 4$ x 11. If blocks of length $N = 6$ are assembled the following codes are built: 2 x binary $(6, 6, 1)$ and $Ham(11, 2) : (6, 4, 3)$. The message is $\mathbf{u} = 132210332211 \in \mathcal{A}_4$ and its first six elements are converted to binary, creating the codes: $\mathcal{C}_0' = 011110$ and $\mathcal{C}_1' = 100100$, both $\in \mathbb{F}_2^6$. The codes $\mathcal{C}_0'$ and $\mathcal{C}_1'$ are modified since $L = 44/2 = 22$, such that $\mathcal{C}_0 = 22 * \mathcal{C}_0' = (0, 22, 22, 22, 22, 0)$ and $\mathcal{C}_1 = 22 * \mathcal{C}_1' = (22, 0, 0, 22, 0, 0)$ respectively. Thus, the selection of the vectors $\mathbf{x}$'(22) and $\mathbf{x}$'(0) in U is settled. The last code is applied over the last six elements of $\mathbf{u}$, i.e. $\mathcal{C}_2 = 613011 \in \mathbb{F}_{11}^6$ $(332211 \in \mathcal{A}_4 \rightarrow 4005 \rightarrow 3011 \in \mathcal{A}_{11})$.
The code rate is $R = 12/(4 * 6) = 0.5$. The distance between $\mathbf{x}$'(22) and $\mathbf{x}$'(0) is $d = n(q-1) = 4 * (4 - 1) = 12$ or $d(\mathbf{x}'(22), \mathbf{x}'(0)) = d(0033, 3300) = 12$. The minimum distance of the code is determined as $d = min\{12, 2 * 3\} = 6$. Thus, $t = (6 - 2)/2 = 2$ errors can be corrected.
The Table 4-1 shows the feasibility of Constructions from 2.2 to 2.5 for different values of q and n. The selection criteria is outlined as follows:

- Cardinalities below 10^5 to ensure low use of memory resources for Construction 2.2.
- Prime powers below 10^3 to secure smooth block code constructions over $GF(Q_j)$.
- Block lengths n's less or equal to 10 to keep a good ratio between t and R.
- Alphabet values q's below 20 to fulfill all of the above.
- Cardinalities expanded by more than one factor for Constructions 2.4 and 2.5.
- Values of N that allow $\log_q(Q_j^{k_j}) \geq 1$ for Constructions $2.4 - 2.5$.
- Even alphabets q_{even} for Construction 2.5.

q	n	Cardinality	Greatest PP.	Factors	2.2	2.3	2.4	2.5
2	4	6	5	2,3	x	x	x	x
2	6	20	19	4,5	x	x	x	x
2	8	70	67	2,5,7	x	x	x	x
2	10	252	251	4,9,7	x	x	x	x
3	2	3	3	3	x	x		
3	3	7	7	7	x	x		
3	4	19	19	19	x	x		
3	5	51	49	3,17	x	x	x	
3	6	141	139	3,47	x	x	x	
3	7	393	389	3,131	x	x	x	
3	8	1107	1103	27,41	x		x	
3	9	3139	3137	43,73	x		x	
4	2	4	4	4	x	x		x
4	4	44	43	4,11	x	x	x	x
4	6	580	577	4,5,29	x	x	x	x
4	8	8092	8089	4,7,289	x		x	x
5	2	5	5	5	x	x		
5	3	19	19	19	x	x		
5	4	85	83	5,17	x	x	x	
5	5	381	379	3,127	x	x	x	
5	6	1751	1747	17,103	x		x	
6	2	6	5	2,3	x	x	x	x
6	4	146	139	2,73	x	x	x	x
6	6	4332	4327	4,3,361	x		x	x
7	2	7	7	7	x	x		
7	3	37	37	37	x	x		
7	4	231	229	3,7,11	x	x	x	
8	2	8	8	8	x	x		x
8	4	344	343	8,43	x	x	x	x
9	2	9	9	9	x	x		
9	3	61	61	61	x	x		
9	4	489	487	3,163	x	x	x	
9	5	3951	3947	9,439	x		x	
9	6	32661	32653	9,19,191	x		x	
10	2	10	9	2,5	x	x	x	x
10	4	670	661	2,5,67	x	x	x	x
10	6	55252	55249	4,19,727	x		x	x
11	2	11	11	11	x	x		
11	3	91	89	7,13	x	x	x	
11	4	891	887	81,11	x	x	x	
11	5	8801	8783	13,677	x		x	
11	6	88913	88903	11,59,137	x		x	
12	2	12	11	4,3	x	x	x	x
12	4	1156	1153	4,289	x		x	x
13	2	13	13	13	x	x		
13	3	127	127	127	x	x		
13	4	1469	1459	13,113	x		x	
14	2	14	13	2,7	x	x	x	x
14	4	1834	1831	2,7,131	x		x	x
15	2	15	13	3,5	x	x	x	
15	3	169	169	169	x	x		
15	4	2255	2251	5,11,41	x		x	
15	5	30381	30367	3,13,19,41	x		x	
16	2	16	16	16	x	x		x
16	4	2736	2731	16,9,19	x		x	x
17	2	17	17	17	x	x		
17	3	217	211	7,31	x	x	x	
17	4	3281	3271	17,193	x		x	
18	2	18	17	2,9	x	x	x	x
18	4	3894	3889	2,3,11,59	x		x	x
19	2	19	19	19	x	x		
19	3	271	271	271	x	x		
19	4	4579	4567	19,241	x		x	
20	2	20	19	4,5	x	x	x	x
20	4	5340	5333	4,3,5,89	x		x	x

Table 4-1: Construction feasibility of Concatenated Codes for $q \leq 20$ and $n \leq 10$.

Chapter 5

Binary Codes Comparison

The aim of this section is the comparison between Knuth-based and Concatenated binary codes. The characteristics: Code Rate R and Digital Sum Variation (DSV) are evaluated. The analysis is based on the code constructions from the Tables 5-1 and 5-2, which are briefly explained along the following lines.

BCH Codes are selected to increase correction capabilities up to 2 errors in Knuth-based binary codes. Shortening over the codes: $(15, 7, 5)$, $(31, 21, 5)$, $(63, 51, 5)$, $(127, 113, 5)$ and $(255, 239, 5)$ is required to obtain even values of n to apply balancing. E.g. the code $(12, 4, 5)$ from the first line in Table 5-1 is determined by subtracting 3 units from n and k, i.e. $(15 - 3, 7 - 3, 5)$.

Prefix constructions are derived from the look-up tables based on the cardinalities of balanced codes with distance $d_2 \geq 4$ in Table 3-3 a). Therefore, only single error over the prefix can be corrected. Equal error correction capabilities for the prefix and the bulk segments ($d_2 = 5$) involved much longer calculation cycles by computer, leading to negligible changes over the DSV and the Code Rate for $n >> p$.

For instance, if the binary message is $\mathbf{u} = 1101$, the balanced vector $\mathbf{x} = 110100101010 \in \mathcal{C}_1$ is generated ($z = 0$) when applying the previous BCH code. Thus, the bulk has correction capabilities up to $t = (5-1)/2 = 4$ errors. The prefix $\rho = 00001111 \in \mathcal{C}_2$ represents the index $z = 0$, since $\mathcal{C}_2 : A_2(8, 4, 4) = 14 \geq n = 12$ from the Table 3-3 a) for single error correction $t = \lfloor (4 - 1)/2 \rfloor = 1$. The code rate for this example is $R = k/(p + n) = 4/(8 + 12) = 0.2$. The binary Knuth-based encoder is shown in the Appendix A-4.

The construction 2.3 is chosen due to its highest rates among all the Concatenated Code configurations. The hamming distance of Construction 2.3 is $d = 6$, given the multiplication of the distance from the Hamming codes by the minimum distance of the balanced set U. Correction up to 2 errors is always possible, however Matlab limitations occur if increasing error correction capabilities. Such software tool only offers block codes over binary finite fields $GF(2^m)$. The development of block code routines over greater prime powers, needed for $t > 2$, is out the scope of this dissertation.

We revise the construction for $l = 5$ and $N = 3$ in Table 5-2 as calculation sample. The parameter l has the related cardinality of $\mathcal{S}_2^{10} = 252$. Thus, the greatest prime power admissible is $Q = 251 < \mathcal{S}_2^{10}$. The Hamming Code $(3, 1, 3)$ over $GF(251)$ is applied for only one data

symbol. If the binary message is the all-ones vector $\mathbf{u} = \mathbf{1}$ of length $n = 7$ its decimal value $\mathbf{u} = 127 \in \mathcal{A}_{251}$ represents an unitary symbol, since $k = \lfloor \log_2(251^1) \rfloor = 7$ bits. Finally, the code rate is calculated as $R = 7/(2 * 5 * 3) = 0.233 \approx \log_2(Q^k)/(2lN)$. The remaining cases in the Table 5-2 are calculated in the same manner. The parameter N varies, thus diverse data lengths k are derived.

k	n	p	R
4	12	8	0.2
6	14	8	0.273
8	18	10	0.286
14	24	10	0.412
18	28	10	0.474
20	30	10	0.5
30	42	12	0.556
38	50	12	0.613
44	56	12	0.647
48	60	12	0.667
50	62	12	0.676
56	70	12	0.683
70	84	12	0.729
82	96	12	0.759
92	106	12	0.78

k	n	p	R
100	114	12	0.794
106	120	12	0.803
110	124	12	0.809
112	126	12	0.812
128	144	14	0.81
148	164	14	0.831
166	182	14	0.847
182	198	14	0.858
196	212	14	0.867
208	224	14	0.874
218	234	14	0.879
226	242	14	0.883
232	248	14	0.885
236	252	14	0.887
238	254	14	0.888

Table 5-1: Binary Knuth-based Code constructions, with correction capabilities up to 2 errors.

5-1 Code Rate Comparison

Binary Concatenated Codes (Construction 2.3) with double error correction capabilities for $l = 2, 3, 4$ & 5 are shown in the Table 5-2. The larger the size of l, the greater the code rate R. The rates of Knuth-based Codes are higher than those from Concatenated Codes for $l = 2$ and $k \geq 9$, as shown in the Figure 5-1 a). Nevertheless, a fair comparison starts approximately from $k \geq 18$, because only single error in the prefix is contemplated. The influence of prefix length over the code rate is minimal at that point, i.e. $R = k/(p + n) = 18/(14 + 28) = 0.429 \approx 0.474$ rather than with $p = 10$ as in Table 5-1, since $p = 14$ according to $A_2(14, 6, 7) = 42 > n = 28$ in [7] for correction up to $t = \lfloor (6 - 1)/2 \rfloor = 2$ errors. Similar assumption can be safely made for $l = 3$ in the Figure 5-1 b), where Knuth-based Codes have better performance for $k \geq 38$, $R = 38/(16 + 50) = 0.576 \approx 0.613$, instead of $k = 20$.
The rates of Knuth-based Codes are higher than those from Concatenated Codes at any time for $l = 4, 5$ in the Figures 5-2 a) & b). However, the comparison is fair for $k \geq 30$ in both cases, since $R = 30/(14 + 42) = 0.536 \approx 0.556$ (the prefix influence over R is marginal).

k	l	N	R
4	2	4	0.25
6	2	5	0.3
9	2	6	0.375
16	2	10	0.4
18	2	11	0.409
20	2	12	0.417
30	2	16	0.469
39	2	20	0.488
44	2	22	0.5
48	2	24	0.5
51	2	25	0.51
58	2	28	0.518
71	2	35	0.507
83	2	40	0.519
92	2	44	0.523
102	2	48	0.531
106	2	50	0.53
111	2	52	0.534
113	2	53	0.533
130	2	60	0.542
148	2	68	0.544
167	2	76	0.549
183	2	83	0.551
197	2	89	0.553
208	2	94	0.553
218	2	98	0.556
227	2	102	0.556
232	2	104	0.558
236	2	106	0.557
239	2	107	0.558

k	l	N	R
4	3	3	0.222
8	3	4	0.333
16	3	6	0.444
21	3	7	0.5
33	3	10	0.55
38	3	11	0.576
46	3	13	0.59
50	3	14	0.595
59	3	16	0.615
72	3	19	0.632
84	3	23	0.609
93	3	25	0.62
101	3	27	0.623
106	3	28	0.631
110	3	29	0.632
114	3	30	0.633
131	3	34	0.642
148	3	38	0.649
169	3	43	0.655
182	3	46	0.659
199	3	50	0.663
208	3	52	0.667
220	3	55	0.667
229	3	57	0.67
233	3	58	0.67
237	3	59	0.669

k	l	N	R
6	4	3	0.25
18	4	5	0.45
30	4	7	0.536
48	4	10	0.6
72	4	14	0.643
84	4	16	0.656
103	4	19	0.678
109	4	20	0.681
115	4	21	0.685
151	4	27	0.699
169	4	30	0.704
218	4	38	0.717
236	4	41	0.72

k	l	N	R
7	5	3	0.233
15	5	4	0.375
23	5	5	0.46
31	5	6	0.517
39	5	7	0.557
47	5	8	0.588
71	5	11	0.645
95	5	14	0.679
103	5	15	0.687
111	5	16	0.694
151	5	21	0.719
167	5	23	0.726
183	5	25	0.732
199	5	27	0.737
239	5	32	0.747

Table 5-2: Binary Concatenated Code Construction 2.3 for $l = 2, 3, 4$ & 5, with correction capabilities up to 2 errors.

5-2 Digital Sum Variation Comparison

This section is intended for the analysis of the accumulated charge from Knuth-based and Concatenated binary codes.
The message lengths of both codes k are restricted to differ by only 3 units between them in all the graphs. This tolerance is necessary given the distinct nature of their constructions. Moreover, the DSV is data dependable, so there are 2^k possible outcomes to plot out. However, this comparison is mostly narrowed down to critical cases with the highest disparity.
Some comparisons of the DSV between Knuth-based and Concatenated binary codes are shown in the Figures 5-3 and 5-4, for the message $\mathbf{u} = \mathbf{1}$ (all-ones vector) of different lengths k. As expected, the DSV of Concatenated Codes always resides between the bounds $\pm l$ for $l = 2, 5, 3$ & 4 regardless the codeword length $(2lN)$.
Furthermore, the DSV of the prefix's segment from the Knuth-based Codes can be easily distinguished in the beginning of each plot. Its impact over the overall DSV of codeword is minimal if $p << n$. The minimum disparity in each case approaches $-n/2$ if $p << n$, characteristic related to the way the balancing is applied over the bulk's segment.

The DSV comparisons between both codes for the message $\mathbf{u} = \mathbf{0}$ (all-zeros vector) of distinct lengths k are plotted out in Figures 5-5 and 5-6. Once more, the DSV of Concatenated Codes always fluctuates within the bounds $\pm l$ ($l = 5, 3, 4$ & 2) for any the codeword length. The influence of the prefix over the whole DSV from the Knuth-based Codeword decreases if $p << n$. The maximum disparity of each case equals $n/2$, because of the balancing applied to the payload.

The Figures 5-7 and 5-8 display several comparisons of the DSV between both codes for randomly chosen messages of different length. The DSV of Concatenated Codes remains in the bounds $\pm l$ ($l = 4, 2, 5$ & 3) in all the cases. The minimum and maximum disparities of Knuth-based Codes are less harsh than those of previous inputs (all-ones/zeros vectors). However, the variation of the disparities is uncontrolled.

The DSV of binary Knuth-based Codes oscillates within the range $(-n/2, n/2]$ for the studied information inputs. On the other hand, the DSV of binary Concatenated codes varies within the range $[-l, l]$ for all the reviewed messages, where $l << n/2$ for large values of n.
The price to pay for keeping the DSV between reasonable bounds is the decrease of code rates, as in Concatenated Codes. The Knuth-based Codes have indeed higher rates, but also have large disparities in some cases.

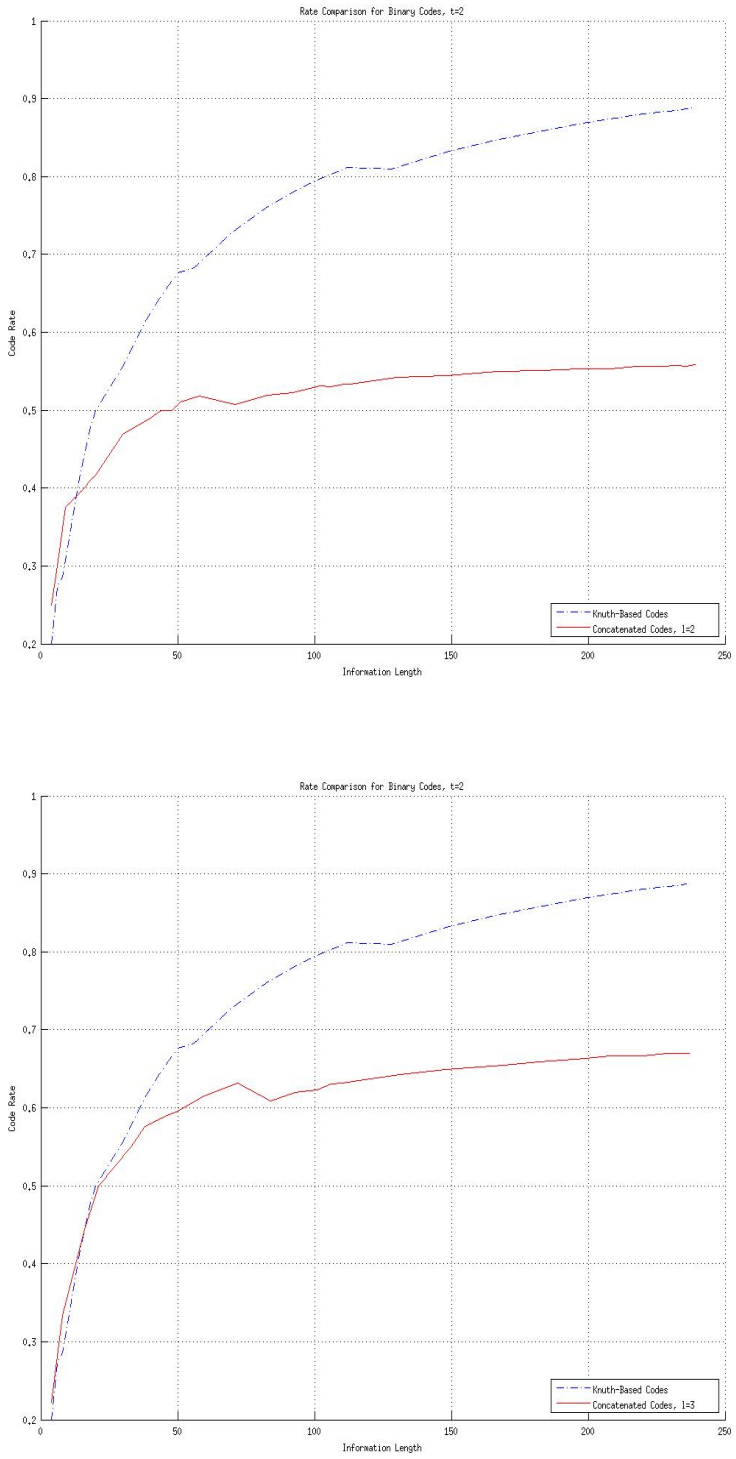

Figure 5-1: Rate of Binary Codes: a) Knuth-based & Concatenated (Construction 2.3 with $l = 2$), b) Knuth-based & Concatenated (Construction 2.3 with $l = 3$). For correction up to $t = 2$ errors, their parameters are listed on Tables 5-1 & 5-2.

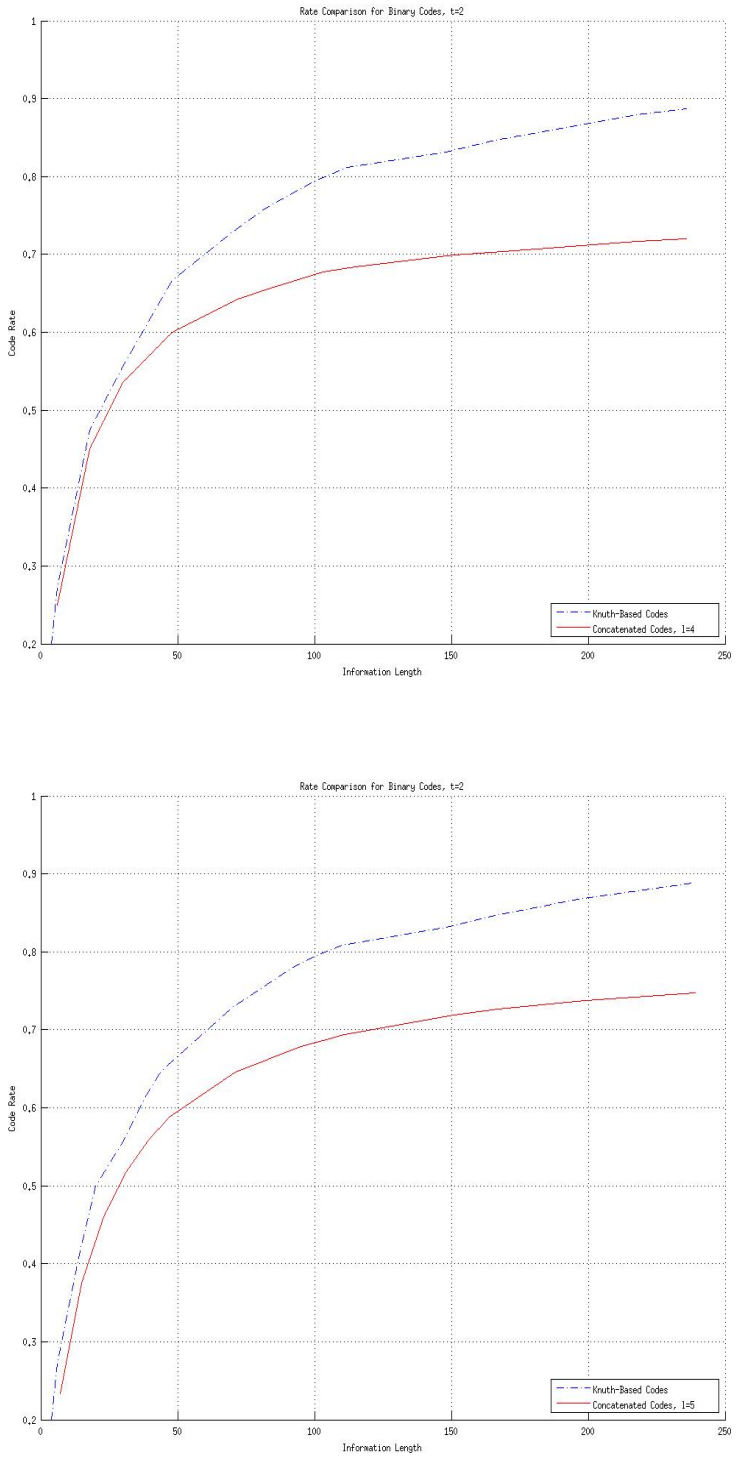

Figure 5-2: Rate of Binary Codes: a) Knuth-based & Concatenated (Construction 2.3 with $l = 4$), b) Knuth-based & Concatenated (Construction 2.3 with $l = 5$). For correction up to $t = 2$ errors, their parameters are listed on Tables 5-1 & 5-2.

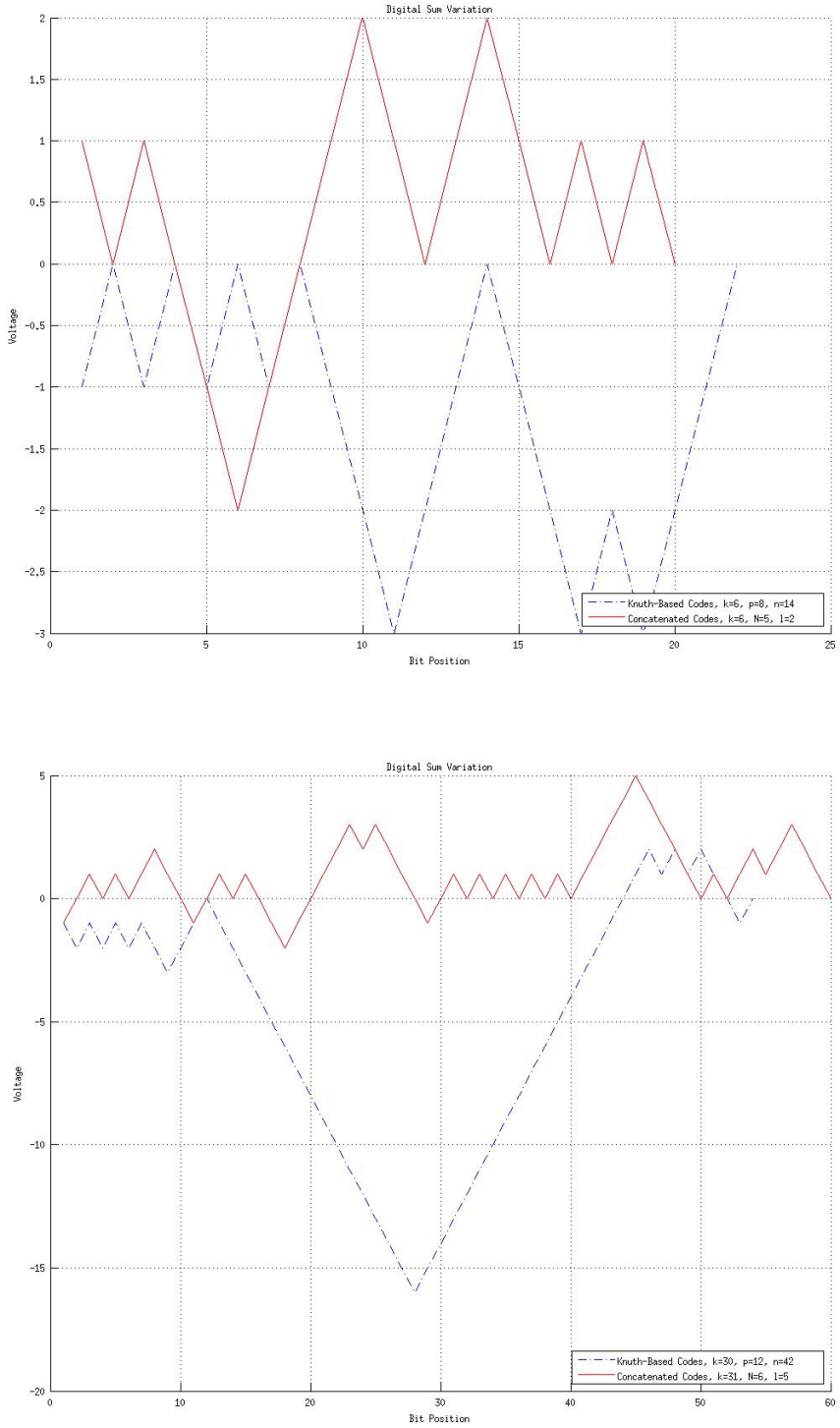

Figure 5-3: DSV of Binary Codes: a) Knuth-based $k = 6$, $p = 8$ & $n = 14$ and Concatenated $k = 6$, $N = 5$ & $l = 2$, b) Knuth-based $k = 30$, $p = 12$ & $n = 42$ and Concatenated $k = 31$, $N = 6$ & $l = 5$, for all-ones message.

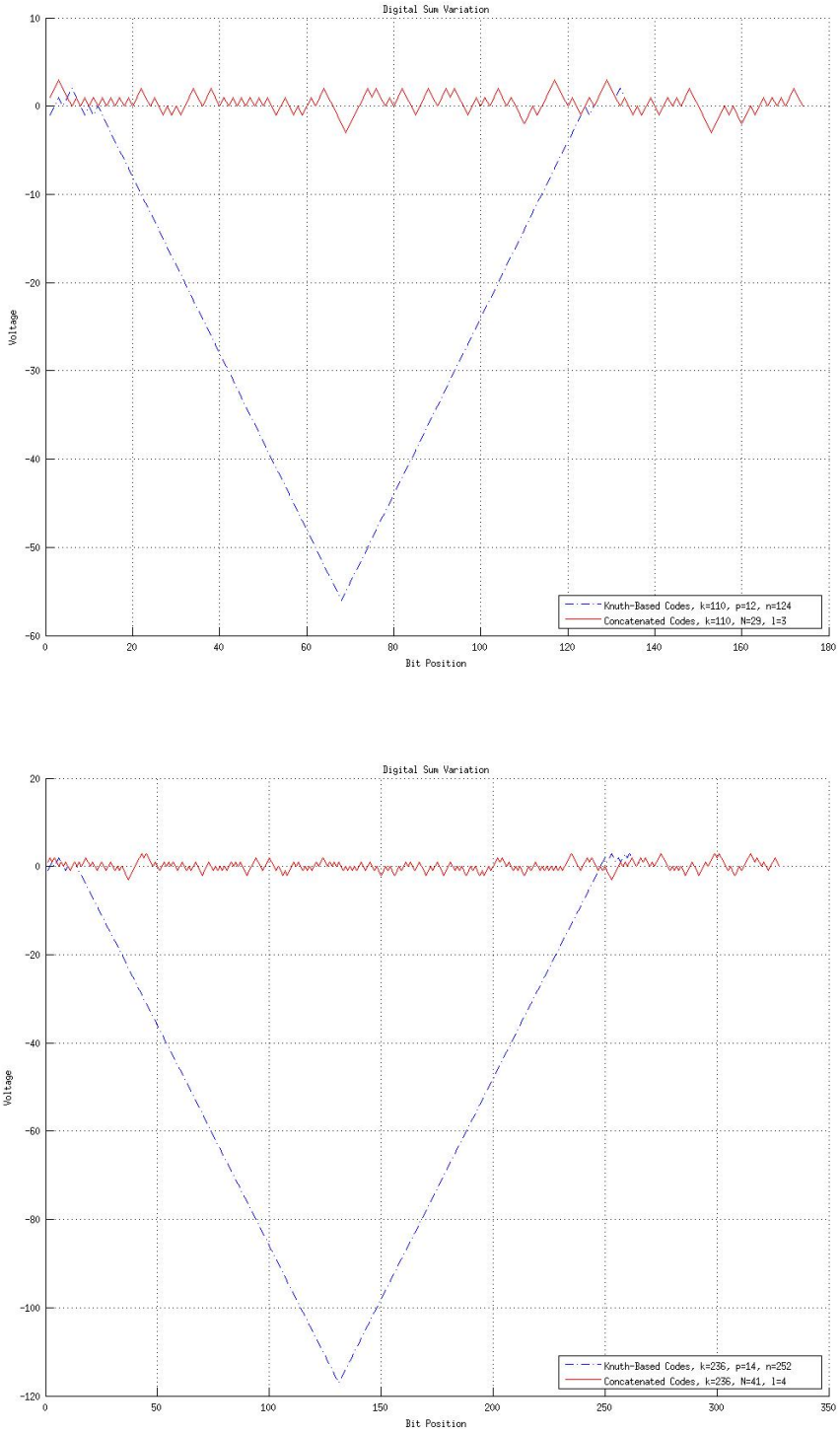

Figure 5-4: DSV of Binary Codes: a) Knuth-based $k = 110$, $p = 12$ & $n = 124$ and Concatenated $k = 110$, $N = 29$ & $l = 3$, b) Knuth-based $k = 236$, $p = 14$ & $n = 252$ and Concatenated $k = 236$, $N = 41$ & $l = 4$, for all-ones message.

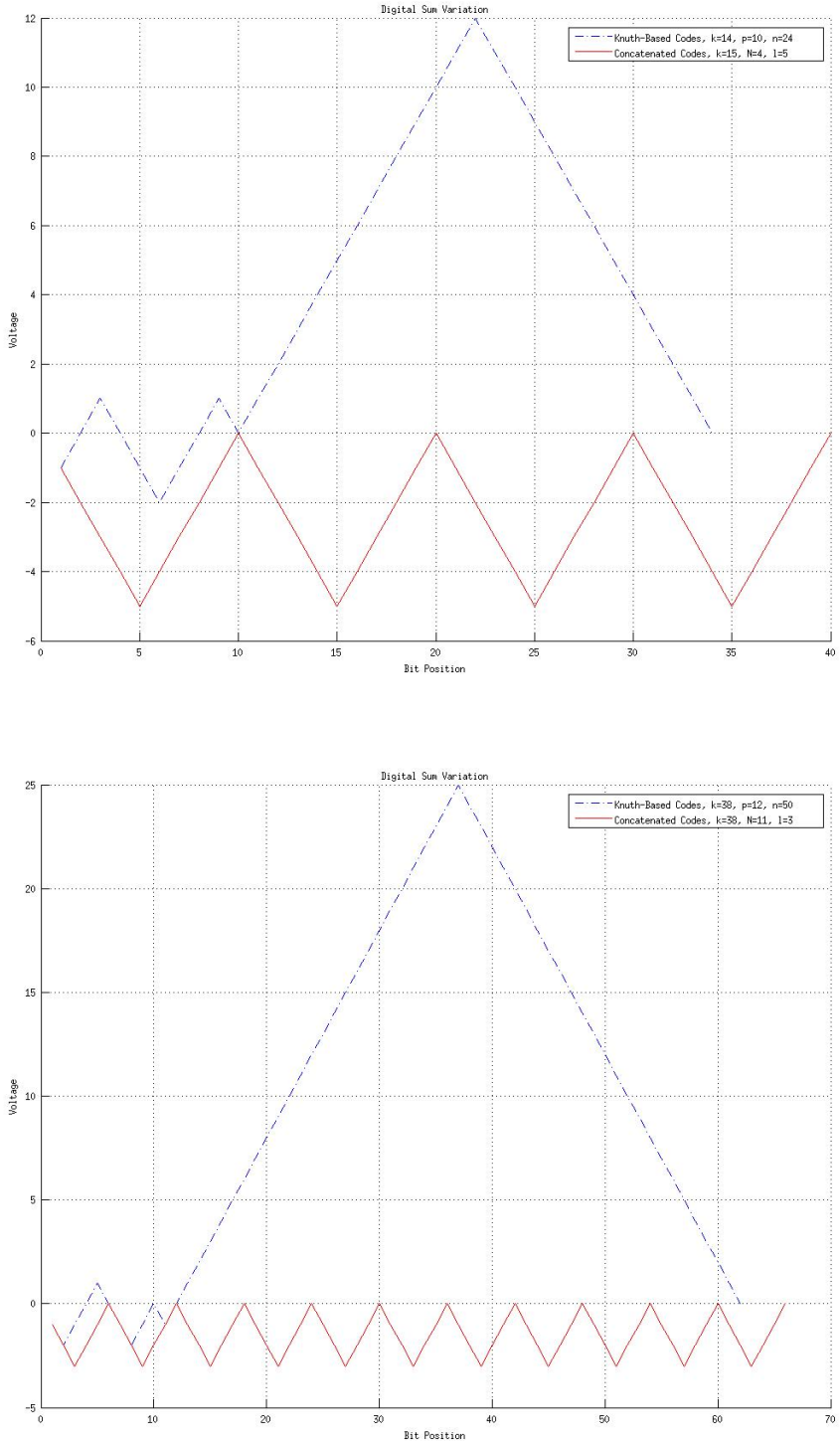

Figure 5-5: DSV of Binary Codes: a) Knuth-based $k = 14$, $p = 10$ & $n = 24$ and Concatenated $k = 15$, $N = 4$ & $l = 5$, b) Knuth-based $k = 38$, $p = 12$ & $n = 50$ and Concatenated $k = 38$, $N = 11$ & $l = 3$, for all-zeros message.

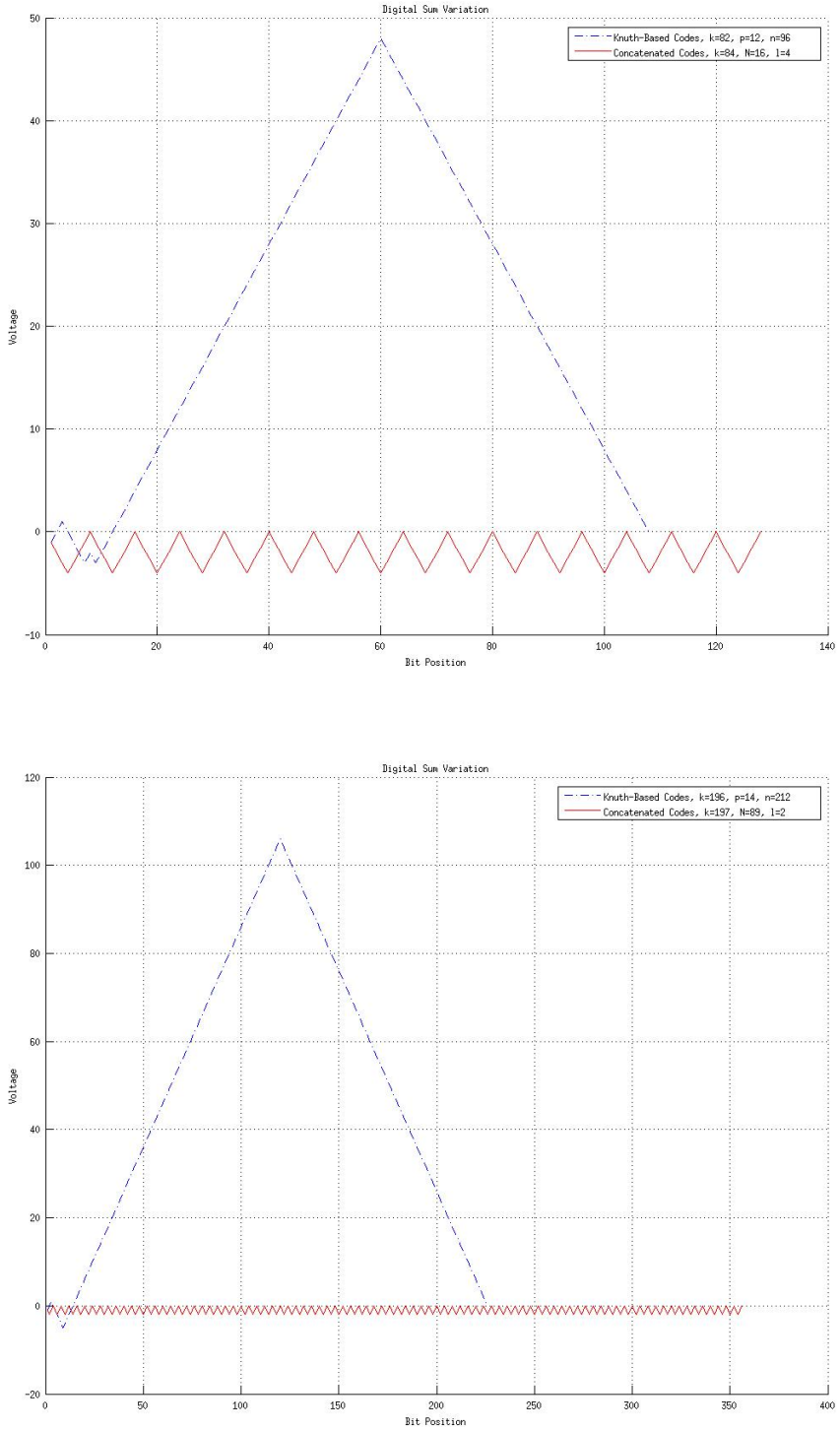

Figure 5-6: DSV of Binary Codes: a) Knuth-based $k = 82$, $p = 12$ & $n = 96$ and Concatenated $k = 84$, $N = 16$ & $l = 4$, b) Knuth-based $k = 196$, $p = 14$ & $n = 212$ and Concatenated $k = 197$, $N = 89$ & $l = 2$, for all-zeros message.

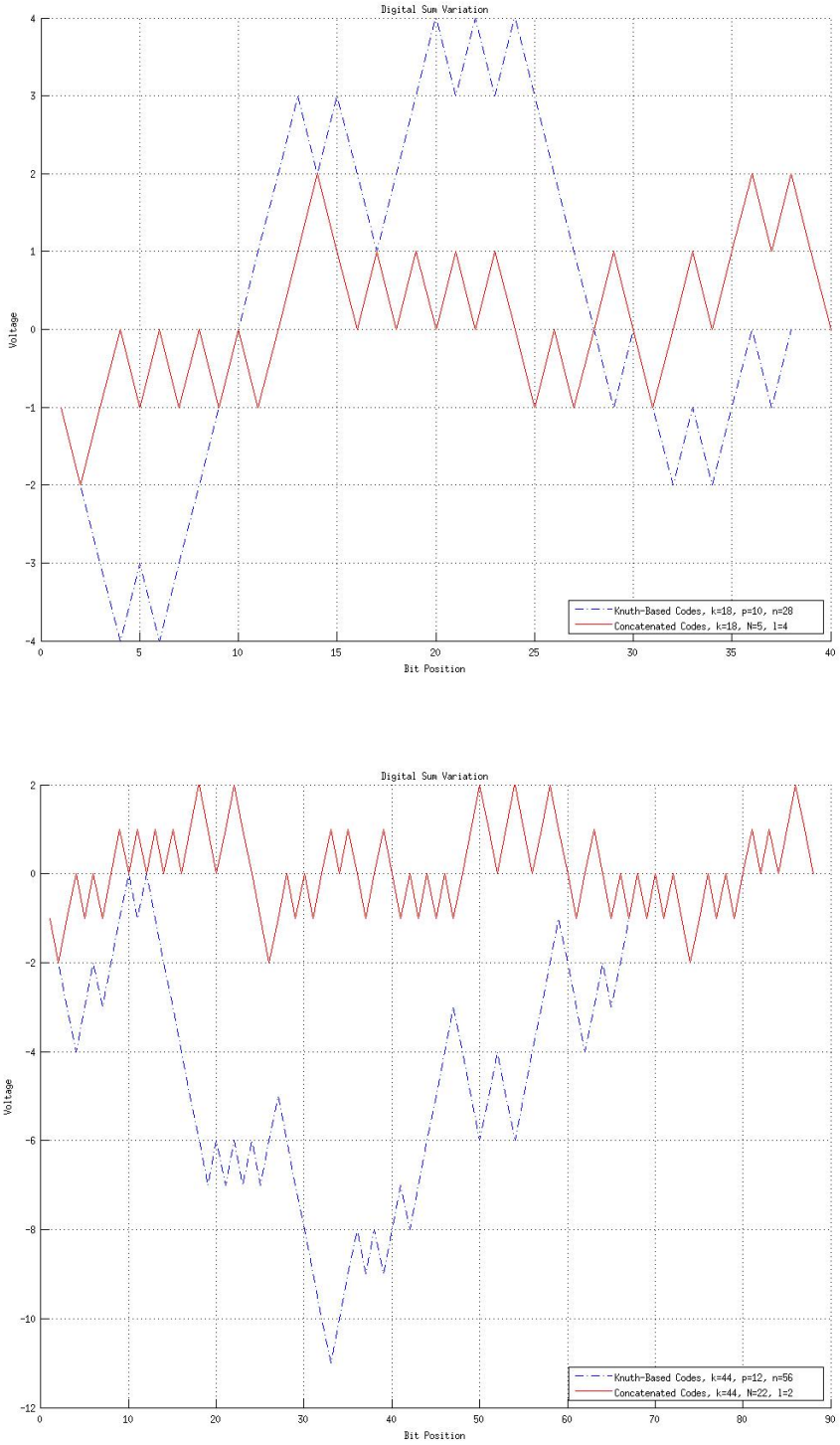

Figure 5-7: DSV of Binary Codes: a) Knuth-based $k = 18$, $p = 10$ & $n = 28$ and Concatenated $k = 18$, $N = 5$ & $l = 4$, b) Knuth-based $k = 44$, $p = 12$ & $n = 56$ and Concatenated $k = 44$, $N = 22$ & $l = 2$, for random messages.

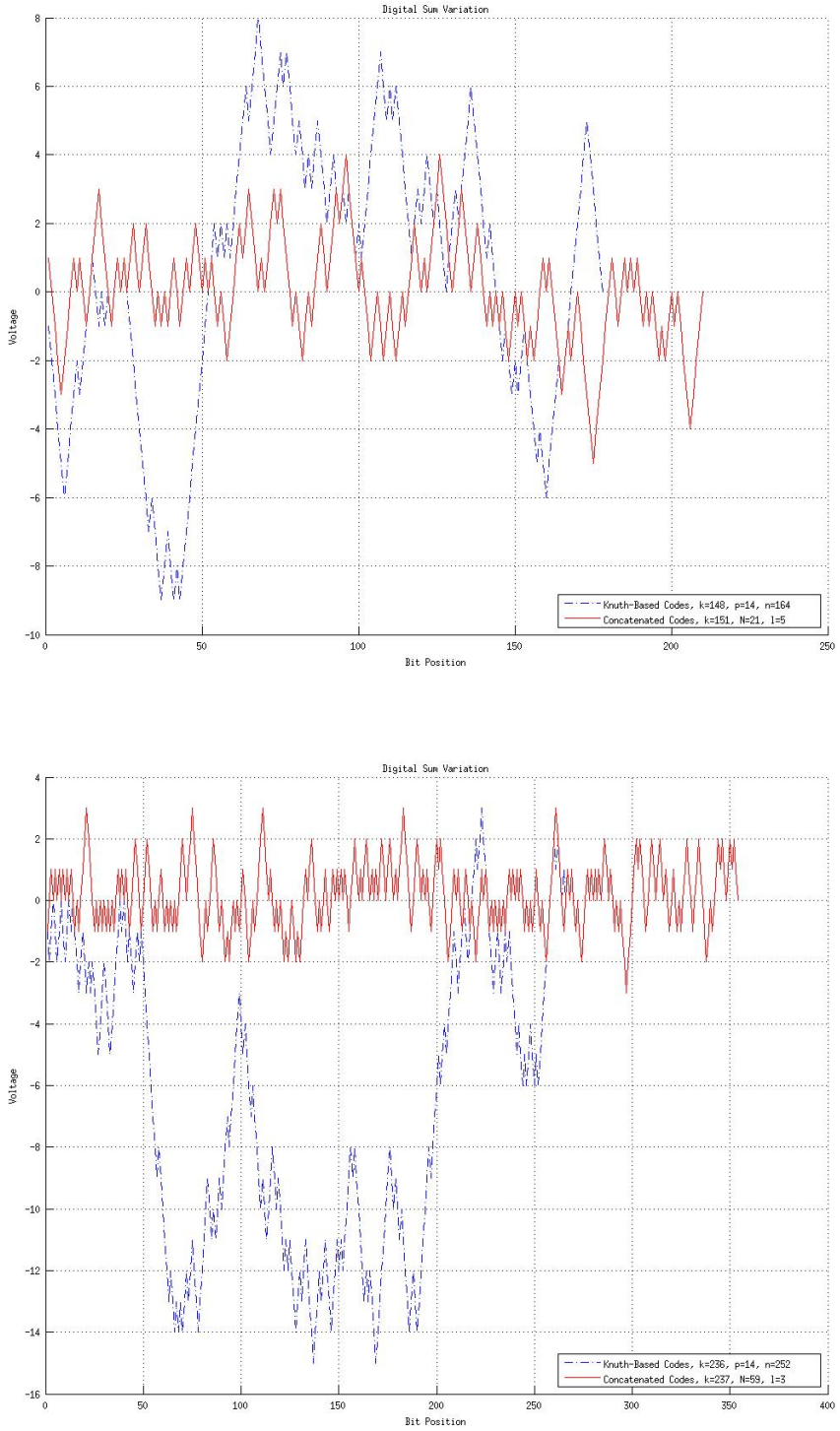

Figure 5-8: DSV of Binary Codes: a) Knuth-based $k = 148$, $p = 14$ & $n = 164$ and Concatenated $k = 151$, $N = 21$ & $l = 5$, b) Knuth-based $k = 236$, $p = 14$ & $n = 252$ and Concatenated $k = 237$, $N = 59$ & $l = 3$, for random messages.

Chapter 6

Conclusions and Recommendations for Future Research

This project was undertaken to explore charge balancing methods with error correction capabilities, and to develop or extend techniques for q-ary codes. After studying the literature of block codes, Hamming encoder and decoder routines for prime numbers of q were implemented via Matlab tools.

This document went over the fundamentals of Knuth's balancing method for q-ary sequences. The Knuth-like coding scheme with error correction capabilities has been extended from binary to q-ary codes. Some prefixes were calculated using Matlab tools, due to the current lack of sources for the bounds from q-ary constant weight codes. The ECC from q-ary Knuth-based Codes were added through Hamming Encoder.

One of the more significant findings to emerge from this study is a technique to reduce iterations to find the balancing indices z's. A higher start point for the search of z's is set up by this algorithm, since it begins from zero in the previous one. An analytical performance measure was carried on, by calculating the average iterations of this method needed to find z's in a full set. Additionally, the number of loops per sequence executed by the existent algorithm is qn. The search of all balancing indices is accelerated by a factor of q, as well as the separation between indices. The necessary cycles if only looking for the first z were only determined for some sequence lengths n shorter than 14.

Most of the computations which involve practical values of q and n were either inconclusive or infeasible via Matlab, due to the long execution when handling large numbers (q^k). Thus, general conclusions were not confirmed in order to reduce the redundancy of q-ary codes, i.e. Transmitted Index, Auxiliary Information Encoding and A Better Variable-Length-Prefix Construction. Nevertheless, some attributes from q-ary codes were identified and the entropy expressions were extended based on them.

This work contributes to existing knowledge of Concatenated Codes, by examining the feasibility of four constructions for values of $q \leq 20$ using Hamming encoding. Several restrictions were mostly traced to ensure low use of memory resources, and to secure smooth block code constructions over finite fields. Moreover, the current findings add substantially to future

understanding of q-ary Concatenated Codes.
The comparison of the code rates and the digital sum variation between Knuth-based and Concatenated binary codes has been presented. However, the generalisability for q-ary codes is subject to certain limitations, such as block codes over binary finite fields are only found in Matlab libraries. Consequently, the comparison between Concatenated and Knuth-based Codes has been restricted to binary and double error correction capabilities. The results of this study suggest that the price to pay for keeping the DSV between reasonable bounds is a decrease of code rates, as in Concatenated codes. Since Knuth-based codes have higher rates, but also large disparities in some cases.

Some recommendations are listed as follows:

- Further investigations is needed to estimate combinatorial expressions for q-ary codes, to describe distributions such as the transmitted index and the balancing positions in a full set, required to carry out the evaluation of redundancy-reductive techniques.
- Further research might explore an optimal algorithm for q-ary constant weight codes, to find the maximum number of sequences in the balanced set with distances that allow error correction, in order to build prefixes for q-ary Knuth-based Codes.
- More broadly, work is also necessary for further comparison between q-ary Knuth-based and Concatenated codes. Thus, block codes over $GF(q)$ with multiple error correction capabilities have to be implemented using software tools.
- It would be interesting to examine techniques for the mapping from extension fields to original alphabet symbols. The latter elements must remain invariable within codewords, since they can have real and physical meaning. Then, Knuth-based Codes can be feasible for any q.

Appendix A

Matlab Routines

A-1 q-ary Hamming Encoder/Decoder

```
function [x]=eHam(u,n,q)
% Hamming Encoder for prime numbers of q
[u,k,r,P]=deal(mod(u,q),length(u),n-length(u),[]);
if ~isprime(q)||r<2,  % Only for prime numbers
    x=[];
    return
end
r0=ceil(log10((q-1)*n+1)/log10(q));
r(r<r0)=r0;
if q==2,
    r(r<3)=3;
    [~,G]=hammgen(r);
    G(:,1:r-n+k)=[];
    x=mod(u*G(1:k,1:n),q);
    return
end
%*********************Parity Check Equations*********************
for i=1:power(q,r)-1,
    p0=dec2vect(power(q,r)-i,q,r);
    for m=1:q-1,
        if any(ismember(mod(m*[eye(r);P],q),p0,'rows')),
            break
        elseif m==q-1,
            P(end+1,1:r)=p0;
        end
    end
    if size(P,1)>=k,
        break
    end
end
```

```
%*****************************************************************
G=[P,eye(k)];
x=mod(u*G,q);
end

function [u]=dHam(x,k,q,CL)
% Hamming Decoder for prime numbers of q
[n,r,P]=deal(length(x),length(x)-k,[]);
if r<=1,
    u=mod(x(1:k),q);
    return
elseif n>floor((power(q,r)-1)/(q-1)),
    u=[];
    return
end
%*********************Parity Check Equations*********************
if q>2,
    for i=1:power(q,r)-1,
        p0=dec2vect(power(q,r)-i,q,r);
        for m=1:q-1,
            if any(ismember(mod(m*[eye(r);P],q),p0,'rows')),
                break
            elseif m==q-1,
                P(end+1,1:r)=p0;
            end
        end
        if size(P,1)>=k,
            break
        end
    end
    H=[eye(r),mod(-P',q)];
%*****************************************************************
elseif q==2,
    r(r<3)=3;
    H=hammgen(r);
    H(1:r-n+k,:)=[];
    H=H(:,end+1-n:end);
end
[s,e]=deal(mod(x*H',q),zeros(1,n));
if any(s),
%********* Decoding by Coset Leader for Low Values of k *********
    if any(CL),
        G=[P,eye(k)];
        for i=0:power(q,k)-1,
            u0=dec2base(i,q,k);
            u0=base2dec(u0(:),q)';
            e=mod(u0*G+[zeros(1,k),s],q);
            if sum(e)==1,            % Minimum Weight Achievable
                break
            end
            coset(i+1,1:n)=e;
        end
        if sum(e)>1,
```

```
49              ind=find(sum(coset,2)==min(sum(coset,2)));
50              e=coset(ind(1),:);
51          end
52  %*************************************************************
53      else
54          for i=1:q-1,
55              e=i*ismember(mod(i*H',q),s,'rows')';
56              if any(e),
57                  break
58              end
59          end
60      end
61      x=mod(x-e,q);
62  end
63  u=mod(x(end-k+1:end),q);
64  end
```

A-2 This function finds Balancing Sequences and Indices of q-ary Codes

```
1   function [BS] = b(x,q)
2   % Balancing of q-ary Sequences
3   % This routine finds all the Balancing Sequences and their Indexes
4   n=length(x);
5   if not(mod(q,2))&&mod(n,2), % q even and n odd
6       BS=[];
7       return
8   end
9   B=n*(q-1)/2;
10  z=B-sum(x)+q*ceil((B-sum(x))/(1-q))*(B<sum(x));
11  i=1;
12  while z<q*n,
13      e=mod(z,n);
14      s=floor(z/n);
15      bs=[repmat(s+1,1,e),repmat(s,1,n-e)];
16      y=mod(x+bs,q);
17      if (sum(y)==B),
18          BS(i,1:n+1)=[s*n+e,bs];
19          %break % Only gets the first Sequence & index
20          i=i+1;
21      end
22      z=z+q;
23  end
24  end
```

A-3 This function performs the inversion of the sequence y

```
1   function [x]=Inversion(y,z,q)
2   % Performs the inversion of a sequence y, returning x
3   n=length(y);
4   e=mod(z,n);
```

```
s=floor(z/n);
bs=[repmat(s+1,1,e),repmat(s,1,n-e)];
x=mod(y-bs,q);
end
```

A-4 q-ary & Binary Knuth-based Encoder

```
function [U,dh]=dMax(q,n,d0)
% This function tries to find the maximum number of vectors
% with distance greater than d0 in a full balanced set
[U,dh]=deal([]);
if ~mod(q,2)&&mod(n,2),
    return
else
    i(any(mod(n,2)))=sum([power(q,floor(n/2)),(q-1)*power(q,0:(n-3)/2)]);
    i(~any(mod(n,2)))=(q-1)*sum(power(q,0:n/2-1));
    while i<power(q,n),
        x=dec2vect(i,q,n);
        if sum(x)==n/2*(q-1),
            [U(end+1,1:n),i]=deal(x,i+q-2);
        end
        i=i+1;
    end
end
S=size(U,1);
[d(1:S,1:S),u]=deal(0,[]);
for i=1:S,
    [j,k]=deal(i,2);
    while 0<length(j),
        j=j(1);
        if ~any(d(j,:)),
            d1=sum(abs(bsxfun(@minus,U,U(j,:))),2)';
            [d1(d1<d0),d1(d1>=d0),dh(1:S,j)]=deal(0,1,d1);
            [d1,d1(j)]=deal(d1.*(1:S),j);
            d(j,1:S)=d1;
        end
        u0=d(i,d(i,:)>0);
        j=u0(~any(d(u0,:),2));
    end
    while k<=length(u0),
        [u0,k]=deal(intersect(u0,d(u0(k),:)),k+1);
        if length(u0)<=size(u,2),
            u0=[];
            break
        end
    end
    [d(i,u0(u0~=i)),d(u0(u0~=i),i)]=deal(0);
    u(end+~isempty(u0),1:length(u0))=u0;
end
u0=sortrows([sum(u~=0,2),u],-1);
u=u0(1,2:end);
dh=reshape(dh(u,u),1,[]);
```

```
dh=min(dh(dh~=0),[],2);
U=U(u,:);
end
```

```
function [y,R,t,n,p]=eKnuth(u,n,q)
% Knuth-based q-ary Encoder
if ~isprime(q),
    [y,R,t,n,p]=deal([]);
end
Knuth=matfile('Knuth.mat','Writable', false);
[pq,k]=deal(Knuth.pq,length(u));
if any(n),              % for Hamming Codes (whether or not n is fixed)
    if ~mod(q,2)&&mod(n,2),
        [y,R,t,n,p]=deal([]);
        return
    end
    r=ceil(log10((q-1)*n+1)/log10(q));
    k=n-r;
    u=u(1:k);
else
    r=2;
    while k>(power(q,r)-1)/(q-1)-r,
        r=r+1;
    end
    r(~mod(q,2)&&mod(k+r,2))=r+1;
    n=k+r;
end
C=eHam(u,n,q);
bs=b(C,q);
[z,z0,x1]=deal(bs(1,1),n,mod(C+bs(1,2:end),q));
z0(q>2)=q*z0;
i=min(find(and(and(pq(:,1)==q,pq(:,3)>=3),pq(:,4)>=z0))); % q p d z U
p=pq(i,2);
if isempty(p),
    [y,R,t,n,p]=deal([]);
    return
end
x2=dec2vect(pq(i,5+z),q,p);
[y,R,t]=deal([x2,x1],round(k/(n+p)*1e3)/1e3,1);
R(sum(y)~=length(y)*(q-1)/2)=[];
end
```

```
function [y,R,n,p]=e2Knuth(u,n,t0)
% Knuth-based binary Encoder
[u,k,Knuth]=deal(mod(u,2),length(u),matfile('Knuth.mat','Writable', false
    ));
m=ceil(log2(n+2));
t=(n-k)/m;
if mod(n,2)||t~=t0,
    [y,R,n,p]=deal([]);
    return
end
fprintf(['m=',num2str(m),', n=',num2str(n),', k=',num2str(k),'\n']);
```

```
C0=comm.BCHEncoder('CodewordLength',n,'MessageLength',k);
[C,pq]=deal(step(C0,u'),Knuth.pq);
bs=b(C,2);
[z,x1]=deal(bs(1,1),mod(C+bs(1,2:end),2));
i=min(find(and(and(pq(:,1)==2,pq(:,3)>=3),pq(:,4)>=n))); % q p d z U
p=pq(i,2);
if isempty(p),
    [y,R,n,p]=deal([]);
    return
end
x2=dec2vect(pq(i,5+z),2,p);
[y,R]=deal([x2,x1],round(k/(n+p)*1e3)/1e3);
R(sum(y)~=length(y)/2)=[];
end
```

A-5 q-ary Concatenated Codes Encoder

```
function [S,U,T]=cardinality(Q,N0)
% This functions finds the exact cardinality
% of Q=q0,q1,q2,... and lengths N=n0,n1,n2,...
% It also determines the construction feasibility of Concatenated Codes
folder='/media/alex/Datos/TU Delft/Thesis/Text/STYLESTUFF/';
Knuth=matfile('Knuth.mat','Writable', true);
Knuth.Sqn(1,1:3)=zeros(1,3);
Sqn=Knuth.Sqn;
T={'q&n&Cardinality&Greatest PP.&Factors&2.2&2.3&2.4&2.5\\\hline'};
for i=1:length(Q),
    q=Q(i);
    nOld=max(Sqn(Sqn(:,1)==q,2));
    nOld(isempty(nOld))=0;
    if all(N0),
        N=N0;
    else
        N=2:nOld;
    end
    for j=1:length(N),
        n=N(j);
        if mod(q,2)||~mod(n,2),
            if n>nOld,
                fprintf(['q=',num2str(q),', n=',num2str(n)]);
                U=[];
                h(any(mod(n,2)))=sum([power(q,floor(n/2)),(q-1)*power(q
                    ,0:(n-3)/2)]);
                h(~any(mod(n,2)))=(q-1)*sum(power(q,0:n/2-1));
                while h<power(q,n),
                    x=dec2vect(h,q,n);
                    if sum(x)==n/2*(q-1),
                        [U(end+1,1:n),h]=deal(x,h+q-2);
                    end
                    h=h+1;
                end
                clc;
```

```
35                  S=size(U,1);                % Cardinality
36                  Knuth.Sqn=sortrows([Knuth.Sqn;q,n,S]);
37              else
38                  U=[];
39              end
40              Sqn=Knuth.Sqn;
41              S=Sqn(and(Sqn(:,1)==q,Sqn(:,2)==n),3);
42              [Q0,Q1]=deal(primes(S),factor(S));
43              m=floor(log10(S)./log10(Q0));
44              P=S-power(Q0,m);
45              P=find(P==min(P));
46              Q0=power(Q0(P),m(P));
47              Q1=power(unique(Q1),histc(Q1,unique(Q1)));
48              if max(Q1)>1e3||S>1e5,
49                  break
50              else
51                  C=[S<1e5,Q0<1e3,max(Q1)<1e3&&length(Q1)>1,~mod(q,2)&&max(
                        Q1)<1e3];
52                  t0=any(C);
53                  C=regexprep(num2str(C),'\s*','&');
54                  C=regexprep(C,'1','x');
55                  C=regexprep(C,'0','');
56              end
57              if t0,
58                  t={strcat(num2str(q),'&',num2str(n),'&',num2str(S),'&',
                        num2str(Q0),'&',regexprep(num2str(Q1),'\s*',',','),'&',C
                        ,'\\ \hline')};
59                  T=[T;t];
60              end
61          end
62      end
63  end
64  end
```

```
1   function [C,k,d]=GF(u,q,Q,N)
2   % This function carries on the data conversion from the alphabet q
3   % to the alphabets Q=q0,q1,q2,... for block codes of length N
4   [C,k,d,j]=deal([],0,[],0);
5   for i=1:length(Q),
6       pr=unique(factor(Q(i)));
7       m=histc(factor(Q(i)),pr);
8       r=ceil(log10((pr-1)*m*N+1)/log10(pr));    % for Hamming Codes
9       k0=m*(N-ceil(r/m));
10      k1=floor(log10(power(pr,k0))/log10(q));
11      if any(k1),
12          u0=u(j+1:j+k1);
13          if pr~=q,
14              u0=dec2vect(sum(u0.*power(q,k1-1:-1:0)),pr,k0);
15          end
16      else
17          [C,k,d]=deal([],0,[]);
18          break
19      end
```

```
    [d(i),C0,k,j]=deal(3,eHam(u0,m*N,pr),k+k1,j+k1);
    if m>1,
        for h=1:N,
         [~,e]=gftuple(C0(m*(h-1)+1:m*h),m,pr);
         e(isinf(e))=-1;
         C=[C,e+1];
        end
    else
        C=[C,C0];
    end
end
end
```

```
function [x,R,t,k]=eCC(u,n,N,q,BC)
% Concatenated q-ary Encoder
n(n<4&&q==2)=4;
n(~mod(q,2)&&any(mod(n,2)))=n+1;
[card,U,~]=cardinality(q,n);
switch BC
    case 2                                  % Construction 2.2
        L=floor(card/2);
        Q=2*L;
        k=floor(log10(power(Q,N-1))/log10(q));
        if q~=Q,
            u=sum(u(1:k).*power(q,k-1:-1:0));
            u0=dec2vect(u,Q,N-1);
        else
            u0=u(1:k);
        end
        C=[u0,ceil(sum(u0)/L)*L-sum(u0)];
        d=2*2;                              % d=2 (Balanced Set) x 2 (I-Set)
    case 3                                  % Construction 2.3
        Q=primes(card);
        m=floor(log10(card)./log10(Q));
        pr=card-power(Q,m);
        pr=find(pr==min(pr));
        Q=power(Q(pr),m(pr));
        [C,k,d]=GF(u,q,Q,N);
        d=2*d;                              % d=2 (Balanced Set) x d of C
    case 4                                  % Construction 2.4
        Q=factor(card);
        Q=power(unique(Q),histc(Q,unique(Q)));
        [C,k,d]=GF(u,q,Q,N);
        d=2*min(d);                         % d=2 (Balanced Set) x d of C
    case 5                                  % Construction 2.5
        Q=factor(card);
        Q(any(mod(q,2)),:)=[];
        if ~isempty(Q),
            m=histc(Q,unique(Q));
            if m(1)<=2,
                m0=m(1);
                Q=power(unique(Q(m0+1:end)),m(2:end));
            elseif m(1)>2,
```

```
41                  [m(1),m0]=deal(m(1)-2,2);
42                  Q=power(unique(Q),m);
43              end
44              [k0(m0==1),k0(m0>1)]=deal(N-1,2*N);
45              k1=floor(log10(power(2,k0))/log10(q));
46              if any(~k1),
47                  C=[];
48              else
49                 u0=sum(u(1:k1).*power(q,k1-1:-1:0));
50                 u0=dec2vect(u0,2,k0);
51                 if m0==1,
52                     C0=[u0,mod(sum(u0),2)];
53                 else
54                     C0=u0;
55                 end
56                 [C,k,d]=GF(u(k1+1:end),q,Q,N);
57                 if ~isempty(C),
58                     C=[floor(max(Q)/2)*C0,C];
59                     k=k1+k;
60                     d=min([n*(q-1)*(m0*N-k0+1),2*d]);
61                 end                                % d=2 (U) x d of C & n(q-1)
62              end
63          end
64      otherwise
65          fprintf('Wrong Construction\n');
66          [x,R,t,k]=deal([]);
67          return
68  end
69  if isempty(Q)||isempty(C)||k==0,
70      [x,R,t,k]=deal([]);
71      return
72  elseif isempty(U),
73      i(any(mod(n,2)))=sum([power(q,floor(n/2)),(q-1)*power(q,0:(n-3)/2)]);
74      i(~any(mod(n,2)))=(q-1)*sum(power(q,0:n/2-1));
75      while size(U,1)<ceil(max(Q)/2),
76          x=dec2vect(i,q,n);
77          if sum(x)==n/2*(q-1),
78              [U(end+1,1:n),i]=deal(x,i+q-2);
79          end
80          i=i+1;
81      end
82  end
83  L=floor(max(Q)/2);
84  U=unique([U(1:L,:);(q-1)*ones(L,n)-U(1:L,:);U(L+any(mod(max(Q),2)),:)],'
        rows','stable');
85  [x,R,t]=deal(reshape(U(C+1,:)',1,[]),k/(N*n),floor((d-2)/2));
86  end
```

A-6 Miscellaneous Routines

```
function [x]=dec2vect(i,q,n)
% This function converts a number i from decimal
% to a vector in base q of length n
for j=n:-1:1,
    P=power(q,j-1);
    x(n-j+1)=floor(i/P);
    i=mod(i,P);
end
end
```

```
function[rw,BS,it0,hlb] = b_s(x,q,step)
% Balancing of q-ary Sequences
% This routine finds:
% 1) rw: Random Walk
% 2) BS: Balancing Sequences and their Indexes.
% 3) it0: Total Iterations performed by this method.
% 4) hbl: Balance Disparity, Higher & Lower Bounds.
n=length(x);
if ~mod(q,2)&&mod(n,2), % q even and n odd
    BS=[];
    return
end
B=n*(q-1)/2;            % Balancing Value
[i,it0(1:2)]=deal(0);   % Balancing Sequence & Iteration Counter
d=B-sum(x);             % Initial Balance Disparity
zl=d;                   % Lower Bound for I-Quadrant
zh=q*(n-1)+mod(d,q);    % Higher Bound for I-Quadrant
if (d<0),
    zh=q*n+d;                 % Higher Bound for II-Quadrant
    zl=d+q*ceil(d/(1-q));     % Lower Bound for II-Quadrant
end
if (q==2),                    % Adjustment for Binary case
    zh=q*n-zl-q*any(~zl);
end
hlb=[d,zl,zh];          % Initial Balance Dis., Higher & Lower Bounds
if step==1,             % Step Size for Random Walk
    zl=0;
    zh=q*n-1;
else
    step=q;             % q-Step size (otherwise)
end
while (zl<=zh),
    p=mod(zl,n);
    s=floor(zl/n);
    bs=[repmat(s+1,1,p),repmat(s,1,n-p)];
    y=mod(x+bs,q);              % Balanced Sequence
    it0(2)=it0(2)+1;            % Iteration Counter for all z's
    rw(it0(2),1:2)=[zl,sum(y)]; % Ramdon Walk
    if (sum(y)==B),
        i=i+1;                  % Balancing Sequence Counter
        BS(i,1:n+1)=[s*n+p,bs]; % Matrix of Balancing Indices & Sequences
        if i==1,
            it0(1)=it0(2);      % Iteration Counter for first z
```

```
        end
        %break
    end
    zl=zl+step;
end
end
```

```
function [trials,bseq,Ne0,Pum]=Test1(q,n)
tic; close all; clc;
folder='/media/alex/Datos/TU Delft/Thesis/Text/STYLESTUFF/';
% Initializing variables
[it1,j,hlb(1,1:3),bseq,BS(1,1:n+1),Ne0(1:n)]=deal(0);
B=n*(q-1)/2;                    % Balancing Value
for i=0:power(q,n)-1,
    x=dec2vect(i,q,n);
    [~,bs,it0,hlb(end+1,1:3)]=b_s(x,q,q); % Balancing indices & sequences
    hlb=unique(hlb,'rows');               % Higher & Lower Bounds
    it1=it1+it0;                          % Iteration Counter
    bp=size(bs,1);                           % Balancing Positions
    zlrange(j+1:j+bp,1)=repmat(sum(x),bp,1); % Plot Coordinates
    zlrange(j+1:j+bp,2)=bs(:,1);             % Balancing Indices
    zlrange=unique(zlrange,'rows');
    j=size(zlrange,1);                    % Number of Plot Coordinates
    e0=mod(bs(1,1),n);                    % First Balancing Position
    Ne0(e0+1)=Ne0(e0+1)+1;          % Occurrence of Transmitted Index
    %*****Cardinality & Balanced Sequences*******
    y=mod(x+bs(1,2:n+1),q);
    if ~ismember(BS(:,1:n),y,'rows'),
        BS(end+1,1:n)=y;        % Balanced Sequences
        BS(~any(BS,2),:)=[];    % Remove rows of zeros
    end
    ind=find(ismember(BS(:,1:n),y,'rows'),1);
    BS(ind,n+1)=BS(ind,n+1)+1; % Occurrence Counter
    %****Balancing Positions & Occurrence*******
    if isempty(find(bseq(:,1)==bp)),  % Description by Column:
        bseq(end+1,1)=bp;             % Balancing Position
        bseq(~any(bseq(:,1),2),:)=[]; % Remove rows of zeros
        bseq(end,3)=sum(x);           % Sequence Weight
        bseq(end,4)=-1;               % End flag
    end
    ind=find(bseq(:,1)==bp);
    bseq(ind,2)=bseq(ind,2)+1;        % Occurrence Counter
    %**********Sequence Weights*****************
    if isempty(find(bseq(ind,3:end-1)==sum(x))),
        ind0=find(bseq(ind,3:end)==-1)+2;
        bseq(ind,ind0)=sum(x);
        bseq(ind,ind0+1)=-1;
    end
    %***********Error Control******************
    if ~isequal(bs,b(x,q)),
        fprintf('ERROR\n');
        return
    else
```

```
47          fprintf(['q=',num2str(q),', n=',num2str(n),' counter: ',num2str(
                power(q,n)-i)]);
48          clc;
49      end
50  end
51  Pum(:,1)=unique(BS(:,n+1));             % u (Occurrence by Balanced
        Sequence)
52  Pum(:,2)=histc(BS(:,n+1),Pum(:,1));     % P(u,m)
53  bseq=sortrows(bseq);
54  card=size(BS,1);                        % Cardinality
55  trials=round(it1/power(q,n)*100)/100;   % Average Iterations
56  %************************Error Control*****************************
57  if ~isequal(sum(bseq(:,2)),sum(BS(:,n+1)),sum(Ne0),power(q,n))||sum(Pum
        (:,2))~=card,
58      fprintf('ERROR\n');
59      return
60  end
61  %*********************************************************************
62  fprintf(['Total Points: ',num2str(j),', Cardinality: ',num2str(card),'\n'
        ]);
63  fprintf(['Approximated Cardinality: ',num2str(power(q,n)*sqrt(6/(pi*n*(
        power(q,2)-1)))),'\n']);
64  fprintf('1) Total Iterations\n');
65  fprintf('2) Balancing Position, Occurrence & Sequence Weights\n');
66  fprintf('3) Occurrence of Transmitted Index by Position\n');
67  fprintf('4) u and P(u,m)\n');
68  %****************************Plots************************************
69  x0=randi(power(q,n)-1);
70  x=dec2vect(x0,q,n);                     % Random Balancing Sequences
71  [~,bs,~,~]=b_s(x,q,q);                  % Balancing Sequences
72  hlb(~any(hlb,2),:)=[];                  % Remove rows of zeros
73  zlinR=figure;
74  set(0,'DefaultTextInterpreter','tex');
75  hold on;
76  grid on;
77  lbound=plot(hlb(:,1),hlb(:,2));
78  hbound=plot(hlb(:,1),hlb(:,3),'color','black');
79  bs0=plot(repmat(B-sum(x),length(bs(:,1)),1),bs(:,1),'-.g');
80  set(bs0,'Displayname',['\sigma(z)s from: ',num2str(x0)]);
81  set(lbound,'Displayname','Lower Bound');
82  set(hbound,'Displayname','Higher Bound');
83  legend('Location','southoutside');
84  plot(B-zlrange(:,1),zlrange(:,2),'r*')
85  xlabel('Balance Disparity \beta-\sigma(x)')
86  ylabel('Index z')
87  title(['Balance Disparity vrs Index z, q=',num2str(q),', n=',num2str(n)])
        ;
88  hold off;
89  %saveas(zlinR,[folder,'zlinR',num2str(q),'x',num2str(n),'.jpg']);
90  %*********************************************************************
91  rw=b_s([2,1,3,2,0,4,1,3,1,4],5,1);
92  if size(rw,1)==50,
93      Rwalk=figure;
```

```
94      hold on;
95      plot([0,49],[20,20],'-.');
96      plot(rw(:,1),rw(:,2),'-.r*')
97      xlabel('Index z')
98      ylabel('Sequence Weight \sigma(z)')
99      title('Index z vrs Sequence Weight \sigma(z), q=5, n=10');
100     hold off;
101     saveas(Rwalk,[folder,'Rwalk.jpg']);
102 end
103 fprintf(['Elapsed Time: ',num2str(round(toc/6)/10),' min\n']);
104 end
```

Appendix B

Calculation Log

B-1 Columns of the Figure 3-4 a)

q	2	2	3	3	4	5	7	8	9	10	11	12	13	14	15	16	31
n	10	14	6	10	8	6	6	4	4	4	4	4	4	4	4	4	3
v	Balancing Position Occurrences $N(v)$																
1	0	0	0	1920	3840	960	8148	256	504	800	1144	1536	2288	3136	4080	5120	7440
2	280	3696	297	7170	11584	3600	27342	1920	2736	4200	6116	9120	11908	16072	21120	28160	7440
3	0	0	0	10020	5376	3600	26964	768	1656	2400	3696	4608	7176	9408	12720	15360	14911
4	280	3696	324	12060	28608	4560	33978	1152	1665	2600	3685	5472	7189	9800	12705	16896	
5	0	0	0	9900	0	2100	15036										
6	240	3360	108	8040	13824	805	6181										
7	0	0		5220	0												
8	160	2688		3090	2304												
9	0	0		1260													
10	64	1792		369													
11		0															
12		896															
13		0															
14		256															

Table B-1: Matlab Calculations: Balancing Positions (v) and their Occurrences $N(v)$, in a full set of sequences $\in \mathcal{A}_q$ of length n.

The minimum averaged iterations if all the indices z's are searched, are calculated using the following formula: $v_1 * Pr_2(v_1) + v_2 * Pr_2(v_2) + ... + v_n * Pr_2(v_n)$.
e.g. $(2 * 280 + 4 * 280 + 6 * 240 + 8 * 160 + 10 * 64)/2^{10} = 4.9219$ for $q = 2$ & $n = 10$[1].

B-2 Calculations to plot out the 2-D Plane for q=3 and n=10

The balancing index is first determined as $\beta = 10 * (3 - 1)/2 = 10$. The calculation of randomly chosen cordinates $(\beta - \sigma(\mathbf{x}), z)$, is carried on as follows:

[1] Using the data from Table B-1.

- Coordinates $(10, 10)$ & $(10, 25)$, located in the I-Quadrant:
  The sequence $\mathbf{x}_1 = 0000000000$, has weight of $\sigma(\mathbf{x}_1) = 0$, thus its balance disparity is $\beta - \sigma(\mathbf{x}_1) = 10 - 0 = 10$. The first balancing sequence found is $\mathbf{b}(1, 0) = 1111111111$, since $\mathbf{y}_1 = \mathbf{x}_1 \oplus_3 \mathbf{b}(1, 0) = 0000000000 \oplus_3 1111111111 = 1111111111$, and $\sigma(\mathbf{y}_1) = 1+1+1+1+1+1+1+1+1+1 = 10$, thus $z_1 = s * n + e = 1 * 10 + 0 = 10$.
  Another balancing sequence is $\mathbf{b}(2, 5) = 3333322222$, ($0000000000 \oplus_3 3333322222 = 0000022222$ and $\sigma(0000022222) = 10$), thus $z_2 = 2 * 10 + 5 = 25$.

- Coordinates $(-4, 2)$ & $(-4, 5)$, located in the II-Quadrant:
  The sequence $\mathbf{x}_2 = 2201222120$, has weight $\sigma(\mathbf{x}_2) = 2+2+0+1+2+2+2+1+2+0 = 14$, thus its balance disparity is $\beta - \sigma(\mathbf{x}_2) = 10 - 14 = -4$. The first balancing sequence found is $\mathbf{b}(0, 2) = 1100000000$, since $\mathbf{y}_2 = 2201222120 \oplus_3 1100000000 = 0001222120$, and $\sigma(\mathbf{x}_2) = 0+0+0+1+2+2+2+1+2+0 = 10$, thus $z_3 = 0 * 10 + 2 = 2$.
  Another balancing sequence is $\mathbf{b}(0, 5) = 1111100000$, since $2201222120 \oplus_3 1111100000 = 0012022120$, and $\sigma(0012022120) = 10$, thus $z_4 = 0 * 10 + 5 = 5$.

The initial values when searching z, denoted as z_0, for the coordinates above are calculated as follows:
$(10, 10)$ & $(10, 25) \rightarrow z_0 = \beta - \sigma(\mathbf{x}_1) = 10$
$(-4, 2)$ & $(-4, 5) \rightarrow j = \lceil \frac{\beta - \sigma(\mathbf{x}_2)}{1-q} \rceil = \lceil \frac{-4}{1-3} \rceil = 2$, $z_0 = \beta - \sigma(\mathbf{x}_2) + q * j = -4 + 3 * 2 = 2$
Notice that the ordinates are spaced by steps of $q = 3$, for the same abscissa as: $z_1 = 10$ & $z_2 = 10 + 3 * 5 = 25$ for $\beta - \sigma(\mathbf{x}_1) = 10$, and $z_3 = 2$ & $z_4 = 2 + 3 * 1 = 5$ for $\beta - \sigma(\mathbf{x}_2) = -4$. The indices z_1 & z_3 are found in the first attempt, and z_2 & z_4 in the 6th and 2nd trial respectively.

B-3 Redundancy

The entropy of the Transmitted Index is: $H_e(n) = -Pr_1(e = 0) * log_q(Pr_1(e = 0)) + ... - Pr_1(e = n - 1) * log_q(Pr_1(e = n - 1))$.
e.g. $H_e(10)$ for $q = 3$ & $n = 10$ is calculated as follows.
The transmitted index probabilities are[2]:
$Pr_1(e = 0) = 10978/59049 = 0.186$, where $3^{10} = 59049$.
$Pr_1(e = 1) = 10899/59049 = 0.185$, $Pr_1(e = 2) = 9843/59049 = 0.167$,
$Pr_1(e = 3) = Pr_1(e = 4) = 5343/59049 = 0.09$, $Pr_1(e = 5) = 4870/59049 = 0.083$,
$Pr_1(e = 6) = 3298/59049 = 0.056$, $Pr_1(e = 7) = 3247/59049 = 0.055$,
$Pr_1(e = 8) = 3139/59049 = 0.053$, $Pr_1(e = 9) = 2089/59049 = 0.035$.

Thus $H_e(10) = -0.186 * log_3(0.186) - 0.185 * log_3(0.185) - 0.167 * log_3(0.167) - 2 * 0.09 * log_3(0.09) - 0.083 * log_3(0.083) - 0.056 * log_3(0.056) - 0.055 * log_3(0.055) - 0.053 * log_3(0.053) - 0.035 * log_3(0.035) = 1.965$.

The entropy of the Auxiliary Data is: $H_a(n) = Pr_2(1) * log_q(1) + Pr_2(2) * log_q(2) + ... + Pr_2(n) * log_q(n)$.

[2] Using the data from Table B-2.

q	e														
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	252	252	112	112	72	72	48	48	28	28	0				
2	3432	3432	1584	1584	1080	1080	800	800	600	600	432	432	264	264	0
3	174	174	174	69	69	69	0								
3	10978	10899	9843	5343	5343	4870	3298	3247	3139	2089	0				
4	10952	10952	10952	10952	5432	5432	5432	5432	0						
5	3093	3061	3013	2780	2304	1374	0								
7	26138	21198	18678	17453	17181	17001	0								
8	1024	1024	1024	1024	0										
9	1947	1640	1502	1472	0										
10	2730	2730	2270	2270	0										
11	3967	3912	3660	3102	0										
12	5184	5184	5184	5184	0										
13	8060	7140	6726	6635	0										
14	10234	10234	8974	8974	0										
15	13432	13292	12656	11245	0										
16	16384	16384	16384	16384	0										
31	10331	9770	9690	0											

Table B-2: Matlab calculations: Transmitted Index Occurrences, in a full set of sequences $\in \mathcal{A}_q$ of length n.

e.g. $H_a(6) = (0 + 297 * log_3(2) + 0 + 324 * log_3(4) + 0 + 108 * log_3(6))/3^6 = 1.0595$ for $q = 3$ & $n = 6^3$.

The entropy using variable-length-prefix scheme is: $H_u(n) = q^{-n} * [u_q P(u_q, n) log_q(u_q) + u_{q+1} P(u_{q+1}, n) log_q(u_{q+1}) + ... + u_{qn} P(u_{qn}, n) log_q(u_{qn})]$.
e.g. $H_u(6) = [3 * 18 * log_3(3) + 4 * 36 * log_3(4) + 5 * 36 * log_3(5) + 6 * 20 * log_3(6) + 7 * 18 * log_3(7) + 8 * 12 * log_3(8) + 9 * 1 * log_3(9)]/3^6 = 1.5336$ for $q = 3$ & $n = 6^4$.

The entropy for the full set of balanced codewords $\in \mathcal{A}_q$ of length n is: $H_0(n) = n - log_q(\mathcal{S}_q^n)$.
e.g. $H_0(4) = 4 - log_8(344) = 1.1912$ for $q = 8$ & $n = 4$.

[3] Using the data from Table B-1.
[4] Using the data from Table B-3.

q	2	2	3	3	4	5	7	8	9	10	11	12	13	14	15	16	31
n	10	14	6	10	8	6	6	4	4	4	4	4	4	4	4	4	3
u	$P(u,n)$																
2	2	2															
3	60	252	18	20													
4	120	1176	36	500	564												
5	60	1260	36	1948	1128	88											
6	10	588	20	2552	1128	530											
7		140	18	1783	1128	222	1920										
8		14	12	1042	1100	248	1800	216									
9			1	599	992	141	508		208								
10				318	828	108	83			302							
11				87	476	74	308		63		384						
12				25	239	41	566		21			724					
13				14	90	49	292		9		31		712				
14				9	63	29	1075		9	42	93			906			
15				12	58	57	318		10	104					1084		
16				4	58	58	495	96	25	42	39		135			1712	
17				8	26	47	359		5		39		45				
18				12	20	20	179		67						59		
19				12	12	13	176		7		72		15		177		
20				8	30	14	187		12	132	24		15	90			
21					54		355		9					208			
22					40		87		6		105		25	90	65		
23					32		78		6				70		65		
24					17		86	24	5	3		324	1				
25					8	12	77		10	6			6				
26					1		64		2	3			231		153		
27							64		14		14		6		51		
28							76		1		14		6	396			
29							39						24				
30							32			16	6		18		337		
31							27				2						525
32							21	8			6		24			768	
33							21				52		24				
34							13			3	6			6			
35							25			6	1		9	12			
36										3	3	76	17	6			
37													2		30		
38													8		30		
39													60				
40										8			8				
41															9		22
42													3	64	3		11
43													1				
44															18		
45															136		
46															18		
48												32	1	6	3	176	
49													3	12	9		
50														6			
51																	8
52																	16
56														32			
60															8		
61																	1
62																	49
63																	1
64																80	
72																	10
73																	5
82																	4
83																	8
93																	61
$\mathcal{S}_q^n$	252	3432	141	8953	8092	1751	9331	344	489	670	891	1156	1469	1834	2255	2736	721

Table B-3: Matlab calculations: $P(u,n)$ with $d(\mathbf{x}) = u$, in the full set of sequences $\in \mathcal{A}_q$ of length n.

Appendix C

Concatenated Code Constructions

q	t	n	$\mathcal{S}_q^n$	N	R
2	1	4	6	4	0.438
2	1	4	6	6	0.5
2	1	6	20	4	0.5
2	1	8	70	4	0.563
2	1	4	6	10	0.575
2	1	10	252	4	0.575
2	1	6	20	6	0.583
2	1	4	6	16	0.594
2	1	4	6	20	0.613
2	1	4	6	24	0.615
2	1	4	6	32	0.625
2	1	8	70	6	0.625
2	1	6	20	10	0.633
2	1	10	252	6	0.65
2	1	6	20	16	0.667
2	1	6	20	20	0.683
2	1	6	20	24	0.688
2	1	8	70	10	0.688
2	1	6	20	32	0.693
2	1	10	252	10	0.71
2	1	8	70	16	0.711
2	1	8	70	20	0.725
2	1	8	70	24	0.729
2	1	8	70	32	0.742
2	1	10	252	16	0.744
2	1	10	252	20	0.755
2	1	10	252	24	0.763
2	1	10	252	32	0.772
3	1	2	3	4	0.125
3	1	2	3	6	0.25
3	1	2	3	10	0.25
3	1	2	3	20	0.275
3	1	2	3	16	0.281
3	1	2	3	24	0.292
3	1	2	3	32	0.297
3	1	3	7	4	0.333
3	1	4	19	4	0.438
3	1	3	7	6	0.444
3	1	3	7	10	0.467
3	1	3	7	16	0.5
3	1	3	7	20	0.5
3	1	5	51	4	0.5
3	1	3	7	24	0.514
3	1	3	7	32	0.521
3	1	4	19	6	0.542
3	1	6	141	4	0.542
3	1	5	51	6	0.567
3	1	7	393	4	0.571
3	1	4	19	10	0.575
3	1	9	3139	4	0.583
3	1	8	1107	4	0.594
3	1	10	8953	4	0.6
3	1	4	19	16	0.609
3	1	6	141	6	0.611
3	1	4	19	20	0.613
3	1	4	19	24	0.625
3	1	4	19	32	0.633
3	1	5	51	10	0.64
3	1	7	393	6	0.643
3	1	8	1107	6	0.646
3	1	5	51	16	0.663
3	1	6	141	10	0.667
3	1	9	3139	6	0.667
3	1	5	51	20	0.67
3	1	5	51	24	0.675

q	t	n	$\mathcal{S}_q^n$	N	R
3	1	10	8953	6	0.683
3	1	7	393	10	0.686
3	1	5	51	32	0.688
3	1	6	141	16	0.698
3	1	6	141	20	0.708
3	1	8	1107	10	0.713
3	1	6	141	24	0.715
3	1	9	3139	10	0.722
3	1	7	393	16	0.723
3	1	6	141	32	0.724
3	1	7	393	20	0.736
3	1	10	8953	10	0.74
3	1	8	1107	16	0.742
3	1	7	393	24	0.744
3	1	7	393	32	0.75
3	1	8	1107	20	0.756
3	1	9	3139	16	0.757
3	1	8	1107	24	0.76
3	1	8	1107	32	0.77
3	1	9	3139	20	0.772
3	1	10	8953	16	0.775
3	1	9	3139	24	0.778
3	1	10	8953	20	0.785
3	1	9	3139	32	0.788
3	1	10	8953	24	0.792
3	1	10	8953	32	0.8
4	1	2	4	4	0.375
4	1	2	4	6	0.417
4	1	2	4	10	0.45
4	1	2	4	16	0.469
4	1	2	4	20	0.475
4	1	2	4	24	0.479
4	1	2	4	32	0.484
4	1	4	44	4	0.5
4	1	4	44	6	0.542
4	1	6	580	4	0.542
4	1	8	8092	4	0.594
4	1	4	44	10	0.6
4	1	6	580	6	0.611
4	1	4	44	16	0.625
4	1	4	44	20	0.638
4	1	4	44	24	0.646
4	1	4	44	32	0.656
4	1	8	8092	6	0.667
4	1	6	580	10	0.683
4	1	6	580	16	0.708
4	1	6	580	20	0.725
4	1	8	8092	10	0.725
4	1	6	580	24	0.729
4	1	6	580	32	0.74
4	1	8	8092	16	0.758
4	1	8	8092	20	0.769
4	1	8	8092	24	0.776
4	1	8	8092	32	0.785
5	1	2	5	4	0.25
5	1	2	5	6	0.333
5	1	2	5	10	0.35
5	1	2	5	16	0.375
5	1	2	5	24	0.396
5	1	2	5	20	0.4
5	1	2	5	32	0.406
5	1	3	19	4	0.417
5	1	3	19	6	0.444
5	1	4	85	4	0.5
5	1	3	19	10	0.533

q	t	n	$\mathcal{S}_q^n$	N	R
5	1	3	19	16	0.542
5	1	4	85	6	0.542
5	1	6	1751	4	0.542
5	1	5	381	4	0.55
5	1	3	19	20	0.567
5	1	3	19	24	0.569
5	1	7	8135	4	0.571
5	1	3	19	32	0.573
5	1	8	38165	4	0.594
5	1	4	85	10	0.6
5	1	5	381	6	0.6
5	1	6	1751	6	0.639
5	1	4	85	16	0.641
5	1	7	8135	6	0.643
5	1	4	85	20	0.65
5	1	4	85	24	0.656
5	1	5	381	10	0.66
5	1	4	85	32	0.664
5	1	8	38165	6	0.667
5	1	6	1751	10	0.683
5	1	5	381	16	0.688
5	1	5	381	20	0.7
5	1	5	381	24	0.7
5	1	5	381	32	0.713
5	1	7	8135	10	0.714
5	1	6	1751	16	0.719
5	1	8	38165	10	0.725
5	1	6	1751	20	0.733
5	1	6	1751	24	0.736
5	1	7	8135	16	0.741
5	1	6	1751	32	0.745
5	1	7	8135	20	0.757
5	1	7	8135	24	0.762
5	1	8	38165	16	0.766
5	1	7	8135	32	0.772
5	1	8	38165	20	0.775
5	1	8	38165	24	0.781
5	1	8	38165	32	0.793
6	1	2	6	4	0.25
6	1	2	6	6	0.417
6	1	2	6	10	0.45
6	1	2	6	16	0.469
6	1	2	6	20	0.475
6	1	2	6	24	0.479
6	1	2	6	32	0.484
6	1	4	146	4	0.5
6	1	4	146	6	0.542
6	1	6	4332	4	0.583
6	1	4	146	10	0.625
6	1	6	4332	6	0.639
6	1	4	146	16	0.641
6	1	4	146	20	0.65
6	1	4	146	24	0.656
6	1	4	146	32	0.672
6	1	6	4332	10	0.7
6	1	6	4332	16	0.729
6	1	6	4332	20	0.733
6	1	6	4332	24	0.743
6	1	6	4332	32	0.75
7	1	2	7	4	0.25
7	1	2	7	6	0.333
7	1	2	7	10	0.4
7	1	2	7	16	0.406
7	1	3	37	4	0.417
7	1	2	7	20	0.425

Table C-1: Construction 2.2 for values: $q = 2$ to 7 and $N = 4, 6, 10, 16, 20, 24, 32$.

q	t	n	$\mathcal{S}_q^n$	N	R
7	1	2	7	24	0.438
7	1	2	7	32	0.438
7	1	3	37	6	0.5
7	1	4	231	4	0.5
7	1	3	37	10	0.533
7	1	4	231	6	0.542
7	1	5	1451	4	0.55
7	1	3	37	16	0.563
7	1	3	37	20	0.567
7	1	7	60691	4	0.571
7	1	3	37	24	0.583
7	1	6	9331	4	0.583
7	1	3	37	32	0.594
7	1	5	1451	6	0.6
7	1	4	231	10	0.625
7	1	6	9331	6	0.639
7	1	4	231	16	0.641
7	1	5	1451	10	0.66
7	1	4	231	20	0.663
7	1	4	231	24	0.667
7	1	7	60691	6	0.667
7	1	4	231	32	0.672
7	1	5	1451	16	0.7
7	1	6	9331	10	0.7
7	1	5	1451	20	0.71
7	1	7	60691	10	0.714
7	1	5	1451	24	0.717
7	1	5	1451	32	0.719
7	1	6	9331	16	0.729
7	1	6	9331	20	0.742
7	1	6	9331	24	0.75
7	1	7	60691	16	0.75
7	1	6	9331	32	0.755
7	1	7	60691	20	0.764
7	1	7	60691	24	0.774
7	1	7	60691	32	0.781
8	1	2	8	4	0.375
8	1	2	8	6	0.417
8	1	2	8	10	0.45
8	1	2	8	16	0.469
8	1	2	8	20	0.475
8	1	2	8	24	0.479
8	1	2	8	32	0.484
8	1	4	344	4	0.5
8	1	4	344	6	0.583
8	1	6	18152	4	0.583
8	1	4	344	10	0.625
8	1	6	18152	6	0.639
8	1	4	344	16	0.656
8	1	4	344	20	0.663
8	1	4	344	24	0.667
8	1	4	344	32	0.68
8	1	6	18152	10	0.7
8	1	6	18152	16	0.729
8	1	6	18152	20	0.742
8	1	6	18152	24	0.75
8	1	6	18152	32	0.76
9	1	2	9	4	0.25
9	1	2	9	6	0.333
9	1	2	9	10	0.4
9	1	3	61	4	0.417
9	1	2	9	20	0.425
9	1	2	9	16	0.438
9	1	2	9	24	0.438
9	1	2	9	32	0.453

q	t	n	$\mathcal{S}_q^n$	N	R
9	1	3	61	6	0.5
9	1	4	489	4	0.5
9	1	3	61	10	0.533
9	1	5	3951	4	0.55
9	1	3	61	16	0.563
9	1	3	61	20	0.583
9	1	3	61	24	0.583
9	1	4	489	6	0.583
9	1	6	32661	4	0.583
9	1	3	61	32	0.594
9	1	5	3951	6	0.6
9	1	4	489	10	0.625
9	1	6	32661	6	0.639
9	1	4	489	16	0.656
9	1	5	3951	10	0.66
9	1	4	489	20	0.663
9	1	4	489	24	0.667
9	1	4	489	32	0.68
9	1	5	3951	16	0.7
9	1	6	32661	10	0.7
9	1	5	3951	20	0.71
9	1	5	3951	24	0.717
9	1	5	3951	32	0.725
9	1	6	32661	16	0.729
9	1	6	32661	20	0.742
9	1	6	32661	24	0.75
9	1	6	32661	32	0.76
10	1	2	10	4	0.375
10	1	2	10	6	0.417
10	1	2	10	10	0.45
10	1	2	10	16	0.469
10	1	2	10	20	0.475
10	1	2	10	24	0.479
10	1	2	10	32	0.484
10	1	4	670	4	0.5
10	1	4	670	6	0.583
10	1	6	55252	4	0.583
10	1	4	670	10	0.625
10	1	6	55252	6	0.639
10	1	4	670	16	0.656
10	1	4	670	20	0.663
10	1	4	670	24	0.667
10	1	4	670	32	0.68
10	1	6	55252	10	0.7
10	1	6	55252	16	0.74
10	1	6	55252	20	0.75
10	1	6	55252	24	0.757
10	1	6	55252	32	0.766
11	1	2	11	4	0.25
11	1	2	11	6	0.333
11	1	2	11	10	0.4
11	1	3	91	4	0.417
11	1	2	11	16	0.438
11	1	2	11	20	0.45
11	1	2	11	32	0.453
11	1	2	11	24	0.458
11	1	3	91	6	0.5
11	1	4	891	4	0.5
11	1	3	91	10	0.533
11	1	5	8801	4	0.55
11	1	3	91	16	0.583
11	1	3	91	20	0.583
11	1	4	891	6	0.583
11	1	6	88913	4	0.583
11	1	3	91	24	0.597

q	t	n	$\mathcal{S}_q^n$	N	R
11	1	5	8801	6	0.6
11	1	3	91	32	0.604
11	1	4	891	10	0.625
11	1	6	88913	6	0.639
11	1	4	891	16	0.656
11	1	4	891	20	0.663
11	1	4	891	24	0.677
11	1	4	891	32	0.68
11	1	5	8801	10	0.68
11	1	5	8801	16	0.7
11	1	6	88913	10	0.7
11	1	5	8801	20	0.71
11	1	5	8801	24	0.725
11	1	5	8801	32	0.731
11	1	6	88913	16	0.74
11	1	6	88913	20	0.75
11	1	6	88913	24	0.757
11	1	6	88913	32	0.766
12	1	2	12	6	0.333
12	1	2	12	4	0.375
12	1	2	12	10	0.45
12	1	2	12	20	0.45
12	1	2	12	24	0.458
12	1	2	12	16	0.469
12	1	2	12	32	0.484
12	1	4	1156	4	0.5
12	1	4	1156	6	0.583
12	1	4	1156	10	0.625
12	1	4	1156	16	0.656
12	1	4	1156	20	0.663
12	1	4	1156	24	0.677
12	1	4	1156	32	0.68
13	1	2	13	4	0.25
13	1	2	13	6	0.333
13	1	2	13	10	0.4
13	1	3	127	4	0.417
13	1	2	13	16	0.438
13	1	2	13	20	0.45
13	1	2	13	24	0.458
13	1	2	13	32	0.469
13	1	3	127	6	0.5
13	1	4	1469	4	0.5
13	1	3	127	10	0.533
13	1	5	17151	4	0.55
13	1	3	127	16	0.583
13	1	3	127	20	0.583
13	1	4	1469	6	0.583
13	1	3	127	24	0.597
13	1	3	127	32	0.604
13	1	4	1469	10	0.625
13	1	5	17151	6	0.633
13	1	4	1469	16	0.656
13	1	4	1469	20	0.675
13	1	4	1469	24	0.677
13	1	5	17151	10	0.68
13	1	4	1469	32	0.688
13	1	5	17151	16	0.713
13	1	5	17151	20	0.72
13	1	5	17151	24	0.725
13	1	5	17151	32	0.731
14	1	2	14	4	0.375
14	1	2	14	6	0.417
14	1	2	14	10	0.45
14	1	2	14	16	0.469
14	1	2	14	20	0.475

Table C-2: Construction 2.2 for values: $q = 7$ to 14 and $N = 4, 6, 10, 16, 20, 24, 32$.

q	t	n	$\mathcal{S}_q^n$	N	R
14	1	2	14	24	0.479
14	1	2	14	32	0.484
14	1	4	1834	4	0.5
14	1	4	1834	6	0.583
14	1	4	1834	10	0.625
14	1	4	1834	16	0.656
14	1	4	1834	20	0.675
14	1	4	1834	24	0.677
14	1	4	1834	32	0.688
15	1	2	15	4	0.25
15	1	2	15	6	0.333
15	1	2	15	10	0.4
15	1	3	169	4	0.417
15	1	2	15	16	0.438
15	1	2	15	20	0.45
15	1	2	15	24	0.458
15	1	2	15	32	0.469
15	1	3	169	6	0.5
15	1	4	2255	4	0.5
15	1	5	30381	4	0.55
15	1	3	169	10	0.567
15	1	3	169	16	0.583
15	1	3	169	20	0.583
15	1	4	2255	6	0.583
15	1	3	169	24	0.597
15	1	3	169	32	0.604
15	1	4	2255	10	0.625
15	1	5	30381	6	0.633
15	1	4	2255	16	0.656
15	1	4	2255	20	0.675
15	1	4	2255	24	0.677
15	1	5	30381	10	0.68
15	1	4	2255	32	0.688
15	1	5	30381	16	0.713
15	1	5	30381	20	0.72
15	1	5	30381	24	0.725
15	1	5	30381	32	0.738
16	1	2	16	4	0.375
16	1	2	16	6	0.417
16	1	2	16	10	0.45
16	1	2	16	16	0.469
16	1	2	16	20	0.475
16	1	2	16	24	0.479
16	1	2	16	32	0.484
16	1	4	2736	4	0.5
16	1	4	2736	6	0.583
16	1	4	2736	10	0.625
16	1	4	2736	16	0.656
16	1	4	2736	20	0.675
16	1	4	2736	24	0.677
16	1	4	2736	32	0.688
17	1	2	17	4	0.25
17	1	2	17	6	0.333
17	1	2	17	10	0.4
17	1	3	217	4	0.417
17	1	2	17	16	0.438
17	1	2	17	20	0.45
17	1	2	17	24	0.458
17	1	2	17	32	0.469
17	1	3	217	6	0.5
17	1	4	3281	4	0.5
17	1	5	50101	4	0.55
17	1	3	217	10	0.567
17	1	3	217	16	0.583
17	1	4	3281	6	0.583
17	1	3	217	24	0.597
17	1	3	217	20	0.6
17	1	3	217	32	0.604

q	t	n	$\mathcal{S}_q^n$	N	R
17	1	4	3281	10	0.625
17	1	5	50101	6	0.633
17	1	4	3281	16	0.656
17	1	4	3281	20	0.675
17	1	4	3281	24	0.677
17	1	5	50101	10	0.68
17	1	4	3281	32	0.688
17	1	5	50101	16	0.713
17	1	5	50101	20	0.72
17	1	5	50101	24	0.725
17	1	5	50101	32	0.738
18	1	2	18	4	0.375
18	1	2	18	6	0.417
18	1	2	18	10	0.45
18	1	2	18	16	0.469
18	1	2	18	20	0.475
18	1	2	18	24	0.479
18	1	2	18	32	0.484
18	1	4	3894	4	0.5
18	1	4	3894	6	0.583
18	1	4	3894	10	0.625
18	1	4	3894	16	0.656
18	1	4	3894	20	0.675
18	1	4	3894	24	0.677
18	1	4	3894	32	0.688
19	1	2	19	4	0.25
19	1	2	19	6	0.333
19	1	2	19	10	0.4
19	1	3	271	4	0.417
19	1	2	19	16	0.438
19	1	2	19	20	0.45
19	1	2	19	24	0.458
19	1	2	19	32	0.469
19	1	3	271	6	0.5
19	1	4	4579	4	0.5
19	1	5	78151	4	0.55
19	1	3	271	10	0.567
19	1	3	271	16	0.583
19	1	4	4579	6	0.583
19	1	3	271	24	0.597
19	1	3	271	20	0.6
19	1	3	271	32	0.604
19	1	4	4579	10	0.625
19	1	5	78151	6	0.633
19	1	4	4579	16	0.656
19	1	4	4579	20	0.675
19	1	4	4579	24	0.677
19	1	5	78151	10	0.68
19	1	4	4579	32	0.688
19	1	5	78151	16	0.713
19	1	5	78151	20	0.72
19	1	5	78151	24	0.733
19	1	5	78151	32	0.738
20	1	2	20	4	0.375
20	1	2	20	6	0.417
20	1	2	20	10	0.45
20	1	2	20	16	0.469
20	1	2	20	20	0.475
20	1	2	20	24	0.479
20	1	2	20	32	0.484
20	1	4	5340	4	0.5
20	1	4	5340	6	0.583
20	1	4	5340	10	0.625
20	1	4	5340	16	0.656
20	1	4	5340	20	0.675
20	1	4	5340	24	0.677
20	1	4	5340	32	0.688

Table C-3: Construction 2.2 for values: $q = 14$ to 20 and $N = 4, 6, 10, 16, 20, 24, 32$.

q	t	n	$\mathcal{S}_q^n$	N	R
3	2	2	3	4	0.25
3	2	2	3	6	0.25
3	2	3	7	4	0.25
3	2	4	19	4	0.313
3	2	6	141	4	0.333
3	2	2	3	10	0.35
3	2	7	393	4	0.357
3	2	2	3	16	0.375
3	2	3	7	6	0.389
3	2	2	3	20	0.4
3	2	3	7	10	0.4
3	2	2	3	24	0.417
3	2	4	19	6	0.417
3	2	2	3	32	0.438
3	2	5	51	6	0.467
3	2	6	141	6	0.472
3	2	3	7	16	0.479
3	2	3	7	20	0.5
3	2	5	51	4	0.5
3	2	7	393	6	0.5
3	2	3	7	24	0.514
3	2	4	19	10	0.525
3	2	3	7	32	0.531
3	2	5	51	10	0.56
3	2	4	19	16	0.578
3	2	4	19	24	0.583
3	2	6	141	10	0.583
3	2	4	19	20	0.6
3	2	4	19	32	0.602
3	2	5	51	16	0.613
3	2	7	393	10	0.614
3	2	5	51	20	0.63
3	2	5	51	24	0.642
3	2	6	141	16	0.646
3	2	5	51	32	0.663
3	2	6	141	20	0.667
3	2	7	393	16	0.67
3	2	6	141	24	0.681
3	2	7	393	20	0.693
3	2	6	141	32	0.698
3	2	7	393	24	0.708
3	2	7	393	32	0.723
4	2	2	4	4	0.25
4	2	4	44	4	0.313
4	2	2	4	6	0.333
4	2	2	4	10	0.35
4	2	6	580	4	0.375
4	2	2	4	16	0.406
4	2	4	44	6	0.417
4	2	2	4	20	0.425
4	2	2	4	24	0.438
4	2	2	4	32	0.438
4	2	6	580	6	0.5
4	2	4	44	10	0.525
4	2	4	44	16	0.578
4	2	4	44	20	0.6
4	2	6	580	10	0.6
4	2	4	44	24	0.615
4	2	4	44	32	0.633
4	2	6	580	16	0.667
4	2	6	580	20	0.683
4	2	6	580	24	0.694
4	2	6	580	32	0.714
5	2	2	5	4	0.25
5	2	3	19	4	0.25
5	2	2	5	10	0.3
5	2	4	85	4	0.313
5	2	2	5	6	0.333
5	2	5	381	4	0.35
5	2	2	5	16	0.375
5	2	3	19	6	0.389
5	2	4	85	6	0.417
5	2	2	5	32	0.422
5	2	2	5	20	0.425
5	2	2	5	24	0.438
5	2	3	19	10	0.467
5	2	5	381	6	0.467
5	2	3	19	16	0.521
5	2	4	85	10	0.525
5	2	3	19	24	0.528
5	2	3	19	20	0.533
5	2	3	19	32	0.552
5	2	5	381	10	0.58
5	2	4	85	16	0.594
5	2	4	85	20	0.613
5	2	4	85	24	0.625
5	2	5	381	16	0.638
5	2	4	85	32	0.641
5	2	5	381	20	0.66
5	2	5	381	24	0.675
5	2	5	381	32	0.688
6	2	2	6	4	0.125
6	2	2	6	6	0.25
6	2	2	6	10	0.3
6	2	4	146	4	0.313
6	2	2	6	16	0.344
6	2	2	6	20	0.375
6	2	2	6	24	0.375
6	2	2	6	32	0.391
6	2	4	146	6	0.458
6	2	4	146	10	0.55
6	2	4	146	16	0.594
6	2	4	146	20	0.613
6	2	4	146	24	0.625
6	2	4	146	32	0.641
7	2	2	7	4	0.25
7	2	3	37	4	0.25
7	2	4	231	4	0.313
7	2	2	7	6	0.333
7	2	2	7	10	0.35
7	2	3	37	6	0.389
7	2	2	7	16	0.406
7	2	2	7	20	0.425
7	2	2	7	24	0.438
7	2	2	7	32	0.453
7	2	4	231	6	0.458
7	2	3	37	10	0.467
7	2	3	37	16	0.521
7	2	3	37	20	0.55
7	2	4	231	10	0.55
7	2	3	37	24	0.556
7	2	3	37	32	0.573
7	2	4	231	16	0.609
7	2	4	231	20	0.625
7	2	4	231	24	0.635
7	2	4	231	32	0.648
8	2	2	8	4	0.25
8	2	2	8	6	0.333
8	2	2	8	10	0.4
8	2	2	8	16	0.438
8	2	2	8	24	0.438
8	2	2	8	20	0.45
8	2	2	8	32	0.453
8	2	4	344	4	0.5
8	2	4	344	6	0.583
8	2	4	344	10	0.625
8	2	4	344	20	0.625
8	2	4	344	24	0.635
8	2	4	344	16	0.656
8	2	4	344	32	0.656
9	2	2	9	4	0.25
9	2	3	61	4	0.25
9	2	4	489	4	0.313
9	2	2	9	6	0.333
9	2	3	61	6	0.389
9	2	2	9	10	0.4
9	2	2	9	16	0.438
9	2	2	9	24	0.438
9	2	2	9	20	0.45
9	2	2	9	32	0.453
9	2	4	489	6	0.458
9	2	3	61	10	0.467
9	2	3	61	16	0.542
9	2	3	61	20	0.55
9	2	4	489	10	0.55
9	2	3	61	24	0.569
9	2	3	61	32	0.583
9	2	4	489	16	0.609
9	2	4	489	20	0.625
9	2	4	489	24	0.635
9	2	4	489	32	0.656
10	2	2	10	4	0.125
10	2	2	10	6	0.25
10	2	4	670	4	0.313
10	2	2	10	10	0.35
10	2	2	10	16	0.406
10	2	2	10	24	0.417
10	2	2	10	32	0.422
10	2	2	10	20	0.425
10	2	4	670	6	0.458
10	2	4	670	10	0.55
10	2	4	670	16	0.609
10	2	4	670	20	0.625
10	2	4	670	24	0.646
10	2	4	670	32	0.656
11	2	2	11	4	0.25
11	2	3	91	4	0.25
11	2	4	891	4	0.313
11	2	2	11	6	0.333
11	2	3	91	6	0.389
11	2	2	11	10	0.4
11	2	2	11	16	0.406
11	2	2	11	20	0.425
11	2	2	11	24	0.438
11	2	2	11	32	0.453
11	2	4	891	6	0.458
11	2	3	91	10	0.467
11	2	3	91	16	0.542
11	2	3	91	20	0.55
11	2	4	891	10	0.55
11	2	3	91	24	0.569
11	2	3	91	32	0.583
11	2	4	891	16	0.609
11	2	4	891	20	0.625
11	2	4	891	24	0.646

Table C-4: Construction 2.3 for values: $q = 3 - 11$ and $N = 4, 6, 10, 16, 20, 24, 32$.

q	t	n	$\mathcal{S}_q^n$	N	R
11	2	4	891	32	0.656
12	2	2	12	4	0.125
12	2	2	12	6	0.25
12	2	2	12	10	0.35
12	2	2	12	16	0.375
12	2	2	12	20	0.4
12	2	2	12	24	0.417
12	2	2	12	32	0.422
13	2	2	13	4	0.25
13	2	3	127	4	0.25
13	2	2	13	6	0.333
13	2	3	127	6	0.389
13	2	2	13	10	0.4
13	2	2	13	16	0.406
13	2	2	13	20	0.425
13	2	2	13	24	0.438
13	2	2	13	32	0.453
13	2	3	127	10	0.5
13	2	3	127	16	0.542
13	2	3	127	20	0.55
13	2	3	127	24	0.569
13	2	3	127	32	0.583
14	2	2	14	4	0.125
14	2	2	14	6	0.25
14	2	2	14	10	0.35
14	2	2	14	16	0.375
14	2	2	14	20	0.4
14	2	2	14	24	0.417
14	2	2	14	32	0.438
15	2	2	15	4	0.125
15	2	2	15	6	0.25
15	2	2	15	10	0.35
15	2	2	15	16	0.375
15	2	2	15	24	0.396
15	2	2	15	20	0.4
15	2	3	169	4	0.417
15	2	2	15	32	0.422
15	2	3	169	6	0.5
15	2	3	169	10	0.5
15	2	3	169	16	0.542
15	2	3	169	20	0.567
15	2	3	169	24	0.569
15	2	3	169	32	0.583
16	2	2	16	4	0.25
16	2	2	16	6	0.333
16	2	2	16	10	0.4

q	t	n	$\mathcal{S}_q^n$	N	R
16	2	2	16	16	0.438
16	2	2	16	20	0.45
16	2	2	16	24	0.458
16	2	2	16	32	0.469
17	2	2	17	4	0.25
17	2	3	217	4	0.25
17	2	2	17	6	0.333
17	2	3	217	6	0.389
17	2	2	17	10	0.4
17	2	2	17	20	0.425
17	2	2	17	16	0.438
17	2	2	17	24	0.438
17	2	2	17	32	0.453
17	2	3	217	10	0.5
17	2	3	217	16	0.542
17	2	3	217	20	0.567
17	2	3	217	24	0.569
17	2	3	217	32	0.583
18	2	2	18	4	0.125
18	2	2	18	6	0.25
18	2	2	18	10	0.35
18	2	2	18	20	0.4
18	2	2	18	16	0.406
18	2	2	18	24	0.417
18	2	2	18	32	0.438
19	2	2	19	4	0.25
19	2	3	271	4	0.25
19	2	2	19	6	0.333
19	2	3	271	6	0.389
19	2	2	19	10	0.4
19	2	2	19	16	0.438
19	2	2	19	24	0.438
19	2	2	19	20	0.45
19	2	2	19	32	0.453
19	2	3	271	10	0.5
19	2	3	271	16	0.542
19	2	3	271	20	0.567
19	2	3	271	24	0.569
19	2	3	271	32	0.594
20	2	2	20	4	0.125
20	2	2	20	6	0.25
20	2	2	20	10	0.35
20	2	2	20	16	0.406
20	2	2	20	24	0.417
20	2	2	20	20	0.425
20	2	2	20	32	0.438

Table C-5: Construction 2.3 for values: $q = 11 - 20$ and $N = 4, 6, 10, 16, 20, 24, 32$.

q	t	n	$\mathcal{S}_q^n$	N	R
2	2	4	6	4	0.25
2	2	4	6	6	0.292
2	2	8	70	4	0.313
2	2	6	20	4	0.333
2	2	10	252	4	0.375
2	2	4	6	10	0.425
2	2	4	6	16	0.469
2	2	6	20	6	0.472
2	2	8	70	6	0.479
2	2	4	6	20	0.5
2	2	6	20	10	0.5
2	2	8	70	10	0.513
2	2	10	252	6	0.517
2	2	4	6	24	0.521
2	2	4	6	32	0.547
2	2	10	252	10	0.58
2	2	6	20	16	0.583
2	2	8	70	16	0.602
2	2	6	20	20	0.608
2	2	6	20	24	0.625
2	2	6	20	32	0.63
2	2	8	70	20	0.631
2	2	8	70	24	0.651
2	2	10	252	16	0.663
2	2	8	70	32	0.672
2	2	10	252	20	0.69
2	2	10	252	24	0.692
2	2	10	252	32	0.713
3	2	5	51	4	0.35
3	2	7	393	4	0.357
3	2	9	3139	4	0.361
3	2	6	141	4	0.375
3	2	5	51	6	0.433
3	2	8	1107	4	0.469
3	2	6	141	6	0.472
3	2	7	393	6	0.476
3	2	9	3139	6	0.519
3	2	8	1107	6	0.521
3	2	5	51	10	0.54
3	2	6	141	10	0.583
3	2	5	51	20	0.59
3	2	5	51	16	0.6
3	2	7	393	10	0.6
3	2	5	51	24	0.617
3	2	6	141	16	0.635
3	2	5	51	32	0.638
3	2	8	1107	10	0.638
3	2	9	3139	10	0.644
3	2	6	141	20	0.658
3	2	7	393	16	0.661
3	2	6	141	24	0.674
3	2	7	393	20	0.679
3	2	6	141	32	0.693
3	2	8	1107	16	0.695
3	2	7	393	24	0.696
3	2	9	3139	16	0.701
3	2	8	1107	20	0.713
3	2	7	393	32	0.719
3	2	9	3139	20	0.728
3	2	8	1107	24	0.729
3	2	9	3139	24	0.741
3	2	8	1107	32	0.746
3	2	9	3139	32	0.76
4	2	4	44	4	0.313
4	2	6	580	4	0.333
4	2	4	44	6	0.417
4	2	6	580	6	0.472
4	2	4	44	10	0.5
4	2	8	8092	4	0.5
4	2	4	44	16	0.547
4	2	6	580	10	0.567
4	2	4	44	20	0.575
4	2	4	44	24	0.594
4	2	8	8092	10	0.6
4	2	8	8092	6	0.604
4	2	4	44	32	0.609
4	2	6	580	16	0.646
4	2	6	580	20	0.658
4	2	6	580	32	0.677
4	2	6	580	24	0.681
4	2	8	8092	16	0.688
4	2	8	8092	20	0.706
4	2	8	8092	24	0.724
4	2	8	8092	32	0.742
5	2	4	85	4	0.313
5	2	6	1751	4	0.333
5	2	5	381	4	0.35
5	2	8	38165	4	0.375
5	2	4	85	6	0.458
5	2	5	381	6	0.467
5	2	4	85	10	0.5
5	2	6	1751	6	0.5
5	2	8	38165	6	0.542
5	2	5	381	10	0.56
5	2	4	85	16	0.563
5	2	4	85	20	0.575
5	2	4	85	24	0.594
5	2	4	85	32	0.609
5	2	6	1751	10	0.617
5	2	5	381	16	0.625
5	2	8	38165	10	0.625
5	2	5	381	20	0.64
5	2	5	381	24	0.658
5	2	6	1751	16	0.667
5	2	6	1751	20	0.667
5	2	5	381	32	0.681
5	2	6	1751	24	0.688
5	2	8	38165	16	0.695
5	2	8	38165	20	0.713
5	2	6	1751	32	0.714
5	2	8	38165	24	0.729
5	2	8	38165	32	0.746
6	2	6	4332	4	0.458
6	2	6	4332	6	0.556
6	2	6	4332	10	0.633
6	2	6	4332	16	0.656
6	2	6	4332	20	0.675
6	2	6	4332	24	0.694
6	2	6	4332	32	0.708
7	2	4	231	4	0.313
7	2	6	9331	4	0.333
7	2	4	231	6	0.375
7	2	7	60691	4	0.393
7	2	4	231	10	0.475
7	2	6	9331	6	0.5
7	2	7	60691	6	0.524
7	2	4	231	16	0.547
7	2	4	231	20	0.575
7	2	4	231	24	0.594
7	2	6	9331	10	0.6

Table C-6: Construction 2.4 for values: $q = 2$ to 7 and $N = 4, 6, 10, 16, 20, 24, 32$.

q	t	n	$\mathcal{S}_q^n$	N	R
7	2	4	231	32	0.617
7	2	7	60691	10	0.643
7	2	6	9331	16	0.667
7	2	6	9331	20	0.683
7	2	7	60691	16	0.696
7	2	6	9331	24	0.701
7	2	6	9331	32	0.719
7	2	7	60691	20	0.721
7	2	7	60691	24	0.732
7	2	7	60691	32	0.75
8	2	4	344	4	0.313
8	2	4	344	6	0.458
8	2	4	344	10	0.55
8	2	4	344	16	0.609
8	2	4	344	20	0.625
8	2	4	344	24	0.625
8	2	4	344	32	0.648
9	2	4	489	4	0.313
9	2	6	32661	4	0.333
9	2	5	3951	4	0.35
9	2	4	489	6	0.417
9	2	5	3951	6	0.5
9	2	6	32661	6	0.5
9	2	4	489	10	0.525
9	2	4	489	16	0.594
9	2	5	3951	10	0.6
9	2	4	489	20	0.613
9	2	6	32661	10	0.617
9	2	4	489	24	0.635
9	2	4	489	32	0.648
9	2	5	3951	16	0.65
9	2	5	3951	20	0.67
9	2	5	3951	24	0.675
9	2	6	32661	16	0.677
9	2	5	3951	32	0.7
9	2	6	32661	24	0.701
9	2	6	32661	20	0.708
9	2	6	32661	32	0.719
10	2	6	55252	4	0.333
10	2	6	55252	6	0.5
10	2	6	55252	10	0.6
10	2	6	55252	16	0.667
10	2	6	55252	24	0.694
10	2	6	55252	20	0.7
10	2	6	55252	32	0.719
11	2	3	91	4	0.25
11	2	5	8801	4	0.35
11	2	6	88913	4	0.375
11	2	3	91	6	0.389
11	2	3	91	10	0.433
11	2	4	891	4	0.438
11	2	5	8801	6	0.467
11	2	3	91	16	0.479
11	2	6	88913	6	0.5
11	2	3	91	20	0.517
11	2	3	91	24	0.542
11	2	4	891	6	0.542
11	2	3	91	32	0.563
11	2	5	8801	10	0.58
11	2	4	891	16	0.594
11	2	4	891	10	0.6
11	2	4	891	20	0.613
11	2	6	88913	10	0.617
11	2	4	891	24	0.635
11	2	5	8801	16	0.638

q	t	n	$\mathcal{S}_q^n$	N	R
11	2	4	891	32	0.648
11	2	5	8801	20	0.66
11	2	6	88913	16	0.667
11	2	5	8801	24	0.675
11	2	6	88913	20	0.692
11	2	5	8801	32	0.7
11	2	6	88913	24	0.715
11	2	6	88913	32	0.734
12	2	4	1156	4	0.438
12	2	4	1156	10	0.525
12	2	4	1156	6	0.542
12	2	4	1156	16	0.594
12	2	4	1156	20	0.625
12	2	4	1156	24	0.635
12	2	4	1156	32	0.648
13	2	4	1469	4	0.313
13	2	4	1469	6	0.458
13	2	4	1469	10	0.55
13	2	4	1469	16	0.594
13	2	4	1469	20	0.625
13	2	4	1469	24	0.635
13	2	4	1469	32	0.656
15	2	4	2255	4	0.25
15	2	4	2255	6	0.417
15	2	4	2255	10	0.525
15	2	4	2255	16	0.578
15	2	4	2255	20	0.613
15	2	4	2255	24	0.625
15	2	4	2255	32	0.641
16	2	4	2736	4	0.313
16	2	4	2736	6	0.458
16	2	4	2736	10	0.55
16	2	4	2736	16	0.609
16	2	4	2736	24	0.625
16	2	4	2736	20	0.638
16	2	4	2736	32	0.641
17	2	3	217	4	0.25
17	2	4	3281	4	0.313
17	2	3	217	6	0.333
17	2	3	217	10	0.433
17	2	4	3281	6	0.458
17	2	3	217	16	0.5
17	2	3	217	20	0.533
17	2	4	3281	10	0.55
17	2	3	217	24	0.556
17	2	3	217	32	0.573
17	2	4	3281	16	0.625
17	2	4	3281	20	0.625
17	2	4	3281	24	0.635
17	2	4	3281	32	0.656
19	2	4	4579	4	0.313
19	2	4	4579	6	0.458
19	2	4	4579	10	0.55
19	2	4	4579	16	0.625
19	2	4	4579	24	0.635
19	2	4	4579	20	0.638
19	2	4	4579	32	0.656

Table C-7: Construction 2.4 for values: $q = 7$ to 19 and $N = 4, 6, 10, 16, 20, 24, 32$.

q	t	n	$\mathcal{S}_q^n$	N	R
2	2	4	6	4	0.375
2	2	4	6	6	0.375
2	2	8	70	4	0.375
2	2	10	252	4	0.475
2	2	4	6	10	0.5
2	2	6	20	4	0.5
2	2	8	70	6	0.521
2	2	4	6	16	0.531
2	2	4	6	20	0.55
2	2	8	70	10	0.55
2	2	4	6	24	0.563
2	2	6	20	6	0.583
2	2	10	252	6	0.583
2	2	4	6	32	0.586
2	2	6	20	10	0.6
2	2	8	70	16	0.633
2	2	10	252	10	0.64
2	2	6	20	16	0.646
2	2	8	70	20	0.656
2	2	6	20	20	0.658
2	2	6	20	24	0.667
2	2	6	20	32	0.672
2	2	8	70	24	0.672
2	2	8	70	32	0.691
2	2	10	252	16	0.7
2	2	10	252	24	0.717
2	2	10	252	20	0.72
2	2	10	252	32	0.738
4	2	6	580	4	0.417
4	2	4	44	4	0.438
4	2	4	44	6	0.5
4	2	6	580	6	0.528
4	2	8	8092	4	0.563
4	2	4	44	10	0.575
4	2	4	44	16	0.594
4	2	4	44	20	0.613
4	2	6	580	10	0.617
4	2	4	44	24	0.625
4	2	8	8092	10	0.638
4	2	4	44	32	0.641
4	2	8	8092	6	0.646
4	2	6	580	16	0.677
4	2	6	580	20	0.683
4	2	6	580	32	0.698
4	2	6	580	24	0.701
4	2	8	8092	16	0.711
4	2	8	8092	20	0.725
4	2	8	8092	24	0.74
4	2	8	8092	32	0.758
6	2	2	6	6	0.167
6	2	2	6	4	0.25
6	2	4	146	4	0.313
6	2	8	135954	4	0.344
6	2	2	6	10	0.35
6	2	2	6	16	0.375
6	2	2	6	20	0.4
6	2	2	6	24	0.417
6	2	4	146	6	0.417
6	2	2	6	32	0.438

q	t	n	$\mathcal{S}_q^n$	N	R
6	2	8	135954	6	0.479
6	2	6	4332	4	0.542
6	2	4	146	10	0.55
6	2	6	4332	6	0.583
6	2	4	146	16	0.594
6	2	8	135954	10	0.613
6	2	4	146	20	0.625
6	2	4	146	24	0.625
6	2	4	146	32	0.641
6	2	6	4332	10	0.667
6	2	6	4332	16	0.677
6	2	8	135954	16	0.688
6	2	6	4332	20	0.692
6	2	6	4332	24	0.708
6	2	8	135954	20	0.719
6	2	6	4332	32	0.724
6	2	8	135954	24	0.724
6	2	8	135954	32	0.746
10	2	6	55252	4	0.375
10	2	6	55252	6	0.528
10	2	6	55252	10	0.633
10	2	6	55252	16	0.688
10	2	6	55252	24	0.708
10	2	6	55252	20	0.717
10	2	6	55252	32	0.734
12	2	4	1156	4	0.5
12	2	4	1156	10	0.575
12	2	4	1156	6	0.583
12	2	4	1156	16	0.609
12	2	4	1156	20	0.65
12	2	4	1156	24	0.656
12	2	4	1156	32	0.664
16	2	2	16	4	0.375
16	2	4	2736	4	0.375
16	2	2	16	10	0.4
16	2	2	16	6	0.417
16	2	2	16	16	0.438
16	2	2	16	20	0.45
16	2	2	16	24	0.458
16	2	2	16	32	0.469
16	2	4	2736	6	0.5
16	2	4	2736	10	0.55
16	2	4	2736	16	0.609
16	2	4	2736	24	0.625
16	2	4	2736	20	0.638
16	2	4	2736	32	0.641
20	2	2	20	4	0.25
20	2	2	20	6	0.333
20	2	2	20	10	0.35
20	2	2	20	16	0.406
20	2	2	20	20	0.45
20	2	2	20	32	0.453
20	2	2	20	24	0.458

Table C-8: Construction 2.5 for values: $q = 2$ to 20 and $N = 4, 6, 10, 16, 20, 24, 32$.

Bibliography

[1] J.H. Weber, “Error-Correcting Codes", *TU Delft: Lecture Notes*, the Netherlands, 2007.

[2] www.networkworld.com/subnets/cisco/chapters/1587052695/graphics/04fig01.jpg

[3] S. Lin and D.J. Costello, Jr.,“Error Control Coding: Fundamentals and Applications", *Pearson Prentice Hall*, 2nd Edition, USA, pp. 1-35, 2004.

[4] R. Morelos-Zaragoza, “The Art of Error Correcting Codes", *Wiley*, 2nd Edition, USA, pp. 4-29, 2006.

[5] W. Huffman and V. Pless, “Fundamentals of Error-Correcting Codes", *Cambridge University Press*, USA, pp. 1-13, 2003.

[6] J.H. Weber, K.A. Schouhamer Immink, P.H. Siegel, and T.G. Swart, “Polarity-Balanced Codes", *Information Theory and Applications Workshop*, USA, 2013.

[7] T.G. Swart and J.H. Weber, “Efficient Balancing of q-ary Sequences with Parallel Decoding", *Proceedings of the 2009 IEEE International Symposium on Information Theory*, Seoul, Korea, pp. 1564-1568, June 28 - July 3, 2009.

[8] D.E. Knuth, “Efficient balanced codes", *IEEE Trans. Inf. Theory*, vol. 32, No. 1, pp. 51-53, Jan. 1986.

[9] K.A. Schouhamer Immink, and J.H. Weber, “Very Efficient Balanced Codes", *IEEE Journal on Selected Areas in Communications*, vol. 28, No. 2, pp. 188-192, Feb. 2010.

[10] J.H. Weber, and K.A. Schouhamer Immink, “Knuth’s Balanced Codes Revisited", *IEEE Transactions on Information Theory*, vol. 56, No. 4, pp. 1673-1679, April 2010.

[11] J.H. Weber, K.A. Schouhamer Immink, and H.C. Ferreira, “Error-correcting balanced Knuth codes", *IEEE Trans. Inf. Theory*, vol. 58, no. 1, pp. 82-89, Jan. 2012.

[12] T.G. Swart, and K.A. Schouhamer Immink, “Prefixless q-ary Balanced Codes with ECC", *IEEE Information Theory Workshop*, Sevilla, Sep. 2013.

[13] R. M. Capocelli, L. Gargano and U. Vaccaro, “Efficient q-ary immutable codes", *Discrete Applied Mathematics*, vol. 33, pp. 25-41, 1991.

[14] H. van Tilborg and M. Blaum, “On error-correcting balanced codes", *IEEE Transactions on Information Theory*, vol. 35, no. 5, pp. 1091-1095, Sep. 1989.

[15] J.C. Bowman, “Math 422 Coding Theory & Cryptography", *Lecture Notes, University of Alberta*, Edmonton Canada, January 17, 2010.

[16] R.A. Carrasco and M. Johnston, “Non-Binary Error Control Coding for Wireless Communication and Data Storage", *John Wiley & Sons, Ltd.*, UK, pp. 51-59, 2008.

[17] K.A. Schouhamer Immink, “Codes for Mass Data Storage Systems", *Shannon Foundation Publishers.*, 2nd Edition, The Netherlands, pp. 51 & 198, 2004.

Glossary

List of Acronyms

GPS	Global Positioning System
ECC	Error Correcting Codes
BPSK	Binary Phase-Shift Keying
QPSK	Quadrature Phase-Shift Keying
QAM	Quadrature Amplitude Modulation
ASK	Amplitude Shift Keying
FSK	Frequency Shift Keying
PSK	Phase Shift Keying
PAM	Pulse Amplitude Modulation
VoIP	Voice over Internet Protocol
DVD	Digital Versatile Disc
CD	Compact Disc
FEC	Forward Error Correction
ARQ	Automatic Repeat Request
SB	Symbol-Balanced
CB	Charge-Balanced
PB	Polarity-Balanced
CPB	Charged and Polarity-Balanced
DC	Direct Current

VLSI	Very Large Scale Integration
DSV	Digital Sum Variation
BCH	Bose, Ray-Chaudhuri and Hocquenghem
GF	Galois Field