Variable-length Non-Binary Constrained Sequence Codes

by

Xin Tu

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Communications

Department of Electrical and Computer Engineering

University of Alberta

Abstract

Block-oriented constrained sequence codes have historically been used in the digital communications and data storage industry. However, we contend that in certain applications, variable-length constrained sequence codes can be better than block codes in terms of efficiency and implementation complexity.

A technique to construct capacity-approaching constrained sequence codes with variable-length codewords was recently developed. This construction process can be divided into four parts. First, it is necessary to define a minimal set of prefix-free words whose concatenation satisfies the constraint. Then, sets of instantaneously decodable codewords are constructed by concatenating words from this minimal set through partial extensions of this minimal set. Third, the optimal mapping between variable-length source words and each set of variable-length codewords is determined using normalized geometric Huffman coding. Lastly, the average code rate of each code constructed is evaluated, and the code with highest code rate is selected as the best code.

Based on the four steps above, in this thesis we construct new non-binary codes for multilevel magnetic recording, and new balanced codes for transmission systems using quaternary phase shift keying modulation. Our codes have higher code rates (within 98% of capacity) than codes designed through any

other construction technique published to date.

Acknowledgements

The first thing I would like to thank my supervisor, Dr. Ivan Fair, for his support and guidance throughout my research.

I would also appreciate the stipend provided by the Natural Sciences and Engineering Research Council of Canada (NSERC). This work would not have been possible without this support.

Lastly, I also need to thank my family and friends, for their consistent love and support, without whom I may lose my passion and motivation.

Contents

List of Tables

List of Figures

List of Abbreviations

List of commonly used abbreviations (in alphabetical order)

BD	Blu-Ray disc
BPSK	Binary phase-shift keying
CD	Compact disc
CS	Constrained sequence
CSCs	Constrained sequence codes
DVD	Digital versatile disc
ECC	Error control coding
EFM	Eight-to-fourteen modulation
GHC	Geometric Huffman Coding
GS	Guided scrambling
i.i.d.	Independent and identically distributed
LFSW	Low frequency spectral weight
ML-RLL	Multi-level runlength limited
nGHC	Normalized Geometric Huffman Coding
OFDM	Orthogonal frequency division multiplexing
PAPR	Peak-to-average power ratio
PMF	Probability mass function
PSD	Power spectral density
PSK	Phase-shift keying
QPSK	Quadrature phase shift keyed
RLL	Runlength-limited
RDS	Running digital sum

Chapter 1

Introduction

With the rapid growth of electronic information technology, the requirement for high data density in transmission and data storage systems also continues to grow. This requirement has led to significant developments in coding technology in recent decades that have enabled high data rate transmission through a variety of mediums including optical fiber, terrestrial wireless, and satellites systems, in addition to high-density recording systems. In all circumstances, the application of coding techniques and information theory, motivated by Shannon's work [1], has led to significant improvement in these communication systems. The goal of coding is to improve the performance of a communication system, making it more robust to errors that occur as the result of noise on the channel or imperfect detection circuitry. In this thesis, we focus on a particular type of coding that we refer to as constrained sequence (CS) coding. This kind of coding is often called line coding in transmission systems and recording coding in storage systems.

A simple block diagram of a typical digital communication system is depicted in Figure 1.1. This system consists of three major sections, of which the first is source coding. Source coding attempts to represent the input data with as few symbols as possible to communicate it more efficiently. For example, shorter code words are usually assigned to symbols with high probability of

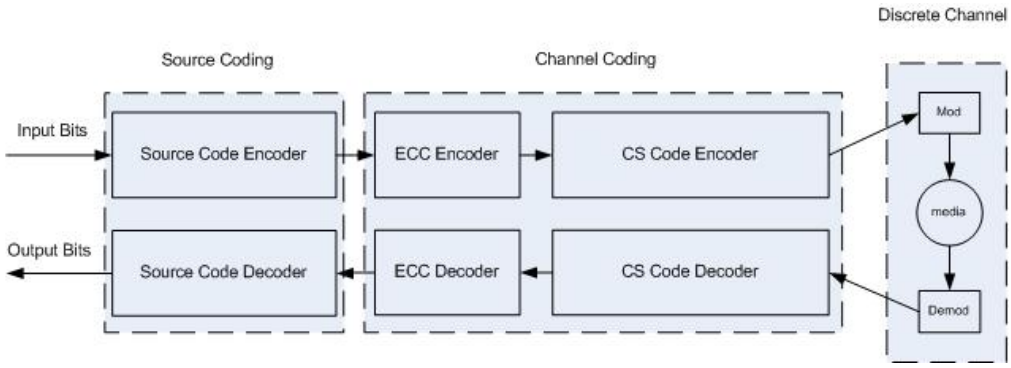

Figure 1.1: Structure of a digital communication system.

occurrence, and longer codewords are assigned to other symbols which rarely occur. In this way, we can reduce the redundancy in the same data sequence and conserve the number of bits to be sent to the channel.

The last block in Figure 1.1 encompasses the functionalities of the modulator, the channel, and the demodulator. This combination of functions is known as the Discrete Data Channel because both its input and output are discrete time signals with one value per symbol period.

The intermediate section is channel coding. Channel coding involves the addition of structured redundancy to the symbol sequence such that the overall performance of the system will be improved when real-world modulation and demodulation circuits with practical limitations are used. One form of channel coding is error control coding (ECC). ECC is a method of detecting and correcting errors that occur due to distortion of the signal on the channel. Specifically, the encoder introduces redundancy to the messages to be transmitted. At the output of the channel, the channel decoder uses this controlled redundancy to detect and correct the errors. By using this coding method, we can reduce the effect of error propagation to a certain extent.

The second form of channel coding used in digital communication systems is CS coding. Coding of this kind is used to translate the sequence of data symbols into a sequence of coded symbols that comply with the given channel constraint. Satisfying the constraint requires the introduction of some redundancy which

will reduce the amount of information that can be represented per symbol compared to that of an unconstrained symbol sequence. With block CS codes, blocks of m data symbols are represented by blocks of n coded symbols, where $n > m$. One performance metric for a code is the code rate R, defined as $R = \frac{m}{n}$. The quantity $1 - R$ is referred to as the redundancy of the code.

There are several different classes of CS codes (CSCs). The two that we consider in this thesis are runlength-limited (RLL) codes and balanced codes. RLL coding is a line coding technique used with bandwidth limited communication channels. These codes ensure limits to the minimum and the maximum number of like-valued consecutive symbols. Balanced codes are widely used in binary communication systems to ensure the presence of many transitions and an equal number of logic 1's and logic 0's in the encoded bit stream. These characteristics enable practical demodulators to accurately recover bit-level synchronization and establish the constant decision thresholds necessary for accurate symbol recovery [2]. Since balanced codes result in a null at DC in the continuous component of the spectrum of the encoded binary sequence, the terms balanced code and DC-free code are often used interchangeably [3].

To date, constrained sequence codes have been widely used in mass storage systems including magnetic hard disks and optical recording systems such as the compact disc (CD), digital versatile disc (DVD), and the Blu-Ray disc (BD). One of the best-known examples of a CS code for storage applications is the eight-to-fourteen modulation (EFM) [4] code that is used in CD players. Binary zeros and ones are represented by pits and lands written on the disc. The decoding circuitry exhibits better performance if the durations of pits and lands are not too short or too long. Therefore, EFM ensures that runlengths are no shorter than 2 bits and no longer than 10 bits. EFM also attempts to ensure that the encoded sequence is balanced.

Although constrained coding has been extensively deployed in the recording industry, there are also applications in other digital communication sys-

tems. Examples include, for instance, limiting the peak-to-average power ratio (PAPR) of signals in orthogonal frequency division multiplexing (OFDM) systems and scrambling techniques used in many transmission systems.

In the past, almost all CS codes (CSCs) used in practice have been fixed-length block codes, which fix the word length of the input user data and output symbols to predetermined fixed values often denoted m and n respectively. A primary reason for the use of fixed-length codes is that this simplifies the implementation of the encoder and decoder. However, recently a straightforward approach for constructing variable-length CSCs was proposed by Andrew Steadman [5]. His approach combines the use of normalized geometric Huffman coding (nGHC) with partial extensions of a minimal set of words that satisfies the constraint. This method can be used to construct simple variable-length CSCs that have a code rate approaching capacity.

1.1 Thesis Objective

The purpose of this thesis is to extend Steadman's recently developed construction method for variable-length constrained codes [5] to the case where the signalling alphabet is larger than binary. Using this approach we design non-binary multilevel RLL codes for magnetic recording systems and balanced codes for quadrature phase shift keyed (QPSK) transmission systems. Prior to the work in this thesis, non-binary constrained sequence codes with such a high code rate have not been constructed.

1.2 Thesis Organization

The thesis is organized as follows.

Chapter 2 presents an overview of constrained sequence codes. This chapter begins with a discussion of fundamental concepts of information theory,

including entropy and capacity. Markov chain modelling of encoding processes is then reviewed. With this model, we can evaluate the capacity of several families of constrained sequence codes. In addition to these concepts, we also highlight two common kinds of constrained sequence codes, RLL codes and balanced codes.

In Chapter 3 we focus on non-binary communication systems. We first present an overview of multilevel magnetic recording systems and outline fixed-length coding techniques that have been published for these recording systems. We then provide a brief introduction to QPSK transmission systems and discuss fixed-length balanced codes that have been proposed for these systems.

Chapters 4 and 5 present our novel work regarding variable-length RLL codes for multilevel magnetic recording systems and variable-length balanced codes for QPSK systems respectively. The codes we construct are simple and have a higher average code rate than any other codes that have been reported for these systems to date.

Lastly, we draw conclusions for this thesis in Chapter 6 and provide recommendations for future work.

Chapter 2

Background

An overview of constrained sequence codes is given in this chapter. This summary includes a discussion of several issues related to constructing and analyzing constrained sequence codes and their properties. Firstly, theoretical foundations of information, entropy and capacity are discussed in Section 2.1. In that section, we consider the fundamental question: how do we measure the amount of information emitted from a source? Two widely used classes of constrained codes, runlength-limited (RLL) codes and DC-free codes, are introduced in Section 2.2. Lastly, an overview of variable-length codes is presented in Section 2.3. With this overview, it becomes apparent that it is possible to construct variable-length codes in a straightforward manner, and that variable-length codes can have advantages when compared to fixed-length codes.

2.1 Information and Entropy

In this section, we give a brief introduction to fundamental concepts of information and entropy. In general, these concepts can be used as a guide in the preliminary stage of code design as well as an assessment tool for the overall performance of the system. This introduction is subdivided into several parts. In Subsections 2.1.1 and 2.1.2, we clarify some basic ideas of information the-

ory, including the definition of information and entropy along with some of their corresponding evaluation methods. In Subsections 2.1.3 through 2.1.6 we focus on Markov chains and the Markov information source which is a model of particular interest in our work. With this background, we can calculate the capacity of practical constrained systems.

2.1.1 Concept of Information

In information theory, the amount of uncertainty is related to the quantity of the information. The premise is that the less likely an event, the more information its occurrence contains. This relationship is satisfied when information is defined as the negative of the logarithm of the probability of an event [1], as shown in Equation (2.1).

$$I\left(x_i\right) = -\log\, p\left(x_i\right) \tag{2.1}$$

In this expression, x_i denotes an event and $p\left(x_i\right)$ denotes the probability of its occurrence. The reason for the use of the logarithm is that the information associated with independent events can be summed using this logarithm expression. The most common units for the amount of information is bits; in this case the logarithm has base 2. Information can also be measured in nats or Hartleys if a logarithm of base e or base 10, respectively, is used during its evaluation.

2.1.2 Entropy of Memoryless Sources

Entropy is defined as the average amount of information in a symbol sequence, and therefore it has the same units as information. A memoryless information source is a source that generates symbols that are statistically independent. Consider a finite set of symbols $X = \{x_1, x_2, ..., x_n\}$, with corresponding probabilities $P = \{p_1, p_2, ..., p_n\}$. Note that the sum of these probabilities

equals unity, that is $\sum_{i=1}^{n} p_i = 1$. By assuming statistical independence of symbols, the entropy of a memoryless source is straightforward to evaluate. It is directly related to the uncertainty or probability of the symbols generated by the source, which is shown as follows:

$$H\left(p\left(x_1\right), p\left(x_2\right), ..., p\left(x_n\right)\right) = -\sum_{i=1}^{n} p\left(x_i\right) \log_2 p\left(x_i\right) \tag{2.2}$$

In the above equation, the base of the logarithm is set to 2, which implies that entropy is measured in average bits of information per symbol. As noted above, other bases also can be used. However, we only consider base 2 in this thesis in order to be consistent with most of the literature. It can be shown that the entropy function achieves a maximum of $\log_2 n$ bits of information per symbol when the source symbols are equiprobable [3].

2.1.3 Markov Chains

A Markov chain is a random process that undergoes transitions from one state to another state in a state space. It possesses the property that the probability distribution of the next state depends only on the current state and not on any previous state. This particular property is called the Markov property. Markov chains have many applications as statistical models for real-world processes. In information theory, a Markov chain is usually represented as a discrete random process with dependent discrete random variables Z_n taken from the set $\{....., Z_0, Z_1, Z_2, ...\}$. These discrete random variables satisfy the Markov property:

$$Pr\left(Z_t = \sigma_{i_t} \mid Z_{t\ 1} = \sigma_{i_{t\ 1}}, Z_{t\ 2} = \sigma_{i_{t\ 2}}, ...\right) = Pr\left(Z_t = \sigma_{i_t} \mid Z_{t\ 1} = \sigma_{i_{t\ 1}}\right) \tag{2.3}$$

where t denotes time and the σ_i denote state values. Also, the notation $Pr\left(A \mid B\right)$ denotes the conditional probability of event A given that event B has occurred.

A Markov chain can be represented by a transition probability matrix $\mathbf{Q}$. Each element of $\mathbf{Q}$ denotes the probability of transitioning from one state to another state:

$$[\mathbf{Q}]_{ij} = Pr\left(Z_t = \sigma_j \mid Z_{t-1} = \sigma_i\right) \tag{2.4}$$

It is important to note that a Markov chain is not driven by an input. The only two features that characterize a Markov chain are the discrete random variables Z_n and the state transition probabilities.

A Markov process can also be described as a directed graph. The states form the vertices of the graph while the edges indicate valid transitions. An example of a directed graph is shown in Fig. 2.1.

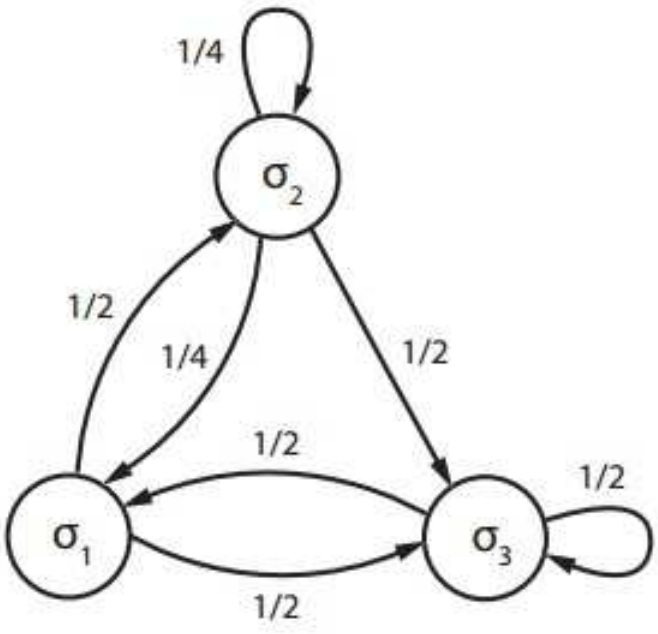

Figure 2.1: A directed graph of a Markov chain

With this directed graph, it is straightforward to obtain the transition probability matrix $[\mathbf{Q}]_{ij}$, shown below:

$$\mathbf{Q} = \begin{bmatrix} 0 & 1/2 & 1/2 \\ 1/4 & 1/4 & 1/2 \\ 1/2 & 0 & 1/2 \end{bmatrix} \tag{2.5}$$

This thesis considers Markov chains that are irreducible and regular, also referred to as ergodic. The irreducible property indicates that a state in the Markov chain can be reached from any other state in some number of transitions. Regularity refers to the fact that transitions between states are non-periodic.

2.1.4 Entropy of Markov Information Sources

In this subsection, we provide a brief introduction to the evaluation of entropy of Markov information sources. In this thesis, an information source is modeled as a Markov information source. Its output sequence $\{X_i\}$ will be described by a generating function ζ whose domain is the set of states. This relationship can be expressed as $X_i = \zeta(\sigma_i)$. Since there is, in general, dependence among symbols emitted from a Markov information source, there is usually some redundancy because each successive symbol has some predictability.

Only unifilar Markov information sources are considered in this thesis. A unifilar source is one which has, for each possible current state, different output symbols associated with each different successor state. A successor state is a state that can be reached in a single step from the present state with a transition probability greater than zero. For the Markov information source to be unifilar, each of these successors states σ_i has a different $\zeta(\sigma_i)$. The unifilar property ensures that there exists a one-to-one relationship between the output sequence and the sequence of states.

For an unifilar Markov information source, the evaluation of the entropy of state σ_i can be described as $H_i = H([\mathbf{Q}]_{i,j_1}, [\mathbf{Q}]_{i,j_2}, ..., [\mathbf{Q}]_{i,j_{n_i}})$, where $\sigma_{j_1}, ..., \sigma_{j_{n_i}}$ are the n_i successor states of σ_i. The overall entropy of the unifilar Markov

information source can then be calculated as:

$$H\{X\} = \sum_{i=1}^{L} \pi_i H_i \tag{2.6}$$

where π_i is the steady-state probability of being in state σ_i. The calculation of these asymptotically steady-state probabilities is considered in [3].

2.1.5 Capacity of Memoryless Discrete Noiseless Channel

The maximization of entropy for a particular channel constraint is called the capacity of the constrained channel. At the beginning of Shannon's well-known paper [1], he provides a definition for the capacity of a memoryless discrete noiseless channel, which is the basis of our thesis. All the concepts of capacity mentioned in this thesis are the capacity of the memoryless discrete noiseless channel. First he chooses the symbols randomly from a specified alphabet. These symbols can be denoted as:

$$\{a_1, a_2, ..., a_n\} \tag{2.7}$$

Each symbol is transmitted through a noiseless channel without alteration but has a duration or cost factor assigned to it. The costs of different symbols are specified as:

$$\{t_1, t_2, ..., t_n\} \tag{2.8}$$

Sequences of symbols are selected from the source according to particular constraint, where, for instance, some sequences of symbols are not to be transmitted. The question posed and answered by Shannon is how to measure the capacity of such a memoryless discrete noiseless channel. Shannon defines the

capacity of this noiseless channel as:

$$C = \lim_{T \to \infty} \frac{\log N(T)}{T} \tag{2.9}$$

where $N(T)$ is the number of distinct permissible messages of total duration or cost T. When all distinct messages are independent and have the same duration or cost, it can be shown that the maximum rate of transmission of information may only be obtained when all these messages are equiprobable.

2.1.6 Capacity of Markov Information Sources

As discussed above, the capacity of a noiseless sequence can be evaluated with Equation (2.9). However, it is often impractical for us to calculate the capacity in that manner, and another approach can be derived based on the representation of a Markov information source [3]. The key to this approach is to enumerate the number of different sequences generated by a Markov source. When the Markov source is unifilar, the enumeration of different sequences equals the enumeration of different paths that lead from one state to another state in the Markov source. Furthermore, matrix operations can help us count the number of paths.

In [3], the author demonstrates that the number of paths of length m is given by the $(i,j)\,th$ entry of $\mathbf{D}^m$, where $\mathbf{D}$ is the $N \times N$ one-step connection matrix of a given directed graph. If we use $[\mathbf{D}]_{ij}^m$ to represent the $(i,j)\,th$ entry of $\mathbf{D}^m$, then an iterative approach to calculate $[\mathbf{D}]_{ij}^m$ is as follows:

$$[\mathbf{D}]_{ij}^m = \sum_{h=1}^{N} [\mathbf{D}]_{ih}^{m-1} [\mathbf{D}]_{hj} \tag{2.10}$$

For large sequence lengths m, we may conveniently approximate the number of sequences $[\mathbf{D}]_{ij}^m$ by:

$$[\mathbf{D}]_{ij}^{m} \cong a_{ij}\lambda_{max}^{m} \tag{2.11}$$

where a_{ij} is a constant and λ_{max} is the largest real eigenvalue of the matrix $\mathbf{D}$ [6]. In other words, λ_{max} is the largest real root of the determinant equation

$$det\left[\mathbf{D} - z\mathbf{I}\right] = 0 \tag{2.12}$$

where $\mathbf{I}$ is the identity matrix with the same dimensions as $\mathbf{D}$. From Equation (2.11), we can draw the important conclusion that the number of different sequences grows exponentially with the sequence length m when m is large enough. Equation (2.11) can be rewritten as:

$$\frac{1}{m}\log\left[\mathbf{D}\right]_{ij}^{m} \simeq \frac{1}{m}\left(\log a_{ij} + m\log\lambda_{max}\right) \tag{2.13}$$

Using this result, the maximum entropy of the noiseless channel may then be evaluated by invoking (2.9) to yield:

$$C = \lim_{m\to\infty}\frac{1}{m}\log\left[\mathbf{D}_{ij}^{m}\right] = \log\lambda_{max} \tag{2.14}$$

2.2 Constrained Sequence Codes

Constrained sequence encoders accept an arbitrary bit stream as their input and convert this stream into a sequence of symbols, called a constrained sequence, that complies with a given constraint. There are a variety of constraints to make full use of a particular channel. We consider two widely enforced constraints here.

2.2.1 Runlength-limited Codes and (d, k) Sequences

A typical class of constrained sequence codes is runlength-limited (RLL) codes. RLL coding can be used to send arbitrary data over a communication channel

with bandwidth limits. The number of symbols between transitions is known as the runlengths. For instance, the runlengths in the sequence '0111100111000000' are 1, 4, 2, 3 and 6.

RLL codes are widely used in hard disk drives and with digital optical disc systems such as the CD, DVD, and Blu-ray systems. The purpose is to bound the length of stretches (runs) of repeated bits during which the signal does not change. If the runs are too long, clock recovery is difficult; if they are too short, the attenuation of high frequencies by the channel makes symbol detection less reliable. RLL codes help ensure that the boundaries between bits can be accurately found while efficiently using the media to store a large amount of data in a given space. Early disk drives used very simple encoding schemes, such as the RLL (0,1) FM code. Higher density RLL (2,7) and RLL (1,7) codes became the industry standard for hard disks by the early 1990s.

The numbers associated with those codes are the values of d and k, where $(d+1)$ and $(k+1)$ are the minimum and maximum runlengths in the encoded sequence. RLL sequences can be generated from (d, k) sequences in which there are at least d and at most k zeros between consecutive logic 1's. A (d, k) sequence is translated into an RLL sequence through change-of-state encoding in which a logic 1 is encoded as a change in value from the previous encoded symbol, and a logic 0 is encoded as the absence of a change.

2.2.2 DC-free Codes

In addition to RLL codes, DC-free codes have achieved widespread use in communication systems. As their name implies, DC-free codes are designed to ensure the presence of a spectral null at zero frequency in the continuous component of the spectrum of the coded signal. These codes are also called balanced codes because they ensure an equal number of logic 1's and logic 0's in the encoded sequence.

In digital communication systems, it is sometimes desirable for the signal

to contain low power at and near zero frequency. When digital communication signals are transmitted through metallic cables, DC-free codes have been applied to reduce the power lost due to coupling circuits and isolation transformers. Furthermore, balanced codes are widely used to ensure the presence of many transitions in the encoded sequence. These characteristics enable practical demodulators to accurately recover bit-level synchronization and establish the constant decision thresholds necessary for accurate symbol recovery.

Any code with balanced codewords, each of which has an equal number of zeros and ones, will have a spectral null at DC. However, there is another way to construct DC-free codes. As discussed in [3], a sequence is DC-free if and only if its running digital sum (RDS) is bounded. The RDS z_i is the accumulation of signalling values from the beginning of the sequence to any point following the ith symbol in the sequence $\{x_1, x_2, ..., x_i\}$, where a logic 1 has symbol value +1 and a logic 0 has symbol value -1. RDS can therefore be expressed as:

$$z_i = \sum_{j=-\infty}^{i} x_j = z_{i-1} + x_i \tag{2.15}$$

If the RDS of the sequence is bounded, the RDS only takes on values from a finite set. It is possible to identify the total number of different RDS values, denoted by N, as:

$$N = \mathbf{Max}\,(z_i) - \mathbf{Min}\,(z_i) + 1 \tag{2.16}$$

In general, the initial value of the RDS can be any value. The definition of N and properties of DC-free codes are equally applicable for all initial values, but for the sake of convenience, the initial value is usually set to 0.

The evaluation of the capacity of DC-free codes can be described as a function of N when there is a correspondence between the states and allowed values of z_i. It has been shown that [3]:

$$C(N) = \log_2 \left(2\cos\frac{\pi}{N+1} \right), N \geq 3 \tag{2.17}$$

The formula clearly demonstrates that capacity approaches one as N increases. For example, for $N = 10$, $C(N) = 0.94$, representing only a 6% reduction in capacity from that of an unconstrained sequence. It occurs because, as the RDS bounds increase, there are more sequences that satisfy the constraint so the capacity increases correspondingly.

2.3 Variable-length Codes

Almost all CSC codes developed to date have been fixed-length codes, also known as block codes since the messages are transmitted by fixed-length codewords that represent fixed-length source words. However, the focus of this thesis is the construction of variable-length codes. There has been limited effort in this regard to date. Some relevant examples include Franaszek's synchronous variable-length codes [7] [8], the variable-length bit stuffing algorithm proposed by Bender and Wolf [9] and the extended bit-flip techniques [10]. The advantage of variable-length codes is increased efficiency in the sense that, on average, fewer digits can be used to represent the same amount information for similar implementation complexity. We now consider a recently developed approach to constrained sequence coding that results in variable-length codes [11]. To achieve this goal, however, we need to know or make assumptions regarding the statistics of the message being sent. We can take advantage of these statistics to make more frequent source words correspond to shorter constrained sequences and less probable source words correspond to longer sequences. This is the basis of Huffman coding, which is a widely used approach for variable-length source coding.

In contrast to block codes, variable-length codes do not have a fixed code rate, therefore in our research we focus on the average code rate. The average

code rate depends on the statistics of the input and output sequences:

$$R_{avg} = \frac{E[L_{in}]}{E[L_{out}]}, \tag{2.18}$$

where L_{in} and L_{out} represent the length of the input and output sequences respectively while $E[L]$ represents the expected value of the random variable L. The value of this average code rate is within the scope of the code rates of individual codewords:

$$R_{min} = \mathbf{Min}[\frac{L_{in,i}}{L_{out,i}}], \tag{2.19}$$

$$R_{max} = \mathbf{Max}[\frac{L_{in,i}}{L_{out,i}}], \tag{2.20}$$

where the subscript i denotes the i^{th} source word-to-codeword relationship. It is important to note that while it is possible for R_{max} (which corresponds to a single codeword) to exceed the capacity of the constraint, the average code rate will be at most equal to capacity for any source and any code that satisfies the constraint.

2.3.1 Methods of Proposed Research

Our research is based on several fundamental features of constrained sequence coding and a recently developed variable-length code construction technique called normalized geometric Huffman coding (nGHC) [12]. We introduce some related concepts here.

Minimal Set

As the length of a sequence grows, the number of constraint-satisfying sequences also increases. We start by asking whether we can find a minimal set of words such that we can construct all the constraint-satisfying sequences by

concatenating words from this set. The answer is yes, and for some constraints this minimal set has a finite number of words, and for other constraints this set might be infinitely large. But there is a minimal set for each constrained sequence code, and we can construct all the constraint-satisfying sequences by concatenating sequences from the minimal set in any order.

As an example of the formation of a minimal set, consider (d, k) constrained sequence codes. First, we draw the state transition graph that describes the constraint. Based on this diagram, we choose one of these states as the initial state. We find all the paths that begin with this initial state and return to this state, and enumerate all the output sequences that arise in response to these paths. These output sequences, taken together, constitute a minimal set. An example will clarify the formation of this minimal set.

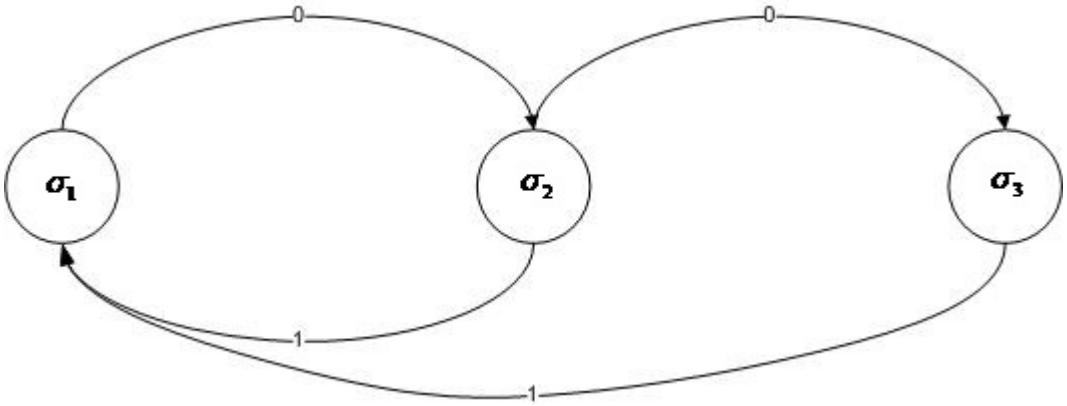

Figure 2.2: (1,2) constrained sequence code

Figure 2.2 is a state transition graph that depicts a $(1, 2)$ CSC. We choose σ_1 to be the initial state. σ_2 and σ_3 represent the states that arise in response to the output '0' and '00' from σ_1, respectively. From this graph, we can quickly find that there are two paths that begin with the initial state σ_1 and return back to it. One is "$\sigma_1 \rightarrow \sigma_2 \rightarrow \sigma_1$", the other one is "$\sigma_1 \rightarrow \sigma_2 \rightarrow \sigma_3 \rightarrow \sigma_1$", with outputs '01' and '001' respectively. Therefore, the minimal set for this constraint is $\{01, 001\}$. It is straightforward to verify that all valid sequences that satisfy the $(1, 2)$ constraint can be constructed through the concatenation

of the sequences in this minimal set.

Partial Extensions

To use nGHC, we require knowledge of the lengths of the codewords that we intend to use in our constrained sequence. With knowledge of these lengths, we can apply the the nGHC construction technique to determine appropriate source word lengths. In this thesis, we use the approach of partial extensions to find valid sets of codeword lengths and the corresponding codewords, and we realize this partial extension algorithm in Java development environment.

First, we set a blank node to be the root node of the tree, and we set this root node to be the current node. We then concatenate all sequences of the minimal set to the codeword sequence represented by the current node to generate the leaves for the current node. This process creates another partial extension and another valid set of codewords. Note that this process can be repeated indefinitely since the sequences of the minimal set satisfy the constraint and can be concatenated arbitrarily without violating the constraint. To limit the complexity of the partial extension process, we set some restrictions on it, which involve limiting the number of codewords Num_{cw} and the maximum length L_{max} of codewords in the partial extensions. Furthermore, Num_{cw} can be used to separate different categories of partial extensions because Num_{cw} is directly reflected to the number of partial extensions for a given minimal set. This relationship can be described as:

$$Num_{cw} = Num_{ms} + n \times (Num_{ms} - 1) \tag{2.21}$$

where Num_{ms} represents the number of words in the minimal set and n represents the number of consecutive partial extensions of this minimal set. For example, the minimal set is $\{01, 001\}$ for the $(1, 2)$ constraint, and the number of words in this minimal set Num_{ms} is 2. In this case, Equation 2.21 can be

simplified into,

$$Num_{cw} = 2 + n \times (2-1) = 2 + n = 2, 3, 4, 5, ... \quad \text{for} \quad n = 0, 1, 2, 3, ... \tag{2.22}$$

Figures 2.3, 2.4 and 2.5 use a tree structure to depict the process of constructing partial extensions. Starting from the blank node, the partial extension with $Num_{cw} = 2$ is $\{01, 001\}$, which is the minimal set of $(1, 2)$ constrained codes. Partial extensions with $Num_{cw} = 3$ include the sets $\{0101, 01001, 001\}$, $\{01, 00101, 001001\}$, and partial extensions with $Num_{cw} = 4$ include the sets $\{010101, 0101001, 01001, 001\}$, $\{0101, 0100101, 01001001, 001\}$, $\{0101, 01001, 001$ $01, 001001\}$, $\{0101, 01001, 00101, 001001\}$, $\{01, 0010101, 00101001, 001001\}$, $\{01,$ $00101, 00100101, 001001001\}$.

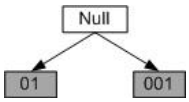

Figure 2.3: Partial Extension with $Num_{cw} = 2$

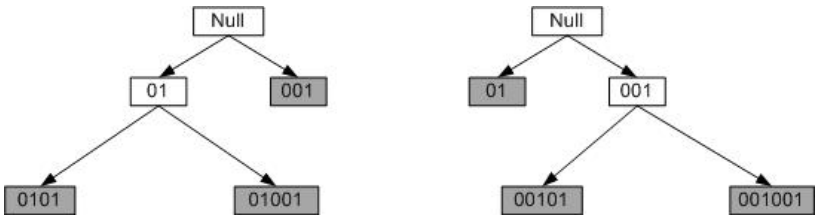

Figure 2.4: Partial Extension with $Num_{cw} = 3$

This approach to constructing partial extensions allows for an exhaustive search of possible codeword sets, up to limits set on Num_{cw} and L_{max}. This follows since we construct a partial extension for each branch node until the limits are reached. Therefore, we will never miss any opportunity for the next category of partial extension. For example, from the codeword set $\{0101, 01001, 001\}$ with $Num_{cw} = 3$, we make extension for each branch node $\{0101\}$, $\{01001\}$ and $\{001\}$ respectively. This generates the possible codeword sets $\{010101, 0101001,$

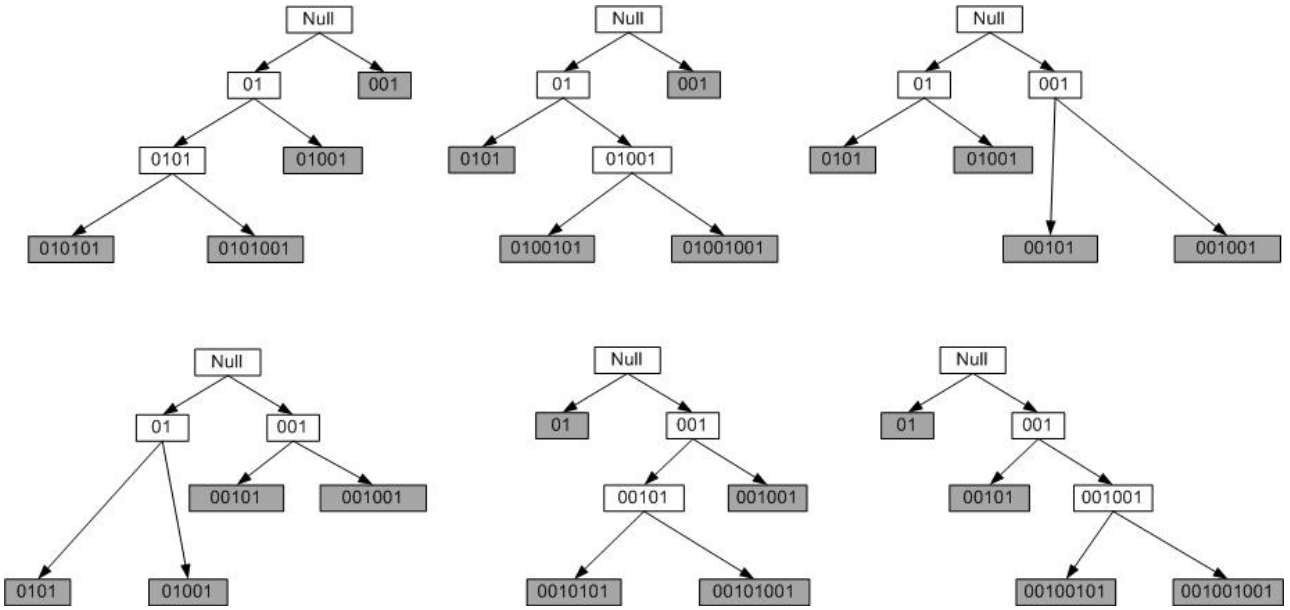

Figure 2.5: Partial Extension with $Num_{cw} = 4$

$01001, 001\}$, $\{0101, 0100101, 01001001, 001\}$ and $\{0101, 01001, 00101, 001001\}$ with $Num_{cw} = 4$, as shown in Fig. 2.5. To verify that our approach is an exhaustive search, we also compare the results of our method and Steadman's method described in [5]. The result listed in Table 2.1 indicates that both approaches achieve the same outcome when setting the same restrictions on the partial extensions.

Constraint	Restrictions of partial extension		Number of partial extensions	If the comparison results are the same?
	longest codeword length	maximum number of codewords		
$(2, 1, \infty)$	12	10	684	same
$(3, 1, \infty)$	12	19	2254	same
$(4, 1, \infty)$	12	28	2991	same
$(5, 1, \infty)$	12	37	3231	same
$(6, 1, \infty)$	12	46	3302	same
$(7, 1, \infty)$	12	55	3322	same
$(8, 1, \infty)$	12	64	3327	same

Table 2.1: Comparison of results for different partial extension methods

Capacity of Constrained Sequence Code

As discussed above, constrained sequence codes are a special case of Markov information sources. Therefore, it is possible to derive the capacity of constrained sequence codes from Equation (2.14). To evaluate the capacity, we first construct a state transition graph that describes the constraint. Here, the constraint may be the (d,k) constraint or any other kind of constraint. We can then derive the connection matrix **D** from the state transition graph, and calculate the capacity of the constrained system as the logarithm of the largest real root of the characteristic equation.

Maxentropic Probability of Constrained Sequence Code

After determining partial extensions and the capacity of the constraint, we apply the nGHC method to generate the source word lengths and a set of source words. nGHC ensures that these source word lengths result in the highest average code rate for that set of codewords [5] [13]; a discussion of this technique follows.

An essential starting point of this method is to determine the desired occurrence probability of codewords, also called the maxentropic probability of the codewords. These probabilities result in the maximum amount of information being conveyed with variable-length codewords. These maxentropic probabilities are [3]:

$$p_i = 2^{-o_i C} = \lambda_{max}^{-o_i} \tag{2.23}$$

where p_i represents the probability of occurrence of the ith codeword and o_i represents its length. For the nGHC procedure, therefore, we require knowledge of the codeword lengths in the partial extension as the values of o_i. The tree structures of Figures 2.3, 2.4 and 2.5, depicted in terms of codeword lengths, are shown in Fig. 2.6. Also, it is necessary for us to verify the unique properties

of different partial extensions when we re-express them in terms of codeword lengths. For example, there are two identical sets of codeword length $\{4, 5, 5, 6\}$ in partial extension with $Num_{cw} = 4$. As discussed above, both of them will generate the same maxentropic probabilities as well as the same result when applying nGHC approach. Thus, we should remove one of them to eliminate this duplication.

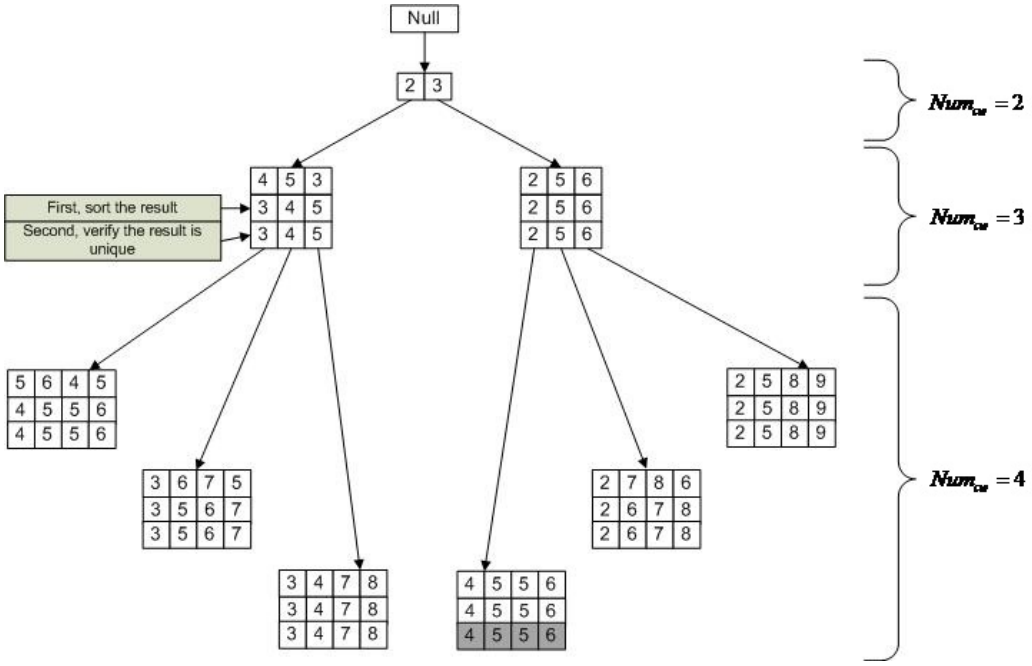

Figure 2.6: Partial Extension in terms of lengths

Geometric Huffman Coding

Channel capacity dictates the highest code rate that can be used while ensuring reliable data transmission. In this thesis, we consider only the noiseless channel. To this point, we have constructed sets of valid codewords and have knowledge of their varying lengths. Our task is then to assign these variable-length codewords to source words such that the codewords will occur in the encoded sequence with probabilities close to the maxentropic probabilities. Throughout this thesis, we assume that binary digits in the source bit sequence are independent and equiprobable. Therefore, source words occur with probability

2^{-l_i} where l_i is the length of the word. With one-to-one mapping of source words to codewords, these are also the probabilities of the corresponding codewords. Our task is then to determine source word lengths and the mapping of source words to codewords such that codeword probabilities are well-matched to Equation (2.21) and that the average code rate given by Equation (2.18) is maximized.

Let the length of the codewords be denoted o_i. Then, Equation (2.18) can be rewritten as follows:

$$R_{avg} = \frac{\sum_i 2^{-l_i} l_i}{\sum_i 2^{-l_i} o_i} \tag{2.24}$$

The maximum code rate is achieved when the input word probabilities are distributed according to the probability mass function (PMF) that maximizes Equation (2.24). To find this PMF, the author of [12] recently introduced an algorithm called geometric Huffman coding (GHC).

The GHC approach is based on a modification of the Huffman coding procedure. As with standard Huffman coding, the least two probabilities are repeatedly merged, forming a tree from leaves to root. The standard Huffman process uses the merging rule $p^{'} = p_m + p_{m-1}$, while GHC uses the merging rule :

$$p^{'} = \begin{cases} p_{m-1}, & \text{if} \quad p_{m-1} \geq 4p_m \\ 2\sqrt{p_m p_{m-1}}, & \text{if} \quad p_{m-1} < 4p_m \end{cases} \tag{2.25}$$

where p_1,p_2,p_3,...,p_m denote the probabilities of the codewords ordered from greatest to least such that $p_1 \geq p_2 \geq ... \geq p_m$, and $p^{'}$ denotes the merged probability. Since merging involves evaluation of a geometric mean, this modified approach is called geometric Huffman coding. An implication of this approach is that a valid output sequence will be discarded if its probability is less than or equal to one quarter the probability of the next likely output sequence. Since merged sequences are assigned source sequences of the same probability, this

prevents the use of unlikely sequences from reducing the overall code rate. It is shown in [12] and [13] that GHC will result in the highest average code rate for a given set of codeword lengths.

Normalized Geometric Huffman Coding

As discussed above, we propose using GHC in our code construction technique. However, the optimal codeword probability $p_i = \lambda_{max}^{-o_i}$ cannot be achieved in practice. Instead, as shown in [12] and [13], the best achievable codeword probabilities have values $\hat{p}_i = 2^{-o_i R_{avg}}$, where R_{avg} is the average codeword rate given in Equation (2.24). It is these codeword probabilities, rather than the probabilities given by (2.21), which are to be used as the starting point for the GHC algorithm. Therefore, using this merging rule requires that we know the code rate in advance, which is not defined until we complete the encoding process.

An iterative method of constructing constrained sequence codes is proposed in [13] to overcome this problem. This iterative approach is known as normalized geometric Huffman coding (nGHC). First, we assume an initial value of code rate equal to capacity, $R_{avg} = C$. Based on this assumption, we calculate the maxentropic probabilities with the equation $p_i = 2^{-o_i R_{avg}} = \lambda_{max}^{-o_i}$. We then use GHC to construct source words with lengths that approach these maxentropic probabilities. Given the source word lengths, we calculate the resulting average code rate R_{avg}. With this new R_{avg} we evaluate the actual codeword probabilities according to $\hat{p}_i = 2^{-o_i R_{avg}}$ and we repeat the GHC process. We continue to repeat the process until the code rate does not change, at which point the procedure has converged to the final result.

By comparing the results of nGHC and standard Huffman Coding, Steadman [11] found that they generate the same codes in almost all cases. But in a very few situations, the effectiveness of nGHC is slightly higher than standard Huffman Coding. This is in agreement with [12] where it is shown that nGHC

is the optimal approach for determining appropriate source word lengths.

Final Code Selection

To this point we have described how to construct source words to correspond to each potential set of codewords by applying the nGHC procedure. From the sets constructed, it remains to simply select the code that has the highest average code rate.

Summary

A summary of the four-step procedure to construct variable-length codes is as follows:

1. Define a minimal set for the constraint;
2. Construct different partial extensions of this minimal set. The number of extensions can be bounded by limiting the depth in the tree structure, or by restricting the maximum length or a maximum number of codewords in the extension;
3. Find the optimal set of source word lengths for each of the partial extensions by using the nGHC approach;
4. Choose the mapping of source words to codewords that results in the highest average code rate.

Chapter 3

Non-binary Communication Systems

Communication systems with more than two symbol values are used in a variety of applications. In this chapter, we consider the application of CS coding to two such systems: multi-level RLL codes for magnetic recording systems and balanced codes for QPSK transmission systems.

3.1 Overview of Multi-level Magnetic Recording Systems

In most magnetic recording systems, data is stored in binary form. Although the applications of binary codes in magnetic recording systems have been successful in the past, considerable effort is now being dedicated towards research on multi-level magnetic recording systems, specifically, the hard disks that are still one of the primary storage mechanism of choice for cloud storage. The reason for this initiative is that the density and transfer rate of information is increased significantly with multi-level magnetic recording. For example, assuming eight recording levels are allowed in a multi-level magnetic recording

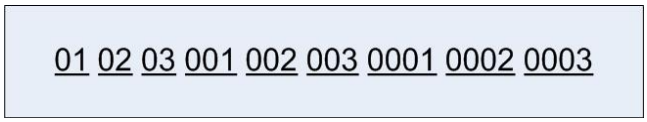

Figure 3.1: Permitted sequences in the $(4, 1, 3)$ constrained code

system, and then 3 bits/stored-symbol are conveyed rather than 1 bit/symbol as in a binary magnetic recording system. In such a system, a multi-level recording code would be employed to help the system approach the capabilities of the media.

3.2 RLL Codes Applied in Multi-level Magnetic Recording Systems

Multi-level runlength limited (ML-RLL) codes are obtained through change-of-state encoding of M-ary (d, k) constrained codes, which are extensions of binary (d, k) constrained codes and denoted as (M, d, k) codes. As mentioned in Chapter 2, binary (d, k) constrained codes comply with the requirement that the number of zeros is at least d and at most k between consecutive ones. M-ary (d, k) constrained codes follow a similar requirement, the only difference being that in M-ary codes we consider the number of consecutive zeros between any two non-zero symbols rather than just those between logic ones. For example, for the 4-ary symbol alphabet $A = \{0, 1, ..., 3\}$, a sequence that satisfies the $(4, 1, 3)$ constraint is shown in Fig. 3.1, where we have underlined the suitable sub-sequences.

For M-ary (d, k) constrained codes, M $(M \geqslant 2)$ represents the number of different symbol values used in the recording system. Note that $M = 2$ denotes the special case of binary (d, k) constrained codes. We also use the equation $\eta = \frac{R_{avg}}{C}$ to evaluate the efficiency of M-ary (d, k) constrained codes, where R_{avg} is the average code rate and C is the channel capacity.

Several fixed-length block coded M-ary (d, k) constrained codes have been

developed. For instance, McLaughlin proposed five different M-ary (d, k) constrained codes constructed using the state-splitting algorithm [14] and [15]. Kumar and Immink proposed a new method to construct close-to-capacity M-ary (d, k) constrained codes [16]. Their method yields efficiencies over 95%, with simple encoder and decoder structures for selected constraints. We provide a brief introduction of their codes in next subsection.

3.2.1 Properties of M-ary (d, k) Constrained Codes

Before we introduce Kumar and Immink's M-ary (d, k) constrained codes, we state two fundamental properties of M-ary (d, k) constrained codes: their capacity, and the counting of M-ary (d, k) constrained sequences. In [16], Kumar presented techniques to determine both of these properties.

Capacity of M-ary (d, k) Constrained Codes

Like binary (d, k) constrained codes, M-ary (d, k) constrained codes can be modeled as Markov information sources. As a result, we can use their characteritic equations to calculate their capacity. French and Wolf first outlined this idea in [17]. Kumar improved and developed a closed form for the characteristic equations for two specific cases [16]: (M, d, ∞) codes and M-ary codes with $d < k < \infty$. The characteristic equation for (M, d, ∞) codes is:

$$z^{d+1} - z^d - (M - 1) = 0 \tag{3.1}$$

The characteristic equation for (M, d, k) codes with $d < k < \infty$ is:

$$z^{k+2} - z^{k+1} - (M - 1)\, z^{k-d+1} + M - 1 = 0 \tag{3.2}$$

The capacity C for either case is obtained as:

$$C = \log_2 \lambda_{max} \tag{3.3}$$

where λ_{max}, as defined in Chapter 2, is the largest real root of the characteristic equation.

Table 3.1 reports Kumar's results [16] for capacity when d is fixed to 1, M ranges from 3 to 7, and k ranges from $d+1$ to ∞. From this table, it is clear that capacity increases as M and k increase. The reason for increasing capacity as M increases is that larger values of M result in more possible sequences and, therefore, a greater information carrying ability with the same number of symbols.

M/k	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10	k= ∞
M=2	0.40569	0.55146	0.61745	0.65090	0.66903	0.67929	0.68525	0.68879	0.69091	0.69424
M=3	0.82317	0.92548	0.96606	0.98390	0.99219	0.99616	0.99810	0.99905	0.99953	1.00000
M=4	1.07300	1.15423	1.18347	1.19503	1.19982	1.20184	1.20271	1.20309	1.20325	1.20337
M=5	1.25276	1.32099	1.34371	1.35196	1.35507	1.35626	1.35672	1.35690	1.35697	1.35702
M=6	1.39362	1.45284	1.47132	1.47755	1.47973	1.48050	1.48078	1.48088	1.48091	1.48093
M=7	1.50961	1.56216	1.57765	1.58256	1.58417	1.58470	1.58487	1.58493	1.58495	1.58496
M=8	1.60830	1.65567	1.66895	1.67294	1.67417	1.67455	1.67467	1.67471	1.67472	1.67472

Table 3.1: Capacities for (M, d, k) codes with d =1 generated by Kumar [16]

Counting of M-ary (d, k) Constrained Sequences

Immink proposed an iterative method to count binary constrained sequences [3]. Kumar extended Immink's recursive relations to M-ary (d, ∞) constrained codes. His recursive relations are:

$$
\begin{aligned}
&N_d(n; M) = 0, \quad for \quad n < 0 \\
&N_d(0; M) = 1, \\
&N_d(n; M) = n(M-1) + 1, \quad 1 \le n \le d+1 \\
&N_d(n; M) = N_d(n-1; M) + (M-1) N_d(n-d-1; M), \quad \text{for} \quad n > d+1
\end{aligned}
\tag{3.4}
$$

where N_d and n represent, respectively, the number and length of fixed-length sequences that satisfy the constraints.

With the above relations, Kumar presented a method to construct fixed-length M-ary (d, k) constrained codes. Firstly, it is necessary to determine

a rational code rate $R = \frac{m}{n}$ as close to capacity as possible. Then, with the knowledge of the desired code rate, he proposed a technique to design an M-ary (d, k) constrained code with high efficiency. The process will be demonstrated in detail in the next subsection.

3.2.2 Previously Published Fixed-length M-ary (d, k) Constrained Codes

The code construction method proposed by Kumar and Immink begins with the evaluation of integers m and n. According to Kumar, the evaluation process can be divided into several steps:

1. With the knowledge of the values of d and M, we can find the corresponding relationship between n and N_d for $k = \infty$ constrained codes by applying Equation (3.4).

2. Then, we can determine the value of m, which must meet the following condition [3]:

$$N_d \geq 2^m \tag{3.5}$$

3. Under this condition, there are multiple values of m that can be chosen. We select the one that results in a code rate $R = \frac{m}{n}$ as close to capacity as possible.

For example, possible values of m and n for the $(4, 1, \infty)$ case are listed in Table 3.2. Note that all of these values of m and n will result in a code rate R within 7% of capacity.

Construction of d=1 M-ary (d, k) Constrained Codes

After determining appropriate values of m and n, Kumar then demonstrated a necessary condition for the design of a finite state encoder. This encoder

m	n	Code Rate R	(1-η)%
6	5	1.20000	0.28
7	6	1.16667	2.7775
8	7	1.14286	5.028
9	8	1.12500	6.5125
13	11	1.18182	1.79
19	16	1.18750	1.318
25	21	1.19048	1.071

Table 3.2: Integers m and n for (M=4,d=1,k=∞) codes

can be used to generate M-ary $(1, \infty)$ constrained codes. His design of a finite state encoder is depicted in Fig. 3.2. This figure implies separation of all possible codewords into four subsets E_{00}, E_{0i}, E_{i0} and E_{ij}, where $i, j \in \{1, 2, ..., M-1\}$. E_{00} is defined as the subset of possible codewords that begin and end with a zero, while E_{0i} is defined as the subset of possible codewords that begin with a zero and end with a non-zero symbol i. Subsets E_{i0} and E_{ij} are defined in a similar manner. The arrows between different subsets indicate permitted concatenation of corresponding codewords. For example, the arrows from subset E_{ij} to subset E_{00} and subset E_{0i} indicate that a codeword from subset E_{ij} must be followed by a codeword from either subset E_{00} or E_{0i} so that the $d = 1$ constraint is maintained. The subsets E_{00} and E_{i0} are shown without arrows; this should be taken as an indication that their codewords can be concatenated with any codewords from any other subset.

With knowledge of the concatenation rules of different codeword subsets, Kumar designed a practical encoder through appropriate definition and analysis of the encoder states. He assumes that the encoder includes r states and divides all of these states into two subsets B_1 and B_2. There are r_1 states in subset B_1 in which all codewords must start with a zero. Subset B_2 involves $r_2 = r - r_1$ encoder states in which all codewords are free to start with a zero or a non-zero symbol.

With these definitions, the state-transition rules of the encoder can be easily summarized. Codewords ending with zero may cause the encoder to enter any

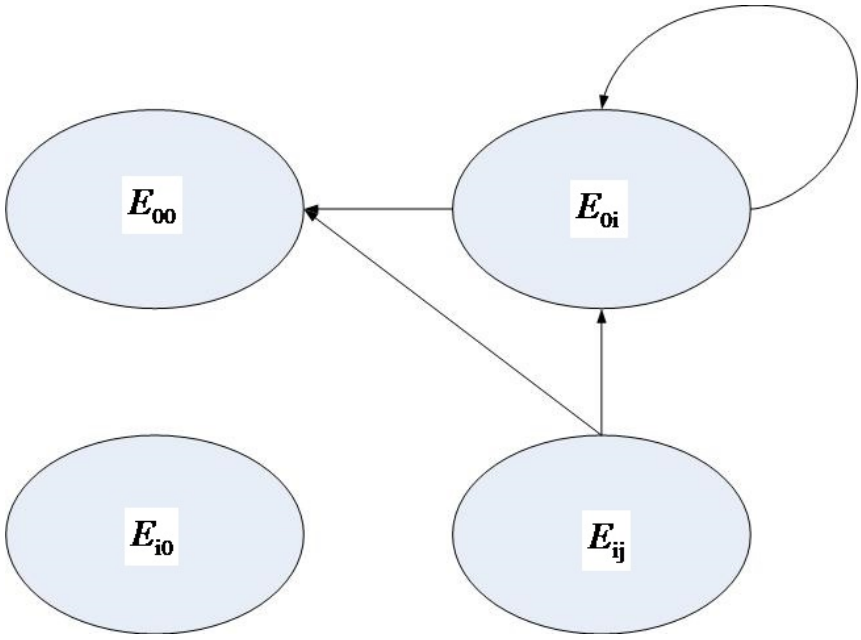

Figure 3.2: Finite state encoder of M-ary $(1, \infty)$ constrained codes

of $r = r_1 + r_2$ states; these are codewords belonging to E_{00} and E_{i0}. Conversely, the codewords ending with a non-zero symbol may cause the encoder to only enter the r_1 states; these states result in the generation of codewords belonging to E_{0i} and E_{ij}. The reason that codewords ending with a non-zero may not result in the encoder moving into any of the r_2 states is that the codewords generated from the r_2 states may start with a non-zero symbol, and when a codeword ending with a non-zero symbol causes the encoder to enter that kind of state, the concatenation rule is violated. Fig. 3.3 clarifies this concept.

In [18], Immink outlines an essential condition for the above approach: the codewords within each encoder state should be unique. He then demonstrates that it is possible for a codeword to correspond to multiple source words since a sliding-block decoder can identify the particular codeword to source word mapping by observing both the current and next state. For example, if the current codeword belongs to the subsets E_{00} and E_{i0}, then it can be followed by the codeword generated from any of $r = r_1 + r_2$ states. This means the codeword itself can be assigned $r = r_1 + r_2$ times to distinct source words.

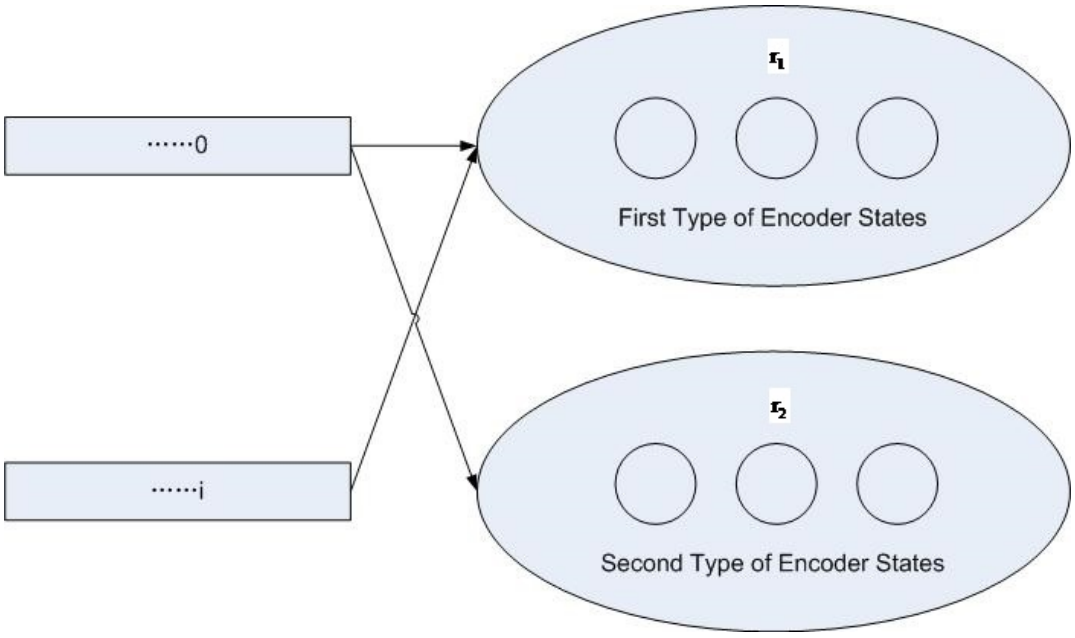

Figure 3.3: Kumar and Immink's fundamental concept concerning encoder states

Similarly, the codewords in subsets E_{0i} and E_{ij} can only be followed by the codewords generated from r_1 states. Thus, this codeword can only be assigned r_1 times to distinct source words.

After developing this model for the encoding and decoding processes, Kumar derived the following inequalities for the values of r_1 and r_2:

$$r\left|E_{00}\right| + r_1 \sum_{i=1}^{M-1} \left|E_{0i}\right| \geq r_1 2^m \tag{3.6}$$

$$r\left(\left|E_{00}\right| + \sum_{i=1}^{M-1} \left|E_{i0}\right|\right) + r_1 \left(\sum_{i=1}^{M-1} \left|E_{0i}\right| + \sum_{i=1}^{M-1} \sum_{j=1}^{M-1} \left|E_{ij}\right|\right) \geq r2^m \tag{3.7}$$

where $|E_{00}|$, $|E_{0i}|$, $|E_{i0}|$ and $|E_{ij}|$ denote the cardinality of the subsets E_{00}, E_{0i}, E_{i0} and E_{ij} accordingly. Equation 3.6 indicates that the maximum number $\left[r\left|E_{00}\right| + r_1 \sum_{i=1}^{M-1} \left|E_{0i}\right|\right]$ of codewords leaving from r_1 states of the first type should be at least equal to $r_1 2^m$, the number of source words entering that

r_1	r_2	r
3	4	7
6	8	14
7	9	16
8	11	19
9	12	21
10	13	23
11	14	25

Table 3.3: Values of r_1 and r that satisfy the $(4, 1, \infty)$ constraint with $R = \frac{6}{5}$

kind of state. Equation 3.7 follows from the similar fact that there should be sufficient codewords leaving from r states to match the corresponding source words entering those states.

Kumar demonstrates that it is possible to evaluate values for r_1 and r by applying Equations 3.6 and 3.7 for selected values of m and n. Thus, the size and the structure of an encoder that follows the $(d = 1)$ constraint can be determined. Constructions of two specific $(d = 1)$ constrained codes are introduced below as examples to demonstrate the process in detail.

Rate 6/5 (4,1,11) Code

As shown in Table 3.1, the capacity of the $(4, 1, \infty)$ constraint is 1.20037. With knowledge of this capacity, Kumar searched for appropriate values of m and n, and found the values listed in Table 3.2. In order to limit the complexity of the encoder, Kumar restricted his search by limiting the maximum value of m to 25. As shown in Table 3.2, the highest code rate $R = \frac{m}{n} = 1.2$ is achieved when $m = 6$ and $n = 5$. For the $(4, 1, \infty)$ constrained code, Equation 3.4 can then be used to calculate the cardinalities $|E_{00}| = |E_{0i}| = |E_{i0}| = 19$ and $|E_{ij}| = 36$. By substituting these values into Equation 3.6 and 3.7, Kumar obtained the possible values of r, r_1 and r_2 listed in Table 3.3.

For the sake of simplicity, Kumar selected the least number of states ($r_1 = 3$, $r_2 = 4, r = 7$) to demonstrate how to design an encoder. Detailed analysis is not discussed here, however, his final result is given in Table 3.4. By eliminat-

	Number of codewords for each state						
	States of R_1			States of R_2			
Codeword subsets	**1**	**2**	**3**	**4**	**5**	**6**	**7**
E_{00}	5	7	7	0	0	0	0
E_{01}	3	2	2	0	0	0	0
E_{02}	3	2	2	0	0	0	0
E_{03}	4	1	1	1	0	0	0
E_{10}	0	0	0	3	1	2	1
E_{20}	0	0	0	3	2	1	1
E_{30}	0	0	0	3	1	1	2
E_{11}	0	0	0	0	1	1	2
E_{12}	0	0	0	0	1	1	2
E_{13}	0	0	0	0	1	1	2
E_{21}	0	0	0	0	2	1	1
E_{22}	0	0	0	0	1	1	2
E_{23}	0	0	0	0	2	1	1
E_{31}	0	0	0	0	1	1	2
E_{32}	0	0	0	0	1	3	0
E_{33}	0	0	0	0	2	2	0

Table 3.4: Distribution of codeword subsets to encoder states for a $(4, 1, 11)$ $R = \frac{6}{5}$ code

ing the all-zero sequence, Kumar restricted the longest runlength to $k = 11$. Table 3.4 therefore describes the encoder structure for a $(4, 1, 11)$ code,

Rate 10/6 (8,1,11) code

As a second example, consider the design of a fixed-length $(8, 1, 11)$ code. The capacity of the $(8, 1, 11)$ constraint is 1.67472, as shown Table 3.1. Kumar found appropriate values of m and n which allow a code rate close to this capacity; these values are listed in Table 3.5. Kumar choose $m = 10$ and $n = 6$ instead of $m = 5$ and $n = 3$ as a basis to design the encoder since the longer words provide additional sequences and therefore additional flexibility in the code design. In this case, with assistance from Equation 3.4, the cardinalities can be evaluated as $|E_{00}| = 176$, $|E_{0i}| = |E_{i0}| = 71$, and $|E_{ij}| = 15$. Kumar then substituted these values into Equations 3.6 and 3.7 and computed the values

m	n	Code rate r	$(1-\eta\%)$
5	3	1.66667	0.481
10	6	1.66667	0.481
8	5	1.6	4.461
11	7	1.57143	6.1675
14	9	1.55556	7.115

Table 3.5: Integers m and n for the $(8, 1, 11)$ constraint

r_1	r_2	r
1	2	3
2	4	6
2	5	7
3	7	10
4	8	12

Table 3.6: Values of r_1 and r that satisfy the constraint $(8, 1, 11)$ when $R = \frac{10}{6}$

of r_1 and r shown in Table 3.6. From these options, the values for the least number of states is selected. The assignment of codewords from each subset to the encoder states is then performed. By eliminating the all-zero sequence, Kumar restricted the parameter k to 11 and generated the rate $R = \frac{10}{6}$ $(8, 1, 11)$ constrained code whose parameters are shown in Table 3.7.

3.3 Overview of QPSK Transmission

Phase-shift keying (PSK) is a modulation technology used in bandpass digital communication systems which represents user data with distinct phases of the carrier. Equal phase shifts are usually chosen to result in a signalling con-

	Number of codewords for each state		
	States in r_1	States in r_2	
Codeword subsets	**1**	**2**	**3**
E_{00}	176	0	0
E_{0i}	497	0	0
E_{i0}	0	371	126
E_{ij}	0	0	735

Table 3.7: Distribution of codeword subsets to encoder states for an $(8, 1, 11)$ $R = \frac{10}{6}$ constrained code

stellation with uniformly distributed points on a circle centered at the origin of the complex plane. The reason for this uniform spacing is that it results in the maximum phase separation between points which ensures the greatest immunity to channel interference. Since all signalling points have the same amplitude, all symbols are transmitted with the same energy.

Two common PSK systems are binary phase-shift keying (BPSK) which uses two phases, and quadrature phase-shift keying (QPSK) which uses four phases. When compared to BPSK, QPSK can double the data rate while maintaining the same bandwidth. A drawback of QPSK is that the design of transmitter and receiver circuitry is somewhat more complicated than that for BPSK.

3.4 DC-free Codes for QPSK Transmission

3.4.1 Introduction of DC-free QPSK Codes

Recall that in a binary system, when the encoded sequence is balanced, the spectrum of the encoded signal contains a null at DC. For this reason, balanced binary codes are also called DC-free codes. An analogous situation holds in QPSK systems. When the encoded system sequence is balanced, the spectrum of the equivalent complex baseband signal contains a null at DC. This translates to a null at the carrier frequency of the transmitted bandpass signal. For this reason, balanced codes have been recently proposed for bandpass QPSK systems [19]. With this null in the spectrum, it is possible to filter out unwanted interference and to insert and remove a pilot tone without reducing the power of the information-carrying portion of the signal.

In this section, we review two approaches that have been published for constructing DC-free QPSK codes. The first one is a tabular construction approach proposed by Jamieson [19]. The second approach involves the application of guided scrambling to construct DC-free QPSK codes, as outlined by Fair and

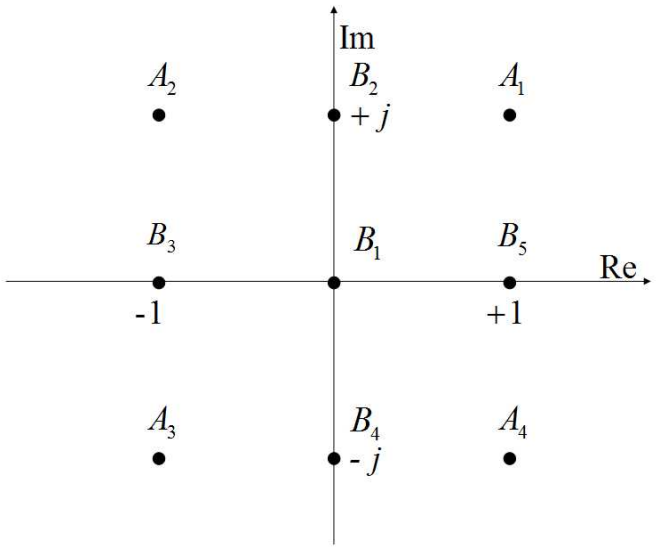

Figure 3.4: RDS values and encoding states in a rate 4/3-balanced QPSK code Martin [20].

3.4.2 Previously Published Fixed-length DC-free QPSK Codes

Tabular Construction of Balanced Codes

A technique for constructing state-independent decodable codes [21] [22] has been reported recently. This algorithm describes the construction of coding tables for the general case of constrained sequence codes. Jamieson extended this general technique by considering symmetry within QPSK codes and proposed a tabular approach to construct DC-free QPSK codes [19].

In order to maintain the DC-free property, the running digital sum (RDS) of the encoded QPSK symbol sequence must be bounded. Since QPSK signalling points take on complex values, so does the RDS of the QPSK symbol sequence. The technique outlined in [19] restricts the RDS of the encoded QPSK sequence to the nine values depicted in Fig. 3.4. Each of these values is equivalent

to an independent state. These states are then separated into three subsets: $\{A_1, A_2, A_3, A_4\}$, $\{B_2, B_3, B_4, B_5\}$ and $\{B_1\}$. For the subsets $\{A_1, A_2, A_3, A_4\}$ and $\{B_2, B_3, B_4, B_5\}$, there is a 90 degree symmetry between each member state. This implies that if a codeword can be emitted from a state, then symmetrical codewords can be emitted from other states within that subset. For example, since the codeword $a_1 = \{-j, -j, -1\}$ can be emitted from state A_1 without violating the RDS constraints, then codewords $a_2 = \{+1, +1, -j\}$, $a_3 = \{+j, +j, +1\}$ and $a_4 = \{-1, -1, +j\}$ can be emitted from states A_2, A_3 and A_4 respectively. However, subset $\{B_1\}$ is self-symmetrical, which means that if a codeword can be emitted from that state, then all three of its symmetrical codewords can also be emitted from that state.

By taking advantage of this symmetry, Jamieson simplified the process of codeword enumeration and table construction. He also reported the simple $R = \frac{4}{3}$ balanced QPSK code that is listed in Table 3.8 and Table 3.9. Also, note that Table 3.8 can be expanded to 16 rows by assigning symmetrical codewords to symmetrical states, and that Table 3.9 lists only one codeword from each subset of four symmetrical codewords. In the design of these coding tables, Jamieson noted that there are extra codewords for some states, which indicates that codeword priorities are necessary to select the appropriate codewords. When considering the construction of his code, he chose codewords that resulted in the greatest suppression of low-frequency components.

As outlined in [3] and [23], the power spectral density (PSD) of a block coded signal can be expressed in the form shown in Equation 3.8:

$$H_x(w) = H_p(\omega)\left(H_{xc}(\omega) + H_{xd}(\omega)\sum_{k=-\infty}^{\infty} 2\pi\delta(\omega - 2\pi k/n)\right) \tag{3.8}$$

where $H_p(\omega) = |S(\omega)|^2$ and $S(\omega)$ is the Fourier transform of the pulse shape, and where $H_{xc}(\omega)$ and $H_{xd}(\omega)$ represent the effect of coding on the continuous

and discrete components of the spectrum respectively. Our interest is the influence of the coding procedures on the continuous component of the spectrum, therefore power spectral density curves shown in this thesis represent only the component $PSD = H_{xc}(\omega)$.

It was also shown in [24] that the PSD close to DC of balanced codes can be accurately approximated as:

$$\begin{aligned} &PSD \approx L\omega^2, \quad \omega \ll 1 \\ &PSD\,[dB] \approx 10\log L + 20\log \omega, \quad \omega \ll 1 \end{aligned} \tag{3.9}$$

where L is the low frequency spectral weight (LSFW) of the code. As reported in [19], this rate $R = \frac{4}{3}$ DC-free QPSK code has LFSW $L = 1.9245$.

State / Row	A_1	A_2	A_3	A_4	B_2	B_3	B_4	B_5	B_1
D_1	a_1	b_1	b_1	c_1	a_1	b_1	c_1	c_1	b_1
E_1	d_1	e_1	f_1	d_1	e_1	f_1	f_1	d_1	f_1
F_1	g_1	h_1	i_1	j_1	h_1	h_1	j_1	g_1	h_1
G_1	k_1	l_1	m_1	n_1	k_1	l_1	n_1	n_1	o_1

Table 3.8: State/codeword assignments in rate 4/3-balanced QPSK code

Codewords					
a_1	$-j,-j,-1$	f_1	$+1,-1,+j$	k_1	$-j,-1,-j$
b_1	$+1,-1,+1$	g_1	$-1,-j,+j$	l_1	$+1,-j,+j$
c_1	$-1,+j,+1$	h_1	$+1,-1,-j$	m_1	$+j,+1,+1$
d_1	$-1,-1,+1$	i_1	$+1,+j,+j$	n_1	$+j,-1,-j$
a_1	$-j,+1,-j$	j_1	$+j,+j,-1$	o_1	$+1,-1,-1$

Table 3.9: Codewords in rate 4/3-balanced QPSK code

Guided Scrambling Construction of Balanced Codes

In [20], Martin proposed another method to construct balanced QPSK codes. His method is based on guided scrambling (GS) which was originally developed

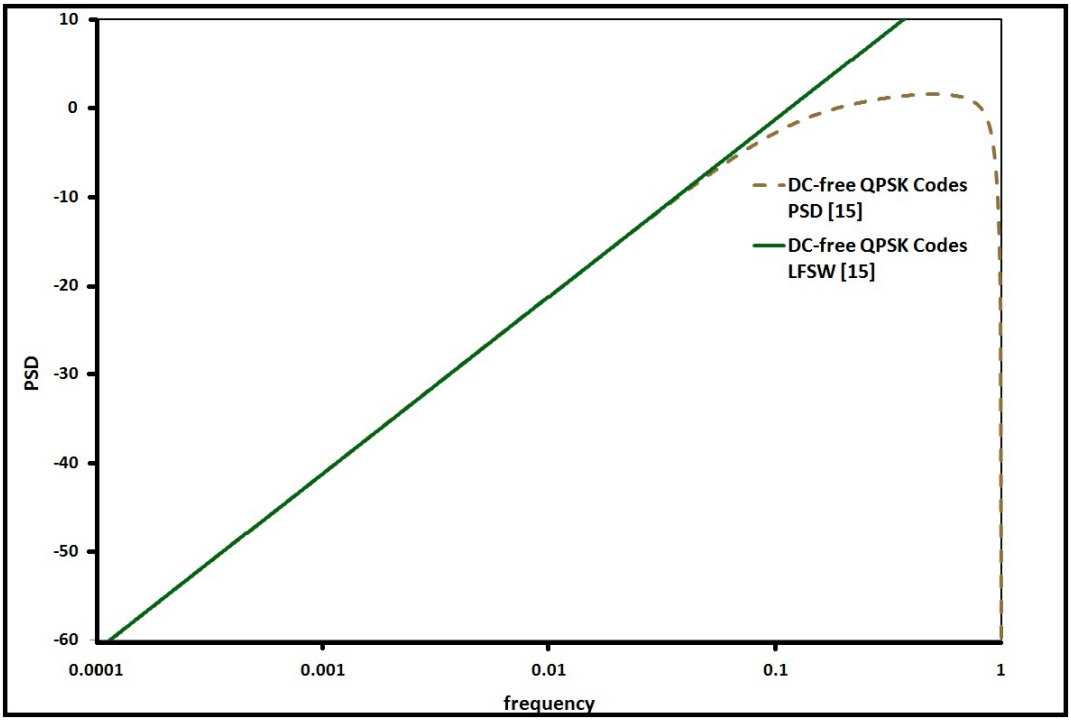

Figure 3.5: Power spectral density of rate 4/3 QPSK codes published in [19]

for binary systems [25] [26] [27]. Because of its simple encoding and decoding procedures, Martin applied this technique to the construction of DC-free QPSK codes [20].

As described in [20], guided scrambling encoding of balanced QPSK codes can be summarized in several steps. First, each length-m source word is augmented with all patterns of A augmenting symbols to create 4^A different augmented words of length $n = m + A$. Then, 4^A different length-n quotients are formed by multiplying each of the augmented words by x^D and dividing it by $d(x)$, where $d(x)$ is a preselected scrambling polynomial of degree D. All arithmetic is within the ring of polynomials defined over the finite field of four elements, GF(4). The element-wise GF(4) operations of addition and multiplication are given in Table 3.10. Lastly, the quotient with the best properties is selected as the length-n codeword. At the receiver, the source words are recovered by multiplying the received symbol sequence by the same scrambling polynomial $d(x)$ and eliminating the A augmenting symbols.

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

x	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

Table 3.10: GF(4) arithmetic

A \ Redundancy	1 in 4		1 in 8		1 in 16		1 in 32		1 in 64		1 in 128	
	n	L	n	L	n	L	n	L	n	L	n	L
1	4	10.4	8	72.7	16	361	32	1559	64	6202	128	21703
2	8	7.13	16	45.6	32	227	64	995	128	3962	256	14336
3	12	5.55	24	30.8	48	148	96	652	192	2635	384	9845
4	16	4.79	32	22.3	64	105	128	459	356	1873	512	7733
5	20	4.20	40	18.0	80	80.0	160	345	320	1435	640	5631
6	24	3.96	48	15.6	96	65.6	192	281	384	1179	768	4660

Table 3.11: LSFW of GS QPSK codes [20]

Using the above method, Martin constructed guided scrambling balanced QPSK codes with different word lengths and code rates. Using the scrambling polynomial $d(x) = x^A + 1$, he obtained the values of LFSW given in Table 3.10 and evaluated the spectral results that are depicted in Fig. 3.6 and 3.7. Note that these spectra contain a null at DC in the spectrum of the equivalent complex baseband signal. This null corresponds to a notch at the center frequency of the bandpass system and confirms the balanced nature of the GS QPSK coded signals.

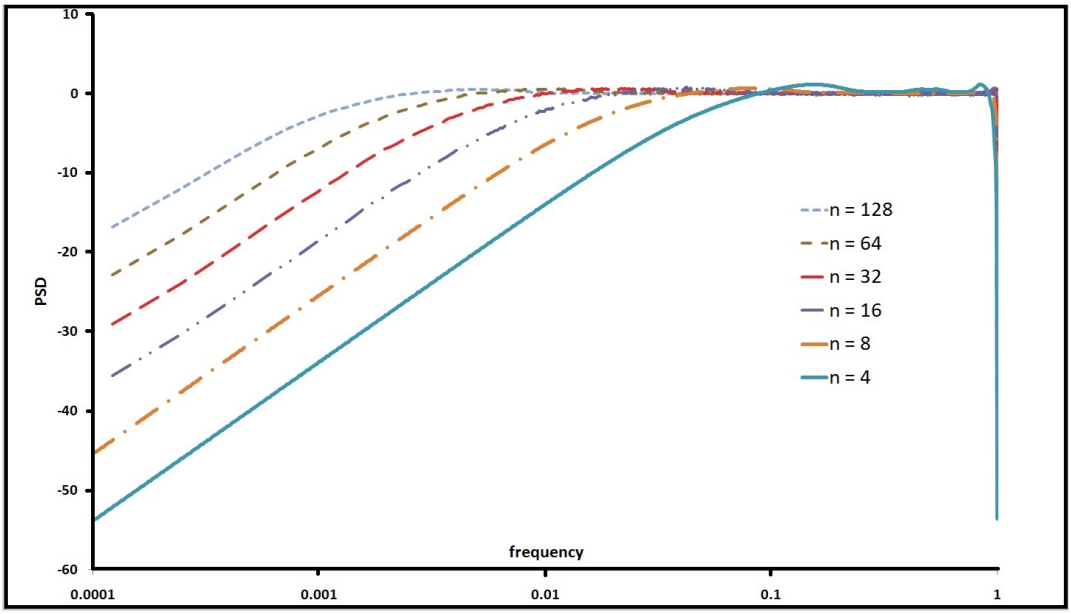

Figure 3.6: Spectra of GS QPSK codes, $A = 1$ and various codeword lengths N [20]

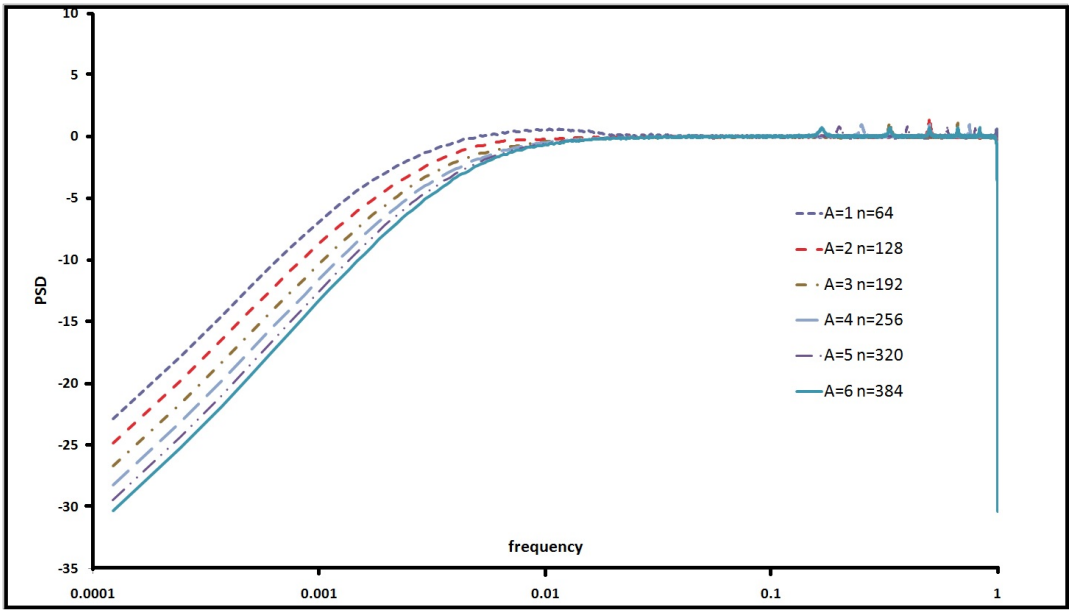

Figure 3.7: Spectra of GS QPSK codes with an average of one redundant symbol in 64 coded symbols [20]

Chapter 4

Variable-length RLL Codes for Multi-level Magnetic Recording Systems

In this chapter, we use Steadman's technique [5] [7] to construct variable-length multi-level runlength limited (ML-RLL) codes for multi-level magnetic recording systems. To demonstrate this approach clearly, we construct two codes for direct comparison with those developed by Kumar [12]. We then present our results for a large number of codes for various values of M and k, for d=1 and d=2. To the best of our knowledge, these new variable-length codes have a higher average code rate and efficiency than any other codes developed to date for these constraints.

4.1 Example of Code Construction : $(4, 1, \infty)$ Code

As described in Chapter 2, the construction of variable-length codes involves the following four steps:

1. Define a minimal set for the constraint.

2. Construct different partial extensions of this minimal set. The number of extensions can be bounded by limiting the depth in the tree structure, or by restricting the maximum length or a maximum number of codewords in the extension.

3. Find the optimal set of source word lengths for each of the partial extensions by using the nGHC approach.

4. Choose the mapping of source words to codewords that result in the highest average code rate.

We now describe these steps in detail for a $(4, 1, \infty)$ code.

4.1.1 Minimal Set

To apply Steadman's technique for construction of variable-length CS codes, we need first to find the minimal set that represents the constraint. As discussed in Chapter 2, a minimal set can be established through observation of the transition graph. For example, consider the binary $(1, \infty)$ constrained codes which can be represented by the transition diagram depicted Fig 4.1.

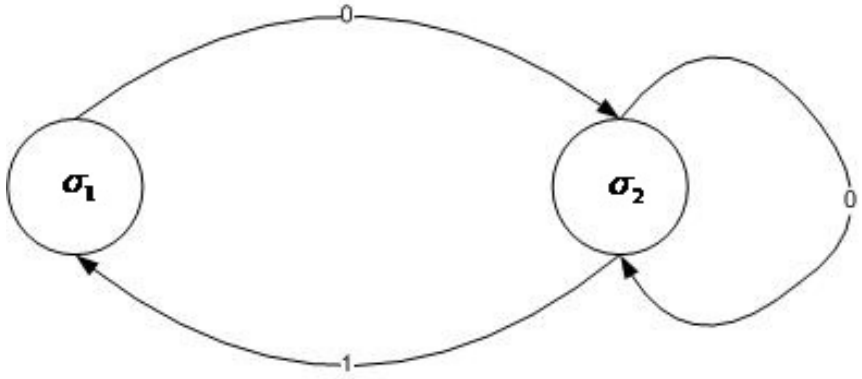

Figure 4.1: Transition graph used to derive the minimal set of binary $(1, \infty)$ codes

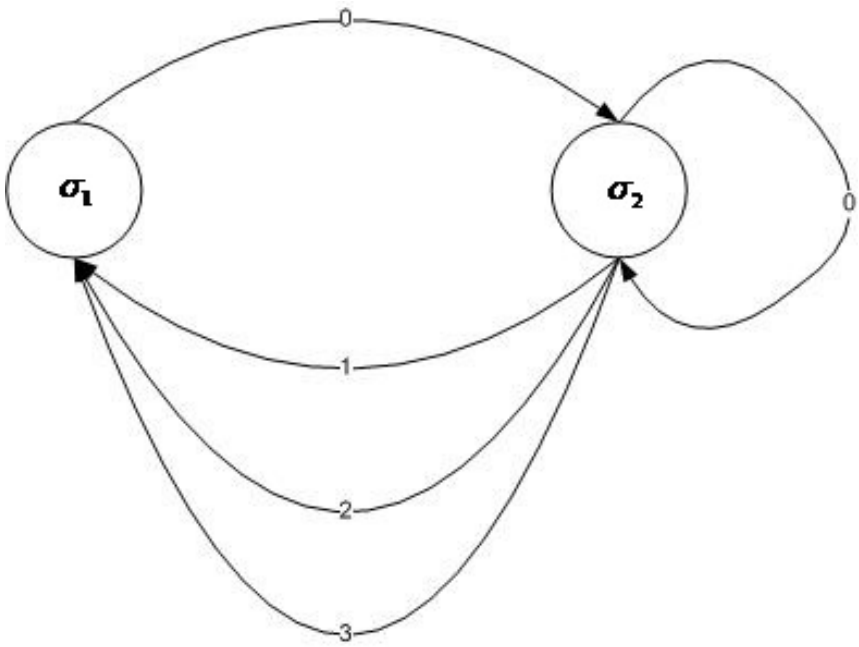

Figure 4.2: Transition graph for $(4, 1, \infty)$ constraint

In this figure, state σ_1 represents the fact that the most recent bit in the encoded sequence is a 1, and state σ_2 denotes the fact that the most recent bit in the encoded sequence is a 0. The oriented edges between these two states indicate the encoded output that arises as a result of transitions between the states. Note that at least one zero must occur between logic ones, and that the number of zeros between ones is unlimited.

To define a minimal set, we select one state as the initial state and list all sequences that can arise as this state is exited and re-entered. When σ_2 is chosen as the initial state, it is straightforward to confirm that the minimal set is $\{0, 10\}$.

We now extend the binary case to the M-ary case for the $(M, 1, \infty)$ constraint. The only difference is that since the number of different signalling levels is now M, there are $M - 1$ oriented edges from state σ_2 to state σ_1. For example, the directed graph for the $(4, 1, \infty)$ constraint is depicted in Figure 4.2. Following the procedure outlined above, it is straightforward to verify that the minimal set for the $(4, 1, \infty)$ constraint is $\{0, 10, 20, 30\}$.

4.1.2 Partial Extensions

With knowledge of the minimal set, we can construct its partial extensions to determine possible sets of codewords. Details of partial extensions were discussed in Section 2.3.1. In this section, we focus on how to construct partial extensions and establish possible limits for construction of these partial extensions for the $(4, 1, \infty)$ constraint.

Because the lengths of codewords are what is required to apply the nGHC method, we consider the lengths of words in the minimal set when establishing the first sublayer in the tree of partial extensions. As indicated in Fig 4.2, these lengths are $\{1, 2, 2, 2\}$. Then for each unique node extending from this sublayer, we sum the lengths of words in the minimal set to a word length in the set of lengths that represent the node to form a set of new lengths. This set of new lengths along with other lengths in the sublayer together constitute the set of lengths representing a new node in the corresponding partial extension.

Taking the first extension in Figure 4.3 for example, we consider the word length $\{1\}$ within the set of word lengths $\{1, 2, 2, 2\}$ as the current node. By adding the length $\{1\}$ with lengths of minimal set $\{1, 2, 2, 2\}$, we generate a set of new lengths $\{2, 3, 3, 3\}$. These new lengths, along with the lengths of the other remaining words, $\{2, 2, 2\}$, together constitute the lengths of corresponding partial extension, which is $\{2, 3, 3, 3, 2, 2, 2\}$.

During the process of partial extension, there are two essential observations. The first one is that each unique value in the same sublayer will only be extended once. For example, for the sublayer $\{1, 2, 2, 2\}$, only one length $\{2\}$ will be extended. The second one is that duplicate sets of lengths must be removed in the extension process. For example, there are two identical sets of lengths $\{2, 2, 2, 3, 3, 3, 3, 4, 4, 4\}$ in Figure 4.3; one of them should be removed. The reason to do so is because extensions from both of these sets of lengths will generate the same result.

Because unrestricted concatenation of words from the minimal set always

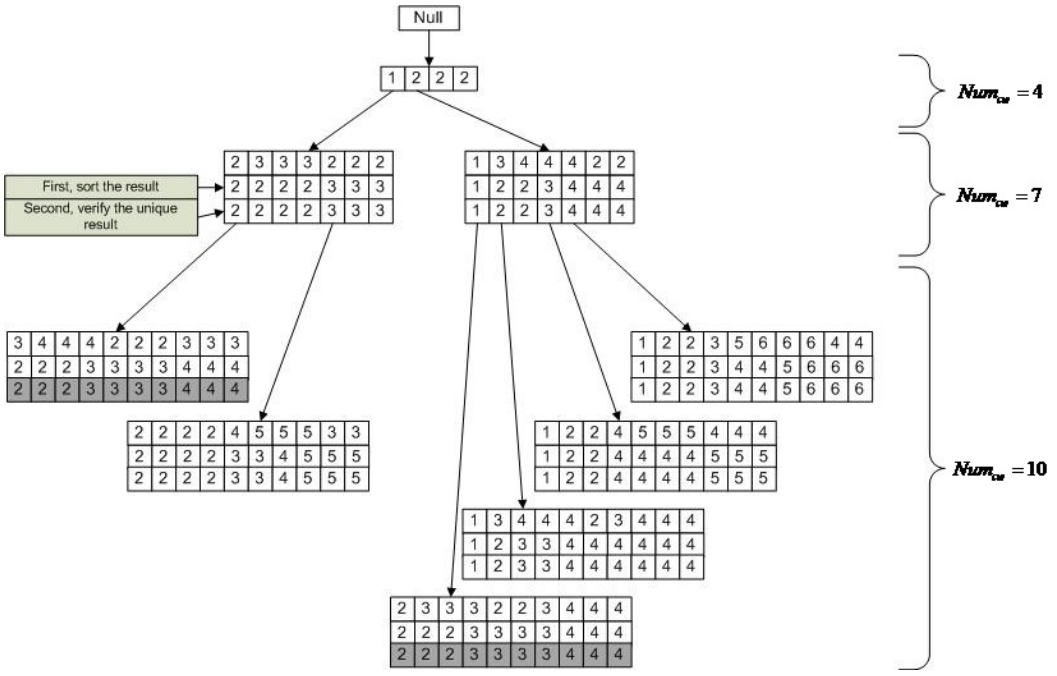

Figure 4.3: Unique word lengths of the partial extensions shown in Figure 4.3

satisfies the constraint, the construction of partial extensions can proceed indefinitely and the number of codewords can increase without bound. However, it is also noted that we cannot continue the recursive concatenation without bound because of limited computing capabilities. It is therefore necessary for this process to be limited by setting one or more thresholds.

There are two standard thresholds available: the number and length of codewords involved in the partial extensions. Once the threshold is set, partial extensions that do not satisfy the thresholds will not be constructed, and the number of partial extensions is thus bounded. In Figure 4.3, we set the upper limit to the number of codewords of partial extension as the threshold, to $Num_{cw} = 10$. The process of constructing partial extensions is repeated until this predetermined threshold is reached. After eliminating duplicate sets of lengths, we obtain the seven simplified and unique partial extensions shown in Figure 4.3.

An algorithm to construct partial extensions can be expressed as follows:

Partial Extension Algorithm

1: **while** number of codewords < predetermined number **do**
2: implement partial extension for each unique length of word in current partial extension
3: **if** any length of words in partial extension > maximum length **then**
4: exclude the partial extension
5: **end**

4.1.3 Capacity and Maxentropic Probability

To apply the nGHC approach, as outlined in Chapter 2, we need to know the capacity of the constraint. With knowledge of the capacity, we can evaluate the maxentropic probabilities of the variable-length codewords. Continuing with the $(4, 1, \infty)$ constraint as an example, its capacity can be evaluated using Equation 3.1. as follows:

$$\begin{aligned} & z^{d+1} - z^d - (M-1) = 0 \\ & z^{1+1} - z^1 - (4-1) = 0 \\ & z^2 - z^1 - (3) = 0 \\ & z_1 = 2.30277, \\ & z_2 = -1.30277, \\ & \lambda_{max} = 2.30277, \\ & C_{4,1,\infty} = \log_2(2.30277) = 1.20337 \end{aligned} \tag{4.1}$$

After determining the capacity, we can then calculate the maxentropic probabilities for the variable-length codewords. Using Equation 2.23, values of these probabilities for two partial extensions shown in Figure 4.3 are given in Figure 4.4.

4.1.4 Normalized Geometric Huffman Coding

Given the maxentropic codeword probabilities, it remains to find source words whose probabilities of occurrence best match the desired codeword probabili-

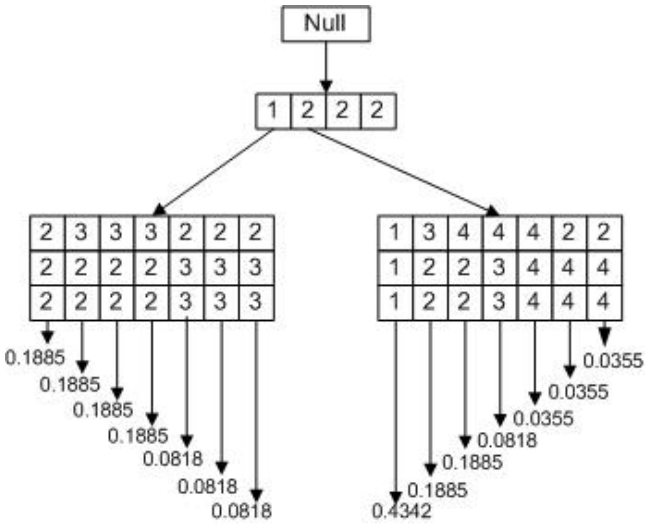

Figure 4.4: Maxentropic probabilities of two sets of codewords in M-ary $(4, 1, \infty)$ codes

ties. As discussed in Chapter 2, this is accomplished with normalized geometric Huffman coding. Details of this method were given in Section 2.3.1; here we summarize the NGHC and GHC algorithms with the following pseudocode:

nGHC Algorithm

$R_{ini} = C$
$R_1 = R_{ini}$
$R_2 = 0$
while $R_2 \neq R_1$ **do**
    $R_2 = R_1$
    $Prob_{cw} = 2^{-o_i R_1}$
    $Prob_{sourcewords} = \mathbf{sort}\left(\mathbf{ghc}\left(Prob_{cw}\right)\right)$
    $l_i = -\log_2\left(Prob_{sourcewords}\right)$
    $m = \mathbf{sum}\left(l_i * Prob_{sourcewords}\right)$
    $n = \mathbf{sum}\left(o_i * Prob_{sourcewords}\right)$
    $R_1 = \frac{m}{n}$
end

GHC Algorithm

$n = \textbf{length}\,(Prob_{codeowords})$
for i=1:n-1 **do**
  $[\sim, index] = \textbf{sort}(Prob_{codeowords}, `ascend')$
  $m = index(1:2)$
  if $2 \times \textbf{sqrt}(Prob_{cw}(m(1)) \times Prob_{cw}(m(2))) \leqslant Prob_{cw}(m(2))$ **then**
    $Prob_{cw} = \textbf{Remove}\,(Prob_{cw}, Prob_{cw}\,(m\,(1)))$
    $cut\_tree\,(i) = true$
  else if **then**
    $m = \textbf{sort}(m, `ascend')$
    allocate a new node z
    $z\_left = Prob_{cw}\,(m\,(1))$
    $z\_right = Prob_{cw}\,(m\,(2))$
    $z\_prob = 2 \times \textbf{sqrt}\,(z\_left \times z\_right)$
    $Prob_{cw} = \textbf{Insert}\,(Prob_{cw}, z\_prob)$
    $Prob_{cw} = \textbf{sort}(Prob_{codeowords}, `ascend')$
    $indices\,(i) = \textbf{index}\,(Prob_{cw}, z\_prob)$
end
end
$L = [0]$
for k=1:n-1 **do**
  if $!cut\_tree(n-k)$ **then**
    $L = \textbf{Remove}\,(L, L\,(indices\,(n-k)))$
    $L = \textbf{Insert}\,(L, L\,(indices\,(n-k)) + 1)$
    $L = \textbf{Insert}\,(L, L\,(indices\,(n-k)) + 1)$
    $L = \textbf{sort}(L, `ascend')$
end
end
return $p = 2^{-L}$

We can apply this algorithm to the codeword lengths generated through partial extensions as described in the previous subsection. As discussed above, the nGHC algorithm requires as input the lengths of the codewords rather than a set of actual codewords. Table 4.1 lists the lengths of both the codewords and the optimal source word lengths, along with the resulting average code rates and efficiencies under the condition that the number of codewords is less than or equal to 10. Following determination of the appropriate source word lengths, the corresponding source words can be constructed as noted below.

	R_{avg}	η	Corresponding length	
$Num_{cw} = 4$	1.1667	96.95%	L_c	1 2 2 2
			L_s	1 2 3 3
$Num_{cw} = 7$	1.1667	96.95%	L_c	1 2 2 3 4 4 4
			L_s	1 2 3 4 5 6 6
	1.1667	96.95%	L_c	2 2 2 2 3 3 3
			L_s	2 2 3 3 3 4 4
$Num_{cw} = 10$	1.1667	96.95%	L_c	1 2 2 3 4 4 5 6 6 6
			L_s	1 2 3 4 6 6 6 7 8 8
	1.1667	96.95%	L_c	1 2 2 4 4 4 4 5 5 5
			L_s	1 2 3 5 5 6 6 6 7 7
	1.1818	98.21%	L_c	1 2 3 3 4 4 4 4 4 4
			L_s	1 3 3 4 5 5 5 5 5 5
	1.1667	96.95%	L_c	2 2 2 2 3 3 4 5 5 5
			L_s	2 2 3 3 3 4 5 6 7 7
	1.1750	97.64%	L_c	2 2 2 3 3 3 3 4 4 4
			L_s	2 2 3 4 4 4 4 4 5 5

Table 4.1: Partial extensions within $Num_{cw} = 10$

4.1.5 Final Code Selection

From the partial extensions considered, we select the one with the highest average code rate as the final code. The codes selected for partial extensions up to $Num_{cw} = 10$ in the tree are listed in Table 4.2. Note that in this instance the code corresponding to $Num_{cw} = 10$ is the most efficient.

	R_{avg}	η	Corresponding length	
$Num_{cw} = 4$	1.1667	96.95%	L_c	1 2 2 2
			L_s	1 2 3 3
$Num_{cw} = 7$	1.1667	96.95%	L_c	1 2 2 3 4 4 4
			L_s	1 2 3 4 5 6 6
	1.1667	96.95%	L_c	2 2 2 2 3 3 3
			L_s	2 2 3 3 3 4 4
$Num_{cw} = 10$	1.1818	98.21%	L_c	1 2 3 3 4 4 4 4 4 4
			L_s	1 3 3 4 5 5 5 5 5 5

Table 4.2: Partial extension with highest code rate

Having found the partial extension that results in the highest code rate, we can redraw the code tree to find codewords with the required lengths. This tree is depicted in Figure 4.5.

A similar tree can also be drawn to find prefix-free source words of the required lengths. This is accomplished using the source word lengths determined from the nGHC algorithm and a tree generated from the minimal set

$\{0, 1\}$. Figure 4.6 depicts the generation of source words corresponding to the codewords in Fig 4.5. Table 4.3 lists the resulting code table outlining possible mapping of source words to code words that satisfies the $(4, 1, \infty)$ constraint with an efficiency of 98.21%.

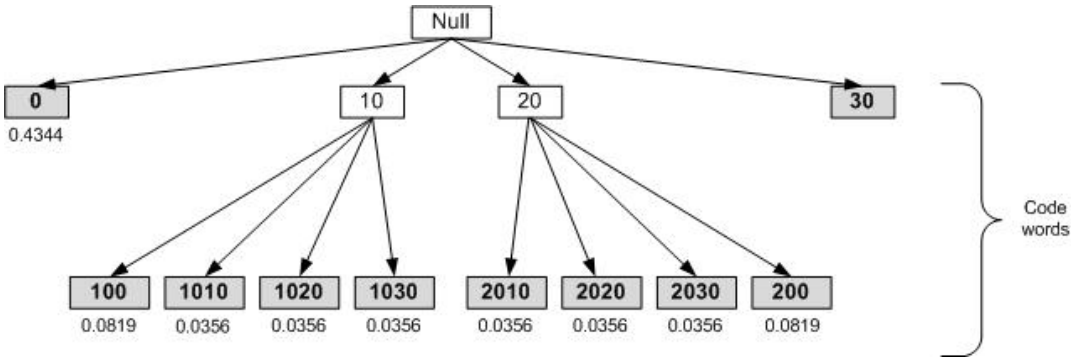

Figure 4.5: Codewords with required lengths for $(4, 1, \infty)$ codes

L_c	1	2	3	3	4	4	4	4	4	4
codewords	0	30	200	100	1010	1020	1030	2010	2020	2030
L_s	1	3	3	4	5	5	5	5	5	5
source words	0	100	101	1100	11011	11010	11100	11101	11110	11111

Table 4.3: Mapping of source words to codewords

4.2 $(4, 1, 11)$ Code

In [12], Kumar constructs a $(4, 1, 11)$ constrained code. He begins his construction by selecting the code rate $R = \frac{6}{5}$ because it is very close to the capacity $C = 1.20331$ for the $(4, 1, \infty)$ constraint. He then constructs a 7-state encoder with code rate $R = \frac{6}{5}$ by limiting the k constraint to a maximum value of 11. His $(4, 1, 11)$ constrained code involves 451 codewords and seven encoder states; its efficiency is 99.72% which is very close to capacity. However, we can construct a more efficient $(4, 1, 11)$ constrained code with fewer codewords by applying our variable-length construction method.

The transition graph for the $(4, 1, 11)$ constraint is depicted in Figure 4.7.

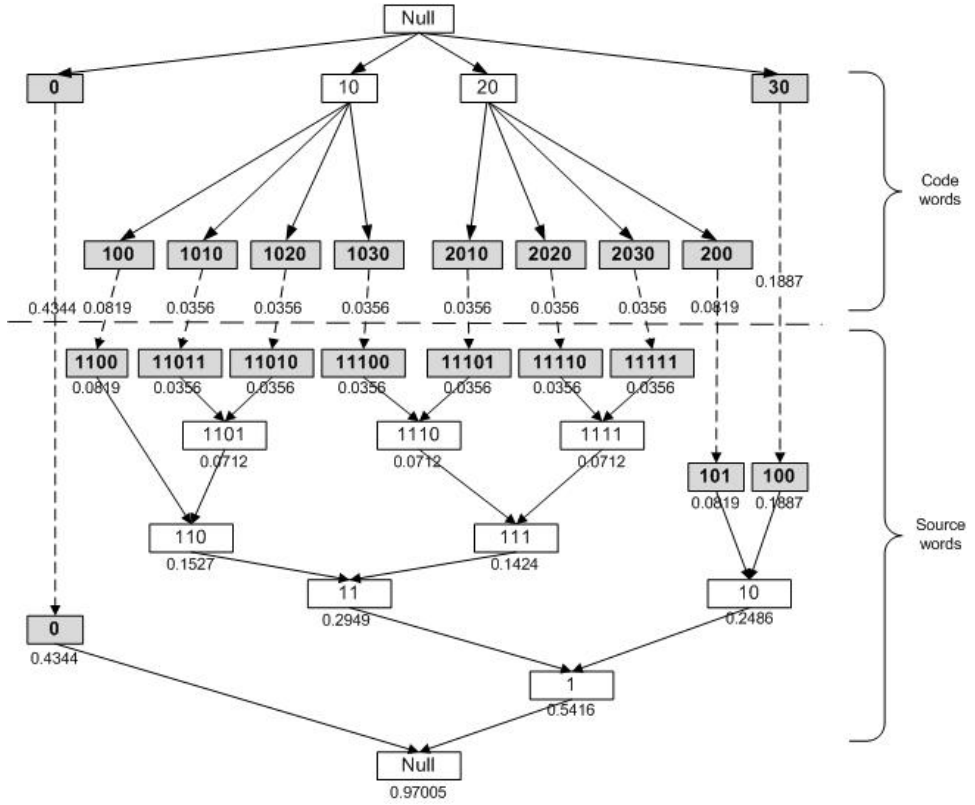

Figure 4.6: Codewords with required lengths for $(4, 1, \infty)$ codes

From this figure, it is straightforward to identify the minimal set and its corresponding word lengths that are listed in Table 4.4.

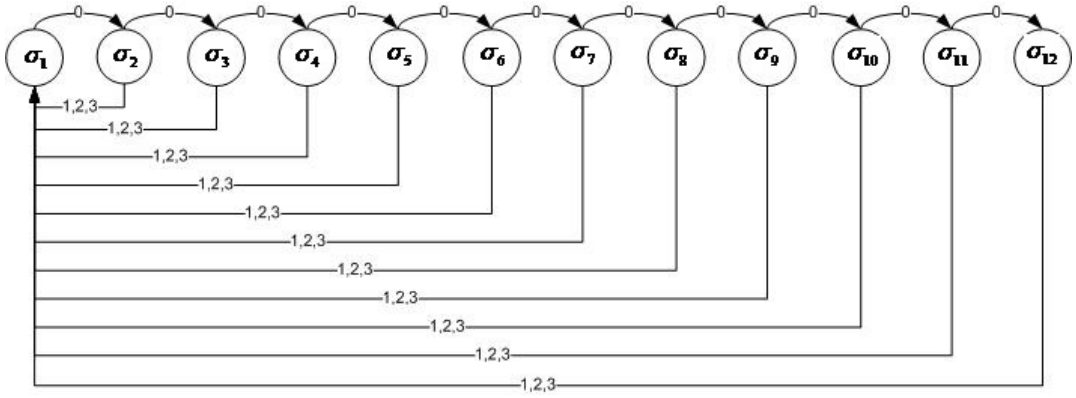

Figure 4.7: Transition graph of M-ary $(4, 1, 11)$ Code

We now consider Equation 3.2 and evaluate the maximum real root λ_{max} in order to find capacity $C_{4,1,11}$:

Minimal Set
01,02,03, 001,002,003, 0001,0002,0003, 00001,00002,00003, 000001,000002,000003, 0000001,0000002,0000003, 00000001,00000002,00000003, 000000001,000000002,000000003, 0000000001,0000000002,0000000003, 00000000001,00000000002,00000000003, 000000000001,000000000002,000000000003,
Minimal Set Length
2,2,2,3,3,3,4,4,4,5,5,5,6,6,6,7,7,7,8,8,8,9,9,9,10,10,10,11,11,11,12,12,12

Table 4.4: Minimal set and corresponding word lengths for $(4, 1, 11)$ constraint

$$\begin{aligned} & z^{k+2} - z^{k+1} - (M-1)\, z^{k-d+1} + M - 1 = 0 \\ & z^{13} - z^{12} - 3z^{11} + 3 = 0 \\ & \lambda_{max} = 2.30268, \\ & C_{4,1,\infty} = \log_2(2.30268) = 1.20331, \end{aligned} \tag{4.2}$$

With knowledge of the capacity $C_{4,1,11}$, we can apply the nGHC method to find partial extensions and corresponding source words for a code with high rate. In our search we limited the lengths of codewords to at most 16. Under this condition, the word lengths for the code with the highest rate is listed in Table 4.5. This code has an average code rate and efficiency of 1.2006 and 99.77% respectively, which are slightly higher than the values for the code constructed by Kumar. Importantly, we achieved this high code rate with only 257 codewords, which is fewer codewords than the code that Kumar constructed.

4.3 $(8, 1, 11)$ Code

In addition to the $(4, 1, 11)$ constrained code, Kumar also constructed a 3-state $(8, 1, 11)$ constrained code with code rate $R = \frac{10}{6}$. His code has an efficiency of 99.52% and contains 3251 codewords.

L_c	L_s	L_c	L_s	L_c	L_s	L_c	L_s	L_c	L_s	L_c	L_s	L_c	L_s
4	5	6	7	8	9	9	11	11	13	13	15	14	17
4	5	6	7	8	10	9	11	11	13	13	16	14	17
4	5	6	7	8	10	9	11	11	13	13	16	15	18
4	5	6	7	8	10	9	11	11	13	13	16	15	18
4	5	6	7	8	10	9	11	11	13	13	16	15	18
4	5	6	7	8	10	9	11	11	13	13	16	15	18
4	5	6	7	8	10	9	11	11	13	13	16	15	18
4	5	6	7	8	10	9	11	11	13	13	16	15	18
4	5	6	7	8	10	10	12	11	13	13	16	15	18
4	5	6	7	8	10	10	12	11	13	13	16	15	18
4	5	6	7	8	10	10	12	11	13	13	16	15	18
5	6	6	7	8	10	10	12	11	13	13	16	15	18
5	6	6	7	8	10	10	12	11	13	13	16	15	18
5	6	6	7	8	10	10	12	11	13	13	16	15	18
5	6	6	7	8	10	10	12	11	13	13	16	16	18
5	6	6	7	8	10	10	12	11	13	13	16	16	19
5	6	7	8	8	10	10	12	12	14	13	16	16	19
5	6	7	8	8	10	10	12	12	14	13	16		
5	6	7	8	8	10	10	12	12	14	13	16		
5	6	7	8	8	10	10	12	12	14	13	16		
5	6	7	8	8	10	10	12	12	14	13	16		
5	6	7	8	8	10	10	12	12	14	14	17		
5	6	7	8	8	10	10	12	12	14	14	17		
5	6	7	8	8	10	10	12	12	14	14	17		
5	6	7	8	9	11	10	12	12	14	14	17		
5	6	7	8	9	11	10	12	12	14	14	17		
5	6	7	8	9	11	10	12	12	14	14	17		
5	6	7	8	9	11	10	12	12	14	14	17		
5	6	7	8	9	11	10	12	12	14	14	17		
5	6	7	8	9	11	10	12	12	14	14	17		
5	6	7	8	9	11	10	12	12	14	14	17		
5	6	7	8	9	11	10	12	12	14	14	17		
6	7	7	8	9	11	11	13	12	14	14	17		
6	7	7	8	9	11	11	13	12	14	14	17		
6	7	7	8	9	11	11	13	12	14	14	17		
6	7	7	8	9	11	11	13	12	14	14	17		
6	7	7	8	9	11	11	13	12	14	14	17		
6	7	7	8	9	11	11	13	12	14	14	17		
6	7	7	8	9	11	11	13	12	15	14	17		
6	7	7	8	9	11	11	13	12	15	14	17		

Table 4.5: Word lengths of partial extension and corresponding source words lengths for high rate $(4, 1, 11)$ code $R_{avg} = 1.2006$, $\eta = 99.77\%$

For purpose of comparison, we report a variable-length $(8, 1, 11)$ constrained code by our method. The transition graph for the $(M = 8, d = 1, k = 11)$ constraint is depicted in Figure 4.8. The minimal set and its word lengths are listed in Table 4.6.

By evaluating the maximum real root λ_{max} of Equation 3.2, we determine the capacity $C_{8,1,11}$:

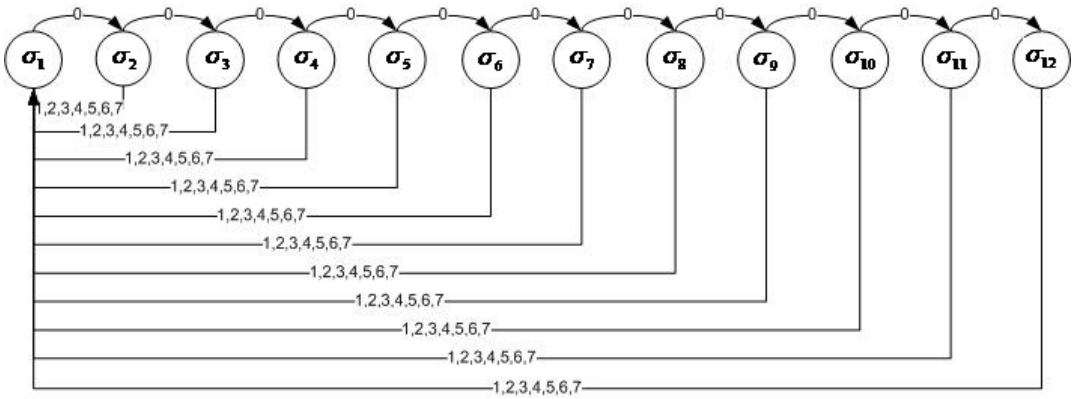

Figure 4.8: Transition graph of M-ary $(8, 1, 11)$ Code

Minimal Sets
01,02,03,04,05,06,07
001,002,003,004,005,006,007
0001,0002,0003,0004,0005,0006,0007
00001,00002,00003,00004,00005,00006,00007
000001,000002,000003,000004,000005,000006,000007
0000001,0000002,0000003,0000004,0000005,0000006,0000007
00000001,00000002,00000003,00000004,00000005,00000006,00000007
000000001,000000002,000000003,000000004,000000005,000000006,000000007
0000000001,0000000002,0000000003,0000000004,0000000005,0000000006,0000000007
00000000001,00000000002,00000000003,00000000004,00000000005,00000000006,00000000007
000000000001,000000000002,000000000003,000000000004,000000000005,000000000006,000000000007
Minimal Sets Length
2,2,2,2,2,2,2
3,3,3,3,3,3,3
4,4,4,4,4,4,4
5,5,5,5,5,5,5
6,6,6,6,6,6,6
7,7,7,7,7,7,7
8,8,8,8,8,8,8
9,9,9,9,9,9,9
10,10,10,10,10,10,10
11,11,11,11,11,11,11
12,12,12,12,12,12,12

Table 4.6: Minimal set and corresponding lengths for $(8, 1, 11)$ Code

$$
\begin{aligned}
& z^{k+2} - z^{k+1} - (M-1)\, z^{k-d+1} + M - 1 = 0 \\
& z^{13} - z^{12} - 7z^{11} + 7 = 0 \\
& \lambda_{max} = 3.1926, \\
& C_{8,1,\infty} = \log_2(3.1926) = 1.6747,
\end{aligned}
\tag{4.3}
$$

With the knowledge of this capacity, we apply the nGHC method to partial extensions of the minimal set to evaluate the source word lengths for a code with high average rate. In our search we limited the number of codewords

of partial extensions to at most 609 and the lengths of codewords to at most 16. The word lengths of the resulting code with the highest rate are listed in Table 4.7. The efficiency and number of codewords of our variable-length code are 99.54% and 609 respectively. This code is therefore superior to the $(8, 1, 11)$ code constructed by Kumar in terms of both average code rate and implementation complexity.

L_c	L_s	L_c	L_s	L_c	L_s	L_c	L_s	L_c	L_s	L_c	L_s	L_c	L_s	L_c	L_s	L_c	L_s	L_c	L_s
2	3	5	8	6	10	7	12	8	13	9	15	10	17	11	18	12	20	14	23
3	5	5	8	6	10	7	12	8	13	9	15	10	17	11	18	12	20	14	23
3	5	5	8	6	10	7	12	8	13	9	15	10	17	11	18	12	20	14	23
3	5	5	8	6	10	7	12	8	13	9	15	10	17	11	18	12	20	14	23
3	5	5	8	6	10	7	12	8	13	9	15	10	17	11	18	12	20	14	23
3	5	5	8	6	10	7	12	8	13	9	15	10	17	11	18	12	20	14	23
3	5	5	8	6	10	7	12	8	13	9	15	10	17	11	18	12	20	14	23
3	5	5	8	6	10	7	12	8	13	9	15	10	17	11	18	12	20	14	23
4	6	5	8	6	10	7	12	8	13	9	15	10	17	11	18	12	20	14	23
4	7	5	8	6	10	7	12	8	13	9	15	10	17	11	18	13	21	14	23
4	7	5	8	6	10	7	12	8	13	9	15	10	17	11	18	13	21	14	23
4	7	5	8	6	10	7	12	8	13	9	15	10	17	11	18	13	21	14	23
4	7	5	8	6	10	7	12	8	13	9	15	10	17	11	18	13	21	14	23
4	7	5	8	6	10	7	12	8	13	9	15	10	17	11	18	13	21	14	23
4	7	5	8	6	10	7	12	8	13	9	15	10	17	12	20	13	21	14	23
4	7	5	8	6	10	7	12	8	13	9	15	10	17	12	20	13	21	14	23
4	7	5	8	6	10	7	12	8	13	9	15	10	17	12	20	13	21	14	23
4	7	5	8	6	10	7	12	8	13	9	15	10	17	12	20	13	21	14	23
4	7	5	8	6	10	7	12	8	13	9	15	10	17	12	20	13	21	14	23
4	7	5	8	6	10	7	12	8	13	9	15	11	18	12	20	13	21	14	23
4	7	5	8	6	10	7	12	8	13	9	15	11	18	12	20	13	21	14	23
4	7	5	8	6	10	7	12	8	13	9	15	11	18	12	20	13	21	14	23
4	7	5	8	6	10	7	12	8	13	9	15	11	18	12	20	13	21	14	23
4	7	5	8	6	10	7	12	8	13	9	15	11	18	12	20	13	21	14	23
4	7	5	8	6	10	7	12	8	13	10	16	11	18	12	20	13	21	14	23
4	7	5	8	6	10	7	12	8	13	10	16	11	18	12	20	13	21	14	23
4	7	5	8	6	10	7	12	8	13	10	17	11	18	12	20	13	21	14	23
4	7	5	8	6	10	7	12	8	13	10	17	11	18	12	20	13	21	14	23
4	7	5	8	6	10	7	12	8	13	10	17	11	18	12	20	13	21	14	23
4	7	5	8	6	10	7	12	9	15	10	17	11	18	12	20	13	21	14	23
4	7	5	8	6	10	7	12	9	15	10	17	11	18	12	20	13	22	14	23
4	7	5	8	6	10	7	12	9	15	10	17	11	18	12	20	13	22	14	23
4	7	5	8	6	10	7	12	9	15	10	17	11	18	12	20	13	22	14	23
4	7	5	8	6	10	7	12	9	15	10	17	11	18	12	20	13	22	14	23
4	7	5	8	6	10	8	13	9	15	10	17	11	18	12	20	13	22	14	23
4	7	5	8	6	10	8	13	9	15	10	17	11	18	12	20	13	22	14	23
4	7	5	8	6	10	8	13	9	15	10	17	11	18	12	20	13	22	14	23
4	7	5	8	6	10	8	13	9	15	10	17	11	18	12	20	13	22	14	23
4	7	5	8	6	10	8	13	9	15	10	17	11	18	12	20	13	22	14	23
4	7	5	8	7	11	8	13	9	15	10	17	11	18	12	20	13	22	14	23
4	7	5	8	7	11	8	13	9	15	10	17	11	18	12	20	13	22	14	23
4	7	5	8	7	11	8	13	9	15	10	17	11	18	12	20	13	22	14	23
4	7	5	8	7	11	8	13	9	15	10	17	11	18	12	20	13	22	14	23
4	7	5	8	7	11	8	13	9	15	10	17	11	18	12	20	13	22	14	23
4	7	6	10	7	11	8	13	9	15	10	17	11	18	12	20	13	22	14	23
4	7	6	10	7	11	8	13	9	15	10	17	11	18	12	20	13	22	14	23
4	7	6	10	7	11	8	13	9	15	10	17	11	18	12	20	13	22	15	24
4	7	6	10	7	11	8	13	9	15	10	17	11	18	12	20	13	22	15	24
4	7	6	10	7	11	8	13	9	15	10	17	11	18	12	20	13	22	15	25
4	7	6	10	7	11	8	13	9	15	10	17	11	18	12	20	13	22	15	25
4	7	6	10	7	11	8	13	9	15	10	17	11	18	12	20	13	22	15	25
4	7	6	10	7	11	8	13	9	15	10	17	11	18	12	20	13	22	15	25
4	7	6	10	7	11	8	13	9	15	10	17	11	18	12	20	13	22	15	25
4	7	6	10	7	11	8	13	9	15	10	17	11	18	12	20	13	22	16	26
4	7	6	10	7	11	8	13	9	15	10	17	11	18	12	20	13	22	16	26
4	7	6	10	7	11	8	13	9	15	10	17	11	18	12	20	13	22	16	26
5	8	6	10	7	11	8	13	9	15	10	17	11	18	12	20	13	22	16	26
5	8	6	10	7	12	8	13	9	15	10	17	11	18	12	20	13	22	16	26
5	8	6	10	7	12	8	13	9	15	10	17	11	18	12	20	14	23	16	27
5	8	6	10	7	12	8	13	9	15	10	17	11	18	12	20	14	23	16	27
5	8	6	10	7	12	8	13	9	15	10	17	11	18	12	20	14	23		

Table 4.7: Word lengths of partial extension and corresponding source words lengths for $(8, 1, 11)$ code $R_{avg} = 1.6671$, $\eta = 99.54\%$

4.4 Variable-length ML-RLL Codes with *d=1*

While we have demonstrated that our approach can yield simpler and more efficient M-ary RLL codes than the fixed-length codes constructed by Kumar, we also stress that our construction method is more flexible and easier to apply to any constraint since it is unnecessary for us, in advance, to find a ratio of block lengths that is close to capacity. As a result, our method will work regardless of the value of capacity and can be applied to a diversity of constraints, while Kumar's approach for constructing fixed-length ML-RLL codes can only be used in particular cases where capacity is close to, but not lower than, a rational number with a low-valued numerator and denominator.

To demonstrate the flexibility of our approach, we constructed variable-length ML-RLL codes for all the constraints where whose capacities were listed in Table 3.1. We present our results in Tables 4.8 - 4.11.

From these tables, it is evident that for all these constraints we constructed codes with efficiencies greater than 99%. Also, our codes are relatively straightforward. The simplest code, for the $(3, 1, \infty)$ constraint, contains only three codewords with maximum length two and has an average code rate equal to capacity. The most complex code, for the $(8, 1, 10)$ constraint, has 415 codewords with a maximum codeword length equal to 13.

M/k	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10	k= ∞
M=2	0.40385	0.54828	0.61436	0.64843	0.66626	0.67629	0.68223	0.68590	0.68821	0.69231
M=3	0.81818	0.91701	0.95911	0.97947	0.98968	0.99482	0.99740	0.99870	0.99935	1.00000
M=4	1.06175	1.14527	1.17834	1.18972	1.19425	1.19622	1.19705	1.19745	1.19759	1.19753
M=5	1.25000	1.31449	1.33636	1.34414	1.34710	1.34830	1.34877	1.34895	1.34902	1.34783
M=6	1.38542	1.44791	1.46702	1.47282	1.47485	1.47557	1.47582	1.47591	1.47594	1.47619
M=7	1.50364	1.55537	1.56952	1.57425	1.57563	1.57621	1.57638	1.57646	1.57648	1.57539
M=8	1.59805	1.64629	1.66073	1.66455	1.66575	1.66612	1.66624	1.66627	1.66628	1.66292

Table 4.8: Highest code rates achieved for $d = 1$ variable-length ML-RLL codes

M/k	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10	k= ∞
M=2	99.55%	99.42%	99.50%	99.62%	99.59%	99.56%	99.56%	99.58%	99.61%	99.72%
M=3	99.39%	99.08%	99.28%	99.55%	99.75%	99.87%	99.93%	99.96%	99.98%	100.00%
M=4	98.95%	99.22%	99.57%	99.56%	99.54%	99.53%	99.53%	99.53%	99.53%	99.51%
M=5	99.78%	99.51%	99.45%	99.42%	99.41%	99.41%	99.41%	99.41%	99.41%	99.32%
M=6	99.41%	99.66%	99.71%	99.68%	99.67%	99.67%	99.67%	99.66%	99.66%	99.68%
M=7	99.60%	99.57%	99.48%	99.47%	99.46%	99.46%	99.46%	99.47%	99.47%	99.40%
M=8	99.36%	99.43%	99.51%	99.50%	99.50%	99.50%	99.50%	99.50%	99.50%	99.30%

Table 4.9: Efficiency of our $d = 1$ variable-length ML-RLL codes

M/k	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10	k= ∞
M=2	7	13	19	25	31	37	43	49	55	5
M=3	13	26	43	55	67	79	91	103	115	3
M=4	31	41	67	85	103	121	139	157	175	19
M=5	36	67	91	115	139	163	187	211	235	17
M=6	55	85	115	145	175	205	235	265	295	31
M=7	67	103	139	175	211	247	283	319	355	37
M=8	79	121	163	205	247	289	331	373	415	43

Table 4.10: Number of codewords in our $d = 1$ variable-length ML-RLL codes

M/k	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10	k= ∞
M=2	9	12	11	13	12	13	14	15	16	4
M=3	6	9	12	14	16	16	16	16	16	2
M=4	8	7	8	9	10	11	12	13	14	5
M=5	5	8	9	10	11	12	13	14	15	4
M=6	7	8	8	9	10	11	12	13	14	5
M=7	6	8	9	10	11	12	13	14	15	6
M=8	7	8	7	8	9	10	11	12	13	4

Table 4.11: Maximum codeword length in our $d = 1$ variable-length ML-RLL codes

4.5 Variable-length ML-RLL Codes with *d=2*

We have also applied our method to the construction of variable-length ML-RLL codes that satisfy the constraint $d = 2$ where M ranges from 2 to 8 and when $k = \infty$ and also ranges from 3 to 10. The capacities of these constraints are listed in Table 4.12. Our code results are listed in Tables 4.13 - 4.16.

Similar to the $d = 1$ constrained codes above, it is evident that we constructed highly efficient codes for all these constraints with efficiencies greater than 99%. However, the complexity of these codes is also acceptably low. The simplest code, for the $(5, 2, \infty)$ constraint, contains only five codewords with maximum length three and has an average code rate equal to capacity. Also, the most complex code, for the $(8, 2, 10)$ constraint, has only 311 codewords with a maximum length equal to 14.

M/k	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10	k=∞
M=2	0.287761	0.405685	0.464958	0.497906	0.517370	0.529340	0.536911	0.541797	0.551463
M=3	0.579692	0.671857	0.714162	0.735555	0.746990	0.753316	0.756894	0.758947	0.761814
M=4	0.752420	0.831349	0.865384	0.881485	0.889498	0.893607	0.895751	0.896883	0.898175
M=5	0.875848	0.946210	0.975110	0.988083	0.994186	0.997133	0.998578	0.999292	1.000000
M=6	0.972085	1.036267	1.061590	1.072473	1.077358	1.079603	1.080648	1.081137	1.081574
M=7	1.051037	1.110467	1.133122	1.142501	1.146544	1.148325	1.149119	1.149474	1.149764
M=8	1.118014	1.173632	1.194203	1.202443	1.205873	1.207328	1.207952	1.208221	1.208425

Table 4.12: Capacities of $d = 2$ variable-length ML-RLL codes

M/k	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10	k=∞
M=2	0.28571	0.40336	0.46218	0.49585	0.51544	0.52664	0.53401	0.53887	0.54839
M=3	0.57609	0.66870	0.71087	0.73171	0.74338	0.74951	0.75291	0.75490	0.75758
M=4	0.74701	0.82698	0.86074	0.87603	0.88377	0.88768	0.88975	0.89103	0.89000
M=5	0.86957	0.93811	0.96924	0.98454	0.99224	0.99611	0.99805	0.99902	1.00000
M=6	0.96539	1.02883	1.05199	1.06267	1.06763	1.07006	1.07127	1.07187	1.07122
M=7	1.04187	1.09967	1.12401	1.13457	1.13893	1.14093	1.14191	1.14238	1.13954
M=8	1.10739	1.16500	1.18710	1.19632	1.19985	1.20142	1.20212	1.20242	1.19817

Table 4.13: Highest code rate achieved for $d = 2$ variable-length ML-RLL codes

M/k	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10	k=∞
M=2	99.29%	99.43%	99.40%	99.59%	99.63%	99.49%	99.46%	99.46%	99.44%
M=3	99.38%	99.53%	99.54%	99.48%	99.52%	99.50%	99.47%	99.47%	99.44%
M=4	99.28%	99.47%	99.46%	99.38%	99.36%	99.34%	99.33%	99.35%	99.09%
M=5	99.28%	99.14%	99.40%	99.64%	99.80%	99.90%	99.95%	99.97%	100.00%
M=6	99.31%	99.28%	99.10%	99.09%	99.10%	99.12%	99.13%	99.14%	99.04%
M=7	99.13%	99.03%	99.20%	99.31%	99.34%	99.36%	99.37%	99.38%	99.11%
M=8	99.05%	99.26%	99.41%	99.49%	99.50%	99.51%	99.52%	99.52%	99.15%

Table 4.14: Efficiency of our $d = 2$ variable-length ML-RLL codes

M/k	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10	k=∞
M=2	3	9	16	21	26	31	36	41	6
M=3	16	21	36	46	56	66	76	86	11
M=4	26	41	56	71	86	101	116	131	16
M=5	36	56	76	96	116	136	156	176	5
M=6	46	71	96	121	146	171	196	221	31
M=7	56	86	116	146	176	206	236	266	31
M=8	79	101	136	171	206	241	276	311	29

Table 4.15: Number of codewords in our $d = 2$ variable-length ML-RLL codes

M/k	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10	k=∞
M=2	8	11	14	14	16	17	20	19	7
M=3	10	10	15	13	14	15	17	17	6
M=4	11	9	10	11	12	14	14	15	7
M=5	7	10	12	14	16	18	20	20	3
M=6	10	10	10	11	12	13	14	15	7
M=7	8	8	9	10	11	12	13	14	6
M=8	8	8	9	10	11	12	13	14	6

Table 4.16: Maximum codeword length in our $d = 2$ variable-length ML-RLL codes

Chapter 5

Variable-length balanced codes for QPSK systems

In Chapter 4, we introduced variable-length RLL codes for multi-level magnetic recording systems. By applying our code construction method to QPSK systems, we extend our approach to constructing balanced codes for these bandpass systems. In this chapter, we demonstrate the construction of capacity-approaching balanced QPSK codes and present parameters of codes we construct including their spectral performance. As in the previous chapter, these codes are comprised of variable-length codewords that can be decoded instantaneously. In this application, however, our balanced QPSK symbol sequences contain, on average, an equal number of different symbol values in order to assist demodulators to recover the original data accurately. The average code rates of these codes are higher than code rates for any previously reported fixed-length balanced QPSK codes.

5.1 Constraint for Balanced QPSK Codes

As noted in Chapter 2, a sequence is balanced if and only if its running digital sum (RDS) is bounded. For this reason, before we introduce the construction

of balanced QPSK codes we give a brief introduction to the particular RDS constraint we consider in this chapter.

Balanced QPSK codes can be considered as an extension of balanced BPSK codes, where in the equivalent baseband representation the real signalling dimension for BPSK is extended into the complex plane for QPSK signals. That is, the modulation signalling points and the RDS constraints for balanced BPSK codes are defined only on the real axis, whereas the signalling points and RDS constraints of balanced QPSK codes are defined in both the real and imaginary dimensions.

Figure 5.1 depicts the QPSK signalling points $\{1, j, -1, -j\}$ and RDS limits we consider in this chapter. It should be noted that our results are independent of the particular one-to-one mapping of symbol values to signaling points. In the following we arbitrarily map the symbol values $\{0, 1, 2, 3\}$ to signaling points $\{1, j, -1, -j\}$ respectively without affecting the final result. These symbol values are also shown in Figure 5.1. Although our method can be used to construct balanced QPSK codes that satisfy various RDS constraints, we consider only the constraint considered in [15]. This constraint limits the RDS of the encoded sequence to the following nine values: $\{0, 1, 1+j, j, -1+j, -1, -1-j, -j, 1-j\}$. These nine values are also depicted in Fig. 5.1.

5.2 Code Construction

In accordance with the method described in Chapter 2, we can construct a variable-length balanced QPSK code using the following steps:

1. Find a minimal set whose words can be freely concatenated to satisfy the given RDS constraint;

2. Construct instantaneously decodable codewords by constructing partial extensions of the minimal set based on a tree structure;

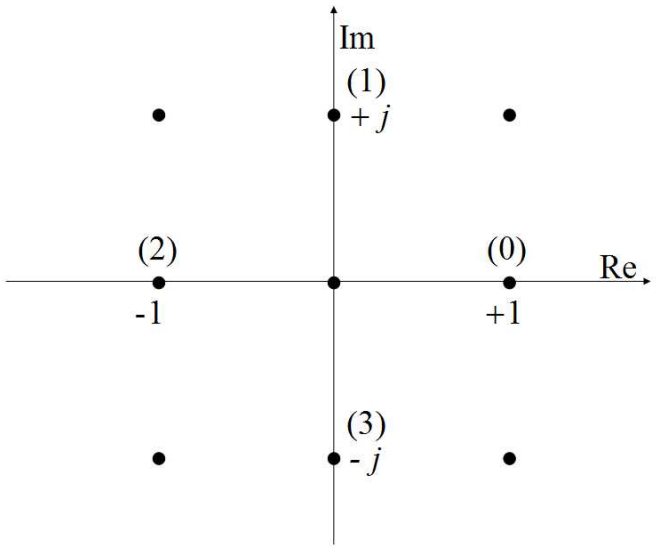

Figure 5.1: Signalling points and permitted RDS values in our balanced QPSK codes

3. Use normalized geometric Huffman coding to generate the optimal mapping of source word lengths to codewords lengths for each set of codewords;

4. Evaluate the average code rate of each constructed code, select the code with the highest rate and construct the corresponding codewords and source words.

We discuss these steps with reference to our balanced QPSK code in the subsection below.

5.2.1 Minimal Set

As with variable-length M-ary RLL constrained codes, the first step in constructing variable-length balanced QPSK codes is to determine the relevant minimal set. In this case, we choose to form the minimal set with words that start and end with RDS = 0. Furthermore, we prove that all the words in this

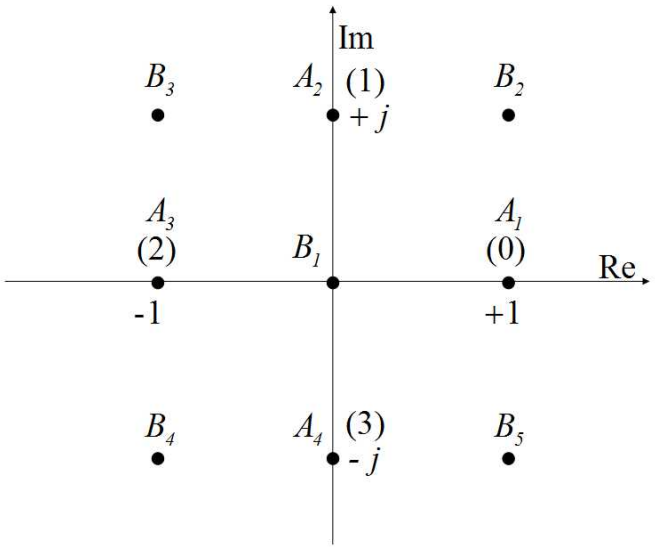

Figure 5.2: Two subsets of permitted RDS values

minimal set are unique and prefix free and therefore that this minimal set and its partial extensions are guaranteed to be instantaneously decodable [15].

We separate the allowable RDS bounds depicted in Fig. 5.1 into two subsets, both of which are shown in Fig. 5.2. The first set includes four positions $\{A_1, A_2, A_3, A_4\}$ from which the RDS can return to 0 only after an odd number of symbols. The second set includes the other five positions $\{B_1, B_2, B_3, B_4, B_5\}$ from which RDS = 0 can occur only after an even number of symbols. For instance, if the RDS after some number of leading symbols of a word is in position $\{A_1\}$, then an appropriate number of odd symbols such as $\{123\}$ will return this word to the origin B_1 where the RDS = 0. Also, since words in the minimal set both start and end with RDS = 0, it follows that the words in the minimal set must have even length. Based on these observations, we can derive the rule upon which we can determine a minimal set for balanced QPSK codes.

We now show that the number of unique words of length l that start with RDS = 0 and end the first time their RDS returns to 0 is 2^l. We take the

simplest case for example where the length of words is $l = 2$ to illustrate how we arrive at this conclusion.

It is straightforward to verify that there are 2^2 words of length = 2 that end with RDS = 0 when they start with RDS = 0: they are $\{02, 13, 20, 31\}$. Note that the word $\{02\}$ describes the only way in which the RDS can be in position A_1 (having RDS = 1) immediately prior to the last symbol and subsequently returns to the origin B_1 (RDS = 0) at the end of the word. Based on their 4-ary symmetry, similar arguments can be applied to the other three length-2 words.

We now derive four unique words of length $l = 4$ that start and end at the origin by replacing the last symbol of the word $\{02\}$. As noted above, the RDS prior to the last symbol in this word has value RDS = 1. Then, it is straightforward to confirm that there are exactly four unique ways in which the RDS can return to the origin B_1 from position A_1 for the first time after three additional symbols. These four unique ways can be described as the replacement of the last symbol 2 with each of the symbol sequences $\{132, 123, 312, 321\}$. This replacement process is shown in Fig. 5.3.

Symmetrical arguments can be made for the other three words of length $l = 2$. Since each of the 2^2 words of length $l = 2$ can be extended to 2^2 words of length $l = 4$ that return to RDS = 0 only at the end of the word, the result is the set of $2^2 2^2 = 2^4$ words of length $l = 4$ shown in Fig. 5.4. .

Note that each word in the set of 2^4 words of length $l = 4$ is unique and returns to RDS = 0 for the first time at the end of the word. Based on this observation, we conclude that the first three leading symbols of each of these words are also unique and can be used to derive the words of length $l = 6$. For instance in the length 4 words, there are four possible ways to reach the position A_1 immediately prior to the last symbol; these four possible ways are described by the three leading symbols of words that end with symbol $\{2\}$. Using the argument above, each of these words of length $l = 4$ can be extended

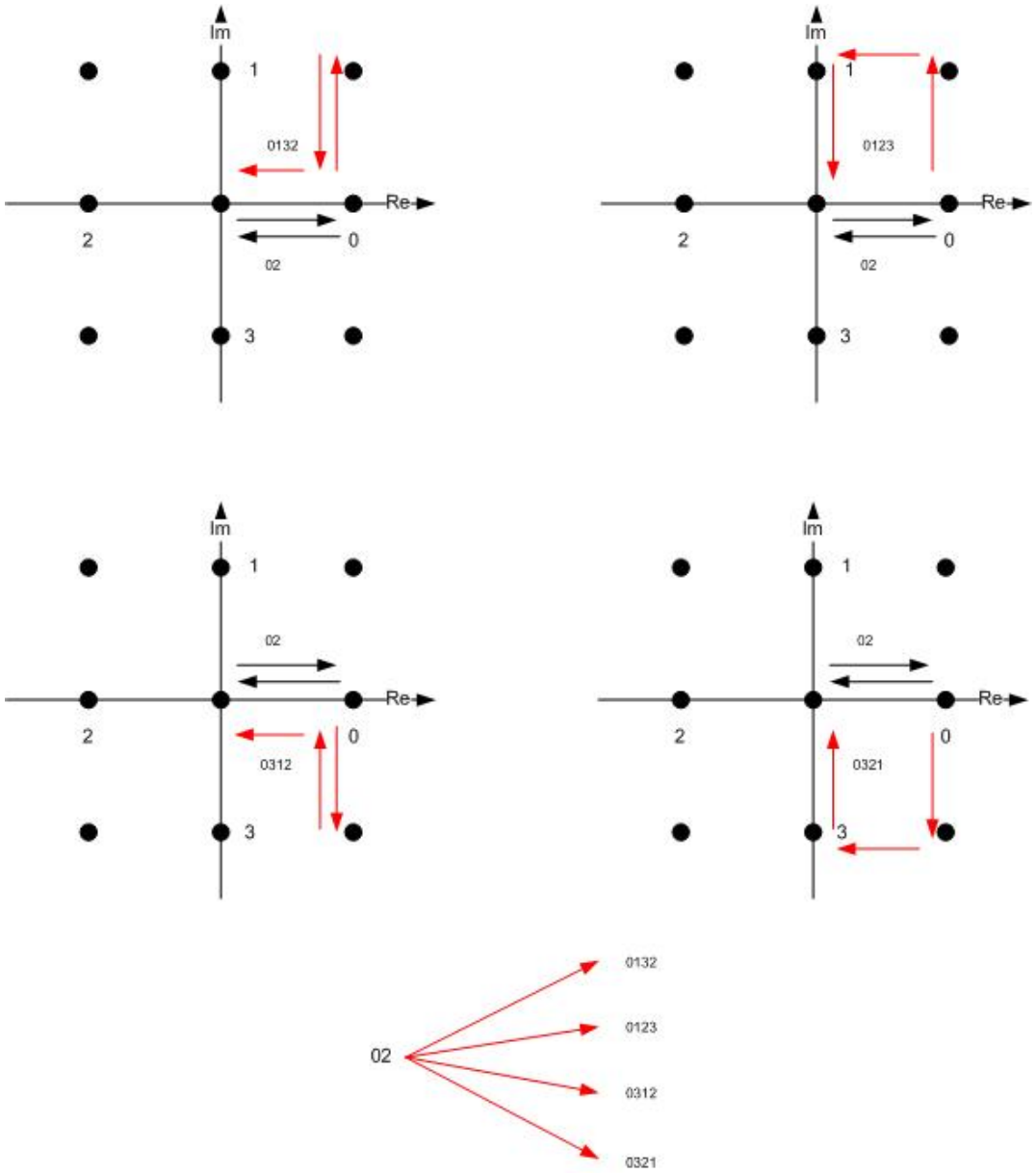

Figure 5.3: Replacement process of the last symbol in the word {02}

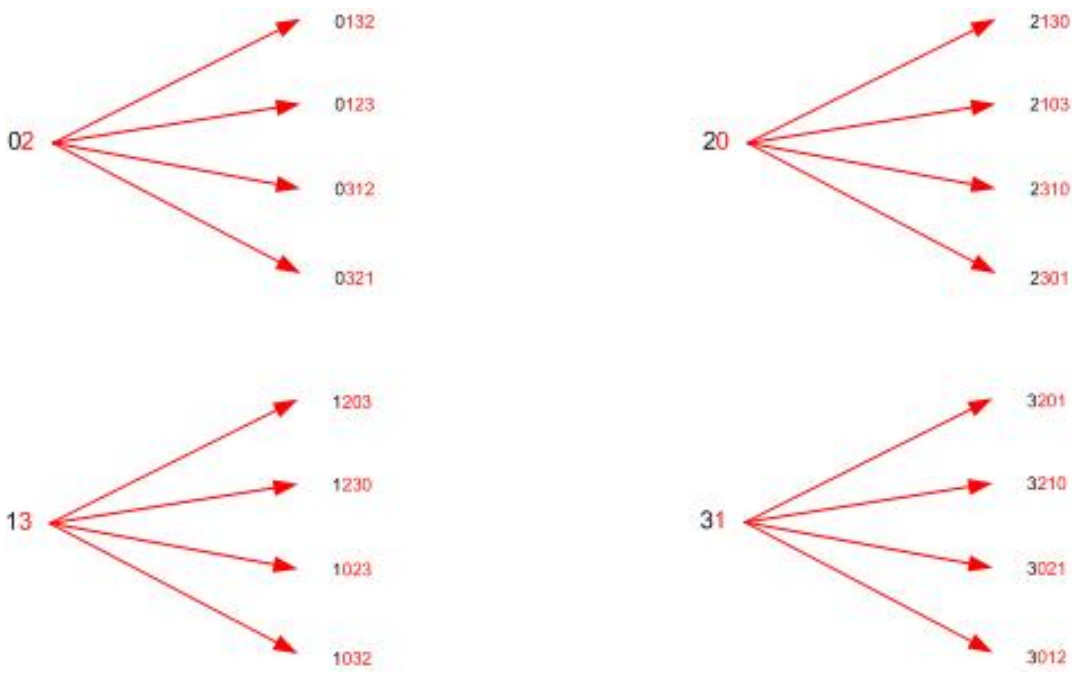

Figure 5.4: Replacement process of last symbol in words of length 2 to construct words of length 4

to four words of length $l = 6$ by replacing the last symbol $\{2\}$ with the four symbol sequences $\{123, 321, 132, 312\}$. Owing to symmetry, this result can also be applied to the other words of length $l = 4$. By replacing the final symbol of the length $l = 4$ words with four appropriate symbols, $2^4 2^2 = 2^6$ balanced words of length $l = 6$ are generated. All these words are shown in Fig. 5.5.

With knowledge of the above replacement process, it is straightforward to enumerate all words of longer length. Since each set of words of length $l - 2$ includes 2^{l-2} unique symbol sequences of length $l - 3$ that describe 2^{l-2} ways to reach each RDS value from the set $\{A_1, A_2, A_3, A_4\}$, we can form balanced words of length l by replacing the last symbol of each word of length $l-2$ with four symbol sequences listed in Fig. 5.4. Therefore, it follows that the number of balanced words of length l is $2^{l-2}2^2 = 2^l$.

Clearly, the length of the words in the minimal set is unbounded, which implies that the complete minimal set contains an infinite number of words. However, it is impractical for us to generate a complete minimal set to apply our

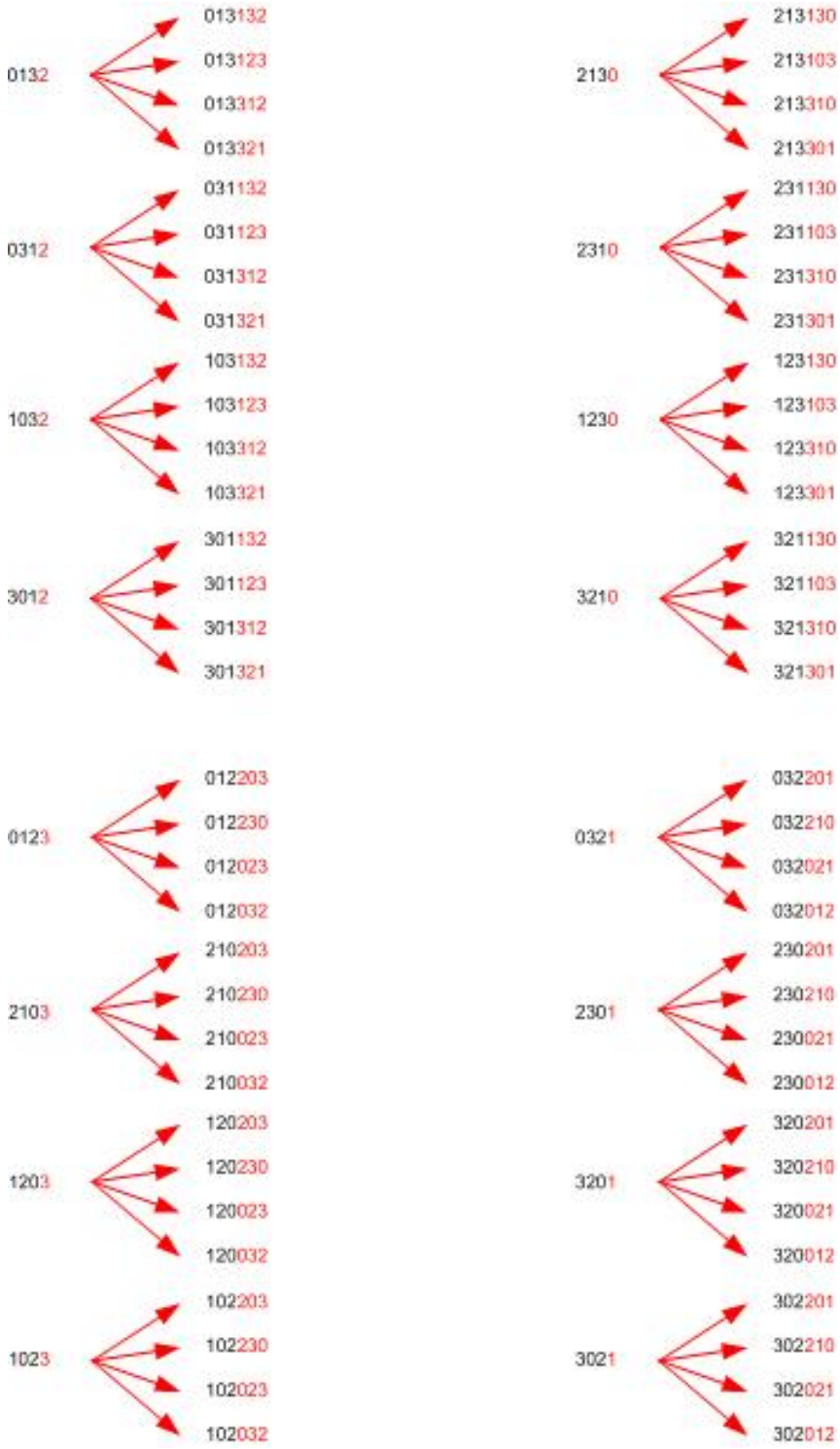

Figure 5.5: Replacement process of last symbol in words of length 4 to construct words of length 6

new method. In this chapter, we consider an incomplete minimal set instead by setting restrictions on the infinitely large complete set to bound the complexity of the code. We propose limiting the length of words in the minimal set to some finite value l_{max}. This implies that some constraint-satisfying sequences will never be used, limiting the rate of the code to a value less than capacity. Table 5.1 lists incomplete minimal sets with l_{max} = 2,4,6,8.

5.2.2 Capacity

Based on the incomplete minimal set, we can apply our new method to construct variable-length balanced QPSK codes. However, we must have knowledge of the capacity of sequences that satisfy the RDS constraint to accomplish this task. Evaluation of the capacity of systems with multi-dimensional signaling alphabets was considered in [28]. As before, the capacity can be calculated by evaluating the largest eigenvalue λ_{max} of the connection matrix $\mathbf{D}$ that describes the overall constraints. In addition, since our system includes independent RDS constraints in the real and imaginary dimensions, it was shown that its connection matrix $\mathbf{D}$ can be found as

$$\mathbf{D} = \mathbf{D}_1 \otimes I_{n_2} + I_{n_1} \otimes \mathbf{D}_2 \tag{5.1}$$

where $\mathbf{D}_1$ is the connection matrix representing the RDS constraint in the real dimension, $\mathbf{D}_2$ is the connection matrix representing the RDS constraint in the imaginary dimension, and the identity matrices are of the appropriate size.

Although we can deduce λ_{max} from the overall connection matrix $\mathbf{D}$, another straightforward relationship to calculate λ_{max} was developed in [29]. Since $\mathbf{D}$ is the Kronecker sum of $\mathbf{D}_1$ and $\mathbf{D}_2$, the maximum eigenvalue of $\mathbf{D}$ is the summation of the maximum eigenvalues of each of $\mathbf{D}_1$ and $\mathbf{D}_2$. That is, because the constraints in each dimension are independent of each other:

incomplete minimal set						
l_max = 2	l_max = 4	l_max = 6	l_max = 8			
02	0123	012023	01202023	01203123	21003123	01313123
20	0132	012032	01202032	01203132	21003132	01313132
13	0312	012203	01202203	01203312	21003312	01313312
31	0321	012230	01202230	01203321	21003321	01313321
	1023	032012	01220023	01223103	21023103	01331123
	1032	032021	01220032	01223130	21023130	01331132
	1203	032201	01220203	01223301	21023301	01331312
	1230	032210	01220230	01223310	21023310	01331321
	2103	102023	03202012	01312023	21310023	03113123
	2130	102032	03202021	01312032	21310032	03113132
	2301	102203	03202201	01312203	21310203	03113312
	2310	102230	03202210	01312230	21310230	03113321
	3012	120023	03220012	01332012	21330012	03131123
	3021	120032	03220021	01332021	21330021	03131132
	3201	120203	03220201	01332201	21330201	03131312
	3210	120230	03220210	01332210	21330210	03131321
		210023	10202023	03112023	23001123	10313123
		210032	10202032	03112032	23001132	10313132
		210203	10202203	03112203	23001312	10313312
		210230	10202230	03112230	23001321	10313321
		230012	10220023	03132012	23021103	10331123
		230021	10220032	03132021	23021130	10331132
		230201	10220203	03132201	23021301	10331312
		230210	10220230	03132210	23021310	10331321
		302012	12002023	03201123	23110023	12313103
		302021	12002032	03201132	23110032	12313130
		302201	12002203	03201312	23110203	12313301
		302210	12002230	03201321	23110230	12313310
		320012	12020023	03221103	23130012	12331103
		320021	12020032	03221130	23130021	12331130
		320201	12020203	03221301	23130201	12331301
		320210	12020230	03221310	23130210	12331310
		013123	21002023	10203123	30112023	21313103
		013132	21002032	10203132	30112032	21313130
		013312	21002203	10203312	30112203	21313301
		013321	21002230	10203321	30112230	21313310
		031123	21020023	10223103	30132012	21331103
		031132	21020032	10223130	30132021	21331130
		031312	21020203	10223301	30132201	21331301
		031321	21020230	10223310	30132210	21331310
		103123	23002012	10312023	30201123	23113103
		103132	23002021	10312032	30201132	23113130
		103312	23002201	10312203	30201312	23113301
		103321	23002210	10312230	30201321	23113310
		123103	23020012	10332012	30221103	23131103
		123130	23020021	10332021	30221130	23131130
		123301	23020201	10332201	30221301	23131301
		123310	23020210	10332210	30221310	23131310
		213103	30202012	12003123	32001123	30113123
		213130	30202021	12003132	32001132	30113132
		213301	30202201	12003312	32001312	30113312
		213310	30202210	12003321	32001321	30113321
		231103	30220012	12023103	32021103	30131123
		231130	30220021	12023130	32021130	30131132
		231301	30220201	12023301	32021301	30131312
		231310	30220210	12023310	32021310	30131321
		301123	32002012	12310023	32110023	32113103
		301132	32002021	12310032	32110032	32113130
		301312	32002201	12310203	32110203	32113301
		301321	32002210	12310230	32110230	32113310
		321103	32020012	12330012	32130012	32131103
		321130	32020021	12330021	32130021	32131130
		321301	32020201	12330201	32130201	32131301
		321310	32020210	12330210	32130210	32131310

Table 5.1: Incomplete minimal set

$$\lambda_{max} = \lambda_{max_1} + \lambda_{max_2} \tag{5.2}$$

It is therefore straightforward to evaluate λ_{max} when we know the maximum eigenvalues of $\mathbf{D}_1$ and $\mathbf{D}_2$. Our balanced QPSK code whose RDS limits are

shown in Fig 5.1 can be regarded as the combination of two one-dimensional balanced codes that each restrict the RDS in their dimension to three different values. It is well known that a one-dimensional signalling alphabet limited to three different RDS values has a capacity of $\frac{1}{2}$ and a $\lambda_{max} = \sqrt{2}$. Therefore, our two-dimensional RDS constraint has $\lambda_{max} = \lambda_{max_1} + \lambda_{max_2} = 2\sqrt{2}$, and $C = \log_2\left[2\sqrt{2}\right] = \log_2\left[8^{\frac{1}{2}}\right] = \frac{3}{2}$ bits of information per quaternary symbol. Note that $\frac{3}{2}$ is the capacity of the constraint, which is also the capacity of the complete minimal set rather than the incomplete minimal set. For an incomplete minimal set, we cannot use Equation 5.2 to calculate the maximum amount of information per coded symbol because not all constraint-satisfying sequences are used. However, the enumeration approach proposed by Immink [3] is an appropriate way to evaluate the upper limit since we already know the number of words in each incomplete minimal set. His approach results in the characteristic equation:

$$2^{l_{max}} z^{-l_{max}} + 2^{l_{max}-2} z^{-(l_{max}-2)} + ... + 4z^{-2} - 1 = 0 \qquad k = 1, 2, ..., K \quad (5.3)$$

The logarithm of the largest real solution for z that satisfies this equation provides us with the upper bound for code rate, $\widehat{C}$, that is possible to obtain with an incomplete minimal set. In order to distinguish it from the capacity of the complete minimal set, we use $\widehat{C}$ to represent this quantity.

The capacities $\widehat{C}$ of our system with incomplete minimal sets when l_{max} = 4, 6, and 8 are listed in Table 5.2. This table also presents the maximum possible efficiency $\eta_{max} = \frac{\widehat{C}}{C}$ for a code constructed with an incomplete minimal set, compared to the capacity $C = \frac{3}{2}$ of the constraint.

l_{max}	4	6	8
$\widehat{C}$	1.3471	1.4396	1.4734
η_{max}	0.8981	0.9597	0.9823

Table 5.2: Capacity of incomplete minimal set

5.2.3 Final Code Selection

With the knowledge of the upper bound, we can construct variable-length balanced QPSK codes by applying normalized geometric Huffman coding. However, similar to the case of M-ary constrained codes, different partial extensions will lead to different mapping between codewords and source words. It is therefore necessary for us to select the code with the highest average code rate. To limit the complexity of our search, we limit the number of codewords and the codeword length during extension of different incomplete minimal sets with $l_{max} = 4, 6, 8$ respectively.

Given these limitations, Table 5.3 lists parameters of the codes that we found to have the highest code rate. In this table, R_{best} and $\eta_{best} = \frac{R_{best}}{\widehat{C}}$ represent the best code rate and best efficiency of codes we constructed starting with each different incomplete minimal set. L_c denotes the maximum length of codewords in the code and L_s is the maximum length of the source words. Finally, Num_{cw} denotes the number of words in highest rate code, and W is a parameter related to the spectral performance of the code, as noted below.

Tables 5.4 and 5.5 list the mapping of source word and codeword lengths for the highest rate codes generated for incomplete minimal sets with $l_{max} = 4$ and $l_{max} = 6$. We don't list the source word to codeword mappings for larger codes because of their large number of words.

l_{max}	4	6	8
R_{best}	1.3375	1.4296	1.4669
η_{best}	0.9929	0.9931	0.9956
L_c	8	10	12
L_s	11	14	17
Num_{cw}	153	665	2713
W	1.543	2.312	2.666

Table 5.3: Optimal metrics of variable-length codes

The optimal variable-length codes l_{max}=4					
R_{best} = 1.338					
η_{best} = 0.892					
L_c	L_s	L_c	L_s	L_c	L_s
2	3	8	11	8	11
2	3	8	11	8	11
2	3	8	11	8	11
4	5	8	11	8	11
4	5	8	11	8	11
4	5	8	11	8	11
4	5	8	11	8	11
4	5	8	11	8	11
4	5	8	11	8	11
4	5	8	11	8	11
4	5	8	11	8	11
4	5	8	11	8	11
4	5	8	11	8	11
4	5	8	11	8	11
4	5	8	11	8	11
4	5	8	11	8	11
4	6	8	11	8	11
6	8	8	11	8	11
6	8	8	11	8	11
6	8	8	11	8	11
6	8	8	11	8	11
6	8	8	11	8	11
6	8	8	11	8	11
6	8	8	11	8	11
6	8	8	11	8	11
6	8	8	11	8	11
6	8	8	11	8	11
6	8	8	11	8	11
6	8	8	11	8	11
6	8	8	11	8	11
6	8	8	11	8	11
6	8	8	11	8	11
6	8	8	11	8	11
6	8	8	11		
6	8	8	11		
6	8	8	11		
6	8	8	11		
6	8	8	11		
6	8	8	11		
6	8	8	11		
6	8	8	11		
6	8	8	11		
6	8	8	11		
6	8	8	11		
6	8	8	11		
6	8	8	11		
6	8	8	11		
6	8	8	11		
6	8	8	11		
6	8	8	11		
6	8	8	11		
6	8	8	11		
6	8	8	11		
6	8	8	11		
6	8	8	11		
6	8	8	11		
6	8	8	11		
6	8	8	11		
8	11	8	11		
8	11	8	11		
8	11	8	11		

Table 5.4: The optimal word length mappings for l_{max} =4

The optimal variable-length codes $l_{max} = 6$																					
$R_{best} = 1.430$																					
$\eta_{best} = 0.954$																					
L_c	L_s	L_c	L_s	L_c	L_s	L_c	L_s	L_c	L_s	L_c	L_s	L_c	L_s	L_c	L_s	L_c	L_s	L_c	L_s	L_c	L_s
2	3	6	9	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14
2	3	6	9	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14
2	3	6	9	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14
2	3	6	9	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14
4	6	6	9	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14
4	6	6	9	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14
4	6	6	9	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14
4	6	6	9	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14
4	6	6	9	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14
4	6	6	9	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14
4	6	6	9	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14
4	6	6	9	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14
4	6	6	9	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14
6	8	6	9	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14
6	8	6	9	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14
6	8	6	9	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	6	9	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	6	9	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	6	9	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	6	9	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	6	9	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	6	9	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	6	9	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	6	9	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	6	9	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	6	9	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	6	9	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	6	9	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	6	9	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	6	9	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	6	9	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	6	9	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	6	9	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	6	9	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	6	9	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	6	9	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	6	9	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	6	9	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	6	9	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	6	9	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	6	9	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	6	9	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	8	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		
6	9	8	11	8	11	10	14	10	14	10	14	10	14	10	14	10	14	10	14		

Table 5.5: The optimal variable-length codes $l_{max} = 6$

5.3 Spectral Analysis

As shown in the previous section, it is possible to use our approach to construct variable-length balanced QPSK codes with a code rate within two percent of capacity. However, excellent spectral performance is also desired. The purpose

of this section is to present spectral results for our codes.

We calculated the power spectral density of our baseband equivalent coded sequences through simulation, in each case considering three million codewords. We also obtained values for low frequency spectral weight (LFSW) W through simulation, where values of LFSW accurately describe the power spectral density [PSD] at low frequencies [19] according to the following relationship.

$$\begin{aligned} \text{PSD} &= W\omega^2, \qquad \omega \ll 1 \\ \text{PSD[dB]} &= 10\log \text{W} + 20\log\omega, \qquad \omega \ll 1 \end{aligned} \tag{5.4}$$

The resulting spectra are given in Fig. 5.6, and values of LFSW W were reported above in Table 5.3:

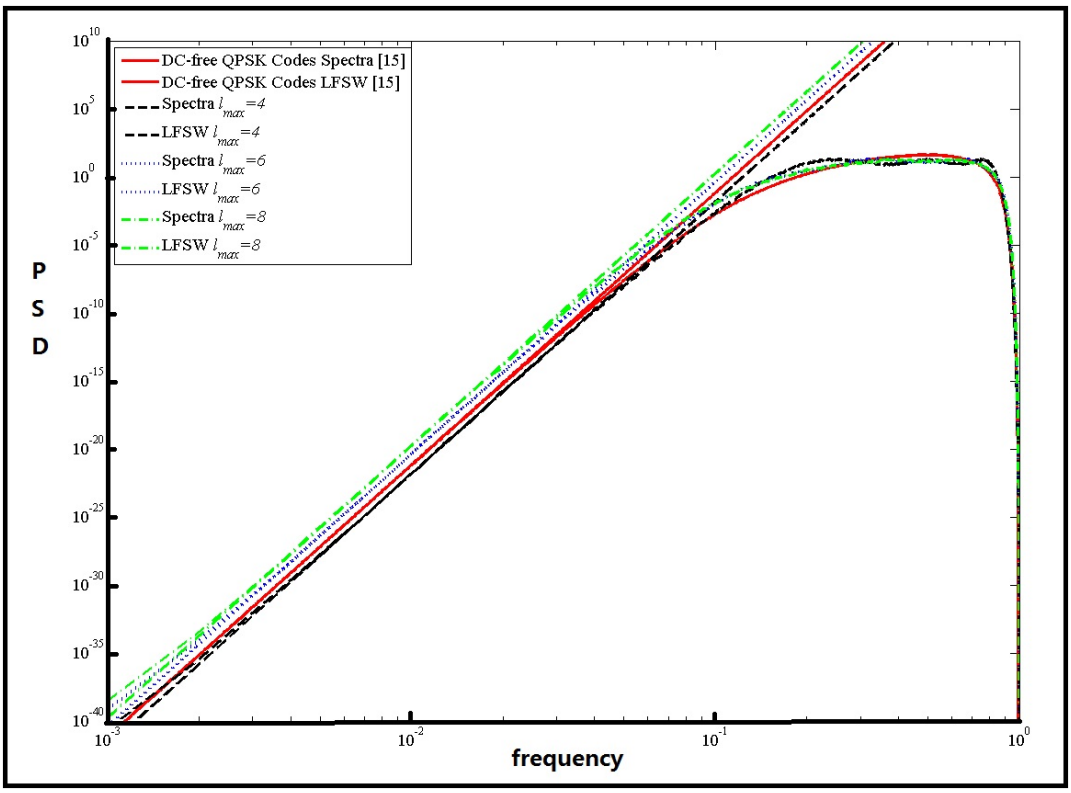

Figure 5.6: PSD of different incomplete minimal set

Comparing the curves in Fig. 5.6 and the values of W, it is clear that the suppression of low frequency components degrades as l_{max} increases, which is

opposite to the increase of code rate. As discussed above, the parameter W represents the performance of low-frequency spectrum where the lower values of W indicate better performance. The visual reflection of this better performance are lower straight-line LFSW approximations in Fig. 5.6.

Also, note that the best fixed-rate code developed to date that satisfies the DC constraint in Fig. 5.1 has a lower code rate ($R = \frac{4}{3}$ [3]) than any of the variable-length codes reported in this Chapter. It also has worse spectral performance ($W = 1.925$) than the variable-rate code with $l_{max} = 4$ and provides only 1.6 dB additional suppression of low frequencies over our variable-length code that has a code rate within two percent of capacity.

Chapter 6

Conclusion

In this chapter we summarize the content and contributions of this thesis and suggest directions for future research.

6.1 Thesis Summary

The purpose of this thesis is to extend the recently developed technique for the construction of variable-length constrained sequence codes to the design of variable-length RLL codes for multi-level magnetic recording systems and the design of variable-length balanced codes for QPSK transmission systems. As shown in this thesis, it is possible in both cases to design codes with code rates within 1% of capacity.

In Chapter 2, fundamental concepts of information theory and constrained sequence codes were introduced. Constrained sequence coding is a channel coding technique that is widely used in optical storage systems and digital transmission systems. In contrast to error control codes, a CS code focuses on error prevention rather than error detection and error correction. It takes arbitrary sequences as its input and converts them into sequences of symbols that satisfy the constraint of the channel. Examples of the most widely used constrained sequence codes include RLL codes and DC-free codes. RLL codes

limit the length of like-valued symbols so that the probability of accurate detection is increased. DC-free codes ensure the presence of many transitions and an equal number of different symbols. These codes guarantee the presence of sufficient timing information for symbol-level synchronization at the receiver and allow transmission of the user data over an AC coupled channel.

Most constrained sequence codes have been constructed as block codes. However, variable-length codes have the advantage that high code rates can be achieved when the variable-length codewords occur with probability close to their maxentropic probability.

The process of constructing variable-length codes can be described with four steps: ① building the model of the constraint of the channel and finding the corresponding minimal set; ② using partial extensions to generate sets of possible codewords; ③ using nGHC to find the optimal one-to-one mapping between source words and code words for each partial extension; ④ selecting the mapping with the highest code rate as the final code selection.

In Chapter 3, several existing fixed-length non-binary codes were presented. Because of their higher-order signalling alphabet, these non-binary codes can result in a high data rate. However, they also have some deficiencies. As noted above, their code rate can be considerably below capacity because of their fixed-length characteristics. For example, a large number of capacities for different constraints were listed in Table 3.1, but fixed-length codes with high code rate exist for very few of these constraints. A tabular method to construct DC-free QPSK codes has also been reported, but the fixed-length codes that have been constructed to meet that constraint have a code rate 12% below capacity. Therefore, we construct variable-length non-binary codes in Chapters 4 and 5.

Variable-length coding enables the design of codes with an average code rate very close to capacity. As shown in this thesis, we generate multi-level RLL codes for magnetic recording systems that satisfy all the constraints from Table

3.1, and all of our codes have an average code rate within 1% of capacity. In addition, we also generated DC-free codes for QPSK transmission systems with spectral performance close to or exceeding the best fixed-rate codes published to date, with code rates higher than those of the best publised fixed-rate codes. The complexity of our codes is also lower than previously published fixed-length codes.

6.2 Thesis Contribution

The primary contribution of this thesis has been to demonstrate the use of variable-length coding for non-binary systems. This contribution can be divided into several parts. ① Through the use of directed graphs, we successfully modeled the system constraints and found the minimal sets of multi-level RLL codes and DC-free QPSK codes. In particular, we found a simple replacement method to determine the incomplete minimal set for DC-free QPSK codes. ② We calculated the capacities of multi-level RLL codes and DC-free QPSK codes, and on this basis we generated a large number of codes satisfying a variety of non-binary constraints. We therefore demonstrated that variable-length coding can be used to construct efficient non-binary constrained codes. ③ Through comparison with reported fixed-length codes, we found that our variable-length codes have not only a higher code rate but also are less complex than the best fixed-rate codes published to date. This indicates that our variable-length codes enjoy considerable advantages over fixed-length coding methods.

6.3 Future Work

In this section, we suggest areas that may be taken into account in future research.

6.3.1 Reducing the Error Propagation

When errors arise on the discrete channel, decoders of constrained sequence codes typically multiply these errors during decoding. Block codes limit the extent of this error propagation to within one fixed-length word, however it is not as straightforward for variable-length codes to achieve this goal. In this thesis, we have not examined the impact of channel errors on the decoding of our instantaneously-decodable variable-length codewords.

6.3.2 Simplifying the Construction of Partial Extensions

Constructing and examining partial extensions is a crucial step in the design of our variable-length codes. However, it can prove difficult for us to find the optimal partial extension by applying an exhaustive search because an exhaustive search requires considerable computing and storage resources. This is of particular importance for non-binary codes because their minimal sets can be much larger than those of binary codes, which translates into the need for considerable resources. It would prove helpful to develope a technique that determines the optimal partial extension without an exhaustive search, as that would significantly reduce the computational complexity of the code design.

6.3.3 Extending the Complex RDS Constraints in QPSK Transmission Systems

In the construction of DC-free QPSK codes, we proposed a new replacement method to find incomplete minimal sets. However, this method is only suitable for the case that the RDS is limited to the nine values depicted in Fig. 5.1. Once the RDS bounds exceed this range, this method is no longer directly applicable. However, after further analysis, we found that the general principle of this method can be applied to larger RDS bounds, which means that we can extend our replacement method when the scope of the RDS bounds is increased.

To do so, we need to divide the new RDS bounds into different regions, and from each region, determine unique ways to return to the origin. We can then develop a new replacement method that represents each unique way to return to the origin with a corresponding symbol sequence. However, further research is still required in order to make it clear how this approach can be applied to other complex-valued signaling transmission systems.

6.3.4 Considering the Non-ideal Source

For the variable-length coding technique used in this thesis, one of the underlying assumptions is that the source symbols are indepedent and identically distributed. However, in many real-world systems, there exists some redundancy in the source symbol stream which makes source symbols and source words correlated. Although source symbols can be randomized through scrambling, it remains to be determined the effects of this randomization on non-ideal sources. It is therefore worthwhile to consider how to apply our variable-length coding technique in a system that contains a non-idea source.

Bibliography

[1] C. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, no. 4, pp. 623–656, July 1948.

[2] K. W. Cattermole, "Principles of digital line coding," *International Journal of Electronics*, vol. 55, no. 1, pp. 3–33, 1983.

[3] K. A. S. Immink, *Codes for mass data storage systems, 2nd ed.* Shannon Foundation Publisher, 2004.

[4] K. Immink, "EFM coding: squeezing the last bits," *IEEE Transactions on Consumer Electronics*, vol. 43, no. 3, pp. 491–495, Aug 1997.

[5] A. Steadman and I. Fair, "Variable-length constrained sequence codes," *Communications Letters, IEEE*, vol. 17, no. 1, pp. 139–142, January 2013.

[6] K. A. Schouhamer Immink, "Constructions of almost block-decodable runlength-limited codes," *IEEE Transactions on Information Theory*, vol. 41, no. 1, pp. 284–287, Jan 1995.

[7] P. Funk, "Run-length-limited codes with multiple spacing," *IEEE Transactions on Magnetics*, vol. 18, no. 2, pp. 772–775, Mar 1982.

[8] P. Franaszek, "Sequence-state methods for run-length-limited coding," *IBM Journal of Research and Development*, vol. 14, no. 4, pp. 376–383, July 1970.

[9] P. Bender and J. Wolf, "A universal algorithm for generating optimal and nearly optimal run-length-limited, charge-constrained binary sequences," in *1993 IEEE*

International Symposium on Information Theory, 1993. Proceedings., Jan 1993, pp. 6–6.

[10] S. Aviran, P. Siegel, and J. Wolf, "An improvement to the bit stuffing algorithm," *IEEE Transactions on Information Theory*, vol. 51, no. 8, pp. 2885–2891, Aug 2005.

[11] A. Steadman, "Variable-length constrained sequence codes," Master's thesis, University of Alberta, 2011.

[12] G. Bocherer and R. Mathar, "Matching dyadic distributions to channels," in *Data Compression Conference (DCC), 2011*, March 2011, pp. 23–32.

[13] G. Bocherer, "Geometric Huffman coding," Available: http://www.georg-boecherer.de/ghc.html, Aug 2011.

[14] A. Calderbank, R. Laroia, and S. McLaughlin, "Coded modulation and precoding for electron-trapping optical memories," *IEEE Transactions on Communications*, vol. 46, no. 8, pp. 1011–1019, Aug 1998.

[15] S. McLaughlin, "Improved distance m-ary (d, k) codes for high density recording," *IEEE Transactions on Magnetics*, vol. 31, no. 2, pp. 1155–1160, March 1995.

[16] A. Kumar and K. A. Schouhamer Immink, "Design of close-to-capacity constrained codes for multi-level optical recording," *IEEE Transactions on Communications*, vol. 57, no. 4, pp. 954–959, April 2009.

[17] C. French, G. Dixon, and J. Wolf, "Results involving (d, k) constrained M-ary codes," *IEEE Transactions on Magnetics*, vol. 23, no. 5, pp. 3678–3680, Sept 1987.

[18] K. A. Schouhamer Immink, J.-Y. Kim, S.-W. Suh, and S. K. Ahn, "Extremely efficient DC-free RLL codes for optical recording," *IEEE Transactions on Consumer Electronics*, vol. 47, no. 4, pp. 910–914, Nov 2001.

[19] I. Fair and C. Jamieson, "Tabular construction of balanced codes," *Electronics Letters*, vol. 49, no. 16, pp. 997–999, Aug 2013.

[20] I. Fair and D. Martin, "Generation of balanced quadrature phase shift keyed sequences through guided scrambling," *IEEE Transactions on Communications*, vol. 9, no. 11, pp. 1404–1411, 2015.

[21] C. Jamieson and I. Fair, "Construction of constrained codes for state-independent decoding," *IEEE Journal on Selected Areas in Communications*, vol. 28, no. 2, pp. 193–199, February 2010.

[22] C. Jamieson, I. Nikolaidis, and I. Fair, "Improved algorithm for constructing constrained codes with state-independent decoding," *Communications Letters, IEEE*, vol. 15, no. 3, pp. 272–274, March 2011.

[23] G. Cariolaro and G. Tronca, "Spectra of block coded digital signals," *IEEE Transactions on Communications*, vol. 22, no. 10, pp. 1555–1564, Oct 1974.

[24] Y. Xin and I. Fair, "A performance metric for codes with a high-order spectral at zero frequency," *IEEE Transactions on Information Theory*, vol. 50, no. 2, pp. 385–394, Feb 2004.

[25] I. Fair, W. Grover, W. Krzymien, and R. MacDonald, "Guided scrambling: a new line coding technique for high bit rate fiber optic transmission systems," *IEEE Transactions on Communications*, vol. 39, no. 2, pp. 289–297, Feb 1991.

[26] B. G. Lee and S. C. Kim, *Scrambling techniques for digital transmission*. Springer-Verlag, 1994.

[27] K. A. Schouhamer Immink, P. Siegel, and J. Wolf, "Codes for digital recorders," *IEEE Transactions on Information Theory*, vol. 44, no. 6, pp. 2260–2299, Oct 1998.

[28] C. Jamieson and I. Fair, "Evaluation of the capacity of constrained codes with multiple constrained signalling dimensions," *IEEE Transactions on Communications*, vol. 8, no. 13, pp. 2238–2245, September 2014.

[29] R. Bellman, *Introduction to Matrix Analysis*. Siam, 1997.