University of Alberta

State-Independent Decodable DC-Free Codes with Complex-Valued Signalling Alphabets

by

Craig Jamieson

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Communications

Department of Electrical and Computer Engineering

Fall 2011
Edmonton, Alberta

To my wife, Athena,
for her constant love and support.

Abstract

Traditionally, constrained sequence coding has been employed exclusively in codes using binary or multilevel signalling. This thesis extends the procedure for constructing DC-free constrained sequence codes to alphabets that use signalling constellations with complex-valued symbols. In particular, the codes are separated into two types: i) constraints with independent logical signalling dimensions and ii) constraints using dependent logical signalling dimensions. In both cases, constraint modelling for the purpose of the evaluation of capacity is explored. Within the case of dependent signalling dimensions, the state machines can contain a finite or an infinite number of states, depending on the signalling constellation that is used. Evaluation of capacity of these types of constrained systems is considered in detail.

Building upon the capacity analysis and constraint modelling techniques, DC-free codes using complex-valued signalling constellations are constructed. The three constellations that are considered in detail in this thesis are quadrature phase shift keying (QPSK), 8 phase shift keying (PSK), and 16 quadrature amplitude modulation (QAM). A number of codes have been constructed for each of the three types for various RDS spans. Further, it is shown that Justesen's relationship for codes using binary-valued symbols, which relates the value of the sum variance to the width of the spectral notch around DC, also holds for codes using complex-valued symbols.

To complete the code construction procedure for DC-free codes using complex-valued signalling alphabets, which involves state-based encoding and decoding, an algorithm was developed to construct codes that can be decoded at the receiver without requiring state information. This algorithm was designed to execute in polynomial time with respect to the size of the input and to be both general and

flexible, so that it can operate on any family of constrained sequence codes. In addition to codes with complex-valued symbols, a number of binary DC-free RLL block codes are constructed using this algorithm, achieving the maximum possible code rates with codeword lengths less than 20. This algorithm is also extended to include the principles of weakly constrained coding.

Acknowledgements

I would first like to thank my supervisor, Dr. Ivan Fair, for his support, guidance, and many suggestions throughout the course of my research.

I would like to express my appreciation for the scholarships I received from National Sciences and Engineering Research Council (NSERC) and the Alberta Informatics Circle of Research Excellence (iCORE). This work would not have been possible without their generous support.

I extend my thanks to all of the faculty members who have taught and assisted me, especially Dr. Witold Krzymień, Dr. Vincent Gaudet, Dr. Hai Jiang, and Dr. Ioanis Nikolaidis for their insightful comments on my research as the members of my examining committee.

Finally, I am grateful to my family and friends for extending their patience, support and love to me. Without them, this work would never have come into existence.

Edmonton, Alberta — Craig Jamieson
July 9, 2011

Table of Contents

List of Tables

List of Figures

List of Symbols

A_u	matrix specifying encoder output function for sourceword u
B	size of base set
C	capacity
C_X	auto-covariance of a random process
C_x	discrete-time auto-covariance function
D	connection (adjacency) matrix
d	$d+1$ specifies the minimum runlength in RLL codes
d_C	Cartesian distance between two points
E	expected value
E_u	matrix specifying valid state transitions for sourceword u
$F(w,P)$	set of states within P from which codeword w can emanate
f	encoder function that determines next state based on current state and output
G	matrix that assists in calculation of R_k
g	encoder function that determines next state based on current state and input
H	entropy function
H_X	power spectral density of a random process
H_x	discrete time power spectral density
h	encoder function that determines output word
I	identity matrix

K	number of dimensions
k	$k+1$ specifies the maximum runlength in RLL codes
k_c	complementation index using Knuth's complementation method
L	total number of states
l_d	lookahead depth
M	number of sequences
M_X	mean of a random process
M_x	mean of discrete time process
m	source word length
N	digital sum variation
N_s	number of sequences
n	codeword length
P	set of principal states
p	probability
Q	transition probability matrix
Q_∞	matrix containing steady-state transition probabilities of each state across each row
q	size of alphabet in codes with multilevel alphabet
R	code rate
R_k	discrete time auto-correlation coefficients in matrix form
R_X	auto-correlation of a random process
R_x	discrete time auto-correlation function
r_f	exponent controlling how constraint is measured for filled score
r_o	exponent controlling how constraint is measured for overlap score

$r_{s(t)}$ auto-correlation of shaping pulse $s(t)$

S size of search set

S_f filled score

S_o overlap score

S_t overall score

$s(t)$ shaping pulse used in communication track

s_z^2 variance of running digital sum

T period

T_b period of bit

$\mathcal{T}$ transpose

t index indicating time

U set of states

u state index

V set of states

v state index

W set of words, or alphabet for a state

W_l number of words left in search rows of a given column

W_s number of spaces remaining in base rows of a given column

w codeword

X finite set of sequences

X_o size of output set

X_t stochastic process

x bit, sequence of bits

y state index

Z	Markov random variable
z	running digital sum
β	source word
Γ	source alphabet
Δ	uniformly distributed random variable
ΔZ	largest RDS difference between any two valid RDS values
ΔZ_i	RDS span in imaginary dimension
ΔZ_r	RDS span in real dimension
ζ	output function of Markov information source
$\lambda_{\max}$	maximum eigenvalue of connection matrix
ν	code efficiency
Π	diagonal matrix with steady-state transition probabilities of each state on its diagonal
π	steady-state probability of state
Σ	set of states
σ	state
τ	time difference
χ	output word
$\psi(\sigma_i, P)$	row sum for a possible principal state
Ω	set of indices corresponding to non-eliminated principal states
ω	frequency variable

List of Abbreviations

BD BluRay disc

CD compact disc

DSV digital sum variation

DVD digital versatile disc

ECC error control codes

EFM eight-to-fourteen modulation

FFT fast Fourier transform

FSK frequency shift keying

FSM finite state machine

ISI intersymbol interference

LFSW low frequency spectrum weight

NRZ non-return-to-zero

NRZI non-return-to-zero inverse

OFDM orthogonal frequency division multiplexing

PAPR peak-to-average power ratio

pdf probability density function

PSD power spectral density

PSK phase shift keying

QAM quadrature amplitude modulation

QPSK quadrature phase shift keying

RDS running digital sum

RLL runlength-limited

Chapter 1

Introduction

The field of digital communications has experienced tremendous growth in the past few decades. This growth can be attributed to the emergence of new hardware technologies and advancements in signal processing, which have enabled high data rate transmission over a variety of mediums including optical fibre, satellites, and radio links. Communication takes place not only across space, however, as in the conventional channels listed above, but also from one time to another, such as in recording systems. In both cases, the use of coding and the application of information theory, motivated by Shannon's work [1], has led to vast improvements in communication systems and stimulated great interest in the field. In coding systems, an encoder performs the task of translating a block of user data, typically a binary sequence, into another symbol sequence. The goal of the encoding procedure is to improve the performance of the system, typically by making the system less prone to errors occurring during transmission and detection. The focus of this thesis is on the specific area of coding referred to as line coding or constrained sequence coding.

Constrained modulation codes used in data storage systems transform blocks of m source symbols into blocks of n coded symbols in a lossless manner such that the encoded bit sequence satisfies specific constraints. These codes ensure satisfactory performance despite limitations of the channel, encoding circuitry, and/or decoding circuitry in digital storage and transmission systems [2]. The set of words from which the code sequences are selected is referred to as a constrained system. For block codes, the code rate, given by $R = \frac{m}{n}$, is a measure of the amount of information a code conveys, typically specified binary digits per symbol. The quantity $1 - R$ is referred to as the redundancy of the code.

To better describe the purpose of a constrained code, a few common applications will be discussed. Constrained codes are widely used in recording systems, and for this reason are sometimes referred to as recoding codes. Storage devices today, such

as hard drives, compact disc (CD) drives, digital versatile disc (DVD), and BluRay disc (BD) drives all employ, or have employed, some form of constrained codes. In fact, advances in storage capacities can be attributed not only to advances in hardware technology, but to improvements in constrained codes. As an example of a constrained code, consider the eight-to-fourteen modulation (EFM) [3] code used in CD players. Data is written to the disc using pits and lands to represent binary zeros and ones. The decoding circuity exhibits better performance if the durations of the pits and lands are not too short or too long. Consequently, a runlength-limited (RLL) code is employed to limit both the minimum and maximum number of consecutive ones and zeros that can be written. Some redundancy is required to enforce runlength limitations, which lowers the storage efficiency, but the improved performance of the decoding circuitry allows more data to be packed onto the disc, resulting in a net increase in storage density. While constrained coding has been traditionally deployed in the recording industry, there are also applications in other digital communication systems, such as the peak-to-average power ratio (PAPR) problem in orthogonal frequency division multiplexing (OFDM) systems, or the ubiquitous scramblers used in most transmission systems.

Digital symbols are represented by physical quantities in order to be written to a physical medium for storage or to be passed through a communication channel. In digital communication systems, the symbols are assigned to one of a finite set of continuous waveforms. These continuous waveforms are, for example, sinusoids in wireless communication systems or pits and lands in optical storage systems. When a single pulse is transmitted in a bandwidth-limited system, convolution of the pulse with the impulse response of the channel can result in the spreading of the pulse over several signalling intervals [2]. This spreading is referred to as inter-symbol interference (ISI) and can be the limiting factor, rather than additive-type noise, in recording systems. While ISI can be reduced or removed using a linear filter at the receiver, there are a number of difficulties in recording systems that prevent the straightforward application of filtering [2]. Rather than employing complex adaptive equalization, a constrained code can be employed. For example, the minimum runlength in a code controls the highest transition frequency in the transmitted waveform and thus directly influences the effects of the ISI over a bandlimited channel.

The guiding principle behind most constrained codes is that particular sequences of bits are difficult or impossible for the communication track to handle. These sequences result in an erroneous version of the user data being detected. Constrained

coding aims to remove these vexatious sequences from the codebook, improving the robustness of the communication track. Widely used classes of constrained codes include RLL codes and balanced codes [2]. An RLL code imposes constraints on the minimum and maximum runlengths allowable for consecutive 0's and 1's within the coded sequence; this can be thought of as a constraint in the time domain. Balanced codes generate sequences of bits with an equal number of 1's and 0's. These sequences have a null at DC in their continuous spectral component, and therefore are referred to as DC-free codes; this can be thought of as a constraint in the frequency domain. It should be noted that while constrained codes can be developed to satisfy many spectral constraints [4], the most common requirement is that the code have no spectral content at DC.

A common procedure for encoding constrained sequence codes is to use the current accumulated running digital sum (RDS) value or run-length as state information. This state information is modelled using a finite-state machine (FSM) with states, edges, tags, and labels. DC-free and RLL codes, along with methods of code construction, are covered in more detail in the next chapter. Further, some variations of these two types of codes, including the codes satisfying both constraints, employing multi-level signalling, or signalling on a two dimensional medium are discussed in Chapter 2.

1.1 Thesis Objective

In this thesis, two major areas of constrained coding are addressed. For the first major area, this thesis aims to extend existing theory of DC-free constrained codes to the case where the signalling alphabet is larger than binary and, in particular, uses a complex-valued signalling alphabet. The construction of codes using complex-valued signalling alphabets requires knowledge of capacity and the development of constraint modelling techniques. Further, practical constrained codes have an additional requirement that the code is decodable without requiring state information, so that error propagation at the decoder is limited. Prior to the work in this thesis, no methods were known for the construction of DC-free constrained codes using complex-valued signalling alphabets. To address this, an approximation algorithm for constructing constrained codes that permit state-independent decoding is the second major area that is considered. The application of this algorithm is not limited to DC-free constrained codes with complex-valued signalling alphabets; instead, the algorithm is designed to function on any type of constrained code

and includes a number of parameters than can be adjusted based on the particular family of codes.

For typical applications of DC-free and RLL codes, the communication system uses only binary valued symbols since the channel or medium supports only two values. However, in this thesis, the case where a larger, complex-valued alphabet is used for signalling is investigated, opening up the possibility for using constrained codes in more diverse applications including, for example, those employing phase shift keying (PSK) or quadrature amplitude modulation (QAM) constellations. While it is possible to use a constrained code on the binary values prior to mapping them onto other symbols for transmission, in general, the constraint will no longer be satisfied. There are some cases where this is possible, for instance by exploiting symmetry in the symbols in the signalling alphabet, however this will not always be possible. For example, in the context of DC-free codes, it is required that the communication system have no frequency content at DC. Typically this is performed by balancing the symbols used for transmission. Balancing a binary sequence prior to mapping it onto PSK symbols will not necessarily mean that the PSK sequence is balanced. In general, balancing a code that uses multiple symbols across two dimensions will be more difficult than balancing a code using binary signalling over a single dimension.

The DC-free constraint is important in optical disc applications for minimizing the effects of fingerprints and other low frequency noise, while the RLL constraints exist in order to facilitate easier bit detection since limitations to the minimum and maximum lengths of a pit or land enable simpler and more reliable demodulation circuitry. However, these types of constraints could potentially be used in any system that employs a larger signalling alphabet. Specifically, controlling the spectral emissions of the communication system is important in a large number of applications, especially in systems that communicate using a shared medium. Further, the system may require the insertion of a pilot tone at a particular frequency and by using a constrained code it is possible to remove all information signal content at the desired frequency. Maximum RLL constraints are used to force transitions in the data stream, for example, to aid in the recovery of timing information or synchronization.

In this thesis, the DC-free constraint using complex-valued signalling alphabets is explored. This constraint is modelled through the use of multiple constrained signalling dimensions. Examples are provided using QPSK, 8 PSK, and 16 QAM signalling alphabets where the encoder divides the signalling constellation into a

number of independent logical dimensions and enforces the DC-free constraint on each logical dimension separately. While these signalling constellations are used as examples, this work is also more general, and the principles can be applied to other forms of communication where the signalling has some form of independence, such as frequency shift keying (FSK).

To design DC-free constrained codes using complex-valued signalling alphabets, important concepts from constrained codes using binary valued signalling are extended. Typically, the constraint is modelled as an FSM and the capacity of the constraint is evaluated. Capacity, in particular, is important because it provides an upper limit on the rate that is achievable when designing the code. In general, for a particular constraint, the code designer should aim for the highest rate possible, bounded by the maximum theoretical capacity, since this means that the coded sequences convey the most information. While it is sometimes possible to design a constrained code without the use of an FSM, for example, by enumerating the total number of sequences that satisfy the constraint, for more complex constraints, which are explored in this thesis, FSM encoder modelling is typically required. By using an FSM, a DC-free code, for example, can ensure that codewords are balanced over a slightly longer period of time than a single codeword interval by having the encoder keep track of relevant state information and concatenating the codewords intelligently. FSM modelling is important because it allows the code designer to have many tools with which to design efficient codes. This includes a relatively simple means for evaluation of capacity, and a straightforward approach to conduct spectral analysis in order to quantify the performance of a code. The approximation algorithm for state-independent decoding that is developed in this thesis implicitly assumes that the constraint has been modelled using an FSM and then partitioned into a table. If the code is simple enough that states are not required for its implementation, then the encoding is performed without state information and so state-independent decoding is straightforward.

1.2 Thesis Organization

The thesis is organized as follows.

Chapter 2 presents the theoretical foundation for constrained codes. The chapter begins by presenting a brief overview of the fundamental idea of information, entropy and capacity. A modelling tool called a Markov chain and the evaluation of its capacity is discussed. Code construction is covered in detail, highlighting a

practical issue: state-independent decoding. The merits of state-independent decoding are highlighted briefly, and an overview of the existing related literature is presented. The spectral analysis of codes is discussed, beginning with discussion of a Markov information source, which serves as a starting point for the spectral analysis of both memoryless block codes and block codes with memory. The chapter ends with a discussion of types of constrained codes that have been considered in the literature. This includes the DC-free and RLL constraints discussed earlier in this chapter, and also constrained codes using multi-level signalling and constrained codes for a two (or more) dimensional medium.

In Chapter 3, the construction of codes that permit state-independent decoding is covered in more detail. The primary focus of the chapter is the development of a approximation algorithm that is able to construct codes that do not require state information at the decoder. Using a representation based on a group of alphabets, the algorithm is able to construct the codebook a single row at a time, using a greedy procedure. Important points of the algorithm, such as optimizing the initial construction of the table, and procedures for constructing the rows are covered. The algorithm has a number of parameters that can be tuned, such as scoring exponents and thresholds, so that it can function for many families of codes. More advanced techniques, such as lookahead and weakly constrained coding, are implemented with the algorithm. Results of applying the algorithm to DC-free RLL codes are presented, including those which employ lookahead or weakly constrained coding.

In Chapter 4, the evaluation of capacity and constraint modelling of DC-free constrained codes using multiple constrained signalling dimensions is considered. In particular, the chapter considers the DC-free constraint for codes using quadrature phase shift keying (QPSK), 8 PSK, or 16 QAM alphabets. Modelling and evaluation of the capacity of these systems is discussed. The signalling constellation is considered as a number of independent signalling dimensions, where the encoder can enforce the DC-free constraint on each of the dimensions separately. These dimensions can then be transformed into a constrained code using dependently constrained signalling dimensions by having the encoder re-consider the overall DC-free constraint. Using this procedure, the constraint modelling and capacity calculations are simplified.

In Chapter 5, the construction of DC-free constrained codes using QPSK, 8 PSK, and 16 QAM signalling alphabets is presented. Using the constraint modelling and capacity evaluation from the previous chapter, and the approximation algorithm for the construction of a state-independent decodable code from Chap-

ter 3, a number of DC-free codes are constructed for each signalling alphabet. The performance of these codes, in the form of the power spectral density (PSD), is presented.

Finally, in Chapter 6, the thesis contributions are summarized and suggestions for future work are offered.

Chapter 2

Background

This chapter discusses the theoretical foundation of constrained codes and its related literature. The first portion of the chapter presents the fundamental ideas and tools required in the design of a constrained code. The ordering in which these topics are discussed is similar to that used by the code designer in the process of constructing a typical code, beginning with the idea of information and entropy, then moving to modelling and encoder design. Spectral analysis is highlighted thereafter. In the final portion of the chapter, the most common types of constrained codes are discussed.

An overview of the concepts of information, entropy, and capacity is presented in Section 2.1. Initially, the discussion is restricted to the entropy of information sources without memory in subsection 2.1.1. The concept of a Markov chain is presented in subsection 2.1.2, and the capacity of a Markov information source is discussed in subsection 2.1.3. In Section 2.2, important elements of code design and construction are presented. This includes modelling of constraints with an FSM, its capacity, and the design of an encoder that produces sequences that satisfy the desired constraint, discussed in subsections 2.2.1 through subsection 2.2.4. The concept of state-independent decoding, the focus of the next chapter, is discussed in subsection 2.2.5 and subsection 2.2.6. In Section 2.3, the spectral analysis of constrained modulation codes is presented. A brief discussion of stochastic processes and stationarity is presented in subsection 2.3.1. The spectral analysis of a Markov information source is discussed in subsection 2.3.2, while the spectral analysis of block-coded signals is discussed in subsection 2.3.3. Common types of constrained codes are discussed in Section 2.4. The two most popular constraints, RLL and DC-free constrained codes are presented in subsections 2.4.2 and 2.4.1, respectively. Additionally, codes satisfying both DC-free and RLL constraints simultaneously are presented in subsection 2.4.3. Finally, codes employing multi-level signalling

are discussed in subsection 2.4.4, while codes designed to write sequences onto two (or more) dimensional surfaces are discussed in subsection 2.4.5.

2.1 Information and Entropy

In this section, a brief summary of the fundamental ideas of information and entropy are presented. These concepts and their associated tools provide the code designer with a guide for the construction of their initial design and mechanisms to assess the performance of their overall design. In particular, this section quantifies how much information is conveyed by an information source, which is related to its entropy. The maximization of entropy for a particular channel constraint, called capacity, is also discussed.

There are two major types of information sources, continuous and discrete. In this thesis, the focus is on discrete information sources, which are information sources that convey symbols from a finite set of letters, called an alphabet. Each time the information source emits a message, some information is transmitted. To measure the amount of information that has been conveyed, the concept of entropy is used.

The information source conveys its information across a channel. In this thesis, the channels that are studied are those that do not permit a certain collection of sequences, called forbidden sequences; this is referred to as an input-restricted or constrained channel. As discussed in Chapter 1, certain sequences may be problematic for the detection circuitry, and so the channel restricts the usage of these sequences. The particular types of constraints commonly used in constrained coding are discussed in more detail towards the end of this chapter.

2.1.1 Entropy of a Memoryless Information Source

A memoryless information source transmits symbols that are statistically independent. It is assumed that the particular symbol that is chosen for transmission is selected by the result of a random experiment. Consider a finite set of sequences $X = \{x_1, x_2, \ldots, x_M\}$, with corresponding probabilities $p_1, p_2, \ldots, p_M$, such that $\sum_{i=1}^{M} p_i = 1$. The symbol generated by the source at time t is indicated by X_t. In [1], Shannon develops the idea of the measurement of information, relating it to entropy or uncertainty. The more unlikely a message is, the more information it

contains. Shannon adopted the definition:

$$H(p_1, \ldots p_M) = -\sum_{i=1}^{M} p_i \log p_i, 0 \leq p_i \leq 1 \tag{2.1}$$

to measure the entropy of an information source. Setting the logarithm to base 2 implies that the entropy is measuring the number of bits required to transmit the sequences generated by the source, yielding units for entropy of bits of information per symbol. Other base values for the logarithm can be used, but in this thesis, a value of two is used, as is commonplace in the literature. It can be shown that the entropy function achieves a maximum of $\log_2 M$ bits of information per symbol when the source symbols are equi-probable [5].

2.1.2 Markov Chains

It is often desirable to introduce correlation into a sequence of coded symbols. This allows, for example, some spectral shaping, which is discussed later in this thesis. In many cases, the appearance of a symbol is a function, in part, of the symbols that appeared before it, and so the source is no longer memoryless, nor are the symbols independent. To model these types of information sources, in this subsection the idea of a Markov chain is discussed.

A Markov chain, with L states, is a discrete random process with dependent discrete random variables, Z_t, taking the form $\{\ldots, Z_{-2}, Z_{-1}, Z_0, Z_1, \ldots\}$ from a state alphabet $\Sigma = \{\sigma_1, \sigma_2, \ldots \sigma_L\}$. These dependent discrete random variables satisfy the Markov condition:

$$\Pr(Z_t = \sigma_{i_t} | Z_{t-1} = \sigma_{i_{t-1}}, Z_{t-2} = \sigma_{i_{t-2}}, \ldots) = \Pr(Z_t = \sigma_{i_t} | Z_{t-1} = \sigma_{i_{t-1}}) \tag{2.2}$$

where $\Pr(A|B)$ is the probability of occurrence of event A, given that event B has already occurred. This condition specifies that the variable Z_t is dependent only on the past sample Z_{t-1} and independent of prior variables $\{Z_{t-2}, \ldots\}$ in the Markov chain. The Markov chain can then be described by its transition probability matrix, Q, which specifies the probabilities of transitioning from a state σ_i to another state σ_j. Mathematically, the entries of the matrix are given by:

$$[Q]_{ij} = \Pr(Z_t = \sigma_j | Z_{t-1} = \sigma_i), 1 \leq i, j \leq L. \tag{2.3}$$

Notice that the Markov chain does not have any inputs; it is characterized by the discrete random variable, Z_t and the state transition probabilities.

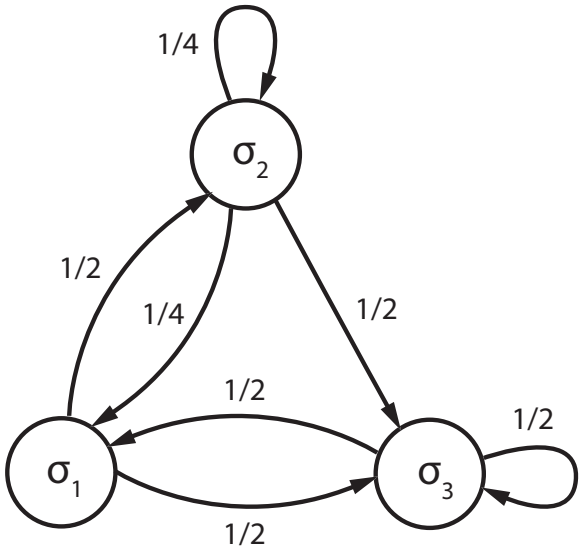

Figure 2.1: Example of a Markov chain represented by a directed graph.

One popular representation of the transition probability matrix of a Markov chain is to use a directed graph. The states form the vertices of the graph, while the edges indicate valid transitions, i.e., transitions from state i to state j where $[Q]_{ij} > 0$. These edges are typically labelled with the value of $[Q]_{ij}$ corresponding to the transitions. For example, for the transition probability matrix

$$Q = \begin{bmatrix} 0 & 1/2 & 1/2 \\ 1/4 & 1/4 & 1/2 \\ 1/2 & 0 & 1/2 \end{bmatrix} \tag{2.4}$$

the directed graph representation is given in Fig. 2.1. While other representations are possible, such as a trellis representation, only the directed graph representation is presented here due to its similarity to the state machine representation discussed in the next section.

In this thesis, only Markov chains that are irreducible and regular, also referred to as ergodic, are considered. A Markov chain is irreducible if from any state the Markov chain can eventually reach any other state (in one or more steps) [6]. Regularity of a Markov chain refers to periodicity; specifically, a Markov chain that is regular is non-periodic [6]. A state in a Markov chain is periodic if that state can only be entered when the time index, t, is multiple of a specified period T. If the largest such value of T is 1, then the Markov chain is non-periodic.

Entropy of Markov Information Sources

In this thesis, an information source with memory is modelled as a Markov information source. Consider a finite Markov chain $\{Z_t\}$ and a function ζ whose domain is the set of states and whose range is the alphabet, Γ. The set of sequences $\{X_t\}$, with $X_t = \zeta(Z_t)$ is the output of a Markov information source corresponding to the chain $\{Z_t\}$ and the function ζ. Since the symbols emitted by a Markov information source are dependent, some redundancy is introduced, as each successive symbol is partially predictable. The Markov information source produces correlated outputs, X_t, based on its current state and the state transition probabilities.

To evaluate the entropy of a Markov information source, a simplification is helpful. In this thesis, only unifilar Markov information sources are considered. A Markov information source is unifilar if for every state, σ_i, the labels of its n_i successor states $\sigma_{i_1}, \sigma_{i_2}, \ldots, \sigma_{i_{n_i}}$ are distinct. A successor state of σ_i is defined as a state that can be reached in a single step with a transition probability greater than zero. For example, in Fig. 2.1, the successor states of state 2 are states 1, 2, and 3, since each can be reached in a single step with probability greater than zero. For the Markov information source to be unifilar, each of these successor states requires a distinct label.

For a unifilar Markov information source the uncertainty of the successor of state σ_i is $H_i = H([Q]_{i,i_1}, [Q]_{i,i_2}, \ldots [Q]_{i,i_{n_i}})$, where $\sigma_{i_1}, \sigma_{i_2}, \ldots, \sigma_{i_{n_i}}$ are the successor states of σ_i. The value of H_i is calculated using the definition of entropy from (2.1). Averaging the entropies of each of the states according to the probability of being in that state gives the overall entropy of the unifilar Markov information source:

$$H\{X\} = \sum_{i=1}^{L} \pi_i H_i. \tag{2.5}$$

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 [5].

2.1.3 Capacity of Constrained Channels

Returning to the input-restricted or constrained channel, a means to measure the entropy of such a channel, or to calculate its capacity, is required. In [1], Shannon defines the capacity, in bits of information per symbol, of a constrained channel as

$$C = \lim_{n\to\infty} \frac{1}{n} \log_2 N_s(n) \tag{2.6}$$

where $N_s(n)$ is the number of sequences of length n allowed by the channel. The units for capacity are in bits of information per symbol since a base two logarithm is used. Evaluation of the capacity of a constrained channel is important because it indicates the upper-bound on the rate of a code that can be constructed that satisfies the channel constraint. However, calculating the maximum number of sequences permitted by a constraint channel is often a complex problem. In the following subsection, another approach is considered.

Capacity of Markov Information Sources

A unifilar Markov source can be characterized in terms of its connection matrix, D. The entries of this matrix, $[D]_{ij}$, are constructed by examining the Markov source and setting $[D]_{ij}$ to be equal to the number of transitions from state i to j, or zero if there is no valid transition from state i to j. These transitions correspond to edges of the directed graph that defines the Markov source. To find the capacity of this Markov information source, the entropy given by:

$$H\{X\} = \sum_{i=1}^{N} p_i H_i \tag{2.7}$$

must be maximized. This is done by choosing the transition probabilities, p_i, such that $H\{X\}$ is maximized. The state transition probabilities that give the maximum entropy for the Markov information source are called the maxentropic state transition probabilities and generate maxentropic sequences. The capacity is then given by:

$$C = \max H\{X\} \tag{2.8}$$

In this thesis, we consider only connection matrices representing strongly connected graphs; therefore, the existence of a positive eigenvalue and corresponding eigenvector with positive elements is guaranteed by the Perron-Frobenius theorems [7]. Shannon [1] showed that $H\{X\}$ is a function of the connections in D. Since the growth factor of the graph described by D is related to the maximum eigenvalue of D, then the maximum number of sequences that can be generated as $m \to \infty$ is related to this maximum eigenvalue. A second method for calculating the capacity is:

$$C = \log_2 \lambda_{\text{max}} \tag{2.9}$$

where λ_{max} is the maximum eigenvalue of D [1]. Similar to before, taking the logarithm to the base of two returns the capacity in bits of information per symbol.

2.2 Code Construction

This thesis focuses on the construction of codes for use with a constrained channel. The constrained modulation codes used in data storage systems transform blocks of m source bits into blocks of n coded bits in a lossless manner such that the encoded bit sequence satisfies specific constraints. The set of words from which the code sequences are selected is referred to as a constrained system. These codes ensure satisfactory performance despite limitations of the channel, encoding circuitry, and/or decoding circuitry in digital storage and transmission systems [2].

The construction and design of constrained modulation codes can be quite involved, and is usually application-specific. Often, the approaches used by code design engineers are ad-hoc, but frequently successful. Despite some elegant mathematics, there is an art to the design of codes. The code designer must be aware of the mathematics and be able to both quantify the trade-offs that are being made and analyze the performance of the code.

While there are many design techniques or approaches used to produce constrained modulation codes, some elements are common to the majority of these approaches. For example, prior to the design of an encoder, it is typical to model the constraint using an FSM, choosing as state variables the important parameters, such as the current run-length or the RDS. With a model for the constraint, the capacity of the code is evaluated (or estimated if exact evaluation is not possible). Capacity serves as an upper bound on the information that it is possible to transmit through the channel, and so provides a guide for the code designer to evaluate their design. Next, an encoder is designed, with values chosen for m and n. The design of an encoder has a significant number of different approaches. This design must satisfy the channel constraints, but may also be required to satisfy other constraints, such as the ability to be decoded at the receiver without the need for state information, which is discussed later in this section. Finally, performance analysis should be completed, so that the various trade-offs that have been made are clear. This could be as simple as comparing the performance of the code relative to a code using maxentropic sequences.

In this section, an overview of an FSM, including a method for calculating their capacity, is provided, along with a discussion of the block-type encoders that are generated from these FSMs. Franaszek's recursive elimination algorithm is then discussed. This procedure is commonly used for determining suitable values for m and n for the block-type encoder, as well as finding a suitable set of encoding states

P. In the remainder of this section, the principles of state-independent decoding are discussed, and their importance is highlighted.

2.2.1 Finite-State Machine Encoder Modelling and Block-type Encoders

In this thesis an encoder is considered as a device with an input, a state, and an output, whose operation can be modelled with an FSM. The FSM models the encoder by using three sets: the inputs, the outputs, and the states, and two logical functions: the output function and the next-state function. For each input to the encoder, the corresponding output is generated according to the output function, which depends on the value of the input and the current state of the encoder. The state of the encoder is updated using the next-state function, which also depends on both the input and the current state of the encoder.

Block-type encoders, which are encoders where both the inputs and outputs are partitioned into blocks, are perhaps the most common and most successful encoders that transform arbitrary data from a user into constrained sequences that satisfy a desired set of properties [2]. In particular, the input is grouped into m symbol blocks, called source words, while the encoder produces a n symbol output, called a codeword, according to the encoder functions. In this thesis, all the codes that have been constructed are restricted to a block-type encoder implementation.

The notation associated with an FSM encoder that will be used in this thesis is as follows. The input set consists of M m-symbol words. A source word in this set is denoted by β_u, $u = 0, 1, \ldots, M-1$. The set of states of size L is denoted by Σ and consists of states $\sigma_i, i = 1, 2, \ldots, L$. The output set consists of X_o n-symbol codewords and are doubly indexed as χ_{iu}, where i corresponds to the state of the encoder and u denotes the source word in the set. The output function of the encoder specifies the mapping $\chi_{iu} = h(\sigma_i, \beta_u)$. In general, recovery of the source word from the codeword requires knowledge of the encoder state. Finally, the next-state function determines which state the encoder enters after mapping a source word into a codeword. If the next-state function is determined on the current state and input, it is called $g()$, however, if it depends on the current state and output, then it is called $f()$. The two functions $g()$ and $h()$, which describe the operation of the FSM, are called the characterizing functions. In this thesis, these functions will be specified in terms of a table, called a transition table, which defines the codebook of the code. Typically, for the codes constructed in this thesis, only the table for $h()$ is included since the states are well-defined, and it is a straightforward exercise to

generate the function $g()$ which determines the next state of the encoder.

Similar to a Markov information source, an FSM encoder is an example of a labelled directed graph. In this thesis, the presentation of a constraint is primarily referred to in terms of a state machine, which is more common in the literature. A graph is typically characterized by a set of vertices, which are referred to as states. This graph consists of a set of edges, which specify valid paths existing between vertices, or states. Graphs may have an edge labeling, which shall be referred to as an alphabet. This edge labeling is often specified using "labels" or "tags." In this thesis, the word tag will be used with input edges, while the word label will be used with output edges. While this thesis generally uses state machine specific terminology, there are times, as indicated in the thesis, where results from graph theory are applied to evaluate the capacity of a constraint or to construct a code.

2.2.2 Types of Block-type Encoders

The FSM encoder description presented in the previous subsection is similar to that of a Markov information source. However, recall that a Markov information source is a machine without inputs, while the block encoders considered in this thesis transform blocks of input symbols into blocks of output symbols and therefore have source words as inputs. When the source words are independent, as they are in all codes considered in this thesis, the sequence of encoder states forms a stationary Markov chain because the next state depends only on the previous state and the input.

While all of the inputs are independent, the process of assigning tags to the input edges and labels to the output edges transforms the Markov chain to an encoder, whose operation can be captured with a codebook. There are a few classifications for block-based encoders that are discussed in this thesis. The most general type is the deterministic encoder, which is an FSM encoder with a deterministic output labeling: each of the outputs is given a specific label, which we refer to as the codeword. A block-decodable encoder is an FSM encoder where any two edges with the same output label have the same input tag. Finally, the most specific type is a block encoder. This is an FSM encoder where any two edges that have the same input tag must also have the same output label.

To illustrate the differences between the three types of encoders, consider the example code based on the state machine presented in Fig 2.2. Assigning labels to the output edges constructs a deterministic encoder. If these output edges are given the input tags according to the Table 2.1, the result is a block encoder. In this table,

the codeword is selected based on the current state σ_j, indicated by a column in the table, and a source word β_u, indicated by a row in the table. For example, if the encoder is in state σ_1 and source word β_1 is selected, then the output word is w_1, the entry in the first row and first column of the table. However, it is not possible to transmit some of the source words in particular states. For example, if the encoder is in state σ_1, source words β_2 and β_3 have no mapping. Typically, a block encoder would be able to transmit the same codeword for a given source word, no matter which state is selected, i.e., each row in the table contains the same w_i across all entries and so both the encoding and decoding do not depend on the current state.

A block-decodable encoder, using the same state machine, is presented in Table 2.2. In this case, each source word is valid in each state, since it is possible to assign two (or more) codewords to the same source word, provided that all edges with the same output label have the same input tag. That is, each codeword can only be listed horizontally in the same row in the table and cannot appear on two different rows. In this way, all block encoders are block-decodable, all of which are deterministic. Later in this section, the idea of block-decodable encoders is examined in more detail to discuss the additional advantages that exists with block-decodable encoders.

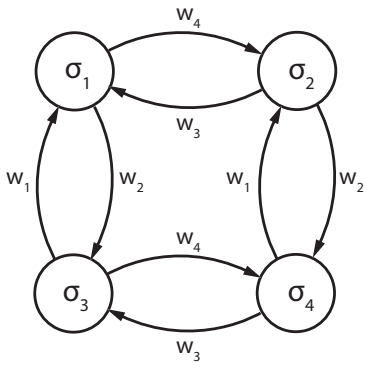

Figure 2.2: Example of a finite state machine.

Table 2.1: Sample block encoder

source	σ_1	σ_2	σ_3	σ_4
β_1	w_1	w_1		
β_2			w_2	w_2
β_3		w_3		w_3
β_4	w_4		w_4	

Table 2.2: Sample block-decodable encoder

source	σ_1	σ_2	σ_3	σ_4
β_1	w_1	w_1	w_2	w_2
β_2	w_4	w_3	w_4	w_3

2.2.3 Capacity of Constraints Modelled by an FSM

For the block-type encoders used in this thesis, the source symbols are always taken from a binary alphabet, while the coded symbols are taken from the particular signalling constellation that is used. For example, in the case of typical constrained codes, such as binary codes enforcing a DC-free or RLL constraint, the coded symbols are taken from $\{0, 1\}$ (or equivalently $\{-1, +1\}$), however, in later chapters, constrained codes are constructed using coded symbols taken from QPSK, 8 PSK, or 16 QAM alphabets. In all cases, a fixed-length block code with words of length m source bits and length n coded symbols has a rate of $R = \frac{m}{n}$ binary digits per coded symbol.

The capacity of a FSM is calculated in the same manner as it was for a Markov information source. The connection matrix, D, (referred to as an adjacency matrix for a graph), is calculated and the maximum eigenvalue λ_{max} of D (the spectral radius of a graph) is evaluated. Using (2.9), the capacity can be evaluated. The capacity of this FSM (and the constrained system it models) is given in units of bits of information per coded symbol.

2.2.4 Franaszek's Recursive Elimination Algorithm

For a given FSM that models a particular constraint, Franaszek's recursive elimination algorithm [8] finds a suitable set of principal states, $P = \{\sigma_1, \sigma_2, \ldots\}$, for a specific set of code parameters (m and n) and can be used as a basis to find a near-capacity achieving code for this constraint. A principal state is a state in which the FSM can exist at the end (or beginning) of each codeword. This set comprises a subset of the original channel states of the FSM since some states might only occur at other positions within codewords. Franaszek's algorithm recursively eliminates states in the FSM that have fewer than 2^m outgoing edges until a suitable set of states is found such that each remaining state has a minimum of 2^m outgoing edges. For codewords of length n, the code designer is interested in how many paths of length n exist between two states. The nth power of the adjacency matrix, D^n, provides this information. To ensure that each source word has a representation in each

state, a valid code exists provided that each state has at least 2^m transitions or edges leaving it, that is, $\sum_j [D]_{ij} \geq 2^m$ for each state σ_i. If any state does not have enough edges, that state is eliminated and the remaining states are considered the set of principal states, P. Each of the remaining row sums of D^n, $\psi(\sigma_i, P) = \sum_{j \in \Omega} [D]^n_{ij}$, where $\Omega = \{j : \sigma_j \in P\}$ must be at least equal to 2^m for each i. Each time a state is eliminated from the set of principal states, this criterion is checked. The algorithm continues, eliminating states recursively, until a set of principal states is found that satisfies this criterion. With 2^m outgoing edges in each state, it is possible to assign a unique sourceword to the outgoing edges in each state, creating a deterministic encoder [9] that will, in general, require state-dependent decoding. If the procedure terminates successfully, the remaining states are principal states.

When the set of principal states is known, it is straightforward to trace through the FSM, starting from each state, to enumerate all the codewords of length n. With this information, it is possible to construct a deterministic encoder. The steps in code construction using this approach are therefore as follows. An FSM is constructed to describe the code, and its capacity, C, is evaluated. Values for m and n of reasonable length (say, $n < 20$) are then selected such that their ratio, m/n, is close to the value of C. With these values of m and n, the Franaszek algorithm is run to determine if a state-dependent code can be constructed. If so, code tables are constructed to facilitate both the encoding and decoding. If not, other values of m and n are chosen and Franaszek's algorithm is run again.

2.2.5 State-Independent Decoding

A desirable property for codes is the capability to decode the received words without requiring the use of state information at the receiver. This is important because keeping track of state information at the receiver can be problematic. If errors are made during decoding and the decoder determines the next state incorrectly, it may continue advancing through states incorrectly and therefore incorrectly decode subsequent words. There are at least two approaches to alleviate this problem: use of a sliding block decoder and state-independent decoding. In a sliding block decoder, each codeword is decoded by either looking ahead or looking back a fixed number of codewords, when ambiguity arises, in order to determine the state of the code. In this way, error propagation is limited to the width of the block considered by the sliding block decoder. State-independent decoding at the receiver eliminates the problem by not requiring that the decoder keep track of any state information. This requires that there be an unambiguous inverse to the output function, $h(\sigma_i, \beta_u)$, that

Table 2.3: Set of codewords partitioned into a group of alphabets.

σ_1	σ_2	σ_3
w_1	w_1	
w_2		w_2
w_3	w_3	w_3
	w_4	w_4
w_5	w_5	w_5

has no dependence on the encoder state σ_i. For a block-decodable code, any two edges with the same output label have the same input tag. By examining only the output labels, the encoder is then able to determine the input tag without requiring knowledge of the state σ_i, satisfying the need for state-independent decoding.

In order to determine whether or not it is possible to design a state-independent decoder, in [8] Franaszek suggests partitioning the code into a representation based on a group of alphabets. Each alphabet, describing a single state, contains codewords that are listed vertically in a table. Codewords that exist in more than one alphabet are placed in the same row in the table. The blank spaces in the table correspond to codewords that violate the constraints for that particular state and therefore do not exist in that particular alphabet. A simple example of this partitioning, containing five codewords and three states in total, is presented in Table 2.3. Upon arranging the codewords in this manner, it is possible to see which areas may prevent state-independent decoding. The "spaces" that exist between two codewords in a vertical line are possibly problematic. To find codes that can be state-independently decoded, the code designer must attempt to fit other codewords into these "spaces", creating a densely packed table of codewords with at least 2^m full rows of words. A single row full of words will comprise the encoder's output function, $h(\sigma, \beta_u)$, for a particular source word, β_u.

Based on this tabular representation of codewords, [8] describes the following necessary and sufficient condition for the existence of state-independent decoding:

Condition 1 *Condition for state-independent decoding [8]: Let $W(\sigma_j)$ denote the set of words, or alphabet, corresponding to state σ_j. A necessary and sufficient condition for the possibility of assigning binary words to the members of $W(\sigma_j), j = 1, 2, \ldots J$, where L is the total number of states, so that decoding is independent of j is that for $1 \leq u \leq v \leq y \leq L$ and for each*

$$w_{i_1} \in W(\sigma_u) \bigcap W(\sigma_y)$$

Table 2.4: Set of codewords that do not permit state-independent decoding.

source	σ_1	σ_2	σ_3
β_1	w_1	w_1	w_2
β_2	w_2	w_4	w_4
β_3	w_3	w_3	w_3
β_4	w_5	w_5	w_5

such that

$$w_{i_1} \notin W(\sigma_v)$$

there exists a $w_{i_2} \in W(\sigma_v)$ *such that there exists no* $\sigma_i, 1 \leq i \leq L$*, for which* $w_{i_1}, w_{i_2} \in W(\sigma_i)$.

Condition 1 states that whenever a codeword emanates from two states, σ_u and σ_y, but does not emanate from another state, σ_v, there must be another word, w_{i_2}, that emanates from state σ_v but does not emanate from either σ_u or σ_y. This condition simplifies to not having the same word in two different rows.

Consider the code in Table 2.4, which is one attempt at constructing a code table for the encoding of four source words given the previous table of codewords. In order to fill all the spaces, codeword w_2 is split and copied into the rows containing codewords w_1 and w_4. Unfortunately, this code cannot be decoded without state information since upon receipt of the word w_2, the decoder needs to know whether it is currently in state σ_1 or σ_3 in order to determine whether the source word was β_2 or β_1.

Condition 1 implicitly allows for discarding some words from the table, should they not be required. However, in the example above, word w_2 corresponding to source word β_2 cannot simply be discarded because there would not be enough codewords (or equivalently, enough edges in the encoder FSM) to satisfy the code construction when there are four source words to be represented. If there were only three different source words, then it would be possible to construct a table permitting state-independent decoding by discarding w_2, and using w_1 and w_4 to represent the same source word, where either w_1 or w_4 can be used in state σ_2. While discarding codewords does lower the rate of that particular code, it must be kept in mind that values for m and n chosen during the implementation are often already lower than the capacity of the code. That is, during evaluation of principal states, it is standard practice to choose m and n such that their ratio is as close as possible to the capacity of the code, but it is usually not possible to match that value exactly. In the construction of most codes, the ability to discard some codewords

becomes a significant degree of freedom to be leveraged. In Chapter 3, we consider in detail the construction of code tables that enable state-independent decoding.

2.2.6 Construction of State-Independent Decodable Codes

As discussed above, the steps to constructing a state-independent decodable code are as follows.

1. Using Franaszek's iterative procedure, suitable candidate values for m, n, and P are selected. Typically the values closest to achieving capacity are chosen first.

2. The constraint is modelled using an FSM, and the designer generates a list of sequences (codewords) that emanate from each state on the FSM.

3. Using the list of sequences, a tabular representation of the code is constructed using the group of alphabets approach proposed by Franaszek.

4. The tabular representation is manipulated in such a manner that the condition for state-independent decoding is not violated, while attempting to form a sufficient number of complete rows such that each source word can be represented regardless of state.

5. If it is not possible to construct a codebook that does not violate the condition, another set of candidate values for m, n, and P are selected and the procedure is started again.

In the next chapter, an approximation algorithm for the construction of codes that permit state-independent decoding is proposed. This algorithm details a method for performing step 4 in the list above. An alternate approach is considered below.

State-Independent Decoding: Set Partitioning and Covers

In [9] the authors consider, in detail, the determination of an optimal set of principal states. Along with their findings, they discuss the problem of constructing a state-independent decodable code, which differs from the representation discussed above. Initially, the authors perform a number of the same steps, including finding a set of principal states using the Franaszek algorithm and choosing values for m and n, as in step 1 above. Step 2 proceeds in a similar fashion: labels are assigned to output edges, and valid edges from each state are enumerated. Starting with step 3,

however, rather than examining the code in terms of a tabular representation using a group of alphabets, the authors present the problem in terms of sets. Codewords are partitioned into these sets, according to the states from which a particular codeword emanates, which the authors refer to as a partition of labels. The overall partitioning can be thought of as a Venn diagram visually displaying the regions. Fig 2.3 shows the set partitioning for a three state code. Regions 1, 2, and 3 denote codewords emanating only from states 1, 2, and 3, respectively. Region 4 denotes codewords that emanate from both states 1 and 2, region 5 denotes codewords common to both states 2 and 3, while region 6 denotes codewords emanating from both states 1 and 3. Finally, region 7 denotes codewords that emanate from all three states. For example, if a codeword w_i emanates from states σ_1, σ_2, and σ_3, then that codeword w_i would be placed in region 7. Upon the completed construction of this Venn diagram, all of the minimal covers are determined by solving a set covering problem. For the three state code, one example of a cover is the union of regions 1 and 5.

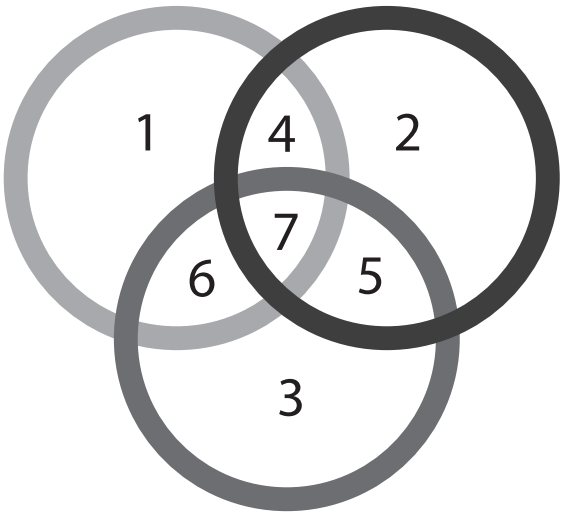

Figure 2.3: Venn diagram showing a set partitioning into regions for a three state code.

With the set covering problem solved, the regions, minimal covers, and their associated codewords are used as input to an integer programming problem. The goal of the integer programming problem is to construct covers for P using the regions in the Venn diagram; the details of the integer programming problem are discussed in [9]. As an example of the construction of a cover for P, consider again the example discussed in the preceding paragraph. If the code contained only those three states, σ_u, σ_v, and σ_y in P, then the word w_i would form a cover for P since it contains all the states within P. If there are additional states, then another word

would have to be combined with w_i in order to construct a cover for P. Comparing this to the tabular representation, a horizontal row in the table is a cover for P if the row contains no spaces. While the two methods of presentation are similar, the method used in [9] provides an alternate method for approaching the problem. In general, both the set covering problem and an integer programming problem are NP-complete. This will be revisited in Chapter 3 in more detail.

2.3 Spectral Analysis

The analysis of random signals can be found in [10], while the analysis of block coded signals is covered in [11]. In this section, a very brief summary of both topics is covered. In general, the main purpose is to establish terminology and notation, while presenting the important results, which is done without proof. Full proofs of the results contained in this section can be found in the two references cited above.

A common constraint for a code designer is that of satisfying particular spectral characteristics. A time-varying signal can be represented by its spectrum, a set of frequency components that occupy a particular range of frequencies, which is commonly evaluated by a Fourier transform. The most commonly used frequency domain characteristic for a stochastic signal is its PSD function.

2.3.1 Stochastic Processes

A stochastic process, X_t, can be defined as an ensemble of sample functions considered at particular time instants $t_1, t_2, \ldots t_n$. The random variables of the stochastic process X_{t_i} have a joint probability density function (pdf) $p(x_{t_1}, x_{t_2}, \ldots x_{t_n})$ or individual pdfs $p(x_{t_i})$. The mean of X_{t_i} is given by $M_X(t_i) = \mathrm{E}\{X(t_i)\}$, where $\mathrm{E}\{.\}$ is the expectation operator. When considering two random variables, X_{t_1} and X_{t_2}, the relevant statistics are the auto-correlation function:

$$R_X(t_1, t_2) = \mathrm{E}\{X_{t_1}, X_{t_2}\} \tag{2.10}$$

the average power of the process, $R_X(t_1, t_2)$ when $t_1 = t_2 = t$:

$$R_X(t, t) = \mathrm{E}\{X_t^2\} \tag{2.11}$$

and the auto-covariance:

$$C_X(t_1, t_2) = \mathrm{E}\{[X_{t_1} - M_X(t_1)][X_{t_2} - M_X(t_2)]\}. \tag{2.12}$$

Stationary Processes

A process is considered to be stationary in the strict sense if all of its statistics are time invariant. That is, the joint probability density function, $p(x_{t_1}, x_{t_2}, \ldots x_{t_n})$ does not change when shifted in time, and so $p(x_{t_1+\tau}, \ldots x_{t_n+\tau}) = p(x_{t_1}, \ldots x_{t_n})$ for an arbitrary time shift τ. In this case, since the pdf of (X_{t_1}, X_{t_2}) is identical to the pdf of (X_{t_1+t}, X_{t_2+t}), the autocorrelation function reduces to $R_X(t_1, t_2) = R_X(|t_1 - t_2|) = R_X(|\tau|)$, where $\tau = t_1 - t_2$. A slightly weaker form of stationarity, wide-sense stationarity, occurs when at least the mean and auto-correlation functions are time invariant, i.e., $\mathrm{E}\{X_t\} = M_X$ and $\mathrm{E}\{X_t X_{t+\tau}\} = R_X(\tau)$. When a process is at least wide-sense stationary, the auto-covariance function is given by $C_X(\tau) = R_X(\tau) - M_X^2$. In this case, the PSD of the process is given by the Fourier transform of the auto-correlation function:

$$H_X(\omega) = \int_{-\infty}^{\infty} R_X(\tau) \mathrm{e}^{-j\omega\tau} d\tau. \tag{2.13}$$

A stochastic process is said to be cyclo-stationary when its statistics, such as probabilities or correlations, vary with some period. A process that is wide-sense cyclo-stationary has the following properties:

$$\begin{aligned} M_X(t) = \mathrm{E}\{X(t)\} &= M_X(t+T) \\ R_X(t, t+\tau) &= R_X(t+T, t+T+\tau) \end{aligned} \tag{2.14}$$

for some period T. The block codes examined in this thesis are cyclo-stationary processes, since their statistics vary with a period equal to the block length. Typically, cyclo-stationary processes are transformed into stationary processes, so that the spectral analysis is tractable. To do this, a random variable Δ is added to the argument of the process, $X(t)$, so that:

$$X_\Delta(t) = X(t - \Delta) \tag{2.15}$$

where Δ is a uniformly distributed random variable over the interval $0 \leq \Delta \leq T$. Including the uniformly distributed random variable removes the time reference, a process which is referred to as phase-averaging [12]. The PSD of the cyclo-stationary process is then equal to the Fourier transform of the auto-correlation of the new stationary process, $X_\Delta(t)$.

In particular, a code designer constructing a constrained code is often interested in shaping the PSD. When a set of symbols $\{x_j\}$ are transmitted with pulse shape $s(t)$ with period T_b, the PSD is given by:

$$H_{X_\Delta}(\omega) = H_x(\omega) H_s(\omega) \tag{2.16}$$

with $H_x(\omega) = \sum_{k=-\infty}^{\infty} R_x(k)\mathrm{e}^{-jk\omega T_b}$ and $H_s(\omega) = \int_{-\infty}^{\infty} r_s(t)\mathrm{e}^{-j\omega t}dt$, where $r_s(t)$ is the autocorrelation of the pulse shape, that is $r_s(t) = \frac{1}{T_b}\int_{-\infty}^{\infty} s(\tau)s(t+\tau)d\tau$. In other words, the PSD of the overall process is related to the auto-correlation of the symbols that are being transmitted, through $H_x(\omega)$, and the pulse shape that is used, through $H_s(\omega)$. The design of constrained codes involves carefully constructing the symbols sequence $\{x_j\}$ such that the desired spectral properties are satisfied, by virtue of the auto-correlation function $R_x(k)$.

2.3.2 Spectral Analysis of Markov Information Sources

Consider an L-state ergodic Markov information source, with transition matrix Q. Its auto-correlation is given by [13]:

$$R_x(k) = \mathrm{E}\{X_t X_{t+k}\} \tag{2.17}$$

while the mean is given by:

$$M_x = \mathrm{E}\{X_t\} = \sum_{i=1}^{L} \zeta(\sigma_i)\pi_i \tag{2.18}$$

where $\zeta(\sigma_i)$ is the symbol emitted when the Markov chain visits state σ_i, and π_i is the steady-state probability of state σ_i as defined in subsection 2.1.2. The auto-covariance is then equal to:

$$\begin{aligned} C_x(k) &= \mathrm{E}\{X_t X_{t+k}\} - M_x^2 \\ &= \sum_{i=1}^{N}\sum_{j=1}^{N} \pi_i\zeta(\sigma_i)\zeta(\sigma_j)[Q]_{ij}^k - (\sum_{i=1}^{N}\zeta(\sigma_i)\pi_i)^2 \\ &= \sum_{i=1}^{N}\sum_{j=1}^{N} \pi_i\zeta(\sigma_i)([Q]_{ij}^k - \pi_j)\zeta(\sigma_j) \end{aligned} \tag{2.19}$$

where $[Q]_{ij}^k$ is the entry in the kth power of the transition matrix when the Markov chain transitions from state σ_i to state σ_j, separated by k intervals. (2.19) can be simplified by writing all variables in matrix form. The symbol outputs are written as a vector, according to $\boldsymbol{\zeta}^{\mathcal{T}} = (\zeta(\sigma_1), \zeta(\sigma_2), \ldots, \zeta(\sigma_L))$, where $\mathcal{T}$ denotes the matrix transpose. The steady-state transition probabilities are written in a diagonal matrix $\Pi = \mathrm{diag}\{\pi_1, \pi_2, \ldots, \pi_L\}$, where diag$\{.\}$ corresponds to creating a matrix with those entries along the diagonal and with all other entries being zero. Finally, $Q_\infty = \mathbf{1}\boldsymbol{\pi}$, where $\mathbf{1}$ is an all ones column vector and $\boldsymbol{\pi}$ is a vector containing the steady-state transition probabilities. The auto-covariance can now be written as:

$$C_x(k) = \boldsymbol{\zeta}^{\mathcal{T}}\Pi(Q^k - Q_\infty)\zeta. \tag{2.20}$$

The PSD of the Markov information source is then given by:

$$\begin{aligned} H_x(\omega) &= \sum_{k=-\infty}^{\infty} R_x(k)\mathrm{e}^{-jk\omega} \\ &= M_x^2 2\pi\delta(\omega) + C_x(0) + 2\sum_{k=1}^{\infty} C_x(k)\cos(k\omega) \end{aligned} \tag{2.21}$$

where the second line follows from use of (2.17), (2.19), and (2.20). This formulation separates the equation for the PSD into two portions: one related to the mean and auto-covariance at zero, and the other being a summation of the remaining terms in the auto-covariance. The usefulness of writing the equation for the PSD in this manner is discussed in the following subsection.

2.3.3 Spectral Analysis of Block Coded Signals

Similar to the case of a Markov information source, the equation defining the PSD of block-coded signals is somewhat difficult to work with; however, it can be simplified using matrix notation. In particular, consider a block code with source word length m and codeword length n, where $\mathbf{x}_j = (x_{j,1}, x_{j,2}, \ldots x_{j,n})$ is the jth transmitted codeword. The block coded sequence, using a pulse shape $s(t)$, is given by:

$$X(t) = \sum_{j=-\infty}^{\infty}\sum_{i=1}^{n} x_{j,i}s[t-(jn+i-1)] \tag{2.22}$$

Block coded signals can exhibit two types of periodicity: one related to the channel bit interval and a second related to the block length. The phase-averaging process takes place over the longer of the two periods, n. Following the procedure in [11], the auto-correlation coefficients can be written in matrix notation using an $n \times n$ matrix, R_k, given by:

$$R_k = \mathrm{E}\{x_t^{\mathcal{T}} x_{t+k}\}, k = 0, \pm 1, \pm 2, \ldots \tag{2.23}$$

The PSD is simply the Fourier transform of the auto-correlation function:

$$H_x(\omega) = \sum_{i=-\infty}^{\infty} R_x(i)\mathrm{e}^{-ji\omega}. \tag{2.24}$$

In this equation, the PSD depends on the phase-averaged correlations, $R_x(i)$, rather than the codeword correlations, R_k. While it is possible to evaluate the values

for $R_x(i)$ (see [11]), (2.24) can be re-written using R_k. Defining $\boldsymbol{\omega}$ as the vector $\boldsymbol{\omega} = ((e)^{jw}, (e)^{j2w}, \ldots, (e)^{jnw})$, (2.24) can be re-written as:

$$H_x(\omega) = \frac{1}{n} \sum_{k=-\infty}^{\infty} \boldsymbol{\omega} R_k \boldsymbol{\omega}^* \mathrm{e}^{-jkn\omega} \tag{2.25}$$

where $\boldsymbol{\omega}^*$ is the conjugate transpose of $\boldsymbol{\omega}$. This formulation performs the phase-averaging process while accepting the codeword correlations as input.

It is desirable to have the PSD written in a form similar to:

$$H_x(\omega) = H_{xc}(\omega) + H_{xd}(\omega) \sum_{k=-\infty}^{\infty} 2\pi\delta(\omega - 2\pi k/n) \tag{2.26}$$

so that the PSD consists of two separate contributions: the continuous component, $H_{xc}(\omega)$, and a series of discrete components, $H_{xd}(\omega)$, which are commonly referred to as spectral lines. The spectral lines, if non-zero, emerge at multiples of $\frac{1}{n}$. Recall that in (2.21), the PSD could be split into two parts. The first part, which is given by $M_x^2 2\pi\delta(\omega) + C_x(0)$, denotes the discrete components, which are related to the mean and auto-covariance at zero. The second portion, the summation, $2\sum_{k=1}^{\infty} C_x(k)cos(k\omega)$, uses the remainder of the auto-covariance, defining the continuous portion of the spectrum. This representation is preferred since it allows the code designer to isolate the effects of two contributions of the spectrum. Generally, a code designer wishes to minimize the discrete components while controlling the shape of the spectrum with the continuous component.

Re-writing (2.25) so that it appears in the form (2.26) gives

$$H_{xc}(\omega) = \frac{1}{n}\boldsymbol{\omega}(R_0 - R_\infty)\boldsymbol{\omega}^* + \frac{2}{n}\mathrm{Re}\sum_{k=1}^{\infty} \boldsymbol{\omega}(R_k - R_\infty)\boldsymbol{\omega}^* \mathrm{e}^{-jkn\omega} \tag{2.27}$$

and

$$H_{xd}(\omega) = \frac{1}{n^2}\boldsymbol{\omega} R_\infty \boldsymbol{\omega}^* \tag{2.28}$$

where R_∞ is the limit of the correlation matrices, R_k, as k tends to ∞. Notice that the discrete components are related only to the correlation matrix, R_∞, which is related only to the mean value of the codewords. Evaluating the continuous spectral components requires the evaluation of all of the correlation matrices, R_k, and so the problem of shaping the spectrum of a code appears difficult. Fortunately, most of the FSM encoders for constrained codes can be modelled as a Markov process, for which the above analysis holds, but which also have structure that makes the above analysis tractable. In the next subsection, the spectral analysis of FSM encoders with memory is considered, and a method to evaluate the R_k is given.

Spectral Analysis of FSM Encoders

Consider an L-state FSM encoder with transition probability matrix, Q, where each state accepts 2^m input words, outputting a codeword based on the current state and input word. The codewords of length n, χ_{iu}, are doubly indexed by the encoder state σ_i and the input word β_u. A matrix A_u, with dimension $L \times n$, is constructed for each source word, β_u, with output codewords comprising each row so that

$$A_u = \begin{bmatrix} \chi_{1u} \\ \chi_{2u} \\ \cdot \\ \cdot \\ \cdot \\ \chi_{Nu} \end{bmatrix}. \tag{2.29}$$

In other words, the collection of these matrices specifies the encoder's output function, $h()$ in matrix notation.

Similarly, the encoder's next state function, $g()$, is also written in a matrix notation. For each input word, β_u, a square matrix E_u is constructed. The entries of this matrix, $[E_u]_{ij}$, are set to unity if for sourceword β_u there is a valid transition from state σ_i to state σ_j. As in subsection 2.3.2, define the matrix $\Pi = \text{diag}\{\pi_1, \pi_2, \ldots \pi_L\}$ to be the diagonal matrix with the steady state transition probabilities $\pi_1, \pi_2, \ldots, \pi_L$ on its diagonal and zeros elsewhere. Similarly, $Q_\infty = \mathbf{1}\boldsymbol{\pi}$ so that Q_∞ has each of its rows set to be equal to the steady-state probabilities in vector form $\boldsymbol{\pi}$.

With this notation, along with derivations contained in [11], the first correlation matrix can be calculated according to

$$R_0 = \sum_{u=1}^{2^m} p_{\beta_u} A_u^{\mathcal{T}} \Pi A_u \tag{2.30}$$

where p_{β_u} is the probability of input word β_u. The remainder of the correlation matrices $(k \geq 1)$ are calculated according to

$$R_k = G_1^{\mathcal{T}} Q^{k-1} G_2 \tag{2.31}$$

where $G_1 = \sum_{u=1}^{2^m} p_{\beta_u} E_u^{\mathcal{T}} \Pi A_u$, $G_2 = \sum_{u=1}^{2^m} p_{\beta_u} A_u$, and Q^{k-1} is the $(k-1)$th power of the transition probability matrix. The zeroth power of the transition probability matrix, Q^0, is set to equal the identity matrix. The final correlation matrix is given by:

$$R_\infty = G_1^{\mathcal{T}} Q_\infty G_2 \tag{2.32}$$

The PSD of the block code is then calculated using (2.27) and (2.28) by substituting the values for the R_k evaluated using the method outlined in this subsection.

Generally speaking, in this thesis, when evaluating the PSD of the codes that have been constructed, a relatively simple code in the family of constructed codes is chosen and a full analytical spectral analysis is completed using the procedure above. This is compared to a PSD evaluation obtained via simulation to confirm the validity of the results. Upon verification, the remaining codes constructed within that family of codes have their PSD evaluated through simulation.

To obtain the PSD of a code through simulation, random source data is generated and fed through the encoder, which outputs a series of codewords. For work in this thesis, to generate accurate PSD plots with an acceptable execution time, five million codewords were analyzed. The codewords are concatenated serially into a vector and that vector is split into a number of equal segments. A fast Fourier transform (FFT) is applied to each segment and the FFT results are averaged to form the PSD. Rather than splitting the codeword vector into a number of equal, non-overlapping segments, it is possible to have the segments overlap. To improve the spectral results further, prior to calculating the FFT of each segment, a windowing function can be applied first. In this thesis, the spectral results obtained via simulation are calculated using non-overlapping segments of length 2048 with a Hamming window used as a windowing function.

2.4 Types of Constrained Codes

In this section, a brief summary is given of the most common types of constrained codes that appear in the literature. This includes both RLL codes and DC-free codes. As the names suggest, RLL codes are designed under the constraint that the run-lengths of the symbols in the coded sequence are restricted in some manner, while a DC-free code is a code that is designed under the constraint that there is a null in the spectrum of the code at DC. These two constraints are not necessarily mutually exclusive; merging the two together into a DC-free RLL constraint is also discussed. In most typical DC-free or RLL codes, the output alphabet is considered to be binary. Constrained codes employing multi-level signalling also exist in the literature, typically to satisfy a spectral constraint, and are also discussed in this section. Finally, multi-dimensional codes, where the constrained code employs binary signalling but is written to a two or more dimensional medium, are discussed briefly.

The overview of common constrained codes presented in this section serves two main purposes. First, the performance of the approximation algorithm developed in the next chapter was tested on a family of codes with a large number of states, those satisfying the the DC-free and RLL constraints. Second, while codes have been developed that use multi-level signalling or write to multi-dimensional surfaces, little work has been completed on codes using two or more dimensional signalling alphabets (natively allowing for multi-level signalling). Capacity calculations for constrained codes with multi-dimensional symbol alphabets is the focus of Chapter 4, while Chapter 5 applies the algorithm for code design to these types of codes.

2.4.1 Run-length Limited Constrained Codes

RLL codes date back to the 1960's, when several authors developed the theoretical foundation of these codes [14]- [17]. The field still remains an active area of research, as researchers continue to search for new codes and design techniques. RLL codes have become a mainstay in the optical storage industry, from the CD to the DVD, as well as the newer BluRay Disc. The EFM code was developed for and used in CDs [18], while its successor EFMPlus was used in DVDs [19]. These codes have also found use in Super Audio CDs and some miniDisc standards. Beginning in the 1980's, RLL codes were widely used in the floppy and hard disk drive storage area for many years. Hard disk drives have employed a synchronous variable length $(2, 7)$ RLL code [20].

RLL codes impose a minimum and/or maximum runlength constraint on the bit sequence that is being communicated across the channel, or equivalently, written to the storage medium. This runlength is usually measured in channel bits between transitions. A minimum runlength loosens the requirements on the detection circuitry by controlling the highest transition frequency. In the case of optical storage devices, such as a CD, the data is stored by a series of tiny indentations, knows as pits, while the areas between these pits are referred to as lands. Increasing the minimum runlength increases the size of these pits and lands, thereby easing the requirements of the detection circuitry that reads the data. In general, in a bandwidth-limited channel, ISI results when the data is transmitted; increasing the length of a symbol will reduce this ISI if the system bandwidth remains constant [2]. The maximum runlength ensures an adequate number of transitions in the data stream. These transitions are used by a phase-locked loop to recover timing information or by the decision threshold in the detection circuitry, to prevent them from drifting

and losing synchronization with the received symbol sequence.

In the literature, the minimum runlength is specified by d, while the maximum runlength is specified by k; together these two parameters specify a (d, k) code. For example, the $(2, 7)$ code used in hard drives has $d = 2$ and $k = 7$. The corresponding minimum and maximum runlengths are equal to $d + 1$ and $k + 1$ respectively. RLL sequences are often designed using another sequence, called a (dk) sequence. A (dk) sequence is characterized by two logical ones separated by at least d zeros, while the maximum runlength of logical zeros between ones is at most k. A (dk) sequence can be converted to a conventional RLL sequence by a simple process. A logical zero in the (dk) sequences specifies that there is no bit transition (zero to one or one to zero) in the RLL sequence, while the presence of a one indicates that there is a bit transition. Satisfying the d and k parameters in the (dk) sequence corresponds to an RLL sequence with minimum and maximum runlengths of $d+1$ and $k + 1$, respectively. Rather than designing an RLL sequence, the code designer can choose to work with (dk) sequences when convenient, as it often is. Additionally, depending on the storage medium, the data written to the storage medium may be written out directly as a (dk) sequence, which is referred to as non-return-to-zero (NRZ) notation. If the (dk) sequence is mapped onto an RLL sequence prior to being written to the storage medium, it is said to be in non-return-to-zero inverse (NRZI) notation.

2.4.2 DC-free Constrained Codes

Similar to RLL codes, DC-free codes have an extensive history [21]- [23]. DC-free codes, also termed DC-balanced or spectral null codes, are constrained codes that are designed to have zero spectral content at and around zero frequency in the continuous component of their PSD. The receiver in the system can leverage this knowledge and employ filtering prior to decoding. This filtering can be used to remove unwanted low frequency disturbances caused by other elements in the communication track, such as fingerprints on CDs, or can be an inherent part of the system, as when components are AC-coupled. Similar to RLL codes, the balanced nature of these codes assists with demodulation. For example, the frequent transitions in the data stream can be used for timing information for clock recovery, and the balanced nature assists with establishing reliable thresholds with which to determine the values of the received symbols [2].

DC-free codes with output symbols taken from a complex signalling alphabet, discussed in the later chapters of this thesis, have some additional advantages. The

position of this null in the frequency response of the code can be shifted by altering the complex values of the coded symbols [24]. This is useful for avoiding interference at a given frequency or for creating a signal-free frequency band for the insertion of a pilot tone.

Any code designed with balanced codewords (i.e., codewords with equal number of zeros and ones) will have a spectral null at DC in the continuous component of its spectrum. In this case, if antipodal signalling is employed, the code will also not have a discrete component in its spectrum at DC. The majority of DC-free codes are designed in this way because content in the discrete components of a spectrum is often viewed as "wasted" power because this power is not being used to carry information. Where possible, the spectrum of the DC-free codes designed in this thesis are kept free of tones. It is, however, generally a straightforward, but time-consuming, process to re-design the code slightly to remove these tones.

A common design technique to ensure that a code contains a null at DC is to use balanced codewords. Another way of stating this constraint is by considering the RDS of a codeword. The RDS, z_i, of a codeword, $\{x_i\} = \{x_1, x_2, \ldots x_n\}$ is defined as the cumulative summation of the bipolar representation of the bits in the word on a bit-by-bit basis. Using binary antipodal signalling values where the logic one has a value of $+1$ and a logic zero has a value of -1, that is, when $\{x_i\} \in \{-1, 1\}$, RDS is evaluated as:

$$z_i = \sum_{j=-\infty}^{i} x_j = z_{i-1} + x_i. \tag{2.33}$$

Generally speaking, only DC-free codes with codewords using a value of n that is even are considered. Codes with odd length codewords can also be balanced, for example by ensuring that each codeword has the same number of ones as all other codewords and the same number of zeros as all other codewords. For example, a length $n = 13$ code could be constructed where each codeword has seven ones and six zeros. These codes, however, will contain discrete components in their PSD and are not considered here. The RDS of a codeword is often referred to as its disparity, with an RDS of zero indicating a zero disparity codeword. A codeword with an equal number of zeros and ones will have a zero disparity, and a code containing only such codewords will have a spectral null at DC in both the continuous and discrete components of its spectrum assuming that antipodal signalling is used.

To determine the possible code rates for codes using zero disparity codewords, or balanced words, a method similar to (2.6) is used. The number of sequences that satisfy this constraint, $N_s(n)$, for a codeword length of n is found using the

Table 2.5: Zero disparity codewords of length n and the corresponding code rate in binary digits per symbol.

n	$N_s(n)$	R
2	2	0.500
4	6	0.646
6	20	0.720
8	70	0.766
10	252	0.798
12	924	0.821
14	3432	0.839
16	12870	0.853
18	48620	0.865
20	184756	0.875

binomial coefficient $N_s(n) = \binom{n}{n/2}$. If each of the symbols in the $N_s(n)$ sequences are binary, length $m = \lfloor \log_2 N_s(n) \rfloor$ source sequences can be represented. The maximum code rate is then $R_{max} = \frac{1}{n} \lfloor \log_2 N_s(n) \rfloor$ with units of binary digits per symbol. Table 2.5 lists values of $N_s(n)$ and R for several values of n. Clearly, as n increases, so does the rate of the code. This table presents only achievable rates, but does not address how the encoding or decoding is performed. Notice that the overall code rate is relatively low, even as n increases.

The code rate of DC-free codes can be increased by relaxing the constraint that each codeword have a zero-valued RDS. Codewords with disparities other than zero can be used, provided that the RDS of the overall coded sequence remains bounded [25]. For example, for $n = 6$ the word 010010, which has disparity -2, can be included along with the word 110110, which has disparity +2. If these two codewords were transmitted sequentially, the combined RDS would be zero and so it is possible for the overall RDS to remain bounded. Therefore, with a proper encoder design it is possible to use words of non-zero disparity. This encoder would choose the sequence of codewords so that the RDS value is bounded, typically by forcing this RDS to as close to zero as possible after each encoding interval. Commonly, codewords with low disparity, $\{\pm 2, \pm 4, \pm 6, \ldots\}$ are chosen, which we refer to as low-disparity codewords. The advantage of this approach is that more binary sequences are included in the codebook, increasing the number of different sequences that are possible, which increases the amount of information that can be carried by the coded sequence. The disadvantage is that since each codeword no longer has a zero RDS, the spectral performance around DC, specifically the width of the spectral notch, will suffer. Typically, the larger the disparity of the words that

Table 2.6: Capacity, in bits of information per symbol, of sequences $\{x_i\}$ as a function of DSV, N.

N	$C(N)$
3	0.500
4	0.694
5	0.792
6	0.895
7	0.886
8	0.910
9	0.928
10	0.940
11	0.950

are included in the code, the greater the degradation in the spectral performance of the code.

More generally, a code is DC-free when the RDS is bounded by a finite value [25]. Rather than ensuring every codeword is balanced, ensuring that the RDS is bounded is often a much simpler constraint to satisfy.

For codes using symbols from complex-valued alphabets, constructed later in this thesis, the definition of RDS is extended in a straightforward manner, using the symbol values in the complex baseband representation of the signalling alphabet in the RDS summation on a symbol-by-symbol basis. Provided that the RDS is bounded, as is the case when using symbols that are binary valued, the code will be DC-free [25]. A DC-free code with symbols from a complex baseband representation implies a null at the carrier frequency of the corresponding bandpass system.

If the RDS, z_i, of a sequence of bits, $\{x_i\}$, is bounded to a maximum, N_2, and minimum, N_1 value, the digital sum variation (DSV), which is the number of different values of RDS that can occur in the bit sequence, is given by $N = N_2 - N_1 + 1$. It should be noted that the DSV is calculated on a bit-by-bit basis and not only at the end of the codeword. This means that a code employing zero disparity codewords will still exhibit a non-zero DSV within each codeword, as it is only constrained to zero at the end of each word. The capacity of this constraint can be calculated as a function of the DSV. This is determined by the eigenvalues of the connection matrix of the resulting finite state machine [21]. Table 2.6 shows the results of the capacity, in bits of information per symbol, for a given DSV, N [2]. It is not surprising that as the DSV is allowed to increase, the resulting capacity increases as more bit sequences are included in the codebook.

The variance of the RDS, called the sum variance $s_z^2 = \mathrm{E}\{z_i^2\}$, gives an indi-

cation of the width of the spectral null around DC [22]. The width of the spectral notch is important, for example, since it relaxes the constraints imposed on the system designer when applying a high-pass filter to remove unwanted low frequency spectral content, i.e., noise or other signal distortion. Under the assumption that the auto-correlation function of the code decays exponentially, a useful relation between s_z^2 and the width of the spectral notch, ω_0, has been developed. This relation is concise and simple [22]:

$$2s_z^2\omega_0 \simeq 1. \tag{2.34}$$

In other words, the width of the spectral notch around DC is inversely proportional to the sum variance. In [22] several examples of codes that obey this relationship are presented. From a system design perspective, this result is of importance as it may be easier to design a code based on the sum variance of the codewords, rather than doing a full spectral analysis, knowing that codewords with minimal sum variance will result in good spectral performance.

2.4.3 DC-free RLL codes

Many of the constrained codes used in practical systems, such as the EFM and EFMPlus codes discussed before, are designed to satisfy both DC-free and RLL constraints simultaneously. There are many design techniques that can be used. For example, in the EFM code, three merging bits are added to the fourteen coded bits. While the addition of these bits was originally intended to ensure that the RLL constraint is satisfied, in some cases there are multiple choices for the three merging bits. In those cases, the merging bits are chosen such that the absolute value of the RDS is minimized at the end of the codeword. While this does not guarantee that the RDS is bounded, the infrequent violations of the constraint are not severe enough to be problematic. In the case of EFMPlus, the code has been designed as a sliding block code to guarantee that both the DC-free and RLL constraints have been satisfied while using a higher code rate. In the case of EFMPlus, there are extra words (or surplus edges) emanating from many of the states on the FSM. The additional words are used to satisfy both constraints simultaneously.

Another approach commonly used for constructing a DC-free RLL code is Guided Scrambling [26]. Instead of designing a code with a number of states, the encoder is able to generate a number of alternative codewords based on an input source word and a number of scrambling bits. The encoder then chooses the "best" codeword, which in this particular case is the codeword which best satisfies

the DC-free RLL constraints.

In the next Chapter, an approximation algorithm is developed that designs codes that can be decoded at the receiver without the need for state information. To quantify the performance of this algorithm, it is used to construct DC-free RLL codes for a wide variety of parameters for both the DC-free and RLL constraints. The encoder enforces both the RLL and DC-free constraints simultaneously, while the surplus edges emanating from a number of the states on the FSM are used to ensure that the code permits state independent decoding.

In DC-free RLL codes, the DC-free constraint is often specified by the maximum number of allowable RDS values, N, while the RLL constraint is usually specified in terms of d and k, as before. These constraints are specified simultaneously using the notation (d, k, N). However, in this case, constructing an FSM model is somewhat difficult when (dk) sequences are used. Instead, a simpler FSM model can be constructed by using states which have state variables indicating the current value for N, the current runlength, and whether the current runlength consists of zeros or ones. To construct this FSM, the constraints are portrayed in two dimensions where one constraint, which in this case is chosen to be the DC-free constraint, is modelled in the horizontal direction, while the other constraint, the runlength limit constraint, is shown in the vertical direction. The movement in the vertical direction is further divided into two halves, as the runlength constraint exists for both zeros and ones. The top half of the FSM is arbitrarily chosen to contain the runlength constraint for ones, while the bottom half represents the runlength constraint for zeros.

An example of such an FSM is presented in Fig. 2.4 for the case where the running digital sum is bounded to values -4 through +4, the maximum runlength is limited to 4, and the minimum runlength is 3. In this FSM, the output of a logic 0 results in movement downwards and to the left; the output of a logic 1 results in movement upwards and to the right. The states indicated with dashed circles do not need to be included in the model since they can never be entered and therefore will never be included as primary states; however, their inclusion might facilitate easier computer programming. Tightening or loosening the RDS constraints results in diminishing or increasing the number of states in the horizontal direction; decreasing or increasing the maximum runlength has a similar effect in the vertical direction. Changing the minimum runlength is accomplished by removing or inserting edges as necessary between states in the FSM.

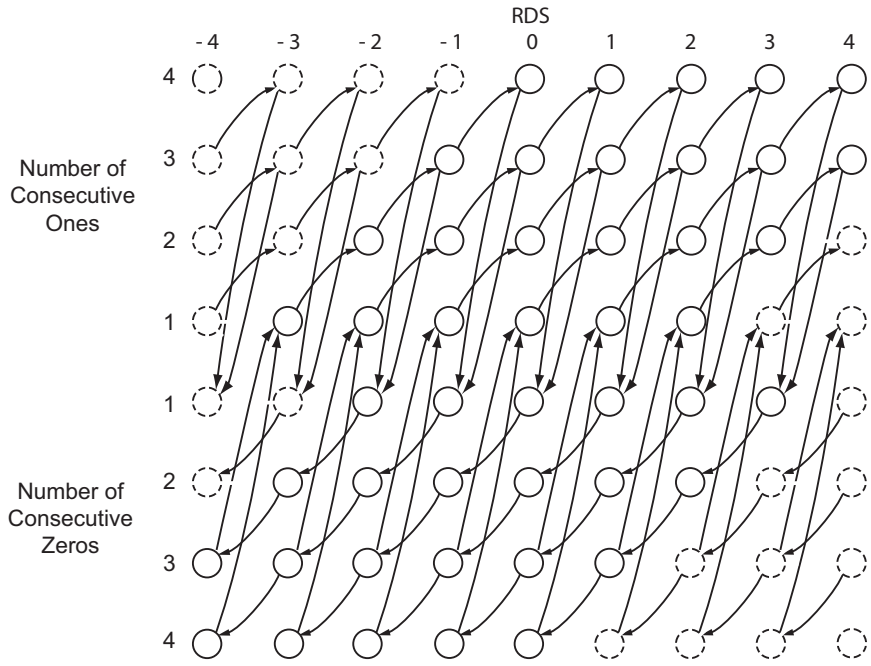

Figure 2.4: Sample FSM for a DC-free RLL code with parameters N = 9, $d = 2$, $k = 3$

2.4.4 Multilevel Constrained Codes

Constrained codes are commonly designed with a binary alphabet since the codes are generally employed on a medium where binary signalling is used. However, constrained codes have been extended to multilevel alphabets [27]. Multilevel codes are commonly designed to satisfy not only the DC-free constraint, with a null occurring at DC, but are also designed to have nulls occurring at certain other frequencies as well. For completeness, these codes can also be designed to satisfy runlength parameters, although they have more often focused on the design of spectral nulls. Examples of such codes can be found in [28] [29] [30]. Multilevel codes afford the code designer some additional advantages over codes using binary signalling alphabets, such as a wider variety of parameters, leading to better codes.

A $q$ary balanced code, with codewords of length n and source alphabet specified with $\Gamma_q = \{0, 1, \ldots q-1\}$, is a code where the weight of each codeword is equal to $\lfloor (q-1)n/2 \rfloor$. The weight of each codeword is calculated by taking the sum each of the symbols of the codeword. Note that when $q = 2$, we require codewords with weight $\lfloor (2-1)n/2 \rfloor = n/2$. Thus, for a length $n = 10$ code, the symbols must sum to $n/2 = 5$, which can only be achieved by transmitting the symbol "0" five times and the symbol "1" five times, in any order, creating the same balanced codewords

that were considered previously.

Perhaps the simplest method of constructing a multilevel code which contains only balanced codewords is to use Knuth's complementation method [31]. In this method, every sequence of symbols is balanced using a simple procedure. This method was originally designed to be used with bit-valued sequences, and was later extended to multi-level sequences. The process for bit-valued sequences is as follows. A sequence of bits is mapped onto a balanced word by complementing the first (or equivalently the last) k_c bits of each source word for an appropriate choice of k_c. For every sequence of bits, at least one valid choice for k_c exists, while a number of sequences may have more than one choice for k_c. The value of k_c is encoded in the coded sequence by appending its appropriate representation to the source sequence as parity bits so that the decoder can undo the mapping. Further, the parity bits must be chosen in such a manner that the overall codeword remains balanced. This method is suitable for very large sequence lengths and when the sequence lengths are long, results in a high code rate.

Extending Knuth's method to non-binary sequences involves generalizing the procedure. In the case of binary sequences, complementation is the bitwise addition of the source sequence with a flipping sequence. For example, complementing the first two bits of a length 8 sequence is the bitwise addition of the sequence 11000000. For multilevel sequences, appropriate "complementation" or balancing sequences are constructed as outlined in [32], one of which is the all zeros sequence. For every source sequence, at least one of the balancing sequences can be added such that a balanced sequence is generated. Examples of efficient codes using this method of design can be found in [33].

2.4.5 Constrained Codes for Multi-Dimensional Media

Recent advances in holographic storage technologies have led to the recording of two-dimensional arrays of data becoming an active area of research. The information stored on hard disk drives, as well as traditional optical drives, is written along a series of tracks, which can be visualized as a one-dimensional sequence. On these surfaces, an adequate length between transitions was important, ensuring an adequate number of transitions for the detection circuitry was required, and robustness to low frequency noise, i.e., fingerprints, was desirable. Some research has been conducted in two-dimensional RLL codes for these surfaces [34]; however, unlike their one-dimensional counterparts where RLL and DC-free constrained codes are commonly used, the two-dimensional media often employ more complex con-

straints. Codes designed for use on these media are referred to as two-dimensional codes and the underlying constraints are referred to as two-dimensional constraints.

Several different types of constraints that help to reduce ISI in two-dimensional recording channels have been proposed [35], [36]. One popular constraint in two-dimensional recording systems is the hard-square model, which is a $(d, k) = (1, \infty)$ RLL constraint extended to two dimensions. In this constraint, there cannot be neighboring pairs of 1's, either horizontally or vertically [37]. These types of constrained codes have proven to be difficult to study for several reasons. First, exact capacities are often not known for most types of constraints, only upper and lower bounds. Second, modelling the constraints with an FSM has proven to be difficult. Third, codewords are no longer sequential in nature but are given by two-dimensional shapes. The shapes themselves can be different, such as rectangles or parallelograms, and the methods for tiling them can vary [38]. Similar to one-dimensional codes, while the shapes themselves may not violate the condition, when tiling shapes together to construct a code, their concatenation at the edges may cause the constraint to be violated. In the one-dimensional case, the concatenation of sequences is easily solved with merging bits, such as in the case of EFM, but in the two-dimensional case, the solution is more complex. Research has been conducted in the study of the capacities of these constraints, along with the design of low complexity encoders and decoders [36].

Consider the case of the hard-square model, where in no place can there be two neighboring ones. The capacity of such a constraint is known to be approximately 0.58789 [39]. A very simple code, called the checkerboard code, satisfies this constraint and has a rate of one half, which is reasonably close to capacity. For the checkerboard code, the two-dimensional surface is partitioned into adjacent white and black squares, with each row beginning alternately with a white and then a black square, resembling a checkerboard. One of squares, arbitrarily chosen to be the black squares, is written with all zeros. The other set of squares can now contain the user's data, since, even when the user's data is a one, it is guaranteed to be surrounded by black squares, which contain zeros. The elegance in the simplicity of this code demonstrates a common scenario in code design. That is, an extremely simple code may come relatively close to achieving capacity; however, researchers are interested in designing codes as close as possible to capacity and will spend significant effort doing so, to achieve an $\frac{0.08789}{0.50000} = 17.6\%$ gain in the amount of information that can be written to the medium.

The constrained codes discussed in this subsection are commonly referred to as

two-dimensional codes or more generally as multi-dimensional codes. In this case, the dimensionality refers specifically to the medium, generally an optical storage device, upon which the coded data is written. In this thesis, constrained codes in multiple dimensions are investigated, but the concept of dimensionality is not the same. Rather than being written to a two-dimensional storage medium, the coded symbols are written to a one dimensional medium as a series of tracks, or equivalently transmitted on a symbol-by-symbol basis as in any conventional transmission system. Dimensionality instead occurs in the sense that the signalling constellation can have multiple dimensions. The encoder can constrain a number of signalling dimensions simultaneously, tracking the constraint on each dimension independently of the others. Examples of these types of multi-dimensional constrained codes are those employing QPSK, 8 PSK, or 16 QAM signalling constellations.

Chapter 3

State-Independent Decoding

In this chapter, the construction of constrained codes that are decoded at the receiver without the use of state information is considered. This property is important to limit error propagation at the decoder. In particular, this chapter focuses on block-decodable encoders, which are finite state encoders such that any two edges with the same output label have the same input tag [9]. Examining only the output sequence of such an encoder, the decoder can unambiguously determine the input sequence.

The construction of block-decodable encoders that admit state-independent decoding is considered in detail in [9]. The authors develop an algorithm to find the globally optimal solution to this problem by partitioning the codewords into regions, solving a set covering problem, and finally solving an integer programming problem. Both the set covering problem and integer programming problem are known to be NP-complete [40]. In this chapter, a simplified approach is considered; an approximation algorithm is developed that finds a constrained code that can be decoded without state information by using a simplified set covering procedure that constructs the codebook one mapping at a time. This procedure operates by finding locally optimal solutions based on a greedy approach. Despite the limitations of employing locally optimal solutions [41], it has been found that the approximation algorithm works very well for typical constraints, such as DC-free RLL codes. Further, since the set covering problem is avoided, the algorithm proposed in this chapter is able to run in polynomial time with respect to the size of the input.

The approximation algorithm proposed in this chapter is flexible, with a number of parameters that can be adjusted based on the needs of the code designer. Further, should the algorithm have difficulty finding a solution, more advanced methods, such as lookahead, can be introduced into the algorithm. The principles of weakly constrained codes are discussed and applied to an example code in a situation where neither the proposed algorithm, nor the algorithm in [9] is able to construct a code

that can be decoded at the receiver without the need for state information.

The approximation algorithm presented is based on a representation proposed by Franaszek [8]. In Section 3.2, an algorithm is developed for generating a code that permits state-independent decoding based on an FSM model of the encoder. This section covers the important aspects in the proper construction of the table based on Franaszek's representation in subsections 3.2.1 and 3.2.3. The cover construction method, referred to as the fitting procedure, is discussed in subsections 3.2.4 and 3.2.5, with a simplification discussed in subsection 3.2.6. In Section 3.3, the complexity of the algorithm is discussed. Section 3.4 presents lookahead as it can be used by the algorithm, along with a discussion of its complexity. In Section 3.5, the use of this algorithm for the construction of DC-free RLL codebooks is illustrated, including two example codes. Weakly constrained codes, along with the construction of a code using these principles, is presented in Section 3.6. A summary is offered in Section 3.7.

3.1 Construction of State-Independent Codes

Using Franaszek's tabular representation of the FSM describing movement among the principal states as a starting point, the goal is to construct a codebook with a complete row for each source word, where the same codeword does not appear in two or more different rows. This goal is accomplished by combining rows from Franaszek's table such that all "spaces" in the table are filled and as few of the entries as possible are discarded. Now considered, in detail, is the construction of a code table that enables state-independent decoding.

3.2 Algorithm for State-Independent Decoding

In this section, the approximation algorithm for the construction of a constrained code that can be decoded without the need for state information is discussed. The optimal solution to this problem is NP-complete, as both a set covering problem and an integer programming problem must be solved [40].

The method proposed in this chapter is a simplified approach to the algorithm presented in [9], which explores the full problem space. The solution presented in this chapter is an approximation algorithm, since the full problem space is not explored to maintain polynomial complexity with respect to the size of the input. While the solution is more precisely a heuristic, in the remainder of this thesis, it is

referred to as an "algorithm" for the sake of brevity. In particular, both the scoring functions and the base set selection procedures are based on an intuitive understanding of the problem. While this algorithm is an approximation, it demonstrates excellent performance on both codes commonly used in constrained coding, as well as the new family of codes developed later in this thesis.

Prior to presenting the proposed algorithm, Franaszek's recursive elimination algorithm and the initial construction of the coding table are revisited; this coding table serves as a starting point for the algorithm. Subsection 3.2.2 provides a general overview of the major steps of the algorithm. In the three subsequent subsections, major design decisions of the algorithm are discussed in more detail. Subsection 3.2.3 discusses the structure of the coding table and how it can be used advantageously, subsection 3.2.4 contains information about the fitting procedure used in the algorithm, and subsection 3.2.6 outlines how states can be reordered to reduce the time required to complete the algorithm. The complexity of the algorithm is discussed in Section 3.3.

3.2.1 Initial Construction of the Coding Table

The first step in constructing a code table is to find suitable values for m, the source word length, n, the codeword length, and P, the set of principal states, using Franaszek's iterative procedure. Generally, the set of parameters that is closest to achieving capacity is chosen first. As the procedure works by making locally optimal decisions, it may not be able to arrive at a suitable solution for a given constraint with the initial set of parameters of m, n, and P. In such a case, the set of values for m, n, and P that are the next closest to capacity can be tried.

Once Franaszek's algorithm terminates successfully and a set of values for m, n, and P is chosen, a coding table is generated in the same fashion as in [8]. In this table, the set of principal states, $P = \{\sigma_j, j = 1, 2, ...|P|\}$, denote the columns, while each different codeword, labeled by an index, i, is given its own row. If a codeword emanates from a given state σ_j then an entry w_i is entered into the table at the ith row and jth column. Thus, each row in the table indicates the set of principal states from which the word w_i emanates. Using a terminology similar to that in [9], this set of states is referred to by:

$$F(w_i, P) = \{\sigma_u \in P : w_i \in W(\sigma_u, P)\} \tag{3.1}$$

where σ_u is a state and $W(\sigma_u, P)$ is the set of all words in the FSM starting in state σ_u and ending in any state in P. As discussed in [9], the $F(w_i, P)$ define a number

of regions that can be visualized in terms of a Venn diagram. A number of words may populate each of these regions if their values for $F(w_i, P)$ are equal. Finding the globally optimal solution to code construction involves finding all of the minimal covers for P using those regions. In the proposed algorithm, examining the global constraints is avoided so that the algorithm does not need to know all of the minimal covers for P formed by those regions. This reduces the complexity of the problem significantly; however, it can no longer be guaranteed that the globally optimal solution will be found. In many cases, however, it is still possible to find a suitable constrained code that can be decoded at the receiver without state information.

For example, consider the state machine shown in Fig. 3.1. To complete the coding table using the group of alphabets approach, consider the process beginning with state σ_1. Word w_1 emanates from state σ_1, and so w_1 is placed in the first row and first column of the table. Further, words w_2, w_3, and w_4 emanate from state σ_1 and so those entries are placed in the second, third, and fourth rows of the first column, respectively. This completes the codewords leaving state σ_1 and so blank entries are left in the fifth, sixth, seventh, and eighth rows of the first column. Moving to state σ_2, the words emanating from that state are w_1, w_3, w_5, and w_6. Hence, the first row, second column is filled with w_1, the third row, second column is filled with w_3 and so on. This process is continued for states σ_3 to σ_6. The finished result is show in Table 3.1. Using the notation presented above, $F(w_1, P) = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$ since word w_1 emanates from states $\sigma_1, \sigma_2, \sigma_3$, and σ_4. As a second example, $F(w_6, P) = \{\sigma_2, \sigma_5, \sigma_6\}$.

Table 3.1: Example table showing group of alphabets.

σ_1	σ_2	σ_3	σ_4	σ_5	σ_6
w_1	w_1	w_1	w_1		
w_2		w_2		w_2	
w_3	w_3	w_3			
w_4		w_4	w_4		w_4
	w_5		w_5	w_5	w_5
	w_6			w_6	w_6
			w_7	w_7	w_7
				w_8	w_8

Rather than partitioning the codewords into regions and finding minimal covers, the alternate procedure starts as follows. First, complete the table as noted above and notice that there will, in general, be open areas since a word, w_i, may not

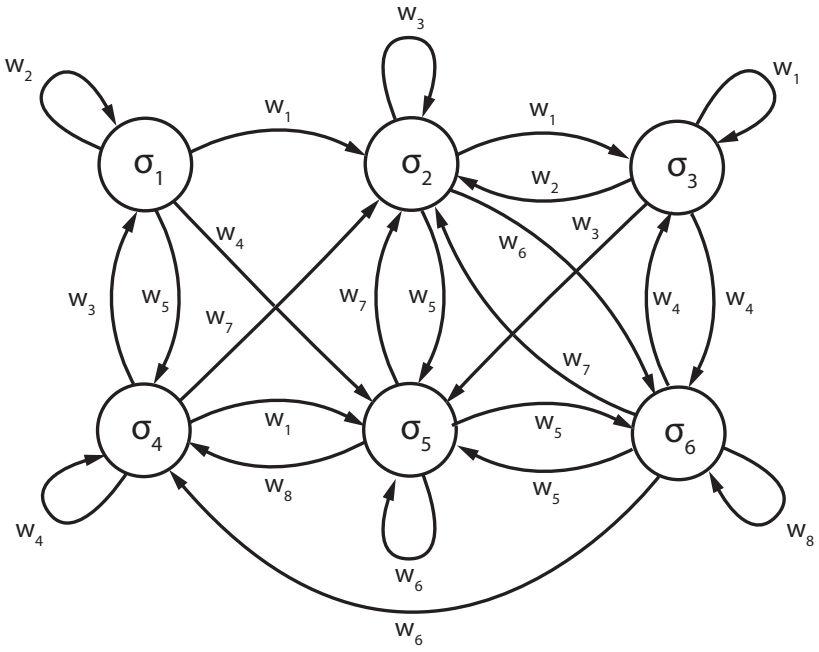

Figure 3.1: Example FSM used to demonstrate the process of translating an FSM into the tabular representation.

emanate from every state in P. In other words, each region does not form a cover for P by itself. These open areas are referred to as "spaces." Generating a code that admits state-independent decoding requires that several rows (i.e., several $F(w, P)$) be joined together such that they form a cover for P, eliminating the spaces in the table. If all the spaces in at least 2^m rows of the table are filled, then a code that admits state-independent decoding has been generated. In the example above, $F(w_1, P)$ and $F(w_6, P)$ form a cover for P since the combination of those two rows would leave no spaces in the table, or equivalently, the union of the states contains all states in P. However, $F(w_1, P)$ and $F(w_8, P)$ and several other combinations also form a cover for P using $F(w_1, P)$.

While it is possible to examine the code using only Condition 1 to determine if state-independent decoding is possible, it is advantageous to not only determine if the code admits state-independent decoding, but to simultaneously find the codebook. In fact, the solution from [9] uses the condition from [8] to guide the integer programming solution and also, if a code exists, generate the codebook. A similar approach is followed here, without having to solve the set covering problem and integer programming problem to explicitly determine whether Condition 1 is satisfied.

The solution in the proposed algorithm aims to complete a single row (a source

word to codeword mapping) in the codebook at a time. Codewords that exist in subsets of P are combined together intelligently such that they cover P and Condition 1 is not violated.

3.2.2 Outline of the Algorithm

The algorithm begins after an initial table is constructed as outlined above. In its most basic version, the major steps of the algorithm are:

1. By combining appropriate rows in the table to form a cover for P, the algorithm attempts to fill as many spaces as possible. Initially the algorithm aims to construct covers for P such that there is no overlap amongst the states. That is, when combining two rows, no state σ_u in the set of states U corresponding to $F(w_{i_1}, P)$ for the first row exists in the set of states V corresponding to $F(w_{i_2}, P)$ for the second row, where i_1 and i_2 are the indices of the two codewords being combined together in the coding table. If, at the completion of this step, there are 2^m complete rows (covers), skip to Step (4).

2. Pass through the table again to combine rows to form covers for P, now considering rows where overlap exists amongst the states. Combining rows in this fashion will result in codewords being discarded from positions of overlap. The choice of which codeword to discard can be arbitrary or, for example, can be chosen to reduce the complexity of the decoder. If, at the conclusion of this step, there are 2^m complete rows, skip to Step (4).

3. The algorithm has determined it cannot complete any further covers for P amongst the remaining $F(w_i, P)$, and so the procedure is stopped. This does not guarantee that it is impossible to decode this code without state information, but it requires that the algorithm re-trace some of its earlier decisions and change the outcome of decisions that were thought to be arbitrary. Alternatively, a solution based on globally optimum decisions, such as that found in [9], could be used instead.

4. If there are 2^m complete rows, then any other (possibly incomplete) rows are discarded and source words are assigned to each row. The simplest construction involves a table look-up, consisting of arbitrary assignment, although an attempt can be made during the assignment to limit the number of decoded errors that occur as a result of errors in the channel. If the mapping is to be performed using combinatorial logic, then the word assignment could be

made in a manner that results in the lowest gate count. However, discussion of how this mapping would take place is beyond the scope of this thesis.

These steps are discussed in more detail in the subsections that follow. Additionally, in subsection 3.2.5, the amalgamation of steps 1 and 2 into a single step with an appropriate scoring mechanism is considered.

3.2.3 Structure of the Table

In order to construct the original table, one state in P is arbitrarily selected as the "first" state, and in the first column, the codewords are listed that emanate from that state and terminate in any state in P. Codewords emanating from the second state are then listed in the second column, where codewords that emanate from both the first and second states are placed horizontally in the same row, and codewords that start in the second but not the first state are listed in additional rows appended to the bottom of the table. This procedure is then repeated for the third and subsequent states. The states can be numbered in any order; it is important only that all of the words emanating from a state be listed prior to considering the next state.

Given this approach to constructing the original table, in general, the set of words starting from the first state will comprise the densest portion of the table. To understand why, consider that a particular codeword w_i often emanates from several different states. When w_i emanates from state one, the ith row and first column is filled in with w_i. If this codeword also exists in some other state, σ_j, then the ith row and jth column will also be filled in with w_i. Thus, each time the algorithm examines a new state during initial table construction, it will, in general, append fewer new rows, since rows corresponding to a number of the codewords emanating from that state will already exist in the table due to consideration of other states. When the final state is reached, only the codewords unique to that state remain to be added to the table. For example, consider Table 3.1. Four rows are created in the first state, while only two rows are created in the second state, even though four words emanated from state σ_2. This is because rows related to two of the words from σ_2, namely w_1 and w_3 have already been added to the table. State σ_3 adds no new rows, while states σ_4 and σ_5 add one new row each.

Recall that in order to construct a valid codebook, it is required that at least 2^m rows be filled. Rows in the original table corresponding to codewords emanating from first state are selected as the "base set," and on a row-by-row basis, the other rows are searched to fill the spaces in these base rows (to form a cover for P). These

rows are selected as base rows since the table has been constructed such that the top portion of the table is the densest, and therefore, in general, fewer other rows will have to be combined to cover P, which simplifies the search. The remaining rows are referred to as the search set. This is because for each row in the base set, the algorithm searches through rows in the search set for the best fit.

A minimum of 2^m rows must be chosen as base rows, however, there are generally significantly more than 2^m possible candidates. For the codes generated in this thesis, the procedure outlined above was followed, that is, one state is chosen as the first state and all codewords emanating from the first state are chosen as base rows. In particular, the state chosen as the first state was the state containing the fewest number of edges, which, in general, yields excellent results. However, it should be noted that there is a significant number of valid selections for the base rows that represents a large degree of freedom for the algorithm. While not explored further in this thesis, depending on the particular application of the code designer, this degree of freedom may be helpful.

All of the rows with codewords emanating from first state are selected to be the base set even though this often results in the selection of slightly more than 2^m rows. This is not problematic for two reasons. Firstly, in Step (4) of the algorithm some of these rows may not contain a full cover of P and these rows can be eliminated or added to the search set. Secondly, because all of words in the base set share a common state (the first), they cannot be copied onto each other directly without having some overlap in the cover for P. It is possible, however, to place any rows fitting this criteria into the search set and re-run Step (2) of the algorithm, provided that at least 2^m rows remain in the base set.

Returning to the example from Table 3.1, if the value of m is set to two, then we require that $2^2 = 4$ rows be selected as base rows. In this case, rows one through four must be selected as base rows, since there are only four in total, while the remainder of the rows, in this case five through eight, are selected as the search set.

While the selection of the first state is arbitrary, as it is desired to have that state be the densest portion of the table, the ordering of codewords within that state does not need to be completely arbitrary. In many codes, the success of local decisions can be sensitive to the order in which rows are examined. In general, the algorithm makes the best decisions when attempting to first find matches for the rows with the fewest number of spaces, leading to better overall results. This is because these rows often have fewer possible matches compared to a row with a significant number of spaces. While this is not always the case, it serves as a good starting point and is

very simple to modify should a code not be generated. In some rare cases, beginning with states with the most number of spaces or even a purely random assignment may work better. To reduce the algorithm's dependence on factors such as this, lookahead can be used, which is discussed in a later section.

As the algorithm progresses, it copies rows from the search set into the base set attempting to create covers for P. If 2^m full rows are constructed, then the code is complete and permits state-independent decoding at the receiver. However, after all base rows have been iterated through several times (Step (1) of the algorithm), it is likely that there will not be 2^m full rows and that there will exist a number of rows in the search set that have not yet been placed. This occurs because the set of states, U, in $F(w_i, P)$ for one row has some overlap with the set of states, V, in $F(w, P)$ for all other rows remaining in the search set. Fortunately, there is another degree of freedom that can be exploited (Step (2) of the algorithm). There are often more words (or edges on the FSM diagram) than the minimum number that is required. Selectively allowing some overlap between U and V is then possible, such that the total amount of overlap across all entries in the column corresponding to state σ_j does not exceed the total amount of surplus edges that are in state σ_j. Stating this in another way, edges from the FSM can be selectively discarded since they cannot be used. Step (2) of the algorithm proceeds in an identical fashion to Step (1), on a row-by-row basis using a greedy algorithm to find a row or set of rows which forms a cover for P. In this case, overlap between the states in U and V is permitted provided that the loss of that edge on the FSM does not reduce the number of available edges in any state in P below 2^m. The solution which has the smallest amount of overlap amongst the states in U and V is chosen by the algorithm.

In the context of the designed code, the overall information flow is not lowered, since the code rate is given by $\frac{m}{n}$. This occurs because there is not an exact match between spaces in a base row and words in rows in the search space; sometimes there are more codewords in the searchable row than spaces in the base row. This means that it is possible to discard some of the words from a particular state (or column in the table). In general, this is avoided initially because it is desirable to save this potential freedom to employ a more intelligent encoding/decoding procedure if possible. However, if 2^m full rows cannot be generated without discarding words, then this degree of freedom must be leveraged to attempt to complete the codebook. Therefore, step (2) of the algorithm attempts to discard words from some of the states in order to allow more rows from the search set to be copied into base rows. The algorithm iterates through each of the "spaces" in the base rows,

attempting to fill them by discarding as few words in a particular column (state) as possible. At any point in time, every state must have at least 2^m words left, so it is important to check, before discarding words, that this minimum is still met.

Finally, in the larger codes that were considered, the size of the search set was often significant. Typically, this search set contains a large number of rows with the same spaces, and so the algorithm benefits from grouping these rows together. This allows the algorithm to compare a base row to many search rows in the set at a time since they have the same structure, effectively reducing the number of candidates it must attempt to try. Compressing the table in this manner can sometimes reduce the size of the search set significantly, allowing for more complex operations, such as lookahead, to be used on these large codes.

3.2.4 Fitting Procedure

When combining $F(w_i, P)$ (rows in the table) to form a cover for P in Steps (1) and (2) of the algorithm, it is necessary to determine how to fit rows together. The algorithm will combine $F(w_i, P)$ until it has constructed at least 2^m complete rows or has determined that it is impossible to do so. Note that several rows in the table may have the same $F(w, P)$; therefore, the same cover for P may be used multiple times. In other words, 2^m unique covers are not required, only that each column in the table must exist in 2^m covers so that a sufficient number of edges leave each state.

With row-by-row decisions, at least two approaches can be used to find suitable covers for P. First, the algorithm could consider a row corresponding to a word w_i with a set of states $U \subseteq P$ and find another row with a set of states $V \subseteq P$ to copy the initial row into such that for each state $\sigma_u \in U$, $\sigma_u \notin V$. This corresponds to focusing on a row that contains entries in certain columns, and searching for another row with spaces in these columns. Alternatively, for each row with a $F(w_{i_1}, P)$ consisting of states denoted by U and hence a set of spaces U' given by $\{\sigma_{u'} \in P : \sigma_{u'} \notin U\}$, the algorithm can find another row $F(w_{i_2}, P)$, with a set of states denoted by Y, such that $Y \subseteq U'$, and copy that row into the spaces of the first row. This corresponds to focusing on the spaces in a certain row, and finding a second row that fills as many of these spaces as possible. The second approach is preferred because it integrates smoothly with the state re-numbering procedure that is discussed in a later subsection. In either case, the two rows "fit" together. Spaces in the table are filled by copying rows of the table into other rows. In Step (1) of the algorithm, rows are copied such that there is no overlap of codewords when the

rows are combined together; in Step (2), the algorithm seeks to combine rows with minimal overlap. $F(w, P)$ for the aggregate row is the union of the $F(w, P)$ values for the two rows that have been copied together. As the rows are merged, and full rows are formed, the states in $F(w, P)$ form covers for P.

Another important consideration is how the fitting should be completed, which for a row-by-row approach, has at least the following two possibilities:

1. First fit, which involves copying a row into the first row in which the algorithm determines it would fit; or

2. Best fit, which involves copying a row into the "best" row that it finds, where "best" is determined according to some predefined criteria.

In other fields, it has been reported [42] [43] that the advantages of a first fit approach often include results very close to the best fit, in terms of utilization, with less overall time to complete. A best fit approach, however, should produce a higher overall utilization (i.e., more filled spaces) at the cost of execution time. In this application, a best fit approach is chosen because it is required that a row be completely filled (i.e. a cover for P is needed) in order to be used in the codebook. Even a single empty state remaining in a row prevents the assignment of a source word, so a higher emphasis is placed on utilization (filling rows) than the execution time of the algorithm. Further, because filling all the spaces in a row is essential, when copying a row with $F(w_i, P)$, corresponding to a set of states Y, into another row with $F(w_i, P)$, corresponding to a set of states U, the best fit is defined as the one in which the size of the set of remaining spaces, i.e., the set of states Y', given by $Y' = \{\sigma_{y'} \in P : \sigma_{y'} \notin U : \sigma_{y'} \notin Y\}$, is the smallest. In essence, the algorithm comprises a greedy algorithm solution on a row-by-row basis. As the rows in the table are iterated through, the locally optimum solution that will leave the fewest number of spaces remaining in the table is chosen. While it is known that a greedy algorithm finds the globally optimal solution only when the optimal solution to the problem contains optimal solutions to the local problems [41], it has been found that the algorithm does perform very well in the cases that were examined, such as the DC-free RLL codes that are considered in Section 3.5.

When determining the best fit for the set of spaces in a row, there are often situations where the row will not be entirely filled and at least one more row would have to be copied to generate a cover. However, there might be two or more alternatives that would leave the same number of spaces, but the set of spaces remaining are different. In this case, according to the best fit criteria, a tie exists, and it is difficult to

determine which is the better of the alternatives. One option is to create a tree-like structure to keep track of these decisions. If the algorithm completes and a code to permit state-independent decoding is not found, then the algorithm could return to this tree, going up levels and trying alternate solutions. Note that with this addition, the search is still not exhaustive since decisions remain based on locally optimum solutions. The converse of this solution, lookahead, is considered in a later section.

The pseudocode listed as Algorithm 1 demonstrates a simple implementation of step one of the proposed algorithm, where B is the size of the base set and S is the size of the search set. Note that the $F(w, P)$ have been separated into the base and search sets via $F_B(w, P)$ and $F_S(w, P)$, respectively.

Algorithm 1 Implementation of step one of the algorithm.

```
while go = true do
  placed ← 0
  for i = 1 → B do
    if F_B(w_i, P) = complete then
      continue
    end if
    U ← F_B(w_i, P)
    U' ← {σ_u' ∈ P : σ_u' ∉ U}
    for j = 1 → S do
      Y ← F_S(w_j, P)
      filled(j) ← |U' ⋂ Y|
    end for
    word = max(filled)
    if word ≠ ∅ then
      F_B(w_i, P) ← F_B(w_i, P) ⋂ F_S(w_word, P)
      placed ← placed +1
    end if
  end for
  if placed = 0 then
    go ← false
  end if
end while
```

3.2.5 Improved Fitting Procedure: Scoring

Rather than splitting the consideration of filled rows with and without overlap separately, it is possible to consider both these constraints simultaneously. In other words, steps 1 and 2 in subsection 3.2.2 are combined into a larger step, using a more sophisticated procedure. Implicit in the design of the proposed algorithm is

the use of two simple scoring mechanisms when selecting rows to be combined in the table. Previously, during the first pass, rows are selected which maximize a filled row score, S_f, and during the second pass, rows are selected which minimize an overlap score, S_o. A single combined scoring metric that considers filling and overlap simultaneously is now proposed, along with new definitions for S_o and S_f.

To find an appropriate balance between the advantage of filling and the disadvantage of overlap, the following overall score S_t is proposed:

$$S_t = S_f - S_o \tag{3.2}$$

and S_f and S_o are defined as outlined below.

In many codes, there are often some states with very few extra edges, while other states more central to the FSM have a large number of excess edges. Removing an edge from a state with a larger number of surplus edges is generally much less constraining than removing an edge from a state with few surplus edges. When combining two rows $F(w_{i_1}, P)$ and $F(w_{i_2}, P)$ with overlap $U_o = F(w_{i_1}, P) \cap F(w_{i_2}, P)$, the following overlap scoring metric has been adopted:

$$S_o = \sum_{\sigma_{u_o} \in U_o} (\frac{W_{s_{u_o}}}{W_{l_{u_o}}})^{r_o} \tag{3.3}$$

where $W_{s_{u_o}}$ is the number of spaces remaining in the base rows of a given column and $W_{l_{u_o}}$ is the number of words left in the search rows of a given column. The metric $(\frac{W_{s_{u_o}}}{W_{l_{u_o}}})$ ranges in value from zero to one and is a measure of how tightly a column is constrained, and thus the algorithm is less likely to discard an edge from a state that is already significantly constrained. The exponent r_o, restricted to positive values, allows the code designer to control the separation from highly constrained columns to columns that are not very constrained. For example, a fully constrained column that has as many words left as there are spaces receives a score of one. Alternatively, a column that is not very constrained will have many words left and few spaces to fill, receiving a score of much less than one.

Using a value of r_o greater than unity preserves the value of a fully constrained column, while lowering the value of columns that are not significantly constrained. However, using a value for r_o less than unity increases the weighting of a column that is not significantly constrained relative to a fully constrained column. Consider the case where one column has three spaces remaining to be filled and three words left with which to fill those spaces, while another column has three spaces remaining to be filled and twelve words left to fill those spaces with. Situations like this are

common in the constrained codes that have been constructed as there are often a few states that are significantly constrained, while other states have a significant number of surplus edges. In this example, $(\frac{W_{s_{u_o}}}{W_{l_{u_o}}})$ evaluates to unity for the first example column, while it evaluates to 0.25 for the second. Clearly, choosing a value of $r_o = 1$ will preserve the values of 1 and 0.25 for the first and second columns in consideration, respectively. Using a value of $r_o = 2$, however, changes the scored values to 1 and 0.0625, and so actions in the first column now carry more weight both with respect to the other column and also compared to the case when $r_o = 1$. Conversely, changing the value of r_0 to 0.5 results in the two scored values becoming 1 and 0.5 and so the second column is now twice as significant as it was, relative to using $r_0 = 1$.

The value chosen for r_o, then, controls the significance the algorithm places upon fully constrained columns, relative to the largely unconstrained columns, when considering the significance of the overlap. In general, the fully constrained columns have little flexibility, and therefore more significance should be placed on completing them. Setting the value of r_o to two provided a good compromise between ensuring that the fully constrained columns are weighted with importance, but not so much so that it prevents the algorithm from combining rows.

The filled row score has been re-defined in a similar fashion. This is because filling a space in a highly constrained column is more valuable than filling a space in a loosely constrained column. The filled row score for two rows $F(w_{i_1}, P)$ and $F(w_{i_2}, P)$ is defined as:

$$S_f = \sum_{\sigma_{y_f} \in Y_f} (\frac{W_{s_{y_f}}}{W_{l_{y_f}}})^{r_f} \tag{3.4}$$

where Y_f is the set of spaces filled by combining $F(w_{i_1}, P)$ and $F(w_{i_2}, P)$, and r_f is a positive exponent. For the codes that have been considered, it has been found that the algorithm works well when the value of r_f matches that of r_o.

The procedure for calculating S_f and S_o discussed above were found to generate excellent results for the cases upon which the algorithm was tested. In the initial version of the algorithm, step 1 assumes that the value of a space to one, or equivalently the value of that column to be one, and maximizes the number of filled spaces. The second step of the algorithm assumes that the value of each discarded edge is one and attempts to minimize the total, using the criteria in the first step as a tiebreaker when necessary. The limitations of this approach are apparent, since the algorithm assumes all spaces are equal, which is not true. Initially, a combined scoring approach was tried where only the overlap score in (3.3) was used, while

the filled row score was calculated assuming using the initial approach, assuming that all spaces are equal. Only the overlap score was updated initially, since it was assumed to be the most restrictive constraint. However, it was found that the algorithm had some difficulty combining together rows in some circumstances and that the filled row score was also significant. Consequently, a similar definition to the overlap score was applied to the filled row score as well, using (3.4). The inclusion of scoring exponents arose in a similar fashion, in order to help the algorithm combined together rows with greater precision. In many cases, a clever code designer can adjust the scoring criteria and form slightly to produce a better result after a quick examination of the initial coding table along with a few key decisions made by the algorithm.

The scoring approach outlined above is flexible and can be changed depending on the needs of the code designer. In particular, the code designer could use a parameter of interest, such as distance properties of the codewords, when selecting between two potential search rows that have the same score, as calculated using the equations above. Further, the code designer could add additional terms to the score computation to accommodate for any additional parameters of interest, depending on their application. The procedure outline in this subsection is intended to be a general solution that works for any family of codes, however, it is flexible enough that it can be modified to accommodate the needs of the code designer.

Using (3.3) and (3.4), S_t is calculated using (3.2). Two potential problems can arise with S_t that can be solved with a scoring threshold. First, because the base rows are considered sequentially in order, sometimes there may be no attractive $F(w_i, P)$ to combine with a particular row, forcing it to be combined with a row that would be better suited elsewhere. To adjust for this, a scoring threshold has been introduced. This scoring threshold ensures that only scores above a particular value will be acted upon. Second, as the algorithm progresses, the scores evaluated at each step will, in general, become lower. This is because more spaces in the table are filled, lowering the filled row score, while discarding words leads to a larger overlap score. Thus, sometimes it is necessary to lower the scoring threshold as the algorithm proceeds. This enables the algorithm to combine $F(w_i, P)$ that may have initially been unattractive. Typically, choosing an initial scoring threshold of zero is sufficient. With this threshold, each time rows are combined, the value of filling the row, given by S_f, outweighs the consequences of any overlap, given by S_o, that might exist. After an iteration where no rows can be further combined, the scoring threshold is lowered, and the process repeats until the table completes or no more

combinations can be formed.

Appropriate selection of the scoring metric, exponents, and thresholds allows for the algorithm to make row-by-row decisions that will lead to a state-independent decodable mapping with high probability. While other methods exist to prevent the algorithm from making locally optimal decisions that lead it away from the globally optimal solution, for example using the lookahead procedure discussed in Section 3.4, careful implementation of the scoring procedure can often produce the same result without the need for additional complexity. For example, of the 70 test cases that were tried, after some optimization of parameters, only 7% of those cases required lookahead to find an acceptable solution.

The pseudocode listed as Algorithm 2 demonstrates an implementation of the proposed algorithm with the scoring procedure.

3.2.6 Re-numbering of States

Consider that as the table grows in size, it may become difficult to efficiently find a suitable row in the search set to copy into a row in the base set. In this section, a simplification to the search procedure is discussed to more efficiently locate suitable pairs of rows that can be combined to form a cover for P. This simplification is amenable to computer implementation. This procedure is most useful when the algorithm operates with steps 1 and 2 separated, that is, without the use of scoring.

Recall that the numbering of the states is not unique, which implies that it is possible to re-number them as necessary. The following approach was found to be helpful. Each time the algorithm attempts to fill a row with $F(w_i, P)$ (corresponding to a set of states U) the states are renumbered such that all of the states in U are moved to the left-most portion of the table. The algorithm then searches for a row which fills as many of the right-most empty positions as possible. This enables an efficient search procedure, as the following example demonstrates.

Consider the code based on Table 3.1, in particular word w_2, which is a base row that contains states $U = \{\sigma_1, \sigma_3, \sigma_5\}$. These states can be temporarily re-numbered such that state three is mapped onto state two and state five is mapped into the third position. The presence of codewords in this row is represented with the binary sequence $\{1, 1, 1, 0, 0, 0\}$, where the ones correspond to the (re-numbered) states where the codeword exists, and the zeros correspond to the states from which the codeword does not emanate. This binary sequence can be represented by its integer equivalent, the number 56. As the algorithm searches the other rows in the table to form a cover for P, with this state re-numbering it now need only look for a row

Algorithm 2 Implementation of the algorithm with scoring.

while go $=$ true **do**
  placed $\leftarrow 0$
  for $i = 1 \rightarrow B$ **do**
    if $F_B(w_i, P)$ = complete **then**
      continue
    end if
    $U \leftarrow F_B(w_i, P)$
    $U' \leftarrow \{\sigma_{u'} \in P : \sigma_{u'} \notin U\}$
    for $j = 1 \rightarrow S$ **do**
      $Y \leftarrow F_S(w_j, P)$
      $Y_f \leftarrow U' \bigcap Y$
      $U_o \leftarrow U \bigcap Y$
      $S_f \leftarrow 0$
      for all $y_f \in Y_f$ **do**
        $S_f(j) \leftarrow S_f(j) + (\frac{W_s(y_f)}{W_l(y_f)})^{r_f}$
      end for
      $S_o \leftarrow 0$
      for all $u_o \in U_o$ **do**
        $S_o(j) \leftarrow S_o(j) + (\frac{W_s(u_o)}{W_l(u_o)})^{r_o}$
      end for
      $S_t(j) = S_f(j) - S_o(j)$
    end for
    word = $\max(S_t)$
    if word $\neq \emptyset$ **and** S_t(word) $\geq$ threshold **then**
      $F_B(w_i, P) \leftarrow F_B(w_i, P) \bigcap F_S(w_{\text{word}}, P)$
      $Y \leftarrow F_S(w_{\text{word}}, P)$
      $Y_f \leftarrow U' \bigcap Y$
      $U_o \leftarrow U \bigcap Y$
      for all $y_f \in Y_f$ **do**
        $W_s(y_f) \leftarrow W_s(y_f) - 1$
      end for
      for all $u_o \in U_o$ **do**
        $W_l(u_o) \leftarrow W_l(u_o) - 1$
      end for
      placed $\leftarrow$ placed $+1$
    end if
  end for
  if placed $= 0$ **then**
    go $\leftarrow$ false
  end if
end while

whose integer representation of the binary sequence is 7 or less. To form a cover for P, it first searches for a row with value equal to 7. If no row with the value of 7 exists, then it instead searches for a row that would fill two of those three spaces, which becomes a bit more difficult since the two spaces to be filled need not be consecutive. When the best fit is found, then the states are reordered, again moving the spaces to the far right of the binary sequence and another possible row, which would complete the cover of P, is sought. This re-numbering and binary mapping executes quickly on a computer.

3.3 Algorithm Complexity

The main loop of the algorithm consists of searching, updating of state variables, updating of tables, and re-numbering of states, all of which run in polynomial time. At most, a base row can choose between S different $F(w_i, P)$ candidates to be combined with to come closer to forming a cover for P, where S is the size of the search set. For one complete iteration through the algorithm, B base rows must choose a candidate, meaning that the overall complexity of a single loop is $O(BS)$, and that each loop through the algorithm maintains a polynomial complexity. For most codes considered in this thesis, the algorithm ran through this loop eight to 10 times at most, owing to the implementation of a scoring threshold. Without a scoring threshold, in a worse case, if only a single row is combined with another on each pass through the entire search set, the algorithm would complete the loop S times in total. This gives an overall complexity of $O(BS^2)$, which maintains a polynomial running time with respect to the size of the input. Generally, the number of operations is significantly less. This compares favorably with the method in [9] since both the set covering problem and the integer programming problem do not run in polynomial time.

The algorithm is centred around the computation of a scoring metric, even in the more sophisticated version employing lookahead discussed in the next section. Fortunately, the score of a particular row in the search set is independent of all other rows in the search set, and so the algorithm lends itself well to an implementation that includes parallelization. Additionally, in terms of overall complexity, the most significant factor is generally the size of the search set, S, since S scores must be computed for each base row and B is typically much smaller than S. However, all of these scores can easily be computed in parallel, speeding up the operation of the algorithm significantly.

In both cases, there is a set of operations with its own complexity that runs prior to the algorithm. This includes operations to generate the initial coding table by walking through the finite state machine to determine which codewords are valid in each of the states. This set of steps to generate the input matrix is required for both the algorithm proposed in this section and the method in [9]. In both cases, it is assumed that this partitioning is completed before either algorithm begins. The complexity of this sets of steps is, however, significant in that it is exponential in the codeword length, n. For each principal state, each of the 2^n codewords must be checked to determine whether that codeword emanates from that state and ends in a valid principal state.

3.4 Lookahead

While it has been found that (3.2) gives a good estimate of which rows should be combined, there may be cases where the algorithm remains unsuccessful. In Sec. 3.2.4, it is suggested that one approach to improving the algorithm is to maintain a decision tree, which can be unraveled if the algorithm is unsuccessful. In practice, the use of l_d-step lookahead is often more successful. The steps involved in the lookahead that has been implemented in the algorithm are as follows.

1. Select a potential $F(w, P)$ to combine with a particular base row to attempt to form a cover for P. Lookahead is only performed if another row must be selected to form the cover, i.e., if the cover is not already complete.

2. With this possible selection, iterate through all other base rows, selecting the best $F(w, P)$ for each row, until the algorithm returns to the base row in step (1).

3. From the rows still remaining in the search set, check to see whether another $F(w, P)$ exists for the base row in step (1) with a score above the scoring threshold to move closer to finding a cover for P.

4. If no $F(w, P)$ exists, the lookahead returns unsuccessful and the $F(w, P)$ with the next highest score is selected from the list of candidates. If another $F(w, P)$ exists, the lookahead returns successful.

These steps can be repeated l_d times for each step of the lookahead. In this particular lookahead implementation, if at any depth the lookahead fails, it returns to the initial potential $F(w, P)$ and re-selects from the list of candidates. For example,

consider a particular code the algorithm is designing with five levels of lookahead. The algorithm selects a candidate and looks ahead one step, selecting candidates for all other base rows. The lookahead algorithm then selects another candidate for the base row and all other base rows, but no further candidates remain for the base row in question. The lookahead algorithm then returns to the initial candidate from the first step and re-selects the next best candidate. In other words, the lookahead algorithm returns to the top-most level of the tree each time, regardless as to where the lookahead fails. The pseudocode listed as Algorithm 3 demonstrates an implementation of the lookahead procedure.

One variation for the lookahead procedure is to only go back one level when the lookahead fails, rather than always returning to the initial level. The advantage of using this approach is that more of the problem space is explored. However, that approach has significantly higher complexity than the approach that is proposed. In general, it was found that, for the codes that required lookahead, returning to the top-most level was sufficient to achieve a code design.

While lookahead increases the complexity of the algorithm, the overall complexity remains polynomial with respect to the size of the input. In step (2) above, the lookahead algorithm must select an $F(w_i, P)$ from the search set for each row in the base set and so the complexity for each row is $O(S)$, similar to the previous section. The overall complexity of step (2) is then $O(BS)$. Step (2) can be completed approximately S times, assuming every candidate in the search set is tried, and thus the complexity of one step of lookahead is $O(BS^2)$. Since, if the lookahead fails on any iteration, the algorithm elects to revert to the top-most decision and try another $F(w_i, P)$ instead, the lookahead step factor is multiplicative, and so the overall lookahead complexity is $O(BS^2 l_d)$, which is polynomial in both B and S. In the codes that were considered, a depth of three to five levels was found to be sufficient, and so l_d is significantly smaller than B and S.

3.5 Example Codes

The algorithm described in the previous sections has several parameters that can be adjusted when applying it to code construction. The code designer has flexibility when selecting r_f and r_o, as well as implementing scoring thresholds. While optimum selection of these parameters is dependent on the code, a quick glance at the coding table and the first round of selections with the recommended parameters from the previous sections ($r_f, r_o = 2$, scoring threshold = 0) can provide a good

Algorithm 3 Implementation of the lookahead procedure.

success ← true {default flag to true}
for $i = 1 \rightarrow l_d$ **do**
  $F'_S(w, P) \leftarrow F_S(w, P)$ {Create a copies of F_S, W_s, W_l to simulate algorithm}
  $W'_s \leftarrow W_s$
  $W'_l \leftarrow W_l$
  row ← contents of $F_B(w, P)$ for the base row in question
  $L_B \leftarrow$ set of all other base rows
  for all $l_B \in L_B$ **do**
    Compute Scores for all $F'_S(w, P)$, storing in S_t
    word = $\max(S_t)$
    if word $\neq \emptyset$ **and** S_t(word) $\geq$ threshold **then**
      remove $F_S(w_{\text{word}}, P)$ from $F'_S(w, P)$
      update W'_s, W'_l
    end if
  end for
  Compute scores for all $F'_S(w, P)$ for row, storing in S_t
  word = $\max(S_t)$
  if word $\neq \emptyset$ **and** S_t(word) $\geq$ threshold **then**
    row ← row $\bigcap F'_S(w_{\text{word}}, P)$
    remove $F'_S(w_{\text{word}}, P)$ from $F'_S(w, P)$
    update W'_s, W'_l
    if row = complete **then**
      break
    end if
  else
    success ← false
    break
  end if
end for

guide for fine tuning these parameters. In the art of code design, examining the code on this level can be very helpful for fine-tuning the parameters of the algorithm, and for better understanding of the code that is being designed. Further, the code designer can rearrange the order of the base rows or select which base rows to complete should there be more than 2^m such rows. As with the other parameters, reviewing the first round of matches with the default ordering of base rows provides a good guide. Finally, the designer can choose to use one or more levels of lookahead to improve the decision-making process of the algorithm. Appropriate selection of these parameters is an important aspect of the art of code design.

To demonstrate the usefulness of the proposed algorithm, the construction of DC-free RLL codes is considered. This particular constraint was chosen for two main reasons. First, this is a constraint that is quite familiar to researchers in the area of constrained codes, since it is widely considered. Second, satisfying both the DC-free and RLL constraints simultaneously is difficult and if the algorithm is able to perform well with this particular constraint, it is anticipated that it will perform well for many other types of constraints. Values of $0 \leq d \leq 3$, $d+1 \leq k \leq 7$, and $5 \leq N \leq 9$, where N is the digital sum variation, were considered. These codes have a range in capacities from $C = 0.15678$ to $C = 0.92760$ bits of information per symbol(see [1], pg. 280 Table 11.1). In each case, the algorithm attempted to construct a valid codebook that can be decoded at the receiver in the absence of state information for the parameters returned by the Franaszek algorithm with the highest code rate for $n \leq 20$. Of the 70 constraints that were considered, the algorithm was able to successfully construct codes for all but one case. Close examination of the one exception, $(d, k, N) = (3, 5, 8)$ with $m = 3$ and $n = 13$, showed that it is in fact impossible to construct a code that can be decoded without state information with this set of parameters. The algorithm was able to successfully construct a code for the parameters returned by the Franaszek algorithm with the second highest code rate, with $m = 4$ and $n = 18$.

Of the 69 codes that the algorithm was able to construct, there were five instances where use of lookahead was necessary. These codes were $(3, 5, 9)$, $(3, 6, 8)$, $(2, 4, 7)$, $(2, 5, 8)$, and $(1, 5, 7)$, and required between one and five steps of lookahead. The remaining 64 codes were constructed using only the scoring improvements in (3.2), (3.3), and (3.4). For example, the $(0, 5, 9)$ code has parameters $m = 18$ and $n = 20$. There are 793028 valid codewords across 50 primary states. Upon completion of the algorithm, 263277 covers for P, slightly more than the minimum requirement of 2^{18} = 262144, were generated. Subsequent simulation

confirmed the code to be DC-free, satisfying both the RLL and RDS constraints.

Two examples of the algorithm successfully designing DC-free RLL codes are presented below. The algorithm designed a state-independent decodable code with (d, k) constraints of $(1, 5)$, along with a DC-free constraint enforced by having a maximum of $N = 7$ different RDS values $\{-3, -2, -1, 0, 1, 2, 3\}$. The capacity of this code is $C = 0.5497$ bits of information per symbol and for $n \leq 20$, Franaszek's recursive elimination algorithm finds viable coding parameters that are closest to capacity to be $m = 4$ and $n = 8$, giving a coding rate of 0.5000 binary digits per symbol for an efficiency of 0.91 binary digits per bit of information. Table 3.2 shows the binary representation for each codeword used in the constructed code, while Table 3.3 presents a mapping that will admit state-independent decoding at the receiver. Since this code has $m = 4$, at least 16 full rows are required. Although this code has 16 primary states, it is straightforward to perform the algorithm by hand since the number of words in each state is small. By inspecting the resulting table, it is evident that row 22 and row 1 together form a cover for P. Similarly, row 23 and row 2 form another cover of P. Continuing downwards, notice that rows 5, 19, and 35 form a cover for P. Row 19 was copied without any overlap, however, four edges were discarded when joining rows 35 and 5. This process was continued for the remainder of the rows in the table. Recall that the requirement for state-independent decoding is that no word exists on two different rows in the table. Careful examination of the table will verify that this constraint has been satisfied.

To verify that the first example code was indeed a DC-free RLL code, a simulation of five million codewords was used to generate the code spectrum, along with analyzing the minimum and maximum runlengths on the entire sequence of codewords. Figure 3.2 shows the PSD of this code, which bears the characteristics of a DC-free code, showing a clear null at DC. In Figure 3.3, the runlengths within the coded sequence are plotted, where the first runlength has been ignored to remove the dependence on initial conditions. A new runlength is marked and "indexed" each time there is a transition from a zero to a one, or vice versa. The figure demonstrates that the runlength stays between 2 and 6, which matches the design specifications of the code.

A second, more complex code, with (d, k) constraints of $(1, 3)$ and a DC-free constraint of $N = 5$ different RDS values was also constructed. The capacity of this code is $C = 0.4248$ bits of information per symbol and the Franaszek algorithm determined that the closest set of code parameters to achieving capacity with $n \leq 20$ is $m = 8$ and $n = 20$ for a code rate of 0.4000 binary digits per symbol and

Table 3.2: Codeword mapping for $d = 1, k = 5, N = 7$ with $m = 4$ and $n = 8$.

Label	Codeword	Label	Codeword
w_1	1 1 0 0 0 1 1 1	w_{27}	0 0 1 1 0 0 1 1
w_2	1 1 0 0 1 1 0 0	w_{28}	0 0 1 1 1 0 0 0
w_3	1 1 0 0 1 1 1 0	w_{29}	1 0 0 0 0 0 0 1
w_4	1 1 1 0 0 0 0 1	w_{30}	1 0 0 0 0 0 1 1
w_5	1 1 1 0 0 0 1 1	w_{31}	1 0 0 0 0 1 1 0
w_6	1 1 1 0 0 1 1 0	w_{32}	1 0 0 0 0 1 1 1
w_7	1 1 0 0 1 1 1 1	w_{33}	1 0 0 0 1 1 0 0
w_8	1 1 1 0 0 1 1 1	w_{34}	0 0 0 1 1 1 1 1
w_9	1 1 1 1 0 0 0 0	w_{35}	0 0 1 1 1 0 0 1
w_{10}	1 1 1 1 0 0 0 1	w_{36}	0 0 1 1 1 1 0 0
w_{11}	1 1 1 1 0 0 1 1	w_{37}	0 0 1 1 1 1 1 0
w_{12}	1 1 1 1 1 0 0 0	w_{38}	1 0 0 0 1 1 1 1
w_{13}	1 0 0 1 1 0 0 1	w_{39}	1 1 0 0 0 0 0 1
w_{14}	1 0 0 1 1 1 0 0	w_{40}	1 1 0 0 0 0 1 1
w_{15}	1 0 0 1 1 1 1 0	w_{41}	1 1 0 0 0 1 1 0
w_{16}	1 0 0 1 1 1 1 1	w_{42}	1 1 1 0 0 0 0 0
w_{17}	0 0 0 0 0 1 1 1	w_{43}	0 1 1 0 0 0 0 1
w_{18}	0 0 0 0 1 1 0 0	w_{44}	0 1 1 0 0 0 1 1
w_{19}	0 0 0 0 1 1 1 0	w_{45}	0 1 1 0 0 1 1 0
w_{20}	0 0 0 0 1 1 1 1	w_{46}	0 1 1 0 0 1 1 1
w_{21}	0 0 0 1 1 0 0 0	w_{47}	0 1 1 1 0 0 0 0
w_{22}	0 0 0 1 1 0 0 1	w_{48}	0 1 1 1 0 0 0 1
w_{23}	0 0 0 1 1 1 0 0	w_{49}	0 1 1 1 0 0 1 1
w_{24}	0 0 0 1 1 1 1 0	w_{50}	0 1 1 1 1 0 0 1
w_{25}	0 0 1 1 0 0 0 0	w_{51}	0 1 1 1 1 1 0 0
w_{26}	0 0 1 1 0 0 0 1	w_{52}	0 1 1 0 0 0 0 0

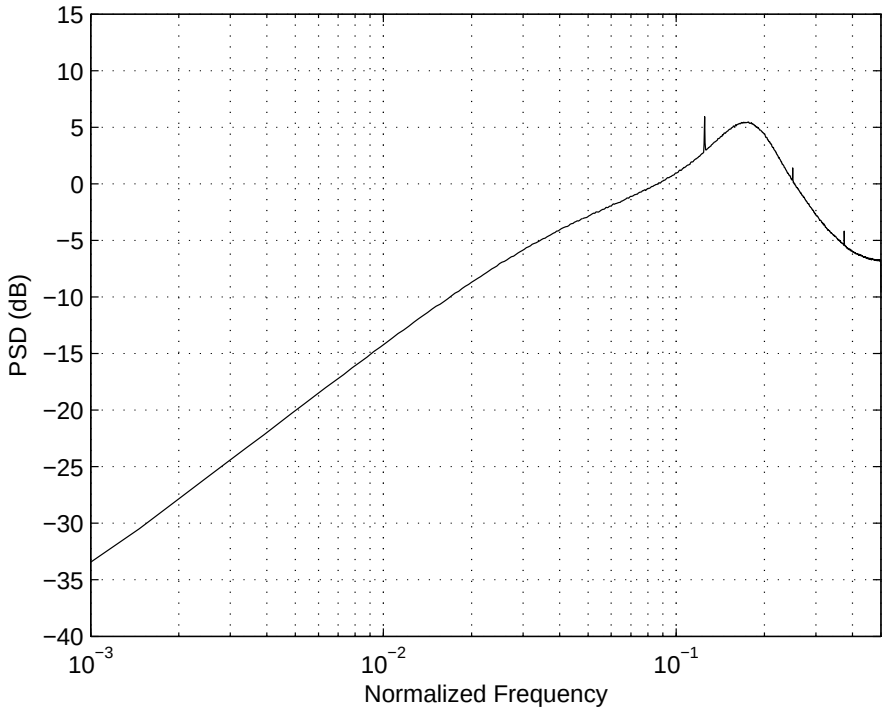

Figure 3.2: PSD of DC-free RLL code with $d = 1, k = 5$ and $N = 7$.

an efficiency of 0.94 binary digits per bit of information. This code has 16 primary states, and the complete code table has 286 rows (slightly more than the minimum requirement of 256) of which the first 15 are presented in Table 3.4. While the algorithm completed the procedure for all 286 rows, they are not all presented here as the table would be very lengthy. The majority of these rows have been constructed through the combination of several rows from the initial table. For example, the second row contains words from four different initial rows: rows 2, 535, 621, and 983. This particular code contains 1658 unique codewords in total. Similar to the previous case, simulations were performed to verify that this code satisfied both the DC-free and runlength constraints.

Table 3.3: Code table for $d = 1, k = 5, N = 7$ with $m = 4$ and $n = 8$.

	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9	σ_{10}	σ_{11}	σ_{12}	σ_{13}	σ_{14}	σ_{15}	σ_{16}
β_1	w_1	w_{22}	w_1	w_{22}	w_1	w_{22}	w_{22}	w_1	w_{22}	w_1	w_1	w_1	w_1	w_1	w_1	w_{22}
β_2	w_2	w_{23}	w_2	w_{23}	w_2	w_{23}	w_{23}	w_2	w_{23}	w_2	w_2	w_2	w_2	w_2	w_2	w_{23}
β_3	w_3	w_{17}	w_3	w_{17}	w_3	w_{17}	w_{17}	w_3	w_{47}	w_3	w_3	w_3	w_3	w_3	w_3	w_{17}
β_4	w_4	w_{18}	w_4	w_{18}	w_4	w_{18}	w_{18}	w_4	w_{34}	w_4	w_4	w_4	w_4	w_4	w_4	w_{18}
β_5	w_5	w_{19}	w_5	w_{19}	w_5	w_{19}	w_{19}	w_5	w_{35}	w_5	w_5	w_5	w_5	w_5	w_5	w_{19}
β_6	w_6	w_{20}	w_6	w_{20}	w_6	w_{20}	w_{20}	w_6	w_{36}	w_6	w_6	w_6	w_6	w_6	w_6	w_{20}
β_7	w_7	w_{24}	w_{24}	w_{24}	w_{24}	w_{24}	w_{24}	w_{38}	w_{24}	w_7	w_7	w_{24}	w_7	w_{24}	w_7	w_{24}
β_8	w_8	w_{26}	w_{26}	w_{26}	w_{26}	w_{26}	w_{26}	w_{39}	w_{26}	w_8	w_8	w_{26}	w_8	w_{26}	w_8	w_{26}
β_9	w_9	w_{27}	w_{27}	w_{27}	w_{27}	w_{27}	w_{27}	w_{41}	w_{27}	w_9	w_9	w_{27}	w_9	w_{27}	w_9	w_{27}
β_{10}	w_{10}	w_{28}	w_{28}	w_{28}	w_{28}	w_{28}	w_{28}	w_{42}	w_{28}	w_{10}	w_{10}	w_{28}	w_{10}	w_{28}	w_{10}	w_{28}
β_{11}	w_{11}	w_{21}	w_{40}	w_{21}	w_{40}	w_{21}	w_{21}	w_{40}	w_{49}	w_{11}	w_{11}	w_{40}	w_{11}	w_{40}	w_{11}	w_{21}
β_{12}	w_{12}	w_{25}	w_{37}	w_{25}	w_{37}	w_{25}	w_{25}	w_{33}	w_{37}	w_{12}	w_{12}	w_{37}	w_{12}	w_{37}	w_{12}	w_{25}
β_{13}	w_{13}	w_{29}	w_{13}	w_{29}	w_{13}	w_{29}	w_{29}	w_{13}	w_{46}	w_{46}	w_{46}	w_{46}	w_{46}	w_{46}	w_{46}	w_{45}
β_{14}	w_{14}	w_{31}	w_{14}	w_{31}	w_{14}	w_{31}	w_{31}	w_{14}	w_{48}	w_{48}	w_{48}	w_{48}	w_{48}	w_{48}	w_{48}	w_{52}
β_{15}	w_{15}	w_{30}	w_{15}	w_{30}	w_{15}	w_{30}	w_{30}	w_{15}	w_{43}	w_{50}	w_{50}	w_{43}	w_{50}	w_{43}	w_{50}	w_{43}
β_{16}	w_{16}	w_{32}	w_{32}	w_{32}	w_{32}	w_{32}	w_{32}	w_{32}	w_{44}	w_{51}	w_{51}	w_{44}	w_{51}	w_{44}	w_{51}	w_{44}

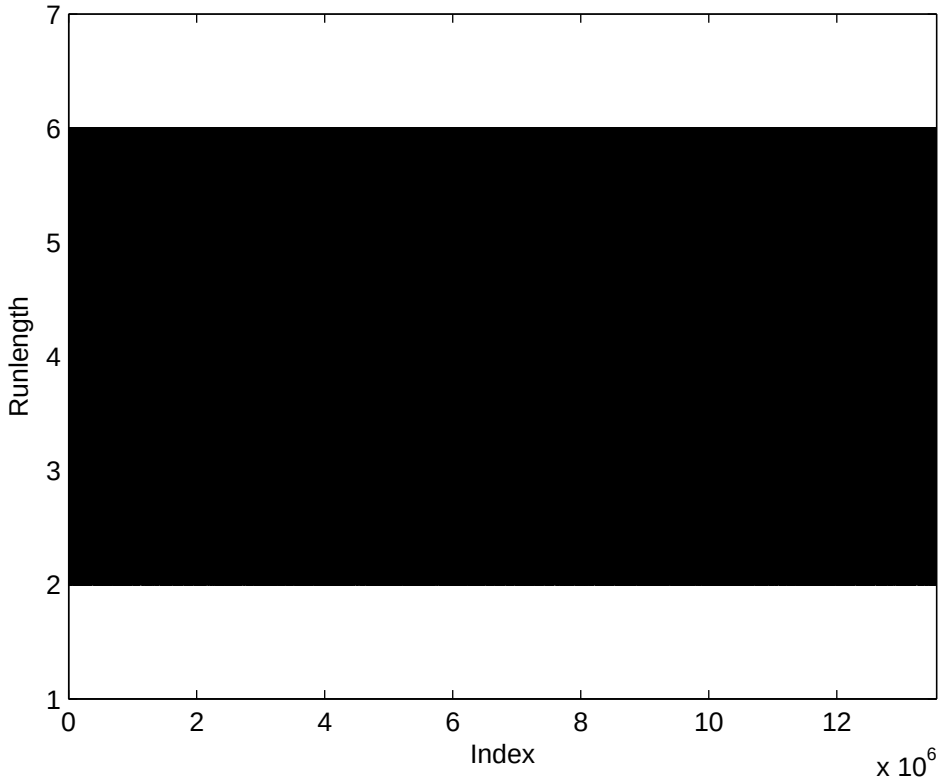

Figure 3.3: Runlengths within a DC-free RLL code with $d = 1, k = 5$ and $N = 7$.

Table 3.4: Portion of code table for $d = 1, k = 3, N = 5$ with $m = 8$ and $n = 20$.

	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9	σ_{10}	σ_{11}	σ_{12}	σ_{13}	σ_{14}	σ_{15}	σ_{16}
β_1	w_1	w_1	w_1	w_1	w_1	w_1	w_{874}	w_{874}	w_1	w_1	w_{874}	w_{874}	w_1	w_{874}	w_{874}	w_{874}
β_2	w_2	w_2	w_2	w_{535}	w_2	w_2	w_{983}	w_{621}	w_{535}	w_2	w_{983}	w_{983}	w_{535}	w_{983}	w_{983}	w_{983}
β_3	w_3	w_3	w_3	w_{786}	w_3	w_3	w_{985}	w_{786}	w_{1115}	w_3	w_{985}	w_{985}	w_{786}	w_{985}	w_{985}	w_{985}
β_4	w_4	w_4	w_4	w_{536}	w_4	w_4	w_{986}	w_{622}	w_{536}	w_4	w_{986}	w_{986}	w_{536}	w_{986}	w_{986}	w_{986}
β_5	w_5	w_5	w_5	w_{787}	w_5	w_5	w_{991}	w_{787}	w_{1116}	w_5	w_{991}	w_{991}	w_{787}	w_{991}	w_{991}	w_{991}
β_6	w_6	w_6	w_6	w_{537}	w_6	w_6	w_{992}	w_{623}	w_{537}	w_6	w_{992}	w_{992}	w_{537}	w_{992}	w_{992}	w_{992}
β_7	w_7	w_7	w_7	w_{788}	w_7	w_7	w_{994}	w_{788}	w_{1119}	w_7	w_{994}	w_{994}	w_{788}	w_{994}	w_{994}	w_{994}
β_8	w_8	w_8	w_8	w_{538}	w_8	w_8	w_{995}	w_{624}	w_{538}	w_8	w_{995}	w_{995}	w_{538}	w_{995}	w_{995}	w_{995}
β_9	w_9	w_9	w_9	w_{789}	w_9	w_9	w_{1009}	w_{789}	w_{1121}	w_9	w_{1009}	w_{1009}	w_{789}	w_{1009}	w_{1009}	w_{1009}
β_{10}	w_{10}	w_{10}	w_{10}	w_{539}	w_{10}	w_{10}	w_{1010}	w_{625}	w_{539}	w_{10}	w_{1010}	w_{1010}	w_{539}	w_{1010}	w_{1010}	w_{1010}
β_{11}	w_{11}	w_{11}	w_{11}	w_{790}	w_{11}	w_{11}	w_{1012}	w_{790}	w_{1122}	w_{11}	w_{1012}	w_{1012}	w_{790}	w_{1012}	w_{1012}	w_{1012}
β_{12}	w_{12}	w_{12}	w_{12}	w_{540}	w_{12}	w_{12}	w_{1013}	w_{626}	w_{540}	w_{12}	w_{1013}	w_{1013}	w_{540}	w_{1013}	w_{1013}	w_{1013}
β_{13}	w_{13}	w_{13}	w_{13}	w_{791}	w_{13}	w_{13}	w_{1014}	w_{791}	w_{1123}	w_{13}	w_{1014}	w_{1014}	w_{791}	w_{1014}	w_{1014}	w_{1014}
β_{14}	w_{14}	w_{14}	w_{14}	w_{541}	w_{14}	w_{14}	w_{1019}	w_{627}	w_{541}	w_{14}	w_{1019}	w_{1019}	w_{541}	w_{1019}	w_{1019}	w_{1019}
β_{15}	w_{15}	w_{15}	w_{15}	w_{792}	w_{15}	w_{15}	w_{1020}	w_{792}	w_{1124}	w_{15}	w_{1020}	w_{1020}	w_{792}	w_{1020}	w_{1020}	w_{1020}

3.6 Weakly Constrained Codes

While the algorithm has been overwhelmingly successful in generating DC-free RLL codes for various values of d, k, and N, it is possible that for some code tables, even modifying the parameters of the algorithm will not generate a state-independent decodable code. It may be the limitation of the algorithm relying on making locally optimum decisions, or it could simply be that it is not possible to find a code that can be decoded at the receiver without state information for the set of parameters with the highest code rate returned by the Franaszek algorithm. Of the 70 DC-free RLL codes for which the algorithm attempted to construct a code table, no codes fell into the former category, while only one code fell into the latter category. That is, for that code, it is not possible to find a code that can be decoded without state information for the set of parameters with the highest code rate. Without lowering the code rate, an alternative approach is to consider the design of a weakly constrained code [44]. For a weakly constrained code, the code designer no longer guarantees that the constraints are not violated, instead, the code designer relaxes the constraints somewhat by allowing them to be violated. Ideally, the violations are few and infrequent. When the algorithm is unsuccessful, generally speaking, few spaces remain in the table. If the algorithm is able to choose a codeword in such a manner that the violation is minimal, a reasonable code should be able to be constructed.

The weakly constrained process is implemented as follows. For each space remaining in the table, the code designer can insert any codeword already existing in that row or any codeword that is not used in the table. In these cases, a next state from P that approximates what would have been the actual next state must be assigned. No codeword will exist in two different rows, so the condition for state-independent decoding has not been violated.

This approach complements the proposed algorithm well because the output of the algorithm meshes well with the input required for a weakly constrained coding algorithm. That is, the proposed algorithm runs as it would normally, attempting to construct a code. If no set of parameters work, the best output is chosen and the principles of weakly constrained coding are applied after that point. The spaces remaining in the table are filled in according to the criteria chosen by the code designer, which can be implemented via a computer program. It is not necessary to re-design the proposed algorithm to handle weakly constrained coding, instead another module can be run thereafter.

While the code rate is not lowered, there will be a penalty incurred. For example, in the case of a DC-free code, its spectral performance will suffer, possibly through the appearance of a flooring effect at low frequencies in the spectrum. By using the algorithm described above to first complete the coding table as thoroughly as possible, however, this penalty should not be significant.

In particular, return to the case where $(d, k, N) = (3, 5, 8)$ with $m = 3$ and $n = 13$. For the sake of clarity the eight values representing the N constraint will be denoted by $\{-3, -2, -1, 0, 1, 2, 3, 4\}$ in the discussion below. To keep this example simple, only words already existing in the current row in the table are considered to be valid candidates to fill the remaining spaces. Recall that any unused codeword also constitutes a valid alternative, but to make the example easier to follow, this has been omitted as a possibility.

For this code, the algorithm generates the codebook shown in Table 3.5 without using any lookahead. Note that this table is shown in two parts for the purposes of presentation. There are two spaces remaining, indicated by "X" in the table, both occurring in the eighth row. The words used in this row include words 8, 10, 19, 34, 38, 41, and 50. The mapping of these codewords into bits is shown in Table 3.6. To complete a valid weakly constrained code, the encoder requires two things: a codeword for the table and a valid next state. A well-designed weakly constrained code will use codewords in each of the two remaining spaces, which will have a minimal impact on the overall operation of the code so that the constraints will be violated as infrequently as possible.

The first space is in state 15, which represents the state with a current runlength of a single logic one and a value of $N = 0$. Looking at the list of candidates in Table 3.6, the best two choices are words 19 and 34. Transmitting word 19 places the encoder in a state where it has two consecutive zeros and a value of $N = 1$, which corresponds to a next state of 20. However, the minimum runlength is violated as there are only three consecutive ones when the previous word and word 19 are concatenated together. Transmitting word 34 places the encoder in a state where it has three consecutive ones and a value of $N = 3$. Unfortunately, no principal state meets these requirements, the closest state being state 9, which has a current runlength of three consecutive ones and a value of $N = 2$. In this case, the code designer can choose to violate either the runlength condition or the DC-free condition, depending on which may be more valuable in the particular application.

Table 3.5: Code table for $d = 3, k = 5, N = 8$ with $m = 3$ and $n = 13$.

	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9	σ_{10}	σ_{11}	σ_{12}	σ_{13}	σ_{14}	σ_{15}
β_1	w_1	w_1	w_1	w_1	w_1	w_1	w_{17}	w_{17}	w_{17}	w_{29}	w_{29}	w_{17}	w_{29}	w_{29}	w_{29}
β_2	w_2	w_2	w_2	w_2	w_2	w_2	w_{18}	w_{18}	w_{18}	w_{30}	w_{30}	w_{18}	w_{30}	w_{30}	w_{30}
β_3	w_3	w_3	w_3	w_{14}	w_3	w_3	w_{26}	w_{26}	w_{20}	w_{26}	w_{26}	w_{26}	w_{32}	w_{26}	w_{26}
β_4	w_4	w_4	w_4	w_{15}	w_4	w_4	w_{27}	w_{15}	w_{15}	w_{27}	w_{27}	w_{15}	w_{27}	w_{27}	w_{27}
β_5	w_5	w_{12}	w_5	w_{16}	w_{16}	w_5	w_{21}	w_{16}	w_{16}	w_{28}	w_{28}	w_{16}	w_{35}	w_{28}	w_{28}
β_6	w_6	w_9	w_6	w_9	w_9	w_6	w_{23}	w_{23}	w_{11}	w_{22}	w_{23}	w_{23}	w_{33}	w_{33}	w_{23}
β_7	w_7	w_{13}	w_7	w_{13}	w_{13}	w_7	w_{25}	w_{13}	w_{13}	w_{25}	w_{25}	w_{24}	w_{31}	w_{25}	w_{25}
β_8	w_8	w_{10}	w_8	w_{19}	w_{10}	w_8	w_{19}	w_{19}	w_{10}	w_{19}	w_{19}	w_{19}	w_{34}	w_{34}	X

	σ_{16}	σ_{17}	σ_{18}	σ_{19}	σ_{20}	σ_{21}	σ_{22}	σ_{23}	σ_{24}	σ_{25}	σ_{26}	σ_{27}	σ_{28}	σ_{29}	σ_{30}
β_1	w_1	w_1	w_1	w_{44}	w_1	w_1	w_{44}	w_{44}	w_{44}	w_{29}	w_{29}	w_{29}	w_{29}	w_{29}	w_{29}
β_2	w_2	w_2	w_2	w_{43}	w_2	w_2	w_{43}	w_{43}	w_{43}	w_{30}	w_{30}	w_{30}	w_{30}	w_{30}	w_{30}
β_3	w_{40}	w_3	w_3	w_{42}	w_{42}	w_{42}	w_{53}	w_{42}	w_{42}	w_{32}	w_{53}	w_{42}	w_{32}	w_{53}	w_{32}
β_4	w_{39}	w_4	w_4	w_{39}	w_{39}	w_{47}	w_{51}	w_{39}	w_{39}	w_{54}	w_{51}	w_{51}	w_{54}	w_{51}	w_{54}
β_5	w_{37}	w_{37}	w_5	w_{37}	w_{37}	w_{37}	w_{49}	w_{37}	w_{37}	w_{35}	w_{28}	w_{28}	w_{35}	w_{28}	w_{35}
β_6	w_9	w_9	w_6	w_{46}	w_9	w_9	w_{46}	w_{46}	w_{48}	w_{33}	w_{33}	w_{46}	w_{33}	w_{33}	w_{33}
β_7	w_{36}	w_{36}	w_{36}	w_{45}	w_{36}	w_{36}	w_{45}	w_{45}	w_{36}	w_{55}	w_{45}	w_{45}	w_{55}	w_{52}	w_{55}
β_8	w_{38}	w_{38}	w_{41}	X	w_{38}	w_{38}	w_{50}	w_{50}	w_{38}	w_{34}	w_{34}	w_{50}	w_{34}	w_{34}	w_{34}

Table 3.6: Codeword mapping for $d = 3, k = 5, N = 8$ with $m = 3$ and $n = 13$.

Label	Codeword
w_8	0 0 0 0 0 0 1 1 1 1 1 1 0
w_{10}	1 0 0 0 0 0 0 1 1 1 1 1 0
w_{19}	1 1 0 0 0 0 1 1 1 1 1 0 0
w_{34}	1 1 1 1 1 0 0 0 0 0 1 1 1
w_{38}	0 0 0 1 1 1 1 1 0 0 0 0 0
w_{41}	0 0 0 1 1 1 1 0 0 0 0 0 0
w_{50}	0 1 1 1 1 0 0 0 0 0 1 1 1

The second space occurs in state 19, which has a current runlength of two consecutive zeros and a value of $N = 0$. Candidates include words 38 and 41. Considering word 38, the encoder ends in a state corresponding to five consecutive zeros and $N = -3$. The closest state to meeting these requirements is state 28, which matches the five consecutive zeros but has $N = -2$. For word 41, the encoder moves to a state with six consecutive zeros and $N = -5$. Recall that the lowest valid N constraint is -3, however, and thus the code must violate the DC-free constraint by at least -2. The state with six consecutive zeros and $N = -3$ is not a principal state and so the closest state is state 30 with six consecutive zeros and $N = -2$, which violates the DC-free constraint by -3. In this case, word 38 comprises the best selection.

There are two possible approaches to completing the weakly constrained code in this section. The first approach is to select word 19 for the first space, along with a next state of 20, which violates the runlength condition, then select word 38 to fill the second space, along with a next state of 28, which violates the DC-free condition. This particular mapping is chosen to demonstrate what the effects of violating each of the constraints. Another code designer may recognize that choosing word 34 for the first space, along with a next state of 9, causes the value of N to be one lower than the actual value of N. Selecting word 38 for the second space, along with a next state of 28, causes the value of N to be one higher than the actual value for N. Since the two violations oppose each other, optimistically, the weakly constrained code will perform very well. Weakly constrained codes were constructed for both options and are presented below.

The codebook has been completed with word 19 in the first space and a next state of 20, violating the runlength condition, along with word 38 in the second space and a next state of 28, which violates the DC-free condition. A simulation of five million codewords was performed to find the PSD of this code, as shown

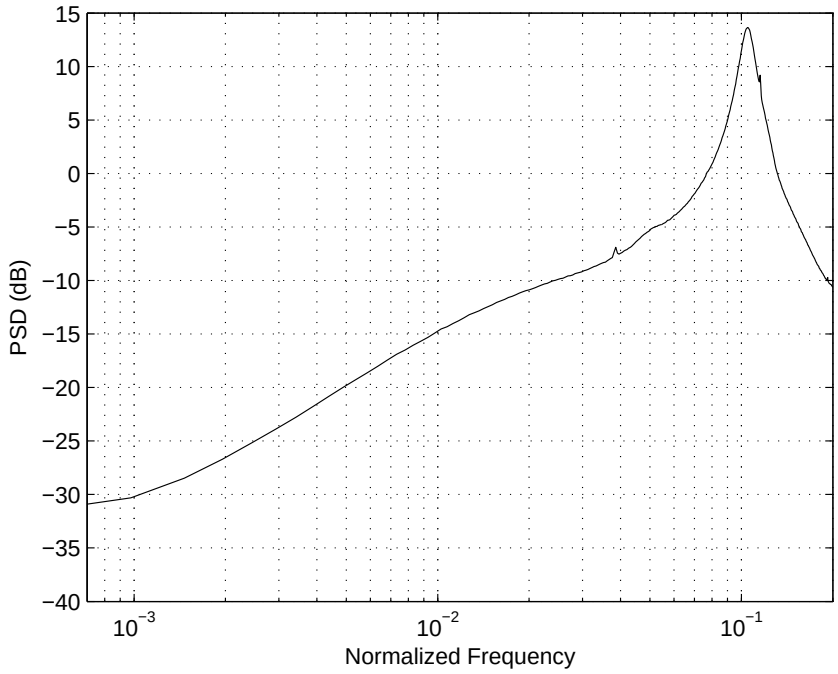

Figure 3.4: PSD of weakly constrained DC-free RLL code with $d = 3, k = 5$ and $N = 8$ using first approach.

in Fig. 3.4 using the codebook in Table 3.5 with the spaces filled in as specified. The flooring effect becomes somewhat visible at frequencies below 10^{-3}. To show the DC-free and runlength constraint violations, both the RDS and runlengths were plotted on a bit-by-bit basis through the simulation. The RDS values during the simulation are shown in Fig. 3.5, while a portion of the runlengths are shown in Fig. 3.6. The RDS violation is a summation of each time the word in row 8, column 19 is transmitted. This violation is fairly minor, causing an error on the scale of 10^4 over nearly 70 million bits. The plot of the runlengths is zoomed in to a particular segment to better show how infrequent the violations are. The segment was chosen randomly, as all of the segments demonstrate comparable performance. Attempting to show all of the runlengths simultaneously results in a plot that obscures the results. In both cases, the constraints are violated in 1 of 240 positions or approximately 0.417% of the time, on average, with equi-probable source words and assuming that each state is entered with roughly the same probability.

For the second weakly constrained code, the codebook has been completed with word 34 in the first space and a next state of 9, and word 38 in the second space with a next state of 28, violating the DC-free constraint only. Fig. 3.7 shows the

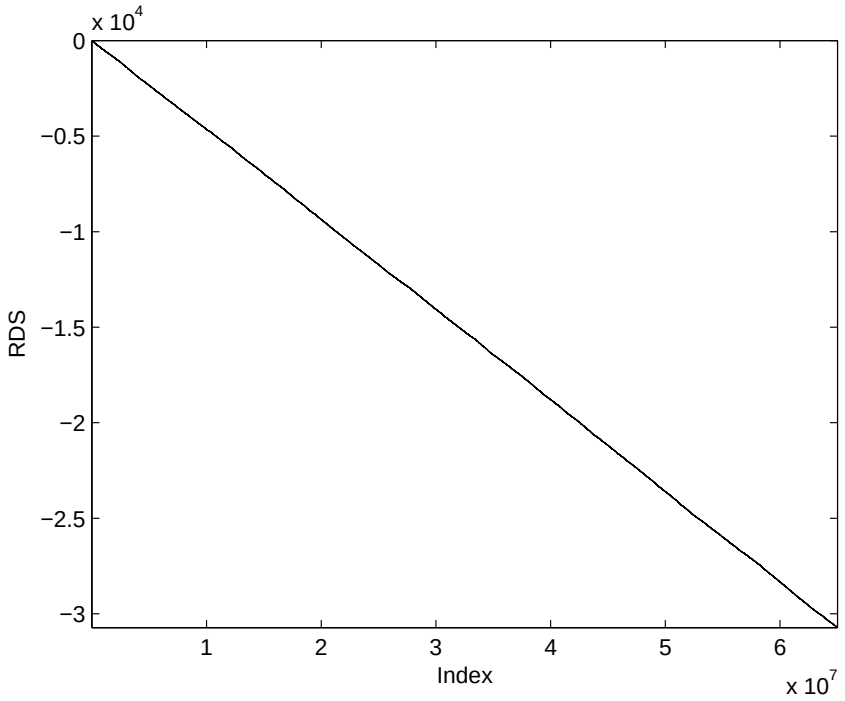

Figure 3.5: RDS of a weakly constrained DC-free RLL code with $d = 3, k = 5$ and $N = 8$ for five million codewords for the first approach.

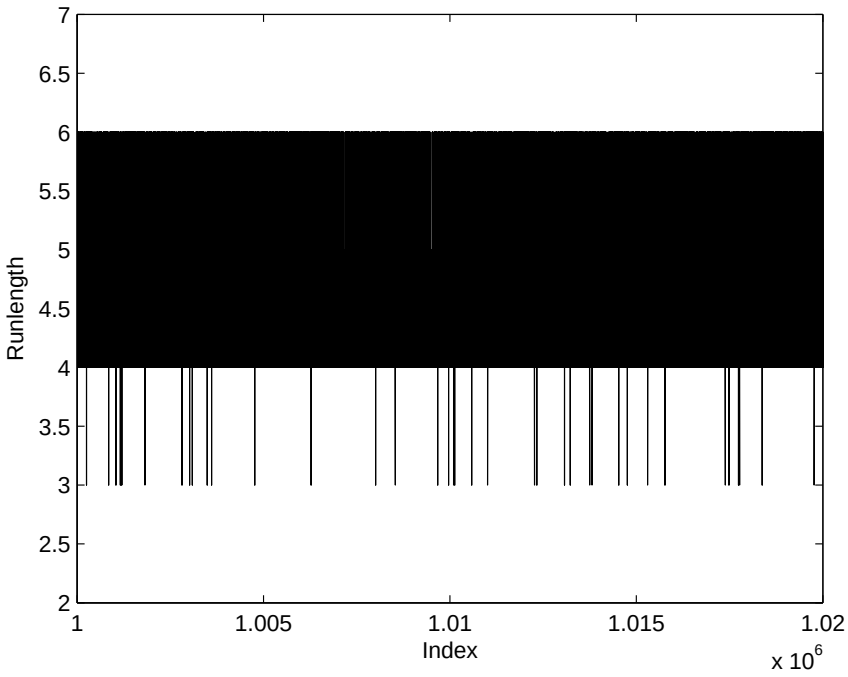

Figure 3.6: Runlengths of a weakly constrained DC-free RLL code with $d = 3, k = 5$ and $N = 8$ for five million codewords for the first approach.

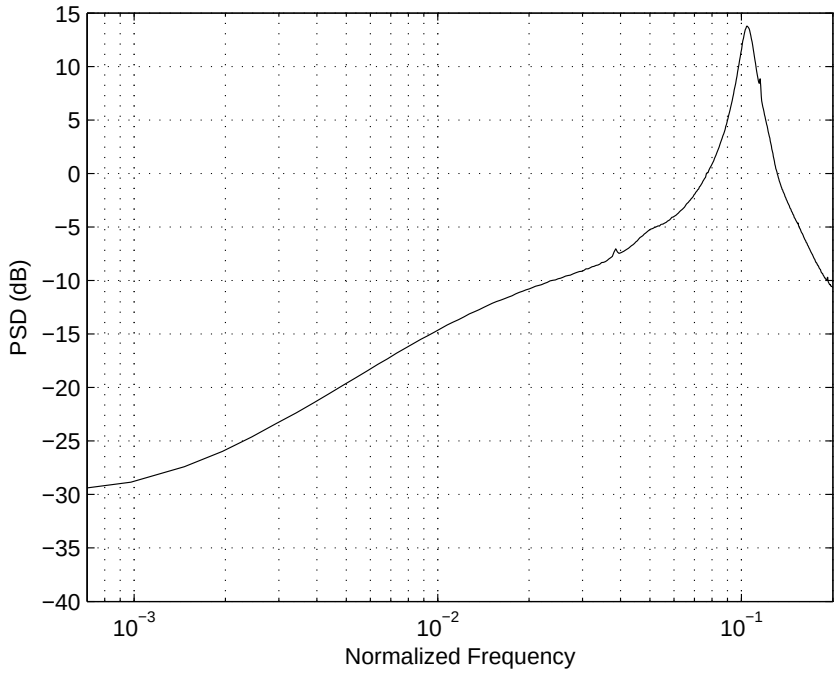

Figure 3.7: PSD of weakly constrained DC-free RLL code with $d = 3, k = 5$ and $N = 8$ for the second approach.

PSD of this code using a simulation of five million codewords. To better illustrate the DC-free violation, the RDS was plotted on a bit-by-bit basis throughout the simulation, as shown in Fig. 3.8. To illustrate the difference between the two codes, the RDS for both codes is included in this figure. Notice that, in the case of the second code, the error accumulated in the RDS is much lower overall. Also, the runlength does tend negative, and so it can be concluded that the state machine is more likely to be in state 19 than state 15. A plot of the runlengths is not included, since the constraint has not been violated, however, it was verified in the simulation that this was indeed the case.

For comparison purposes, the algorithm constructed a $(d, k, N) = (3, 5, 8)$ code with the set of parameters giving the second highest code rate, $m = 4$ and $n = 18$. The PSD of all three codes is shown in Fig. 3.9, demonstrating that the weakly constrained codes have a PSD very similar to that of the $m = 4$, $n = 18$ code, with the advantage that their rate is roughly 4% higher. With this in mind, the code designer can now evaluate the tradeoffs between code rate and frequency of constraint violation.

While the accumulated error of the second method is lower than that of the

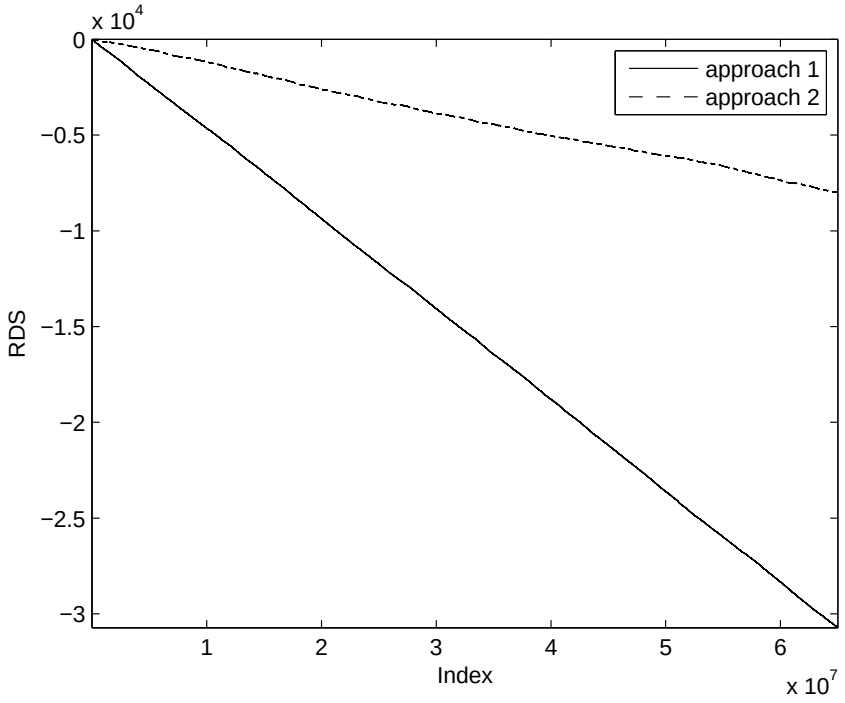

Figure 3.8: Comparison of RDS for a weakly constrained DC-free RLL code with $d = 3, k = 5$ and $N = 8$ with five million codewords for the two approaches.

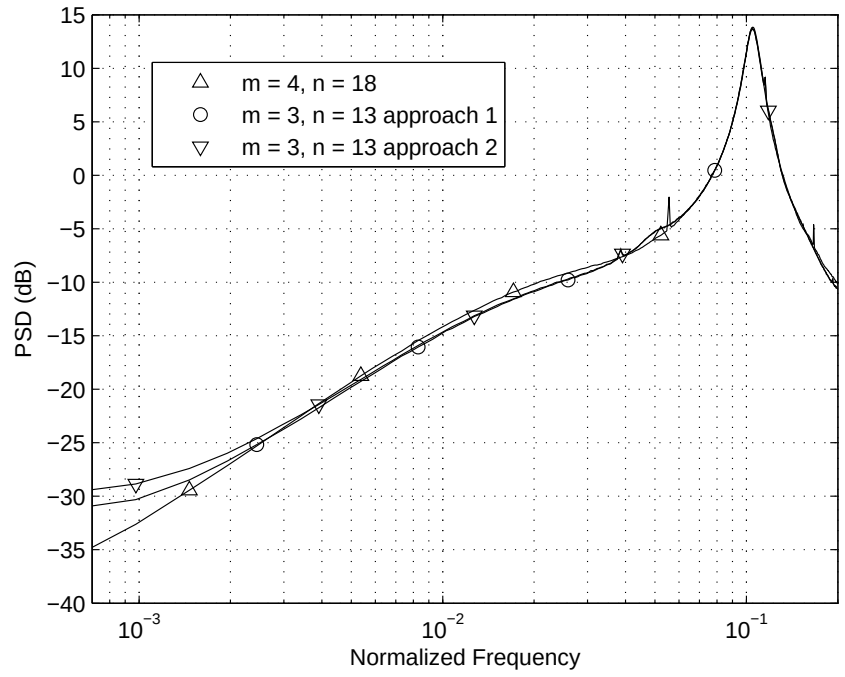

Figure 3.9: Comparison of PSD with two weakly constrained codes and a non-weakly constrained code.

first, the actual spectral performance is slightly worse, indicating that the spectral performance is not solely related to the slope of the accumulated RDS error. The spectrum of a code can be written as a Taylor expansion [45]. In this expansion, as frequency values decrease, different terms in the expansion become significant.

Between frequency values of 8×10^{-3} and 1×10^{-2}, the spectral performance is dominated by a term related to the low frequency spectrum weight (LFSW), which is related to the spectrum of the first derivative of the RDS evaluated at zero [45]. The RDS of this code can be estimated by $z_{\text{overall}} = z_{\text{original}} + z_{\Delta}$. Constructing a weakly constrained code does not change the original RDS bounds, rather the change in RDS can be represented using the z_{Δ} term. The LFSW is given largely by z_{original} and so the three codes look similar in this region.

Below frequency values of about 1×10^{-3}, another term, which is influenced largely by the RDS error z_{Δ}, begins to dominate the Taylor expansion. If the RDS error was the same in each encoding interval, the RDS error would be given by a straight line. In the actual coded sequence, this does not happen, since the RDS error accumulates only when one of the words that violates the constraint is transmitted. However, based on Fig. 3.8, it is reasonable to approximate the RDS error with a linear function. However, a second term, related to the variance of the error term around that line is needed, that is, $z_{\Delta} \approx z_{\Delta_{\text{line}}} + z_{\Delta_{\text{var}}}$. Fig. 3.10 shows a portion of Fig. 3.8 that has been zoomed in to more clearly demonstrate this variance. The spectral performance at frequencies below 1×10^{-3} is dominated by z_{Δ}, and so is influenced by not only the slope of the error line, but also the variability of the line. While the second weakly constrained code improves the first term in the RDS error, $z_{\Delta_{\text{line}}}$, the second term, $z_{\Delta_{\text{var}}}$, is more significant. As a result, the second approach has slightly worse spectral performance at very low frequencies than the first approach, but only violates the DC-free constraint, while the first approach violates both the DC-free and RLL constraints.

3.7 Summary

The construction of constrained codes that permit decoding without state information was discussed in this chapter. In particular, an algorithm that makes locally optimum decisions to maintain reasonable complexity was proposed. Building upon previous work by Franaszek, a code table containing a group of alphabets is constructed in such a manner that the spaces within the table are eliminated through intelligent searching and scoring techniques. This algorithm has a number of pa-

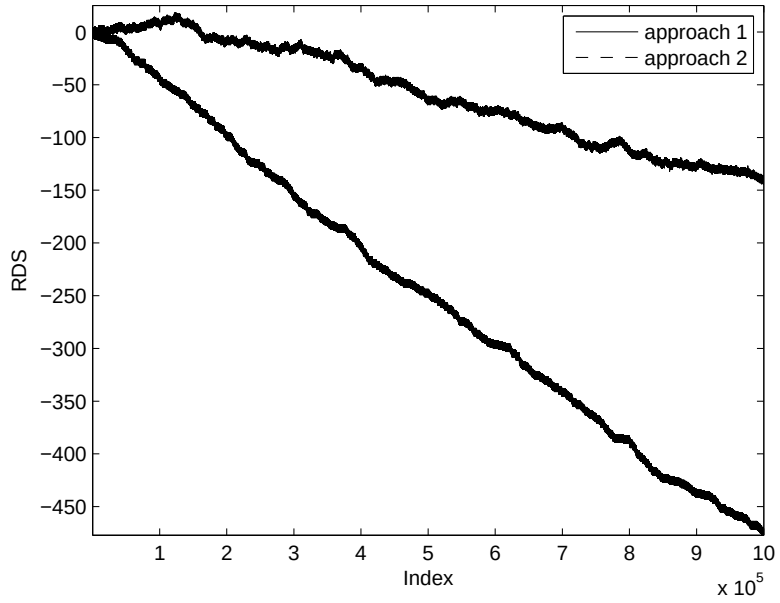

Figure 3.10: Comparison of RDS for a weakly constrained DC-free RLL code with $d = 3, k = 5$ and $N = 8$ with five million codewords for the two approaches zoomed in to show variability.

rameters, such as scoring exponents, scoring thresholds, and the selection of base rows that can be adjusted based on the type of codes being constructed by the code designer. Further, the scoring implementation allows the code designer freedom and flexibility to change the criteria according to their specific application. A number of optimizations were discussed, such as lookahead, state re-numbering, and weakly constrained coding. Finally, the usefulness of this algorithm was demonstrated with the construction of DC-free RLL codes that admit state-independent decoding. Many codes that fulfill all constraints were designed, and an example was also given of a code that follows the principles of weakly constrained codes.

Chapter 4

Evaluation of the Capacity of Constrained Codes with Multiple Constrained Signalling Dimensions

In this chapter, constrained codes using a signalling alphabet that is larger than binary, which can be regarded as containing symbols from multiple signalling dimensions, are considered. Representative systems that are considered include QPSK, 8 PSK, and 16 QAM. Note that this is a different situation than consideration of systems with signalling constrained in multiple physical dimensions, such as binary digits written on the two-dimensional surface of a disc or in three-dimensional volumetric holographic storage [46] [47] [48].

In the process of code design, the first major step is to evaluate the system capacity; in this chapter, the evaluation of the capacity of constrained codes with constraints in multiple signalling dimensions is considered. First, consider the case when signalling dimensions are independently constrained. This independence may occur naturally, as is the case of QPSK, or can be the result of an encoding process that regards the signalling constellation as several logical sub-constellations and enforces constraints on each logical sub-constellation independently. Evaluation of capacity is straightforward in this case.

Moving to more complex situations, analysis of the capacity of codes with multiple dependently constrained dimensions is considered. In these codes, there exists at least one state in which at least one symbol is impacted by or impacts the constraints in at least two dimensions simultaneously. It is demonstrated how the capacity of some of these systems can be accurately evaluated, and how the capacity of others can be estimated and upper bounded.

An overview of this chapter is as follows. State variables are reviewed in Sec-

tion 4.1, while DSV is discussed in more general terms as an RDS span in Section 4.2. The difference between multiple independently constrained logical dimensions and multiple dependently constrained logical dimensions is discussed in Section 4.3. In Section 4.4 the evaluation of the capacity of DC-free codes when the signalling alphabet consists of independently constrained logical dimensions is outlined. Presented in Section 4.5 are capacity calculations for the case where the signalling constellation has one or more signalling points that leads to dependently constrained logical dimensions. Finally, a summary is provided in Section 4.6.

4.1 FSM Encoders and State Variables

The state machines commonly used in constrained codes characterize their current state with one or more state variables. For example, in DC-free codes, all allowable RDS values, on a symbol-by-symbol basis, are enumerated and given a state, and encoding can be performed by tracking the current RDS and mapping the current source word to a codeword such that the RDS bounds are maintained. RLL codes can use a similar method, tracking the current state by the number of consecutive ones and zeros that have occurred. An example of a code with multiple state variables is a DC-free RLL code, where both the DC-free and RLL constraints are enforced by the encoder. In this case, the transmission of a one or zero changes the value of both state variables simultaneously, and so the two constraints cannot be easily separated. A method to model this type of constraint is given in Section 4.3.

4.2 Digital Sum Variation and RDS Span

One-dimensional FSMs for DC-free codes are commonly characterized by their DSV, N, which is the maximum number of different RDS values that the code can take on. A more general model, which is useful when considering multi-dimensional FSMs, is to consider the RDS bounds on the DC-free constrained code. In the one-dimensional case, for a given value of N, RDS bounds can be computed by associating each state with its RDS. To calculate the RDS associated with a state, a value is assigned to one particular state as a starting point (typically zero) and the RDS associated with all other states is computed by adding the value of the complex baseband representation of the symbol that has been transmitted. For example, for a binary code using symbols $\{+1, -1\}$ with a value of $N = 5$, where the middle (third) state is denoted as the zero point, states one and two have RDS values of -2

and -1, respectively, while states four and five have RDS values of $+1$ and $+2$, respectively.

To characterize the overall RDS bound, the largest span between RDS values is calculated, which is referred to as ΔZ. In the previous example, $\Delta Z = 2-(-2) = 4$, which is one less than the number of RDS values permissible. When considering the two-dimensional RDS plane, the RDS span can be characterized in terms of ΔZ, which is the largest overall span between permissible RDS values, or in terms of ΔZ_r and ΔZ_i, which is the largest RDS span in the real or imaginary dimensions, respectively. In general, it is the RDS span that is significant, not the actual RDS values since the initial state is arbitrarily assigned. It is also well established that the smaller the RDS span of the DC-free code, the greater the suppression of the continuous spectrum at low frequencies [2].

In this thesis, N, ΔZ_r, ΔZ_i, and ΔZ are used where appropriate. Typically, when describing a one-dimensional constraint, a value for the DSV N will be given. When describing multi-dimensional constraints simultaneously a characterization using the RDS span ΔZ is useful. However, when considering constraints in the real and imaginary dimensions separately, ΔZ_r and ΔZ_i are used.

4.3 Constraint Modelling with Multiple Signalling Dimensions

In this section, state machine models for constrained codes with multiple constrained signalling dimensions are considered. Two different types of constrained codes are considered: those with independently constrained dimensions and those with dependently constrained dimensions. These two cases are discussed and several examples of each are presented to provide context. Further, recall that the dimensions can occur as a result of natural independence in the signalling constellation, as in QPSK, or through the partitioning of a signalling constellation into a number of logical sub-constellations or dimensions for the encoder.

4.3.1 Constraint Modelling with Independent Signalling Dimensions

When modelling a constrained code with multiple signalling dimensions using an FSM, the overall operation of the FSM is described through multiple state variables. If, for all states, each dimensional constraint can be tracked with separate state

variables, and if each symbol affects the values of the state variables from only one dimension, the constraints are said to be modelled with independent dimensions.

Consider such a code modelled with K dimensions, $\{k = 1, 2, \ldots, K\}$, when the constraint in each dimension can be modelled with its own FSM with a connection matrix D_k. To construct the FSM describing the overall constraint, with connection matrix D, the FSMs for each dimension can be combined through a Cartesian product. This construction preserves the state transition structure of the constituent FSMs, allowing the states and state variables for each dimension to remain independent of one another. The Cartesian product creates "copies" of the FSM describing one dimension, joining these copies in accordance with the FSM of another dimension. The overall connection matrix can be found by considering the connections in the overall FSM, or by evaluating the Kronecker sum of the constituent connection matrices according to:

$$D = \sum_{k=1}^{K} I_{n_1} \otimes \ldots \otimes I_{n_{k-1}} \otimes D_k \otimes I_{n_{k+1}} \otimes \ldots \otimes I_{n_k}, k = 1, 2, \ldots K \qquad (4.1)$$

where D_k are the connection matrices for each of the dimensions, K is the total number of dimensions, and the I_{n_k} are appropriately sized identity matrices [49]. For example, for $K = 3$, the relationship becomes $D = D_1 \otimes I_{n_2} \otimes I_{n_3} + I_{n_1} \otimes D_2 \otimes I_{n_3} + I_{n_1} \otimes I_{n_2} \otimes D_3$.

Example: DC-free QPSK

A straightforward extension of a binary DC-free code is a DC-free QPSK code, where the running digital sum is constrained in both the real and imaginary dimensions of the complex baseband representation. Note that a null at DC in the complex baseband representation implies a null at the carrier frequency of the transmitted signal. This code is an example of a signalling constellation with two independently constrained signalling dimensions. The signalling points $\{+1, -1\}$ and $\{+j, -j\}$ naturally form two orthogonal signalling dimensions, when one state variable is defined to track the RDS in the real dimension and a second state variable is defined to track the RDS in the imaginary dimension. Each signalling point in the constellation affects only one of the state variables. A maximum DSV can be chosen for the number of different RDS values in each of the dimensions independently. These DSV values, which are denoted N_1 and N_2, dictate the size of the one-dimensional FSMs that describe the constraints in each of the independent dimensions, as depicted in Fig. 4.1 a) and b) when $N_1 = 4$ and $N_2 = 3$, respectively. The connection

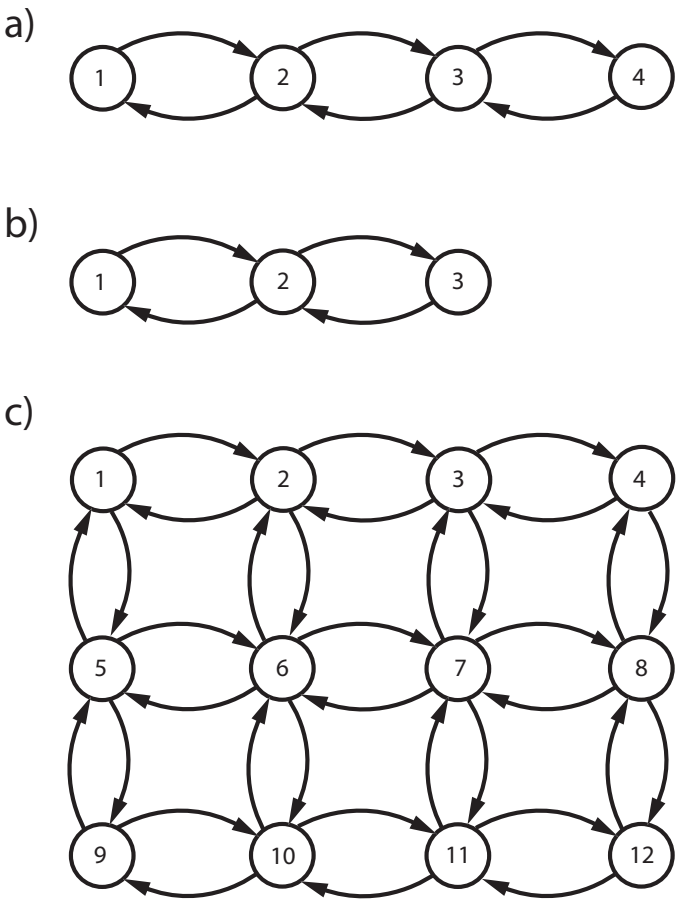

Figure 4.1: One-dimensional FSMs with (a) $N_1 = 4$ and (b) $N_2 = 3$. A two-dimensional FSM (c) is constructed by the Kronecker product of FSMs (a) and (b)

matrices for these one-dimensional FSMs are:

$$D_1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{4.2}$$

and

$$D_2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{4.3}$$

The FSM describing the overall constraint is depicted in Fig. 4.1 c) with overall

connection matrix:

$$D = \left[\begin{array}{cccccccccccc} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{array}\right]. \tag{4.4}$$

This connection matrix can be found by inspection of the FSM, or by using the two-dimensional Kronecker sum result:

$$D = D_1 \otimes I_{n_2} + I_{n_1} \otimes D_2 \tag{4.5}$$

where I_{n_1} and I_{n_2} are appropriately sized identity matrices (in this case, 3 by 3 and 4 by 4, respectively) and $\otimes$ is the Kronecker product operation. This is an example of a rectangular-type RDS bound since bounding the DSV on each dimension independently results in a rectangular FSM structure. In this example, the RDS span in the real dimension is $\Delta Z_r = 3$ and in the imaginary dimension is $\Delta Z_i = 2$. This gives an overall RDS span of $\Delta Z = \sqrt{13}$ for this code.

Example: DC-free 8PSK

The concept of independent signalling dimensions can be extended to larger signaling alphabets that are commonly used in communications. Consider the 8-PSK constellation shown in Fig. 4.2. If the state variables are defined in the same manner as in the QPSK case, the signalling points that are not on either axis alter the state variables of both dimensions simultaneously. This is an example of dependently constrained signalling dimensions, which is considered in the next subsection.

However, the manner in which the encoder operates can be changed to create a constrained code with multiple independently constrained logical dimensions. The encoder can construct a DC-free code by separating the signalling constellation into four logically independent dimensions by constraining the RDS of the following four pairs of points independently: $\{+1, -1\}$, $\{+j, -j\}$, $\{+\frac{1}{\sqrt{2}}+j\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}-j\frac{1}{\sqrt{2}}\}$, $\{+\frac{1}{\sqrt{2}} - j\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}} + j\frac{1}{\sqrt{2}}\}$. This creates four independent logical dimensions,

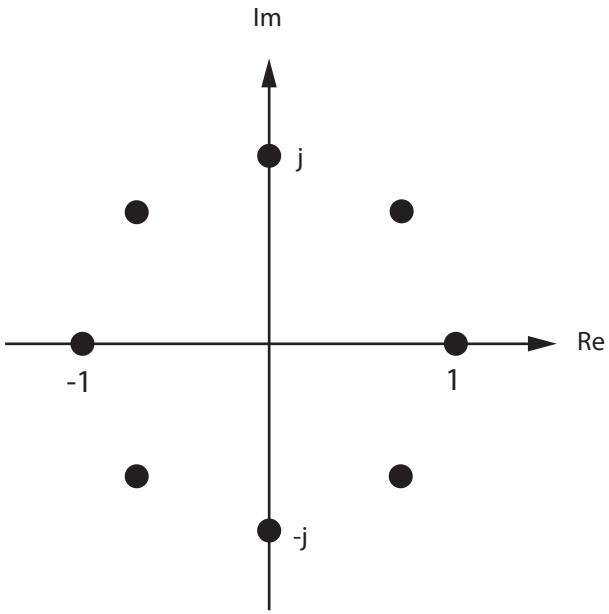

Figure 4.2: 8PSK signalling constellation.

with respect to the RDS constraints, where the state variable for each of the logical dimensions is the accumulated RDS in that dimension. Starting with a one-dimensional FSM that describes the constraint in each dimension, the overall FSM can be constructed as a four-dimensional FSM with an overall number of states equal to the product of the number of states in each dimension. The connection matrix of this FSM can be found using (4.1). The RDS span can be evaluated by determining the maximum distance between the complex RDS values represented by the four dimensional FSM. For example, if the DSV in each dimension is $N = 5$, then the RDS span is 4 for that dimension. Taking into consideration all four dimensions simultaneously, it can be shown that the overall RDS span for the code is $\Delta Z = 10.453$.

Example: DC-free 16 QAM

Consider the 16 QAM constellation, centered at the origin, with neighboring signalling points separated by a distance of two. Similar to the 8PSK case, defining two state variables to track the RDS values in the real and imaginary dimensions causes all signalling points to affect both state variables, and results in two dependently constrained dimensions. However, the encoder can be designed to regard the 16 QAM constellation as consisting of multiple independent logical signalling dimensions, and can bound the DSV independently on each of the logical dimensions.

There are at least two ways in which the encoder can partition the constellation into logical dimensions. In a manner similar to the approach for 8 PSK, the code can be partitioned into eight independently constrained dimensions by separating the signalling points into pairs with odd symmetry about the origin. A second partitioning consists of only six independently constrained logical dimensions. Note there are two sets of four points each that exist on straight lines through the origin: $\{-3-3j, -1-j, 1+j, 3+3j\}$ and $\{-3+3j, -1+j, 1-j, 3-3j\}$. The remaining eight points lie on four different signalling lines in pairs with odd symmetry about the origin: $\{1+3j, -1-3j\}$, $\{3+j, -3-j\}$, $\{-1+3j, 1-3j\}$, $\{-3+j, 3-j\}$. These six sets of points can be regarded as six independent dimensions, and the encoder can bound the DSV of each of these dimensions independently. With either approach, the overall connection matrix D can be calculated through the Cartesian product of the connection matrices representing the constituent FSMs.

4.3.2 Constraint Modelling with Dependent Signalling Dimensions

Dependent signalling dimensions arise when the state variables from different dimensions cannot be tracked separately or at least one symbol from at least one state affects the state variables assigned to two or more different dimensions. The FSMs describing these types of constrained codes can be significantly more complex to construct than the state machines which model independently constrained signalling dimensions. In general, FSMs describing dependently constrained signalling dimensions cannot be constructed simply through the use of Cartesian products; instead, care must be taken to ensure that the dependence between dimensions is modelled appropriately.

A drawback of codes using independently constrained signalling dimensions is that the RDS span is often larger than that of their dependently constrained counterparts. For example, consider again the 8PSK constellation using four independently constrained signalling dimensions, each with digital sum variation $N = 5$. It can be shown that this code has an RDS span in the real and imaginary dimensions of $\Delta Z_r = \Delta Z_i = 9.657$, and as mentioned above, it has an overall RDS span of $\Delta Z = 10.453$, whereas the RDS span on each of the independent dimensions is 4. However, the independently constrained model is useful to first generate an initial FSM, from which some states can be removed to enforce a tighter RDS span. Some examples are considered below.

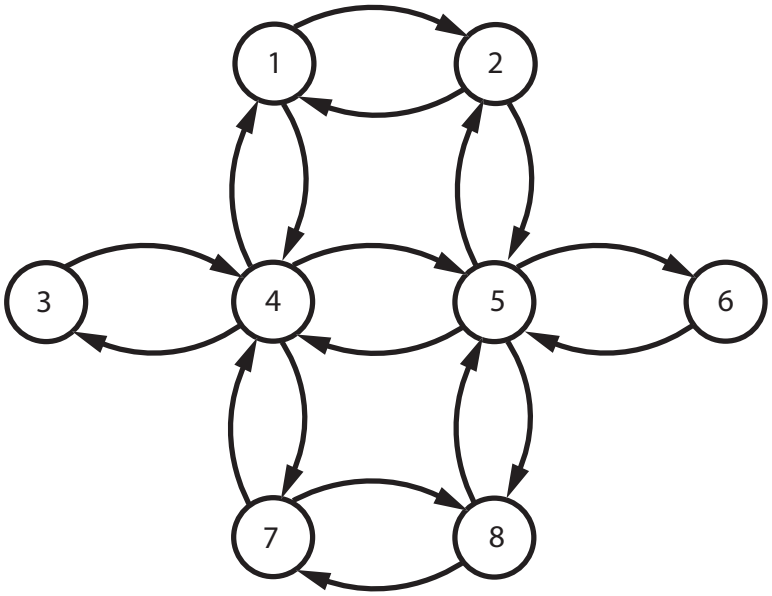

Figure 4.3: FSM for QPSK DC-free code with $N_1 = 4$ and $N_2 = 3$ and a circular-type RDS bound.

Example: DC-free QPSK

In the previous section, a QPSK code with a rectangular-type RDS bound was considered. However, this approach results in an overall RDS span that exceeds the constituent RDS spans owing to the magnitude of the states at the corners of the overall rectangular FSM. A circular-type (or elliptical) RDS bound produces a smaller overall span. To enforce a circular-type RDS bound, an FSM is constructed as described above, but all states that violate the prescribed RDS span are removed. This introduces a dependence into the signalling dimensions since, for example, the number of states in the horizontal direction is limited when the RDS of the vertical constraint is large, as compared to the number of states in the horizontal direction when the RDS in the vertical direction is small. This relationship is depicted in Fig. 4.3 for an RDS constraint of $N_1 = 4$ and $N_2 = 3$, when the corner states of the rectangular FSM are removed. The RDS span of this code is $\Delta Z = 3$.

Example: DC-free 8PSK

An 8PSK signalling constellation, using state variables to track the real and imaginary RDS values, naturally forms two dependently constrained signalling dimensions since the off-axis points in this constellation alter the values of both state variables. To construct a state machine that models the operation of the constrained code, an initial value for each of the state variables is chosen (typically zero), mini-

mum and maximum bounds on the RDS in both the real and imaginary dimensions are selected, and the RDS values (i.e. states) that can exist within the bounds are enumerated. Since the values of the constellation points are incommensurable in the real and imaginary dimensions, however, there exists an infinite number of states within the RDS span. Therefore, the straightforward approach of finding the connection matrix D and its maximum eigenvalue λ_{max} to evaluate the capacity of such a code cannot be used. In the next section, the estimation and upper bounding of the capacity of this type of constrained code is considered.

Example: DC-free 16 QAM

In the case of 16 QAM, two state variables are used, the RDS in the real and imaginary dimensions, to construct a code with two dependently constrained signalling dimensions. In this case, every signalling point has both real and imaginary components and therefore each symbol simultaneously affects both state variables. Since the signalling points are commensurable, then given a finite RDS span, the resulting state machine is described by a finite number of states. This state machine is constructed by choosing an initial starting state and enumerating all possible valid transitions. For example, starting at the state with zero accumulated RDS in both dimensions, selection of each of the 16 signalling points is considered. If the selection of a symbol results in an RDS value that lies within the RDS span, a new state is created, this transition is entered into the connection matrix, and the procedure is repeated at each new state until no additional states are created. During this process, new states are numbered and the connection matrix is constructed. The RDS constraint can be enforced using either rectangular-type or circular-type RDS bounds.

4.4 Capacity with Independently Constrained Signalling Dimensions

In this section, the evaluation of the capacity of a code with independently constrained signalling dimensions is considered. As described previously, when the dimensions are independent and the constraint imposed on the kth dimension by the encoder is modelled by an FSM with a connection matrix D_k, the overall connection matrix D is constructed by using (4.1). The largest eigenvalue λ_{max} of this connection matrix can then be found. Taking $\log_2$ of this value gives the capacity C of the code in bits of information per symbol.

Table 4.1: Maximum eigenvalue and capacity, in bits of information per symbol, of one-dimensional DC-free codes for $N = 2$ through 9.

N	λ_{max}	C
2	1.000	0.000
3	1.414	0.500
4	1.618	0.694
5	1.732	0.793
6	1.802	0.850
7	1.848	0.886
8	1.879	0.910
9	1.902	0.928

Alternatively, it is straightforward to evaluate λ_{max} if the maximum eigenvalues of the D_k are known. Since D is a Kronecker sum of the D_k, it can be shown that the eigenvalues of D are given by the summation of all possible combinations of eigenvalues from the D_k matrices [49]. As a consequence, λ_{max}, the maximum eigenvalue of D, is given by:

$$\lambda_{max} = \sum_{k=1}^{K} \lambda_{max_k} \tag{4.6}$$

where λ_{max_k} is the maximum eigenvalue for connection matrix D_k. Taking $\log_2$ of this value gives the capacity of the code in bits of information per symbol.

4.4.1 Examples: DC-free QPSK, 8PSK, and 16 QAM

It is straightforward to use the second approach when evaluating the capacity of the DC-free example codes in Section 4.3.1. Table 4.1 lists the maximum eigenvalue of the connection matrix D and capacity for several one-dimensional FSMs that model DC-free sequences with running digital sum variation N. Such tables are commonplace in the literature [cf. 2]. Eigenvalues listed in this table can be used to efficiently calculate the capacity of DC-free codes with independently constrained signalling dimensions.

For example, for the DC-free QPSK code with bounds of $N_1 = 4$ and $N_2 = 3$, the maximum eigenvalue is $\lambda_{max} = 1.618 + 1.414 = 3.032$, and $C = 1.600$ bits of information per symbol. Consider the 8PSK code with four independent logical signalling dimensions with a DSV of $N = 5$ on each of the dimensions. This code has a maximum eigenvalue of $\lambda_{max} = 4 \times 1.732 = 6.928$ and $C = 2.792$ bits of information per symbol. Table 4.2 lists the capacity of a few DC-free codes for an

Table 4.2: Maximum eigenvalue, capacity (in bits of information per symbol), and number of states in 8PSK DC-free codes with four independent signalling dimensions.

N	$\lambda_{\max}$	C	L
3	5.657	2.500	81
4	6.472	2.694	256
5	6.928	2.793	625
6	7.208	2.850	1296
7	7.391	2.886	2401
8	7.518	2.910	4096
9	7.609	2.928	6561

8PSK constellation considered as four independent dimensions, as well as the total number of states L. In this table, for simplicity, only results for codes when the DSV is the same in each dimension are listed.

Capacities for DC-free codes with symbols from the 16 QAM constellation depends on the partitioning that is chosen for the constellation. In the case of the eight dimension partitioning, the maximum eigenvalue for each of the dimensions can be obtained from Table 4.1 above. Care should be taken when selecting the appropriate value of N in each dimension because of the difference in RDS span due to the differing signalling amplitudes.

The six dimension partitioning resolves the difference in amplitudes in a different fashion. For the same DSV, the logical dimensions containing four points have different capacities than the dimensions with two signalling points. It is straightforward to calculate capacities of one-dimensional DC-free codes with four evenly-spaced signalling points; some results are listed in Table 4.3. The remaining four logical dimensions, each with two signalling points, are modelled as in the previous examples. The maximum eigenvalue of each dimension is obtained from Tables 4.1 and 4.3, and these values are summed together to find λ_{max} and the corresponding capacity. For example, if the dimensions with four signalling points are constrained to $N = 7$ and the dimensions with two signalling points are constrained to $N = 3$, the overall maximum eigenvalue is 11.581, and the capacity is 3.533 bits of information per symbol. For this code, the largest RDS value in either the real or imaginary dimensions is 14, and so $\Delta Z_r = \Delta Z_i = 28$. The overall RDS span is $\Delta Z = 28$. In this particular case, the real and imaginary RDS spans are equal to the overall RDS span since the worst case RDS span exists on the axes.

Table 4.3: Maximum eigenvalue, and capacity, in bits of information per symbol, for one-dimensional DC-free codes for $N = 4$ through 9 with multi-level signalling values of $\{\pm 1, \pm 3\}$.

N	$\lambda_{\max}$	C
4	2.000	1.000
5	2.450	1.293
6	2.732	1.450
7	2.962	1.567
8	3.140	1.651
9	3.274	1.711

4.5 Capacity with Dependently Constrained Signalling Dimensions

In the previous section, it was shown how the capacity of a DC-free code using signalling constellations consisting of multiple independently constrained dimensions can be evaluated. In this section, the evaluation of capacity for constrained codes with multiple dependently constrained dimensions is presented. These codes are characterized by state machines that have either a finite or infinite number of states, which are considered in turn in this section.

4.5.1 Finite Number of States

When a constrained code with dependently constrained signalling dimensions is described with an FSM, exact capacity analysis is possible through the methods detailed previously in this chapter. That is, the FSM and corresponding D matrix is constructed and its maximum eigenvalue is evaluated to find the capacity.

In Section 4.3.2, codes with dependently constrained signalling dimensions, which were constructed by first forming FSMs with rectangular bounds and then removing states that exceeded the RDS span, were considered. DC-free codes constructed in this manner with commensurable signalling points, such as QPSK or 16 QAM, tend to have significantly fewer states than their incommensurable counterparts. Tables 4.4 and 4.5 list the capacity of DC-free QPSK with circular-type RDS bounds and 16 QAM codes with rectangular-type RDS bounds and the six dimension partitioning. The relatively small number of states in these codes is due to the fact that many states along different dimensions have the same complex-valued RDS values and can be merged since they describe the same state. This simplifies the FSM. In the case of 8 PSK, however, the codes generally have a significantly

Table 4.4: Maximum eigenvalue and capacity, in bits of information per symbol, of QPSK DC-free codes with two dependent dimensions and RDS bounded with circular-type bounds.

N	ΔZ	$\lambda_{\max}$	C	L
3	2	2.000	1.000	5
5	4	3.000	1.585	13
7	6	3.482	1.800	29
9	8	3.690	1.884	49

Table 4.5: Maximum eigenvalue and capacity, in bits of information per symbol, of 16 QAM DC-free codes with six dependent dimensions and RDS bounded with rectangular-type bounds.

N	$\Delta Z_r, \Delta Z_i$	$\lambda_{\max}$	C	L
3	6	8.772	3.133	25
5	12	12.902	3.690	85
7	18	14.333	3.841	181
9	24	14.968	3.904	313

higher number of states. Table 4.6 lists the capacity of DC-free 8 PSK codes for several rectangular-type RDS bounds.

Fig. 4.4 shows a plot of the capacity of DC-free 8PSK codes using a signalling constellation split into four independent logical dimensions with rectangular-type two-dimensional RDS bounds on $\Delta Z_r, \Delta Z_i$, forcing dependent dimensions. Note that even for reasonably small RDS spans (for example $\Delta Z_r = \Delta Z_i = 8$, giving $C = 2.877$ bits of information per symbol) the capacity of the unconstrained system, $\log_2 8 = 3$, is approached. The short plateaus visible on the graph, for example from 3.0 to 3.4, exist because increasing the RDS span in this range does not increase the number of states in the FSM. There are a number of states with RDS values of $\sqrt{2}$, which results in a significant jump between 2.82 and 2.83, but no new states are added until the RDS span exceeds 3.42.

Table 4.6: Maximum eigenvalue and capacity, in bits of information per symbol, of 8 PSK DC-free codes with four dependent dimensions and RDS bounded with rectangular-type bounds.

N	$\Delta Z_r, \Delta Z_i$	$\lambda_{\max}$	C	L
3	2	4.168	2.060	37
5	4	6.110	2.611	289
7	6	6.955	2.798	1161
9	8	7.349	2.878	3301

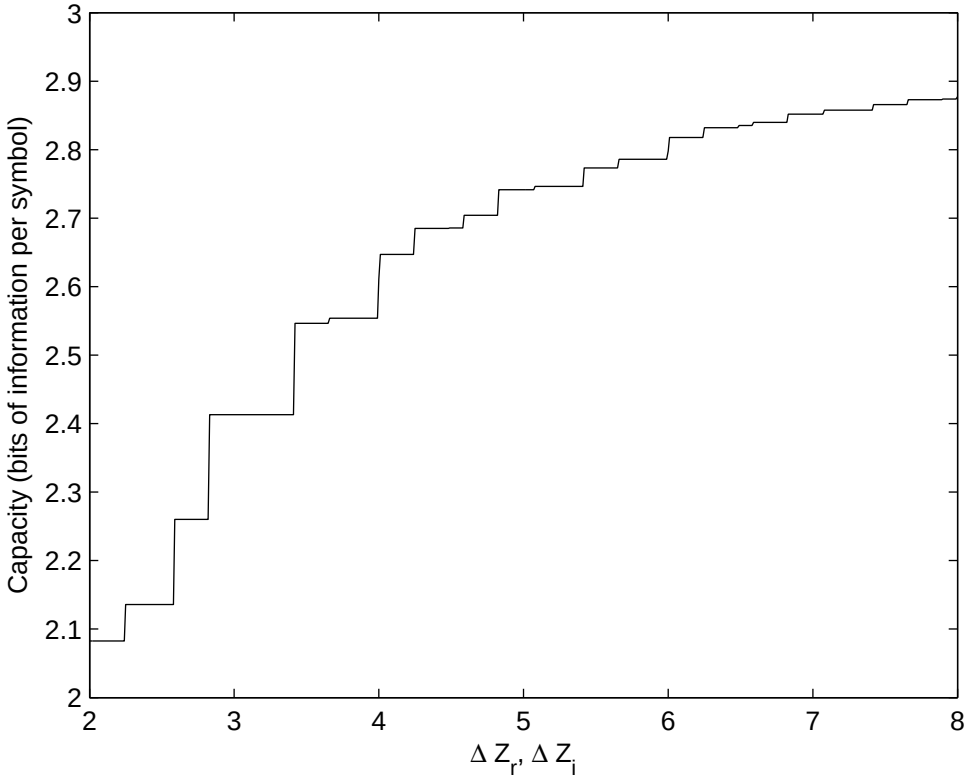

Figure 4.4: Capacity, in bits of information per symbol, as a function of RDS span for DC-free 8PSK codes with dependence introduced by using rectangular-type RDS bounds.

4.5.2 Infinite Number of States

In cases where two or more signalling points are incommensurable, construction of a state machine that enumerates all valid RDS values requires an infinite number of states. This type of situation arises with 8 PSK when the contribution of each point to the RDS in the real and imaginary dimensions is tracked. This subsection considers estimating the capacity of such systems through rounding, and upper bounding the capacity by partitioning the states into regions.

Estimating Capacity Using Rounding

In this approach, the number of states is limited by rounding the value of one or more of the signalling points to ensure that all signalling point values are commensurable. Using the rounded constellation points, an FSM is constructed by choosing an initial, valid state. From this state, all possible transitions are considered, and those that satisfy the constraint are retained. Valid transitions may correspond to a new state in the FSM, which is later analyzed in the same manner for all possible valid transitions. This process continues until no new states are created. The D matrix is constructed, its maximum eigenvalue is evaluated, and the corresponding capacity, which serves as an estimate of the capacity of the original system, is found.

For example, consider the 8PSK constellation where the real and imaginary dimensions are each bounded with an RDS span. Since the RDS contribution of the signalling points $\pm\frac{1}{\sqrt{2}} \pm j\frac{1}{\sqrt{2}}$ is incommensurable with the points on the axes, an infinite number of states arise. The capacity is estimated by rounding the value of $\frac{1}{\sqrt{2}}$ to 0.7, 0.71, and 0.705 to show the effect of increasing the number of states used in the approximate FSM. A rectangular-type bound is imposed on the RDS, extending from -1 to 1 in the real dimension and -1 to 1 in the imaginary dimension. While the resulting FSMs have a reasonably large number of states (221, 20201, and 80401 states, for rounding values 0.7, 0.71 and 0.705 respectively), it is possible to find the exact value of capacity for these systems. The corresponding maximum eigenvalues in these three cases are $\lambda_{max} = 4.2526, 4.2502,$ and 4.2502, giving rise to capacity estimates of $2.0883, 2.0875,$ and 2.0875 bits of information per symbol. Note that there is not a significant difference amongst these values, and so it can be inferred that the estimation of the capacity of this system is reasonably accurate.

To further validate this approach, consider Fig. 4.5, which shows a plot of the probability of being in each of the 221 states, in ascending probability, for the case

when a rounding value of 0.7 is used. These probabilities were obtained from simulation assuming equiprobable output symbols from each state. Notice the appearance of six distinct plateaus. These six plateaus correspond to six major regions of the FSM, with each state within these regions having approximately the same probability of occurrence. The states within each of these regions have similar probabilities of occurrence because they all share a common number of entry points; they also have the same number of output edges. These regions are presented visually in Fig. 4.6. Region 1 consists of a single point, $(0, 0)$; this is the only state with eight incoming edges. Region 2 consists of all points within $(\pm(1 - \frac{1}{\sqrt{2}}), \pm(1 - \frac{1}{\sqrt{2}}))$ lying on the axes; each state in this region has seven incoming edges. Region 3 consists of all points within $(\pm(1 - \frac{1}{\sqrt{2}}), \pm(1 - \frac{1}{\sqrt{2}}))$ excluding all points along the axes; these states have six incoming edges. Region 4 consists of all states along the axes with at least one coordinate having an absolute value greater than $(1 - \frac{1}{\sqrt{2}})$; these states have five incoming edges. Region 5 consists of all points not along the axes which have one coordinate with an absolute value smaller than $(1 - \frac{1}{\sqrt{2}})$ and one coordinate with an absolute value larger than $(1 - \frac{1}{\sqrt{2}})$; these states have four incoming edges. Finally, region 6 consists of all remaining states, where each state has three incoming edges. Regions 1 through 6 correspond to the plateaus of decreasing probability of occurrence in Fig. 4.5. Rounding the value of $(\frac{1}{\sqrt{2}})$ differently adjusts the size of these regions, but only slightly, and does not change the number or type of the regions. Further, the probability of being in any given region changes only slightly as the rounded value is adjusted.

In Chapter 2, it is shown that the entropy of a system is given by a summation of a sequence of terms, as in (2.5) on page 12. Recall that in the 8PSK example above, all states within each region have the same number of allowable output symbols, so they share a common H_i, and that there is a nearly uniform probability distribution across all the states in each region. Therefore, each of the six different regions contributes a term to this summation: $H\{X\} = \sum_{i=1}^{6} \pi_i H_i$, where π_i is the probability of being in a particular region and H_i is the entropy associated with that region. Since neither the size of the regions nor the probability of being in a particular region changes significantly as different values for rounding are chosen, the overall entropy of the system, $H\{X\}$, will not change significantly, nor will its maximization over the region probabilities, which is the capacity. For these reasons, it is concluded that the estimate obtained through the rounding approach provides a good approximation to the actual capacity of the code.

Recall that in subsection 4.5.1, above a DC-free 8PSK code with indepen-

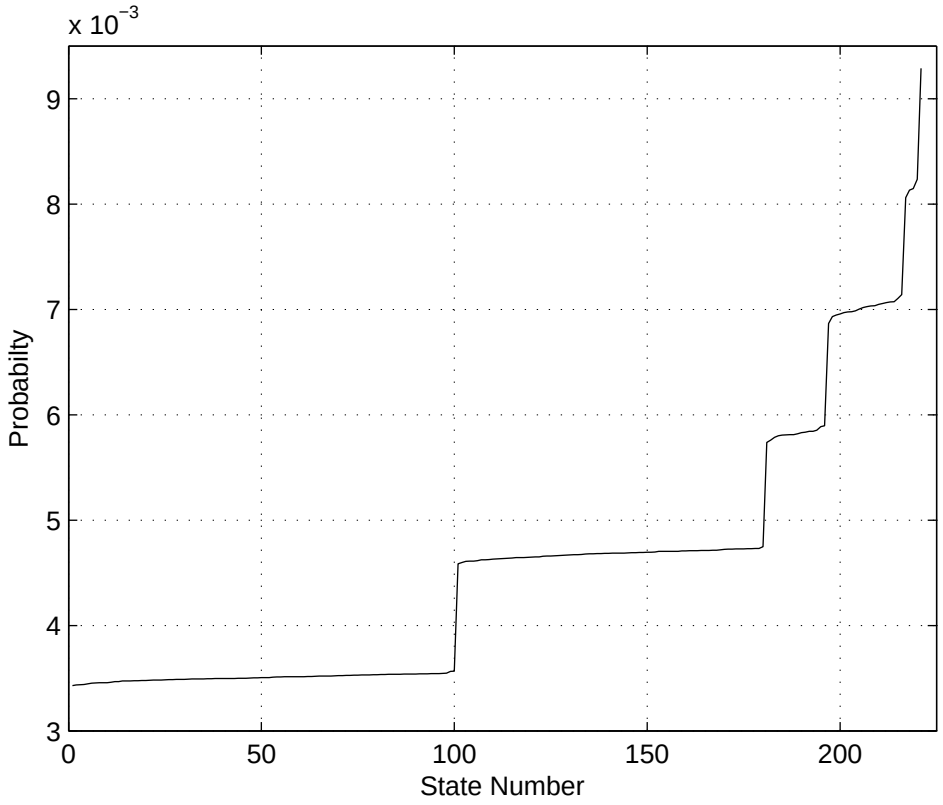

Figure 4.5: Steady-state probability of each of the 221 states in the FSM when $\frac{1}{\sqrt{2}}$ is rounded to 0.7.

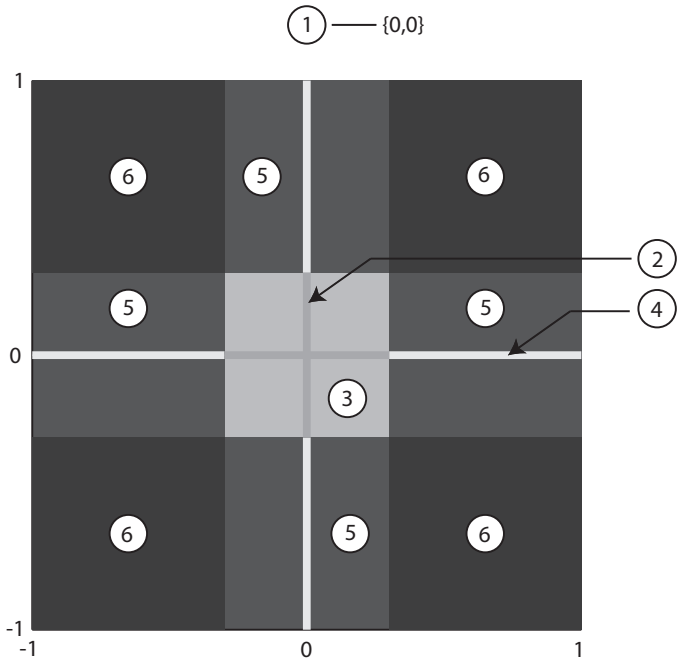

Figure 4.6: Graphical representations of the regions which contain a cluster of states with the same steady-state probability.

dently constrained signalling dimensions is considered, with states lying outside of a rectangular-type RDS span $\Delta Z_r = \Delta Z_i = 2$ being removed, forcing dependence among the four signalling dimensions. The result was a code with dependently constrained dimensions, 37 states, and a capacity of 2.060 bits of information per symbol. Note that this is within 1.5% of the estimated capacity of the DC-free 8PSK code constructed using the rounding approach with the same RDS span in both the real and imaginary dimenions. Given the similar capacities, the code with only 37 states might prove more practical. In that code, states have been removed from the FSM, but these states would not have provided a significant increase in the overall system capacity because they would have been entered with very low probability and had few exit paths.

Upper Bounding Capacity Using Regions

As outlined above, in some cases it is possible to partition states into regions if these groups of states have similar characteristics and properties. Each group of states can be represented by a "meta-state" and valid transitions among the meta-states can be considered. An edge from one meta-state to another is included if there is at least one state in the first meta-state that transitions to a state in the second meta-state. Therefore, the transitions among the meta-states may not be

valid for every state within the meta-states, but may represent a surplus of edges in the actual state machine. Since capacity is a function of the number of options available at any point in time, it is affected by the number of outgoing edges from the states. The system capacity will then be upper bounded if the outgoing edges from each meta-state represent at least all of the outgoing edges from every state they represent.

For example, if each of these six regions is represented with a meta-state in the DC-free 8PSK case considered above, and the outgoing transitions from these six states are constructed, the following connection matrix is obtained:

$$D = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{4.7}$$

Note, however, that this connection matrix overestimates the number of edges in the actual state machine. For example, no state in region 6 can transition to all six regions, only five. Some of the states within region 6 can transition to regions 1, 2, 3, 4 and 6, while other states in region 6 can transition to regions 1, 2, 3, 4 and 5. Overestimating the number of edges in this manner results in an upper bound on capacity. The maximum eigenvalue of this connection matrix is 4.322, giving an upper bound on the capacity of 2.111 bits of information per symbol. Note that the earlier estimate using the rounding approach is within 1.5% of this upper bound.

4.6 Summary

This chapter presented state machine modelling and capacity evaluation of constrained codes using larger-than-binary signalling alphabets. Two different methods for state machine modelling are outlined: codes using independently or dependently constrained signalling dimensions. In the case of independently constrained signalling dimensions, a straightforward approach for evaluation of the exact capacity of the system based on the summation of maximum eigenvalues of the connection matrices representing each independent signalling dimension was presented. In the case of dependently constrained signalling dimensions, both the evaluation and estimation of the system capacity were considered. Estimation was achieved using three methods: *i)* the removal of states from the state machine which contribute

insignificantly to the overall capacity and evaluation of the capacity of the resulting model, *ii)* rounding of constellation points so that symbols are commensurable, resulting in FSMs whose analysis is tractable, and *iii)* upper bounding the system capacity by partitioning the state machine into regions. A number of examples throughout this chapter were presented to provide context and show the applicability of this work to DC-free codes with signalling constellations using complex-valued symbols.

Chapter 5

DC-free Codes with Complex-Valued Signalling Constellations

In this chapter, DC-free codes using complex-valued signalling alphabets, including QPSK, 8PSK, and 16 QAM, that are state-independent decodable are constructed. Capacity analysis of such codes was considered in detail in Chapter 4. The FSMs constructed in that chapter serve as a starting point for generating the codebook of the codes presented in this chapter. Franaszek's algorithm [8] is used to find a suitable set of coding parameters for a state-dependent code, and to generate an encoding table. This encoding table is manipulated according to the approximation algorithm described in Chapter 3 to construct a code that will admit state-independent decoding at the receiver.

The encoding process that bounds the RDS is modelled with a finite state machine, where the state variables of this FSM are RDS values, since the encoder outputs codewords in response to the current RDS value and the input word. To calculate the capacity of such codes, the maximum eigenvalue of the connection matrix that describes this FSM is found [2] and the capacity, $C = \log_2 \lambda_{\max}$ [1], is evaluated, which has units of bits of information per symbol. Constructing a state-dependent encoder is straightforward when the edges of the FSM are used to enumerate all valid codewords for each state. To prevent error propagation at the receiver, the code is designed so that it is state-independently decodable, in that the code can be decoded at the receiver without the need for state information.

An overview of this chapter is as follows. In Section 5.1, the DSV and RDS span conventions used in this thesis are reviewed and sum variance is discussed, while a brief review of constraint modelling is given in Section 5.2. Section 5.3 presents several examples of DC-free QPSK codes, including their spectral plots. In Section 5.4, DC-free codes using 8PSK signalling alphabets are constructed,

while in Section 5.5, codes using the 16 QAM signalling alphabet are constructed. A summary is offered in Section 5.6.

5.1 RDS, DSV, and Sum Variance

As in the previous chapter, the DSV and RDS span are used to model the constraint imposed by the code. The conventions remain the same in this chapter: DSV is typically used when referring to a one-dimensional code, while $\Delta Z_r, \Delta Z_i$, and ΔZ are used when referring to codes with a two-dimensional RDS, depending on the type of RDS bound used.

It has been shown that for binary DC-free codes, the variance of the RDS, s_z^2, is a good indication of the spectral performance of the code [22]. The variance of the RDS is commonly referred to as the sum variance [2]. In particular, [22] shows that for most one-dimensional codes, $2s_z^2\omega_0 \approx 1$, where ω_0 is defined as the cut-off frequency, which is the frequency where the PSD is equal to one half, that is, the value of ω_0 when $H_x(\omega_0) = 1/2$. In the case of two-dimensional DC-free codes, the variance of a zero-mean sequence of complex-valued RDS values is given by $s_z^2 = \mathrm{E}[d_C^2]$ where d_C is the Cartesian distance of the RDS values from the origin. In other words, the variance of the RDS is related to the average squared distance of the RDS from the origin of the code. Note that the evaluation of the sum variance for binary codes is just a special case of this more general definition.

5.2 Constraint Modeling with DC-free Codes Using Multiple Signalling Dimensions

In the previous chapter, methods to calculate the capacity of DC-free codes using QPSK, 8PSK, and 16 QAM alphabets were developed. To find the capacity of these types of codes, FSMs modelling the constraints of the codes were generated. The FSMs generated during that process now serve as a starting point for the code construction process. These codes are separated into two categories: those with independently constrained signalling dimensions and those with dependently constrained signalling dimensions. Dependent signalling dimensions arise when the state variables from two or more dimensions cannot be tracked separately, or at least one symbol from at least one state affects the state variables assigned to two or more different dimensions. Conversely, codes using independent signalling dimensions are those in which every symbol affects the value of the state variables

from only one signalling dimension at a time. These independent signalling dimensions could result from the signalling constellation, such as the real and imaginary dimensions in the complex plane, or as a result of the encoding process, where the signalling constellation is subdivided into several logical sub-constellations.

Capacity evaluation and FSM construction of codes using QPSK alphabets is largely straightforward since the constellation consists of two independent sub-constellations, one on the real axis and the second on the imaginary axis. Therefore, in their simplest form these codes naturally fit the independently constrained dimensions model. Additional constraints can be imposed on the FSM that may cause this independence to be lost.

Capacity evaluation and FSM construction of 16 QAM codes is slightly more complex, since the signalling constellation does not have the independence inherent in QPSK codes. Each signalling point on the traditional 16 QAM constellation affects the RDS in both the real and imaginary dimensions simultaneously. However, it is possible to construct an FSM for these codes using the independently constrained dimensions method as a starting point. It is done so by partitioning the signalling constellation into several logically independent dimensions, highlighted in the previous chapter. For example, one dimension is composed of the signalling points $\{3 + j3, 1 + j1, -1 - j1, -3 - j3\}$; the five other dimensions also consist of sets of points that exist on lines through the origin. If the encoder enforces the RDS constraint on each of the dimensions independently, then these dimensions are independent of one another. An FSM is constructed that models this operation. If required, to enforce a tighter overall RDS span, states that violate this constraint are removed from the overall FSM, introducing dependence into the code.

Codes using 8PSK alphabets can be more difficult to model because of the presence of signalling points whose values are incommensurable, and therefore a model that tracks all possible RDS values requires an infinite number of states. While it is possible for the encoder to consider the 8PSK constellation as four independently constrained logical dimensions, the overall RDS span with that approach is significantly larger than the dependently constrained case. In this thesis, only 8PSK codes using dependently constrained signalling dimensions and a finite number of states are considered. These FSMs are constructed using one of two methods. First, rounding of signalling points so that their values are commensurable to ensure that there is a finite number of states is considered. Second, construction of the FSM starts with independently constrained logical dimensions as a starting point. Then a tighter RDS span is enforced by removing states that violate the desired RDS span

constraint at the cost of introducing dependence between the dimensions.

5.3 Construction of DC-free Codes Using QPSK Signalling Alphabets

In this section, the construction of DC-free codes using the QPSK signalling alphabet is considered. These codes have the simplest construction of the cases considered in this thesis because the signalling constellation naturally forms two independent sub-constellations. QPSK DC-free codes using rectangular-type RDS bounds have independently constrained dimensions. Other types of RDS bounds, such as a circular bound that restricts the magnitude of the RDS from the zero point, introduce dependence into the signalling dimensions, but use the independently constrained model as a starting point.

Constructing a DC-free QPSK code starts by choosing a desired RDS span on each of the real and imaginary dimensions, and modelling symbol-by-symbol movement in each of the two dimensions with individual one-dimensional FSMs. To construct the FSM describing the overall code, the FSMs on each of the two dimensions are combined through a Cartesian product. This construction results in rectangular-type RDS bounds. If circular (or other) bounds are required, states that violate these bounds are removed from the overall FSM at this point. As shown in the previous chapter, it is straightforward to calculate the D matrix, maximum eigenvalue, and capacity of such a code. The D matrix describing the FSM is used as an input to Franaszek's algorithm, which will return the set of parameters that give the highest code rate of a viable state-dependent decodable code. Finally, using these parameters, a state-independent decodable code is constructed using the approximation algorithm in Chapter 3.

For illustration purposes, consider a DC-free QPSK code with rectangular bounds $\Delta Z_r = \Delta Z_i = 2$ such that $\Delta Z = 2\sqrt{2}$. Using the approaches outlined in Chapter 4, it is straightforward to find that this code has a capacity of 1.5 bits of information per symbol. Franaszek's algorithm determines that, for $n \leq 10$, the highest code rate is given by $m = 13$ and $n = 9$. The tables specifying the codebook for such a code, however, would be lengthy. Instead, Franaszek's algorithm continues until a simpler code emerges, one for which it is practical to list the codebook. The construction of this simple code has parameters $m = 4$ and $n = 3$, which has a code rate of $R = 1.333$ binary digits per symbol. This code has an efficiency of $\nu = 0.889$, where efficiency is evaluated as $\nu = \frac{R}{C}$ with units binary digits per bit

Table 5.1: Codebook for QPSK DC-free code with $m = 4, n = 3$, and $\Delta Z_r = \Delta Z_i = 2$ as a function of encoding state σ_i.

source	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9
β_1	w_1	w_1	w_{27}	w_1	w_1	w_{27}	w_1	w_1	w_{27}
β_2	w_2	w_2	w_2	w_2	w_2	w_2	w_{51}	w_{51}	w_{51}
β_3	w_3	w_3	w_{21}	w_3	w_{44}	w_{44}	w_{50}	w_{44}	w_{44}
β_4	w_4	w_4	w_{29}	w_4	w_{42}	w_{42}	w_{52}	w_{42}	w_{42}
β_5	w_5	w_5	w_{24}	w_5	w_5	w_{41}	w_{38}	w_{41}	w_{41}
β_6	w_6	w_{19}	w_{19}	w_6	w_{45}	w_{45}	w_{32}	w_{45}	w_{45}
β_7	w_7	w_7	w_{26}	w_7	w_7	w_{43}	w_{53}	w_{43}	w_{43}
β_8	w_8	w_8	w_{30}	w_8	w_8	w_{46}	w_{39}	w_{39}	w_{46}
β_9	w_9	w_{17}	w_{17}	w_9	w_{17}	w_{17}	w_9	w_{17}	w_{17}
β_{10}	w_{10}	w_{10}	w_{10}	w_{34}	w_{34}	w_{34}	w_{34}	w_{34}	w_{34}
β_{11}	w_{11}	w_{18}	w_{18}	w_{11}	w_{18}	w_{18}	w_{37}	w_{37}	w_{48}
β_{12}	w_{12}	w_{20}	w_{20}	w_{36}	w_{36}	w_{47}	w_{36}	w_{36}	w_{47}
β_{13}	w_{13}	w_{22}	w_{22}	w_{31}	w_{22}	w_{22}	w_{31}	w_{55}	w_{55}
β_{14}	w_{14}	w_{25}	w_{25}	w_{40}	w_{25}	w_{25}	w_{40}	w_{56}	w_{56}
β_{15}	w_{15}	w_{15}	w_{23}	w_{33}	w_{33}	w_{23}	w_{33}	w_{33}	w_{54}
β_{16}	w_{16}	w_{16}	w_{28}	w_{35}	w_{35}	w_{28}	w_{35}	w_{35}	w_{49}

of information. Franaszek's algorithm has determined that there are nine principal states. The minimum number of codewords in any state is 16, and so the approximation algorithm attempts to fill the code table with 16 words across 9 states. After placing the 16 words of state one in the table, the table contains 90 spaces in the $16 \times 9 = 144$ total entries, which the algorithm successfully fills. Table 5.1 presents the codebook for one possible implementation of such a code that has nine principal states denoted $\sigma_i = 1, 2, \ldots 9$. Codewords are denoted as $w_j, j = 1, 2, \ldots 56$; source words are denoted as $\beta_k, k = 1, 2, \ldots 16$. The three-symbol sequence for each codeword is presented in Table 5.2; the mapping of four-bit binary sequence to source word tag is arbitrary. It can be verified that regardless of the source statistics this code is DC-free and, because each codeword appears in only one row of the code table, the code does not require state information in order to be decoded.

Table 5.3 lists parameters of a few DC-free QPSK codes, using both rectangular and circular RDS bounds, that were constructed using this method. The parameters listed include the RDS span, capacity, R, ν, as well as values for m, n, and number of principal states, $|P|$. In Table 5.3, as well as all subsequent tables in this chapter, the units for capacity, rate, and efficiency are bits of information per symbol, binary digits per symbol, and binary digits per bit of information, respectively. For rectangular bounds, the highest efficiency codes that have been constructed using

Table 5.2: Word index to codeword mapping for QPSK code with $\Delta Z_r, \Delta Z_i = 2$, $m = 4$ and $n = 3$.

Label	Codeword			Label	Codeword		
w_1	-1	$+1$	-1	w_{29}	$+1$	$-j$	$+1$
w_2	$-j$	$+j$	$-j$	w_{30}	$-j$	$+1$	$+1$
w_3	-1	$+1$	$-j$	w_{31}	$+j$	-1	$+1$
w_4	-1	$-j$	$+1$	w_{32}	$+j$	-1	-1
w_5	-1	$-j$	$+j$	w_{33}	$+j$	-1	$-j$
w_6	$-j$	$+j$	-1	w_{34}	$+j$	$-j$	$+j$
w_7	$-j$	-1	$+1$	w_{35}	$+j$	$-j$	-1
w_8	$-j$	-1	$+j$	w_{36}	-1	$+1$	$+j$
w_9	-1	-1	$+1$	w_{37}	-1	$+j$	$+1$
w_{10}	$-j$	$-j$	$+j$	w_{38}	-1	$+j$	-1
w_{11}	-1	-1	$-j$	w_{39}	-1	$+j$	$-j$
w_{12}	-1	$-j$	-1	w_{40}	-1	-1	$+j$
w_{13}	-1	$-j$	$-j$	w_{41}	$+1$	$+j$	-1
w_{14}	$-j$	-1	-1	w_{42}	$+1$	$+j$	$-j$
w_{15}	$-j$	-1	$-j$	w_{43}	$+1$	-1	$+j$
w_{16}	$-j$	$-j$	-1	w_{44}	$+j$	$+1$	-1
w_{17}	$+1$	-1	$+1$	w_{45}	$+j$	$+1$	$-j$
w_{18}	$+1$	-1	$-j$	w_{46}	$+j$	$-j$	$+1$
w_{19}	$+1$	$-j$	$+j$	w_{47}	$+1$	$+1$	$+j$
w_{20}	$+1$	$-j$	-1	w_{48}	$+1$	$+j$	$+1$
w_{21}	$+1$	$-j$	$-j$	w_{49}	$+j$	$+1$	$+1$
w_{22}	$-j$	$+1$	$+j$	w_{50}	$+j$	$+j$	-1
w_{23}	$-j$	$+1$	-1	w_{51}	$+j$	$+j$	$-j$
w_{24}	$-j$	$+1$	$-j$	w_{52}	$+j$	-1	$+j$
w_{25}	$-j$	$+j$	$+1$	w_{53}	-1	$+j$	$+j$
w_{26}	$-j$	$-j$	$+1$	w_{54}	$+1$	$+j$	$+j$
w_{27}	$+1$	$+1$	-1	w_{55}	$+j$	$+1$	$+j$
w_{28}	$+1$	$+1$	$-j$	w_{56}	$+j$	$+j$	$+1$

Table 5.3: Parameters of QPSK DC-free codes with $\Delta Z_r, \Delta Z_i = 2, 4$ and 6.

Rectangular Bounds

$_{\Delta Z_r \Delta Z_i}$	m	n	R	C	ν	$\lvert P \rvert$	s_z^2	f_0
2	4	3	1.333	1.500	0.889	9	1.021	0.0824
2	10	7	1.429	1.500	0.952	9	1.009	0.0895
4	10	6	1.667	1.793	0.930	21	2.286	0.0343
6	16	9	1.778	1.886	0.943	25	3.763	0.0222

Circular Bounds

ΔZ	m	n	R	C	ν	$\lvert P \rvert$	s_z^2	f_0
4	3	2	1.500	1.585	0.946	4	1.250	0.0715
6	10	6	1.667	1.807	0.923	21	2.397	0.0362

$\Delta Z_r, \Delta Z_i = 2, 4,$ and 6 have efficiencies of $95.2\%, 93.0\%,$ and 92.8%, respectively. Using circular bounds, DC-free QPSK codes with $\Delta Z = 4$ and 6 with effiencies of 94.6% and 92.3% have been constructed.

Fig. 5.1 shows the spectral performance of each of these codes based on a simulation of five million codewords, assuming equi-probable source symbols. The validity of the simulations for the simpler codes has been confirmed through comparison with analytical results computed using the approach of [11] extended to include consideration of complex-valued coded symbols. Values of sum variance, s_z^2, and cutoff frequency, f_0, were also obtained from these simulations. As expected, as the RDS span increases, the width of the spectral notch at DC decreases. Also note that for the same $\Delta Z_r, \Delta Z_i$, the codes using circular bounds exhibit better spectral performance than those using rectangular bounds. In particular, when $\Delta Z = 4$, the code using circular bounds has approximately 5 dB more spectral suppression than the code with rectangular bounds, while the code with $\Delta Z = 6$ has approximately 3 dB more spectral suppression than its rectangular bound counterpart.

When comparing the spectral performance of these QPSK codes, notice that, similar to one-dimensional codes, the variance of the RDS gives a very good indication of the spectral performance. For example, the codes constructed with circular RDS bounds have lower sum variance than those constructed with rectangular RDS bounds for the same ΔZ, which evidences itself in terms of better spectral performance. Taking the values for s_z^2 and f_0 and calculating $2s_z^2\omega_0$ gives values ranging from 1.014 to 1.135, indicating that Justesen's relationship holds for QPSK DC-free codes.

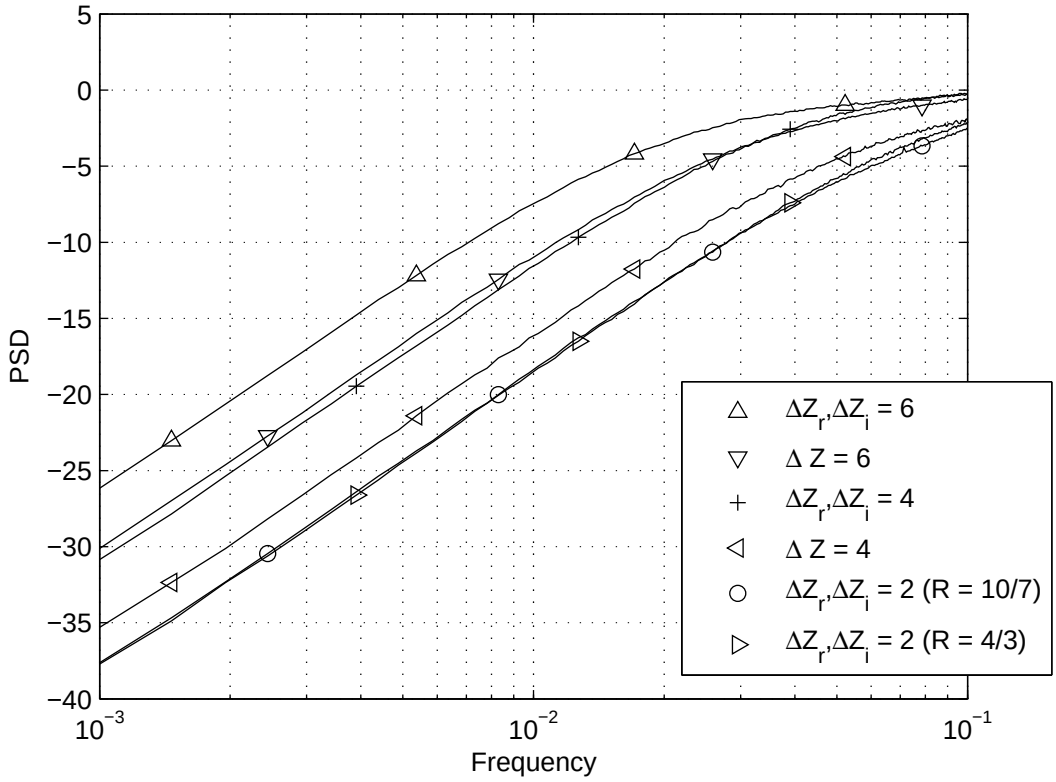

Figure 5.1: PSD of DC-free QPSK codes with $\Delta Z_r, \Delta Z_i = 2, 4$ and 6.

5.4 Construction of DC-free Codes Using 8PSK Signalling Alphabets

The construction of DC-free codes using an 8PSK signalling alphabet is considered in this section. This is the most complex case of the three signalling constellations considered in this chapter because it contains signalling points that are incommensurable. As a result, there exist an infinite number of states in a state machine that accurately tracks bounded RDS values, making the code construction process difficult.

One possible solution to limiting the state machine to a finite number of states is to round the values of some of the signalling points. In the previous chapter, the problematic value of $\sqrt{2}$ is rounded to a rational number, such as 0.7 or 0.71, and this value is used during the construction of an FSM. Although the resulting FSMs have a significant number of states, the FSMs, in theory, can be used for code construction. As the number of significant digits in the approximation of the symbol value increases, however, the number of states grows rapidly, and soon it becomes impractical to use this approach as a basis for code construction.

Instead, to simplify the code construction procedure, independently constrained dimensions are used as a starting point. The signalling constellation is considered to consist of four logical dimensions, instead of only the real and imaginary dimensions. Each point on the signalling constellation is paired with its reflection across the origin to form the four logical dimensions. An RDS bound is then enforced on each of these dimensions independently. The benefit of this approach is a significant reduction in the number of states while maintaining a capacity that is very close to that of the rounding approach described above.

To construct the required FSM, the procedure outlined in the previous chapter is followed. First, the constellation is split into its logical dimensions that are considered to be independent of one another. Second, an RDS span is chosen for each dimension of this signalling constellation and a one-dimensional FSM is constructed for each dimension corresponding to this bound. Next, the overall FSM is constructed by taking the Cartesian product of all four of the one-dimensional FSMs. The RDS for each state on this overall FSM is then calculated. Finally, based on the desired RDS span for this code, states representing RDS values that would violate this RDS span are removed from the FSM. This leaves an FSM that contains only states that satisfy the desired RDS span. This removal of states also results in dimensions that are no longer independent.

Table 5.4: Parameters of 8PSK DC-free codes with $\Delta Z_r, \Delta Z_i$ between 2 and 4.

Rectangular Bounds

$\Delta Z_r \Delta Z_i$	m	n	R	C	ν	$\lvert P \rvert$	s_z^2	f_0
2.0	11	6	1.800	2.083	0.880	29	0.738	0.124
2.4	8	4	2.000	2.136	0.936	16	0.698	0.125
3.0	11	5	2.200	2.413	0.912	141	1.048	0.0822
3.6	14	6	2.333	2.547	0.916	145	1.353	0.0749
4.0	14	6	2.333	2.647	0.882	25	1.572	0.0508

Circular Bounds

ΔZ	m	n	R	C	ν	$\lvert P \rvert$	s_z^2	f_0
3.0	8	4	2.000	2.315	0.864	121	0.842	0.124
3.5	11	5	2.200	2.439	0.902	145	1.039	0.0873

Table 5.4 summarizes codes that have been constructed using the method described above along with the approximation algorithm presented in Chapter 3. As shown in the table, DC-free codes using an 8PSK alphabet have been generated with rectangular RDS bounds that have efficiencies ranging from 88% to over 93%. Additionally, two DC-free 8PSK codes using circular RDS bounds have been constructed with efficiencies of 86% and 90%. In general, the number of encoding states increases as ΔZ increases. There is one exception however. Observe that the 8PSK DC-free code with $\Delta Z_r, \Delta Z_i = 4$ has significantly fewer principal states than the code with $\Delta Z_r, \Delta Z_i = 3.6$. This behaviour results from the fact that both codes are constructed with the same rate, while the code with $\Delta Z_r, \Delta Z_i = 4.0$ has a higher capacity. Thus, there are significantly more edges available in each state for the code with $\Delta Z_r, \Delta Z_i = 4.0$ and Franaszek's algorithm is able to use this to eliminate a significant number of states from the set of principal states. Therefore, while the number of encoding states typically increases significantly as ΔZ increases, it is sometimes possible to offset this increase in states with a reduction in efficiency.

The power spectral density of each of the codes listed in Table 4.2 is presented in Fig. 5.2. One unusual case emerges in the PSD plots. The spectral performance of the 8PSK DC-free code for $\Delta Z_r, \Delta Z_i = 2.4$ is actually better than that of the code with $\Delta Z_r, \Delta Z_i = 2.0$. The primary reason for this is that while the RDS span is larger, the code spends less of its time in states with RDS values that are further from the origin. This can be seen by comparing the sum variance of the RDS for the two codes, which measures the average squared distance the code is from the origin. For the case of the $\Delta Z_r, \Delta Z_i = 2.0$ code, s_z^2 is 0.738, while $s_z^2 = 0.698$ for the case of the $\Delta Z_r, \Delta Z_i = 2.4$ code. It is concluded that while $\Delta Z_r, \Delta Z_i$ specify

the RDS span used in the FSM design, the code construction procedure that occurs thereafter is still of great importance in order to obtain good code performance.

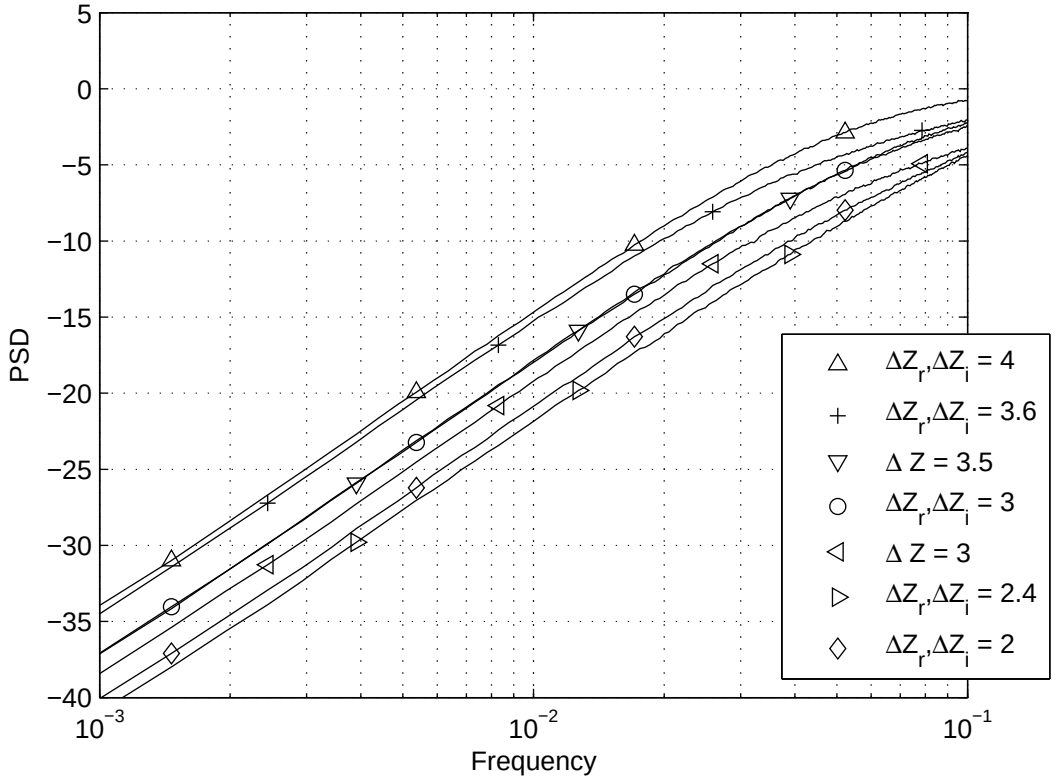

Figure 5.2: PSD of 8PSK DC-free codes with $\Delta Z_r, \Delta Z_i$ between 2 and 4.

Using the values for s_z^2 and f_0 for the codes that have been designed, the values for Justesen's relationship were computed. In the case of DC-free codes using 8PSK signalling alphabets, the value of $2\omega_0 s_z^2$ falls between 1.005 and 1.314, indicating that the relationship holds.

When comparing the QPSK DC-free codes to the 8PSK DC-free codes for the same values of ΔZ, notice that the 8PSK DC-free codes have better spectral suppression. For example, when considering $\Delta Z = 4$, the 8PSK DC-free code has approximately 3 dB more spectral suppression than the QPSK DC-free code at low frequency values. This superior spectral suppression is reflected in the variance of the RDS for these codes.

5.5 Construction of DC-free Codes Using 16 QAM Signalling Alphabets

In this section, the construction of DC-free codes using encoded symbols from the 16 QAM signalling alphabet is considered. Similar to the QPSK DC-free codes, the symbol values in the signalling constellation are commensurable and the code construction process is simpler than for 8PSK.

The code construction process begins by partitioning the constellation into several independent logical sub-constellations, where an RDS span is enforced upon each dimension. The overall FSM is constructed by taking the Cartesian product of the FSMs describing the operation of each of the sub-constellations. The overall D matrix, its maximum eigenvalue, and capacity can then be calculated, using the constraint modelling from the previous chapter. Using this approach, the overall RDS span will be larger than the RDS span on each of the sub-constellations, but this can be adjusted by enforcing an RDS span constraint on the overall FSM by removing states that violate this overall constraint. The D matrix describing the final FSM is used as an input to Franaszek's algorithm, which returns a set of parameters for m, n and P. The large alphabet size, however, places practical limitations on the set of parameters that can be used since as n increases, the size of the codebook increases significantly. Upon fixing the values for m, n, and P, the approximation algorithm in Chapter 3 is used to construct a code that can be decoded at the receiver without the need for state information.

Table 5.5 lists parameters for DC-free codes that have been constructed using a 16 QAM signalling alphabet for various RDS spans. As shown in the table, these codes have efficiencies ranging from 88% up to almost 96%. Note that ΔZ_r and ΔZ_i are larger than in other codes considered in this paper because, when the 16 QAM constellation has distance of 2 between adjacent points, the smallest possible bound is $\Delta Z_r = \Delta Z_i = 6$. Fig. 5.3 shows the spectral performance of each of these codes based on a simulation of five million codewords. For a fair comparison of spectra with the QPSK and 8PSK DC-free codes, the symbol values have all been divided by $\sqrt{10}$, so that the average energy per symbol in each constellation is equal to one. Trends similar to the QPSK and 8PSK DC-free codes can be observed. In particular, observe that the value of $2s_z^2\omega_0$ is between 0.958 and 1.080, indicating that Justesen's relationship holds.

Table 5.5: Parameters of 16 QAM DC-free codes with $\Delta Z_r, \Delta Z_i$ between 6 and 15.

Rectangular Bounds

$\Delta Z_r \Delta Z_i$	m	n	R	C	ν	$\lvert P \rvert$	s_z^2	f_o
6	12	4	3.000	3.133	0.958	21	0.565	0.141
9	13	4	3.250	3.422	0.950	16	0.872	0.0939
12	13	4	3.250	3.690	0.881	81	1.701	0.0501
15	14	4	3.500	3.758	0.931	77	1.769	0.0486

Circular Bounds

ΔZ	m	n	R	C	ν	$\lvert P \rvert$	s_z^2	f_o
9	13	4	3.250	3.3978	0.956	16	0.825	0.0923
12	13	4	3.250	3.6062	0.901	61	1.274	0.0612

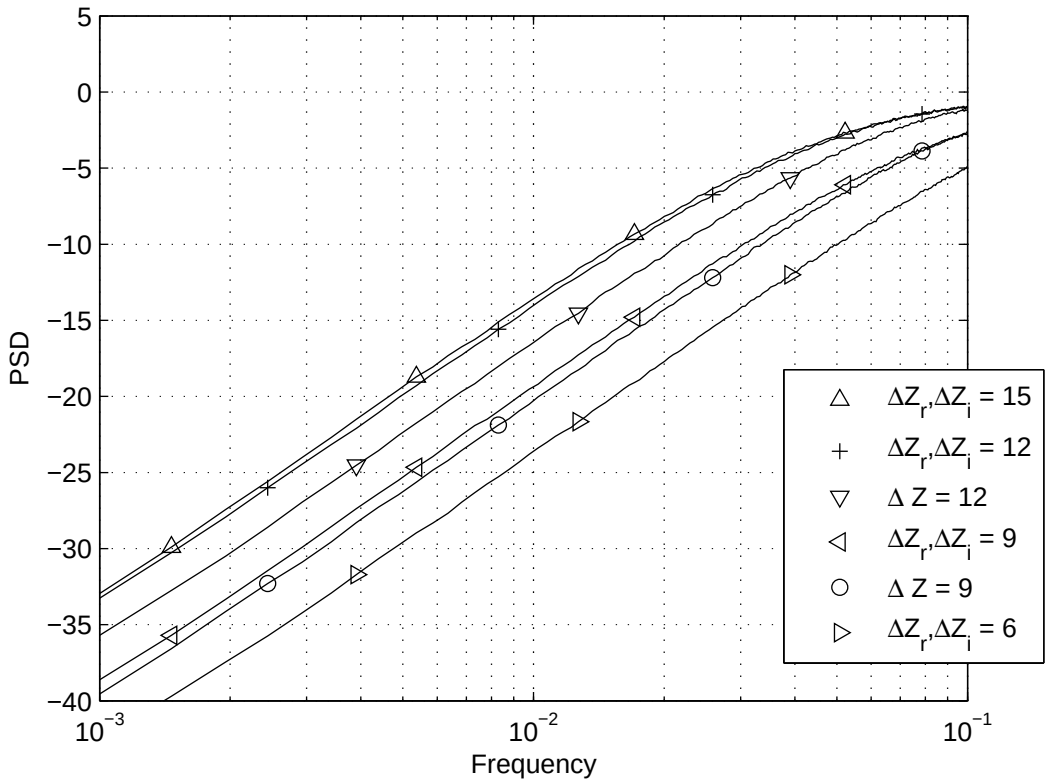

Figure 5.3: PSD of 16 QAM DC-free codes with $\Delta Z_r, \Delta Z_i$ between 6 and 15.

5.6 Summary

In this chapter, several examples of QPSK, 8PSK, and 16 QAM DC-free codes have been constructed for various RDS spans. The various steps involved in the code construction procedure were discussed, from constraint modelling to using Franaszek's algorithm and the approximation algorithm described in Chapter 3 to generate a state-independent decodable code. The codes that have been constructed with complex-valued signalling alphabets have efficiencies ranging from 88% to 96%. The spectral performance of these codes was presented and it was shown that, similar to codes using one-dimensional constellations, the variance of the RDS provides a good indication of the spectral performance. Further, values of $2s_z^2\omega_0$ were evaluated, and it was shown that Justesen's relationship, $2s_z^2\omega_0 \approx 1$, holds for DC-free codes using these complex-valued signalling alphabets.

Chapter 6

Conclusions and Future Work

In this chapter, an overview of this thesis is provided, along with suggestions for possible future work. Section 6.1 details the three major contributions of this thesis, while Section 6.2 suggests potential future work.

6.1 Thesis Contributions

The first major contribution of this thesis is the introduction of an approximation algorithm to construct constrained codes that permit state-independent decoding at the receiver. This algorithm was designed with all types of constrained codes in mind, although it is applied to two specific types in this thesis. The major goals of the design of the algorithm were i) to have a reasonable computational complexity (at most polynomial running time), ii) to be flexible so that it is applicable to a wide range of codes, and iii) to be able to handle simplistic codes, for example, codes with less than 100 total codewords and 10 or fewer principal states, and to also handle significantly more complex codes, for example, codes with up to $2^{18} = 262144$ codewords and 150 or fewer principal states. The algorithm developed in this thesis was able to meet all three of these targets. This algorithm was tested first upon a well-known class of codes, DC-free RLL constrained codes, and then applied to a new class of codes, DC-free codes using signalling alphabets with points taken from complex alphabets.

The second major contribution of this thesis is the development of techniques to model the constraints of DC-free codes using signalling alphabets with symbols taken from a complex signalling alphabet. This constraint modelling, typically in terms of a state machine, allows for the determination of the capacity of such systems. Of the representative systems that were studied, two major types emerged: signalling alphabets with commensurable symbols, leading to a finite number of

states, and signalling alphabets with incommensurable symbols, leading to an infinite number of states. In both cases, it was demonstrated how the constraint can be modelled and the capacity calculated or an estimation of capacity made. The constraint modelling techniques are used in conjunction with the approximation algorithm developed in the first portion of this thesis to design DC-free codes using QPSK, 8 PSK, and 16 QAM signalling alphabets.

The specific contributions of this thesis are described in more detail below.

1. **An approximation algorithm for constructing state-independent decodable codes**

   While many techniques exist for the construction of a constrained code, an important property, in order to prevent an unbounded string of errors, is that the code be decodable at the receiver without requiring state information. Previous work includes an algorithm that develops codes that achieve this property, but at the expense of significant (NP-complete) computational complexity. An algorithm for generating constrained codes that permit decoding without state information was developed in this thesis. This algorithm was designed to run in polynomial time and be able to construct constrained codes that are complex, both in terms of the number of required codewords and the number of principal states. Further, this algorithm is flexible and includes a number of parameters that can be adjusted to achieve code design, including scoring thresholds, scoring exponents and the selection of base rows. More advanced techniques, such as lookahead and weakly constrained coding were also discussed and implemented. While lookahead and weakly constrained coding are reasonably generic principles, it was shown that, in the case of the algorithm developed in this thesis, both of these techniques mesh well with the algorithm. The successful design of this algorithm was demonstrated through its application to DC-free RLL codes using binary signalling. This algorithm was applied to a number of DC-free RLL codes, two of which were presented in detail. In the one case where it was not possible to find a code that permits state-independent decoding for the most efficient set of parameters returned by Franaszek's algorithm, a weakly constrained code was constructed and presented in detail.

2. **Constraint modelling and evaluation of capacity for constrained codes using multiple signalling dimensions**

Constrained codes constructed for a one-dimensional signalling medium have usually been constructed using binary digits or with multilevel signalling. In this thesis, constrained codes are extended to include multi-dimensional signalling alphabets written to a one-dimensional signalling medium. In particular, representative systems using QPSK, 8 PSK, and 16 QAM were used as examples to illustrate the techniques for evaluating or estimating the capacity of these constrained systems. Techniques were developed to model constraints using these signalling alphabets to construct state machine representations. For the case of codes using both QPSK and 16 QAM signalling alphabets, since the signalling points are commensurable, the state machines describing these constraints were limited to a finite number of states, and so exact capacity analysis was possible. Constraint modelling and capacity evaluation in the case of 8 PSK codes is more difficult since the points in the signalling constellation are incommensurable, leading to an infinite number of states. The capacity of these systems was estimated using several different methods, including rounding of signal point values and upper bounding.

3. **Construction of highly efficient DC-free constrained codes with multiple signalling dimensions**

   While there exist many examples of DC-free codes in the literature, [cf. 2], these DC-free codes use a one-dimensional signalling alphabet that is typically binary. Further, there exist many codes designed for two (or more) dimensional media [34]- [39]. However, to date, there are few to no examples of codes using multi-dimensional signalling. Using the constraint modelling techniques developed in this thesis, constrained codes using two-dimensional signalling alphabets, including QPSK, 8 PSK, and 16 QAM were constructed. The capacity analysis demonstrates that the codes that have been constructed are highly efficient, in most cases greater than 90%. PSD plots of the codes that have been constructed were presented, and it was verified that Justesen's relationship, $2s_z^2\omega_0 \approx 1$, which was developed for one-dimensional codes, holds also for these multi-dimensional codes.

The content of the individual chapters of this thesis can be summarized as follows:

Chapter 3 proposed an approximation algorithm for the construction of constrained codes that permit state-independent decoding. Building on Franaszek's

work [8], which organizes the constraint in a tabular format, the important considerations for the design of the algorithm were considered, such as advantageous table construction and the fitting procedure used to combine rows in the table. Advanced techniques, such as lookahead and weakly constrained coding, were discussed and integrated into the algorithm. Highly efficient DC-free RLL codes were presented for a wide range of $d, k,$ and N values, along with an example of a weakly constrained DC-free RLL code.

Chapter 4 proposed constraint modelling techniques and capacity analysis of codes with multiple constrained signalling dimensions. Examples of these types of constraints were presented, using QPSK, 8 PSK, and 16 QAM signalling constellations. For the case when state machine modelling of the constraint resulted in an infinite number of states, the capacity of the constrained system was estimated through rounding and upper bounding.

Chapter 5 used the constraint modelling techniques developed in the previous chapter to construct DC-free constrained codes with QPSK, 8 PSK, and 16 QAM signalling constellations. The spectral performance of these codes was evaluated and simulated, to both demonstrate that the codes are DC-free and verify that Justesen's relationship holds.

6.2 Suggested Future Work

In this section, some related topics for future research are suggested.

1. **Improving the values of scoring exponents and scoring thresholds for code construction**

   The approximation algorithm for state-independent decoding has a number of parameters that can be adjusted to improve its ability to construct a code. In the majority of cases that were tried, values of $r_f, r_o = 2$ with an initial scoring threshold of zero produced the best results. While efficiency is calculated directly using the values for m and n that are chosen, sometimes the algorithm cannot complete the codebook. In some cases, this is due to the lack of surplus edges constraining the algorithm's freedom. A more common trend is that as the number of principal states increases to over 100, the algorithm has some trouble finding efficient codes. Below 100 principal states, efficiencies close to 95% are generally possible, but efficiencies drop to around 86% to 88% as the number of principal states increases significantly beyond 100.

In other words, as the number of principal states gets large, the algorithm is unable to construct a code for the set of parameters (m, n, P) returned by the Franaszek algorithm with the highest code rate. It continues trying the next most efficient set of parameters, eventually constructing a code, but the efficiency lowers each time a new set is tried. In those cases, there are a significant number of columns in the code tables, and so fine tuning the scoring parameters might give better results. In the simpler cases, for example 30 to 50 principal states, it is somewhat easier to examine a number of the decisions the algorithm makes and adjust the parameters slightly. However, with a large number of principal states, this analysis becomes difficult. Other families of constrained codes might benefit from a deeper look at these parameters.

2. **Weakly constrained codes**

   In Chapter 3, the weakly constrained coding approach was applied to a particular DC-free RLL code where it is impossible to construct a code that permits state-independent decoding for the set of parameters with the highest code rate returned by the Franaszek algorithm. The code construction algorithm is amenable to this approach because it does not require any changes to allow a weakly constrained coding algorithm to run thereafter. That is, the algorithm fills as many spaces as possible so that the constraint is violated as infrequently as possible, and the few remaining spaces, typically $0.1 - 1\%$ of the table, are filled by the weakly constrained coding algorithm. Constructing a weakly constrained code does not require that the code rate be lowered and, as indicated by the example in this thesis, with careful design, the spectral performance need not suffer significantly. A flooring effect does appear in the case of weakly constrained DC-free codes, but only at very low frequencies. Prior to that point, the PSD of a weakly constrained code is very similar to that of a non-weakly constrained code. In particular, Franaszek's algorithm returns the set of parameters with the highest code rate for a given constraint. In the case of a weakly constrained code, one can design a code that has a higher code rate since the constraint is being violated. For the same reason, a clever code designer can construct a code with a code rate that is above capacity since the capacity calculation is specific to the constraint not being violated. In that sense, Franaszek's algorithm is no longer required; however, it does provide an excellent guide to which states have the most edges and are thus best suited to being principal states. It is anticipated that

an improved version of Franaszek's algorithm, designed specifically for the subsequent design of weakly constrained codes, would improve the final code construction results for weakly constrained codes.

3. **Improved code construction techniques**

   There has not been a significant amount of research in the area of constrained codes using complex-valued signalling. This thesis serves as a preliminary work in this area, outlining some techniques for the construction of DC-free codes using complex-valued signalling alphabets. In particular, these techniques use a lookup table approach for encoding and decoding, over a large number of states. Further, the capacity evaluation and constraint modelling is formulated based on the idea of independent and dependently constrained signalling dimensions. It is of interest to consider other construction techniques, which, for example, would not require a table-based lookup. Alternatively, more sophisticated graph-based operations could be used, particularly in the case where encoding initially requires an infinite number of states. Improving these code construction techniques allows the code designer to have more options with which to build their communication system.

4. **Spectral null coding**

   In this thesis, the codes using complex-valued signalling alphabets are all designed to be DC-free. Recall from (2.16) that the spectrum of a sequence of symbols can be shaped by introducing correlation into the symbol sequence. This procedure is not mutually exclusive with pulse shaping, facilitating an implementation with many existing communication systems. In particular, these codes can be used as a starting point to construct codes with a movable notch in their spectrum. By multiplying the coded symbol sequence by a set of complex values, the spectral null can be shifted to anywhere in the spectrum. Since the shifting values will be complex, this means that the symbols entering the channel will be complex-valued, even if the initial symbol sequence was not. These types of codes could have applications in communication systems where the transmitter may wish to avoid any spectral content at a particular frequency or a range of frequencies. Further, it may be possible to construct codes, using techniques similar to those described in this thesis, with multiple spectral nulls.

Bibliography

[1] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, pp. 379–423, July 1948.

[2] K. A. S. Immink, *Codes for Mass Data Storage Systems 2nd Ed.* Shannon Foundation, 2004.

[3] K. A. S. Immink and H. Ogawa, "Method for encoding binary data," U.S. Patent 4,501,000, Feb., 1985.

[4] D. Slepian, "On maxentropic discrete stationary processes," *Bell Syst. Tech. J.*, vol. 51, no. 3, pp. 629–653, Mar. 1972.

[5] R. Togneri and C. J. S. deSilva, *Fundamentals of Information Theory and Coding Design*. Chapman and Hall/CRC, 2002.

[6] J. G. Kemeny and J. L. Snell, *Finite Markov Chains*. Van Nostrand, 1960.

[7] R. S. Varga, *Matrix Iterative Analysis*. Prentice-Hall, Inc., 1962.

[8] P. A. Franaszek, "Sequence-state encoding for digital transmission," *Bell Syst. Tech. J.*, vol. 47, pp. 143–157, Jan. 1968.

[9] P. Chaichanavong and B. H. Marcus, "Stabilization of block-type-decodability properties for constrained systems," *SIAM J. Discrete Math*, vol. 19, no. 2, pp. 321–344, 2005.

[10] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill Book Company, 1965.

[11] G. L. Cariolaro and G. P. Tronca, "Spectra of block coded digital signals," *IEEE Trans. Commun.*, vol. 28, no. 10, pp. 1555–1564, Oct. 1974.

[12] K. W. Cattermole and J. J. O'Reilly, *Problems of Randomness in Communication Engineering*. Pentech Press, 1984, vol. 2.

[13] P. Galko and S. Pasupathy, "The mean power spectral density of Markov chain driven signals," *IEEE Trans. Inf. Theory*, vol. 27, no. 6, pp. 746–754, Nov. 1981.

[14] P. A. Franaszek, "Sequence-state encoding for digital transmission," *Bell Syst. Tech. J.*, vol. 47, pp. 143–157, Jan. 1968.

[15] M. Berkoff, "Waveform compression in NRZI magnetic recording," *Proc. IEEE*, vol. 52, pp. 1271–1272, Oct. 1964.

[16] A. Gabor, "Adaptive coding for self-clocking recording," *IEEE Trans. Electron. Comput.*, vol. EC-16, pp. 866–868, Dec. 1967.

[17] W. H. Kautz, "Fibonacci codes for synchronization control," *IEEE Trans. Inf. Theory*, vol. 11, no. 2, pp. 284–292, Apr. 1965.

[18] J. P. J. Heemskerk and K. A. S. Immink, "Compact disc: System aspects and modulation," *Philips Techn. Review*, vol. 40, no. 6, pp. 157–164, 1982.

[19] K. A. S. Immink, "The digital versatile disc (DVD): System requirements and channel coding," *SMPTE Journal*, vol. 105, no. 8, pp. 483–489, Aug. 1996.

[20] T. D. Howell, "Analysis of correctable errors in the IBM 3380 disk file," *IBM J. Res. Develop.*, vol. 28, no. 2, pp. 206–211, Mar. 1984.

[21] T. M. Chien, "Upper bound on the efficiency of DC-constrained codes," *Bell Syst. Tech. J.*, vol. 49, pp. 2267–2287, Nov. 1970.

[22] J. Justesen, "Information rates and power spectra of digital codes," *IEEE Trans. Inf. Theory*, vol. 28, no. 3, pp. 457–472, May 1982.

[23] J. Justesen and T. Hoholdt, "Maxentropic Markov chains," *IEEE Trans. Inf. Theory*, vol. 30, no. 4, pp. 665–667, July 1984.

[24] I. J. Fair, "Construction and characteristics of codewords and pulse shapes that satisfy spectral constraints," *IEEE Trans. Commun.*, vol. 56, no. 10, pp. 1585–1590, Oct. 2008.

[25] G. Pierobon, "Codes for zero spectral density at zero frequency," *IEEE Trans. Inf. Theory*, vol. 31, no. 7, pp. 853–861, March 1984.

[26] 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 Trans. Commun.*, vol. 39, no. 2, pp. 289–297, Feb. 1991.

[27] E. Gorog, "Redundant alphabets with desirable frequency spectrum properties," *IBM J. Res. Develop.*, vol. 12, no. 3, pp. 234–241, May 1968.

[28] A. R. Calderbank, M. A. Herro, and V. Telang, "Redundant alphabets with desirable frequency spectrum properties," *IEEE Trans. Inf. Theory*, vol. 35, no. 3, pp. 579–583, May 1989.

[29] L. Botha, H. C. Ferreira, and I. Broere, "New multilevel line codes," in *Proc. IEEE GLOBECOM '90*, San Diego, CA, Dec. 1990, pp. 714–719.

[30] ——, "Multilevel sequences and line codes," in *IEE Proc. I Communications, Speech, and Vision*, Aug. 1993, pp. 255–261.

[31] D. E. Knuth, "Efficient balanced codes," *IEEE Trans. Inf. Theory*, vol. 32, no. 1, pp. 51–53, Jan. 1986.

[32] A. Baliga and S. Boztas, "Balancing sets of non-binary vectors," in *Proc. IEEE Int'l Symp. Inform. Theory (ISIT'2002)*, Lausanne, Switzerland, June 2002, p. 300.

[33] L. G. Tallini and U. Vaccaro, "On efficient $m$ary balanced codes," in *Proc. IEEE Int'l Symp. Inform. Theory (ISIT'1997)*, Ulm, Germany, June 1997, p. 217.

[34] E. Soljanin and C. N. Georghiades, "Coding for two-head recording systems," *IEEE Trans. Inf. Theory*, vol. 41, no. 3, pp. 794–755, May 1995.

[35] J. J. Ashley and B. H. Marcus, "Two dimensional low-pass filtering codes," *IEEE Trans. Commun.*, vol. 46, no. 6, pp. 724–727, June 1998.

[36] R. M. Roth, P. H. Siegel, and J. K. Wolf, "Efficient coding schemes for the hard-square model," *IEEE Trans. Inf. Theory*, vol. 47, no. 3, pp. 1166–1176, March 2001.

[37] A. Sharov and R. M. Roth, "Two-dimensional constrained coding based on tiling," in *Proc. IEEE Int'l Symp. Inform. Theory (ISIT'2008)*, Toronto, Canada, July 2008, pp. 1468–1472.

[38] W. Weeks and R. E. Blahut, "The capacity and coding gain of certain checkerboard codes," *IEEE Trans. Inf. Theory*, vol. 44, no. 3, pp. 1193–1203, May 1998.

[39] S. Forchhammer and J. Justesen, "Entropy bounds for constrained 2-D random fields," *IEEE Trans. Inf. Theory*, vol. 45, pp. 118–127, Jan. 1999.

[40] S. Cook, "The complexity of theorem proving procedures," in *Proc. 3rd Annual ACM symposium on Theory of Computing*, Shaker Heights, OH, May 1971, pp. 151–158.

[41] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. McGraw-Hill, 2002.

[42] D. E. Knuth, *The Art of Computer Programming. Vol. 1: Fundamental Algorithms*. Addison-Wesley, 1973.

[43] R. P. Brent, "Efficient implementation of the first-fit strategy for dynamic storage allocation," *ACM Transactions on Programming Languages and Systems*, vol. 11, no. 3, pp. 388–403, July 1989.

[44] K. A. S. Immink, "Weakly constrained codes," *Electronic Letters*, vol. 33, no. 23, pp. 1943–1944, Nov. 1997.

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

[46] J. F. Heanue, M. C. Bashaw, and L. Hesselink, "Channel codes for digital holographic data storage," *J. Opt. Soc. Am. A*, vol. 12, pp. 2432–2439, Nov. 1995.

[47] R. M. Roth, P. H. Siegel, and J. K. Wolf, "Efficient coding schemes for the hard-square model," *IEEE Trans. Inf. Theory*, vol. 47, no. 3, pp. 1166–1176, Mar. 2001.

[48] S. Halevy, J. Chen, P. H. Siegel, and J. K. Wolf, "Improved bit-stuffing bounds on two-dimensional constraints," *IEEE Trans. Inf. Theory*, vol. 50, no. 5, pp. 824–838, May 2004.

[49] R. Bellman, *Introduction to Matrix Analysis*. SIAM, 1997.