Capacity-Approaching Variable-Length Constrained Sequence Codes

by

Congzhe Cao

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

Doctor of Philosophy

in

Communications

Department of Electrical and Computer Engineering

University of Alberta

Abstract

In this thesis we consider construction techniques and applications of variable-length constrained sequence codes. First we present background information related to constrained sequence coding theory and review a recently reported technique for constructing variable-length constrained sequence codes. We then outline original work. We demonstrate two new general algorithms that we have developed to construct variable-length constrained sequence codes for various types of constraints. The first is based on an encoder with a single encoding state, while the second is based on an encoder with multiple encoding states which retains the property of state-independent decoding. We present examples of these construction algorithms. Then we apply our coding technique to develop constrained sequence codes for flash memory with multi-page programming in order to reduce the impact of inter-cell interference and cell leakage. Lastly we study the synchronization properties of the codes we developed, and show that it is possible to design variable-length constrained sequence codes with good synchronization properties such that once the receiver loses synchronization, it regains synchronization within a limited number of codewords.

Preface

This thesis contains contents that appear in the following publications. For all the publications listed in the preface, the technical analysis, computer simulations and writing are conducted by myself under the supervision of Dr Ivan Fair.

In Chapter 2.1 we propose a construction approach to construct minimal sets of variable-length constrained sequence (CS) codes for a variety of constraints based on the finite state machine (FSM) description of constraints. We also introduce FSM partitions and propose a recursive construction algorithm to establish the minimal set of the specified state. The content of Chapter 2.1 has been published in the following publications:

- C. Cao and I. Fair, "Construction of minimal sets for capacity-approaching variable-length constrained sequence codes," *2016 Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, USA, 2016, pp. 255–259.

- C. Cao and I. Fair, "Minimal sets for capacity-approaching variable-length constrained sequence codes," *IEEE Transactions on Communications*, September 2018, vol. 67, no. 2, pp. 890-902, 2019.

In Chapter 2.2 we develop a variable-length CS coding technique with multi-state encoders that permit state-independent decoding. The construction technique is based on the FSM description of the constraint as well as n-step FSMs. Examples are given for a variety of constraints, including the runlength-limited (RLL) constraint, the DC-free constraint,

and the DC-free RLL constraint. The content of Chapter 2.2 has been published in the following publication:

- C. Cao and I. Fair, "Multi-state encoding of capacity-approaching variable-length constrained sequence codes with state-independent decoding," *IEEE Access*, vol. 7, pp. 54746-54759, 2019.

In Chapters 3.1 and 3.2 we describe the application of the proposed coding technique in flash memories such that inter-cell interference (ICI) is mitigated. We describe the ICI mitigation in single-level cell and multi-level cell flash memories. The content of Chapters 3.1 and 3.2 has been published in the following publications:

- C. Cao and I. Fair, "Variable-length constrained sequence codes for mitigating inter-cell interference in all-bit-line flash memory with multi-page programming," *28th Biennial Symposium on Communications*, Kelowna, BC, Canada, 2016.

- C. Cao and I. Fair, "Mitigation of inter-cell interference in flash memory with capacity-approaching variable-length constrained sequence codes," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 9, pp. 2366-2377, 2016.

In Chapter 3.3 we describe the application of the proposed coding technique such that variable-length capacity-approaching Pearson codes are constructed that can be used to reduce the impact of cell leakage in flash memories. The content of Chapter 3.3 has been published in the following publication:

- C. Cao and I. Fair, "Capacity-approaching variable-length Pearson codes," *IEEE Communications Letters*, vol. 22, no. 7, pp. 1310-1313, 2018.

In Chapter 4 we describe how variable-length CS codes with high synchronizing probability can be constructed. We develop algorithms for construction of variable-length CS codes with high synchronization

probability, and present simulation results that show that these codes exhibit good synchronization properties such that once the receiver loses synchronization, it regains synchronization within a limited number of codewords. The content of Chapter 4 is under preparation for submission:

- C. Cao and I. Fair, "Synchronization of variable-length constrained sequence codes," *submitted to IEEE Access*.

To my whole family

For your love and constant support.

They who know the truth are not equal to those who love it, and they who love it are not equal to those who delight in it.
(知之者不如好之者,好之者不如乐之者。)

– Confucius, ancient Chinese philosopher.

Acknowledgements

I would first like to give my thanks to my supervisor, Dr. Ivan Fair, for his strong and constant support and supervision during my entire program. I very much appreciate it.

I would like to express my appreciation for the scholarships I received from Alberta Innovation Technology Futures (AITF) and National Sciences and Engineering Research Council (NSERC) of Canada. This work would not have been possible without generous support.

I extend my thanks to all the faculty members who have taught and assisted me, especially Dr Majid Khabbazian, Dr Witold Krzymien, Dr Yindi Jing and Dr Ioanis Nikolaidis for their comments on my research as the members of my candidacy and final examination committees. I like to thank Dr Jos Weber from Delft University of Technology, Netherlands, for reviewing my thesis as the external examiner.

I would also like to extend my thanks to Dr Zesong Fei from Beijing Institute of Technology who supervised my Masters thesis, Dr Ming Xiao from KTH Royal Institute of Technology, Dr Toshiaki Koike-Akino and Dr Ye Wang from Mitsubishi Electric Research Laboratories, for their consistent help and collaboration.

Congzhe Cao,
in Edmonton, Alberta

Table of Contents

List of Tables

List of Figures

List of Symbols

c_{max}	the maximum length of codewords in a codebook during partial extensions
$\mathcal{C}$	a codeword
C	capacity of a constrained sequence
$\widetilde{C}_M$	maximum possible code rate of a constrained sequence constructed from the set M
$\mathbf{C}$	codebook
$\mathbf{D}$	adjacency matrix of a finite state machine (FSM)
$\mathbf{D}^n$	the adjacency matrix of an n-step FSM
$H\{X\}$	entropy of a Markov FSM
$H(\sigma_j)$	the set of next states corresponding to words in $W(\sigma_j)$
l^b	number of bits stored in each cell in flash memory
l_I	length at which, for two different minimal sets, the maximum possible code rate starts to diverge
$\|l_k\|$	the number of words that are of length l_k in a set
l_{max}	the maximum length of words in a minimal set
l_{th}	the maximum length of words in a full extension of a minimal set
L_M	the set of lengths of words in M
$\|L_M\|$	the number of words in a minimal set M
L_S	lengths of source words in a codebook
M_c	a complete extension of a minimal set M
M_f	a full extension of a minimal set M
$M_f^{(l_{th})}$	the full extension of a minimal set M containing all possible words with length no greater than l_{th}
M_p	a partial extension of a minimal set M

n_{max}	the maximum number of codewords in a codebook during partial extensions
N	number of RDS values in a DC-free sequence
N_b	the number of coded bits required for the decoder to regain synchronization once synchronization is lost
$\widetilde{N}_b$	upper bound of N_b
N_c	the number of codewords required for the decoder to regain synchronization once synchronization is lost
$\widetilde{N}_c$	upper bound of N_c
N^c	the number of codewords in a codebook
N_m	number of words in a minimal set
P	synchronization probability
$\mathbf{Q}$	single step transition matrix of an FSM
$\mathbf{Q}^n$	the n-step transition matrix of an FSM
$\bar{R}$	average code rate
$\mathbf{S}$	the set of suffixes of synchronizing words
$\mathcal{U}(m)$	the number of constraint-satisfying sequences of length m
$W(\sigma_j)$	the set of words generated by the j-th principal state
$\lvert\boldsymbol{X}\rvert$	the size of vector $\boldsymbol{X}$
$\alpha(\sigma_j)$	the set of codewords generated through extension of $W(\sigma_j)$
$\beta(\sigma_j)$	the set of next states corresponding to $\alpha(\sigma_j)$
γ	redundancy of a code
ξ	the number of entries in each state in the multi-state codebook
η	efficiency of a code
$\widetilde{\eta}$	maximum possible efficiency of a minimal set
$\boldsymbol{\pi}$	steady-state distribution of an FSM
$\varnothing$	a void codeword
Ψ	the set of principal states in the multi-state code construction technique

Acronyms

AWGN additive white gaussian noise

BSC binary symmetric channel

CS constrained sequence

CD compact disc

DAT digital audio tape

DCC digital compact cassette

DVC digital video cassette

DVD digital video disc

DVR digital video recorder

EFM eight-to-fourteen modulation

FSM finite state machine

ICI inter-cell interference

MFM modified frequency modulation

MLC multi-level cell

NGH normalized geometric Huffman

NRZI non-return-to-zero inverted

OOK on-off keying

RDS running digital sum

RLL runlength-limited

SLC single-level cell

TLC triple-level cell

VLC visible light communication

Chapter 1

Background and Motivation

1.1 Introduction to constrained sequence codes

In communication and data storage systems, constrained sequence codes, which are also called recording codes and modulation codes (in data storage systems), or line codes (in optical and wireline communications systems), have been widely used [1]. Constrained sequence codes have also been proposed for wireless energy harvesting [2] and visible light communications [3]. As for the next-generation data storage systems, constrained sequence codes are also proving to be promising for several emerging data storage devices such as flash memories [4], phase change memories [5] and DNA-based storage [6].

It is known that the storage and retrieval of information can be modeled as a communication process that provides information transfer over time. With channel coding modules integrated into the data storage system, the reliability of communication can be improved. The reliability of a communication or data storage system is often measured by the probability that the transmitted information is correctly recovered at the receiver, or inversely, by the probability that the original transmitted information is not accurately recovered. The fundamental problem of the limits of reliable transmission was considered in the pioneering work of Shannon in his 1948 paper [7] where he proved that it is possible to transmit information over a

noisy channel with an arbitrarily small error probability, provided that the transmission rate is smaller than the channel capacity. Following Shannon's early work, theorists and engineers invested tremendous effort to design practical codes that would enable the performance that Shannon promised to be approached.

In recording systems and some transmission systems, channel coding is often performed in two steps: error control coding and constrained sequence coding. Powerful error control coding techniques such as turbo codes [8], low density parity check (LDPC) codes [9], rateless codes [10] and polar codes [11] have resulted in performance approaching the Shannon limit on additive white Gaussian noise (AWGN) channels. Constrained sequence codes, as another family of channel coding techniques, attempt to ensure that the characteristics of transmitted information sequences meet the physical constraints of the non-white channel. The typical order of coding procedures for error control codes and constrained sequence codes is shown in Fig. 1.1. The error control decoder is responsible for dealing with the error propagation that results from the constrained sequence decoder.

Constrained sequence codes have found many applications in storage devices such as the compact disc (CD), Mini Disc, digital video disc (DVD), digital video recorder (DVR), digital audio tape (DAT), magnetic hard disk, digital compact cassette (DCC) and digital video cassette (DVC). The majority of constrained sequence codes used in those devices are runlength-limited (RLL) codes, DC-free codes, or the integration of these techniques such that sequences have both RLL and DC-free properties.

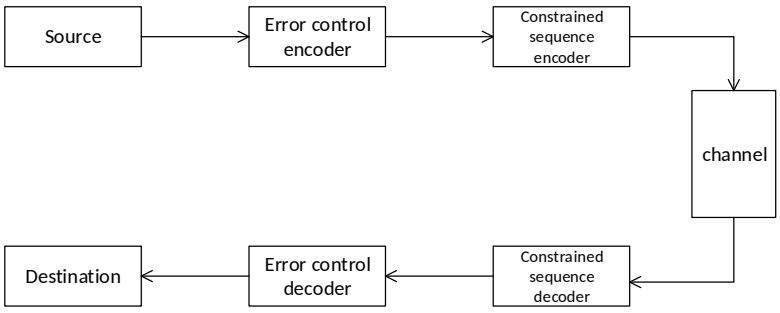

Figure 1.1: The successive coding steps of error control codes and constrained

sequence codes

RLL coded sequences are sequences in which the number of encoded bits between consecutive transitions is bounded. In many recording systems, long sequences of like-valued bits are harmful to system performance because the timing recovery and the adaptive equalization algorithms often rely on the presence of an appropriate number of transitions. Very long sequences of like-valued bits may cause the timing recovery circuit to lose synchronization. On the other hand, it is well known that the time-domain output of a channel is the convolution of the input sequence and the channel response. If the time duration of like-valued bits is too short, severe inter-symbol-interference (ISI) will arise in the channel output.

RLL codes are often referred to as (d, k) codes where d and k denote the minimum and maximum number of zeros between consecutive ones. The (d, k) constraints, followed by a non-return-to-zero inverted (NRZI) step where bit one represents a change of logic value and bit zero represents no change, result in an RLL coded sequence where the number of successive like-valued bits is at least $d + 1$ and at most $k + 1$. Such sequences avoid both long and short sequences of like-valued bits and thus effectively deal with the issue of timing recovery, adaptive equalization and ISI.

DC-free encoded sequences are sequences that do not accumulate charge. In other words, the running digital sum (RDS) value of the encoded sequence is bounded [12]. RDS is the ongoing summation of encoded bit weights in the sequence, where a logic one is represented by weight +1 while logic zero is represented by weight –1. It has been shown that if the RDS value is bounded (i.e., only N different RDS values are allowed), the coded sequence will have suppressed low frequency components [12]. In recording systems, it is desired to have low power near zero frequency in the coded sequence because recorders often are unable to respond to low frequency signals. DC-free codes have found many applications in those products. DC-free codes are also used in AC-coupled transmission systems in order to reduce power loss and baseline wander in those systems.

The last section of this chapter provides context, contributions and the significance of the work in this thesis. In Sections 1.2 to 1.4, we review fixed-length and variable-length constrained sequence codes some of which have found wide application in the data storage and communication industry. These are codes that impose an RLL constraint, or a DC-free constraint, or both of these constraints simultaneously.

1.2 Review of several constrained sequence codes

In this section we review several fixed-length and variable-length constrained sequence codes. For fixed-length codes we introduce the widely-used eight-to-fourteen modulation (EFM) code, Modified Frequency Modulation (MFM) code and Three Position Modulation code, and we discuss the encoding and decoding rules. We then introduce three types of variable-length constrained sequence codes, including synchronous variable-length constrained sequence codes.

1.2.1 Fixed-length codes

EFM code

The EFM code was designed to be used with CDs [14]. EFM coded sequences satisfy the RLL property with $d = 2$ and $k = 10$, and have a code rate of 8/17 and efficiency 86.86%, meaning that the code rate achieves 86.86% of the maximum rate that is possible with this contraint. Each source block consists of 8 bits, which is encoded into 14 coded bits using a code table. Each sequence of 14 coded bits is followed by 3 additional bits called merging bits. The 14 coded bits and the 3 merging bits together form a coded block of 17 coded bits.

The three merging bits are used to control the low frequency components of the coded sequences. They are chosen in a way that attempts to bound the RDS of a coded block. An example of the EFM coding process is shown in Fig.

1.2. As shown in this figure, the first block of 8 source bits 0110001 is encoded into 14 coded bits 10000100100010 by referring to the EFM codebook. When considering the merging bits, it is evident that in order not to violate the $d = 2$ constraint, the first merging bit must be a 0. Therefore we have three choices for the 3 merging bits: 000, 001 and 010, bearing in mind that sequence 011 is not allowed since it violates the $d = 2$ constraint. The one we choose should result in the lowest magnitude of RDS value at the end of a new coded block. In this example, assume this is the beginning of the coded sequence and the initial RDS value is 0. 001, 010, 000 result in the RDS value of 5, 7, -1 at the end of the next coded block, respectively. Therefore, 000 should be chosen as the merging bits.

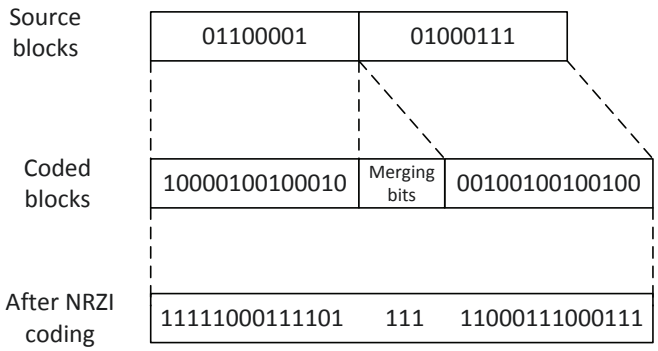

Figure 1.2: The successive coding step of error control codes and constrained sequence codes

It has also been reported that improved selection criteria by looking ahead for one more block and choosing the merging bits that result in the smallest RDS at the end of the next two blocks can improve the performance [15]. The reason is that minimizing the RDS value for the short term (at the end of the next block) does not always minimize the RDS value for the longer term (for example, at the end of the next two blocks). However, because of the complexity, this approach is not used in real equipment, although it is in full agreement with the standard for the Compact Disc system.

MFM code

The MFM code is simple and easy to implement, and as a result became the standard in flexible and "Winchester"-technology disc drives [12]. The MFM code has the parameters $d = 1$, $k = 3$, the code rate $R = 0.5$ and efficiency 90.66%. This code maps one source bit into two coded bits as shown in Table 1.1, and as explained below.

Table 1.1: The code table of MFM code

Source bit	Coded bits
0	$x0$
1	01

The mapping from source bits to coded bits is very simple. If the source bit is 1, then the coded bits are 01. If the source bit is 0, the coded bits are $x0$ with x depending on the previous coded bit. If the previous coded bit is 0, x is 1, otherwise x is 0. It can be seen that x also works as the "merging bit", which is similar to the EFM code. The finite state machine (FSM) that describes the encoder is shown in Fig. 1.3, where state A denotes the situation in which the previous coded bit is 0, while state B denotes the situation in which the previous coded bit is 1.

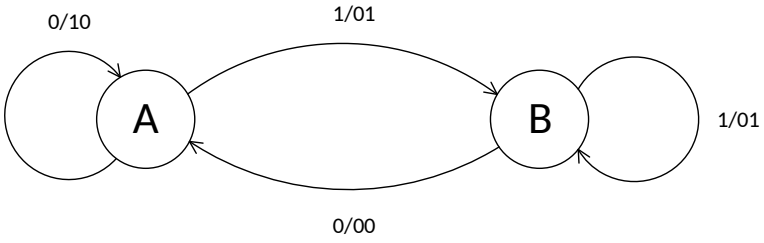

Figure 1.3: The FSM description of the encoder of MFM code

Decoding the MFM code is straightforward. One only needs to discard the first received bit, and the second received bit is decoded as the source bit.

Three Position Modulation code

Three Position Modulation code [16], which is also called the 3PM code, has also been used in disk systems. The parameters of 3PM code are $d = 2$,

$k = 11$, with a code rate of $R = 0.5$. The basic encoding table of the 3PM code is shown in Table 1.2.

Table 1.2: The encoding table of 3PM code

Source bits	Coded bits
000	000010
001	000100
010	010000
011	010010
100	001000
101	100000
110	100010
111	100100

In the encoding process, 3 source bits are encoded into 6 coded bits. However, it can be observed that in some situations the $d = 2$ constraint may be violated. Therefore, a merging process is introduced. If the fifth coded bit in the current codeword and the first coded bit in the next codeword are both 1, then the $d = 2$ constraint will be violated. In such situations, the sixth coded bit in the current codeword is set to 1, the fifth coded bit in the current codeword and the first bit in the next codeword are set to 0 in order to maintain the $d = 2$ property.

The decoding process is also simple to implement. Whenever a codeword enters the decoder and the sixth coded bit of the codeword is observed to be 1, the decoder can easily undo the process described above, otherwise the received codeword stays the same. Then, by using Table 1.2, one can easily find the corresponding source bits.

In the above we have introduced several types of fixed-length constrained sequence codes. In the next subsection, we discuss several types of variable-length codes and compare them with fixed-length codes.

1.2.2 Variable-length codes

Variable-length constrained sequence codes refer to those having variable-length codewords, while the source words can either be fixed-length or variable-length. One issue related to variable-length constrained sequence codes is the need to unambiguously identify the end of each variable-length codeword. Since the lengths of codewords are different, it is necessary that one codeword is not the prefix of another, i.e., there is no whole codeword in the codebook that is an initial segment of any other codeword in this codebook, otherwise it may not be possible for the decoder to correctly identify the end of a codeword. Codes in which no codewords are prefixes of other codewords are called prefix codes, or codes with the prefix property. In most cases variable-length constrained sequence codes should be prefix codes in order to facilitate the decoding process. Several codes that satisfy this property are considered below.

A $(1, \infty)$ code

Design of variable-length constrained sequence codes can be straightforward for some types of constraints. For example, here we present a variable-length constrained sequence code to satisfy the $(1, \infty)$ constraint [12]. Since the only constraint imposed is that consecutive ones are not allowed, we can design a code table with only two codewords, i.e. 0 and 10. The assignment of source words and codewords can be arbitrary. In Table 1.3 we show one possible mapping between source words and codewords.

Table 1.3: The code table of a $d = 1, k = \infty$ code

Source bit	Coded bits
0	0
1	10

Assuming equiprobable and independent source bits, the average code rate of this code is 2/3. Note that the capacity of the $(1, \infty)$ constraint is $\log_2 \frac{1+\sqrt{5}}{2} = 0.6942$, thus this codebook has an efficiency of 96.03%. This code

has the advantage that it has a high code rate, and is very simple. However, it is mentioned in [12] that this code, like other variable-length codes, also suffers from the following drawbacks. If the input sequences are not sufficiently random, then with an all-one input sequence the code rate decreases to 0.5. However, one may use a scrambler to make the input sequence sufficiently random prior to encoding. The more important concern is that the output block will have variable length, which may introduce difficulties in some systems. The effort required to accommodate the variable length output sequence may require additional overhead, which may preclude the use of this variable-length code because efficient fixed-length codes have been designed for this particular constraint.

Synchronous variable-length constrained sequence codes

To deal with the issue of variable-length encoded sequences, synchronous variable-length constrained sequence codes with fixed code rates have been proposed. These codes ensure that the ratio of the lengths of the variable length source words to variable length codewords is constant. Several synchronous constrained sequence variable-length codes are introduced below.

Example: (2, ∞) RLL code A synchronous variable-length constrained sequence code with parameters $d = 2$, $k = \infty$ and code rate $R = 1/2$ is shown in Table 1.4. It is important to note that, even though source words are not of the same length, each source word is mapped to a codeword exactly twice its length. Therefore, given a block of source bits, the block of coded bits is fixed-length, which is straightforward to implement in real systems. Furthermore, as shown in [12], a fixed-length code with the same RLL parameters and the same code rate has 128 words in the code table while the code in Table 1.4 only has three codewords. This shows one of the advantages of variable-length codes.

The encoding and decoding process is simple due to the prefix property. The encoder first recognizes the first bit of the input source bit stream. If the

Table 1.4: The code table of synchronous variable-length (2, ∞) code

Source bit	Coded bits
0	00
10	0100
11	1000

bit is 0, it is recognized as a source word. Otherwise the encoder checks the second bit and recognizes either 10 or 11 as a source word. In this way, the input source bit stream is partitioned into 1-bit or 2-bit source words. Then, by referring to the code table, the input bit partitions can be mapped to the corresponding output sequences.

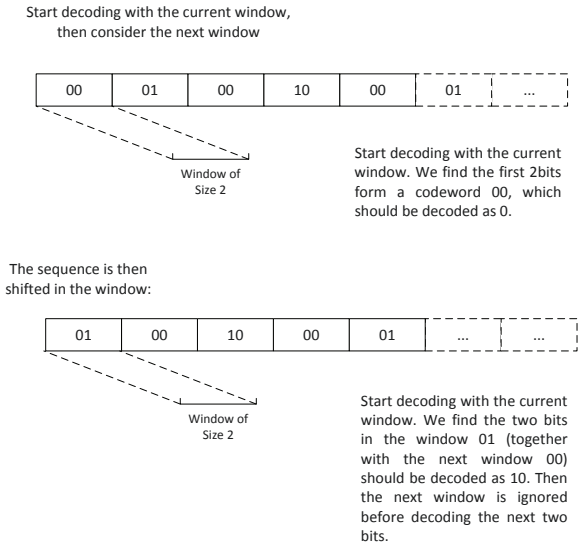

Figure 1.4: The sliding-block decoder for the (2, ∞) synchronous variable-length RLL code

Since no codeword appears in the most significant bit positions of another, the codewords are all prefix-free. One way to decode them is to use a sliding-block decoder with a window size of 2 bits, which is explained as follows. The received coded sequence is sent into the decoder. The received bits are checked two by two in a window. If the received bits in the window are 00, the corresponding source bit is 0. If the received bits in the window are 01 or 10, the corresponding source bit is 10 or 11, respectively, and the next two

bits 00 are ignored before starting decoding the bits in the next window. The sliding-block decoding process is shown in Fig. 1.4.

Table 1.5: The code table of synchronous variable-length (2, 7) code

Source bit	Coded bits
10	1000
11	0100
011	000100
010	001000
000	100100
0011	00100100
0010	00001000

Example: (2, 7) RLL code

The $d = 2$, $k = 7$ synchronous variable-length RLL code with code rate $R = 1/2$ has constituted the bedrock of hard-disk drives [17]. The code table is shown in Table 1.5.

Thanks to the prefix property of the code, the encoding and decoding process is similar to the synchronous variable-length (2, ∞) code. The input sequences are divided into blocks of 2, 3 and 4 bit partitions in accordance with the words in the code table. For example, if the input source bit sequence is the 11 bit stream 01100111000..., the output coded bit sequence is the 22 bit stream 0001000010010001001001000.... The decoding can also be implemented with a sliding-block decoder with window size of 2 bits, similar to the sliding block decoder described above.

Lastly, we also note that a fixed-length code with the same RLL parameters can be designed with a codeword length of 34 [12]. It is therefore evident that the complexity and hardware requirements of fixed-length codes may be much higher than that of variable-length codes.

1.3 Motivation of the proposed variable-length codes

Although we introduced some variable-length constrained sequence codes in the previous section, to the best of our knowledge, most constrained sequence codes that have been employed to date are fixed-length codes. In those codes, source sequences are divided into fixed-length blocks that are sent to the encoder. The encoder then outputs fixed-length coded blocks to the transmission or recording channel. In the following we provide justification why developing variable-length constrained sequence codes is now of value.

With fixed-length codes, the code rate is a rational number. However, in most cases, the capacity of a constrained system is an irrational number, making it impossible to design capacity-achieving fixed length codes [12, 13]. In order to even approach capacity, either long codewords, complex design procedures or complex encoding and decoding techniques are usually required. Although synchronous variable-length constrained sequence codes are proposed, they still have a fixed and rational code rate, thus they do not achieve the full flexibility of variable-length codes, and their code rates are typically still a few percent below capacity.

We note that it could be worthwhile to develop variable-length constrained sequence codes for the following reasons. First, we claim that capacity-approaching variable-length constrained sequence codes with code rates only a few hundredths of a percent away from capacity can be developed. Examples of such codes are given in [18, 19, 20, 21, 22], and will be discussed later in this thesis. The design and implementation of those codes can also be very simple. Therefore, we believe it can be beneficial to accommodate the variable-length nature of the sequences in order to achieve the excellent efficiency possible with these codes, which can be implemented with fairly simple hardware. Second, variable-length blocks fit naturally into modern systems such as variable-length packet-based transmission systems, in software solutions such as software defined radio systems, and in today's

high-capacity storage systems and in systems using variable-length compression techniques. For example, TCP/IP communications, which is based on variable-length data frames, is now widely used in transmission systems. Recently, TCP/IP packets integrated with network coding have been designed for wireless communications and have been reported to improve the system throughout several times [23]. Since TCP/IP packets are by definition variable-length, additional variability introduced by variable-length codes does not introduce any significant drawbacks. File compression systems also have a variable-length output file size since the size of the compressed file is unknown beforehand. In contrast, fixed-length constrained sequence codes designed for CD and DVD were developed in 1980s and 1990s. With new technology being used in modern systems, we believe variable-length codes should be considered as an option for emerging transmission systems and storage devices.

In addition to the arguments above, we note that the implementation of variable-length constrained sequence codes can be adapted to systems that require fixed-length input blocks transmitted to the channel, as shown in Fig. 1.5. The output of the encoder is a variable-length coded block, which can then be sent to a buffer. The buffer then outputs fixed-length blocks row-by-row to fit into the requirement of fixed-length input stream for the channel. If the coded sequence ends before the end of the output block, then padding is required in order for the output block to be full. Decoding can be performed with a sliding-block decoder or another type of decoder. As has already been demonstrated in some of the examples given above and as will be shown later in this thesis, it is expected that in some situations, simple, variable-length codes can achieve higher code rates than fixed-length codes, resulting in higher efficiency and lower implementation complexity. The design and application of capacity-approaching variable-length constrained sequence codes is the subject of this thesis.

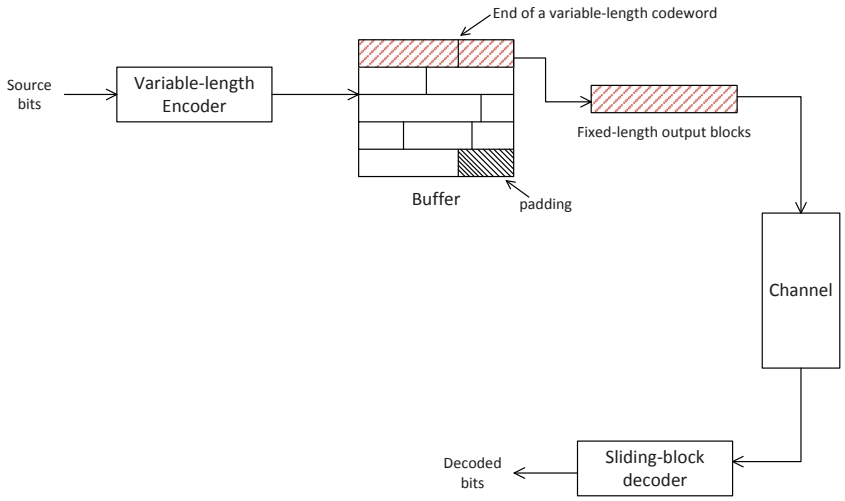

Figure 1.5: An overview of implementation of variable-length constrained sequence codes

In the rest of this chapter, we briefly introduce the theory of constrained sequence codes, and then review a construction technique for capacity-approaching variable-length constrained sequence codes originally reported in [24]. We extend the development of those codes throughout the remainder of this thesis.

1.4 Review of capacity-approaching variable-length constrained sequence codes

1.4.1 Brief review of constrained coding theory

It is well known that constraints can be modeled by Markov chains. In this thesis we restrict our discussion to ergodic Markov chains. An ergodic Markov chain is one in which any state can eventually be reached from any other state [12]. For an FSM with S states, the matrix of the directed graph underlying the constraint is denoted by an $S \times S$ adjacency matrix $\mathbf{D} = \{d_{ij}\}$, where d_{ij} is the number of edges going from state i to state j. The single step transition matrix is denoted by an $S \times S$ matrix $\mathbf{Q} = \{q_{ij}\}$, where q_{ij} is

the transition probability of going from state i to state j. With the transition matrix $\mathbf{Q}$ and an initial probability vector $\mathbf{w^{(1)}}$ of length S, the steady-state probability distribution of the Markov chain $\boldsymbol{\pi}$ is defined as

$$\boldsymbol{\pi} = \lim_{t\to\infty} \mathbf{w^{(1)}}\mathbf{Q}^{t-1} \tag{1.1}$$

where t is number of the transition steps. Given the steady-state distribution, the entropy of the S-state Markov chain $H\{X\}$ can be evaluated as

$$H\{X\} = \sum_{k=1}^{S} \pi_k H_k \tag{1.2}$$

where H_k is the entropy of the k-th state in the Markov chain which is defined as

$$H_k = -\sum_{i=1}^{M} h_i \log_2 h_i, \tag{1.3}$$

and $h_1, h_2, ..., h_M$ are the transition probabilities along the M edges that exit the k-th state. The *capacity* of a Markov chain representing the information source is the value of $H\{X\}$ maximized over the transition probabilities.

Shannon defined the capacity of a constrained sequence C as [7]

$$C = \lim_{m\to\infty} \frac{\log_2 \mathcal{U}(m)}{m} \tag{1.4}$$

where $\mathcal{U}(m)$ is the number of constraint-satisfying sequences of length m. With the FSM description of the constraint and its corresponding adjacency matrix $\mathbf{D}$, he showed that the capacity can also be evaluated by calculating the logarithm of λ_{max}, where λ_{max} is the largest real root of the determinant equation

$$\det[\mathbf{D} - z\mathbf{I}] = \mathbf{0} \tag{1.5}$$

and where $\mathbf{I}$ is an identity matrix. The capacity, with units of bits of information per symbol, can therefore be written as

$$C = \log_2 \lambda_{max}. \tag{1.6}$$

As mentioned above, capacity is achieved with maxentropic transition probabilities, which can be calculated as follows. Let vector $\mathbf{p}$ be the

eigenvector associated with the eigenvalue λ_{max}, which is

$$\mathbf{Dp} = \lambda_{max}\mathbf{p}. \tag{1.7}$$

The maxentropic state transition probability from state i to state j is [12]

$$q_{ij} = \lambda_{max}^{-1} d_{ij} \times \frac{p_j}{p_i} \tag{1.8}$$

where p_i, p_j are the i-th and j-th element of the eigenvector $\mathbf{p}$. If the constraint can be equivalently represented by a single-state FSM, the maxentropic probability of a variable-length codeword of length o_i is given as [24]

$$p_i = {\lambda_{\max}}^{-o_i} = 2^{-o_i C}. \tag{1.9}$$

where o_i is the length of the i-th codeword.

The *maximum possible code rate* $\widetilde{C}$ of constrained coded sequence constructed with single-state codewords whose lengths are from the set M, is

$$\widetilde{C}_M = \log_2 \widetilde{\lambda}_{max} \tag{1.10}$$

where $\widetilde{\lambda}_{max}$ is the largest eigenvalue of the characteristic equation $\sum_{i \in M} \lambda^{-l_i} = 1$ and l_i is the length of i-th word in M. We use L_M to denote the set of lengths of words in M, and $|L_M|$ to denote the size of L_M, i.e., the number of words in M. It should be emphasized that a word length l_i could appear more than once in L_M, since different words might have the same length.

Given a one-to-one correspondence between variable-length source words and variable-length encoded codewords, and assuming equiprobable and independent bits in the source bit stream, the average code rate $\bar{R}$ is

$$\bar{R} = \frac{\sum_{s_i} 2^{-s_i} s_i}{\sum_{o_i} 2^{-s_i} o_i} \tag{1.11}$$

where s_i is the length of i-th source codeword that is mapped to the i-th codeword of length o_i. The efficiency of a code is defined as

$$\eta = \bar{R}/C. \tag{1.12}$$

1.4.2 Minimal sets and extensions

Sequences satisfying a constraint can be generated through free concatenation of words in a minimal set of the constraint [24]. A *complete minimal set* of a constrained sequence code is a set of words whose concatenation generates all possible constraint-satisfying sequences. As discussed in [24], this set can be established by enumerating all words that originate from and end in any specified state in the FSM describing the constraint. However, sometimes there exists an infinite number of words in the minimal set, and this set must be truncated in order to generate a practical codebook. In this case, the truncated set is known as an *incomplete minimal set*. For instance, in the FSM of the $(d = 1, k = \infty)$ RLL constraint shown in Fig. 1.6, a complete minimal set is $A = \{0, 10\}$ if state 1 is selected as the specified state. An incomplete minimal set can be established if state 2 is selected as the specified state, resulting in the set $B = \{01, 001, 0001, ...\}$ which is infinite because of the loop at state 1. Therefore, in practice we must truncate B to force it to be finite.

Let minimal set M contain k words. A *partial extension* of M, denoted M_p, is formed by concatenating all k words in M to any single word in M, generating $k + k - 1 = 2k - 1$ words, where k words are newly generated words and $k - 1$ words are from the previous partial extension. Subsequent partial extensions can be performed by appending the k words from M onto any word from the previous extension. On the other hand, a *complete extension* of M, denoted M_c, is formed by concatenating all k words in M with each of the k words in M to generate k^2 words, and then recursively concatenating all k words in M with each of the words in the newly generated set to generate additional complete minimal sets. A *full extension* of M, denoted M_f, is formed through continuously performing complete extensions of M until asymptotically M_f contains an infinite number of words.

Based on A, two possible partial extensions are $A_{p_1} = \{00, 010, 10\}$ or $A_{p_2} = \{000, 100, 010, 0010, 1010\}$, based on which further extensions can be

constructed, as shown in Fig. 1.7. A complete extension is $A_{c1} = \{00, 010, 100, 1010\}$ based on which another complete extension can be generated as $A_{c2} = \{000, 0010, 0100, 01010, 1000, 10010, 10100, 101010\}$. After establishing the minimal set and performing extensions, we obtain a set of variable-length codewords. Note that because the minimal sets that we construct have the characteristic that no word in the set is a prefix of another, this characteristic extends also to our sets of variable-length codewords, which enables instantaneous decoding of the encoded sequence [24, 25].

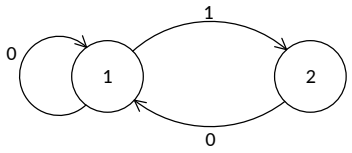

Figure 1.6: FSM of a $(1, \infty)$ code

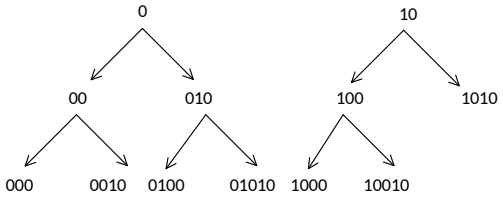

Figure 1.7: An example of a partial extension of the minimal set $\{0, 10\}$

As noted above, we describe the maximum possible code rate with minimal set M as $\widetilde{C}_M$. If all constraint-satisfying sequences can be generated by concatenating codewords in M, then M is a complete minimal set and is able to achieve capacity, i.e. $\widetilde{C}_M = C$. However, in some cases there are an infinite number of codewords in M because of loops that exist in the FSM. In those cases, let l_{max} be the maximum length of words in a truncated minimal set. It can be shown that $\widetilde{C}_M \to C$ as $l_{max} \to \infty$. However in practical code construction, l_{max} should be finite. In those cases, $\widetilde{C}_M < C$, and $\widetilde{C}_M$ can be determined according to (1.10).

1.4.3 Normalized geometric Huffman coding

After constructing a set of potential codewords through partial extensions, the next task is to assign these codewords to corresponding

source words such that the maximum information density is approached. This is done with normalized geometric Huffman (NGH) coding [26] [27]. For each set of partial extensions, NGH coding can be used to obtain the optimal mapping between variable-length source words and variable-length codewords. Starting with the desired codeword probabilities

$$q_{o_i} = 2^{-o_i C} \tag{1.13}$$

as the input probabilities, NGH coding merges the two smallest probabilities in each encoding stage by evaluating their geometric mean, unless these probabilities are so different that there is no advantage to merging these values. In this situation, the least probable codeword is simply discarded [26] [27]. More specifically: in the merging process of q_i and q_j where $q_i \geq q_j$, the merged probability q_{merged} is

$$q_{merged} = \begin{cases} 2\sqrt{q_i q_j} & \text{if } q_i < 4q_j \\ q_i & \text{if } q_i \geq 4q_j. \end{cases} \tag{1.14}$$

The smaller probability is pruned from the Huffman tree when the lower condition is satisfied. After obtaining $\bar{R}$, we replace C by $\bar{R}$ in (1.13) and repeat the above process, until $\bar{R}$ converges.

An example of NGH coding is shown in Fig. 1.8. In this example, a rate 0.6923 variable length ($d = 1, k = \infty$) constrained sequence code is constructed that is within 0.28% of the capacity of this constraint. The codewords $\{000, 100, 010, 0010, 1010\}$ represent the source words $\{11, 10, 01, 001, 000\}$. As will be the case throughout this thesis, we evaluate the code rate for this code assuming equiprobable and independent bits in the source sequence. More detailed discussions on NGH coding can be found in [26] [27] .

Since different partial extensions result in different codebooks with different efficiencies η as discussed in [25], we establish limits such as the maximum number of codewords in the codebook n_{max} or the maximum length of codewords in the codebook c_{max}, and search over all codebooks within these bounds to determine $\bar{R}$ of each codebook and choose the one with the highest efficiency.

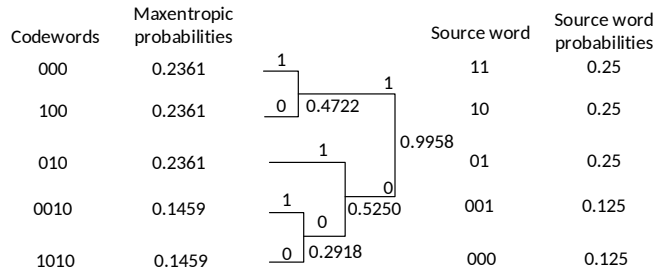

Figure 1.8: Construction of a rate 0.6923 ($d = 1, k = \infty$) code via NGH coding, $\eta = 0.9972$

Example ($(d = 1, k = 3)$ RLL constraint): the FSM of a (1,3) RLL code is shown in Fig. 1.9. The minimal set based on state 1 is $\{01, 001, 0001\}$. If we perform NGH coding with this minimal set, we obtain a simple codebook with efficiency of 98.9% as shown in Table 1.6. If we perform extensions with the $c_{max} = 10$, the highest efficiency we can achieve is 99.24%, with the codebook shown in Table 1.7. As a point of comparison, we note that a widely used (1,3) RLL code is the MFM code with $\eta = 91\%$ [12].

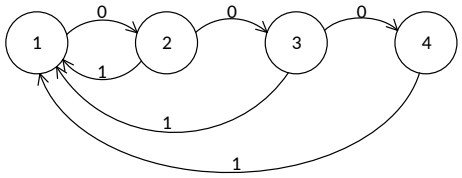

Figure 1.9: FSM of a $(1, 3)$ RLL code

Table 1.6: A codebook of (1,3) RLL code with efficiency of 98.9%

Source word	Codeword
0	01
10	001
11	0001

Example (DC-free constraint with $N = 3$): the FSM of a DC-free code that takes only three different RDS values, i.e., $N = 3$, is shown in Fig. 1.10. The minimal set based on state 2 is $\{01, 10\}$. We may directly perform NGH coding over the words in this minimal set to obtain a codebook with $\eta = 100\%$, as shown in Table 1.8. There is then no purpose in examining

Table 1.7: A codebook of (1,3) RLL code with efficiency of 99.24%

Source word	Codeword
0	01
100	00101
1010	0010001
1011	0001001
1100	00010001
1101	00100101
11100	00010101
11101	001001001
11110	000101001
111110	0010010001
111111	0001010001

partial extensions and developing a longer code. Note that, in this case, the result happens to be a fixed-length code and this is in fact the Manchester code used in many applications such as visible light communications [3].

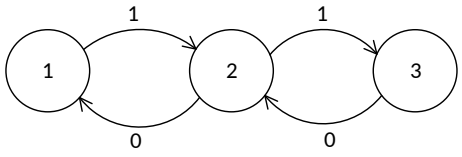

Figure 1.10: FSM of a DC-free code with $N = 3$

Table 1.8: A codebook of $N = 3$ DC-free code with efficiency of 100%

Source word	Codeword
0	01
1	10

1.5 Context and contributions of work in this thesis

In this section we highlight the context, contributions, and the significance of the work in this thesis.

We begin by noting that constrained sequence codes, which were developed and have evolved independently from error control codes, serve a fundamentally different purpose than error control codes. Error control codes attempt to overcome the effects of signal distortion after it has occurred. In contrast, constrained sequence codes attempt to reduce the amount of signal distortion that occurs on the channel by shaping the signal such that it matches the constraints of the channel. The effect of constrained sequence coding is therefore to reduce the likelihood of errors occurring during demodulation. Both constrained sequence coding and error control coding serve critical functions in high performance transmission and recording systems.

Placing constraints on the types of sequences that can be used on the channel means that sequences that violate system constraints cannot be used, which places a limit on the number of sequences of a given length that can be employed. Assuming that all possible sequences may occur at the input to the constrained sequence encoder, since there will be fewer valid encoded sequences of the same length, the length of the encoded constrained sequences must be longer than the length of the source sequence in order to convey all possible source messages. If the source sequence contains one bit of information in each source bit, then the amount of information per encoded bit must be less than one. The maximum amount of information per encoded bit is known as the capacity of the constraint [7], and can be calculated as given previously in Eq. (1.4).

We note that this definition of capacity is fundamentally different than the channel capacity which is of interest for error control codes. The capacity of the channel is defined as the maximum mutual information between the transmitter and the receiver. The maximum code rate of an error control

code that one can use in hopes of achieving error-free communication is upper bounded by the capacity of the channel, when capacity is expressed as bits of information per coded symbol.

Code rates of constrained sequence codes cannot exceed the capacity of the constraint. We define the efficiency of a code as the ratio of the code rate to the capacity, and seek to construct codes with efficiencies as close to 1 as possible. As noted earlier, the capacities of almost all constraints are irrational [12, 13]. Since fixed-length codes have rational code rates, it may not be possible for these code rates to approach capacity with practical word lengths. We note that the average code rate of our variable length codes, as given by Eq. (1.11), is also rational. However, the construction of variable-length codes provides considerable flexibility in the mapping of variable-length source words to variable-length codewords, hence many more rational rates are possible with practical variable-length codes so there are many more opportunities for constructing practical codes with rates close to capacity.

Variable-length codes are commonly used in source coding applications, also referred to as compression. The most widely known variable-length compression technique is Huffman coding. However, variable length techniques are not widely used in constrained sequence applications. Drawbacks and advantages of variable-length constrained sequence techniques were discussed earlier in this chapter.

The variable-length techniques developed in this thesis are an extension of the code construction approach first outlined in [24], which was summarized above in Section 1.4. To the best of our knowledge, the approach outlined in [24] is the first general technique that has been developed to construct highly-efficient variable-length codes for any constraint that can be modeled by a finite state machine.

In this thesis, we extend that work in the following ways.

- In Chapter 2 we further develop the approach outlined in [24] by identifying the appropriate state within the finite state machine upon which to base the code construction, and also developing a multi-state

construction technique. In doing so we propose two systematic approaches to construct capacity-approaching variable-length constrained sequence codes that have higher code efficiency than most codes in the literature.

- In Chapter 3 we consider the application of constrained sequence codes in flash memories and propose variable-length constrained sequence coding schemes to improve the error rate performance when reading from a flash memory.

- In Chapter 4 we consider the synchronization of variable-length codes in terms of correctly identifying the codeword boundaries, and we show that these variable-length codes exhibit good synchronization properties which enable the receiver to recover from mis-synchronization quickly once synchronization is lost.

Overall, this thesis develops construction techniques for highly efficient variable-length constrained sequence codes and demonstrates their practicality in high performance communication systems.

Chapter 2

Novel code construction techniques for general constraints

In this chapter we propose two extensions of the technique outlined in [24] that result in novel, systematic approaches to design variable-length constrained sequence codes for a wide variety of constraints. These techniques are based on an encoder that uses a single encoding state, and an encoder that uses multiple encoding states. Both approaches constitute original work that enables the construction of codes with capacity-approach code rates that outperform many codes in the literature. This original work has been published in [18, 19] and [22].

2.1 Construction of capacity-approaching codes with a single encoding state [18, 19]

Although the construction technique summarized in Section 1.4, and originally described in [24], is efficient for some types of constraints, the construction of minimal sets in [24] is still empirical and lacks a rigorous methodology. Therefore, the generalized construction of minimal sets given

any type of constraint is still an open research area. For example, the FSM of a general DC-free constraint is shown in Fig. 2.1. Since multiple states exist in the FSM of DC-free codes, prior to this work it was not known how to decide which state should be selected as the specified state in order to establish the minimal set. Moreover, it is not straightforward to enumerate the words in the minimal set directly due to the loops that exist in the FSM.

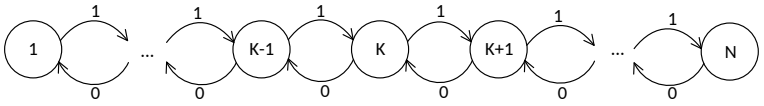

Figure 2.1: FSM of general DC-free codes

In this section we first introduce criteria for choosing the specified state which results in the minimal set with the highest maximum possible code rate, and then we propose a general recursive algorithm to establish the minimal set of a given type of constraint that could be used to construct high-efficiency constrained sequence codes. Examples of the proposed algorithm are given for different constraints.

2.1.1 Selection of specified states

In this section, we outline three criteria for selecting the specified state which results in the minimal set that has the highest maximum possible code rate. We discuss examples to illustrate each of the criteria, starting with a discussion of the complete/incomplete minimal set.

Criterion 2.1

Consider again the $(d = 1, k = \infty)$ RLL constraint whose FSM is shown in Fig. 1.6, and its possible minimal sets A and B. From a capacity point of view, minimal set A is capacity-achieving because use of (1.10) with the characteristic equation $\lambda^{-1} + \lambda^{-2} = 1$ yields exactly the same result as (1.6): $C_A = \widetilde{C}_A = \log_2 \frac{1+\sqrt{5}}{2} = 0.6942$, which is the capacity of the $(d = 1, k = \infty)$ RLL constraint [12]. With an infinite number of words in B, however, the characteristic equation has an unbounded number of terms, and use of (1.10)

with a truncated characteristic equation results in $\widetilde{C}_B < 0.6942$, reflecting the use of an incomplete minimal set and thus not all constraint-satisfying sequences in the encoded bit stream. Moreover, it can be shown that the characteristic equation can be written:

$$\sum_{i=2}^{\infty} \lambda^{-i} = \lambda^{-2} + \frac{\lambda^{-3}(1 - \lambda^{-(N_m-1)})}{1 - \lambda^{-1}} = 1 \tag{2.1}$$

where N_m is the total number of words in B. The left-hand side indicates the potentially unlimited number of words in B. As $N_m \to \infty$, $\lambda^{-(N_m-1)} \to 0$ and (2.1) reduces to $\lambda^{-2} + \lambda^{-1} = 1$ and hence $\lim_{N_m \to \infty} \widetilde{C}_B = \widetilde{C}_A = 0.6942$. Therefore capacity can be achieved if we allow an infinite number of words in B. This trend is shown in Fig. 2.2 where it is evident that as l_{max} increases, $\widetilde{C}_B$ asymptotically approaches capacity. However, in practice we can only allow a finite number of words in B, hence the capacity is not achievable with practical encoding schemes. Therefore, when possible, we recommend selecting a state that results in a complete minimal set such that capacity can be achieved. This leads to Lemma 2.1

Lemma 2.1 Given two states A and B, if A results in a minimal set with a finite number of words and if B results in a minimal set with an infinite number of words, then A is superior to B in terms of maximum possible code rate, except as $l_{max} \to \infty$ in which case they are equal.

Lemma 2.1 guides selection between states resulting in a complete minimal set and an incomplete minimal set. We now introduce Lemma 2.2 regarding selection between two complete minimal sets.

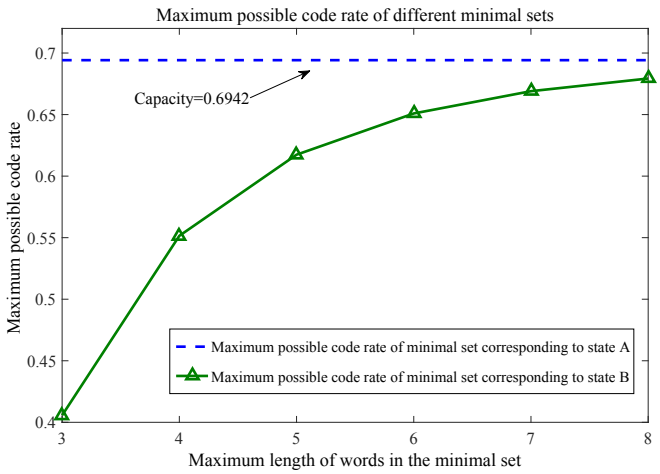

Figure 2.2: Maximum possible code rates of different minimal sets in a $(d = 1, k = \infty)$ RLL code

Lemma 2.2 If there exist two complete minimal sets A and B in an FSM description of a certain constraint, we have $L_A = L_B$, meaning that the sets of word lengths in A and B are identical.

Proof. Since A and B are complete minimal sets, there is no loop associated with a third state S_C when enumerating words exiting and re-entering the specified states S_A or S_B. For an ergodic Markov chain, the only loops that are possible either start from S_A, go to S_B, and come back to S_A, or start from S_B, go to S_A, and come back to S_B. Therefore, the words generated based on state S_A and S_B have the same length since they are generated by tracing the same edges in the FSM, only in different order. □

We now generalize Lemmas 2.1 and 2.2 to the following criterion.

Criterion 2.1 When selecting among several minimal sets based on an FSM description of a constraint, any set that has a finite number of words is equally preferred in terms of maximum possible code rate.

Example ($(d = 1, k = 2)$ RLL constraint): The FSM of the $(d = 1, k = 2)$ RLL constraint is shown in Fig. 2.3. Based on Criterion 2.1, we conclude that states 1 and 2 are equally preferred since they both result in complete minimal sets. Furthermore, although the minimal sets have different words, their words have the same length, i.e., $L_1 = L_2 = \{2, 3\}$. Therefore, the codes

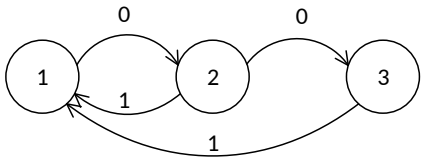

Figure 2.3: FSM of a $(d = 1, k = 2)$ RLL code

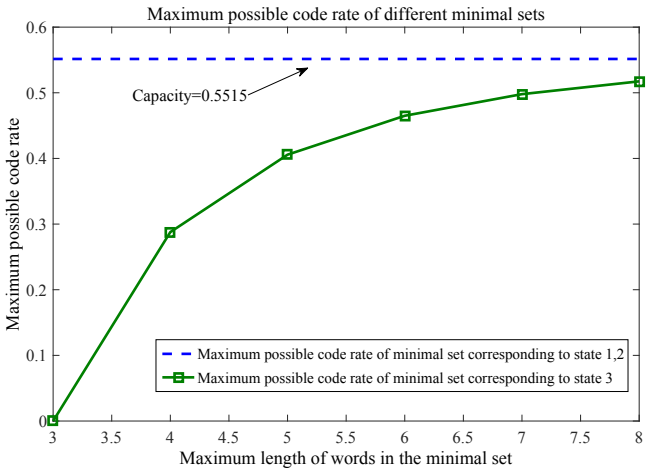

Figure 2.4: Maximum possible code rates of different minimal sets in a $(d = 1, k = 2)$ RLL code

constructed have the same rate, although the codebooks are different. State 3 is not preferred since it results in an incomplete minimal set. We also note that for all (d, k) RLL constraints, there always exists at least one state that results in a complete minimal set and hence should be selected as the specified state such that capacity is achievable.

The maximum possible code rates of the states in the $(d = 1, k = 2)$ RLL constraint are shown in Fig. 2.4. It can be seen that the maximum possible code rate for codes constructed from minimal sets based on states 1 and 2 is exactly the capacity of the $(d = 1, k = 2)$ RLL constraint; the maximum possible code rate for codes constructed based on state 3 increases with increasing l_{max}, but capacity is not achievable with a finite number of words in the minimal set.

Criteria 2.2 and 2.3

With Criterion 2.1, we are able to select an optimal state based on the FSM description of a constraint that can be represented by a complete minimal set. However, Criterion 2.1 does not apply when we are unable to construct a complete minimal set for a constraint. In this section, we discuss selection criteria when selecting among incomplete minimal sets.

As an example to motivate the consideration of incomplete minimal sets, we consider DC-free codes because of their importance in transmission and data storage systems. Minimal sets for most DC-free codes are incomplete. The FSM in Fig. 2.1 describes bit-by-bit generation of DC-free binary sequences that take on at most N different RDS values. Note that if $N \geq 4$, once any state in the FSM depicted in Fig. 2.1 is exited, there is no limit to the number of state transitions that may occur before it is re-entered. (States 1 and 3 also exhibit this property when $N = 3$.) Therefore there is no limit to the length of sequences that can be generated before a state is re-entered, and it is not possible to construct a complete minimal set.

It is, however, possible to construct incomplete minimal sets with maximum possible code rates $\widetilde{C}_M$ approaching C. To do so, denote $|l_k|$ as the number of words that are of length l_k in the minimal set. Our task is to determine $|l_k|$ for all $l_k, 1 \leq l_k \leq l_{max}$ that can occur as a particular state in the FSM is exited and then re-entered, starting with the shortest possible sequence length up to l_{max}. We then use these sequences as words in the minimal set. Since this FSM does not generate all possible sequences that satisfy the RDS constraint, the maximum possible code rate $\widetilde{C}_M$ of sequences generated by this single-state FSM will be strictly less than the capacity C for the original constraint, and the collection of these edge labels comprises an incomplete minimal set. $\widetilde{C}_M$ can be evaluated according to (1.10) as shown in the example below.

The values of $|l_k|$ can vary depending on the particular state considered in the FSM. Since $\widetilde{C}_M$ is a function of the edge labels $|l_k|$, then $\widetilde{C}_M$ can depend on the state from the FSM that is used when enumerating $|l_k|$. For example,

Table 2.1: Capacity of single-state DC-balanced FSM with $N = 4$. Analysis of the multi-state FSM yields $\lambda_{max} = 1.6180$ and $C = 0.6942$.

l_{max}	Based on states 1 or 4			Based on states 2 or 3		
	$\widetilde{\lambda}_{max}$	$\widetilde{C}_{1,4}$	$\frac{\widetilde{C}_{1,4}}{C}$	$\widetilde{\lambda}_{max}$	$\widetilde{C}_{2,3}$	$\frac{\widetilde{C}_{2,3}}{C}$
2	1.0000	0.0000	0.0000	1.4142	0.5000	0.7202
4	1.2720	0.3471	0.5000	1.5538	0.6358	0.9158
6	1.4142	0.5000	0.7202	1.5959	0.6744	0.9714
8	1.4903	0.5756	0.8291	1.6100	0.6871	0.9896
10	1.5341	0.6174	0.8893	1.6150	0.6916	0.9961
12	1.5610	0.6425	0.9254	1.6169	0.6932	0.9985
14	1.5783	0.6584	0.9483	1.6176	0.6939	0.9994
16	1.5898	0.6688	0.9634	1.6179	0.6941	0.9998

consider the DC-balanced code with $N = 4$, a code which has $\lambda_{max} = 1.6180$ and $C = 0.6942$ [12]. When considering sequences that emanate from and return to either state 1 or state 4 in Fig. 2.1, it can be verified that all such sequences have even length, that there is a single sequence of length 2, a single sequence of length 4, and $|l_k| = 2^{(\frac{l_k}{2}-2)}$ for even l_k, $l_k \geq 4$. In contrast, when either state 2 or state 3 is exited and re-entered, it is straightforward to show that $|l_2| = 2$ and $|l_k| = 1$ for all even l_k, $l_k \geq 4$, and zero otherwise.

Solving (1.10) with these values of $|l_k|$ yields the results in Table 2.1 which show how $\widetilde{C}_M$ increases with increasing values of l_{max}. From this table it is clear that regardless of the specified state, $\widetilde{C}_M$ approaches C as l_{max} increases, but that the minimal set based on states 2 or 3 results in a significant capacity advantage for low l_{max}. While $\widetilde{C}_{2,3}$ is within 0.4% of capacity for that model when $l_{max} = 10$, it can be shown that this $\frac{\widetilde{C}_{1,4}}{C} = 0.996$ is not attained for the minimal set based on states 1 or 4 until $l_{max} = 32$. However, in either case it is possible to form an incomplete minimal set that can be used as the basis for a variable-length code whose code rate is upper bounded by a value of $\widetilde{C}_M$ that is close to capacity C.

Based on the discussions above, we conclude that the optimal state to select for DC-free codes with $N = 4$ is state 2 or 3 with $l_{max} \leq 32$. However,

when l_{max} is large, it is not straightforward to determine which state is optimal without counting the number of words in the minimal sets. More importantly, not only for DC-free constraints with large N, but more generally, for arbitrary constraints, the selection of the optimal state that results in the highest maximum possible code rate is not an easy task because the counting of words may become intractable. To tackle these problems, we now discuss criteria for selection of the specified state when all minimal sets have infinite size. We start with the following lemma.

Lemma 2.3: For a minimal set M and the set of word lengths L_M where the length of the longest word is l_{max} and the highest maximum possible code rate is $\widetilde{C}_M$, $\forall l_1, l_2, l_3$ consecutive values $l_1, l_2, l_3 \in L_M$ that satisfy $l_2 - l_1 = l_3 - l_2$, $\widetilde{C}_M = f(l_{\max})$ is a concave monotonically increasing function with respect to l_{max} in L_M.

Proof. We first prove the monotonicity of $\widetilde{C}_M = f(l_{\max})$, and then proceed to prove concavity.

1) We choose two values of l_{max}, i.e. l_1 and l_2. For all $0 < l_1 < l_2 < \infty$, we have the following characteristic equations:

$$\begin{cases} \sum\limits_{l_i \le l_1, l_i \in L_M} \lambda_1^{-l_i} = 1 \\ \sum\limits_{l_i \le l_2, l_i \in L_M} \lambda_2^{-l_i} = 1. \end{cases} \tag{2.2}$$

It is straightforward to show that $\lambda_2 > \lambda_1$ where λ_1 and λ_2 are the largest real roots of the above characteristic equations, since the second equation has more terms on left-hand side. According to Eq. (1.10), we have $\widetilde{C}_1 = f(l_1) < \widetilde{C}_2 = f(l_2)$. Therefore $\widetilde{C}_M = f(l_{\max})$ is a monotonically increasing function.

2) Based on Eq. (2.2), as the size of the minimal set grows, if there exist any three consecutive values of $l_1, l_2, l_3 \in L_M$ where $0 < l_1 < l_2 < l_3 < \infty$ and $l_2 - l_1 = l_3 - l_2$, then we have that $\lambda_1 < \lambda_2 < \lambda_3$. In accordance with Eq. (1.10), the corresponding highest maximum possible code rate then satisfies

$$\widetilde{C}_1 < \widetilde{C}_2 < \widetilde{C}_3 \tag{2.3}$$

When $l_3 \to \infty$, we have the characteristic equation $\sum\limits_{l_i \le l_2, l_i \in L_M} \lambda_3^{-l_i} + \lambda_3^{-l_3} = 1$ where $\lambda_3^{-l_3} \to 0$, and hence $\lambda_3 = \lambda_2$, and $\widetilde{C}_3 = \widetilde{C}_2$ by comparing this equation

and the second equation in (2.2). Considering the asymptotic behavior we know that the rate of increase of $\widetilde{C}$ diminishes and becomes negligible as $l_{max} \to \infty$. We now give the proof of the aforementioned observation for finite l_{max}.

From equation (1.4) we know

$$\begin{aligned} \lambda &= 2^{\lim_{m\to\infty} \frac{\log_2 \mathcal{U}(m)}{m}} \\ &= \lim_{m\to\infty} \sqrt[m]{\mathcal{U}(m)}. \end{aligned} \tag{2.4}$$

We have

$$\begin{aligned} \widetilde{C}_2 - \widetilde{C}_1 &= \log_2 \tfrac{\lambda_2}{\lambda_1} \\ &= \lim_{m\to\infty} \sqrt[m]{\tfrac{\mathcal{U}_2(m)}{\mathcal{U}_1(m)}} \end{aligned} \tag{2.5}$$

and

$$\begin{aligned} \widetilde{C}_3 - \widetilde{C}_2 &= \log_2 \tfrac{\lambda_3}{\lambda_2} \\ &= \lim_{m\to\infty} \sqrt[m]{\tfrac{\mathcal{U}_3(m)}{\mathcal{U}_2(m)}}. \end{aligned} \tag{2.6}$$

We compare $\frac{\mathcal{U}_3(m)}{\mathcal{U}_2(m)} = 1 + |l_3| \frac{\sum_{i=1}^{\mathcal{U}_2(m)} \Delta_{3,i}}{\mathcal{U}_2(m)}$ and $\frac{\mathcal{U}_2(m)}{\mathcal{U}_1(m)} = 1 + |l_2| \frac{\sum_{i=1}^{\mathcal{U}_1(m)} \Delta_{2,i}}{\mathcal{U}_1(m)}$, where $\Delta_{k,i}$ is the number of additional allowable sequences generated based on the i-th sequence in $\mathcal{U}_{k-1}(m)$ when the maximum length l_{max} increases to l_k.

We first consider $|l_2| = |l_3| = 1$ which means that only one loop has to be taken into account when counting words in the minimal set. We will later extend this approach to the general situation where $|l_2| \neq |l_3|$ to consider the possibility of multiple loops. To demonstrate, we show $\mathcal{U}_1(m), \mathcal{U}_2(m), \mathcal{U}_3(m)$ in Fig. 2.5. As shown in that figure, $\mathcal{U}_2(m)$ consists of two parts, i.e. $\mathcal{U}_1(m)$ and the set Ψ where the number of sequences in Ψ is $|\Psi| = \mathcal{U}_2(m) - \mathcal{U}_1(m) = \sum_{i=1}^{\mathcal{U}_1(m)} \Delta_{2,i}$. $\mathcal{U}_3(m)$ consists of $\mathcal{U}_1(m), \mathcal{U}_2(m) - \mathcal{U}_1(m)$ and $\sum_{i=1}^{\mathcal{U}_2(m)} \Delta_{3,i} = \mathcal{U}_3(m) - \mathcal{U}_2(m)$ where the last part comprises of two sets, $\mathbf{X}$ and $\mathbf{Y}$, where the number of sequences in $\mathbf{X}$ and $\mathbf{Y}$ is $|\mathbf{X}| = \sum_{i=1}^{\mathcal{U}_1(m)} \Delta_{3,i}$ and $|\mathbf{Y}| = \sum_{i=\mathcal{U}_1(m)+1}^{\mathcal{U}_2(m)} \Delta_{3,i}$, respectively. Ψ and $\mathbf{X}$ are the additional sequences generated from $\mathcal{U}_1(m)$ when l_{max} increases to l_2 and l_3 respectively. Since $l_3 > l_2$, it is readily apparent that on average, for each sequence in $\mathcal{U}_1(m)$ the

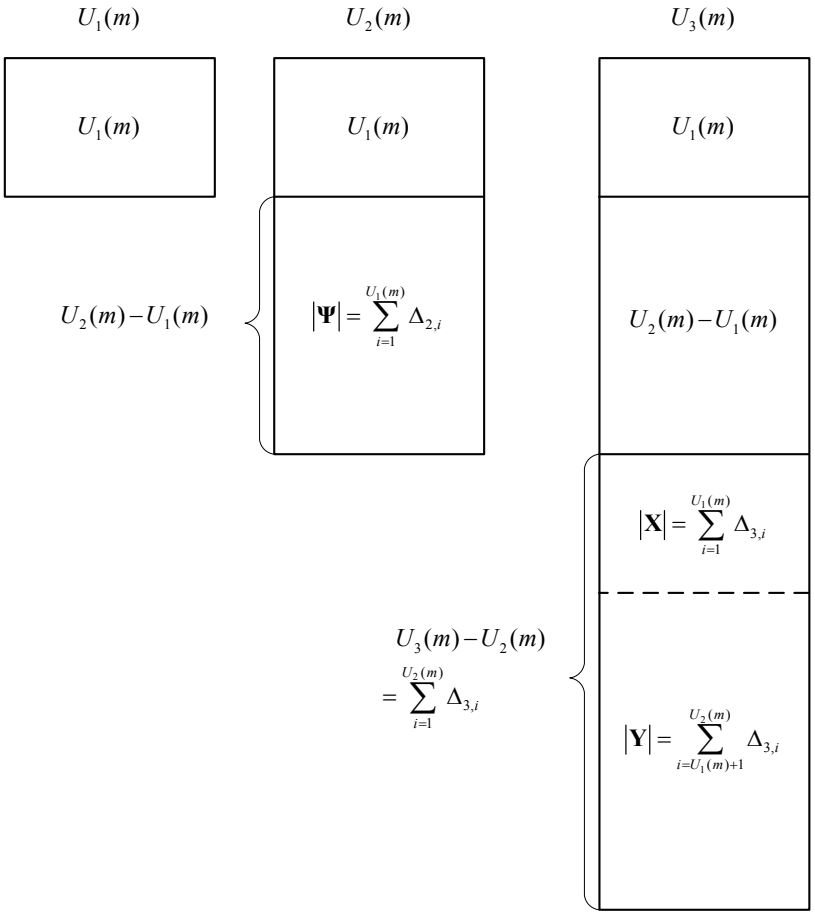

Figure 2.5: Explanation of $\mathcal{U}_1(m), \mathcal{U}_2(m), \mathcal{U}_3(m)$

number of additional allowable sequences when generating $\mathbf{X}$ and Ψ satisfies $\frac{|\mathbf{X}|}{\mathcal{U}_1(m)} < \frac{|\Psi|}{\mathcal{U}_1(m)}$ because for sequences in $\mathcal{U}_1(m)$ the number of additionally generated allowable sequences decreases with larger l_{max}. Furthermore, since we consider $m \to \infty$, $\frac{|\mathbf{X}|}{\mathcal{U}_1(m)} = o(\frac{|\Psi|}{\mathcal{U}_1(m)})$, i.e. $\frac{|\mathbf{X}|}{\mathcal{U}_1(m)}$ is the higher-order infinitesimal of $\frac{|\Psi|}{\mathcal{U}_1(m)}$. To prove that, we consider

$$\begin{cases} \sum\limits_{l_i \le l_1, l_i \in L_M} \lambda_1^{-l_i} = 1 \\ \sum\limits_{l_i \le l_1, l_i \in L_M} \lambda_2^{-l_i} + \lambda_2^{-l_2} = 1 \\ \sum\limits_{l_i \le l_1, l_i \in L_M} \chi^{-l_i} + \chi^{-l_3} = 1. \end{cases} \tag{2.7}$$

It follows that $\lambda_1 < \chi < \lambda_2$, and $\lambda_2^m > \chi^m > \lambda_1^m$. Comparing $|\mathbf{X}| = \lim\limits_{m\to\infty}(\chi^m - \lambda_1^m)$ with $|\Psi| = \lim\limits_{m\to\infty}(\lambda_2^m - \lambda_1^m)$, we obtain $|\mathbf{X}| = o(|\Psi|)$, and thus $\frac{|\mathbf{X}|}{\mathcal{U}_1(m)} = o(\frac{|\Psi|}{\mathcal{U}_1(m)})$.

On average, for each sequence in Ψ, the additional number of allowable sequences $\frac{|\mathbf{Y}|}{|\Psi|}$ when generating $\mathbf{Y}$ is even smaller than $\frac{|\mathbf{X}|}{\mathcal{U}_1(m)}$ because the number of bits that can be substituted by the new word of length l_3 per sequence in Ψ is fewer than in $\mathcal{U}_1(m)$. Combining the results above, it is straightforward to derive

$$\frac{\sum\limits_{i=1}^{\mathcal{U}_2(m)} \Delta_{3,i}}{\mathcal{U}_2(m)} = o(\frac{\sum\limits_{i=1}^{\mathcal{U}_1(m)} \Delta_{2,i}}{\mathcal{U}_1(m)}). \tag{2.8}$$

Now consider the generalized situation where $|l_2| \neq |l_3|$ which means multiple states are associated with loops in the FSM. Generally, $|l_2| \le |l_3|$ because we have more freedom to choose the loops with larger l_i. Let there be Γ states associated with loops in the FSM when counting the words in the minimal set, and let γ_m be the maximum length of a word in the minimal set generated without taking the loops into consideration. If $l_{max} > \gamma_m$, loops will be taken into account when generating words of length $l_{max} \ge l_k > \gamma_m$. If $\gamma_m \ge 2$, $|l_k| > 1$. We upper bound the number of $|l_k|$ as (2.9), which corresponds to the situation where we are free to choose the positions of the $l_k - \gamma_m$ extra bits, and we are also free to assign each loop to any extra bit.

$$|l_k| \leq \left\{ \begin{pmatrix} \Gamma \\ 1 \end{pmatrix} + \begin{pmatrix} \Gamma \\ 2 \end{pmatrix} 2^{l_k-\gamma_m} + \begin{pmatrix} \Gamma \\ 3 \end{pmatrix} 3^{l_k-\gamma_m} + ... + \begin{pmatrix} \Gamma \\ \Gamma \end{pmatrix} \Gamma^{l_k-\gamma_m} \right\} \times \begin{pmatrix} l_k \\ l_k - \gamma_m \end{pmatrix} \tag{2.9}$$

Since we consider finite l_{max} and practical FSMs, Γ, γ_m are finite numbers. Therefore, $|l_3| \,/\, |l_2|$ is also a finite number. According to (2.8), we have

$$|l_2| \frac{\sum_{i=1}^{\mathcal{U}_2(m)} \Delta_{3,i}}{\mathcal{U}_2(m)} = o(|l_3| \frac{\sum_{i=1}^{\mathcal{U}_1(m)} \Delta_{2,i}}{\mathcal{U}_1(m)}) \tag{2.10}$$

It therefore follows that $\frac{\mathcal{U}_3(m)}{\mathcal{U}_2(m)} < \frac{\mathcal{U}_2(m)}{\mathcal{U}_1(m)}$ as $m \to \infty$, and

$$\widetilde{C}_2 - \widetilde{C}_1 > \widetilde{C}_3 - \widetilde{C}_2 \tag{2.11}$$

for l_1, l_2, l_3. This completes the proof that $\widetilde{C}_M$ is a concave function of l_{max}. □

Consider two minimal sets A and B constructed from two different states. When selecting the specified state with a given l_{max}, we consider $\widetilde{C}_A = f(l_{\max})$ and $\widetilde{C}_B = g(l_{\max})$ for the two minimal sets. It could happen that in the range of $0 < l_{max} < l_I$, the maximum possible rates are the same for two minimal sets, and that starting from $l_{max} = l_I$, $\widetilde{C}_A$ and $\widetilde{C}_B$ differ. Also as $l_{\max} \to \infty$, the maximum possible rates of both minimal sets approach capacity. Therefore we consider $\widetilde{C}_A = f(l_{\max})$ and $\widetilde{C}_B = g(l_{\max})$ where $l_I \leq l_{max} < \infty$.

Without loss of generality, assume $f(l_{max} = l_I) > g(l_{max} = l_I)$, and consider the asymptotic behavior. As $l_{\max} \to \infty$, due to the concave nature of $f(l_{\max})$ and $g(l_{\max})$, if $\forall a \to 0$, $\exists l_x = l_{\max} - a$ where $f(l_x) > g(l_x)$, then for any $l_{\max} \in [l_I, \infty)$, $f(l_{\max}) > g(l_{\max})$; otherwise the concavity of $f(l_{max})$ and $g(l_{max})$ will be violated. This observation provides us with the following Criterion 2.2 to determine the specified state: we only need to compare $f(l_I)$ with $g(l_I)$, and $f(l_x)$ with $g(l_x)$. The latter comparison can be accomplished simply by comparing the number of longest words in the minimal set as $l_{\max} \to \infty$. This is possible because as $l_{max} \to \infty$, $f(l_{\max}) = g(l_{\max}) = C$, and by removing

the longest words, the values of $f(l_{\max})$ and $g(l_{\max})$ decrease. The minimal set with the greater number of longest words will lose more information-carrying capacity since more available patterns are lost, and hence will have a lower maximum possible rate. This criterion is summarized as follows.

Criterion 2.2: Given any two minimal sets A, B with $\widetilde{C}_A = f(l_{\max})$ and $\widetilde{C}_B = g(l_{\max})$, if $f(l_I) > g(l_I)$ and A has fewer number of longest words as $l_{\max} \to \infty$, then $f(l_{\max}) > g(l_{\max})$ for $l_{max} \in [l_I, \infty)$, which means that A is superior to B in terms of maximum possible code rate for finite l_{max}.

Example (constrained codes that forbid the 101 pattern): In flash memories, inter-cell interference (ICI) is considered as one of the most dominant sources of errors [20, 28, 29, 30, 31]. ICI refers to the phenomenon that variations of the electrical charge on one floating-gate transistor can change the voltage levels of its neighboring transistors via the parasitic capacitance-coupling effect. Many constraints have been exploited in ICI mitigation [20, 28, 29, 30, 31]. Constrained sequence codes that forbid the pattern 101 have been designed to limit ICI [20], where the FSM describing this constraint is shown in Fig. 2.6. All states result in minimal sets with an infinite number of words. States 1 and 3 are equivalent in terms of maximum possible code rate due to symmetry. When comparing states 1 and 2, first we find $l_I = 3$ and $\widetilde{C}_1 = f(l_I) > \widetilde{C}_2 = g(l_I)$. We then consider the longest codeword. State 2 has one more loop available than state 1 to construct the longest codeword, therefore state 1 has fewer number of longest words than state 2. For example, if the longest word in the minimal set of state 1 has the form $00\underbrace{0.....0}_{\textit{the loop at state 3}}1$, there is a longer word $0\underbrace{0.....0}_{\textit{the loop at state 3}}1\underbrace{1.....1}_{\textit{the loop at state 1}}0$ in the minimal set of state 2. According to Criterion 2.2, state 1 is better than state 2 in terms of maximum possible code rate.

Fig. 2.7 shows the maximum possible efficiency of different minimal sets of constrained sequence codes that forbid the 101 pattern. It can be seen that states 1 and 3 result in higher maximum possible code rates than state 2 for $l_{max} \in [3, \infty)$, which is consistent with the analysis above.

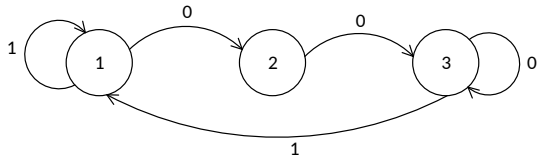

Figure 2.6: FSM of a constrained sequence code that forbids 101 pattern

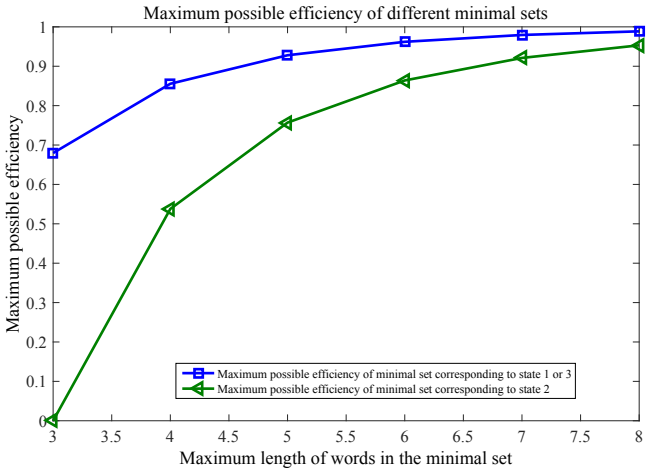

Figure 2.7: Maximum possible code rate of different minimal sets of constrained sequence codes that forbid the 101 pattern

Example (DC-free codes): As noted previously, all states of DC-free codes with $N > 3$ result in minimal sets with an infinite number of words. By applying Criterion 2.2, we can determine that the best state is state $\frac{N+1}{2}$ when N is odd, and $\frac{N}{2}$ or $\frac{N}{2}+1$ when N is even. We discuss odd and even N separately.

When N is odd, we denote set A as the minimal set resulting from state $\frac{N+1}{2}$ and denote set B as the minimal set from any other state. Consider the number of words of increasing length in the minimal sets, and determine the length l_I at which those numbers diverge. It can be seen that $l_I = 2$ when A is compared with the set generated from state 1, i.e. $\widetilde{C}_A = f(l_{max} = 2) > \widetilde{C}_B = g(l_{max} = 2)$. $l_I = 4$ when A is compared with the set generated from state 2, i.e. $\widetilde{C}_A = f(l_{max} = 4) > \widetilde{C}_B = g(l_{max} = 4)$, etc. That is, when comparing state $\frac{N+1}{2}$ with state $1, 2, 3, ..., \frac{N-1}{2}$, we have that, respectively, $l_I = 2, 4, 6, ..., N-1$, and $\widetilde{C}_A = f(l_{max} = l_I) > \widetilde{C}_B = g(l_{max} = l_I)$, because with $l_{max} < l_I$ the

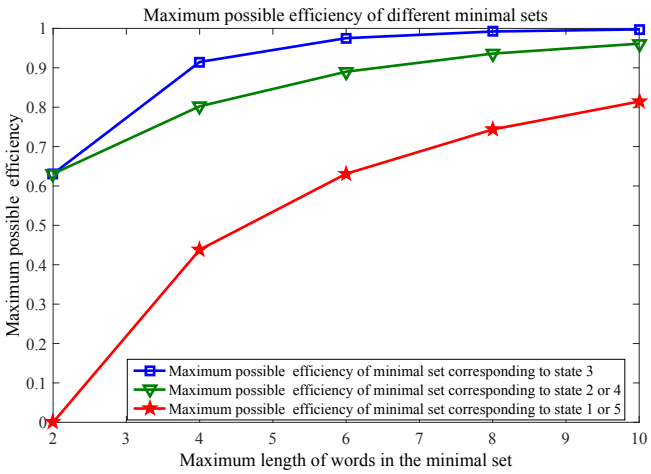

Figure 2.8: Maximum possible code rate of different minimal sets of DC-free codes with $N = 5$

word lengths are identical in the two minimal sets. Considering the longest codeword, it can be observed that every longest codeword in A corresponds to more than one codeword in B of the same length because sequences of edges that generate B include more loops. Therefore state A has fewer number of longest words than B. Comparison of A and minimal sets from states $\frac{N+3}{2}, ..., N$ is similar due to symmetry in the FSM. According to Criterion 2.2, the minimal set from state $\frac{N+1}{2}$ is the best in terms of maximum possible code rate. This is confirmed by the curves shown in Fig. 2.8 for the case when $N = 5$.

When N is even, the procedure is similar to the one discussed above. First we note that due to symmetry, the maximum possible code rates of minimal sets from states $\frac{N}{2}$ and $\frac{N}{2}+1$ are the same since they contain words of the same lengths. We denote set A as the minimal set from state $\frac{N}{2}$ or $\frac{N}{2}+1$, and set B as the minimal set resulting from any other state. Consider the value of l_I. It can be seen that when comparing state $\frac{N}{2}$ or $\frac{N}{2}+1$ with state $1, 2, 3, ..., \frac{N}{2}-1$, $l_I = 2, 4, 6, ..., N-2$ respectively, and $\widetilde{C}_A = f(l_{max} = l_I) > \widetilde{C}_B = g(l_{max} = l_I)$. When considering the longest codeword, it can be observed that A has fewer longest words than B for the reason outlined above. Comparison of A and minimal sets of states $\frac{N}{2}+1, ..., N$ results in the same observations due to

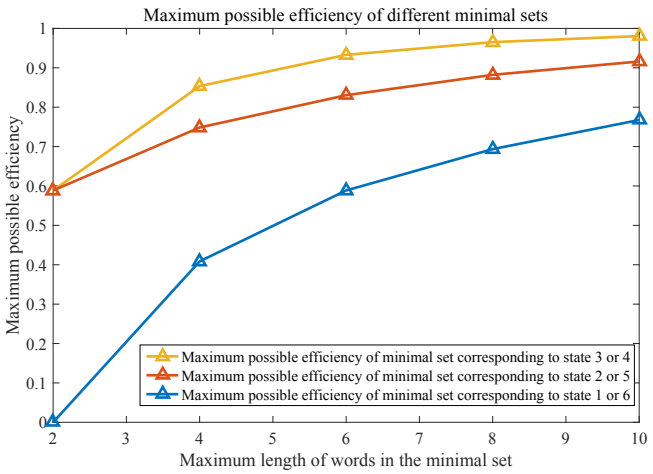

Figure 2.9: Maximum possible code rate of different minimal sets of DC-free codes with $N = 6$

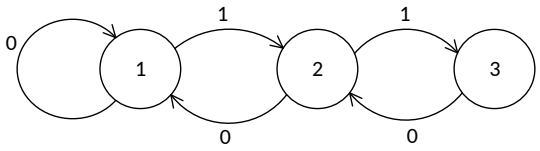

Figure 2.10: FSM of a constrained sequence code that forbids 111 and 11011 patterns

symmetry in the FSM. According to Criterion 2.2, the minimal sets from states $\frac{N}{2}$ or $\frac{N}{2}+1$ are superior to those from any other state in terms of maximum possible code rate. This is confirmed by the curves shown in Fig. 2.9 for $N = 6$.

Example (constrained codes that forbid the patterns 111 and 11011): We note that even if it is not obvious how to compare the number of longest codewords in different minimal sets, Criterion 2.2 can still be applied. For example, the constraint that forbids the pattern 111 and 11011 is of interest when we wish to limit the inter-cell interference in flash memories [20]. In the corresponding FSM shown in Fig. 2.10, it can be seen that states 1, 2 and 3 all have minimal sets with an infinite number of words.

Let minimal sets A and B be constructed from states 1 and 2 respectively. It can be seen that $L_A = \{1, 2, 4, 6, 8, ...\}$ and $L_B = \{2, 2, 3, 4, 5, 6, 7, 8, ...\}$.

When $l_I = 2$, $\widetilde{C}_A = f(l_{max} = l_I) > \widetilde{C}_B = g(l_{max} = l_I)$. Now consider the number of longest words. Although it is not easy to directly determine which minimal set has the greater number of longest codewords, the following observations can be made.

If we set $l_{max} = 2n$ for A and $l_{max} = 2n + 1$ for B where $n = 1, 2, 3, ...$, the following equations can be obtained:

$$\begin{cases} \lambda_1^{-1} + \sum\limits_{l_i=2,4,6...}^{2n} \lambda_1^{-l_i} = 1 \\ \lambda_2^{-2} + \sum\limits_{l_i=2,3,4...}^{2n+1} \lambda_2^{-l_i} = 1. \end{cases} \tag{2.12}$$

where λ_1 and λ_2 are the largest real eigenvalues of the characteristic equations of minimal sets A and B, respectively. We establish another equation as follows:

$$\lambda^{-1} + \sum_{l_i=2,4,6...}^{2n} \lambda^{-l_i} = \lambda^{-2} + \sum_{l_i=2,3,4...}^{2n+1} \lambda^{-l_i} \tag{2.13}$$

The solution to this equation is

$$\lambda^{-1} = \sum_{l_i=2,3,5,7...}^{2n+1} \lambda^{-l_i} \tag{2.14}$$

which is exactly the solution to (2.12). Therefore, the maximum possible code rate of A and B is the same if $l_{max} = 2n$ for A and $l_{max} = 2n + 1$ for B. This property holds until asymptotically capacity is achieved as $n \to \infty$. When considering the longest codeword, B has a codeword of length $2n + 1$ that is longer than the longest codeword in A of length $2n$ by one bit. According to criteria 2, state 1 is better than state 2 in terms of maximum possible code rate.

We then apply Criterion 2.2 to compare minimal sets A and B constructed from states 1 and 3, with maximum possible code rates $\widetilde{C}_A = f(l_{max})$ and $\widetilde{C}_B = g(l_{max})$. It can be seen that when $l_I = 2$, $f(l_{max} = l_I) > g(l_{max} = l_I)$. Now consider the longest words. It is easily seen that the longest codeword in A is shorter than that in B, since there is one more loop available associated with state 1 when generating B. Therefore, A has fewer number of longest

words than B. According to criteria 2, state 1 is the best state in terms of maximum possible code rate.

Fig. 2.11 shows the achievable efficiency of state 1, 2 and 3 with different l_{max} in the FSM describing constrained sequence codes that forbid the pattern 111 and 11011. It can be seen that state 1 results in a minimal set that has the highest achievable efficiency. The achievable efficiency of state 2 at $l_{max} = 2n + 1$ is the same as that of state 1 at $l_{max} = 2n$, which is also consistent with the analysis above.

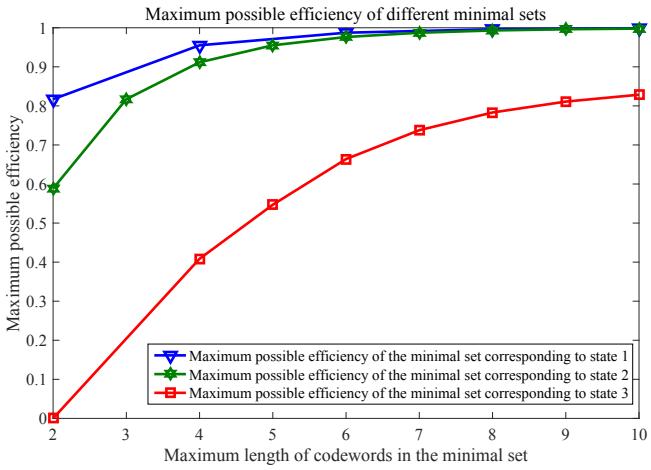

Figure 2.11: Achievable efficiency of different minimal sets of constrained sequence codes that forbids 111 and 11011 patterns

We also infer that due to the concavity of the function $\widetilde{C}_M = f(l_{\max})$, a minimal set with a lower maximum possible code rate corresponds to a higher increasing rate in $\widetilde{C}_M$ at that value of l_{max} for $l_{max} \in [l_I, \infty)$. The higher increasing rate arises from the fact that more words become available due to more loops in the FSM. Generally speaking, once there are more loops available for a minimal set at some value of $l_{max} = l_1$, that set always has more words available with $l_{max} = l_2 > l_1$ since there is greater freedom with more loops when generating new words with length l_2. Thus, once a minimal set has a higher increasing rate, it will maintain the higher increasing rate until capacity is asymptotically achieved. In other words, once a minimal set is worse in terms of maximum possible code rate, it is always worse until it

asymptotically achieves capacity. This implication leads us to Criterion 2.3.

Criterion 2.3: Given two minimal sets A, B with $\widetilde{C}_A = f(l_{\max})$ and $\widetilde{C}_B = g(l_{\max})$, if $f(l_I) > g(l_I)$, then $f(l_{\max}) > g(l_{\max})$ for $l_{max} \in [l_I, \infty)$. This implies that A is superior to B in terms of maximum possible code rate.

Proof. We prove by contradiction. Given $f(l_I) > g(l_I)$ and $v > w > I$, suppose $f(l_w) > g(l_w)$ and $f(l_v) < g(l_v)$. Asymptotically for $l_x = l_{\max} - a \ \forall a \to 0$, due to the concavity of $f(l_{\max})$ and $g(l_{\max})$, $f(l_x)$ should be smaller than $g(l_x)$ under this assumption.

The fact that $f(l_w) > g(l_w)$ and $f(l_v) < g(l_v)$ implies that $|l_{max} = l_v|$ in A is smaller than $|l_{max} = l_v|$ in B because B has gained more information-carrying capacity with $l_{max} = l_v$. With incomplete minimal sets A and B, we know that B has more freedom to choose loops when enumerating words of length l_v. It follows that for $l_v \leq l_{max} = l_k < \infty$, $|l_{max} = l_k|$ in B is always greater than $|l_{max} = l_k|$ in A. However, consider the longest word as $l_{\max} \to \infty$. It is evident that $f(l_x) > g(l_x)$ since $|l_{\max}|$ in B is greater than in A and hence more information-carrying capacity is lost in B than in A with $l_{max} = l_x$. This is contradictory to the condition that $f(l_x) < g(l_x)$. Therefore the assumption does not hold, and hence the two curves of $f(l_{max})$ and $g(l_{max})$ cannot cross when $l_{max} > l_I$, i.e. $f(l_{\max}) > g(l_{\max})$ for $l_{max} \in [l_I, \infty)$. □

Criterion 2.3 is a strengthened version of Criterion 2.2. Based on this criterion, there is no need to compare the number of longest codewords in minimal sets: one just needs to compare the maximum possible code rates $\widetilde{C}_A$ and $\widetilde{C}_B$ at $l_{max} = l_I$. The minimal set with a higher maximum possible code rate at $l_{max} = l_I$ will remain superior until capacity is asymptotically achieved. For example, if we revisit all the examples discussed above, it is readily seen that once a state A is better than B in terms of code rate, i.e. $\widetilde{C}_A = f(l_I) > \widetilde{C}_B = g(l_I)$, A is always better for $l_{max} \in [l_I, \infty)$.

Lastly, we note that the actual curves of $\widetilde{C} = f(l_{\max})$ are not continuous but discrete. However, this does not change the results above when comparing different minimal sets with the same l_{max}.

2.1.2 Code construction algorithm

Given selection of an appropriate state, we now discuss how to construct the minimal set for a general constraint. For some constraints like the $(1, \infty)$ RLL constraint shown in Fig. 1.6, it is straightforward to determine the minimal set by inspection of the FSM. However, for a variety of constraints the FSMs are much more complicated and contain many loops, such as the DC-free constraints which are depicted in Fig. 2.1. From our brief discussion of DC-free codes in the above section, we observe that counting the words in minimal sets is not an easy task. In this section we present an algorithm to construct minimal sets based on FSM partitions. Note that the proposed code construction technique we present is a general algorithm that can be used to find the minimal set for a large variety of constraints, and hence can be applied in many transmission and data storage systems where constrained coding is essential. We also note that the appropriate number of FSM partitions to use is an engineering practice, and should be chosen to best fit the scenario under consideration. The algorithm starts with definition of word $\varnothing$, which indicates that it is possible to merely stay in a state rather than exit from it.

Definition 2.1 For a specified state, the word $\varnothing$ in the corresponding minimal set is a void codeword. Coded sequence $a\varnothing b$ is indeed ab where a and b denote any possible constraint-satisfying sequences.

In an extension of a minimal set, the concatenation of $\varnothing$ operates as follows. Appending any word w in the minimal set to the end of $\varnothing$ results in w. $\varnothing$ is not allowed to be appended to the end of any word. For a minimal set M containing word $\varnothing$, $M_f^{(l_{th})}$ denotes the full extension of M containing all possible words with length no greater than l_{th}. Detailed discussion of these definitions is given later in this section.

An *FSM partition* is obtained by partitioning the FSM into several parts. For example, the FSM in Fig. 2.6 is partitioned into two FSM partitions F_1 and F_2 in Fig. 2.12, where F_1 has one state and F_2 has two states. If we choose state 3 as the specified state in Fig. 2.12, we begin with the partition farthest

from this state and move toward state 3 in order to recover the minimal set of the original FSM. Starting from state 1 and excluding any transitions across the partition line separating F_1 and F_2, the minimal set of state 1 in FSM partition F_1 is $M_{1|p_{12}} = \{\varnothing, 1\}$. Explanation of $M_{1|p_{12}}$ is as follows.

It is possible to emanate from state 1 and re-enter state 1. In this way we obtain the codeword 1. It is also possible to just stay in state 1 rather than emanate from it. This generates the void codeword $\varnothing$ which is necessary when taking into account the next FSM partition. As defined above in Section 1, a full extension of $M_{1|p_{12}}$ with $l_{th} = 4$ is $M_{1|p_{12},f}^{(l_{th}=4)} = \{\varnothing, 1, 11, 111, 1111\}$ which is obtained by following the rules of extensions with $\varnothing$, as outlined above.

Now consider state 3 in the FSM partition F_2. The minimal set of state 3 is established as $M_2 = \{0, 1M_{1|p_{12},f}00\}$, which is explained as follows. The stand-alone 0 represents exiting and reentering state 3 on the rightmost loop. Bit 1 corresponds to the edge emitting from F_2 (and entering F_1). The subsequent 00 corresponds to the edge entering F_2 (emitting from F_1). Since $M_{1|p_{12},f}$ contains all possible words in F_1 including codeword $\varnothing$, M_2 is comprised of all words whose concatenation generates all constraint-satisfying sequences. Therefore, $M_2 = \{0, 1M_{1|p_{12},f}00\}$ is the complete set of words in the minimal set with state 3 selected as the specified state. For example, with $l_{th} = 4$, $M_{1|p_{12},f}^{(l_{th}=4)}$ is $\{\varnothing, 1, 11, 111, 1111\}$ and hence $M_2 = \{0, 1M_{1|p_{12},f}^{(l_{th}=4)}00\}$ is $\{0, 100, 1100, 11100, 111100, 1111100\}$. As $l_{th} \to \infty$, $M_2 = \{0, 1M_{1|p_{12},f}^{(l_{th})}00\}$ then becomes $\{0, 1M_{1|p_{12},f}00\}$, which is a minimal set comprising an infinite number of words. Note that since F_2 is the last FSM partition, although $\varnothing$ could be included in M_2, it would serve no purpose since F_2 is not included in other FSM partitions. Therefore, $\varnothing$ is not included as a word in the minimal set of the final partition in a sequence of partitions.

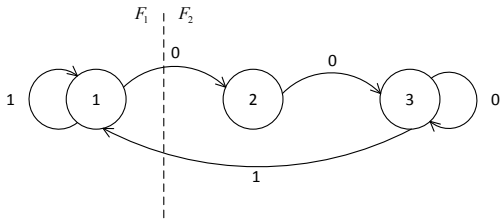

Figure 2.12: FSM partitions of a constraint that forbids 101 pattern

An algorithm to construct a minimal set using this approach is as follows.

- Choose a specified state which results in the highest maximum possible code rate.
- Partition the FSM into n FSM partitions $F_i, i = 1, 2, 3...n$ where the full extension of minimal set of F_k should be fully included in the minimal set of F_j, for $\forall 1 \leq k < j \leq n$.
- Denote $M_{i|p_{ij}}$ as the minimal set of FSM partition F_i excluding any transitions across the partition line separating partition F_i and partition F_j. $M_{i|p_{ij},f}$ is the full extension of $M_{i|p_{ij}}$. Starting from $M_{1|p_{12}}$, generate the minimal set $M_{i|p_{i(i+1)}}$ of F_i recursively until reaching minimal set M_n of the last FSM partition F_n. Note that, to limit the size of the resulting minimal set, it might be necessary to prune words to limit the maximum length of words l_{th} in minimal sets of $F_i, i = 1, 2, 3...n-1$, in order to limit the maximum length of words l_{max} in the minimal set M_n we eventually establish. In practice, $M^{(l_{th})}_{i|p_{i(i+1)},f}$ is utilized instead of $M_{i|p_{i(i+1)},f}$ which may contain an infinite number of words and is not practical. Appropriate pruning techniques are considered below.

We now provide complexity analysis for this recursive construction algorithm. The enumeration of words depends on the specific constraint, the number of FSM partitions K, the number and lengths of words in each $M_{i|p_{ij}}, i = 1, 2, 3...K-1$, and the limit on the maximum length of words l_{th}. For simplicity, in this complexity analysis we assume that the number of complete extensions of $M_{i|p_{ij}}$ performed during generation of $M^{(l_{th})}_{i|p_{i(i+1)},f}$ is

$c_i, i = 1, 2, 3...K-1$, i.e., $\left| M^{(l_{th})}_{1|p_{12,f}} \right| = \Theta(\left| M_{1|p_{12}} \right|^{c_1+1})$ where $\left| M_{1|p_{12}} \right|$ denotes the number of words in $M_{1|p_{12}}$. Thus the number of words in $M^{(l_{th})}_{1|p_{12},f}$ is $O(\left| M_{1|p_{12}} \right|^{c_1})$ as c_1 becomes large. Similarly, the number of words in $M^{(l_{th})}_{2|p_{23},f}$ is $O(\left| M_{2|p_{23}} \right|^{c_2}) = O(\{\left| M_{1|p_{12}} \right|^{c_1} + \epsilon\}^{c_2})$, where ϵ is the number of words in $M_{2|p_{23}}$ except those in $M^{(l_{th})}_{1|p_{12},f}$, and hence ϵ is the higher-order infinitesimal of $O(\left| M_{1|p_{12}} \right|^{c_1})$. Therefore, we have $\left| M^{(l_{th})}_{2|p_{23},f} \right| = O(\left| M_{1|p_{12}} \right|^{c_1 c_2})$. Recursively repeating this procedure, we find that the complexity of enumerating words in F_K is $O(\left| M_{1|p_{12}} \right|^{\prod_{i=1}^{K-1} c_i})$.

The proposed algorithm involves performing extensions until reaching the maximum allowed word length l_{max} in the minimal sets. In practice, one could set l_{th} as an appropriate value. With larger l_{th}, more constraint-satisfying words exist in the minimal set and the code will have increased efficiency, at the cost of increased storage and computational resources.

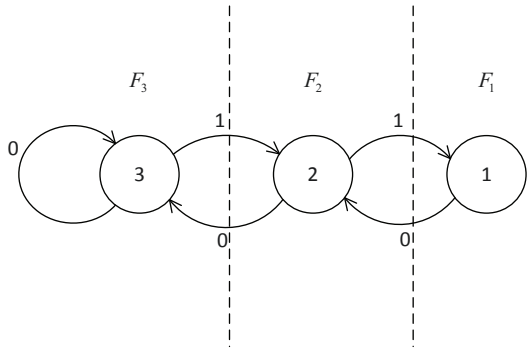

Figure 2.13: FSM partitions in constrained sequence codes that forbids 111 and 11011 patterns

Example (constrained codes that forbid the 111 and 11011 patterns): the FSM in Fig. 2.10 is partitioned into three FSM partitions F_1, F_2, F_3, as shown in Fig. 2.13. According to Criterion 2.3, state 3 results in the highest achievable code rate and is selected as the specified state. To construct its minimal set, we begin with the farthermost partition and recursively move toward the specified state. Therefore we begin by constructing the minimal set for state 1. Starting from F_1, the minimal set $M_{1|p_{12}}$ of F_1 excluding

transitions across the partitioning line between F_1 and F_2 is obtained as $M_{1|p_{12}} = \{\varnothing\}$. Recursively, the minimal set $M_{2|p_{23}}$ of F_2 is obtained as $M_{2|p_{23}} = \{\varnothing, 1M_{1|p_{12},f}^{(l_{th})}0\}$, and finally the minimal set of state 3 is obtained as $M_3 = \{0, 1M_{2|p_{23},f}^{(l_{th})}0\}$. If we set l_{max} in M_3 to be 6, it is straightforward to obtain $M_{1|p_{12},f} = \{\varnothing\}$, $M_{2|p_{23}} = \{\varnothing, 10\}$ and $M_{2|p_{23},f}^{(l_{th}=4)} = \{\varnothing, 10, 1010\}$. Hence the final minimal set $M_3 = \{0, 10, 1100, 110100\}$. Based on M_3, we can construct codebooks for this constraint. In Table 2.2 we choose to present a simple code that achieves 96.65% of capacity.

Table 2.2: A codebook of the constraint that forbids the 111 and 11011 patterns with $\eta = 96.65\%$

Source words	Codewords
00	10
01	00
10	010
110	1100
1110	01100
11110	110100
11111	0110100

We have now presented the proposed construction algorithm with simple examples. Apart from its successful application in reducing inter-cell interference in flash memories as discussed in [20], and in constructing codes for the Pearson constraint as discussed in [21], in the subsequent sections we apply this technique in two emerging applications, VLC and DNA-based storage systems.

2.1.3 Example: codes for visible light communications

DC-free codes for VLC

VLC has gained much attention recently [3, 32, 33, 34]. The simplest VLC relies on on-off keying (OOK) modulation, which is realized with DC-free codes to generate a constant dimming level of 50% and reduce flicker

perception. Three types of DC-free codes, the Manchester code, a 4B6B code and an 8B10B code, have been used in VLC standards for flicker mitigation and dimming control [3].

Manchester encoding represents a logic zero as an OOK symbol 01 and a logic one as an OOK symbol 10, hence the code rate is 0.5. Note that Manchester codes satisfy the DC-free constraint with $N = 3$ and capacity 0.5, and therefore have a code rate equal to capacity. The 4B6B code satisfies the DC-free constraint with $N = 5$, which has a capacity of 0.7925 [1]. The codebook has 16 source words with a code rate R of 2/3, and therefore has an efficiency of $\eta = R/C = 84.12\%$. 8B10B codes are a class of rate $R = 8/10$ codes that satisfy DC-free constraints with $N = 6$ or $N = 7$. Many 8B10B codes have been constructed with different parameters; a survey of 8B10B codes can be found in [12]. DC-constrained systems with $N = 6$ and $N = 7$ have capacities 0.8495 and 0.8858 respectively. The efficiency of an 8B10B code is then $\eta = 94.17\%$ for the $N = 6$ constraint and $\eta = 90.31\%$ for the $N = 7$ constraint.

In the following, we first present an algorithm that constructs minimal sets specifically for DC-free constraints with an arbitrary RDS value. Then we consider the DC-free $N = 5$ and $N = 7$ constraints that the 4B6B code and 8B10B code satisfy, respectively. We construct simple variable-length codes based on the proposed construction process, and demonstrate that they have higher efficiency than the fixed-length 4B6B and 8B10B codes.

Algorithm for constructing the minimal set

In this section, we specifically illustrate how our algorithm for construction of minimal sets can be applied to construct high-efficiency variable-length DC-free codes with any given value of N. With the aforementioned criteria and procedures, we base our designs on the FSM in Fig. 2.1 and choose state $S_K, K = \lfloor \frac{N+1}{2} \rfloor$ as the specified state.

We partition the FSM into N partial FSMs $F_{S_1}, F_{S_2}, ..., F_{S_N}$, each having one state, as shown in Fig. 2.14. Algorithm 1 shows the procedure to construct

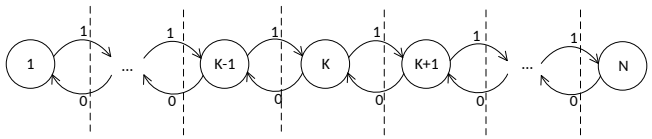

Figure 2.14: FSM partition of general DC-free codes

the minimal set M_{S_K} of state S_K. States S_1 and S_N do not have edges that exit and re-enter them without crossing partition boundaries, so the minimal sets of FSM partitions F_{s_1} and F_{s_N} are initialized as $M_{S_1|p_{S_1S_2}} = M_{N|p_{S_NS_{(N-1)}}} = \{\varnothing\}$.

Algorithm 1 Constructing the minimal set of general DC-free codes

---

Initialize:

1: Choose state S_K as the specified state, $K = \lfloor \frac{N+1}{2} \rfloor$

2: Set the minimal set of FSM partitions F_{s_1} and F_{s_N} as $M_{S_1|p_{S_1S_2}} = M_{S_N|p_{S_NS_{(N-1)}}} = \{\varnothing\}$.

Start:

3: **for** $i = 2, 3, ..., K-1,\ \ j = N-1, N-2, ..., K+1$ **do**

4: Construct the minimal set of FSM partitions F_{S_i} and F_{S_j} as $M_{S_i|p_{S_iS_{(i+1)}}} = \{\varnothing, 0(M^{(l_{th}^{(i)})}_{S_{(i-1)}|p_{S_{(i-1)}S_i},f})1\}$ and $M_{S_j|p_{S_{(j)}S_{(j-1)}}} = \{\varnothing, 1(M^{(l_{th}^{(i)})}_{S_{(j+1)}|p_{S_{(j+1)}S_j},f})0\}$, respectively.

5: **end for**

6: **Output:** The minimal set of state S_K is $M_{S_K} = \{0(M^{(l_{th}^{(i)})}_{S_{(K-1)}|p_{S_{(K-1)}S_K},f})1, 1(M^{(l_{th}^{(i)})}_{S_{(K+1)}|p_{S_{(K+1)}S_K},f})0\}$

Pruning

As discussed above, l_{th} can be set to a predetermined value in order to limit the size of the minimal set. We now discuss the appropriate value of l_{th} for DC-free codes when pruning is utilized for the sake of limiting storage and computational resources.

In order to limit the maximum wordlength in the minimal set to l_{max}, it is straightforward to verify that, in each full extension $M^{(l_{th}^{(i)})}_{S_{(i)}|p_{S_{(i)}S_{i+1}},f}$, $i = 2, 3, ..., (K-1)$ and $M^{(l_{th}^{(j)})}_{S_{(j)}|p_{S_{(j)}S_{j-1}},f}$, $j = N, (N-1), ..., (K+1)$, it is sufficient

to dynamically set l_{th} as

$$l_{th}^{(i)} = l_{\max} - 2(K-i), \quad i = 2, 3, ..., K-1 \tag{2.15}$$

$$l_{th}^{(j)} = l_{\max} - 2(j-K), \quad j = N, N-1, ..., K+1. \tag{2.16}$$

Code construction

We first consider a special case of construction of a DC-free code with $N = 3$ using our approach, and compare the result with the Manchester code. With a DC-free FSM of $N = 3$, it is readily seen that state 2 should be selected as the specified state, according to Criterion 2.1, from which it is straightforward to obtain the complete minimal set $\{01, 10\}$. Our proposed method gives a mapping of logic one to 10 and logic zero as 01, which results in the fixed-length Manchester code with $\eta = 100\%$.

We now use our approach to construct variable-length DC-free codes with $N = 5$ for comparison with the 4B6B code. Based on the criteria outlined above, we select state 3 as the specified state to construct a variable-length code; for purpose of comparison we have also constructed codes based on states 1&5 and 2&4. In Table 2.3 we present the number of words in the minimal set $|L_M|$, $\widetilde{C}_M$, $\widetilde{\eta}$, the number of codewords in the codebook N^c, the maximum length of codewords in the codebook c_{max}, $\bar{R}$, and η for the best code constructed with $l_{max} = 6$. We present results in Table 2.4 when $l_{max} = 8$. It is evident from these tables that with state 3 as the specified state, $\widetilde{C}_M$ and $\bar{R}$ are better than for codes constructed based on other states, which is in agreement with the results of Fig. 2.8. With $l_{max} = 6, c_{max} = 8$ and $N^c = 16$, we constructed a codebook with $\bar{R} = 0.7703$ and $\eta = 97.19\%$ which is shown in Table 2.5. Comparing the code with the 4B6B code with $R = 2/3$ and $\eta = 84.12\%$, we note that our code contains the same number of words but is 13% more efficient and is only 2.81% from capacity. With $l_{max} = 8, c_{max} = 10$ and $N^c = 22$, a codebook with $\eta = 98.66\%$ can be constructed. Note that codebooks with still higher efficiency can be constructed with larger l_{max}, c_{max} and N^c.

We also constructed DC-free codes with $N = 7$, which has capacity 0.8858

Table 2.3: Parameters of $N = 5$ DC-free codes with $l_{max} = 6$

State	$\|L_M\|$	$\widetilde{C}_M$	$\widetilde{\eta}$	N^c	c_{max}	$\bar{R}$	η
1&5	4	0.5000	63.09%	4	6	0.5000	63.09%
2&4	5	0.7054	89.01%	17	10	0.7012	88.48%
3	6	0.7729	97.53%	16	8	0.7703	97.19%

Table 2.4: Parameters of $N = 5$ DC-free codes with $l_{max} = 8$

State	$\|L_M\|$	$\widetilde{C}_M$	$\widetilde{\eta}$	N^c	c_{max}	$\bar{R}$	η
1&5	9	0.5892	74.35%	25	14	0.5827	73.53%
2&4	9	0.7418	93.60%	25	10	0.7381	93.13%
3	8	0.7863	99.22%	22	10	0.7819	98.66%

Table 2.5: A DC-free code with $N = 5$, $\bar{R} = 0.7703, \eta = 97.19\%$

Source word	Codeword	Source word	Codeword
000	0011	11010	010011
001	1100	11011	011100
010	0101	11100	100011
011	0110	11101	101100
100	1001	111100	01001011
101	1010	111101	01110100
11000	001011	111110	10001011
11001	110100	111111	10110100

Table 2.6: Maximum possible code rates of minimal sets with different l_{max} for a DC-free code with $N = 7$

l_{max}	$\|L_M\|$	$\widetilde{C}_M$	$\widetilde{\eta}$
6	8	0.8101	91.45%
8	16	0.8471	95.63%
10	32	0.8650	97.65%
12	64	0.8743	98.70%
14	128	0.8793	99.27%

Table 2.7: Codes constructed for a DC-free code with $N = 7$

l_{max}	N^c	c_{max}	$\bar{R}$	η
6	15	8	0.8034	90.70%
8	61	12	0.8416	95.01%
10	94	14	0.8574	96.80%
12	127	16	0.8661	97.78%

Table 2.8: Words in the minimal set of a DC-free code with $N = 7$, $l_{max} = 10$, $\widetilde{C}_M = 97.65\%$

Word length	Word	Word length	Word
2	10	10	1110101000
2	01	10	0001010111
4	1100	10	1101101000
4	0011	10	0010010111
6	111000	10	1110011000
6	000111	10	0001100111
6	110100	10	1110100100
6	001011	10	0001011011
8	11101000	10	1101010100
8	00010111	10	0010101011
8	11011000	10	1101011000
8	00100111	10	0010100111
8	11100100	10	1101100100
8	00011011	10	0010011011
8	11010100	10	1110010100
8	00101011	10	0001101011

[12], for comparison with the 8B10B code. We use state 4 to find the minimal set. Table 2.6 summarizes the maximum possible code rates $\widetilde{C}_M$ of minimal sets with different l_{max}, and Table 2.7 summarizes parameters of practical codes we have constructed based on those minimal sets. We also include $|L_M|$, N^c and c_{max} in the tables. As is shown in Table 2.6, with $l_{max} = 10$, a minimal set with $\widetilde{\eta} = 97.65\%$ can be constructed; this minimal set is presented in Table 2.8. Using this minimal set we constructed a code with $N^c = 94, L_m = 14, \bar{R} = 0.8574$ and $\eta = 96.80\%$. Note that this code contains fewer codewords and has a higher code rate than the 8B10B code with $R = 0.8$. Codebooks with still higher efficiency can be constructed with larger l_{max}, c_{max} and N^c. We also note that codewords in our codes satisfy the prefix condition [24] and therefore can be instantaneously decoded given knowledge of the codebook.

2.1.4 Example: codes for DNA-based storage

The first large scale DNA-based storage was implemented in 2012 and has attracted considerable interest [35]. In this scheme, binary source bits are translated into a strand of nucleotides adenine (A), thymine (T), guanine (G) and cytosine (C), which can be represented by a 4-ary alphabet set $\{0, 1, 2, 3\}$. Long repetitions (e.g., more than 4) of the same nucleotide may significantly increase sequencing errors in DNA-based storage, and hence should be avoided. A recent work [6] proposed using fixed-length 4-ary k-constrained codes with modulo 4 precoding to limit the maximum runlength of nucleotides to be $k + 1$. We now apply our proposed code construction technique to construct variable-length 4-ary k-constrained codes that achieve higher efficiency and significantly lower implementation complexity.

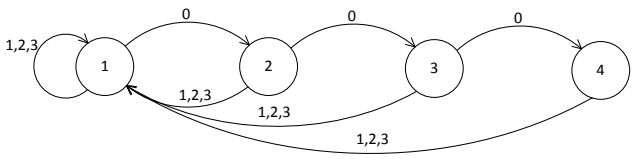

Figure 2.15: FSM of 4-ary k-constrained codes for DNA-based storage

The FSM of 4-ary k-constrained codes with $k = 3$ is shown in Fig. 2.15. According to Criterion 2.1, we select state 1 as the specified state. The minimal set is established as

$$M = \{1, 2, 3, 01, 02, 03, 001, 002, 003, 0001, 0002, 0003\}.$$

Following NGH coding, we construct a codebook with $\bar{R} = 0.9971$ quaternary bits per coded symbol and $\eta = 99.92\%$ as shown in Table 2.9, where $N^c = 12$. To compare, the highest rate of the fixed-length codes proposed in [6] (with method B) is $147/148 = 0.9932$, and $N^c = 4^{147}$. Comparison to [6] with other k values is listed in Table 2.10, where we present the highest code rate R and N^c of the fixed-length codes in [6] and our results. It is clear that our proposed variable-length coding technique achieves higher code rates with significantly smaller codebooks.

Table 2.9: A 4-ary $k = 3$ RLL codebook, $\bar{R} = 0.9971$

Source word	Codeword	Source word	Codeword
10	1	111110	001
01	2	111101	002
00	3	111100	003
1110	01	1111110	0001
1101	02	11111110	0002
1100	03	11111111	0003

Table 2.10: Comparison of highest code rates and sizes of codebooks with [6]

k	R [6]	N^c [6]	$\bar{R}$	N^c
1	0.9091	4^{10}	0.95	6
2	0.9744	4^{38}	0.9881	9
3	0.9932	4^{147}	0.9971	12
4	0.9983	4^{580}	0.9993	15

2.2 Construction of capacity-approaching codes with multiple encoding states and state-independent decoding [22]

In the previous section we demonstrated the construction of constrained sequence codes with a single encoding state. Although codebooks with high η can be constructed with the single-state variable-length construction technique, two potential drawbacks of this approach should be considered. First, the codewords can be long, especially when long words exist in the minimal set, which increases the complexity of the encoding and decoding circuits. This occurs, for instance, with a large value of k in RLL constraints and a large value of N in DC-free constraints. Second, for some types of constraints, a minimal set consisting of a finite number of words does not capture all the constraint-satisfying sequences because of loops that exist in the FSMs. This results in a loss in the achievable code rate, as discussed in the previous section and in [18] [19]. Typical examples are DC-free constraints with $N \geq 4$ and most DC-free RLL constraints. To overcome these drawbacks, in this section we extend the single-state encoding technique by proposing a construction technique for variable-length constrained sequence codes that involves multiple states in the codebook.

2.2.1 Multi-state encoding based on an FSM

In this section we discuss the encoding technique with multiple encoding states based on an FSM. We start with the selection of principal states.

Selection of principal states

Similar to the single-state technique, the first step of our proposed multi-state technique is to determine which multiple states of the FSM that describes the constraint should be considered when generating the words in the minimal set. We call these states the principal states. Denote the j-th principal state as $\sigma_j, \sigma_j \in \Psi = \{\sigma_1, \sigma_2, ..., \sigma_{|\Psi|}\}$, where Ψ is the set of principal states and $|\Psi|$

is the size of Ψ. We first define how concatenation of words in the minimal set is performed.

Definition 2.2 (concatenation of words) Denote

$$W(\sigma_j) = \{w(\sigma_j)_1, w(\sigma_j)_2, ..., w(\sigma_j)_{|W(\sigma_j)|}\}$$

as the set of words generated by the j-th principal state. Given $W(\sigma_j)$, let the set of next states corresponding to words in $W(\sigma_j)$ be

$$H(\sigma_j) = \{h(\sigma_j)_1, h(\sigma_j)_2, ..., h(\sigma_j)_{|W(\sigma_j)|}\}$$

where $h(\sigma_j)_i \in \Psi, 1 \leq i \leq |H(\sigma_j)|$. The words in the minimal set are $W(\Psi) = \{W(\sigma_1), W(\sigma_2), ..., W(\sigma_{|\Psi|})\}$. When considering concatenation of words in $W(\Psi)$, $w(\sigma_j)_i$ is only allowed to be concatenated with words in the set $W(h(\sigma_j)_i)$.

Based on this definition of concatenation, we introduce the definition of principal states and the criterion to select them.

Definition 2.3 (principal states) Ψ is determined such that all constraint-satisfying sequences can be generated through the concatenation of words in $W(\Psi)$. In addition, to ensure that it is possible to instantaneously decode the received sequence, no word in $W(\sigma_i)$ is the prefix of a word in $W(\sigma_j)$ $\forall \sigma_i, \sigma_j \in \Psi$.

From Definition 2.3 it follows immediately that if a state σ_j has a loop associated with itself in the FSM, then $\sigma_j \in \Psi$ otherwise not all constraint-satisfying sequences can be generated with finite-length codewords due to the loop at σ_j. The principal states should also be selected such that $|W(\sigma_i)| = |W(\sigma_j)|$ $\forall \sigma_i, \sigma_j \in \Psi$, in order that each state will have a codeword associated with each source word in the codebook. Examples of the appropriate selection of principal states follow discussion of establishing the minimal set.

Minimal set

After determining the principal states, we establish the minimal set of the constraint based on its underlying FSM. Given $|\Psi|$ principal states, the minimal set in multi-state encoding is a tabular representation that contains

$2\,|\Psi|$ columns, where $|\Psi|$ of the columns indicate the words generated by the principal states, i.e. $W(\Psi) = \{W(\sigma_1), W(\sigma_2), ..., W(\sigma_{|\Psi|})\}$, and the other $|\Psi|$ columns indicate the next states corresponding to each word, i.e. $H(\Psi) = \{H(\sigma_1), H(\sigma_2), ..., H(\sigma_{|\Psi|})\}$.

Assignment of words in a minimal set with multiple states will, in general, result in the necessity for state-dependent decoding, which requires knowledge of both the received codeword and the corresponding encoding state in order to correctly determine the corresponding source word. Decoding that can be performed with knowledge of only the received codeword and without tracking the encoder state is called state-independent decoding. To enable state-independent decoding, the following necessary and sufficient condition [36] must be satisfied.

Condition 2.1 (state-independent decoding): When assigning words from $W(\Psi)$ in the minimal set, the necessary and sufficient condition for state-independent decoding is that for $1 \leq u \leq v \leq y \leq |\Psi|$ and for each

$$W_r \in W(\sigma_u) \cap W(\sigma_y)$$

such that

$$W_r \notin W(\sigma_v)$$

there exists a $W_q \in W(\sigma_v)$ such that there exists no $\sigma_l, 1 \leq l \leq |\Psi|$, for which $W_r, W_q \in W(\sigma_l)$.

As will become evident in the examples below, this condition implies that in the minimal set table, a word does not appear in more than one row. Therefore, enabling state-independent decoding requires satisfying Condition 2.1, which implies careful design of the encoder. It should be mentioned, however, that it may not be possible for $|W(\sigma_i)|$ and $|W(\sigma_j)|$ to be equal $\forall i, j$. In such cases we can extend some of the words in $W(\Psi)$ using $H(\Psi)$ with the goal of generating an *extended minimal set* with $|W(\sigma_i)| = |W(\sigma_j)|$ for all i, j. We note that, without adequate care, this concatenation of words in $W(\Psi)$

may result in a situation where one word becomes a prefix of another, meaning that the codewords are not prefix-free and that the decoder would not be able to instantaneously decode the received sequence. In this section we focus on situations where it is possible to have $|W(\sigma_i)| = |W(\sigma_j)|$ without causing the prefix problem, and in the next section we extend the construction technique to consider situations where the prefix problem arises.

Example ($(d = 1, k = 3)$ code): Consider the FSM for the $(d = 1, k = 3)$ RLL constraint shown in Fig. 2.16. We choose states 1 and 3 as the principal states, which we denote σ_1 and σ_2, respectively. With these states, it follows that $W(\sigma_1) = \{01, 00\}, H(\sigma_1) = \{\sigma_1, \sigma_2\}$ and $W(\sigma_2) = \{01, 1\}, H(\sigma_2) = \{\sigma_1, \sigma_1\}$. We note that this selection of states and codewords satisfies Condition 2.1 and the prefix condition, therefore instantaneous state-independent decoding is viable. The minimal set is given in tabular form in Table 2.11.

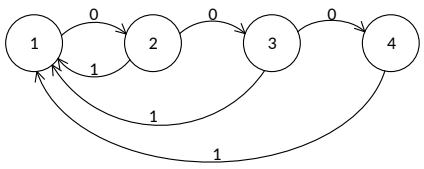

Figure 2.16: FSM of a $(d = 1, k = 3)$ RLL code

Table 2.11: The minimal set of a two-state $(d = 1, k = 3)$ code

$W(\sigma_1)$	$H(\sigma_1)$	$W(\sigma_2)$	$H(\sigma_2)$
01	σ_1	01	σ_1
00	σ_2	1	σ_1

Example (DC-free code with $N = 5$): The FSM of a DC-free sequence with $N = 5$ is shown in Fig. 2.17. Similar to the previous example, we follow the steps of the construction technique to select states 2 and 4 as the principal states, i.e. $\sigma_1 =$ state 2 and $\sigma_2 =$ state 4. The minimal set of this DC-free code with $N = 5$ is shown in Table 2.12. As in the example above, this minimal set enables the construction of codes with instantaneous state-independent decoders.

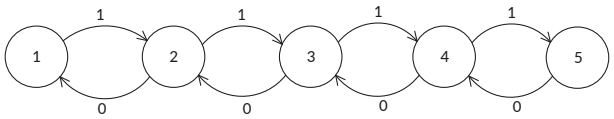

Figure 2.17: FSM of a DC-free code with $N = 5$.

Table 2.12: The minimal set of a multi-state DC-free code with $N = 5$

$W(\sigma_1)$	$H(\sigma_1)$	$W(\sigma_2)$	$H(\sigma_2)$
11	σ_2	00	σ_1
10	σ_1	10	σ_2
01	σ_1	01	σ_2

Example (DC-free RLL codes with $(d = 1, k = 3, N = 5)$): We also employ the proposed multi-state encoding technique to construct codes that satisfy both DC-free and RLL constraints. We describe the code construction process with an example of a code that limits to $N = 5$ the RDS of the sequence that arises after NRZI encoding a sequence that satisfies the $(d = 1, k = 3)$ constraint. We will later present results of other codes with different d, k and N values. The FSM of the $(d = 1, k = 3, N = 5)$ DC-free RLL constraint (after NRZI coding) is shown in Fig. 2.18, where the coded sequence has runlengths between $d + 1 = 2$ and $k + 1 = 4$, and the RDS is limited to five different values.

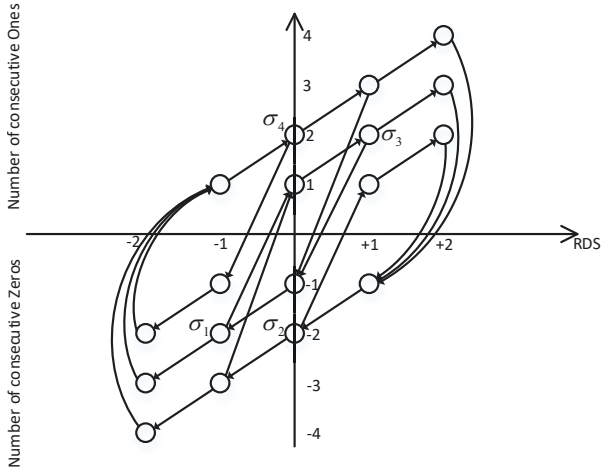

Figure 2.18: FSM of the DC-free RLL constraint corresponding to an NRZI encoded $(d = 1, k = 3)$ code with $N = 5$.

We choose four states as the principal states, and refer to them by their locations in the x-y coordinates in Fig. 2.18, i.e. $\sigma_1 = (-1,-2), \sigma_2 = (0,-2), \sigma_3 = (1,2), \sigma_4 = (0,2)$. With this selection of principal states, all loops in the FSM contain at least one principal state, and therefore the minimal set will contain a finite number of words. This ensures that all constraint-satisfying sequences can be generated with concatenation of words in the minimal set. We establish the minimal set shown in Table 2.13. Note that $|W(\sigma_1)| = |W(\sigma_3)| = 2$, and $|W(\sigma_2)| = |W(\sigma_4)| = 3$. Therefore, we extend some of the words in $W(\sigma_1)$ and $W(\sigma_3)$ by referring to $H(\sigma_1)$ and $H(\sigma_3)$, in order to have the same number of rows in all columns of the minimal set. After extension of the word 11 in $W(\sigma_1)$ and the word 00 in $W(\sigma_3)$, we obtain the extended minimal set as shown in Table 2.14 where $|W(\sigma_i)| = 3\ \forall i$.

Table 2.13: The minimal set of a DC-free RLL code corresponding to an NRZI encoded $(d = 1, k = 3)$ code with $N = 5$

$W(\sigma_1)$	$H(\sigma_1)$	$W(\sigma_2)$	$H(\sigma_2)$	$W(\sigma_3)$	$H(\sigma_3)$	$W(\sigma_4)$	$H(\sigma_4)$
011	σ_4	011	σ_3	100	σ_2	100	σ_1
11	σ_3	1100	σ_2	00	σ_1	1100	σ_2
		0011	σ_4			0011	σ_4

Table 2.14: The extended minimal set of a DC-free RLL code corresponding to an NRZI encoded $(d = 1, k = 3)$ code with $N = 5$

$W(\sigma_1)$	$H(\sigma_1)$	$W(\sigma_2)$	$H(\sigma_2)$	$W(\sigma_3)$	$H(\sigma_3)$	$W(\sigma_4)$	$H(\sigma_4)$
011	σ_4	011	σ_3	100	σ_2	100	σ_1
1100	σ_1	1100	σ_2	00011	σ_4	1100	σ_2
11100	σ_2	0011	σ_4	0011	σ_3	0011	σ_4

Note that if we use the single-state encoding technique, minimal sets for DC-free constraints with $N \geq 4$ and for most DC-free RLL constraints consist of an infinite number of words, so to be practical, these sets must be truncated to result in sets with a finite number of words. Since valid words are removed from the minimal set, the achievable code rate is reduced, as noted in the previous section. However, with the multi-state encoding technique described above, all constraint-satisfying sequences can be generated with the words in the minimal set, hence full capacity can potentially be approached.

Partial extensions

After obtaining a minimal set or an extended minimal set, we may perform partial extensions to obtain sets of codewords. However, as opposed to the single-state encoding technique where words in a minimal set can be freely concatenated, concatenation as defined in Definition 2.2 must be performed with multi-state encoding. Therefore, in addition to the words in $W(\Psi)$, we must have knowledge of $H(\sigma_j)_i$ to determine how to extend the word $w(\sigma_j)_i$, $1 \leq j \leq |\Psi|$, $1 \leq i \leq |W(\sigma_j)|$. A partial extension in multi-state encoding is the simultaneous extension of words $w(\sigma_j)_i$ $\forall j$ for a fixed i, where extension is according to the concatenation of words in Definition 2.2. Similar to single-state encoding, a partial extension can be applied to the result of any previous partial extension where partial extensions start from the minimal set.

We denote the set of codewords generated through extension of $W(\sigma_j)$ as $\alpha(\sigma_j)$ and the corresponding set of next states as $\beta(\sigma_j)$[1]. The size of each set is denoted as ξ. Note that when a word $w(\sigma_j)_i$ is extended in the table, all the other words in the i-th row that are generated from other principal states are simultaneously extended to ensure $|\alpha(\sigma_1)| = |\alpha(\sigma_2)| = ... = \left|\alpha(\sigma_{|\Psi|})\right| = \xi$.

Example ($(d = 1, k = 3, N = 5)$ DC-free RLL code) We perform partial extensions of Table 2.14 to obtain a set of codewords for the $(d = 1, k = 3, N = 5)$ DC-free RLL constraint. We extend the words $W(\sigma_j)_1, \forall j$ and then extend the words $W(\sigma_j)_2, \forall j$ based on the previous partial extension, by following Definition 2.2. Our set of codewords is shown in Table 2.15, where $\xi = 7$.

NGH coding and code rate evaluation

The last step of the encoding technique is to perform NGH coding over the codebook to assign source words to codewords $\alpha(\sigma_1), \alpha(\sigma_2), ..., \alpha(\sigma_{|\Psi|})$ in the codebook. To approach capacity, we attempt to approximate the maxentropic probabilities of codewords in the codebook as closely as possible.

We first obtain the maxentropic transition probabilities of the constraint

[1]When we use the words in the minimal set as the set of codewords directly, for consistency, we still denote $W(\sigma_j)$ as $\alpha(\sigma_j)$ and $H(\sigma_j)$ as $\beta(\sigma_j)$.

Table 2.15: Partial extension of the extended minimal set of a DC-free RLL code corresponding to an NRZI encoded $(d = 1, k = 3)$ code with $N = 5$

$\alpha(\sigma_1)$	$\beta(\sigma_1)$	$\alpha(\sigma_2)$	$\beta(\sigma_2)$	$\alpha(\sigma_3)$	$\beta(\sigma_3)$	$\alpha(\sigma_4)$	$\beta(\sigma_4)$
011100	σ_1	011100	σ_2	100011	σ_3	100011	σ_4
0111100	σ_2	01100011	σ_4	1001100	σ_2	1001100	σ_1
0110011	σ_4	0110011	σ_3	1000011	σ_4	10011100	σ_2
1100011	σ_4	1100011	σ_3	00011100	σ_1	1100011	σ_3
11001100	σ_1	11001100	σ_2	000111100	σ_2	11001100	σ_2
110011100	σ_2	11000011	σ_4	000110011	σ_4	11000011	σ_4
11100	σ_2	0011	σ_4	0011	σ_3	0011	σ_4

based on its FSM representation, which is well studied in [12]. Based on the maxentropic transition probabilities, we evaluate the maxentropic probability of each codeword in the final codebook and the steady-state distribution of Ψ. Denote the maxentropic probability of the i-th codeword in α_j as $p(\alpha_j)_i$, the steady-state distribution as $\boldsymbol{\pi} = [\pi(\sigma_1), \pi(\sigma_2), ..., \pi(\sigma_{|\Psi|})]$, and the vector of input probabilities to NGH coding as $\boldsymbol{p}_{NGH} = [p_1, p_2, ..., p_\xi]$. The desired probability of the i-th source word is then

$$p_i = \sum_{j=1}^{|\Psi|} \pi(\sigma_j) \times p(\alpha_j)_i, \ 1 \leq i \leq \xi. \tag{2.17}$$

With this vector of desired input probabilities, NGH coding is performed to generate the corresponding source words.

After constructing the codebook, we must evaluate the average code rate. We assume independent and equiprobable input bits, and denote codeword $\alpha(\sigma_j)_i$ as the codeword assigned to a source word of length l_i. The probability of occurrence of that codeword when the encoder is in state j is $p(\alpha(\sigma_j)_i) = 2^{-l_i}$. Note that since these probabilities are not in general equal to the maxentropic probabilities, the steady-state probabilities of the principal states are not exactly the probabilities in the steady-state distribution of the FSM. Based on the probability of occurrence of each codeword in the codebook, it is possible to evaluate the steady-state distribution $\widetilde{\boldsymbol{\pi}} = [\widetilde{\pi}(\sigma_1), \widetilde{\pi}(\sigma_2), ..., \widetilde{\pi}(\sigma_{|\Psi|})]$ of all the principal states Ψ by solving:

$$\widetilde{\boldsymbol{\pi}}\boldsymbol{P} = \widetilde{\boldsymbol{\pi}} \tag{2.18}$$

where $\mathbf{P}$ is a $|\Psi| \times |\Psi|$ matrix, p_{ji} is the element in j-th row and i-th column and

$$p_{ji} = \sum_{k, \forall h(\sigma_j)_k = \sigma_i} p(\alpha(\sigma_j)_k). \tag{2.19}$$

Given the steady-state distribution of the codebook, the average code rate $\bar{R}$ is evaluated as

$$\bar{R} = \frac{\sum\limits_{l_i} l_i \times 2^{-l_i}}{\sum\limits_{l_i} (\sum\limits_{j=1}^{|\Psi|} \widetilde{\pi}(\sigma_j) \times o(\sigma_j)_i) \times 2^{-l_i}} \tag{2.20}$$

where $o(\sigma_j)_i$ is the length of the codeword emitted from state σ_j due to the occurrence of the i-th source word.

Similar to single-state encoding, by performing partial extensions, different codebooks can be generated depending on different concatenations of words in partial extensions. We can establish limits on n_{max} or c_{max}, use an exhaustive search to compare all codebooks that are within these limits, and choose the one that has the highest $\bar{R}$.

Example ($d = 1, k = 3$ RLL code): If we do not perform partial extensions but directly perform NGH coding over the minimal set shown in Table 2.11, we obtain the simple yet efficient codebook shown in Table 2.16. Although $|\alpha(\sigma_1)| = |\alpha(\sigma_2)| = 2$, the number of unique codewords in Table 2.16 is only three. It can be verified that the steady-state distribution for this codebook is $\widetilde{\pi} = [\frac{2}{3}\ \frac{1}{3}]$ and that the average code rate is

$$\bar{R} = \frac{1 \times \frac{1}{2} + 1 \times \frac{1}{2}}{\frac{1}{2} \times 2 + \frac{1}{2} \times (\frac{2}{3} \times 2 + \frac{1}{3} \times 1)} = \frac{6}{11}$$

which achieves 98.9% of capacity. Note that this code is as efficient as the single-state code given in Table 1.6, but that it has shorter codewords and source words. Higher efficiency can be achieved with partial extensions and a larger codebook. To compare, another variable-length coding technique [37]

gives a rate 0.5 variable-length ($d = 1, k = 3$) code with efficiency 90.7%, which demonstrates the effectiveness of our proposed construction technique.

Table 2.16: Codebook of a ($d = 1, k = 3$) RLL code with two states and $\eta = 98.91\%$

Source word	$\alpha(\sigma_1)$	$\beta(\sigma_1)$	$\alpha(\sigma_2)$	$\beta(\sigma_2)$
0	01	σ_1	01	σ_1
1	00	σ_2	1	σ_1

Example (DC-free codes with $N = 5$): Using the minimal set shown in Table 2.12 as the codebook, we are able to construct a code with efficiency $\eta = 96.46\%$. After performing partial extensions with maximum codeword length $c_{max} = 4$, we obtain the codebook shown in Table 2.17 which has efficiency $\eta = 99.14\%$. We also note that with a ternary source, it is possible to achieve 100% of capacity simply by using words in the minimal set as the codewords, as demonstrated in Table 2.18.

Table 2.17: Codebook of a $N = 5$ DC-free code with $\eta = 99.14\%$.

Source word	$\alpha(\sigma_1)$	$\beta(\sigma_1)$	$\alpha(\sigma_2)$	$\beta(\sigma_2)$
00	11	σ_2	00	σ_1
010	0111	σ_2	1000	σ_1
011	0101	σ_1	0101	σ_2
100	0110	σ_1	0110	σ_2
101	1011	σ_2	0100	σ_1
110	1001	σ_1	1001	σ_2
111	1010	σ_1	1010	σ_2

Table 2.18: Codebook of a $N = 5$ DC-free code with ternary source and $\eta = 100\%$.

Source word	$\alpha(\sigma_1)$	$\beta(\sigma_1)$	$\alpha(\sigma_2)$	$\beta(\sigma_2)$
0	11	σ_2	00	σ_1
1	10	σ_1	10	σ_2
2	01	σ_1	01	σ_2

Example (DC-free RLL codes with $(d = 1, k = 3, N = 5)$): Based on the extended minimal set shown in Table 2.14, if we use the words in the extended minimal set as the codewords, the average code rate is $\bar{R} = 0.4167$ and $\eta = 98.09\%$. The corresponding codebook is shown in Table 2.19. Based on the partial extension in Table 2.15, we construct the codebook with $\bar{R} = 0.4183$

Table 2.19: Codebook of a DC-free RLL code corresponding to an NRZI encoded $(d = 1, k = 3)$ code with $N = 5$, $\eta = 98.09\%$

Source word	$\alpha(\sigma_1)$	$\beta(\sigma_1)$	$\alpha(\sigma_2)$	$\beta(\sigma_2)$	$\alpha(\sigma_3)$	$\beta(\sigma_3)$	$\alpha(\sigma_4)$	$\beta(\sigma_4)$
0	011	σ_4	011	σ_3	100	σ_2	100	σ_1
10	1100	σ_1	1100	σ_2	00011	σ_4	1100	σ_2
11	11100	σ_2	0011	σ_4	0011	σ_3	0011	σ_4

Table 2.20: Codebook of a DC-free RLL code corresponding to an NRZI encoded $(d = 1, k = 3)$ code with $N = 5$, $\eta = 98.47\%$

Source word	$\alpha(\sigma_1)$	$\beta(\sigma_1)$	$\alpha(\sigma_2)$	$\beta(\sigma_2)$	$\alpha(\sigma_3)$	$\beta(\sigma_3)$	$\alpha(\sigma_4)$	$\beta(\sigma_4)$
000	011100	σ_1	011100	σ_2	100011	σ_3	100011	σ_4
001	0111100	σ_2	01100011	σ_4	1001100	σ_2	1001100	σ_1
010	0110011	σ_4	0110011	σ_3	1000011	σ_4	10011100	σ_2
011	1100011	σ_4	1100011	σ_3	00011100	σ_1	1100011	σ_3
100	11001100	σ_1	11001100	σ_2	000111100	σ_2	11001100	σ_2
101	110011100	σ_2	11000011	σ_4	000110011	σ_4	11000011	σ_4
11	11100	σ_2	0011	σ_4	0011	σ_3	0011	σ_4

Table 2.21: Codebook of a DC-free RLL code with $(d = 2, k = 3, N = 5)$, $\eta = 98.62\%$

Source word	$\alpha(\sigma_1)$	$\beta(\sigma_1)$	$\alpha(\sigma_2)$	$\beta(\sigma_2)$
0	111000	σ_1	000111	σ_2
1	0111	σ_2	1000	σ_1

Table 2.22: Codes constructed that satisfy different DC-free RLL constraints

Constraint	η	Number of states	Number of source words
$d = 1, k = 3, N = 5$	98.09%	4	3
$d = 2, k = 3, N = 5$	98.62%	2	2
$d = 1, k = 4, N = 6$	97.61%	6	4
$d = 1, k = 5, N = 7$	98.27%	8	5
$d = 2, k = 5, N = 7$	97.66%	6	4

Table 2.23: Codebook of a DC-free RLL code with $(d = 1, k = 5, N = 7), \eta = 98.27\%$

Source word	$\alpha(\sigma_1)$	$\beta(\sigma_1)$	$\alpha(\sigma_2)$	$\beta(\sigma_2)$	$\alpha(\sigma_3)$	$\beta(\sigma_3)$	$\alpha(\sigma_4)$	$\beta(\sigma_4)$
11	011	σ_2	100	σ_1	011	σ_8	011	σ_6
01	1100	σ_1	0000011	σ_6	1100	σ_3	1100	σ_4
00	0011	σ_8	0011	σ_2	0011	σ_5	111100	σ_3
101	00011	σ_5	00011	σ_8	00011	σ_6	1111100	σ_1
100	000011	σ_6	000011	σ_5	11100	σ_1	11100	σ_7

Source word	$\alpha(\sigma_5)$	$\beta(\sigma_5)$	$\alpha(\sigma_6)$	$\beta(\sigma_6)$	$\alpha(\sigma_7)$	$\beta(\sigma_7)$	$\alpha(\sigma_8)$	$\beta(\sigma_8)$
11	100	σ_8	100	σ_4	011	σ_5	100	σ_3
01	1100	σ_3	1100	σ_8	1100	σ_8	110	σ_1
00	0011	σ_5	0011	σ_6	0011	σ_6	0011	σ_7
101	00011	σ_6	111100	σ_1	111100	σ_1	00011	σ_5
100	11100	σ_1	11100	σ_3	11100	σ_3	000011	σ_6

and $\eta = 98.47\%$ shown in Table 2.20. Further improvement of efficiency can be obtained via partial extensions with larger c_{max} and/or n_{max}.

Example (Other DC-free RLL codes): In Table 2.21 we present a very simple, but highly efficient, multi-state code for the $(d = 2, k = 3, N = 5)$ constraint. In Table 2.22 we list parameters of other DC-free RLL codes that we have constructed. Note that still higher efficiencies can be achieved for these values of d, k and N with larger codebooks.

To compare, a state-of-the-art fixed-length code construction technique for the $(d = 1, k = 3, N = 5)$ DC-free RLL constraint gives a rate 0.4 code with $\eta = 94.16\%$, and the codebook has 18 states, 256 source words and hundreds of codewords [38]. As shown in Tables 2.19 and 2.22, however, our proposed construction gives a codebook with only 4 states, 3 source words and 6 different codewords, and has efficiency $\eta = 98.09\%$. [38] also proposed a $(d = 1, k = 5, N = 7)$ DC-free RLL code with 20 states and 16 source words that results in an efficiency of $\eta = 90.96\%$. Our construction gives a code with 8 states, 5 source words and $\eta = 98.27\%$ demonstrating that our proposed variable-length construction technique can significantly reduce the complexity and improve the efficiency of constrained sequence codes. The codebook is shown in Table 2.23. Other examples of DC-free RLL codes that we have constructed are summarized in Table 2.22.

As is evident in the above examples, all the codes we have presented

(and the codes we construct in the rest of this paper) are state-independently decodable. This requires attention during the construction process. For instance, consider the minimal set where $W_r \in W(\sigma_u) \cap W(\sigma_y)$ and $W_r \notin W(\sigma_v)$. This indicates that state σ_v cannot generate W_r because of the constraint described by the FSM. But with an appropriate selection of principal states, state σ_v would generate another word W_s such that $W_s \notin W(\sigma_u)$ and $W_s \notin W(\sigma_y)$. State-independent code design would have W_r and W_s constitute a row, thus satisfying Condition 2.1. For example in Table 2.13, $w(\sigma_1)_1$ and $w(\sigma_2)_1$ are 011 while $w(\sigma_3)_1$ and $w(\sigma_4)_1$ are 100, where it is clear that states σ_1 and σ_2 can generate 011 but not 100, and vice versa for states σ_3 and σ_4. Words 100 and 011 constitute the first row, and Condition 2.1 is satisfied. Given a minimal set that has the state-independent decoding property, it is readily seen that codebooks constructed through its partial extensions can be state-independently decoded.

2.2.2 Multi-state encoding based on n-step FSM

As demonstrated above, different principal states may have a different number of words, i.e. there may exist i, j such that $|W(\sigma_i)| \neq |W(\sigma_j)|$. This will cause imbalance in the number of words in different states in the minimal set, i.e., the number of words in different states is unequal, and hence this may cause a different number of codewords associated with different states in the codebook after partial extensions. Although, as we illustrated in the previous section, it may be possible to construct an extended minimal set where $|W(\sigma_i)| = |W(\sigma_j)|$ by extending some words, this approach does not apply for all constraints. For example, with a DC-free $N = 6$ constraint, if we select states 2, 4 and 6 as principal states, i.e. $\sigma_1 = 2, \sigma_2 = 4, \sigma_3 = 6$, we have $W(\sigma_1) = \{01, 10, 11\}, W(\sigma_2) = \{01, 10, 11, 00\}$, and $W(\sigma_3) = \{01, 00\}$. A feasible codebook requires the number of codewords in $\alpha(\sigma_i), 1 \leq i \leq |\xi|$ to be the same. If we choose to concatenate words in $W(\sigma_3)$ to compensate for

the imbalance, some words in $W(\sigma_1)$ and $W(\sigma_2)$ become prefixes of words in $W(\sigma_3)$, and after partial extensions, some codewords in $\alpha(\sigma_1)$ and $\alpha(\sigma_2)$ become prefixes of words in $\alpha(\sigma_3)$, eliminating the possibility of state-independent decoding. Alternatively, it is possible to eliminate words from $W(\sigma_1)$ and $W(\sigma_2)$ to force the same number of words in all the principal states, however this will result in rate loss.

In this section we extend our construction technique to include the use of n-step FSMs, and illustrate this extension with DC-free codes because of their importance in recently-developed VLC systems. In the next section we consider a special case of n-step FSMs for DC-free codes that can result in even further enhancement.

n-step FSM

An n-step FSM describes transitions among states where the edge labels represent the concatenation of n successive edges of the initial FSM. For example, when the initial FSM contains edge labels of a single symbol, the labels in the n-step graph have length n. The adjacency matrix of an n-step FSM is $\mathbf{D}^n$ and the n-step transition matrix is $\mathbf{Q}^n$. The asymptotic steady-state distribution $\boldsymbol{\pi}$ of a n-step FSM is the same as that of the initial FSM.

Principal states and minimal sets

Given the n-step FSM, the general code construction procedure is similar to the one introduced above. The concatenation of words and selection of principal states is the same as that introduced in the previous section. Should the number of words in the principal states be unequal, we perform concatenation over some of the words in $W(\Psi)$ in an attempt to construct an extended minimal set with the same number of rows in each state. Care must be taken in this step to ensure that the prefix condition is maintained, and that similarity in the number of words is improved. For example, it might occur that $w_r \in W(\sigma_1)$ and $w_r \in W(\sigma_2)$, where

$|W(\sigma_1)| < |W(\sigma_2)|$. If we concatenate w_r only in $W(\sigma_1)$ to increase the number of words in state σ_1, w_r in $W(\sigma_2)$ will become a prefix of some words in $W(\sigma_1)$. If we concatenate w_r in both $W(\sigma_1)$ and $W(\sigma_2)$, the inequality in number of words might become more pronounced. To address this problem, we must choose an appropriate value of n such that some words can be concatenated without those problems occurring in the new minimal set. We call the new minimal set the n-step minimal set.

We observe that in n-step FSMs of DC-free codes with N RDS values, state 1 and state N have fewer words than state $\lfloor \frac{N}{2} \rfloor$. Note that with $n = N-1$, the all-one word of length $N-1$ is generated by state 1 and the all-zero word of length $N-1$ is generated by state N. In addition, those two words do not occur in any other states in the minimal set. Therefore, it is possible to concatenate those two words with other words in the minimal set according to Definition 2.2 to compensate for the imbalance of words without causing the prefix problem to arise.

This observation that the imbalance of words in the n-step minimal set can be reduced also holds for n-step FSMs with $n = N-2$ where the all-one word is generated by states 1 and 2 and the all-zero word is generated by states $N-1$ and $N-2$. It is straightforward to verify that this observation applies when n is in the range

$$n = [\left\lceil \frac{N}{2} \right\rceil, N-1]. \tag{2.21}$$

From this range, we select the n that results in the highest achievable code rate, as will be discussed in the next subsection.

Example (DC-free $N = 6$ code) Consider the construction of an n-step minimal set for the DC-free $N = 6$ constraint. Using (2.21), we obtain the range of n as $[3, 5]$. Selecting all states as principal states, the minimal set for the 3-step FSM is shown in Table 2.24.

It is clear from this table that there is an unequal number of words associated with different states in this minimal set. As has been discussed, if we extend words in this table without due care, we may violate the prefix

Table 2.24: The minimal set of a 3-step DC-free code with $N = 6$

$W(\sigma_1)$	$H(\sigma_1)$	$W(\sigma_2)$	$H(\sigma_2)$	$W(\sigma_3)$	$H(\sigma_3)$	$W(\sigma_4)$	$H(\sigma_4)$	$W(\sigma_5)$	$H(\sigma_5)$	$W(\sigma_6)$	$H(\sigma_6)$
101	σ_2	101	σ_3	101	σ_4	101	σ_5	101	σ_6	010	σ_5
110	σ_2	110	σ_3	110	σ_4	110	σ_5	001	σ_4	001	σ_5
111	σ_4	011	σ_3	011	σ_4	011	σ_5	011	σ_6	000	σ_3
		010	σ_1	010	σ_2	010	σ_3	010	σ_4		
		100	σ_1	100	σ_2	100	σ_3	100	σ_4		
		111	σ_5	001	σ_2	001	σ_3	000	σ_2		
				111	σ_6	000	σ_1				

condition or cause greater imbalance to arise. For example, if we extend the word 101 in $W(\sigma_1)$, all other occurrences of the word 101 in the same row should be extended as well, otherwise they will become prefixes of the newly concatenated word. However, the extension of the word 101 in other columns in this row will make the imbalance more severe.

Motivated by the observation above, we perform concatenation of the all-one words in state $\sigma_1, \sigma_2, \sigma_3$ and of the all-zero words in state $\sigma_4, \sigma_5, \sigma_6$. The resulting table is shown in Table 2.25.

Table 2.25: An extended 3-step minimal set of a DC-free code with $N = 6$, $n = 3$

$W_c(\alpha_1)$	$H_c(\alpha_1)$	$W_c(\alpha_2)$	$H_c(\alpha_2)$	$W_c(\alpha_3)$	$H_c(\alpha_3)$	$W_c(\alpha_4)$	$H_c(\alpha_4)$	$W_c(\alpha_5)$	$H_c(\alpha_5)$	$W_c(\alpha_6)$	$H_c(\alpha_6)$
101	σ_2	101	σ_3	101	σ_4	101	σ_5	101	σ_6	000010	σ_2
111101	σ_5	010	σ_1	010	σ_2	010	σ_3	010	σ_4	010	σ_5
110	σ_2	110	σ_3	110	σ_4	110	σ_5	000011	σ_3	000011	σ_3
111100	σ_3	111100	σ_4	001	σ_2	001	σ_3	001	σ_4	001	σ_5
111011	σ_5	100	σ_1	100	σ_2	100	σ_3	100	σ_4	000100	σ_2
111110	σ_5	011	σ_3	011	σ_4	011	σ_5	011	σ_6	000001	σ_2
111010	σ_3	111010	σ_4	111010	σ_5	000101	σ_2	000101	σ_3	000101	σ_3
111001	σ_3	111001	σ_4	111001	σ_5	000110	σ_2	000110	σ_3	000110	σ_3
111000	σ_1	111000	σ_2	111000	σ_3	111000	σ_4	111000	σ_5	111000	σ_6
		111101	σ_6					000010	σ_1		
		111011	σ_6					000100	σ_1		

Pruning and maximum possible code rate

Careful extension of words should reduce the imbalance between the number of words from different states while ensuring that the prefix condition remains satisfied. However, should an inequality in the number of words associated with different states remain, it is possible to truncate some of the words to obtain a pruned version of the extended minimal set that has the same number of words in each state. We denote this pruned set as

$W^p(\Psi)$.

The number of words that must be truncated from state σ_j, denoted $u(\sigma_j)$, is

$$u(\sigma_j) = |W(\sigma_j)| - \min\{|W(\sigma_1)|, |W(\sigma_2)|, ..., \left|W(\sigma_{|\psi|})\right|\}. \tag{2.22}$$

Given $\mathbf{Q}^n$ and $\mathbf{D}^n$, we can evaluate the probability of the i-th word in $W(\sigma_j), 1 \leq j \leq |\Psi|$, which we denote $p(W(\sigma_j))_i$. Then, $u(\sigma_j)$ words with the lowest probabilities in $W(\sigma_j)$ are eliminated. We denote the set of words in each state of $W^p(\Psi)$ as $W^p(\sigma_i), 1 \leq i \leq |\Psi|$, and the set of next states corresponding to words in $W^p(\sigma_i)$ as $H^p(\sigma_i)$.

Since some words that satisfy the constraint are not used, we are not able to achieve full capacity. The achievable code rate of $W^p(\Psi)$, denoted as $\widetilde{C}_n$, is given by (2.23) where $l(W^p(\sigma_k))_v$ denotes the length of the v-th word in $W^p(\sigma_k)$, and $\widetilde{\boldsymbol{\pi}}^p = [\widetilde{\pi}^p(\sigma_1), \widetilde{\pi}^p(\sigma_2), ..., \widetilde{\pi}^p(\sigma_{|\Psi|})]$ is the steady-state distribution of $W^p(\Psi)$.

$$\widetilde{C}_n = \frac{\sum_{k=1}^{|\Psi|} \widetilde{\pi}^p(\sigma_k) \times \left(\sum_{v=1}^{|W^p(\sigma_j)|-u(\sigma_j)} -p(W^p(\sigma_k))_v \times \log_2 p(W^p(\sigma_k))_v \right)}{\sum_{k=1}^{|\Psi|} \widetilde{\pi}^p(\sigma_k) \times \left(\sum_{v=1}^{|W^p(\sigma_j)|-u(\sigma_j)} p(W^p(\sigma_k))_v \times l(W^p(\sigma_k))_v \right)}. \tag{2.23}$$

Note that $\widetilde{\boldsymbol{\pi}}^p$ is different from the steady-state distribution of the initial FSM of the constraint. It is evaluated similar to (2.18), but with $p(\alpha(\sigma_j))_i$ in (2.18) replaced with $p(W^p(\sigma_j))_i$. After evaluating $\widetilde{C}_n$, we choose to work with the $W^p(\Psi)$ with the highest $\widetilde{C}_n$ and the highest achievable efficiency $\widetilde{\eta}_n = \widetilde{C}_n/C$.

Example (DC-free $N = 6$ code): Based on (2.23), using the approach outlined in this section where the all-one and all-zero words are extended and $u(\sigma_j)$ words are pruned, the achievable code rates of different n-step FSMs are shown in Table 2.26. Note that although for illustration we use $n = 3$ as an example throughout this section, of the values of n considered $\widetilde{C}_n$ is highest with $n = 4$.

Encoding

Since we now have $W^p(\Psi)$ which contains the same number of words in each state, we can perform partial extensions and NGH coding to obtain

Table 2.26: Achievable code rates of different n-step FSMs with different values of n for the DC-free constraint with $N = 6$

	$n = 3$	$n = 4$	$n = 5$
$\widetilde{C}_n$	0.8448	0.8475	0.8331
$\widetilde{\eta}_n$	99.45%	99.76%	98.07%

the codebook in a manner similar to the FSM-based method in the previous section. As outlined above, the evaluation of the average code rate is given by (2.20). Within predetermined limits on n_{max} and/or c_{max}, an exhaustive search can be performed to determine the codebook with the highest $\bar{R}$.

Example (DC-free $N = 6$ code) Based on Table 2.25 and (2.22), the words 111101 and 111011 from state σ_2, and the words 000010 and 000100 from state σ_5, are removed to result in an equal number of words in all states in the 3-step minimal set. If we use this 3-step minimal set as the codebook, and perform NGH coding to obtain the assignment of source words, we construct the codebook shown in Table 2.27 that has an efficiency of 92.8%.

By performing partial extensions over this 3-step minimal set, we are able to construct codebooks with higher average code rates. Some results are listed in Table 2.28.

Lastly, we note that the construction process introduced in this section can be used for a variety of constraints, and in some instances can result in a codebook with few principal states, or an extended minimal set with an equal number of words in all states so that so pruning is not needed. In the next section we focus on DC-free codes for VLC systems, and we show that appropriately designed DC-free codebooks can benefit from both of these conditions.

2.2.3 Examples: codes for visible light communications

In the previous section we showed that, with our proposed encoding method, codebooks that satisfy the DC-free $N = 5$ constraint can be constructed with over 99% efficiency and with fewer codewords than the 4B6B code described earlier. Therefore, our codes can be considered superior in terms of both efficiency and implementation complexity. In this

Table 2.27: Codebook of a pruned version of the extended 3-step minimal set of a DC-free code with $N = 6$

Source word	$\alpha(\sigma_1)$	$\beta(\sigma_1)$	$\alpha(\sigma_2)$	$\beta(\sigma_2)$	$\alpha(\sigma_3)$	$\beta(\alpha_3)$
000	101	σ_2	101	σ_3	101	σ_4
001	110	σ_2	110	σ_3	110	σ_4
10	111101	σ_5	010	σ_1	010	σ_2
110	111110	σ_5	100	σ_1	100	σ_2
010	111011	σ_5	011	σ_3	011	σ_4
11100	111010	σ_3	111010	σ_4	111010	σ_5
011	111100	σ_3	111100	σ_4	001	σ_2
1111	111001	σ_3	111001	σ_4	111001	σ_5
11101	111000	σ_1	111000	σ_2	111000	σ_3

Source word	$\alpha(\sigma_4)$	$\beta(\sigma_4)$	$\alpha(\sigma_5)$	$\beta(\sigma_5)$	$\alpha(\sigma_6)$	$\beta(\sigma_6)$
000	101	σ_5	101	σ_6	000010	σ_2
001	110	σ_5	000011	σ_3	000011	σ_3
10	010	σ_3	010	σ_4	010	σ_5
110	100	σ_3	100	σ_4	000100	σ_2
010	011	σ_5	011	σ_6	000001	σ_2
11100	000101	σ_2	000101	σ_3	000101	σ_3
011	001	σ_3	001	σ_4	001	σ_5
1111	000110	σ_2	000110	σ_3	000110	σ_3
11101	111000	σ_4	111000	σ_5	111000	σ_6

Table 2.28: Highest average code rates of DC-free code with $N = 6$ codebooks with different size

n_{max}	25	33	41
$\bar{R}$	0.801	0.8047	0.8054
η	94.29%	94.72%	94.81%

subsection, we now focus on coding for the DC-free constraint with $N = 7$, and compare our results with the 8B10B codes.

We present codes constructed for VLC systems based on n-step FSMs. We show that based on n-step FSMs with an even n, the number of principal states in a DC-free code can be reduced to $N/2$ when N is even and either $\lfloor N/2 \rfloor$ or $\lceil N/2 \rceil$ when N is odd, and that an extended minimal set with an equal number of words in each state can be obtained such that $\widetilde{\eta} = 100\%$.

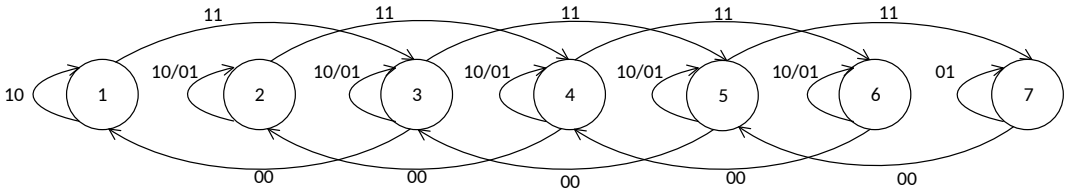

Figure 2.19: 2-step FSM of the DC-free constraint with $N = 7$.

The reduction of states is based on the observation that with DC-free constraints, when n is even, the n-step edge graphs subdivide into two non-intersecting FSMs. An example of this phenomenon is shown in Fig. 2.19, where it is evident that in this 2-step edge graph of the DC-free constraint with $N = 7$, the FSM comprised of the even-numbered states is not connected to the FSM comprised of the odd-numbered states. However, as discussed in [39], each of these smaller FSMs generate all constraint satisfying sequences, and therefore either one can be used as the basis for our variable-length code design. It can also be verified that the 2-step FSM comprised of the three even-numbered states has the steady-state probability distribution $\boldsymbol{\pi} = [0.2929.0.4142, 0.2929]$, whereas the 2-step FSM of the four odd-numbered states has the steady-state probability distribution $\boldsymbol{\pi} = [0.1464, 0.3536, 0.3536, 0.1464]$.

Following the construction technique introduced above with the 2-step FSM, we now consider the construction process specifically for DC-free codes with any value of N. We show that it is always possible to construct an extended minimal set with a maximum achievable efficiency $\widetilde{\eta} = 100\%$ with only $N/2$ principal states when N is even, and with $\lfloor N/2 \rfloor$ or $\lceil N/2 \rceil$

principal states when N is odd, depending on whether we work with the set of even-numbered states or the set of odd-numbered states. We begin with the following example for $N = 7$ with the set of even-numbered states.

Example (DC-free codes with $N = 7$): When we consider the set of even-numbered states, the minimal set of the 2-step FSM is shown in Table 2.29. The achievable code rate of this codebook is 0.8858, which is the capacity of DC-free constraint with $N = 7$, confirming that all constraint-satisfying sequences are generated by this three-state FSM.

We extend the words 11 in $W(\sigma_2), W(\sigma_4)$, and 00 in $W(\sigma_4), W(\sigma_6)$ by tracing the edges corresponding to 11 and 00 to construct Table 2.30. Then, we extend word 1111 in $W(\sigma_2)$, and 0000 in $W(\sigma_6)$ by once again tracing the edges corresponding to 11 and 00, and obtain an extended minimal set with $\xi = 9$ without causing the prefix problem. Note that with this extended minimal set, the achievable efficiency is 100% since no pruning is performed. If we use this minimal set as the codebook and perform the encoding procedure as outlined in this section, we obtain the codebook in Table 2.31 with $\bar{R} = 0.8462$ and $\eta = 95.53\%$. By performing partial extensions with $n_{max} = 15$, we have constructed a codebook with $\bar{R} = 0.8535$ and $\eta = 96.35\%$.

Table 2.29: A 2-step minimal set of DC-free code $N = 7$

$W(\sigma_2)$	$H(\sigma_2)$	$W(\sigma_4)$	$\beta(\sigma_4)$	$H(\sigma_6)$	$W(\sigma_6)$
10	σ_2	10	σ_4	10	σ_6
01	σ_2	01	σ_4	01	σ_6
11	σ_4	11	σ_6	00	σ_4
		00	σ_2		

Table 2.30: An extended 2-step minimal set of DC-free code $N = 7$

$W(\sigma_2)$	$H(\sigma_2)$	$W(\sigma_4)$	$\beta(\sigma_4)$	$H(\sigma_6)$	$W(\sigma_6)$
10	σ_2	10	σ_4	10	σ_6
01	σ_2	01	σ_4	01	σ_6
1110	σ_4	1110	σ_6	0010	σ_4
1101	σ_4	1101	σ_6	0001	σ_4
1111	σ_6	1100	σ_4	0011	σ_6
1100	σ_2	0010	σ_2	0000	σ_2
		0001	σ_2		
		0011	σ_4		

Table 2.31: A DC-free $N = 7$ codebook, $\eta = 95.53\%$

Source words	$\alpha(\sigma_2)$	$\beta(\sigma_2)$	$\alpha(\sigma_4)$	$\beta(\sigma_4)$	$\alpha(\sigma_6)$	$\beta(\sigma_6)$
10	10	σ_2	10	σ_4	10	σ_6
11	01	σ_2	01	σ_4	01	σ_6
0111	111101	σ_6	0010	σ_2	0010	σ_4
001	111100	σ_4	0011	σ_4	0011	σ_6
0110	111110	σ_6	0001	σ_2	0001	σ_4
0101	1101	σ_4	1101	σ_6	000010	σ_2
000	1100	σ_2	1100	σ_4	000011	σ_4
0100	1110	σ_4	1110	σ_6	000001	σ_2

Example (DC-free codes with $N = 7$): We now consider the construction of a codebook with the set of odd-numbered states as principal states. The minimal set is shown in Table 2.32, where the principal states are $\sigma_1, \sigma_3, \sigma_5, \sigma_7$. As in the example above, we perform extensions by tracing the edges corresponding to 11 and 00 to obtain an extended minimal set with an equal number of words in each principal state. By performing NGH coding over this extended minimal set, we obtain a codebook with $\bar{R} = 0.8405$ and $\eta = 94.88\%$, which is shown in Table 2.33. By performing partial extensions with $n_{max} = 17$, we have constructed a codebook with $\bar{R} = 0.8468$ and $\eta = 95.59\%$.

The above examples demonstrate that when $N = 7$, it is possible to extend the all-one and all-zero words are extended until there are an equal number of words associated with each state. The fact that this is possible for an arbitrary N where $N \geq 3$ is given in the proof of the following theorem. Note that when $N = 2$ the code construction is straightforward based on the minimal set.

Table 2.32: A 2-step minimal set of DC-free code $N = 7$ with the set of odd states as principal states

$W(\sigma_1)$	$H(\sigma_1)$	$W(\sigma_3)$	$\beta(\sigma_3)$	$H(\sigma_5)$	$W(\sigma_5)$	$H(\sigma_7)$	$W(\sigma_7)$
10	σ_1	10	σ_3	10	σ_5	01	σ_7
11	σ_3	01	σ_3	01	σ_5	00	σ_5
		11	σ_5	11	σ_7		
		00	σ_1	00	σ_3		

Table 2.33: A DC-free $N = 7$ codebook, $\eta = 94.88\%$

Source words	$\alpha(\sigma_1)$	$\beta(\sigma_1)$	$\alpha(\sigma_3)$	$\beta(\sigma_3)$	$\alpha(\sigma_5)$	$\beta(\sigma_5)$	$\alpha(\sigma_7)$	$\beta(\sigma_7)$
10	10	σ_1	10	σ_3	10	σ_5	000001	σ_3
010	1110	σ_3	1110	σ_5	0001	σ_3	0001	σ_5
0110	1101	σ_3	1101	σ_5	1101	σ_7	00000010	σ_1
0111	1100	σ_1	1100	σ_3	1100	σ_5	00000011	σ_3
00	111110	σ_5	01	σ_3	01	σ_5	01	σ_7
11000	111101	σ_5	111101	σ_7	000010	σ_1	000010	σ_3
11001	111100	σ_3	111100	σ_5	000011	σ_3	000011	σ_5
1101	11111101	σ_7	0010	σ_1	0010	σ_3	0010	σ_5
111	11111100	σ_5	0011	σ_3	0011	σ_5	0011	σ_7

Theorem 2.1 For DC-free constraints with any N where $N \geq 3$, we can obtain an extended minimal set with an equal number of words in all states based on extension of the all-zero and all-one words, where only $N/2$ states are selected as principal states when N is even, and when either $\lfloor N/2 \rfloor$ or $\lceil N/2 \rceil$ states are selected as principal states when N is odd. These codes have achievable efficiency $\widetilde{\eta} = 100\%$, and are instantaneously decodable.

Proof. When $N = 3, 4$ the proof is straightforward. We now prove for the situations where $N > 4$. We consider 2-step FSMs, and consider odd and even N separately.

i) We first consider odd N with the set of $\lfloor \frac{N}{2} \rfloor$ even-numbered states as principal states, i.e. $\Psi = \{\sigma_2, \sigma_4, ..., \sigma_{N-1}\}$. It is readily seen that the number of words in each state in Ψ is $N_\Psi = \{3, 4, 4, ..., 4, 3\}$, because for a state $\sigma_j \in \{\sigma_4, \sigma_6, ..., \sigma_{N-3}\}$, $W(\sigma_j) = \{01, 10, 00, 11\}$, and for the other two states, $W(\sigma_2) = \{01, 10, 11\}$ and $W(\sigma_{N-1}) = \{01, 10, 00\}$. Starting from a state σ_j, another state $\sigma_i, i \in \{j + 2, j - 2\}$ is reached in a single extension with label 11 (when $i > j$) or 00 (when $i < j$). With each extension we reach another state in Ψ. Since $|\Psi| = \lfloor N/2 \rfloor$, the maximum number of extensions that result in the all-one or all-zero sequence is $\lfloor N/2 \rfloor - 1$.

We denote $\Delta_{N_{\sigma_j}}$ as the number of new words generated from an extension of the edge with label 11 or 00. Consider $\sigma_j = \sigma_2$, and consider the number of words that can occur as an extension of the edge 11. Since that edge has reached state σ_4, when $4 < N - 1$ there are four possible words:

$1101, 1110, 1100, 1111$ since $W(\sigma_4) = \{01, 10, 00, 11\}$, and hence $\Delta_{N_{\sigma_j}} = 4$. Since the word 1111 has reached state σ_6, when $6 < N - 1$ there are four extended words $111101, 111110, 111100, 111111$ hence $\Delta_{N_{\sigma_j}} = 4$ (otherwise we reach extension number $\lfloor N/2 \rfloor - 1$). Continuing in this manner, it can be deduced that in the first $\lfloor N/2 \rfloor - 2$ extensions, $\Delta_{N_{\sigma_j}} = 4$. In extension number $\lfloor N/2 \rfloor - 1$, however, $\Delta_{N_{\sigma_j}} = 3$ since it reaches state σ_{N-1} where $|W(\sigma_{N-1})| = 3$, and the extended words do not include the all-one word. Therefore, the total number of words N_{σ_j} in state σ_j once the all-one word is no longer in the set is:

$$\begin{aligned} N_{\sigma_2} &= \sum_{k=2,4,6,\ldots,N-1} \Delta_{N_{\sigma_k}} \\ &= 3 + 4(\lfloor N/2 \rfloor - 2) + 3 - (\lfloor N/2 \rfloor - 1) \\ &= 3\lfloor N/2 \rfloor - 1. \end{aligned} \tag{2.14}$$

Similar analysis holds for σ_{N-1}. The first extension of the edge 00 from σ_{N-1} results in four extended words $0001, 0010, 0011, 0000$ since $W(\sigma_{N-3}) = \{01, 10, 11, 00\}$, hence $\Delta_{N_{\sigma_j}} = 4$. It can be deduced that in the first $\lfloor N/2 \rfloor - 2$ extensions $\Delta_{N_{\sigma_j}} = 4$, and the all-zero word remains in the set. In extension number $\lfloor N/2 \rfloor - 1$, $\Delta_{N_{\sigma_j}} = 3$, and this is the first extension that does not include the all-zero word. Therefore, the total number of words N_{σ_j} in state σ_j once the all-zero word no longer appears in this state is also $N_{\sigma_j} = 3\lfloor N/2 \rfloor - 1$.

For $\sigma_j \in \{\sigma_4, \sigma_6, ..., \sigma_{N-3}\}$, both 11 and 00 in are traced during extensions. It can be verified that the number of extensions of the all-one word is $\frac{N-1}{2} - \frac{j}{2}$, where $\Delta_{N_{\sigma_j}} = 4$ in the first $\frac{N-1}{2} - \frac{j}{2} - 1$ extensions and $\Delta_{N_{\sigma_j}} = 3$ in the last extension, since it has reached state σ_{N-1}. Similarly, the number of extensions of the all-zero word is $\frac{j}{2} - 1$ where $\Delta_{N_{\sigma_j}} = 4$ in the first $\frac{j}{2} - 2$ extensions and $\Delta_{N_{\sigma_j}} = 3$ in the last extension, since it has reached state σ_2. Therefore, the total number of words N_{σ_j} in state σ_j once the all-one and the all-zero words are no longer in the set is:

$$
\begin{aligned}
N_{\sigma_j} &= 4 + 4(\lfloor N/2 \rfloor - \frac{j}{2} - 1) + \\
&\quad 3 + 4(\frac{j}{2} - 2) + 3 - (\lfloor N/2 \rfloor - 1) \\
&= 3\lfloor N/2 \rfloor - 1.
\end{aligned}
\tag{2.15}
$$

Thus if all principal states are extended just to the point where they no longer contain either the all-zero or all-one words, each of the principal states $\Psi = \{\sigma_2, \sigma_4, ..., \sigma_{N-1}\}$ have $3\lfloor N/2 \rfloor - 1$ words in the extended minimal set, and hence $\widetilde{\eta} = 100\%$ since no pruning is required to construct a set in which all principal states have the same number of words.

ii) When we choose odd-numbered N with the set of odd states, similar to the above analysis, the total number of words N_{σ_j} in state $\sigma_j \in \{\sigma_1, \sigma_N\}$ once the all-one or all-zero words are no longer in the set is

$$
\begin{aligned}
N_{\sigma_j} &= 2 + 4(\lceil N/2 \rceil - 2) + 2 - (\lceil N/2 \rceil - 1) \\
&= 3\lceil N/2 \rceil - 3,
\end{aligned}
\tag{2.16}
$$

and the total number of words N_{σ_j} in state $\sigma_j \in \{\sigma_3, \sigma_5, ..., \sigma_{N-2}\}$ once the all-one or all-zero word is no longer in the set is

$$
\begin{aligned}
N_{\sigma_j} &= 4 + 4(\lceil N/2 \rceil - \frac{j+1}{2} - 1) + \\
&\quad 2 + 4(\frac{j+1}{2} - 2) + 2 - (\lceil N/2 \rceil - 1) \\
&= 3\lceil N/2 \rceil - 3.
\end{aligned}
\tag{2.17}
$$

Therefore all states have $3\lceil N/2 \rceil - 3$ words, and $\widetilde{\eta} = 100\%$.

iii) Similarly, when we choose even N with either the set of even-numbered or odd-numbered states, the total number of words N_{σ_j} in state $\sigma_j \in \{\sigma_1, \sigma_3, ..., \sigma_{N-1}\}$ or $\sigma_j \in \{\sigma_2, \sigma_4, ..., \sigma_N\}$ after all extensions is

$$
N_{\sigma_j} = 3(\frac{N}{2}) - 2, \tag{2.18}
$$

so there is the same number of words in all principal states of the extended minimal set and $\widetilde{\eta} = 100\%$.

iv) We now prove the words in the extended minimal set are prefix-free such that they are instantaneously decodable. First we observe that in the

minimal set $W(\Psi)$, no word is a prefix of another. Therefore, the prefix problem could only have occurred if a word $W_q \in W(\sigma_u), W(\sigma_v)$ is extended in σ_u, but is not extended in σ_v. During the extensions described in this proof, only the all-one word or all-zero word is extended. Therefore, if the all-one word $W_q \in \{W(\sigma_u), W(\sigma_v)\}$, it is extended in σ_u and it is also extended in σ_v, since extensions continue until the all-one word is no longer in the set. Similar analysis holds for the all-zero word. Hence, the prefix problem is avoided and codebooks constructed based on these balanced extended minimal sets are instantaneously decodable. □

Recall that the 8B10B code employed in VLC has $R = 0.8$. Tables 2.31 and 2.33 present simple codes with code rates $\bar{R} = 0.8462$ and $\bar{R} = 0.8405$, respectively. With similar high code rates, the codes proposed with the single-state variable-length coding scheme in [18] include significantly more and longer words. Thus with multi-state encoding, we can construct codes with fewer and shorter codewords to satisfy DC-free constraints for VLC.

Chapter 3

Applications

In this chapter we discuss applications of our proposed variable-length constrained sequence codes in emerging data storage devices especially flash memories. We propose novel constrained coding schemes to mitigate inter-cell interference and deal with cell leakage. The proposed schemes have less redundancy compared with conventional schemes, and result in better bit error rate performance. The contents of this chapter were published in [20] and [21].

3.1 Flash memory basis

3.1.1 Structure and programming schemes

In this subsection we review the basics of flash memory, its different structures, and its programming schemes.

The basic storage unit of flash memories is a floating-gate transistor called a cell. Electrons can be injected or removed from a cell, and information is represented by modulating the amount of charge present on the floating gate. Flash memory is classified based on the number of bits each cell represents: the single-level cell (SLC), multi-level cell (MLC) and triple-level cell (TLC) architectures which store 1, 2 and 3 bits in each cell, respectively. In flash memory, *writing* or *programming* means injecting electrons into cells while *erasing* is the process of removing electrons from cells.

Cells are organized as blocks in flash memory, where a block is the basic

unit of erasure. One block has several word-lines which represent a series of cells in either an all-bit-line structure or an even/odd bit-line structure. In flash memories of an all-bit-line structure, each cell stores l^b bits where each word-line consists of l^b pages. That is, for SLC flash memory, one word-line contains only one page, while for MLC and TLC, a word-line contains two and three pages respectively. In contrast to [40], we denote the lower voltage level in SLC as a bit of value 1 and the higher voltage level as a bit of value 0. For MLC and TLC, we assume that bits are stored in a cell according to the Gray mappings given in Tables 3.1 and 3.2. In these representations, each bit corresponds to one specific page: s_0 is the lowest voltage level, s_3 is the highest voltage level in MLC and s_7 is the highest voltage level in TLC. In flash memories using an even/odd bit-line structure, even and odd bit-lines are interleaved along pages, dividing one page in an all-bit-line structure into two pages. Therefore, with the even and odd bit-line structure, some circuits and resources can be shared at the expense of more severe ICI, as discussed in [41] [42].

There are two basic programming schemes in flash memory: multi-page programming and full-sequence programming. In multi-page programming, data is programmed on a page-by-page basis. The first bit in each cell, which corresponds to page 1, is programmed first, before proceeding to the second and third bits corresponding to pages 2 and 3 respectively. When programming bits on higher-level pages, a read operation is first required to determine the level to be programmed. Therefore in flash memories storing l^b bits per cell, l^b programming steps are needed to fully program a cell with multi-page programming. In contrast, in full-sequence programming, all bits in a cell are programmed simultaneously regardless of the number of pages on a word-line. Full-sequence programming can achieve a higher throughput but suffers from greater ICI [42]. In this chapter we focus on multi-page programming with an all-bit-line structure, and consider the use of constrained codes to limit ICI in this structure.

Table 3.1: Gray mapping for MLC

State	s_0	s_1	s_2	s_3
page 1	1	1	0	0
page 2	1	0	0	1

Table 3.2: Gray mapping for TLC

State	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7
page 1	1	1	1	1	0	0	0	0
page 2	1	1	0	0	0	0	1	1
page 3	1	0	0	1	1	0	0	1

3.1.2 Inter-cell interference

Flash memory has gained much attention for non-volatile storage systems because of its low cost and high densities [43] [44]. Apart from its many merits, however, flash memory also suffers from some deficiencies, one of which is inter-cell-interference (ICI). ICI arises from parasitic capacitances between physically adjacent cells such that when charge is being inserted into one cell, charge may also be added to the floating gates of the neighboring cells, resulting in a rise of their voltage levels. ICI is becoming increasingly problematic because with advances in scaling technology, the density of cells is increasing dramatically. Simultaneously, more information bits are being stored in each cell, resulting in narrower threshold voltage distributions. Both issues lead to ICI having a more severe impact in flash memory.

Recently, constrained sequence codes have been successfully applied in flash memory systems to reduce the effect of ICI [40] [45] [46] [47]. The capacity of ICI-free balanced codes and ICI-free write-once memory codes is derived in [40]. In [45], constrained codes are used to limit ICI by setting an ICI severity function beforehand; this approach is extended to two-dimensional cases in [46]. RLL codes are employed to mitigate ICI in [47] by forbidding the most severe ICI pattern.

In the following, we apply our variable-length coding technique to limit or remove ICI in flash memory with multi-page programming.

3.2 Coding for flash memory with multi-page Programming [20]

In flash memory, ICI occurs when a cell with low charge is surrounded by cells with high charge. In all-bit-line systems we consider the patterns written to cells along an individual word-line. In [47], the use of $d = 1$ RLL codes is proposed in order to forbid the occurrence along a word-line of the pattern that introduces the most severe ICI. For example, the scheme proposed in [47] that uses an RLL code and NRZI coding forbids the pattern 010 which can result in significant ICI in SLC flash memory. However, this scheme also forbids the pattern 101 which does not contribute significantly to ICI. Forbidding this second pattern lowers the code rate unnecessarily. According to [48], in MLC $s_3s_0s_3, s_3s_1s_3$ and $s_3s_2s_3$ are the patterns that are the most susceptible to ICI. [47] also proposes the use of $d = 1$ RLL codes on page 2 in MLC to forbid the pattern $s_3s_0s_3$. However, RLL codes are not optimal for mitigating ICI because, while they forbid the most severe ICI pattern, they also forbid other patterns that do not contribute to ICI which results in an unnecessary reduction in code rate. In this section we derive constraints and minimal set representations for SLC, MLC and TLC flash memories that more precisely mitigate ICI based on the observation of the Gray mapping schemes previously shown in Tables 3.1 and 3.2. In the next section we present highly efficient codes that satisfy these constraints.

3.2.1 Page-1 constraint

In this subsection we present the Page-1 constraint, where constrained coding is performed on page 1 to forbid pattern 010. We first discuss the Page-1 constraint for SLC which is a special case since SLC only has one page in each word-line. We then discuss the Page-1 constraint for MLC and TLC.

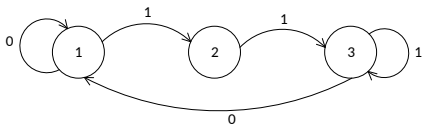

Figure 3.1: FSM of Page-1 constraint.

SLC

For SLC, it is sufficient to forbid only the pattern 010 in order to remove the effect of ICI. We denote this constraint as the *Page-1 constraint*. The FSM of this Page-1 constraint is shown in Fig. 3.1. The corresponding adjacency matrix is

$$\mathbf{D} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}, \tag{3.1}$$

from which the capacity of this Page-1 constraint is derived as $C_1 = 0.8114$, as was previously reported in [40].

As discussed in Chapter 2.1 of this thesis, according to criteria 2 and 3, state 1 is selected as the specified state since it has highest maximum possible code rate. The corresponding minimal set S_1 is established as $S_1 = \{0, 110, 1110, 11110, ...\}$. Truncating this set to make it practical results in an incomplete set because of the loop that exists in state 3. Codes constructed for this constraint are reported later in this section.

MLC/TLC

Recall that with multi-page programming, which we are considering in this chapter, data is programmed and fetched in the unit of pages. As a result, in order to avoid patterns that cause significant ICI, we now show that it is sufficient to impose the Page-1 constraint on MLC/TLC.

As shown in Tables 3.1 and 3.2, the bit value 1 on page 1 corresponds to the half of the states with lower voltages, while bit 0 on that page corresponds to the other half of the states with higher voltages. Accordingly, it follows

that if we perform Page-1 constrained coding to forbid the pattern 010 only on page 1, we eliminate the possibility of occurrence of the most severe ICI inducing patterns.

To demonstrate the usefulness of this constraint, note that introducing the Page-1 constraint in MLC forbids the following patterns: $s_3s_0s_3, s_3s_1s_3, s_2s_0s_2$, $s_2s_1s_2, s_3s_0s_2, s_3s_1s_2, s_2s_0s_3$ and $s_2s_1s_3$. In these patterns, the victim cell is in state s_0 or s_1. Since errors in state s_0 and s_1 contribute to most of the error events caused by ICI [41] [48], this Page-1 constraint is able to effectively mitigate the impact of ICI.

The same reasoning holds for TLC. Introducing the Page-1 constraint forbids the patterns $s_is_js_i, i \in \{4,5,6,7\}, j \in \{0,1,2,3\}$, where the victim cell is in state s_0, s_1, s_2 or s_3 which contribute to most of the error events caused by ICI. Note that all the patterns that we forbid by imposing the Page-1 constraint are ICI patterns; no other patterns are unnecessarily forbidden.

We now consider the capacity of the Page-1 constraint for MLC and TLC. Note that since we perform constrained coding on only one page while leaving data on the other pages uncoded, the capacity of the Page-1 constraint for MLC is $\frac{C_1+1}{2} = 0.9057$ bit/page/cell and for TLC it is $\frac{C_1+2}{3} = 0.9371$ bit/page/cell.

3.2.2 Page-2 constraints

Page-2A constraint

Based on the use of the Gray mapping, we propose constraints for MLC/TLC that have a higher capacity than the Page-1 constraint, at the cost of reduced ICI mitigation. Focus on page 2 instead of page 1. By considering state sequences that are possible with bit values on page 2, it is seen that the most severe ICI patterns are associated with the pattern 111 on page 2. Accordingly, if we perform constrained coding on page 2 to forbid the pattern 111 on that page, the sequence $s_3s_0s_3$ is prohibited in MLC, and the sequences $s_is_js_i, i \in \{6,7\}, j \in \{0,1\}$ will not occur in TLC. These are the patterns that induce the most severe ICI. We note that some patterns

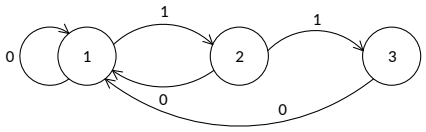

Figure 3.2: FSM of Page-2A constraint.

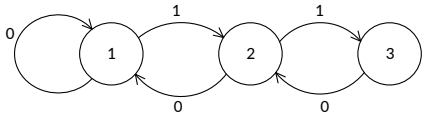

Figure 3.3: FSM of Page-2B constraint.

that do not induce ICI are also prohibited, however, as we now demonstrate, the code rate possible with this constraint still exceeds that which is possible with the Page-1 constraint.

The FSM of the Page-2 constraint is shown in Fig. 3.2. The adjacency matrix is

$$\mathbf{D} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \tag{3.2}$$

from which it follows that the capacity of the FSM in Fig. 3.2 is $C_{2A} = 0.8791$.

Since only data on page 2 is encoded, the capacity of the Page-2A constraint for MLC and TLC is $\frac{C_{2A}+1}{2} = 0.9396$ bit/page/cell and $\frac{C_{2A}+2}{3} = 0.9597$ bit/page/cell, respectively. According to criterion 2.1 as discussed in Chapter 2.1 of this thesis, we select state 1 as the specified state since it results in a complete minimal set. The minimal set of the Page-2A constraint is $S_{2A} = \{0, 10, 110\}$.

Page-2B constraint

The Page-2A constraint has a relatively high capacity at the cost of reduced ICI suppression, since fewer ICI patterns are forbidden. As a compromise, we propose a modification of the Page-2A constraint to eliminate more ICI patterns with only a small penalty in capacity.

Consider MLC flash memories. As discussed in the previous section, by imposing the Page-2A constraint, the most severe ICI pattern 303 is forbidden, and hence the ICI pattern 30303 is also forbidden. However, we observe that the patterns 30203 and 30103 also result in considerable ICI, and therefore can significantly impact the performance. Thus, we propose the Page-2B constraint to not only forbid the pattern 303, but also the patterns 30203 and 30103. This can be accomplished by imposing a constraint on page 2 that forbids patterns 111 and 11011, in conjunction with the Grey mapping as shown in Table 3.1. For TLC flash memories, the Page-2B constraint forbids sequences $s_i s_j s_k s_j s_i, i \in \{6,7\}, j \in \{0,1\}, k \in \{2,3,4,5\}$ and hence also effectively suppresses ICI.

The FSM of the Page-2B constraint is shown in Fig. 3.3. The adjacency matrix is

$$\mathbf{D} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \tag{3.3}$$

from which it follows that the capacity of the FSM in Fig. 3.3 is $C_{2B} = 0.849549161 \approx 0.84955$.

Note that since only data on page 2 is encoded, the capacity of the Page-2B constraint for MLC and TLC is $\frac{C_{2B}+1}{2} = 0.92478$ bit/page/cell and $\frac{C_{2B}+2}{3} = 0.94985$ bit/page/cell, respectively. Note that as with the Page-2A constraint, although the Page-2B constraint excludes some non-problematic patterns, high capacities can still be achieved.

Because of the loops in the FSM, the minimal set of the Page-2B constraint has an infinite number of words. As discussed in Chapter 2.1, according to criteria 2 and 3, state 1 is selected as the specified state. The corresponding minimal set S_{2B} is established as $S_{2B} = \{0, 10, 1100, 110100, 11010100, 1101010100, ...\}$. Practical minimal sets representing this constraint are therefore incomplete.

Clearly, the capacity of the Page-2A and Page-2B constraints is higher than that of the Page-1 constraint but with reduced ICI mitigation since some

of the less severe ICI patterns are not forbidden. The choice between the Page-1, Page-2A and Page-2B constraints is up to the designer, depending on the system cost and performance requirement. We note that other constraints are straightforward to derive, model and analyze using this approach.

3.2.3 Results of codes constructed with capacity-approaching code rates

Upper bound on code rate

As previously discussed, the minimal set S_{2A} has a finite number of words while the minimal sets S_1 and S_{2B} have an infinite number of words. When designing practical codebooks using the construction technique outlined in [24], we must construct minimal sets S_1' and S_{2B}' with a finite number of words by eliminating words in the original minimal sets. Since this results in some constraint-satisfying sequences being abandoned, we are not able to achieve capacity with S_1' and S_{2B}', but by retaining an appropriate number of shorter words in the minimal sets, we are able to approach capacity. From (1.9) it is evident that longer codewords have less of an impact on the achievability of capacity, therefore we retain the $|L_M|$ words with the shortest lengths in S_1 and S_{2B} to construct S_1' and S_{2B}'. The upper bound on the code rate for the Page-1 constraint of SLC, $\widetilde{C}_1$, can be determined according to Eq. (1.10). Similar analysis also holds for the Page-2B constraint.

Along with other results, we present the upper bounds on code rate for the Page-1 and Page-2 constraints for various values of $|L_M|$ in Tables 3.3 and 3.5. Similar to the analysis of capacity, for the Page-1 constraint the sum rate for MLC and TLC flash memory is $\frac{\widetilde{C}_1+1}{2}$ and $\frac{\widetilde{C}_1+2}{3}$ respectively. For the Page-2B constraint the sum rate of MLC and TLC flash memory is $\frac{\widetilde{C}_{2B}+1}{2}$ and $\frac{\widetilde{C}_{2B}+2}{3}$ respectively. For the Page-2A constraint, the code rate upper bound is 0.9396 bit/page/cell and 0.9597 bit/page/cell for MLC and TLC respectively.

Table 3.3: Parameters of codes constructed to satisfy the Page-1 constraint. Capacity of SLC: 0.8114, capacity of MLC: 0.9057, capacity of TLC: 0.9371

$\|L_M\|$	4	6	8	10	12	14
N^c	28	46	64	64	67	118
c_{max}	11	15	13	15	17	23
$\widetilde{C}_1$ for SLC	0.7529	0.7947	0.8062	0.8097	0.8108	0.8112
η_{max}	92.79%	97.94%	99.36%	99.79%	99.93%	99.98%
$\bar{R}$	0.7507	0.7917	0.8032	0.8065	0.8069	0.8082
η	92.52%	97.57%	98.99%	99.39%	99.45%	99.61%
$\widetilde{C}_1$ for MLC	0.8764	0.8973	0.9031	0.9048	0.9054	0.9056
η_{max}	96.77%	99.08%	99.71%	99.91%	99.97%	99.99%
$\bar{R}$	0.8754	0.8959	0.9016	0.9033	0.9034	0.9041
η	96.65%	98.91%	99.55%	99.73%	99.75%	99.82%
$\widetilde{C}_1$ for TLC	0.9176	0.9316	0.9354	0.9366	0.9369	0.9371
η_{max}	97.92%	99.41%	99.82%	99.95%	99.98%	99.99%
$\bar{R}$	0.9169	0.9306	0.9344	0.9355	0.9356	0.9361
η	97.87%	99.30%	99.71%	99.83%	99.84%	99.89%

Table 3.4: Parameters of codes constructed to satisfy the Page-2A constraint. Capacity of MLC: 0.9396, capacity of TLC: 0.9597

$\|L_M\|$	3
N^c	27
c_{max}	9
$\bar{R}$ for MLC	0.9373
η for MLC	99.76%
$\bar{R}$ for TLC	0.9582
η for TLC	99.83%

Table 3.5: Parameters of codes constructed to satisfy the Page-2B constraint. Capacity of MLC: 0.92478, capacity of TLC: 0.94985

$\|L_M\|$	4	6	8	10
N^c	28	41	50	73
c_{max}	10	19	22	27
$\widetilde{C}_{2B}$ for MLC	0.91939	0.92429	0.92473	0.92477
η_{max}	99.417%	99.947%	99.995%	99.999%
$\bar{R}$	0.91721	0.92240	0.92280	0.92285
η	99.181%	99.743%	99.786%	99.791%
$\widetilde{C}_{2B}$ for TLC	0.94627	0.94953	0.94982	0.94985
η_{max}	99.623%	99.966%	99.997%	99.999%
$\bar{R}$	0.94477	0.94827	0.94853	0.94857
η	99.465%	99.833%	99.861%	99.865%

Code construction

We now present results for variable-length codes constructed for the constraints introduced above.

For the Page-1 constraint, the Page-2B constraint and for each value of $|L_M|$ shown in Tables 3.3 and 3.5, we list parameters of high-rate codes we have constructed, including the number of codewords in the codebook N^c, and the maximum length of codewords in the codebook c_{max}. As noted above, we also present the upper bounds on code rate for SLC and the sum rates for MLC and TLC flash memories and the highest achievable efficiency η_{max}. We then present the average code rate $\bar{R}$ and the efficiency η of high-rate codes we have constructed. For the Page-2A constraint, since the minimal set is complete with $|L_M| = 3$ words, in Table 3.4 we present parameters of a high rate code that satisfies this constraint. Although codes in these tables have rates very close to capacity, we note that codes with still higher efficiencies can be constructed by expanding the search space.

The code tables for many of the codes referred to in Tables 3.3, 3.4 and 3.5 are too large to list in this thesis, therefore we provide codebooks for some other simpler codes as examples. For example, as noted in Table 3.3, starting

with a minimal set with $|L_M| = 12$, we constructed a code for MLC with average rate 0.9034 that achieves 99.75% of capacity of the Page-1 constraint. The codebook for that code has 67 codewords and the longest codeword has length 17. Since that codebook is too large to list here, in Table 3.6 we list another codebook for the Page-1 constraint that contains only 12 codewords but that still achieves 99.36% of capacity. Note that the codewords in this code are just the words in the minimal set with lengths less than or equal to 13.

Table 3.6: A codebook for the Page-1 constraint that achieves 99.36% of capacity for MLC

Source words	Codewords
0	0
10	110
110	1110
1110	11110
11110	111110
111110	1111110
1111110	11111110
11111110	111111110
111111110	1111111110
1111111110	11111111110
11111111110	111111111110
11111111111	1111111111110

Table 3.7: A codebook for the Page-2A constraint that achieves 98.83% of capacity for MLC

Source words	Codewords
0	0
10	10
11	110

For the Page-2A constraint, although we constructed a code with 27 codewords that achieves 99.76% of capacity for MLC, we choose to present a

Table 3.8: A codebook for the Page-2B constraint that achieves 99.12% of capacity for MLC

Source words	Codewords
00	00
10	10
01	010
110	1100
1110	01100
11110	110100
111110	0110100
1111110	11010100
11111110	011010100
111111110	1101010100
1111111110	01101010100
11111111110	110101010100
111111111110	0110101010100
1111111111110	11010101010100
11111111111110	011010101010100
111111111111110	1101010101010100
1111111111111110	01101010101010100
11111111111111110	110101010101010100
111111111111111110	011010101010101010100
1111111111111111110	1101010101010101010100
11111111111111111110	0110101010101010101010100
11111111111111111111	01101010101010101010100

very simple codebook in Table 3.7 for MLC with rate 0.92855 that achieves 98.83% of capacity. For the Page-2B constraint with $|L_M| = 12$, we have constructed a code for MLC with rate 0.9230 that achieves 99.81% of capacity. The codebook for that code contains 111 codewords with a maximum codeword length of 32. Due to space limitations, we list a different codebook in Table 3.8 that has only 23 codewords and a maximum codeword length of 23 but still achieves 99.12% of capacity.

Recall that the $d = 1$ binary RLL code proposed in [47] has rate $0.667, 0.833$ and 0.889 for SLC, MLC and TLC flash memories. When we consider our codes, we note that not only do our codes have higher average code rates but that they also ensure that more ICI inducing patterns will not occur. These benefits arise from a more explicit description of the constraints and utilization of variable-length constrained sequence codes.

Lastly, we note that it is straightforward to extend our code construction approach to the design of constrained sequence codes for other constraints of importance in emerging flash memories in which each cell can store more than three bits.

3.2.4 Error control inherent in the constrained sequence codes

Potential error control properties of constrained sequence codes can be exploited during their decoding, and a well-known problem that arises during decoding of constrained sequence codes is error propagation. As we outline in this section, there is inherent error control capability in our proposed Page-1 and Page-2 constrained sequence codes. We now discuss how we can take advantage of the error control capability inherent within these codes to improve overall performance.

The decoder of the Page-1 and Page-2 constrained sequence codes is implemented using a codebook. The words in the minimal set satisfy the prefix condition, as do codewords in its partial extensions [25]. Therefore, the decoding process can be implemented with an *increasing block decoder*. This

decoder keeps a list of codeword lengths $c_i, c_i \in [c_{min}, c_{max}]$ where c_{min} is the minimum length of codewords. The size of the decoding block increases from c_{min} to c_{max}. As the received sequence enters the decoder, the decoding block recognizes the next codeword with length starting from the initial size c_{min}. If the first c_{min} bits in the received bit sequence do not comprise a codeword, the block size increases to the next value of c_i and decides if the corresponding received bit sequence is a codeword or not. This process is repeated until the size reaches c_{max}. For illustration purposes we take the code in Table 3.6 as an example; the detailed decoding procedure for that codebook is demonstrated in Algorithm 2.

Next we introduce the implicit error correction capabilities in the Page-1 and Page-2 constraint. In Page-1 constrained codes, the pattern 010 is forbidden. At the receiver, the occurrence of the forbidden pattern is possible only if one or more errors have occurred during transmission. Note that if a codeword is corrupted such that pattern 010 occurs at the receiver, the receiver does not recognize it as a valid codeword, as shown in Algorithm 2. Similarly, in the Page-2A and Page-2B constraints, if a codeword is corrupted such that the pattern 111 (or 111 and 11011 in Page-2B constrained systems) occurs at the receiver respectively, the receiver does not recognize it as a valid codeword and indicates that an error event has occurred.

We now discuss how the decoder could implement a simple error correction algorithm to correct many such errors. In this chapter, we consider the most probable case when errors due to ICI occur when a state corresponding to a lower voltage level is detected as another state corresponding to a higher voltage level. The detector detects and compares the voltages representing the bit sequences 010 (or 111 and 11011) to find the error positions and complement incorrect bit values for the Page-1 (and Page-2) constraint. We denote this as *layer-I error correction*. For MLC and TLC cells, due to the voltage distribution of cells, we set a different detection threshold for some bits on the constrained coded page that can reduce the number of erroneous detected bits, which we denote as *layer-II error correction*. We note that for SLC flash memories the proposed layer-I error

Algorithm 2 Increasing block decoding process for the Page-1 constrained code in Table 3.6

Initialize:

$c_{min} = 1$, $c_{max} = 13$

N_r: the total number of received bits

r_i: the i-th bit of the received coded sequence at the decoder

$i = 1$

Start:

```
while i ≤ N_r do
    if r_i==0 then
        Output source word 0
        i = i + 1;
    else
        if r_{i+1}==1 then
            i = i + 2;
            while i ≤ c_max do
                if r_i==0 then
                    Output the corresponding source word
                    i = i + 1;
                    break;
                end if
                i = i + 1;
            end while
        else
            Indicate error event has happened because the pattern 010 has occurred
        end if
    end if
end while
```

correction can be used. For MLC and TLC cell flash memories, the layer-I error correction and layer-II error correction can be used individually or simultaneously. In the following we consider the most probable situation of at most one incorrectly detected bit in a codeword.

Page-1 constraint

Layer-I error correction As a result of the Gray mapping used in SLC, MLC and TLC cells, the only errors that ICI causes during the reading process of Page-1 constrained coded sequences are that coded logic ones are detected as coded logic zeros, whether in SLC, MLC or TLC cells. In the increasing block decoding process introduced above, consider the situation where the decoder recognizes the pattern 010, i.e. $r(i) = 0, r(i-1) = 1, r(i-2) = 0$, and the corresponding detected voltage values are $v(i), v(i-1), v(i-2)$, respectively. Denote the read threshold voltage of page 1 as v_{th}, and recall that a logic zero is stored as a high voltage level. We identify the position of erroneous bit j using soft information from the detector as follows:

$$j = \begin{cases} i & \text{if } v(i-2) - v_{th} \geq v(i) - v_{th} \\ i-2 & \text{if } v(i-2) - v_{th} < v(i) - v_{th} \end{cases} \tag{3.4}$$

That is, we check the detected voltages of the two positions corresponding to logic zeros, and select the position whose voltage is closer to the threshold as the position of the erroneous bit and complement its value to a logic one since it is more likely to be the erroneous bit due to ICI. The decoder then returns to the end of the last codeword that was decoded without errors and resumes the decoding process.

Layer-II error correction We introduce the layer-II error correction for MLC and TLC cells, which exploits the fact that the voltage distributions of MLC and TLC cell flash memories are much narrower than in SLC flash memory for all states except state s_0 [49, 50, 51, 52, 53, 54, 55, 56].

As a result of the Gray mapping, for a cell with q bits, the most probable errors on page 1 are due to incorrect detection of bits in state $s_{2^{q-1}-1}$, which is

incorrectly identified as state $s_{2^{q-1}}$. This corresponds to a coded logic one being erroneously detected as a logic zero. Denote u_i and σ_i as the mean value and standard deviation of the voltage distribution of state i. Owing to the much narrower distributions of state $s_{2^{q-1}-1}$ and $s_{2^{q-1}}$ than for s_0 [49, 50, 51, 52, 53, 54, 55, 56], in some flash memory structures the voltage distributions give rise to distance between v_{th} and $u_{2^{q-1}} - t\sigma_{2^{q-1}}$, where t is a parameter describing the distance between distributions. Assuming that the mean value $u_{2^{q-1}}$ and the standard deviation $\sigma_{2^{q-1}}$ of state $s_{2^{q-1}}$ can be accurately estimated, then the received symbols with voltages that fall in the range $(v_{th}, u_{2^{q-1}} - t\sigma_{2^{q-1}})$ should not be detected as state $s_{2^{q-1}}$ since they are likely to be symbols of state $s_{2^{q-1}-1}$ whose voltages have been shifted by ICI. Therefore, we establish a new threshold value that is closer to $u_{2^{q-1}} - t\sigma_{2^{q-1}}$ than v_{th}. During the detection process, we compare the voltages of bits interpreted as 0 on page 1 with the new threshold. If a voltage is smaller than the new threshold, the corresponding bit is corrected to value 1, and the decoder returns to the end of the last decoded codeword where decoding resumes.

Page-2 constraint

Layer-I error correction In Page-2A constrained codes, the pattern 111 is forbidden. Whenever the decoder receives the patten 111, it indicates that errors have occurred. In accordance with the Gray mapping, pattern 111 occurs when coded bit zero is incorrectly detected as logic value one on page 2. Note that there are two read threshold voltages on page 2. We denote the higher threshold voltage as v_h and the lower threshold as v_l such that a voltage $v(i)$ is detected as a logic zero when $v_l < v(i) < v_h$. It is obvious that pattern 111 occurs only when the bit on Page 1 is detected as zero. Therefore, we identify the position of erroneous bit j as follows. When the decoder recognizes pattern 111, i.e. $r(i) = 1, r(i-1) = 1, r(i-2) = 1$ with the corresponding detected voltage values $v(i), v(i-1), v(i-2)$, the erroneous detected bit position is determined to be the one whose detected voltage is closest to v_h.

For the Page-2B constraint, apart from pattern 111, pattern 11011 is also forbidden. Therefore, in addition to the position selection criterion above, we establish another criterion to determine the erroneous detected bit position j when pattern 11011 is recognized, i.e. $r(i) = 1, r(i-1) = 1, r(i-2) = 0, r(i-3) = 1, r(i-4) = 1$ with the corresponding detected voltage values $v(i), v(i-1), v(i-2), v(i-3), v(i-4)$. Assume the most likely situation of at most one bit error in each codeword. The bit with value 0 is not regarded as the erroneous bit otherwise the original sequence would be 11111 which violates the Page-2B constraint. Therefore, the four potential erroneous bit positions are $i, i-1, i-3, i-4$. Similar to the Page-2A constraint, it is assumed that the position of the erroneous bit is the one whose detected voltage is closest to v_h.

After identifying the erroneous bit position, we correct the value of the bit from 1 to 0. The decoder returns to the end of last decoded codeword and resumes the increasing block decoding process.

Layer-II error correction In accordance with the Gray mapping that is used on page 2, a coded bit with value 0 on page 2 could be corrupted to value 1 when the bit on page 1 is detected as 0. However, as noted earlier, because of the much narrower voltage distribution in high voltage states, we can establish a new threshold to take into account voltage shifts due to ICI for MLC and TLC flash memories.

Consider page 2. For a cell with q bits, if the corresponding bit on page 1 is detected as a logic one, then the most probable error that can occur is when state $s_{2^{q-2}-1}$ is detected as state $s_{2^{q-2}}$. Since, as discussed in [49, 50, 51, 52, 53, 54, 55, 56], there is distance between v_l and $u_{2^{q-2}} - t\sigma_{2^{q-2}}$, the detection threshold can be set to be closer to $u_{2^{q-2}} - t\sigma_{2^{q-2}}$ than v_l to deal with voltage shift caused by ICI. If the corresponding bit on page 1 is detected as a logic zero, then the most probable error occurs when state $s_{2^q-2^{q-2}-1}$ is detected as state $s_{2^q-2^{q-2}}$. Similarly since there is distance between v_h and $u_{2^q-2^{q-2}} - t\sigma_{2^q-2^{q-2}}$, the detection threshold can be set closer to $u_{2^q-2^{q-2}} - t\sigma_{2^q-2^{q-2}}$. As bits are detected, we compare the voltages of bits with value 1 on page 2 in the

sequence with the new thresholds. If this results in a different logic value, the corresponding bit is corrected to 0 and decoding resumes from the beginning of the codeword.

Results

We now demonstrate the effect of taking advantage of the inherent error control capabilities introduced above. We consider ICI as the only source of impairment, and we consider ICI only within the same word-line to better observe the effect of constrained coding with multi-page programming. As suggested by [47], the threshold voltage shift Δv_i of the i-th cell on a specific word-line is modeled as

$$\Delta v_i = \alpha(\Delta v_{i-1} + \Delta v_{i+1}) \tag{3.5}$$

where α is the coupling ratio and Δv_{i-1} and Δv_{i+1} are the voltage shifts of the neighboring cells of the i-th cell. For an SLC flash block, the voltage distribution before ICI is set to $N(-1, 0.25^2)$ when programming bit one and $N(1, 0.25^2)$ when programming bit zero, which is consistent with [47].

The BER performance of the uncoded scheme and the Page-1 constrained coding scheme for SLC flash memories is presented in Fig. 3.4. The horizontal axis represents a range of values for the coupling ratio α. We use the codebook in Table 3.6 for the constrained coding scheme. We consider only Layer-I error correction, and we present the BER before decoding, i.e. the raw BER at the output of the detector, and the BER after decoding. As shown in Fig. 3.4, the BER performance improves with smaller α. Generally, constrained sequence codes result in error propagation after decoding. However, due to the error correction capability of our constrained code, some detection errors are corrected and error propagation is limited. In the simulation we consider blocks of size 512, 128 and 64. The results show that there is less error propagation with smaller source block sizes because error propagation is truncated at the end of each block. The BER of the constrained coding scheme with source block sizes of 64 and 128 is comparable to the BER before decoding.

We also demonstrate the BER performance of the Page-2A and Page-2B constraints before and after decoding for MLC. For MLC flash memories, the voltage distribution of state s_0 is set to $N(1.1, 0.2^2)$ and the distributions of states s_1, s_2, and s_3 are set to $N(2.7, 0.03^2)$, $N(3.3, 0.03^2)$ and $N(3.9, 0.03^2)$ respectively. The codebook used for the Page-2A constrained code is the one shown in Table 3.7, and for the page-2B constraint we use a codebook with rate 0.9163 bit/page/cell whose size is smaller than the code listed in Table 3.8 and achieves 99.08% of capacity. Both layer-I and layer-II error correction are used. In layer-II error correction, the detection threshold is set to 2.5 rather than $v_l = 1.9$, and we consider a source block size of 256. It can be seen from Fig. 3.5 that with layer-I and layer-II error correction, error propagation is reduced and lower BER is obtained after constrained decoding. We also observe that layer-II error correction is independent of block size since false correction occurs very rarely, especially with moderate t values. Note that the range of values of α considered in Fig. 3.5 is smaller than the range in Fig. 3.4 because ICI is more severe in MLC than in SLC.

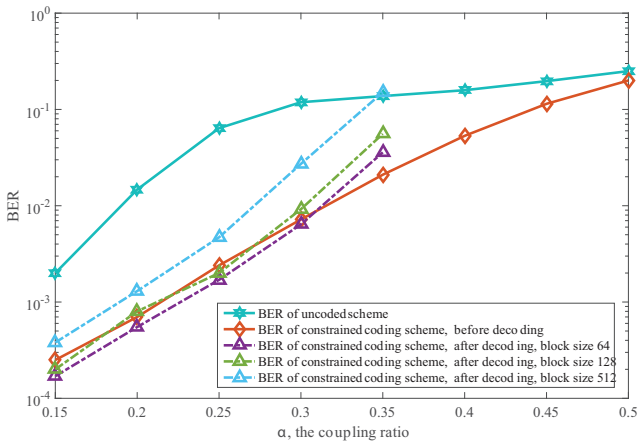

Figure 3.4: BER performance of uncoded scheme and the Page-1 constrained coding scheme for SLC flash memories

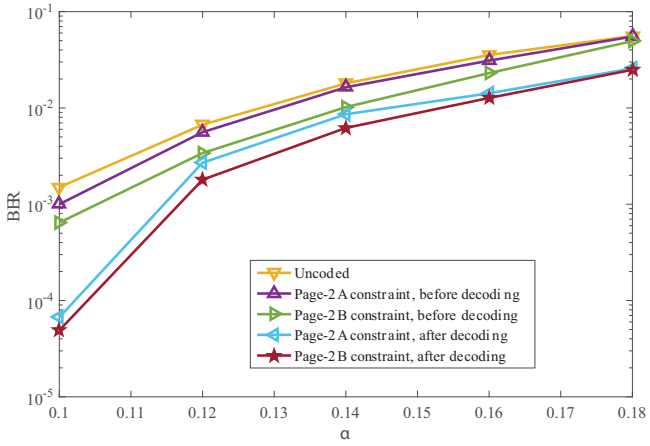

Figure 3.5: BER performance of uncoded scheme and the Page-2 constrained coding schemes for MLC flash memories

3.2.5 Concatenation of constrained sequence codes with error control codes

When integrating error control codes with constrained sequence codes, the concatenated coding scheme shown in Fig. 1.1 is often used. The source bits are first sent to an ECC encoder, and then are sent to an encoder of a constrained sequence code so that the coded sequence that is forwarded to the channel satisfies the channel constraint. At the receiver side, the received sequence is first decoded by a constrained sequence decoder, and then by an ECC decoder.

In this section we simulate the system described in Fig. 1.1 and present the overall bit error rate (BER) performance for various flash memory systems. We present results for the proposed Page-1, Page-2A and Page-2B constrained sequence codes with inherent error control capabilities for SLC and MLC flash memories. We employ Bose, Chaudhuri, and Hocquenghem (BCH) codes as the ECC code.

Results for the Page-1 constraint

First, to demonstrate the ICI mitigation effect achieved with the proposed constrained codes, we compare the voltage distributions of a scheme using

only a BCH code and a system using only a constrained sequence code. We use the code that satisfies the Page-1 constraint for SLC given in Table 3.6 with rate 0.7999. To compare, we use a (511,412) BCH code with rate 0.806. The encoding processes are illustrated in Fig. 3.6. We evaluate the voltage distribution of cells in each flash memory block assuming a coupling ratio α of 0.2.

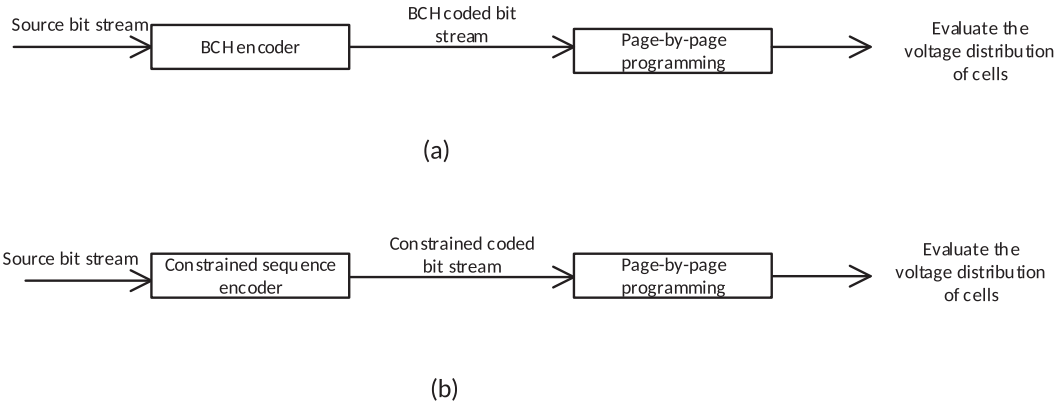

Figure 3.6: The encoding process and voltage evaluation of (a) the conventional scheme and (b) the constrained coding scheme

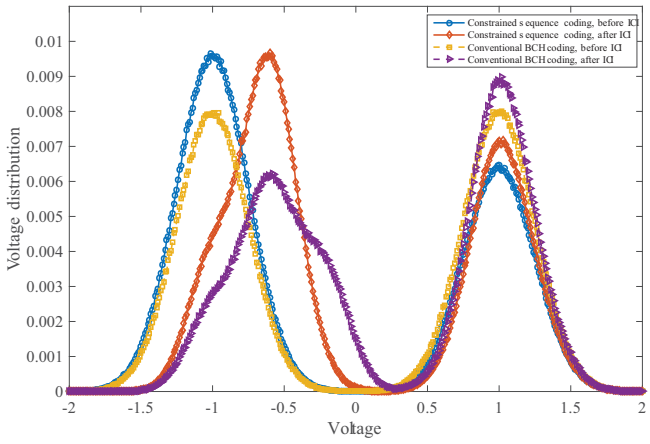

Figure 3.7: The voltage distribution of conventional scheme and Page-1 constrained coding scheme

The results of this evaluation are shown in Fig. 3.7. Threshold zero is used to distinguish between logic zero and logic one symbols. It can be seen from this figure that ICI causes severe voltage shifts when only error control

coding is used. This will result in incorrect detection of coded bits at the receiver, which may exceed the error correction capability of the ECC code, especially with larger α. In contrast, the constrained coding scheme effectively limits ICI, and the occurrence of incorrect detection at the receiver is limited.

We now consider the concatenation of ECC codes and constrained codes, and compare the overall performance with that of a system using ECC with the same code rate. The concatenated scheme is shown in Fig. 3.8. We compare the concatenation and the BCH-only schemes with approximate code rates 0.7 and 0.8. The source bit stream is divided into blocks of size 358, and is encoded with a (412,358) shortened BCH code. The BCH coded stream is then encoded with the Page-1 constrained code in Table 3.6 to generate 515 constrained coded bits per block on average. The overall rate of the (515,412,358) BCH-CS concatenated code is 0.695. An interleaver and a deinterleaver are used between the BCH encoder and the constrained encoder, and between the BCH decoder and the constrained decoder, respectively. We use a simple block interleaver that write bits row-by-row and extract bits column-by-column. The deinterleaving process helps to translate the burst errors at the output of the constrained decoder that might arise due to error propagation into more random errors for the BCH code to correct effectively. In the ECC-only system we used a (511,358) BCH code with code rate 0.701. As shown in Fig. 3.9, the BER of the BCH-CS scheme outperforms the BCH-only scheme. With smaller α, the error rates of both schemes decrease dramatically.

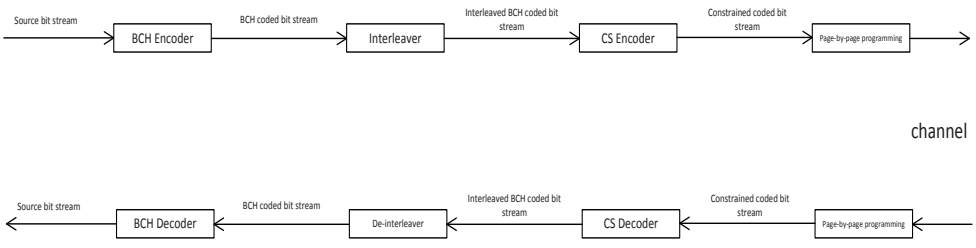

Figure 3.8: The encoding and decoding process of the concatenated coding scheme

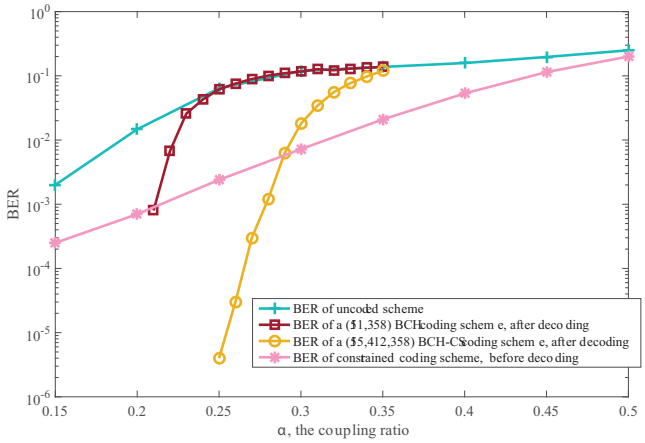

Figure 3.9: BER performance of the BCH coded scheme with rate 0.701, and the concatenated coded scheme of a BCH code and Page-1 constrained code with average rate 0.695 for SLC flash memories

We also designed a (511,409,391) BCH-CS scheme which has the higher overall rate of 0.765. A rate 0.771 (511,394) BCH code is simulated for purposes of comparison. BER results of the concatenated scheme and the BCH-only scheme are shown in Fig. 3.10. It is clear from these results that the concatenated scheme performs better than the BCH-only scheme. In the BCH-CS scheme, if we allocate the minimum possible number of redundant bits for the BCH code, we could design a rate 0.783 (511,409,400) BCH-CS scheme. The comparison of this code with a rate 0.789 (511,403) BCH code is also shown in Fig. 3.10. It can be seen that, since the error correction capability of the BCH-CS scheme is limited, its BER curve flatter for low α when compared to the BCH-only scheme. We note that this is a limiting case of the BCH-CS scheme and we recommend that additional redundant bits be allocated to the BCH code to provide sufficient error correction.

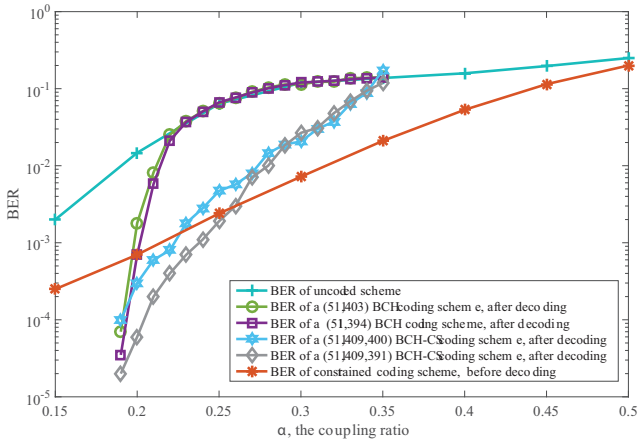

Figure 3.10: BER performance of the BCH coded scheme with rate 0.771, and the concatenated coded scheme of a BCH code and Page-1 constrained code with average rate 0.765 for SLC flash memories

As shown in Fig. 3.4, with smaller source blocks the error propagation is alleviated. Therefore, larger performance gain can be obtained with the BCH-CS scheme with smaller source block size, compared with the BCH-only scheme. To demonstrate, we compare a rate 0.768 (259,207,199) BCH-CS scheme where the fewest possible redundant bits are allocated for the BCH code. We simulated a rate 0.780 (255,199) BCH code as comparison. It can be seen in Fig. 3.11 that, with shorter block size, the error propagation of the constrained decoder is alleviated, and compared with the BCH-only scheme, larger performance gain can be achieved with concatenation with even a very weak BCH code.

We now consider lower code rates and compare a rate 0.51 (121,97,62) BCH-CS scheme and a rate 0.45 (127,57) BCH-only scheme in Fig. 3.12, a rate 0.48 (255,207,123) BCH-CS scheme and a rate 0.48 (255,123) BCH-only scheme in Fig. 3.13, a rate 0.514 (504,403,259) BCH-CS scheme and a rate 0.507 (511,259) BCH-only scheme in Fig. 3.14. It is again shown that the BCH-CS schemes outperform the BCH-only schemes. The outer BCH codes in the BCH-CS schemes are able to correct errors that exist after CS decoding due to their lower code rates.

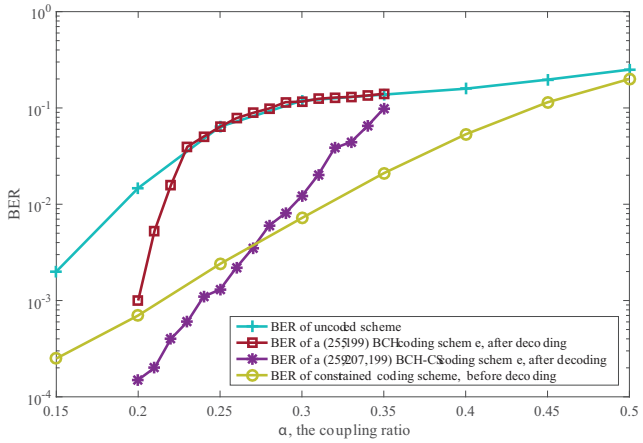

Figure 3.11: BER performance of the BCH coded scheme with rate 0.78, and the concatenated coded scheme of a BCH code and Page-1 constrained code with average rate 0.768 for SLC flash memories

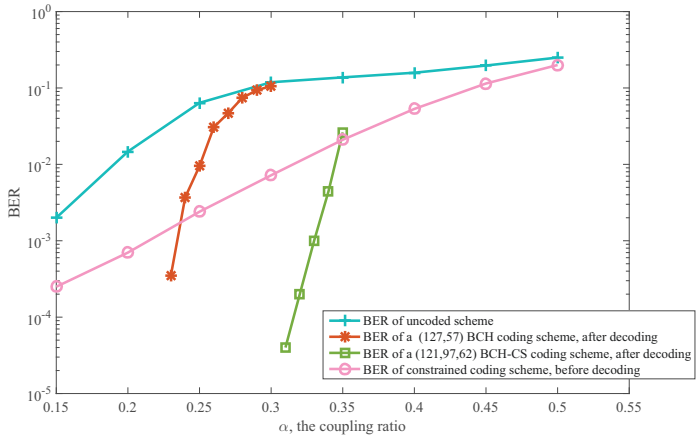

Figure 3.12: BER performance of the BCH coded scheme with rate 0.45, and the concatenated coded scheme of a BCH code and Page-1 constrained code with average rate 0.51 for SLC flash memories

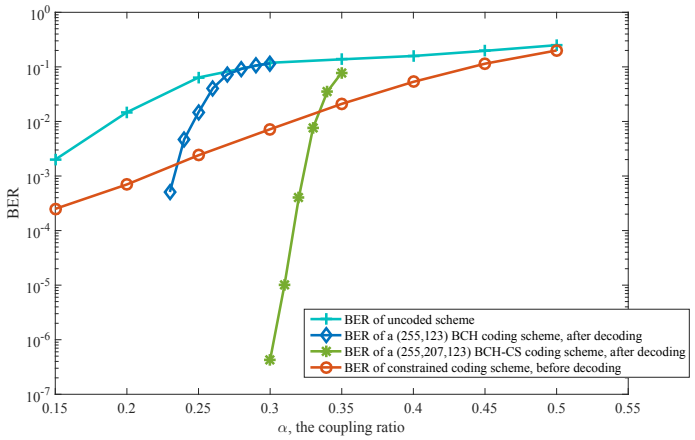

Figure 3.13: BER performance of the BCH coded scheme with rate 0.48, and the concatenated coded scheme of a BCH code and Page-1 constrained code with average rate 0.48 for SLC flash memories

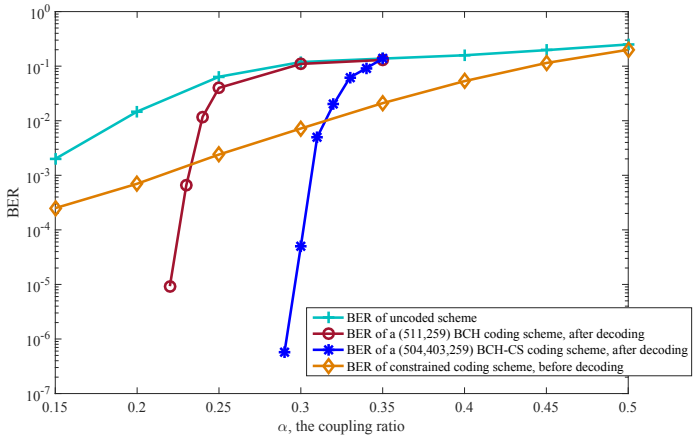

Figure 3.14: BER performance of the BCH coded scheme with rate 0.507, and the concatenated coded scheme of a BCH code and Page-1 constrained code with average rate 0.514 for SLC flash memories

Results for the Page-2A and Page-2B constraints

We first present the voltage distribution of two schemes in MLC flash memories, the BCH coded scheme and the Page-2B constrained coded scheme. From Fig. 3.15 we see that the ICI mitigation effect is not obvious, which is in agreement with Fig. 3.5 where the BER improvement before constrained decoding is limited compared with the uncoded scheme. However, we note

that when constrained sequence codes are concatenated with BCH codes, the effect of constrained sequence coding is to mitigate the ICI which reduces the number of bit errors for the BCH codes to correct.

We now compare the concatenated coding scheme with other conventional schemes in MLC. We consider a BCH-only scheme with no extra error correction that uses a rate 0.806 (511,412) BCH code on both page 1 and page 2. For the Page-2A constraint, 439 source bits are encoded into 511 BCH coded bits with a shortened (511,439) BCH code on page 1. On page 2, 375 source bits are encoded into 438 BCH coded bits with a shortened (438,375) BCH code. The BCH coded bits are then encoded with the codebook in Table 3.7 into 511 coded bits on average, resulting in a rate $\frac{439+375}{511+511} = 0.796$ (511,439,511,438,375) BCH-CS scheme. Block interleavers and deinterleavers are used between the ECC and constrained sequence coding procedures. Similarly, a rate 0.801 (511,448,511,
425,371) BCH-CS scheme for the Page-2B scheme is simulated based on the codebook introduced above. We also simulate the performance of a rate 0.803 (511,457,511,409,364) BCH-CS scheme for the Page-1 constraint.

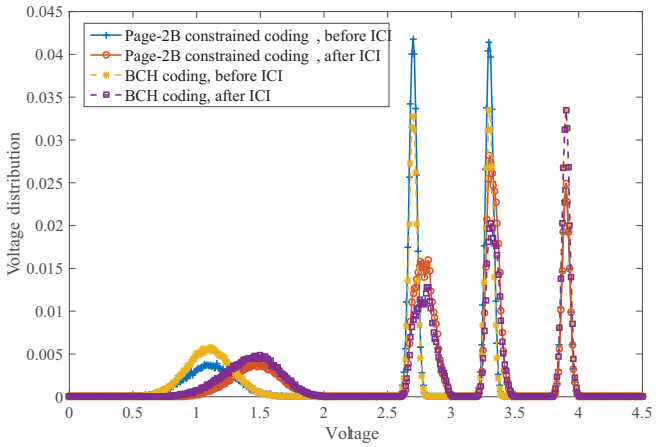

Figure 3.15: The voltage distribution of ECC-only scheme and Page-2B constrained coding scheme

We compare all schemes above with a rate 0.797 (511,493,510,340,322) BCH-RLL concatenated scheme, where a (1,7) RLL code from Table II and III in [47] is used. On page 1, 493 source bits are encoded into 511 BCH coded

bits with a rate (511,493) BCH code. On page 2, 322 source bits are first encoded into 340 BCH coded bits with a shortened (340,322) BCH code, and then are encoded into 510 (1,7) RLL constrained coded bits.

It can be seen in Fig. 3.16 that our proposed BCH-CS scheme outperforms the BCH-RLL scheme and the BCH-only scheme. In this case, the code rate of the RLL code is so low that the number of redundant bits in the BCH code of the BCH-RLL scheme is such that the BCH-RLL scheme has limited error correction capability, and hence performs even worse than the BCH-only scheme. We note that although the layer-II error correction is proposed for protection of CS codes, it can also be used for the BCH-only scheme. We present the BER performance of the BCH-only scheme with this extra error correction in Fig. 3.16. It is seen that with BER of interest (smaller than 10^{-2}), the BCH-CS schemes perform better since in this region the ICI reduction resulted from CS codes leads to fewer errors for the outer BCH code to correct.

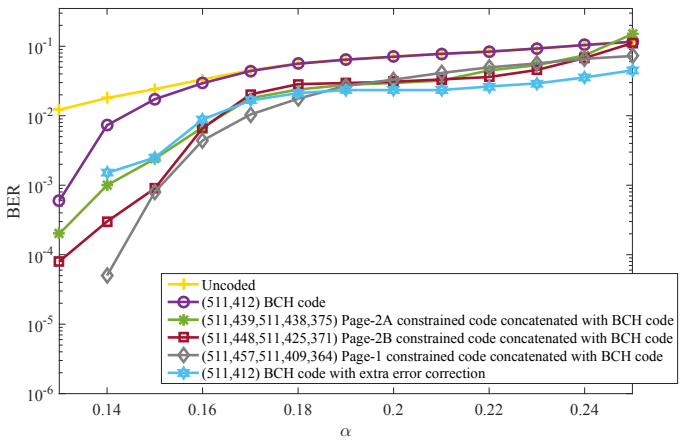

Figure 3.16: BER performance of the BCH coded scheme with rate 0.806, the BCH-RLL scheme with rate 0.797, the concatenated coded scheme of a BCH code and Page-2A constrained code with average rate 0.796, the concatenated coded scheme of a BCH code and Page-2B constrained code with average rate 0.801, and the concatenated coded scheme of a BCH code and Page-1 constrained code with average rate 0.803 for MLC flash memories.

We now compare different schemes with higher average code rates in MLC. We design a rate 0.858 (511,475,511,438,402) BCH-CS scheme for the Page-2A scheme, a rate 0.854 (511,484,511,425,389) BCH-CS scheme for the Page-2B scheme, and a rate 0.856 (511,484,511,409,391) BCH-CS scheme for the Page-1 scheme. A rate 0.859 (511,439) BCH-only scheme is used as comparison. It can be seen in Fig. 3.17 that all BCH-CS schemes outperform the BCH-only scheme. For comparison with the BCH-only scheme with extra error correction, a similar trend is observed as in Fig. 3.16, except that the advantage of BCH-CS schemes become less obvious since the code rate of the outer BCH code is higher, which results in weaker error correction capability.

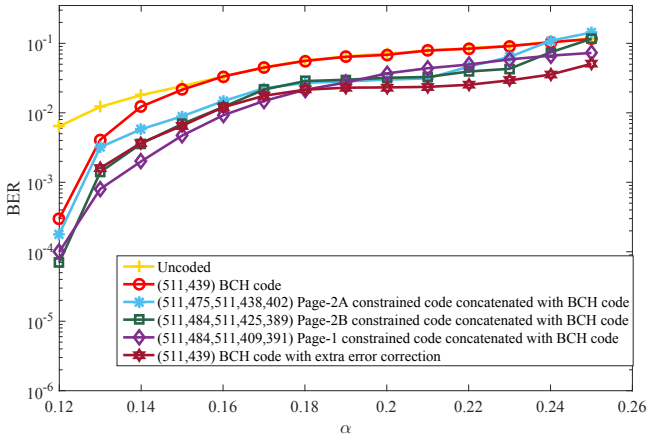

Figure 3.17: BER performance of the BCH coded scheme with rate 0.859, the concatenated coded scheme of a BCH code and Page-2A constrained code with average rate 0.858, the concatenated coded scheme of a BCH code and Page-2B constrained code with average rate 0.854, and the concatenated coded scheme of a BCH code and Page-1 constrained code with average rate 0.856 for MLC flash memories.

In Fig. 3.18 we present the performance of a rate 0.506 BCH-only scheme with layer-II error correction, a rate 0.506 (511,259,512,439,259) Page-2A BCH-CS scheme, and a rate 0.512 (511,259,506,421,259) Page-2B BCH-CS scheme. Again it is shown that the CS-BCH schemes outperform the BCH-only scheme with BERs of interest.

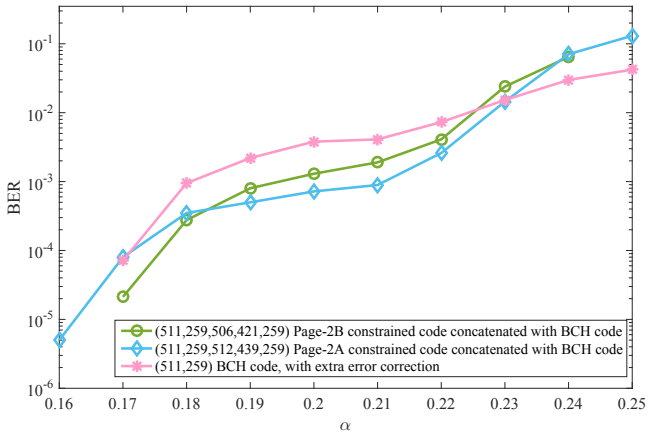

Figure 3.18: BER performance of the BCH coded scheme with rate 0.506, the concatenated coded scheme of a BCH code and Page-2A constrained code with average rate 0.506, the concatenated coded scheme of a BCH code and Page-2B constrained code with average rate 0.512 for MLC ash memories.

3.3 Pearson codes for cell leakage [21]

In this section we discuss the use of the proposed single-state code construction technique to construct capacity-approaching variable-length Pearson codes that are immune to channel gain and/or offset mismatch at the demodulator that cause performance loss in modern data storage and communication systems. The contents of this section was published in [21].

3.3.1 Background

As has been described, in flash memory devices, information is represented by modulating the amount of charge present on floating gates. However, cell leakage — the phenomenon that charge may leak from these gates — causes drift of threshold voltages. Fingerprints on optical disks may cause channel gain and offset variations of the signal at the receiver [57, 58], therefore effective coding and signal processing methods that deal with gain and offset mismatch of the received signals are critical for recovery of source information. Pearson codes, a class of constrained sequence codes that can

be decoded with Pearson-distance-based detection, which is immune to channel gain and offset mismatch, were proposed in [58] and [59]. Practical systematic Pearson codes were proposed in [60] where the authors studied fixed-to-fixed (FF) and variable-to-fixed (VF) mappings from source words to codewords. It is shown in [60] that the VF codes have less redundancy than their FF counterparts. In the following sections, we explore variable-to-variable (VV) mapping from source words to codewords to improve the efficiency and reduce the implementation complexity of Pearson codes. Based on the variable-length code construction technique proposed in Section. 2.1, we construct VV Pearson codes, analyze their performance, and discuss their implementation.

3.3.2 Finite state machine description

We present the FSM for binary and non-binary Pearson codes. Pearson codes can be regarded as a type of T-constrained code where T pre-defined symbols each appear at least once in each codeword [57]. As discussed in [60], a known construction method for Pearson codes is to ensure that every q-ary codeword contains at least one symbol zero and one symbol one. This constraint is described by the finite state machine (FSM) description of q-ary Pearson codes in Fig. 3.19 which guarantees that at least one symbol zero and one symbol one occur in each sequence of length G. When $q = 2$, this FSM reduces to the one shown in Fig. 3.20. It is straightforward to verify that, regardless of the state from which one starts and ends in these FSMs, every length-G sequence generated according to these figures satisfies the Pearson constraint. In this section we focus on binary Pearson codes, but note that it is possible to apply our code construction technique to q-ary Pearson codes based on the FSM in Fig. 3.19 rather than the FSM in Fig. 3.20.

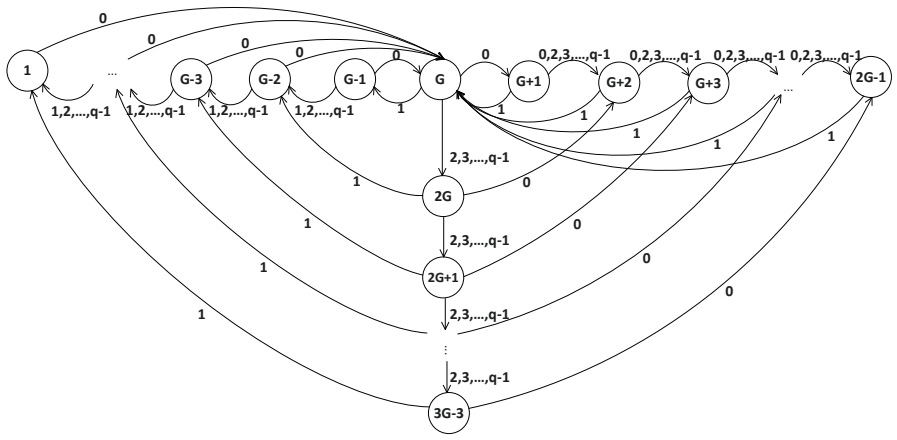

Figure 3.19: FSM for q-ary Pearson codes.

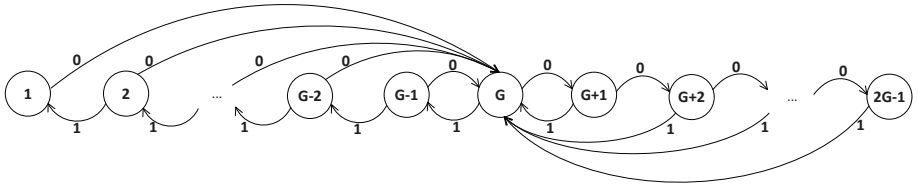

Figure 3.20: FSM for binary Pearson codes.

The FSM in Fig. 3.20 has $2G-1$ states. As $G \to \infty$, the FSM of binary Pearson codes extends to an infinite number of states. As outlined in Chapter 1, the capacity C of this FSM is equation (1.6)

$$C = \log_2 \lambda_{\max} \tag{3.6}$$

where $\lambda_{\max}$ is the largest real root of the characteristic equation $\sum_{i=2}^{\infty} 2\lambda^{-i} = 1$. By solving the characteristic equation for $\lambda_{\max}$ and using this value in (3.6), we obtain $C = 1$ which agrees with the observation that the sequence becomes unconstrained as $G \to \infty$.

In practical construction of Pearson codes, we truncate this FSM to finite G and construct the code based on a finite number of states and a minimal set M that consists of a finite number of words in which the longest codeword is of length $l_{\max}$. The maximum possible code rate $\widetilde{C}_M$ is determined as (3.7), and

the corresponding maximum possible efficiency $\widetilde{\eta} = \widetilde{C}_M/C$. Again we denote L_M as the set of lengths of words in M. $\widetilde{C}_M$ is evaluated as equation (1.10)

$$\widetilde{C}_M = \log_2 \widetilde{\lambda}_{\max} \tag{3.7}$$

where $\widetilde{\lambda}_{\max}$ is the largest real root of the characteristic equation $\sum_{l_i \in L_M} \lambda^{-l_i} = 1$ and l_i is the length of the i-th word in M [18, 19]. As shown in Chapter 2.1, $\widetilde{C}_M = f(l_{\max})$ is a concave increasing function with respect to $l_{\max}$. We define the redundancy of the code as

$$\gamma = \sum_{o_i} 2^{-s_i} o_i \times (1 - \bar{R}). \tag{3.8}$$

3.3.3 Code construction

As noted above, our construction of Pearson codes is based on the technique proposed in Section 2.1. It includes:

- Selecting the state upon which to construct the minimal set. Based on the analysis in Section 2.1, it can be shown that in the FSM of Fig. 3.20, state G will generate minimal sets with the highest $\widetilde{C}_M$ for $l_{\max} \in [2, \infty)$, therefore we specify this state for construction of the minimal set.

- Constructing the minimal set by enumerating all words of length no greater than $l_{\max}$ that can occur when state G is exited and subsequently re-entered. It is straightforward to verify that when state G is the specified state, the words in this minimal set M have lengths $L_M = \{2, 2, 3, 3, 4, 4, ..., l_{\max} - 1, l_{\max} - 1, l_{\max}, l_{\max}\}$.

- Forming partial extensions of the words in the minimal set, performing NGH coding over the minimal set and these partial extensions to generate different codes with one-to-one correspondence between variable-length source words and variable-length codewords, and selecting the code with highest efficiency. However, to simplify analysis of these codes here, we do not consider codes constructed based on partial extensions but consider only the simplified case when the

minimal set M is indeed the set of codewords in the codebook. Consideration of partial extensions will result in higher efficiency codes at the cost of increased implementation complexity.

Table 3.9: Codebook of a binary Pearson code with $l_{\max} = 10, \bar{o} = 2.9961, \bar{R} = 0.9987, \gamma = 0.0039$

Source word	Codeword	Source word	Codeword
00	10	111101	000001
01	01	1111100	1111110
100	110	1111101	0000001
101	001	11111100	11111110
1100	1110	11111101	00000001
1101	0001	111111100	111111110
11100	11110	111111101	000000001
11101	00001	111111110	1111111110
111100	111110	111111111	0000000001

Example (A codebook with $l_{\max} = 10$): Using the construction process above we construct a code with $l_{\max} = 10$ and present the codebook in Table 3.9. We use as codewords all words of length less than or equal to 10 that exit and re-enter state G in Fig. 3.20. We then perform NGH coding to construct the source words and source-to-codeword mapping in the codebook. Codewords in this code have an average length of $\bar{o} = 2.9961; \bar{R} = 0.9987$, and $\gamma = 0.0039$. Detailed performance analysis of this construction technique is given in the next section.

3.3.4 Performance analysis

Code rate and redundancy

We now analyze the average length of codewords $\bar{o}$, average code rate $\bar{R}$, and redundancy γ of VV Pearson codes that can be constructed using the technique outlined above. First, we introduce and prove the following propositions.

Proposition 3.1 In VV Pearson codes constructed based on a minimal set, the set of codeword lengths $L_M = \{2, 2, 3, 3, 4, 4, ..., l_{\max} - 1, l_{\max} - 1, l_{\max}, l_{\max}\}$ has the corresponding source word lengths $L_S = \{2, 2, 3, 3, 4, 4, ..., l_{\max} - 2, l_{\max} - 2, l_{\max} - 1, l_{\max} - 1, l_{\max} - 1, l_{\max} - 1\}$.

Proof. Consider the NGH coding process that involves recursively merging the two largest words on a Huffman tree. With $\bar{R}$ initially set to C, it is straightforward to verify that applying the merging rule given by equation (1.14) to the set of codeword lengths L_M will result in the set of source word lengths L_S. It is also straightforward to show that as $\bar{R}$ converges to its final value, further iterations will not change this set of word lengths because the only possible set of source word lengths that would allow an unconstrained source sequence to be mapped to codewords with lengths L_M with a higher code rate is $L'_S = \{2, 2, 3, 3, 4, 4, ..., l_{\max} - 1, l_{\max} - 1, l_{\max} - 1, l_{\max}\}$. This set of source word lengths is not possible because with L'_S, the two longest source words with length $l_{\max} - 1$ and $l_{\max}$ must correspond to the two codewords with length $l_{\max}$ and $l_{\max}$, however, the construction process ensures that source words corresponding to these two codewords must have the same length. Therefore L'_S does not exist, source word lengths converge to L_S, and the proposition above holds. □

Proposition 3.2 As $l_{\max} \to \infty$, the average length of codewords $\bar{o}$ in a binary Pearson codebook is 3, and the average length of source words $\bar{s}$ is also 3. The asymptotic average code rate $\bar{R}_\infty$ as $l_{\max} \to \infty$ is 1.

Proof. Assuming independent equiprobable bits in the source sequence, based on Proposition 3.1 it follows that

$$\begin{aligned} \bar{o} &= 2\left\{\sum_{i=2}^{l_{\max}-1} (\tfrac{1}{2})^i i + (\tfrac{1}{2})^{l_{\max}-1} l_{\max}\right\} \\ &= 2\left\{1 + 2\Big[\sum_{i=3}^{l_{\max}-1} (\tfrac{1}{2})^i\Big] - 2\Big[(\tfrac{1}{2})^{l_{\max}}(l_{\max} - 1)\Big] + (\tfrac{1}{2})^{l_{\max}-1} l_{\max}\right\}. \end{aligned} \tag{3.9}$$

As $l_{\max} \to \infty$, $\bar{o}$ becomes

$$\bar{o} = 2\left\{1 + 2\Big[\sum_{i=3}^{\infty} \Big(\frac{1}{2}\Big)^i\Big]\right\} = 3. \tag{3.10}$$

Similarly, the average length of source words $\bar{s}$ is given by

$$\begin{aligned}\bar{s} &= 2\left\{\sum_{i=2}^{l_{\max}-1}(\tfrac{1}{2})^i i + (\tfrac{1}{2})^{l_{\max}-1}(l_{\max}-1)\right\} \\ &= 2\left\{1+\ 2\Big[\sum_{i=3}^{l_{\max}-1}(\tfrac{1}{2})^i\Big] - 2\Big[(\tfrac{1}{2})^{l_{\max}}(l_{\max}-1)\Big] + (\tfrac{1}{2})^{l_{\max}-1}(l_{\max}-1)\right\}.\end{aligned} \tag{3.11}$$

As $l_{\max} \to \infty$, $\bar{s}$ becomes

$$\bar{s} = 2\left\{1+\ 2\Big[\sum_{i=3}^{\infty}\Big(\frac{1}{2}\Big)^i\Big]\right\} = 3. \tag{3.12}$$

Therefore, from equations (1.11), (1.12) and (3.8), it follows that as $l_{\max} \to \infty$, the asymptotic code rate and the asymptotic redundancy are 1 and 0 respectively, and the asymptotic average efficiency is 1. As noted earlier, this is expected since the sequence becomes unconstrained as $l_{\max} \to \infty$. Of interest is how rapidly this occurs, which is discussed below. □

Rate of convergence

Theoretical upper bound: As noted above, limiting the maximum length of codewords to $l_{\max} < \infty$ in the truncated FSM results in $\widetilde{C}_M < C$. Using (1.10), the relationship of maximum possible efficiency $\widetilde{\eta}$ with $l_{\max}$ is given in Table 3.10, where it is shown that $\widetilde{\eta}$ approaches 1 quickly even with moderate $l_{\max}$. Therefore, the reduction in efficiency arising from truncation is negligible with moderate to large $l_{\max}$.

Table 3.10: $\widetilde{\eta}$ with $l_{\max}$

$l_{\max}$	4	5	6	7	8	9	10
$\widetilde{\eta}$	0.925	0.966	0.984	0.992	0.996	0.998	0.999

Practical codes Given the asymptotic behavior of $\bar{o}$ and γ as outlined in Proposition 3.2, we now analyze the convergence to their limits in practical codes. We do so by deriving the first and second order derivatives of $\bar{o}$ and γ versus $l_{\max}$. As shown in (3.13) and (3.14), $\bar{o}$ increases with $l_{\max}$, but the rate of increase declines asymptotically as $\bar{o} \to 3$.

$$\frac{d\bar{o}}{dl_{\max}} = -\Big(\frac{1}{2}\Big)^{l_{\max}-2}\ln\Big(\frac{1}{2}\Big) > 0. \tag{3.13}$$

$$\frac{d^2\bar{o}}{dl_{\max}{}^2} = -\left(\frac{1}{2}\right)^{l_{\max}-2}\left[\ln\left(\frac{1}{2}\right)\right]^2 < 0. \tag{3.14}$$

As shown in (3.15) and (3.16), γ decreases as $l_{\max}$ grows and the rate of decrease declines asymptotically as $\gamma \to 0$.

$$\frac{d\gamma}{dl_{\max}} = \left(\frac{1}{2}\right)^{l_{\max}-2}\ln\left(\frac{1}{2}\right) < 0. \tag{3.15}$$

$$\frac{d^2\gamma}{dl_{\max}{}^2} = \left(\frac{1}{2}\right)^{l_{\max}-2}\left[\ln\left(\frac{1}{2}\right)\right]^2 > 0. \tag{3.16}$$

We plot redundancy versus average length of codewords in Fig. 3.21. In this figure we compare the redundancy of our proposed VV Pearson codes with the redundancy of the binary VF Pearson codes proposed in [60] which is shown to be $\frac{3}{2}\times(\frac{1}{2})^{(o-2)}$ where o denotes the length of the fixed-length codewords [60]. It is evident from Fig. 3.21 that our proposed VV codes achieve a specified redundancy with significantly shorter average codeword lengths.

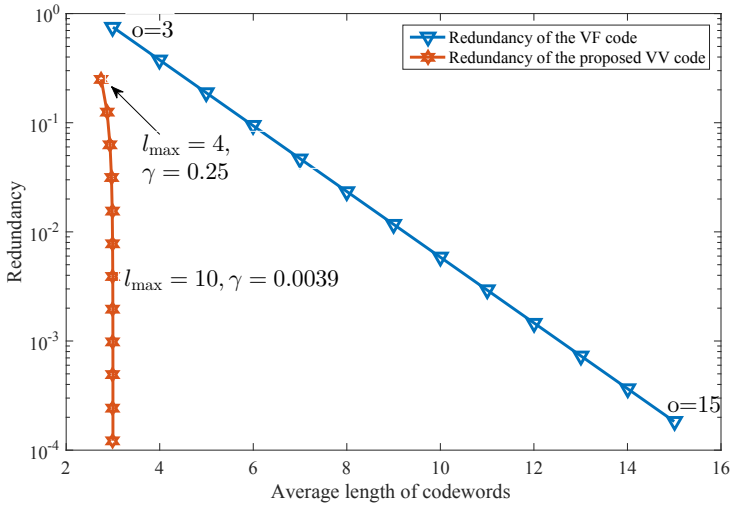

Figure 3.21: Comparison of redundancy of the proposed VV code with the VF code in [60]

Implementation

We now show that our binary VV Pearson codes can be encoded and decoded in a straightforward fashion. The codeword and source word lengths

described in Proposition 3.1 facilitate well-structured codebooks where simple encoding and decoding can be implemented without table look-up, given a predefined $l_{\max}$. Consider, for instance, the codebook with $l_{\max} = 10$ in Table 3.9. While table-lookup encoding/decoding is possible, we note that the structure of the source words and codewords instead allows encoding and decoding using Algorithms 3 and 4 outlined below.

Algorithm 3 Encoding of binary VV Pearson codes

Initialize:

N_s: the total number of source words

$b_{ij} \leftarrow j$-th bit of the i-th source word $\mathbf{b_i}$

$c_{ij} \leftarrow j$-th bit of the i-th codeword $\mathbf{c_i}$

$i = 1$

Start:

while $i \leq N_s$ **do**

  $j = 1$

  while $j \leq l_{\max} - 1$ **do**

    Read in source bit b_{ij}

    $c_{ij} = b_{ij}$

    if $b_{i(j-1)} == 0$ && $b_{ij} == 0$ **then**

      // *source words with length* $\leq l_{\max} - 1$ *that end with 00*

      Encode by changing $c_{i,j-1}$ to 1.

      Break;

    else if $b_{i(j-1)} == 0$ && $b_{ij} == 1$ **then**

      // *source words with length* $\leq l_{\max} - 1$ *that end with 01*

      Encode by changing first $j - 2$ bits in $\mathbf{c_i}$ to 0

      Break;

    else if $j == l_{\max} - 1$ && $b_{ij} == 0$ **then**

      // *source words with length* $l_{\max} - 1$ *that end with 0*

      Insert a 1 before c_{ij}

    else if $j == l_{\max} - 1$ && $b_{ij} == 1$ **then**

      // *source words with length* $l_{\max} - 1$ *that end with 1*

      Convert all bits in $\mathbf{c_i}$ to 0 and then append a 1

    end if

    $j = j + 1$

  end while

  $i = i + 1$

end while

Algorithm 4 Decoding of binary VV Pearson codes

Initialize:

N_r^w: the total number of received words

$r_{ij} \leftarrow j$-th bit of the i-th received codeword $\mathbf{r_i}$

$d_{ij} \leftarrow j$-th bit of the i-th decoded word $\mathbf{d_i}$

$i = 1$

Start:

while $i \leq N_r^w$ **do**

  $j = 1$

  while $j \leq l_{\max}$ **do**

    Read in received bit r_{ij}

    $d_{ij} = r_{ij}$

    if $j == l_{\max}$ && $r_{ij} == 0$ **then**

      // codewords with length $l_{\max}$ that end with 0

      Drop $d_{i(j-1)}$

    else if $j == l_{\max}$ && $r_{ij} == 1$ **then**

      // codewords with length $l_{\max}$ that end with 1

      Drop d_{ij} and convert all remaining bits in $\mathbf{d_i}$ to 1

    else if $r_{ij} \neq r_{i(j-1)}$ && $r_{ij} == 0$ **then**

      // codewords with length $< l_{\max}$ that end with 0

      Decode by changing $d_{i,j-1}$ to 0.

      Break;

    else if $r_{ij} \neq r_{i(j-1)}$ && $r_{ij} == 1$ **then**

      // codewords with length $< l_{\max}$ that end with 1

      Decode by changing first $j - 2$ bits to 1

      Break;

    end if

    $j = j + 1$

  end while

  $i = i + 1$

end while

Chapter 4

Synchronization of variable-length constrained sequence codes

In the preceding chapters we demonstrated that improved code efficiency and simpler implementation can be achieved with variable-length constrained sequence codes as compared to fixed-length codes. Apart from their advantages, however, variable-length constrained sequence codes have the drawback that because of noise that occurs during transmission, erroneously received sequences may result in loss of codeword boundary synchronization at the decoder. Mis-synchronization typically results in insertion or deletion errors, which may cause burst errors at the output of the constrained sequence decoder that may be difficult for the outer ECC to correct. To enable practical implementation of variable-length constrained sequence codes, we aim to develop codes with good synchronization properties to make it feasible for ECCs to correct errors that occur at the output of the constrained sequence decoder. Alternatively, with automatic repeat request (ARQ), it is desired that error propagation be limited to within the current received packet, to ensure that subsequent packets are unaffected.

In this section, we consider the variable-length constrained sequence codes constructed by the approaches in Section 2.1 where to the point the design guideline has been to achieve the highest possible code rate. We show that

different strategies should be considered when we aim to develop codes that achieve both high efficiency and good synchronization properties. We consider a variety of constraints that are studied previously, including the runlength limited (RLL) constraint, constraints that mitigate ICI in flash memories, the Pearson constraint, and the DC-free constraint. We show that these variable-length constrained sequence codes have good synchronization properties such that the receiver regains synchronization quickly once it loses synchronization. This ensures that error propagation is limited during decoding, which confirms the practicality of these variable-length codes.

4.1 Synchronization

We wish to develop variable-length constrained sequence codes with good codeword synchronization properties, in which their codebooks have as many *synchronizing codewords* as possible. A synchronizing codeword guarantees that whenever it is received, the decoder achieves synchronization because it is guaranteed to correctly identify the end of this codeword and therefore maintain or recover synchronization, regardless of the correctness of the previously received codewords [63]. According to [63], a synchronizing codeword $\mathcal{C} = c_1c_2...c_n$ in the set of codewords from codebook $\mathbf{C}$ must satisfy the following two conditions:

Condition 4.1: $\forall X = x_1x_2...x_m$ in $\mathbf{C}$ such that $m > n$ and $\mathcal{C}$ is a substring of X, $c_1c_2...c_n = x_{m-n+1}x_{m-n+2}...x_m$ and $c_1c_2...c_n \neq x_ix_{i+1}...x_{i+n-1}$ $\forall i \neq m-n+1$.

Condition 4.2: $\forall j < n$ such that $c_1c_2...c_j$ is a suffix of any codeword in $\mathbf{C}$, $c_{j+1}c_{j+2}...c_n$ is a valid codeword in $\mathbf{C}$.

Condition 4.1 indicates that if a synchronizing codeword $\mathcal{C}$ is an internal bit string of any codeword X, the last bit of $\mathcal{C}$ must also be the last bit of X. This guarantees that correct identification of $\mathcal{C}$ results in correct recognition of the end of a codeword, hence synchronization is maintained or recovered. Condition 4.2 ensures that whenever mis-synchronization occurs that results in the receiver incorrectly identifying c_j as the end of a codeword,

resynchronization is guaranteed to occur at the end of codeword $\mathcal{C}$ because c_n will be identified as the end of $\mathcal{C}$. It is shown in [63] that some Huffman codes exhibit good synchronizing properties because their codewords re-synchronize the decoder regardless of the previous synchronization status.

We denote the synchronization probability (denoted as *sync probability*) *P of a codebook* $\mathbf{C}$ as the sum of occurrence probabilities of all synchronizing codewords. Note that synchronizing codewords appear more frequently in coded sequences constructed from codebooks with higher values of P, and therefore that decoders will typically re-synchronize more quickly once synchronization is lost, resulting in fewer burst errors for the outer ECC to correct. Our goal is therefore to construct codes with high synchronization probability.

To construct these codes, we first consider the minimal set, and then consider partial extensions of this minimal set to achieve high efficiency and high sync probability. Note that, prior to NGH coding, a minimal set and its partial extensions are not codebooks because their words have not been assigned source words, and therefore it is not possible to obtain their occurrence probabilities. However, because we expect good codes to have probabilities that are close to maxentropic, in our construction approach we approximate the sync probability P of a minimal set M and its partial extensions M_p as the sum of the maxentropic probabilities of each synchronizing word.

Lastly, we note that existence of synchronizing codewords is a sufficient but not necessary condition for the receiver to regain synchronization, since in some situations the receiver may correctly regain synchronization on nonsynchronizing words, as we will show in our simulation results. Thus, we expect there to be a strong correlation, but not necessarily a direct relationship, between the sync probability and synchronization performance.

4.2 Criteria for minimal set selection

The criteria introduced in Section 2.1 aim to select the specified state that results in the minimal set with the highest possible code rate. In situations where we wish to construct codes with good synchronization properties, criteria should be developed that result in codebooks with high sync probabilities. In this section we consider selection of the specified state from the minimal set, and illustrate our selection criteria with a variety of constraints. In the next section we consider partial extensions of this minimal set.

Observation 4.1: In the FSM description of a constraint, if a state has a loop associated with itself, then selection of this state as the specified state is likely to correspond to a minimal set with a small sync probability, unless the loop corresponds to a synchronizing word.

Observation 4.1 arises when a single bit 1 or 0 that is associated with the specified state appears as a word in the minimal set. The word 1 or 0 is likely to violate Condition 4.1, therefore this case should be avoided unless codeword 0 or 1 is a synchronizing word. We demonstrate Observation 4.1 with the following examples.

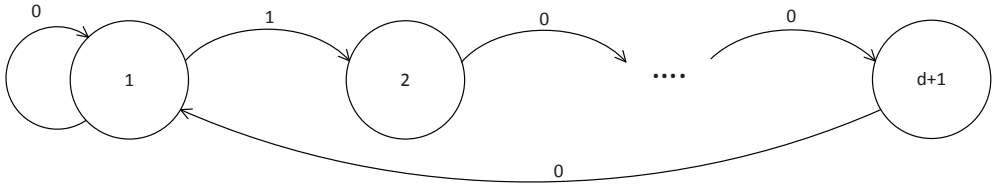

Figure 4.1: FSM of the $(d, k = \infty)$ RLL constraint.

Example ((d, ∞) constraint): the FSM of the $(d, k = \infty)$ RLL constraint is shown in Fig. 4.1. We consider the $(d = 2, k = \infty)$ RLL constraint as an example. To maximize the efficiency of the code, in accordance with the Criterion 2.1 in Section 2.1 we would select state 1 as the specified state, and find the minimal set to be $M_1 = \{0, 100\}$. More generally, as is evident from Fig. 4.1, the minimal set of a (d, ∞) constraint is $M_1 = \{0, 1\underbrace{0.....0}_{d\ zeros}\}$.

It can be verified that the second word is a synchronizing word. However, it can be observed that word 0 in M_1 is not a synchronizing word since it does not satisfy Condition 4.1. But it can also be verified that, if we select any other state as the specified state, the minimal set will have a sync probability of 100%. For example in the FSM of the $(d = 2, k = \infty)$ RLL constraint, $M_2 = \{001, 0001, 00001, ...\}$ in which every word in M_2 is a synchronizing word. However this minimal set contains an unlimited number of words, and therefore will result in a less efficient code than that constructed using M_1 as proposed in Section 2.1. This demonstrates that a code optimized for efficiency may not be optimized for sync probability, and vice versa.

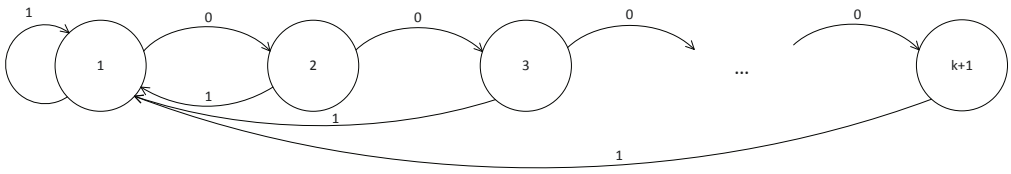

Figure 4.2: FSM of the $(d = 0, k)$ RLL constraint.

Example $((0, k)$ constraint): the FSM of the $(d = 0, k)$ RLL constraint is shown in Fig. 4.2. We consider the $(d = 0, k = 3)$ RLL constraint as an example. To maximize the efficiency of the code, in accordance with the Criterion 2.1 in Section 2.1, we select state 1 as the specified state, and establish the minimal set as $M_1 = \{1, 01, 001, 0001\}$. It can be observed that for the $(d = 0, k = 3)$ RLL constraint, and more generally for any $(d = 0, k)$ constraint, every word in M_1 is a synchronizing word, and hence the sync probability is 100%. In this case, the selection criteria in Section 2.1 also result in the minimal set with the highest sync probability. Note that this is in agreement with Observation 4.1 since the single-bit word 1 associated with state 1 is a synchronizing word.

Observation 4.2: In the FSM description of a constraint, if outgoing edges and incoming edges of a state correspond to the same bit sequence, it is likely that selection of this state as the specified state results in a minimal set with few synchronizing words. This occurs because the prefix of a codeword $\mathcal{W}$

will be suffix of one or more codewords in the minimal set, and the remaining bits of $\mathcal{W}$ may not be a valid codeword, thus violating Condition 4.2.

Observation 4.2 arises when a word in the minimal set (that corresponds to the outgoing edges) can be divided into a suffix of another word (that corresponds to the incoming edges) plus the remaining bits, and therefore it might occur that these remaining bits are not a valid word, which violates Condition 4.2. We explain Observation 4.2 with the following example, in which we use the notation $\mathcal{W}\backslash\zeta$ to represent a substring of codeword $\mathcal{W}$ where the prefix ζ of $\mathcal{W}$ is excluded, and denote $M\backslash\mathcal{W}$ as the set of words in the minimal set M where word $\mathcal{W}$ is excluded.

Example (general (d,k) constraints): we consider general (d,k) constraints where $d \neq 0$ and $k \neq \infty$. The corresponding FSM is shown in Fig. 4.3. According to the Criterion 2.1 in Section 2.1, any state from states 1 to $d+1$ can be selected as the specified state because they all result in the highest maximum possible code rate. For example, the minimal set with state 1 as the specified state is $M_1 = \{\underbrace{00...0}_{d\ zeros}1,\ \underbrace{00...0}_{d+1\ zeros}\ 1,\ \underbrace{00...0}_{d+2\ zeros}\ 1, ...,$ $\underbrace{00...0}_{k\ zeros}1\}$. Since every word is a synchronizing word, the sync probability is 100%. However, different from the criteria in Section 2.1, as introduced in Observation 4.2, state $\sigma, 1 < \sigma \leq d$ should not be selected as the specified state. Consider state 2 as an example. One of the words $\mathcal{W}$ in its minimal set is $\underbrace{00...0}_{d-1\ zeros}\ 10$ which is not a synchronizing word, since 0 is a suffix of $\mathcal{W}$ itself but the minimal set does not contain the word $\mathcal{W}\backslash 0 = \underbrace{00...0}_{d-2\ zeros}\ 10$. Therefore, Condition 4.2 is not satisfied and hence $\mathcal{W}$ is not a synchronizing word, making the sync probability of M_2 less than 100%. A similar observation can be made for states $\sigma, 2 \leq \sigma \leq d$, where we can see that these states have both an outgoing edge and an incoming edge that are associated with bit zero, which violates Condition 4.2. $\mathcal{W}$ could be a synchronizing word iff $\mathcal{W}\backslash 0$ was a valid word, which is not the case.

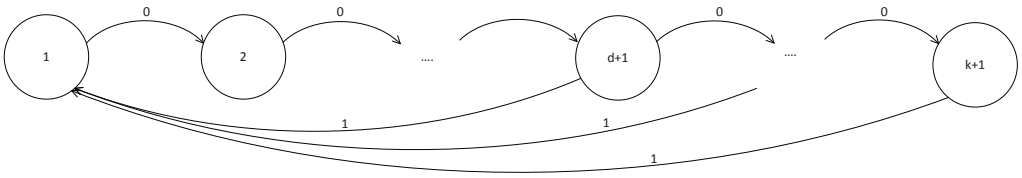

Figure 4.3: FSM of general (d, k) RLL constraints, $d \neq 0, k \neq \infty$.

Based on the above observations and examples, we propose the following criteria to select the specified state that results in high sync probability.

Criterion 4.1: if a state has a loop associated with itself, this state should not be selected as the specified state unless the loop corresponds to a synchronizing word.

Criterion 4.2: if outgoing edges and incoming edges of a state correspond to the same bit sequence, this state should not be selected as the specified state.

Note that these two criteria are guidelines for general constraints when selecting the specified state. Therefore one should consider the specific characteristics of each constraint when selecting the specified state. In addition to the (d, k) RLL constraints that we have discussed, we now illustrate state selection for a variety of other constraints.

4.2.1 Constraints that mitigate ICI in flash memories

As discussed in previous sections, constrained sequence codes that forbid the pattern 101 have been designed to limit ICI [20, 28, 29, 30, 31]; the FSM describing this constraint is shown in Fig. 4.4, which is Fig. 2.6 in Section 2.1.

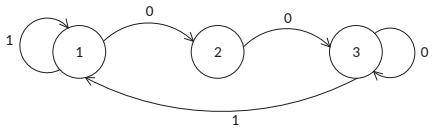

Figure 4.4: FSM of the constraint that forbids pattern 101 for ICI mitigation.

According to Criteria 2.2 and 2.3 in Section 2.1, states 1 and 3 are equally preferred as the specified state due to the fact that they both result in the highest maximum possible code rate. However, according to Criterion

4.1 described in this section, state 3 is inferior to state 1 in terms of sync probability since state 3 has a loop associated with itself that corresponds to word 0, and the word 0 is not a synchronizing word. A closer look at $M_3 = \{0, 100, 1100, 11100, ...\}$ reveals that word 0 is not a synchronizing word, since 0 does not satisfy Condition 4.1. However, every word in $M_1 = \{1, 001, 0001, 00001, ...\}$ is a synchronizing word, including the one-bit word on the self-loop on state 1, resulting in the sync probability of M_1 to be 100%. Therefore, considering the criteria in both Section 2.1 and in this section, state 1 is preferred.

Now consider ICI mitigation in multi-level cell (MLC) flash memories, where each coded symbol can be 0, 1, 2 or 3 and the pattern 303 is forbidden by the constraint. The FSM that represents this constraint is shown in Fig. 4.5. Different from the FSM in Fig. 4.4, according to the study in Section 2.1, state 3 is better than state 1 in terms of maximum possible code rate. As demonstrated in the previous example, however, based on Criterion 4.1, M_3 has the word 0 that is not a synchronizing word whereas as we demonstrate below, every word in M_1 is a synchronizing word. This results in the sync probability of M_1 to be 100%. Therefore, we note that in general there is a tradeoff between the maximum possible code rate and the sync probability, and it is the choice of the system designer as to which to optimize when selecting the specified state and constructing the corresponding minimal set.

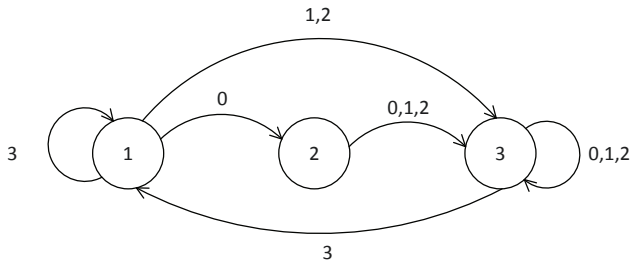

Figure 4.5: FSM of the constraint that mitigates ICI in MLC flash memories.

To show that the sync probability of M_1 is 100%, consider first the quaternary bit 3 in M_1. It can be observed that 3 does not violate the

synchronization conditions and hence is a synchronizing word. Consider the set $M_1\backslash 3$. All words in M_1 end with a quaternary bit 3, and no word in $M_1\backslash 3$ starts with a quaternary bit 3, therefore Condition 4.2 is satisfied for all words in $M_1\backslash 3$. Furthermore, since the quaternary bit 3 only appears at the end of each word, Condition 4.1 is satisfied for all words in $M_1\backslash 3$. Thus, M_1 has a sync probability of 100%.

4.2.2 The Pearson constraint

In this section we discuss the selection of the specified state in the FSM of the Pearson constraint that is introduced in Section 3.3.

The FSM of the binary Pearson constraint is shown in Fig. 3.20 in Section 3.3. According to Criteria 2.2 and 2.3 in Section 2.1, we select state G as the specified state to achieve the highest maximum possible code rate. However, it can be observed that state G does not satisfy Condition 4.2 since both an outgoing edge and an incoming edge correspond to bit 0 (and bit 1). Therefore, state G results in a sync probability lower than 100%. The minimal set is $M_G = \{01, 10, 001, 110, 0001, 1110, ...\}$. It can be verified that every word in M_G other than the words 01 and 10 is a synchronizing word, and the sync probability is 50%.

State $G-1$ and $G+1$ are inferior to state 1 in terms of the maximum possible code rate as discussed in Section 3.3, however, it can be verified that only words 01 and 101 are not synchronizing words in M_{G-1}, and the sync probability of M_{G-1} (and M_{G+1}) is 62.5%. Therefore, it is again observed that a tradeoff exists between the maximum possible code rate and the sync probability.

4.2.3 DC-free constraints

We consider DC-free constraints with N different RDS values, as depicted in the FSM shown in Fig. 2.1. According to Criteria 2.2 and 2.3 in Section 2.1, state $\lceil N/2 \rceil$ should be selected as the specified state since its minimal set has the highest possible code rate. We first consider the case of

the DC-free constraint with $N = 5$. We have that $M_3 = \{10, 01, 1100, 0011, 110100, 001011\}$ with $l_{max} = 6$. Unfortunately it can be verified that M_3 does not contain a synchronizing word, thus M_3 has a sync probability of 0%.

Now we consider the general case where $N > 5$ and generate the codebooks using the construction algorithm in Section 2.1. To illustrate, in Table 2.8 shown in Section 2.1 we list a minimal set when $N = 7$. Note that since state $\lceil N/2 \rceil$ has sequences 11 and 00 associated with both its outgoing and incoming edges, it does not satisfy Criterion 4.2 and thus may not be preferred in terms of sync probability. In fact, we can prove that with $N \geq 5$, the minimal set $M_{\lceil N/2 \rceil}$ has sync probability 0%. The proof is as follows.

It is clear that words 01 and 10 are not synchronizing words since they violate Condition 4.1. Furthermore, we observe that any of the remaining words $\mathcal{W}$ in $M_{\lceil N/2 \rceil}$ ends with 00 or 11. Therefore, $\mathcal{W}$ is a synchronizing word iff $\mathcal{W} \backslash 00$ is a valid word when 00 is the prefix of $\mathcal{W}$, and similarly iff $\mathcal{W} \backslash 11$ is a valid word when 11 is the prefix of $\mathcal{W}$. However, $\mathcal{W} \backslash 00$ cannot be a valid word since a valid word in $M_{\lceil N/2 \rceil}$ must have an equal number of zeros and ones, while $\mathcal{W} \backslash 00$ has two more ones than zeros. A similar argument can be made for $\mathcal{W} \backslash 11$. Hence none of the words in $M_{\lceil N/2 \rceil}$ is a synchronizing word and therefore the sync probability is 0%.

However, we note that state 1 does not violate Criterion 4.2, and that the minimal set associated with state 1 has a nonzero sync probability. For $N = 5$ and $l_{max} = 6$, $M_1 = \{10, 1100, 110100, 111000\}$, where it can be verified that 110100 and 111000 are synchronizing words, resulting in a sync probability of 7.4% which is equal to what is possible with M_5 and is higher than that can be achieved in other states.

We also note that with additional prior knowledge at the decoder, performance can be improved in terms of sync probability. For example, assume the decoder exploits its knowledge that all DC-free codewords are of even length. With $N = 5$, the sync probability of M_1 is improved since one more word, 1100, becomes a synchronizing word. We will present more results assuming that the decoder has additional knowledge related to the

constraint.

In this section we investigated the sync properties of a variety of constraints, and demonstrated that states that satisfy Criteria 4.1 and 4.2 can result in minimal sets with high sync probabilities. In the next section, we show that variable-length constrained sequence codes constructed via partial extensions of minimal sets can achieve both high efficiency and high sync probability.

4.3 Partial extensions

Using words in the minimal set as the set of codewords may not result in capacity-approaching codes since higher efficiency is often achieved with larger codebooks. Therefore, we perform partial extensions over the minimal sets to generate larger codebooks. However, different from Sections 2.1 and 2.2 where partial extensions are exhaustively performed and the one that has the highest efficiency is selected, in this section we introduce an algorithm that efficiently guides the partial extension process such that the resulting codebook has a high sync probability. Note that performing partial extensions without care can reduce the sync probability, as we show in the following example.

Example ($d = 1, k = 3$ RLL constraint) As discussed in the previous section, we select state 1 in Fig. 4.3 as the specified state that results in a minimal set of sync probability 100%, i.e., $M_1 = \{01, 001, 0001\}$. If we extend word 0001, we have $M_{1,p} = \{01, 001, 000101, 0001001, 00010001\}$ where $01, 001$ and 00010001 are not synchronizing words, resulting in a sync probability of only 17%. In contrast, extension of the words 01 and 001 results in sync probabilities of 78% and 43% respectively.

This example motivates us to design a guided partial extension algorithm that aims to simultaneously achieve both high code efficiency and high sync probability.

4.3.1 Extending synchronizing versus nonsynchronizing words

We first consider whether to extend synchronizing words or nonsynchronizing words when our aim is to keep the sync probability high. We consider the following proposition and observation.

Proposition 4.1 Extending a synchronizing word lowers sync probability.

Proof. Consider a synchronizing word $\mathcal{W}$ in M. If we extend $\mathcal{W}$, the $|M|$ resulting words from extension of $\mathcal{W}$ cannot all be synchronizing words since one of the resulting words is $\mathcal{W}' = \mathcal{W} + \mathcal{W}$. $\mathcal{W}$ becomes a suffix after this extension, and $\mathcal{W}'\backslash\mathcal{W} = \mathcal{W}$ can no longer be a valid word. Therefore, $\mathcal{W}'$ is not a synchronizing word, which lowers the sum of maxentropic probabilities of all synchronizing words. □

Observation 4.3: Extending a nonsynchronizing word may increase the sync probability.

It is often the case that an extended word $\mathcal{W}'$ constructed through an extension of a nonsynchronizing word $\mathcal{W}$ is a synchronizing word, since $\mathcal{W}'$ consists of $\mathcal{W}$ concatenated with a valid word. $\mathcal{W}$ becomes the suffix of the extended word $\mathcal{W}'' = \mathcal{W}+\mathcal{W}$, hence any $\mathcal{W}' \neq \mathcal{W}''$ is likely to satisfy Condition 4.2.

For example, consider the ICI constraint in Fig. 4.5. $M_3 = \{0, 1, 2, 31, 300,$
$301, 302, 331, 332, 3300, 3301, 3302...\}$ where word 0 is not a synchronizing word. If we extend word 0, the resulting set is $\{\underbrace{00, 01, 02, 031, 0300, 0301, 032, 0331}_{partial\ extension}, \underbrace{0332, 03300, 03301, 03302}_{partial\ extension}, 1, 2, 31, 300,$
$301, 302, 331, 332, 3300, 3301, 3302, ...\}$ where only 00 is not a synchronizing word. Note that the extended words $\{01, 02, 031, 0300, 0301, 032, 0331, 0332,$
$03300, 03301, 03302\}$ are synchronizing words, thus the sync probability has increased.

Based on Proposition 4.1 and Observation 4.3, we propose extending nonsynchronizing words whenever possible. Only when the minimal set does

not contain a nonsynchronizing word do we recommend extending synchronizing words. Selection of synchronizing words that will be extended is discussed below.

4.3.2 The guided partial extension algorithm

We start from the minimal set M, i.e., $M_p = M$. In each partial extension we first obtain the set of suffixes $\mathbf{S}$ where $\forall \mathcal{S} \in \mathbf{S}$ is a suffix of a synchronizing word $\mathcal{W}$ in M_p, i.e., $\mathcal{W} = \zeta + \mathcal{S}$ where ζ denotes any suffix that exists in M_p. We search for a nonsynchronizing word $\mathcal{N}$ such that $\mathcal{N} \notin \mathbf{S}$ is the target word that we would like to extend. The reason for this is as follows. Suppose a synchronizing word $\mathcal{W}$ can be represented as $\mathcal{W} = \zeta + \mathcal{N}', \mathcal{N}' \in M_p \wedge \mathbf{S}$. In this case, extending $\mathcal{N}'$ would make $\mathcal{W}$ nonsynchronizing since $\mathcal{N}'$ is not a valid word any more, which would reduce the sync probability. Therefore, we extend a nonsynchronizing word $\mathcal{N} \notin \mathbf{S}$. At the same time, $\mathcal{N}$ should not have a synchronizing word as its suffix, i.e., $\mathcal{N} = \Lambda + \mathcal{W}$ (where Λ denotes any sequence) should not be extended since extension of $\mathcal{N}$ will result in $\mathcal{W}$ violating Condition 4.1 and becoming a nonsynchronizing word. If we cannot find such a word $\mathcal{N}$, then a partial extension will result in a synchronizing word becoming nonsynchronizing. In this case we choose to perform extension of each of the nonsynchronizing words and choose the one that results in the highest synchronization probability.

If there are no nonsynchronizing words in M_p, we must extend a synchronizing word. We note that extending a longer synchronizing word is more likely to reduce the sync probability than extending a shorter synchronizing word, since a longer synchronizing word $\mathcal{W}$ may correspond to a greater number of valid words $\mathcal{W}''$ where $\mathcal{W} = \Lambda + \mathcal{W}''$, and extension of $\mathcal{W}$ results in $\mathcal{W}''$ violating Condition 4.1. Returning to the example above, it is straightforward to confirm that extension of word 0001 is not preferred since it results in words 01 and 001 violating Condition 4.1. On the other hand, extension of word 001 only excludes word 01 from being a synchronizing word, and extension of the word 01 does not result in any word violating

Condition 4.1. Therefore, we choose to extend the shortest synchronizing word, and we propose Proposition 4.2 based on the following lemma.

Lemma 4.1 Under the condition that M_p does not contain nonsynchronizing words, the shortest synchronizing word cannot be represented as a suffix plus a valid word.

Proof. The proof is straightforward and is omitted. □

Proposition 4.2 Under the condition that M_p does not contain nonsynchronizing words, extending the shortest synchronizing word $\mathcal{W}$ reduces the sync probability by $\lambda_{max}^{-2l_{\mathcal{W}}}$ where $l_{\mathcal{W}}$ is the length of word $\mathcal{W}$.

Proof. Since $\mathcal{W}$ is a synchronizing word, it satisfies Condition 4.1. Therefore words resulting from extension of $\mathcal{W}$ also satisfy Condition 4.1. From Lemma 4.1 we know that these words also satisfy Condition 4.2 except for the extended word $\mathcal{W}'' = \mathcal{W} + \mathcal{W}$. It can be easily checked that all words in $M_p \backslash \mathcal{W}$ remain synchronizing words, because they satisfy both Conditions 4.1 and 4.2. The reduction of sync probability as the result of extending $\mathcal{W}$ is therefore the maxentropic probability of word $\mathcal{W}''$, which is $\lambda_{max}^{-2l_{\mathcal{W}}}$. □

Based on the above discussion, the proposed guided partial extension algorithm is shown in Algorithm 5. This algorithm is initialized with $M_p = M$ and $J = 0$ where J is the recursion depth, and is recursively called a number of times until J exceeds the pre-established limits. With each recursion depth the algorithm outputs a codebook with the highest synchronization probability at the current depth.

4.4 Capability to quickly re-synchronize

In this section, we present results regarding the efficiency and sync probability of codes constructed based on the procedures outlined above. We evaluate, under the case of a binary symmetric channel (BSC), the sync properties in terms of the average number of bits and average number of codewords that the decoder requires to regain synchronization once

Algorithm 5 The guided_partial_extension_algorithm

Data: words in M_p, the recursion depth J

Result: the updated M_p

Initialization: $\mathbf{S}$, the set of nonsynchronizing words $\mathbf{P}$, the set of synchronizing words $\mathbf{Q}$, the set of words $\mathbf{\Gamma}$ that will be extended in the current recursion, $\mathbf{\Gamma} = \Phi$

Sort($\mathbf{P}$) // *sort by length from shortest to longest*

Sort($\mathbf{Q}$) // *sort by length from shortest to longest*

if $\mathbf{P} \neq \varnothing$ **then**

  foreach *word* $\mathcal{N} \in \mathbf{P}$ **do**

    if $\mathcal{N} \in \mathbf{S}$ *or* $\mathcal{N} = \mathcal{N}'' + \mathcal{W}$ $\exists \mathcal{W} \in \mathbf{Q}$ **then**

      continue;

    else

      if $\mathbf{\Gamma}$.*empty*() *or* $\mathcal{N}$.*size*() == $\mathbf{\Gamma}$.*back*().*size*() **then**

        $\mathbf{\Gamma}$.*push_back*($\mathcal{N}$)

      else

        break;

      end

    end

  end

  if $\mathbf{\Gamma}$.*empty*() **then**

    construct $\mathbf{\Gamma}$ with all words in $\mathbf{P}$

  end

else

  construct $\mathbf{\Gamma}$ with the shortest words in $\mathbf{Q}$

end

/* *recursion with depth* $J + 1$ */

foreach *word* $\tau \in \mathbf{\Gamma}$ **do**

  extend τ in M_p, obtain $M_{p,\tau}$ and the corresponding synchronization probability

  call *guided_partial_extension_algorithm*($M_{p,\tau}, J + 1$)

  undo the extension of τ in $M_{p,\tau}$, backtrack to M_p

end

Output: $M_{p,\tau}$ that has the highest synchronization probability $\forall \tau \in \mathbf{\Gamma}$

synchronization is lost. The decoding algorithm that we consider is the conventional bit-by-bit decoding algorithm described in the Appendix of [65], and which is reported here as Algorithm 6.

4.4.1 Upper bounds of the average number of codewords and bits before resynchronization

We first derive an upper bound on the number of codewords and the number of coded bits that are required for the decoder to regain synchronization once synchronization is lost. We denote the upper bound on the number of codewords N_c and the number of coded bits N_b as $\widetilde{N}_c$ and $\widetilde{N}_b$, respectively.

To evaluate $\widetilde{N}_c$, under the condition that there are no errors in the received symbols during synchronization, we consider the case when synchronization occurs only as a result of the occurrence of a sync word. We note that, after loss of synchronization, if the next received codeword is a synchronizing codeword (that occurs with probability P) then the receiver will regain synchronization. However, if the next codeword is not a synchronizing codeword (with probability $1-P$) but the subsequent word is a sync word, then synchronization will occur after two words with probability $(1-P)P$. Continuing, we have that

$$\begin{aligned}\widetilde{N}_c &= \lim_{i\to\infty}\{P + 2(1-P)P + ... + i(1-P)^{i-1}P\} \\ &= \frac{1}{P}.\end{aligned} \tag{4.1}$$

Now consider the case when errors on the binary symmetric channel occur with probability p_c. The probability that a codeword is correctly received is, on average, $(1-p_c)^{\bar{o}}$ where $\bar{o}$ denotes the average length of codewords, i.e., $\bar{o} = \sum_{s_i} 2^{-s_i} o_i$. Therefore, we have

$$\widetilde{N}_c = \frac{1}{P \times (1-p_c)^{\bar{o}}}, \tag{4.2}$$

Algorithm 6 Conventional bit-by-bit variable-length constrained sequence decoding when errors occur during transmission

Data: the received sequence $\hat{\mathbf{v}}$

Result: the decoded output, an estimation of the source sequence

Initialization: the codebook, l_{max}, $cur\_pos\_head \leftarrow 1, cur\_pos \leftarrow 1$

while $cur\_pos\_head \leq |\hat{\mathbf{v}}|$ **do**

  if $cur\_pos - cur\_pos\_head + 1 \leq l_{max}$ **then**

    if $\hat{\mathbf{v}}_{cur\_pos\_head}^{cur\_pos}$ *is a valid codeword* **then**

      decode $\hat{\mathbf{v}}_{cur\_pos\_head}^{cur\_pos}$ into the corresponding source word

      $cur\_pos \leftarrow cur\_pos + 1$

      $cur\_pos\_head \leftarrow cur\_pos$

    else

      $cur\_pos \leftarrow cur\_pos + 1$

    end

  else

    /* *no match is found in the codebook* */

    $cur\_pos \leftarrow cur\_pos\_head + 1$

    while $cur\_pos \leq |\hat{\mathbf{v}}|$ **do**

      foreach tmp_start_pos *in* $[cur\_pos\_head, cur\_pos]$ **do**

        if $\hat{\mathbf{v}}_{tmp\_start\_pos}^{cur\_pos}$ *is a valid codeword* **then**

          decode $\hat{\mathbf{v}}_{tmp\_start\_pos}^{cur\_pos}$

          $cur\_pos \leftarrow cur\_pos + 1$

          $cur\_pos\_head \leftarrow cur\_pos$

          goto line 5: *if* $cur\_pos - cur\_pos\_head + 1 \leq l_{max}$ *then*

        else

          $tmp\_start\_pos \leftarrow tmp\_start\_pos + 1$

        end

      end

      $cur\_pos \leftarrow cur\_pos + 1$

    end

  end

end

In a similar fashion, $\widetilde{N}_b$ is derived as

$$\widetilde{N}_b = \frac{1}{P \times (1-p_c)^{\bar{o}}} \times \bar{o} + \bar{o} - 1 \tag{4.3}$$

where $\bar{o} - 1$ is, on average, the maximum number of bits of the currently received codeword that has caused mis-synchronization.

As noted above, in the derivation of (4.1) – (4.3), we assume that resynchronization occurs only on synchronizing codewords. However, it can be observed in Algorithm 6 that it is possible for resynchronization to also occur on nonsynchronizing codewords. Therefore, $\widetilde{N}_c$ and $\widetilde{N}_b$ are indeed upper bounds on the average number of codewords and the average number of bits that the receiver requires to regain synchronization, since the actual number of codewords the receiver requires to resynchronize can be smaller. As we will show in the simulation results, good synchronization properties can be observed even with a small value of P and correspondingly large values of $\widetilde{N}_c$ and $\widetilde{N}_b$.

4.4.2 Simulation results

We now consider simulation results for synchronization with several classes of constrained sequence codes.

RLL constraints

Consider, for example, the $(d = 1, k = 3)$ RLL constraint. In accordance with the discussion in this Section, we select state 1 as the specified state, hence $M_1 = \{01, 001, 0001\}$. We perform the guided partial extension algorithm over M_1 with J recursions according to Algorithm 5, $J = 0 \rightarrow 9$. The code efficiency and sync probability of the resulting codes are shown in Fig. 4.6, where $J = 0$ on the horizontal axis represents the code constructed with codewords from the minimal set. From this figure we can see that after several iterations of extension, we can construct codes with code efficiency near 99% and sync probability near 100%. The decrease at $J = 1$ is due to the fact that $\mathbf{P} = \varnothing$ and we therefore have to extend a synchronizing word in the first

extension, resulting in $M_{1,p} = \{001, 0001, 0101, 01001, 010001\}$ where 0101 is not a synchronizing word since it does not satisfy Condition 4.2.

For comparison purposes, consider the codebook that corresponds to $J = 4$ shown in Table 4.1. This code achieves $\eta = 98.90\%$ and $P = 96.88\%$. Note that this codebook has the same number of codewords and an efficiency very close to the code given in Table 1.7, however, the sync probability of the code in Table 1.7 is only 21.88%, which is much lower than the code in Table 4.1. This demonstrates that the guided partial extension algorithm can effectively generate codebooks with high sync probabilities.

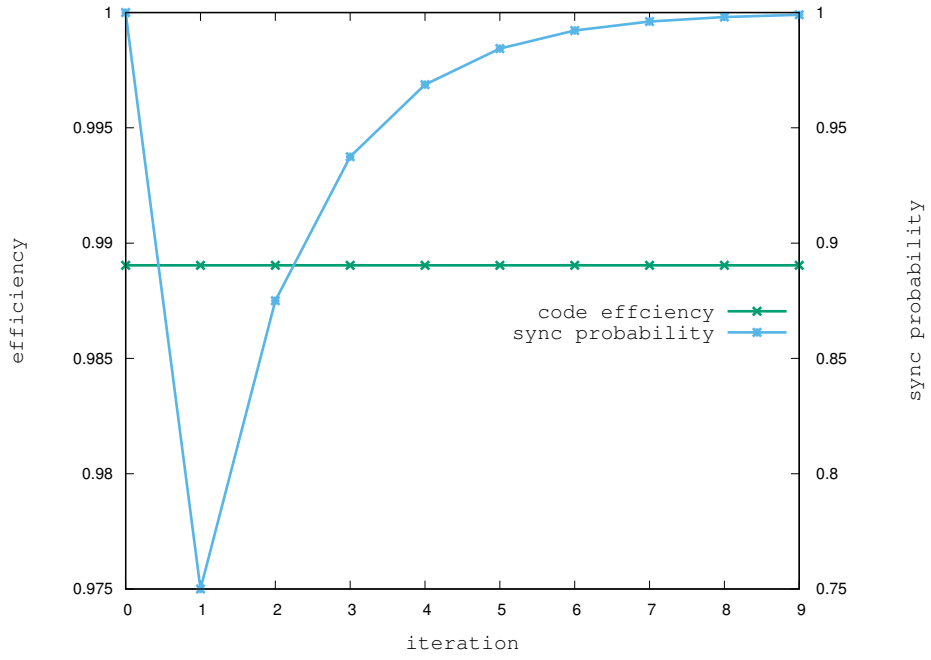

Figure 4.6: Code efficiency and sync probability of $(d = 1, k = 3)$ RLL constrained sequence codes.

We now consider the sync properties of the $(d = 1, k = 3)$ RLL codes we constructed. The coded sequence is transmitted over a BSC with crossover probability 0.1. A source sequence of 50000 bits is randomly generated and encoded into a constrained sequence with the number of codewords ranging from ∼9000 to ∼18000 with $J = 0 \rightarrow 9$, according to the codebook. Once synchronization is lost due to errors that occur during transmission through simulation of Algorithm 6, we obtain the number of bits and the number of

Table 4.1: Codebook of a $(d = 1, k = 3)$ RLL code with efficiency of 98.90% and sync probability of 96.88%

Source word	Codeword	Source word	Codeword
01	0001	00000	0101010001
11	001	00001	0101010101
001	010001	10001	010101001
101	01001	100000	010101010001
0001	01010001	100001	01010101001
1001	0101001		

codewords before the receiver regains synchronization. We consider all the occurrences that synchronization is lost, and report the average number of bits and average number of codewords that the receiver receives before it re-synchronizes. The results are shown in Figs. 4.7 and 4.8. It can be seen that the receiver generally requires less than one codeword to regain synchronization, demonstrating that these codes have good synchronization properties in the sense that once synchronization is lost, they recover synchronization quickly. N_c first increases from $J = 0 \rightarrow 1$ and then decreases from $J = 1 \rightarrow 9$, which is consistent with the sync probability shown in Fig. 4.6. Fig. 4.8 shows that N_b is around 8 for $J = 1 \rightarrow 9$. This is because codebooks with larger J have longer codewords, therefore N_b does not reduce as dramatically as N_c.

In Fig. 4.9 we demonstrate the ratio of the number of events when synchronization is achieved on synchronizing codewords to the total number of synchronization events, for $J = 0 \rightarrow 9$. As demonstrated in Fig. 4.9, these ratios are less than 100% indicating that some synchronization events occur on nonsynchronizing codewords. This demonstrates that Algorithm 6 permits synchronization to occur on nonsynchronizing codewords, as was mentioned above. This explains why in Figs. 4.7 and 4.8 the actual average number of codewords and average number of bits that the receiver requires for resynchronization are lower than $\widetilde{N}_c$ and $\widetilde{N}_b$, since $\widetilde{N}_c$ and $\widetilde{N}_b$ assume that synchronization only occurs on synchronizing codewords.

Finally, as is clear from these figures, for the $(d = 1, k = 3)$ RLL constraint there is no significant advantage to using a codebook other than the minimal set because it satisfies Criteria 4.1 and 4.2, and hence has excellent sync properties, while also having high efficiency. In contrast, in the next subsection we examine situations in which $J = 0$ is not the best choice when we compare with other codebooks constructed using our guided partial extension algorithm.

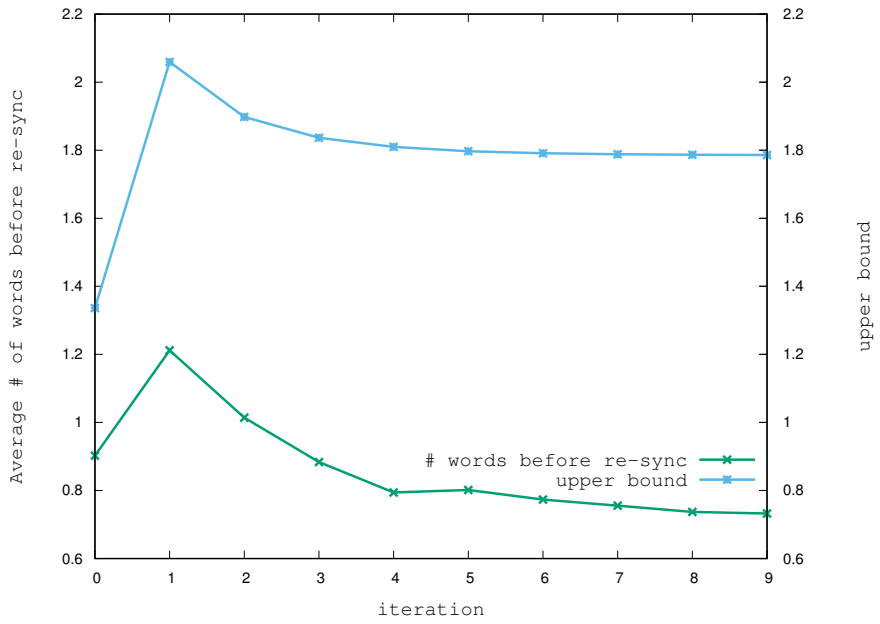

Figure 4.7: Average number of words required to regain synchronization for the constructed $(d = 1, k = 3)$ RLL constrained sequence codes.

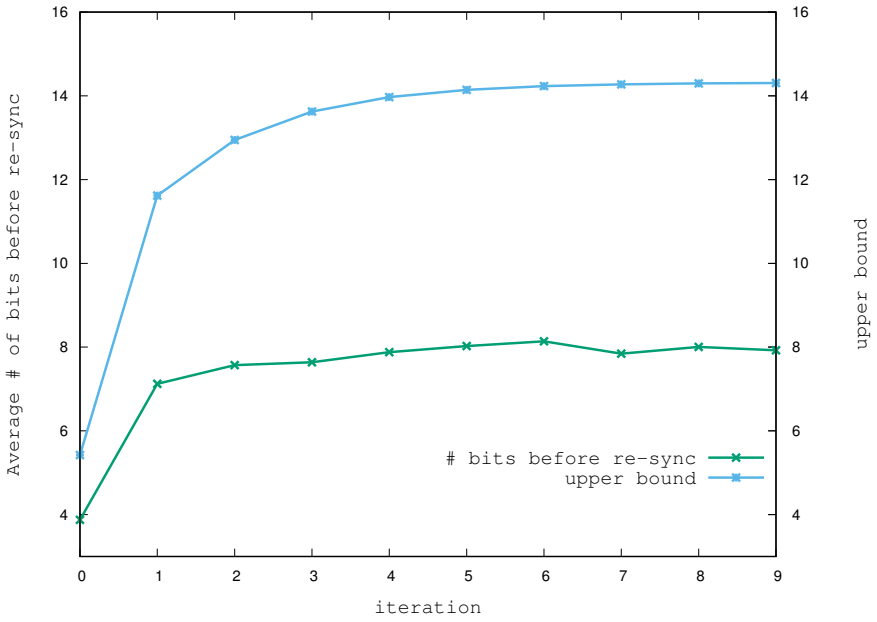

Figure 4.8: Average number of bits required to regain synchronization for the constructed $(d = 1, k = 3)$ RLL constrained sequence codes.

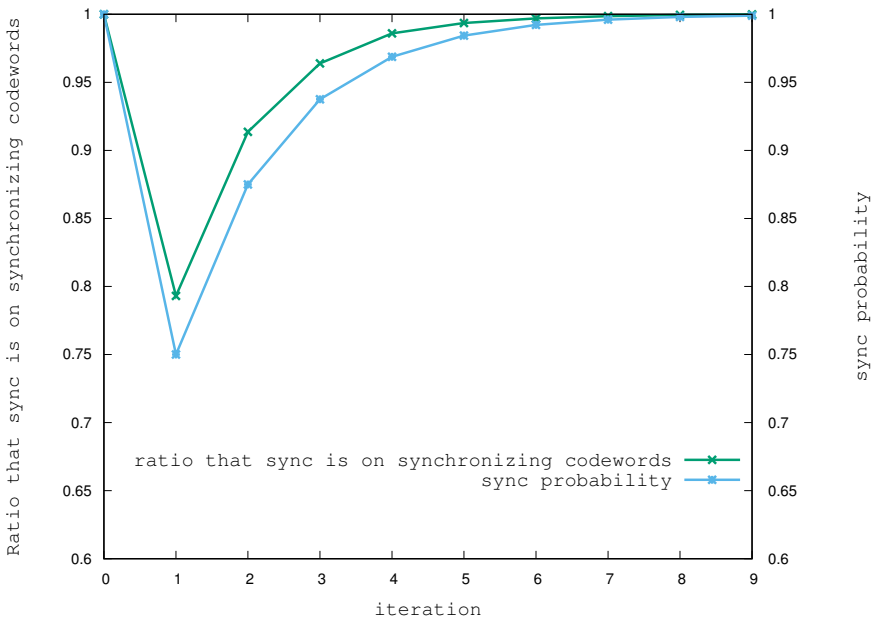

Figure 4.9: The ratio that synchronization is achieved on synchronizing codewords for the constructed $(d = 1, k = 3)$ RLL constrained sequence codebooks.

Flash memories

We consider constraints that mitigate ICI in flash memories, including the single-level cell (SLC) and multi-level cell (MLC) flash memories. For the SLC flash memories, the constraint was shown previously in Fig. 2.6; discussion in Section 4.2.1 reveals that it is sufficient to use M_1 as the minimal set. Therefore, we construct our codebooks based on state 1. The code efficiency and sync probability are shown in Fig. 4.10 where it can be seen that, similar to the situation with the $(d = 1, k = 3)$ RLL constraint, the sync probability decreases as $J = 0 \rightarrow 1$ since the synchronizing word 1 is extended, resulting in the nonsynchronizing word 11. For $J = 1 \rightarrow 9$, the sync probability increases up to 99.6%. The sync performance is shown in Figs. 4.11 and 4.12, where it is evident that on average less than one codeword and less than 8 bits are required for the receiver to regain synchronization.

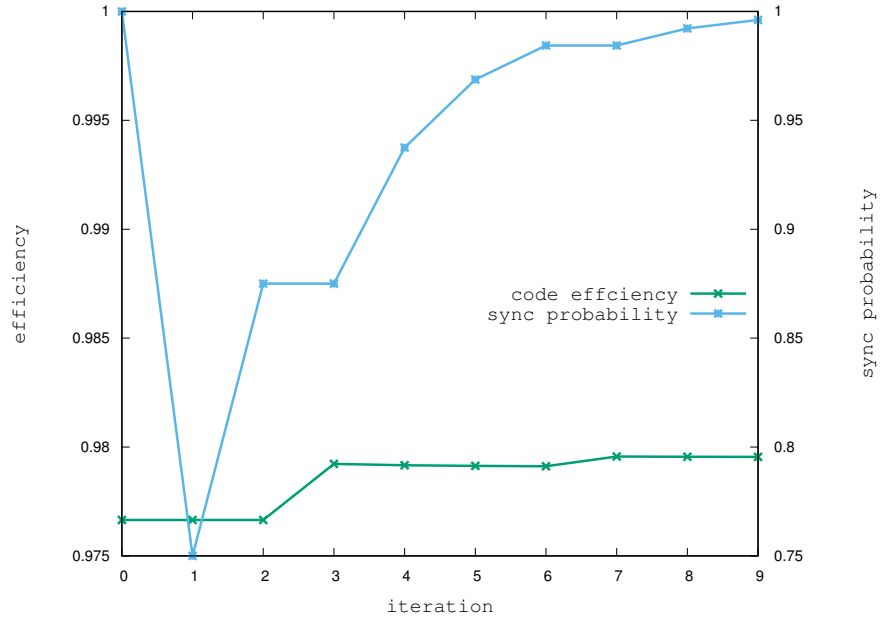

Figure 4.10: Code efficiency and sync probability of the codes for SLC flash memory.

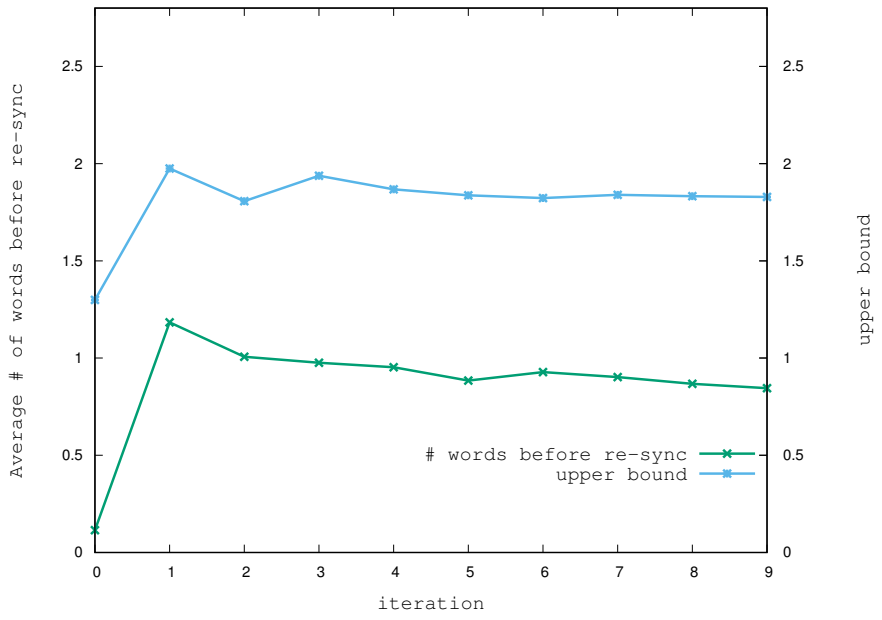

Figure 4.11: Average number of words required to regain synchronization for the codes for SLC flash memory.

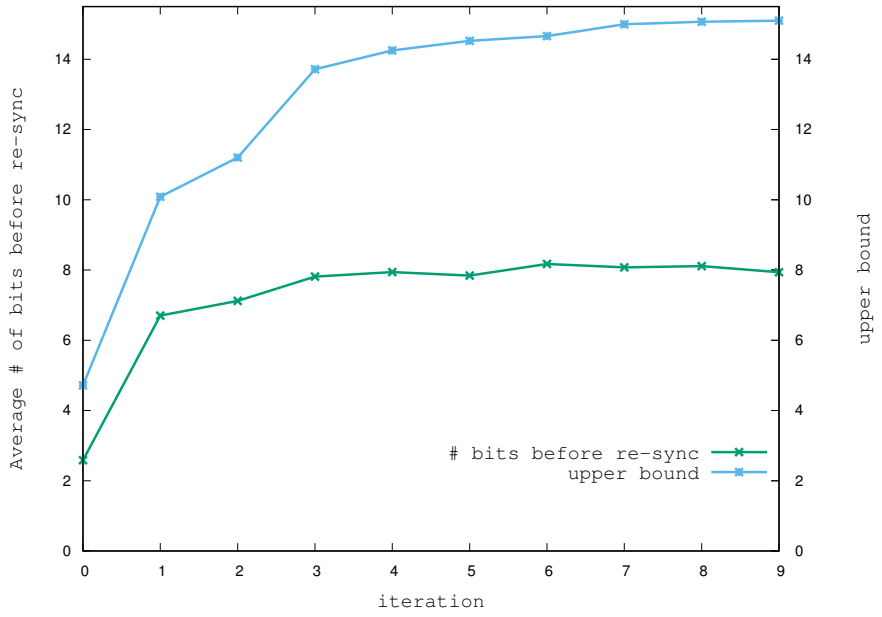

Figure 4.12: Average number of bits required to regain synchronization for the codes for SLC flash memories.

Now we show that in situations where the minimal set has low sync probability, such as when the code efficiency is of high priority and the specified state is selected according to the criteria in Section 2.1, the guided

partial extension algorithm will likely help improve the sync probability. For example, we consider MLC flash memories and the FSM of the constraint that forbids pattern 303 as shown in Fig. 4.5. The capacity of this constraint is 1.978 bit/symbol. According to our criteria, state 1 has the best sync probability, but a lower code rate than states selected according to the criteria described in Section 2.1. We instead consider selecting state 3 as the specified state which has the best maximum possible code rate, but worse sync probability compared to state 1. Note that state 3 does not satisfy Criterion 4.1, because word 0 is not a synchronizing word in M_3. If we directly perform NGH coding over M_3, the resulting codebook, shown in Table 4.2, achieves 99.6% of the capacity and has a sync probability of $P = 75\%$ since codeword 0, which occurs with 25% probability, is not a synchronizing codeword. We now show that the sync probability increases with the proposed guided partial extension algorithm.

Table 4.2: A constrained sequence codebook for ICI mitigation of MLC flash memory that achieves 99.6% of capacity

Source words	Codewords	Source words	Codewords
00	2	01	1
10	0	1100	32
1101	31	111000	302
111001	301	111010	300
111011	332	111100	331
1111010	3302	1111011	3301
1111100	3300	1111101	3332
11111100	3331	111111010	33302
111111011	33301	111111100	33300
111111101	33332	111111110	33331
11111111100	333302	11111111101	333301
111111111100	333300	111111111101	333332
111111111110	333331	1111111111110	3333302
11111111111110	3333301	11111111111111	3333300

Fig. 4.13 shows the code efficiency and sync probability of codebooks we have constructed for $J = 0 \rightarrow 9$. It can be seen that along with small increases in efficiency, the sync probability increases from 75% to 99.99%, which demonstrates the effectiveness of the proposed algorithm. Fig. 4.14 and 4.15 show the average number of codewords and the average number of bits that the decoder requires to receive to regain synchronization. It can be seen that on average, less than 0.4 codewords and less than 5 bits are needed to regain synchronization, which illustrates that the constructed codebooks have good synchronization properties. N_c is smaller than 1 because even when errors exist in the received bit sequence, Algorithm 6 usually correctly identifies the end of the current codeword.

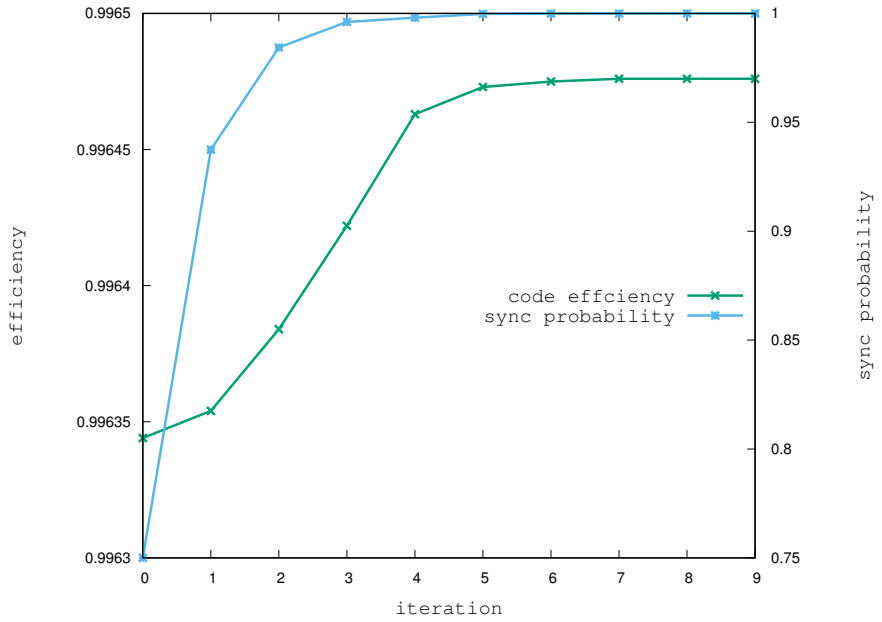

Figure 4.13: Code efficiency and sync probability of the constructed codes for MLC flash memory.

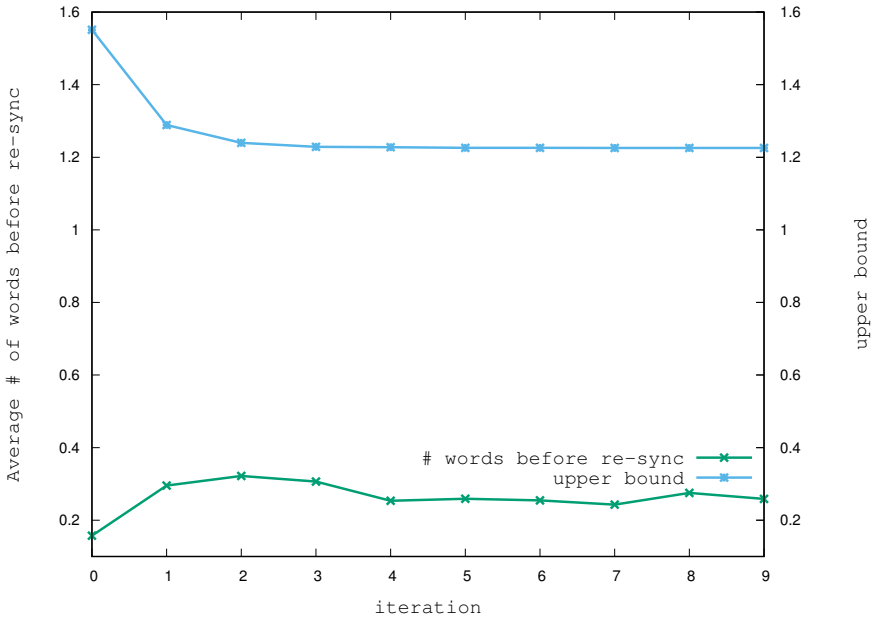

Figure 4.14: Average number of words required to regain synchronization for the constructed codes for MLC flash memory.

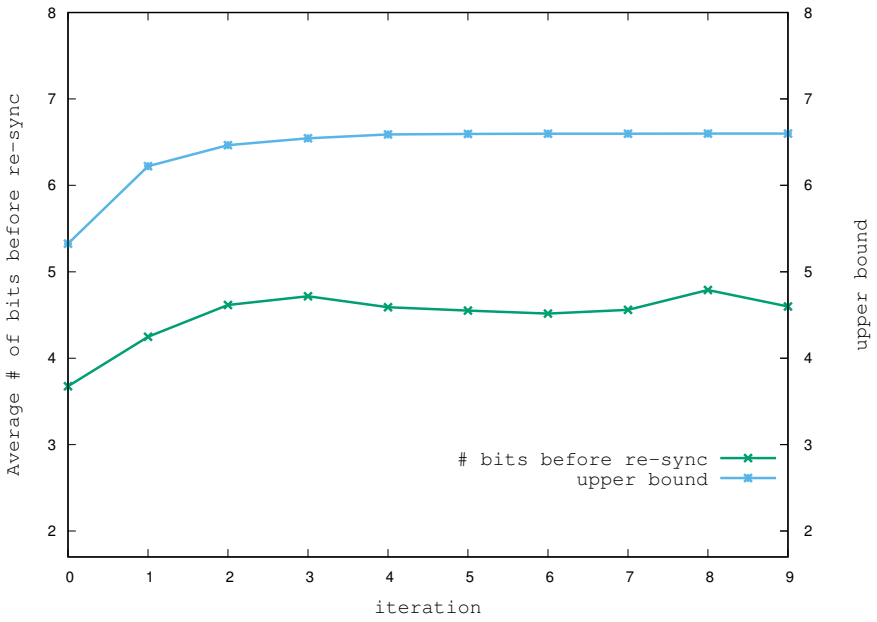

Figure 4.15: Average number of bits required to regain synchronization for the constructed codes for MLC flash memory.

The Pearson constraint

With the Pearson constraint, we present the results with codes that are constructed using state G in Fig. 3.20 as the specified state. Fig. 4.16 shows the code efficiency and sync probability of the constructed codebooks with $J = 0 \rightarrow 9$. It can be seen that the sync probability increases from 50% to 61.5%. Figs. 4.17 and 4.18 show the average number of codewords and the average number of bits that the decoder needs to receive to regain synchronization. It can be seen that even though the sync probabilities are relatively low, on average approximately one codeword and fewer than 7 bits are needed to regain synchronization.

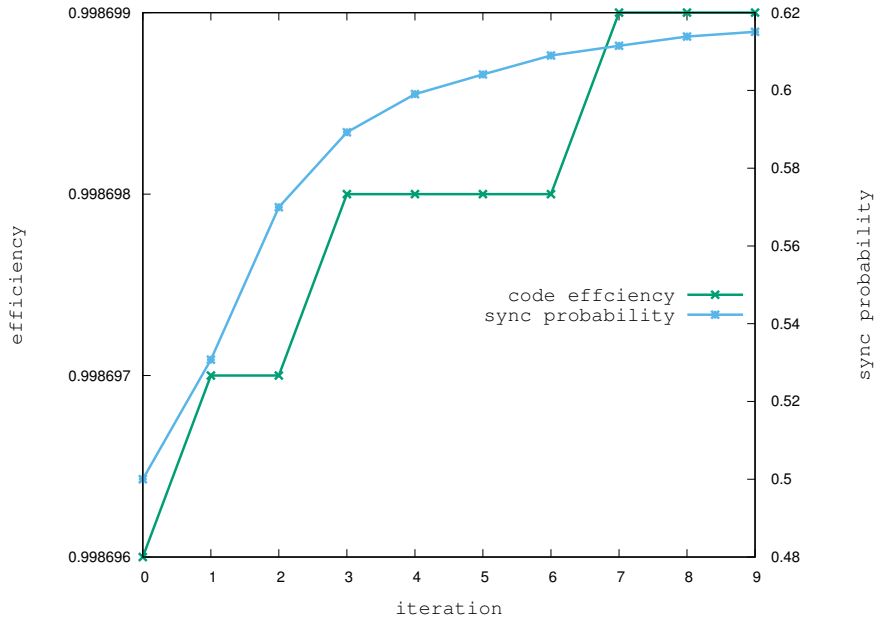

Figure 4.16: Code efficiency and sync probability of the Pearson codes.

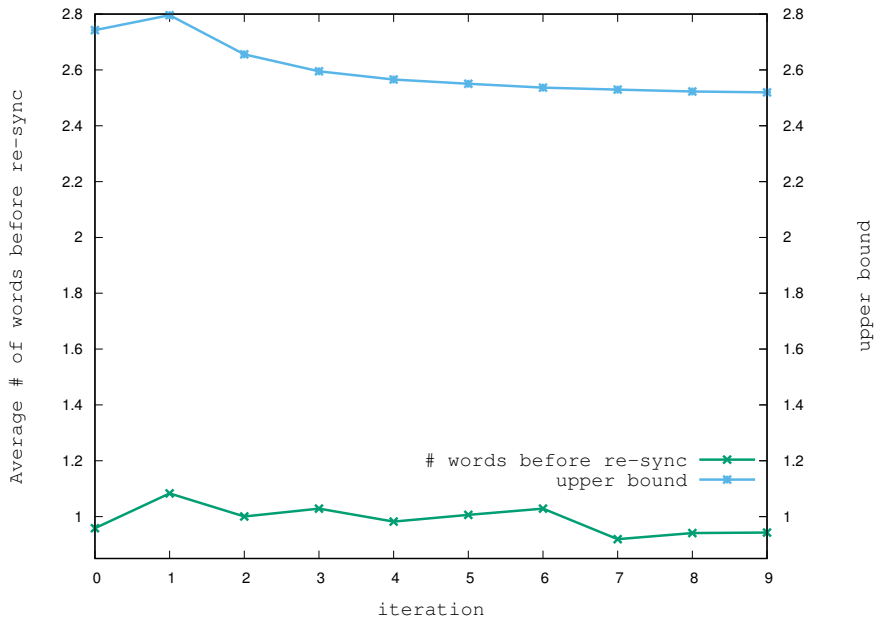

Figure 4.17: Average number of words required to regain synchronization for the constructed Pearson codes.

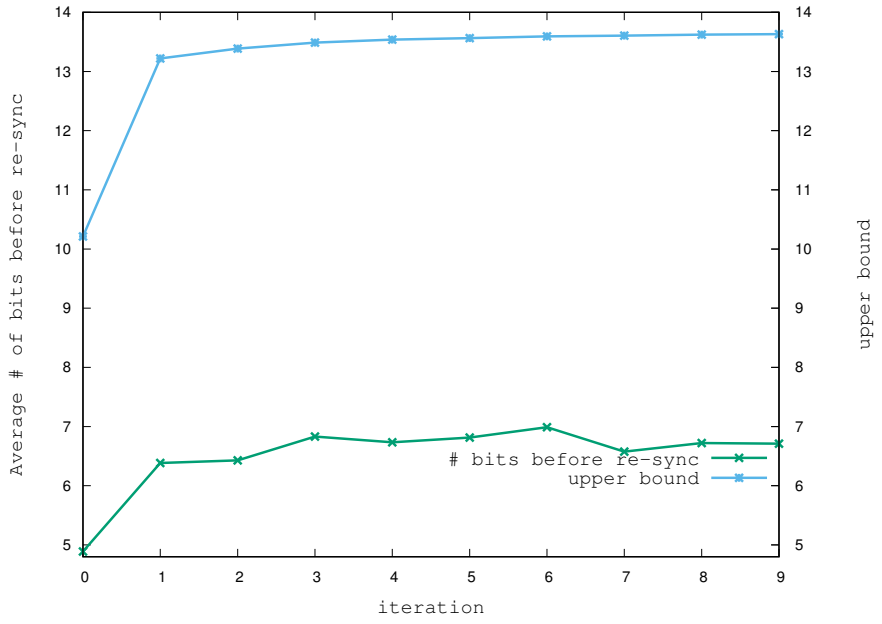

Figure 4.18: Average number of bits required to regain synchronization for the constructed Pearson codes.

The DC-free constraint

We present the results of the DC-free constraint with $N = 5$, and consider the tradeoff between code efficiency and sync probability. According to the discussion in this Section, state 1 is better than state 2, which is better than state 3 in terms of sync probability. However, according to the study in Section 2.1, the opposite is true when attempting to maximize the code rate.

In Fig. 4.19 we present the result of code efficiency and sync probability for codes constructed with states 1, 2 and 3 as the specified state, with $J = 0 \rightarrow 9$. It can be seen that the above-mentioned conclusion is verified, and the expected tradeoff between code efficiency and sync probability that arises from different states as the specified state is clearly seen. Figs. 4.20 and 4.21 show the average number of codewords and the average number of bits before the decoder regains synchronization. It can be seen that on average, between 2 to 7 codewords and between 10 to 35 bits are needed to regain synchronization. Note that $\widetilde{N}_c$ and $\widetilde{N}_b$ for states 2 and 3 are infinity since $P = 0$. However, because it is possible for the decoder to regain synchronization on nonsynchronizing codewords, these codes demonstrate relatively good synchronization properties even though $P = 0$.

To improve the synchronization properties, we use the prior knowledge that the codewords are of even length, and process two bits per decoding attempt. In this case, the sync probabilities are improved as shown in Fig. 4.22, and the average number of codewords and binary coded bits that are needed to regain synchronization are reduced as shown in Figs. 4.23 and 4.24. It can be seen that on average, approximately one codeword and fewer than 7 bits are needed to regain synchronization.

Note that in this case, the results do not indicate that the synchronization performance for state 1 > state 2 > state 3. The reason is illustrated in Fig. 4.25, and is explained as follows. Codeword 2 in the minimal set with state 1 as the specified state is not a synchronizing word since it does not satisfy Condition 4.1. However, the chances that reception of codeword 2 result in mis-synchronization is only when the quaternary bit before 2 (which can be

2 or 0) is incorrectly detected as 3 and the quaternary bit after 2 (which can be 2 or 3) is incorrectly detected as 0. The probability of this case is low, hence codeword 2 can be regarded as an "almost synchronizing codeword". It follows that the sync probability of the minimal set of state 1 can be regarded as "almost 100%". Similar reasoning holds for states 2 and 3, making their sync probability "almost 100%", and the number of codewords that are needed to regain synchronization is similar for all three states.

We also note that, with prior knowledge that all codewords have even length, in case that the receiver misses a single bit in the detection process, the above-mentioned decoding process with two bits per decoding attempt will never re-synchronize. Therefore, we propose starting the two-bit-grouping at both odd and even positions and performing decoding with both alternatives. In situations where the receiver misses a bit or mistakenly clocks in an extra bit, the decoding attempt that starts at odd positions will re-synchronize, otherwise the decoding attempt that starts at even positions will re-synchronize the received sequence.

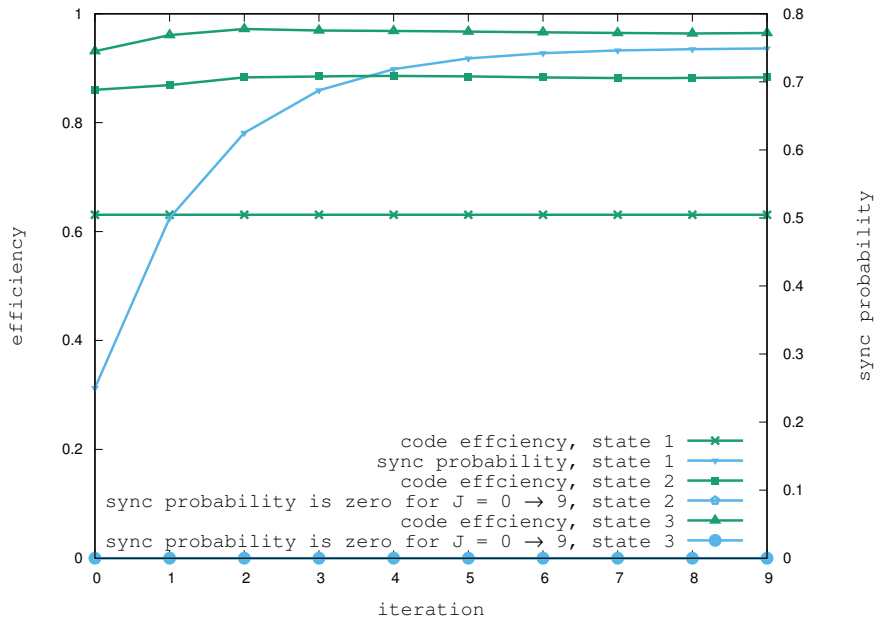

Figure 4.19: Code efficiency and sync probability of the constructed DC-free codes with $N = 5$.

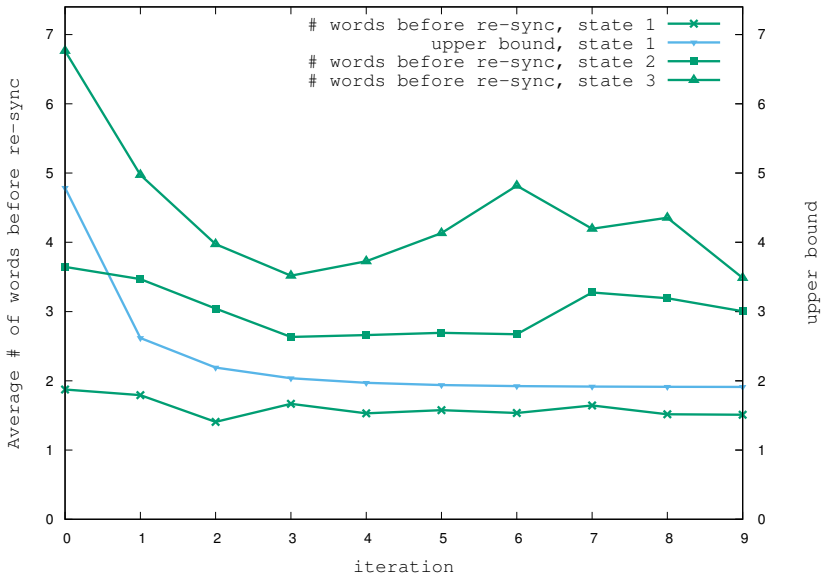

Figure 4.20: Average number of words required to regain synchronization for the constructed DC-free codes with $N = 5$. The upper bounds for states 2 and 3 are infinity since $P = 0$.

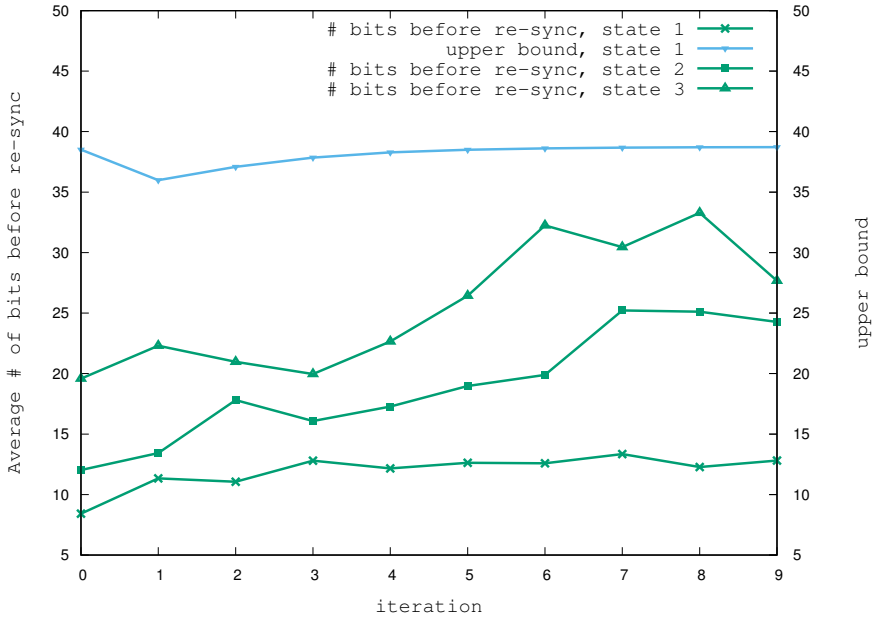

Figure 4.21: Average number of bits required to regain synchronization for the constructed DC-free codes with $N = 5$. The upper bounds for states 2 and 3 are infinity since $P = 0$.

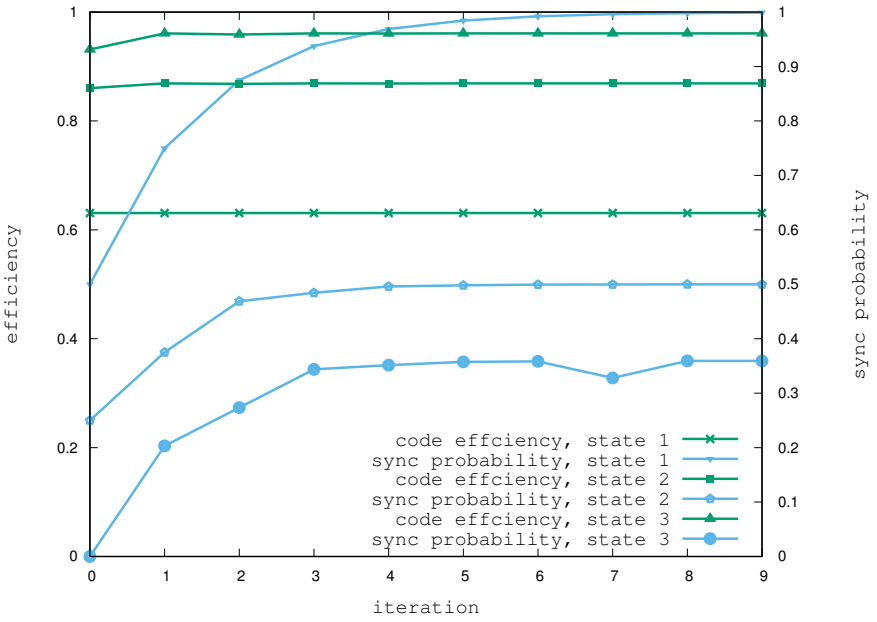

Figure 4.22: Code efficiency and sync probability of the constructed DC-free codes with $N = 5$, when the decoder has knowledge that codewords have even length.

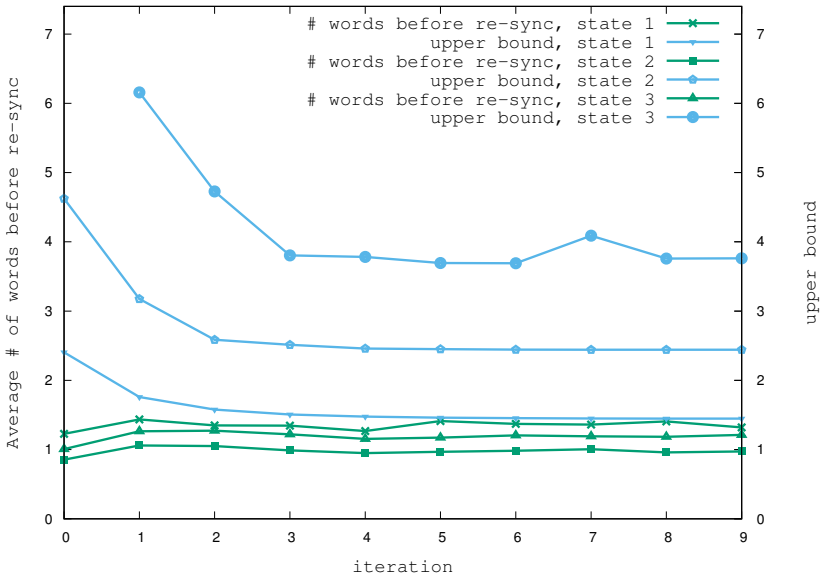

Figure 4.23: Average number of words required to regain synchronization for the constructed DC-free codes with $N = 5$, when the decoder has knowledge that codewords have even length.

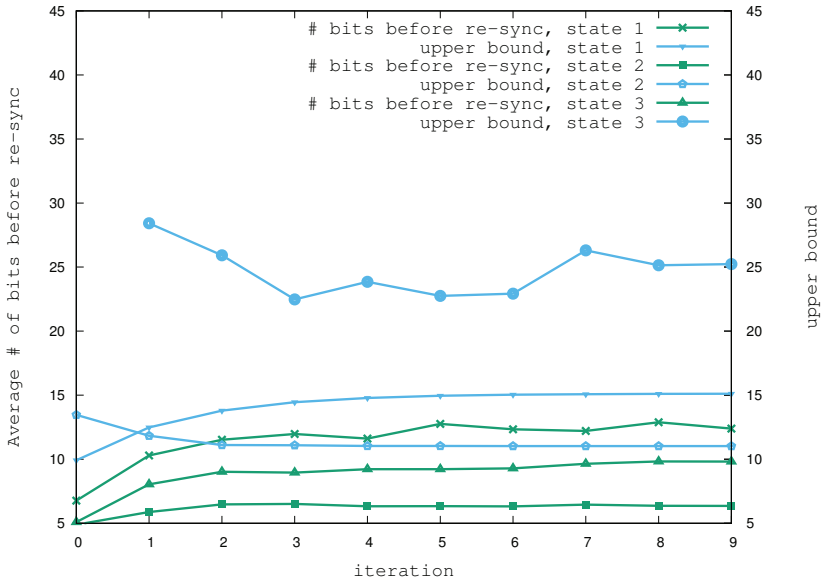

Figure 4.24: Average number of bits required to regain synchronization for the constructed DC-free codes with $N = 5$, when the decoder has knowledge that codewords have even length.

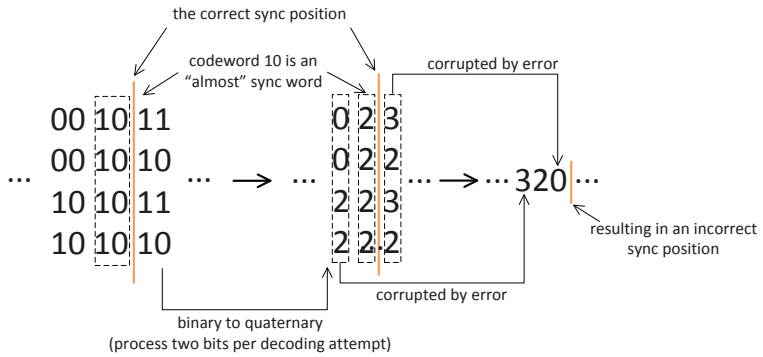

Figure 4.25: All cases that codeword 2 in the minimal set constructed based on state 1 results in mis-synchronization. Note that the probabilities of these cases are small, making codeword 2 an "almost synchronizing codeword".

Chapter 5

Conclusion

5.1 Thesis summary

In this thesis, we have discussed the design and application of variable-length constrained sequence codes. Construction of these codes consists of the following steps: building a minimal set, performing partial extensions, and NGH coding.

In Chapter 2 we studied the single-state code construction technique where the state selection criteria that result in the highest code efficiency were discussed. Then a recursive minimal set construction algorithm was developed and applied to a variety of constraints such as DC-free constraints. We also studied a multi-state code construction technique that results in high-efficiency variable-length constrained sequence codes with state-independent properties. We showed that the multi-state construction technique can result in a higher code efficiency than the single-state construction technique for some constraints.

In Chapter 3 we discussed the application of the codes constructed by the construction techniques in Section 2. In particular, we studied the inter-cell interference in flash memories and developed constrained sequence codes that mitigate inter-cell interference. Simulation results show that reduced inter-cell interference and improved BER performance can be achieved when utilizing these codes. Then we developed variable-length Pearson codes to deal

with cell leakage. Numerical results demonstrate that these codes have than conventional fixed-to-variable mapping Pearson codes.

In Chapter 4 we studied the synchronization properties of these variable-length constrained sequence codes. We demonstrated that when the primary goal is high sync probability instead of high code efficiency, new criteria need to be developed when building the minimal set. We presented these new criteria, and then we developed a guided partial extension algorithm aiming at improving the sync probability. Simulation results show that the codes we constructed have good synchronization properties such that once the receiver loses synchronization, it regains synchronization quickly within only a few codewords.

5.2 Future work

One possibility for future work is to consider the design of variable-length constrained sequence codes for use with full-sequence programming in flash memory. Instead of applying binary constrained sequence codes on each of the pages as was described in the case of multi-page programming in Chapter 3, one could consider designing non-binary constrained sequence codes and applying them with full-sequence programming where each cell is programmed exactly once to achieve the desired voltage. This would involve the consideration of additional constraints that would result in different levels of ICI mitigation. For example, in TLC flash memory, pattern 707 is the pattern that corresponds to the most severe ICI, and it is also desired to forbid patterns 706, 607 and 606, etc. A more complex constraint would involve forbidding some or all of these patterns, which would result in different levels of ICI mitigation. Construction of these constraints would involve developing FSMs that describe the constraints, calculating their capacity, and then constructing codes for those constraints based on the algorithms proposed in this thesis.

A second goal is to improve the error rate performance of systems using constrained sequence codes. One promising direction is to apply machine learning methods in constrained sequence decoding. With fixed-length codes,

it is promising to use a multiple layer perception network or a convolutional neural network to decode capacity-approaching fixed-length codes that results in bit error rates close to maximum a posteriori (MAP) decoding. With variable-length codes, it is possible to perform one-shot batch-processing of variable-length CS codes such that an entire sequence is decoded at once instead of using bit-by-bit decoding as we described in Chapter 4. As has been shown in our preliminary results [64, 65], it is possible to improve the error rate performance of constrained sequence codes by making use of soft information and global information at the receiver.

Another possibility for improving the error rate performance is to consider the integration of error control codes and constrained sequence codes. Error control codes and constrained sequence codes are usually designed separately in a system. As illustrated in Fig. 1.1, the source information is first encoded by an error control encoder and then by a constrained sequence encoder so that the characteristics of the coded bit stream match the physical constraint of the channel. At the receiver, the bit stream is first decoded by a constrained sequence decoder and then an error control decoder. This approach has two drawbacks. First, the constrained sequence decoder is often not able to exploit the soft information of channel bits; second, the constrained sequence decoder may cause error propagation, which has significant impact on the error control decoder. We would like to investigate integration of the two codes in a way that the output bit steam of the channel is sent into the error control decoder first, and then sent into the constrained sequence decoder while the order of two encoders stays the same. In this way, soft information from the channel can be exploited, the error propagation problem could be alleviated, and the input bit stream of the channel still matches the channel constraint.

Depending on different types of constraints, many methods can be investigated to accomplish this reversal. One general method that may worth investigating is to use systematic error control codes [62]. First, the source bits $\mathcal{S}$ are encoded with a constrained sequence encoder to generate a sequence $\mathcal{S}_1'$ that satisfies the constraint, and then a systematic error control code is used to encode $\mathcal{S}_1'$ and generate the redundant bits $\mathcal{R}$. Then,

constrained sequence coding is performed over $\mathcal{R}$ to generate $\mathcal{S}_2'$. $\mathcal{S}_1'$ and $\mathcal{S}_2'$ are transmitted over the channel. At the receiver side, the decoder first recovers $\mathcal{R}$ with constrained sequence decoding, and then performs error control decoding using the redundant bits $\mathcal{R}$ to recover $\mathcal{S}_1'$. Finally constrained sequence decoding is performed over $\mathcal{S}_1'$ to recover $\mathcal{S}$. In this way, error control decoding is performed before the constrained sequence decoding of source bits.

References

[1] K. W. Cattermole, "Principles of digital line coding," *International Journal of Electronics*, 1983, 55(1), pp. 3-33.

[2] A. M. Fouladgar, O. Simeone and E. Erkip, "Constrained codes for joint energy and information transfer," *IEEE Transactions on Communications*, vol. 62, no. 6, pp. 2121-2131, 2014.

[3] IEEE standard for local and metropolitan area networks-part 15.7: short-range wireless optical communication using visible light, *IEEE Standard 802.15.7*, 2011, pp. 248-271.

[4] F. Sala, K. A. Schouhamer Immink, L. Dolecek, "Error control schemes for modern flash memories: solutions for flash deficiencies," *IEEE Consumer Electronics Magazine*, vol. 4, no. 1, pp. 66-73, 2015.

[5] A. Jiang, J. Bruck, H. Li, "Constrained codes for phase-change memories," *IEEE Information Theory Workshop (ITW)*, Dublin, 2010, pp. 1-5.

[6] K. A. Schouhamer Immink and K. Cai, "Design of capacity-approaching constrained codes for DNA-based storage systems," *IEEE Communications Letters*, vol. 22, pp. 224-227, 2018.

[7] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, pp. 379-423, 1948.

[8] C. Berrou, A. Glavieux, "Near optimum error correcting coding and decoding: Turbo-codes," *IEEE Transactions on Communications*, vol. 44, no. 10, pp.1261-1271, 1996.

[9] R. G. Gallager, "Low-density parity-check codes," MIT press, 1963.

[10] A. Shokrollahi, "Raptor codes," *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp.2551-2567, 2006.

[11] E. Arikan, "Channel polarization: a method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on Information Theory*, vol. 55, no. 7, pp. 3051-3073, 2009.

[12] K. A. S. Immink, *Codes for mass data storage systems,* The Netherlands: Shannon Foundation Publishers, 2004.

[13] S.W. McLaughlin, J. Luo, and Q. Xie, "On the capacity of m-ary runlength-limited codes," *IEEE Transactions on Information Theory*, vol. 41, no. 5, pp. 1508-1511, 1995.

[14] K. A. Schouhamer Immink and H. Ogawa, "Method for encoding binary data," US Patent 4,501,000, 1985.

[15] K. A. Schouhamer Immink and U. Gross, "Optimization of low-frequency properties of eight-to-fourteen modulation," *The Radio and Electronic Engineer*, vol. 53, no. 2, pp. 36-66, 1983.

[16] G. V. Jacoby, "A new look-ahead code for increasing data density," *IEEE Transactions on Magnetics*, vol. MAG-13, no. 5, pp.1202-1204, 1977.

[17] T. D. Howell, "Analysis of correctable errors in the IBM 3380 disk file," *IBM Journal of Research & Development*, vol 28, no. 2, pp. 206-211, 1984.

[18] C. Cao and I. Fair, "Construction of minimal sets for capacity-approaching variable-length constrained sequence codes," *2016 Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, USA, 2016, pp. 255–259.

[19] C. Cao and I. Fair, "Minimal sets for capacity-approaching variable-length constrained sequence codes," *IEEE Transactions on Communications*, September 2018, vol. 67, no. 2, pp. 890-902, 2019.

[20] C. Cao and I. Fair, "Mitigation of inter-cell interference in flash memory with capacity-approaching variable-length constrained sequence codes," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 9, pp. 2366-2377, 2016.

[21] C. Cao and I. Fair, "Capacity-approaching variable-length Pearson codes," *IEEE Communications Letters*, vol. 22, no. 7, pp. 1310-1313, 2018.

[22] C. Cao and I. Fair, "Multi-state encoding of capacity-approaching variable-length constrained sequence codes with state-independent decoding," *IEEE Access*, vol. 7, pp. 54746-54759, 2019.

[23] J. K. Sundararajan, D. Shah, M. Medard, "Network coding meets TCP: Theory and implementation," *Proceedings of the IEEE*, vol. 99, no. 3, pp. 490-512, 2011.

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

[25] A. Steadman, I. Fair, "Simplified search and construction of capacity-approaching variable-length constrained sequence codes," *IET Communications*, vol. 10, no. 14, pp. 1697-1704, 2016.

[26] G. Böcherer, "Capacity-achieving probabilistic shaping for noisy and noiseless channels," PhD thesis, RWTH Aachen University, 2012. [Online]. Available: http://www.georg-boecherer.de/capacityAchievingShaping.pdf.

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

[28] E. Hemo and Y. Cassuto, "D-imbalance WOM codes for reduced inter-cell interference in multi-level NVMs," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 9, pp. 2378-2390, 2016.

[29] V. Taranalli, H. Uchikawa, and P. H. Siegel, "Error analysis and inter-cell interference mitigation in multi-level cell flash memories," *IEEE International Conference on Communications (ICC)*, London, 2015, pp. 271-276.

[30] Y. M. Chee, C. Johan, H. M. Kiah, S. Ling, T. T. Nguyen, and V. K. Vu, "Efficient encoding/decoding of capacity-achieving constant-composition ici-free codes," *2016 IEEE International Symposium on Information Theory (ISIT)*, Barcelona, 2016, pp. 205-209.

[31] S. Buzaglo, P. H. Siegel, "Row-by-row coding schemes for inter-cell interference in flash memory," *IEEE Transactions on Communications*, vol. 65, no. 10, pp. 4101-4113, 2017.

[32] H. Wang and S. Kim, "New RLL decoding algorithm for multiple candidates in visible light communication," *IEEE Photonics Technology Letters*, vol. 27, no. 1, pp. 15-17, 2015.

[33] C. E. Mejia, C. N. Georghiades, M. M. Abdallah and Y. H. Al-Badarneh, "Code design for flicker mitigation in visible light communications using finite state machines," *IEEE Transactions on Communications*, vol. 65, no. 5, pp. 2091-2100, 2017.

[34] C. E. Mejia, C. N. Georghiades and Y. H. Al-Badarneh, "Code design in visible light communications using color-shift-keying constellations," *Proceedings of the 2016 IEEE Global Communications Conference (GLOBECOM)*, Washington, DC, 2016, pp. 1-7.

[35] G. M. Church, Y. Gao, and S. Kosuri, "Next-generation digital information storage in DNA," *Science*, vol. 337, no. 6012, pp. 1628-1628, 2012.

[36] P. A. Franaszek, "Sequence-state encoding for digital transmission," *Bell System Technical Journal*, vol. 47, pp. 143-157, 1968.

[37] M. Béal, "The method of poles: a coding method for constrained channels," *IEEE Transactions on Information Theory*, vol. 36, no. 4, pp. 763-772, 1990.

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

[39] I. Fair, Y. Zhu and A. Hughes, "Spectra of multimode coded signals," *IEE Proceedings - Communications*, vol. 153, no. 3, pp. 383-391, 2006.

[40] M. Qin, E. Yaakobi, and P. H. Siegel, "Constrained codes that mitigate inter-cell interference in read/write cycles for flash memories," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 836-846, 2014.

[41] Y. Cai, E. F. Haratsch, O. Mutlu et al, "Error patterns in mlc nand flash memory: measurement, characterization and analysis," *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, March 2012, pp. 521-526.

[42] G. Dong, S. Li, T. Zhang, "Using data postcompensation and predistortion to tolerate cell-to-cell interference in MLC NAND flash memory," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol.57, no.10, pp. 2718-2728, 2010.

[43] S. K. Lai, "Flash memories: Successes and challenges," *IBM Journal of Research and Development*, vol. 52 , no. 4.5, pp. 529-535, 2008.

[44] L. Dolecek and A. Jiang, "Coding methods for emerging storage systems," *Asilomar Conference Tutorial*, Pacific Grove, CA, 2012.

[45] A. Berman and Y. Birk, "Constrained flash memory programming," *IEEE International Symposium on Information Theory (ISIT)*, St. Petersburg, 2011, pp. 2128-2132.

[46] A. Berman and Y. Birk, "Low-complexity two-dimensional data encoding for memory inter-cell interference reduction," *2012 IEEE 27th Convention of Electrical & Electronics Engineers in Israel (IEEEI)*, Eilat, November 2012, pp. 1-5.

[47] Y. Kim, B. Kumar, K. L. Cho, H. Son, J. Kim, J. J. Kong, and J. Lee, "Modulation coding for flash memories," *2013 International Conference on Computing, Networking and Communications (ICNC)*, San Diego, CA, 2013, pp. 961-967.

[48] V. Taranalli, H. Uchikawa and P. H. Siegel, "Error analysis and inter-cell interference mitigation in multi-level cell flash memories," *2015 IEEE International Conference on Communications (ICC)*, London, 2015, pp. 271-276.

[49] R. Motwani, "Hierarchical constrained coding for floating-gate to floating-gate coupling mitigation in flash memory," *2011 IEEE Global Telecommunications Conference (GLOBECOM 2011)*, Houston, TX, USA, 2011, pp. 1-5.

[50] R. Motwani and C. Ong, "Robust decoder architecture for multi-level flash memory storage channels," *2012 International Conference on Computing, Networking and Communications (ICNC)*, Maui, HI, 2012, pp. 492-496.

[51] R. Motwani and C. Ong, "Design of LDPC coding schemes for exploitation of bit error rate diversity across dies in NAND flash memory," *2013 International Conference on Computing, Networking and Communications (ICNC)*, San Diego, CA, 2013, pp. 950-954.

[52] G. Dong, N. Xie and T. Zhang, "Techniques for embracing intra-cell unbalanced bit error characteristics in MLC NAND flash memory," *2010 IEEE Globecom Workshops*, Miami, FL, 2010, pp. 1915-1920.

[53] Y. Cai, Y. Luo, E. F. Haratsch, K. Mai and O. Mutlu, "Data retention in MLC NAND flash memory: Characterization, optimization, and recovery," *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, Burlingame, CA, 2015, pp. 551-563.

[54] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, O. Unsal, A. Cristal, and K. Mai, "Neighbor-cell assisted error correction for MLC NAND flash memories," *2014 ACM international conference on Measurement and modeling of computer systems (SIGMETRICS)*, New York, NY, USA, 2014, pp. 491-504.

[55] Y. Cai, O. Mutlu, E. F. Haratsch and K. Mai, "Program interference in MLC NAND flash memory: Characterization, modeling, and mitigation," *2013 IEEE 31st International Conference on Computer Design (ICCD)*, Asheville, NC, 2013, pp. 123-130.

[56] Y. Cai, E. F. Haratsch, O. Mutlu and K. Mai, "Threshold voltage distribution in MLC NAND flash memory: Characterization, analysis, and modeling," *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, 2013, pp. 1285-1290.

[57] K. A. Schouhamer Immink, "Coding schemes for multi-level flash memories that are intrinsically resistant against unknown gain and/or offset using reference symbols," *IET Electronics Letters*, vol. 50, pp. 20-22, 2014.

[58] K. A. Schouhamer Immink and J. H. Weber, "Minimum Pearson distance detection for multi-level channels with gain and/or offset mismatch," *IEEE Transactions on Information Theory*, vol. 60, no. 10, pp. 5966-5974, 2014.

[59] J. H. Weber, K. A. Schouhamer Immink, and S. Blackburn, "Pearson codes," *IEEE Transactions on Information Theory*, vol. 62, no. 1, pp. 131-135, 2016.

[60] J. H. Weber, T. G. Swart and K. A. Schouhamer Immink, "Simple systematic Pearson coding," *2016 IEEE International Symposium on Information Theory*, Barcelona, 2016, pp. 385-389.

[61] H. Wang and S. Kim, "Soft-input soft-output run-length limited decoding for visible light communiation," *IEEE Communications Letters*, 2016, 28(3), pp. 225-228.

[62] W. G. Bliss, "Circuitry for performing error correction calculations on baseband encoded data to eliminate error propagation," *IBM Technical Disclosure Bulletin*, vol. 23, pp. 4633-4634, 1981.

[63] T. Ferguson and J. Rabinowitz, "Self-synchronizing Huffman codes," *IEEE Transactions on Information Theory*, vol. 30, no. 4, pp. 687-693, 1984.

[64] C. Cao, D. Li and I. Fair, "Deep learning-based decoding for constrained sequence codes," *2018 IEEE Global Telecommun. Conf. (GLOBECOM) workshops*, Abu Dhabi, United Arab Emirates, 2018, pp. 1-7.

[65] C. Cao, D. Li and I. Fair, "Deep learning-based decoding of constrained sequence codes," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 11, pp. 2532-2543, 2019.