Improved Coding Techniques for MPPM-like Systems

by

Siyu Liu

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science,
The Edward S. Rogers Sr. Department of
Electrical & Computer Engineering
University of Toronto

Your file Votre référence
ISBN: 978-0-494-59259-5
Our file Notre référence
ISBN: 978-0-494-59259-5

Abstract

Improved Coding Techniques for MPPM-like Systems

Siyu Liu

Master of Applied Science,

The Edward S. Rogers Sr. Department of

Electrical & Computer Engineering

University of Toronto

2009

Multipulse pulse position modulation (MPPM) has been widely proposed to improve data rate over the traditional pulse position modulation (PPM) in free-space optical communication systems. However, there is no known efficient method of encoding MPPM codewords. Furthermore, MPPM is not the optimal coding scheme (in terms of data rate) given the two main constraints of optical systems (duty cycle and zero runlength). In this work, an improved encoding technique for MPPM is provided as well as an analysis of regions where significant rate gain over MPPM is achievable. A new coding technique based on constrained coding is introduced that allows construction of codes which achieves considerable rate gain over comparable MPPM systems. In addition, our new codes allow for convenient concatenation with an outer-code and are suitable for iterative decoding. Simulation results show that these codes can achieve a 6 dB coding gain over comparable MPPM systems.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Prof. Frank R. Kschischang. It has been a truly rare pleasure to work with an advisor who not only provided me with academic guidance, but also insightful discussions on all aspects of life. His extraordinary patience and devotion to his students cannot be overstated. His dedication has made my last two years a truly wonderful life experience.

I wish to thank the members of my evaluation committee: Prof. Ben Liang, Prof. Teng Joon Lim and Prof. Li Qian for taking their time to help evaluate my thesis. Their comments, suggestions and advices have benefitted me greatly in improving this thesis.

I am deeply grateful to my colleague Ben Smith for providing me with vital research advice as well as providing me with useful computer codes. His assistance has been most valuable to the completion of this thesis. I would also like to thank David Han, Yan Liang and Mingfai Wong for proofreading my thesis and providing useful comments.

For their friendship and advice, I would like to thank the members of FRK group: Chen Feng, Azadeh Khaleghi, Sarah Lyons, Chunpo Pan, Danilo Silva, Mansoor Yousefi.

For all the joy and laughter that makes BA4162 the best office, I like to extend my thanks to my officemates: Hayssam Dahrouj, Peyman Razaghi, Soroush Tabatabaei, Ameer Youssef and Weifei Zeng.

Finally, I am, as always, grateful to my family and friends for their unwavering support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Pulse position modulation (PPM) and multipulse position modulation (MPPM) are very useful coding schemes for a wide range of optical communication systems [9]. In this work, we shall examine new coding techniques to improve achievable rates and decoding capabilities over comparable optical communication systems.

This chapter provides a brief introduction to both PPM and MPPM, highlighting the reasons that make them desirable for optical communication systems. In addition, we shall also draw attention to the limitations of PPM and MPPM. These limitations serve as our motivation to study better coding techniques.

1.1 PPM and MPPM

We shall first motivate the idea of pulse position modulation (PPM) and discuss multipulse position modulation (MPPM) as its generalization.

1.1.1 PPM

Pulse position modulation (PPM) has been shown to be a desirable modulation scheme for optical communication systems that use photon-counting techniques for direct detection

[1]. Especially for low-power, low-data-rate applications (such as deep space optical communication), PPM is known to be particularly suitable [1] [2].

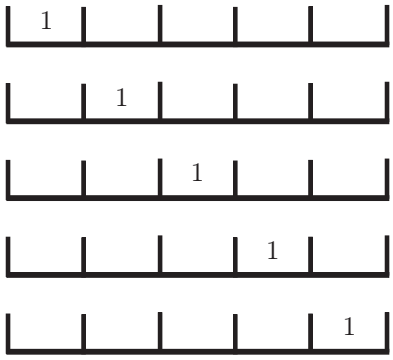

Figure 1.1: Example of $n = 5$ PPM system

For simplicity, we shall only consider binary PPM where we are restricted to sending 0 and 1. In an optical device such as a laser, a 0 corresponds to an *off* signal (the laser is turned off) while a 1 corresponds to an *on* signal (the laser is turned on). A general binary PPM codeword has n *slots*. We can view each slot as an allocation of a block of time for data transmission. In *exactly one* of the n slots, we are allowed to send a 1 with 0s in all other slots. Thus, a binary PPM codeword is identified by the position of its 1. The n slot binary PPM codebook is simply the set of all binary sequences of length n with exactly a single 1. Clearly, it has exactly n codewords. Figure 1.1 shows an example of a 5-slot PMM codebook.

As we can easily see, PPM suffers from a very low data rate. As n grows large, the number of codewords grows linearly in n while the total number of possible binary sequences of length n grows as 2^n. Even if we were to allow the alphabet size of PPM to increase from binary, it has been shown that PPM still suffers severe throughput limitations [9]. Hence, to improve on PPM, two generalizations of PPM have been studied: overlapping PPM (OPPM) [10] and multipulse or combinatorial PPM (MPPM) [11]. It has been shown that MPPM can achieve superior rates compared to both PPM and OPPM [9], and thus will serve as the basis of this work.

1.1.2 MPPM

Multipulse pulse position modulation (MPPM) was introduced in [6] as an improved modulation scheme over PPM. The idea of MPPM is simple. Instead of only allowing one pulse (or sending a 1 in our above example) per n slots, we allow exactly k of them. Figure 1.2 shows an MPPM codebook with 5 slots and allowing exactly two ($k = 2$) 1s to be transmitted in the 5 slots. This coding scheme will allow a significant rate gain over PPM even for small values of n and k. As an example, take a $n = 8$ binary PPM, we

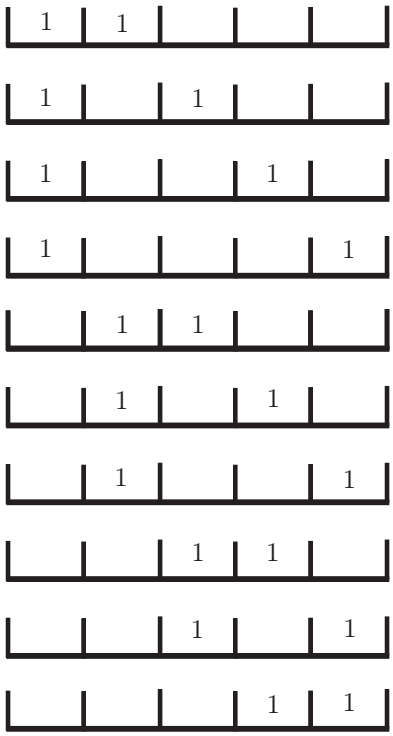

Figure 1.2: Example of $n = 5$ $k = 2$ MPPM system

have exactly 8 codewords and a code rate of $3/8 = 0.375$. Now consider MPPM with the same $n = 8$, by simply allowing $k = 2$, we have $\binom{8}{2} = 28$ codewords and a code rate of $\log 28/8 = 0.601$. Thus we see almost a 2 times rate gain by using MPPM modulation.

Besides the gain in rate, the main reason why generalizing PPM to MPPM is valid and useful is that MPPM maintains all the important characteristics that make PPM a desirable coding scheme. These important characteristics are outlined in the following

section and will serve as recurring constraints for our new approach to code design for optical communication systems.

1.2 Two Key Constraints

Comparing PPM and MPPM, we see that both schemes has the property that every codeword contains exactly the same number of 1s. We shall henceforth call the fraction of 1s the *duty cycle* of the code. We see that PPM system has a duty cycle of $1/n$ and MPPM in general as a duty cycle of k/n. This requirement translates to a maximal runlength of consecutive 0s (we shall henceforth call this "zero runlength"). Not surprisingly, MPPM shares the same characteristics as PPM. It has constant weight, duty cycle of k/n and a maximal zero runlength.

It turns out that for free-space optical systems where PPM and MPPM are used, these characteristics are of great importance. The duty cycle requirement is induced by the power constraint on the optical system. The zero runlength constraint is induced by the synchronization sensitivity of the optical system; as it has been shown that long sequences of zeros or ones are ill-suited for symbol synchronization [5] [12].

Thus, the duty cycle and zero runlength constraints are fixed by the underlying optical system and *must* be satisfied. Therefore, in order to improve on MPPM and still be applicable to the same optical system, we need to satisfy both these constraints. We shall henceforth call the duty cycle and zero runlength constraints the *two key constraints*.

1.3 Previous Works on MPPM

In a landmark paper [9], Georghiades gave an in depth analysis of the error probability, cutoff rate and capacity of PPM, OPPM and MPPM systems. He showed while both MPPM and OPPM can achieve high rates, OPPM requires large indices of overlap for high rates that make synchronization difficult in practical implementations. Furthermore,

he proposed trellis-coded modulation (TCM) to encode MPPM and showed that a 3 dB coding gain over PPM is achievable.

In [3], the maximum achievable throughputs (rates) of uncoded OPPM and MPPM are analyzed. It was concluded that MPPM should be used for high rate transmission. Specifically, MPPM is superior to OPPM if the underlying optical system allows a transmission efficiency exceeding 0.027 nats/photon. (For our purposes, this just means that MPPM is preferred over OPPM).

In [7, 8], Reed-Solomon (RS) coded MPPM was analyzed. Particularly, in [8], it was shown that RS-coded MPPM can achieve a 3 dB gain over RS-coded PPM. In [7], it was shown that RS-coded MPPM has better performance over uncoded MPPM only when the number of oness in each codeword is large.

In [4], it was shown through a design example that for very low duty cycle (hence low rate), MPPM has only an negligible gain over PPM. This result shows that we should only consider MPPM for duty cycles where significant gain over PPM can be achieved.

In [5, 12], the issue of *symbol synchronization* for MPPM systems was examined. The main result that is useful for our discussion is that long sequences of consecutive zeroes are undesirable for symbol synchronization.

In sum, previous works have shown that MPPM is the desired modulation scheme (compared to PPM and OPPM) for high data rates. Furthermore, symbol synchronization is a major concern in designing MPPM codebooks.

1.4 Drawbacks of MPPM

From the list of previous works, we can see that much analysis has been done to show the superiority of MPPM over PPM. We shall now show that there are in fact several drawbacks of MPPM on which significant improvements are desired.

Firstly, the mapping (encoding) from bit sequences to MPPM codewords is typically

done through a lookup table. On the decoding end, MPPM decoding is done through the same brute force table lookup. We observe that for even moderate size of n and k such as 20 and 10, $\binom{20}{10} = 184756$ requires a large lookup table. Thus, the lack of an efficient encoding and decoding algorithm severely limits the MPPM codebook size in practice, which in turn limits the rates we can achieve.

Secondly, we shall show in Chapter 3 that MPPM is not the optimal code (in terms of data rate) that satisfies our key conditions. Thus, it is natural to ask if we can find better codes that achieve better rates than MPPM.

Lastly, MPPM itself can be viewed as more of a modulation scheme than a coding scheme in the sense that it has very limited error correcting capabilities. Thus, it would be desirable to concatenate it with an outer error-correcting code. In a concatenated system, it would then be desirable to take advantage of iterative decoding, i.e., the so-called Soft-In-Soft-Out (SISO) decoding technique. However, the brute force mapping of MPPM codewords is ill-suited for iterative decoding in concatenation with an outer error-correcting code.

With these drawbacks in mind, we would like to look for coding systems that can improve over MPPM in the aforementioned areas while still satisfying our two key constraints. We shall list the three areas of improvements as follows:

1. efficient mapping algorithm;

2. higher rate;

3. suitable for iterative decoding.

In this thesis, we shall show that we can indeed find such coding systems. Furthermore, we shall provide both the construction and analysis of such systems in concatenation of outer error-correcting codes and demonstrate their superiority over MPPM.

1.5 Overview of Thesis

The thesis is organized as follows.

In Chapter 2, we shall provide a more in depth analysis of MPPM and describe a technique that allows reductions in the size of the required MPPM lookup tables.

In Chapter 3, we analyze the asymptotic rates while satisfying our 2 key conditions. We shall show that for many parameters significant gain over MPPM systems are theoretically possible.

In Chapter 4, we shall provide a constrained coding approach as an alternative to MPPM. We shall show that this construction allows us not only to achieve much better rates than MPPM, but also enables an efficient codeword mapping algorithm defined in terms of a finite state machine.

In Chapter 5, we shall use Extrinsic Information Transfer (EXIT) charts to design outer LDPC codes that can be serially concatenated with our constrained codes. Superior performance of our codes in comparison with MPPM systems is demonstrated.

In Chapter 6, we summarize our contributions, and suggest directions for future research in this area.

Chapter 2

Analysis of MPPM

In this chapter, we briefly introduce the MPPM model and provide an improved mapping algorithm that reduces the size of MPPM lookup tables.

2.1 MPPM Codebooks

MPPM is also called combinatorial PPM [11]. When restricted to the binary alphabet $\{0, 1\}$, each MPPM symbol is a $\{0, 1\}$ sequence of length n having exactly k ones and $(n - k)$ zeros. Let $M(n, k)$ denote the set of all possible MPPM codewords for a fixed pair (n, k). The size of $M(n, k)$ is given by :

$$|M(n, k)| = \binom{n}{k} = \frac{n!}{k!(n-k)!}. \tag{2.1}$$

With this formulation, we shall now state a few key definitions.

Definition 2.1.1 *A (n, k) MPPM codebook is a nonempty subset of $M(n, k)$.*

In general, we allow any subset of $M(n, k)$ to define a MPPM codebook. We shall later outline some desirable properties that we would like our MPPM codebooks to satisfy. These properties will guide us in making judicious choices for MPPM codebooks.

Definition 2.1.2 *For a (n, k) MPPM codebook, the ratio k/n is called the duty cycle and will henceforth be denoted by $\mathcal{D}$.*

In optical systems, each transmission of a 1 translates to an *on* signal. For example, in a laser device, a 1 translates to turning the laser on for a given time slot. In this context, we can relate the duty cycle as an average power constraint determined by a given optical system.

Given a finite binary sequence x, a subsequence of z consecutive 0s (or 1s) in x is called a *run* of 0s (or 1s) and has a *runlength* of z. The *zero runlength* of a sequence x is the longest run of 0s in x. Given an (n, k) MPPM codebook, we can easily see that the zero runlength of any codeword is upper bounded by $n-k$. In a practical optical system, it is useful to think of sending MPPM codewords sequentially. Thus, we are concerned with the concatenation of MPPM codewords. In this setting, the longest run of 0s we can achieve is the concatenation of a codeword with $n-k$ 0s at the end with a codeword with $n-k$ 0s in the beginning. This particular concatenation results in a zero runlength of $2(n-k)$. This motivates the definition of zero runlength for general (n, k) MPPM codebook.

Definition 2.1.3 *Zero runlength $\mathcal{Z}$ of M(n,k) is given by $\mathcal{Z} = 2(n-k)$.*

In physical systems, it has been shown that long runs of 0s have a detrimental effect on signal synchronization [9, 12]. Thus, we can view $\mathcal{Z}$ as a constraint determined by the timing sensitivity of the physical device.

Now, given an (n, k) MPPM codebook Ω, we define its rate $C_\Omega(n, k)$, in the standard way as:

$$C_\Omega(n, k) = \frac{\log |\Omega|}{n}, \tag{2.2}$$

where log denotes the base 2 logarithm here and in all following discussions. We shall drop the subscript of Ω when there is no confusion over the particular (n, k) codebook

in question. For $\Omega = M(n,k)$, we can approximate $|M(n,k)|$ as follows:

$$|M(n,k)| = \binom{n}{k} = 2^{n(H(\mathcal{D})+o(1))}, \tag{2.3}$$

where $\mathcal{D} = k/n$ and $H(\mathcal{D})$ is the binary entropy function given by the formula

$$H(\mathcal{D}) = -\mathcal{D}\log\mathcal{D} - (1-\mathcal{D})\log(1-\mathcal{D}) \tag{2.4}$$

and $o(1)$ denotes the little-o notation defined by $f(x) \in o(g(x))$ if

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = 0. \tag{2.5}$$

Hence, $o(1)$ is very small and we can approximate $C(n,k) \approx H(\mathcal{D})$.

For example, suppose $k/n = p = 0.5$. Then $H(0.5) = 1$, so asymptotically $C(n,k) \approx 1$. To actually come close to this rate, however, we would need very large values for n and k. For example, picking $n = 20$ and $k = 10$ will only give us $C(20,10) \approx 0.875$, while $|M(20,10)| = 184756$, a considerably large codebook size. The difficulty with large MPPM codebooks is that there is no known efficient way of encoding input messages to MPPM codewords. We shall make the notion of encoding precise in the the next section, and analyze the standard encoding and decoding method.

2.2 Standard Encoding and Decoding

We shall first define a correspondence between an input bit sequence and an (n,k) MPPM codeword.

Definition 2.2.1 *Let $\{0,1\}^m$ be the set of all possible m-bit input messages. An encoding of a message $x \in \{0,1\}^m$ to a MPPM codeword $y \in M(n,k)$ is an injective (one-to-one) function $\mathcal{E} : \{0,1\}^m \to M(n,k)$.*

From the above definition, the injectivity assumption tells us that it is sufficient to consider a MPPM codebook that has the same cardinality as the set of input messages.

Since we are assuming the set of inputs has cardinality a power of 2 (2^m for some m), we can restrict the size of our MPPM codebook to the same power of 2. This motivates the following mapping $b : \mathbb{Z}_+ \to \mathbb{Z}_+$:

$$b(x) = 2^{\lfloor \log x \rfloor}. \tag{2.6}$$

We are generally interested in (n, k) MPPM codebooks of size $b(\binom{n}{k})$. We shall henceforth denote a (n, k) MPPM codebook of size $b(\binom{n}{k})$ by $\bar{M}(n, k)$ and let $\bar{C}(n, k)$ denote its rate. As an example, consider $\bar{M}(20, 10)$. We have $|\bar{M}(20, 10)| = 2^{17} = 131072$ and $\bar{C}(20, 10) = 17/20 = 0.85$. Note that the above map only tell us the value of $|\bar{M}(n, k)|$ and does not tell us which elements of $M(n, k)$ should be included in $\bar{M}(n, k)$. In fact there is no canonical choice of elements to be included in $\bar{M}(n, k)$. Choices for constructing $\bar{M}(n, k)$ will be discussed in the following section and the next chapter.

Now given an input message set $\{0, 1\}^m$ and a corresponding $\bar{M}(n, k)$ with $|\bar{M}(n, k)| = 2^m$, we shall analyze the complexity of the standard MPPM encoding and decoding method.

For both the input message set $\{0, 1\}^m$ and $\bar{M}(n, k)$, we can order their elements by a lexicographical ordering. More precisely, we shall define the following.

Definition 2.2.2 *Let $X \subset \{0, 1\}^n$ be a nonempty set of binary sequences of length n, and let N be the cardinality of X. Define $\eta : X \to \{0, 1, \ldots, N - 1\}$ as the mapping that takes $x \in X$ to its lexicographical index in $\{0, 1, \ldots, N - 1\}$.*

Since both $\{0, 1\}^m$ and $\bar{M}(n, k)$ have the same cardinality, the lexicographical ordering on both sets gives a natural bijection between them. This bijection induces the following encoding function:

Definition 2.2.3 *Let $x \in \{0, 1\}^m$ and $y \in \bar{M}(n, k)$. The standard MPPM encoding function $\mathcal{E}_S : \{0, 1\}^m \to \bar{M}(n, k)$ is given by $\mathcal{E}_S(x) = y$ if $\eta(x) = \eta(y)$.*

With this encoding function, we need a lookup table with a list of all codewords of $\bar{M}(n, k)$ (listed in lexicographical order). This lookup table has a total of 2^m elements

Table 2.1: Complexity of Standard MPPM Encoding and Decoding

	Encoding	Decoding
Space Complexity (bits)	$2^m n$	$2^m n$
Time Complexity (operations)	$O(1)$	$O(m)$

and each element has a size of n bits. Thus, a total space of $2^m n$ bits is needed. For encoding time, we shall assume the time to look up an element takes constant time.

On the decoding side, since $\mathcal{E}_s$ is a bijection, we just need to find the inverse map. Namely, given a $\bar{M}(n,k)$ codeword y, we need to find $\eta(y)$. Towards this end, we can use the same lookup table as the encoder and perform binary search to deduce $\eta(y)$. Since there are 2^m elements on our list, we need $O(m)$ search operations to deduce $\eta(y)$.

The encoding and decoding complexity are summarized in Table 2.1.

2.3 Mapping MPPM using Integer Partition

From the above analysis, the standard encoding method results in a lookup table that has as many elements as the number of codewords in our MPPM codebook. In practice, there are physical limitations on the size of lookup tables. These limitations in turn restrict the size of an MPPM codebook. Therefore, it is desirable to find encoding methods that can handle large codebooks without large lookup tables. We shall now describe a method based on *integer partitions* that can reduce the size of MPPM lookup table. This reduction proves to be significant for moderate codebook sizes that are generally used in practice.

2.3.1 Correspondence of MPPM and Ordered Partition

Let n be a positive integer. Following the notations of [13], we define an *ordered partition* or *composition* of $n+1$ into $k+1$ parts as an *ordered* $(k+1)$-tuple $\alpha = (m_1, m_2, \ldots, m_{k+1})$ where m_i are positive integers and $m_1 + m_2 + \ldots + m_{k+1} = n + 1$. Denote the set of all ordered partitions of $n+1$ into $k+1$ parts as $\mathcal{P}(n+1, k+1)$. From basic combinatorics, we know that $|\mathcal{P}(n+1, k+1)| = \binom{n}{k}$. Thus we see that $|\mathcal{P}(n+1, k+1)| = |M(n, k)|$.

There is a natural bijection between elements of $\mathcal{P}(n+1, k+1)$ and elements of a $M(n, k)$. This bijection is given by corresponding each m_i in the composition of n with a binary sequence of zeros of length $m_i - 1$ and each $+$ sign as the position of a one. Namely, we define the following map $\mathcal{B} : \mathcal{P}(n+1, k+1) \to M(n, k)$ by:

$$\begin{aligned} \mathcal{B}(\alpha) &= \mathcal{B}(m_1, m_2, \ldots, m_{k+1}) && (2.7) \\ &= 0^{m_1-1}10^{m_2-1}1 \ldots 0^{m_k-1}10^{m_{k+1}-1} && (2.8) \end{aligned}$$

where 0^j is a string of j 0s and follows the convention that 0^0 is the empty string.

For example, the combination $2 + 2 + 1 = 5$ will correspond to the 4-bit sequence 0101. Below is an example of the set $\mathcal{P}(5, 3)$ and its corresponding $M(4, 2)$ codewords:

$\mathcal{P}(5,3)$	$M(4,2)$
$1+1+3$	1100
$1+3+1$	1001
$3+1+1$	0011
$1+2+2$	1010
$2+1+2$	0110
$2+2+1$	0101

2.3.2 Ordered Partition under Cyclic Permutation

In [13], the action of a cyclic permutation on the set $\mathcal{P}(n,k)$ is investigated. Let θ be the map on $\mathcal{P}(n,k)$ defined via

$$\theta[(m_1, m_2, \ldots, m_k)] = (m_2, m_3, \ldots, m_k, m_1). \tag{2.9}$$

It is clear that θ generates a cyclic group G_θ of order k under function composition, with $\theta^k = e$ (the identity element in the group) and $\theta^{-1} = \theta^{k-1}$. Whenever we have a group G acting on a set (also called a group action), we would naturally like to examine the set of *orbits* of G under this action. We shall define orbits of $\mathcal{P}(n,k)$ under the action of G_θ as follows.

Definition 2.3.1 *The orbit of $\alpha \in \mathcal{P}(n,k)$ under the action of G_θ is the set*

$$G_\theta \alpha = \{\alpha, \theta\alpha, \theta^2\alpha, \ldots, \theta^{k-1}\alpha\} \tag{2.10}$$

For example, the orbit of the 3-tuple $(1,0,2)$ is the set $\{(1,0,2),(2,1,0),(0,2,1)\}$, which is also the orbit of $(2,1,0)$ and $(0,2,1)$. From this example, it is easy to see that many elements can have the same orbit. In fact, orbits form an equivalence relation on $\mathcal{P}(n,k)$ and hence induce a partition of $\mathcal{P}(n,k)$. Thus, it is natural to determine the total number of orbits of $\mathcal{P}(n,k)$.

For our given group action on $\mathcal{P}(n,k)$, define its set of orbits as $\mathcal{O}(n,k)$ and further, denote the set of orbits with exactly k-elements as $\mathcal{O}_k(n,k)$. It has been proven in [13] that the size of the orbits follows the following elegant formulae:

$$|\mathcal{O}_k(n,k)| = \frac{1}{n} \sum_{d|\{n,k\}} \mu(d) \binom{n/d}{k/d} \tag{2.11}$$

and

$$|\mathcal{O}(n,m)| = \frac{1}{n} \sum_{d|\{n,k\}} \phi(d) \binom{n/d}{k/d}, \tag{2.12}$$

where μ is the Möbius function and ϕ is the Euler Totient function defined below, and the sum is taken over all positive common divisors of n and k.

Definition 2.3.2 *Let n be a positive integer. The Möbius function is defined as*

$$\mu(n) = \begin{cases} 1 & \text{if } n \text{ is square-free with an even number of distinct prime factors;} \\ -1 & \text{if } n \text{ is square-free with an odd number of distinct prime factors;} \\ 0 & \text{if } n \text{ is not square-free.} \end{cases} \tag{2.13}$$

Putting in more a compact form: let $\omega(n)$ be the number of distinct primes dividing n and $\Omega(n)$ be the number of prime factors of n, counted with multiplicities (clearly, $\omega(n) \leq \Omega(n)$). Then:

$$\mu(n) = \begin{cases} (-1)^{\omega(n)} = (-1)^{\Omega(n)} & \text{if } \omega(n) = \Omega(n); \\ 0 & \text{if } \omega(n) < \Omega(n). \end{cases} \tag{2.14}$$

The Euler totient function, $\phi(n)$, is defined as the number of positive integers less or equal to n and *relatively prime* to n. The totient function is deeply related to the Möbius function via the Möbius inversion formula:

$$\phi(n) = \sum_{d|n} d \cdot \mu(\frac{n}{d}). \tag{2.15}$$

Now (2.11) is particularly interesting to us. Let $M_{\mathcal{O}_k}(n,k)$ be the union of all the orbits in $\mathcal{O}_k(n,k)$. Then $|M_{\mathcal{O}_k}(n,k)| = k \cdot |\mathcal{O}_k(n,k)|$. Now, we consider $M_{\mathcal{O}_{k+1}}(n+1,k+1)$, keeping in mind that elements of $M_{\mathcal{O}_{k+1}}(n+1,k+1)$ are in natural correspondence with elements of $M(n,k)$. Now if$|M_{\mathcal{O}_{k+1}}(n+1,k+1)| \geq |\bar{M}(n,k)|$, then we can take a subset of $M_{\mathcal{O}_{k+1}}(n+1,k+1)$ to generate a codebook of size $|\bar{M}(n,k)|$. To see how we can achieve this algorithmically, we first need the following definition.

Definition 2.3.3 *Suppose that $|M_{\mathcal{O}_{k+1}}(n+1,k+1)| = (k+1) \cdot \mathcal{O}_{k+1}(n+1,k+1) \geq |\bar{M}(n,k)|$. Let p be the smallest positive integer such that $(k+1)p \geq |\bar{M}(n,k)|$, i.e., p is such that*

$$(k+1)(p-1) < \ |\bar{M}(n,k)| \leq (k+1)p. \tag{2.16}$$

We shall now give an example illustrating the above concepts. Take $M(9,2)$, we have $|M(9,2)| = 36$ and $|\bar{M}(9,2)| = 32$. Then

$$|\mathcal{O}_3(10,3)| = \frac{1}{10}\binom{10}{3} = 12 \tag{2.17}$$

Thus, we have 12 orbits of size 3. We only require 11 of these, since $11 \times 3 > 32$. Thus, in our example, we have $p = 11$.

Now we shall give an algorithm for constructing $\bar{M}(n,k)$.

1. Via our bijection $\mathcal{B}$, we can view elements of $M_{\mathcal{O}_{k+1}}(n+1, k+1)$ as elements of $M(n,k)$.

2. From each orbit, we can select a representative element. We can make a canonical choice for the representative by picking the *smallest* element in the orbit (viewed as element of $M(n,k)$) in lexicographical order.

3. Within each orbit $\mathcal{O}_j$, we can index its elements by the action of θ on the orbit representative. More precisely, let α be the orbit representative. Each $\beta \in \mathcal{O}_j$ is given as $\beta = \theta^i \alpha$ for some $i \in \{0, 1, \ldots, k\}$, so i is the index assigned to β. This i is the assigned index of β. This method of index assignments gives the elements of each orbit an index from 0 to k.

4. Given a set of orbit representatives, we can index the orbits from 0 to $|\mathcal{O}_k(n,k)| - 1$ in *ascending* lexicographical order of the orbit representatives.

5. Construct $\bar{M}(n,k)$ with p orbits (possibly eliminating some elements from the last orbit to match the size of $\bar{M}(n,k)$).

With the above construction, we can see that every codeword of $\bar{M}(n,k)$ is completely specified by the orbit it belongs to and its position in the orbit. Thus, let a be the orbit number and r be the position in the orbit, the 2-tuple (a,r) completely specifies the codewords of $\bar{M}(n,k)$. This motivates the following mapping:

Definition 2.3.4 *Let* $\tau : \bar{M}(n,k) \to \{0,\ldots,p-1\} \times \{0,\ldots,k\}$ *be the bijection that sends the codewords of* $\bar{M}(n,k)$ *to their* 2*-tuple representations. Namely, for* $y \in \bar{M}(n,k)$, $(a,r) \in \{0,\ldots,p-1\} \times \{0,\ldots,k\}$, *our above construction defines the mapping* $\tau(y) = (a,r)$.

In the next section, we shall see that the above construction leads to an improved encoding and decoding method.

2.4 Improved Encoding and Decoding

With the particular $\bar{M}(n,k)$ given by the above construction, we can now define our improved encoding function. We shall again assume that $|\bar{M}(n,k)| = |\{0,1\}^m| = 2^m$. A key observation is that for $x \in \{0,1\}^m$, with $\eta(x) = w$ for some $w \in \{0, 2^m - 1\}$, we can represent w as:

$$\begin{aligned} \eta(x) &= w & (2.18)\\ &= a(k+1) + r, & (2.19) \end{aligned}$$

where $0 \leq r \leq k$.

Of course, it is no coincidence that we choose $(k+1)$ and the letters a and r. With this observation, we can define the following improved encoding function.

Definition 2.4.1 *Let* $x \in \{0,1\}^m$ *with* $\eta(x) = a(k+1) + r$. *The improved MPPM encoding function* $\mathcal{E}_I : \{0,1\}^m \to \bar{M}(n,k)$ *is given by* $\mathcal{E}_I(x) = y$, *where* $y \in \bar{M}(n,k)$ *and* $\tau(y) = (a,r)$.

With this improved encoding function, we see that we only need to specify a lookup table with p elements (recall p is the number of orbits used in constructing $\bar{M}(n,k)$). This will require pn bits. Note that we need not to store all the elements of a particular orbit because we can find every element in the orbit easily by knowing the orbit representative

and the r value. Thus, the encoding is essentially a table lookup together with a cyclic shift (θ^r) of an orbit representative. We shall assume that the table lookup takes constant time and a cyclic shift can be done in $O(k)$ time.

For decoding, we will still use the same lookup table with p elements. The decoding procedure can be described as follows:

1. Suppose $y \in \bar{M}(n,k)$ is received, map y to its integer partition representation y' via $\mathcal{B}^{-1}$.

2. Perform the inverse cyclic shift θ^{-1} on y' until we arrive at an orbit representative. Note that our canonical choice orbit representative $\bar{y}$ is the cyclic shift of y' that has the smallest lexicographical value. Thus we can determine the orbit representative in at most k cyclic shifts (recall θ generates a group of order $(k+1)$).

3. Let r be defined by $\theta^{-r}y' = \bar{y}$.

4. Given orbit representative $\bar{y}$, find $\bar{y}$ in our lookup table of orbit representatives. Since the orbit representatives are lexicographically ordered, we can use binary search.

5. Let a be defined by the position of $\bar{y}$ in our lookup table.

6. With r and a, we can recover our input message $x \in \{0,1\}$ by the formula $\eta(x) = a(k+1) + r$

With this decoding algorithm, we again need pn bits to store a lookup table with p elements. For time complexity, $O(k)$ operations are needed to determine a canonical orbit representative and $O(\log p)$ operations are needed to perform binary search on our lookup table.

Since we have $2^m \approx (k+1)p$, we can now compare the encoding and decoding complexity of our improved method against the standard method. This comparison is summarized in Table 2.2. In general, we have reduced the lookup table size by a factor of

Table 2.2: Complexity of standard MPPM method vs orbit partition method

	Standard MPPM Method		Orbit Partition Method	
	Input	Output	Input	Output
Space Complexity (bits)	$(k+1)pn$	$(k+1)pn$	pn	pn
Time Complexity (operations)	$O(1)$	$O(\log(kp))$	$O(k)$	$O(k)+O(\log p)$

$k+1$. Also note that in most cases we have $p \gg k$, and $\log p > k$. Thus we also have a slight gain in decoding time complexity over the standard method.

2.4.1 Example of Improved Encoding and Decoding

To further illustrate our improved encoding and decoding algorithm, recall our earlier example with $\bar{M}(9,2)$. We can now explicitly list out its 11 orbit representatives in Table 2.3. Note that $\mathcal{O}_3(10,3)$ has one more orbit $(4,3,3)$, which is eliminated. Furthermore, we only need 2 elements in the last orbit in Table 2.3 to get a total of 32 elements.

Now suppose that we received the codeword 001000001. We wish to find its a and r value. Using our decoding algorithm, map 001000001 back to its integer partition representation via $\mathcal{B}^{-1}$. We have $001000001 \rightarrow (3,6,1)$. Now, applying θ^{-2} gives us that $\theta^{-2}(3,6,1) = (1,3,6)$ Since there are only 3 elements in the orbit, we see that $(1,3,6)$ is the element with the smallest lexicographical value. Thus, it must be our canonical orbit representative. As we have used θ^{-2}, the r value is 2. Now a binary search on orbit representatives gives us $a = 5$. With these values, we see that $ak + r = 5 \times 3 + 2 = 17$. This corresponds to the 5-bit input of 10001.

2.4.2 List of Orbits for Various MPPM Systems

In the above example, we saw that the orbits of size 3 actually covers the whole set $M(9,2)$ and hence we were able to construct $\bar{M}(9,2)$. In this section, we shall show

Table 2.3: Table of orbits representatives of $\bar{M}(9,2)$

$\mathcal{O}(10,3)$	$\bar{M}(9,2)$	Index Value (a)
$(8,1,1)$	000000011	0
$(7,2,1)$	000000101	1
$(7,1,2)$	000000110	2
$(6,3,1)$	000001001	3
$(6,2,2)$	000001010	4
$(6,1,3)$	000001100	5
$(5,4,1)$	000010001	6
$(5,3,2)$	000010010	7
$(5,2,3)$	000010100	8
$(5,1,4)$	000011000	9
$(4,4,2)$	000100010	10

Table 2.4: Table of orbits with $k = 2$

n-value	$\lvert M(n,k)\rvert$	$\lvert\mathcal{O}(n+1,k+1)\rvert$	$(k+1)\times\lvert\mathcal{O}(n+1,k+1)\rvert$	$\lvert\bar{M}(n,k)\rvert$
9	36	12	36	32
12	66	22	66	64
17	136	45	135	128
24	276	92	276	256
33	528	176	528	512

Table 2.5: Table of orbits with $k = 3$

n-value	$\lvert M(n,k)\rvert$	$\lvert\mathcal{O}(n+1,k+1)\rvert$	$(k+1)\times\lvert\mathcal{O}(n+1,k+1)\rvert$	$\lvert\bar{M}(n,k)\rvert$
7	35	8	32	32
9	84	20	80	64
11	165	40	160	128
13	286	70	280	256
16	560	140	560	512
20	1140	285	1140	1024

for most practical MPPM codebooks (relatively small codebooks), $|M_{\mathcal{O}_k}(n,k)|$ is always greater than $|\bar{M}(n,m-1)|$. In Table 2.4 to 2.7, we give a list of MPPM codebooks with k values ranging from 2 to 5. In most cases, the n values are appropriately chosen so that $\binom{n}{k}$ is close to a power of 2. We see that for all cases in Table 2.4 to 2.7, we have enough orbits to construct $\bar{M}(n,k)$.

Finally, we note that this result only provides a significant gain (in terms of size of lookup table) when we have relatively small codebooks. This is because in general we can only reduce the lookup table size by a factor of k which converts to $\lfloor \log k \rfloor$ bits. To put this into perspective, recall our first example with $n = 20$, $k = 10$, we

Table 2.6: Table of orbits with $k = 4$

n-value	$\lvert M(n,k)\rvert$	$\lvert\mathcal{O}(n+1,k+1)\rvert$	$(k+1)\times\lvert\mathcal{O}(n+1,k+1)\rvert$	$\lvert\bar{M}(n,k)\rvert$
7	35	7	32	32
8	70	14	70	64
9	126	25	125	64
11	330	66	330	256
13	715	143	715	512
15	1365	273	1365	1024

Table 2.7: Table of orbits with $k = 5$

n-value	$\lvert M(n,k)\rvert$	$\lvert\mathcal{O}(n+1,k+1)\rvert$	$(k+1)\times\lvert\mathcal{O}(n+1,k+1)\rvert$	$\lvert\bar{M}(n,k)\rvert$
10	252	42	252	128
13	1287	212	1272	1024
16	4368	728	4368	4096
18	8568	1428	8568	8192
23	33649	5598	33588	32768
26	65780	10962	65772	65536

have $\bar{M}(20, 10) = 2^{17}$, while $\lfloor \log 10 \rfloor = 3$. Hence, we still need a lookup table of size $2^{17}/10 \approx 13108$ or 14-bits (rounding up to the nearest bit value). Therefore, while useful, this mapping algorithm still does not allow us to construct large codebooks that approach the asymptotic rate of $H(k/n)$.

Chapter 3

Theoretical Rate Improvements over MPPM

In this chapter, we shall examine theoretical rate improvements over MPPM while satisfying the same duty cycle and zero runlength constraints as an MPPM codebook.

3.1 Fixed Duty Cycle and Zero Run-length Constraint

As mentioned earlier, both the duty cycle and the zero runlength constraint are imposed by the optical system. The duty cycle constraint translates to an average power constraint, while the zero runlength constraint corresponds to the fact that long sequences of zeros are undesirable for synchronization purposes [9]. These two constraints exactly fix the parameters of MPPM (i.e., the n and k values). Thus, it is useful to examine if we can achieve a rate gain by considering other modulation schemes while keeping both $\mathcal{D}$ and $\mathcal{Z}$ constant.

The first observation is that, in order to keep the $\mathcal{D}$ parameter constant, it suffices to keep the ratio k/n constant. Thus, it is natural to investigate how much improvement one can make by increasing n and k simultaneously while keeping the same $\mathcal{D}$ and $\mathcal{Z}$. First, we shall make the following definition:

Definition 3.1.1 *Let $x \in \{0,1\}^n$. Let $wt(x)$ denote the number of 1s in x and $z(x)$ denote the zero runlength of x. We define the set $S(n,k,\mathcal{Z})$ as:*

$$S(n,k,\mathcal{Z}) = \{x \in \{0,1\}^n : wt(x) = k, z(x) \leq \mathcal{Z}\}. \tag{3.1}$$

A simple example shows that we can gain quite a bit by increasing n and k. Suppose we start with $M(4,2)$. We have $|M(4,2)| = 6$, $\mathcal{D} = 50\%$ and $\mathcal{Z} = 2$. Now we examine the effect of increasing n and k simultaneously. Clearly, if we set $n = 6$ and $k = 3$, we still satisfy the same duty cycle. Now we ask how many sequences of length $n = 6$, $k = 3$ has a zero runlength of $\mathcal{Z} = 2$? Namely, we would like to find $|S(6,3,2)|$. Towards this end, we can, in a brute force manner, list out all such sequences as in Table 3.1.

As we can count, there are 16 codewords of $S(6,3,2)$. To make a fair comparison, let us concatenate $M(4,2)$ sequentially. A concatenation of 3 consecutive $M(4,2)$ codewords is a sequence of length 12; and this concatenated codebook has a total of $|M(4,2)|^3 = 216$ codewords. Likewise, we can concatenate $S(6,3,2)$ sequentially. A concatenation of 2 consecutive $S(6,3,2)$ codewords is also a sequence of length 12. This concatenated codebook, however, contains a total of $|S(6,3,2)|^2 = 256$ codewords. Thus, we gain 40 codewords over a length of 12 by simply increasing both n and k by 1!

If we keep increasing n and k, we quickly find that we cannot list the codebook by a brute force manner. Unfortunately, there is no known asymptotic formula for $S(n,k,\mathcal{Z})$. There is, however, a method to analyze this problem for reasonably large n and k values. This method is related to yet another integer partition problem.

3.1.1 Counting using Integer Composition

We shall now show that counting codewords of $S(n,k,\mathcal{Z})$ is related to counting the number of compositions of a positive integer n into k parts with each part $\leq \mathcal{Z}+1$. Namely, $n = m_1 + m_2 + \ldots + m_k, m_i \leq \mathcal{Z}+1, \forall i$. We can think of each part m_i as a binary sequence starting with one followed by $|m_i| - 1$ zeros. This combinatorial problem

Table 3.1: List of Codewords of S(6,3,2)

$n = 6, k = 3, \mathcal{Z} = 2$
001011
001101
001110
010011
010101
010110
011001
011010
011100
100101
100110
101001
101010
101100
110010
110100

has a well-known generating function given by [14]:

$$G(x) = (x + x^2 + \ldots + x^{\mathcal{Z}+1})^k. \tag{3.2}$$

Let a_k denote the coefficient of x^k. We see that the coefficient a_n corresponds to the number of binary sequences of length n that begin with a one; and in general, the coefficient a_{n-j} corresponds to the number of binary sequences of length n that begin with exactly j zeros. In our case, we allow up to $\mathcal{Z}$ zeros. Thus, the total number of sequences is given by the following formula:

$$|S(n, k, \mathcal{Z})| = \sum_{i=0}^{\mathcal{Z}} a_{n-i}. \tag{3.3}$$

Now, looking back at our $S(6, 3, 2)$ example. Take our generating function approach, we have:

$$(x + x^2 + x^3)^3 = x^9 + 3x^8 + 6x^7 + 7x^6 + 6x^5 + 3x^4 + x^3 \tag{3.4}$$

Since $n = 6$ and $\mathcal{Z} = 2$, we need to sum the coefficients from x^4 to x^6. Thus, $|S(6, 3, 2)|$ is given by:

$$\begin{aligned} |S(6, 3, 2)| &= a_6 + a_5 + a_4 && (3.5)\\ &= 7 + 6 + 3 && (3.6)\\ &= 16, && (3.7) \end{aligned}$$

exactly the same as our brute force approach. Furthermore, referring back to Table 3.1, there are indeed $a_4 = 3$ sequences that begin with two consecutive 0s, $a_5 = 6$ sequences that begin with one 0 and and $a_6 = 7$ sequences that begin with a 1.

3.1.2 Rate Improvements for Various Duty Cycle and Zero Run-length

Now equipped with the above generating function method, we can investigate the effect of increasing n and k simultaneously. In each case, we shall start with a base MPPM

system that induces a duty cycle $\mathcal{D}$ and zero runlength $\mathcal{Z}$. In Figures 3.1 to 3.4, we start with $M(20, 2)$, $M(10, 2)$, $M(10, 3)$, $M(10, 4)$ respectively. These give us duty cycles of 10%, 20%, 30% and 40%. In each case, we let n increase to the nearest value less than 100 that satisfies the given duty cycle constraint. The two points on the figures where the n and R values are explicitly shown are the extremum points of the graph. The R values compares the rate gain over the base MPPM system for the largest $n < 100$.

As we can see from Figures 3.3 and 3.4, for 30% and 40% duty cycle, the rate gain is quite significant. For lower duty cycles, as shown in Figures 3.1 and 3.2, we do not get a large rate gain. Lastly, we note that we have limited our analysis to duty cycle up to 40%. This is because duty cycles higher than 40% are of no practical interest.

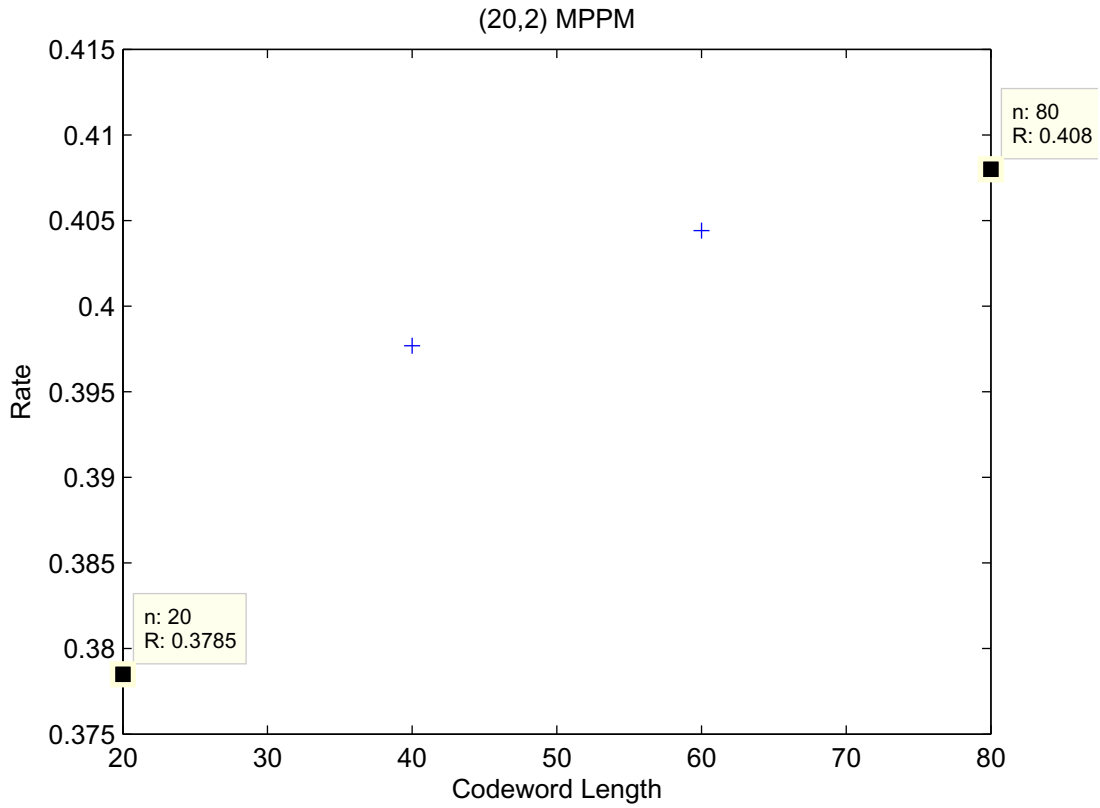

Figure 3.1: Rate improvement starting from a (20,2)-MPPM base code, 20% duty cycle

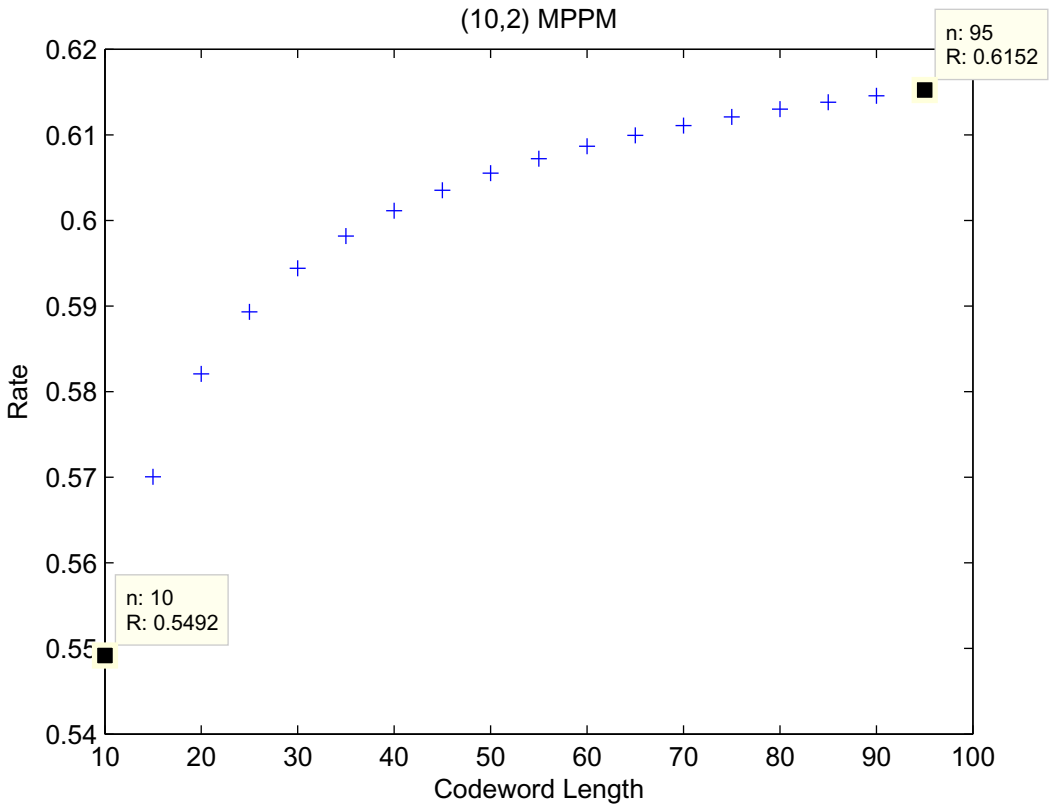

Figure 3.2: Rate improvement starting from a (10,2)-MPPM base code, 20% duty cycle

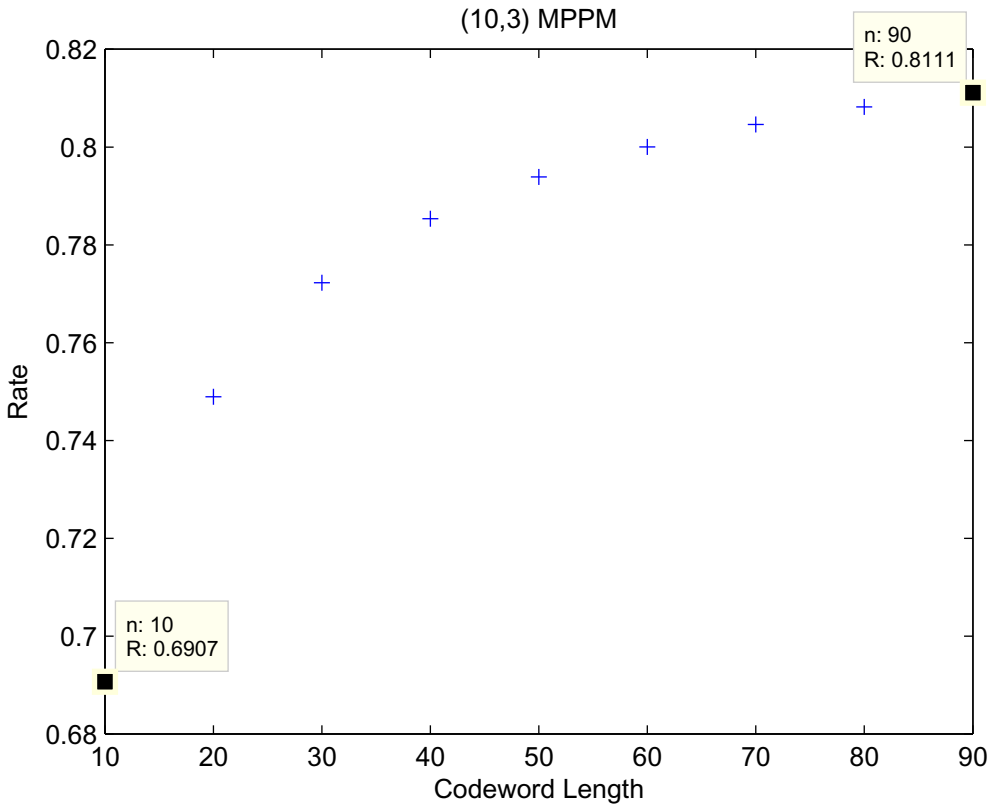

Figure 3.3: Rate improvement starting from a (10,3)-MPPM base code, 30% duty cycle

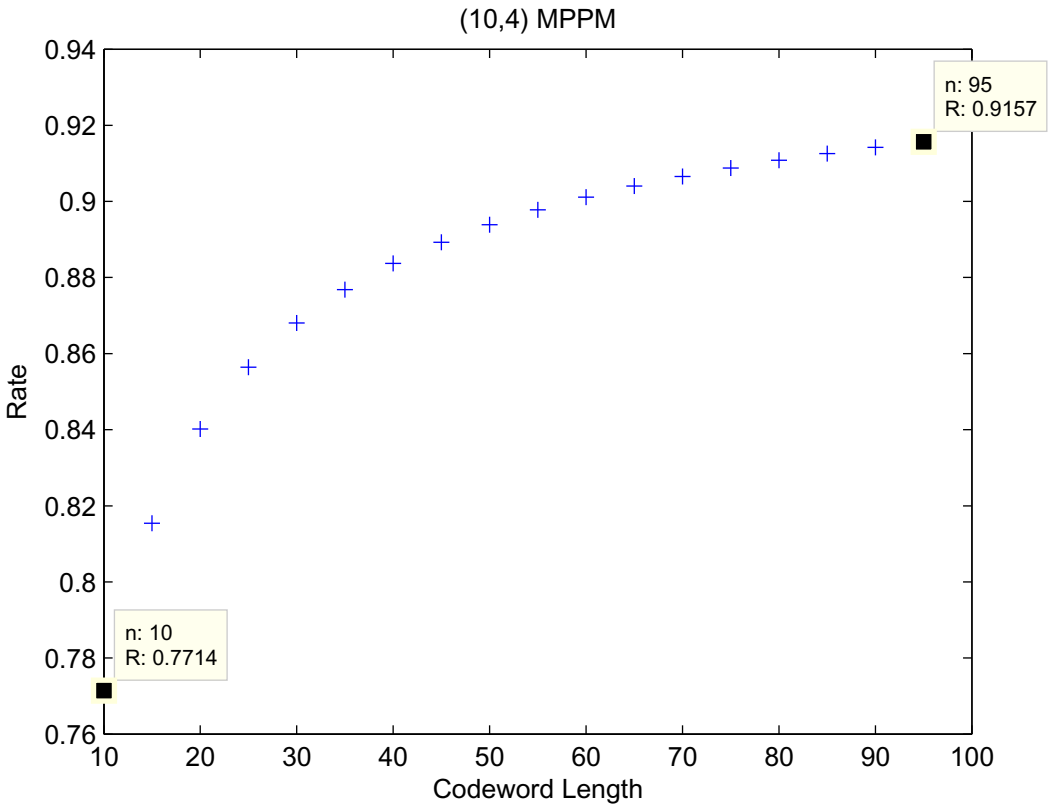

Figure 3.4: Rate improvement starting from a (10,4)-MPPM base code, 40% duty cycle.

3.2 Achievable Regions with Fixed Duty Cycle

In the above section, we saw that by allowing n and k to increase, we can get a significant rate gain. In this section, we shall keep the duty cycle fixed, and pick a large n and k that satisfy the given duty cycle. Now we allow the zero runlength to vary. Each $\mathcal{Z}$ value determines an MPPM system. Thus, for various $\mathcal{Z}$ values, we can compare the rate gain of $S(n, k, \mathcal{Z})$ over the corresponding MPPM system determined by $\mathcal{Z}$.

In the previous section, we considered $M(n, k)$ and $S(n, k, \mathcal{Z})$. In both codebooks, the zero runlength is indeed $\mathcal{Z}$. In practice, however, we are not actually dealing with $M(n, k)$, but $\bar{M}(n, k)$. With an appropriate choice of codewords in $\bar{M}(n, k)$, we can actually reduce the zero runlength. We shall first illustrate this with an example.

Consider $M(5, 2)$, which has $|M(5, 2)| = 10$ elements as listed in Table 3.2. If we are concatenating $M(5, 2)$ codewords, we will have $\mathcal{Z} = 6$. But in practice, we only require $\bar{M}(5, 2)$ with $|\bar{M}(5, 2)| = 8$. Thus, we are free to eliminate 2 codewords from $M(5, 2)$. Suppose that we choose to eliminate codewords 11000 and 00011, we see that the maximum zero runlength upon concatenation is actually only 4. This is because 11000 and 00011 are the only sequences that either *begin* or *end* with three 0s. Without them, all leftover codewords have a maximum of two 0s at either the beginning or the end. Thus, upon concatenation, these codewords can only create maximum zero runlength of 4.

To make this notion precise, we shall make the following definitions.

Definition 3.2.1 *The front zero runlength of a finite binary sequence is the number of consecutive zeros occurring at the beginning of the sequence; and the end zero runlength of a finite binary sequence is the number of consecutive zeros occurring at the end of the sequence.*

Definition 3.2.2 *Let X be a codebook containing finite binary sequences (e.g. $M(n, k)$). The maximum front zero runlength $U_f(X)$ of X is the largest front zero runlength over*

Table 3.2: List of Codewords of (5,2) MPPM

Codewords of $M(5,2)$
11000
10100
10010
10001
01100
01010
01001
00110
00101
00011

all codewords in X; and the maximum end zero runlength $U_e(X)$ of X is the largest end zero runlength over all codewords in X.

Starting with a $M(n,k)$, we can systematically eliminate codewords with long zero runlength (both front and end), such that the resulting $\bar{M}(n,k)$ has the *smallest* maximum zero runlength (front and end). We shall call this particular choice of codewords in $\bar{M}(n,k)$ the $\bar{M}(n,k)$ that minimizes zero runlength. With this choice, we can define our new zero runlength parameter $\bar{\mathcal{Z}}$ as follows:

Definition 3.2.3 *Let $\bar{M}(n,k)$ be chosen to minimize zero runlength. Let z_i be the zero runlength of a codeword $x_i \in \bar{M}(n,k)$, and let $Z_{\max}$ be defined as:*

$$Z_{\max} = \max_{x_i \in \bar{M}(n,k)} z_i. \tag{3.8}$$

The zero runlength parameter $\bar{\mathcal{Z}}$ is defined by:

$$\bar{\mathcal{Z}} = \max(U_f(\bar{M}(n,k)) + U_e(\bar{M}(n,k)), Z_{\max}). \tag{3.9}$$

Note that the above definition tells us that if $U_f(\bar{M}(n,k)) + U_e(\bar{M}(n,k)) < Z_{\max}$, the zero runlength is not determined by the concatenation of codewords, but it is instead determined by the longest zero runlength of the codewords.

With the above setup, we shall compare the achievable regions for 3 systems as a function of zero runlength constraint for a *fixed* duty cycle. First, we consider our standard MPPM $M(n,k)$, governed by the parameter $\mathcal{Z}$. Since such a system is not practical, we shall give it the name "naive" MPPM. The second system is the above choice of $\bar{M}(n,k)$ that minimizes the zero runlength with its associated parameter $\bar{\mathcal{Z}}$, we shall call this type of system *practical* MPPM. Lastly, we choose a large n (≈ 200) and the corresponding k (given by the fixed duty cycle) and calculate the rate for $S(n,k,\mathcal{Z})$. This is denoted as "Zero Runlength" on the figures. Note that as the zero runlength grows sufficiently large, the asymptotic rate for $S(n,k,\mathcal{Z})$ approaches $H(\mathcal{D})$ where H is the bin ary entropy function and $\mathcal{D}$ is our duty cycle. This shows that $n \approx 200$ is sufficiently large in giving approximately the asymptotic achievable rate for given $\mathcal{Z}$ and $\mathcal{D}$ parameters.

The following (Figures 3.5 to 3.7) are a few of the results with duty cycles of practical interest. The points on each graph where z and R values are explicitly shown are chosen to illustrate the significant difference in achievable rates in each of the systems for comparable zero runlengths. We can easily see that the performance of practical MPPM system achieves a much lower rate than the asymptotic rate.

With this set of results, it is natural to ask if we can achieve better rates using other coding schemes. Towards this end, we shall consider a constrained coding approach as described in the following chapter.

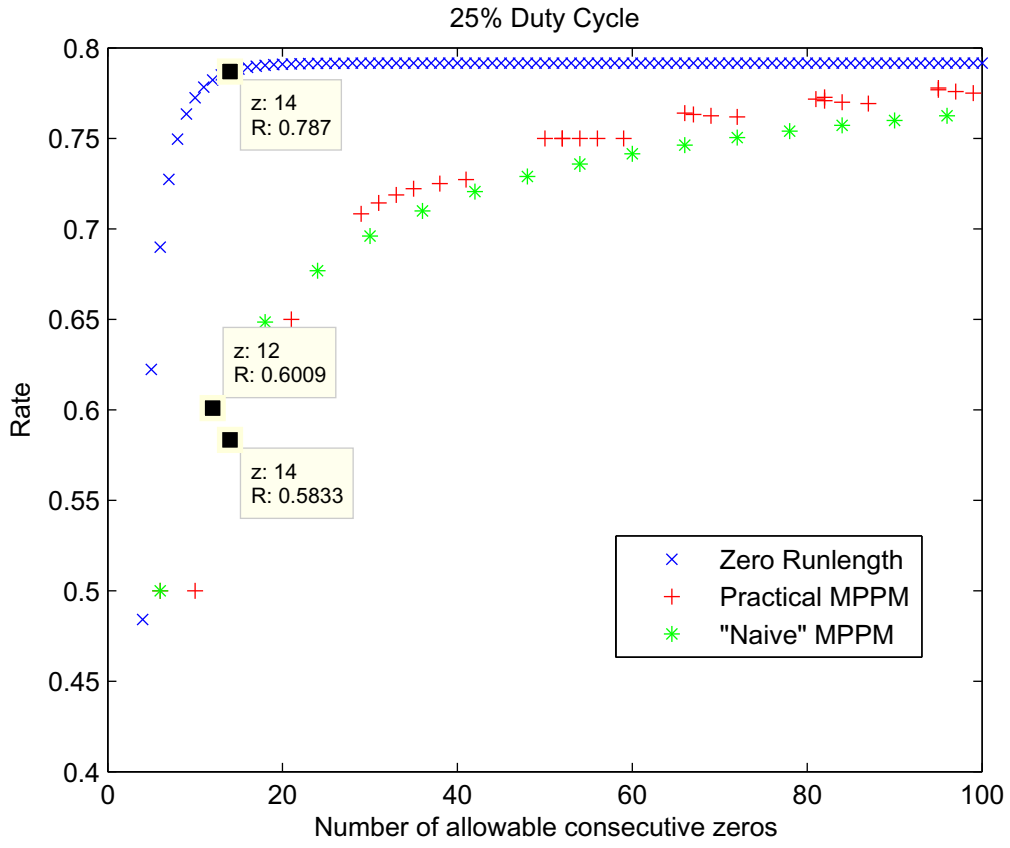

Figure 3.5: Rate as a function of zero runlength constraint for 25% duty cycle

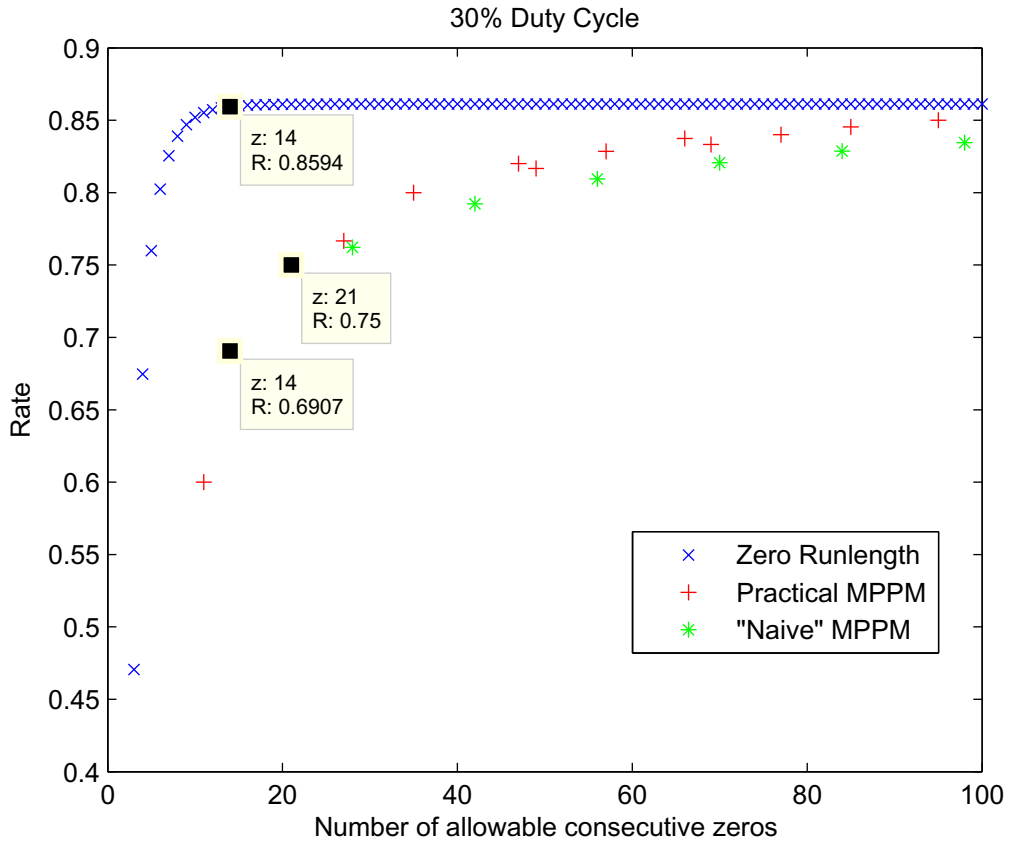

Figure 3.6: Rate as a function of zero runlength constraint for 30% duty cycle

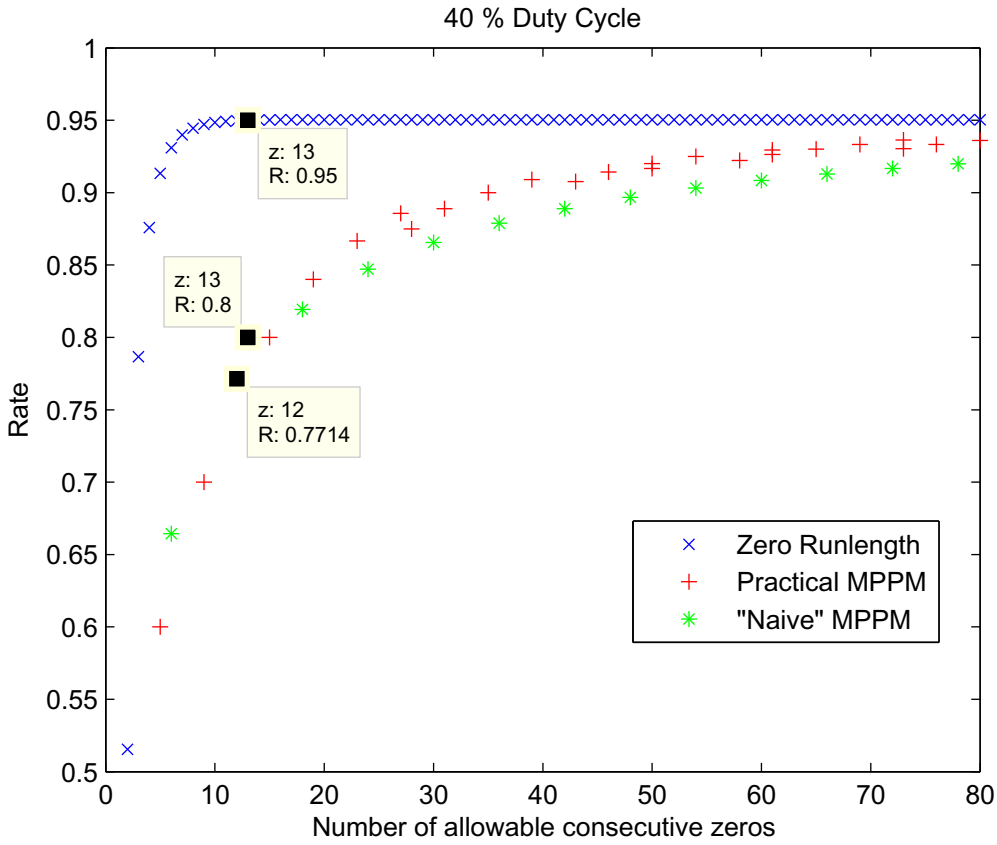

Figure 3.7: Rate as a function of zero runlength constraint for 40% duty cycle

Chapter 4

Constrained Coding Approach

In this chapter, we introduce a constrained coding method to satisfy both the duty cycle and zero runlength constraints.

4.1 Runlength-Limited Sequences

Runlength-limited (RLL) sequences have been widely studied for many years. They are by far the most frequently used coding schemes in digital recording. An in depth analysis on this subject is given in [17]. For our purpose, we only require the basics.

RLL sequences are binary sequences that are completely characterized by two parameters d and k. The d constraint forces two $1s$ to be separated by *at least* d consecutive zeros, while the k constraint allows any run of consecutive $0s$ to have length at most k. Thus, an RLL sequence is normally denoted as a (d, k) sequence. As an example, a $(1, \infty)$ sequence is a sequence without two consecutive ones. Similarly, a $(0, 1)$ sequence is a sequence without two consecutive zeros.

It is easy to see how RLL sequences are of interest as an alternative to MPPM. If we are interested in a particular zero runlength constraint $\mathcal{Z}$ on our system, we can simply use the corresponding set of $(0, \mathcal{Z})$ RLL sequences. The only difficulty will be trying to satisfy the given duty cycle constraint. Towards this end, we propose a greedy *power*

graph approach based on *constraint graphs*.

4.2 Finite State Encoders

RLL sequences and other constrained sequences can be analyzed using a *symbolic dynamics* approach. Symbolic dynamics is a branch of mathematics that studies dynamical systems of discrete spaces. The tools of symbolic dynamics provides us with a way to build *finite state encoders*, which shall prove to be particularly useful for our coding problem. In this section, we shall give a brief introduction to finite state encoders and outline some of their key properties that we will use. A more detailed mathematical treatment is given in Appendix A. For a thorough analysis, readers are encouraged to consult [18].

4.2.1 Constraint Graphs

Many constrained codes, and in particular RLL sequences, can be represented by a *constraint graph*. In loose terms, a *constraint graph* is a graph with an *output* edge labeling that forbids certain output sequences from occurring. Figure 4.1 is an example of a constraint graph that represents the set of $(1, \infty)$ RLL sequences. The circles represent

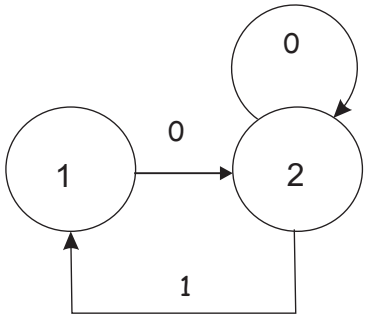

Figure 4.1: Constraint graph representation of $(1, \infty)$ runlength limited sequence

vertices with a set of vertex labels. We can think of the vertices as *states* of a finite state machine. The labeling of the edges gives us a set of *outputs* of our finite state machine.

In Figure 4.1, we can see that the only edge that outputs a 1 is the edge transition from state (vertex) 2 to state 1. When we are in state 1, we can only go to state 2 with an edge label of 0. Thus, it is easy to see that an output of two consecutive 1s can never occur. The self-loop on state 2 allows us to output arbitrarily long sequences of 0s. Hence, we have verified that the above constraint graph indeed represents the set of $(1, \infty)$ RLL sequences.

4.2.2 Encoders from Constraint Graphs

In the last example, we called the labeling of our edges *outputs* because they represent the set of all possible sequences that occur as a result of traversing the edges of our constraint graph. We can view these outputs as the output of an *encoder*. As an example, consider state 2 in Figure 4.1. There are two outgoing edges from state 2, thus each time at state 2, we are given a choice of which edge to traverse. A set of choices can be defined by a set of *inputs* for our constraint graph. At each state, an *encoding* is an one-to-one mapping from inputs to outputs.

To illustrate this notion, consider Figure 4.2 as an example. At each state, we have two possible outgoing edges. The choice of an edge is determined by a binary *input* of either a 0 or 1. Each choice of an input uniquely determines which edge is taken, and hence also determines the output. For example, at state 1, an input of 0 determines an output of 00. Thus, at state 0, the input 0 is *encoded* into the output 00. A constraint graph where every edge is specified with an input and an output is called an *encoder graph*. The resulting code is referred to as a *finite state code*.

Note that in Figure 4.2 we have an encoding from 1-bit input to a 2-bit output. This gives us a code rate of 1/2. In general, we are interested in rational code rates of $r = p/q$ with $p, q \in \mathbb{Z}_+$. Of course, there are limitations on what rates we can achieve for a given constraint graph. For example, no matter what the underlying graph is, we can never achieve a rate greater than 1. To fully capture the achievable rates of a constraint graph,

we shall develop the following notion of *entropy*.

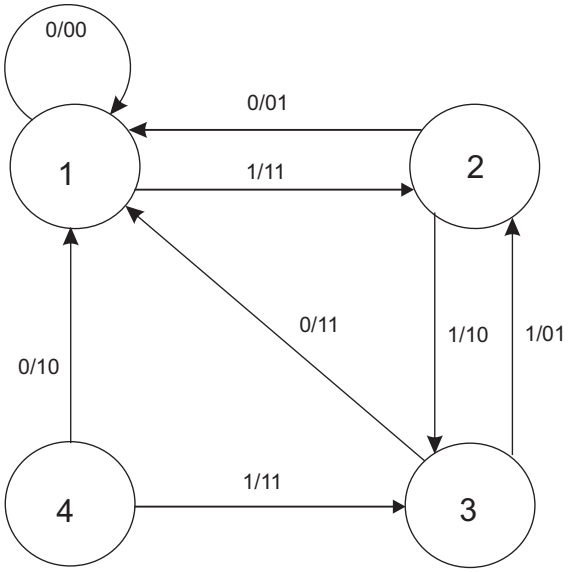

Figure 4.2: A rate 1/2 finite state code

4.2.3 Entropy of Constraint Graphs

For a given constraint graph G, the maximal code rate it can achieve is governed by its *entropy*. Entropy can be easily computed based on Perron-Frobenius theory (implicitly we are assuming that our underlying graph is *irreducible*). Namely, given the *adjacency* matrix A of G, the entropy is given by:

$$h(G) = \log \lambda_A \tag{4.1}$$

where λ_A is the largest eigenvalue of A. With this and as a consequence of the *Finite-State Coding Theorem* from [18], we can construct a rate p/q code from G as long as $h(G) \geq p/q$.

4.2.4 p:q Encoders and Power Graphs

A natural way to construct rate p/q finite state code is to consider a mapping from p input bits to q output bits. This mapping can be easily done if each of the vertices in our

constraint graph has at least 2^p outgoing edges. However, using the $(1, \infty)$ RLL sequence in Figure 4.1 as example, we see that this condition cannot even be satisfied for $p = 1$.

One way to remedy this problem is by taking *power graphs*. Again using the $(1, \infty)$ example, we see that it has an adjacency matrix

$$A((1,\infty)) = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}.$$

Taking the second power graph, we get (see Figure 4.3):

$$A((1,\infty))^2 = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}.$$

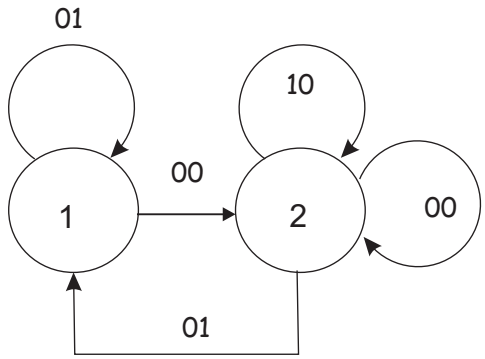

Figure 4.3: Second power graph of $(1, \infty)$ runlength limited sequence

We see that the second power graph has at least 2^1 outgoing edges at each vertex. This will allow us to build $1/q$ codes. Note that any power graph satisfy the same constraints (on the output sequence) as the original constraint graph. Thus taking power graphs is an useful tool for constructing finite state codes. More generally, we can take power graphs and use an algorithm called the *State Splitting Algorithm* (outlined in [18]) to construct finite state codes. For our purposes, we shall see that taking power graphs alone is sufficient to achieve a high rate.

4.3 $(0, k)$ Sequences and Power Graphs

With a general notion of capacity for constraint graphs, we shall now shift our attention back to RLL sequences. Specifically, we will now focus on $(0, k)$ sequences. As previously mentioned, $(0, k)$ sequences are particularly interesting to us as the k constraint exactly gives us a zero runlength of k.

4.3.1 $(0, k)$ Capacity

It is easy to analyze the capacity of general (d, k) sequences. One way is through their constraint graph representations. As shown in [17], the characteristic equations (of the adjacency matrices) of (d, k) sequences have the following general form:

$$z^{k+2} - z^{k+1} - z^{k-d+1} + 1 = 0. \tag{4.2}$$

Putting $d = 0$, we get

$$z^{k+2} - 2z^{k+1} + 1 = 0. \tag{4.3}$$

With this, we shall now list the capacity of $(0, k)$ sequences for various k parameters in Table 4.1.

4.3.2 Power Graphs of $(0, k)$ Sequences

As 4.1 shows, the $(0, k)$ sequences have very high capacity for $k > 2$. This allows us a great deal of flexibility in designing p/q finite state codes. As mentioned earlier, a standard way of designing p/q finite state codes is through taking power graphs of the underlying constraint graph. In this section, we shall prove a theorem that demonstrates a useful property of the power graphs of $(0, k)$ sequences. In the following section, we shall exploit this property in our code design. Throughout this section, we shall assume the following set of definitions:

Table 4.1: Capacity of $(0, k)$ Sequences for Various k Values

k	Capacity
1	0.6942
2	0.8791
3	0.9468
4	0.9752
5	0.9881
6	0.9942
7	0.9971
8	0.9986
9	0.9993
10	0.9996
∞	1.000

Definition 4.3.1 *Let G_k be the constraint graph representation of the set of $(0, k)$ sequences. Let $A(G_k)$ be its adjacency matrix and $A(G_k)^n$ be the adjacency matrix of the nth power graph. Let R_i^n be the ith row in $A(G_k)^n$ with $R_i^n(j)$ denoting its jth element and $|R_i^n|$ the associated row sum (sum over all $R_i^n(j)$). Similarly, let C_i^n be the ith column in $A(G_k)^n$ with $C_i^n(j)$ denoting its jth element and $|C_i^n|$ the associated column sum (sum over all $C_i^n(j)$).*

We shall prove the following main theorem of this section.

Theorem 4.3.2 *Given a G_k, for $1 \leq j \leq k+1$, $1 \leq i \leq k+1$, we have $|R_i^j| \geq 2^{j-1}$, with equality exactly when $i = k+1$ (i.e, on the last row).*

Before proving the theorem, we first observe that $A(G_k)$ is a $(k+1) \times (k+1)$ matrix with entries A_{ij} having the following form:

$$A_{ij} = \begin{cases} 1 & \text{if } j = 1, \text{ or } j = i+1 \text{ and } i \leq k \\ 0 & \text{otherwise} \end{cases} \tag{4.4}$$

From this, or from the constraint graph, it is easy to first prove the following lemma.

Lemma 4.3.3 *For all $n > 0$, we have $|R_i^n| > |R_{k+1}^n|$ for $1 \leq i \leq k$.*

Proof: Looking at the adjacency matrix A, we have $|R_i| > |R_{k+1}|$ for $1 \leq i \leq k$. Furthermore, R_{k+1} has only one non-zero element, namely the first element $R_{k+1}(1) = 1$. Now for $1 \leq i \leq k$, $R_i(1) = 1$ and $R_i(j) = 1$ for some $1 < j \leq k+1$. This means that for all $1 \leq i \leq k$, $R_i(j) \geq R_{k+1}(j)$ for all $1 < j \leq k+1$; and for *each* i, $1 \leq i \leq k$ there exist exactly one j, $1 < j \leq k+1$ such that $R_i(j) > R_{k+1}(j)$. In words, all entries of any row R_i is bigger or equal to entries of R_{k+1}, with exactly one element strictly greater. Now, observe that A is positive definite with a column (C_1) with strictly positive entries. This means that for $1 \leq i \leq k$, $R_i C_1 > R_{k+1} C_1$ and $R_i C_j \geq R_{k+1} C_j$ for $1 < j \leq k+1$. This shows that for $1 \leq i \leq k$, $|R_i^2| > |R_{k+1}^2|$. The rest follows by induction. □

Next, we observe that $R^n_{k+1} = R^{n-1}_1$. This is by the fact that applying R_{k+1} to A^{n-1} simply copies the values of the first row of A^{n-1} (recall that R_{k+1} has $R_{k+1}(1) = 1$ and the rest of the entries being 0). Thus, we should focus our attention to the evolution of R^n_1 (as we increase n). Note that since $R_1(1) = 1$, $R_1(2) = 1$ and $R_1(j) = 0$ for $j \geq 2$, we have $R^n_1 = R^{n-1}_1 + R^{n-1}_2$. I.e., applying R_1 to A^{n-1} gives us the sum of the values of the first two rows of A^{n-1}. Now, the key observation here is that the first two rows of A^n follow a very neat pattern for $n \leq k + 1$. As an example, we shall illustrate this fact by looking at the adjacency matrix of G_4. We have the following:

$$A(G_4) = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$A(G_4)^2 = \begin{bmatrix} 2 & 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 & 1 \\ 2 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$A(G_4)^3 = \begin{bmatrix} 4 & 2 & 1 & 1 & 0 \\ 4 & 2 & 1 & 0 & 1 \\ 4 & 2 & 1 & 0 & 0 \\ 3 & 2 & 1 & 0 & 0 \\ 2 & 1 & 1 & 0 & 0 \end{bmatrix}$$

$$A(G_4)^4 = \begin{bmatrix} 8 & 4 & 2 & 1 & 1 \\ 8 & 4 & 2 & 1 & 0 \\ 7 & 4 & 2 & 1 & 0 \\ 6 & 3 & 2 & 1 & 0 \\ 4 & 2 & 1 & 1 & 0 \end{bmatrix}$$

$$A(G_5)^5 = \begin{bmatrix} 16 & 8 & 4 & 2 & 1 \\ 15 & 8 & 4 & 2 & 1 \\ 14 & 7 & 4 & 2 & 1 \\ 12 & 6 & 3 & 2 & 1 \\ 8 & 4 & 2 & 1 & 1 \end{bmatrix}$$

Focusing on the first two rows, we see that for $n < k+1$, $R_1^n(j) = R_2^n(j)$ for $1 \leq j \leq n$. Furthermore, for $n < k$, we have $R_1^n(n+1) = 1$, $R_2^n(n+1) = 0$, $R_1^n(n+2) = 0$ and $R_2^n(n+1) = 1$. In fact, this observation is true for *any* G_k. We shall prove this as the following lemma.

Lemma 4.3.4 *For any $A(G_k)$, for $n < k+1$, $R_1^n(j) = R_2^n(j)$ for $1 \leq j \leq n$. For $n < k$, we have $R_1^n(n+1) = 1$, $R_2^n(n+1) = 0$, $R_1^n(n+2) = 0$ and $R_2^n(n+1) = 1$.*

Proof: Consider the constraint graph G_k. Elements of $A(G_k)^n$ correspond to paths of length n in G_k. Now for $n < k$, consider any state j with $j \leq n$. Any path from state 1 to state j of length n *must* pass through state 1 at least once (excluding the initial state from state 1). This is because $j \leq n$ and the only loop in the graph is through state 1. Note that, as we can see from the constraint graph, it is not possible to have path of length n between two states that differ in state index less than n *without* a loop. Similarly, any path from state 2 to state j has the same property.

With this observation, we see that we can form a bijective correspondence between state transitions from state 1 to j and state 2 to j by simply shifting the index of all state transitions by 1 except state transitions that ends with state 1 (and of course the last

state transition to state j). For example, consider $A(G_4)^3$ from above, the two paths of length 3 from state 1 to state 2 are $1 \to 2$, $2 \to 1$, $1 \to 2$ and $1 \to 1$, $1 \to 1$, $1 \to 2$. The two corresponding paths from state 2 to 2 are $2 \to 3$, $3 \to 1$, $1 \to 2$ and $2 \to 1$, $1 \to 1$, $1 \to 2$. By our restriction on the state index of j, it is easy to see that this mapping is indeed a bijection (our state index can never be greater than k).

Lastly, it is easy to see that there is exactly one path of length n from state 1 to state $n+1$ and one path of length n from state 2 to state $n+2$. With this, the proof is completed. $\square$.

With the above lemmas, we are now in the position to prove Theorem 4.3.2.

Proof of Theorem 4.3.2: We shall proceed by induction. Clearly, $R_1(1) = 1$ and $R_1(2) = 1$ with all other entries zero. Now suppose for some $n \leq k$ we have $R_1^n(j) = 2^{n-j}$ for $1 \leq j \leq n$. By Lemma 4.3.4 we know that $R_2^n(j) = 2^{n-j}$ for $1 \leq j \leq n$ as well. Furthermore, we know that $R_1^n(n+1) = 1$ and $R_2^n(n+2) = 1$, with all other entries zero. Since R_1^{n+1} is the sum of these two rows, we have $R_1^{n+1}(j) = 2^{n-j+1}$ for $1 \leq j \leq n+1$ and $R_1^{n+1}(n+2) = 1$. This gives us $|R_1^{n+1}| = 2^{n+1}$. Since we have $R_{k+1}^{n+2} = R_1^{n+1}$, we have $|R_{k+1}^{n+2}| = 2^{n+1}$. By our restriction on n, this works exactly up to $n+2 = k+1$ (recall $n < k$). The proof is complete with Lemma 4.3.4. $\square$

4.4 Minimal Duty Cycle Design

At this point, as a guiding design example for the rest of our discussion, we will start with $(0, 5)$ RLL sequence and construct a rate $4/5$ code from it. First observe that the $(0, 5)$ RLL sequence can be represented by the following constraint graph $\mathcal{G}$ (Fig. 4.4).

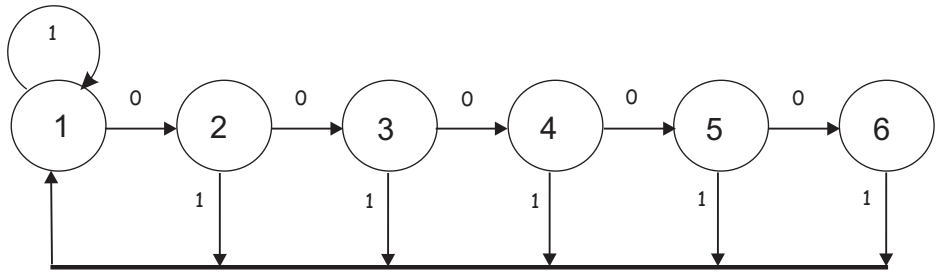

Figure 4.4: Constrained graph representation of (0,5) runlength limited sequence

Associated with this constrained graph is its adjacency matrix given by:

$$A(\mathcal{G}) = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

4.4.1 Design Algorithm via Power Graph

Now, we shall outline our design algorithm by taking power graphs.

1. Given a constraint graph G_k of set of $(0, k)$ sequences , find its adjacency matrix $A(G_k)$

2. Compute $h(A(G_k)$.

3. Choose integers p and q such that $p < q$, $q \leq k + 1$ and $h(A(G_k)) \geq p/q$

4. Construct G_k^q, observing that $h(A(G_k^q)) \geq \log 2^p$.

5. Observe that by Theorem 4.3.2 each vertex of G_k^q has at least 2^p outgoing edges.

6. Prune away extra edges from G_k^q if necessary to obtain $\hat{G_k}^q$, which has exactly 2^p outgoing edges at each vertex.

Following the above algorithm, in order to construct a 4/5 code from a constraint graph, we would need to construct $\mathcal{G}^5$. Namely, this is the graph of all paths of length 5 in $\mathcal{G}$ and thus has the following adjacency matrix:

$$A(\mathcal{G}^5) = A(\mathcal{G})^5 = \begin{bmatrix} 16 & 8 & 4 & 2 & 1 & 1 \\ 16 & 8 & 4 & 2 & 1 & 0 \\ 15 & 8 & 4 & 2 & 1 & 0 \\ 14 & 7 & 4 & 2 & 1 & 0 \\ 12 & 6 & 3 & 2 & 1 & 0 \\ 8 & 4 & 2 & 1 & 1 & 0 \end{bmatrix}$$

Looking at the sum of each row, we see that every row sums up to at least $2^4 = 16$. Thus, each vertex in $\mathcal{G}^5$ has at least 16 outgoing edges. Also observe that by our construction, we have $h(A(\mathcal{G}^5)) = 4.9405 \geq \log 2^p = 4$.

Now each of the edges in $\mathcal{G}^5$ correspond to a length 5 path in $\mathcal{G}$. As we labeled edges in $\mathcal{G}$ by 1-bit output labels, we can label edges in $\mathcal{G}^5$ by its corresponding 5-bit output label. Note that we only require 16 outgoing edges at each vertex. So if a vertex has more than 16 outgoing edges, we would like to keep the 16 edges that minimizes the overall duty cycle of code. To achieve this, we shall use the following greedy approach:

1. Take the 16 edges with the least number of 1s (least weight) in their output labels.

2. If there is a tie between two edges, we always pick the edge that goes to the lower vertex (state) index.

As an example, at the outgoing edge of vertex 3, if we have a choice between picking 10011 which goes to vertex 1 and 10110 which goes to vertex 2 (both have weight 3), we will pick 10011 because vertex 1 has a lower vertex index. The rationale behind this is that in general the higher the vertex index, the higher the overall weight of the output edges (by that we mean the sum of all weights) at the vertex. Thus, we would like to go back to the vertices with lower indices as much as possible. Clearly, this greedy approach

tries to minimize the duty cycle of the code. However, we currently have no proof that the resulting code does indeed have a minimal duty cycle.

Following the above greedy algorithm, we can list outputs for each state in Table 4.2 and 4.3. The tables are listed in term of vertices similar to the structure of an adjacency matrix. The initial vertex is listed in the first column, and the next 6 columns list the output edge labeling from the initial vertex to given state in the particular column. For example, from state (vertex) 1, there are 2 edges going into state 4, and they are 01000 and 11000. Listing edges in this manner, we can easily see that most of the edges go into state 1.

Given the list in Table 4.2 and 4.3, we can construct a finite-state code by choosing a set of input labels. Finally, we give the constraint graph representation of this new finite-state code C. For simplicity, we shall represent it by its adjacency matrix:

$$A(C) = \begin{bmatrix} 5 & 4 & 3 & 2 & 1 & 1 \\ 6 & 4 & 3 & 2 & 1 & 0 \\ 6 & 4 & 3 & 2 & 1 & 0 \\ 7 & 3 & 3 & 2 & 1 & 0 \\ 9 & 2 & 2 & 2 & 1 & 0 \\ 8 & 4 & 2 & 1 & 1 & 0 \end{bmatrix}$$

4.4.2 Duty Cycle Analysis

As our key goal is to analyze the asymptotic duty cycle of this code, we need knowledge of the weight (number of ones) associated with each state. Such a list is given in Table 4.4.

What we are really interested in is actually the *average weight* of the code. To compute this, we note that the adjacency matrix tells us the probability of each state transition. Namely, the probability of a state transition from state i to state j is given by $\frac{1}{16}A(C)_{i,j}$. Thus, we can model the state transitions as a Markov chain. From this

Table 4.2: State Table Part 1

States	1	2	3	4	5	6
From 1	00001	00010	00100	01000	10000	00000
	00011	00110	01100	11000		
	00101	01010	10100			
	01001	10010				
	10001					
From 2	00001	00010	00100	01000	10000	
	00011	00110	01100	11000		
	00101	01010	10100			
	01001	10010				
	10001					
	00111					
From 3	00011	00010	00100	01000	10000	
	00101	00110	01100	11000		
	01001	01010	10100			
	10001	10010				
	00111					
	01011					

Table 4.3: State Table Part 2

States	1	2	3	4	5	6
From 4	00101	00110	00100	01000	10000	
	01001	01010	01100	11000		
	10001	10010	10100			
	00111					
	01011					
	10011					
	10101					
From 5	01001	01010	01100	01000	10000	
	10001	10010	10100	11000		
	01011					
	10011					
	10101					
	11001					
	01111					
	10111					
	11011					
From 6	10001	10010	10100	11000	10000	
	10011	10110	11100			
	10101	11010				
	11001	11110				
	10111					
	11011					
	11101					
	11111					

Table 4.4: Table of output weights

States	Output Weight
1	25
2	28
3	30
4	33
5	40
6	48

the steady state distribution P_∞ is given by the equation:

$$P_\infty = \frac{1}{16} A(C) P_\infty \tag{4.5}$$

In our particular case, we have $P_\infty = [0.3740, 0.2345, 0.1821, 0.1235, 00625, 0.0234]$. Given our weight distribution $W = [25, 28, 30, 33, 40, 48]$, we have $P_\infty W^T \approx 29.0775$. So the average weight is 29.0775, this corresponds to an asymptotic duty cycle of $29.0775/(16 \times 5) \approx 36.347\%$.

Thus, with our new design, we are able to achieve a rate of 4/5 with duty cycle 36.347 and zero runlength constraint of 5. In the following section, we shall see how this code compares to MPPM systems.

4.5 Achievable Rates and MPPM Comparison

Following the code construction we described in the previous section, we constructed 5 different constrained codes with different duty cycles and zero runlengths. As shown in Figures 4.5 to 4.7, they all achieve superior rate against comparable MPPM systems. Note that their duty cycle is *not* exactly same as the fixed duty cycle on the figures. In all cases, our constrained codes actually have a slightly *lower* duty cycle. Taking the

best constrained code in each of the figures, Table 4.5 shows the exact properties of our codes.

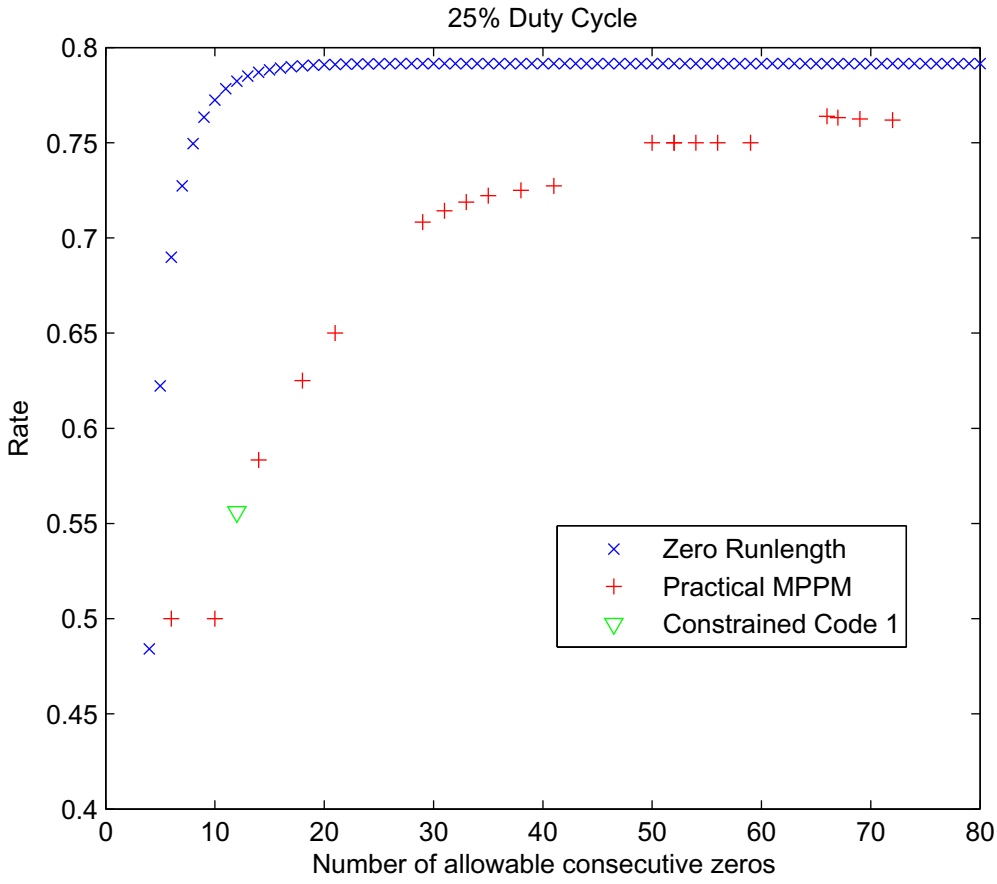

Figure 4.5: Comparison of constrained codes with MPPM systems for 25% duty cycle

From Table 4.5, we can see that our 4/5 code provides the best gain over MPPM. Both $\bar{M}(5,2)$ and $\bar{M}(10,4)$ MPPM systems achieve only a rate of 0.6. $\bar{M}(5,2)$ has a zero runlength of 4 while $\bar{M}(10,4)$ has a zero runlength of 10. (Recall we started with a $(0,5)$ RLL sequence, hence have a zero runlength of 5). To further put our gain in perspective, even a $\bar{M}(40,10)$ system only achieves a rate of 0.725 and it requires a zero runlength of 54 and has $2^{29} = 536870912$ codewords!

In the next chapter, we shall use our rate 4/5 constrained code in serial concatenation with an outer LDPC code. Using iterative decoding, we shall show that our concatenated

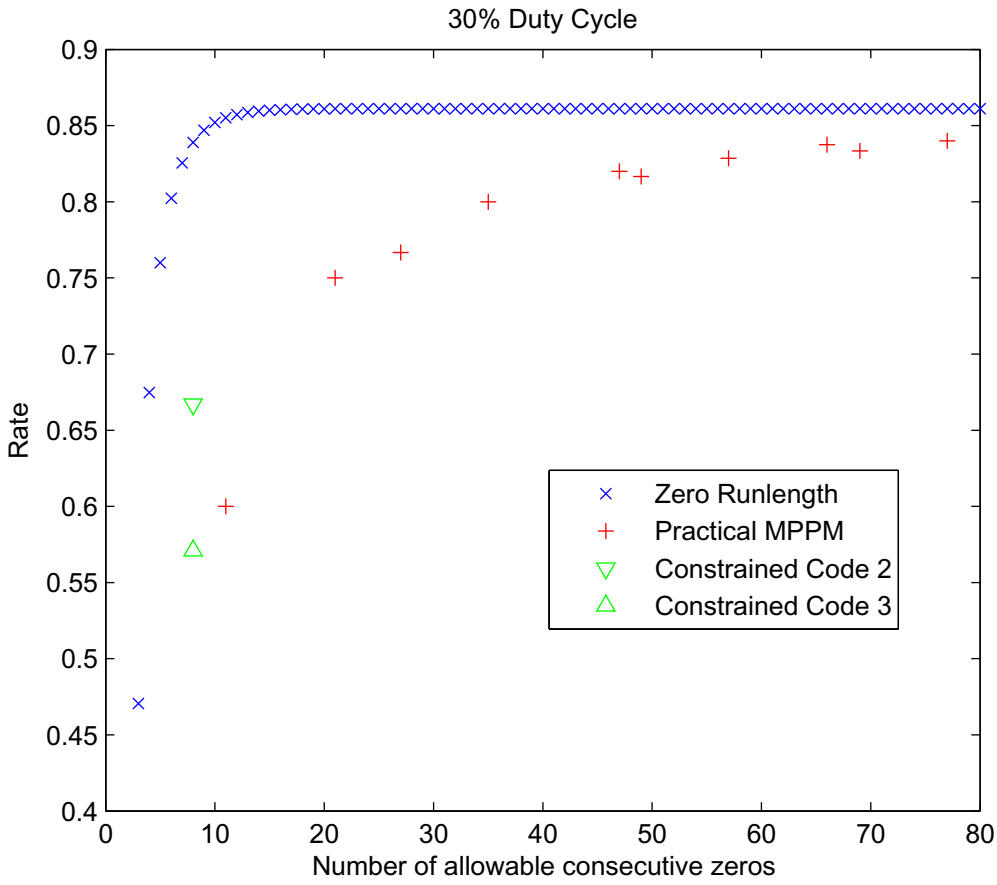

Figure 4.6: Comparison of constrained codes with MPPM systems for 30% duty cycle

system can achieve much better bit-error-rate (BER) for the same SNR as compared to MPPM systems.

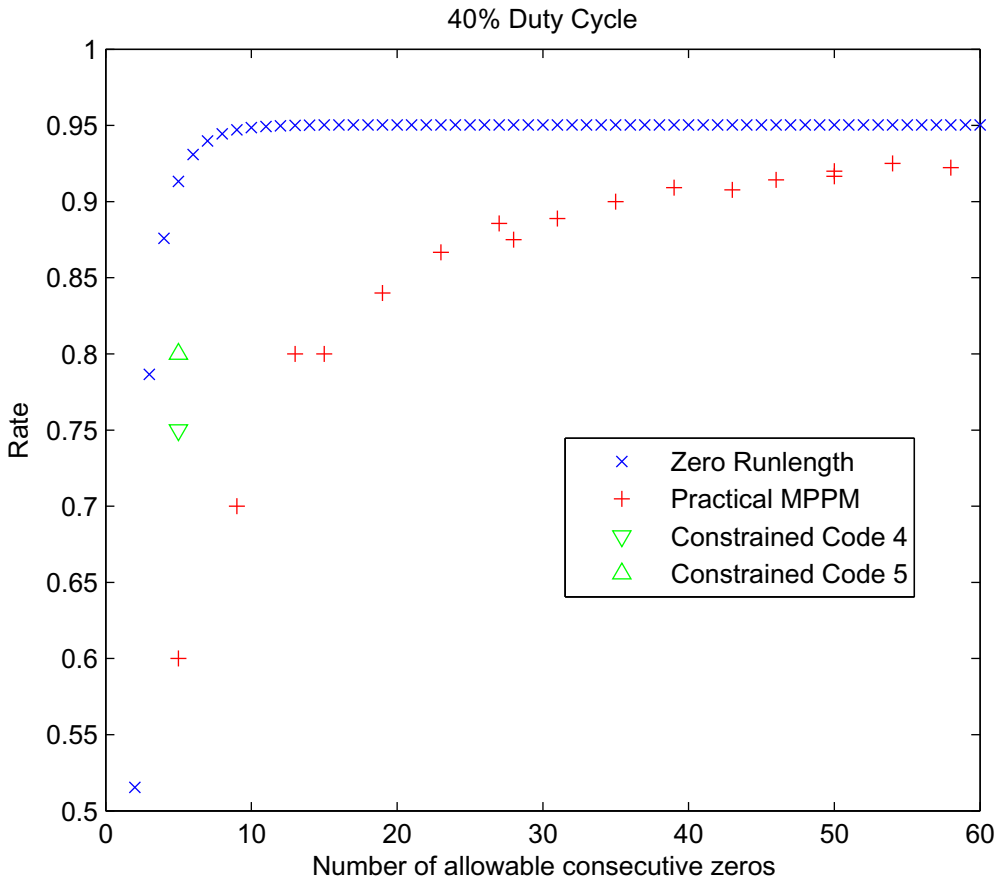

Figure 4.7: Comparison of constrained codes with MPPM systems for 40% duty cycle

Table 4.5: Table of Constrained Code

Code	Asymptotic Duty Cycle	p:q	Rate
Code 1	22.2%	5 : 9	0.556
Code 2	28.3%	2 : 3	0.667
Code 5	36.47%	4 : 5	0.8

Chapter 5

Concatenation with Outer Code

As mentioned in the previous section, one of the main advantages of having a constrained code is to allow for soft decoding. In this chapter, we create a serially concatenated code with our constrained system as an inner code. We shall first describe our iterative decoding system, followed by explaining a methodology of designing outer codes using Extrinsic Information Transfer (EXIT) charts [19, 20]. Lastly, we shall compare the performance of our iterative decoding system against comparable MPPM systems.

5.1 Serially Concatenated System

Since the discovery of *turbo codes* [22], the concept of iterative decoding has been widely used due to its excellent performance. In our work, we propose a serially concatenated system with our constrained code as an inner code and a low density parity check (LDPC) code as the outer code. Figure 5.1 provides an overview of our communication system. Iterative decoding is performed between the inner trellis decoder and the outer LDPC decoder. The choice of LDPC codes and the exact message passing involved in the iterative decoding will be explained in the following sections.

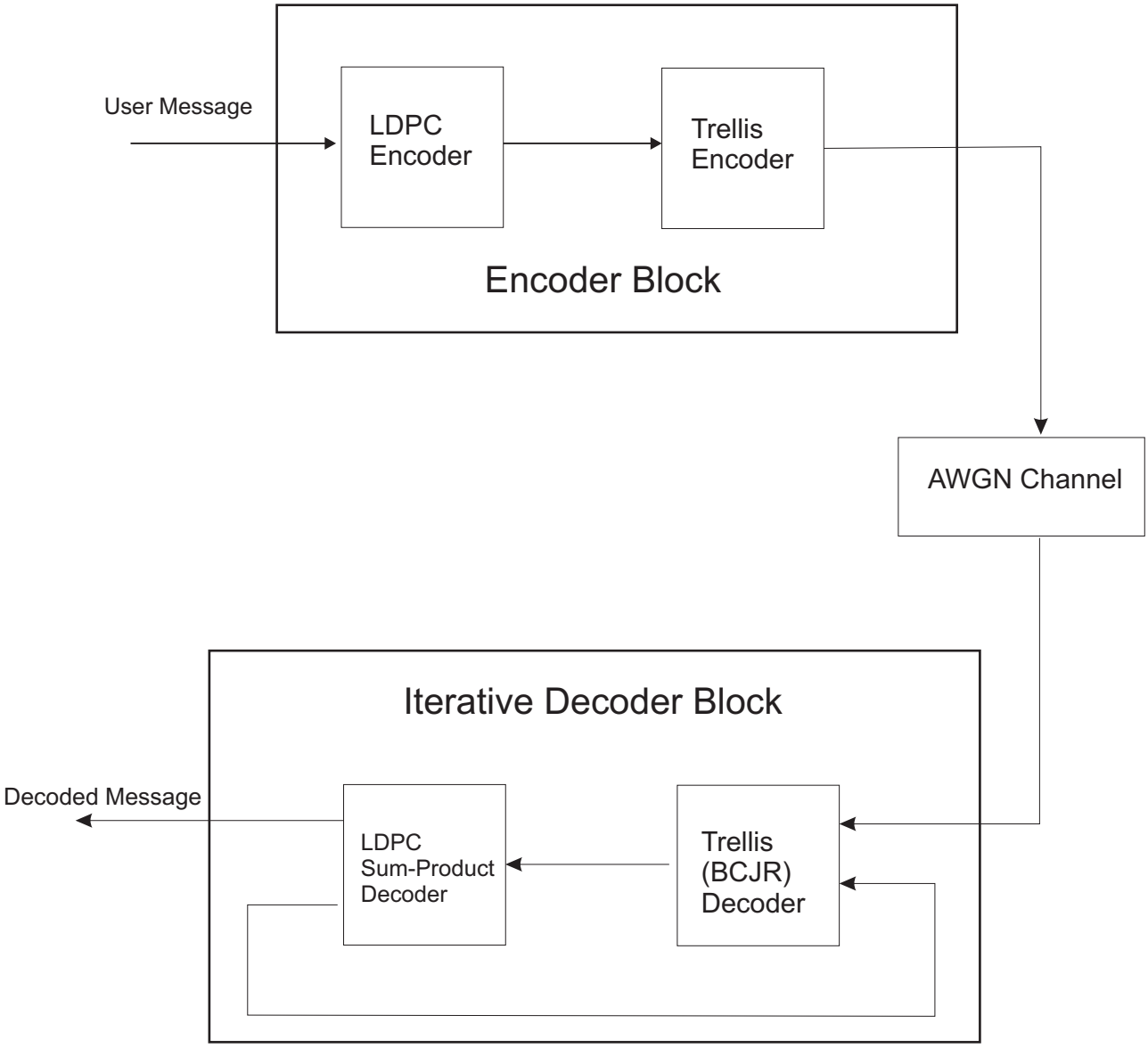

Figure 5.1: Full communication model with constrained (trellis) inner code and LDPC outer code

5.2 LDPC code and EXIT Chart Analysis

In this section, we shall give a brief introduction to LDPC codes and EXIT charts. We shall use EXIT charts to design LDPC outer codes for our serially concatenated system.

5.2.1 LDPC Codes

LDPC codes were first invented by Gallager [26]. As its name implies, an LDPC code has a *sparse* parity-check matrix. More precisely, it has a parity-check matrix of size $r \times n$ and code rate of $R = 1 - \frac{r}{n}$. As $n \to \infty$, the fraction of zero elements goes to one.

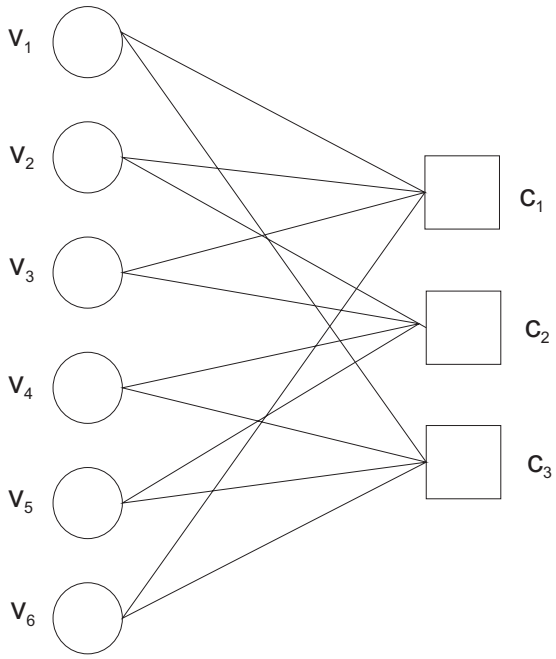

Figure 5.2: Tanner Graph Representation of (6,3) Regular LDPC Code

It is convenient to represent the parity-check matrix H as a bipartite graph with n left *variable* nodes and r right *check* nodes, and an edge joining a left node and a right node if H has a non-zero element in the corresponding position. This representation is called a Tanner graph [27], which is a special case of a *factor graph* [28].

Associated with an LDPC code are two sets of parameters called *variable degrees* and *check degrees*. For a given variable node, its variable degree is the number of edges connected to it. Similarly, for a given check node, its check degree is the number of edges it is connected to. If every variable node has variable degree j and every check node has check degree k, the LDPC code is called a (j, k) regular LDPC code.

With some analysis [29, 31], one can show that *increasing* the variable degree j im-

proves the performance of the LDPC code, and *decreasing* the check degree also improves the performance. However, it is clear that for a regular LDPC code, it is impossible to satisfy both conditions simultaneously. This calls for the implementation of *irregular LDPC codes*.

Irregular LDPC codes were first studied by Luby et al. [30]. As one might guess, instead of having one fixed variable degree and check degree, irregular LDPC codes have a *distribution* of variable degrees and check degrees. The only requirement is that one maintains an average variable degree $\bar{d}_v$ and an average check degree $\bar{d}_c$ such that $n \cdot \bar{d}_v = r \cdot \bar{d}_c$. We shall see that given an EXIT chart of the inner code, we can use curve fitting to design the degree distribution of the outer LDPC code.

5.2.2 EXIT Charts

Extrinsic information transfer (EXIT) charts were first introduced by ten Brink [19]. They provide a very intuitive method for designing iterative decoding systems. Conceptually, an EXIT chart keeps track of the flow of information in an iterative decoding system, much like *density evolution* as proposed by Richardson and Urbanke [23, 24]. The advantage of EXIT charts is that, while they are not exact, they allow information to be characterized in terms of *mutual information*, which can be efficiently computed.

Following ten Brink's treatment, we focus our attention on a section of an iterative decoding system. Figure 5.3 shows a decoder within an iterative decoding system. Here Z represents the channel observation, D is the decoded output, A is the *a priori* information from a previous decoder in the system and E is the *extrinsic information* to be passed to the next decoder in the system. From the perspective of our particular serially concatenated system in Figure 5.1, focusing on the trellis decoder, A corresponds to the message passed back to the trellis *from* the outer LDPC decoder while E corresponds be the message that the trellis decoder pass *to* the outer LDPC decoder.

The key observation by ten Brink is that the *a priori* input A can be modeled by

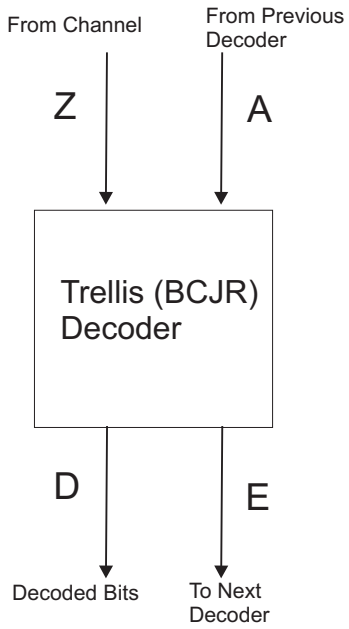

Figure 5.3: A portion of an iterative decoding system that takes in a priori information and passes out extrinsic (a posterior) information.

applying an independent Gaussian random variable n_A with zero mean and variance σ_A^2 in conjunction with the *known* transmitted bits x, given by:

$$A = \mu_A x + n_A \tag{5.1}$$

With this, we can compute the mutual information $I_A = I(X; A)$ given by:

$$I_A = \frac{1}{2} \sum_{x=-1,1} \int_{-\infty}^{+\infty} \log \frac{2 \cdot p_A(\xi | X = x)}{p_A(\xi | X = -1) + p_A(\xi | X = +1)} d\xi \tag{5.2}$$

Now since $p_A(\xi | X = x)$ is a function of σ_A, we can characterize I_A as a function of σ_A. Namely, define:

$$J(\sigma) = I_A(\sigma_A = \sigma) \tag{5.3}$$

$$\sigma_A = J^{-1}(I_A) \tag{5.4}$$

Note that this J function cannot be expressed in closed form.

Furthermore, the mutual information of the extrinsic output $I_E = I(X; E)$ is given

by:

$$I_E = \frac{1}{2} \sum_{x=-1,1} \int_{-\infty}^{+\infty} \log \frac{2 \cdot p_E(\xi|X=x)}{p_E(\xi|X=-1) + p_E(\xi|X=+1)} d\xi \tag{5.5}$$

With the above analysis, we can see that via σ_A we can write I_E as a function of I_A and E_b/N_0. If we fix E_b/N_0, we can write $I_E = T(I_A)$, a function of I_A alone.

On a trellis type decoder, we can use the BCJR algorithm [21] to do iterative decoding (details in the next section). Using the same construct, we can feed in I_A as our *a priori* probability values to the BCJR algorithm and extract I_E through the *a posterior* probability values from the BCJR. This will give us numeric values of I_E in terms of I_A.

Using our constrained code as the inner code, Figure 5.4 is an example of EXIT charts for $E_s/N_0 = 3$dB.

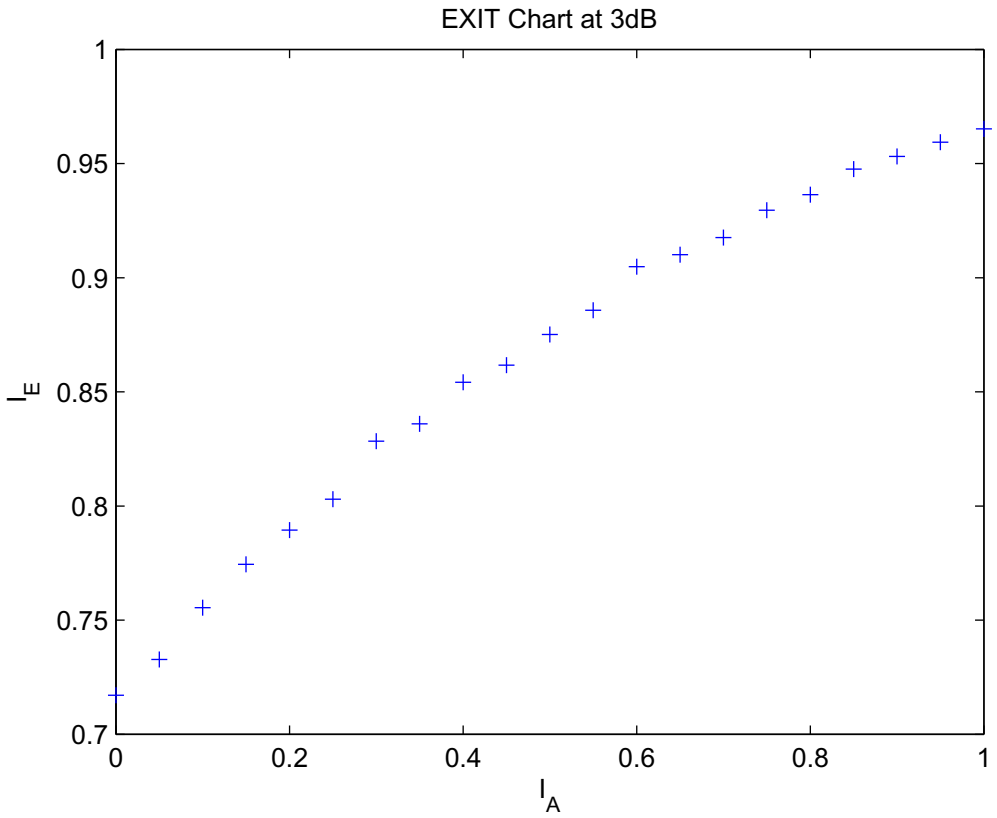

Figure 5.4: EXIT Chart for $E_s/N_0 = 3$dB

Given the EXIT chart of the inner code, we can design outer LDPC codes by a method

illustrated by ten Brink [20]. The essence of the method is to find the appropriate LDPC parameters that matches the EXIT chart curve of the inner code. This turns out to be an optimization problem that can be solved via linear programming [31]. In brief, it suffices to know that given an EXIT chart, we can find the optimal set of LDPC degree distributions. We leave the detailed discussion to Appendix B.

5.3 Message Passing Algorithm

Having designed trellis code and outer LDPC code, we shall now turn our attention to the iterative decoder and briefly explain how iterative decoding is done. Recall that we have the following overall decoder block diagram shown in Figure 5.5.

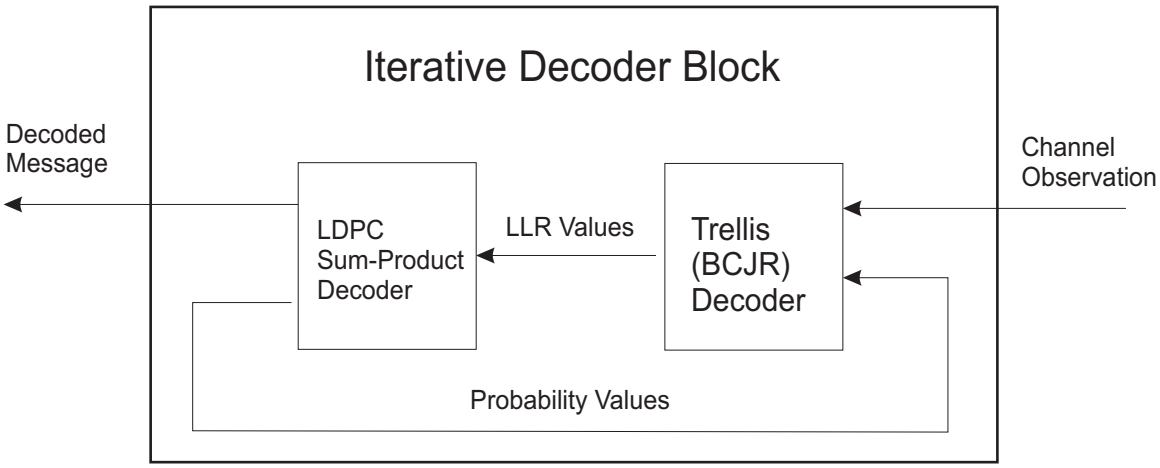

Figure 5.5: Iterative decoding block of our communications system.

We shall first focus on the trellis decoder. Following the the standard convention of [28], the trellis decoder is best represented as a factor graph as shown in Figure 5.6.

The channel observations are in the form of $f(y_i|x_i)$, note that y_i and x_i can be a vector of many bits (5 in our particular model). Symbols x_i, u_i, s_i denote the input variables, output variables, and state variables respectively. The functions T_i are indicator functions of the form $T_i(s_{i-1}, x_i, u_i, s_i)$. Namely, T_i is 1 if and only if s_{i-1}, x_i, u_i, s_i correspond to a valid state transition.

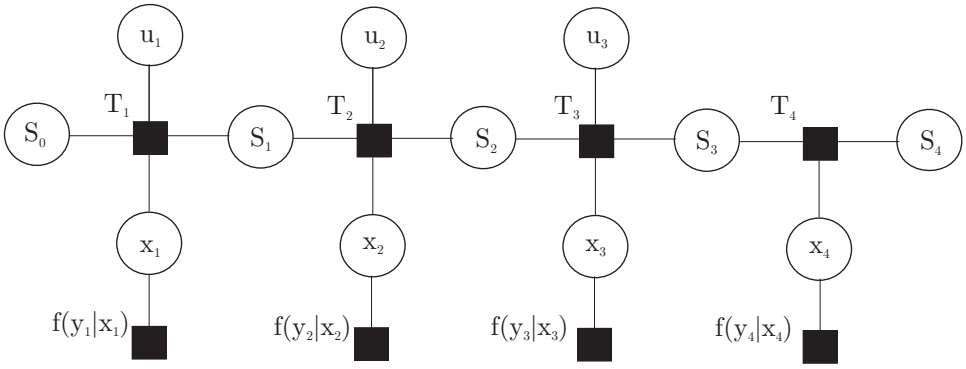

Figure 5.6: Factor graph representation of a trellis

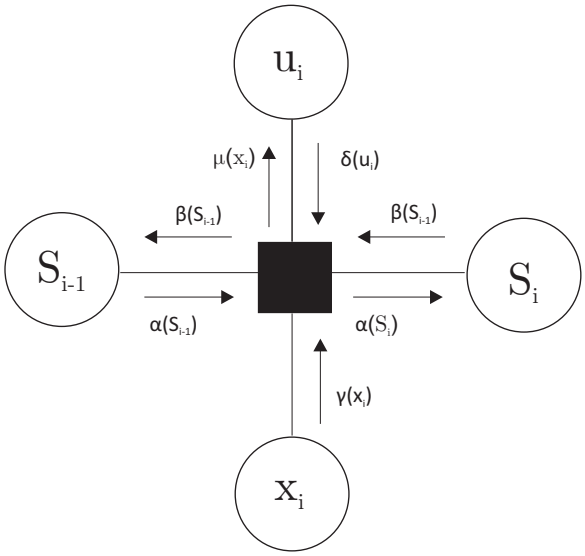

Figure 5.7: Detailed view of a trellis section

With this model, we can focus our attention at a particular trellis section and analyze the message passing around that particular node. Figure 5.7 is a detailed view of a particular trellis section and the corresponding messages. $\gamma(x_i)$ is the message from the channel observation, based on $f(y_i|x_i)$. $\alpha(x_i)$ and $\beta(x_i)$ are messages from the state variables (left and right respectively). $\delta(u_i)$ is the message we receive *from* the outer LDPC code, while $\mu(u_i)$ is the message we wish to pass "upwards" to the LDPC decoder. Note in the case of $\mu(u_i)$, we will convert this to a *log likelihood ratio* (LLR) and pass it to the LDPC decoder. The iterative message passing on the trellis section can be

summarized with the following summations. Note that $\sim x_i$ refers to summation over all variables except x_i, a standard notation in [28].

$$\alpha(s_i) = \sum_{\sim s_i} T_i(s_{i-1}, u_i, x_i, s_i)\alpha(s_{i-1})\gamma(x_i)\delta(u_i) \tag{5.6}$$

$$\beta(s_{i-1}) = \sum_{\sim s_{i-1}} T_i(s_{i-1}, u_i, x_i, s_i)\beta(s_i)\gamma(x_i)\delta(u_i) \tag{5.7}$$

$$\mu(s_i) = \sum_{\sim s_i} T_i(s_{i-1}, u_i, x_i, s_i)\alpha(s_{i-1})\beta(s_i)\gamma(x_i) \tag{5.8}$$

In our particular trellis, we have the setup shown in Figure 5.8. The $e(k)$ in the Figure 5.8 denote the edges involved in the state transition from one state to another. Multiple edges are represented by 1 edge on the graph with an associated list of edge indices. Also associated with each edge are a γ and δ value. We shall use $\gamma(k)$ and $\delta(k)$ to denote the values corresponding to edge k. The transitions into state 2 are highlighted. Let $s_i(j)$ denote the j state at stage s_i. As an example, the summation of $\alpha(s_i)$ at state 2 can be computed as follows:

$$\begin{aligned}
\alpha(s_i(2)) &= \sum_{k \to s_i(2)} \alpha(s_{i-1})\gamma(k)\delta(k) && (5.9)\\
&= \sum_{j=1}^{6} \sum_{k \to s_i(2)} \alpha(s_{i-1}(j))\gamma(k)\delta(k) && (5.10)\\
&= \sum_{j=1}^{6} \sum_{k \in \{5-9,23-26,39-42,56-58,74-75,89-92\}} \alpha(s_{i-1}(j))\gamma(k)\delta(k), && (5.11)
\end{aligned}$$

where the notation $k \to s_i(2)$ denotes the set of edges that go into $s_i(2)$.

Now, passing the message μ up to the LDPC decoder, we need to specify the message passing from a variable node to a check node and vice versa (see Figure 5.9 and 5.10). Towards this end, we shall refer to the treatment presented in [24, 31]. Let the set of messages towards a variable node v with degree d_v be denoted by the set $\{m_i\}$. Then the outgoing message via edge i is summation of all incoming values from all other edges plus the μ value from our inner code. Namely:

$$message_{v \to c} = \sum_{j=1, j \neq i}^{d_v} \log \frac{p(m_j|v=+1)}{p(m_j|v=-1)} + \mu_v \tag{5.12}$$

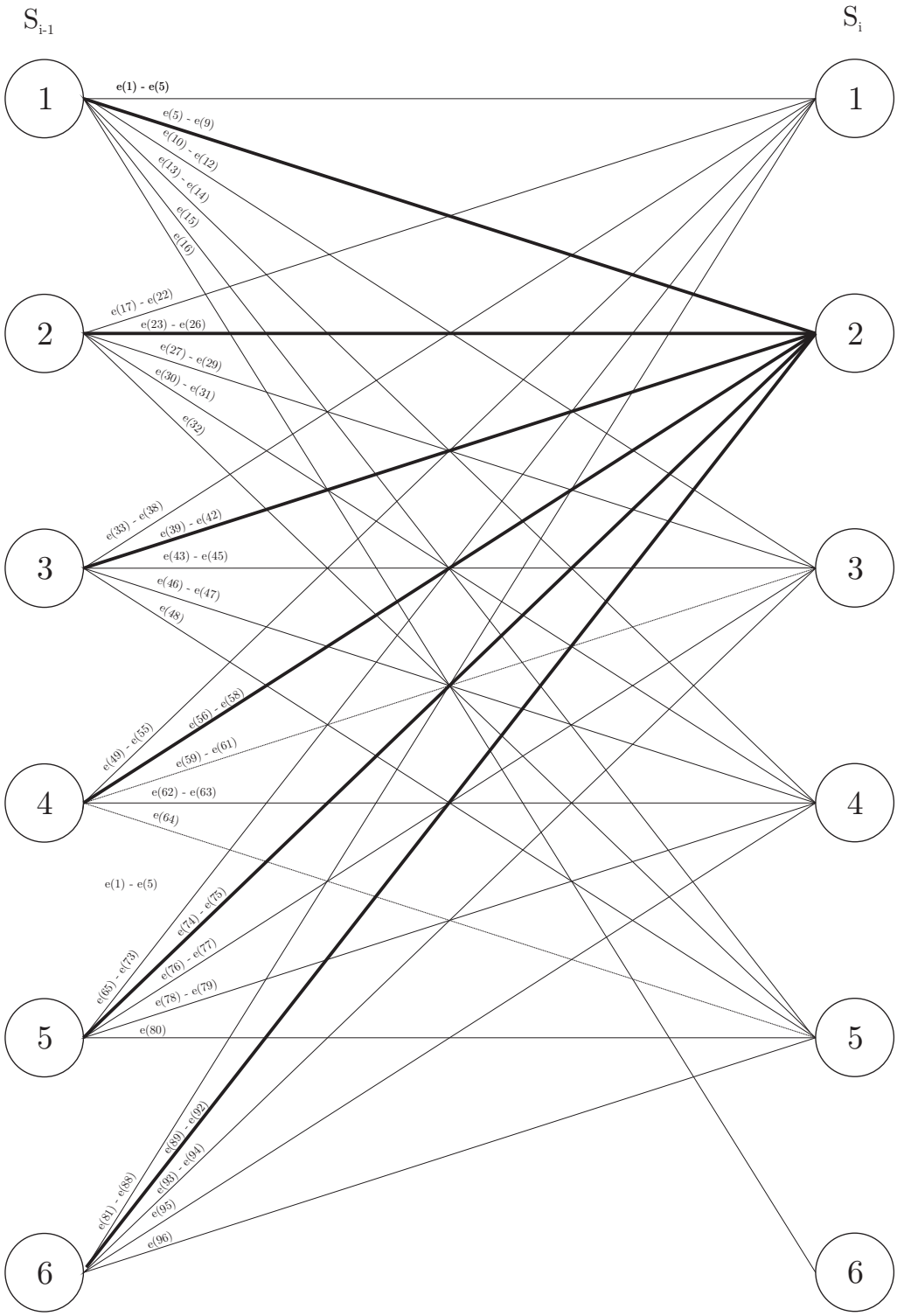

Figure 5.8: Detail view of trellis transition with edges going into state 2 highlighted

Note that in LLR domain $+1$ and -1 corresponds to the binary value 0 and 1 respectively. μ_v refers to the μ value that corresponds to our particular variable node v.

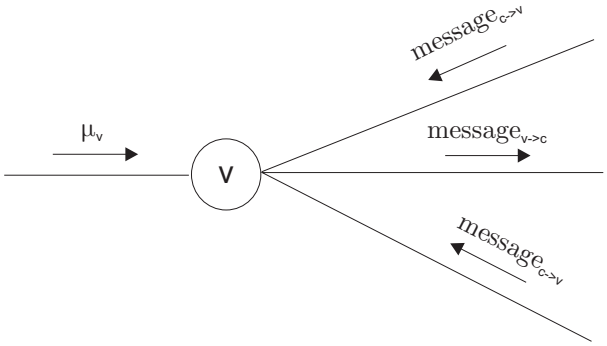

Figure 5.9: Message passing on a variable node

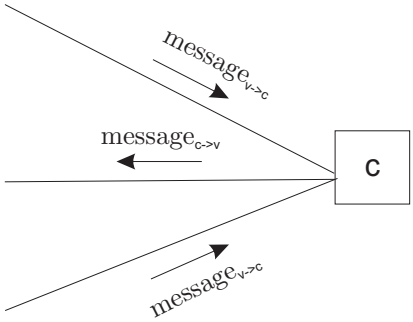

Figure 5.10: Message passing on a check node

In the check node to variable node direction, the set of messages towards a check node of degree d_c are the LLR values, denoted by a set $\{L_i\}$. The outgoing message from the check node via a particular edge i is given by:

$$message_{c \to v} = 2 \tanh^{-1} \Big(\prod_{j=1, j \neq i}^{d_c} \tanh(\frac{L_j}{2}) \Big) \tag{5.13}$$

With the above list of equations, we have completely specified the message passing in our iterative decoding system.

5.4 Simulation Results

In this section we shall put together all the theory we have described in the above sections and simulate our entire communication system and evaluate its performance.

5.4.1 Result from EXIT Chart Analysis

From our EXIT chart analysis, we can calculate the SNR required to achieve certain rates for our outer LDPC code. Since we do not know the overall code rate of the concatenated system, we shall measure SNR in terms of E_s/N_0. We recall that our inner code has duty cycle $\mathcal{D} = 36.47\%$. Thus, we shall set $E_s = 0.3647$ and calculate N_0 according.

As a first step analysis, we restrict the check degree of our outer LDPC code to be ≤ 20. Figure 5.11 shows that we can achieve rate of 0.8 and 0.9 at SNR of 2.2 dB and 4.1 dB respectively. Setting rates of 0.8 and 0.9 as our target rates for designing outer codes, we examine the check degrees required to achieve such rates as a function of SNR. We shall limit ourselves to maximum check degree of 50 as decoding complexity increases for large check degrees. Figure 5.12 shows that for rate 0.8, the minimal check degree required is 10 and for check degree upper bounded by 50, we require at least a SNR value of 1.9 dB. Figure 5.13 shows that for rate 0.9, the minimal check degree required is 20 and for check degree upper bounded by 50, we require at least a SNR value of 3.9 dB.

In Table 5.1 and 5.2, we plot the variable node distributions associated with each of the check degrees in Figure 5.12 and 5.13. We shall use V_d to denote the variable node degree, and λ_{V_d} to denote the percentage of edges incident to the given V_d.

From Table 5.1 and 5.2, we can see that for large check degrees, a significant portion of high variable degree nodes are necessary. Thus, in order to arrive at a lower SNR, we not only need to increase the complexity at the check nodes, we also have to significantly increase the complexity at the variable nodes. In the other extreme, however, we see that for low check degrees, we can achieve low variable degree distributions . Specifically, for

Table 5.1: Table of Variable Node Distribution for a Fixed Check Node (Rate =0.8)

C_d	V_d	λ_{V_d}
47	2	0.1904
	72	0.0251
	73	0.7844
33	2	0.2785
	57	0.4218
	58	0.2997
23	2	0.4080
	43	0.5821
	44	0.0099
16	2	0.6015
	29	0.3985
13	2	0.7557
	28	0.2443
10	2	1

Table 5.2: Table of Variable Node Distribution for a Fixed Check Node (Rate =0.9)

C_d	V_d	λ_{V_d}
28	2	0.6805
	18	0.2781
	19	0.0414
22	2	0.9012
	19	0.0988
20	2	1

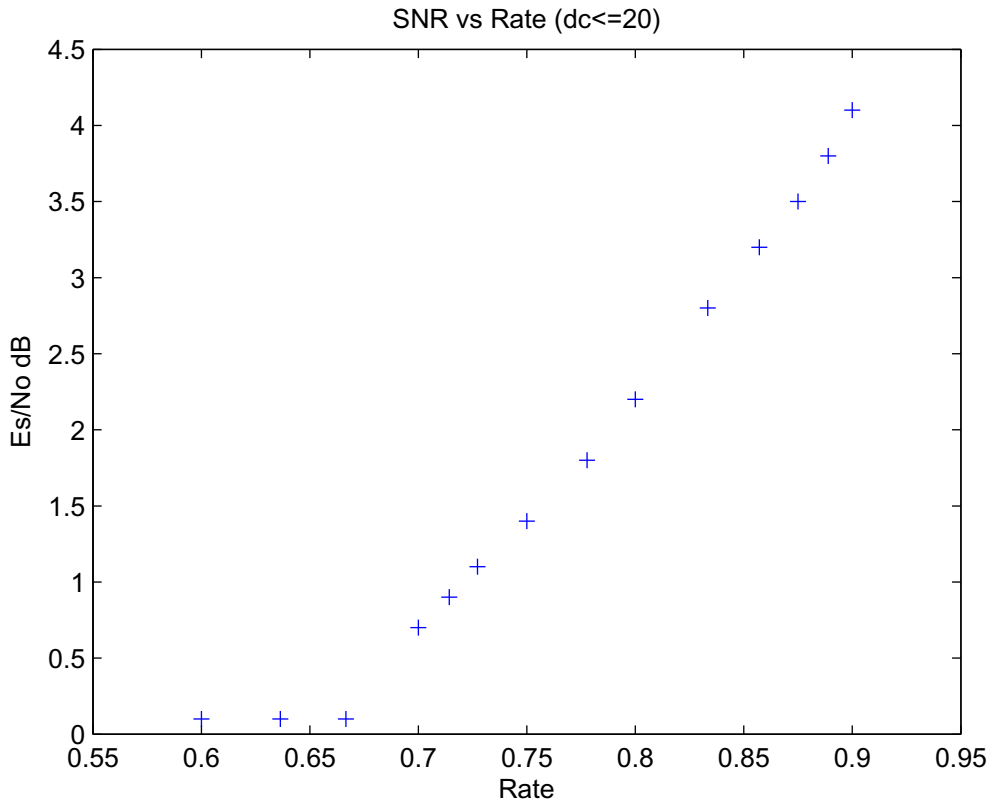

Figure 5.11: Required SNR to achieve desired rates for outer LDPC codes

the case of rate 0.8 code, we can use a regular $(10, 2)$ LDPC code with a threshold value of 2.4 dB. Similarly, for rate 0.9 code, we can use a regular $(20, 2)$ LDPC code with a threshold value of 4.1 dB. In the former case, we only sacrifice 0.5 dB for a significant reduction in complexity. And in the latter case, we actually only sacrifice 0.2 dB. With this trade off between complexity and SNR gain in mind, we will henceforth fix our outer codes to be the $(10, 2)$ and $(20, 2)$ LDPC code for rates 0.8 and 0.9 respectively.

Lastly, we shall note that in the LDPC literature, a LDPC code with every variable node having degree 2 is called a *cycle code*, and has been shown to satisfy many nice properties [33].

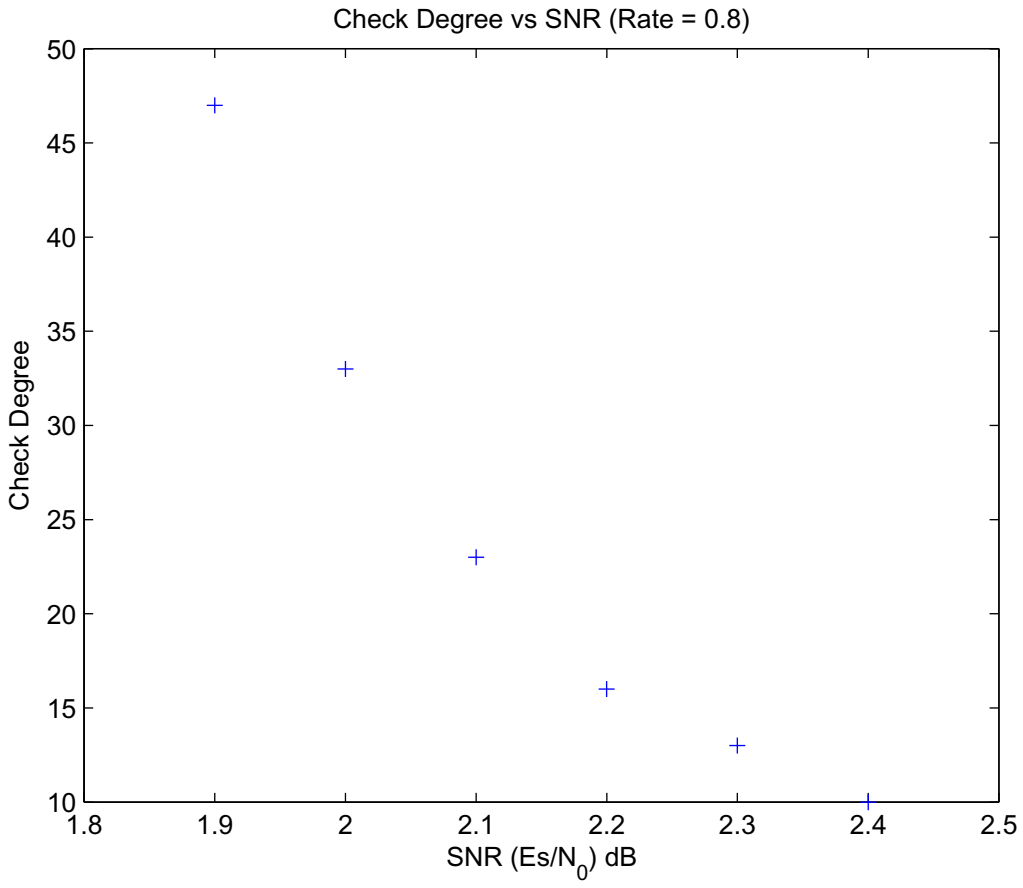

Figure 5.12: Check degree versus SNR required to achieve LDPC code rate of 0.8

5.4.2 Simulation of Concatenated Systems

We shall now present the simulation results of our concatenated system. In our simulations, we allow up to 100 iterations of message passing between the inner and outer decoders. In Figure 5.14 and 5.15, we present the bit-error-rate (BER) curves for rate 0.8 and 0.9 outer LDPC code respectively. In each case, we used outer LDPC codes of blocklength 10^4 and 10^5.

Figure 5.14 and 5.15 shows the performance with LDPC code of length 10^4 and 10^5. In the blocklength of 10^4 case, for rate 0.8, we can achieve a BER of 10^{-5} within 0.5 dB of EXIT chart threshold; and for rate 0.9, we can achieve a BER of 10^{-5} at 0.4 dB from the EXIT chart threshold. In the blocklength 10^5 case, the performance is even better. For

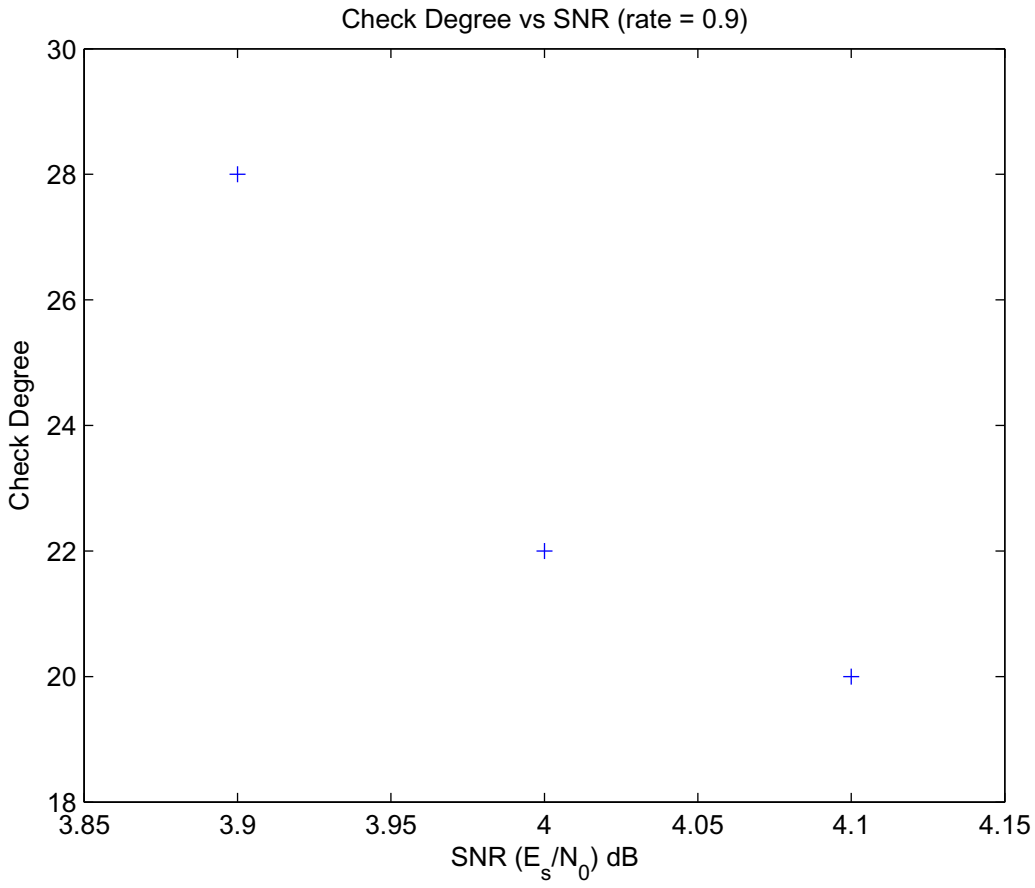

Figure 5.13: Check degree versus SNR required to achieve LDPC code rate of 0.9

rate 0.8, we can achieve a BER of 10^{-6} within 0.2dB of the EXIT chart threshold; and for rate 0.9 case, we can achieve a BER of 10^{-6} within a mere 0.15 dB of the the EXIT chart threshold. We shall henceforth use the results for blocklength 10^5 to compare our systems with comparable MPPM systems. In concatenation of inner code, the rate 0.8 LDPC gives an overall rate 0.64 concatenated system and the rate 0.9 LDPC gives an overall rate 0.72 concatenated system.

5.4.3 Estimation of Shannon Limit

While we can approach the threshold predicted by the EXIT chart, it is more valuable to find out how far we are from the Shannon limit. Specifically, we want to find out the

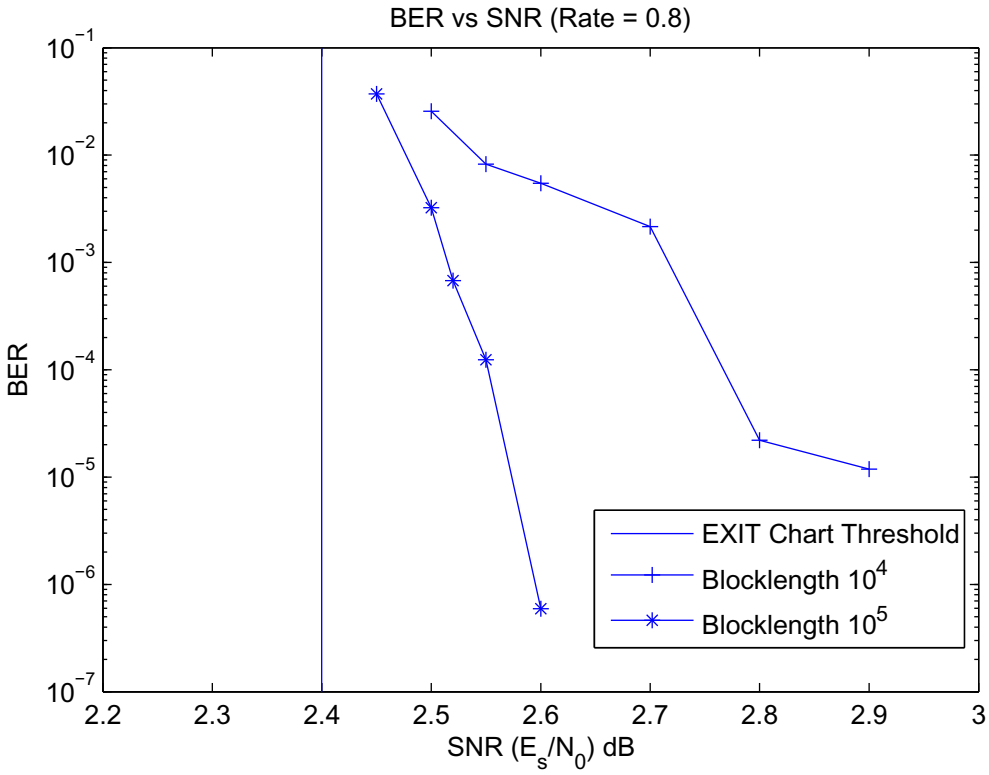

Figure 5.14: BER curve for the concatenated system with rate 0.8 outer LDPC code

Shannon limit of sending binary signals over AWGN channel that satisfies both our duty cycle and zero runlength constraints at our target rates of 0.64 and 0.72. Towards this end, we try to find the SNR needed to achieve these target rates. This problem proves to be difficult; and so far we have not been able to find a good way to incorporate both constraints into one capacity calculation. Thus, in this section we shall only provide a *lower bound* on the Shannon limit.

It is easy to calculate the capacity (as a function of SNR) if we ignore the zero runlength constraint and focus only on the duty cycle. Namely, if we are sending $x \in \{0, 1\}$, and receive $Y = X + Z$, where Z is a zero mean Gaussian with variance σ^2, then the mutual information between X and Y is given by:

$$I(X : Y) = \sum_{x=0,1} \int_{-\infty}^{+\infty} f(y|x)p(x) \log \frac{f(y|x)}{f(y)} dy \tag{5.14}$$

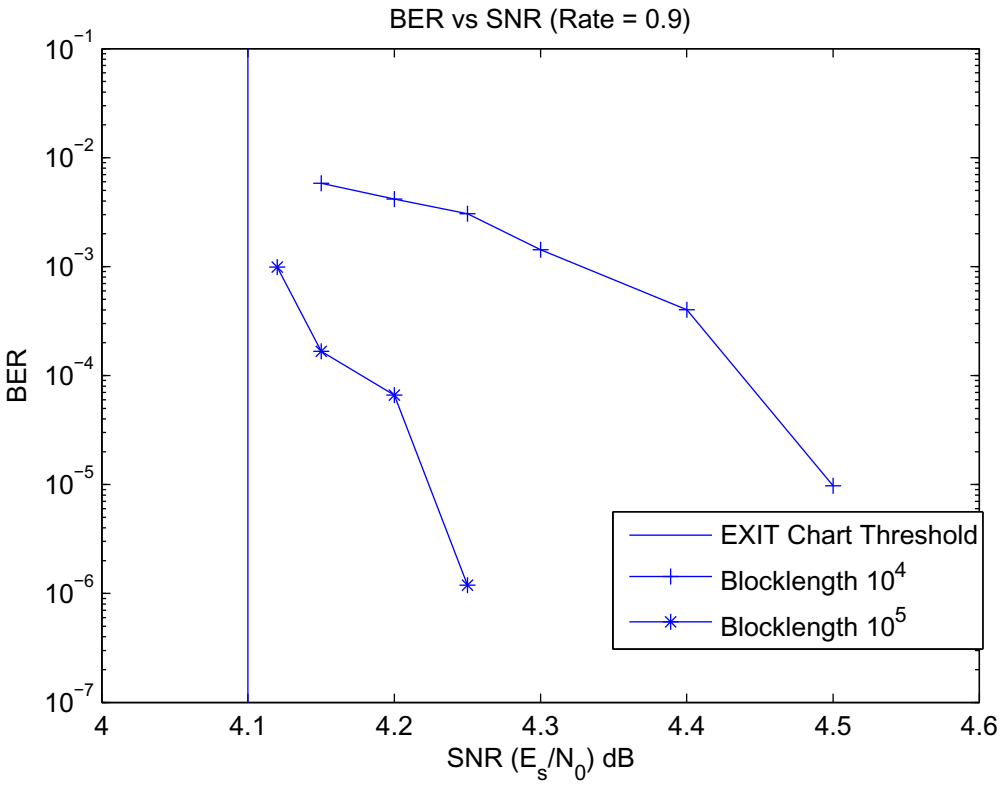

Figure 5.15: BER curve for the Concatenated System with rate 0.9 outer LDPC code

We can express $f(y)$ by the following:

$$f(y) = p(x = 0)f(y|x = 0) + p(x = 1)f(y|x = 1), \tag{5.15}$$

and $f(y|x)$ is given by:

$$f(y|x = a) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y-a)^2}{2\sigma^2}}. \tag{5.16}$$

Our duty cycle constraint tells us exactly the distribution $p(x)$. With this, varying the value of σ, we can find the required σ values that give us capacity of value of 0.64 and 0.72 respectively. Since we ignored the zero runlength constraint, this method gives us an under-estimate on the SNR needed to achieve the given capacity values.

Using this method, we arrived at Shannon limit values of approximately $E_s/N_0 = 1$ dB for rate 0.64 and $E_s/N_0 = 2.2$ dB for rate 0.72. We shall further discuss the implications of these values in the following section. Lastly, we emphasize that the Shannon limit

values obtained are *lower bounds*.

5.4.4 MPPM Comparison

In this section, we state our main results. We shall show that our concatenated system achieves a significant coding gain (in terms of BER vs SNR) as compared to MPPM systems. Furthermore, our code actually comes relatively close to the Shannon limit.

To make a fair comparison with MPPM, throughout this section we shall measure SNR in terms of E_b/N_0. Figures 5.16 and 5.17 shows the performance of our concatenated systems against comparable MPPM systems as well as our lower bound Shannon limit estimate. As shown in Figure 5.16, our rate 0.64 concatenated system performs over 6 dB better than a comparable MPPM system at a BER of 10^{-5}. More importantly, its performance is within 1.5 dB of the Shannon limit. Similarly, in Figure 5.17, our rate 0.72 concatenated system performs around 5dB better than the comparable MPPM system at a BER of 10^{-5} and is within 2 dB of the Shannon limit.

In Table 5.3 and 5.4, we list a few other comparable MPPM systems. From these, we can verify that our concatenated system achieves similar coding again over all these systems as well.

Lastly, we shall again emphasis that our Shannon limit estimate is a *lower* bound, so we are actually closer to the real Shannon limit value than our graphs indicate. In addition, if we were to employ more complex LDPC codes (i.e., those with higher check and variable degrees), we can push our performance even closer to the limit.

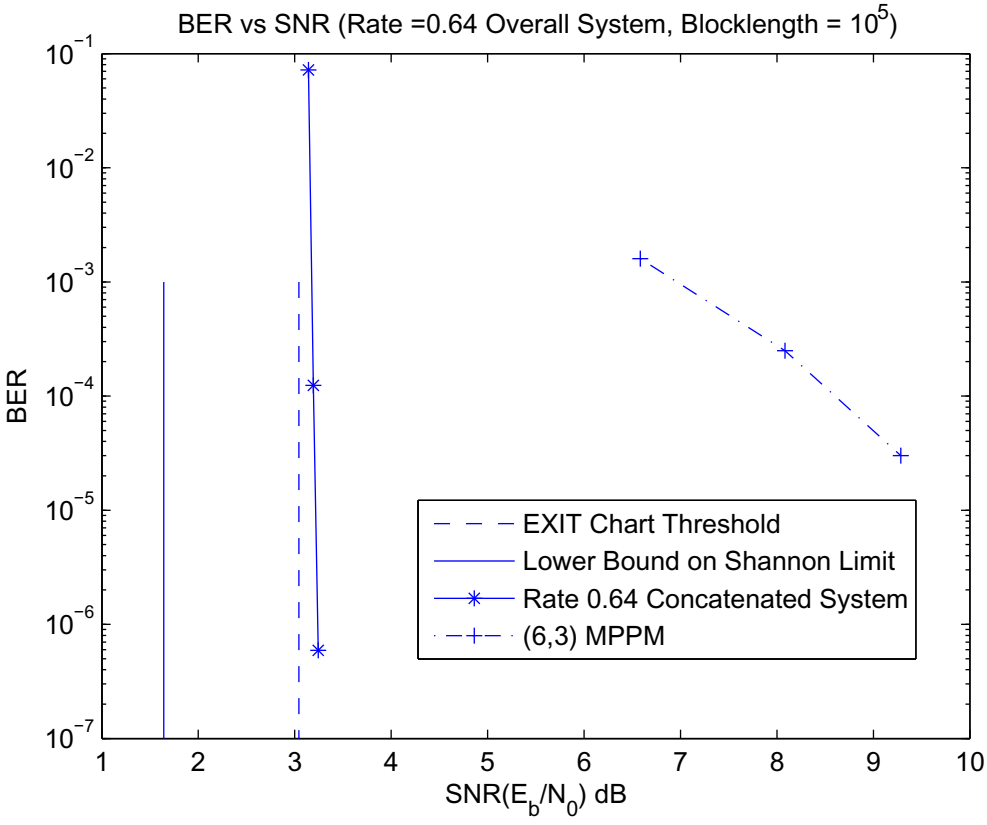

Figure 5.16: Comparison against MPPM system using overall rate 0.64 concatenated system

Table 5.3: Comparable MPPM systems for overall rate 0.64 concatenated system

$M(n,k)$	$\lvert M(n,k)\rvert$	$\lvert\bar{M}(n,k)\rvert$	$\bar{C}(n,k)$	$\bar{\mathcal{Z}}$	SNR for BER $= 10^{-5}$
$M(5,2)$	10	8	0.6	4	9.3
$M(6,3)$	20	16	0.667	4	9.4
$M(9,3)$	84	64	0.667	8	9.4

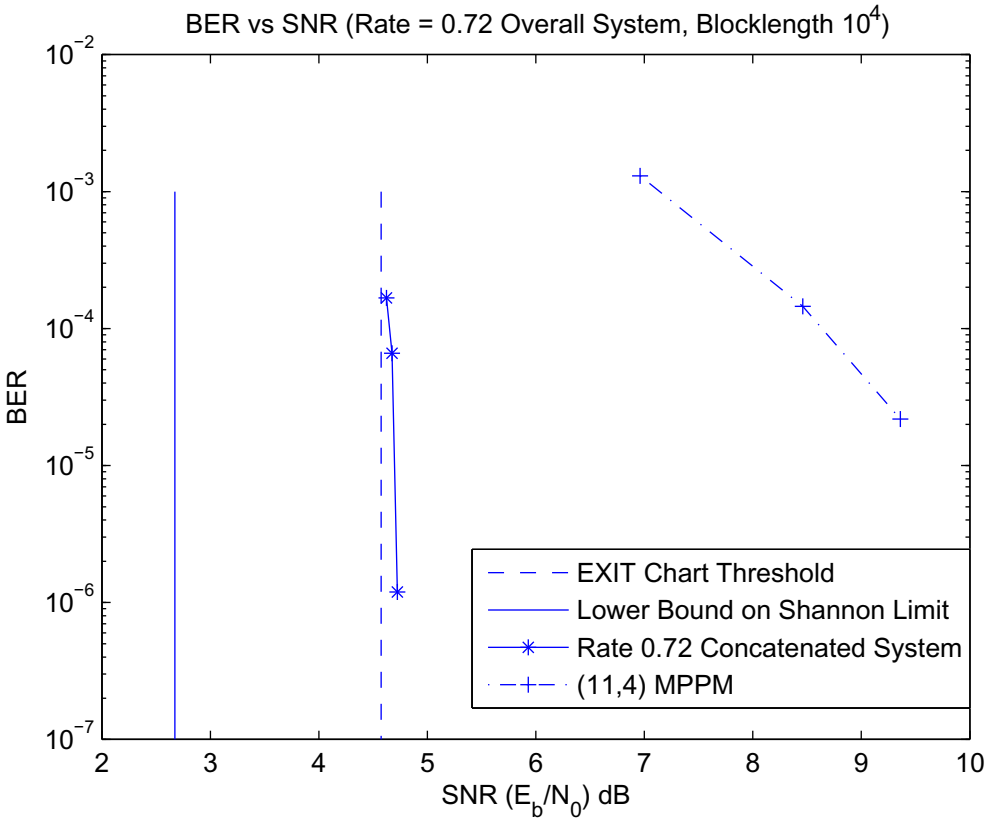

Figure 5.17: Comparison against MPPM system using overall rate 0.72 concatenated system

Table 5.4: Comparable MPPM systems for overall rate 0.72 concatenated system

$M(n,k)$	$\lvert M(n,k)\rvert$	$\lvert \bar{M}(n,k)\rvert$	$\bar{C}(n,k)$	$\bar{\mathcal{Z}}$	SNR for BER $= 10^{-5}$
$M(10,4)$	210	128	0.7	7	9.4
$M(11,4)$	330	256	0.727	9	9.5
$M(13,5)$	1287	1024	0.769	11	9.5

Chapter 6

Conclusion

6.1 Summary of Contributions

In this thesis, we outlined three key areas where significant improvements of current MPPM coding systems are desired. The three areas are as follows:

1. efficient mapping algorithm;

2. higher rate;

3. suitable for iterative decoding.

In Chapter 2, we provided a more efficient mapping algorithm within the framework of MPPM. Comparisons showed that our mapping algorithm can provide significant reductions in the sizes of MPPM lookup tables.

In Chapter 3, we showed through simulation results that significant rate improvement over MPPM systems are indeed possible (while satisfying the same constraints). This set of results also provided us with a region in which we can look for higher rate codes.

Chapter 4 highlights our most important contributions to this subject. We took a constrained coding approach and in particular investigated the prosperities of $(0, k)$ RLL sequences. By taking power graphs and using a greedy edge pruning algorithm, we were

able to construct codes with comparable duty cycle and zero runlength constraints as MPPM systems while achieving much higher rates. Furthermore, by the constraint graph structure of our construct, our new constrained codes automatically have an efficient mapping algorithm and are suitable for iterative decoding.

In Chapter 5, we concatenated an example of our constrained code with an outer LDPC code and performed iterative decoding. Simulation results showed that we were able to achieve an enormous gain of 6 dB over comparable MPPM systems for the same BER. Furthermore, analysis showed that our concatenated system can perform *at least* within 1.5 dB of the Shannon limit.

In sum, our new code design was able to achieve all three areas of significant improvements over MPPM that were desired. Furthermore, in concatenation with outer LDPC code, our code demonstrated overwhelmingly better performance than comparable MPPM systems.

6.2 Directions for Future Research

While we have shown codes that provide significant improvements over MPPM systems, there are still numerous open questions left unanswered.

In Chapter 2, we outlined a particular method that can reduce the size of MPPM lookup tables. It is natural to ask if there are even better mapping algorithms out there. The existence and construction (or non-existence) of such mapping algorithms are interesting theoretical questions to be answered.

In Chapter 3, we used simulations to approximate the capacity rates for fixed duty cycle and zero runlength constraints. It is useful to ask if we can find close forms of the capacity rates (as n goes to ∞). Possible approaches could be attempting to derive close forms based on the generating function we had. This question is only of theoretical interest and does not affect practical implementations.

In Chapter 4, we provided an algorithm as an attempt to minimize the overall duty cycle of the code. However, we did not prove that our method actually gives the minimal construction (in terms of duty cycle). For both theoretical and practical interest, it is important to either prove that our construction is indeed duty cycle minimizing or provide an algorithm that gives the actual duty cycle minimizing construction.

Furthermore, there are many aspects of our code constructions in which further analysis is needed. For example, it would be interesting to analyze the duty cycle tradeoff of constructing a p/q code versus a kp/kq (for integer $k > 1$) code. An example of such would be the trade off between constructing a 2/3 code versus a 4/6 code. Clearly, the 4/6 code is more complex. But at the same time, do we get a reduction in duty cycle? Analysis of such tradeoffs would result in better code designs.

In Chapter 5, we were only able to provide a lower bound on the Shannon limit calculation. It is desirable to find an exact formula or at least an accurate approximation of the actual Shannon limit. Such an analysis will shed light on exactly how powerful our codes are and how far away we are from the threshold. Furthermore, in our simulations, we assumed Gaussian noise and coherent detection. It would also be interesting to consider noncoherent detection.

Lastly, we note that we really only found one special method of constructing codes that satisfy both duty cycle and zero runlength constraints. It will be interesting to find entirely different construction methods that could potentially perform better than the systems we provided.

Appendix A

Finite State Codes

In this appendix, we shall go through some key definitions in the language of symbolic dynamics and constraint graphs. These definitions will lead us to the *Finite-State Coding Theorem* which puts the work in Chapter 4 in solid theoretical framework. Readers who wish to gain deeper understanding of this theory are encouraged to read [18]. In our discussion, we shall faithfully follow the conventions used in [18].

A.1 Basis Definitions

Definition A.1.1 *Let $\mathcal{A}$ be a finite alphabet. The full $\mathcal{A}$-shift is the collection of all bi-infinite sequences of symbols from $\mathcal{A}$, and is denoted by $\mathcal{A}^{\mathbb{Z}}$.*

Given an alphabet $\mathcal{A}$, a *block* over $\mathcal{A}$ is a finite sequence of symbols from $\mathcal{A}$. A *k-block* is simply a block of length k. For example, let $x = (x_i)_{i\in\mathbb{Z}}, x_i \in \mathcal{A}$ be a sequence. We can denote $x_{[i,j]} = x_i x_{i+1} \dots x_j$ as a block of length $j - i + 1$.

Let $\mathcal{F}$ be a collection of blocks over $\mathcal{A}$. Define $X_{\mathcal{F}}$ to be the subset of $\mathcal{A}^{\mathbb{Z}}$ which do *not* contain any block in $\mathcal{F}$. In some sense, we can think of $\mathcal{F}$ as a set of *forbidden blocks*. This allows us to define the following:

Definition A.1.2 *A shift space (or simply shift) is a subset X of $\mathcal{A}^{\mathbb{Z}}$ such that $X = X_{\mathcal{F}}$*

for some collection $\mathcal{F}$ of forbidden blocks over $\mathcal{A}$.

To this point, we have talked about shifts over a alphabet $\mathcal{A}$, we can further expand this notion by looking at alphabets in terms of blocks. Namely, let $\mathcal{A}_X^{[N]}$ be the set of all N-blocks that are allowed in a shift X. We can regard $\mathcal{A}_X^{[N]}$ as an *alphabet* on its own, and over which we could form the full $\mathcal{A}_X^{[N]}$-shift $(\mathcal{A}_X^{[N]})^{\mathbb{Z}}$. This allows us to define the *higher block code* $\beta_N : X \to (\mathcal{A}_X^{[N]})^{\mathbb{Z}}$ by:

$$(\beta_N(x))_{[i]} = x_{[i,i+N-1]} \tag{A.1}$$

Now, it is natural for us to define the *higher block shift* $X^{[N]}$ as the image (of X) $X^{[N]} = \beta_N(X)$ in the full shift over $\mathcal{A}_X^{[N]}$. This definition will be useful when we consider codewords in blocks, as they arise naturally by taking sequences in blocks each time. Lastly, we shall define the shifts that characterize our RLL sequences.

Definition A.1.3 *A shift of finite type is a shift space that can be described by a finite set of forbidden blocks. Furthermore, a shift of finite type is M-step if it can be described by a collection of forbidden blocks all of which have length $M + 1$.*

As a cumulating example justifying the above definitions, consider the $(1, \infty)$ RLL sequence. This is a shift space with forbidden block $\mathcal{F} = \{11\}$. Thus, it is a shift of finite type; and since the only forbidden block has length 2, it is 1-step. If we let X be this shift space, then looking at $X^{[2]}$ we have sequences over the alphabet $\{0, 1\}^2$ where the $11 \in \{0, 1\}^2$ is forbidden.

A.2 Graph Representations of Shifts

As we have justified RLL sequences as shifts of finite type, we shall now show it is no coincidence that they have graph representations. First we need a few definitions.

Definition A.2.1 *A graph G is a collection consisting of a finite set $\mathcal{V} = \mathcal{V}(G)$ of vertices (or states) together with a finite set $\mathcal{E} = \mathcal{E}(G)$ of edges. Each edge $e \in \mathcal{E}(G)$ starts at a vertex denoted by $i(e) \in \mathcal{V}(G)$ and terminates at a vertex $t(e) \in \mathcal{V}(G)$.*

Definition A.2.2 *Let G be a graph with edge set $\mathcal{E}$. The edge shift X_G is the shift space over the alphabet $\mathcal{A} = \mathcal{E}$ given by:*

$$X_G = \{\xi = (\xi_i)_{i\in\mathbb{Z}} \in \mathcal{E}^{\mathbb{Z}} : t(\xi_i) = i(\xi_{i+1}) \forall i \in \mathbb{Z}\} \tag{A.2}$$

From this we can see that if we traverse a graph and record its edge labeling, the edge labeling gives us a shift space. Thus, we can produce shift spaces from graphs. The converse to this is guaranteed by the next theorem proven in [18]:

Theorem A.2.3 *If X is an M-step shift of finite type, then there is a graph G such that $X^{[M+1]} = X_G$.*

A.3 Finite-State Code Theorem

Now that we have justified using graph representations of shifts. We now show how we can construct *finite-state codes* using graphs.

Definition A.3.1 *Let G be a graph with edge set $\mathcal{E}$. A road-coloring $\mathcal{C}$ of G is a labeling $\mathcal{C} : \mathcal{E} \to \mathcal{A}$ that establishes, for each state I of G, a one-to-one correspondence between $\mathcal{A}$ and the edges of G starting at I.*

Definition A.3.2 *A labeled graph is right-closing with delay D if whenever two paths of length $D + 1$ start at the same state and have the same label, then they must have the same initial edge. A labeled graph is right-closing if it is right-closing with some delay $D \geq 0$.*

These two definitions serve the purpose of eliminating graphs that have ambiguous labeling. With them in mind, we can finally define finite-state codes.

Definition A.3.3 *A finite-state code is a triple $(G, \mathcal{J}, \mathcal{O})$, where G is a graph called the encoder graph, $\mathcal{J}$ is a road-coloring of G called input labeling, and $\mathcal{O}$ is a right-closing labeling of G called the output labeling. If G has constant out-degree n and if X is a shift containing all output sequences (from the output labeling) of G, then $(G, \mathcal{J}, \mathcal{O})$ is called a finite-state (X, n)-code.*

Finally, we shall state the finite-state coding theorem:

Theorem A.3.4 ***(Finite-State Coding Theorem).*** *Let X be a shift of finite-type and $n \geq 1$. Then there is a finite state (X, n)-code if and only if $h(X) \geq \log n$.*

Appendix B

LDPC Code Design from EXIT Charts

In this appendix, we shall briefly describe a method of designing LDPC codes from EXIT charts. We shall follow the treatment detailed in [20].

B.1 Intuitive Idea

From our message passing between the check nodes and variable nodes of an LDPC code, we see that the *extrinsic* information computed through the variable nodes serves as *a priori* information for the check nodes. This extrinsic information, of course, will depend on the exact underlying LDPC structure. Namely, the degree distributions of both variables nodes and check nodes.

Now key the observation is the following. Let the EXIT chart of a prior decoder (in our case the inner decoder) be given. Suppose that from the EXIT chart we can characterize the extrinsic information at the variable nodes as a function of the degree of the variable nodes (v_d); and we can equally characterize the a priori information at the check nodes as a function of the degree of the check nodes (c_d). Then, knowing the fact that these two functions should match (or come close to matching), we can deduce the

appropriate values for v_d and c_d so that these functions match. This is the key idea of designing LDPC codes from EXIT charts.

B.2 Derived Equations

We shall first define a set of notations. We shall denote the a priori and extrinsic information of the inner decoder $I_{A,DET}$ and $I_{E,DET}$ respectively. Similarly, we shall denote the extrinsic information at the variable nodes and the a priori information at the check nodes $I_{E,VND}$ and $I_{A,CND}$ respectively.

With these, we recall (from Chapter 5) that the EXIT chart of the inner decoder is simply a graphical representation of $I_{E,DET}$ as a function of $I_{A,DET}$. Also, recall that we have a function J that relates a priori mutual information and its variance σ_A (equations (5.3) and (5.4)). Now, it has been shown in [20] that

$$I_{A,DET}(I_{A,VND}, d_v) = J(\sqrt{d_v} J^{-1}(I_{A,VND})) \tag{B.1}$$

where $I_{A,VND}$ is the a priori information to the variable nodes.

Thus, if we allow $I_{A,VND}$ to vary from 0 to 1, we get a set of values for $I_{A,DET}$, which in turn gives us values for $I_{E,DET}$ via the EXIT chart of the inner decoder. Now, [20] further showed that $I_{E,VND}$ can be modeled as

$$I_{E,VND}(I_{A,VND}, I_{E,DET}, d_v) = J(\sqrt{(d_v - 1)[J^{-1}(I_{A,VND})]^2 + [J^{-1}(I_{E,DET}]^2}). \tag{B.2}$$

Note that $I_{E,DET}$ is determined by $I_{A,VND}$. Furthermore, it was also shown that $I_{A,CND}$ can be modeled by

$$I_{A,CND}(I_{A,VND}, d_c) \approx 1 - J\Big(\frac{J^{-1}(1 - I_{A,VND})}{\sqrt{d_c - 1}}\Big). \tag{B.3}$$

Now, for fixed $I_{A,VND}$, both $I_{E,VND}$ and $I_{A,CND}$ are one variable functions of v_d and c_d respectively. Thus, it suffices to find values of v_d and c_d such that $I_{E,VND}$ and $I_{A,CND}$ agrees for all values of $I_{A,VND}$. In other words, we want to find v_d and c_d such that the

graphs of $I_{E,VND}$ and $I_{A,CND}$ as (a function of $I_{A,VND}$) match (more precisely terms, we want the EXIT chart to be "open", see [19] for details). In addition, we also want to add the constraint that we attain an LPDC code with the highest possible rate. This procedure can be easily done using linear programming as shown in [31].

Bibliography

[1] J.R. Pierce,"Optical channels: practical limits with photon counting," *IEEE Trans. Commun.*, vol. COM–26, pp. 1819–1821, 1978.

[2] D. L. Snyder and I. B. Rhodes, "Some implications of the cutoff rate criterion for coded direct-detection optical communication systems," *IEEE Trans. Inform. Theory*, vol. IT–26, pp. 237–338, Jul. 1981.

[3] H. M. H. Shalaby, "Maximum achievable throughputs for uncoded OPPM and MPPM in optical direct-detection channels," *IEEE J Lightwave Technol.*, vol. LT–13, No. 11. pp. 2121–2128, Nov. 1995.

[4] J. Hamkins and B. Moision, "Multi-pulse PPM on memoryless channels," in *Proc. Int. Symp. Information Theory*, Chicago, IL, June–July 2004, pp. 336.

[5] R. Velidi and C.N. Georghiades, "On symbol synchronization of MPPM sequences," *IEEE Trans. Commun.*, vol.46, no. 5, pp. 587–589, May 1998.

[6] H. Sugiyama and K. Nosu, "MPPM: A method for improving the band-utilization efficiency in optical PPM," *IEEE J Lightwave Technol.*, vol. LT–7, no.3. pp. 465-472, Mar. 1989.

[7] K. Sato, T. Ohtsuki, I. Sasase and S. Mori,"Performance analysis of $(m, 2)$ MPPM with imperfect slot synchronization," *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, vol. 2, pp. 765–768, May 1993.

[8] T. Ohtsuki, H. Yashima, I. Sasase and S. Mori, "Cutoff rate and capacity of MPPM in noiseless photon counting channel," *Trans. IEICE Japan*, vol. E74, no. 12, pp. 4080–4084, Dec. 1991.

[9] C. N. Georghiades, "Modulation and coding for throughput-efficient optical systems," *IEEE Trans. Inform. Theory*, vol. 40, no. 5, Sep. 1994.

[10] G.M. Lee and G. W. Schroeder, "Optical pulse-position modulation with multiple positions per pulsewidth," *IEEE Trans. Commun.*, vol. COM–25, pp. 360–364, Mar. 1977.

[11] J. M. Budinger, M. Vanderaar, P. Wagner, and S. Bibyk, "Combinatorial pulse position modulation for power-efficient free-space laser communications," *SPIE Proc.*, vol. 1866, Jan. 1993.

[12] H. Park and J. R. Barry, "Trellis-coded multiple-pulse-position modulation for wireless infrared communication," *IEEE Trans. Commun.*, vol. 52, no. 4, pp. 643–651, 2004.

[13] R. Razen, J. Seberry, and K. Wehrhahn, "Ordered partitions and codes generated by circulant matrices," *J. Comb. Theory*, Ser. A, 27(3): 333–341, 1979.

[14] A. K. Agarwal, Padmavathamma, and M.V. Subbarao, *Partition Theory*. Chandigarh, India: ATMA RAR & SONS, 2005.

[15] T. T. Nguyen and L. Lampe, "Capacity-maximizing MPPM constellations for free-space optical communications," *Proc. 6th Symposium on Communication Systems, Networks and Digital Signal Processing*, Graz, Austria, July 2008.

[16] S. Hranilovic, "On the design of bandwidth efficient signalling for indoor wireless optical channels," *International Journal of Communication Systems - Special Issue on*

Indoor Optical Wireless Communication Systems and Networks, Wiley Interscience, vol. 18, no. 3, pp.205–228, April 2005.

[17] K. A. S. Immink, *Coding Techniques for Digital Recoders*. Hertfordshire, UK: Prentice Hall, 1991.

[18] D. Lind and B. Marcus, *An Introduction to Symbolic Dynamics and Coding*. New York: Cambridge University Press, 1995.

[19] S. ten Brink, "Convergence behavior of iteratively decoded parallel concatenated codes," *IEEE Trans. Commun.*, vol. 49, pp. 1727–1737, Oct. 2001.

[20] S. ten Brink, G. Kramer, and A. Ashikhmin, "Design of low-density parity-check codes for modulation and detection," *IEEE Trans. Commun.*, vol. 52, pp. 670–678, Apr. 2004.

[21] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minmizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. IT–20, pp. 284–287, Mar. 1974.

[22] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting codes and decoding: Turbo codes," in *Proc. IEEE International Conference on Communications*, Geneva, Switzerland, 1993, pp. 1064–1070.

[23] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inform. Theory,* vol. 47, no. 2, pp. 619–637, Feb. 2001.

[24] T. J. Richardson and R. L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inform. Theory,* vol. 47, no. 2, pp. 599-618, Feb. 2001.

[25] T. J. Richardson and R. L. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 638–656, Feb. 2001.

[26] R. G. Gallager, *Low-Density Parity-Check Codes.* Cambridge, MA: MIT Press, 1963.

[27] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory*, vol. 27, no. 5, pp. 533–547, Sept. 1981

[28] F. R. Kschischang, B. J. Frey, H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inform. Theory*, vol. 47. no 2, pp. 498–519, Feb. 2001.

[29] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman, and V. Stemann, "Practical loss-resilient codes," *Proc. 29th Sympoisum on Theory of Computing,* 1997, pp. 150–159.

[30] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Improved low-density parity-check codes using irregular graphs," *IEEE Trans. Inform. Theory,* vol. 47, no. 2, pp. 585–598, Feb. 2001.

[31] B. Smith, "Low-density partiy-check codes with reduced decoding complexity," M.A.Sc. Thesis, University of Toronto, 2007.

[32] X.-Y. Hu, E. Eleftheriou, and D. M. Arnold, "Regular and irregular progressive edge-growth Tanner graphs," *IEEE Trans. Inform. Theory*, vol. 51, no. 1, pp. 3886–3898, Jan. 2005.

[33] G. Horn, "Iterative decoding and pseudo-codewords," Ph.D. disseration, California Institue of Technology, 1999.