Decoding of LDPC Codes Over Channels with Binary Additive Markov Noise

by

Christopher Michael Nicola

A thesis submitted to the
Department of Mathematics and Statistics
in conformity with the requirements for
the degree of Master of Science (Engineering)

Queen's University
Kingston, Ontario, Canada
September 2005

Abstract

Error-correcting coding schemes using low-density parity-check (LDPC) codes and belief propagation decoders based on the sum-product algorithm (SPA) have recently achieved some of the highest performance results in the literature. In particular, the performance of irregular LDPC codes over memoryless channel models is unrivaled. This thesis explores LDPC decoding schemes which are capable of exploiting the memory of channels based on Markov models to achieve performance gains over systems which use interleaving and assume the channel is memoryless. To this end, we present a joint channel-state estimation decoder for binary additive M^{th}-order Markov noise channels (BAMNCs) using the SPA. We then apply this decoder to the queue-based channel (QBC), which is a simple BAMNC with only four channel parameters. The QBC enjoys closed form equations for the channel capacity and the block transition distribution. The QBC is a versatile model which has been shown to be able to effectively model other channels with memory including the Gilbert-Elliot channel (GEC) and Rayleigh and Rician fading channels. In particular, it is a good model for slow fading phenomena, which is difficult to model using channel models like the GEC. We obtain simulation results that compare SPA decoding of LDPC codes over the QBC to SPA decoding over the memoryless binary symmetric channel (BSC) relative to their respective Shannon limits. These results demonstrate that the performance of the QBC decoding scheme scales well with the QBC's higher Shannon limit and that the decoder effecively exploits the channel memory. We also use the QBC and BAMNC to model the GEC by attempting to match its M^{th}-order channel noise statistics. Simulations are run using the QBC and the more general BAMNC to model the GEC for decoding data transmitted over the simulated GEC. The results show us the effectivenes of this technique. Finally, we examine the performance of irregular LDPC codes over the QBC model. Using irregular codes designed for the GEC and the additive white Gaussian noise (AWGN) chan-

nel we simulate their performance over two different QBCs as well as over the BSC and GEC they were designed for. We measure their performance relative to the performance of regular LDPC codes over the same channels and examine the effect of using irregular LDPC codes over channels they were not designed for. We find that the 'memorylessness' of the channel affects the performance of these two codes more than the type of channel model used. The overall goals of this work are to develop tools and obtain results that allow us to better understand the relationship between LDPC/SPA based error-correcting coding schemes and their performance over channel models with memory. In real-world communication systems channel noise is rarely, if ever, memoryless. Even so, the majority of coding systems use random interleaving to compensate for the burstiness of typical communication channels rather than exploiting the channel's memory. The primary goal of this work is to aid in the design of practical error-correcting coding schemes that can outperform traditional memoryless-model-based schemes through exploiting the memory of the channel.

Acknowledgments

I would like to sincerely thank both of my advisors, Dr. Fady Alajaji and Dr. Tamas Linder, for their guidance and input, as well as their significant contribution to my work and the writing of this thesis. I am most grateful for their patience and time and for agreeing to supervise my research.

I would also like to acknowledge my colleague Dr. Libo Zhong for her help with the queue-based channel, Dr. Radford Neal of the University of Toronto, who's software for decoding LDPC codes was an invaluable tool and Dr. Andrew Eckford, who provided advice on the decoding of LDPC codes over the Gilbert-Elliot channel.

I would like to thank all those in the Department of Math and Statistics at Queen's who have made this such a wonderful place to work and learn. I have thoroughly enjoyed my time here; thanks to all of you.

Finally, a very special thanks goes to my parents for supporting me in my continuous endeavor to be a perpetual student and to my wonderful fiancee for her love and support.

Contents

List of Figures

List of Tables

List of Acronyms

- LDPC - Low-density parity-check
- SPA - Sum-product algorithm
- BSC - (Memoryless) binary symmetric channel
- BAMNC - Binary (symmetric) additive Markov noise channel
- QBC - Queue-based channel
- UQBC - Uniform queue-based channel (equivalent to a FMCC)
- FMCC - Finite memory contagion channel
- GEC - Gilbert-Elliot channel
- AWGN - Additive white Gaussian noise
- HMM - Hidden Markov model
- RDSL - Rate-distortion Shannon limit
- CBER - Channel bit-error rate
- BCJR - Bahl Cocke Jeliniek and Raviv (algorithm)

- DE - Density evolution
- MPF - Marginalize product-of-functions
- ML - Maximum likelyhood
- MAP - Maximum a posteriori probability

Chapter 1

Introduction

1.1 Description of the Problem

Traditional communication systems are made up of three major components: the sender, the channel and the receiver. The sender transmits a signal across a noisy channel which introduces distortion to that signal. The receiver receives the now distorted signal and attempts to recover the original signal.

In the design of any communication system the designers must consider the channel distortion as it will cause errors possibly rendering the received data unusable to the receiver. In general a certain level of signal distortion may be acceptable but it may be necessary to design a system in which the receiver is capable of correcting the errors in the received data in order to bring the distortion down to an acceptable level. This can be accomplished through the use of an *error-correcting coding* scheme.

An error-correcting coding scheme adds two additional components to the communication system described above. A channel encoder, which adds redundancy to the transmitted data and a channel decoder which exploits this redundancy in order to find and correct er-

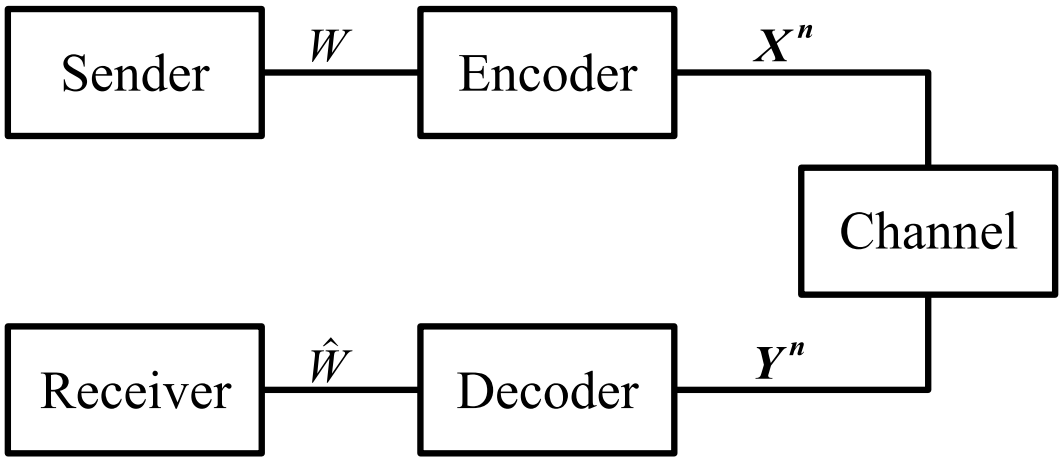

Figure 1.1: Diagram of a basic communication system utilizing an encoder and decoder to correct errors.

rors caused by the channel noise. In Fig. 1.1 we present a block diagram of this system. The *sender* produces a message W to be transmitted which is then encoded by the *encoder* to produce the encoded, length n block of data $\mathbf{X}^n$. The *channel* introduces distortion to that signal producing $\mathbf{Y}^n$ at its output. The *decoder* then attempts to correct the errors in the received signal to produce $\hat{W}$ for the *receiver* which is an estimate of what the original signal W was.

In his pioneering work "*A mathematical theory of communication*" [29], Claude Shannon defines the capacity of a channel as the maximum rate at which data can be sent such that an arbitrarily low probability of error can be achieved in the recovery of the data. The capacity of a binary channel is a value $C \leq 1$. When we say that it is the maximum rate at which data can be reliably sent it means that we must send $r \leq C$ bits of data per bit transmitted across the channel for it to be received reliably.

This implies that we send redundant data across the channel. Assuming that the data is binary, this means sending n bits of data for every $k \leq n$ bits that we wish to communicate across the channel; thus, there are $m = n - k$ redundant bits in the transmitted signal.

Carefully chosen redundant bits can aid the decoder in correcting the errors caused by the channel noise and allow it to recover the original signal.

Example. Consider sending three bits for every one bit of data we wish to send. If we send the sequence $(1, 1, 1)$ every time we wish to send the binary value 1 and $(0, 0, 0)$ every time we wish to send 0, then if there is only a single error in any bit the decoder can look at the other two and correct that error. Unfortunately, the decoder is limited to a single error, for if two errors occur we will decode incorrectly

$$\begin{aligned} 1 : (1, 1, 1) &\implies (1, 0, 1) : 1, \ \mathsf{correct}, \\ 1 : (1, 1, 1) &\implies (0, 0, 1) : 0, \ \mathsf{error}. \end{aligned}$$

Many different types of error-correcting codes have been developed in the almost 60 years since Shannon's original work such as the repetition code shown in the example above. In this work, we focus on LDPC codes, which are binary linear block codes based on linear transformations defined by binary matrices. They have the dual advantage of both being very simple and very effective. In fact, they are some of the best error correcting codes known today. This is particularly interesting since they were actually first proposed [11] only 15 years after Shannon's paper, but the impressive performance of these codes was discovered only just recently [18].

One of the most important considerations in the design of any error-correcting coding scheme is the channel, or more precisely the channel model, it is being designed for. It is important to design the coding scheme based on a channel model that accurately describes the statistics of the channel the code will eventually be used on.

Most commonly, coding schemes are based on simple memoryless channel models. These are models where the noise process is independent. One common example is the

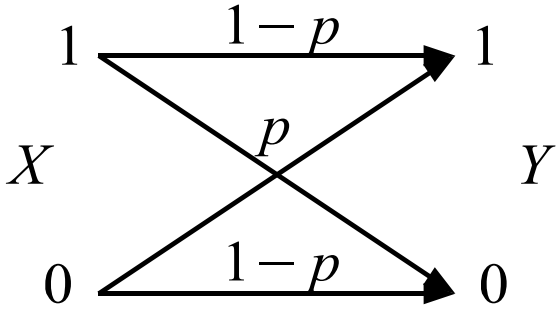

Figure 1.2: Channel transition diagram for the BSC.

memoryless binary symmetric channel (BSC) in which errors are distributed uniformly across the received data. Error occurs with probability p and cause the bits to be 'flipped' from 0 to 1 and vice versa (see Fig. 1.2).

This work examines LDPC decoder designs based on channel models with memory. These are channel models where the channel noise is dependent on past channel noise symbols (as opposed to memoryless channel models where the channel noise is independent). It is significantly more difficult to design decoders for channels with memory as the noise process which describes them is more complex than that of memoryless channels. Nonetheless, channel models with memory are generally better models for real world communication channels since real channel noise is rarely, if ever, truly independent. Since most channels do in fact have memory an interleaver can be used to randomly order the data and make the noise appear memoryless to the decoder. This is typically how many communication systems function in the real world.

There is generally a significant gain in performance achievable by considering the channel memory for most channels with memory. Decoders designed to exploit the memory of the channel are capable of achieving better results than those which assume the channel noise to be memoryless. One of the main goals of this work is to extend the class of channels for which practical estimator-decoders exist to the queue-based channel (QBC)

from [35] for LDPC codes. More generally, this work defines an estimator-decoder for LDPC codes over channels with binary additive Markov noise. The QBC model is a type of binary additive Markov noise channel.

We compare results for our decoder on the QBC to the Gilbert-Elliot channel (GEC) [10, 14] which is a very popular two-state Markov modulated channel. Much work has been done on LDPC decoding for the GEC [6, 12, 25] making it an obvious choice to compare our work on the QBC with. Both the GEC and QBC are binary symmetric channels with additive binary noise and so they are both comparable to the equivalent memoryless channel, the BSC. The BSC is a model which is based on the ideal interleaving of channel noise which has memory.

Both of these channels are based on finite-state Markov models, where the channel state is determined by a Markov process. For each state, the channel has a probability of error associated with it. Usually some states have a particularly high probability of error while others are low. This simulates channels for which errors occur in bursts which correspond to states with a high probability of error.

In order to fully exploit the channel memory in our decoder, we need to estimate the state of the channel for each received bit. This helps produce a better estimate of the probability a particular bit is an error. To achieve this, we use a joint estimator-decoder that performs state estimation and decoding simultaneously using the *sum-product algorithm* (SPA) [15].

The SPA is an iterative decoder which uses belief propagation [21] to estimate the probability of error for each bit based on the relationships between bits and parity-checks for an LDPC code. By further extending the belief propagation to include the relationship between the probability of error and the channel state, the decoder can use the estimates produced by the decoding process to help estimate the channel state, which can then be

used to obtain a more accurate estimate of the probability of error.

This is the basis for the design of SPA based joint estimator-decoders for the GEC presented in [6, 12, 20], and it is also the basis for the design of a joint estimator-decoder for the QBC and more generally the class of binary additive Markov noise channels (BAMNCs) in this work.

1.2 Review of Literature

Low-density parity-check codes were first proposed by Gallager in 1963 [11] as an approach to linear block codes based on sparse parity-check matrices. Each column and row of the parity-check matrix was required to contain a relatively small and fixed number of 1's. The reasoning behind this design was to be able to decode these codes in linear time.

Gallager proposed two different probabilistic decoding methods based on message passing between code-bits and the parity-checks they participate in. This type of decoder is called a *message-passing* decoder. Since each bit participates in a small fixed number of parity-checks, the complexity of this algorithm is linear. Unfortunately, the block lengths and the calculations required for the decoder were still too difficult for computers of the time and hence the performance of these codes was not fully explored.

The discovery of convolutional codes [9] began a new area of study into so called 'non-algebraic codes'. Non algebraic because they were not based on linear transformations using generator and parity-check matrices, nor were they based on the algebraic constructions used for cyclic codes. Convolutional codes are encoded using a finite-state process which gives them a linear order encoding method. They also have linear order [1] decoders such as the the Viterbi algorithm [32] and the BCJR algorithm [2].

[1] Linear order refers to the complexity of the decoding algorithm growing linearly in the length of the code (i.e., the number of computations needed for decoding grows linearly in the length of the code).

Much later, convolutional codes led to the discovery of a class of codes called Turbo codes [3], which are a type of concatenated convolutional codes that use an interleaver to randomize the order of some of the bits. These so-called 'random' codes have excellent performance characteristics and achieve performance well within 0.5dB of the Shannon limit. Turbo codes also utilize a message passing probabilistic decoder to achieve these results in linear time.

Both the BCJR algorithm and the Turbo decoder can be shown [15] to be instances of Pearl's belief propagation [21] on a Bayesian network. This is a method of computing complex probability functions of multiple variables by organizing the relationships between the variables into a network or graph.

Shortly after the discovery of Turbo codes, work on other classes of codes that could be decoded effectively using belief propagation led to the rediscovery of LDPC codes by MacKay and Neal [18] where they demonstrate that LDPC codes are good codes with near Shannon limit performance approaching that of Turbo codes. In later work [17] MacKay also proves that under the assumption of an optimal decoder there exists regular LDPC codes which are in fact capacity achieving.

The decoding of LDPC codes was done through belief propagation on the factor graph representation of the parity-check matrix. The factor graph representation for LDPC codes was first analyzed by Tanner [30] and it is often referred to as a Tanner graph. The message passing algorithm over a factor graph is known as the sum-product algorithm (SPA) and is described in [15]. The authors demonstrate how Pearl's belief propagation, the BCJR algorithm and Turbo decoding can all be implemented as instances of the sum-product algorithm.

These impressive results regarding the performance and low complexity of LDPC codes brought attention back to these long forgotten codes and a significant amount of new work

has been done recently on the analysis of their performance. In particular, a technique called *density evolution* (DE) was developed [27] as a means of determining the limit of performance for a class of LDPC codes under the assumption of the sub-optimal SPA decoding rather than the less practical assumption of optimal decoding used in [17].

The DE technique computes the performance of SPA decoding numerically in the limit of long block length codes and assumes that a large number of iterations are used for decoding. This technique enables the comparison of the average performance of sub-classes of LDPC codes based on the code parameters (i.e., the weights of the rows and columns of the parity-check matrix).

For regular LDPC codes each sub-class of LDPC codes is defined by the constant row and column weights for the parity-check matrix. It was shown that the best rate-1/2 regular LDPC code for the AWGN and BSC channels under SPA decoding is the (3,6) code (three parity-checks per bit and six bits per parity-check) [27].

In order to find LDPC codes with even better performance the definition of LDPC codes was loosened to allow for *irregular* LDPC codes where the number of 1's in the rows and columns of the parity-check matrix was allowed to vary from column to column and row to row. Irregular LDPC codes are determined by their *degree distribution*, which determines the proportion of rows or columns of a particular weight in the parity-check matrix. Using density evolution, Richardson, Shokrollahi and Urbanke were able to evaluate the performance of various degree distributions for the additive white Gaussian noise (AWGN) channel and BSC [26]. They used DE to search for degree distributions which showed particularly good performance limits. The best degree distribution found for the AWGN using this technique has a performance limit under SPA decoding within 0.0045dB of the Shannon limit [4]. While this result assumes impractically long block lengths, this LDPC code still outperforms even Turbo codes over the AWGN when simulated with similar block

lengths and decoder iterations achieving results withing 0.04dB of the Shannon limit at a BER of 10^{-6}.

The next logical step in the construction of irregular LDPC codes is to design codes which perform this well for channels with memory. Recently, there has been work done on extending the SPA decoding algorithm for LDPC codes to exploit the channel memory in models of channels with memory, such as the GEC. In [6, 12, 25], SPA decoding schemes are developed which perform joint channel estimation-decoding for the GEC and this decoder is also generalized to the larger class of Markov modulated channels based on hidden Markov models with more than two states, though the results of these papers have been restricted to the GEC.

Finding good irregular LDPC codes using DE is much more complex for most channels with memory than for memoryless channels. Channels with memory generally have multiple parameters unlike memoryless channels like the AWGN channel or the BSC. Analysing the performance of an irregular code must be done in multiple dimensions instead of simply one. Therfore, it is necessary to characterize the parameters of these channels in such a way that allows one to easily determine how the parameters improve or degrade the channel quality.

In [6, 8] the authors first develop a joint estimator-decoder for hidden Markov model (HMM) based channels, and then take it a step further by also characterizing the GEC parameters to greatly simplify density evolution for the GEC. This is then used to analyze the performance limits of a regular-(3,6) LDPC code. In [7], the authors apply density evolution to the GEC to devise a number of irregular codes for a specific set of GEC parameters which outperform the standard regular LDPC codes. Even with the simplification provided by this characterization of the GEC paramters, the analysis done in [6, 8] is only performed in two dimentions and in [7], only one dimention. It would still be very difficult to do a

complete analysis over the entire parameter space using DE.

In [1], the authors propose a binary symmetric channel with memory based on Polya's urn scheme for modeling contagions. They describe both infinite memory and finite memory versions of the channel. The finite memory contagion model (FMCC) is an M^{th}- order additive Markov model in which the channel noise sequence is itself a Markov chain. The probability of channel error is a function of the last M channel noise symbols. The FMCC is designed to be a simple model of fading or burst error noise which is easy to analyze. The FMCC has closed form solutions for both the capacity and the block transition distributions which is what defines any channel with memory.

Finite-state Markov models like the GEC are based on an underlying or 'hidden' Markov process to describe state transitions and for each state there is an associated probability of error. While finite-state Markov models like the GEC are simple to describe and implement, they do not have closed form solutions for capacity or their block transition distributions. As a result, they can be difficult to analyze.

Decoding of LDPC codes over the FMCC for the simple case of $M = 1$ was explored in [20]. The authors developed a joint estimator-decoder for the FMCC similar to that used for the GEC in [6]. They also develop a two-dimentional characterization of the parameters of the FMCC for $M = 1$ so that they can apply DE.

The FMCC model was further generalized in [35] as a queue-based channel (QBC), where the state of the M^{th}-order Markov process is described by a length M queue containing the last M channel noise symbols. This is a four parameter model which adds an additional degree of freedom to the FMCC but retains closed form solutions for capacity and the block transition distribution.

Both the FMCC and the QBC are finite-state additive Markov noise channels where the state transition process is not a 'hidden' variable of the noise process. The state transition

is instead determined exactly by the noise process. We refer to these types of channels as binary additive Markov noise channels (BAMNCs) and describe this class in detail later on.

Markov process based channel models are useful models both because they are simple to understand and implement and because they have proven to be good models of burst errors and fading phenomena. In [23, 28, 33] hidden Markov models (including the GEC) are used to model discretized versions of both Rician and Rayleigh correlated fading channels. The BAMNC model is also used to model the same channels in [23] with excellent results (in terms of matching of capacity and the autocorrelation function for the channels) for fast and medium fading channels. The authors of [37] further demonstrate that the QBC model is in fact a better model for Rician slow fading phenomena than the GEC and much easier to analyze and implement than the BAMNC for the long memory lengths required to model slow fading since it only requires 4 parameters to describe regardless of the memory length.

In this work, we focus on the design of a joint channel estimator-decoder for the QBC and the more general family of BAMNCs. This gives us the ability to simulate the experimental performance of LDPC codes over the QBC and compare the QBC model to the GEC and the general class of BAMNCs. It also provides the framework for the design of QBC or BAMNC based decoders for Rayleigh and Rican fading channels or other appropriate channel models.

1.3 Overview of this Work

We begin in Chapter 2 with a background review of information theoretic concepts used in this work. We review Shannon's noisy channel coding theorem and the Shannon limit

used for performance comparison in our simulations. We define noisy channels both with and without memory and give the specific definitions of the types of channels used in this thesis. These include the memoryless BSC, BAMNCs like the QBC and binary symmetric channels based on HMMs like the GEC. Each of these channel models is described in detail and where possible expressions for capacity, the steady state and block transition distributions are given. We also describe a method of modeling general channel statistics using an M^{th}-order additive Markov noise model to match the M^{th}-order channel noise statistics of another channel.

Chapter 3 covers the topic of LDPC codes by covering the relevant background material needed to understand LDPC codes and sum-product algorithm based decoding. Beginning with linear block codes and their parity-check matrix and Tanner graph expressions, LDPC codes, both regular and irregular, are defined and discussed to give an understanding of their construction and the method of encoding used. The SPA is first discussed as a general belief propagation algorithm, then more specifically as a decoder for memoryless channels and then as a joint estimator-decoder for finite-state Markov modelled channels like those described in Chapter 2.

With the necessary background covered we move on to the coding system design in Chapter 4. We begin with the encoder design for both regular and irregular LDPC codes, then the SPA decoder design and finally the design of the channel model simulations. The SPA decoder is again broken down into two parts. First is the message passing algorithm on the Tanner graph. This part can be taken on its own and used for decoding over memoryless channels. The second part is the message passing algorithm on the Markov chain graph which can also be described as a forward-backward algorithm (much like the BCJR algorithm). Both parts are then connected bit-wise to produce the joint-estimation decoder design. The chapter concludes with a step-by-step breakdown of the coding system and

finally a brief discussion of algorithm parallelization for the SPA based joint estimator-decoder.

In Chapter 5 results from the numerous simulations are given. Simulation results involving the QBC, GEC and BSC with both regular and irregular codes are shown. Additionally examples of QBC and BAMNC modeling of the GEC are demonstrated.

Finally, we conclude in Chapter 6 by discussing the significance of the results. We also describe several possible avenues of future research indicated by this work.

Chapter 2

Information Theory and Channel Modeling

This chapter is a brief review of the key concepts of Shannon's channel coding theorem as well as a detailed overview of the channel models we present in this work. All of coding theory, including that of LDPC codes, is based on the channel coding theorem. It provides us with the achievable limits of communications allowing us to measure how close practical communications schemes come to this ultimate limit. The channels we present in this work attempt to accurately model real world communications channels in order to allow the design of coding schemes that can approach the capacity of these channels. These channels model the burstiness of typical real world channels which generally makes them better models than the corresponding memoryless channel model that is commonly used in communication system design.

2.1 Information Theory and Channel Coding

We begin our discussion of information theory and channel modeling with the basics of information theory as laid out by Claude E. Shannon in [29]. After giving the necessary definitions of information and channel capacity, we state Shannon's channel coding theorem and then derive general form channel capacity equations for the types of channels we consider in this work. Finally, we define the rate-distortion Shannon limit, which is used to compare the performance of our simulations to these ultimate limits defined by Shannon.

2.1.1 Information Measures

To begin any discussion of Shannon's information theory we must start with the definition of the basic measure of information. Dubbed *entropy* by Shannon, it is a measure of the uncertainty in a random variable. Entropy tells us that the more uncertainty there is about a random variable, the more *information* it carries with it. Conversely, this can be expressed as, the more certain we are about the result of a random event, the less information we obtain by knowing the result.

We are concerned with discrete data sources and discrete channels defined over discrete alphabets, so our definitions will be based on this model of information and communications. A discrete alphabet $\mathcal{X}$ is a countable set of elements or *symbols*. from which we can draw a value $x \in \mathcal{X}$ for the random variable X. The simplest and most common example of a discrete alphabet is the binary alphabet $\mathcal{X} = \{0, 1\}$. Another very common discrete alphabet is the Latin alphabet $\mathcal{X} = \{a, b, c, \ldots, z\}$, which is a source alphabet for many Western written languages.

We describe a random variable X over a discrete alphabet $\mathcal{X}$ by it's probability distribution $p(x) = \Pr(X = x)$, which is the probability that X takes on the value $x \in \mathcal{X}$.

Definition 2.1.1. The *entropy* of a random variable X, which takes values from the discrete alphabet $\mathcal{X}$, with probability distribution $p(x) = \Pr(X = x)$, where $x \in \mathcal{X}$ is given by

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x),$$

where the logarithm is to the base 2. Let us also define the *conditional entropy* of X given Y. Let Y be another random variable over the discrete alphabet $\mathcal{Y}$ and we define the probability distributions $p(y) = \Pr(Y = y)$, $p(x, y) = \Pr(X = x, Y = y)$ and $p(x|y) = \Pr(X = x|Y = x)$. Then the conditional entropy of X given Y is

$$H(X|Y) = -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log p(x|y).$$

While the entropy can be thought of as a measure of the amount of information a random variable contains, the conditional entropy can be thought of as the amount of information carried by X given that we already know Y. It can be shown (and is intutitively clear) that $H(X|Y) \leq H(X)$, with equality *iff* X is independent of Y [5, p.27].

We extend the concept of entropy to sequences of random variables, or random processes where $\mathbf{X}^n = (X_1, X_2, \ldots, X_n)$ is a length n sequence of random variables. Let $\mathbf{X} = \{X_i\}_{i=1}^{\infty}$ be a random process, then we would like to know the *entropy rate* or the average entropy per symbol. The entropy of a sequence of random variables is given by

$$H(\mathbf{X}^n) = -\sum_{\mathbf{x}^n \in \mathcal{X}^n} p(\mathbf{x}^n) \log p(\mathbf{x}^n),$$

where $\mathbf{x}^n$ is a length n sequence of values and $p(\mathbf{x}^n)$ is the joint probability of that sequence of values.

Definition 2.1.2. The *entropy rate* of the random process $\mathbf{X} = \{X_i\}_{i=1}^{\infty}$ is the average

entropy per variable (or per symbol) given by

$$\begin{aligned} H(\mathbf{X}) &= \lim_{n\to\infty} \frac{1}{n} H(X_1, X_2, \ldots, X_n) \\ &= -\lim_{n\to\infty} \frac{1}{n} \sum_{x_1,\ldots,x_n} \Pr(X_1 = x_1, \ldots, X_n = x_n) \log \Pr(X_1 = x_1, \ldots, X_n = x_n). \end{aligned}$$

This limit may not exist for all random processes, however it does exist for processes which are stationary and for a stationary process $\mathbf{X}$, $H(\mathbf{X}) = H'(\mathbf{X})$, where $H'(\mathbf{X})$ is defined as

$$H'(\mathbf{X}) = \lim_{n\to\infty} H(X_n | X_1, X_2, \ldots, X_{n-1}),$$

and $H(X_n|X_1, X_2, \ldots, X_{n-1})$ is the conditional entropy of the last symbol in the sequence given the preceeding symbols.

The final quantity we need to define is *mutual information.* This is the amount of information that can be known about one random variable by observing another. It can be thought of as the amount of information shared by two random variables.

Definition 2.1.3. The *mutual information* between random variables X and Y defined over the alphabets $\mathcal{X}$ and $\mathcal{Y}$, respectively, is defined as

$$\begin{aligned} I(X;Y) &= \sum\sum p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \\ &= H(X) - H(X|Y) \\ &= H(Y) - H(Y|X) \\ &= I(Y;X). \end{aligned}$$

Entropy, conditional entropy, mutual information and the entropy rate provide the basic definitions needed to discuss information theory. They are the measures of information

used to describe the limits of communications which we define in the next section.

2.1.2 Shannon's Channel Coding Theorem

The communication system shown in Fig. 1.1 is the basis for Shannon's channel coding theorem. We assume for this work that both the source and channel are *discrete*. In the following definitions we use the superscript n to denote a sequence of variables of length n.

Definition 2.1.4. A *discrete channel* is a process defined by the sequence of block transition probabilities $\{\Pr(\mathbf{Y}^n = \mathbf{y}^n | \mathbf{X}^n = \mathbf{x}^n\}_{n=1}^{\infty}$ where $\mathbf{x}^n \in \mathcal{X}^n$ and $\mathbf{y}^n \in \mathcal{Y}^n$ and $\mathcal{X}$ and $\mathcal{Y}$ are discrete alphabets. The block transition probability $\Pr(\mathbf{Y}^n = \mathbf{y}^n | \mathbf{X}^n = \mathbf{x}^n)$ is the probability that we receive the sequence $\mathbf{y}^n$ from the output of the channel given that the encoder sends the sequence $\mathbf{x}^n$.

The capacity of a discrete channel can be defined using the mutual information bwtween the input and output sequences of the channel.

Definition 2.1.5. We define the *capacity* of the channel as the maximization of the mutual information rate between the input and the output of the channel over the distribution of the input. Essentially, this is the maximum amount of information about the source data that can be reliably conveyed across the channel and this is given by

$$\begin{aligned} C &= \lim_{n\to\infty} \max_{p(\mathbf{x}^n)} \frac{1}{n} I(\mathbf{X}^n; \mathbf{Y}^n) \\ &= \lim_{n\to\infty} \max_{p(\mathbf{x}^n)} (\frac{1}{n} H(X_1, X_2, \ldots, X_n) - \frac{1}{n} H(X_1, X_2, \ldots, X_n | Y_1, Y_2, \ldots, Y_n)), \end{aligned}$$

where the maximization is over all possible input distributions for $\mathbf{x}^n$.

We note that this definition is not always useful as this limit may not exist for certain

channels. It is also very difficult to compute in closed form for most channels in general unless the channel model used allows for a significant simplification of this definition.

The encoder and decoder in Fig. 1.1 together comprise a channel code. A channel code is defined by its encoding function $f(\cdot)$, and its decoding function $g(\cdot)$.

Definition 2.1.6. An (M, n) channel code maps the input messages from the set $\{1, \ldots, M\}$ to the discrete alphabet $\mathcal{X}^n$ through the encoding function

$$f: \ \{1, \ldots, M\} \rightarrow \ \mathcal{X}^n,$$

and maps messages from the discrete alphabet $\mathcal{Y}^n$ to $\{1, \ldots, M\}$ through the decoding function

$$g: \ \mathcal{Y}^n \rightarrow \ \{1, \ldots, M\}.$$

The encoder maps the set of input messages to the set $\mathcal{C} = \{f(1), \ldots, f(M)\}$, which is the *codebook* of the code. The decoder attempts to make the best guess at what message $W \in \{1, \ldots, M\}$ was sent given that it received Y^n and assigns the value $\hat{W}$ as its estimate of W.

The *rate* of an (M, n) code is given by $R = \log(M)/n$ (bits/channel use). The probability of decoder error for a code is the probability that $\hat{W} \neq W$ (i.e. that the decoder chooses the wrong message). The conditional probability of error given that $W = i$ is

$$P_e^i \ = \ \Pr(g(Y^n) \neq i | X^n = f(i)),$$

for each message $i \in \{1, \ldots, M\}$. We define the maximum probability of error as

$$P_e^{(n)} = \max_{i \in \{1,\ldots,M\}} P_e^i.$$

Definition 2.1.7. For a given channel, a rate R is said to be *achievable* for that channel, if there is a sequence of $(\lceil 2^{nR} \rceil, n)$ codes such that $P_e^{(n)} \to 0$ as $n \to \infty$. We define the ceiling operation $\lceil \cdot \rceil$ as rounding up to the next integer value.

Shannon's channel coding theorem proves that the maximum *achievable* rate at which data can be transmitted on a channel is the capacity of the channel. We are now ready to give Shannon's channel coding theorem, which holds for information stable channels [1] for C as defined in Def. 2.1.5.

Theorem 2.1.8. *For every rate $R < C$, there exists a sequence of $(\lceil 2^{nR} \rceil, n)$ codes such that the maximum probability of error $P_e^{(n)} \to 0$ as $n \to \infty$. In other words, every rate $R < C$ is achievable according to the definition above.*

Conversely, for any sequence of $(\lceil 2^{nR} \rceil, n)$ which have the property that $P_e^{(n)} \to 0$ as $n \to \infty$, then the code must have rate $R \leq C$.

See [29] or [5, pp.199] for the proof of this theorem for the case of discrete memoryless channels.

Shannon's channel coding theorem defines the optimal limits of communications systems. No system may exceed the capacity of the channel without suffering a probability of error which is bounded away from zero, though it is still possible to consider the limits of systems which do not have arbitrarily small probability of error. This is done through the use of Shannon's rate-distortion theorem and the Shannon limit described in Section 2.1.4.

2.1.3 Capacity of Additive Binary Symmetric Channels

We now show how to derive the capacity for the channel models which are considered in this thesis. These are all channels where both the input and output alphabets are binary (i.e. $\mathcal{X} = \mathcal{Y} = \{0, 1\}$). Furthermore, all of the channels we consider are *additive* binary noise channels and the noise process is *symmetric*.

An additive noise channel is one in which the channel noise is determined by a random sequence $\mathbf{E}^n$ which is additively combined with the input sequence such that $\mathbf{Y}^n = \mathbf{X}^n + \mathbf{E}^n$. Furthermore we assume, $\mathbf{E}^n$ is independent of $\mathbf{X}^n$ so that

$$\begin{aligned} \Pr(\mathbf{Y}^n = \mathbf{y}^n | \mathbf{X}^n = \mathbf{x}^n) &= \Pr(\mathbf{X}^n + \mathbf{E}^n = \mathbf{y}^n | \mathbf{X}^n = \mathbf{x}^n) \\ &= \Pr(\mathbf{E}^n = \mathbf{y}^n - \mathbf{x}^n | \mathbf{X}^n = \mathbf{x}^n) \\ &= \Pr(\mathbf{E}^n = \mathbf{y}^n - \mathbf{x}^n). \end{aligned}$$

For binary input-binary output channels $\mathbf{E}^n \in \{0, 1\}^n$ and addition is taken bitwise modulo-2. We represent modulo-2 addition with the symbol $\oplus$, so we write $\mathbf{Y}^n = \mathbf{X}^n \oplus \mathbf{E}^n$ for binary additive noise channels.

The term symmetric refers to the relationship between the two possible channel outputs and the two possible channel inputs. Simply put, for any single bit, the probability of error is the same for both inputs (see Fig. 1.2).

$$\begin{aligned} \Pr(Y_i = 1 | X_i = 0) &= \Pr(Y_i = 0 | X_i = 1) \\ &= \Pr(E_i = 1), \\ \Pr(Y_i = 1 | X_i = 1) &= \Pr(Y_i = 0 | X_i = 0) \\ &= \Pr(E_i = 0) \\ &= 1 - \Pr(E_i = 1). \end{aligned}$$

The simplest example of this kind of channel is the memoryless binary symmetric channel (BSC) in which the noise samples E_i are independent and the probability of error is fixed, $\Pr(E_i = 1) = p$, where $p \leq 0.5$. For this channel the capacity equation is much simpler and we get

$$\begin{aligned} C &= \max_{p(x)} I(X;Y) \\ &= \max_{p(x)} H(Y) - H(Y|X) \\ &= \max_{p(x)} H(Y) - \sum p(x)H(E) \\ &= 1 - H(E) = 1 + p\log p + (1-p)\log(1-p). \end{aligned}$$

The distribution of X which maximizes C is clearly the uniform distribution for a binary random variable. We note that if X is uniform then Y must be uniform as well, due to the symmetry of the channel and thus $H(Y) = 1$, which is the maximum entropy a binary random variable can have.

This work is primarily concerned with two types of channels with memory. One type has a simple and exact expression for the channel capacity, while the other type only allows for upper and lower bounds. The first is the binary symmetric additive Markov noise channel (BAMNC) which is the main focus of this work and the queue-based channel (QBC) is an example of this type of channel. The second type is the class of Markov modulated binary symmetric channels based on hidden Markov models (HMMs). As we stated earlier in all cases when we are referring to BAMNCs or HMM based channels, we are referring to additive binary symmetric channels. The Gilbert-Elliot channel (GEC) is an example of this type of channel.

In all cases, the additive channel noise process $\mathbf{E}$ is independent of the channel input process $\mathbf{X}$ and since the channel is symmetric the capacity is maximized by input blocks

$\mathbf{X}^N$ which are uniformly distributed. So we assume a uniform input distribution in each case. These channels are also stationary so we have that $H(\mathbf{E}) = H'(\mathbf{E})$; thus, the equation for capacity simplifies somewhat to become

$$\begin{aligned} C &= \lim_{n\to\infty} \max_{p(x^n)} \frac{1}{n} I(\mathbf{X}^n; \mathbf{Y}^n) \\ &= 1 - \lim_{n\to\infty} \frac{1}{n} H(E_1, E_2, \ldots, E_n) \\ &= 1 - \lim_{n\to\infty} H(E_n | E_1, E_2, \ldots, E_{n-1}). \end{aligned}$$

For a BAMNC the noise process $\mathbf{E}$ is an M^{th}-order stationary Markov process. This means that

$$\begin{aligned} &\Pr(E_n = e_n | E_{n-1} = e_{n-1}, \ldots, E_1 = e_1) \\ &\qquad = \Pr(E_n = e_n | E_{n-1} = e_{n-1}, \ldots, E_{n-M} = e_{n-M}) \\ &\qquad = \Pr(E_{M+1} = e_{M+1} | E_1 = e_1, \ldots, E_M = e_M), \end{aligned}$$

In other words, the probability of the next symbol given the entire past is equal to the probability of the next symbol given only the past M symbols. This means that the capacity simplifies further and we can compute the limit as

$$\begin{aligned} C &= \lim_{n\to\infty} \max_{p(x^n)} \frac{1}{n} I(\mathbf{X}^n; \mathbf{Y}^n) \\ &= 1 - \lim_{n\to\infty} H(E_n | E_1, E_2, \ldots, E_{n-1}) \\ &= 1 - H(E_{M+1} | E_1, E_2, \ldots, E_M) \\ &= 1 - \sum_{\mathbf{e}^M} p(\mathbf{e}^M) \log p(e_M | \mathbf{e}^{M-1}), \end{aligned}$$

where $\mathbf{e}^M = (e_1, \ldots, e_M)$.

An M^{th}-order binary Markov process can also be thought of as a first-order Markov process with 2^M states. Each state is a representation of a unique sequence of values for $\mathbf{e}^M = (e_1, \ldots, e_M)$. which can take on 2^M possible values since each variable is a binary number. The state at time t is a random variable we will denote by S_t. Each of these states has associated with it a steady-state probability, which is the average probability we are in that state and it is denoted π_i, the steady-state probability of being in state i, where $i \in \{0, \ldots, 2^{M-1}\}$. Any two states i and j also have a probability of state transition associated with them, denoted $P_{ij} = \Pr(S_{t+1} = j | S_t = i)$, which is the probability of going from state i at time t to state j at time $(t+1)$.

We note that the state S_t is simply another way of representing the length M sequence $(E_{t-1}, E_{t-2}, \ldots, E_{t-M})$, and thus

$$\begin{aligned} C &= \lim_{n\to\infty} \max_{p(x^n)} \frac{1}{n} I(X^n; Y^n) \\ &= 1 - H(E_M | E_0, E, \ldots, E_{M-1}) \\ &= 1 - H(S_{t+1} | S_t) \\ &= 1 - \sum_{i,j} \pi_i P_{ij} \log P_{ij}. \end{aligned} \tag{2.1}$$

So for BAMNCs, where we have closed form equations for the steady-state and block transition distributions, we can compute a closed form expression for the capacity of the channel. This makes models like the QBC, for which this is true, very attractive for information theoretic analysis purposes.

Markov modulated channels based on HMMs are not as simple. A hidden Markov model is one in which the Markov process is an underlying or hidden process of some observed random variables that are related to the hidden Markov process through a 'random function' of the state of the Markov process. The capacity of channels based on HMMs

can usually only be upper and lower bounded in a closed form for any finite value of n.

We define a HMM by a 'hidden' state sequence $S_1, \ldots, S_n$, where the variables s_t are defined over the set $\mathcal{S}$, and a set of probability distribution functions (PDF) $\phi_{s_t}(\cdot)$. There is a PDF associated with each state in $\mathcal{S}$. Then we define the sequence $E_1, \ldots, E_n$ such that $\Pr(E_t = e_t) = \phi_{s_t}(e_t)$ which is the 'visible' sequence of the HMM.

For channels based on HMMs, $\mathbf{E}$ is the additive noise process of the channel. We wish to know the entropy rate $H(\mathbf{E})$ of this process so that we may evaluate the capacity. We use the following theorem to obtain tight upper and lower bounds for the entropy rate of hidden Markov models.

Theorem 2.1.9. *If $S_1, \ldots, S_n$ is a stationary Markov chain and $\Pr(E_t = e_t) = \phi_{s_t}(e_t)$, then*

$$H(E_n|E_{n-1}, \ldots, E_1, S_1) \leq \quad H(\mathbf{E}) \quad \leq H(E_n|E_{n-1}, \ldots, E_1)$$

and

$$\lim_{n\to\infty} H(E_n|E_{n-1}, \ldots, E_1, S_1) = \quad H(\mathbf{E}) \quad = \lim_{n\to\infty} H(E_n|E_{n-1}, \ldots, E_1).$$

Proof. The proof of this can be found in [5, p. 71]. □

Using these bounds we can place upper and lower bounds on the capacity of HMM based channel models

$$\begin{aligned} 1 - H(E_n|E_{n-1}, \ldots, E_1) \leq \quad C \quad &\leq 1 - H(E_n|E_{n-1}, \ldots, E_1, S_1) \\ 1 - \lim_{n\to\infty} H(E_n|E_{n-1}, \ldots, E_1) = \quad C \quad &= 1 - \lim_{n\to\infty} H(E_n|E_{n-1}, \ldots, E_1, S_1). \end{aligned} \tag{2.2}$$

Either of these bounds can be used with a sufficiently large value of n to approximate the

capacity of an HMM based channel such as the GEC.

Each of these channels represents a significant simplification of real world channels where channel noise is dependent on many complex factors. Models like these allow for practical simulation and analysis by simplifying the computations needed to both simulate the channel model and to compute its statistical properties and limits. It is these simple models that allow us to design practical decoders. In particular, models with memory like the GEC and QBC allow us to design decoders that can exploit the property that errors occur in bursts in many real world channels.

2.1.4 The Rate-Distortion Shannon Limit

In Shannon's original work [29], he defines the capacity of the channel as the highest rate at which data can be sent such that an arbitrarily low probability of error can be achieved. In practice we wish to analyze a fixed rate code and would like to know what is the worst channel on which this code can achieve an arbitrarily low probability of error. In other words, for a code of rate r we want to find the channel parameters for which the capacity of the channel $C = r$. This is often referred to as the Shannon limit for that code.

To find the Shannon limit of a rate 1/2 code over the BSC we need to find the value of p, the probability of error for the channel, which gives a channel which has $C = 1/2$. This turns out to be at $p = 0.11$; thus, the Shannon limit of a rate 1/2 code of the BSC is $p = 0.11$.

In most practical applications where we can accept a certain level of error we need not require an arbitrarily low error rate. In this case, the question becomes, "for a given code of rate r, and an acceptable probability of decoding error P_e, what is the worst channel for which we can achieve P_e with this code?"

Shannon also provides us with a tool for considering the rates at which a certain level of error or distortion can be achieved called *rate-distortion theory* [5, 13]. Rate-distortion theory is often used in source coding involving lossy compression. For a Bernoulli source, rate-distortion theory states that for an acceptable probability of bit error P_e (under the Hamming distortion measure), we need only send $R(P_e) \leq 1$ bits per bit of data where $R(\cdot)$ is the rate distortion function and is given by

$$\begin{aligned} R(P_e) &= 1 - h_b(P_e), \\ &= 1 - [-P_e \log_2(P_e) - (1 - P_e) \log_2(P_e)], \end{aligned}$$

where $0 \leq P_e \leq 1/2$. This means that only $R(P_e)$ bits are needed to describe each bit of data and still be able to recover the original source with a probability of bit error as low as P_e.

Suppose we were to apply lossy compression to our source data before encoding it. If we used ideal lossy compression on the source data then the rate-distortion function tells us that we can achieve a probability of bit error as low as P_e by encoding only $R(P_e) \leq 1$ bits per bit of data. If the channel encoder then encodes this data at a rate of $r < C$ then we can achieve an arbitrarily low probability of error with respect to the compressed bits and we can still achieve a probability of error as low as $P_e$with respect to the original source data.

Relative to the original source data we have encoded the source data at an overall rate of $r \cdot R(P_e)$ and we can achieve a probability of error as low as P_e. Another way to look at this is to say that if we send data at a rate $r < C/R(P_e)$ then Shannon guarantees that we can achieve a probability of error as low as P_e.

We can now reword this result in terms of the Shannon limit to find the worst channel

for which a code of rate r can achieve a probability of bit error as low as P_e. This is referred to as the rate-distortion Shannon limit (RDSL).

The RDSL is generally evaluated numerically since this is simpler than evaluating the inverse of the capacity equations. It is also usually evaluated in terms of a single parameter, the BSC has only one parameter p, which is the channel bit-error rate (CBER) and for the QBC the parameter used is also the CBER p (all the other parameters are fixed). For these examples the RDSL is computed as

$$S_{RD}(r, P_e) \quad = \quad p^*,$$

where $p*$ satisfies

$$C(p^*) \quad = \quad r(1 - h_b(P_e)) \qquad 0 < p^* < 0.5.$$

In examining the performance of the simulations presented in Chapter 5, we use the RDSL to compare the performance of actual codes over simulated channels at different CBERs to the theoretical limit of performance at the same CBERs for both the BSC and QBC.

2.2 Binary Symmetric Channels with Markov Memory

This section describes, in detail, the channel models used in the design of our LDPC decoder. We are concerned with decoding over channels with memory, which can be modeled using finite-state Markov chains. To this end we present two different families of binary symmetric Markov modeled channels: channels with additive M^{th}-order Markov noise and channels with additive noise based on finite-state HMMs.

The former channel models are finite-memory models where the channel state is a direct function of the past M channel noise outputs. By contrast, HMMs have effectively infinite memory (in terms of having no exact finite-length block transition description) and the channel state is a 'hidden' variable of the channel noise outputs; thus, the state cannot be determined by direct observation of these outputs.

2.2.1 Binary Additive (M^{th}-order) Markov Noise Channels

We define here a general class of binary symmetric channels with M^{th}-order Markov noise and we will refer to them as binary additive Markov noise channels (BAMNCs).

Definition 2.2.1. Let $\mathbf{X}^n$ be a random input sequence of length n and let $\mathbf{Y}^n$ be the output of a BAMNC. Then $\mathbf{Y}^n = \mathbf{X}^n \oplus \mathbf{E}^n$, where $\oplus$ represents block-wise addition modulo-2 and $\mathbf{E}^n$ is the length n additive noise sequence of the BAMNC. Now we let $E_t \in \{0, 1\}$ be a random variable representing the single binary noise symbol at time t and we define $S_t = (E_{t-1}, E_{t-2}, \ldots, E_{t-M})$ as the channel state at time t. We then require that

$$\begin{aligned}\Pr(E_t = e_t | E_{t-1} = e_{t-1}, \ldots, E_1 = e_1) & \\ &= \Pr(E_t = e_t | E_{t-1} = e_{t-1}, \ldots, E_{t-M} = e_{t-M}) \\ &= \Pr(E_t = e_t | S_t = s_t),\end{aligned}$$

and thus for all $s_i \in \{0, 1\}^M$,

$$\Pr(S_{t+1} = s_{t+1} | S_t = s_t, S_{t-1} = s_{t-1}, \ldots, S_1 = s_1) = \Pr(S_{t+1} = s_{t+1} | S_t = s_t).$$

In other words, we require that the noise process of the BAMNC be the M^{th}-order Markovian and thus the channel-state process form a Markov chain.

This channel model is fully characterized by its state transition probability since each state transition corresponds to a particular channel noise output value. Equivalently, this implies that there are only two possible state transitions into or out of any state. In order to fully describe an M^{th}-order binary additive Markov channel we require up to 2^M parameters, one for each state. We can enumerate the states using the set of integers $\{0, \ldots, 2^M - 1\}$, where the state corresponding to the sequence $(e_{t-1}, ..., e_{t-M})$ is given by

$$i \;=\; e_{t-1}2^{M-1} + e_{t-2}2^{M-2} + \cdots + e_{t-i}2^{M-i} + \cdots + e_{t-M}.$$

For each state, we define the probability $P_i^e = \Pr(E_t = 1 | S_t = i)$, which is both the probability of error for that state as well as the probability of state transition from $i \rightarrow \left\lfloor \frac{i+2^M}{2} \right\rfloor$. The probability of no error for state i is simply $(1 - P_i^e)$, which is also the state transition probability from $i \rightarrow \left\lfloor \frac{i}{2} \right\rfloor$. We define the operation $\lfloor \cdot \rfloor$ as rounding down the the nearest integer.

The state transition probability matrix for this channel is sparse, since there are only two non-zero values in each row: P_i^e and $(1 - P_i^e)$. Because channel errors and state transitions directly determine each other, we need no more than one parameter per state to describe any BAMNC. Thus this model is simpler than other, more general, finite-state Markov models.

In [35], the authors propose using a queue of size M to describe the channel-state. One can think of channel state transitions as shifting the values of the queue to the right by one and placing the most recent channel noise output in the first queue position (the most significant bit). This leads to the description of an even simpler subset of this family, the queue-based channel which is a generalization of the finite memory Polya contagion channel from [1].

For real world channels it is possible to try to develop binary additive M^{th}-order Markov

approximations. M^{th}-order Markov processes are often used in source coding to model real-world random sources [13, 5] and could be applied using the BAMNC model to develop decoders or based on M^{th}-order approximations to actual channels. We will show how this can be done in Section 2.2.4

2.2.2 Binary Channels Based on Hidden Markov Models

The GEC is one of the most popular finite-state Markov channel models in the literature. This is due in part to its early adoption in the history of communication research but mostly due to its simplicity as a model.

It is a two-state HMM, because the channel state is a hidden variable of the observations of channel noise. The channel-state sequence cannot be directly inferred from observing the channel noise sequence. This is different from the BAMNC where the knowledge of the past channel noise sequence allows us to determine exactly what the channel-state sequence was.

Binary symmetric HMM based channels, as we define them here, are Markov modulated channels where channel state transitions occur according to a Markov process in the same way as state transitions for the BAMNC model occur. The difference is that for HMMs, each channel-state has an associated memoryless BSC which generates the channel noise when the channel is in that state. For each state, we need to define the state-transition probabilities as well as the probability of error for the BSC associated with that state. We can see this relationship between the channel state and it's associated BSC in Fig. 2.2 for the simple GEC model.

As mentioned in Section 2.2.1, for the M^{th}-order BAMNC, channel-state transitions are a function of the noise symbol. As a result we only need one parameter per state. The two-state (first-order) BAMNC needs only two parameters to define the channel, while

the two-state GEC requires four parameters. In general an n-state binary HMM channel requires up to n^2 parameters, $n \times (n-1)$ for state transition probabilities and n parameters for the channel bit-error rates for each state. The n-state M^{th}-order BAMNC ($n = 2^M$) requires only up to n parameters.

Finite-state HMM based binary symmetric channels do not have an exact finite-length channel block transition probability description (although an approximation is possible). This means that HMMs do not have a finite-order Markov distribution for the noise process even though the underlying state transition model is first-order Markov. As a result of this HMMs are harder to analyse since channel capacity can only be upper and lower bounded and the noise process statistics must be approximated. Having a finite-order Markov distribution of the noise process is the defining characteristic of the BAMNC model and we note that it is possible to approximate the noise process of an HMM with a BAMNC of finite-order by using an M^{th}-order approximation of the channel noise process. This is described in Section 2.2.4.

2.2.3 Specific Implementations

There are two channel models in the literature based on an additive Markov noise model that we will consider in this work. The finite memory contagion channel (FMCC) and the queue-based channel (QBC). The FMCC [1] is based on Polya's model for the spread of a contagion in a population. This model was generalized to the QBC in [35] which is the channel we focus on in our simulations. The GEC is the example used for a hidden Markov model since it is a very popular example of this type of channel in the literature. This channel is also used in our simulations and is compared with the QBC.

Finite Memory Contagion Channel

Alajaji and Fuja presented a novel description of a channel with binary additive Markov noise in [1], where each error affects the probability of future errors based on Polya's model for the spread of a contagion. It models how testing a single person for an infection affects the probability of finding infected people in future tests. Each positive test increases the probability of a future positive test and a negative test decreases the probability of future positive tests.

In their paper, they develop a channel model where the channel noise sequence is chosen according to the urn scheme used by Polya for modeling contagions [24]. It proceeds as follows: the urn initially contains T balls with R red balls and S black balls ($T = R + S$). At each turn, a ball is drawn and we examine the color. If it is red then an error occurs and if it is black no error occurs. Afterwards we place $1 + \Delta$ balls of the same color in the urn ($\Delta > 0$).

The authors note that this channel has infinite memory and the effect of memory does not diminish over time. The effect of the first ball drawn on the probability of the next ball is the same as the effect of the last ball. They also determine that the capacity of this channel is zero. As a result they propose a finite memory contagion channel, where the model has an M^{th}-order Markov noise process and any balls added to the urn more than M draws in the past are removed from the urn.

The FMCC scheme is defined as follows: an urn contains T balls with R red balls and S black balls. As before, we draw from the urn and an error occurs if and only if we draw a red ball. Then we return $1 + \Delta$ ($\Delta > 0$) balls of the same color as before. After M more draws from the urn, we retrieve Δ balls of that color from the urn. This ensures that any draw will only affect the probabilities of the next M draws and no more. This channel is described by three parameters M, $\rho = R/T$ and $\delta = \Delta/T$. We note that ρ is the stationary

probability of error for the channel. Recalling our definition of S_t as the last M channel noise outputs, the probability distribution of E_t given S_t is:

$$\begin{aligned} \Pr(E_t = 1 | S_t = s_t) &= \frac{R + (e_{t-1} + e_{t-2} + \cdots + e_{t-M})\Delta}{T + M\Delta} \\ &= \frac{\rho + (e_{t-1} + e_{t-2} + \cdots + e_{t-M})\delta}{1 + M\delta}, \end{aligned} \tag{2.3}$$

where $s_t = (e_{t-1}, \ldots, e_{t-M})$.

The FMCC's additive noise process is stationary ergodic M^{th}-order additive Markovian with closed form equations for its steady-state distribution and the entropy rate. Like all additive Markov noise processes we can define the state as a value, $i \in \{0, \ldots, 2^M - 1\}$, where $i = e_{t-1}2^{M-1} + e_{t-2}2^{M-2} + \cdots + e_{t-i}2^{M-i} + \cdots + e_{t-M}$. Using this definition, the one-step state transition probabilities for the channel are given by

$$P_{ij} = \begin{cases} \frac{1-\rho+(M-w(i))\delta}{1-M\delta}, & \text{if } j = 2i \ (mod\ 2^M), \\ \frac{\rho+w(i)\delta}{1+M\delta}, & \text{if } j = (2i+1) \ (mod\ 2^M), \\ 0, & \text{otherwise}, \end{cases} \tag{2.4}$$

where $w(i)$ is the weight (number of 1's) of the binary representation of i. The steady-state distribution of the channel state is given by

$$\pi_i = \frac{\prod_{j=0}^{w(i)-1}(\rho + j\delta) \prod_{k=0}^{M-1-w(i)}((1-\rho) + k\delta)}{\prod_{l=1}^{M-1}(1 + l\delta)}, \tag{2.5}$$

where π_i is the steady-state probability that the Markov process of the channel is in state i.

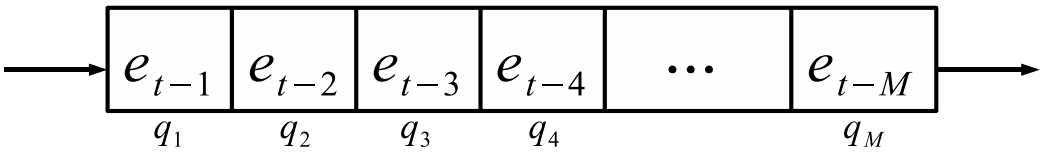

Figure 2.1: Queue diagram for the QBC

Using (2.4) and (2.5) we can compute the channel capacity exactly.

$$C_{FMCC} = 1 - H(E_{M+1}|E_M, E_{M-1}, \ldots, E_1) \tag{2.6}$$

$$= 1 - \sum_{i,j=0}^{2^M-1} \pi_i h_b(p_{ij}) \tag{2.7}$$

$$= 1 - \sum_{k=0}^{M} \binom{M}{k} L_k h_b\left(\frac{\rho + k\delta}{1 + M\delta}\right), \tag{2.8}$$

where

$$L_k = \frac{\prod_{j=0}^{k-1}(\rho + j\delta)\prod_{l=0}^{M-1-k}((1-\rho) + l\delta)}{\prod_{m=1}^{M-1}(1 + m\delta)},$$

and $h_b(\cdot)$ is the binary entropy function.

Queue-Based Channel

In [35] Zhong, Alajaji and Takahara generalize the FMCC by using a finite queue to describe an M^{th}-order additive Markov noise process. The channel is described by four parameters M, p, ε and α.

The noise process is generated according to the following scheme: we have a size M queue which contains the values of the past M channel noise outputs (Fig. 2.1). We choose randomly between two 'parcels' where we choose parcel 1 with probability ε and parcel 2 with probability $1 - \varepsilon$.

Parcel 1 is the length M queue which contains the last M values from the channel output. The next channel noise output is chosen from the queue according to:

$$\Pr(E_t = q_n) = \begin{cases} 1/(M-1+\alpha), & \text{if } n = 1, \ldots, M-1, \\ \alpha/(M-1+\alpha), & \text{if } n = M. \end{cases}$$

Parcel 2 is a memoryless BSC noise process with probability of error p:

$$\Pr(E_t = 1) = p.$$

After each use of the channel, the value, E_t, enters the queue from the left like a shift register and the entries of the queue are shifted to the right by one. E_t becomes the new entry in the first position and the last entry, E_{t-M}, is shifted out of the queue. Just like the FMCC, noise symbols older than E_{t-M}, where t is the current time, have no effect on the next noise symbol.

The contents of the queue fully describe the state of the channel. The probability distribution of the next noise symbol E_t, is given by

$$\Pr(E_t = 1|S_t) = \frac{\varepsilon(e_{t-1} + e_{t-2} + \cdots + \alpha e_{t-M})}{M-1+\alpha} + (1-\varepsilon)p. \tag{2.9}$$

We note that for $\alpha = 1$, the QBC reduces to the FMCC with parameters M, $\rho = p$ and $\delta = \varepsilon/[(1-\varepsilon)M]$. Like the FMCC, the QBC has a stationary ergodic noise process with closed form expressions for the channel noise block distribution and capacity.

The one-step channel state transition probabilities are given by

$$p_{ij}^{(M)} = \begin{cases} \frac{(M-w(i)-1+\alpha)\varepsilon}{M-1+\alpha} + (1-\varepsilon)(1-p), & \text{if } j = \frac{i}{2}, \text{ and } i \text{ is even,} \\ \frac{(M-w(i))\varepsilon}{M-1+\alpha} + (1-\varepsilon)(1-p), & \text{if } j = \left\lfloor \frac{i}{2} \right\rfloor, \text{ and } i \text{ is odd,} \\ \frac{w(i)\varepsilon}{M-1+\alpha} + (1-\varepsilon)p, & \text{if } j = \frac{i+2^M}{2}, \text{ and } i \text{ is even,} \\ \frac{(w(i)-1+\alpha)\varepsilon}{M-1+\alpha} + (1-\varepsilon)p, & \text{if } j = \left\lfloor \frac{i+2^M}{2} \right\rfloor, \text{ and } i \text{ is odd,} \\ 0, & \text{otherwise;} \end{cases} \quad (2.10)$$

where $p_{ij}^{(M)}$ is the M-state channel transition probability from state i to state j and $w(i)$ is the weight of the binary sequence representation of i as before. The steady-state distribution for the channel is

$$\pi_i = \frac{\prod_{j=0}^{w(i)-1}[j\frac{\varepsilon}{M-1+\alpha} + (1-\varepsilon)p] \prod_{k=0}^{M-w(i)-1}[k\frac{\varepsilon}{M-1+\alpha} + (1-\varepsilon)(1-p)]}{\prod_{l=1}^{M-1}(1+(\alpha+l)\frac{\varepsilon}{M-1+\alpha})},$$

where the state $i = 0, 1, \ldots, 2^M - 1$.

Finally, the channel capacity is given by

$$\begin{aligned} C_{QBC} &= 1 - H(E_{M+1}|E_M, E_{M-1}, \ldots, E_1) \\ &= 1 - \sum_{i,j=0}^{2^M-1} \pi_i h_b(p_{ij}) \\ &= 1 - \sum_{k=0}^{M-1} \binom{M-1}{k} L_k h_b\left(\frac{k\varepsilon}{(M-1+\alpha)} + (1-\varepsilon)p\right) \\ &- \sum_{k=1}^{M} \binom{M-1}{k-1} L_k h_b\left(\frac{(k-1+\alpha)\varepsilon}{(M-1+\alpha)} + (1-\varepsilon)p\right), \end{aligned}$$

where

$$L_k = \frac{\prod_{j=0}^{k-1}[j\frac{\varepsilon}{M-1+\alpha} + (1-\varepsilon)p]\prod_{l=0}^{M-k-1}[l\frac{\varepsilon}{M-1+\alpha} + (1-\varepsilon)(1-p)]}{\prod_{m=1}^{M-1}(1+(\alpha+m)\frac{\varepsilon}{M-1+\alpha})},$$

In [35] the authors show that the parametrization of the QBC has certain properties which are useful for analysis of the channel. The first is that the capacity is non-decreasing in α and the second is that for $0 \leq \alpha \leq 1$ capacity is non-decreasing in M.

Theorem 2.2.2. *The capacity C_{QBC} of the QBC increases as the parameter α increases for fixed M, p, and correlation of the channel given by*

$$Cor = \frac{\frac{\varepsilon}{(M-1+\alpha)}}{1-\frac{(M-2+\alpha)\varepsilon}{(M-1+\alpha)}}, \tag{2.11}$$

and the capacity converges to 1 as α approaches infinity for all M, p, and $Cor \neq 0$.

Proof. Given in [35]. □

The correlation of the QBC channel is a measure of the relationship between consecutive channel noise symbols and is defined as

$$Cor \triangleq \frac{E[E_1E_2] - E[E_1]E[E_2]}{E[E_1^2] - E[E_1]^2},$$

where $E[\cdot]$ denotes expectation. For discrete random variables correlation ranges between -1 and 1 and $Cor = 0$ if E_1 and E_2 are independant although the converse is not necessarily true. We note here that it is true for the QBC and $Cor = 0$ implies not only that E_1 and E_2 are independant but that $\varepsilon = 0$ and therefore the channel is a completely memoryless BSC. Furthermore the correlation of the QBC must be between 0 and 1 which is clear from (2.11).

Theorem 2.2.3. *The capacity C_{QBC} of the QBC is non-decreasing in M for fixed p, Cor, and $0 \leq \alpha \leq 1$.*

Proof. Noting that a QBC with fixed Cor, M and $\alpha = 1$, is equivalent to a QBC with $Cor' = Cor$, $M' = M + 1$ and $\alpha' = 0$, we can use Theorem 2.2.2 and write

$$\begin{aligned} C_{QBC}^{M}(\alpha) &\leq C_{QBC}^{M}(1) \\ &= C_{QBC}^{M+1}(0) \\ &\leq C_{QBC}^{(M+1)}(\alpha). \end{aligned}$$

□

Monotonicity of capacity in p, assuming all other parameters are fixed, is clear, since p is the steady-state probability of error for the channel. In general, the capacity will be non-increasing in p. Proving that capacity is monotonic in ε (or equivalently Cor), assuming all other parameters are fixed, is not as simple. Numerical results plotted in [35], for several examples, show that as $Cor \rightarrow 1$ capacity is monotonic and $C_{QBC} \rightarrow 1$. Additionally, for the simplest case of $M = 1$, it is straightforward to show that $\frac{\partial C_{QBC}^{1}}{\partial Cor} \geq 0$ for all fixed values of p. We hypothesize that the capacity is in fact monotonic and non-decreasing in the correlation of the channel and equivalently ε.

Both the QBC and FMCC have a constant number of parameters needed to describe the channel regardless of the value of M. This is not true in general for all additive M^{th}-order Markov noise channels. Thus, these models are simpler to implement and analyze, particularly for large values of M.

Gilbert-Elliot Channel

The GEC [14, 10] is one of the most widely used binary additive noise channels with memory. It is a simple two-state Markov modulated channel based on a hidden Markov model.

Usually, one state is referred to as the 'good' state and the other as the 'bad' state. The good state is associated with a BSC which has a lower probability of error than the BSC which is associated with the bad state. The channel operates by transitioning between the low error 'good' state and the high error 'bad' state, where errors occur more frequently. The bad state models bursts of errors typical of many real world communications systems.

The Markov process is shown in Fig. 2.2 as a simple finite state machine model. There are four parameters needed to describe the GEC: g the probability of transition from the bad state to the good state, b the probability of transition from the good to the bad state, P_g the BSC probability of error in the good state and P_b the BSC probability of error in the bad state. In general we require that $P_g < P_b$.

As discussed in Section 2.1.3 there is no closed form solution for the capacity of the GEC (or any other HMM based channel) its capacity can be estimated numerically using the following equation from [19], which is a lower bound on the capacity for finite values of t

$$C_{GEC} = \lim_{t\to\infty} C_{GEC}^{t} = \lim_{t\to\infty} E[1 - h_b(\Pr(e_{t+1}|\mathbf{e}^t))], \tag{2.12}$$

where $\mathbf{e}^t = (e_t, e_{t-1}, \ldots, e_1)$. We approximate C_{GEC} numerically by computing C_{GEC}^{t} for a sufficiently large value of t.

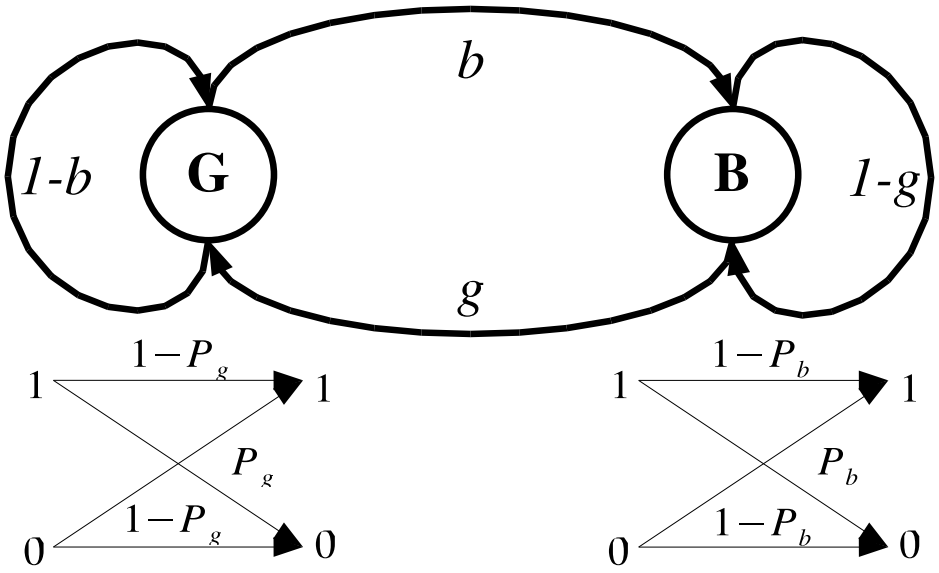

Figure 2.2: Finite state machine model for the GEC

2.2.4 Binary Additive M^{th}-order Markov Approximations

The binary additive M^{th}-order Markov noise channel is a versatile channel model which can be used to approximate the statistics of channels for which no simple model exists. We can design a BAMNC as an M^{th}-order Markov approximation to a real world channel noise process or for a channel model which does not have a finite-order additive Markov process such as the GEC. This can be done by determining the length M block transition probabilities and stationary block distribution for the channel noise.

M^{th}-order Markov approximations of real world random processes are often used to model random sources for data compression. Markov processes can be used as a probabilistic model for text, images, audio and video in order to design better data compression codes for these types of data. English language based text makes for a good example of Markov modeling.

If we want to use a random process to model English text, we could begin by randomly

generating letters and spaces according to the frequency at which they occur in a sufficiently long English book or set of books. Using this model to generate text we would get something like this [29]:

xfoml rxkhrjffjuj zlpwcfwkcyj ffjeyvkcqsghyd qpaamkbzaacibzlhjqd.

While the letters and spaces appear in similar proportion as in English text, this text does not resemble English at all. We can improve the model significantly by using a Markov model. In order to do this, we compute the probability of each letter given the past M letters. Each letter is chosen according to the probability that it follows the previous M letters that came before it. For example, a third order Markov approximation has the probability distribution of the next letter generated as

$$p(x) \quad = \quad \Pr(L_i = x | L_{i-1} = l_{i-1}, L_{i-2} = l_{i-2}, L_{i-3} = l_{i-3}),$$

where L_i and l_i are the random variable and variable, respectively, associated with the i^{th} letter in the sequence. Using this model would produce something like this [29]:

in no ist lat whey cratict froure birs grocid pondenome of demonstures of the reptagin is regoactiona of cre.

This sentence is still gibberish, however its structure more closely resembles English. The words, though nonsense, could probably be pronounced and they contain consonants and vowels in a similar manner to real English words. Additionally, we see the occurrence of a few real English words such as 'the', 'is' and 'of'. We can clearly see that this is a much better random model of English text than the first. However the complexity of the higher-order models increase exponentially with the order of the model.

The same technique can be used to model the statistics of real channels by computing the M^{th}-order Markov statistics for the channel. In order to do this we must first observe sufficient channel noise statistics to compute accurately the length M block stationary distribution and transition probabilities for the channel. In other words, for a binary symmetric channel we need to determine, for each $\mathbf{e}_M = (e_1, e_2, \ldots, e_M)$, where $e_i \in \{0, 1\}$, the finite-length block distributions of the channel noise process

$$\Pr(\mathbf{e}^M) = \Pr(E_1 = e_1, E_2 = e_2, \ldots, E_M = e_M), \tag{2.13}$$

$$\Pr(e_{M+1}|\mathbf{e}^M) = \Pr(E_{M+1} = e_{m+1}|E_1 = e_1, \ldots, E_M = e_m). \tag{2.14}$$

This gives the steady state distribution (2.13) and the channel transition probability (2.14) for the M^{th}-order approximation of the channel. The channel state at time t is represented by $\mathbf{e}^t = (e_{t-1}, e_{t-2}, \ldots, e_{t-M})$ as it is for all BAMNCs.

Using an M^{th}-order approximation allows us to model the statistics of any stationary binary symmetric additive noise channel as closely as complexity will allow. The complexity of the channel model and the decoder increases exponentially with M. Thus, there is a practical limit to the order of the model that can be used for actual decoding.

We can use this method to derive an M^{th}-order approximation of the GEC by computing the probability of each length M bit sequence under the GEC as well as the probability of error in the $M + 1^{st}$ noise symbol given the previous M noise symbols.

First let us define a matrix which contains binary probability distributions

$$\mathbf{P}_{GEC}(e_i) = \begin{bmatrix} P_{E_i,S_{i+1}|S_i}(e_i, G|G) & P_{E_i,S_{i+1}|S_i}(e_i, B|G) \\ P_{E_i,S_{i+1}|S_i}(e_i, B|B) & P_{E_i,S_{i+1}|S_i}(e_i, G|B) \end{bmatrix},$$

where $P_{E_i,S_{i+1}|S_i}(e_i, s_{i+1}|s_i)$ is the joint probability of the state transition to s_{i+1} with noise

symbol e_i given that the current state is s_i. For $e_i \in \{0, 1\}$ we get

$$\begin{aligned}
\mathbf{P}_{GEC}(1) &= \begin{bmatrix} (1-b)P_g & bP_g \\ (1-g)P_b & gP_b \end{bmatrix}, \\
\mathbf{P}_{GEC}(0) &= \begin{bmatrix} (1-b)(1-P_g) & b(1-P_g) \\ (1-g)(1-P_b) & g(1-P_b) \end{bmatrix}.
\end{aligned}$$

This matrix can be used to calculate $P(e_1, \ldots, e_M)$ as well as $P(e_{M+1}|e_1 \ldots, e_M)$ for every sequence $(e_1, \ldots, e_M)$. Let

$$\pi_{GEC} = \begin{bmatrix} \frac{g}{g+b} \\ \frac{b}{g+b} \end{bmatrix},$$

which is the stationary distribution for the state of the GEC. Then we get

$$\begin{aligned}
P(e_1, \ldots, e_M) &= \mathbf{u}^T \mathbf{P}_{GEC}(e_M) \cdot \mathbf{P}_{GEC}(e_{M-1}) \cdots \mathbf{P}_{GEC}(e_1)\pi_{GEC}, \\
P(e_{M+1}|e_1, \ldots, e_M) &= \frac{P(e_1, \ldots, e_{M+1})}{P(e_1, \ldots, e_M)},
\end{aligned}$$

where $\mathbf{u}^T$ is the all '1' row vector (implying the sum over all the values of the vector obtained from the matrix multiplications to the right of it).

This computation must be computed for each binary sequence $\{e_1, \ldots, e_M\}$; thus, computing the M^{th}-order BAMNC statistics is of exponential complexity, which limits how large M can practically be.[1]

[1] We use the same computations to compute the conditional probability distribution for the GEC in order to get a lower bound on the capacity of the channel as described in Section 2.2.3.

Chapter 3

LDPC Codes and SPA Decoding

In this chapter we review the two related topics of low-density parity-check (LDPC) codes and the sum-product algorithm (SPA) which is used to decode them.

We begin in Section 3.1 with a review of the definition of LDPC codes for both regular and irregular variants. The basics of linear block codes, parity-check equations and Tanner graphs are discussed to give a complete understanding of LDPC codes.

In Section 3.2, we describe the sum-product algorithm which is the basis for our joint channel-estimation/decoder design in Chapter 3. The SPA is an algorithm for simplifying the computation of a complex global probability distribution function by factoring it into a product of simple local probability functions and representing this factorization on a graph.

3.1 Low-Density Parity-Check Codes

Here we define the characteristics of LDPC codes. These are a class of linear block codes named for their sparse parity-check matrix, which is their defining characteristic. We begin

with the basics of linear block codes and parity-check matrices and define the main subclasses of LDPC codes from the literature.

This chapter makes use of the following standard notation for representing vectors and matrix multiplication. We denote vectors using boldface letters (e.g. $\mathbf{x} = (x_1, x_2, \ldots, x_n)$) and for the purposes of matrix multiplication we use the standard definition of vectors as column vectors. Matrices are denoted using a capital bold face letter. For any given $m \times n$ matrix $\mathbf{A}$, and length n vector $\mathbf{x}$, we perform the multiplication of matrix with vector using the standard method of matrix-column vector multiplication

$$\mathbf{Ax} = \mathbf{A} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}.$$

Since we are concerned only with linear block codes over the binary alphabet $\{0, 1\}$, we perform all multiplication and addition over the binary alphabet modulo-2. We use the symbol $\oplus$ to denote the operation of addition modulo-2.

3.1.1 Linear Block Codes

Linear block codes are channel codes defined as the null-space or kernel of a size $m \times n$ matrix known as a *parity-check* matrix. This forms a linear vector space denoted by $\mathcal{C}$, which is called the *codebook* for the code and is the set of all valid codewords for the code. Each codeword is a length n vector (n is the codeword block size of the code). While linear block codes are not restricted to variables over a binary alphabet, this work is only concerned with binary-input/binary-output channels so we will only define codes in terms of a binary alphabet.

Definition 3.1.1. Let $\mathbf{H}$ be an $m \times n$ *parity-check matrix* with entries taken from the binary set $\{0, 1\}$. Then a vector $\mathbf{c} = (c_1, c_2, \ldots, c_n)$, where $c_i \in \{0, 1\}$, is said to be a codeword of the *linear block code* defined by $\mathbf{H}$, if and only if $\mathbf{Hc} = \mathbf{0}_m$ where $\mathbf{0}_m$ is the all zero vector. We refer to the set $\mathcal{C} = \{\mathbf{c} : \mathbf{Hc} = \mathbf{0}_m\}$ as the *codebook* for the code.

Example 3.1.2. A Hamming code with $n = 7$ and $m = 3$ can be represented by the following parity-check matrix:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}. \tag{3.1}$$

Each row of the parity-check matrix represents a parity-check equation on the code while each column corresponds to a code-bit. A parity-check equation enforces the condition that the modulo-2 sum of the code-bits that participate in that parity-check is zero. For the Hamming code example in (3.1) the parity-check equations are

$$\begin{aligned} \tilde{h}_1(\mathbf{c}) &= c_1 \oplus c_4 \oplus c_6 \oplus c_7, \\ \tilde{h}_2(\mathbf{c}) &= c_2 \oplus c_4 \oplus c_5 \oplus c_7, \\ \tilde{h}_3(\mathbf{c}) &= c_3 \oplus c_5 \oplus c_6 \oplus c_7. \end{aligned} \tag{3.2}$$

Therefore, a vector c, is a codeword if and only if $\tilde{h}_i(\mathbf{c}) = 0$ for all i.

Linear block codes can be encoded through the use of a *generator matrix*, which is an $n \times k$ matrix $\mathbf{G}$, where $k = n - m$ and for which the rows are linearly independent. The matrix $\mathbf{G}$ forms a linear mapping from a k-dimensional binary vector-space to a n-dimensional binary vector-space that is the null-space of the parity-check matrix.

Definition 3.1.3. The *generator matrix* for a linear block code $\mathcal{C}$, defined by the size $m \times n$

parity-check matrix $\mathbf{H}$, is a size $n \times k$ matrix $\mathbf{G}$ such that $\mathbf{HG} = \mathbf{0}_{m\times k}$, where $\mathbf{0}$ is the all zero matrix.

Example 3.1.4. The Hamming code defined by (3.1) can be encoded using the following generator matrix:

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}. \tag{3.3}$$

Encoding a length k block of binary information $\mathbf{v} = (v_1, v_2, \ldots, v_k)$, using the code defined by $\mathbf{H}$, can be done by simply obtaining the product $\mathbf{x} = \mathbf{Gv}$; thus, $\mathbf{x} \in \mathcal{C}$ is the length n codeword for the input vector $\mathbf{v}$.

It is a simple process to determine if a given vector $\mathbf{x}$ is a valid codeword by computing the syndrome $\mathbf{Hx}$. If each of the parity-checks evaluate to zero, then it is a valid codeword. This provides an effective method for detecting if a received vector contains errors or for determining if error correction was successful (though this will not guarantee that it is the correct codeword). In decoding LDPC codes using iterative decoders such as the SPA, we use the parity-checks to terminate the decoding process once it has resulted in a valid codeword.

Another property of all linear block codes is that they can be represented graphically using a Tanner factor-graph [30]. This is a bipartite graph where parity-checks and code-bit variables are represented by nodes on the graph. A code-bit node is connected to a parity-

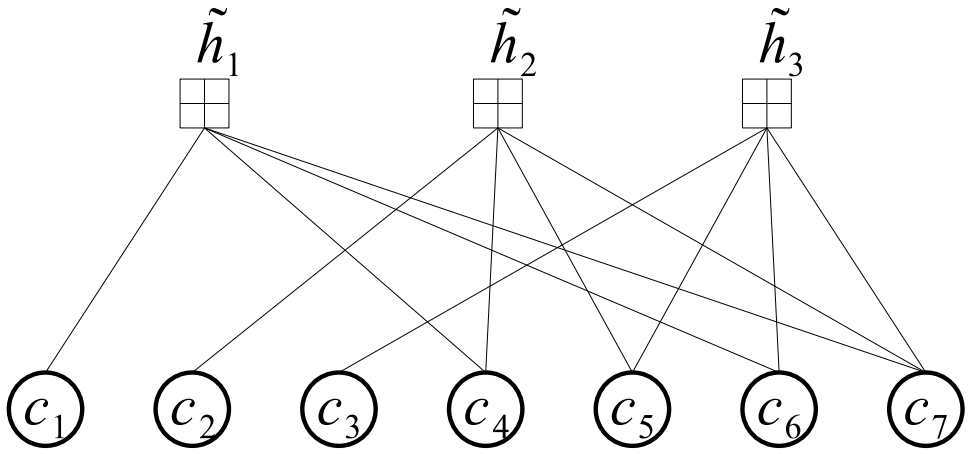

Figure 3.1: Tanner graph for the Hamming code from (3.1).

check node in the graph if that code-bit participates in that parity-check. Fig. 3.1 shows the parity-check matrix from Example 3.1.4 in its Tanner graph representation.

There are many families of linear block codes such as Hamming codes, Reed-Muller codes, cyclic codes like BCH codes and LDPC codes with which we are concerned.[1]

3.1.2 Regular LDPC Codes

As mentioned earlier, low-density parity-check codes have a sparse parity-check matrix (i.e., the proportion of non-zero entries of the matrix is small). Indeed, the density of the LDPC matrix goes to zero as the length of the code increases. Ee define regular-LDPC codes as they were defined in Gallager's original work [11].

Definition 3.1.5. A length n (d_v, d_c)-regular LDPC code is defined by an $m \times n$ parity-check matrix where each row has d_c 1's and each column has d_v 1's. The rate of this code is $r \geq 1 - d_v/d_c$ (with equality *iff* the rows of H are linearly independant) and we have the constraint that $m = (d_v/d_c)n$.

[1] More detailed descriptions of each of these classes of codes can be found in [16].

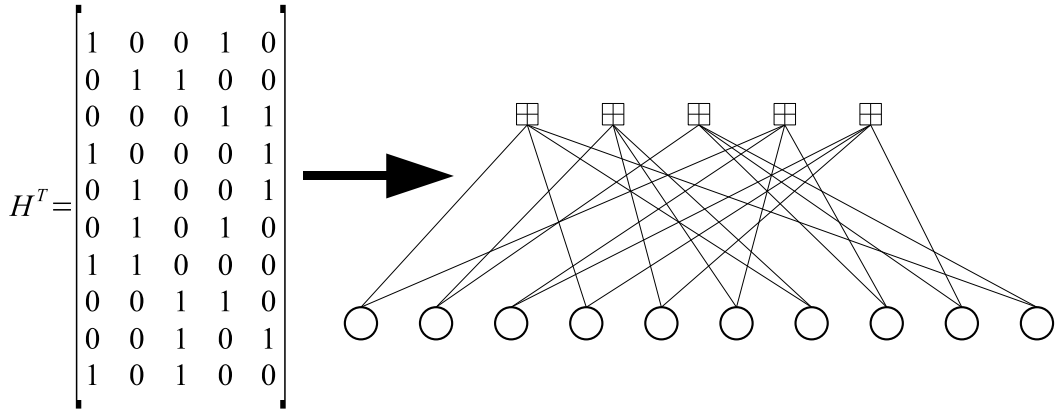

Figure 3.2: Example of a length 10 (2,4)-regular LDPC code in both parity-check and Tanner graph forms.

The density of a (d_v, d_c)-regular LDPC code is simply $\rho = d_c/n = d_v/m$, and clearly $\lim_{n\to\infty} \rho = 0$. An example of a length 10 (2,4)-regular LDPC code can be seen in Fig. 3.2 with both its parity-check matrix and Tanner graph.

Often LDPC codes are subject to the additional constraint that no two rows share more than one column for which they both have a '1' in that column. This results in a Tanner factor graph for the code which does not contain cycles of length four. Cycles of length four are the shortest cycles possible in a bipartite graph (Fig. 3.3). The shortest cycle in a graph defines the *girth* so as a result of this constraint the girth of this LDPC code will be at least 6. The reason for requiring this will become clearer when the sum-product algorithm decoder is discussed, but for now it is sufficient to say that short cycles negatively affect the performance of message-passing decoders like the SPA.

The sparse nature of the parity-check matrix makes it possible to use message-passing decoding algorithms with linear complexity.[2] Message-passing algorithms estimate the probability of error for each bit by passing messages containing estimates of the probability

[2]Linear complexity means that the number of computations needed to decode a block increases linearly with the block length n.

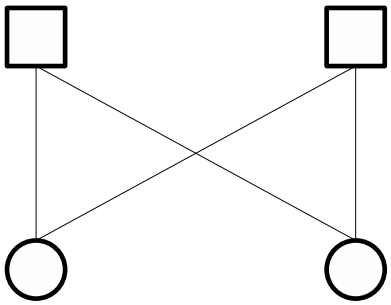

Figure 3.3: Example of a length 4 cycle in a Tanner graph.

of error along the edges between code-bits and parity-checks on the factor graph iteratively. For LDPC codes, the number of edges on the graph grows linearly with n. For a (d_v, d_c)-regular LDPC code,

$$N_{Edges} \quad = \quad nd_v = md_c.$$

The number of messages passed grows linearly in n; thus, we say that the decoding algorithm is of order n which is often written as $O(n)$. The most effective message passing decoder is the SPA [15] which is a belief propagation [21] algorithm for factor graphs.

The existence of a linear-time belief propagation decoder for LDPC codes is something they have in common with Turbo codes. In fact the SPA can also be used to perform belief propagation decoding of Turbo codes and convolutional codes. The second thing they have in common is excellent performance under belief propagation decoding. Simulated performance of LDPC codes is within a fraction of a decibel of the Shannon limit for the AWGN channel [4, 26]. This is one reason why LDPC codes have generated so much attention recently.

LDPC codes also have some advantages over Turbo codes. Unlike Turbo codes, the decoding process for LDPC codes is verifiable by computing the parity-check of the decoder output. This allows for early termination of the decoding algorithm once a valid

codeword is obtained, as well as allowing for the decoder to determine if the decoding of a received block has failed. Detecting failures allows the receiver to request retransmission if needed. LDPC codes also do not require the use of a long interleaver which is an integral component of Turbo encoders and decoders.

Furthermore, Turbo codes generally experience a phenomenon known as 'error-flare' (often incorrectly referred to as 'error-floor') at a higher bit-error rate than LDPC codes. Error-flare is when the bit-error rate stops decreasing exponentially as the channel quality improves causing the plotted error curve to 'flare' outwards. This is due to the poor distance properties of their constituent convolutional codes [22]. In fact, under the assumption of optimal decoding, the existence of capacity-achieving LDPC codes can be proven [17], a property which has not been demonstrated for Turbo codes.

3.1.3 Irregular LDPC Codes

Just like regular LDPC codes, irregular LDPC codes are defined by their sparse parity-check matrix. Unlike regular LDPC codes, the definition for irregular LDPC codes is looser and allows the number of 1's in each row and column to vary. An irregular LDPC code is defined by a *degree distribution*. The degree of a variable (code-bit) or check node is the number of edges that intersect with that node. For a variable node, it is the number of parity-checks it participates in, and for a check node, it is the number of variables which participate in that parity-check.

Definition 3.1.6. The degree distribution of an LDPC code is defined by the sets, $\{\rho_2, \rho_3, \ldots, \rho_{d_c}\}$ and $\{\lambda_2, \lambda_3, \ldots, \lambda_{d_v}\}$, where ρ_i (λ_i) is the proportion of non-zero (i.e. containing a '1') in the parity-check matrix in a row (column) of weight i. More specifically it is defined as the proportion of edges connected to check (variable) nodes of degree i and clearly

$\sum_i \rho_i = \sum_i \lambda_i = 1$. Using this definition we can categorize regular LDPC codes as a subset of irregular LDPC codes which have $\rho_{d_c}, \lambda_{d_v} = 1$.

Using (3.4) and (3.5), we can determine the number of variable nodes V_i, and parity-check nodes C_i, of degree i that are in the Tanner graph. (3.6) gives the relationship between the number of checks and the block length of the code which in turn gives us the rate of the code in (3.7):

$$V_i = n\frac{\lambda_i/i}{\sum_{j=2}^{d_v}\lambda_j/j}, \tag{3.4}$$

$$C_i = m\frac{\rho_i/i}{\sum_{j=2}^{d_c}\rho_j/j}, \tag{3.5}$$

$$m = n\frac{\sum_{j=2}^{d_c}\rho_j/j}{\sum_{j=2}^{d_v}\lambda_j/j}, \tag{3.6}$$

$$r = 1 - \frac{\sum_{j=2}^{d_c}\rho_j/j}{\sum_{j=2}^{d_v}\lambda_j/j}. \tag{3.7}$$

The family of irregular LDPC codes is clearly a super-set of the family of regular LDPC codes and it has been shown that there exist irregular degree distributions that greatly outperform regular LDPC codes for a variety of channels under SPA decoding. Such channels include the AWGN channel and the BSC [26, 4] and some Gilbert-Elliot channels [7]. The design of a good degree distribution generally involves a combination of trial and error, careful searching and testing and the use of analysis techniques such as density evolution (DE) [27], EXIT chart analysis [31] or other techniques.

These techniques are generally computationally complex though significantly simpler than brute force searching and testing. This is even more difficult for channels with memory than for channels without memory. Despite these difficulties DE techniques have been applied to the GEC [6] and the FMCC [20] for a few simple examples of these channels. However, very little work has been done to design irregular codes for channels with mem-

ory due to the difficulty in performing this type of analysis.

3.2 The Sum-Product Algorithm

The Sum-Product Algorithm (SPA) is a form of Pearl's belief propagation [21] that uses message passing on factor graphs. Many different algorithms in computer science, probabilistic modeling, statistics, signal processing and digital communications can be shown to be instances of the SPA [15] and can be performed using the SPA. In fact, Turbo decoding, the BCJR algorithm and other forms of belief propagation, can all be performed as instances of the SPA [15].

In this section, we describe in detail how the SPA is derived and give examples of message passing on factor graphs. We also define factor graphs and show how global functions can be factored as a product of local functions to be represented as a factor graph.

We will show how the SPA can be combined with the Tanner graphs of LDPC codes to perform approximate belief propagation decoding, as well as explain why this decoding method is only an approximate belief propagation technique and is sub-optimal. Furthermore, we extend this decoding method by factoring the Markov chain representation of the channel state for the GEC and for BAMNCs to show how the SPA can be used to design a joint channel-estimator/decoder.

3.2.1 Factor Graphs

Factor graphs are a simple way to graphically represent the statistical dependence between variables. Imagine we have a global function of several variables $g(x_1, \ldots, x_n)$ which can

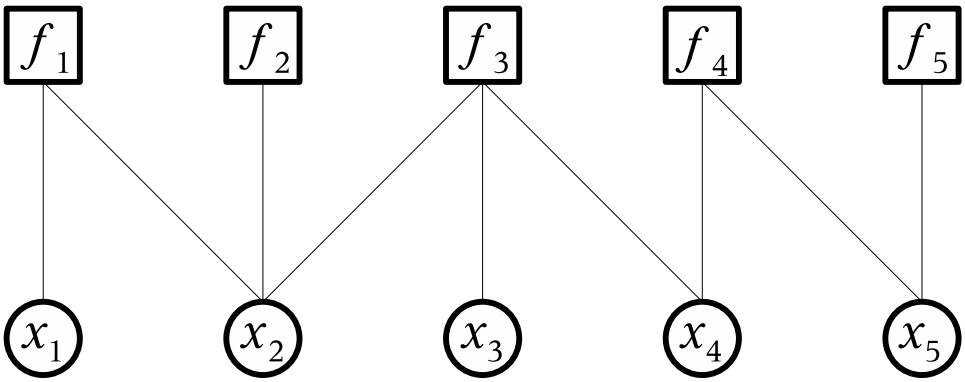

Figure 3.4: Factor graph for (3.9).

be factored into a product of local functions

$$g(x_1, \ldots, x_n) \quad = \quad \prod_{j=1}^{N} f_j(X_j), \tag{3.8}$$

where $X_j \subset \{x_1, \ldots, x_n\}$ and is the set of variables participating in the j^{th} factor and N is the number of factors.

A factor graph is a bipartite graph where the graph connects nodes representing each of the variables x_i to nodes representing the functions f_j so that there is an edge between x_i and f_j if and only if $x_i \in X_j$.

Example 3.2.1. For example, let us define $g(\cdot)$ as follows

$$g(x_1, \ldots, x_5) \quad = \quad f_1(x_1, x_2) f_2(x_2) f_3(x_2, x_3, x_4) f_4(x_4, x_5) f_5(x_5), \tag{3.9}$$

which can be represented as the factor graph shown in Fig. 3.4.

This work is mainly concerned with two types of factor graphs: Tanner factor graphs of parity-check codes [30] and Wiberg-type factor graph representations of Markov chains [34]. Both of these types of graphs are discussed in [15]. Tanner graphs have previously been

defined in Section 3.1 along with the linear block codes they are used to describe. They are simply factor graphs where we connect variable nodes representing code-bit variables to factor nodes representing the parity-checks that those code-bits participate in.

For a parity-check matrix with n bits and m parity-checks we define the global function of a parity-check matrix $\mathbf{H}$ as follows:

$$\begin{aligned} g_{\mathbf{H}}(x_1, \ldots, x_n) &= \begin{cases} 1 \text{ if } (x_1, \ldots, x_n) \in \mathcal{C}, \\ 0 \text{ otherwise.} \end{cases} \\ &= \prod_{i}^{m} h_i(X_i), \end{aligned}$$

where

$$h_i(X_i) = \begin{cases} 1 \text{ if } \bigoplus_{x_j \in X_i} x_j = 0 \\ 0 \text{ otherwise,} \end{cases} \tag{3.10}$$

and $\bigoplus_{x_j \in X_i}$ is the modulo-2 sum taken over all the elements $x_j \in X_i$. The global function $g_{\mathbf{H}}(\cdot)$ is the indicator function of the code (i.e., $g_{\mathbf{H}}(\mathbf{c}) = 1$ iff $\mathbf{c} \in \mathcal{C}$). The local function $h_i(X_i)$ is the indicator function of the i^{th} parity-check for the code and X_i is the set of all variables x_j which participate in the i^{th} parity-check.

The Tanner-graph represents the factorization of a global indicator function which tests if a vector is a valid codeword, into a product of local indicator functions. Each indicator function represents a parity-check equation for the code. Recall that in Section 3.1 we showed that the parity-check matrix, which defines the set of parity-check equations, is sufficient to completely characterize the codebook for a linear block code.

The global function for the state-sequence of a Markov chain can be factored into local functions which describe the probability distribution of the next state given the current

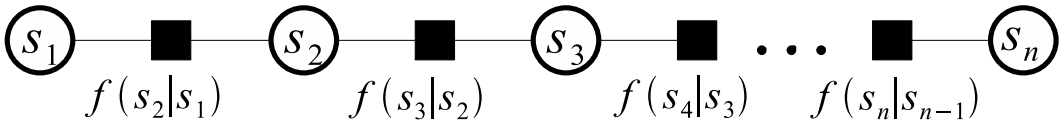

Figure 3.5: Factor graph for the state sequence of a Markov process

state.

$$g(s_1, \ldots, s_n) \quad = \quad \Pr(s_1) \prod_{i=1}^{n-1} \Pr(s_{i+1}|s_i).$$

This is because the Markov property of Markov processes states that the probability of the next state conditioned on the infinite past sequence of states is equal to the probability of the next state conditioned only on the most recent state

$$\Pr(s_{i+1}|s_i, s_{i-1}, \ldots) = \Pr(s_{i+1}|s_i).$$

We obtain a Markov chain factor graph which can be seen in Fig. 3.5. The Markov factor-graph can be extended to represent a hidden Markov model to obtain what is called a Wiberg-type graph [34]. In a Wiberg-type representation, the state of the Markov chain is related through each factor node to an observed or *visible* variable. The SPA can then be used to infer the probability of the state sequence from these observations through the relationship given by the factors. The Wiberg-type factor graph is shown in Fig. 3.6. The Wiberg-type graph represents the factorization of the global function

$$g(s_1, \ldots, s_n, e_1, \ldots, e_n) \quad = \quad \Pr(s_1) \prod_{i=1}^{n-1} \Pr(s_{i+1}|s_i) \Pr(e_i|s_{i+1}, s_i).$$

In an SPA decoder we observe variables which are the bits received after being transmitted across the channel. These can be considered 'visible' variables of the 'hidden' state

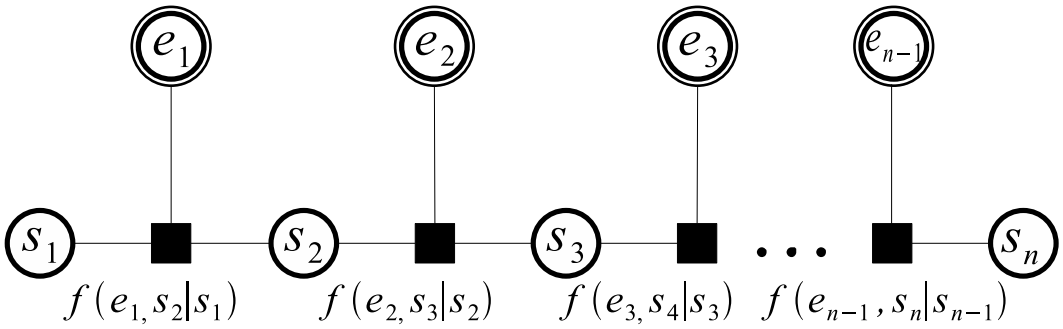

Figure 3.6: A factor graph for the state-sequence of a hidden Markov model (Wiberg-type graph).

sequence for the channel.

3.2.2 Message Passing

The SPA is a 'message-passing' algorithm. This means that it operates by passing messages along the edges of the factor graph. At each node, messages are received by the node and uses the SPA update rules to compute the outgoing messages. One rule is for the factor nodes and one is for the variable nodes. Outgoing messages are computed and passed to the connected nodes on the outgoing edges. A *message* in the SPA is a function $\mu(x)$ which carries a conditional probability distribution for the variable x. The distribution is conditional on the messages distributions from other variable and factor nodes which were passed to compute it (using the message update rules). First, we will define the message passing rules for both the variable nodes and factor nodes. Second, we will show step-by-step how the SPA is used to compute the *marginalize product-of-functions* (MPF) rule using Example 3.2.1.

Definition 3.2.2. We define the SPA update rules for the messages in Fig. 3.7 as follows. Let $\mathfrak{X}$ be the set of all variables and $\mathfrak{F}$ the set of all factor functions. Let $x \in \mathfrak{X}$ denote a variable in the set and $f \in \mathfrak{F}$ a function. We denote $n(x)$ as the local neighbourhood of

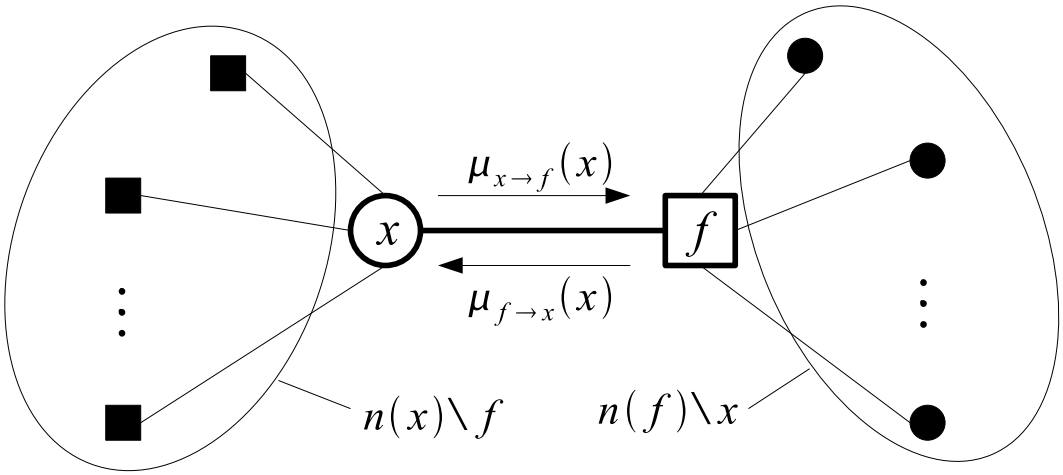

Figure 3.7: The messages passed between variable nodes and factor nodes by the SPA from Def'n. 3.2.2.

functions connected to the node representing x in the graph and $X = n(f)$ as the local neighbourhood of variables connected to f. The update rules for the SPA are expressed as follows:

variable-to-factor:

$$\mu_{x\to f}(x) \quad = \quad \prod_{f'\in n(x)\backslash\{f\}} \mu_{f'\to x}(x). \tag{3.11}$$

factor-to-variable:

$$\mu_{f\to x}(x) \quad = \quad \sum_{n(f)\backslash\{x\}} f(X) \left(\prod_{x'\in n(f)\backslash\{x\}} \mu_{x'\to f}(x') \right). \tag{3.12}$$

As previously mentioned the SPA computes the MPF for the variable nodes in the factor

graph. Using Example 3.2.1 the MPF for x_1 can be written as

$$g_1(x_1) = \sum_{\sim\{x_1\}} g(x_1, x_2, x_3, x_4, x_5), \tag{3.13}$$

where $\sim \{x_1\}$ is called the *summary* and denotes the set of all variables *not* being summed over. Using the factorization of $g(\cdot)$ in (3.9) we can write this as

$$\begin{aligned} g_1(x_1) = & \sum_{x_2} f_1(x_1, x_2) f_2(x_2) \cdot \\ & \cdot \left(\sum_{x_3, x_4} f_3(x_2, x_3, x_4) \left(\sum_{x_5} f_4(x_4, x_5) f_5(x_5) \right) \right). \end{aligned} \tag{3.14}$$

Using the SPA we can compute the MPF of this problem by arranging the factor graph from Fig. 3.4 as a tree with x_1 as the root node. Then we pass messages from the leaves of the tree to the root as follows.

Example 3.2.3. Using the equations and factor graph from Example 3.2.1 and the update rules (3.11) and (3.12), we can compute the MPF from (3.14) as follows:

$$\begin{aligned} \mu_{f_5 \to x_5}(x_5) &= f_5(x_5) \\ \mu_{x_5 \to f_4}(x_5) &= f_5(x_5) \\ \mu_{f_4 \to x_4}(x_4) &= \sum_{x_5} f_4(x_4, x_5) f_5(x_5). \end{aligned}$$

$$\begin{aligned} \mu_{x_4 \to f_3}(x_4) &= \mu_{f_4 \to x_4}(x_4) \\ &= \sum_{x_5} f_4(x_4, x_5) f_5(x_5) \\ \mu_{x_3 \to f_3}(x_3) &= 1. \end{aligned}$$

$$\begin{aligned}
\mu_{f_3\to x_2}(x_2) &= \sum_{x_3,x_4} f_3(x_2,x_3,x_4)\mu_{x_3\to f_2}(x_3)\mu_{x_4\to f_2}(x_4) \\
&= \sum_{x_3,x_4} f_3(x_2,x_3,x_4)\sum_{x_5} f_4(x_4,x_5)f_5(x_5) \\
\mu_{f_2\to x_2}(x_2) &= f_2(x_2).
\end{aligned}$$

$$\begin{aligned}
\mu_{x_2\to f_1}(x_2) &= \mu_{f_2\to x_2}(x_2)\mu_{f_2\to x_2}(x_2) \\
&= f(x_2)\left(\sum_{x_3,x_4} f_3(x_2,x_3,x_4)\left(\sum_{x_5} f_4(x_4,x_5)f_5(x_5)\right)\right) \\
\mu_{f_1\to x_1}(x) &= \sum_{x_2} f_1(x_1,x_2)\mu_{f_2\to x_2}(x_2) \\
&= \sum_{x_2} f_1(x_1,x_2)f_2(x_2)\left(\sum_{x_3,x_4} f_3(x_2,x_3,x_4)\left(\sum_{x_5} f_4(x_4,x_5)f_5(x_5)\right)\right) \\
&= g_1(x_1).
\end{aligned}$$

To truly take advantage of the computations performed by the SPA one must compute the complete set of MPFs for the factor graph. It is easily seen that passing the messages back in the other direction computes the MPF for every variable. So while we pass eight messages to compute the MPF for x_1, in order to compute all five MPF solutions we need only pass sixteen messages. This makes the SPA very efficient for computing large sets of MPFs as it reuses the computations to solve for each variable.

3.2.3 SPA Decoder for LDPC Codes

We now describe how the SPA decoder is used to decode LDPC codes. In fact this method can be applied to any code defined by a parity-check matrix, but for LDPC codes the complexity of the decoder is of linear order in the length of the code. We will show how the SPA decoding algorithm can be used as a decoder for memoryless channels as well as in

the design of a joint channel-state estimator/decoder for finite-state Markov channels such as those described in Section 1.2.

Factorizations for SPA Decoding

The SPA can be applied to the decoding of LDPC codes through factoring the probability distribution

$$\hat{\mathbf{x}} = \arg\max_{\mathbf{x}\in\mathcal{C}}(f_{\mathbf{X}|\mathbf{Y}}(\mathbf{x}|\mathbf{y}))$$

where $\mathcal{C}$ is the set of all codewords. The rule is: given that we receive $\mathbf{y}$ then $\hat{\mathbf{x}}$ is the decoding decision which minimizes the probability of block error. The distribution equation $f_{\mathbf{X}|\mathbf{Y}}(\mathbf{x}|\mathbf{y})$ can be rewritten as

$$g(\mathbf{x}) = K f_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x}) f_{\mathbf{X}}(\mathbf{x}) = f_{\mathbf{X}|\mathbf{Y}}(\mathbf{x}|\mathbf{y}),$$

where K is a normalizing constant, $\mathbf{x}$ and $\mathbf{y}$ are length n vectors representing the input and output of the channel respectively, $f_{\mathbf{Y}|\mathbf{X}}(\cdot|\cdot)$ is the conditional probability of the channel output and $f_{\mathbf{X}}(\cdot)$ is the probability distribution of the transmitted codewords. We assume $\mathbf{y}$ is a known quantity since it is what the decoder receives; thus, $g(\cdot)$ is only a function of $\mathbf{x}$.

The decoding rule is: given $\mathbf{y}$, we choose $\mathbf{x}$ to maximize $g(\cdot)$, which optimally minimizes the probability of block error. If we assume that the channel input vectors are uniformly distributed (i.e., the probability of sending any given codeword is uniform) then this is equivalent to the *maximum likelihood* (ML) decoding rule which maximizes the quantity $g(\mathbf{x}) = f_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x})$ over $\mathbf{x}$.

For a binary-input memoryless channel using a linear block code defined by a parity-check matrix $\mathbf{H}$ and assuming uniform input, $g(\cdot)$ factors as

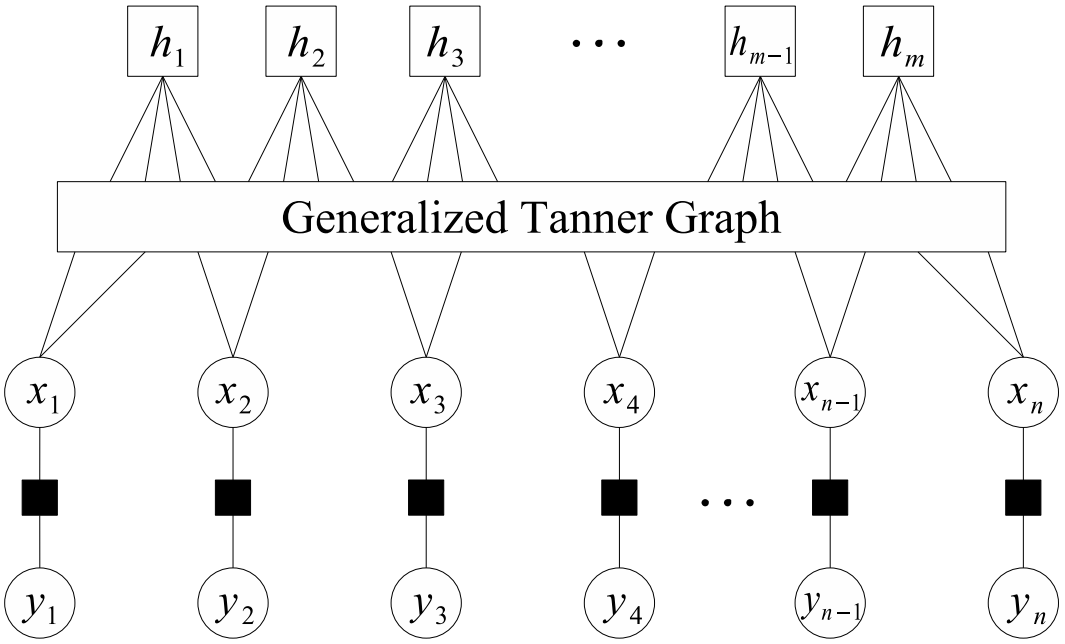

Figure 3.8: Factor graph of (3.15) for decoding parity-check codes over memoryless channels.

$$g(x_1, x_2, \ldots, x_n) = K\,|\mathcal{C}|^{-1} \prod_{i=1}^{m} h_i(X_i) \prod_{j=1}^{n} f_{Y|X}(y_j|x_j), \tag{3.15}$$

where $|\mathcal{C}| = 2^k$ is the size of the codebook for our linear block code and $h_i(\cdot)$ is the indicator function of the i^{th} parity-check from (3.10).

The factor graph for (3.15) is shown in Fig. 3.8. We see how the code-bits, x_i's, are related to the received bits, y_i's, through the factors, $f_{Y|X}(y_i|x_i)$, which is the conditional distribution of the channel transition model. We only need to determine the variable values which maximize $g(\cdot)$ so we can ignore the constants in (3.15).

For channels with memory, where the y_i's are not conditionally independent given the x_i's, we cannot factor $f_{\mathbf{Y}|\mathbf{X}}(\cdot|\cdot)$ as in (3.15). Fortunately, for channels with a Markov channel-state process, like the ones described in the previous section, we can factor the Markov chain of the channel state sequence as we have just shown. Thus, the function $g(\cdot)$

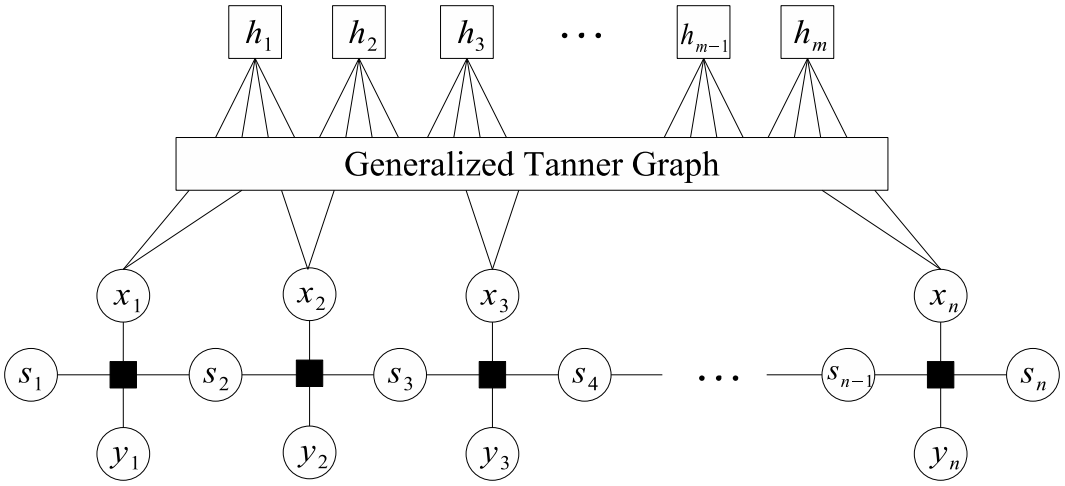

Figure 3.9: Factor graph of (3.16) for decoding parity-check codes over finite-state Markov channels.

becomes

$$g(x_1, x_2, \ldots, x_n) = K\,|\mathcal{C}|^{-1} \prod_{i=1}^{m} h_i(X_i)\,\Pr(s_1) \prod_{j=1}^{n} \Pr(s_{j+1}|s_j)\,\Pr(y_j|x_j, s_{j+1}, s_j). \tag{3.16}$$

This factorization is represented graphically by the factor graph in Fig. 3.9.

At each iteration, the SPA explicitly computes the function $g_i(x_i|n(n(x_i)))$ for each variable node in the graph, where $n(n(x_i))$ is the set of all received bits which share parity-checks with x_i. Due to cycles in the graph this can only approximate the solution to $g_i(x_i|\mathbf{y}^n)$, even after many iterations. The maximization of this function over x_i is the bit-wise maximum a posteriori probability (MAP) decoding rule. The SPA is an approximate MAP decoder. We note that while exact MAP rule decoding of LDPC codes is order NP (no polynomial time algorithm exists), the SPA is a linear time algorithm.

Message Passing for SPA Decoding

Message passing for decoding using the SPA is, in general, exactly the same as it was described in Section 3.2.2. There is one key difference however; the algorithm does not have a termination point since the factor graphs for linear block codes generally contain cycles so their Tanner graph cannot be arranged as a tree.

While it is entirely possible to create a parity-check matrix for which the factor graph does not contain cycles, codes constructed in this manner have extremely poor properties and are not good choices for error-correcting. Cycles result in messages being passed around continuously that will never return to a termination point. Messages passed to a node in a cycle in the graph will also be dependent on the message passed from that node a number of iterations earlier which can affect the performance of the algorithm.

Cycles in LDPC codes make the SPA decoding algorithm an iterative decoding algorithm. For each iteration we pass messages from the set of variable nodes along each edge in the graph to the parity-check nodes and then pass the updated messages back to the variable nodes. While this process is not capable of computing the MPFs exactly even after a large number of iterations, it has been shown that for sufficiently large and sparse graphs the local neighbourhoods within the graph are almost always cycle free and the algorithm converges to the optimal solution [27]. In other words, for sufficiently large LDPC codes the probability of short cycles is small and the effect of the larger cycles on the convergence of the SPA is also small.

LDPC codes are ideal for decoding using the SPA since they form sparse graphs, in fact the density of the parity-check matrix decreases as the block length is increased. Furthermore, for a particular regular-LDPC code, the number of checks-per-bit is fixed, which implies that the number of messages computed per bit is constant; thus SPA decoding is of linear order complexity in the length of the code.

For memoryless channels, using the graph in Fig. 3.8, the SPA operates by passing messages containing the probability that each transmitted bit x_i was either a zero or a one. To simplify this, we pass the likelihood ratio,

$$\mathcal{L}(x_i) = \Pr(x_i = 0)/\Pr(x_i = 1),$$

for each bit. The algorithm is initialized with the messages passed from the received bit variable-nodes through the channel factor nodes which simply passes the value

$$\zeta_i(y_i) = \Pr(x_i = 0|y_i)/\Pr(x_i = 1|y_i)$$

to each variable node for the x_i's.

For each iteration, the messages are computed for each edge in the Tanner factor graph, first going from variable to parity-check and then from parity-check to variable. The messages are computed using the SPA update rules (3.11) and (3.12) combined with the factorization in (3.15). The message equations are as follows:

$$S_{ij} = \prod_{k \neq j} P_{ki}, \tag{3.17}$$

$$P_{ij} = \frac{1 + \prod_{k \neq j} (1 - 2(1 + S_{ki})^{-1})}{1 - \prod_{k \neq j} (1 - 2(1 + S_{ki})^{-1})}, \tag{3.18}$$

where S_{ij} is the message passed from the i^{th} variable node to the j^{th} parity-check node and P_{ij} is the message passed from the i^{th} parity-check node to the j^{th} variable node. To begin, the algorithm initializes the S_{ij} messages to the messages received from the channel factors; thus, $S_{ij} = \zeta(x_i)$ for all i, j as mentioned above.

For each iteration we have an estimate of each bit determined by

$$\hat{x}_i = \begin{cases} 0 \text{ if } \left(\prod_j P_{ij}\right) \geq 1, \\ 1 \text{ if } \left(\prod_j P_{ij}\right) < 1, \end{cases}$$

where $\hat{x}_i$ is our estimate for x_i.

Using the parity-checks, we can verify if the estimate for transmitted bits is a valid codeword and if so the algorithm is terminated. Otherwise, the algorithm continues until a valid codeword is obtained or until a maximum number of iterations has been performed.

The algorithm can be extended to pass messages from the variable nodes to a channel factor graph representing the Markov chain of a finite-state channel with Markov memory such as is shown in Fig. 3.9. This requires four additional messages to be passed between the channel factor graph and the Tanner factor graph of the code and along the channel factor graph. The entire set of messages for both graphs can been seen in Fig. 3.10.

The equations for messages passed between the code-bit variable nodes and the parity-check factor nodes are unchanged from those given in (3.17) and (3.18). The four additional messages are: χ_i, the extrinsic information passed from the Tanner graph to the channel graph, ζ_i, the channel message passed back from the channel graph, α_i, the forward message on the channel factor graph and β_i, the backward message. The α_i and β_i messages are the SPA estimates of the probability distribution for the state of the channel at time i.

The channel messages equations are derived from (3.11) and (3.12) combined with the factorization in (3.16) and are

$$\chi_i = \prod_j P_{ji}, \tag{3.19}$$

$$\alpha_{i+1}(s_{i+1}) = \sum_{x_i,s_i} \alpha_i(s_i)\chi_i(x_i)\Pr(s_{i+1}|s_i)\Pr(x_i|y_i,s_i,s_{i+1}), \tag{3.20}$$

$$\beta_i(s_i) = \sum_{x_i,s_{i+1}} \beta_{i+1}(s_{i+1})\chi_i(x_i)\Pr(s_{i+1}|s_i)\Pr(x_i|y_i,s_i,s_{i+1}), \tag{3.21}$$

$$\zeta_i = \frac{\sum_{s_i,s_{i+1}} \alpha_i(s_i)\beta_{i+1}(s_{i+1})\Pr(s_{i+1}|s_i)\Pr(x_i=0|y_i,s_i,s_{i+1})}{\sum_{s_i,s_{i+1}} \alpha_i(s_i)\beta_{i+1}(s_{i+1})\Pr(s_{i+1}|s_i)\Pr(x_i=1|y_i,s_i,s_{i+1})}. \tag{3.22}$$

We note that while χ_i was originally defined as a likelihood ratio in (3.19), in (3.20) and (3.21), we represent it as a distribution function of x_i to make the equations easier to express. The two representations of the message are equivalent, since x_i is a binary number.

These equations are similar to those used in [8] and the same notation for the channel message has been used to make comparisons between this work on the BAMNC and the work in [8] on the GEC easier. The main difference is that the relationship between channel state transitions and the noise process for the BAMNC model is deterministic rather than probabilistic. Furthermore BAMNCs only have two possible transitions for each state regardless of the number of states, which is also true for the GEC because it has only two states but is not true in general for HMM based channels. The effect of these diferences will become clearer when we derive the exact computations used for decoding in Chapter 4.

The extended SPA decoding algorithm is a joint channel state estimator/decoder. The Tanner graph portion of the decoder can perform decoding on the code itself, while the channel graph portion performs channel state estimation. The advantage of the joint system is that the decoder gains from the knowledge of the channel model while the channel estimation gains from the knowledge of the code structure. This improves the ability of both parts to produce an accurate estimate of the code-bits and channel state.

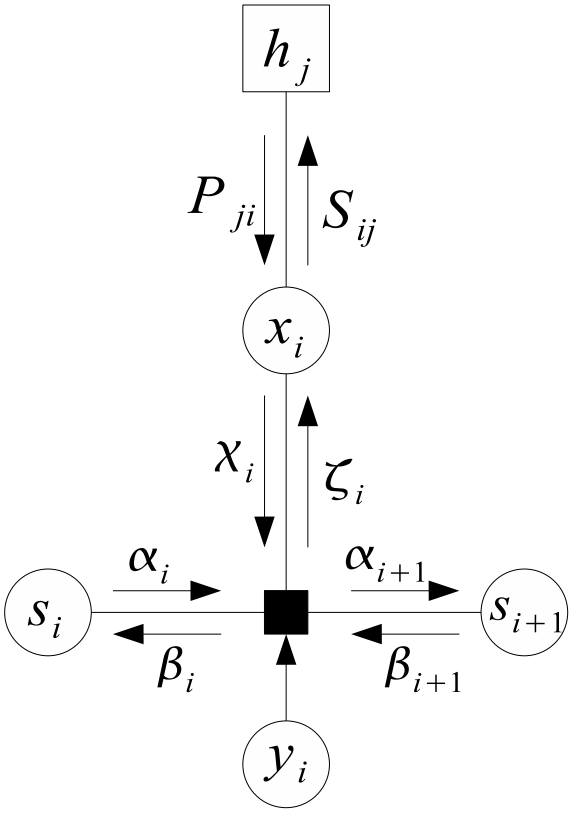

Figure 3.10: All the local messages passed for the extended SPA decoder

Chapter 4

System Design

This chapter describes the design of a joint channel-state estimator/decoder for binary channels with Markov memory, specifically the GEC and the QBC

The purpose of such a system is to design an error-correcting coding scheme capable of recovering errors in data transmitted over these channels, or over channels similar enough to these channels that they can be considered effective models. In order to be effective, such a scheme must use both knowledge of the encoder and of the channel model to decode the received data. In other words, the decoder must be able to exploit the relationship between code bits as well as the relationship between channel errors.

In order to create practical systems for decoding using methods designed for memoryless channels, techniques like interleaving are used to make the data appear to be memoryless to the decoder. The memoryless equivalent channel often has lower capacity than the original channel. The performance of this scheme is necessarily bounded away from capacity relative to the true Shannon limit when channel memory is considered. So even though it may be very close to the performance limit of the memoryless channel, there may still be a large gap to the capacity of the actual channel with memory.

Exploiting the channel memory to achieve better performance than schemes based on memoryless channels can lead us to better performance in terms of the bit-error rate of the decoder; however, this performance generally comes at the cost of increased computational complexity in the design of the decoder. It is a central goal of decoder design to achieve excellent decoder performance while also maintaining a sufficiently small decoding delay.[1] These two objectives are often at odds with one another.

It is also necessary to use 'good' channel models in order to exploit channel memory effectively. Channel models should attempt to be 'good' models of real world channel noise phenomena such as fading, multi-path propagation, burst error noise, etc. This is important because the decoder will rely on how close a 'fit' the channel model is to the real channel when it is operating in the real world. A good fit means it will be able to exploit the channel memory and possibly outperform an equivalent memoryless system. Additionally, a 'good' channel model is also important for simulation and testing of the system. We would like the results of simulating the communication system to give us results that are representative of the performance of that system in the real world.

The three main objectives in the design of this error-correcting decoding scheme are: to be able exploit channel memory, to obtain performance as close to the Shannon limit as possible and to do so with minimum cost in terms of computational complexity.

We will compare our scheme with the traditional scheme used for decoding LDPC codes over the memoryless binary symmetric channel in terms of performance and complexity. In order to make these comparisons, we develop a computer simulation of the noisy channel coding scheme for channels with and without memory using the designs for the encoder, channel models and the decoder.

[1] The decoding delay is the time between the decoder receiving a block of encoded data from the channel and the time it completes decoding that block.

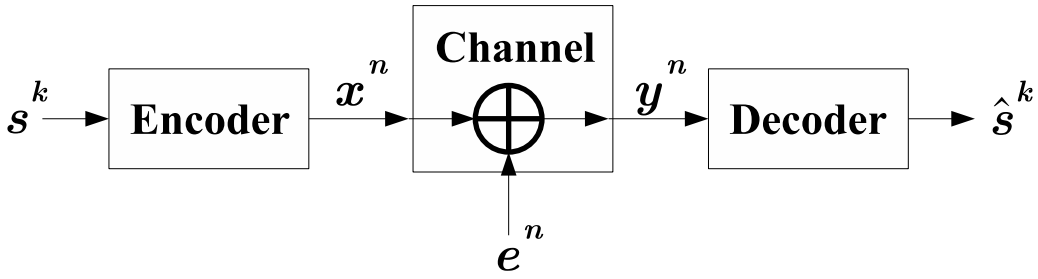

Figure 4.1: Block diagram of an error-correcting coding communication system with an additive noise channel.

4.1 Overview of the System

As described in Chapter 1, an error-correcting coding scheme is comprised of two components: the encoder and the decoder. For LDPC codes (and linear block codes in general) the encoder takes binary sequences of length k and produces encoded binary sequences of length $n > k$ which contain $m = n - k$ parity-check bits. For simplicity we assume the source data is uniformly distributed[2] and that the channel has a binary stationary ergodic noise process. The decoder receives the encoded data after it has been affected by the channel noise and attempts to correctly determine what was sent from the encoder by correcting the errors in the received data.

The basic LDPC encoder functions via matrix multiplication using the generator matrix for the code. Since, this work is mainly concerned with decoder design, we do not explore other encoding schemes here, which may have lower order complexity than matrix multiplication.

The decoder is based on the MAP decoding rule. The basic idea is to decode for each bit to the value which was most probably sent given the channel's block transition probability, the probability distribution of the code and the data received by the decoder. This problem

[2]This is equivalent to assuming that the source is ideally compressed before encoding.

is extremely difficult, and no known algorithm solves it exactly for LDPC codes in linear order time. The SPA decoder can be used to sub-optimally compute the bitwise MAP decoding rule with excellent results.

The SPA decoder is an iterative decoder operating on the principle of belief propagation. The probability distribution associated with the optimal decoding rule is factored into a product-of-functions operating on local variable subsets and calculations are carried out locally to get an estimate of the global distribution. The decoder computes the probability distribution for each bit based on its local relationship to other sent bits via parity-check equations. In accounting for a channel with memory, we also consider the related channel state at the time it was sent. Furthermore, the probability distribution for the channel state at each time is computed using the SPA based on its relationship with both the previous and subsequent states.

The channel models are based on finite-state Markov chains and have been described in detail in Section 2.2. Each state of the channel has a probability of bit error and a probability of state transition associated with it. For the purpose of computer simulation of these channels we use pseudo-random number generation to determine the channel noise output and channel state transition for each channel use.

The encoder, channel simulation and decoder are concatenated as shown in Fig. 4.1 in order to simulate a complete communication system. We test the performance limits of the LDPC coding scheme for these channels and compare performance against other similar systems using this simulation method.

4.2 LDPC Encoder Design

For encoding, we use a systematic encoder, for which the first k bits of the length n encoded sequence $\mathbf{c} = (c_1, \ldots, c_n)$ are the uncoded length k input sequence $\mathbf{v} = (v_1, \ldots, v_k)$, and the last m bits are the parity-check bits $\mathbf{p} = (p_1, \ldots, p_m)$. All linear block codes can either be put into a *strictly-systematic* form or they are equivalent[3] to a code which can. By strictly-systematic we mean that the form of G is

$$\mathbf{G} = \left[\frac{\mathbf{I}_k}{\mathbf{A}}\right],$$

where $\mathbf{G}$ is the generator matrix, $\mathbf{I}_k$ is the $k \times k$ identity matrix and $\mathbf{A}$ is a matrix which computes the parity-check bits for the codeword.

Encoding the length k vector $\mathbf{v}$ results in $\mathbf{Gv} = \mathbf{c}$. Matrix-vector multiplication is $O(m \times k)$ or $O(n^2)$, since m and k are both linearly proportional to n. There are LDPC code designs which are structured in such as way as to have linear order ($O(n)$) encoders, but these are outside the scope of this work.

4.2.1 Regular LDPC Codes

In this work, we use a common class of regular LDPC codes of rate 1/2 which has three parity-check nodes per bit and six bits per parity-check node(a (3,6)-regular LDPC code). The codes used are created by pseudo-randomly generating a parity-check matrix with constrained row and column weights. This can be done by randomly generating columns of weight three and then making each row weight six by shifting bits along the columns or vice-versa.

[3]By equivalent we mean that they can be made to be strictly-systematic via. a permutation of the columns which results in a different codebook, but does not change any of the other properties of the code.

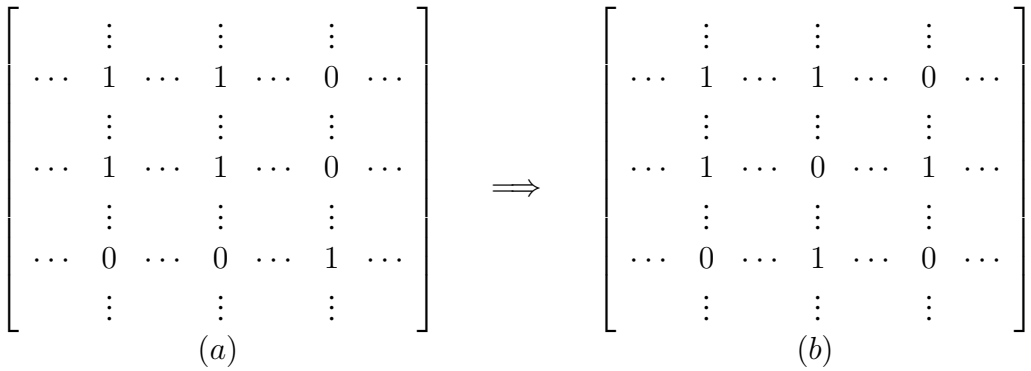

Figure 4.2: Parity-check matrix with a 4-cycle (a) and then with it removed (b).

After generating a random matrix, cycles of length 4 are removed by further shifting of the bits within the columns (see Fig. 4.2). Cycles of length 4 are the shortest possible cycles in the Tanner graph and represent the highest dependency between messages. They are removed so that the messages passed by the SPA are less dependent.

The SPA would compute the bitwise MAP decoding rule exactly if it were not for the cycles in the Tanner graph. Unfortunately, as mentioned previously, codes designed without any cycles are not good codes. It is widely believed that short cycles in particular, negatively affect the convergence of the SPA towards a near-optimal solution.[4]

Removing the shortest cycles, the length 4 cycles, is quite simple using an iterative technique that first looks for 4-cycles, then shifts the bits in the column to break them up and then checks again to see if any new 4-cycles were created. This method usually works after only a few iterations. If it does not succeed after a maximum number of attempts a new parity-check matrix can be generated that will hopefully be more accommodating. An example of a removing 4-cycles is shown in Fig. 4.2.

It is possible to remove cycles of length 6 and higher, however the recursion is becomes

[4]The exact nature of the relationship between cycles in the graph and the performance of the SPA is still an open problem.

more complex with the length of the cycles and it becomes more likely that we will simply create new cycles in the process of removing them. If one wants LDPC codes with girth larger than 6 it is possible to structure the LDPC codes in such a way to accomplish this. We do not explore these LDPC code designs in this work.

Once the parity-check matrix $\mathbf{H}$ has been created, the generator matrix is computed by finding a strictly-systematic matrix $\mathbf{G}$ which solves $\mathbf{HG} = \mathbf{0}_{m\times k}$. We can do this using the following decomposition of H

$$\begin{aligned} \mathbf{H} &= [\mathbf{C}_1|\mathbf{C}_2], \\ \mathbf{G} &= \left[\frac{\mathbf{I}_k}{\mathbf{A}}\right], \\ \mathbf{HG} &= [\mathbf{C}_1 \oplus \mathbf{C}_2\mathbf{A}], \\ \therefore \mathbf{A} &= \mathbf{C}_2^{-1}\mathbf{C}_1, \end{aligned}$$

where $\mathbf{C}_1$ is an $m \times k$ matrix and $\mathbf{C}_2$ is a $m \times m$ invertible matrix. To compute the generator matrix we simply need to compute $\mathbf{C}_2^{-1}$.

4.2.2 Irregular LDPC Codes

Irregular LDPC codes are also generated pseudo-randomly, but in order to generate irregular LDPC codes we must use a degree distribution to determine the individual row and column weights of the matrix to be generated. The degree distribution of an irregular LDPC code determines the proportion of edges in the Tanner graph connected to check nodes and variable nodes of a particular degree. Equivalently, it can be represented by the number of check nodes and variable nodes of a particular degree.

Finding 'good' degree distributions is very hard. They must be tested experimentally to determine if they perform well (in particular if they perform better than a regular LDPC

code). Since there is an effectively infinite number of possible degree distributions, we require some means of direct analysis. Fortunately, techniques such as density evolution [27] (DE) offer a reduced complexity method to determine the performance limit of a degree distribution.

The analysis performed by DE determines what the 'worst' channel for which the SPA decoder will decode with an arbitrarily small probability of error in the limit of long block lengths and using a large number of iterations of the SPA to decode. DE is based on the average performance of an ensemble of LDPC codes of a particular degree distribution. One can then state with certainty that there exists a code in this ensemble which outperforms the average. For example, DE can be used to determine what is the highest CBEP for a BSC for which the average performance of the ensemble goes to zero as the block length of the codes in the ensemble go to infinity and the number of SPA iterations is taken to be very large.

In [27] the authors prove that DE determines the worst channel for which a code using a particular degree distribution can achieve an arbitrarily low probability of error under SPA decoding. For any channel worse than this 'limit' the probability of error will be bounded away from zero. This is a sort of 'channel capacity' for the LDPC/SPA encoder/decoder. The difference between this and Shannon's capacity is that Shannon gives us the performance limit of the best channel code which assumes optimal decoding and as we have stated earlier the SPA is sub-optimal.

Using this technique the authors were able to search for good degree distributions (those that maximized the DE 'capacity' for the BSC and AWGN channels) [26]. DE has also been applied to an example of the GEC channel in [7] although not as extensively.

This work does not present any density evolution techniques to find good degree distributions for the QBC or the GEC. Instead, we have used results from [26] and [7] in our

simulations to see how degree distributions designed for one type of channel perform on another.

In order to generate the irregular code, we need the number of nodes of each particular degree; however, in the literature the proportion of edges is generally used for the degree distribution. We compute the number of nodes using the following equations from Section 3.2:

$$m = n\frac{\sum_{j=2}^{d_c}\rho_j/j}{\sum_{j=2}^{d_v}\lambda_j/j}, \tag{4.1}$$

$$V_i = n\frac{\lambda_i/i}{\sum_{j=2}^{d_v}\lambda_j/j}, \tag{4.2}$$

$$C_i = m\frac{\rho_i/i}{\sum_{j=2}^{d_c}\rho_j/j}, \tag{4.3}$$

where ρ_i and λ_i are the proportion of edges connected to check nodes and variable nodes of degree i, respectively, d_v and d_c are the maximum degree for variable and check nodes respectively, n is the length of the code, m the number of parity-checks and V_i and C_i are the number of variable and check nodes of degree i, respectively which are the quantities we use to generate the parity-check matrix.

The n columns are generated randomly with weights according to the degree distribution and (4.2) and then arranged in random order. Then using the degree distribution for the rows and (4.3), the row weights are placed in a random ordering. Then the row weights are applied to the existing matrix by shifting bits within the columns (preserving the column weights) so that the row weights of the matrix match the ordering. This results in a random parity-check matrix with row and column weights determined by the degree distribution

given.

At this point, the 4-cycles can be removed by shifting bits in the columns just as before. This technique preserves both the row and column weights and does not affect the degree distribution. Interestingly, no significant gain in performance was achieved by doing this. In fact, performance appeared to be very slightly worse for the codes tested, so 4-cycles were not removed for the irregular code simulations.

4.3 SPA Decoder Design

As mentioned in Section 3.2, the SPA decoder is an iterative decoder design that performs probability propagation on factor graphs. It can be applied to the belief propagation decoding of convolutional codes, Turbo codes and LDPC codes as a linear time decoding algorithm. Furthermore, the SPA decoder can be extended to consider the channel state model for channels with finite-state Markov memory.

Here, we consider the message passing rules used by the decoder for passing probabilities along the Tanner factor graph and the Markov factor graph for the channel state.

4.3.1 Tanner Graph Messages (Decoding)

The messages update rules for the Tanner graph have been given in Section 3.2; below we derive the equations for SPA decoding from those message update rules.

First we define the messages passed as $\mu_{x_i \to h_j}(x_i) = \mu_{h_j \to x_i}(x_i) = \hat{\Pr}(x_i)$, the current estimate for the probability distribution for x_i. Since $x_i \in \{0, 1\}$, we can send a single

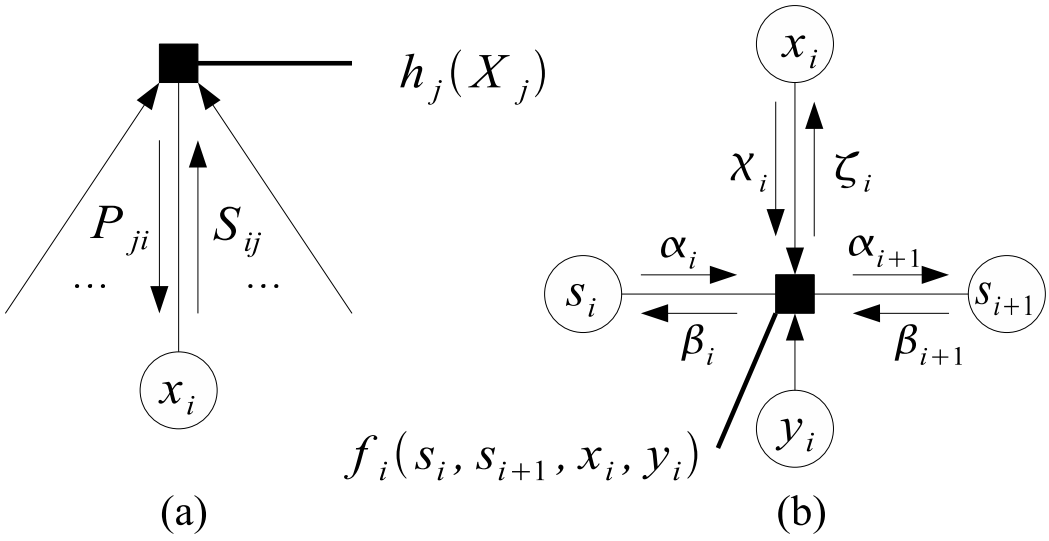

Figure 4.3: Graphs showing the messages and factor nodes associated with the parity-check factors (a) and the channel factors (b).

value in place of the distribution message. For this we use the likelihood ratio (LR):

$$\begin{aligned} S_{ij} &\triangleq \mu_{x_i \to h_j}(0)/\mu_{x_i \to h_j}(1), \\ P_{ji} &\triangleq \mu_{h_j \to x_i}(0)/\mu_{h_j \to x_i}(1), \end{aligned}$$

where S_{ij} is the message passed from the code-bit variables to the parity-check factors and P_{ij} is the message passed from the parity-check factors to the code-bit variables.

The variable node for x_i receives messages from its neighbouring parity-check nodes and computes the outgoing message to the parity-check node for h_j as

$$\begin{aligned} \mu_{x_i \to h_j}(x_i) &= \prod_{k \in n(x_i) \setminus \{j\}} \mu_{h_k \to x_i}(x_i), \\ \Rightarrow S_{ij} &= \prod_{k \in n(x_i) \setminus \{j\}} P_{ki}, \end{aligned} \tag{4.4}$$

where $n(x_i)$ is the set of indicies for the parity-check nodes connected to x_i.

The message update equation for the parity-check nodes is more complicated because we need to take the sum of a product-of-functions involving several variables over all of the combinations of those variables which satisfy the parity-check condition. This can be simplified considerably using the likelihood difference (LD):

$$\begin{aligned} S_{ij}^{LD} &\triangleq \mu_{x_i \to h_j}(0) - \mu_{x_i \to h_j}(1), \\ P_{ji}^{LD} &\triangleq \mu_{h_j \to x_i}(0) - \mu_{h_j \to x_i}(1), \end{aligned}$$

and further noting that

$$\begin{aligned} S_{ij}^{LD} &= 1 - 2(1 + S_{ij})^{-1}, \\ P_{ji}^{LD} &= 1 - 2(1 + P_{ji})^{-1}. \end{aligned}$$

Thus, we can use

$$\begin{aligned} \mu_{h_j \to x_i}(x_i) &= \sum_{X_j} h_j(X_j) \prod_{x_k \in X_j \backslash \{x_i\}} \mu_{x_k \to h_j}(x_k), \\ \Rightarrow \mu_{h_j \to x_i}(0) - \mu_{h_j \to x_i}(1) &= \prod_{k \in X_j \backslash \{i\}} \left(\mu_{x_k \to h_j}(0) - \mu_{x_k \to h_j}(1) \right), \\ \Rightarrow P_{ji} &= \frac{1 + \prod_{k \in X_j \backslash \{i\}} \left(1 - 2(1 + S_{ki})^{-1}\right)}{1 - \prod_{k \in X_j \backslash \{i\}} \left(1 - 2(1 + S_{ki})^{-1}\right)}, \end{aligned} \tag{4.5}$$

where $h_j(\cdot)$ is the identity function for the j^{th} parity-check equation and X_j is the set of variables corresponding to the code bits that participate in the j^{th} parity-check. We note that the second step comes from observing that all the negative terms in the product correspond to an odd number of 1's in the set $X_j \backslash \{x_i\}$ (implying $x_i = 1$) and all the positive terms correspond to an even number of 1's (implying $x_i = 0$).

Another way of representing the messages passed is with log-likelihood ratios (LLRs),

which uses the log of the LRs: $S_{ij}^{LLR} \triangleq \ln(S_{ij})$ and $P_{ji}^{LLR} \triangleq \ln(P_{ji})$. This simplifies the multiplication of (4.4) to a sum. Unfortunately, (4.5) becomes much more complex, since we must compute the hyperbolic tangent. When running computer simulations, any gain in performance obtained by changing products into sums is negated by the operations needed to compute the hyperbolic tangent function.

One way to simplify these calculations further is by applying the min-sum update rule. This is based on using LLRs and an approximation of the check function involving the relation: $\ln(\cosh(x)) \approx |x| - \ln(2)$, for $x \gg 1$. The update equations become

$$\begin{aligned} S_{ij}^{LLR} &= \sum_{k \in n(x_i) \backslash \{j\}} P_{ki}^{LLR}, \\ P_{ji}^{LLR} &= \min_{k \in X_j \backslash \{i\}} (S_{kj}^{LLR}) \left(\prod_{k \in X_j \backslash \{i\}} \operatorname{sgn}(S_{kj}^{LLR}) \right), \end{aligned}$$

where

$$\operatorname{sgn}(x) = \begin{cases} 1 & if\ x \geq 0 \\ -1 & if\ x < 0 \end{cases}.$$

The min-sum algorithm trades multiplication operations with summation and comparison operations. While the min-sum update rule is not as effective as the sum-product update rule, the cost of slightly reduced bit-error rate performance is often worth the significant reduction in the computational complexity of the algorithm. This can be used in applications where computational complexity is more important than achieving the best possible performance. While we do not use the min-sum rule in any of our simulations it is important to not that simplifications to the SPA are possible in order to achieve reduced complexity.

4.3.2 Markov Graph Messages (Estimation)

The Markov chain graph models the relationship between the channel state transition sequence and the channel noise sequence. The relationship of the factor node of the joint probability between the variables s_{i+1}, s_i, x_i and y_i is given by

$$f_i(s_{i+1}, s_i, x_i, y_i) = \Pr(s_{i+1}|s_i)\Pr(y_i|x_i, s_{i+1}, s_i).$$

The message update rules related to the Markov graph are

$$\begin{aligned}
\mu_{x_i \to f_i}(x_i) &= \prod_{j \in n(x_i)} \mu_{h_j \to x_i}(x_i), \\
\mu_{f_i \to s_{i+1}}(s_{i+1}) &= \sum_{s_i, x_i} f_i(s_{i+1}, s_i, x_i, y_i)\mu_{s_i \to f_i}(s_i)\mu_{x_i \to f_i}(x_i), \\
\mu_{f_i \to s_i}(s_i) &= \sum_{s_{i+1}, x_i} f_i(s_{i+1}, s_i, x_i, y_i)\mu_{s_{i+1} \to f_i}(s_i)\mu_{x_i \to f_i}(x_i).
\end{aligned}$$

The variable nodes on the Markov graph simply pass on the messages they receive since they are of degree two. Using the likelihood ratios of the previous section we defined the extrinsic message χ_i, as the likelihood ratio passed from the Tanner graph to the Markov graph this is computed based on (3.19):

$$\chi_i = \prod_{j \in n(x_i)} P_{ji}.$$

This ratio is also used for the decision rule on each bit when determining the current decoder estimate for the codeword.

We define the messages α_i and β_i as

$$\alpha_i = \begin{bmatrix} \mu_{s_i \to f_i}(0) \\ \mu_{s_i \to f_i}(1) \\ \vdots \\ \mu_{s_i \to f_i}(s_{\max}) \end{bmatrix},$$

where $s_{\max}$ is the last state. β_i is defined similarly, with $\mu_{s_i \to f_{i-1}}(s_i)$ used for the entries.

The message update rules from (3.20) and (3.21) can be computed as:

$$\begin{aligned} \alpha_{i+1}(s_{i+1}) &= \sum_{s_i \in \mathcal{S}, x_i \in \{0,1\}} \Pr(s_{i+1}|s_i) \Pr(y_i|s_{i+1}, s_i, x_i) \alpha_i(s_i) \psi_{x_i}(\chi_i), \\ \beta_i(s_i) &= \sum_{s_{i+1} \in \mathcal{S}, x_i \in \{0,1\}} \Pr(s_{i+1}|s_i) \Pr(y_i|s_{i+1}, s_i, x_i) \beta_{i+1}(s_{i+1}) \psi_{x_i}(\chi_i), \end{aligned}$$

where $\psi_{x_i}(\chi_i) = \left(1 + \chi_i^{x_i - 1}\right)^{-1} = \mu_{x_i \to f_i}(x_i)$ and $\mathcal{S}$ is the set of state indexes for the channel ($\mathcal{S} = \{0, \ldots, s_{\max}\}$). The functions $\alpha(s)$ and $\beta(s)$ correspond to the entries in α and β corresponding to state s.

We can write this as a matrix multiplication by defining matrices $\mathbf{E}$ and $\mathbf{C}$ to correspond to the terms associated with errors ($y_i \neq x_i$) and valid transmissions ($y_i = x_i$). Since y_i is a known variable, this is equivalent to the terms in which $x_i = 0$ and $x_i = 1$. Let $e_i = y_i \oplus x_i$, then define the $(j, k)^{th}$ entry of the matrices $\mathbf{E}$ and $\mathbf{C}$ as follows:

$$\begin{aligned} e_{jk} &= \Pr(s_{i+1}|s_i) \Pr(x_i \neq y_i|s_{i+1}, s_i, y_i), \\ c_{jk} &= \Pr(s_{i+1}|s_i) \Pr(x_i = y_i|s_{i+1}, s_i, y_i). \end{aligned}$$

The matrices are defined for the GEC as

$$\begin{aligned} e_{jk} &= P_{jk}P_j, \\ c_{jk} &= P_{jk}(1-P_j), \end{aligned}$$

where P_{jk} is the one-step channel transition probability from state j to state k and P_j is the probability of error for state j (either P_g or P_b).

For the QBC and in general for binary channels with additive Markov noise, the matrices are

$$\begin{aligned} e_{jk} &= \begin{cases} P_{jk} & \text{if } k \geq 2^{M-1}, \\ 0 & otherwise, \end{cases} \\ c_{jk} &= \begin{cases} P_{jk} & \text{if } k < 2^{M-1}, \\ 0 & otherwise. \end{cases} \end{aligned}$$

Using these matrices, the update rules become simplified. First we define

$$P_e(\chi_i, y_i) = \left(1+\chi_i^{-y_i}\right)^{-1},$$

as the probability y_i is in error based on the extrinsic message χ_i. Then the forward message, backward message and the return message ζ_i, are computed using matrix and vector multiplication as

$$\begin{aligned} \alpha_{i+1} &= [\mathbf{C}^T(1-P^e(\chi_i,y_i)) + \mathbf{E}^T P^e(\chi_i,y_i)]\alpha_i, \\ \beta_i &= [\mathbf{C}(1-P^e(\chi_i,y_i)) + \mathbf{E}P^e(\chi_i,y_i)]\beta_{i+1}, \\ \zeta_i &= \left(\frac{\alpha_i \mathbf{C}\beta_{i+1}}{\alpha_i \mathbf{E}\beta_{i+1}}\right)^{-y_i}. \end{aligned}$$

We note that the matricies are quite sparse for the QBC and BAMNC since there are only two entries per row. For a more general Markov process this will not always be the case so using BAMNCs allows us to compute these equations in linear order with respect to the number of states, where it would be a squared order algorithm in general.

4.4 Channel Model Simulations

The channel models used for simulation are the QBC and the GEC. For both channels, we simply generate $U = \mathrm{Uniform}(0, 1)$ pseudo-random numbers to determine state-transition and error events according to their probability distribution.

For the GEC there are two events at each channel use: the state transition and the noise symbol value. First we determine the noise symbol value based on the state s_i and the probability of error for that state P_g, if it is the 'good' state and P_b, if it is the bad state. In other words, we generate U_1, and if $U_1 < P_{s_i}$, where P_{s_i} is the probability of error in the state s_i at time i, then an error occurs. Then we generate a second pseudo-random number U_2, to determine if a state transition occurs. If we are in the 'good' state then state transition to the 'bad' state occurs only if $U_2 < g$. If we are in the 'bad' state, then state transition to the good state occurs if $U_2 < b$: otherwise the state remains unchanged.

The QBC model is more complex because it can have many more than two states. Fortunately, for each state there are only two possible events: a state transition corresponding to an error event or a state transition corresponding to the no-error event. Using the one-step state transition probabilities we can determine p_{s_i} for each state (probability of error/error state transition) and if $U_1 < p_{s_i}$ an error event occurs, otherwise no error occurs.

Alternatively, it is possible to use the QBC process described in Chapter 2 and generate separate random variables to determine if we choose the queue or BSC process and then to

determine which entry from the queue is chosen or whether or not an error occurs according to the BSC process. This method is a bit more complex. Furthermore, the former method is universal and will work for any binary additive M^{th}-order Markov channel where the one-step transition probabilities have been defined.

4.5 Step-by-Step

The simulation process proceeds in three main parts: encoding, transmission and decoding. Encoding and transmission are very simple processes, while the SPA decoder is clearly more involved.

Encoding:

- Generate a set of uniformly distributed pseudo-random source bits to encode.
- Encode the source bits one block at a time (k bits at a time) through matrix multiplication with the generator matrix.
- The output result is the encoded bit sequence.

Transmission:

- Take encoded bits one at a time to simulate transmission.
- Determine the noise symbol for each bit transmission according to the noise process using pseudo-random number generation.
- Output the modulo-2 sum of the input bit and the noise symbol.

Decoding:

1. Initialize the decoder:

   (a) Set each message in the Tanner graph based on the bit error rate of the channel and the received bit.

   (b) Set each message in the Markov graph to the stationary distribution of the channel state.

2. Perform an iteration on the Tanner graph:

   (a) Pass messages from the variable to the factor nodes and back.

   (b) Compute the new variable node messages.

   (c) Compute the extrinsic message to be passed to the Markov graph.

3. Check exit conditions:

   (a) Use the extrinsic message to get an estimate for each bit and determine if this forms a valid codeword by computing the parity-check. Stop decoding and return codeword if it is valid.

   (b) Check if the number of iterations performed is greater than the maximum number of iterations set for the algorithm. If so, stop decoding and report that decoding failed.

4. Perform an iteration on the Markov graph (forward-backward):

   (a) Starting with the state at time zero, pass messages forward. The last forward message becomes first (time n) backward message.

   (b) Starting with the state at time n, pass messages backward. The last backward message becomes the first (time zero) forward message.

(c) For each bit compute the channel message from the prior forward message and posterior backward message and pass it to the Tanner graph.

5. Return to step 2 and repeat.

4.6 Algorithm Parallelization

One of the major advantages of the SPA is that it is a completely parallelizable design. A multi-processor system can, in theory, be designed so that each node in the graph has a single processor computing the update rules and passing the messages. We will generally not have that many processors so we suggest a system based on computing the nodes N at a time.[5]

A multi-processor system with N processors can be utilized so that parity-check update messages are computed N at a time and then variable update messages are computed N at a time. The forward-backward scheduling cannot be parallelized as we have described it; however, it is possible to change the scheduling without affecting the long run calculations of the channel messages. The channel factor node messages can be computed in parallel, N at a time. The messages would no longer pass updated values serially from beginning to end and back again. Instead, they would use the existing messages from the last iteration for computing the updates and thus, would be able to compute the messages in parallel.

Theoretically, using this scheduling the speed of the algorithm should scale linearly with the number of processors used, up to a maximum of m processors for the parity-check updates and n processors for the variable and channel updates.

[5]Assuming $N < m, n$.

Chapter 5

Results

In this section we present the results from several simulations designed to analyze the performance of the SPA decoder design outlined in the previous chapter. We compare memoryless decoding using the BSC as well as decoding for the GEC using a joint estimator-decoder designed for the GEC. We compare these with the performance of the QBC decoder using both regular LDPC codes and irregular LDPC codes designed for channels other than the QBC.

The purpose of comparing with the BSC is to see how much BER performance is improved by taking the channel memory into consideration. This is as opposed to using an interleaver to make the channel noise appear memoryless. Furthermore, we want to know if the gap between simulated performance and the Shannon limit of the QBC is comparable to that for the BSC. We will see that for regular LDPC codes the performance gap to the Shannon limit remains relatively constant regardless of the channel used.

We make comparisons with the GEC using the results from [36]. There the authors took several different parametrizations of the GEC and produced parametrizations of the QBC in which they minimized the divergence rate between the two channels. As a re-

sult, they obtain QBC channels that are statistically similar to the GEC for several different parametrizations. We take two of these examples, which are suitable for simulation using a rate 1/2 code, to obtain simulated results to compare these models. In particular, we wanted to examine how much the performance degrades if we use one model to decode data transmitted over the other model (i.e., using the QBC decoder to decode data sent over the simulated GEC and vice versa). We see that while performance is degraded by the channel/decoder mismatch when compared with matched channel and decoder, the performance is still significantly better than the memoryless strategy assuming an ideal interleaver.

Lastly, we use irregular LDPC codes with the QBC decoder. In [26] and [7], irregular codes are designed for the AWGN channel and the GEC, respectively. In both cases, the authors used density evolution to find these codes. We test these codes with the QBC to see the effect that channel mismatch between the channel used for code design and the channel used in the simulation has on performance. The choice of channel used for design of irregular codes has a very significant effect on performance particularly when the effect of the channel memory is strong. In fact, an irregular LDPC code which significantly improves performance relative to the regular LDPC code on one channel, may actually show degraded performance on another channel.

5.1 Performance of the QBC vs. BSC

To begin analysis of the performance of the joint estimator-decoder for the QBC, we have compared the results of three different QBCs with those obtained by ordinary SPA decoding for the BSC. The QBCs have varying channel correlation to examine channels with significantly different memory effects. We wanted to see if performance of the LDPC codes scaled with the capacity of the channels used; so two of the channels have significantly

Channel	ε	α	M	Cor
QBC 1	0.26	1.0	4	0.08
QBC 2	0.625	1.5	2	0.4
QBC 3	0.8	1.0	4	0.5

Table 5.1: Table of parameters for the three QBC channels used in the simulations shown in Fig. 5.1

higher capacity than the BSC for the same CBER.

The channel parameters for the three QBCs are given in Table 5.1. The first QBC is a UQBC (i.e., a FMCC) with a very small correlation and thus its capacity is not significantly different than the BSC. The effect of memory for this channel is quite weak since it has a very small correlation value. In the second channel, we use a smaller queue of length two with a larger ε, resulting in a significantly increased correlation. Furthermore, the dependency of the next noise symbol on the last queue entry is larger than for the first entry due to a higher α. The last channel is again a UQBC with queue length four and a value of ε, which is the highest of the three channels. This channel has the highest correlation and capacity of the three. The QBCs are numbered in order of increasing capacity for the same CBER allowing us to examine how the simulated performance scales with channel capacity.

In Fig. 5.1, we present the results from simulations performed on the three QBCs and the BSC. The code used was a randomly generated regular-(3,6) LDPC code of length 100,000 bits. The decoder performed a maximum of 200 iterations per block and simulations were run using up to 10,000 blocks (1,000,000,000 bits), though simulations were terminated early once a sufficient number of bits had been sent to produce accurate results (as a rule of thumb, in order to measure a Bernoulli statistic on the order of 10^{-x} one should send at least $10^{(x+2)}$ bits; we use up to $10^{(x+3)}$ bits).

The Shannon limit curve shown is the rate-distortion Shannon limit (RDSL) and shows

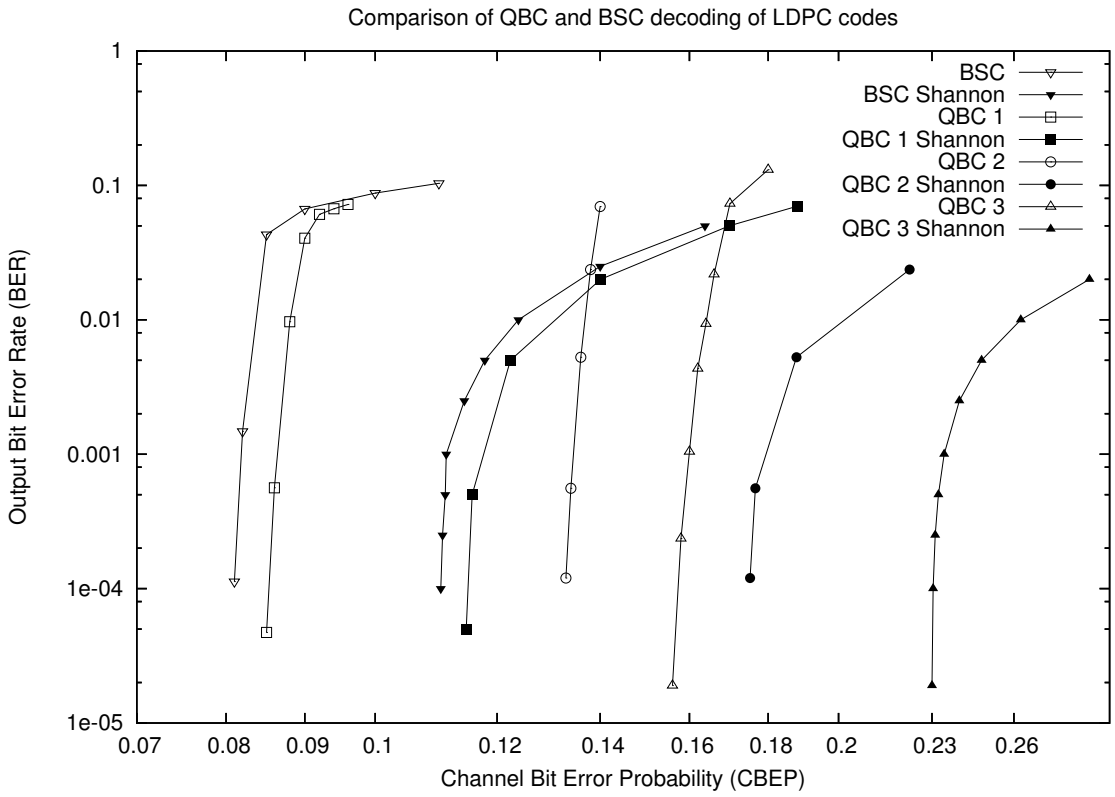

Figure 5.1: Comparison of the QBCs from Table 5.1 and the BSC. For these simulations a length 10^5, rate-1/2, (3,6)-regular code was used with a maximum of 200 iterations for decoding.

the best CBER achievable at the plotted output BER by a rate-1/2 code. In each case the RDSL was computed using the method described in Section 2.1.4.

The main result that we observe is that the performance of the decoder scales well with the Shannon limit of the channels. This shows that for channels with memory, that have considerably higher capacity than the corresponding memoryless channel, significant performance gains are achievable using the joint estimation/decoding scheme relative to the interleaved decoding scheme without state-estimation.

The gap between the Shannon limit and the simulated performance (measured at $\sim 10^{-4}$ BER) is quite consistent for each channel. For the BSC performance is 73% of Shannon limit, for QBC 1 it is 74%, QBC 2 it is 76% and QBC 3 it is 68%. By comparing performance as a percentage of the Shannon limit, we can compare the gap relative to the limit for channels with different capacities. This is similar to comparing the Shannon limit gap in decibels for channels such as the AWGN channel where signal and noise power are given in decibels.

This indicates that for regular LDPC codes, we obtain nearly the same performance across all the channels. The effectiveness of these codes appear to be universal for all the channels we have tested. This appears to be a property of regular-LDPC codes because as we will see in Section 5.4, a different behaviour is observed for irregular LDPC codes.

5.2 Using the QBC to Model GEC Statistics

While the QBC and GECs are in general very different channel models, it is demonstrated in [36] that for certain parametrizations there exist QBCs and GECs with similar block transition distributions. This fact can be exploited to produce QBC parametrizations that closely model or 'match' a particular parametrization of the GEC. This also suggests that

the QBC could be used to model other more general channel models such as Rayleigh or Rician correlated fading channels.[1]

In [36], the authors derive a number of these channel matchings between the GEC and QBC by minimizing the Kullback-Leibler divergence rate between the channels. They demonstrate numerically that the capacity of the two channels is nearly the same. Using the decoding scheme we have developed for the QBC and the decoding scheme for the GEC from [6, 12, 25], we simulate the transmission of data over one model, coupled with the decoder for the other. In other words, we mismatch the channel and the decoder to give us a good idea of how practical these channel matchings are in terms of decoder performance.

In Table 5.2, we study two GECs and their corresponding 'matched' QBC. The match is obtained by searching for a parametrization of the QBC that minimizes the divergence of the two channels. The CBEP for the GEC is (like the QBC) the long-run proportion of errors produced by the channel and can be obatined from

$$CBEP \;=\; \frac{g}{b+g}P_g + \frac{b}{b+g}P_b.$$

We can see from this table that while the performance of each decoder when operating on the other channel is worse than when it is operating on its own channel (which we expect), the performance is still much better than the performance of the memoryless (interleaved) channel using the BSC model. Thus, channel matching by minimizing the divergence between the two channels is an effective technique for exploiting channel memory. We are capable of producing a match that performs better than the alternative of interleaving the data and assuming the channel is memoryless.

[1] Modeling correlated Rayleigh and Rician fading using Markov models is the subject of [23]. They also explore modeling using general BAMNCs and the GEC.

GEC Channel Parameters:

GEC	CBEP	P_g	P_b	g	b
Exp. 1	0.09	0.0519	0.6118	0.0450	0.0033
Exp. 2	0.08	0.0439	0.5746	0.0450	0.0033

QBC Channel Parameters

QBC	CBEP	ε	α	p	M
Exp. 1	0.09	0.5705	0.4168	0.0900	5
Exp. 2	0.08	0.5711	0.4312	0.0800	5

Results:

	GEC	QBC	GEC w/ QBC Decoder	QBC w/ GEC Decoder	BSC
Exp 1	1.4E-05	1.2E-05	2.0E-03	5.0E-04	6.5E-02
Exp 2	$< 10^{-6}$	$< 10^{-6}$	1.2E-05	$< 10^{-6}$	6.5E-03

Table 5.2: Results from the comparison of decoding over two GECs and two QBCs that approximate those GECs. We show performance over the GEC using the QBC decoder and vice-versa compared with the BSC and correctly matched decoders. For these simulations a length 10^5, rate-1/2, (3,6)-regular code was used with a maximum of 200 iterations for decoding.

It should be mentioned that although the GEC decoder appears to decode data transmitted over the QBC better than the QBC decoder does with data transmitted over the GEC, this does not imply that one model is superior. The fact that both results significantly outperform the memoryless channel serve to demonstrate that the match between the two channels is good.

5.3 Using Additive Markov Approximations to Model GEC Statistics

Similar to the concept of matching the QBC to the GEC above, it is possible to use a more general additive Markov noise channel to model other channels such as the GEC. As was proposed in Chapter 2, we can use an additive Markov noise channel which has exactly the same finite-length block distribution and block transition probabilities as the GEC for a particular parametrization.

We do this by computing the length M block stationary distribution for the noise process of the GEC as well as the probability of error given the last M noise symbols for each possible noise sequence. These are the block stationary distribution and state transition probabilities of the length M BAMNC model of this GEC.

So for each length M sequence of channel noise symbols $(e_{i-1}, \ldots, e_{i-M})$ we compute the probability of that sequence occurring for that GEC channel and this gives us the stationary distribution for the BAMNC

$$\begin{aligned} P_{BAMNC}(s_i) &= P_{GEC}(e_{i-1}, \ldots, e_{i-M}), \\ &= P_{GEC}(e_1, \ldots, e_M), \\ &= P_{GEC}(e_1) \cdot P(e_2|e_1) \cdot \cdots \cdot P(e_M|e_1, ..., e_{M-1}). \end{aligned}$$

Channels Parameters:

GEC	CBEP	P_g	P_b	g	b
Exp. 1	0.09	0.0519	0.6118	0.0450	0.0033
Exp. 2	0.08	0.0439	0.5746	0.0450	0.0033

Results:

	GEC	GEC w/ BAMNC Decoder, M=2	GEC w/ BAMNC Decoder, M=4	GEC w/ BAMNC Decoder, M=6	BSC
Exp. 1	1.4E-05	5.047e-03	8.094e-04	1.978e-04	6.5E-02
Exp. 2	$< 10^{-6}$	3.532e-04	5.654e-06	$< 10^{-6}$	6.5e-03

Table 5.3: Results from the comparison of decoding over the GEC using BAMNC approximation decoders versus the GEC decoder. For these simulations a length 10^5, rate-1/2, (3,6)-regular code was used with a maximum of 200 iterations for decoding.

The equations and method used to compute the stationary distribution and state-transition model for the M^{th}-order BAMNC are given in Section 2.2.4.

Using the class of BAMNCs to match the GEC should give us more flexibility than using the QBC since it is ultimately a subset of that class. For those GECs that are modelled well by a finite-length stationary block distribution and state transition distribution we should expect decoding results as good or better than those obtained by the QBC model in the last section.

Our results in Table 5.3 use the GECs from the last section to show how the BAMNC model performs for different memory lengths. We can see that for $M = 4$, the BAMNC outperforms the matched QBC from Table 5.2 which has $M = 5$. Even for $M = 2$, the BAMNC still greatly outperforms the BSC which represents decoding using ideal interleaving.

In Fig. 5.2, we use a similar GEC to that used in Exp. 1 from Table 5.3 to plot the performance of the GEC decoder vs. the BAMNC modeled decoder for various memory lengths. As the memory is increased the performance of the BAMNC improves, and we

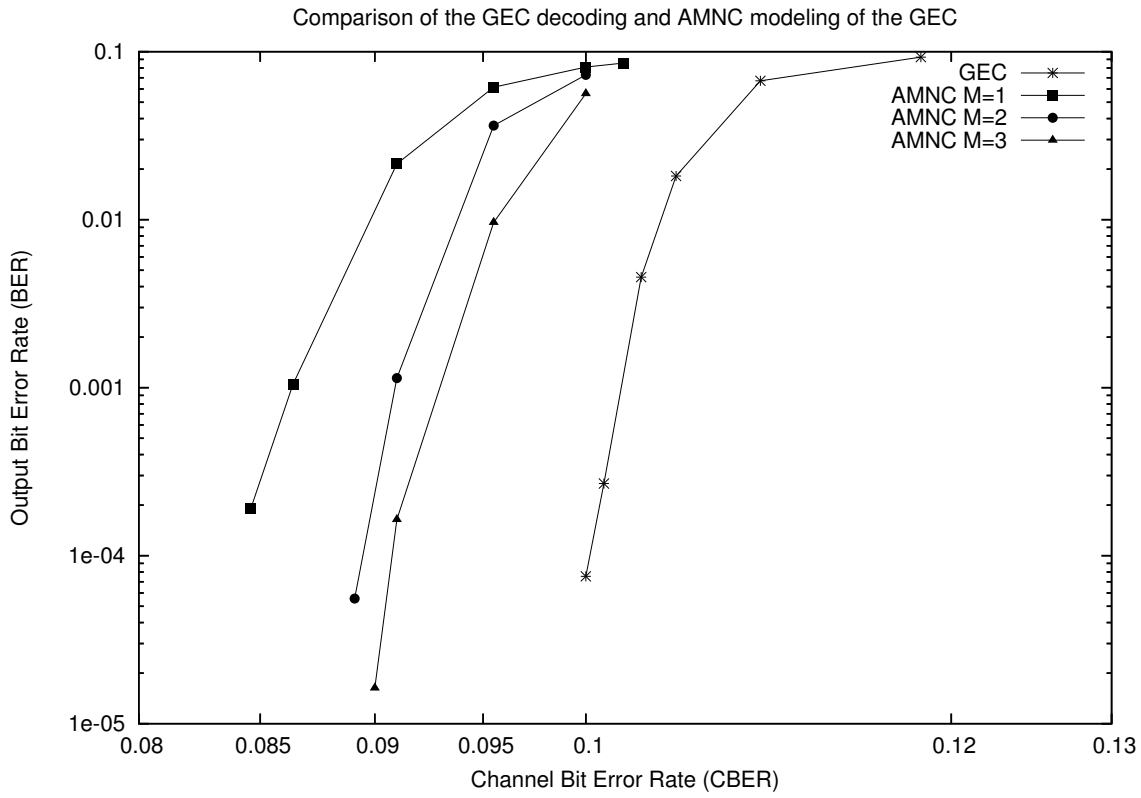

Figure 5.2: BER performance of the BAMNC based decoder ($M = 1, 2, 3$) and the GEC based decoder over a simulated GEC ($P_b = 0.5$, $g = 0.02$, $b = 0.002$) and P_g varies. For these simulations a length 10^5, rate-1/2, (3,6)-regular code was used with a maximum of 200 iterations for decoding.

obtain excellent results even for the relatively short memory lengths shown.

The main drawback to the BAMNC model is its complexity. The BAMNC model used here is a numerical model produced by finite-length analysis of either another channel model, or observed statistics. It requires two parameters for each state to define it, one for the stationary probability of that state and another for the probability of error/state-transition probability for that state. The QBC on the other hand requires only four parameters for any number of states.

Regardless of this fact, the decoder we use is of linear order complexity in the number of states and not the number of parameters used. This means that both a BAMNC and a

QBC of length M will have the same decoder complexity using this finite-state Markov chain modeled decoder. Thus, the BAMNC modeling technique provides a useful method for designing a versatile SPA based decoder for channels with memory, even though it may not be a useful analytical model.

5.4 Performance of Irregular LDPC Codes

Irregular LDPC codes have been shown to significantly outperform regular LDPC codes for a number of channels. In this work we did not develop any techniques for finding good irregular codes for the QBC, since it is very difficult for channels with multiple parameters. Instead, we have examined the performance of irregular codes developed for other channels over the QBC.

Irregular LDPC codes are generally designed with a particular channel model in mind using density evolution (DE) [27]. While the work on designing irregular LDPC codes has focused almost solely on memoryless channels, there has also been some work done for the GEC [6, 7].

The results for regular LDPC codes showed that the performance of the code was not significantly affected by the choice of channel, but remained consistent relative to the capacity of the channel. We are curious if the same will be true for irregular codes, or if the choice of channel model is more important in the design and implementation of irregular codes than it is for regular LDPC codes. We used two different codes for these simulations. The degree distributions for the two codes are given in Table 5.4.

The first set of simulations were run using Code 1 which is taken from [26] and was designed using density evolution over the AWGN channel. This rate-1/2 code shows a significant gain in performance over the (3,6) regular LDPC code for both the AWGN and

Variable Node Degree Distribution:

d_v	λ_2	λ_3	λ_4	λ_5	λ_8	λ_{10}	λ_{12}
Code 1	0.24426	0.25907	0.01054	0.05510	0.01455	0.01275	0.40373
Code 2	0.08	0.63	-	-	0.29	-	-

Parity-Check Node Degree Distribution:

d_c	ρ_6	ρ_7	ρ_8	ρ_9
Code 1	-	0.25475	0.73438	0.01087
Code 2	0.0113	0.9887	-	-

Table 5.4: Variable node (d_v) and parity-check node (d_c) degree distributions for the two irregular codes used in the simulation results below. The two codes generated were of length 10^5 bits

the BSC. In Fig. 5.3, we plot simulations of the BSC and QBC 1 and 2 from Table 5.1. We can see that the performance improvement for the BSC is quite significant. The two QBCs give very different results however. QBC 1 shows performance gains similar to those obtained by the BSC but QBC 2 shows performance degradation relative to the regular-(3,6) code. The irregular code actually performs worse than the regular code on this channel.

The second set of simulations were run using Code 2 which is taken from [7] which is a rate 1/2 code designed using density evolution over the GEC with parameters $P_g = g = b = 0.01$ and P_b varying. In Fig. 5.4, we once again notice a significant improvement over the (3,6) regular LDPC code for both the GEC and QBC 1. However, it is clear that this performance gain is less than it was for QBC 1 using Code 1. The QBC 2 simulation still shows degradation relative to the regular code.

Due to the fact that QBC 1 performed well with both codes we ran a final simulation comparing Code 1 and Code 2 for the BSC, GEC and QBC 1 with results shown in Fig. 5.5.

Clearly it is important that irregular LDPC codes be designed with the channel model intended in mind. However, as can be seen in the comparison of the GEC and the QBC

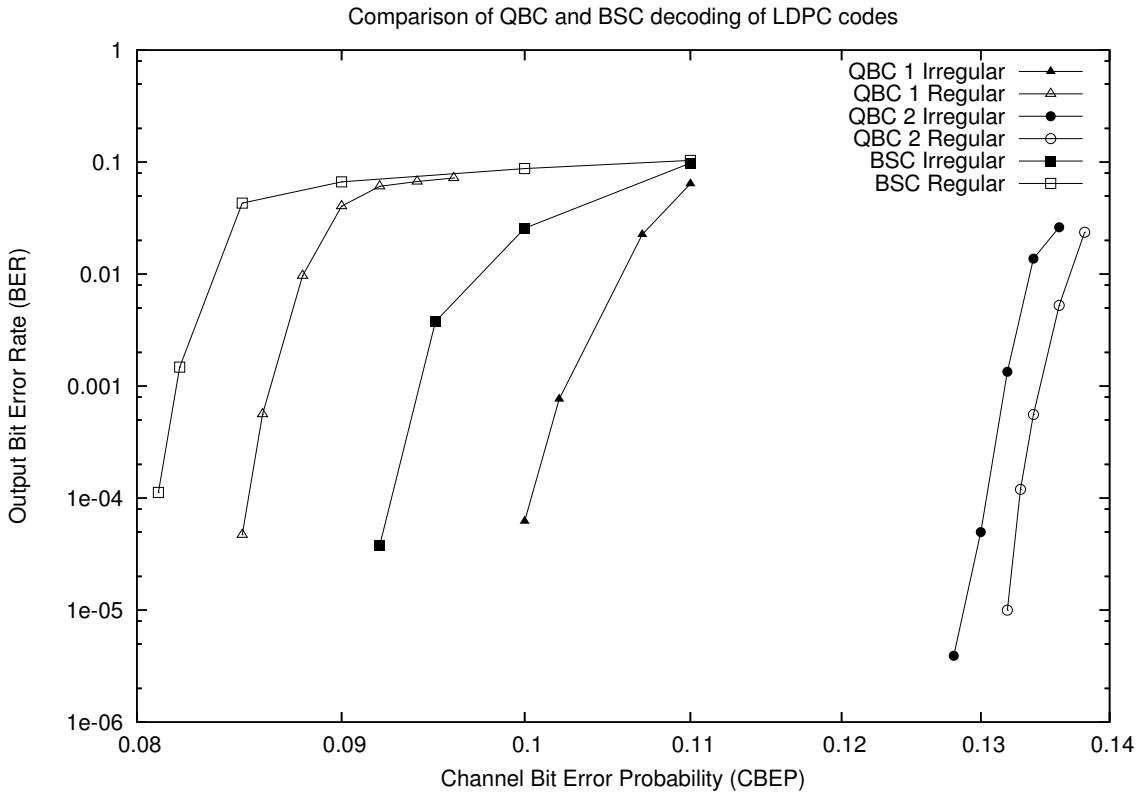

Figure 5.3: Comparison of decoding of the QBC 1 and 2 and the BSC using Code 1 from [26].

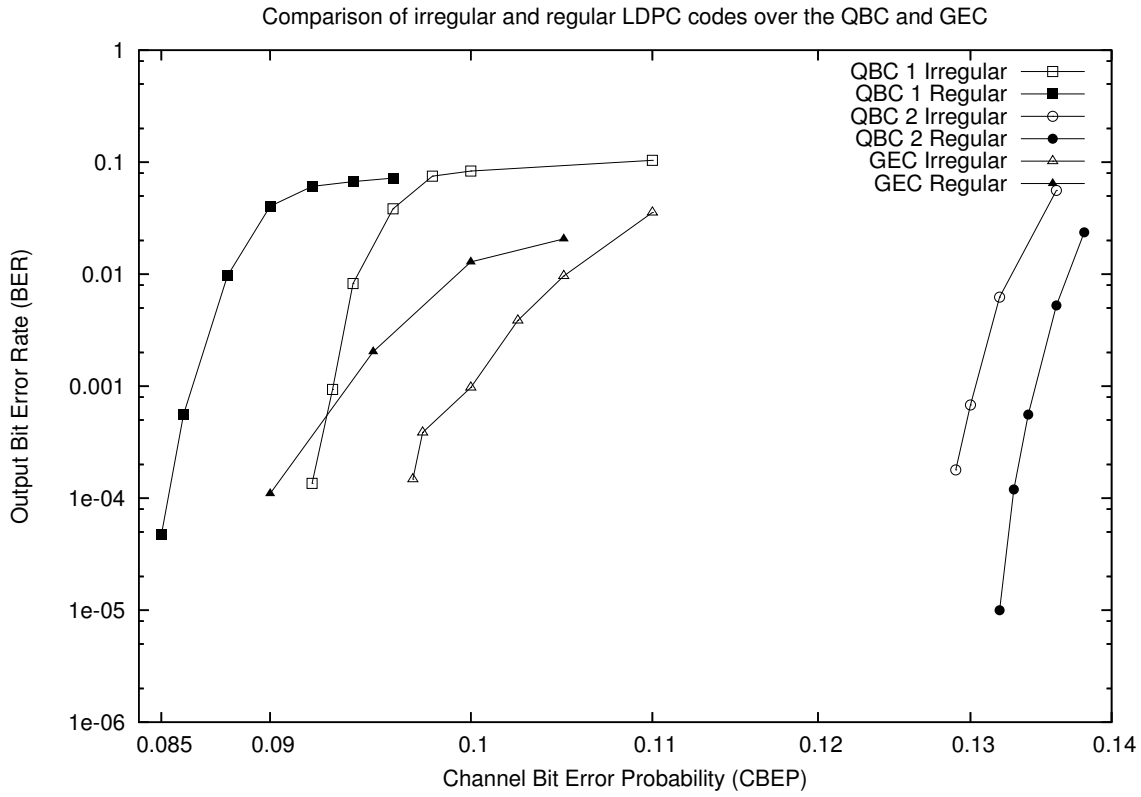

Figure 5.4: Comparison of decoding of the QBC 1 and 2 and a GEC using Code 2 from [7]. GEC parameters are $P_g = g = b = 0.01$ and P_b varies.

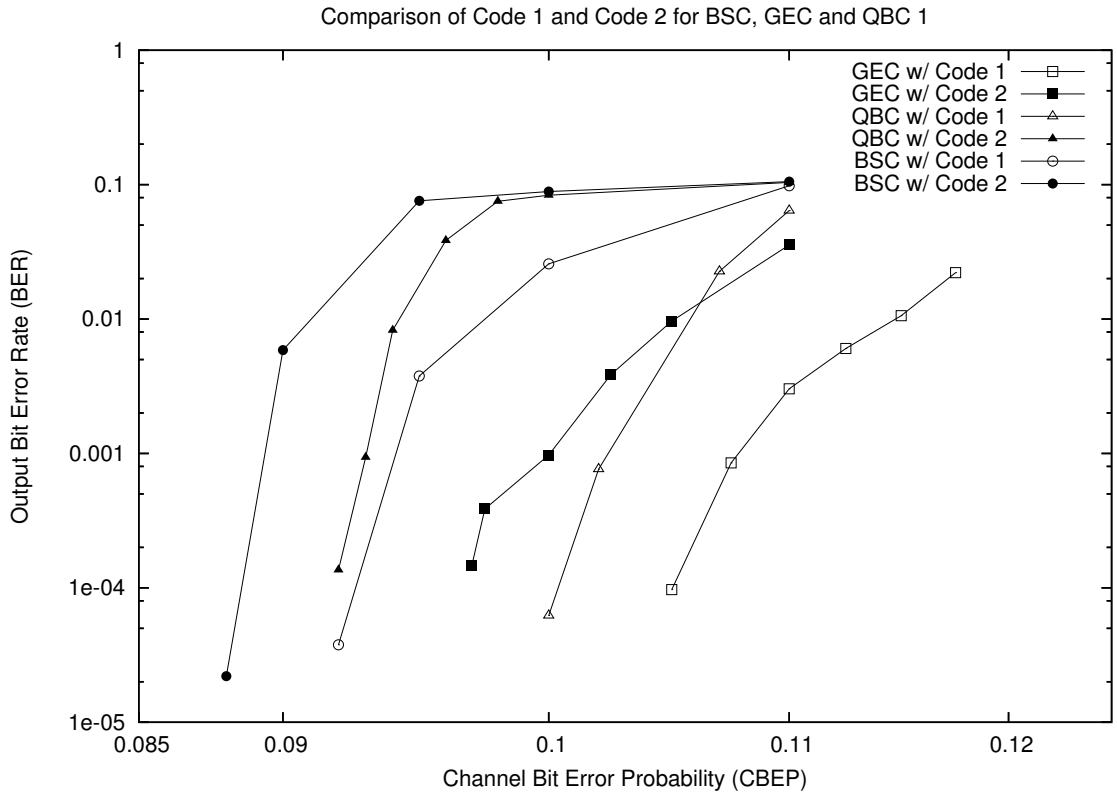

Figure 5.5: Comparison of Code 1 from [26] and Code 2 from [7] on the BSC, GEC and QBC 1. GEC parameters are again $P_g = g = b = 0.01$ and P_b varies.

1, though these models are not statistically similar they both benefit greatly from the same irregular code designed for that GEC (Code 2). Interestingly, the QBC 1 also benefited greatly from the irregular code designed for the AWGN channel which is a completely memoryless channel.

The QBC 1 which performed well using both codes is, in fact, a nearly memoryless channel with very weak correlation. Furthermore, the GECs used also has very weak correlation. Thus, in both cases these channels are nearly memoryless channels to begin with.

Due to the nearly memoryless nature of QBC 1, it is not that surprising that it performs well on a code designed for a memoryless channel. Furthermore, since the GEC used in Fig. 5.4 is also a nearly memoryless channel, then it makes sense that a code designed for this channel is essentially a code designed for a nearly memoryless channel. Therefore, we would also expect that it performs well when used on other nearly memoryless or memoryless channels.

The results we have obtained seem to suggest that irregular codes designed for memoryless or nearly memoryless channels perform well and in fact, nearly equally well on other channels which are memoryless or nearly memoryless. The comparison of the two codes Fig. 5.5 further suggest that even though Code 1 was not designed for either QBC 1 and the GEC that it is a better code overall for all three channels than Code 2, likely due to the fact that it is a more complex degree distribution. It is clear however that the GEC used will achieve good results with codes designed for memoryless channels.

Unfortunately, these results do not tell us much about the performance of irregular codes for channel which do have significant memory effects such as QBC 2 and 3. What we do learn is that for a channel with high correlation like QBC 2 it is still better to exploit the channel memory than it is to use an irregular code with excellent performance over an interleaved version of the channel.

Chapter 6

Conclusion

In this work we have presented a method of decoding low-density parity-check codes over channels with binary additive Markov noise using the sum-product algorithm.

We factor the probability distribution for the optimum decoding rule which maximizes the probability that $\hat{\mathbf{x}}^{\mathbf{n}}$ was sent given that $\mathbf{y}^n$ was received, where $\hat{\mathbf{x}}^{\mathbf{n}}$ is our decoders estimate of the value of $\mathbf{x}^n$, the transmitted codeword. This is given by the joint probability of the block-channel transition probabilities and the parity-check functions of the code. By incorporating the block transition probabilities for a channel with Markov memory this rule can be extended to channels with memory as well. Thus, we can obtain a factorization which includes the factored channel-state transition probabilities of the Markov process for the channel.

The factorization is represented graphically as a relationship between factor equations and the variables which participate in them. By iteratively applying the SPA to this factorization we are able to compute results that converge very close to the optimal bitwise MAP decoding rule, which minimizes the probability of bit error. The SPA decoder is a linear time algorithm with respect to the length of the codewords. The result obtained is,

admittedly, sub-optimal due to cycles in the graphs of LDPC codes; however, the SPA is a significant reduction in complexity compared with the optimal solution to MAP decoding, which is NP hard.

This extended SPA decoder is a joint estimator-decoder capable of simultaneous channel-state estimation and decoding. The SPA based joint estimator-decoder is also a linear time algorithm with respect to both the block-length of the code and the number of states of the channel (though it is exponential in M, the memory of the channel for the QBC and BAMNC models).

6.1 Significance of the Results

The results from the simulated performance of the SPA based joint estimator-decoder show that joint estimation-decoding performs as well on the QBC as the standard SPA decoder performs on the memoryless BSC. In other words, the SPA is just as capable of computing the probabilities for approximate MAP decoding of more complex channel models such as the QBC, as it is for decoding very simple memoryless models like the BSC.

This demonstrates that significant performance gains are achievable using joint estimation-decoding vs. interleaving and decoding assuming the channel is memoryless. These gains would seem to outweigh the added complexity of the estimator-decoder at least for channels which have sufficiently greater capacity than the equivalent memoryless channel. However, in order to achieve these gains in real world communication systems, it is necessary to be able to model the real world channel statistics accurately.

By minimizing the divergence rate between QBC and GEC models as well as computing M^{th}-order BAMNC models for the GEC, our simulations demonstrate that channel matching based on finite-order block statistics is an effective tool. While the results ob-

tained show that the decoder performs best when it has precise knowledge of the channel, we see that an approximation is sufficient to effectively exploit the channel memory. In real-world communications systems, developing exact models of the channel is impractical. These simulations demonstrate that for the GEC, QBC and BAMNC models, reasonable performance gains are still achievable through channel matching. This suggests that channel matching using the QBC or BAMNC model could be used to exploit channel memory in real channels with worthwhile results.

There are two parts to an error-correcting coding system, the encoder and the decoder. We have shown the effectiveness of the joint estimator-decoder in improving performance over channels with memory. We also examined the effect of code design on the system's performance through the use of irregular LDPC codes. Simulations are performed using two different irregular LDPC codes, which have been designed to perform much better than the (3,6)-regular LDPC code on the BSC and a nearly memoryless GEC. We observe that the parameters of the QBC that the code is used on have a substantial effect on the performance of the code. Both irregular codes were designed for channels with very weak correlation (in the case of the first code, no correlation at all). We see that even when used over the QBC, these codes perform very well if the QBC's correlation is also weak. However, when we simulate these codes over a QBC with strong correlation the performance of these codes is actually worse than the performance of the regular LDPC code on this channel though still better than the weakly correlated QBC.

It would seem that for channels with a strongly correlated noise process the choice of channel model used in the code design is much more important than it is for channels with weak correlation. Unfortunately, no work has been done constructing irregular LDPC codes for strongly correlated channels, so there is still much about the design of irregular LDPC codes for channels with Markov memory that remains to be explored.

6.2 Future Work

A number of areas of future research are indicated by the results of this thesis are:

- Developing and testing a decoding algorithm with linear complexity in M for the QBC (i.e., sub-linear in the number of states of the channel).
- Extending the work from [23] and [37] to explore modeling hard-decision correlated Rayleigh and Rican fading with the QBC and BAMNC and testing SPA decoder performance over using these models.
- Characterizing the QBC parameters to modify density evolution for this channel as was done for the GEC in [8].
- Design of irregular LDPC codes which outperform regular LDPC codes for the QBC using density evolution.

Bibliography

[1] F. Alajaji and T. Fuja, "A communication channel modeled on contagion," *IEEE Trans. Inform. Theory*, vol. 40, no. 6, pp. 2035–2041, November 1994.

[2] L. R. Bahl, J. Cocke, F. Jeliniek, and J. Raviv, "Optimal decoding of linear codes for minimal symbol error rate," *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 284–87, March 1974.

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

[4] S.-Y. Chung, J. G. David Forney, T. J. Richardson, and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit," *IEEE Comm. Letters*, vol. 5, no. 2, pp. 58–60, February 2001.

[5] T. M. Cover and J. A. Thomas, *Elements of Information Theory* 2^{nd} *Ed.* Wiley, 1991.

[6] A. W. Eckford, F. R. Kschischang, and S. Pasupathy, "Analysis of LDPC codes in channels with memory," in *Proc. of the 21st Queen's Biennial Symposium on Communications*, Kingston, Ontario, Canada, 2002.

[7] ——, "Designing very good low-density parity-check codes for the Gilbert-Elliot channel," in *Proc. of the 8th Canadian Workshop on Information Theory*, Waterloo, Ontario, Canada, 2003.

[8] A. W. Eckford, *Low-Density Parity-Check Codes for Gilbert-Elliot and Markov-Modulated Channels*. Ph.D. thesis, University of Toronto, 2004.

[9] P. Elias, "Coding for noisy channels," in *Proc. of IRE Conv. Rec.*, 1955, pp. 4:37–47.

[10] E. O. Elliot, "Estimate of error rates for codes on burst-noise channels," *IEEE Trans. Inform. Theory*, vol. 35, no. 6, pp. 1277–1290, November 1989.

[11] R. G. Gallager, *Low-Density Parity-Check Codes*. M.I.T. Press, 1963.

[12] J. Garcia-Frias, "Decoding of low-density parity-check codes over finite-state binary Markov channels," *IEEE Trans. Commun.*, vol. 52, no. 11, November 2004.

[13] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1991.

[14] E. N. Gilbert, "Capacity of a burst-noise channel," *Bell System Technical Journal*, vol. 42, pp. 1977–1997, September 1963.

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

[16] S. Lin and D. J. Costello, *Error Control Coding* 2^{nd} *Ed.* Prentice Hall, 2004.

[17] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol. 45, no. 2, pp. 399–431, March 1999.

[18] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 32, pp. 1645–1646, 1996.

[19] M. Mushkin and I. Bar-David, "Capacity and coding for the Gilbert-Elliott channels," *IEEE Trans. Inform. Theory*, vol. 35, no. 6, November 1989.

[20] V. Nagarajan and O.Milenkovic, "Performance analysis of structured LDPC codes for the Polya-urn channel with finite memory," in *Proc. of the CCECE 2004*, Niagara Falls, Canada, 2004.

[21] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufman, 1988.

[22] L. C. Perez, J. Seghers, and J. D. J. Costello, "A distance spectrum interpretation of Turbo codes," *IEEE Trans. Inform. Theory*, vol. 42, no. 6, pp. 1698–1709, November 1996.

[23] C. Pimentel, T. H. Falk, and L. Lisboa, "Finite-state Markov modeling of correlated Rician-fading channels," *IEE Trans. Vehicular Tech.*, vol. 53, no. 5, September 2004.

[24] G. Polya, "Sur quelques points de la théorie des probabilities," *Ann. Inst. H. Poincarre*, vol. 1, pp. 117–161, 1931.

[25] E. Ratzer, "Low-density parity-check codes on Markov channels," in *Second IMA Conference on Mathematics in Communications*, December 2002.

[26] 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, February 2001.

[27] 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, February 2001.

[28] P. Sadeghi and P. Rapajic, "Capacity analysis for finite-state Markov mapping of flat-fading channels," *IEEE Trans. Commun.*, vol. 53, May 2005.

[29] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. vol. 27, pp. 379–423 and 623–656, July and October 1948.

[30] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory*, vol. IT-27, pp. 533–547, September 1981.

[31] S. ten Brink, "Convergence of iterative decoding," *Electronics Letters*, vol. 35, pp. 806–808, May 1999.

[32] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 260–69, April 1967.

[33] H. S. Wang and P.-C. Chang, "On verifying the first-order Markovian assumption of a Rayleigh fading channel model," *IEEE Trans. Vehicular Tech.*, vol. 45, no. 2, May 1996.

[34] N. Wiberg, H.-A. Loeliger, and R. Kotter, "Codes and iterative decoding on general graphs," *Eur. Trans. Telecomm.*, vol. 6, pp. 513–525, Sept./Oct. 1995.

[35] L. Zhong, F. Alajaji, and G. Takahara, "A queue-based model for binary communication channels," in *Proc. of the 41st Annual Allerton Conference on Communication, Control and Computing*, Monticello, IL, October 2003.

[36] ——, "An approximation of the Gilbert-Elliott channel via a queue-based channel model," in *Proc. of the IEEE International Symposium on Information Theory*, Chicago, June-July 2004.

[37] ——, "A model for correlated Rican fading channels based on a finite queue," *Submitted to IEEE Trans. Vehicular Tech.*, July 2005.