5-2012

Utilizing Correct Prior Probability Calculation to Improve Performance of Low-Density Parity-Check Codes in the Presence of Burst Noise

David A. Neal
Utah State University

UTILIZING CORRECT PRIOR PROBABILITY CALCULATION TO IMPROVE PERFORMANCE OF LOW-DENSITY PARITY-CHECK CODES IN THE PRESENCE OF BURST NOISE

by

David A. Neal

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

of

MASTER OF SCIENCE

in

Electrical Engineering

Approved:

Dr. Todd K. Moon Major Professor	Dr. Jacob Gunther Committee Member
Dr. Donald Cripps Committee Member	Dr. Mark R. McLellan Vice President for Research and Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2012

Abstract

Utilizing Correct Prior Probability Calculation to Improve Performance of Low-Density Parity-Check Codes in the Presence of Burst Noise

by

David A. Neal, Master of Science

Utah State University, 2012

Major Professor: Dr. Todd K. Moon
Department: Electrical and Computer Engineering

Low-density parity-check (LDPC) codes provide excellent error correction performance and can approach the channel capacity, but their performance degrades significantly in the presence of burst noise. Bursts of errors occur in many common channels, including the magnetic recording and the wireless communications channels. Strategies such as interleaving have been developed to help compensate for bursts errors. These techniques do not exploit the correlations that can exist between the noise variance on observations in and out of the bursts. These differences can be exploited in calculations of prior probabilities to improve accuracy of soft information that is sent to the LDPC decoder.

Effects of using different noise variances in the calculation of prior probabilities are investigated. Using the true variance of each observation improves performance. A novel burst detector utilizing the forward/backward algorithm is developed to determine the state of each observation, allowing the correct variance to be selected for each. Comparisons between this approach and existing techniques demonstrate improved performance. The approach is generalized and potential future research is discussed.

(106 pages)

Public Abstract

Utilizing Correct Prior Probability Calculation to Improve Performance of Low-Density Parity-Check Codes in the Presence of Burst Noise

by

David A. Neal, Master of Science

Utah State University, 2012

Major Professor: Dr. Todd K. Moon
Department: Electrical and Computer Engineering

Error correcting codes help correct errors in transmitted data. These errors are caused by noise and distortion that are introduced during transmission. In some cases, there is more noise on some portions of a transmitted message than on others. This extra noise is referred to as burst noise and causes bursts of errors.

Low-density parity-check (LDPC) codes are a family of error correcting codes. They provide excellent performance in many cases, but perform poorly when burst noise is present. Methods have been developed that help compensate for bursts of errors. In most cases, these do not completely account for the burst noise. Instead, they manipulate the data to spread the errors out across the received data. This helps the LDPC code correct the errors more easily in some cases.

As an alternative approach, a method is developed to identify the locations of the bursts and properly account for them in order to improve LDPC code error correcting performance. This is shown to exceed the performance of existing techniques used with LDPC codes. In addition, this technique provides an opportunity to utilize LDPC codes in scenarios where they could not previously be used.

Acknowledgments

I extend gratitude and thanks to my parents, Tom and Barbara Neal, who supported me in my undergraduate education, helped me become an engineer, and have always believed that I can do anything.

I am also grateful to Dr. Todd Moon, who provided me with an opportunity to pursue a master's degree and shared with me his love of signal processing and error correcting codes.

David Alexander Neal

Contents

Page

Abstract . . . . . **iii**

Public Abstract . . . . . **iv**

Acknowledgments . . . . . **v**

List of Tables . . . . . **viii**

List of Figures . . . . . **ix**

Acronyms . . . . . **xiv**

1 Introduction **1**
1.1 Error-Correcting Codes 1
1.1.1 LDPC Codes 1
1.1.2 Noise in Channels 1
1.1.3 Burst Noise Channels 2
1.1.4 Existing Techniques 2
1.1.5 LDPC Codes and Burst Noise 4
1.2 Correct Prior Calculations as a New Approach 5

2 Codes and Error Bursts **6**
2.1 Low-Density Parity-Check Codes 6
2.2 Burst Noise Model 8
2.2.1 Burst Sources 8
2.2.2 Noise Types 9
2.2.3 Burst Model Characterization 11
2.3 LDPC Codes with Burst Noise 15

3 Burst Error Compensation **23**
3.1 Interleaving 23
3.2 *Lying* to the Decoder 28
3.2.1 Calculating Priors Using Burst Variance 30
3.2.2 Conclusions on Lying 34
3.3 Proper Prior Calculation 34
3.3.1 Priors for Uncorrelated Burst Noise 35
3.3.2 Priors for Correlated Noise 35
3.3.3 Performance Comparison 37

4 Burst Estimation **43**
4.1 Existing Techniques 43
4.1.1 Algebraic Structure 43
4.1.2 Parity-Check Thresholding 44
4.1.3 Other Techniques 45
4.1.4 Conclusion on Existing Techniques 45
4.2 New Technique: Markov State Detection 45
4.2.1 Forward-Backward Algorithm 46
4.2.2 Correlated Forward-Backward Algorithm 50
4.2.3 Performance 53
4.2.4 Conclusion 57

5 Results **58**
5.1 Uncorrelated Burst Noise Results 58
5.2 Correlated Burst Noise Results 70
5.3 Conclusions 81

6 Conclusions **83**
6.1 Summary of Conclusions 83
6.2 Generalized Approach 84
6.2.1 Optimality 84
6.2.2 Complexity 85
6.2.3 Extending Beyond LDPC Codes 86
6.3 Future Work 86
6.3.1 Burst Modeling and Detection 86
6.3.2 Code Performance Investigation 87
6.3.3 Priors Calculations Using Incorrect Variance Values 87
6.3.4 Burst Time Effects 88
6.3.5 Interleaving and Correct Prior Calculations 88
6.4 Conclusion 89

References **90**

List of Tables

Table		Page
2.1	Coding gain for three LDPC codes.	7
2.2	SBNR and σ_b^2 for test codes at P_b of 10^{-4}.	10
2.3	Estimated and actual burst times and average burst length for 100,000 bits.	15
3.1	Prior probabilities for various observations and variances.	29
3.2	Coding gain for three LDPC codes with and without burst noise.	31
3.3	Coding gain comparison with correct priors with 5% burst time.	42
5.1	Coding gain comparison with an uncorrelated burst and 1% burst time.	68
5.2	Coding gain comparison with an uncorrelated burst and 2% burst time.	68
5.3	Coding gain comparison with an uncorrelated burst and 5% burst time.	69
5.4	Coding gain comparison with a correlated burst and 1% burst time.	80
5.5	Coding gain comparison with a correlated burst and 2% burst time.	80
5.6	Coding gain comparison with a correlated burst and 5% burst time.	80

List of Figures

Figure Page

2.1 Bit-error rate curves for three LDPC codes. 7

2.2 Two-state Markov chain. 13

2.3 1000 samples from burst distribution across a 100,000 state sampling, 5% burst time. 14

2.4 Histogram of burst lengths for 100,000 bits, 5% burst time. 15

2.5 Bit-error rate curve for rate .333 code with 1% burst time and SBNR of 1.6 dB. 17

2.6 Bit-error rate curve for rate .333 code with SBNR of 1.6 dB. 17

2.7 Bit-error rate curve for rate .700 code with SBNR of 4.6 dB. 18

2.8 Bit-error rate curve for rate .936 code with SBNR of 3.6 dB. 18

2.9 Bit-error rate curve for rate .333 code with 5% burst time and scaled burst variance. 20

2.10 Bit-error rate curve for rate .936 code with 5% burst time and scaled burst variance. 21

2.11 Bit-error rate curve for rate .700 code with 5% burst time and scaled burst variance. 22

3.1 Bit-error rate curve for interleaved rate .333 code with 1% burst time and SBNR of 1.6 dB. 24

3.2 Bit-error rate curve for interleaved rate .333 code with SBNR of 1.6 dB. . . 25

3.3 Bit-error rate curve for interleaved rate .700 code with SBNR of 4.6 dB. . . 25

3.4 Bit-error rate curve for interleaved rate .936 code with SBNR of 3.6 dB. . . 26

3.5 Interleaved versus non-interleaved for rate .333 code with 5% burst time and SBNR of 1.6 dB. 26

3.6 Interleaved versus non-interleaved for rate .700 code with 5% burst time and SBNR of 4.6 dB. 27

3.7 Interleaved verus non-interleaved for rate .936 code with 5% burst time and SBNR of 3.6 dB. 27

3.8 Comparison between non-burst and burst variance prior calculations for .333 code with 5% burst time, correlated burst and SBNR of 1.6 dB. 31

3.9 Comparison between non-burst and burst variance prior calculations for .700 code with 5% burst time, correlated burst, and SBNR of 4.6 dB. 32

3.10 Comparison between non-burst and burst variance prior calculations for .936 code with 5% burst time, correlated burst, and SBNR of 3.6 dB. 32

3.11 Comparison between burst and non-burst calculated priors for rate .333 code with 5% burst time, correlated burst, and four times burst variance. 33

3.12 Low and high estimates for variance for rate .333 code with 5% burst time, correlated burst, and 100 times burst variance. 33

3.13 Correct prior calculation comparison for rate .333 code with 5% burst time and SBNR of 1.6 dB. 39

3.14 Correct prior calculation comparison for rate .700 code with 5% burst time and SBNR of 4.6 dB. 39

3.15 Correct prior calculation comparison for rate .936 code with 5% burst time and SBNR of 3.6 dB. 40

3.16 Close-up of correct prior calculation comparison for rate .333 code with 5% burst time and SBNR of 1.6 dB. 40

3.17 Close-up of correct prior calculation comparison for rate .700 code with 5% burst time and SBNR of 4.6 dB. 41

3.18 Close-up of correct prior calculation comparison for rate .936 code with 5% burst time and SBNR of 3.6 dB. 41

4.1 Estimated burst performance for rate .333 code with 5% burst time, uncorrelated burst, and SBNR of 1.6 dB. 54

4.2 Estimated burst performance for rate .333 code with 5% burst time, correlated burst, and SBNR of 1.6 dB. 54

4.3 Estimated burst performance for rate .700 code with 5% burst time, uncorrelated burst, and SBNR of 4.6 dB. 55

4.4 Estimated burst performance for rate .700 code with 5% burst time, correlated burst, and SBNR of 4.6 dB. 55

4.5 Estimated burst performance for rate .936 code with 5% burst time, uncorrelated burst, and SBNR of 3.6 dB. 56

4.6 Estimated burst performance for rate .936 code with 5% burst time, correlated burst, and SBNR of 3.6 dB. 56

5.1 Uncorrelated burst comparison for rate .333 code with 1% burst time and SBNR of 1.6 dB. 59

5.2 Close-up of uncorrelated burst comparison for rate .333 code with 1% burst time and SBNR of 1.6 dB. 59

5.3 Uncorrelated burst comparison for rate .333 code with 2% burst time and SBNR of 1.6 dB. 60

5.4 Close-up of uncorrelated burst comparison for rate .333 code with 2% burst time and SBNR of 1.6 dB. 60

5.5 Uncorrelated burst comparison for rate .333 code with 5% burst time and SBNR of 1.6 dB. 61

5.6 Close-up of uncorrelated burst comparison for rate .333 code with 5% burst time and SBNR of 1.6 dB. 61

5.7 Uncorrelated burst comparison for rate .700 code with 1% burst time and SBNR of 4.6 dB 62

5.8 Close-up of uncorrelated burst comparison for rate .700 code with 1% burst time and SBNR of 4.6 dB. 62

5.9 Uncorrelated burst comparison for rate .700 code with 2% burst time and SBNR of 4.6 dB. 63

5.10 Close-up of uncorrelated burst comparison for rate .700 code with 2% burst time and SBNR of 4.6 dB. 63

5.11 Uncorrelated burst comparison for rate .700 code with 5% burst time and SBNR of 4.6 dB. 64

5.12 Close-up of uncorrelated burst comparison for rate .700 code with 5% burst time and SBNR of 4.6 dB. 64

5.13 Uncorrelated burst comparison for rate .936 code with 1% burst time and SBNR of 3.6 dB. 65

5.14 Close-up of uncorrelated burst comparison for rate .936 code with 1% burst time and SBNR of 3.6 dB. 65

5.15 Uncorrelated burst comparison for rate .936 code with 2% burst time and SBNR of 3.6 dB. 66

5.16 Close-up of uncorrelated burst comparison for rate .936 code with 2% burst time and SBNR of 3.6 dB. 66

5.17 Uncorrelated burst comparison for rate .936 code with 5% burst time and SBNR of 3.6 dB. 67

5.18 Close-up of uncorrelated burst comparison for rate .936 code with 5% burst time and SBNR of 3.6 dB. 67

5.19 Correlated burst comparison for rate .333 code with 1% burst time and SBNR of 1.6 dB. 71

5.20 Close-up of correlated burst comparison for rate .333 code with 1% burst time and SBNR of 1.6 dB. 71

5.21 Correlated burst comparison for rate .333 code with 2% burst time and SBNR of 1.6 dB. 72

5.22 Close-up of correlated burst comparison for rate .333 code with 2% burst time and SBNR of 1.6 dB. 72

5.23 Correlated burst comparison for rate .333 code with 5% burst time and SBNR of 1.6 dB. 73

5.24 Close-up of correlated burst comparison for rate .333 code with 5% burst time and SBNR of 1.6 dB. 73

5.25 Correlated burst comparison for rate .700 code with 1% burst time and SBNR of 4.6 dB. 74

5.26 Close-up of correlated burst comparison for rate .700 code with 1% burst time and SBNR of 4.6 dB. 74

5.27 Correlated burst comparison for rate .700 code with 2% burst time and SBNR of 4.6 dB. 75

5.28 Close-up of correlated burst comparison for rate .700 code with 2% burst time and SBNR of 4.6 dB. 75

5.29 Correlated burst comparison for rate .700 code with 5% burst time and SBNR of 4.6 dB. 76

5.30 Close-up of correlated burst comparison for rate .700 code with 5% burst time and SBNR of 4.6 dB. 76

5.31 Correlated burst comparison for rate .936 code with 1% burst time and SBNR of 3.6 dB. 77

5.32 Close-up of correlated burst comparison for rate .936 code with 1% burst time and SBNR of 3.6 dB. 77

5.33 Correlated burst comparison for rate .936 code with 2% burst time and SBNR of 3.6 dB. 78

5.34 Close-up of correlated burst comparison for rate .936 code with 2% burst time and SBNR of 3.6 dB. 78

5.35 Correlated burst comparison for rate .936 code with 5% burst time and SBNR of 3.6 dB. 79

5.36 Close-up of correlated burst comparison for rate .936 code with 5% burst time and SBNR of 3.6 dB. 79

6.1 Generic burst compensation system. 85

Acronyms

AWGN	Additive White Gaussian Noise
BCH	Bose and Ray-Chaudhuri
BER	Bit-Error Rate
BPSK	Binary Phase-Shift Keying
ECC	Error-Correcting Codes
ISI	Inter-Symbol Interference
LDPC	Low-Density Parity-Check
SBNR	Signal-to-Burst-Noise Ratio
SNR	Signal-to-Noise Ratio
TA	Thermal Asperity

Chapter 1

Introduction

1.1 Error-Correcting Codes

Communication between electronic devices is ubiquitous in modern society. Transmitted data is sent through a channel which introduces distortion and noise. This is especially true in the wireless case, where effects such as Rayleigh fading can significantly degrade the signal. Additional noise is introduced by the transmitter and receiver. Storage devices face similar problems, as the hardware storing the data is exposed to many effects over time and is expected to provide the uncorrupted data to the user on demand. All these effects make error-correcting codes (ECC) integral in communications and data storage applications in order to overcome these effects and ensure high data reliability.

1.1.1 LDPC Codes

Low-density parity-check (LDPC) codes provide a fast, effective means of encoding data for reliable transmission or storage by exploiting soft data. They have been shown to approach channel capacity [1–3]. Making use of LDPC coding schemes in applications where other coding schemes have traditionally been used can allow for improved speed and greater data reliability.

1.1.2 Noise in Channels

LDPC codes can be of particular use in memory storage devices, where increasing memory density has decreased the reliability of the data storage and readback [4–6]. This manifests itself as higher noise in the channel. The high coding gain of LDPC codes could help overcome this noise. One problematic aspect is the presence of burst noise in storage device channels. In burst noise channels, LDPC code performance is still good when the

bursts are short enough not to overlap multiple bits on the same parity check [7]. However, when bursts overlap multiple bits in the same parity check, performance degrades to the point that other codes are better choices [2, 8, 9].

1.1.3 Burst Noise Channels

Burst noise in a channel occurs when some event, acting over a period of time, adds noise and causes a group of errors that are adjacent to each other. In some cases, the burst will be for the length of an entire message, but not on other messages, severely corrupting that particular message. In other cases, it can occur over parts of a single message, but not affect other parts of that same message. Multiple bursts from multiple sources may be present simultaneously.

As an example, consider transmitted symbols representing a codeword $\mathbf{c}$ of length i. The codeword may be sent through an Additive White Gaussian Noise (AWGN) channel, adding noise such that the signal $\mathbf{r}$ is received as $\mathbf{r} = \mathbf{c} + \mathbf{n}$, where $\mathbf{n}$ is noise distributed $N(0, \sigma^2)$. Let a burst start at symbol j and end at k. This means that for symbols from j to k, the received value is actually $\mathbf{r} = \mathbf{c} + \mathbf{n} + \mathbf{b}$ where $\mathbf{b}$ is the burst noise, versus only $\mathbf{r} = \mathbf{c} + \mathbf{n}$ without the burst. In this model, the burst noise adds with the AWGN.

Different types of burst noise occur and can drastically affect the observations of data. If the locations of the burst are not known or the bursts are not accounted for, calculations based on observations of data may become inaccurate and negatively affect bit-error rates (BER).

1.1.4 Existing Techniques

Techniques for dealing with burst noise vary. Prevalent among these techniques are interleaving, concatenated codes, and codes specifically designed for the types of bursts seen. Combinations of these techniques are also employed in different cases.

Interleaving

Interleaving is a common technique for dealing with burst noise. It involves reordering

the data to be transmitted, mixing symbols from different messages together into new messages of the same size, sending them through the channel, and then de-interleaving them back into the original messages at the receiver. This results in spreading out the bursty bits among multiple messages, increasing the likelihood that the number of errors will be less than the error-correcting capability of the ECC [10].

When employing interleaving, it is important to choose an interleaving pattern that is optimal for the given noise. Some examples are mixing patterns that are optimized for 2-dimensional noise [10] and adaptive interleaving to account for distortion from Rayleigh fading channels [11].

Concatenated Codes

Concatenated codes utilize multiple codes of either the same or different types to assist in coding. Encoding in a random error correcting code and then encoding in a code that is good at correcting bursts often provides improved performance. At decoding, the burst error correcting code corrects the bursts and then the random error correcting code can fix any remaining errors. A common example is to use a Reed-Solomon (RS) code as an outer burst-correcting code and an LDPC code as an inner random-error-correcting code. Performance is typically good, but this can have negative effects on code rate, due to the additional coding, and complexity [8].

Another approach is to use an additional inner code to help even more with burst errors. After using an outer Bose and Ray-Chaudhuri (BCH) code, the data is split up into blocks, all but one of which are encoded using LDPC codes. The remaining block is coded using an RS and the blocks are then interleaved into one message. The decoding of the RS code helps identify burst regions in the interleaved signal, allowing the bursty bits that are in the LDPC codes to be erased [12]. This method, then, makes use of all three of the different techniques discussed so far.

Burst Codes

Codes designed for dealing with burst errors obviously provide an ideal solution. One

example of an LDPC-type code that targets burst errors is based on cyclic LDPC matrices. By exploiting the structure of the parity-check matrix, a burst of a certain size or smaller can be detected, located, and corrected. The code is only LDPC-like in that it uses a low-density parity check matrix. It decodes in an entirely different manner [2]. The code is very effective, but longer bursts or two random errors simply separated by more than the codes' burst-correcting capability cause the method to break down. Given the presence of random noise on real channels, this does not seem an effective method for combatting bursts of errors.

Other codes exist that target burst errors and are effective for correcting burst errors [13, 14]. Fire codes are an excellent example and have been developed to correct bursts of a certain length. In many cases, these only correct one burst in a message, which does not work well with our burst model [15]. A most interesting approach is to simply solve for the Bayesian-optimum solution of the observed data. This method both detects the burst and interpolates the original data underneath the burst. The given derivation is only applicable in the framework of the article, especially since it assumes an auto-regressive (AR) data source [16]. A variation could be developed for other systems.

Compatibility with LDPC Codes

While these techniques have been used with great success in many applications, they do not necessarily work well with LDPC codes. This costs us the channel-capacity approaching performance of the LDPC codes.

Of the methods suggested, only one does not preclude using only LDPC codes. Interleaving can easily be combined with any decoding method, as it simply reorders bits before and after transmission. It does not in any way affect our ability to use LDPC encoding and decoding and is often used with LDPC codes in practice to help combat burst errors.

1.1.5 LDPC Codes and Burst Noise

While interleaving has been shown to be effective for combatting burst errors, it is effective by distributing the noise amongst a number of transmitted codewords, reducing the

overall noise on each codeword, allowing the decoder to work successfully. It is interesting to consider if we can directly improve the performance of LDPC codes when burst noise is present on received codewords.

1.2 Correct Prior Calculations as a New Approach

It is common in communication systems to assume that all noise is AWGN. This is true in calculating the soft prior proabilities that are input into LDPC decoders.

As a new approach to improving LDPC decoding performance in burst noise channels, instead of migrating to complicated coding schemes or simply shuffling the noise amongst multiple messages, the burst noise shall be exploited to provide additional information for identifying the burst region(s) and for calculating priors that are fed into the LDPC decoder. It will be shown that properly including the noise in the prior calculations greatly improves LDPC decoding performance when burst noise is present.

Several LDPC codes will be tested both with and without bursts for comparison. An exploration of the effects of ignoring bursts will be conducted in order to provide better understanding of how the bursts are affecting the decoder performance. Modifications to prior calculations will be made to reflect the presence of bursts. These results will be compared to both the performance without bursts and performance with bursts with no code modifications. A comparison will be made with the use of interleaving with an LDPC code as a baseline against standard techniques.

A novel burst detection technique will be developed and used in the decoder as a comparison against performance when the burst is perfectly known. This will demonstrate the practicality of the code in real applications where the burst location may be unknown. All comparisons will be performed on binary codes.

Chapter 2

Codes and Error Bursts

2.1 Low-Density Parity-Check Codes

Decoding of LDPC codes can be accomplished by several different algorithms. Gallager originally proposed two methods: the bit-flipping algorithm and a probabilistic approach [17]. The decoder that will be used herein makes use of the sum-product algorithm, which is the probabilistic approach, and derivations which are readily available [18]. Variations on this decoder exist that propose modifications to improve speed or accuracy.

Several codes have been selected to test the techniques that will be developed later. Different code rates and lengths were selected to measure performance under different conditions.

The codes selected are a one-third-rate $(1920, 1280)$ code [19], a seven-tenths-rate $(273, 82)$ code [20], and a .936-rate $(4376, 282)$ code [21]. These codes were selected to provide both low and high rate codes and variable lengths as well. The codes were tested over signal-to-noise ratio (SNR) values that were specific to each code. Figure 2.1 shows the probability of bit error versus SNR for each code in a standard AWGN channel. The x-axis shows Eb/No in dB, which is bit energy (Eb) over noise energy (No). This is also referred to as SNR. The codes are run versus SNR values in dB, which is converted to a noise variance and used to generate the random noise samples. The y-axis shows Pb, which is the probability of bit error. For a given code run, this is found by summing up all errors at a given SNR and dividing by the total number of bits sent.

The gain for the three codes is listed in Table 3.2, where the gain is measured at probability of bit error of 10^{-6}. The codes are compared with binary phase-shift keying (BPSK) since they are binary codes.

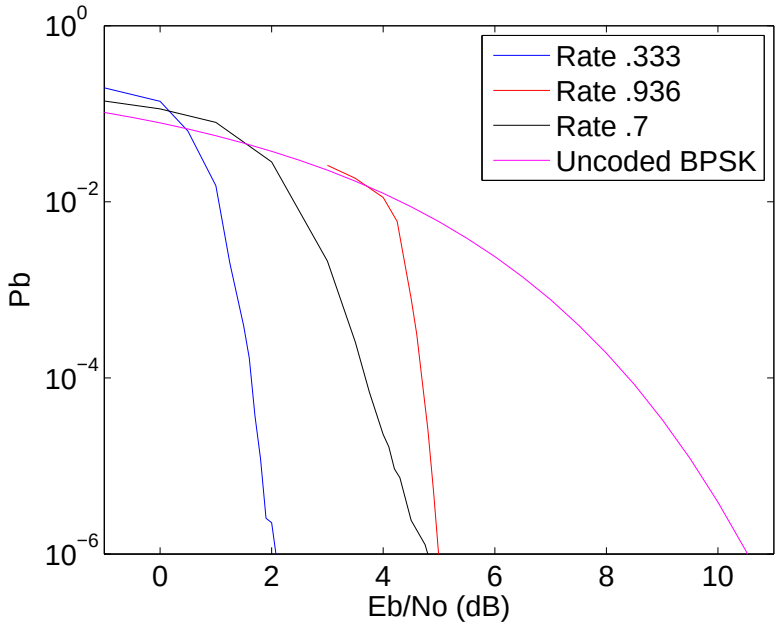

Fig. 2.1: Bit-error rate curves for three LDPC codes.

Table 2.1: Coding gain for three LDPC codes.

Code Rate	Coding Gain
.333	8.4
.936	5.5
.7	5.7

The code used to generate the curves shown in Figure 2.1 allows for varying the maximum number of messages to run at each SNR value, the minimum number of errors to wait for, and the maximum number of failed messages that are allowed before it moves to the next SNR value. The code moves to the next SNR when any one of the three conditions is met. For all three codes, the maximum number of failed messages was ten and the minimum number of errors allowed was 1000. Due to the different lengths of the codes, different values for the maximum number of messages were used to ensure that a sufficient number of data points were available to calculate the probability of bit-error at higher SNR values. A target of 100 million points was set to allow for the probability of bit-error to go below 10^{-6}, which is a typical measurement point for coding gain. For the one-third rate code, the maximum number of messages was set to 85000, for the .936 rate code it was set to

30000, and for the seven-tenths rate code, it was set to 400000.

2.2 Burst Noise Model

In order to find probability of bit-error curves for the codes in burst noise, a generative model needs to be developed for burst noise. While models for simulating AWGN channels are simple and easily implemented, several variations for burst noise models exist. The most common models are variations on Markov models. In the most basic form, burst channels are modeled by two-state Markov models [22]. In cases where this does not accurately reflect the channel properties, Markov models with more states are typically applied. This additional accuracy comes at the cost of greater complexity, but improves the match of the model's statistical properties to the burst channel [23, 24]. The difference between different burst noise models mainly focuses on how to set parameters in these models to match the burst characteristics of different channels [25].

In order to generalize the burst model, a two-state Markov model will be used to generate the burst locations. There are multiple types of bursts to be addressed, but no specific channels have been chosen for study. Rather, noise characteristics will be used that reflect several types of bursts. More specific models could be developed for specific applications, both in the noise models and in the burst-generating model.

Because the burst sources are separate from the AWGN sources, the channels that will be explored have both AWGN and burst noise present. This means that in one of the states of the Markov model, only AWGN is present, representing the non-burst state. In the other state, both AWGN and burst noise are present, representing the burst state.

2.2.1 Burst Sources

There are various factors that cause bursts. In many cases, these are actually interference sources as opposed to noise, but they can be modeled as noise bursts. Effects that cause interference sometimes result in attenuation or information replacing the intended data. We can take the characteristics of the interference and model them as a noise source

that produces the same effects. For simplicity in discussion, we will consider the sources simply as burst noise.

Some examples of burst sources occur in electronic storage device channels. Alpha particle strikes, thermal asperity (TA) noise, and write errors are common sources of burst noise. Burst phenomena also occur in other channels, including the mobile wireless channel [11]. In this channel, multipath fading occurs that changes with respect to time [26]. In the case of Viterbi decoding, errors in decoding occur in a burst [27–29].

Alpha particles introduce errors by changing the energy levels in cells. This can result in higher noise on some bits than others. As the size of such devices decreases, they become more susceptible to such errors [5, 6, 30]. Thermal asperity noise adds extra noise on the symbols over the times where it occurs. This creates groups of errors in these regions [8]. Write errors occur when writing data to the incorrect locations, and erasure bursts caused by attenuation can also create burst noise [12].

Multipath fading results from reflection and diffraction of the signal as it travels between the transmitter and receiver. The result is fluctuation in the received signal phase and amplitude. Delay can also occur, possibly resulting in inter-symbol interference (ISI) [26].

For Viterbi decoding, errors tend to be grouped togehter in bursts. This is a result of errors causing a deviation in the maximum-likelihood path. The errors result in a path being chosen that moves through the states that the errors cause. In order to get to these states, additional errors are introduced. In this way, a burst is formed [27–29]. Since single events are causing the burst of errors, there will be correlation between the elements in the burst. While Viterbi decoders are not utilized in LDPC decoding, in concatenated codes where Viterbi and LDPC decoders are used together, the LDPC decoder would see burst errors from the Viterbi decoding.

2.2.2 Noise Types

Each source of noise provides different characteristics for the burst noise. However, they exhibit some common characterstics that allow for simple modeling of their effects. By defining some basic burst types with certain probabilistic characterstics, general models

representing multiple sources of noise show how the properties of each type of noise can be exploited. Two general types of burst noise have been chosen. While the models chosen do not represent all types of noise, they do provide a setting in which to test the effectiveness of the techniques that will be developed. They also provide a framework for developing methods for noise types that are not investigated.

In the model that will be used, AWGN and burst noise will both be added to the signal. In order to provide some comparison between the burst and AWGN, the burst noise variance will be considered as a multiple of the AWGN variance.

Additional Uncorrelated Noise

The most simplistic burst noise type is simply additional Gaussian noise. Random errors caused by AWGN can be compounded by adding more noise at the burst locations. This noise will combine with the existing AWGN to increase the noise variance, as variances of multiple Gaussian random variables add together. So, for this case, the non-burst state will have variance σ^2 and the burst state will have a constant variance σ_b^2, which will be set to a level that causes a probability of bit error of approximately 10^{-4} for the given code, plus the non-burst variance. We will refer to this combination $\sigma^2 + \sigma_b^2$ as the burst variance for convenience. Based on the performance of each code, the values of the signal-to-burst-noise ratio (SBNR) will be set as given in Table 2.2.

The variance σ_b^2 is calculated from the SBNR given $E_b = 1$ and adjusted for the specific code rate.

This noise can be considered as a simple model for bursts of attenuation or ISI that impact the signal in the mobile channel or TA and write errors in the storage device channel.

Table 2.2: SBNR and σ_b^2 for test codes at P_b of 10^{-4}.

Code Rate	SBNR (dB)	σ_b^2
.333	1.6	1.04
.936	3.6	.233
.7	4.6	.248

Correlated Noise

Correlated noise, in the context of this paper, will be represented as Gaussian noise where adjacent elements are correlated. In application, more elements may be correlated, but that will not be addressed. In the case where a burst is of length one, the noise will not be correlated with any adjacent bits.

The correlation will be introduced using the Cholesky factorization of the autocorrelation matrix and will be applied to each burst individually. The autocorrelation matrix will be tri-diagonal with $\sigma^2 + \sigma_b^2$ in the major diagonal and $\rho\sigma_b^2$ in the other two diagonals, where ρ is the correlation factor. For example, a 4×4 matrix R would appear as

$$R = \begin{bmatrix} \sigma^2 + \sigma_b^2 & \rho\sigma_b^2 & 0 & 0 \\ \rho\sigma_b^2 & \sigma^2 + \sigma_b^2 & \rho\sigma_b^2 & 0 \\ 0 & \rho\sigma_b^2 & \sigma^2 + \sigma_b^2 & \rho\sigma_b^2 \\ 0 & 0 & \rho\sigma_b^2 & \sigma^2 + \sigma_b^2 \end{bmatrix}. \tag{2.1}$$

Note that in the variance on each bit, the variance is the combination of the burst and non-burst variance, as in the uncorrelated noise. However, in the correlated elements, only the burst noise is accounted for, since only the noise from the burst is correlated. The values for σ_b^2 are the same as for the uncorrelated burst as listed in Table 2.2. The value for the correlation factor ρ is set to .5 to show a weak correlation between the observations.

For the mobile wireless channel, when the transmitter is stationary, the channel response is highly correlated [26]. This could result in a burst of highly correlated noise over the time period in which it is stationary. Since many burst phenomena are generated by singular events acting over a short period of time on the transmitted signal, correlation could occur in other bursts as well.

2.2.3 Burst Model Characterization

Next, we must define the parameters of the burst noise model. In the two-state Markov chain used to generate the burst state information, there are three parameters that can

be selected that determine the statistics of the burst. The first two are the transition probabilities. A Markov chain can be represented as in Figure 2.2.

It can also be defined by a transition matrix as

$$P = \begin{bmatrix} p_{0,0} & p_{0,1} \\ p_{1,0} & p_{1,1} \end{bmatrix} = \begin{bmatrix} 1-p & p \\ q & 1-q \end{bmatrix}. \tag{2.2}$$

In (2.2), p and q represent the transition probabilities as in Figure 2.2, while the $p_{i,j}$ are the same transition probabilities expressed as going from state i to j. Selection of p and q defines the transition matrix P and and also the third value, burst time.

Burst time is the average amount of time that the process spends in state 1, the burst state. This is also referred to as the steady state probability of being in the burst state. This steady state probability can be found by

$$\pi_j = \sum_i \pi_i P_{ij},$$

and

$$1 = \sum_i \pi_i,$$

where the π_i are the fractions of time in each state [31]. Switching the $p_{i,j}$s for p and q, the equations for finding the π_i for a two-state Markov chain are

$$\pi_0 = \pi_0(1-p) + \pi_1 q, \tag{2.3}$$

and

$$1 = \pi_0 + \pi_1. \tag{2.4}$$

Since π_1 is the number of most interest, we can manipulate (2.3) and (2.4) to find π_1 as

$$\pi_1 = \frac{1}{1+\frac{q}{p}}. \tag{2.5}$$

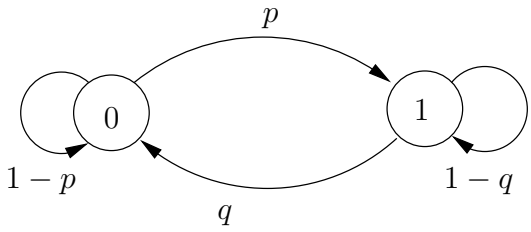

Fig. 2.2: Two-state Markov chain.

Since p and q are not directly related, any arbitrary values can be chosen and the resulting π_1 can be found. This approach is useful if the burst transition probabilities are known or can be estimated. Since the models being used do not represent specific channel properties, it is more useful to select a burst time π_1 and the transition probability q. This selects how much time the process spends in the burst and allows for some control on the average length of the bursts. We then solve (2.5) for p, which is found as

$$p = \frac{\pi_1 q}{1 - \pi_1}. \tag{2.6}$$

For use in testing, burst times of 1%, 2%, and 5% total length were chosen. This provides some variation in the amount of burst noise that is added without overwhelming the decoder with the additional noise.

Several values for q were selected and tested to see how it would affect the length of the bursts. A value of $p = .2$ was chosen because it gives an average burst length of about 5 symbols and generally provides a large number of bursts as opposed to a few extremely long bursts. Using (2.6) and the selected value of p, we can find the transition matrices for each chosen burst time as

$$P_{1\%} = \begin{bmatrix} .998 & .002 \\ .20 & .80 \end{bmatrix}, \tag{2.7}$$

$$P_{2\%} = \begin{bmatrix} .996 & .004 \\ .20 & .80 \end{bmatrix}, \tag{2.8}$$

$$P_{5\%} = \begin{bmatrix} .989 & .011 \\ .20 & .80 \end{bmatrix}. \tag{2.9}$$

In order to test the Markov model for the burst and the values that were selected, 100,000 test state samples were tested for each of the burst lengths. Table 2.3 shows the results for the estimated and actual burst times, as well as the average burst length.

A histogram of the burst lengths from the burst sample is shown in Figure 2.4. The burst length distribution is a decaying exponential with an average burst length of approximately 5. The output of the burst noise algorithm with the transition matrices chosen are close to this length regardless of burst time, as shown in Table 2.3. The results were satisfactory for the targeted performance.

Figure 2.3 shows a 1000 bit sample of the resulting burst locations for 5% burst time. The burst state occurrences are represented by the value 1 and are spread across the sample fairly evenly.

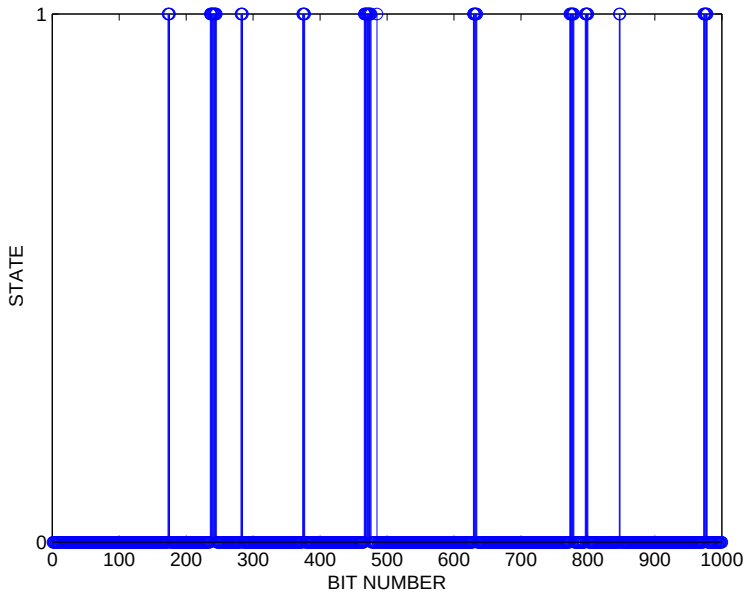

Fig. 2.3: 1000 samples from burst distribution across a 100,000 state sampling, 5% burst time.

Table 2.3: Estimated and actual burst times and average burst length for 100,000 bits.

Estimated Burst Time	Actual Burst Time	Average Burst Length
1.0 %	1.012 %	5.4 Samples
2.0 %	2.062 %	4.9 Samples
5.0 %	5.229 %	5.2 Samples

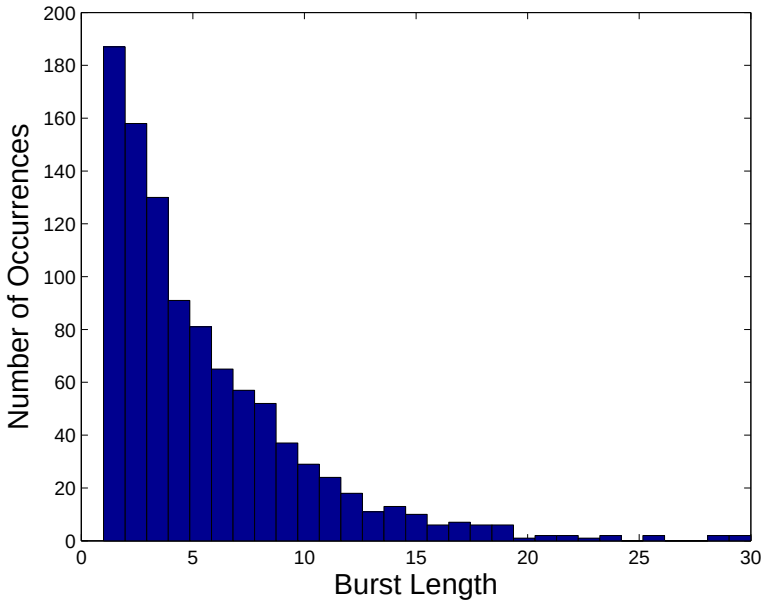

Fig. 2.4: Histogram of burst lengths for 100,000 bits, 5% burst time.

2.3 LDPC Codes with Burst Noise

Regular decoding assumes that the noise seen on the transmitted message is AWGN and has the same distribution on all bits. This does not account for any type of burst noise. In order to see some of the effects of the burst noise on decoding performance, probability of bit error curves for each code were generated with burst noise added. Each type of burst noise will be added with the variance estimated as the variance for the AWGN without the burst.

Figure 2.5 shows the performance for the rate .333 code with 1% burst time. This plot is typical of all the codes at the different burst rates. In examining the code performance, it is worth considering the expected performance. The correlated and uncorrelated burst noise are not expected to differ, since the correlation tends to make the magnitudes stay closer

together in correlated observations, but the overall magnitudes are drawn from random variables with the same variance. It is easy to observe in the plot that there is little difference between the correlated burst and uncorrelated burst performance. Since this is the case, the remaining data will be presented with only the correlated burst case shown. All three burst times will be shown for comparison.

As far as actual code performance, the expectation is that as the SNR increases, the code performance should improve. The addition of the burst noise at a constant SBNR should introduce some kind of noise floor resulting from the constant extra noise. Since the burst noise is not on all the bits in the codeword, it is not clear at what probability of bit-error it will be.

The actual performance, as shown in Figures 2.6 through 2.8, improves with increasing SNR, as expected. However, before settling to a constant probability of bit-error, as would be expected given that the burst variance is constant, it first decreases to a minimum and then increases before leveling off at a constant probability of bit-error. Further, this constant level is higher than one would expect, since if the burst noise alone was added to the entire codeword, the expected performance should be at probability of bit-error of 10^{-4}. Instead, with the burst length only a small portion of the codeword, we get much higher probability of bit-error.

The reason for the decrease before coming back up is the interaction between the burst noise and non-burst noise. The zero-mean noise values for both the burst and non-burst noise take on both positive and negative values with equal likelihood, due to the Gaussian nature of the noise. Then, on the bits with burst and non-burst noise, there are four potential combinations of signs that occur with equal likelihood. On average, half of the combinations will result in the noise combining additively and the other half the noise will combine destructively.

In regions where the non-burst noise dominates, this will not have a large impact on the noise, as there will be many errors from the non-burst noise and from the burst regions where the noise constructively adds. Here, the curve will be dominated by random errors.

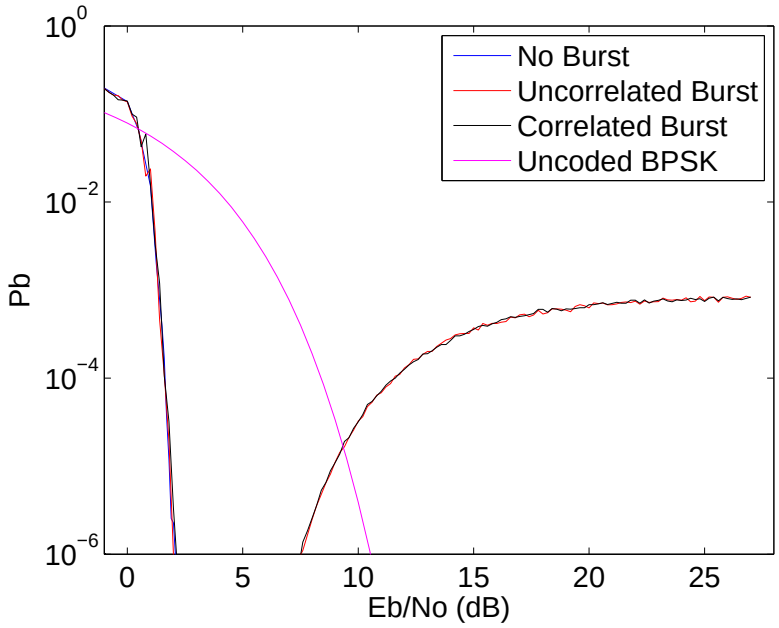

Fig. 2.5: Bit-error rate curve for rate .333 code with 1% burst time and SBNR of 1.6 dB.

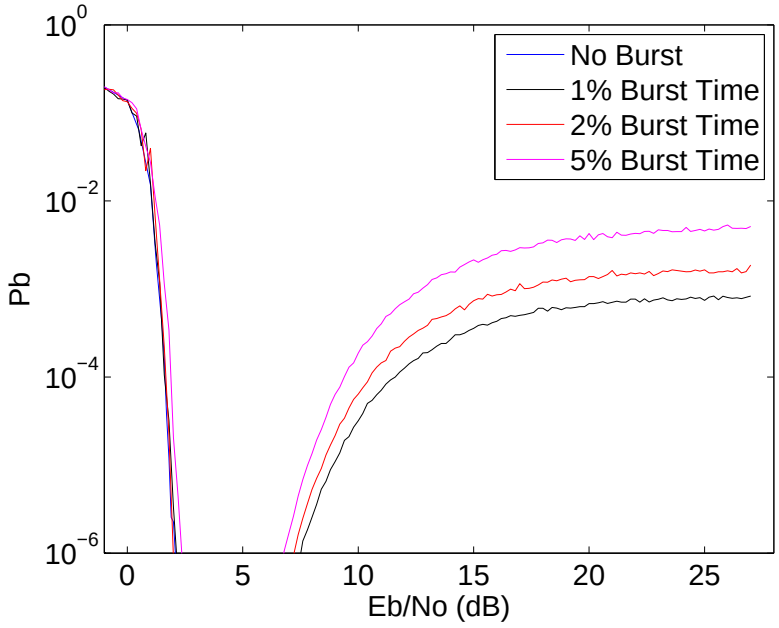

Fig. 2.6: Bit-error rate curve for rate .333 code with SBNR of 1.6 dB.

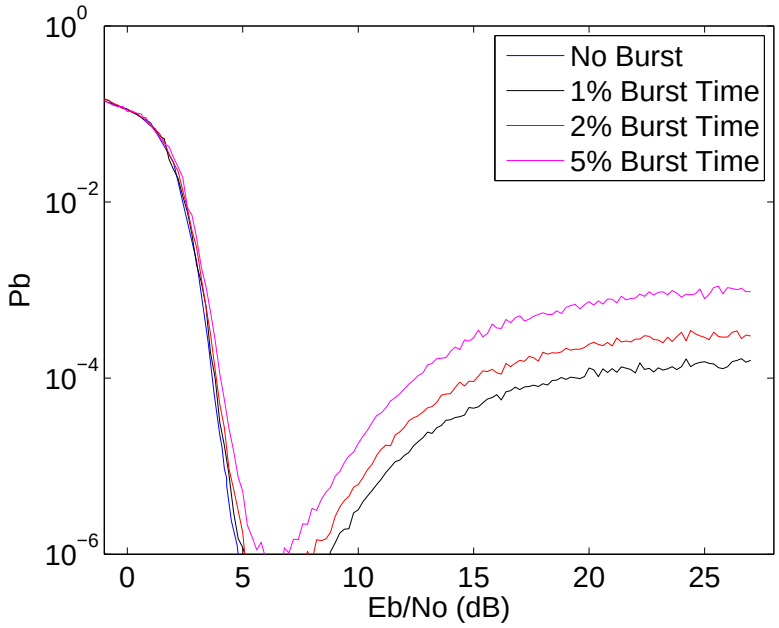

Fig. 2.7: Bit-error rate curve for rate .700 code with SBNR of 4.6 dB.

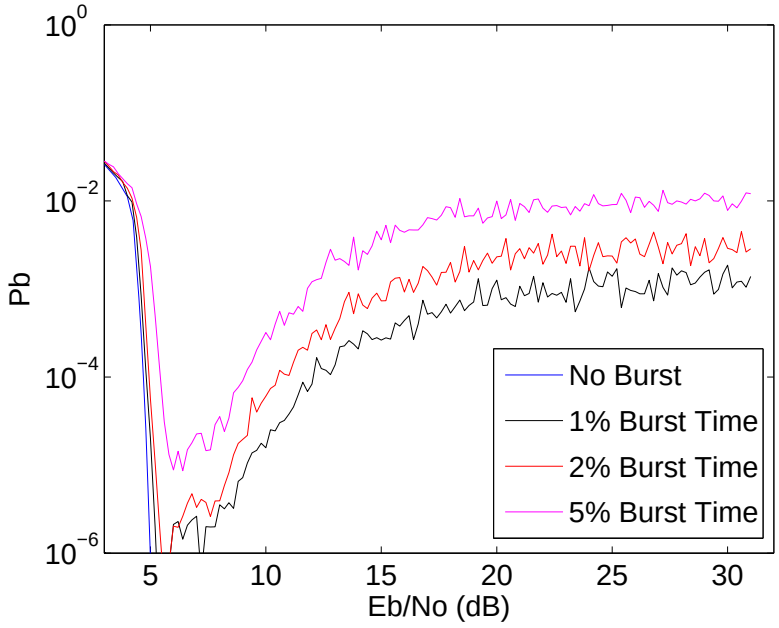

Fig. 2.8: Bit-error rate curve for rate .936 code with SBNR of 3.6 dB.

However, when the burst noise and non-burst noise are close in magnitude, the destructive combinations will actually reduce the amount of errors and overall noise, especially considering that in these regions of the plot, the non-burst noise has dropped to levels that the code can normally correct the errors it causes. This results in the code performing better and dropping below the noise floor that the burst noise would normally cause.

As the burst noise begins to dominate, the regular noise does not reduce the burst noise in the destructive combinations enough to remove errors. As this effect continues to decrease, the burst noise is the source of almost all the errors, so the probability of bit-error tends to a constant level of noise.

Since the SNR is high here, the prior probabilities effectively amount to hard decisions based on a threshold, ignoring the effects of burst noise. If a significant number of bits are flipped by the burst, this can result in decoding to another codeword, which greatly increases the amount of error, as seen in the higher than expected noise floor. This reflects what is seen in the data plots.

The other main observation is that the noise floor increases as burst time increases. This is expected, as the amount of noise added by the burst increases with burst time.

In the papers reviewed, this effect was not seen. For example, in papers [2, 7, 9, 10, 12, 32, 33], the authors utilized burst models that had burst noise in the burst state and no noise otherwise. However, we have noted that the effect results in the interaction between the burst and non-burst noise. In actual practice, burst and non-burst noise can be present at the same time and usually are, so this suggests that our model is more complete.

However, since this type of model has not been investigated in other works, the proposed interpretation of the performance does bear some extra investigation. As a test, the code was run with the same burst model, except that the burst noise variance was a multiple of the non-burst noise variance. In this case, the code was run with the burst variance set at four times and 100 times the non-burst variance. Each code displays some different characteristics that are useful to identify, both for our investigation of the unusual probability of bit-error curves and for future evaluation of the code performance.

In Figure 2.9, we see that the curve for the four times burst variance looks similar to the curves for the constant burst noise, except that it rolls off at the end as the burst noise decreases with the non-burst noise. All the codes display this similar roll off, which is expected with the decreasing noise. The 100 times burst variance has a slightly different shape, staying flat, decreasing a little, going flat again, and then rolling off. Considered in light of the interaction between the burst and non-burst noise, we can see that what is happening is, at first, the random errors dominate in the first flat region. As the non-burst noise decreases, the burst noise starts to become the main source of errors. In this burst error-dominated region, the burst variance continues to decrease until it has decreased enough for the code to correct the errors and the probability of bit-error rolls off.

It is interesting to note that in the intermediate region, the probability of error decreases and levels off before the burst errors begin to be corrected by the code. Looking at Figures 2.10 and 2.11, there is a similar change for the 100 times variance curves. However, as the code rate increases, the probability of bit-error at which the code levels off increases.

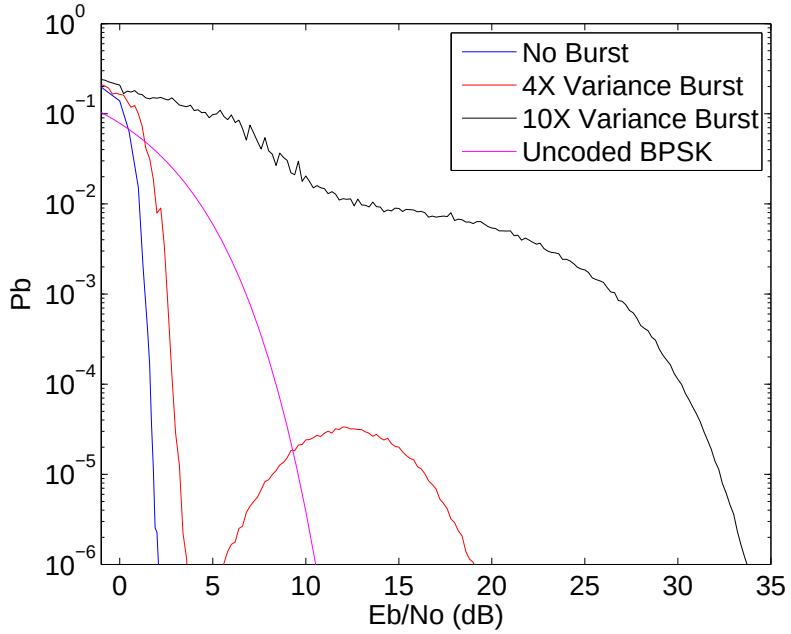

Fig. 2.9: Bit-error rate curve for rate .333 code with 5% burst time and scaled burst variance.

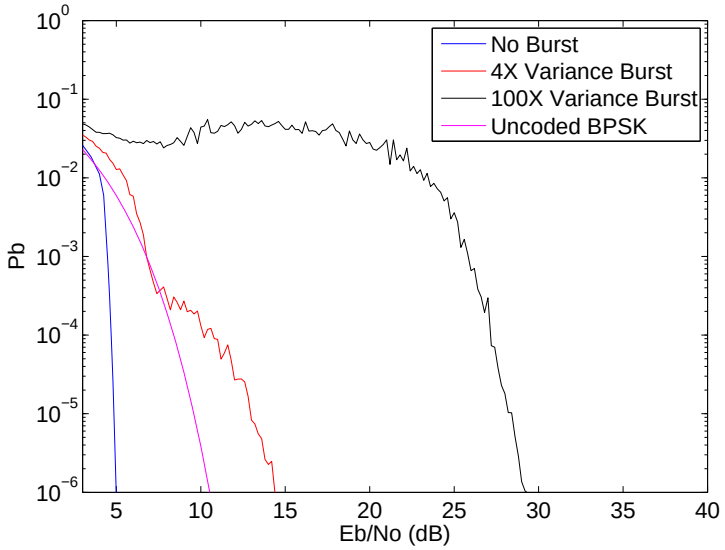

Fig. 2.10: Bit-error rate curve for rate .936 code with 5% burst time and scaled burst variance.

For the two higher rate codes, the four times variance curves have a similar shape. Both decrease, then change shape, jutting out slightly when the burst noise takes over, then continuing to roll as the code overcomes the burst noise. This contrasts with the constant burst performance and the .333 rate code, which dip down and then come back up. In Figure 2.8, this dip does not go as far down in probability of bit-error, which, combined with Figure 2.10, suggests that this is due to the increased code rate. Figure 2.7 shows a slight increase in the bottom of the dip for the .700 rate code as well, which combined with Figure 2.11 supports this idea. Since the higher code rate means poorer decoding performance, general coding theory supports this idea.

An interesting observation is that, in spite of this improved performance in the transition regions between non-burst and burst noise dominated errors, looking at Figures 2.9 through 2.11, the point at which the code actually reaches probability of bit-error of 10^{-6} is at higher SNR for the lower rate codes. The .333 rate code crosses at about 19 dB for the four times variance curve and 34 dB for the 100 times variance curve, while the .700 rate code crosses at 15 dB and 30 dB and the .936 rate code crosses at 14 dB and 29 dB. The overall performance improves with code rate, which is contrary to the expectation.

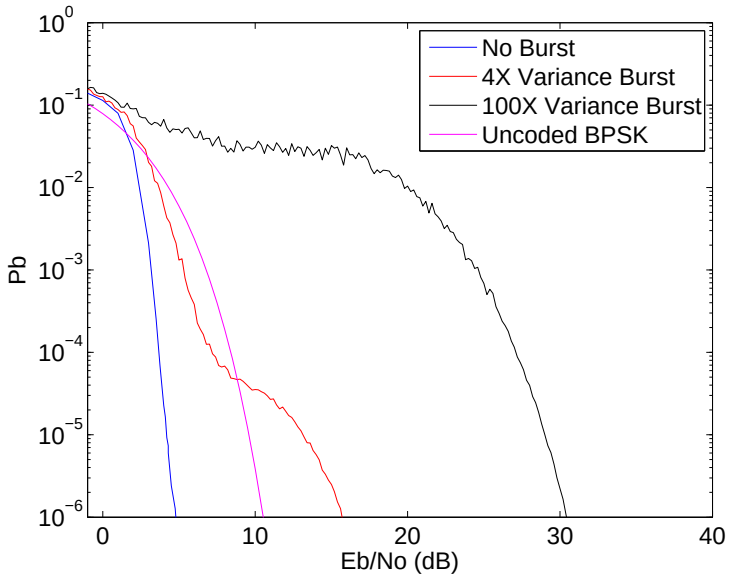

Fig. 2.11: Bit-error rate curve for rate .700 code with 5% burst time and scaled burst variance.

Combining this with the previous observation in the transition region suggests that lower code rates deal well with the random error-dominated region, but that the higher rate codes has slightly better performance in the burst error-dominated region.

Regardless of these other observations, the performance of the code with SNR-varying SBNR supports the interpretation of the curve shapes that was observed with the constant SBNR curves. Understanding that LDPC codes perform poorly in the presence of burst noise, we now move on to an approach to compensating for the burst noise.

Chapter 3

Burst Error Compensation

Techniques do exist to improve code performance when burst noise is present. One of the most common, interleaving, is used as a baseline for performance against our burst model.

3.1 Interleaving

Interleaving is commonly used with codes in order to combat burst errors. As discussed in the introduction, a group of messages can be encoded and then shuffled together to form a new group of messages of the same length. Each new message will have some of the symbols from each of the original group of messages. The messages can be sent, un-mixed, and decoded. The mixing takes place according to a mapping between all the messages, which is what allows the messages to be un-mixed [10, 11]. The first advantage of doing this is that, for cases where there is correlated noise-whether burst or otherwise-the correlation is broken up within a message. This can be useful to codes where there is an assumption of no correlation between elements. The second advantage is seen when a noise burst occurs on only one message in a group as it is sent through the channel. In this case, the interleaving allows that burst of errors to be distributed amongst the group during the de-interleaving. This usually allows the error correcting code to correct the errors, which it might not be able to do if the errors were all in one message [10, 11].

A simple interleaving pattern for twenty messages was developed randomly using a random shuffling algorithm [34]. Twenty messages were interleaved together and passed through the same channels. The burst model developed before was utilized with the same constant burst variances and burst time percentages as developed in Chapter 2.

A baseline plot is given in Figure 3.1 and shows the rate .333 code with 1% burst time.

There is no difference between correlated and uncorrelated performance. This plot is typical for all the codes and burst times. As with the baseline data, the different code rates are shown separately with the correlated data shown and the burst time varying.

The results shown in Figures 3.2 through 3.4 show that the interleaving performs about the same as the code with no interleaving. As a comparison, an example from each code with the 5% is compared for interleaved versus non-interleaved.

The comparison, seen in Figures 3.5 through 3.7, shows that there is effectively no difference between the performance with and without interleaving.

This may come as a surprise until the burst model is considered. The bursts that are being considered are distributed across all the messages, so interleaving does not provide any benefit in this case. If the burst, even with our model, was only on one message, the interleaving would effectively reduce the burst percentage to one-twentieth of what it was before, which should provide some improvement, as we can see better performance with some of the codes when decreasing the burst time from 5% to 1%. Regardless, given the burst model that we are using, interleaving provides no added benefit.

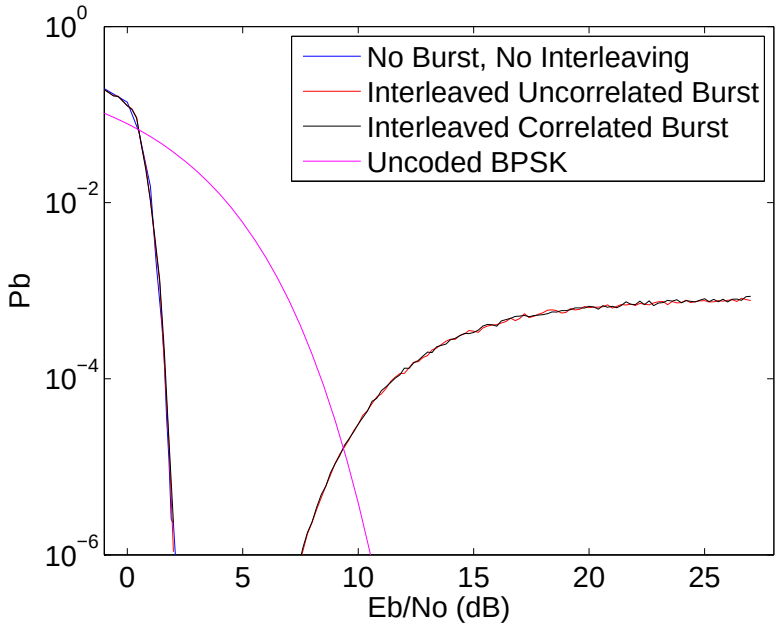

Fig. 3.1: Bit-error rate curve for interleaved rate .333 code with 1% burst time and SBNR of 1.6 dB.

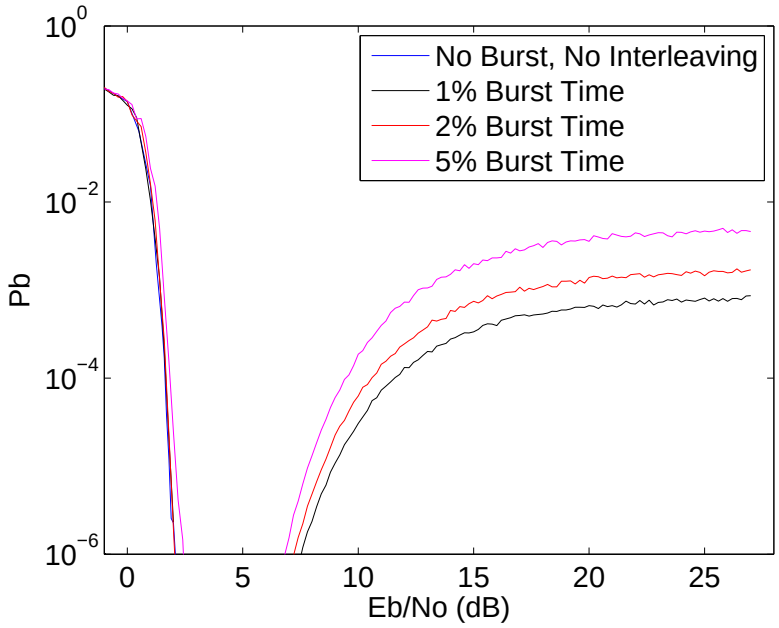

Fig. 3.2: Bit-error rate curve for interleaved rate .333 code with SBNR of 1.6 dB.

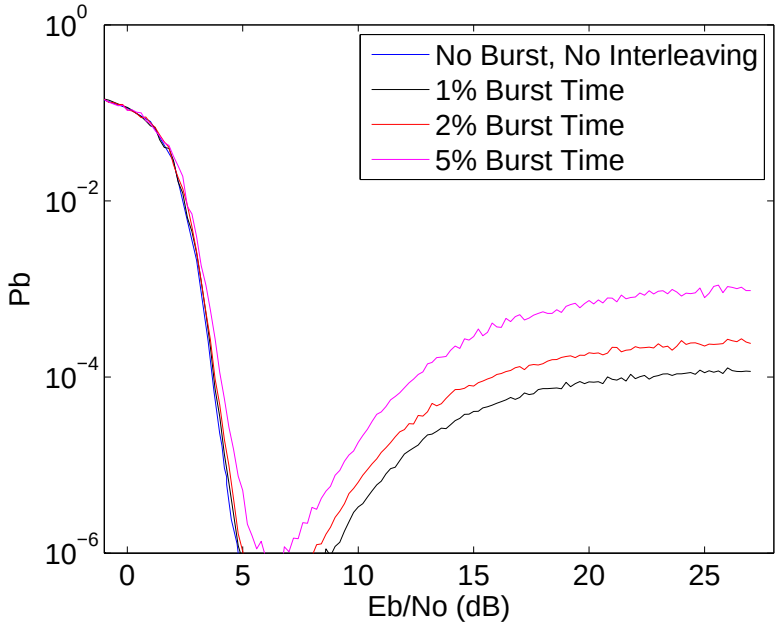

Fig. 3.3: Bit-error rate curve for interleaved rate .700 code with SBNR of 4.6 dB.

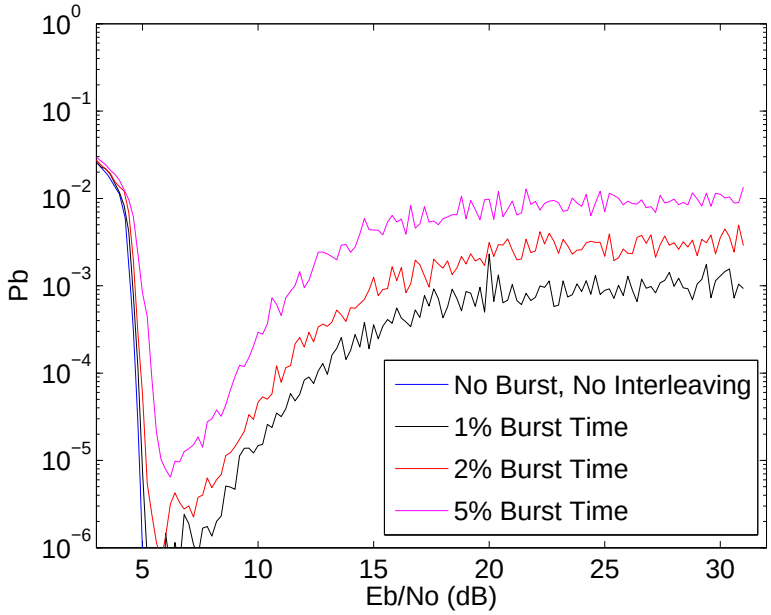

Fig. 3.4: Bit-error rate curve for interleaved rate .936 code with SBNR of 3.6 dB.

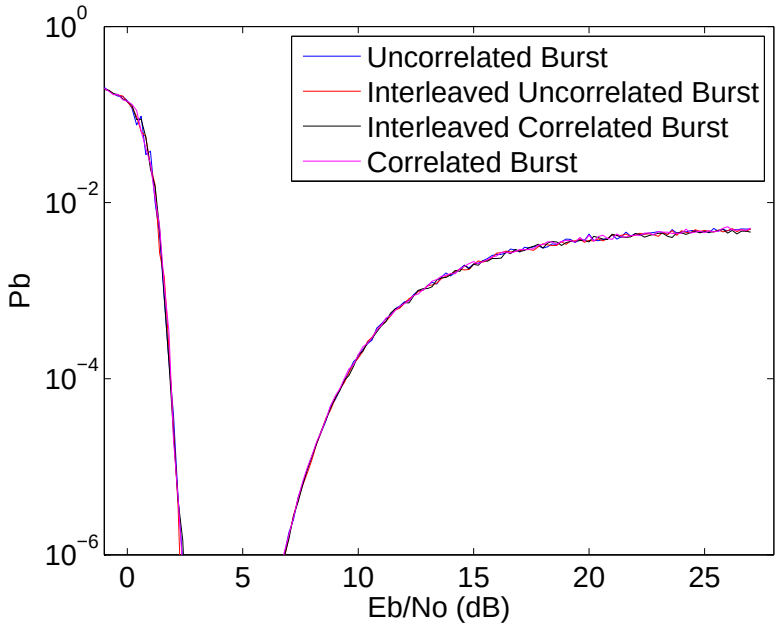

Fig. 3.5: Interleaved versus non-interleaved for rate .333 code with 5% burst time and SBNR of 1.6 dB.

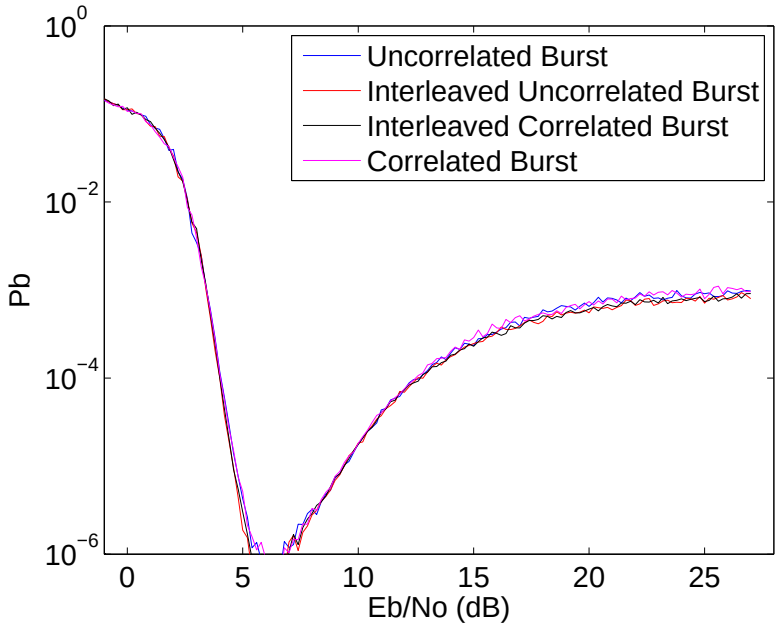

Fig. 3.6: Interleaved versus non-interleaved for rate .700 code with 5% burst time and SBNR of 4.6 dB.

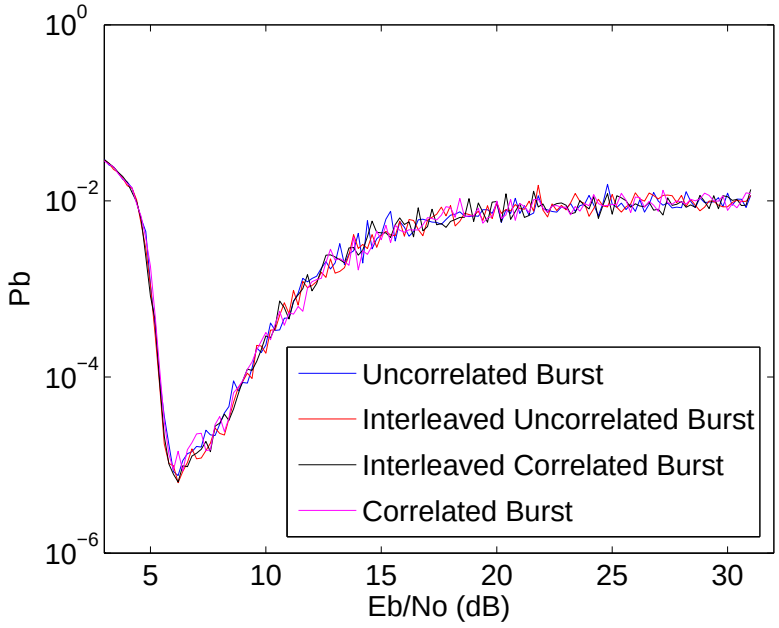

Fig. 3.7: Interleaved verus non-interleaved for rate .936 code with 5% burst time and SBNR of 3.6 dB.

3.2 *Lying* to the Decoder

All the tests that we have run so far have not accounted for the burst noise. Even with interleaving, we only mixed up the values. But where does this actually impact the decoder? The difference between codewords with burst noise and without burst noise is the amount of noise, as measured by the variance of the noise. The variance on the non-bursty bits is σ^2. However, for all the bits where burst noise is present, the variance would actually be $\sigma^2 + \sigma_b^2$ and, in the correlated case, will be correlated with adjacent bursty bits. The variance value is only used in calculation of the prior probabilities which are input into the decoder.

Examining the formula for the prior probability provides some insight into what is going on. The calculation of the prior probability of having sent the signal $x_t = 1$, given the observation r_t, is given by

$$p(x_t = 1|r_t) = \frac{1}{1 + e^{-2r_t/\sigma^2}}, \tag{3.1}$$

which was derived by Moon [18]. In this case, the bit values are modulated to 1 and -1. If we assume that all bits are non-bursty, as we have, we are essentially lying to the decoder about the probability of the observations in a burst being a 1.

Investigating the effects of using incorrect variance shows some of the dangers of using the wrong variance to calculate prior probabilities. As an example, the prior probabilty of several values for r_t were calculated with variances $\sigma^2 = 1$, $\sigma^2 = .1$, and $\sigma^2 = .01$, and the combined $\sigma^2 + \sigma_b^2$ using

$$p(x_t = 1|r_t) = \frac{1}{1 + e^{-2r_t/\sigma_p^2}}. \tag{3.2}$$

This is the prior probability calculation that is typically used for an LDPC code [18]. In (3.2), σ_p^2 indicates that different values of p will indicate the variance that is being used. The burst variance values for σ_b^2 are taken from Table 2.2 and added to σ^2. For all the calculations, it is assumed that a modulated 1 is sent.

We can see in Table 3.1 that using a different variance changes the calculated prior.

For example, with an observation of .01 and a variance of $\sigma^2 = .01$, there is high probability of it being a 1. This is a result of insufficient noise to flip the transmitted signal from the sent 1. However, adding in the burst variance in every case pushes the probability close to .5, which reflects very low confidence, which would be the case with the much higher variance. This also supports the observations in Chapter 2, where the code performance begins to degrade when the burst noise is much larger than the non-burst noise.

While not all the changes are so drastic, the burst noise does affect the prior probabilities. This, in turn, impacts decoding. Effects occur across the length of the message, as seen in the results above and in Chapter 2.

Further, we recognize that in the variable burst case, the poorer performance should be expected. This is due to the burst noise being significantly larger than the non-burst noise throughout the entire performance curve. This will create less confidence on the bits close to the decision threshold.

Another point to note about the results in Table 3.1 is that there is high confidence in the observations when they are more negative than a 0. If the burst noise were large enough to switch a sent 1 to even -2, there is such high confidence that it was a sent 0 that it would be difficult for a decoder to change it.

Table 3.1: Prior probabilities for various observations and variances.

Variance $\sigma^2 + \sigma_b^2$	Observation Value r_t			
	.75	.01	-2	-10
1	.8176	.5050	.0180	$2*10^{-9}$
1+1.04	.6760	.5025	.1234	$5*10^{-5}$
1+.233	.7715	.5041	.0375	$9*10^{-8}$
1+.248	.7689	.5040	.0390	$1*10^{-7}$
.1	1.0	.5498	$4*10^{-18}$	$1*10^{-87}$
.1+1.04	.7885	.5044	.0291	$2*10^{-8}$
.1+.233	.9891	.5150	$6*10^{-6}$	$8*10^{-27}$
.1+.248	.9867	.5144	10^{-5}	10^{-25}
.01	1.0	.8808	$2*10^{-174}$	0 (rounded)
.01+1.04	.8067	.5048	.0217	$5*10^{-9}$
.01+.233	.9979	.5206	$7*10^{-8}$	$2*10^{-36}$
.01+.248	.9970	.5194	$2*10^{-7}$	$2*10^{-34}$

3.2.1 Calculating Priors Using Burst Variance

As an experiment, the codes were run exactly the same as in Chapter 2, except that the burst variance, or $\sigma^2 + \sigma_b^2$, was used in the calculation of the prior probabilities. This will actually result in more mistakes in the prior calculations since more of the bits will not be in the burst, but it will provide correct prior probabilities for the bits with the most noise.

Figures 3.8 through 3.10 are typical results for the code performance when calculating priors with the burst variance. Only the correlated bursts with 5% burst time are shown, as there is little variation between the different burst times and the correlated versus uncorrelated data. The results show that the performance is almost identical in the random error-dominated region, but there is no increase of probability of bit-error as the transition to the burst error-dominated region occurs. This is quite a bit of improvement over the performance with no burst compensation. The main difference is there is a some loss in gain where the curve starts to roll. This is due to assuming that the variance is higher. In effect, the decoder is being told that there is more noise, so there is less confidence in the non-burst observations, thus decreasing the code performance. The performance is still better than without any compensation, seeing as there is no longer a noise floor, so the coding gain can actually be measured.

The coding gain for using the burst variance are listed in Table 3.2, along with the non-burst coding gain for comparison. We can see that there is some loss in the gain, but it is on the order of 1 dB. When we consider that there is extra noise being added in the form of burst noise, this should be expected.

For comparison, the scaled variance code is again used, with the .333 rate code as an example. Figures 3.11 and 3.12 show the performance for the four times variable burst variance and the 100 times variable burst variance. Here the loss of coding gain is much more pronounced. The amount of noise at SNR of 3 dB, which is about where the .333 rate coding gain is measured for the burst variance calculated priors case, for an input of 1 is a variance of $\sigma^2 = .752$. For the .333 rate code, the burst noise is only about 1.4 times larger.

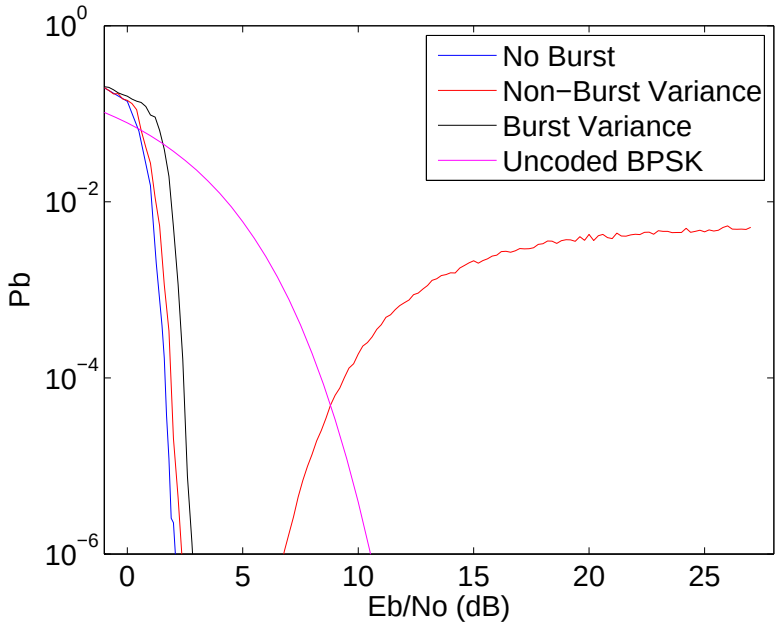

Fig. 3.8: Comparison between non-burst and burst variance prior calculations for .333 code with 5% burst time, correlated burst and SBNR of 1.6 dB.

When we move up to four times larger, as in Figure 3.11, the performance has dropped significantly. It takes longer for the code to correct the errors due to the increased noise. This increase is due both to the noise that is present in the burst and the noise that we are telling the decoder is present by the variance value chosen.

While a direct comparison is not really useful, given the changing ratio between the burst and non-burst noise in the constant burst case, it does suggest that if the burst noise is larger, there will be more coding gain lost if we use the burst variance to calculate the prior probabilities.

Table 3.2: Coding gain for three LDPC codes with and without burst noise.

Code Rate	Non-Burst Coding Gain	Coding Gain for 5% Correlated Burst
.333	8.4	7.7
.936	5.5	4.1
.7	5.7	5.0

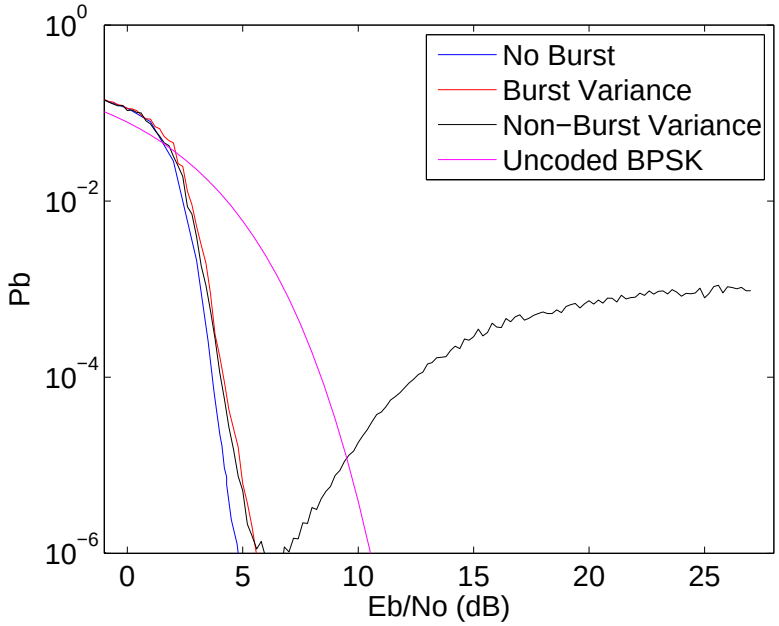

Fig. 3.9: Comparison between non-burst and burst variance prior calculations for .700 code with 5% burst time, correlated burst, and SBNR of 4.6 dB.

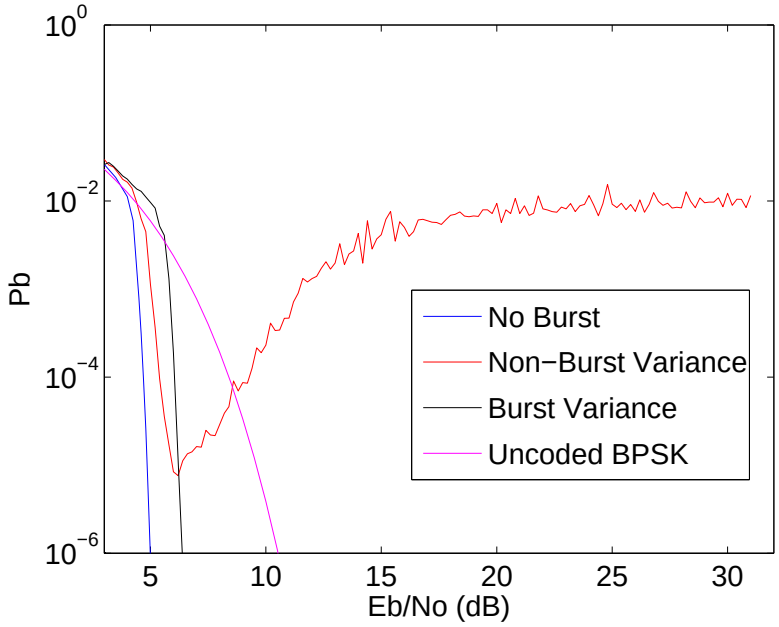

Fig. 3.10: Comparison between non-burst and burst variance prior calculations for .936 code with 5% burst time, correlated burst, and SBNR of 3.6 dB.

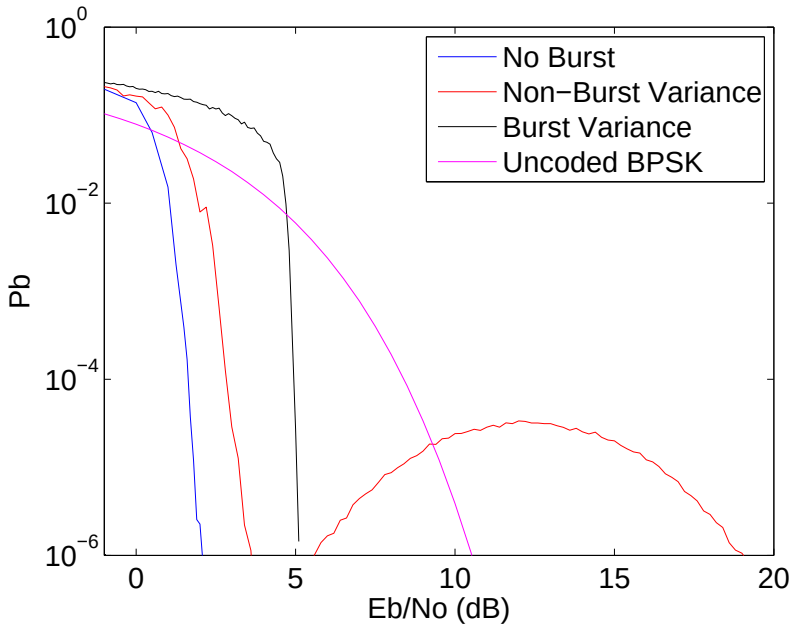

Fig. 3.11: Comparison between burst and non-burst calculated priors for rate .333 code with 5% burst time, correlated burst, and four times burst variance.

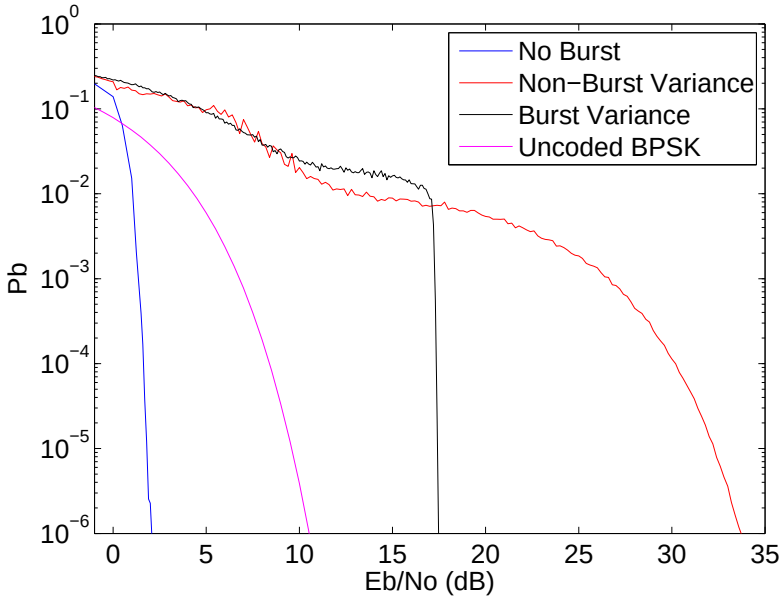

Fig. 3.12: Low and high estimates for variance for rate .333 code with 5% burst time, correlated burst, and 100 times burst variance.

3.2.2 Conclusions on Lying

The use of the non-burst variance in prior probability calculations has a negative effect on the performance. Some improvement can be made by using the higher burst variance, which allows for the code to overcome the burst noise floor, assuming that the burst length is short enough for the code to correct. This appears to be the case for the test cases. There is some loss in coding gain when the burst variance is used to calculate the prior probabilities, which is a negative effect, but is preferable to the code being unuseable, as in the non-burst variance case. In general, lying to the decoder degrades code performance, but using the burst variance at least makes it usable, which suggests that some lies are better than others, where decoding is concerned.

3.3 Proper Prior Calculation

Some research has been done on LDPC code performance, without bursts, when the variance is varied from the true value. The variance can be set to a value that optimizes code performance. Depending on the code and the channel, this value may vary [35]. While this does not apply directly to our case, it does suggest that we could adjust the variance based on the burst and non-burst values and find an optimal value. This could improve performance over what we saw in the burst variance case.

However, even if this optimized the performance, there might still be some loss in coding gain. Considering the extra noise that is present in the burst, this may be acceptable. However, the research on varying the variance shows that only minor variations have a positive effect. As the SNR mismatch increases, the code begins to perform more poorly [35]. It seems then, that when the burst variance and non-burst variance are far apart, some performance is being left on the table.

Instead of using only a single variance, we propose using variable calculations that take into account the characteristics of the noise on each observation. This will result in correct priors being provided across the code and should improve performance.

3.3.1 Priors for Uncorrelated Burst Noise

Utilizing the correct variance for calculating the prior probabilities requires different calculations for bits in the burst and bits not in the burst. Since the only difference between the two is the variance, we can use (3.2). Here, p is the state that the system is in at time t. The value of σ_p^2 is determined by p, with a 0 indicating to use the non-burst variance and a 1 indicating to use the burst variance. The calculation is relatively straight forward and simply requires knowledge of the state.

3.3.2 Priors for Correlated Noise

For correlated noise, more significant modifications to the prior calculations are required. This is due to the effects of the correlation on the derivation of the decoder.

The decoder starts with $q_t(c)$, the *pseudoposterior* probability of the observation sequence x. This is given as

$$q_t(c) = \frac{1}{P(\{z_m = 0, m \in \mathcal{M}_t\})} P(x_t = c|\mathbf{r}) P(\{z_m = 0, m \in \mathcal{M}_t\}|x_t = c, \mathbf{r}), \tag{3.3}$$

as developed by Moon [18]. In (3.3), the uncorrelated derivation assumes that the bit and observations are independent excepting the one at the same time, so the conditional probability $P(x_t = c|\mathbf{r})$ is reduced to $P(x_t = c|r_t)$ since the bits are independent. This is the only factor in (3.3) with which we need be concerned, so we will focus on it.

For correlated noise, we cannot exploit the independence in the same way. Instead, depending on the state, there may be correlation between adjacent observations, so this sets up a slightly more complicated case system. When no adjacent observations are in the burst or when not in the burst state at the current time, we are still able to break up the factor. Thus, we can continue to use (3.2) as before, relying on the current state to determine the variance value.

When in the burst state and only one adjacent observation is in the burst state, we neglect all except for the adjacent observation. For example, when the previous observation

is correlated with the current observation, we have

$$P(x_t = c|\mathbf{r}) = \sum_{i\in\{-1,1\}} p(x_t = c, x_{t-1} = i|r_t, r_{t-1}).$$

Conditional factoring allows us to change this to

$$\sum_{i\in\{-1,1\}} p(x_t = c, x_{t-1} = i|r_t, r_{t-1}) = \frac{\sum_{i\in\{-1,1\}} p(r_t, r_{t-1}|x_t = c, x_{t-1} = i)p(x_t = c)p(x_{t-1} = i)}{p(r_t, r_{t-1})}.$$

In order to calculate the denominator, we must introduce the bit values for the observations, so we sum them in and conditionally factor to get

$$\frac{\sum_{i\in\{-1,1\}} p(r_t, r_{t-1}|x_t = c, x_{t-1} = i)p(x_t = c)p(x_{t-1} = i)}{p(r_t, r_{t-1})} = \frac{\sum_{i\in\{-1,1\}} p(r_t, r_{t-1}|x_t = c, x_{t-1} = i)p(x_t = c)p(x_{t-1} = i)}{\sum_{j\in\{-1,1\}}\sum_{k\in\{-1,1\}} p(r_t, r_{t-1}|x_t = j, x_{t-1} = k)p(x_t = j)p(x_{t-1} = k)}.$$

Since the prior probabilities for the sent bits are equal and do not change with time, we can factor them out and cancel them in the fraction, resulting in

$$\sum_{i\in\{-1,1\}} p(x_t = c, x_{t-1} = i|r_t, r_{t-1}) \frac{\sum_{i\in\{-1,1\}} p(r_t, r_{t-1}|x_t = c, x_{t-1} = i)}{\sum_{j\in\{-1,1\}}\sum_{k\in\{-1,1\}} p(r_t, r_{t-1}|x_t = j, x_{t-1} = k)}.$$

Each of the terms in the summations can be calculated by using the multi-variate Gaussian likelihood,

$$p(r_t, r_{t-1}|x_t = a, x_{t-1} = b) = \frac{1}{2\pi|R|^{.5}} e^{-\frac{1}{2}(\mathbf{r}-\mathbf{x}_{a,b})^T R^{-1}(\mathbf{r}-\mathbf{x}_{a,b})}.$$

Here, R is the autocorrelation matrix between the two observations which, as given in (3.4).

$$R = \begin{bmatrix} \sigma^2+\sigma_b^2 & \rho\sigma_b^2 & 0 & 0 \\ \rho\sigma_b^2 & \sigma^2+\sigma_b^2 & \rho\sigma_b^2 & 0 \\ 0 & \rho\sigma_b^2 & \sigma^2+\sigma_b^2 & \rho\sigma_b^2 \\ 0 & 0 & \rho\sigma_b^2 & \sigma^2+\sigma_b^2 \end{bmatrix} \tag{3.4}$$

The vector $\mathbf{r}$ is the two observations r_t and r_{t-1} and the vector $\mathbf{x}_{a,b}$ is mean of the observations and depends on the values that are assumed to be sent. For example, if a 1 and a 0 were sent, then $a = 1$ and $b = -1$, so $\mathbf{x}_{1,-1} = [1, -1]^T$. Since the constant coefficient in the likelihood is the same for all the terms, it cancels out and we can find

$$P(x_t = c|\mathbf{r}) = \frac{\sum_{i\in\{-1,1\}} \tilde{p}(r_t, r_{t-1}|x_t = c, x_{t-1} = i)}{\sum_{j\in\{-1,1\}} \sum_{k\in\{-11\}} \tilde{p}(r_t, r_{t-1}|x_t = j, x_{t-1} = k)}, \tag{3.5}$$

where $\tilde{p}(r_t, r_{t-1}|x_t = a, x_{t-1})$ is given by

$$\tilde{p}(r_t, r_{t-1}|x_t = a, x_{t-1} = b) = e^{-\frac{1}{2}(\mathbf{r}-\mathbf{x}_{a,b})^T R^{-1}(\mathbf{r}-\mathbf{x}_{a,b})}. \tag{3.6}$$

If the correlation was between the current time and the next time, simply change r_{t-1} to r_{t+1} and x_{t-1} to x_{t+1} in (3.5) and (3.6).

A similar derivation follows for the case where both the current and both adjacent bits are in the burst. In this case, we find the value of $P(x_t = c|\mathbf{r})$ as

$$P(x_t = c|\mathbf{r}) = \frac{\sum_{i\in\{-1,1\}} \sum_{j\in\{-1,1\}} \tilde{p}_3(r_t, r_{t-1}, r_{t+1}|x_t = c, x_{t-1} = i, x_{t+1} = j)}{\sum_{k\in\{-1,1\}} \sum_{m\in\{-1,1\}} \sum_{n\in\{-1,1\}} \tilde{p}_3(r_t, r_{t-1}, r_{t+1}|x_t = k, x_{t-1} = m, x_{t+1} = n)}. \tag{3.7}$$

For the likelihood part, we find

$$\tilde{p}_3(r_t, r_{t-1}|x_t = a, x_{t-1} = b) = e^{-\frac{1}{2}(\mathbf{r}-\mathbf{x}_{a,b})^T R^{-1}(\mathbf{r}-\mathbf{x}_{a,b})}. \tag{3.8}$$

In this case, the subscript 3 has been added to the $\tilde{p}$ to indicate in (3.7) that this is a three element correlation, so (3.8) should be used.

Having defined the proper calculation of the prior probabilities for the correlated noise, we have covered both of the burst models that we are using and can move on to investigating performance.

3.3.3 Performance Comparison

Having set up the prior calculations, the same codes with the same bursts were run

again. This time, when the burst was generated, it was fed to the prior calculations along with the burst type and used to decide which of the prior formulas should be used.

As an example of the improved performance, the 5% bursts from each code for the correlated and uncorrelated case are included with the baseline, the interleaved code and the code with only the burst variance used to calculate the priors for comparison. Since the comparison cases are the same for both bursts, only the correlated burst is included.

It is easy to see in Figures 3.13 through 3.15 that the performance is much better than the interleaving, as it does not have a noise floor. It appears to be better than using the burst variance for all the priors, but it is hard to see how much.

Figures 3.16 through 3.18 zoom in to better show the comparison. We can see that the correct prior calcultions perform as well as, if not better than using the non-burst variance. They greatly outperform using the burst variance. The interleaving does not compare due to the noise floor, but the performance is better or comparable with the random error-dominated region of the interleaved code. The performance in general is comparable to the performance without the burst, with a small amount of loss. Considering there is more overall noise due to the burst, this is exceptional performance. Table 3.3 compares the coding gain for all the codes.

The results show that there is only a loss of a few tenths of a dB from the non-burst case. The code is at least .5 dB better than using only the burst variance. This is exceptionally good performance. The shorter burst times also show better performance.

Having shown that this technique provides superior performance to using only the burst variance to calculate priors, we recognize that there is one key assumption: that we know the burst location. We may be able to identify that a burst is present by performance loss, but we may not know where exactly it is present. Given that there are expected types of bursts and we may know something about the properties of that burst, we turn to exploiting that information in order to identify the burst locations.

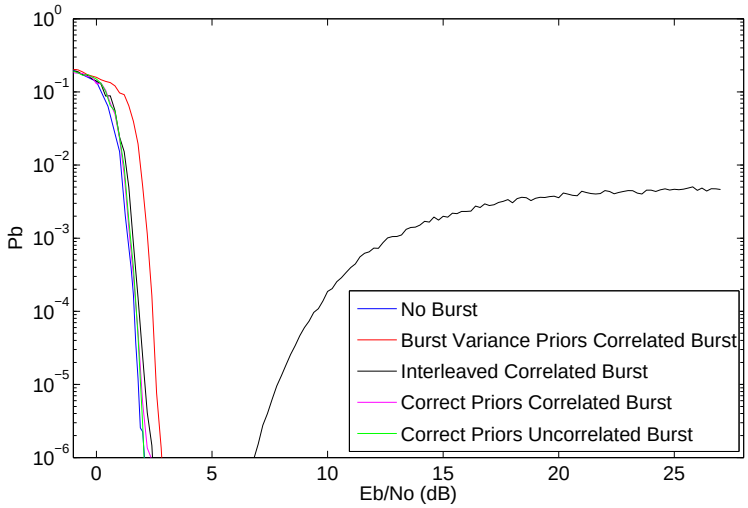

Fig. 3.13: Correct prior calculation comparison for rate .333 code with 5% burst time and SBNR of 1.6 dB.

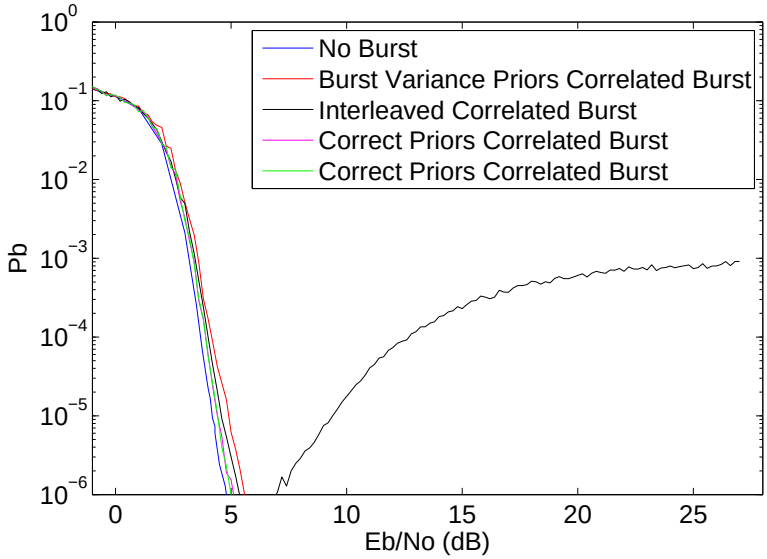

Fig. 3.14: Correct prior calculation comparison for rate .700 code with 5% burst time and SBNR of 4.6 dB.

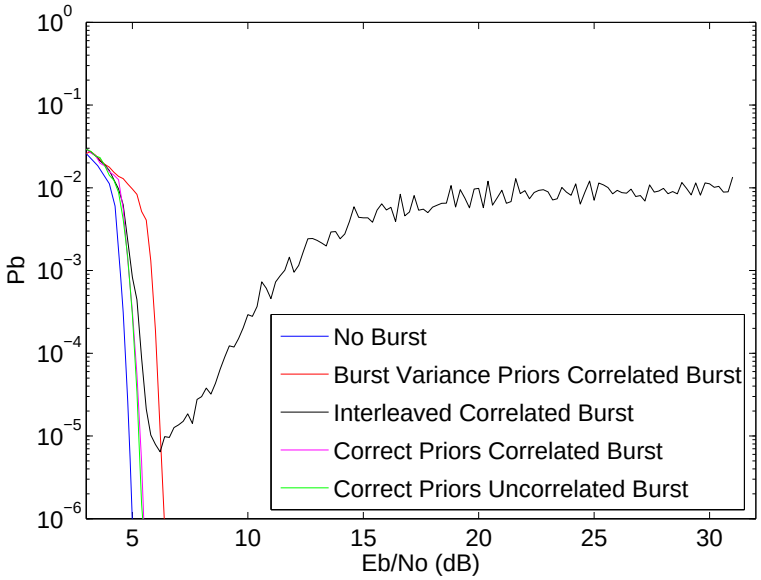

Fig. 3.15: Correct prior calculation comparison for rate .936 code with 5% burst time and SBNR of 3.6 dB.

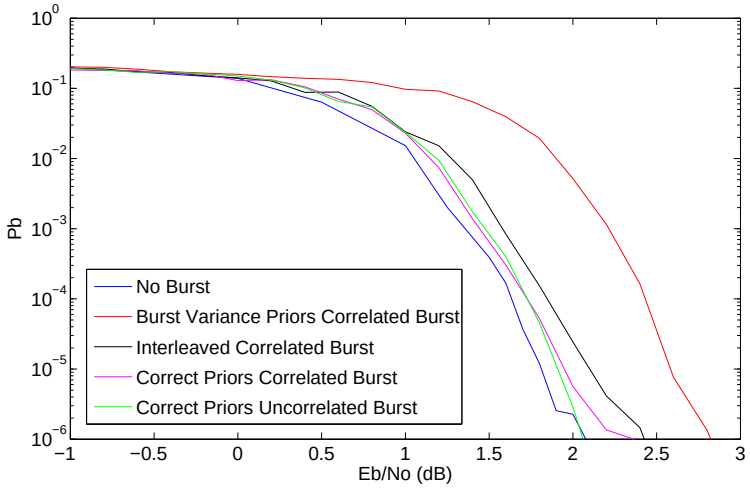

Fig. 3.16: Close-up of correct prior calculation comparison for rate .333 code with 5% burst time and SBNR of 1.6 dB.

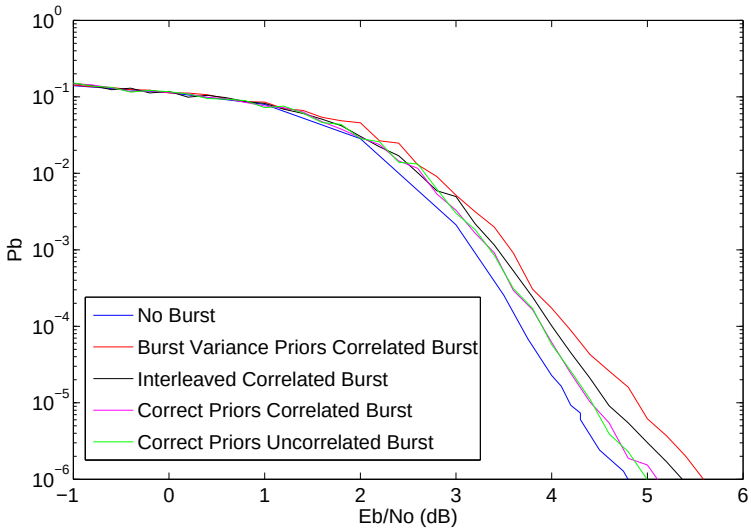

Fig. 3.17: Close-up of correct prior calculation comparison for rate .700 code with 5% burst time and SBNR of 4.6 dB.

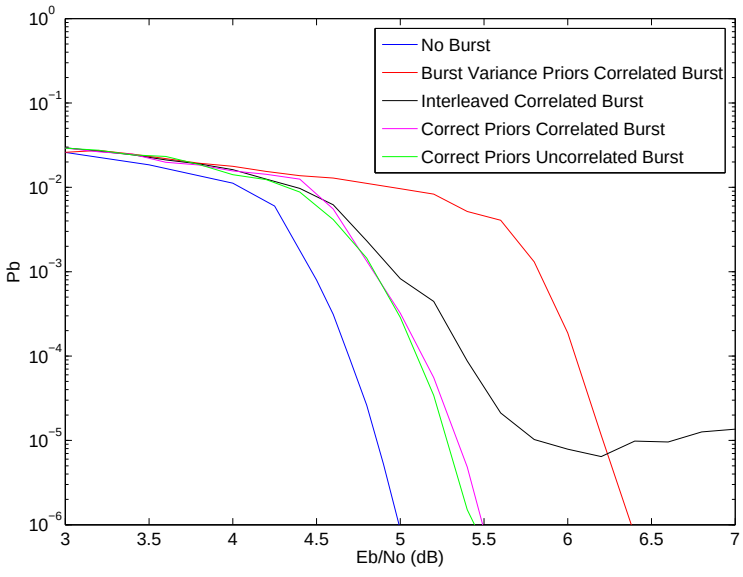

Fig. 3.18: Close-up of correct prior calculation comparison for rate .936 code with 5% burst time and SBNR of 3.6 dB.

Table 3.3: Coding gain comparison with correct priors with 5% burst time.

Code Rate	No Burst	Burst Variance Priors	Correct Priors, Correlated Burst	Correct Priors, Uncorrelated Burst
.333	8.4	7.7	8.4	8.2
.936	5.5	4.1	5.0	5.0
.7	5.7	5.0	5.5	5.4

Chapter 4

Burst Estimation

Having identified a method for improving code performance when a burst is present, we now turn to identifying a method for locating the bursts. We will investigate existing methods as well as developing a new one that will be used in performance tests.

4.1 Existing Techniques

Several techniques have been published that can locate bursts. Various attributes can be used to identify bursts in LDPC codes. A review of two of the more interesting and useful ones follows.

4.1.1 Algebraic Structure

This first technique was briefly addressed in the introduction as a method for correcting bursts. The setting for this approach is in the case of bursts with no random errors. The syndrome calculations reveal the location of the bursts if they are restricted to a total distance requirement defined by the code structure. It also allows for easy correction of these bursts. When restricting the technique to correcting bursts that are within the code capabilities — that is, all bursts and random errors are contained within the required length — the code works well [2].

The technique does not have much application in the setting discussed here, though, as multiple bursts across the codeword can easily break the code. Random errors cause the most problems, though, as they can occur far apart. The algorithm response to errors that occur outside the code requirement is to end decoding and not correct the received codeword [2]. This results in poor performance whenever the random errors or bursts violate the code performance constraints. A test program was created to examine performance given

the burst model described in Chapter 2, using only the short 15-bit example code from the paper [2]. The code performed worse than the binary symmetric channel when subjected to the burst model used in Chapter 2. This is expected, since the code is not being tested in a channel that actually meets the code design requirements.

4.1.2 Parity-Check Thresholding

In this method, the failed parity checks are used to locate bursts. In the case where only one burst is present and there is no random noise, it is likely that only a single bit in the burst will be involved in any given parity check. By making a hard decision on the initial soft decoder data for a received message and then checking parity, the checks with erroneous bits in them are able to be identified. All the bits in these checks are flagged. These bits are considered suspect. Since two bits are unlikely to be involved in the same two parity checks, but each bursty bit will most likely be in multiple failed checks, the number of these flags can indicate the bursty bits. A summation is made of the number of times each bit is flagged. Since some checks will be entirely outside of a burst, not all bits will be flagged, but the majority of bits inside the burst should receive, on average, more flags. Simple filtering is applied to help identify the burst range. This method is shown to estimate the burst as larger than the actual burst being used, but the actual burst is within this identified larger burst location. This method can be used to identify the bursty data for erasure so that the decoder can effectively decode, although other techniques could also be applied [9].

This approach also creates problems given the model developed in Chapter 2. First, the bursts are spread out across the codeword, meaning that the possibility of two or more bursts being in the same check are higher, which could result in bursts not leaving a failed check as a signature. Second, the bursts are shorter and may even be separated by only one non-burst bit. In this case, the error margin on the edge of the detected burst would overlap these bursts and lump them together. Depending on the percentage of the bits selected, this could result in performance similar to estimating the variance as the burst noise plus non-burst noise as shown in Section 3.2. Finally, the random errors would additionally

throw off the parity check counts, affecting the burst locations.

4.1.3 Other Techniques

A number of other techniques were identified, but the techniques do not really tie in with the burst model established. In one case, burst erasures are exploited to identify locations of the burst [36]. This ties in poorly with our approach since, with erasures, the received information does not relate to the sent information, so the prior probability calculation does not mean anything. As discussed in the introduction specific codes have been developed that correct bursts well. In some cases, these codes could be used to identify burst locations. In many cases, these only correct one burst in a message, which does not work well with our burst model [15]. Further, this would preclude using an LDPC code. As such, these will not be investigated further.

4.1.4 Conclusion on Existing Techniques

While effective techniques exist to detect bursts, none of the techniques where an LDPC code is used account for the presence of burst and regular noise simultaneously. In order for the bursts to be identified, a new technique will need to be developed.

4.2 New Technique: Markov State Detection

Identifying the bits where a burst is present can be a difficult task. However, when we consider that the underlying model uses a Markov chain with different states, a solution presents itself. A common technique for detecting which state a system is in given observations is the forward-backward algorithm. Variations of this algorithm are commonly used to detect which bit or symbol state a system is in given the set of observations. In contrast to this, we want to determine the burst state at time t, which we will call Ψ_t, given the set of received observations $\mathbf{r}$. The starting point for the derivation is taken from Deller et al. [37], but the derivation is developed here due to some differences in the objectives.

4.2.1 Forward-Backward Algorithm

We will begin by examining the uncorrelated burst case. We want to find the state Ψ at time t given the observations of the message, $\mathbf{r}$, so we begin with $P(\Psi_t = p|\mathbf{r})$. The posterior probability of being in state p given the observed sequence $\mathbf{r}$ is

$$P(\Psi_t = p|\mathbf{r}) = p(\Psi_t = p, \mathbf{r})/p(\mathbf{r}) = p(\Psi_t = p, \mathbf{r}_{\leq t}, \mathbf{r}_{>t})/p(\mathbf{r}), \tag{4.1}$$

where we divide by $p(\mathbf{r})$ to remove conditioning on the observations and then break the obserations into two groups, $\mathbf{r}_{>t}$ and $\mathbf{r}_{\leq t}$. Then we conditionally factor the observations $\mathbf{r}_{\leq t}$ and the state $\Psi_t = p$ as

$$p(\Psi_t = p, \mathbf{r}_{\leq t}, \mathbf{r}_{>t})/p(\mathbf{r}) = \alpha(\mathbf{r}_{\leq t}, p)\beta(\mathbf{r}_{>t}|p)/p(\mathbf{r}), \tag{4.2}$$

where the factors are defined as

$$\alpha(\mathbf{r}_{\leq t}, p) = p(\mathbf{r}_{\leq t}, \Psi_t = p) \tag{4.3}$$

$$\beta(\mathbf{r}_{>t}|p) = p(\mathbf{r}_{>t}|\mathbf{r}_{\leq t}, \Psi_t = p) = p(\mathbf{r}_{>t}|\Psi_t = p), \tag{4.4}$$

and the conditioning on the $\mathbf{r}_{\leq t}$ is removed in (4.4) due to Markovity between observations. We next examine the recursion that allows for the forward and backward passes. We start with (4.3) and examine α at time $t + 1$.

$$\alpha(\mathbf{r}_{\leq t+1}, q) = p(\mathbf{r}_{\leq t+1}, \Psi_{t+1} = q)$$

Here we see that if we sum in the previous state p at time t that leads to state q and separate the observation r_{t+1} from the rest, we get

$$\sum_{p=0}^{1} p(\mathbf{r}_{\leq t+1}, \Psi_{t+1} = q, \Psi_t = p) = \sum_{p=0}^{1} p(\mathbf{r}_{\leq t}, r_{t+1}, \Psi_{t+1} = q, \Psi_t = p).$$

We then conditionally factor Ψ_t and the observations $\mathbf{r}_{\leq t}$ to get

$$\sum_{p=0}^{1} p(\mathbf{r}_{\leq t}, r_{t+1}, \Psi_{t+1} = q, \Psi_t = p) = \sum_{p=0}^{1} p(\mathbf{r}_{\leq t}, \Psi_t = p) p(r_{t+1}, \Psi_{t+1} = q | \mathbf{r}_{\leq t}, \Psi_t = p).$$

Then we can conditionally factor Ψ_{t+1} in the second factor to get

$$\sum_{p=0}^{1} p(\mathbf{r}_{\leq t}, \Psi_t = p) p(r_{t+1}, \Psi_{t+1} = q | \mathbf{r}_{\leq t}, \Psi_t = p) =$$
$$\sum_{p=0}^{1} p(\mathbf{r}_{\leq t}, \Psi_t = p) p(\Psi_{t+1} = q | \mathbf{r}_{\leq t}, \Psi_t = p) p(r_{t+1} | \Psi_{t+1} = q, \mathbf{r}_{\leq t}, \Psi_t = p).$$

In the second factor, the observations $\mathbf{r}_{\leq t}$ provide no information about Ψ_{t+1} except to estimate Ψ_t, which is already given. In the third factor, Markovity between the observations removes the dependence on $r_{\leq t}$, and Ψ_t does not provide any information about r_{t+1}. We can remove all of these dependencies, giving

$$\sum_{p=0}^{1} p(\mathbf{r}_{\leq t}, \Psi_t = p) p(\Psi_{t+1} = q | \mathbf{r}_{\leq t}, \Psi_t = p) p(r_{t+1} | \Psi_{t+1} = q, \mathbf{r}_{\leq t}, \Psi_t = p) =$$
$$\sum_{p=0}^{1} p(\mathbf{r}_{\leq t}, \Psi_t = p) p(\Psi_{t+1} = q | \Psi_t = p) p(r_{t+1} | \Psi_{t+1} = q).$$

The first factor is $\alpha(\mathbf{r}_{\leq t}, p)$. The probability of state transition $P_{(p,q)}$ that we used in Chapter 2 before is the second factor. We will define

$$b(r_t | p) = p(r_t | \Psi_t = p), \tag{4.5}$$

and use it to represent the third factor. So we can write $\alpha(\mathbf{r}_{\leq t+1}, \Psi_{t+1} = q)$ as

$$\alpha(\mathbf{r}_{\leq t+1}, q) = \sum_{p=0}^{1} \alpha(\mathbf{r}_{\leq t}, p) P_{(p,q)} b(r_{t+1} | q). \tag{4.6}$$

Using (4.6), we can recursively calculate $\alpha(\mathbf{r}_{\leq t}, p)$'s for the forward pass using the previous α.

For the backwards pass involving $\beta(\mathbf{r}_{>t}|p)$, we start with (4.4). We sum in the next state q at time $t+1$ and split the observations to get

$$\beta(\mathbf{r}_{>t}|p) = \sum_{q=0}^{1} p(\mathbf{r}_{>t+1}, r_{t+1}, \Psi_{t+1} = q|\Psi_t = p).$$

We can conditionally factor $\mathbf{r}_{>t+1}$ and Ψ_{t+1} to get

$$\sum_{q=0}^{1} p(\mathbf{r}_{>t+1}, r_{t+1}, \Psi_{t+1} = q|\Psi_t = p) =$$
$$\sum_{q=0}^{1} p(\mathbf{r}_{>t+1}, \Psi_{t+1} = q|\Psi_t = p)p(r_{t+1}|\mathbf{r}_{>t+1}, \Psi_{t+1} = q, \Psi_t = p),$$

and then conditionally factor Ψ_{t+1} in the first factor as

$$\sum_{q=0}^{1} p(\mathbf{r}_{>t+1}, \Psi_{t+1} = q|\Psi_t = p)p(r_{t+1}|\mathbf{r}_{>t+1}, \Psi_{t+1} = q, \Psi_t = p) =$$
$$\sum_{q=0}^{1} p(\mathbf{r}_{>t+1}|\Psi_{t+1} = q, \Psi_t = p)p(\Psi_{t+1} = q|\Psi_t = p)p(r_{t+1}|\mathbf{r}_{>t+1}, \Psi_{t+1} = q, \Psi_t = p).$$

In the first factor, Ψ_t does not provide any information about the observations $\mathbf{r}_{>t+1}$. In the third factor, knowing the state Ψ_t and the observations $\mathbf{r}_{>t+1}$ does not provide any information about the observation r_{t+1}, so we can remove conditioning on these, yielding

$$\sum_{q=0}^{1} p(\mathbf{r}_{>t+1}|\Psi_{t+1} = q, \Psi_t = p)p(\Psi_{t+1} = q|\Psi_t = p)p(r_{t+1}|\mathbf{r}_{>t+1}, \Psi_{t+1} = q\Psi_t = p) =$$
$$\sum_{q=0}^{1} p(\mathbf{r}_{>t+1}|\Psi_{t+1} = q)p(\Psi_{t+1} = q|\Psi_t = p)p(r_{t+1}|\Psi_{t+1} = q).$$

We recognize here that the first factor is $\beta(\mathbf{r}_{>t+1}|q)$, the second factor is $P_{(p,q)}$. and the third factor is $b(r_{t+1}|q)$. So the recursion for finding $\beta(\mathbf{r}_{>t}|p)$ is given as

$$\beta(\mathbf{r}_{>t}|p) = \sum_{q=0}^{1} \beta(\mathbf{r}_{>t+1}|q)P_{(p,q)}b(r_{t+1}|q). \tag{4.7}$$

In order to calculate α and β, the values of the b must be found. By (4.5), we can see that b is simply the likelihood of receiving the observation r_t given the state at time t. In order to find this, we must include the transmitted bit value x_t, which must be summed in. Then, the likelihood is given by

$$p(r_t|\Psi_t = p) = \sum_{x_t \in \{-1,1\}} \frac{1}{\sqrt{2\pi\sigma_p^2}} e^{-\frac{1}{2\sigma_p^2}(r_t - x_t)^2}. \tag{4.8}$$

Here, σ_p^2 represents the noise variance in the state that p indicates. It is assumed in this step that the prior probabilities of each symbol being sent are equal, so they are canceled out when we normalize later. Then using (4.8) to calculate each b, we can move on to the forward and backward passes.

To initialize the algorithm, we consider the first element in the forward pass, $\alpha(r_1, q)$, where there are no observations earlier in time. As such, there is no state transition and (4.6) at $t = 1$ becomes

$$\alpha(r_1, \Psi_1 = q) = P_q b(r_1|q). \tag{4.9}$$

The P_q factor is the probability of being in state q and the b factor is defined as before.

In the backward pass, we consider an element outside the range of observations which occurs at time $T + 1$, where T is the length of the vector of observations. Obviously, this element is fictitious, but we can define $\beta(r_{T+1}|\Psi_{T+1} = q)$ as 1 if q is a legal final state and zero otherwise. We normalize to put equal weight on all the valid states. In the case of our model, this means that we divide the probability evenly between both states, since we can end in either state. This provides with a starting point for recursing back to $\beta(r_T|\Psi_T = p)$ from this initialization as defined in (4.7).

The algorithm works by combining the initialized and calculated forward and backwards passes into (4.2). By normalizing over all the different states, we account for the $p(\mathbf{r})$ and are able to find $P(\Psi_t = p|\mathbf{r})$.

4.2.2 Correlated Forward-Backward Algorithm

Since the previous derivation assumes that the elements are uncorrelated in the burst, we must also work through the derivation when there is correlation between the burst elements.

For the correlated algorithm, the derivation comes out with some slight differences. We can follow the uncorrelated derivation through to (4.2) with α defined as before in (4.3). When we conditionally factor as before, the β factor comes out as $p(\mathbf{r}_{>t}|\mathbf{r}_{\leq t}, \Psi_t = p)$. When the observations at time $t+1$ and t are uncorrelated, the conditioning on the observations $\mathbf{r}_{\leq t}$ is removed as before. However, when they are correlated, the r_t term remains. As such, we will define β for the correlated version of the algorithm as

$$\beta(\mathbf{r}_{>t}|r_t, p) = p(\mathbf{r}_{>t}|r_t, \Psi_t = p), \tag{4.10}$$

and we will account for the correlation or lack thereof as appropriate, given the appropriate state information.

We must determine the recursions for the forward and backwards passes, as there are some minor differences. Starting with (4.3) and examining α at time $t+1$ while summing in the state p at time t, we get

$$\alpha(\mathbf{r}_{\leq t+1}, q) = \sum_{p=0}^{1} p(\mathbf{r}_{\leq t}, r_{t+1}, \Psi_{t+1} = q, \Psi_t = p).$$

Conditionally factoring twice, as in the uncorrelated derivation, we get

$$\sum_{p=0}^{1} p(\mathbf{r}_{\leq t}, r_{t+1}, \Psi_{t+1} = q, \Psi_t = p) =$$

$$\sum_{p=0}^{1} p(\mathbf{r}_{\leq t}, \Psi_t = p) p(\Psi_{t+1} = q|\mathbf{r}_{\leq t}, \Psi_t = p) p(r_{t+1}|\Psi_{t+1} = q, \mathbf{r}_{\leq t}, \Psi_t = p).$$

In the second factor, the observations $\mathbf{r}_{\leq t}$ still provide no additional information about Ψ_{t+1}. In the third factor, there is always Markovity between observations that are more

than one away in time from r_{t+1}, so we can remove the dependance on $r_{\leq t-1}$. Removing the conditioning results in

$$\sum_{p=0}^{1} p(\mathbf{r}_{\leq t}, \Psi_t = p)p(\Psi_{t+1} = q|\mathbf{r}_{\leq t}, \Psi_t = p)p(r_{t+1}|\Psi_{t+1} = q, \mathbf{r}_{\leq t}, \Psi_t = p) =$$

$$\sum_{p=0}^{1} p(\mathbf{r}_{\leq t}, \Psi_t = p)p(\Psi_{t+1} = q|\Psi_t = p)p(r_{t+1}|r_t, \Psi_t = p, \Psi_{t+1} = q).$$

The first factor is still $\alpha(\mathbf{r}_{\leq t}, p)$ and the second factor is still the probability of transition $P_{(p,q)}$. In the third factor, when $p = 0$, there is no correlation and this factor becomes b as defined in (4.5). When $p = 1$ and $q = 1$, there is correlation between r_t and r_{t+1}, so the conditioning on r_t must still be considered. For the correlated case, then, we will define

$$c(r_{t+1}|r_t, p, q) = p(r_{t+1}|r_t, \Psi_t = p, \Psi_{t+1} = q), \tag{4.11}$$

and use it to represent the third factor. So we can write $\alpha(\mathbf{r}_{\leq t+1}, q)$ as

$$\alpha(\mathbf{r}_{\leq t+1}, q) = \sum_{p=0}^{1} \alpha(\mathbf{r}_{\leq t}, p)P_{(p,q)}c(r_{t+1}|r_t, p, q). \tag{4.12}$$

Using (4.12), we can recursively calculate $\alpha(\mathbf{r}_{\leq t}, p)$'s for the forward pass.

For the backwards pass involving $\beta(\mathbf{r}_{>t}|r_t, p)$, we start with (4.10). We sum in the next state q at time $t + 1$, split the observations, and conditionally factor twice as in the uncorrelated algorithm to get

$$\beta(\mathbf{r}_{>t}|r_t, p) = \sum_{q=0}^{1} p(\mathbf{r}_{>t+1}|r_{t+1}, r_t, \Psi_{t+1} = q, \Psi_t = p)$$

$$*p(\Psi_{t+1} = q|r_t, \Psi_t = p)p(r_{t+1}|r_t, \Psi_{t+1} = q, \Psi_t = p).$$

In the first factor, Ψ_t and r_t do not provide any information about the observations $\mathbf{r}_{>t+1}$, as they are too far away to be correlated. In the second factor, r_t provides no information

about the state at time $t+1$. Removing conditioning on these yields

$$\sum_{q=0}^{1} p(\mathbf{r}_{>t+1}|r_{t+1}, r_t, \Psi_{t+1}=q, \Psi_t=p)p(\Psi_{t+1}=q|r_t, \Psi_t=p)$$

$$*p(r_{t+1}|r_t, \mathbf{r}_{>t+1}, \Psi_{t+1}=q, \Psi_t=p) =$$

$$\sum_{q=0}^{1} p(\mathbf{r}_{>t+1}|r_{t+1}, \Psi_{t+1}=q)p(\Psi_{t+1}=q|\Psi_t=p)p(r_{t+1}|r_t, \Psi_{t+1}=q, \Psi_t=p).$$

We recognize here that the first factor is $\beta(\mathbf{r}_{>t+1}|r_{t+1}, q)$, the second factor is $P_{(p,q)}$, and the third factor is $c(r_{t+1}|r_t, p, q)$. So the recursion for finding $\beta(\mathbf{r}_{>t}|r_t, p)$ is given as

$$\beta(\mathbf{r}_{>t}|r_t, p) = \sum_{q=0}^{1} \beta(\mathbf{r}_{>t+1}|r_{t+1}, q)P_{(p,q)}c(r_{t+1}|r_t, p, q). \tag{4.13}$$

In order to calculate α and β, the value of the c must be found. Starting with (4.11), we can remove the condtioning on r_t by writing

$$p(r_{t+1}|r_t, \Psi_t=p, \Psi_{t+1}=q) = p(r_{t+1}, r_t|\Psi_t=p, \Psi_{t+1}=q)/p(r_t|\Psi_t=p, \Psi_{t+1}=q).$$

Here, the conditioning on Ψ_{t+1} in the denominator is removed as it does not provide any information about r_t, so it becomes

$$p(r_{t+1}, r_t|\Psi_t=p, \Psi_{t+1}=q)/p(r_t|\Psi_t=p, \Psi_{t+1}=q) = p(r_{t+1}, r_t|\Psi_t=p, \Psi_{t+1}=q)/p(r_t|\Psi_t=p). \tag{4.14}$$

The numerator of (4.14) is simply the likelihood of the observations r_{t+1} and r_t given the states. If p or q (or both) are not in the burst state, then the likelihood of r_t factors out and cancels with the denominator. In this case, the likelihood can be found using (4.8), replacing p with q. If both are in the burst state, then there is correlation and we utilize the likelihood for correlated elements given as

$$p(r_t, r_{t+1}|\Psi_t=p, \Psi_{t+1}=q) = \sum_{x_t\in\{-1,1\}} \sum_{x_{t+1}\in\{-1,1\}} \frac{1}{2\pi\sqrt{|R|}} e^{-\frac{1}{2}(\mathbf{r}-\mathbf{x})^T R^{-1}(\mathbf{r}-\mathbf{x})}. \tag{4.15}$$

Here, R is the covariance matrix between r_t and r_{t+1}, $\mathbf{r}$ is the vector of the observations in question, r_t and r_{t+1}, and $\mathbf{x}$ is the vector of the bit values x_t and x_{t+1}. The denominator can be found using (4.8). So, depending on the state values, we can calculate the c terms.

Initialization of the algorithm is the same as before for the forward and backwards passes. The α and β terms are combined as before and normalization over the different state values accounts for the $p(\mathbf{r})$ with one difference. In the uncorrelated case, there are only two values for b, when $p = 0$ and when $p = 1$. In the correlated case, since both p and q are used, there are four values accounting for the four possible combinations of the states p and q. This only mildly impacts the calculations, though, as the states in the factors in (4.12) and (4.13) indicate which c factors to use.

4.2.3 Performance

As test of performance, the detected burst state was used in place of the actual burst state to evaluate the performance. Figures 4.1 through 4.6 show the estimated burst versus true burst performance for the 5% burst time for each code. The results are typical for each code in that the performance is about the same for each burst time.

For the uncorrelated bursts, seen in Figures 4.1, 4.3, and 4.5, the performance is very poor and the burst still overcomes the code. It appears that the burst detection in this case is not able to identify the burst well enough to help performance.

For the correlated burst, seen in Figures 4.2, 4.4, and 4.6, both the uncorrelated and correlated detectors are used. In both cases, the performance is excellent. The burst does not create a high noise floor that would degrade performance. For the rate .333 code, the performance for both is better than the true burst by about .5 dB. For the other two codes, the performance is about the same, except that the correlated algorithm does not work well on the rate .936 code. The fact that the correlated burst is effectively decoded even when using the uncorrelated burst detector is interesting. Since the correlation provides more information than is available in just the uncorrelated burst, this suggests that the extra information from the correlation in the prior probabilities for the LDPC decoder is actually making up for whatever the burst detector is missing.

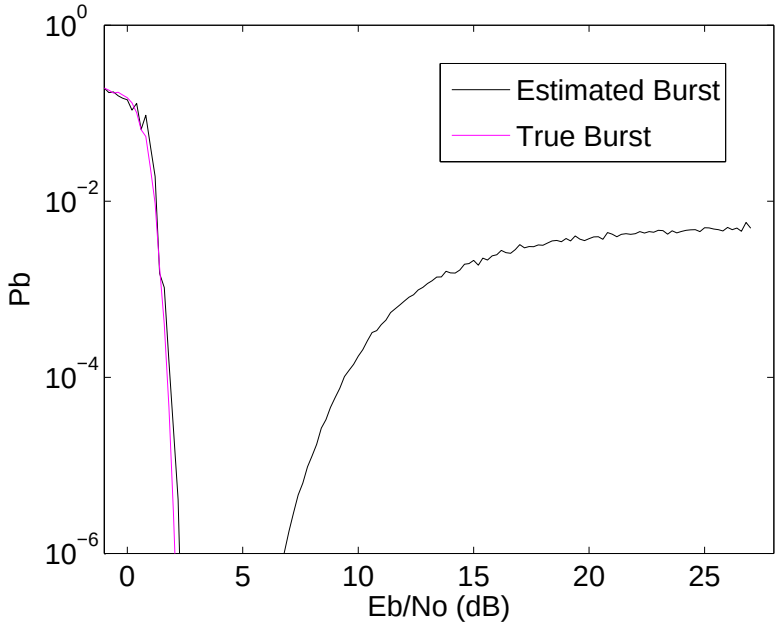

Fig. 4.1: Estimated burst performance for rate .333 code with 5% burst time, uncorrelated burst, and SBNR of 1.6 dB.

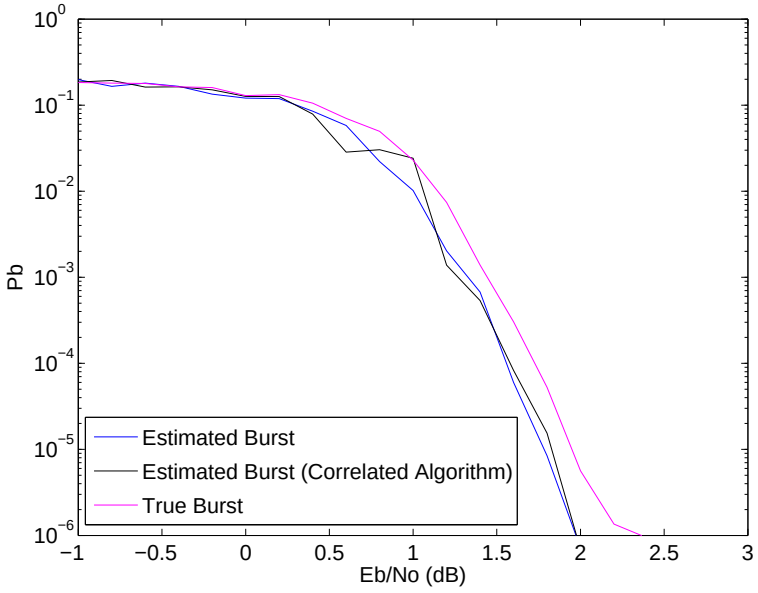

Fig. 4.2: Estimated burst performance for rate .333 code with 5% burst time, correlated burst, and SBNR of 1.6 dB.

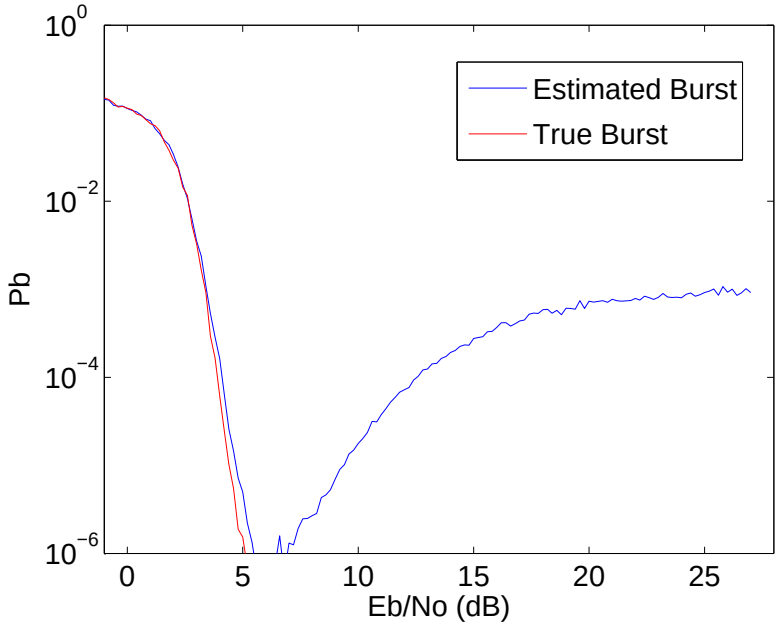

Fig. 4.3: Estimated burst performance for rate .700 code with 5% burst time, uncorrelated burst, and SBNR of 4.6 dB.

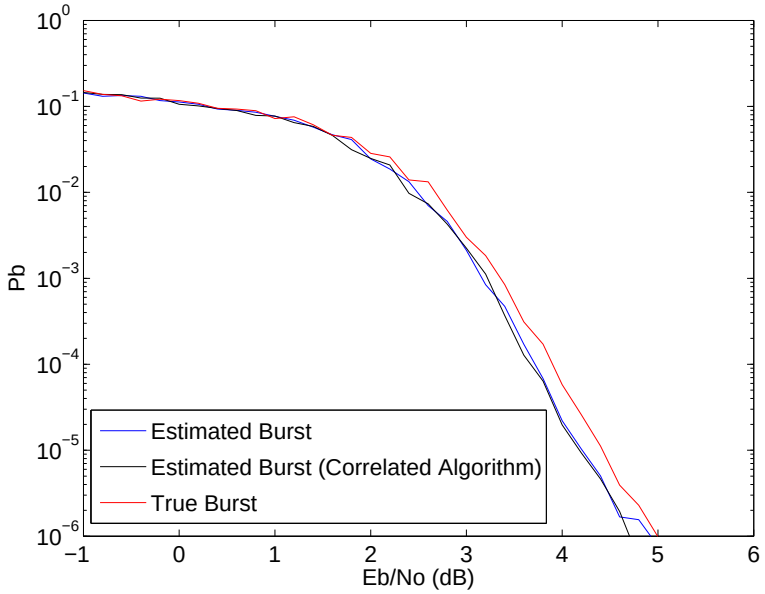

Fig. 4.4: Estimated burst performance for rate .700 code with 5% burst time, correlated burst, and SBNR of 4.6 dB.

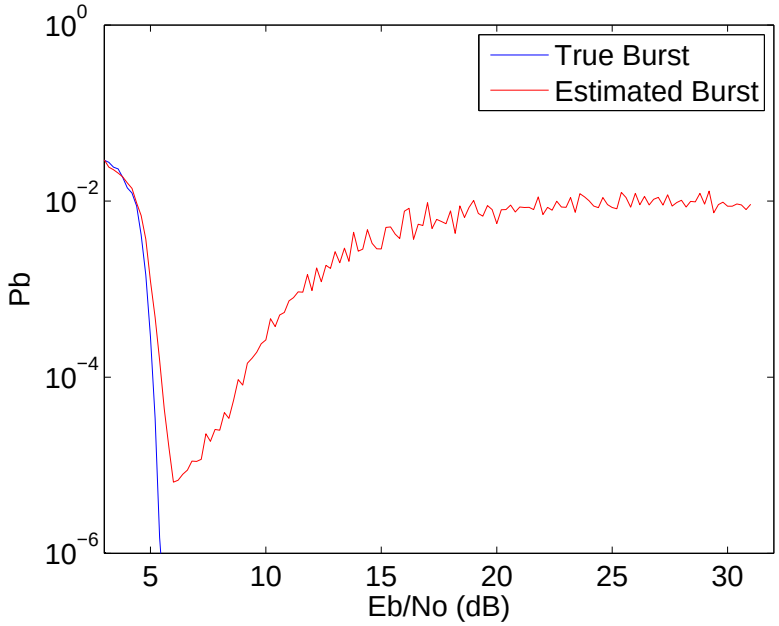

Fig. 4.5: Estimated burst performance for rate .936 code with 5% burst time, uncorrelated burst, and SBNR of 3.6 dB.

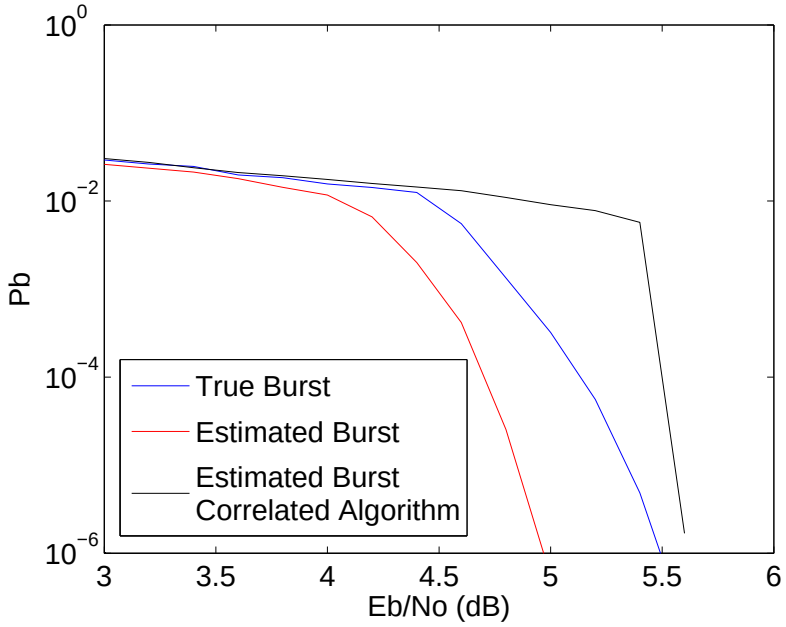

Fig. 4.6: Estimated burst performance for rate .936 code with 5% burst time, correlated burst, and SBNR of 3.6 dB.

4.2.4 Conclusion

Having identified a method for detecting bursts, we can now move on to discussing the overall results and their implications.

Chapter 5

Results

In considering the overall results of using the correct prior calculations with an estimated burst, we initially simply compared to knowing the true burst. The full results require a comparison with the other techniques that have been presented, namely interleaving and using only the burst variance to calculate priors. Since the interleaving performance is essentially identical to the code performance with no compensation, at least with our burst model, this will also represent the baseline code performance. This also makes sense because interleaving is the primary method for dealing with bursts and would be applied if the channel was known to have burst noise present.

The results will be plotted together for each code and burst time. Due to the differences in performance when using the estimated burst, the correlated and uncorrelated bursts will be addressed separately. A comparison of the coding gains and the recommended best approach, amongst those considered, will be given.

5.1 Uncorrelated Burst Noise Results

The uncorrelated burst noise results are documented in Figures 5.1 through 5.18. Starting with Figure 5.1, we recognize that it is difficult to compare the different curves in the 0 to 5 dB range, where the curves are all very close together. A close-up of this region is provided for each plot for further comparison.

Further investigation of Figure 5.1 shows that the estimated burst does not perform well, as we saw in Chapter 4. In fact, when compared with the baseline interleaving, the performance is identical at higher SNR values, where the burst errors dominate. At lower SNR values, where random errors dominate, Figure 5.2 shows that the performance is identical here as well.

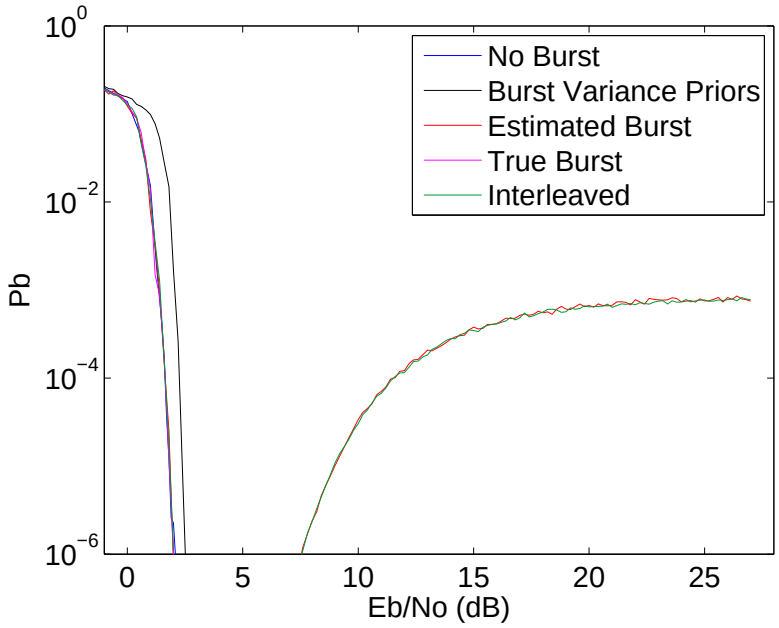

Fig. 5.1: Uncorrelated burst comparison for rate .333 code with 1% burst time and SBNR of 1.6 dB.

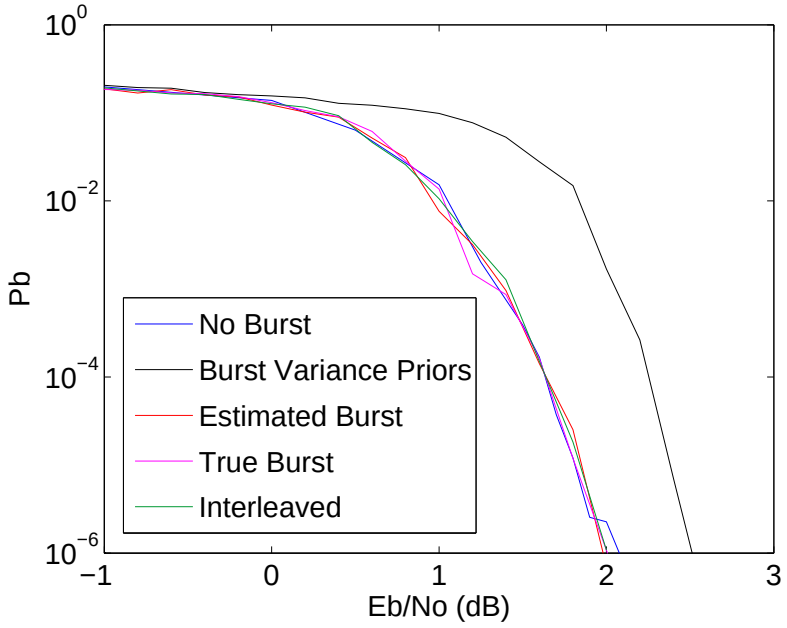

Fig. 5.2: Close-up of uncorrelated burst comparison for rate .333 code with 1% burst time and SBNR of 1.6 dB.

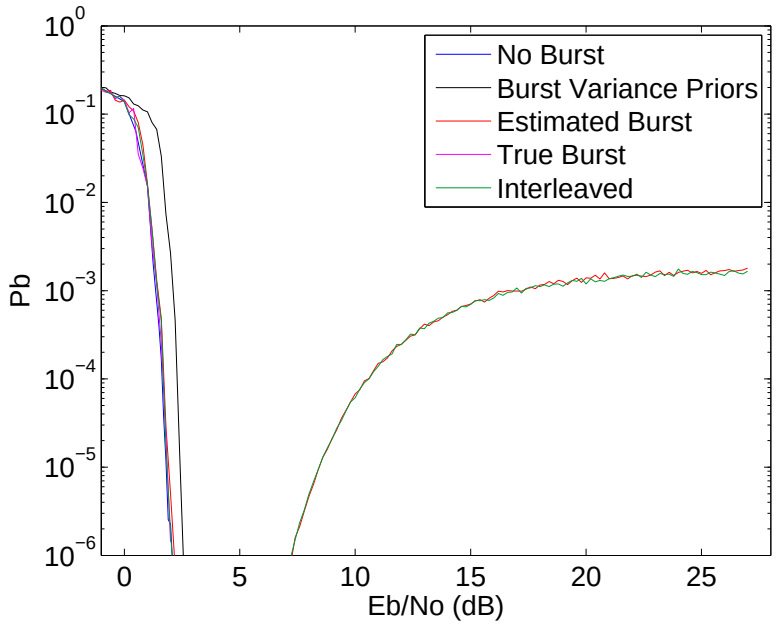

Fig. 5.3: Uncorrelated burst comparison for rate .333 code with 2% burst time and SBNR of 1.6 dB.

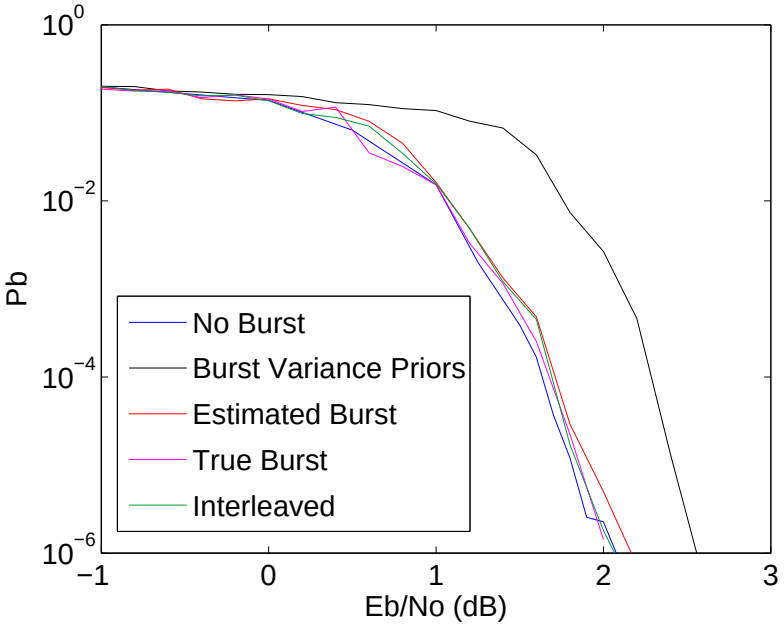

Fig. 5.4: Close-up of uncorrelated burst comparison for rate .333 code with 2% burst time and SBNR of 1.6 dB.

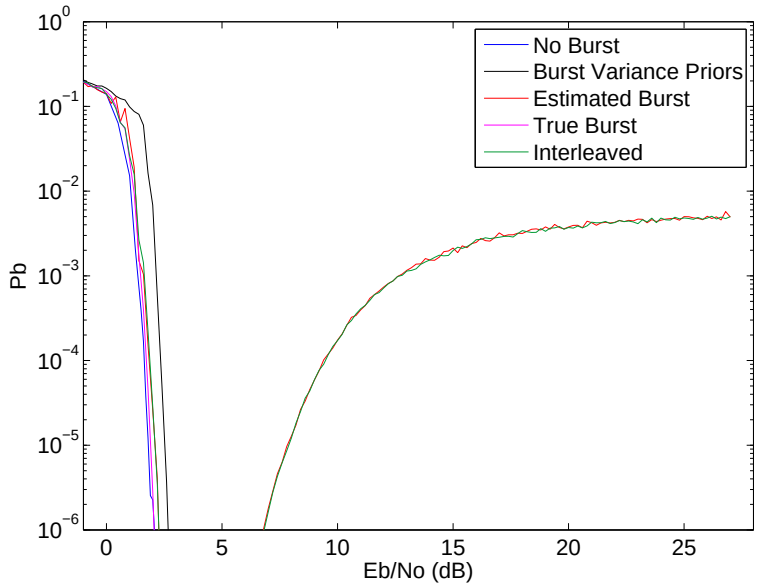

Fig. 5.5: Uncorrelated burst comparison for rate .333 code with 5% burst time and SBNR of 1.6 dB.

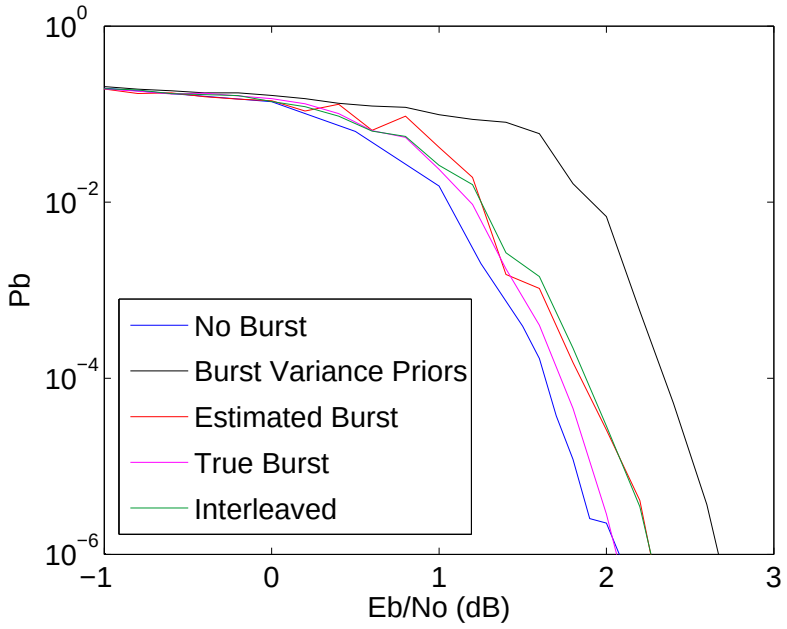

Fig. 5.6: Close-up of uncorrelated burst comparison for rate .333 code with 5% burst time and SBNR of 1.6 dB.

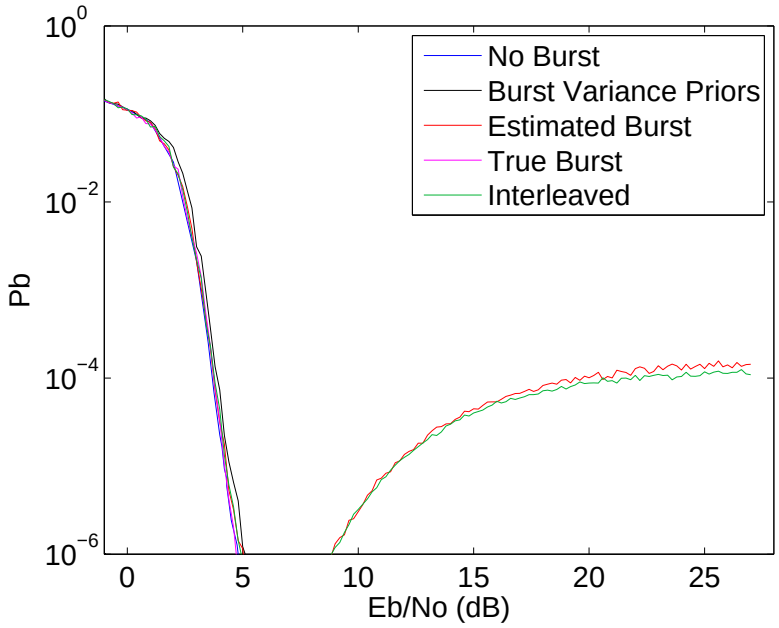

Fig. 5.7: Uncorrelated burst comparison for rate .700 code with 1% burst time and SBNR of 4.6 dB

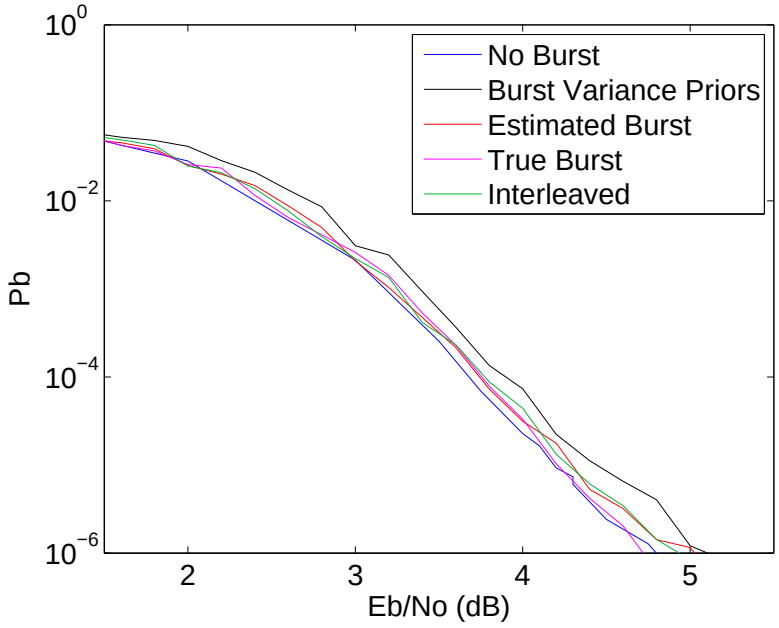

Fig. 5.8: Close-up of uncorrelated burst comparison for rate .700 code with 1% burst time and SBNR of 4.6 dB.

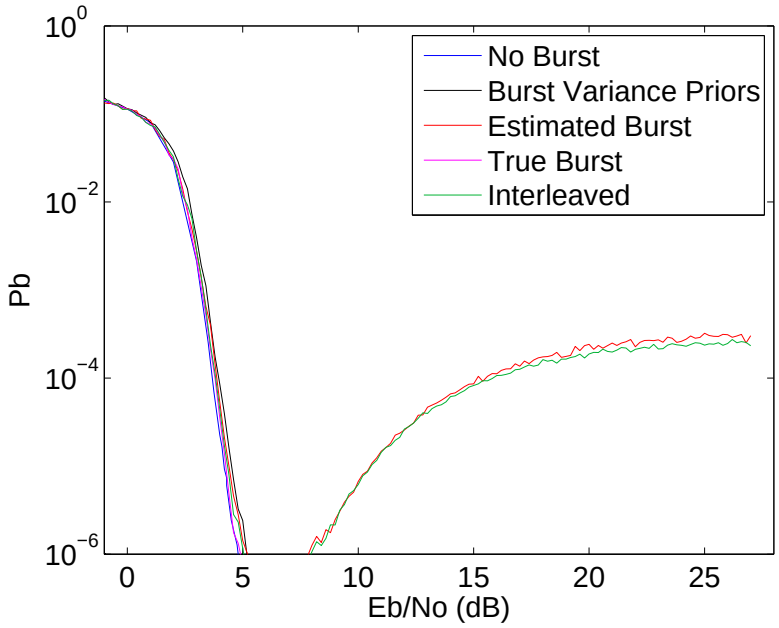

Fig. 5.9: Uncorrelated burst comparison for rate .700 code with 2% burst time and SBNR of 4.6 dB.

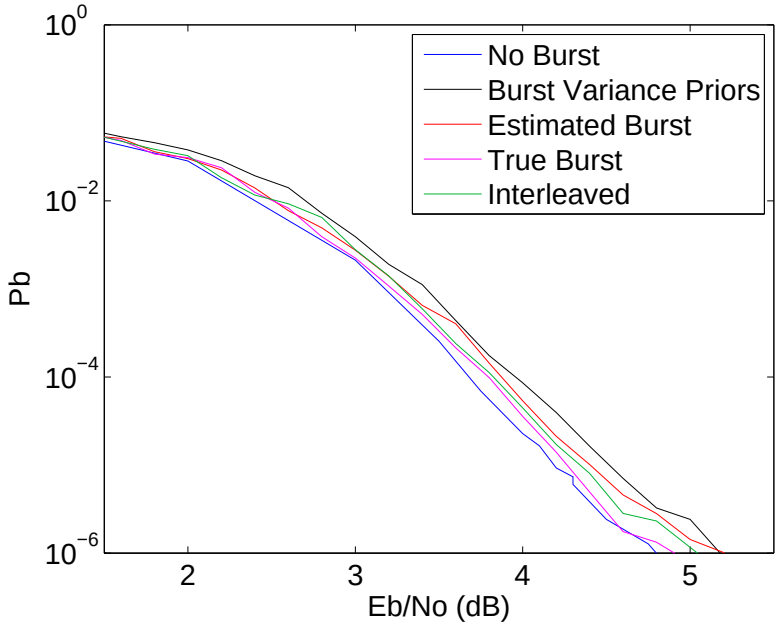

Fig. 5.10: Close-up of uncorrelated burst comparison for rate .700 code with 2% burst time and SBNR of 4.6 dB.

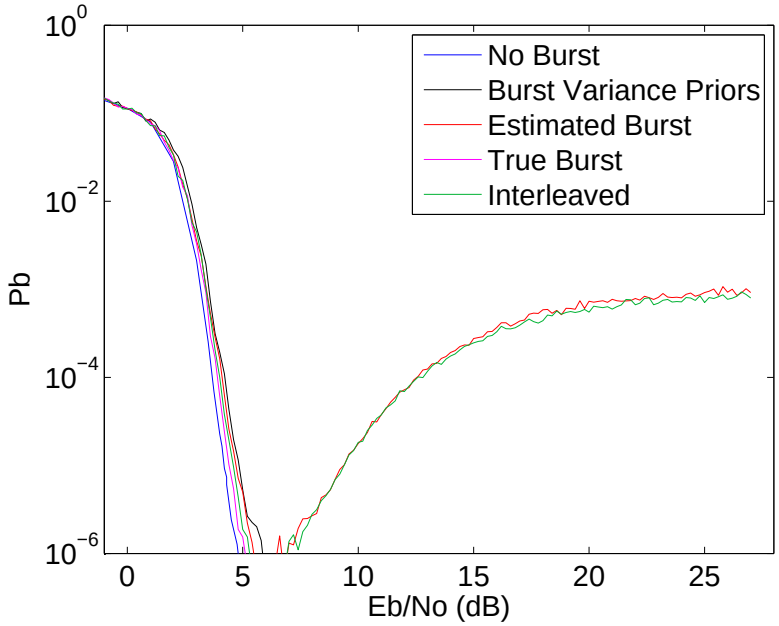

Fig. 5.11: Uncorrelated burst comparison for rate .700 code with 5% burst time and SBNR of 4.6 dB.

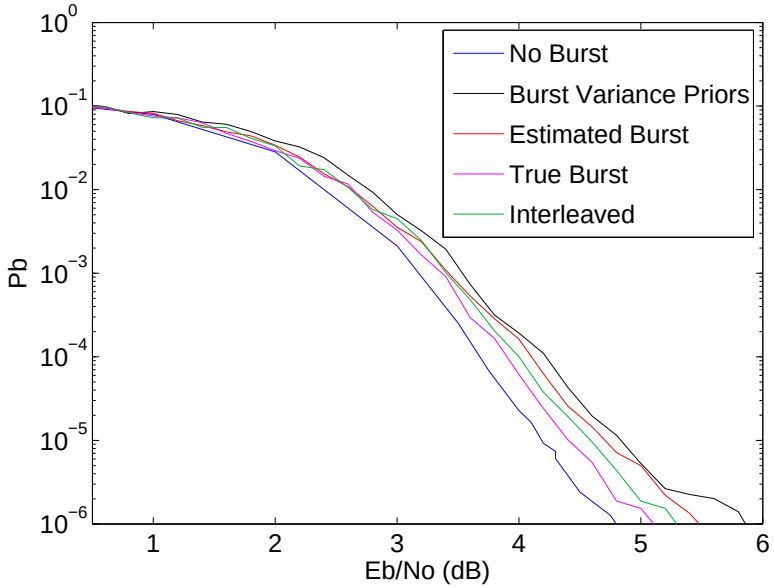

Fig. 5.12: Close-up of uncorrelated burst comparison for rate .700 code with 5% burst time and SBNR of 4.6 dB.

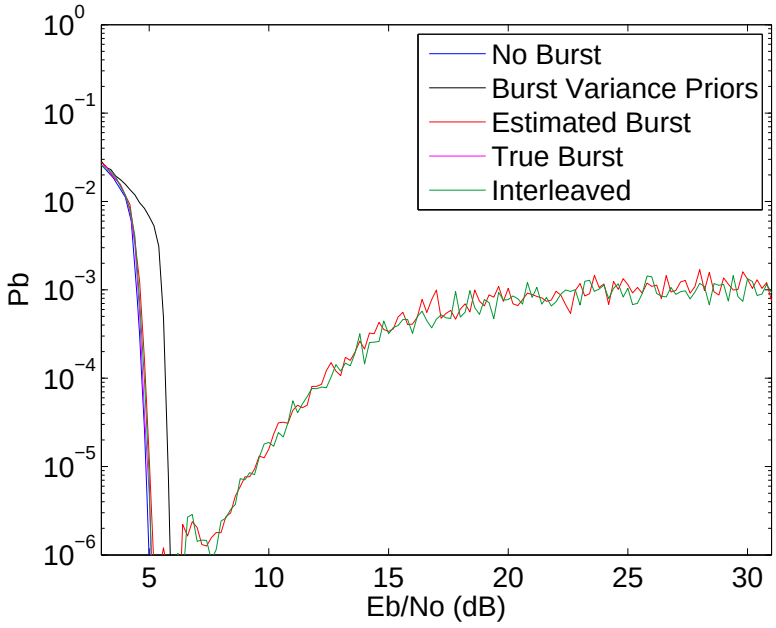

Fig. 5.13: Uncorrelated burst comparison for rate .936 code with 1% burst time and SBNR of 3.6 dB.

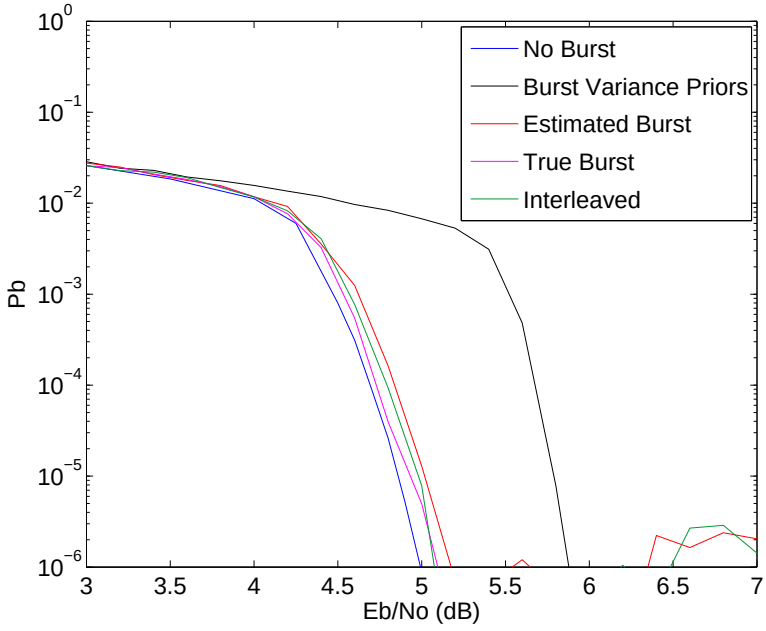

Fig. 5.14: Close-up of uncorrelated burst comparison for rate .936 code with 1% burst time and SBNR of 3.6 dB.

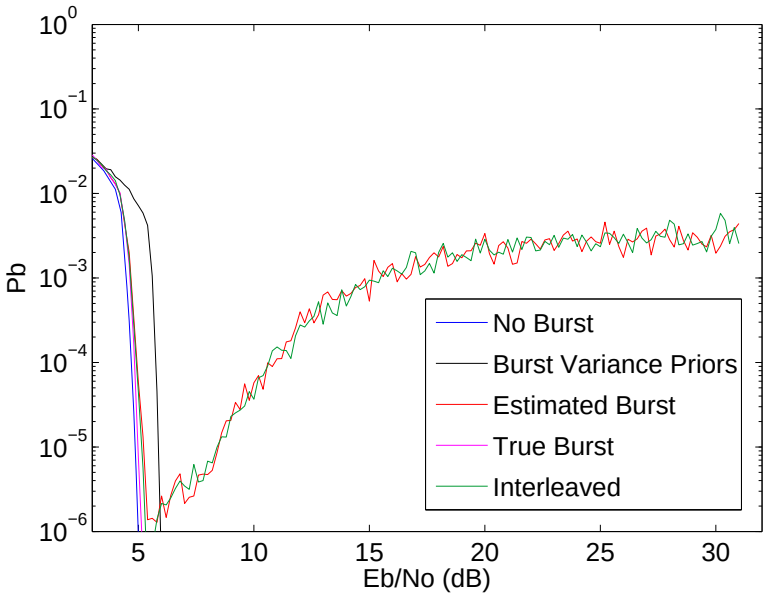

Fig. 5.15: Uncorrelated burst comparison for rate .936 code with 2% burst time and SBNR of 3.6 dB.

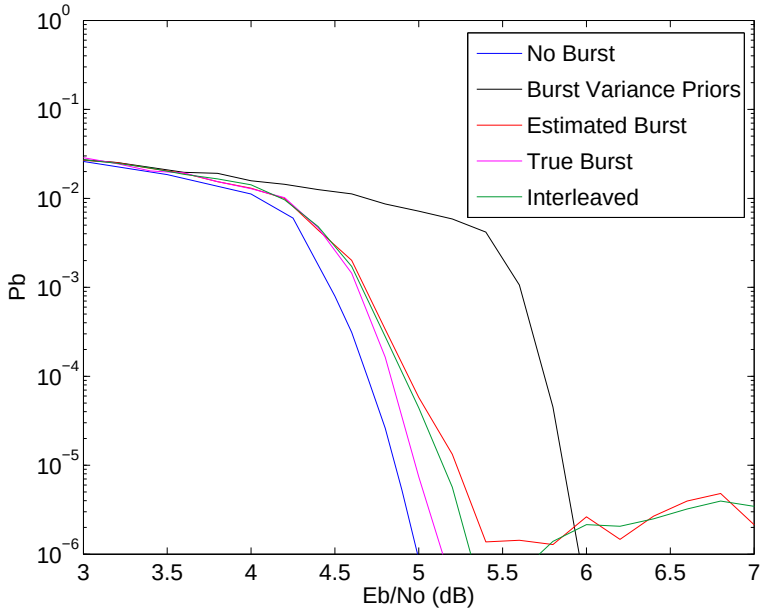

Fig. 5.16: Close-up of uncorrelated burst comparison for rate .936 code with 2% burst time and SBNR of 3.6 dB.

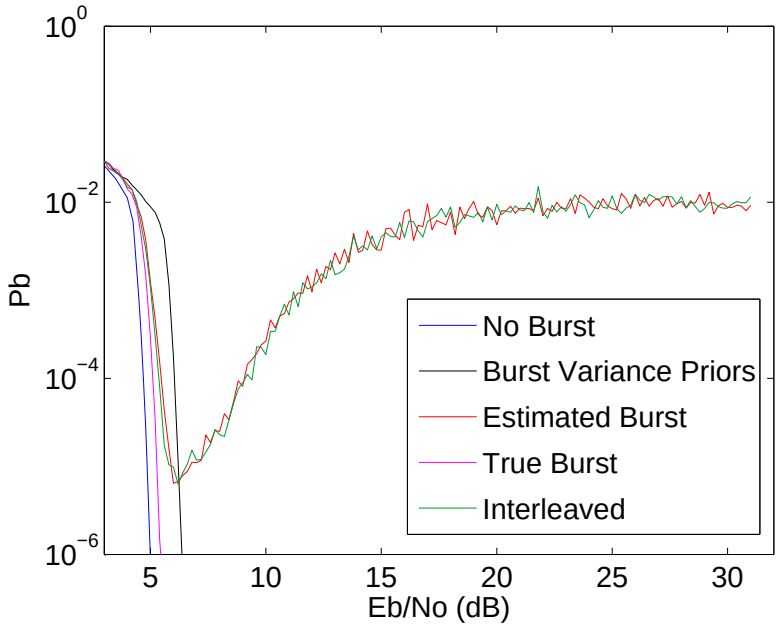

Fig. 5.17: Uncorrelated burst comparison for rate .936 code with 5% burst time and SBNR of 3.6 dB.

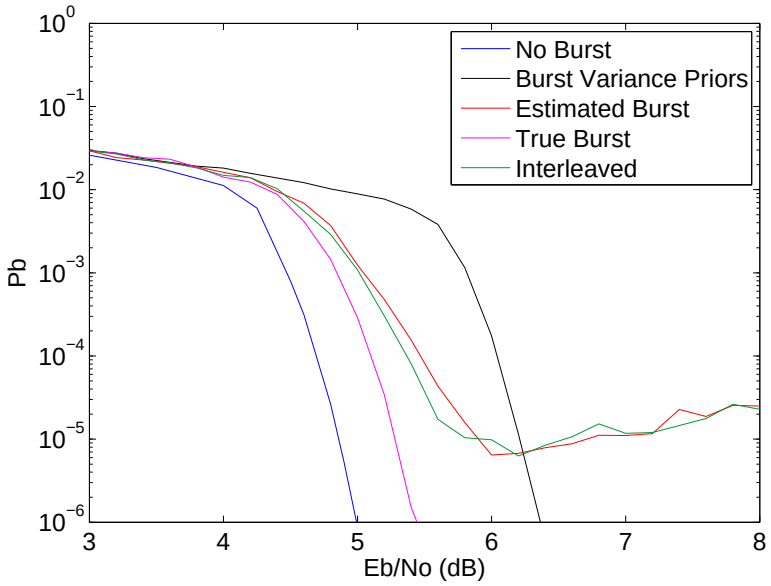

Fig. 5.18: Close-up of uncorrelated burst comparison for rate .936 code with 5% burst time and SBNR of 3.6 dB.

Looking at the performance for the other burst times and the other codes, this is consistently true in each plot. There are some minor variations for the rate .7 code. In Figures 5.7 and 5.9 we see that in the higher SNR regime, the interleaving appears to have a slightly lower probability of bit-error. Also, the interleaved code performs slightly better in the low SNR regime in Figure 5.12. In both cases, the difference is very slight and, given that the code still has a high noise floor, the change does not really provide a significant improvement in code performance.

Given that the estimated burst with correct priors does not perform well, the best approach that has been investigated for compensating for uncorrelated bursts is using the burst variance to calculate all the prior probabilities.

Code performance for priors calculated using the high burst variance will be compared to the non-burst case and correct priors with the true burst. This provides a reference against the typical code performance and also a reference for potential code performance with the burst present if we were able to identify the burst correctly. The coding gain is listed in Tables 5.1 through 5.3.

The results with $*$ did not actually achieve probability of bit-error of 10^{-6}, so the result is estimated if the curve had continued downward. This introduces some error in those values.

Table 5.1: Coding gain comparison with an uncorrelated burst and 1% burst time.

Code Rate	Interleaving	Burst Variance Priors	Correct Priors	Estimated Priors
.333	-.4	-.4	.1	.1
.7	-.1	-.3	.1	-.2
.936	-.1	-.9	-.1	-.2

Table 5.2: Coding gain comparison with an uncorrelated burst and 2% burst time.

Code Rate	Interleaving	Burst Variance Priors	Correct Priors	Estimated Priors
.333	0.0	-.5	.1	-.1
.7	-.3	-.4	-.1	-.4
.936	-1.0	-1.0	-.1	-.4*

Table 5.3: Coding gain comparison with an uncorrelated burst and 5% burst time.

Code Rate	Interleaving	Burst Variance Priors	Correct Priors	Estimated Priors
.333	-.2	-.6	0.0	-.2
.7	-.5	-.9	-.3	-.7
.936	-.8*	-1.3	-.4	-1.4*

Examining the results in Tables 5.1 through 5.3 we see that there is a loss in coding gain due to using the burst variance to calculate priors. For the rate .333 code, the loss is about .5 dB, with the true priors providing equal or better coding gain compared to no burst. Performance for the rate .7 starts similarly, but as the burst time increases, performance using the burst variance only decreases until about 1 dB of loss in coding gain is seen in the 5% burst time case. Again, performance with the true priors is equal to or better than the no burst performance. The rate .936 code has the worst loss, seeing about 1 dB in coding gain loss for all rates. The performance with the true priors is slightly worse than the performance with no burst.

Evaluating this comparison suggests several obvious conclusions. First, as burst time increases, the extra burst noise due to the larger amount of bursts decreases the amount of coding gain. This makes sense in both the case where the true variance and the burst variance only are used to calculate priors.

Second, despite some observations in Chapter 2 suggesting that high rate codes handle the burst noise well with no compensation, lower code rates really do perform better when compensating for bursts. The lost coding gain is much lower for the rate .333 code and even in the rate .7 code, the lost gain is not as severe as for the rate .936 code.

The most interesting observation is that the coding gain actually seems to be better for the burst noise case using the true priors when the burst is shorter. While there is only a difference of about .1 dB, it is seen on the rate .333 code for both 1% and 2% burst times and for the rate .7 code for 1% burst time. Most likely, this is due to the fact that the burst errors have low confidence because of high variance. This can help the decoder correct them more easily, as the higher confidence bits without burst noise will push the codeword

towards the correct value.

Unfortunately, the burst detector does not have enough information to properly identify the burst. The best we can do is to use the burst variance, which greatly improves performance over other options, but still can cause significant loss in coding gain.

5.2 Correlated Burst Noise Results

The results for the correlated burst noise are documented in Figures 5.19 through 5.36. Close-ups are again provided. In all the figures, the interleaved code is the only one that does not overcome the noise as the SNR increases. Recognizing this fact, we will restrict our discussion to the close-up figures.

The main observation is that in all of the rate .333 and rate .7 figures, all of the different configurations perform very similarly except for the case where the burst variance is used to calculate the prior probabilities. It can also be seen that the spread between the different curves increases as the burst time increases. This supports what was seen in the uncorrelated burst noise case.

In general, the rate .936 code performs similarly, except that the estimated burst using the correlated algorithm has performance much more similar to the burst variance only case. While this will clearly cause a performance impact, we can also see that in Figure 5.36, the performance is within .1 dB of the true burst case.

An interesting observation is that, as seen in Chapter 4, the correlated burst is much more easily decoded, even with the uncorrelated burst detector, with the exception of the correlated burst detector for the rate .936 code. The extra information supplied to the decoder by the correlation, even when it is weak as in this burst model, greatly improves performance. Even in the case of the rate .936 code, the correlation improves performance over using the burst variance on every observation.

Having evaluated the different plots, the main conclusion is that the best performance is achieved using the estimated burst without correlation in the algorithm. This performance lines up well with the no burst case for every plot.

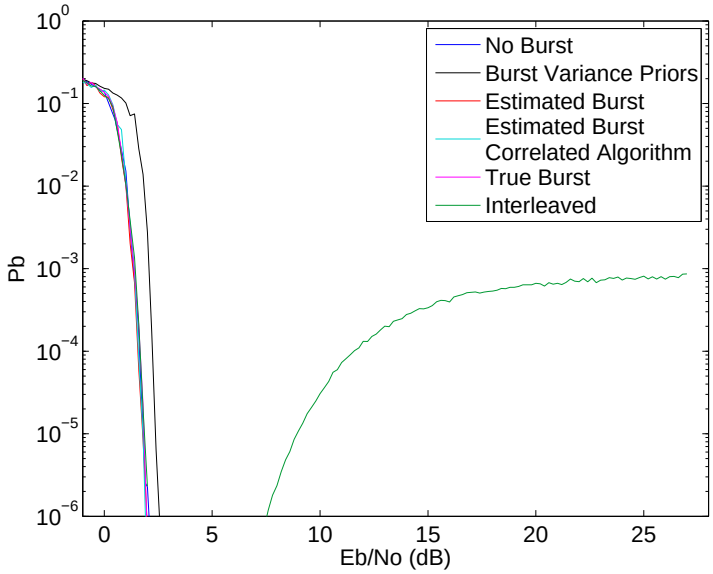

Fig. 5.19: Correlated burst comparison for rate .333 code with 1% burst time and SBNR of 1.6 dB.

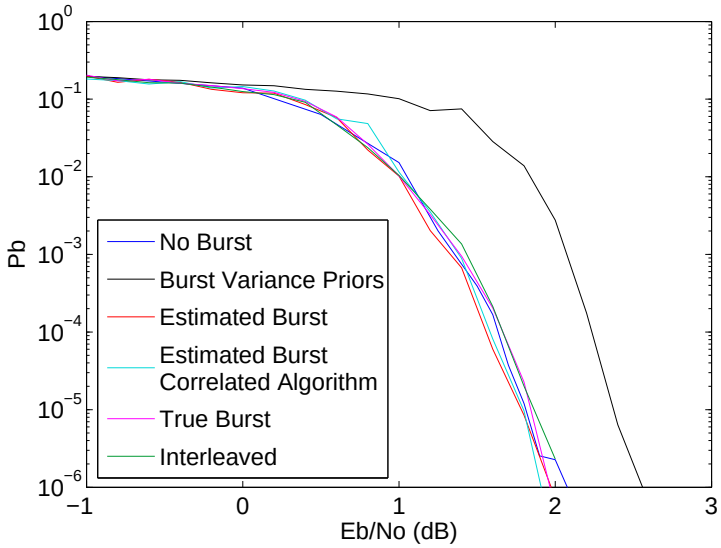

Fig. 5.20: Close-up of correlated burst comparison for rate .333 code with 1% burst time and SBNR of 1.6 dB.

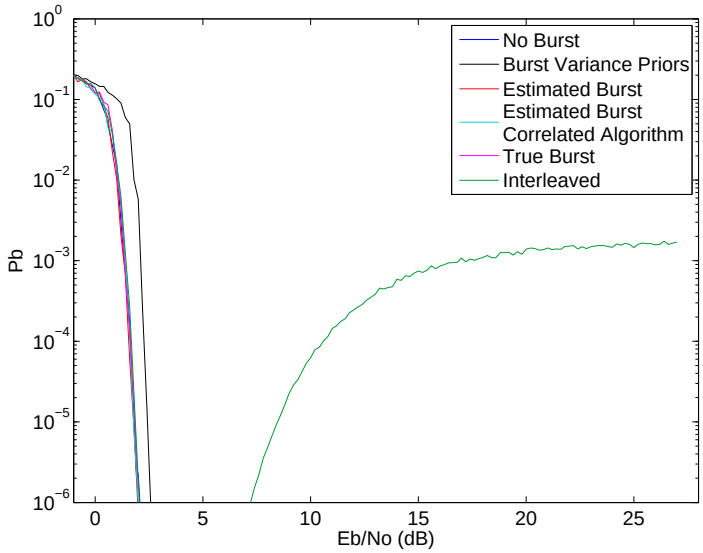

Fig. 5.21: Correlated burst comparison for rate .333 code with 2% burst time and SBNR of 1.6 dB.

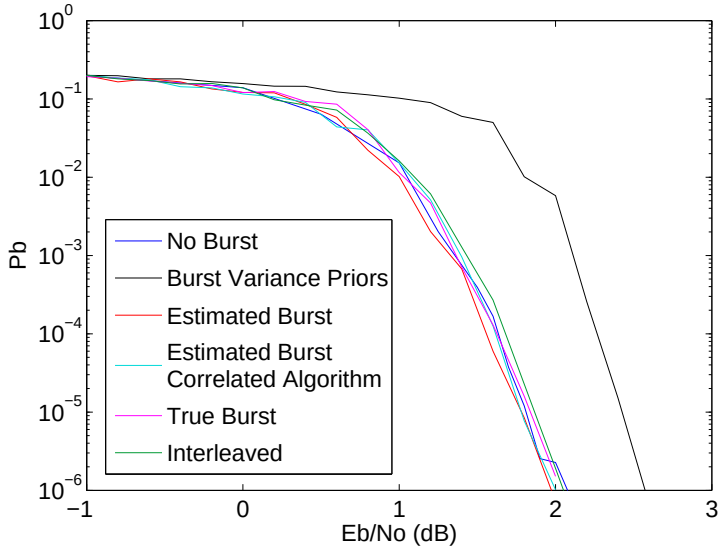

Fig. 5.22: Close-up of correlated burst comparison for rate .333 code with 2% burst time and SBNR of 1.6 dB.

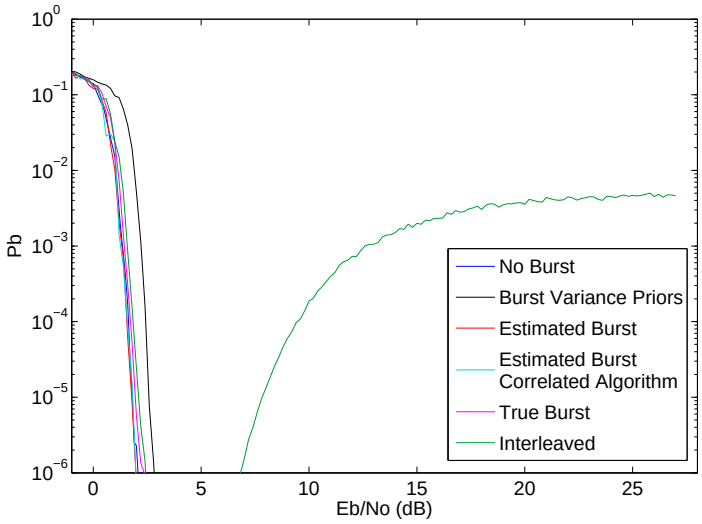

Fig. 5.23: Correlated burst comparison for rate .333 code with 5% burst time and SBNR of 1.6 dB.

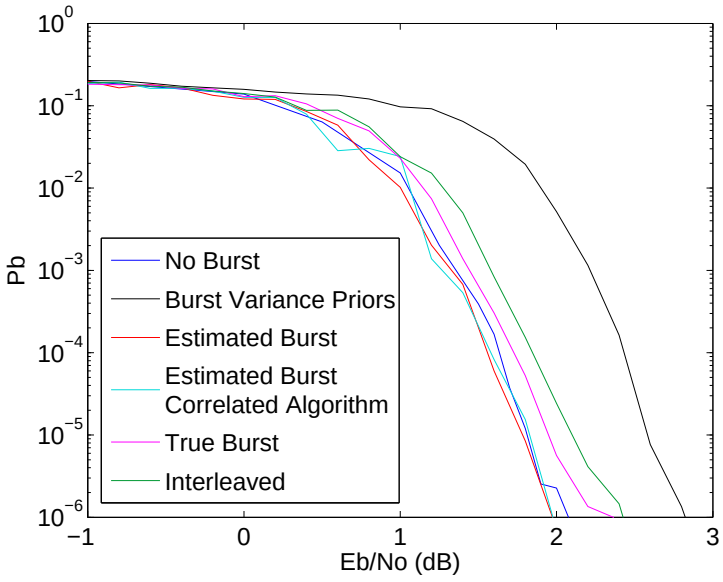

Fig. 5.24: Close-up of correlated burst comparison for rate .333 code with 5% burst time and SBNR of 1.6 dB.

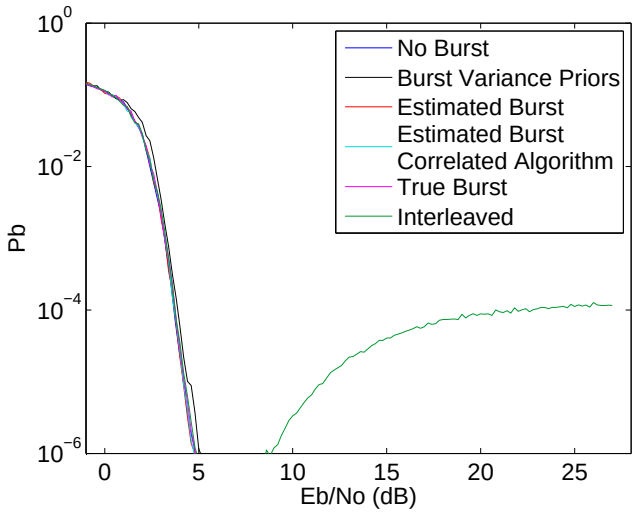

Fig. 5.25: Correlated burst comparison for rate .700 code with 1% burst time and SBNR of 4.6 dB.

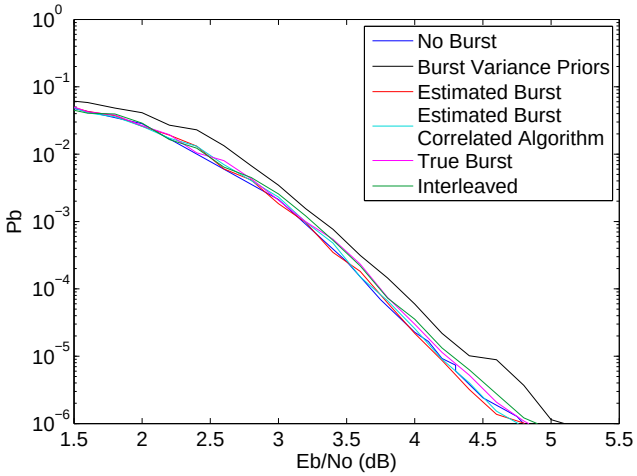

Fig. 5.26: Close-up of correlated burst comparison for rate .700 code with 1% burst time and SBNR of 4.6 dB.

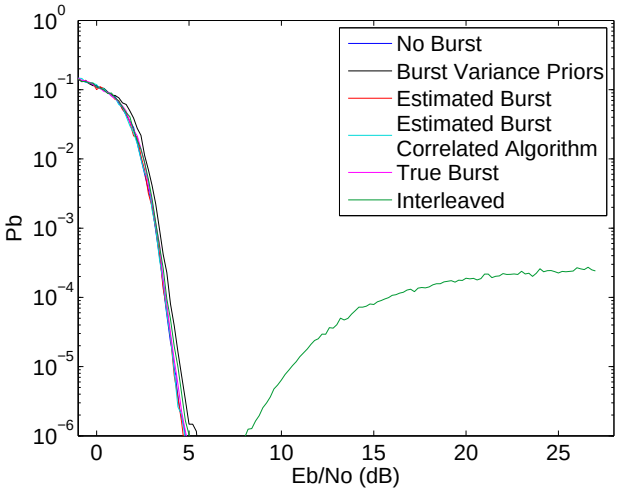

Fig. 5.27: Correlated burst comparison for rate .700 code with 2% burst time and SBNR of 4.6 dB.

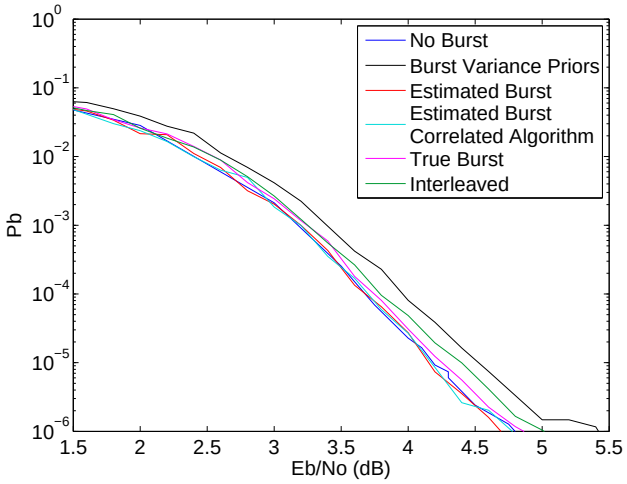

Fig. 5.28: Close-up of correlated burst comparison for rate .700 code with 2% burst time and SBNR of 4.6 dB.

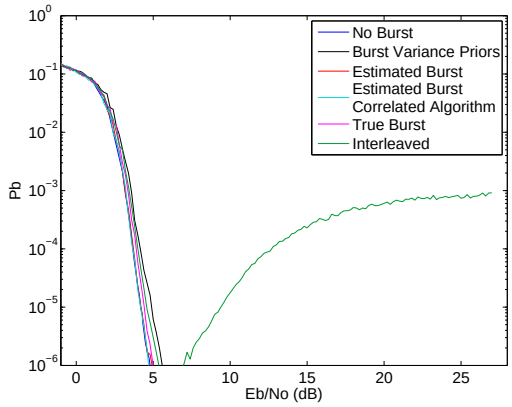

Fig. 5.29: Correlated burst comparison for rate .700 code with 5% burst time and SBNR of 4.6 dB.

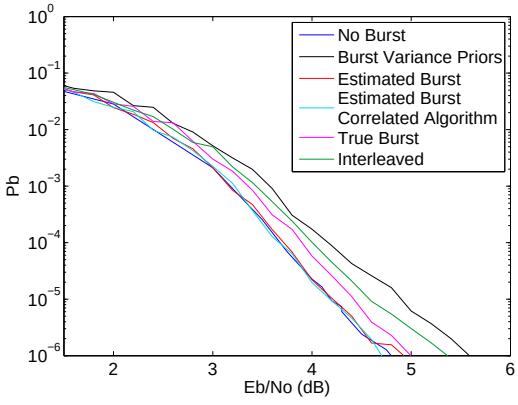

Fig. 5.30: Close-up of correlated burst comparison for rate .700 code with 5% burst time and SBNR of 4.6 dB.

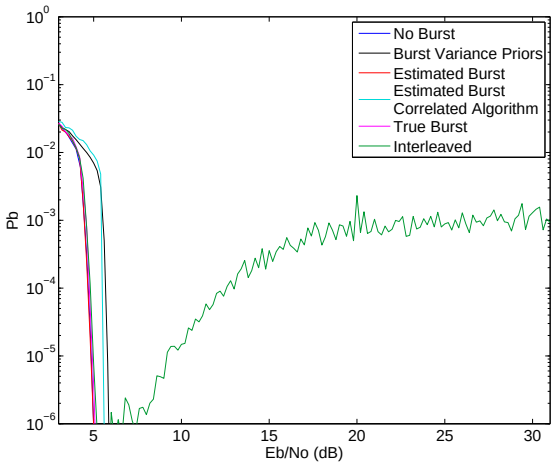

Fig. 5.31: Correlated burst comparison for rate .936 code with 1% burst time and SBNR of 3.6 dB.

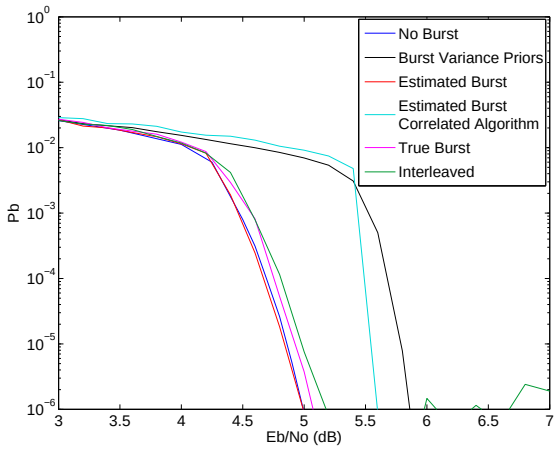

Fig. 5.32: Close-up of correlated burst comparison for rate .936 code with 1% burst time and SBNR of 3.6 dB.

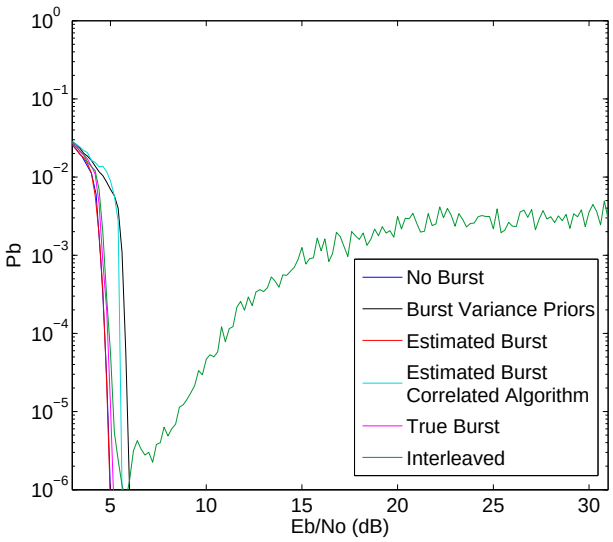

Fig. 5.33: Correlated burst comparison for rate .936 code with 2% burst time and SBNR of 3.6 dB.

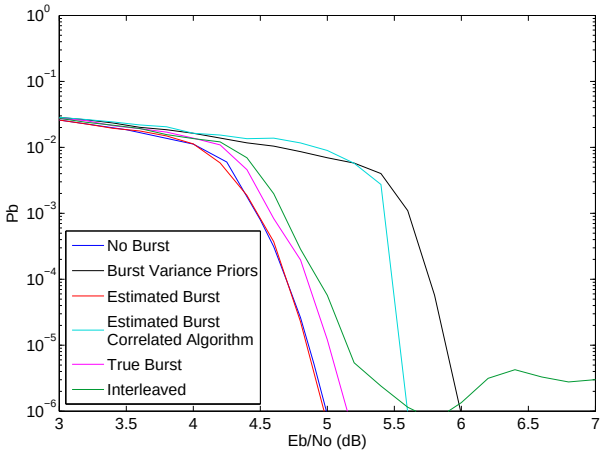

Fig. 5.34: Close-up of correlated burst comparison for rate .936 code with 2% burst time and SBNR of 3.6 dB.

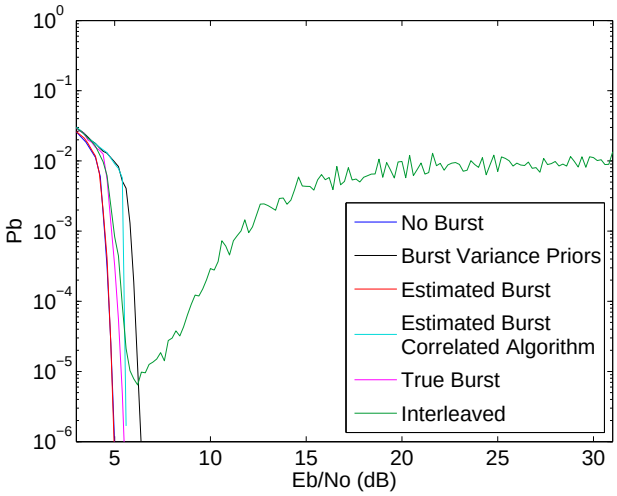

Fig. 5.35: Correlated burst comparison for rate .936 code with 5% burst time and SBNR of 3.6 dB.

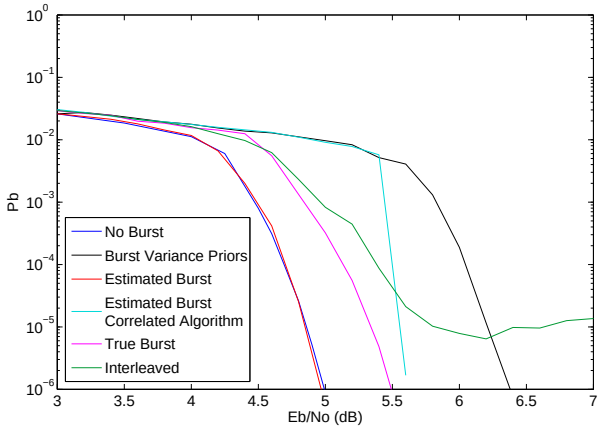

Fig. 5.36: Close-up of correlated burst comparison for rate .936 code with 5% burst time and SBNR of 3.6 dB.

Again, the $*$ indicates that the code did not achieve the desired probabiltiy of bit-error and the value is estimated.

As in the uncorrelated case, the performance in many cases is actually better with the burst than without. The rate .333 code loses between .5 and .7 dB in gain, while the other codes are much better. Strangely, the true priors actually degrade in performance as the noise goes up. This suggests that the extra noise is decreasing performance. More puzzling is that the estimated bursts are superior to the true burst. In fact, the performance does decrease as the burst time increases.

The coding gain for the correlated burst figures is summarized in Tables 5.4 through 5.6. The values for the burst variance only, the true priors, and the two estimated bursts are all compared to the no burst case.

Table 5.4: Coding gain comparison with a correlated burst and 1% burst time.

Code Rate	Interleaving	Burst Variance Priors	Correct Priors	Est. Burst Uncorr. Alg.	Est. Burst, Corr. Alg.
.333	0.0	-.5	.1	.1	.2
.7	-.1	-.4	-.1	0.0	0.0
.936	-.2	-1.0	-.1	0.0	-.6

Table 5.5: Coding gain comparison with a correlated burst and 2% burst time.

Code Rate	Interleaving	Burst Variance Priors	Correct Priors	Est. Burst Uncorr. Alg.	Est. Burst, Corr. Alg.
.333	0.0	-.5	0.0	-.1	-.1
.7	-.2	-.6	-.1	.1	0.0
.936	-1.6*	-1.0	-.2	0.0	-.6

Table 5.6: Coding gain comparison with a correlated burst and 5% burst time.

Code Rate	Interleaving	Burst Variance Priors	Correct Priors	Est. Burst Uncorr. Alg.	Est. Burst, Corr. Alg.
.333	-.4	-.7	-.4	.1	.1
.7	-.6	-.8	-.3	-.1	.1
.936	-1.5*	-.4	-.5	0.0	-.6

This performance is reflected in the rate .7 code, although there is some gain lost as the burst time goes up. This is most likely due to the higher code rate. Still, the performance with the estimated burst still exceeds the no burst case at 5% burst time for the estimated burst using the correlated algorithm.

Performance for the rate .936 code is still not as good as the other two, but with the estimated burst, the performance does not decrease with burst time. The correct priors perform more poorly than the estimated burst and there is more loss for the burst variance case. The performance in general shows the value of not ignoring the correlation in the prior probabilties.

The results support the conclusion in the uncorrelated burst case where the lack of confidence that the burst priors have helps the decoder correct the errors.

5.3 Conclusions

It is interesting to observe that the errors introduced by the burst detector can help in some cases, such as correlated bursts, but hurt in others, as for the uncorrelated burst. Even more interesting, the interaction between these errors and the code rate is critical for determining how well one actually wishes to identify the burst. There may be advantage in mis-identifying some state values. This could relate to our observation in Chapter 3 that some lies are better than others, at least where decoding is concerned. This suggests that additional research into burst detection and what it is actually doing could be fruitful.

The specific results we have shown support using the burst variance only to calculate prior probabilities in the uncorrelated burst case. In the correlated burst case, it is instead better to use the burst detector. Experimentation may be necessary to determine whether the correlated or uncorrelated algorithm is superior. There may be cases where superior performance is found when using the extra calculations required for the correlated detection algorithm. However, in our example cases, the performance is identical. This suggests that we would be equally well served to use the uncorrelated algorithm, which has the added benefit of increasing speed due to less computations.

For a general case, this suggests that general experimentation across multiple approaches may help optimize performance and complexity, as in some cases, the additional complexity has actually degraded our performance.

Chapter 6

Conclusions

6.1 Summary of Conclusions

We have demonstrated a method for improving LDPC decoding when burst errors are present. This method introduces several important contributions for combatting burst noise. These include:

- Developing a burst noise model that includes both random and burst noise.
- Documenting the effects of the combination of burst and random noise on LDPC codes.
- Evaluating the effects of using incorrect prior calculations to feed a soft decoder.
- Showing that interleaving combats bursts by spreading out noise, but in doing so, ignores information about the bursts.
- Demonstrating the added decoding benefits of considering the burst noise in prior probability calculations.
- Using the forward/backward algorithm to develop a novel burst detector.
- Showing that correlated burst errors result in correlated observations.
- Exploiting the added information from that correlation to improve coding gain even beyond the code performance in a burstless channel.
- Showing that LDPC decoders can include correlation information, unlike other block codes.
- Showing that LDPC codes can effectively deal with bursts when the bursts are properly considered.

- Showing that when correlation information is included, performance superior to interleaving is achievable. The performance of the decoder can actually be equal to the performance with no burst present.

This approach was presented for a specific burst model. We now extend this to a generalized method for improving LDPC decoding performance for any burst model.

6.2 Generalized Approach

While the models used were based on our specific burst model, the techniques applied are not restricted to that burst model only. In any system where burst noise is present, if the burst can be characterized, the techniques used herein can be applied. Using the burst statistics and characteristics, the forward/backward algorithm can be modified for any m-state Markov chain with any transition matrix. The prior probabilities for the decoder can also be modified to reflect the burst and non-burst characteristics. The approach can be generalized further by replacing the forward/backward algorithm with any generic burst detector.

Then, in order to employ the proposed approach, select an LDPC code, identify an appropriate model for the burst noise, apply the chosen burst detector, and choose the appropriate prior calculations. These priors are then fed to the decoder. Figure 6.1 shows a block diagram for this generic system.

There are some considerations for applying this method that should be considered. These include the optimality of the approach, the complexity of the calculations, and the generality of the approach to other soft decoders.

6.2.1 Optimality

By no means have we presented a proof of the optimality of this technique. However, one would hope that by using the correct prior probabilities, the decoder would perform better. As we noted in the discussion of the results, if the burst detector is not working well enough, the decoder may still fail to decode properly. We ended up using the burst

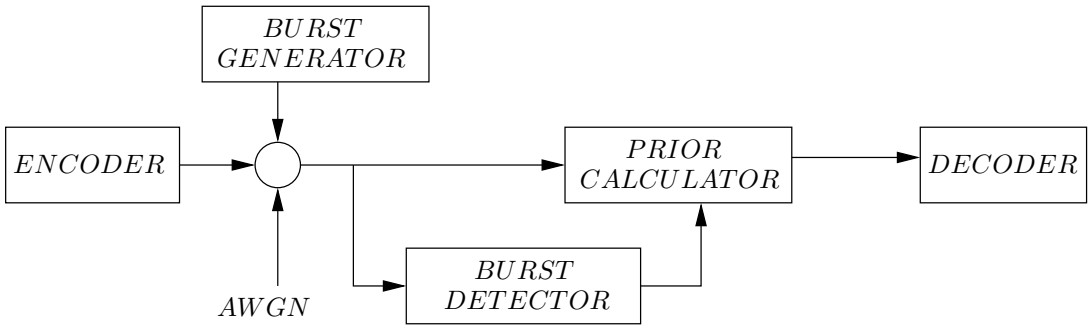

Fig. 6.1: Generic burst compensation system.

variance to optimize performance for the uncorrelated burst because of this. However, it was also clear that the true priors performed better than the burst variance priors if we knew the true burst.

It is clear, then, that the burst detector needs to work well enough to help the decoder. In the case of the forward/backward algorithm, it was good enough to improve performance. This was not the case for the uncorrelated burst. So each burst detector must be evaluated to see if it works well enough, as defined by the decoding capability of the system.

The ability of the detector to adequately find the burst will depend on the amount of difference between the different burst states. It will also depend on how good of a burst detector is used. The conclusion of this is that for each burst and detector, the system must be tested to see how it performs.

6.2.2 Complexity

The use of a burst detector may add increased complexity over other approaches. The complexity of the correlated algorithm is much greater than that of the uncorrelated burst. For more complex burst models, this could become worse, especially given how the increase in the number of states in the burst model increases complexity in the forward/backward algorithm.

Choosing a different burst detector could also help with complexity. For example, in the correlated burst case, the performance is very similar with the correlated and uncorrelated forward/backward algorithm. Since it is less complex, in this case it would be useful to

use the uncorrelated algorithm. Tradeoffs will have to be made depending on the desired performance and the allowable complexity.

6.2.3 Extending Beyond LDPC Codes

In this work, we have restricted ourselves to LDPC codes. However, in theory, the techniques that we have developed could be applied to any soft decoder. It would greatly depend on how the decoder calculated and utilized the soft information. Additional work would need to be done in order to test and verify the performance and utility for a given code. Additionally, any performance improvement would depend on the robustness of the code against burst noise and errors without any additional modifications. Regardless, it would be interesting to investigate the utility of the technique for other decoders.

6.3 Future Work

The development of the general model for other cases presents a large opportunity for additional work in this area. However, a good portion of this work is simply application of the techniques developed herein to a different system. This would be important for developing our techniques for application in various fields. Other opportunities for research exist to expand or improve the capabilities of what we have presented.

6.3.1 Burst Modeling and Detection

As mentioned in the development of the burst model, a simple two state burst model may not accurately represent a given burst. Different burst models could require more states. Investigation into different models is its own area of research, but if generalized models were to be developed for common burst phenomena, it would provide a useful tool for applying the techniques that have been developed herein.

Further, understanding the burst models will help develop burst detection algorithms. For example, in the discussion on interleaving, we mentioned long bursts on a single message. This would be vastly different from what we saw with our short burst times. This might require transition states so that the burst occurs and then stays in the burst state for a given

amount of time. This could create difficulty in detecting the burst as the burst variance would not be changing in the different states, so the information that defines states might depend entirely on detecting the first bit in the burst. This could place a premium on a burst detector that optimally identified the starting and ending points of a burst, rather than the entire burst, since the burst model would force the bits in the middle to be in the burst.

It is possible that a simple model that does not truly represent the burst would be sufficient if the burst detector was able to identify it. Exploring the sufficiency of a simple model and its impact on performance over a true model could provide gains in complexity as well as provide insight into the robustness of the burst detection and modeling techniques.

6.3.2 Code Performance Investigation

A number of interesting observations were made in the code performance when burst noise was introduced. It is especially interesting how the curves take on different shapes than would be typically expected, especially without burst compensation, as seen in Chapter 2. It would be useful to characterize the curves for code performance over different burst variances to investigate the different effects. It would also be useful to explore the impact of code rate and length on how the code performs to see if any conclusions could be drawn. As seen in Section 2.3, there are some performance trends that run contrary to standard coding performance.

It was also noted in a number of cases that the code performs better in the transition region between the random error dominated and burst error dominated regions. It could be useful to explore the practicality of using a system in this transition region while ignoring the burst or employing interleaving to see if the predicted performance could be realised. Careful controls would have to be developed in order to ensure that the SNR did not get too high or low.

6.3.3 Priors Calculations Using Incorrect Variance Values

We noted that there can be some performance improvement by lying to the decoder

and using an incorrect SNR to estimate priors [35]. It was also seen that the performance of the decoder using the burst variance only to calculate priors may be sufficient to improve performance, especially in the uncorrelated burst case.

Using this as a basis, it would be useful to find the optimal SNR to assume when decoding using only one value to calculate the priors. This might preclude the need, in some cases, of a burst detector. Instead, the variance of the burst and non-burst noises could be examined and used to determine an optimal operating point.

This could be extended to the case where we use multiple variances. It is possible that performance could be improved by varying the burst and/or non-burst variances used in the correct prior calculations. This could also be applied to the burst detector to see if any improvement can be made.

6.3.4 Burst Time Effects

It was noted that the performance improves as the burst time decreases. The decrease in burst time lowers the amount of burst noise added, improving the overall SNR. While it is not clear what this overall SNR is due to how the bursts vary in length, it is obvious that the burst is adding some amount of extra noise.

Insight into burst effects and the requirements of burst detection performance could be found if burst time effects were better understood. It is possible that a code has a threshold of burst time that prevents it from decoding successfully. Burst detection may only need to identify a certain percentage of the bursts in order to allow decoding.

6.3.5 Interleaving and Correct Prior Calculations

Considering the burst time suggests another approach to dealing with burst noise. While interleaving does not perform well given our burst model, it has proven to be effective in certain cases. If bursts were only present on some messages, even if they were sufficient to overwhelm the interleaver, by combining the interleaving effects with the use of burst detection and correct prior calculation, some performance gain could be seen. This would be useful in cases where the bursts alone on a single message overwhelmed the decoder. The

interleaver would reduce the amount of burst noise present, effectively lowering the overall burst times and improving performance.

6.4 Conclusion

Burst noise provide a serious challenge for any communications channel. We have demonstrated that proper inclusion of noise in the prior calculations greatly improves LDPC decoding performance when burst noise is present. Proper application of the techniques developed herein can help reduce complications caused by burst noise. This important contribution to communication and coding techniques provides a framework for future application as well as additional research opportunities.

References

[1] R. G. Gallager, "Low density parity check codes," *IRE Transactions on Information Theory*, vol. IT-8, no. 1, pp. 21–28, 1962.

[2] S. Song, S. Lin, and K. Abdel-Ghaffar, "Burst-correction decoding of cyclic LDPC codes," *Information Theory, 2006 IEEE International Symposium*, pp. 1718–1722, July 2006.

[3] D. J. C. Mackay, "Good error-correcting codes based on very sparse matrices," *IEEE Transactions on Information Theory*, vol. 45, no. 2, pp. 399–431, Mar. 1999.

[4] T. C. May and M. H. Woods, "Alpha-particle-induced soft errors in dynamic memories," *IEEE Transactions on Electron Devices*, vol. ED-26, no. 1, pp. 2–9, 1979.

[5] I. Leon Lantz, "Tutorial: Soft errors induced by alpha particles," *IEEE Transactions on Reliability*, vol. 45, no. 2, pp. 174–178, 1996.

[6] F. Matsuoka and F. Masuoka, "Numerical analysis of alpha-particle-induced soft errors in floating channel type surrounding gate transistor (FC-SGT) DRAM cell," *IEEE Transactions on Electron Devices*, vol. 50, no. 7, pp. 1638–1644, 2003.

[7] J. Chen, L. Wang, and Y. Li, "Performance comparison between non-binary LDPC codes and Reed-Solomon codes over noise bursts channels," *Communications, Circuits and Systems. Proceedings. International Conference*, vol. 1, pp. 1–4, May 2005.

[8] T. Morita, Y. Sato, and T. Sugawara, "ECC-less LDPC coding for magnetic recording channels," *IEEE Transactions on Magnetics*, vol. 38, no. 5, pp. 2304–2306, 2002.

[9] Y. Nakamura, Y. Okamoto, H. Osawa, H. Muraoka, and H. Aoi, "Burst detection by parity check matrix of LDPC code for perpendicular magnetic recording using bit-patterned medium," *Information Theory and Its Applications. ISITA 2008. International Symposium*, pp. 1–6, Dec. 2008.

[10] Y. Q. Shi, X. M. Zhang, Z.-C. Ni, and N. Ansari, "Interleaving for combating bursts of errors," *IEEE Circuits and Systems Magazine*, vol. First Quarter, pp. 29–42, 2004.

[11] S.-W. Lei and V. K. N. Lau, "Performance analysis of adaptive interleaving for OFDM systems," *IEEE Transactions on Vehicular Technology*, vol. 51, no. 3, pp. 435–444, 2002.

[12] N. Xie, T. Zhang, and E. Haratsch, "Improving burst error tolerance of LDPC-centric coding systems in read channel," *IEEE Transactions on Magnetics*, vol. 46, no. 3, pp. 933–941, Mar. 2010.

[13] S. Al-Bassam, B. Bose, and R. Venkatesan, "Burst and unidirectional error detecting codes," *Fault-Tolerant Computing. FTCS-21. Digest of Papers, Twenty-First International Symposium*, pp. 378–384, June 1991.

[14] Y. Saitoh and H. Imai, "Generalized concatenated codes for channels where unidirectional byte errors are predominant," *IEEE Transactions on Information Theory*, vol. 39, no. 3, pp. 1014–1022, 1993.

[15] S. Lin and D. J. Costello, Jr., *Error Control Coding Fundamentals and Applications*, 1st ed. Englewood Cliffs, NJ: Prentice Hall, Inc., 1983.

[16] S. J. Godsill and P. J. W. Rayner, "A Bayesian approach to the detection and correction of error bursts in audio signals," *Acoustics, Speech, and Signal Processing. ICASSP-92, IEEE International Conference*, vol. 2, pp. 261–264, Mar. 1992.

[17] R. G. Gallager, "Low-density parity-check codes," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, 1963.

[18] T. K. Moon, *Error Correction Coding: Mathematical Methods and Algorithms*. Hoboken, NJ: John Wiley & Sons, Inc., 2005.

[19] D. J. C. Mackay. (1999) .333 rate (1920,1280) LDPC code parity check matrix in sparse form. Available: http://www.inference.phy.cam.ac.uk/mackay/codes/matrices/1920.1280.A.

[20] ——. (1998) .7 rate (273,82) LDPC code parity check matrix in sparse form. Available: http://www.inference.phy.cam.ac.uk/mackay/codes/EN/C/273.82.3.353.

[21] ——. (1998) .936 rate (4376,282) LDPC code parity check matrix in sparse form. Available: http://www.inference.phy.cam.ac.uk/mackay/codes/EN/C/4376.282.4.9598.g%z.

[22] E. N. Gilbert, "Capacity of a burst-noise channel," *The Bell System Technical Journal*, vol. 39, pp. 1253–1265, Sept. 1960.

[23] B. D. Fritchman, "A binary channel characterization using partitioned Markov chains," *IEEE Transactions on Information Theory*, vol. IT-13, no. 2, pp. 221–227, Apr. 1967.

[24] S. Tsai, "Markov characterization of HF channel," *IEEE Transactions on Communication Technology*, vol. 17-1, no. 1, pp. 24–32, Feb. 1969.

[25] L. N. Kanal and A. R. K. Sastry, "Models for channels with memory and their applications to error control," *Proceedings of the IEEE*, vol. 66, no. 7, pp. 724–744, July 1978.

[26] B. Sklar, "Rayleigh fading channels in mobile digital communication systems part 1: Characterization," *IEEE Communications Magazine*, pp. 724–744, July 1997.

[27] D. Reed and W. Draheim, "Viterbi decoding burst errors and their effects on digital communication performance," *Military Communications Conference. IEEE*, vol. 1, pp. 220–225, Oct. 1985.

[28] C.-C. Chao and Y.-L. Yao, "Hidden markov models for the burst error statistics of Viterbi decoding," *Communications. Geneva. Technical Program, Conference Record, IEEE International Conference*, vol. 2, pp. 751 –755, May 1993.

[29] A. Franchi and R. Harris, "On the error burst properties of Viterbi decoding," *Communications. Geneva. Technical Program, Conference Record, IEEE International Conference*, vol. 2, pp. 1086–1091, May 1993.

[30] M. Ashouei, S. Bhattacharya, and A. Chatterjee, "Improving SNR for DSM linear systems using probabilistic error correction and state restoration: A comparative study," *Test Symposium. Eleventh IEEE European*, pp. 35–42, May 2006.

[31] A. Leo-Garcia, *Probability and Random Processes for Electrical Engineering*, 2nd ed. Reading, MA: Addison-Wesley Publishing Company, Inc., 1994.

[32] M. Horstein, "Efficient communication through burst-error channels by means of error detection," *IEEE Transactions on Communication Technology*, vol. COM-14, no. 2, pp. 117–126, 1966.

[33] C.-X. Wang and M. Patzold, "A novel generative model for burst error characterization in Rayleigh fading channels," *Personal, Indoor and Mobile Radio Communications. 14th IEEE Proceedings*, pp. 960–964, 2003.

[34] D. E. Knuth, *The Art of Computer Programming: Volume 2*, 3rd ed. Boston, MA: Addison-Wesley, 1998.

[35] W. Tan and J. R. Cruz, "Signal-to-noise ratio mismatch for low-density parity-check coded magnetic recording channels," *IEEE Transactions on Magnetics*, vol. 40, no. 2, pp. 498–506, 2004.

[36] W. Tan, H. Xia, and J. R. Cruz, "Erasure detection algorithms for magnetic recording channels," *IEEE Transactions on Magnetics*, vol. 40, no. 4, pp. 3099–3101, 2004.

[37] J. R. Deller, Jr., J. G. Proakis, and J. H. L. Hansen, *Discrete-Time Processing of Speech Signals*, 1st ed. New York, NY: Macmillan Publishing Company, 1993.