INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®
CONSTRAINED CODING AND SOFT ITERATIVE DECODING FOR STORAGE

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

John L. Fan
December 1999
I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

John M. Cioffi
(Principal Advisor)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

John T. H. Hill III
John Gill

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Brian H. Marcus

Approved for the University Committee on Graduate Studies:

Thomas Wasow
Abstract

Constrained coding, in which certain restrictions known as a modulation constraint are placed on the sequence that can be transmitted through a digital communications channel, is commonly used in magnetic and optical storage. Error-correcting codes (ECC) introduce redundancy into a sequence so that the receiver can correct errors that may occur during transmission over a noisy channel. The implementation of the modulation constraint, however, can often interfere with the error-correcting code. For hard-decision ECCs, such as Reed-Solomon codes, it turns out that a modified concatenation scheme, in which the positions of the modulation code and the ECC are reversed, can reduce error-propagation.

Error-correcting codes with soft iterative decoding algorithms, such as turbo codes and low-density parity check (LDPC) codes, have shown extremely good performance, approaching theoretical limits. The decoding algorithms for these codes are considered using the framework of factor graphs and the message-passing algorithm. Then turbo codes and LDPC codes are compared for the magnetic recording channel.

Finally, the modified concatenation scheme is applied to the problem of soft decoding, allowing the ECC decoder to obtain accurate soft information from the channel decoder. The interaction between the constrained code and the ECC is of fundamental importance for the application of turbo and LDPC codes to digital recording systems that use a modulation constraint. In addition, it is possible to introduce a constraint decoder that updates the probabilities based on the knowledge of the modulation constraint. Iterating between the constraint decoder and the ECC decoder can yield additional coding gain beyond the gain of the ECC.
Acknowledgements

"Think deeply of simple things." – Prof. Arnold E. Ross

Various mentors and managers have guided and influenced me during the course of my graduate studies, including Prof. Laci Babai, with whom I began graduate studies in mathematics at the University of Chicago; Rob Calderbank of AT&T Labs, where I worked for a year before coming to Stanford; Prof. Hesselink and Prof. John Gill at Stanford University; Masayuki Hattori and Norihisa Shirota at Sony Corporation in Tokyo, where I spent two summers; Don Shaver at Texas Instruments; and Brian Marcus of IBM Almaden, with whom I have had the pleasure of many discussions on constrained coding and iterative decoding. Finally, I would like to recognize the wisdom, patience and support of my Ph.D. advisor, Prof. John Cioffi, who has allowed me the freedom to pursue my research interests, and provided enthusiastic encouragement and advice. This work continues (and possibly concludes) a long series of dissertations in Prof. Cioffi’s research group in the area of magnetic storage.

I would like to thank Prof. Cioffi, Prof. Gill, and Brian Marcus for serving on my orals committee, and Prof. Shan X. Wang for chairing the committee. In addition, I would like to recognize Prof. Cioffi, Prof. Gill, and Brian Marcus for their time and effort in serving on my thesis reading committee. I would also like to express my appreciation to Texas Instruments for its extended research support, and to Hitachi America for providing me with a fellowship. And I would like to recognize the National Storage Industry Consortium (NSIC) for its support and for the opportunity to participate in the quarterly meetings.

In addition, I would like to express my appreciation to Jorge Campello, David Forney, Arnon Friedmann, Ehud Gelblum, Masayuki Hattori, Errozan Kurtas, Joseph
Lauer, David MacKay, Steve McLaughlin, Christof Paar, Tom Richardson, Ronny Roth, Emina Soljanin, Adriaan van Wijngaarden, Bruce Wilson, and Zining Wu for helpful discussions and collaborations related to magnetic storage, error-correcting codes and iterative decoding.

My fellow students in Prof. Cioffi’s group, including Acha, Susan, Kok-wui, Joonsuk, Ardy, Wonjong, Wonjoon, Jose, Louise, Atul, Carlos, Wei, Jeannie, and Steve, as well as other students such as Sriram, Xin, Jiapei, Yiannis and Sunil have been great to work with, and I hope to have the opportunity to do so again someday. In addition, I would like to thank Joice for her tireless administrative support and interesting conversation, and Denise for thinking up creative ISL activities. A word of recognition goes to my quals study groups, including Brian, Scott, Liyi, Krishna, Oskar, and especially to Pablo, for creating Calendus (Stanford’s web calendaring system) with me.

Finally, I would like to thank my family, Mom, Dad, Linda and Robert, for their encouragement and support, and my friends, who gave meaning to my time here at Stanford, including Andy, Ike, Pauline, Erik, Yoshi, Elizabeth, Huey, Sophia, Chiao-Chuan, Veronica, Hung-ken, Shine, Sydney, San-San, Emily, and others from EE, AAGSA, SOTA, STSA, and Monte Jade.

This thesis is dedicated to my parents.
Contents

Abstract iv

Acknowledgements v

1 Introduction 1
  1.1 Storage systems ............................................. 1
  1.2 Soft iterative decoding ...................................... 3
    1.2.1 Probabilities ............................................ 4
    1.2.2 Soft decoding modules .................................. 5
    1.2.3 Low density parity check codes ......................... 6
  1.3 Overview ..................................................... 7

2 Constrained coding for hard decoders 9
  2.1 Constrained coding .......................................... 9
  2.2 Concatenation ............................................... 13
    2.2.1 Standard concatenation .................................. 14
    2.2.2 Modified concatenation .................................. 14
  2.3 Analysis of error propagation ............................... 19
    2.3.1 Decoder performance, with independent errors ........... 19
    2.3.2 Error propagation ....................................... 22
    2.3.3 Decoder performance, with burst errors .................. 25
  2.4 Lossless compression ....................................... 27
    2.4.1 Lossless compression for modified concatenation .......... 27
    2.4.2 Block codes for lossless compression .................... 30
3 Soft iterative decoding

3.1 Factor graphs and message-passing
3.1.1 Definition
3.1.2 Example: Counting soldiers
3.1.3 Combining multiple decoders

3.2 Low-density parity check codes
3.2.1 Message-passing (probabilities)
3.2.2 Message-passing (log-likelihood ratios)
3.2.3 Approximations
3.2.4 Performance on AWGN channel
3.2.5 Complexity

3.3 Forward-Backward Algorithm
3.3.1 Derivation
3.3.2 Intersymbol interference channel
3.3.3 Turbo codes
3.3.4 Complexity

3.4 Comparison of turbo and LDPC codes
3.4.1 Complexity and performance
3.4.2 Comparison for magnetic storage channel
3.4.3 Thermal asperities

4 Constrained coding for soft decoders

4.1 Soft decoding and constrained coding
4.1.1 Standard concatenation
4.1.2 Modified concatenation
4.1.3 Bit insertion technique
C.2 Correcting 1 error and 1 erasure ........................................... 138
C.3 Correcting 1 error and r-2 erasures ........................................... 139

Bibliography 142
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Comparison of concatenation schemes</td>
<td>18</td>
</tr>
<tr>
<td>2.2</td>
<td>Comparison of standard and modified concatenation for rate 16/17</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>modulation code</td>
<td></td>
</tr>
<tr>
<td>2.3</td>
<td>Comparison of standard and modified concatenation for DC-free</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>modulation code</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>Comparison of complexity of turbo and LDPC codes, per bit per iteration</td>
<td>79</td>
</tr>
<tr>
<td>3.2</td>
<td>Some examples of turbo and LDPC decoding complexity, per bit per iteration</td>
<td>80</td>
</tr>
<tr>
<td>4.1</td>
<td>Comparison of systematic modulation and capacity for the (0, k)</td>
<td>88</td>
</tr>
<tr>
<td></td>
<td>constraint</td>
<td></td>
</tr>
<tr>
<td>4.2</td>
<td>Lower and upper bounds for use of bit insertion</td>
<td>93</td>
</tr>
<tr>
<td>4.3</td>
<td>Rates for systematic (d, k) modulation codes</td>
<td>99</td>
</tr>
<tr>
<td>4.4</td>
<td>Systematic modulation rate vs. capacity</td>
<td>99</td>
</tr>
<tr>
<td>4.5</td>
<td>Comparison of soft demodulation and the soft constraint decoder</td>
<td>105</td>
</tr>
<tr>
<td>4.6</td>
<td>Coding gain from the soft constraint decoder</td>
<td>109</td>
</tr>
<tr>
<td>B.1</td>
<td>Systematic modulation codes for the (0, G/I) constraints</td>
<td>133</td>
</tr>
<tr>
<td>B.2</td>
<td>Maximum rates for (0, G/I) systematic modulation codes</td>
<td>133</td>
</tr>
<tr>
<td>B.3</td>
<td>Bounds for bit insertion for (0, G/I) codes</td>
<td>136</td>
</tr>
</tbody>
</table>
# List of Figures

1.1 Standard framework for concatenating the channel decoder, the modulation constraint and the error-correcting code. ........................................ 2

2.1 Graph for \((d, k)\) constraint ................................................. 10
2.2 Constraint graph for \((2, \infty)\) constraint ............................... 11
2.3 Block diagram of standard and modified concatenation .............. 15
2.4 Standard and modified concatenation ..................................... 15
2.5 Interleaved Reed-Solomon codewords .................................... 20
2.6 Modified concatenation with lossless compression ................... 28
2.7 Performance comparison for 16/17 code .................................. 38
2.8 A comparison of the effects of long bursts for the 16/17 code ...... 39
2.9 Performance comparison for DC-free codes ............................ 42

3.1 Factor graph ................................................................. 44
3.2 Message-passing of probabilities on a cycle-free graph .............. 46
3.3 Counting soldiers as an example of message-passing ................. 51
3.4 Message-passing works on graphs without cycles .................... 52
3.5 Iterating amongst multiple decoders ................................... 53
3.6 Iterating between two decoders ......................................... 54
3.7 Low Density Parity Check (LDPC) codes ................................ 56
3.8 \(L\) bits that satisfy an even parity check constraint ............... 57
3.9 Performance of an LDPC decoder after different numbers of iterations 66
3.10 Forward-backward algorithm as message-passing .................... 69
3.11 Trellis for intersymbol interference channel .......................... 74
3.12 Three configurations for turbo coding .......................... 77
3.13 Comparison of turbo and LDPC codes for an AWGN channel .... 80
3.14 Channel block diagram ........................................... 82
3.15 Message passing between the channel decoder and the LDPC decoder 83
3.16 Performance of turbo and LDPC codes on an ideal EPR4 channel .. 84
3.17 AWGN channel with thermal asperities (rate 8/9) .................. 85
3.18 EPR4 channel with thermal asperities, LDPC vs. parallel turbo code 86
3.19 EPR4 channel with thermal asperities, LDPC vs. serial turbo code . 86

4.1 Configurations for using soft information with a modulation constraint 90
4.2 Bit insertion technique ............................................. 90
4.3 Ranges of the ECC rate where bit insertion is more efficient .... 94
4.4 Systematic modulation code ....................................... 98
4.5 Trellis for MFM code ............................................. 103
4.6 Comparison of soft demodulation for MFM code ................... 104
4.7 Soft decoder for the constraint .................................. 107
4.8 Iterating with the soft constraint decoder ........................ 108
4.9 Performance gains from iterating with the soft constraint decoder . 110
4.10 LDPC and constraint decoder for an optical disc with a (2,10)-RLL constraint ............................................. 111
4.11 Applying the $d = 1$ constraint to the EPR4 channel ............. 113
4.12 Iterating a constraint decoder with the channel .................. 114

5.1 Decoding an array code using soft decoding ...................... 122
Chapter 1

Introduction

Recent breakthroughs in the design of error-correcting codes that use iterative probabilistic decoding will have a significant impact on many communication systems. This thesis considers various issues in the application of these advanced error-correcting techniques to magnetic and optical storage systems. With the increasing requirements of computer systems and databases, as well as the exponential growth of the Internet, the need for high-capacity data storage has grown substantially. The use of these coding techniques improves the performance of the hard disk in the presence of noise, allowing for higher recording densities to be used; meanwhile, the decreasing costs of VLSI technology have made it possible to consider more sophisticated coding schemes.

1.1 Storage systems

The standard framework for the signal processing and coding in a disk storage system is depicted in Figure 1.1. At a very high level, the three basic components of the system are:

- **Channel decoder**. This decodes the channel, which introduces intersymbol interference and noise.
• **Modulation code.** This restricts the transmitted sequence to satisfy certain constraints, for various reasons such as avoiding common error events and timing recovery.

• **Error-Correcting Code (ECC).** This introduces redundancy into the sequence to create distance between coded sequences, so that the decoder can correct errors that may result from transmission through a noisy channel.

![Diagram of ECC and Channel Processing]

Figure 1.1: Standard framework for concatenating the channel decoder, the modulation constraint and the error-correcting code.

In a typical magnetic recording system, the properties of the magnetic media require that it is polarized entirely one way or the other, so that the input to the channel is constrained to binary signals. The data is read and processed in sectors, with the typical disk drive containing sectors that correspond to 512 bytes of user data each.

There is intersymbol interference (ISI) on the channel, in which the received samples are influenced by the neighboring bits. In partial response maximum likelihood (PRML), the channel is equalized to a partial response polynomial which roughly matches the spectrum of the channel; the channel samples are then decoded using the channel decoder, which is based on the Viterbi algorithm. [Kob71]. This channel decoder takes the received samples and produces a sequence of bits, corresponding to its estimate of the maximum likelihood sequence. Non-linearities in the intersymbol interference may require some appropriate adjustments to the Viterbi decoder, and in addition, some alternative approaches for handling the intersymbol interference of the channel are Decision Feedback Equalization (DFE) and signal-space approaches.
CHAPTER 1. INTRODUCTION

The modulation code encodes the sequence so that it meets a particular modulation constraint, which may be required for reasons such as avoiding common error events, timing recovery, DC-balance, and physical restrictions. These modulation codes are usually implemented by a look-up table or dedicated circuitry. Finally, the error-correcting code (ECC), which is usually a byte-based Reed-Solomon code, is used to correct multiple random symbol errors. In addition, errors may occur in bursts due to error propagation caused by the channel and constraint decoders, as well as due to defects on the media and phenomena such as thermal asperities. As a result, the Reed-Solomon codewords are usually interleaved, so that a burst of errors is spread across different codewords, allowing for the correction of error bursts.

It is desirable to have modulation codes and error-correcting codes that are high rate and low complexity. To compensate for the loss in disk capacity due to the overhead due to these codes, one can increase the recording density (the number of bits per unit area of a disk). But due to the limitations of the magnetic recording channel, increasing the density results increases the intersymbol interference and introduces various non-linear noise effects. In channels where the density is already pushed very high, it is then essential that the coding redundancy is kept to a minimum. In addition, read channels operate at speeds of several hundred megabits per second, which calls for fast, low complexity implementations of the coding schemes. Some other important issues for the read channel include timing recovery, channel identification, equalization, and quantization. [Abb91][Ber96].

1.2 Soft iterative decoding

The discovery of turbo codes [BGT93] and the rediscovery of low density parity check (LDPC) codes [MN97][Mac99], which were first introduced in 1962 by Gallager [Gal62], have shown that reliability information about the channel bits can be used to greatly improve the performance of decoding systems, achieving performance that is close to the theoretical limits with reasonable complexity. Error-correcting code (ECC) refers to any code in which redundancy is introduced for the purpose of correcting errors, and is traditionally associated with algebraic hard-decision decoders,
such as Reed-Solomon codes. We introduce the terminology *soft error-correcting code (soft ECC)* to refer specifically to codes that make use of soft information, such as low density parity check (LDPC) codes and turbo codes.

### 1.2.1 Probabilities

For a binary variable $x \in \{0, 1\}$, the probability $p$ that $x = 1$ can be represented in several ways. The *log-likelihood ratio*, $\text{LLR}(p) = \log \frac{p}{1-p}$, is useful since it converts multiplications into additions. The *soft bit* $\chi(p) = 2p - 1$ lies in the range $[-1, 1]$. The relationship between the soft bit (ref. [HOP96]) and the log-likelihood ratio can be expressed using the hyperbolic tangent $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.

$$\chi(p) = \tanh \left( \frac{1}{2} \text{LLR}(p) \right)$$

It is useful to review some terminology for probabilities. If $p = P(x = 1)$ is the probability that a bit $x$ is equal to 1, then with respect to an observation $y$, the *prior* (or *a priori*) probability $p = P(x = 1)$ refers to what was known before the observation, while the *posterior* (or *a posteriori*) probability $P(x = 1 | y)$ is the conditional probability given the observation $y$. Finally, the *extrinsic* information is what has been learned from the observation $y$, and is the difference of the posterior and prior information, when expressed as log-likelihood ratios:

$$\text{LLR}_{\text{prior}}(x) + \text{LLR}_{\text{extrinsic}}(x) = \text{LLR}_{\text{posterior}}(x)$$

In terms of probabilities, the relation is given as:

$$p = \frac{ae}{ae + (1-a)(1-e)}$$

where $p = P_{\text{posterior}}(x)$, $a = P_{\text{prior}}(x)$, and $e = P_{\text{extrinsic}}(x)$. 
1.2.2 Soft decoding modules

Following the example of the soft ECC decoder, which operates as a "soft-in, soft-out" decoder, it is desirable to transform all aspects of the decoding system to fit the soft decoding paradigm. The overall decoding system consists of "soft-in, soft-out" modules which exchange probabilistic information. The extrinsic principle, which follows from the message-passing algorithm, is used to connect the various modules together, as described in §3.1.3. The input to a module is the prior information with respect to that module, while the updated information based on that decoding module is called the posterior information. The extrinsic information indicates the new information which has been obtained by that module. These decoder modules are also referred to as a posteriori probability (APP) decoders.

The components of the magnetic recording channel—the error-correcting code, demodulation, and the decoder for the intersymbol interference channel—all produce hard-decision outputs in traditional magnetic storage systems. To apply a turbo code or LDPC code to magnetic storage, there are some issues that need to be addressed to accommodate the soft ECC decoder. A soft decoder module is introduced for the decoder for the intersymbol interference channel, in order to provide soft information about the decoded bits. This soft channel decoder is based on the forward-backward algorithm (ref. Section 3.3), also known as the BCJR algorithm [BCJR74]. Then, we introduce a revised configuration for the modulation scheme such that soft information is allowed to pass through, as well as a soft decoder module for decoding the constraint.

Thinking in terms of implementations, it is useful to classify the decoder modules based on how the algorithms can be run: serial processing, where the decoder operates on a sliding window of data, and packet processing, where the decoder reads in a packet (or sector) and operates over its entirety. In addition, a distinction can be made as to whether the algorithm is parallelizable—i.e. whether it is possible to break the decoding procedure into small steps that can be run simultaneously, allowing for efficient hardware implementations. Examples of serial processing include the Decision-Feedback Equalizer (DFE) and the Viterbi algorithm. On the other
hand, the forward-backward algorithm (ref. Section 3.3) requires a packet processing approach, but is not parallelizable. The message-passing algorithm for LDPC codes (ref. Section 3.2), however, requires packet processing but is parallelizable. Most existing signal processing and coding algorithms for storage work in a serial processing manner, but the feasibility of implementing a packet-based approach for magnetic recording systems is considered in [Lin99].

1.2.3 Low density parity check codes

Low density parity codes (LDPC) is a binary linear error-correcting codes with a sparse parity check matrix $\mathbf{H}$. The code is defined by the set of all words $\mathbf{x}$ such that $\mathbf{Hx} = 0$. As a very simple example of a parity check, consider the following code, whose codewords satisfy the following equation:

$$
\begin{bmatrix}
1 & 1 & 1
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3
\end{bmatrix} = 0

x_1 \oplus x_2 \oplus x_3 = 0
$$

Suppose that the decoder is given prior probabilities $p_i = \Pr(x_i = 1)$ on all the bits. Then it is possible to use the parity check to update the values of the probabilities.

For example, suppose that $p_1 = p_2 = 0.9$ and $p_3 = 0.5$. In other words, $x_1$ and $x_2$ are very likely to be 1, but no information is available about $x_3$. Then it is possible to estimate an updated probability on $x_3$ based on this parity check, by only considering the probabilities for combinations of $x_1$, $x_2$, and $x_3$ that satisfy this parity check.

$$
\Pr(x_3 = 1 \mid \mathbf{Hx} = 0) = \frac{\Pr(x_3 = 1, \mathbf{Hx} = 0)}{\Pr(\mathbf{Hx} = 0)}
$$

$$
= \frac{\Pr(x = 011) + \Pr(x = 101)}{\sum_{\mathbf{Hx} = 0} \Pr(x)}
$$

$$
= \frac{0.09}{0.5} = 0.18
$$
So the probability estimate for $x_3$ is lowered to 0.18, using the probabilities $p_1$ and $p_2$. In other words, since $x_1$ and $x_2$ are more likely to be 1, then $x_3$ is likely to be 0. This example illustrates the basic principle for updating probabilities.

As discussed in Section 3.2, the message-passing algorithm is a probabilistic decoding algorithm for decoding LDPC codes that iteratively updates the probabilities based on the parity check constraints in the above manner. The algorithm works by performing an update to each of the bits involved in the parity check, which is determined by the location of the 1’s in the parity check matrix $H$. For the algorithm to work effectively, it is necessary to use sparse matrices $H$, and hence the name “low-density” parity check codes.

As discussed in Chapter 3, the decoding of LDPC codes and turbo codes can be understood in the same framework, known as the message-passing algorithm on factor graphs. Section 3.4 shows that for the magnetic recording channel, LDPC codes perform as well as turbo codes, but require lower complexity.

1.3 Overview

This thesis considers the dual themes of constrained coding and soft iterative decoding. Chapter 2 describes constrained coding, and shows how a modified concatenation scheme provides a method for decreasing error propagation with a hard-decision error-correcting code, such as a Reed-Solomon code. Chapter 3 introduces soft iterative decoding, using the framework of factor graphs to explain message-passing for LDPC decoding and the forward-backward algorithm for channel decoding and for turbo codes. LDPC and turbo codes are then compared for use in magnetic recording. Next, Chapter 4 considers the question of how to incorporate constrained coding into the soft iterative decoding paradigm. The modified concatenation scheme is used to make soft information accessible to the ECC decoder, and a method is provided for decoding the modulation constraint in a soft manner.

Finally, some additional topics related to soft iterative decoding are briefly considered in Chapter 5, which gives an array code as an example of an algebraic error-correcting code that is also a low-density parity check code, and in Appendix A,
which discusses the use of soft cancellation as a low-complexity soft decoder for inter-symbol interference. Both topics deserve further investigation.

The original contributions in this thesis include the following:

- Modified concatenation reduces the error propagation with conventional ECCs, leading to fewer interleaves in a disk drive sector (Chap. 2) [FC98].

- Sliding-block version of lossless compression codes can be constructed using a variation of the state-splitting algorithm for constructing encoders (Chap. 2) [FMR00].

- A comparison is made of turbo codes and LDPC codes for magnetic recording (Chap. 3) [FKFM99][FKFM00].

- Modified concatenation allows the soft ECC decoder to soft information using constrained coding (Chap. 4) [FCi99].

- Systematic modulation codes can be constructed for the \((d, k)\) and \((0, G/I)\) constraints, and a method is described for constructing soft demodulation codes (Chap. 4) [FCi99].

- A soft decoder module for the modulation constraint is constructed using the forward-backward algorithm, which provides additional coding gain in the iterative decoding algorithm (Chap. 4) [FCi99].

- It is observed that array codes can also be decoded as low density parity check codes, and guidelines are given for avoiding short cycles in the factor graph (Chap. 5).
Chapter 2

Constrained coding for hard decoders

In this chapter, the problem of using a modulation constraint with a hard-decision error correcting code (ECC), such as a Reed-Solomon code, is discussed. Section 2.1 introduces constrained coding. Section 2.2 describes how error-propagation occurs when a modulation codes is concatenated with an ECC in the standard way, and addresses this problem with a modified concatenation scheme, which involves reversing the order of modulation and ECC. Section 2.3 analyzes the error-propagation and Section 2.5 gives examples of modulation codes for magnetic recording that benefit from this scheme. Meanwhile, Section 2.4 describes how a lossless compression scheme can be integrated into the modified concatenation scheme to improve its performance. This material has been published in [FC98] and [FMR00].

This modified concatenation scheme can also be applied to the problem of handling soft information, as discussed in Chapter 4.

2.1 Constrained coding

Constrained coding is a process in which unconstrained user sequences are encoded in a lossless manner into a sequences that satisfy certain modulation constraints, and is used in various data storage applications, such as magnetic disk drives, magnetic tape
drives, optical discs and digital video tape recorders. Recent comprehensive surveys of constrained coding can be found in [ISW98, KSS99, MRS99].

A *constrained system* consists of a set \( S \) of sequences that satisfy a constraint. This constraint can be defined through a labeled finite directed graph \( G \), known as the *constraint graph*, whose vertices known as states and whose edges are labeled by elements of a finite alphabet \( \Sigma \). The constrained sequences are the finite words that are obtained by reading the labels of paths in \( G \). A labeled graph is said to have *finite memory* if there exists some integer \( N \) such that all paths of length \( N \) that generate the same word terminate at the same state, so that for a sufficiently long sequence, it is possible to determine the corresponding path on the constraint graph.

A special class of modulation constraints that are important for storage are the \((d, k)\)-run-length-limited (RLL) constraints [Imm91]. These are defined as the set of binary words whose 1's are separated by runs of zeros of length at least \( d \), and at most \( k \), as pictured in Figure 2.1. Since magnetic recording systems work by detecting transitions in the magnetization of the media, a convention known as NRZI is used, in which a 1 indicates a transition and 0 indicates the lack of a transition. Having a non-zero minimum separation \( d \) can reduce the intersymbol interference or distortion associated with adjacent transitions. Having an upper bound on the maximum separation \( k \) forces occasional transitions, which may be useful for timing recovery.

![Figure 2.1: Graph for \((d, k)\) constraint](image-url)
The capacity of a constraint $S$ is defined by

$$\text{cap}(S) = \lim_{n \to \infty} \frac{1}{n} \log_2 |S_n|,$$

where $|S_n|$ is the number of sequences of length $n$ that satisfy the constraint $S$. For a constrained sequence, it is possible to compute the capacity of the constraint as follows. [MRS99]. The transition matrix $A$ is a square matrix that represents the possible transitions on the constraint graph, with $a_{ij} = 1$ if there is a directed edge from the $i$-th state to the $j$-th state, and $0$ otherwise. For example, the $(2, \infty)$-RLL constraint is generated by the constraint graph $G_{2,\infty}$ in Figure 2.2. The corresponding transition matrix for the $(2, \infty)$-RLL constraint is

$$A = \begin{bmatrix}
0 & 1 & 0 \\
0 & 0 & 1 \\
1 & 0 & 1
\end{bmatrix}.$$

Then from Perron-Frobenius theory, it turns out that the capacity can be computed as $\log_2 (|\lambda|)$, where $\lambda$ is the maximum eigenvalue of the transition matrix $A$ of the constraint graph. In the case of the $(2, \infty)$-RLL constraint, the capacity is $0.5515$.

There are various techniques for creating encoders for the constraints, including block modulation codes, sliding-block codes, enumerative techniques and sequence replacement techniques. Block modulation codes consist of binary block codes of rate $K/N$, where the modulated blocks (of length $N$) all satisfy the modulation constraint, and in addition maintain the constraints when modulation blocks are
arbitrarily placed side by side. As a block code, these are in general nonsystematic and nonlinear, and usually consist of fairly arbitrary assignments. In [Imm97], Immink provides an efficient construction for modulation block codes for \((d, k)\) constraints which are hundreds of bits long and approach the capacity of the constraint, which would be useful if there is a way to use block codes with long block lengths.

Sliding-block codes are encoded by a finite-state machine in such a way that demodulation of sliding-block codes is determined by a sliding block window, which may involve memory \(m\) and anticipation \(a\). In other words, the decoding is determined by looking at a window of previous \(m\) blocks, the current block, and the next \(a\) blocks, for a total of \(m + a + 1\) blocks of \(N\) bits each. For any constraint graph, it is possible to apply the state-splitting algorithm to construct an encoder for a sliding-block decodable modulation code for any rate less than the capacity of the modulation constraint.

**Theorem 1** (Adler, Coppersmith, Hassner [ACH83]) Let \(S\) be a finite memory constrained system, and suppose \(\frac{K}{N} \leq \text{cap}(S)\). Then for this constrained system \(S\), there exists a modulation code of rate \(K/N\) that is sliding-block decodable.

Other modulation techniques include enumerative coding schemes, which create constrained sequences using an algorithmic procedure rather than a look-up table, as discussed in [Cov73][IJ99]. In the sequence replacement technique [WI97], minor modifications (deletions and shifts) are performed on the sequence to make it satisfy the modulation constraint. Our focus will be on block modulation codes, although the discussion applies to the other constrained coding methods.

Other constraints include the \((0, G/I)\) constraint, which is often used in magnetic drives using partial response maximal likelihood (PRML), and is considered in Appendix B. [Wol91]. This constraint divides the sequence into two interleaves (i.e. the odd interleave \(\{x_{2i+1}\}\) and the even interleave \(\{x_{2i}\}\)) and specifies a \((0, I)\)-RLL constraint on each interleave, as well as a global \((0, G)\)-RLL constraint. Some constraints for partial response systems that use higher-order polynomials are the maximum transition run (MTR) constraints. [ISW98], which eliminate the minimum distance error events. One example of an MTR constraint for the EPR4 channel is given by the
(d, k_1, k_2) = (0, 2, 2) MTR constraint for the EPR4 channel, where in addition to d and k_1, the minimum and maximum run of 0’s in the sequence, the MTR constraint specifies k_2, the maximum allowable run of 1’s in the NRZI sequence. Finally, restrictions on the spectral properties and running digital sum (RDS) are also appropriate for partial response channels are discussed in [KSS99, TIH95].

For systems that allow multilevel recording, there has been work on multilevel runlength limited (RLL) constraints. [McL97]. In addition, for page-oriented memories such as holographic data storage, the use of two-dimensional modulation codes is of great interest. [AM98][VBSS96]. With appropriate modifications, the ideas described in this thesis for combining the modulation constraint with the ECC can also be applied to these other constraints.

2.2 Concatenation

The modified concatenation scheme, in which the input to the ECC is modulated by the constraint, was introduced by Bliss [Bli81] and considered by Mansuripur [Man91], Immink [Imm97], and Fan and Calderbank [FC98]. These papers consider the case where the ECC is an algebraic code such as a Reed-Solomon code, and show that a primary benefit of modified concatenation is to reduce the error propagation caused by the demodulation step. For soft ECCs as shown in Chapter 4, a key benefit of the modified concatenation scheme is to allow the ECC decoder to obtain accurate reliability information about the channel bits, which would otherwise be obscured by the demodulation of the modulation constraint.

The ECC operates on symbols of size B bits (corresponding usually to a finite field GF(2^B)), in which the systematic encoder takes m message symbols and generates r parity symbols, for a total of n = m + r symbols. The ECC is assumed to be a high rate code, so that r/n is typically less than 0.1. This chapter focuses on hard-decision ECCs, such as Reed-Solomon codes, where a typical value of B is 8. For soft ECCs, which are considered in Chapter 4, the symbol size is usually B = 1. The modulation code will be assumed to be a block code, but modified concatenation also applies to other constrained coding methods.
2.2.1 Standard concatenation

In the standard method of concatenation, shown in top half of Figure 2.3, the modulation coding lies strictly in between the encoder and decoder for the ECC code. The systematic encoder takes \( m = m_0 \) message symbols of \( B \) bits each, and encodes to obtain \( p \) parity symbols, giving a total of \( n = m + r \) symbols. These are then modulated into \( \frac{K}{N} (m + r)B \) bits of data sent through the channel, so the overall rate is \( \frac{K}{N} \cdot \frac{m}{n} \).

Suppose that the modulation code is a binary block code of rate \( K/N \). Unless the encoder is specially constructed, the encoder essentially consists of an arbitrary assignment of \( K \) input bits to \( 2^K \) words of length \( N \) bits that satisfy the modulation constraint (and maintain the constraint across blocks). As a result, a bit error (or a few bit errors) in a block of \( N \) bits will lead to the choice of the wrong word upon demodulation, resulting in error propagation. In particular, a single bit error can propagate into \( \lceil K/B \rceil \) symbol errors in the worst case, and a short error burst on the boundary of two blocks can result in \( \lceil 2K/B \rceil \) symbol errors. The effect is worse for large \( K \), so that to reduce the error-propagation due to demodulation, it is generally good to choose smaller values of \( K \), such as \( K = B \), which imposes a restriction on choice of the block modulation code.

2.2.2 Modified concatenation

The modified concatenation method, in which the order of the modulation code and ECC decoder is reversed, can allow the use of an arbitrary modulation codes, reduce error propagation, and facilitate the use of soft decoding. The modified concatenation scheme was introduced by Bliss [Bli81] and Mansuripur [Man91], and studied by Immink [Imm97], and Fan and Calderbank [FC98] as a method of preventing error-propagation with Reed-Solomon codes. The bottom half of Figure 2.3 shows the modified concatenation scheme, which uses two modulation codes for the same constraint. Another representation of these concatenation schemes is given in Figure 2.4.

Consider two types of modulation codes which produce modulated sequences for
CHAPTER 2. CONstrained CODING FOR HARD DECODERS

Standard concatenation

Modified concatenation

Figure 2.3: Block diagram of standard and modified concatenation

Standard concatenation

Modified concatenation

Italics indicate that the word satisfies the constraint

Figure 2.4: Standard and modified concatenation
the same constraint. The first modulation code $C_1$ has no restrictions, so it can have a rate $K_1/N_1$ arbitrarily close to the capacity of the constraint. The second modulation code $C_2$ is chosen in such a way as to prevent error propagation, much as with the modulation code in standard concatenation. In other words, for a block implementation of the code, if the encoder takes $K_2$ user bits and outputs $N_2$ modulated bits, then it is desirable to choose $K_2 = B$ in order to prevent error propagation. In summary, the use of modified concatenation guarantees that no error-propagation occurs on the message bits, so that the code $C_1$ can have arbitrary design, with arbitrary block lengths, so in particular, $K_1$ and $B$ need not match. Meanwhile, the parity bits do not suffer error-propagation due to the choice of $K_2 = B$. Hence, all the data reaching the channel satisfies the modulation constraint, and there is no error-propagation due to the modulation code $C_1$.

In addition, with block codes for the modulation codes, it is often difficult to associate reliability information for the demodulated output. With modified concatenation, it is directly obtain soft reliability information for the message bits. This soft information can be used for soft ECCs such as turbo codes and LDPC codes, as explored in Chapter 4, as well as for erasure decoding of Reed-Solomon codes [Coo94].

**Comparison of overhead**

Comparing these two concatenation schemes more precisely, standard concatenation starts with $m = m_0$ message symbols that are encoded using the systematic ECC encoder to obtain $r$ parity symbols, for a total of $n = m + r$ symbols. These symbols are then modulated into $\frac{N}{K} B(m + r)$ bits, which are sent through the channel, so the overall rate is $\frac{K}{N} \cdot \frac{m}{n}$. On the other hand, in modified concatenation, the initial $m_0$ message symbols are first modulated into roughly $\frac{N_a}{K_1} Bm_0$ bits using code $C_1$, and these are transmitted over the channel. Meanwhile, the modulated message bits are also passed into a systematic ECC encoder that appends $Br'$ parity bits, for a total of $\frac{N_a}{K_1} Bm_0 + Br'$ bits in an ECC codeword. Finally, the parity bits, which do not satisfy the constraint, must be passed into an encoder for a modulation code $C_2$ with rate $K_2/N_2$, resulting in a total of $\frac{N_a}{K_1} Bm_0 + \frac{N_a}{K_2} Br'$ bits transmitted through the channel. Hence, there is an increase of $\left(\frac{r' N_a}{K_2} - \frac{r N_a}{K_1}\right) B$ bits over the standard concatenation.
scheme, where \( r' \) can be freely chosen but is usually equal to \( r \). The value of \( r' \) depends on the details of the ECC, and there are two likely candidates:

- The ECC encoder might consistently add a fixed number of symbols (regardless of the expansion of the number of input symbols), so that \( r' = r \). Then the increase in the number of channel bits is \( r\left(\frac{N_2}{K_2} - \frac{N_1}{K_1}\right)B \).

- The ECC encoder might maintain a fixed coding rate, so that the transmitter will need to expand the number of parity symbols proportionally, so \( r' = \frac{N_1}{K_1}r \). Then the increase in the number of channel bits is

\[
r\left(\frac{N_2}{K_2} - 1\right) \frac{N_1}{K_1}B.
\]

When the rate of the ECC is high (so that \( r/n \) is small), the cost of this modified concatenation scheme is slight since the increase in redundancy that applies only to the parity bits and not to the message bits of the ECC codeword. The resulting overhead of modified concatenation is only slightly larger than with standard concatenation using modulation code \( C_1 \), and is much smaller than standard concatenation using code \( C_2 \).

This modified concatenation scheme provides a practical approach for reducing the error-propagation associated with ECC decoders with constrained coding. In short, modified concatenation allows a system to have the high code rate of \( C_1 \) as well as the desirable properties of code \( C_2 \). Table 2.1 gives a comparison of the different configurations.

It should be mentioned that when alternating between a block of modulated message bits and a block of modulated parity bits, it may be necessary to insert a few bits in between these blocks to maintain the modulation constraint. This number is usually not significant so our analysis omits the extra bits.

**Codeword expansion**

The modified concatenation technique suffers from an expansion of the input to the ECC code, due to the modulation encoding. The message is of length \( m' = \frac{N_1}{K_1}m_0 \) and
CHAPTER 2. CONSTRAINED CODING FOR HARD DECODERS

<table>
<thead>
<tr>
<th>Concatenation scheme</th>
<th>channel bits</th>
<th>soft info. for ECC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard concatenation with $C_1$</td>
<td>$\frac{N_1}{K_1}m_0S + \frac{N_1}{K_1}pS$</td>
<td>No</td>
</tr>
<tr>
<td>Modified concatenation with $C_1, C_2$</td>
<td>$\frac{N_1}{K_1}m_0S + \frac{N_2}{K_2}pS$</td>
<td>Yes</td>
</tr>
<tr>
<td>Standard concatenation with $C_2$</td>
<td>$\frac{N_2}{K_2}m_0S + \frac{N_2}{K_2}pS$</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 2.1: Comparison of concatenation schemes

$n' = \frac{N_1}{K_1}m_0 + r'$, so there is an $\alpha = \frac{N_1}{K_1} > 1$ increase in the input to the ECC decoder and a factor of $\frac{n'}{n} = \frac{\alpha m_0 + r'}{m_0 + r} \approx \alpha$ increase in the codeword length, as compared with standard concatenation using code $C_1$.

Increasing the codeword length results in an increased complexity of ECC decoding. In addition, for some ECCs, such as Reed-Solomon codes, there may be an upper limit on the codeword length. Finally, for a long burst of errors on the channel, the demodulation step actually has the beneficial effect of shrinking the error, so that the effect of long bursts on modified concatenation is increased by a factor of $\alpha$ in comparison with standard concatenation. In other words, if there is a burst of $L$ bits, then ignoring edge effects, this burst affects $L$ bits in the decoder in modified concatenation, but only roughly $\frac{K_1}{N_1}L = \frac{1}{\alpha}L$ bits using standard concatenation.

Immink [Imm97] has proposed the use of a block code for lossless compression to counteract this expansion due to modulation. As pictured in Figure 2.6, a block code is used to losslessly compress the modulated message sequence before it is input to the ECC encoder, and also before the ECC decoder, so that the expansion is reduced to $\alpha \frac{K_c}{N_c}$. For a hard-decision ECC (such as a Reed-Solomon code) with symbol size $B$, the block compression code has rate $K_c/N_c$ such that $K_c$ matches the symbol size $B$. All possible constrained sequences of length $N_c$ must be in the code book for this code, so that the rate $K_c/N_c$ of the compression code should be greater than the capacity of the constraint. Section 2.4 describes the general problem of designing lossless compression codes.

It should be noted that this lossless compression technique is not applicable for soft information, since the compression step would only further obscure the reliability information. In general, the compression code is a fairly arbitrary assignment of codewords, so that the computation of post-compression reliability information
from the input reliability information is a difficult task, similar to computing post-
demodulation reliability information for an arbitrary modulation code, as in Section
4.2.

2.3 Analysis of error propagation

In [Bla91] and [ABW95], comparisons are made of the error-propagation properties of
block codes and sliding-block codes. It turns out that block codes perform favorably
when the number of bits matches the Reed-Solomon symbol size, but if the block
modulation codes have long block lengths, then error-propagation results, so that
the sliding-block method is superior. In comparison, as studied in [FC98], the modi-
ﬁed concatenation scheme gives a method that allows the use of modulation codes
with arbitrarily long block lengths without the fear of error-propagation. In this sec-
tion, the error propagation of standard and modiﬁed concatenation are analyzed and
compared.

For magnetic recording, a sector is typically divided into multiple ECC codewords
that are interleaved in such a way that a burst of error symbols becomes separated
into the different codewords, as shown in Figure 2.5. The number of interleaves is $ID$,
the interleave depth. First in §2.3.1, it is assumed that the interleaving is sufﬁcient
to separate bursts of symbol errors, so that the decoders see independent symbol
errors. An analysis of error propagation is given in §2.3.2, and then §2.3.3 considers
the situation where the error bursts exceed the interleave depth, causing more than
one symbol error in a single interleave.

Many approximations are made in this section, such as ones of the form $\Pr (\bigcup A_i) \approx
\sum \Pr (A_i)$. These are accurate in the situation of high signal to noise ratio (SNR),
where each sectors has very few error events. These approximations are intended to
give an intuitive understanding of the primary factors that affect performance.
2.3.1 Decoder performance, with independent errors

Hard-decision ECCs, such as Reed-Solomon decoders, decode the received word to a codeword lying within Hamming distance \( t \), if such a codeword exists. In the usual terminology (ref. [Wic95]), when no codeword exists within distance \( t \), this detectable malfunction is called decoder failure. If the decoder outputs a wrong decision because the received word lies within distance \( t \) of a different codeword, this undetectable event is called decoder error. (For an analysis of the probability of decoder error, refer to [MS86].) The sum of decoder failure rate and the decoder error rate is the probability of having more than \( t \) symbol errors in a received word, which we denote by \( P_{\text{decoder}} \). In addition, the assumption is made that any bit error will cause the entire block to become corrupted during demodulation.

If error bursts are short or are sufficiently dispersed by interleaving, then the Reed-Solomon (RS) decoder sees symbol errors that occur independently. With independent errors, the expected number of errors is binomially distributed, with the probability of having \( a \) errors given by \( \binom{n}{a} p^a (1 - p)^{n-a} \), where \( p \) is the symbol error rate. The probability of an unsuccessful decoding (decoder error or decoder
failure) is given by

$$P_{\text{decoder}}(n, t, p) = \sum_{i=t+1}^{n} \binom{n}{i} p^i (1 - p)^{n-i}$$

(2.1)

for a $t$-error-correcting RS decoder. For small $p$ and large $n$, this summation is dominated by the first term $\binom{n}{t+1} p^t$, and the Poisson approximation to the binomial distribution, $\binom{n}{t+1} p^t \approx e^{-np} \frac{(np)^t}{t!} \approx \frac{(np)^t}{t!}$, gives a rough estimate of performance.

We compare the performance of the two concatenation methods under the assumption that the errors are single bit errors occurring independently with probability $b$. For standard concatenation, an error bit in any of $N_1$ bits in a block can corrupt the block, and the probability of symbol error is equal to the block error probability $P_{\text{symbol}}^{\text{std}} = P_{\text{block}} = N_1 b$. It is assumed that interleaving has dispersed the burst of $K_1 b$ symbol errors that occur when a block is in error.

On the other hand, for modified concatenation, any of the $B$ bits can cause a symbol to be in error, so $P_{\text{symbol}}^{\text{mod}} = B b$. Note that the error in the parity bits are not considered in this analysis, since it is assumed that the code $C_2$ does not cause error propagation. Hence for independent errors, the symbol error probabilities $p = P_{\text{symbol}}^{\text{std}}$ and $p' = P_{\text{symbol}}^{\text{mod}}$ are related by

$$p' \approx \frac{B}{N_1} p.$$ 

(2.2)

Now if there is the same amount of user data per codeword, then it is necessary to increase the code length for modified concatenation to $n' = \frac{N_1}{K_1} (n - 2t) + 2t \approx \frac{N_1}{K_1} n$.

Comparing the schemes using the Poisson approximation gives the following:

$$P_{\text{decoder}}^{\text{mod}} \approx \left( \frac{n'}{t+1} \right) p' \approx \frac{1}{(t+1)!} \left( \frac{N_1}{K_1} n \right) \left( \frac{B}{N_1} p \right)^{t+1}$$

$$\approx \left( \frac{B}{K_1} \right)^{(t+1)} \binom{n}{t+1} p \approx \left( \frac{B}{K_1} \right)^{(t+1)} P_{\text{decoder}}^{\text{std}}$$

(2.3)

This rough calculation indicates that the amount which modified concatenation can be
expected to perform better than standard concatenation depends on the ratio \( B/K_1 \) and the number of symbols errors \( t \) that the Reed-Solomon decoder can correct.

A similar result holds when a lossless compression code is used with modified concatenation. For a lossless compression block code of rate \( K_c/N_c \), the probability of symbol error is \( P_{\text{symbol}}^{\text{mod}} = \frac{N_c}{K_c} Bb = N_c b \). Using the codeword length \( n' \approx an = \frac{K_c}{N_c} K_1 n \)
gives the approximate relationship:

\[
P_{\text{decoder}}^{\text{mod}} \approx \frac{1}{(t + 1)!} \left( \frac{K_c}{N_c} \frac{N_1}{K_1} \frac{N_c}{N_1} \right)^{t+1} \approx \left( \frac{K_c}{K_1} \right)^{t+1} P_{\text{decoder}}^{\text{std}}
\]

If \( K_c = B \), then this expression is the same as equation (2.3).

A more relevant measure of the performance of a system is \( P_{\text{sector}} \), the probability that a sector contains at least one interleave that is not decodable, which depends on the distribution of the errors with respect to the interleaving. In general, this is a complicated expression, which is related to \( P_{\text{decoder}} \) in the following way:

\[
P_{\text{decoder}} \leq P_{\text{sector}} \leq ID \cdot P_{\text{decoder}}
\]

For small interleave depth, it is reasonable to estimate \( P_{\text{sector}} \) using \( P_{\text{decoder}} \), so that for our purposes, it is sufficient to estimate for \( P_{\text{decoder}} \).

### 2.3.2 Error propagation

When demodulating a burst of error bits using a block code of rate \( K/N \), it is possible for a number of block errors to occur. A single bit in error affects one block; two adjacent bits in error have a \( \frac{N-1}{N} \) chance of causing one block error and a \( \frac{1}{N} \) chance of overlapping two blocks. Given non-negative integers \( A \) and \( B \), let \( m(A, B) \) be a modulo function, equal to the unique integer satisfying \( 0 \leq m(A, B) \leq B - 1 \) and \( m(A, B) \equiv A \pmod{B} \). In general, a burst of length \( L \) bits has a probability \( m(L-1,N) \) of causing \( \left\lfloor \frac{L}{N} \right\rfloor + 1 \) block errors and probability \( 1 - m(L-1,N) \) of causing \( \left\lfloor \frac{L}{N} \right\rfloor \) block errors. In this context, the appropriate definition of an error burst is a pattern of bits
in error such that the burst begins and ends with error bits, and there are enough error bits in between to cause a contiguous burst of block errors upon demodulation.

Let \( b_L \) denote the probability of observing a burst of length \( L \), which can be measured by taking the number of bursts of length \( L \) and dividing by the total number of bits. The distribution of bit error lengths is then described by \( \{b_1, b_2, b_3, \ldots\} \), or \( b(z) = \sum_{i=1}^{\infty} b_i z^i \) in polynomial form. Another way to consider the bit error distribution is to observe that the probability of having an error event (of any length) begin at a particular bit is \( b = \sum_i b_i \). Then the conditional probability of having a burst of length \( L \), given that an error event has occurred, is \( \tilde{b}_L = b_L / b \). In Section 2.5, the conditional distribution \( \tilde{b}(z) = \frac{b(z)}{b} \) is assumed to be fixed while the error event rate \( b \) varies.

Suppose that a single error event has occurred while reading a sector. The effect of that error event upon demodulation with a block code of rate \( K/N \) is given by \( p_{\text{block}}(m) \), the probability that \( m \) consecutive blocks in error occur:

\[
\begin{align*}
    p_{\text{block}}^{\text{std}}(1) &= \frac{1}{bN} \left( b_1 N + b_2 (N - 1) + b_3 (N - 2) + \cdots + b_N \right) \\
    p_{\text{block}}^{\text{std}}(m) &= \frac{1}{bN} \left( \sum_{i=2}^{N} b_{(m-2)N+i}(i - 1) + \sum_{i=1}^{N} b_{(m-1)N+i}(N - i + 1) \right)
\end{align*}
\]

This can be rewritten as the compact expression

\[
    p_{\text{block}}^{\text{std}}(m) = (\Delta_N(z) \tilde{b}(z))_{(1+(m-1)N)}
\]

for \( m \geq 1 \), where \( \tilde{b}(z) = \frac{1}{b} b(z) = (\sum_i b_i)^{-1} \sum_i b_i z^i \), and the function \( \Delta_N(z) \) is a triangular function

\[
    \Delta_N(z) = \frac{1}{N} \left( z^{-(N-1)} + 2z^{-(N-2)} + \cdots + (N-1)z^{-1} + N + (N-1)z + \cdots + z^{(N-1)} \right).
\]

The subscript \( 1+(m-1)N \) in (2.4) indicates the coefficient of \( z^{1+(m-1)N} \) in the product \( \Delta_N(z)b(z) \).

This expression (2.4) describes the error propagation from single bits to blocks of size \( N \). In general, it is possible to consider the effect of error-propagation from blocks
of size $A$ to blocks of size $B$. If $A$ and $B$ are relatively prime, then it is assumed that the $A$-block boundaries are randomly and uniformly distributed with respect to the $B$-block boundaries. Let $p_A(z)$ be a distribution of errors in terms of number of $A$-blocks. In terms of bits, the distribution is $b(z) = p_A(z^A)$. Then $p_B(z) = \sum_m p_B(m)$, where

$$p_B(m) = (\Delta_B(z)p_A(z^A))_{(1+(m-1)B)}.$$  

Now if $G = \gcd(A, B)$ is greater than 1, then it can be assumed that the block boundaries have been aligned to match as often as possible (to minimize the error propagation). In that case, the appropriate expression becomes:

$$p_B(m) = (\Delta_{(B/G)}(z)p_A(z^{(A/G)}))_{(1+(m-1)(B/G))}$$

For example, if $A = 2B$, then $G = B$, so that $p_B(m) = (p_A(z^2))_m$, and $p_B(z) = p_A(z^2)$.

**Standard Concatenation**

As derived above, for standard concatenation with code $C_1$ of rate $K_1/N_1$, the distribution of block errors resulting from a single error event is given by

$$p_{\text{block}}^{\text{std}}(m) = (\Delta_{N_1}(z)b(z))_{(1+(m-1)N_1)}.$$  

A block error is assumed to cause $K_1/B$ symbol errors, so that if $K_1$ is a multiple of $B$, the probability that a symbol is the start of a burst of $m$ consecutive symbol errors is

$$p_{\text{symbol}}^{\text{std}}(m) = \begin{cases} p_{\text{block}}^{\text{std}} \left( \frac{m}{(K_1/B)} \right) & \text{if } m \text{ is a multiple of } K_1/B \\ 0 & \text{otherwise} \end{cases} \quad (2.5)$$

which can be rewritten as

$$p_{\text{symbol}}^{\text{std}}(z) = p_{\text{block}}^{\text{std}} (z^{K_1/B}).$$

To generalize to the case where $K_1$ is not a multiple of $B$, let $g = \gcd(B, K_1)$, which gives
\[ p_{\text{symbol}}^{\text{std}}(m) = (\Delta_{(B/g)}(z)p_{\text{block}}^{\text{std}}(z^{(K_1/g)}))_{(1+(m-1)(B/g))} \] (2.6)
where it is assumed that the block boundaries are aligned in a way as to minimize the error propagation. In other words, the block boundaries and symbol boundaries coincide every \( \frac{BK_1}{g} \) bits.

**Modified Concatenation**

Only the error propagation in the message is considered, since it is assumed that the fraction of parity bits is small, and there is limited error propagation because of the code \( C_2 \) (where \( K_2 = B \)) that modulates parity bits. If a lossless compression code of rate \( K_c/N_c \) is used (where \( K_c = B \)), then the symbol error distribution in terms of \( b(z) \) is given by:

\[ p_{\text{symbol}}^{\text{mod}}(m) = (\Delta_{N_c}(z)b(z))_{(1+(m-1)N_c)} \quad (2.7) \]

\[ = \frac{1}{bN_c} \left( \sum_{i=2}^{N_c} b_{(m-2)N_c+i}(i-1) + \sum_{i=1}^{N_c} b_{(m-1)N_c+i}(N_c - i + 1) \right) \]

If no lossless compression step is used, then simply let \( N_c = B \).

### 2.3.3 Decoder performance, with burst errors

The Reed-Solomon codewords are usually interleaved to reduce the possibility that a single burst error leads to multiple symbol errors in the same codeword. Let us assume that the interleave depth is \( ID \), and consider the effect of a single burst of \( L \) symbols on the first interleave. The burst of \( L \) symbols has probability \( \frac{m(L,ID)}{ID} \) of causing \( \lfloor \frac{L}{ID} \rfloor \) error(s) in an interleave and probability \( 1 - \frac{m(L,ID)}{ID} \) of causing \( \lfloor \frac{L}{ID} \rfloor - 1 \) error(s). Hence, if \( p(z) = \sum_{m=1} p(m)z^m \) gives the distribution of the symbol burst length due to a single error event, then the probability of a burst of length \( m \) in the first interleave can be evaluated as

\[ q(m) = (\Delta_{ID}(z)p(z))_{mID}, \quad (2.8) \]
CHAPTER 2. CONstrained CODING FOR HARD DECODERS

using the same triangular function as before. Note that the average symbol error rate stays the same ($\sum mq(m) = \sum mp(m)$).

In particular, the probabilities of single errors and double errors in a single interleave are

$$q(1) = \frac{1}{ID} (p(1) + 2p(2) + \cdots + ID \cdot p(ID) + \cdots + p(2ID - 1))$$

$$q(2) = \frac{1}{ID} (p(ID + 1) + 2p(ID + 2) + \cdots + ID \cdot p(2ID) + \cdots + p(3ID - 1))$$

and so on. Now recall that $\{q(m)\}$ represents the probability distribution resulting from a single error event. Let $q(0) = 1 - \sum_{i=1}^t q(i)$, and consider the distribution given by $q(z) = \sum_{i=0}^t q(i)z^i$. Then in the case of two error events, the probability distribution for the number of errors in the first interleave is $q(z)^2$. For $e$ error events, it is $q(z)^e$.

Since the Reed-Solomon code can correct $t$ errors in an interleave, the decoder fails to decode when there are $t+1$ or more errors. Define an operator on polynomials $f(x)$, which sums up the coefficients that are larger than $t$:

$$S_t : \mathbb{R}[z] \to \mathbb{R}$$

$$S_t : f(z) \to \sum_{i=t+1}^{\text{deg}(f)} f_i$$

Then the probability of not decoding correctly is

$$P_{\text{decoder}} = \sum_{e=1}^t \Pr(e \text{ error events}) \Pr(\text{undecodable} \mid e \text{ error events})$$

$$= \sum_{e=1}^t \left( \begin{array}{c} N_{\text{sector}} \\ e \end{array} \right)_b S_t(q(z)^e)$$

$$= \sum_{e=1}^t \left( \begin{array}{c} N_{\text{sector}} \\ e \end{array} \right) b^e (1 - b)^{N-e} \left( \sum_{j=t+1} \left( q(z)^e \right)_j \right)$$

Then it is possible to evaluate $P_{\text{decoder}}$ by conditioning on the number of error events in a sector. The probability of $e$ error events is given by $\left( \begin{array}{c} N_{\text{sector}} \\ e \end{array} \right)_b$ where $b = \sum_{i=1} b_i$.
is the event error rate, and \( N_{\text{sector}} \) is the total number of recorded bits for a sector. Note that \( N_{\text{sector}} \) is given by

\[
N_{\text{sector}}^{\text{std}} = \left[ \frac{ID \cdot (M + 2t) \cdot B}{K_1} \right] N_1
\]

\[
N_{\text{sector}}^{\text{mod}} = \left[ \frac{ID \cdot M \cdot B}{K_1} \right] N_1 + ID \cdot 2t \cdot N_2
\]

for standard and modified concatenation.

The performance can be estimated from the burst error distribution \( b(z) \). The symbol error rates can be found using (2.6) and (2.7), from which the post-interleaving symbol error rates can be found using (2.9). Then expression (2.10) calculates the decoder error-failure rates for both concatenation schemes. In this way, it is possible to make an fair approximate evaluation of the effects of error propagation.

It should be mentioned that an alternative approach is given by Weldon [Wel92], in which the performance of the Reed-Solomon decoder is evaluated in terms of the probabilities of single and double symbol errors. In comparison with Weldon’s approach, our approach simplifies the calculations by using an approximation based on conditioning on the number of error events per sector, which is appropriate in the situations of high SNR where most sectors will have very few error events.

### 2.4 Lossless compression

This section studies the design of lossless compression codes, which are used to improve the performance of modified concatenation for hard-decision ECCs. Examples of block compression codes for modified concatenation were given in [Imm97] and [FC98]. A construction of sliding-block lossless compression codes is presented, which has advantages over block codes. This section represents joint work with Brian Marcus and Ron Roth [FMR99][FMR00].
2.4.1 Lossless compression for modified concatenation

While constrained coding involves encoding unconstrained sequences as constrained sequences, lossless data compression takes constrained sequences and encodes them into unconstrained sequences in such a way so that no distortion occurs upon decompression. In this sense, lossless data compression is the reverse of constrained coding. This duality between constrained coding and lossless data compression has been observed by several authors [Ari90], [Ker91], [MLT83], [TLM83]), who have applied data compression techniques such as arithmetic coding to constrained coding.

![Diagram of modified concatenation with lossless compression](image)

Figure 2.6: Modified concatenation with lossless compression

The new construction for lossless data compression can be incorporated in the modified concatenation scheme, as shown in Figure 2.6. The output \( w \) of the encoded \( C_1 \) is longer than the input \( u \), so that it may be necessary to increase the number of parity symbols \( r \) for the error-correcting code to achieve the same level of error protection. In addition, for long bursts, the effect of a burst of channel errors is magnified relative to the standard concatenation scheme, since the bursts are not first demodulated. Immink's solution to this problem in [Imm97] is to compress the sequence \( w \) in a lossless manner into a sequence \( s \), and then compute the sequence of parity symbols \( r \) based on \( s \). For instance, \( s \) could be the message and \( r \) could be the parity portions of a Reed-Solomon codeword. The parity sequence \( r \), and therefore the modulated parity sequence \( y \), can be made shorter as a consequence, lowering the overhead of the error-correction scheme and reducing the complexity of decoding. At the channel output, the receiver obtains the possibly incorrect words \( \hat{w} \) and \( \hat{y} \). The
 CHAPTER 2. CONstrained CODING FOR HARD DEcodERS

received message sequence \( \hat{w} \) is compressed to a sequence \( \hat{s} \), and the ECC decoder uses \( \hat{r} \) and \( \hat{s} \) to construct a corrected version \( \hat{\bar{s}} \). Then the decompressor recovers \( \hat{w} \), and the constrained decoder \( C_1 \) recovers \( \bar{u} \).

As an extreme form of lossless compression, one could compress \( w \) back to \( u \) (in which case \( s \) would be the same as \( u \)). This makes the system equivalent to standard concatenation, since lossless compression is equivalent to the demodulation step. A small channel error in \( \hat{w} \) could corrupt all of \( \hat{s} \) before error correction, thereby undermining the benefits of modified concatenation. A good compression scheme will have limited error propagation (for example, by using a block code of rate \( K_c/N_c \), where \( K_c \) equals the symbol size \( B \)), so that a channel error affects a limited number of bits symbols in \( s \).

To understand the duality between modulation coding and lossless compression, consider the case of a block modulation code, with an encoder that takes \( K \) bit to \( N \) bits, such that the output bitstream satisfies the constraints. In other words, the block code corresponds to a 1-1 map

\[
\varphi : \{0,1\}^K \rightarrow \{0,1\}^N
\]

where any combination of words in the image of \( \varphi \) satisfies the modulation constraint. Let \( \text{Im} \varphi \) be the image of the mapping \( \varphi \). Then the set of all possible sequences created from putting such blocks together is denoted by

\[
(\text{Im} \varphi)^\infty = \{(\ldots, w_{-1}, w_0, w_1, \ldots) \mid \, w_i \in \text{Im} \varphi \subset \{0,1\}^N\}.
\]

Then any bi-infinite sequence of bits consisting of blocks of length \( N \) from the image of \( \varphi \) will satisfy the modulation constraint. If \( S \) denotes the constrained system, consisting of all possible sequences satisfying the constraint, then

\[
(\text{Im} \varphi)^\infty \subset S. \tag{2.11}
\]

Of necessity, the modulation rate \( K/N \) is less than or equal to the capacity of the constraint \( \text{Cap}(S) \).
CHAPTER 2. CONSTRAINED CODING FOR HARD DECODERS

On the other hand, a block code for lossless compression code is described by a 1-1 map known as an \textit{excoder} (in analogy to “encoder”),

\[ \psi : \{0, 1\}^{K_c} \rightarrow \{0, 1\}^{N_c} \]

such that the image covers the set of constrained words of length \(N_c\). In other words, the map \(\psi\) should satisfy:

\[ (\text{Im} \psi)\infty \supset S \]  \hspace{1cm} (2.12)

Then the map \(\psi^{-1}\) is defined on \(S_{N_c}\), the set of all constrained sequences of length \(N_c\), and gives a lossless compression map corresponding to a block code of rate \(K_c/N_c\), which is a bijection from \(\text{Im} \psi\) onto \(\{0, 1\}^{K_c}\), and any sequence in \(S\) can be divided into blocks of length \(N_c\) and then compressed using \(\psi^{-1}\). In the next subsection, it is shown that \(K_c/N_c \geq \text{Cap}(S)\). Comparing (2.11) and (2.12), the duality between modulation coding and lossless compression is evident. It turns out that this duality extends to the application of the state-splitting algorithm for constructing sliding-block codes, as explored in §2.4.3.

It may be appropriate to develop block compression codes in conjunction with the modulation code. In that case, it is sufficient that the map \(\psi\) merely cover the sequences generated by the modulation map \(\varphi\) (as opposed to the entire constrained system \(S\)), so that the condition becomes

\[ (\text{Im} \psi)\infty \supset (\text{Im} \varphi)\infty. \]

The co-design of modulation codes and lossless compression codes poses an interesting challenge.

2.4.2 Block codes for lossless compression

A lossless block code for compression code of rate \(K_c : N_c\) maps constrained words of length \(N_c\), called \(N_c\)-codewords, to unconstrained binary words of length \(K_c\), called \(K_c\)-frames, in a 1-1 fashion. A necessary and sufficient condition for such a code to
exists is

\[ |S_{Nc}| \leq 2^{Kc}, \]

(2.13)

where \( S_{Nc} \) denotes the set of constrained words of length \( Nc \) for a given constraint \( S \).

Two examples are given in [Imm97]: a rate 8:11 block code for the (1,12)-RLL constraint and a rate 8:13 block code for the (2,15)-RLL constraint. For \( Kc = 8 \), these values of \( Nc \) are optimal, as condition (2.13) would be violated for any rate 8:12 block code for the (1,12)-RLL constraint and any rate 8:14 block code for the (2,15)-RLL constraint.

Clearly, \( Kc = 8 \) is a good choice owing to the availability of high performance, high efficiency, off-the-shelf Reed-Solomon codes with \( B = 8 \) bits per symbol. Allowing other values of \( Kc \) can give added flexibility in the choice of compression schemes (provided that \( Kc \) and the symbol size \( B \) of the ECC are somewhat compatible). It is desirable to have a low compression rate \( Kc/Nc \), which can be achieved by larger block lengths. The capacity of the constraint imposes a lower bound on the compression rates.

Recall that the capacity, \( \text{cap}(S) \), of a constraint \( S \) is defined by

\[ \text{cap}(S) = \lim_{n \to \infty} \frac{1}{n} \log_2 |S_n|, \]

(2.14)

Since \( |S_{Ncm}| \leq |S_{Nc}|^m \) for any choice of positive integers \( Nc \) and \( m \), it follows that

\[ \text{cap}(S) = \lim_{m \to \infty} \frac{1}{Ncm} \cdot \log |S_{Ncm}| \leq \frac{1}{Nc} \cdot \log |S_{Nc}|. \]

Combining this with (2.13) yields

\[ \text{cap}(S) \leq \frac{1}{Nc} \cdot \log |S_{Nc}| \leq Kc/Nc. \]

(2.15)

Thus, to obtain compression rates \( Kc/Nc \) close to capacity, it is necessary take \( Nc \) (and hence \( Kc \)) sufficiently large so that \( \frac{1}{Nc} \cdot \log |S_{Nc}| \) is close enough to capacity. As a result, the schemes can be rather complex. Moreover, if the typical burst error length is short relative to \( Nc \), then the compression code may actually expand the burst.
Finally, for a burst error comparable to \( N_c \) bits, it may be aligned so as to affect two or more consecutive \( N_c \)-codewords, and therefore two or more consecutive \( K_c \)-frames. This "edge-effect" can counteract the benefits of using compression codes.

### 2.4.3 Sliding-block compressible codes

A more general class of lossless compression codes is given by a sliding-block construction. Such a code consists of a compressor and a decompressor, which can be also called an encoder (short for expanding coder). The compressor is a sliding-block decoder from sequences of \( N_c \)-codewords of the constrained system \( S \) to unconstrained sequences of \( K_c \)-frames. An \( N_c \)-codeword (in \( w \)) is compressed into a \( K_c \)-frame (in \( s \)) as a time-invariant function of the current codeword, and perhaps some \( m \) preceding and a upcoming \( N_c \)-codewords. The sliding-block window length is defined as the sum \( m + a + 1 \). The encoder, on the other hand, has the form of a finite-state machine. Recall Theorem 1 in Section 2.1 for constructing sliding-block modulation codes, which is proved using the state-splitting (ACH) algorithm. There exists an analogous result for constructing sliding-block lossless compression codes, which also uses a state-splitting algorithm.

**Theorem 2** (Fan, Marcus, Roth [FMR00]) Let \( S \) be a finite memory constrained system with \( K_c/N_c \geq \text{cap}(S) \). Then there is a rate \( K_c : N_c \) finite-state encoder for \( S \) that is sliding-block compressible.

While the state-splitting algorithm for constrained codes is guided by an "approximate eigenvector" that satisfies a certain inequality (of the form \( Ax \leq nx \)), this construction for compression codes is guided by similar kind of approximate eigenvector that satisfies a reversed inequality (of the form \( Ax \geq nx \)).

An \((m, a)\)-sliding-block compressible code is defined by a mapping from the set of constrained sequences of length \( N_c \) \((m + a + 1)\) to binary words of length \( K_c \):

\[
\phi : S_{N_c(m+a+1)} \rightarrow \{0, 1\}^{K_c}
\]

\[
\phi : w_{-m}w_{-m+1} \cdots w_0w_1 \cdots w_a \rightarrow s_0
\]
On the other hand, an encoder for decompression is defined by a finite state diagram with edges labeled by inputs consisting of \( K_c \)-frames \( (s_i) \) and outputs consisting of \( N_c \)-codewords \( (w_i) \). As a trivial example, a block encoder is defined by a finite state diagram with a single state, and sliding-block compressor has \( m = a = 0 \). A precise definition of encoders and sliding-block compression is presented in [FMR00].

2.4.4 Application to burst correction

When a lossless compression code is used in modified concatenation, there are a number of factors that may affect the choice of a compressor-encoder pair, such as the complexity of the compression and expansion, and the error-propagation associated with the application of the compression on the receiving end. In particular, we consider how compressors handle raw channel bursts, and their suitability for use with a symbol-based ECC.

The compressor is applied to the channel bit sequence right after the channel, so that a benefit of the compression is that the length of a raw channel burst will be roughly decreased by the compression factor \( K_c/N_c \), when the length of the burst is long (relative to \( N_c \)). On the other hand, edge effects due to the compressor can expand the error length, and this error-propagation dominates for short bursts. Moreover, for sliding-block compression codes, the sliding-block window length will also extend the burst. Ultimately, the choice of a compression code involves a balance of these four factors:

1. compression rate \( K_c : N_c \)

2. edge effects (how many extra \( K_c \)-frames are affected by the phasing of a burst)

3. effect of the sliding-block window length \( m + a + 1 \) (how many extra \( K_c \)-frames are affected by each error)

4. compatibility between the frame length \( K_c \) and the symbol alphabet \( B \) of the ECC.
In this case, consider a channel burst of length $L$ bits. The length of burst in a sequence over a given symbol alphabet is the number of symbols between (and including) the first and last erroneous symbols. A simplified model will be used, in which any error in the $N_c$-codeword will result in an entirely erroneous $K_c$-frame upon compression, although in practice it might be possible to mitigate this effect through careful design of the compressor.

The maximum number of $N_c$-codewords (including the edge effect) that can be affected by a channel burst of length $L$ bits is either $\left\lceil \frac{(L-1)}{N_c} \right\rceil + 1$ or $\left\lceil \frac{(L-1)}{N_c} \right\rceil + 1$, depending on the phasing within an $N_c$-codeword where the channel burst starts. For $(m,a)$-sliding-block compression codes, the effect of the memory and anticipation is to expand the number of affected $K_c$-frames by $m+a$, so that there are a maximum of

$$N(L) = \left\lceil \frac{(L-1)}{N_c} \right\rceil + m + a + 1$$  

(2.16)

affected $K_c$-frames due to a burst of $L$ bit errors.

Next, the erroneous $K_c$-frames result in a number of symbol errors for the ECC. The sequence of $K_c$-frames is regarded as a long bitstream and sub-divided into non-overlapping blocks of length $B$ bits, corresponding to the symbols of the ECC. It is assumed that the system has been designed so that the boundaries between $K_c$-frames align with the boundaries between ECC symbols as often as possible, i.e. every $(K_cB)/\gcd(K_c,B)$ bits. Then the maximum number of ECC symbols that are in error due to a channel burst of length $L$ bits can be calculated as follows.

Consider a basic unit of size $\gcd(p,B)$ bits, so that there is a burst of length $(NK_c)/\gcd(K_c,B)$ units. Each symbol is a block consisting of $B/\gcd(p,B)$ units. Then similar to (2.16), the maximum number of symbols in error is given by

$$D(L, B) = \left\lceil \frac{(NK_c)/\gcd(K_c,B) - 1}{B/\gcd(K_c,B)} \right\rceil + 1$$

$$= \left\lceil \frac{NK_c - \gcd(K_c,B)}{B} \right\rceil + 1$$
Putting this together with (2.16) yields

\[ D(L, B) = \left\lfloor \frac{((L - 1)/N_c) + m + a + 1)K_c - \gcd(K_c, B)}{B} \right\rfloor + 1. \]

Further analysis and examples of error-propagation are given in [FMR00].

### 2.5 Examples for magnetic recording

In a typical magnetic recording system, the sectors are 512 user bytes, the Reed-Solomon code has symbol size \( B = 8 \), and the bytes are arranged in interleaved codewords, with \( M \approx 512/ID \) user bytes per interleave, where \( M < 2^B - 2t \) due to the restrictions of the Reed-Solomon code design. Hence the interleave degree must satisfy \( ID \geq 3 \). Also, the interleave degree \( ID \) should be large enough to disperse bursts into different interleaves. On the other hand, the performance of the ECC is determined largely by \( t \), so that the amount of redundancy needed is roughly \( 2t \cdot ID \) bytes. To minimize the redundancy, therefore, \( ID \) should be kept as small as possible.

Partial response techniques, which involve the Viterbi algorithm for maximum likelihood sequence detection, are widely applied in commercial hard disk drives. [CDHHS92]. For a receiver system comprising a partial response detector for PR4, where the channel is equalized to a \( 1 - D^2 \) response, the performance of standard and modified concatenation for two codes is compared for two modulation codes. A rate \( \frac{16}{17} \) modulation code shows that with a Reed-Solomon code with 8-bit symbols, modified concatenation permits the use of three interleaves per sector, whereas standard concatenation requires four interleaves for good performance. An example of DC-free block code of length 40 illustrates the use of lossless compression. While these small examples illustrate the flexibility and improved performance allowed by modified concatenation, it should be noted that the advantages of modified concatenation increase with block length of the modulation code. The ability to use longer block lengths with modified concatenation enables the use of higher rate and more sophisticated modulation codes. [TIH95][KSS99].
2.5.1 A rate 16/17 code for PR4

As an improvement to a rate \( K/N = 8/9 \) modulation code for a \((0, G/I) = (0, 4/4)\) constraint, consider a code of rate 16/17 that satisfies a slightly weaker constraint that is still sufficient for the channel. One simple way to do this is to alternate uncoded bytes with modulation blocks from a rate 8/9 code, and there will be no error propagation upon demodulation, since \( K = B = 8 \). For a general code for rate 16/17, however, error propagation occurs where a small error burst (that lies across block boundaries) can result in 4 bytes in error. (It has been shown [W196] that there exist constructions can provide rate 16/17 codes with some \((0, G/I)\) constraints without causing error propagation, but in this example, which applies to much longer block lengths as well, the code is assumed to not be specially designed.) It will be seen that modified concatenation then can reduce the error propagation from four bytes to two bytes.

For standard concatenation with \( ID = 3 \), the interleaves have 170, 170 and 172 user bytes each (where \( M \) is chosen to be even in order to accommodate the modulation code, and \( K_1 = 2B \)). Suppose the error-correction capability of the ECC is \( t = 3 \) errors per interleave. Then the length of the RS code is \( n = k + 2t \) for standard concatenation, so the total number of bytes is \( 178 + 176 + 176 = 530 \). Then there are a total of \( 530 \cdot B \cdot \frac{N_1}{K_1} \) channel bits per sector. For \( ID = 4 \) interleaves per sector, there are \( M = 128 \) user bytes per interleave, so for standard concatenation, the codeword length is \( n = 128 + 6 = 134 \), corresponding to \( ID \cdot n \cdot B \cdot \frac{N_1}{K_1} \) channel bits. Note that this requires an additional \( 2t \) parity symbols per sector, along with an additional use of the Reed-Solomon decoder.

In addition to the rate \( K_1/N_1 = 16/17 \) modulation code, consider another code with rate \( K_2/N_2 = 8/9 \). Since the rate is close to 1, it is not necessary to use a lossless compression step, so \( K_c = N_c = B \). For modified concatenation, all the user bytes are modulated first, followed by the parity, giving a total of \( k' = \frac{1}{B \cdot ID} \left( \left\lceil \frac{ID \cdot M \cdot B}{K_1} \right\rceil N_1 \right) \) input bytes per interleave, for a total of \( \left\lceil \frac{ID \cdot M \cdot B}{K_1} \right\rceil N_1 + ID \cdot 2t \cdot N_2 \) channel bits. For \( ID = 3 \), there are \( k' = 181 \) for 2 interleaves and 182 for one interleave, and for \( ID = 4 \), there are \( k' = 136 \) bytes. The rates listed in the table 2.2 are given by
<table>
<thead>
<tr>
<th>16/17 code</th>
<th>RS input</th>
<th>RS output</th>
<th>Bits/Sector</th>
<th>Rate</th>
<th>% of Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Std $ID = 3$</td>
<td>$k = 170, 172$</td>
<td>$n = 176, 178$</td>
<td>4505</td>
<td>0.9092</td>
<td>100%</td>
</tr>
<tr>
<td>Mod $ID = 3$</td>
<td>$k' = 181, 182$</td>
<td>$n' = 187, 188$</td>
<td>4514</td>
<td>0.9074</td>
<td>99.8%</td>
</tr>
<tr>
<td>Std $ID = 4$</td>
<td>$k = 128$</td>
<td>$n = 134$</td>
<td>4556</td>
<td>0.8990</td>
<td>98.9%</td>
</tr>
<tr>
<td>Mod $ID = 4$</td>
<td>$k' = 136$</td>
<td>$n' = 142$</td>
<td>4568</td>
<td>0.8967</td>
<td>98.6%</td>
</tr>
</tbody>
</table>

Table 2.2: Comparison of standard and modified concatenation for rate 16/17 modulation code

\[
\frac{\# \text{ user bits}}{\# \text{ channel bits}} \text{ and includes both the modulation and ECC. The maximum rate possible under these assumptions is } \frac{16}{17} \cdot \frac{512}{512+18} = 0.9092.
\]

Using the earlier analysis and formulas, it is possible to compare the modified concatenation and standard concatenation schemes using only the distribution of the bit errors out of the detector. It is assumed that for an error event on a $1 - D$ channel, the distribution of the bit lengths is given by

\[
\frac{1}{2}z^2 + \frac{1}{4}z^3 + \frac{1}{8}z^4 + \cdots + \frac{1}{32}z^6 + \frac{1}{32}z^7
\]

where it is assumed that the error length is limited due to the modulation code. The PR4 channel is given by the bit-wise interleaving of two $1 - D$ channels, so the distribution of bit error lengths is roughly

\[
b(z) = b \cdot \left(\frac{1}{2}z^3 + \frac{1}{4}z^5 + \frac{1}{8}z^7 + \cdots + \frac{1}{32}z^{11} + \frac{1}{32}z^{13}\right),
\]

where $b$ is the error event rate. The actual error distribution will depend on the channel characteristics and the choice of modulation code. Due to the precoding used for the PR4 channel, each error event often shows up as two bit errors with many non-errors in between, so $b$ is roughly half the usual bit error rate.

Given this model of the burst error distribution, Figure 2.7 plots the performance of the decoder as a function of the event error rate, and compares standard and modified concatenation for $ID = 3$ and $ID = 4$. There is a close match between the predicted performance (the solid lines) and semi-analytical simulation results (the crosses and circles), indicating that the formulas derived in the previous section provide an accurate measure of the performance. From this figure, it can be concluded that standard concatenation requires 4 interleaves for satisfactory performance, while
modified concatenation requires only 3 interleaves. In this situation, modified concatenation always outperforms standard concatenation.

Next, a hypothetical situation is considered in which the errors on the channel are exactly of length $L$, with probability $b_L = 1$, so that $b(z) = b \cdot z^L$. Figure 2.8 plots the performance as a function of the length $L$, where the error event rate is fixed at $b = 10^{-5}$. The sudden upward jumps correspond to the loss in performance as it becomes possible that a single error event causes two or more errors in a single interleave.

### 2.5.2 A DC-free code for PR4

For partial response channels, matched-spectral null (MSN) codes can provide significant coding gain [KS91]. For the $1 - D$ channel, DC-free codes, which are balanced in the number of ones and zeros, are matched to the spectrum and provide approximately $3\, \text{dB}$ of coding gain. The PR4 channel can be treated as two completely
Figure 2.8: A comparison of the effects of long bursts for the 16/17 code

independent $1 - D$ channels. One method of satisfying these DC-free constraints efficiently is to use block modulation codes with long length. The design and use of DC-constrained block codes is discussed in [Imm91]. It should be noted that our analysis of error propagation assumes a simple detector, which does not depend on the modulation code. This analysis does not accurately describe the situation of detection when using a time-varying trellis [Sol97a][Sol97b] or of post-processing schemes [KWM93] for DC-free block codes, but it should be possible in those cases to draw the similar general conclusions regarding modified concatenation.

Consider a hypothetical modulation constraint consisting of 10-bit blocks which are DC-free. There are $\binom{10}{5} = 252$ DC-free words of length 10, and concatenating these words provides a theoretical data rate of $\frac{1}{10} \log_2 252 = 0.7977$ bits per symbol. The encoder is restricted to process one 10-bit DC-free word at a time, the data rate would be $\frac{1}{10} \left[ \log_2 252 \right] = 0.7$. To increase this rate, Gelblum and Calderbank [GC97] describe a generalized cross-constellation method, giving an example in which four of these 10-bit words are grouped together to obtain a rate $31/40 = 0.775$ code, utilizing
240 of the 252 available codewords. This code serves as $C_1$, with $K_1/N_1 = 31/40$. This construction allows for a simple lossless compression scheme, where each of the 252 possible 10-bit DC-free words is mapped to a different 8-bit word (leaving four 8-bit words unassigned), so that $K_c/N_c = 8/10$.

To complete the construction for modified concatenation, it is necessary to select a modulation code for the parity bits, where $K_2 = B = 8$. A code with balanced DC content is desired, so since $\log_2 \binom{11}{6} > 8$, one possible choice is a code $C_2$ of rate $K_2/N_2 = 8/11$, where a word with 6 ones and 5 zeros alternates with a word with 5 ones and 6 zeros. The capacity of this constraint is $\frac{1}{11} \log_2 \binom{11}{6} = 0.8047$, which is just larger than $\frac{1}{10} \log_2 \binom{10}{5} = 0.7977$. It is reasonable to assume that for channels where a DC-free constraint over every block of 10 bits is useful, the code $C_2$ will yield very similar performance. For simplicity, it is assumed that modulating with $C_1$ and with $C_2$ give identical performances, resulting in the same raw bit error rate on the channel.

These are the alternative configurations for this channel:

- standard concatenation using code $C_2$, with rate $K_2/N_2 = 8/11 = 0.7272$.
- standard concatenation using code $C_1$, with rate $K_1/N_1 = 31/40 = 0.775$. (Error propagation occurs.)
- modified concatenation using code $C_1$ on the message portion, and $C_2$ on the parity portion, along with a lossless compression code of rate $K_c/N_c = 8/10$.

Suppose that $ID = 4$, so the number of user bytes per interleave is $M = 128$. For standard concatenation, the number of channel bits is $\left\lceil \frac{ID \cdot M \cdot B}{K} \right\rceil N$ for a code of rate $K/N$. Meanwhile, for modified concatenation, there are a total of $\left\lceil \frac{ID \cdot M \cdot B}{K_1} \right\rceil = 133$ modulation blocks of four 10-bit words, which become compressed into 4 bytes. Hence, there are $k' = \frac{1}{B \cdot ID} \left( \left\lceil \frac{ID \cdot M \cdot B}{K_1} \right\rceil N_1 \right) \frac{K_c}{N_c} = 133$ modulated message bytes per interleave, and the number of channel bits is given by

$$\left\lceil \frac{ID \cdot M \cdot B}{K_1} \right\rceil N_1 + ID \cdot 2t \cdot N_2 = 133 \cdot 40 + 264 = 5584.$$
The efficiency is calculated by comparing with the maximum possible rate, which is $(\frac{1}{10} \log_2 252) \frac{128}{134} = 0.762$. These results are summarized in Table 2.3.

Table 2.3: Comparison of standard and modified concatenation for DC-free modulation code

The actual error characteristics may differ depending on the implementation of the DC-free code. In the figure, modified concatenation performs almost exactly the same as standard concatenation with $C_2$, while standard concatenation with $C_1$ suffers greatly due to error propagation. Hence using modified concatenation, it is possible to use the generalized cross constellation code as code $C_1$, increasing the rate from 0.6947 to 0.7335 without any loss in performance.

In summary, this example shows that there is no loss in performance from using modified concatenation, while the reduction in the modulation code overhead can be significant. In addition, this example demonstrates how a lossless compression code can be effective in keeping down the size of the Reed-Solomon code, so that the expansion factor in this case is merely $\alpha = \frac{N_1 K_c}{K_1} = 1.032$, rather than $\frac{N_1}{K_1} = \frac{40}{31} = 1.29$ without the compression code.
Figure 2.9: Performance comparison for DC-free codes
Chapter 3

Soft iterative decoding

This chapter discusses a framework for understanding probabilistic decoding. Section 3.1 defines factor graphs and explains the message-passing algorithm. Section 3.2 uses this framework to discuss the decoding algorithm for low-density parity check (LDPC) codes, and then gives some reduced complexity implementations. Section 3.3 explains the forward-backward algorithm (FBA), also known as the BCJR algorithm or MAP decoder in turbo decoding, and interprets it in terms of the message-passing algorithm. Finally, Section 3.4 makes a complexity and performance comparison of turbo and LDPC codes for magnetic recording.

3.1 Factor graphs and message-passing

The theory of factor graphs and the message-passing algorithm turns out to be well suited for understanding the iterative probabilistic decoding algorithms for turbo codes and low density parity check (LDPC) codes. [For97][Fre98][KFL98]. Factor graphs provide a method for expressing how a global function factors into local functions. Starting with prior probabilities corresponding to observed evidence, the message-passing algorithm provides a computationally efficient method for evaluating the posterior probabilities that incorporate the knowledge of the structure described by the factor graph. Similar representations that serve the same purpose include Tanner graphs [Tan81], Tanner-Wiberg-Loeliger graphs [Wib97], Generalized State
Realizations [For99], and belief propagation in Bayesian networks [Pea97].

### 3.1.1 Definition

A factor graph is a bipartite graph consisting of two types of nodes, variable nodes $x$ and function nodes $f$, connected by undirected edges. To each function node $f$, it is possible to associate a function that is dependent on all the messages from the neighbors of $f$. The variables are assumed to be discrete-valued. In Figure 3.1, the variable nodes are denoted by circles, and function nodes by black squares. In addition, it is useful to introduce state nodes, which are variable nodes that do not directly correspond to an observable quantity, denoted by two concentric circles.

In the message-passing algorithm, also known as the sum-product algorithm, messages are passed both ways along each edge $e$ between a variable node $x$ and a function node $f$. A message is passed for each possible value of the variable node, so that for a variable that has $M$ values, the message along the edge is a real vector of length $M$ that lies in $[0, 1]^M$. In accordance with the extrinsic property, the message coming out of a node along an edge $e$ depends on the messages that were passed into that node on the previous iteration, except for the message that was passed in the opposite direction along edge $e$.

![Factor Graph](image)

**Figure 3.1: Factor graph**
The decoder starts with prior information $P_{\text{prior}}(x)$ on each variable node. The message from variable node $x$ to function node $f$ is denoted by $\mu_{x \rightarrow f}(\xi)$, where $\xi$ ranges over all possible values of the variable $x$, and depends on all the incoming messages from the other function nodes adjacent to $x$. In other words, it depends on $\mu_{x \rightarrow f'}(\xi)$ for $f' \in \text{Nbr}(x) \setminus \{f\}$, where $\text{Nbr}(x)$ is the set of neighboring function nodes of $x$, and $\{f\}$ is excluded from the set. At the start of the algorithm, the decoder has prior probabilities $P_{\text{prior}}(x)$ on each of the variable nodes. Then the message from a variable node to a function node is given by

$$
\mu_{x \rightarrow f}(\xi) = P_{\text{prior}}(x = \xi) \prod_{f' \in \text{Nbr}(x) \setminus \{f\}} \mu_{f' \rightarrow x}(\xi).
$$

This represents the probability that $x = \xi$ given all the information from the neighboring function nodes except for $f$. Similarly, the message $\mu_{f \rightarrow x}(\xi)$ is the message from the function node $f$ to the variable node $x$, based on the incoming messages $\mu_{x' \rightarrow f}(\xi)$ for $x' \in \text{Nbr}(f) \setminus \{x\}$. The message-passing algorithm takes all possible combinations of values for the adjacent variables, and weights accordingly using the function $f$.

$$
\mu_{f \rightarrow x}(\xi) = \sum_{(\xi_1, \xi_2, \ldots, \xi_l)} f(\xi, \xi_1, \xi_2, \ldots, \xi_k) \prod_{i=1}^{l} \mu_{x_i \rightarrow f}(\xi_i)
$$

This represents the probability that $x = \xi$ given all the information about the neighbors of $f$ (except for $x$). The summation is over all possible values of the variables in the set $\text{Nbr}(f) \setminus \{x\} = \{x_1, x_2, \ldots, x_l\}$.

Finally, to obtain the updated values for a variable, the message-passing algorithm takes the product of all messages coming into that node, along with the prior information.

$$
P_{\text{posterior}}(x = \xi) = c \cdot P_{\text{prior}}(x = \xi) \prod_{f \in \text{Nbr}(x)} \mu_{f \rightarrow x}(\xi)
$$

The constant $c$ is a normalizing factor so that the sum $\sum_{\xi} P_{\text{posterior}}(\xi) = 1$. Leaving
out the prior probabilities gives a similar expression for the extrinsic information

\[ P_{\text{extrinsic}}(x = \xi) = c' \cdot \prod_{f \in \text{Nbr}(x)} \mu_{f \rightarrow x}(\xi) \]

for appropriate normalization \( c' \). This completes the message-passing algorithm.

![Diagram of message-passing](image)

**Figure 3.2:** Message-passing of probabilities on a cycle-free graph

It is possible to understand the message-passing rules (3.1) and (3.2) more precisely in terms of probabilities. Suppose that the factor graph is a cycle-free graph, i.e. a tree, as pictured in Figure 3.2. Given initial prior probabilities on each variable node, the goal is to compute updated posterior probabilities based on the relationships described by the graph.

The messages that are propagated on the edge between a variable node \( x \) and a function node \( f \) in Figure 3.2 can be interpreted in terms of probabilities. The message from variable \( x \) to function node \( f \) is proportional to the joint probability of \( x \) and all the information to the left of \( x \), denoted by \( L \). On the other hand, if \( R \) represents the information to the right of the variable \( x \), the message from function node \( f \) to variable \( x \) is proportional to the conditional probability of \( R \) given \( x \). Also,
let \( C \) represent the prior information on the variable \( x \). Then

\[
\mu_{z \to f}(\xi) = c' \cdot \Pr(x = \xi, L, C)
\]

\[
\mu_{f \to z}(\xi) = c \cdot \Pr(R | x = \xi)
\]

for some constants \( c \) and \( c' \).

There information on this graph is assumed to obey a Markovian structure, so that if \( z \) is a node that links two subgraphs with associated information \( A \) and \( B \), then \( A \) and \( B \) are independent conditioned on \( z \). In other words, \( \Pr(A, B | z) = \Pr(A | z) = \Pr(B | z) \), or alternatively \( \Pr(A | z, B) = \Pr(A | z) \). Then \( L_1, L_2, L_3, \) and \( C \) are independent given \( x \). Similarly, \( R_i \) is independent from the rest of the graph given \( x_i \).

The message-passing rules can then be derived by expanding these probability expressions. The message from \( x \) to \( f \) is given by

\[
\Pr(x, L, C) = \Pr(x, C) \Pr(L | x, C)
\]

\[
= \Pr(x, C) \Pr(L_1, L_2, L_3 | x)
\]

\[
= \Pr(x, C) \prod_{i=1}^{3} \Pr(L_i | x)
\]

which corresponds to the expression derived from (3.1).

\[
\mu_{z \to f}(\xi) = P_{\text{prior}}(x = \xi) \prod_{i=1}^{3} \mu_{f \to z}(\xi)
\]

Let \( R_1^3 \) denote \( R_1, R_2 \) and \( R_3 \), and \( x_1^3 \) denote \( x_1, x_2 \) and \( x_3 \). Then

\[
\Pr(R_1^3, x | x_1^3) = \left( \prod_{i=1}^{3} \Pr(R_i | x_i) \right) \Pr(x | x_1^3)
\]

using the Markov property. Now \( R \) is all the information to the right of \( x \), which includes \( R_1, R_2 \) and \( R_3 \), and includes the prior information about \( x_1, x_2 \) and \( x_3 \).
Hence,
\[
\Pr(R, C, x) = \sum_{x_1^3} \Pr(R_1^3, C, x \mid x_1^3) \Pr(x_1^3)
\]
\[
= \sum_{x_1^3} \left( \prod_{i=1}^3 \Pr(R_i \mid x_i) \Pr(x_i) \right) \Pr(x, C \mid x_1^3)
\]
\[
= \sum_{x_1^3} \left( \prod_{i=1}^3 \Pr(R_i, x_i) \right) \Pr(x, C \mid x_1^3)
\]

The right most term can be written as
\[
\Pr(x, C \mid x_1^3) = \Pr(C \mid x) \Pr(x \mid x_1^3)
\]
\[
= c_{\text{prior}}(x) f(x, x_1, x_2, x_3)
\]

where \(\Pr(C \mid x)\) is proportional to \(P_{\text{prior}}(x)\), as indicated by the constant \(c\), and \(f(x, x_1, x_2, x_3)\) is an indicator function, describing whether the inputs are compatible. For example, for the case of an even parity check,
\[
f(x, x_1, x_2, x_3) = \begin{cases} 
1 & \text{if } x \oplus x_1 \oplus x_2 \oplus x_3 = 0 \\
0 & \text{otherwise}
\end{cases}
\]

Then the expression for messages from \(f\) to \(x\) becomes
\[
\Pr(R \mid x) = c \sum_{x_1^3} f(x, x_1, x_2, x_3) \left( \prod_{i=1}^3 \Pr(R_i, x_i) \right)
\]
which matches equation (3.2).
\[
\mu_{f \rightarrow x}(x) = \sum_{x_1^3} f(x, x_1, x_2, x_3) \prod_{i=1}^3 \mu_{x_i \rightarrow f}(x_i)
\]

Note that constants are not important since it is the relative proportion of the messages (e.g. \(\frac{\mu_{f \rightarrow x}(x=1)}{\mu_{f \rightarrow x}(x=0)}\)) that is important.
Finally, the posterior probability on $x$ can also be found by taking the product of the two messages on an edge, since this turns out to be proportional to $P(x \mid L, R, C)$, which is the probability of $x$ including all the information on the graph.

$$
\mu_{x \rightarrow f}(\xi) \mu_{f \rightarrow x}(\xi) = c \cdot \Pr(x = \xi, L, C) \Pr(R \mid x = \xi) = c \cdot \Pr(x = \xi, L, R, C)
$$

where $c$ is some constant. Then normalizing gives the posterior probability

$$
P_{\text{posterior}}(x = \xi) = c' \cdot \Pr(x = \xi, L, R, C)
$$

where $c'$ is a constant such that the sum of the posterior probabilities is 1. Hence, this simple example shows how message-passing on a factor graph relates to the propagations of probabilities on a cycle-free graph.

In this way, the message-passing algorithm can be seen to yield the exact values of the posterior probabilities if there are no cycles (i.e. closed paths) in the factor graph. If the graph does contain cycles, however, the algorithm can be iterated indefinitely and there are few guarantees on convergence, since the presence of cycles invalidates the independence assumptions in the message-passing algorithm. Fortunately, it has been found empirically that for many situations (such as turbo and LDPC codes), the algorithm converges to the desired probabilities despite the presence of cycles. While this approach only yields an approximation of a full maximum-likelihood decoder, the result can be very good in many cases. This method uses the simple rules of Bayesian probability, but structures the computations in an efficient manner for decoding of some otherwise computationally infeasible problems.

The order in which messages are passed is referred to as the message-passing schedule. There can be wide variability in the message-passing schedule. For graphs without cycles, it turns out that convergence does not depend on the message-passing schedule. For graphs with cycles, however, there may be preferred message-passing schedules, which reduce the decoding time and complexity, or improve performance. If messages are passed along all edges at all times simultaneously, this is referred to
as the "flooding schedule". Alternatively, each node can send messages whenever it receives an updated incoming message. When the factor graph is a linear graph, there is a natural schedule, in which a decoder uses a single forward sweep and a single backward sweep of message-passing to compute the posterior probabilities. This is called the forward-backward algorithm (see Section 3.3).

### 3.1.2 Example: Counting soldiers

Pearl [Pea97] gives a simple example that illustrates the mechanics of message-passing. Suppose there is a formation of \( n \) soldiers standing in a row, and each soldier can only communicate with the soldier directly ahead and behind. Message-passing provides a distributed computing method to count the total number of soldiers and transmit this information to all of them. This problem can be solved using the message-passing algorithm on the factor graph pictured in Figure 3.3. Each soldier obeys the following rule: *If you hear a number from one neighbor, add 1 and pass the number to the other neighbor.* The algorithm begins by passing in the number 0 to the two soldiers standing at the ends of the line, and the messages get incremented as they pass along to the left and to the right. Finally, each soldier sums the numbers from the two neighbors to get \( n - 1 \), which corresponds to the extrinsic information. Then the soldier adds 1 to count himself or herself, which corresponds to the prior information, and obtains the posterior information, which is \( n \), the total number of soldiers.

To express the soldier counting problem in terms of the message-passing algorithm, each soldier is represented by a variable node \( x \) and a function node \( f \). The variable \( D \) is a placeholder, where the exponent of \( D \) indicates the number of soldiers. Then each of the variable nodes \( x \) starts off by sending a message \( \mu_{x-f} = D \), corresponding to the prior information that each soldiers counts as 1.

State nodes are used to connect function nodes \( f \) and \( f' \) corresponding to adjacent soldiers. These nodes simply pass the message along, so that the state nodes can be ignored, and message passing occurs directly between function nodes. If \( s \) is the state node between function nodes \( f \) and \( f' \), then a simplification can be made in the
Figure 3.3: Counting soldiers as an example of message-passing

notation as follows

$\mu_{f_i \leftarrow f} \overset{\Delta}{=} \mu_{f_i \leftarrow s} = \mu_{s \leftarrow f}$.

The forward and backward messages are given by

$\mu_{f_i \leftarrow f_{i+1}} = D^{i+1}$

$\mu_{f_i \leftarrow f_{i-1}} = D^{(n-i)}$

The extrinsic information that each soldier receives is that there are $n - 1$ other soldiers in the formation,

$\mu_{f_i \leftarrow z_i} = D^{n-1}$

and the posterior information is $n$, the total number of soldiers.

$\mu_{f_i \leftarrow z_i, x_i \leftarrow f_i} = D^n$

This message-passing algorithm generalizes to soldiers lined up in a tree formation,
as seen in Figure 3.4. On the other hand, it can be seen that this message-passing algorithm fails badly when the soldier stand in a circle. This corresponds to the fact that the message-passing algorithm converge to the exact answer for graphs which are cycle-free, but are not guaranteed to converge for graphs with cycles.

![Diagram showing message-passing in cycle-free and cycle-containing graphs]

Figure 3.4: Message-passing works on graphs without cycles

3.1.3 Combining multiple decoders

An important decoding principle is to use several simple decoding modules to approach the performance of a more complicated decoder. To understand this configuration for multiple decoder modules, represent the code by an single variable node, as in Figure 3.5. Each decoder acts essentially like a function node, and the input (prior information) to a module $f$ corresponds to the message $\mu_{z \rightarrow f}$ from the variable node to a function node. On the other hand, the output (extrinsic information) of a module $f$ corresponds to the message $\mu_{f \rightarrow z}$ from a function node to the variable node, giving the following relation for passing information between multiple decoding modules:
\[ \mu_{z \rightarrow f}(\xi) = \prod_{f' \neq f} \mu_{f' \rightarrow z}(\xi). \]

When the variables are binary, it is convenient to represent the messages as log-likelihood ratios (LLRs), in order to convert multiplications into additions:

\[ \text{LLR}(\mu_{a \rightarrow b}) = \log \frac{\mu_{a \rightarrow b}(1)}{\mu_{a \rightarrow b}(0)}. \]

The relation then becomes

\[ \text{LLR}(\mu_{x \rightarrow f}) = \sum_{f' \neq f} \text{LLR}(\mu_{f' \rightarrow z}), \]

so that in terms of LLRs, the input to each module is the sum of the extrinsic outputs of all the other modules. Finally, the decoder computes the posterior probability as follows

\[ \text{LLR}^{\text{posterior}}(x) = \sum_{f'} \text{LLR}(\mu_{f' \rightarrow z}) \]

in order to obtain the final estimates of the LLRs for all the bits.

Figure 3.5: Iterating amongst multiple decoders
CHAPTER 3. SOFT ITERATIVE DECODING

As in turbo codes (§3.3.3), a code can be designed such that it can be decoded by two decoding modules, which pass soft information between them, as pictured in Figure 3.6. The input to the combined decoder is global prior information, which is passed into both decoders. Each module takes its input as prior information and produces extrinsic information that is passed to the other decoder and added to the global prior information (in terms of log-likelihood ratios). The output of the first module is taken as the input to the second module, and vice versa.

![Diagram of two decoders iterating](image)

The input LLR to a decoder is the sum of the extrinsic LLR of the other decoder and the prior LLR.

Figure 3.6: Iterating between two decoders

3.2 Low-density parity check codes

Low density parity check (LDPC) codes are linear error-correcting codes defined by parity check matrices where the number of non-zero entries is a small proportion of the matrix. These are also known as Gallager codes, in tribute to their inventor [Gal62][Gal63]. These ideas were expanded by Tanner [Tan81], MacKay [Mac99] and others. Gallager [Gal62] introduced regular LDPC codes, which have a parity-check matrix with a uniform number of non-zero entries in each column or row (denoted by $t_c$ and $t_r$, respectively). It turns out that irregular LDPC codes, with parity check matrices that have varying numbers of non-zero entries in each column and row, can yield better performance than the regular constructions if designed properly.
[Mac99][MWD99]. Furthermore, it has been shown that these irregular LDPC codes can perform better than the best known turbo codes, performing extremely close to theoretical limits. [RSU99]. LDPC codes can be defined over various symbol alphabets, as discussed in [DM98], but the focus here is placed on binary LDPC codes.

Binary LDPC codes which are binary parity check matrices with a small number of 1's. Suppose that the parity check matrix $\mathbf{H}$ has $M$ rows by $N$ columns, where $N$ is the length of a codeword, and the location of a 1 in the parity check matrix indicates that a particular bit are involved in a parity check. The codewords correspond to column vectors that satisfy the parity check constraint.

$$\mathbf{Hx} = 0$$

Without loss of generality, the parity-check matrix $\mathbf{H}$ is assumed to have full row rank.

For encoding, it is useful to put $\mathbf{H}$ into systematic form $\begin{bmatrix} P & I_M \end{bmatrix}$, where $I_M$ is the identity matrix of size $M \times M$, and $P$ is of size $M \times (N - M)$. This can be accomplished using Gaussian elimination, and some rearrangement of the columns might be necessary. Then the codeword $\mathbf{x}$ can be divided into message and parity portions $\mathbf{x} = \begin{bmatrix} \mathbf{x}^m \\ \mathbf{x}^p \end{bmatrix}$, so that the systematic encoding is performed as follows

$$\begin{bmatrix} P & I_M \end{bmatrix} \begin{bmatrix} \mathbf{x}^m \\ \mathbf{x}^p \end{bmatrix} = 0$$

$$-P\mathbf{x}^m = \mathbf{x}^p$$

and the generator matrix is given by $\mathbf{G} = \begin{bmatrix} I_{N-M} \\ -P \end{bmatrix}$.

The corresponding factor graph consists of $N$ variable nodes and $M$ function nodes, with an edge between the appropriate variable and function nodes for each 1 in the parity check matrix, as seen in Figure 3.7. Then the message-passing algorithm can be used on this factor graph for decoding the LDPC code. Some issues in the
design of LDPC codes are that the code should have good distance properties and the graphs should not have cycles of short length. Short cycles cause the independence assumptions in the message-passing algorithm to be violated, resulting in suboptimal performance. The message-passing decoder attempts to approach the performance of a maximum likelihood decoder, so that the distance properties of the code strongly determine the performance. For a randomly constructed code, the minimum distance almost always increases linearly with the blocklength [Gal63].

$$
\begin{bmatrix}
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5 \\
x_6 \\
x_7 \\
x_8 \\
x_9 
\end{bmatrix}
=
\begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 
\end{bmatrix}
$$

![LDPC Code Diagram]

Figure 3.7: Low Density Parity Check (LDPC) codes

### 3.2.1 Message-passing (probabilities)

As a preliminary calculation, suppose two bits satisfy a parity check constraint $x_1 \oplus x_2 = 0$, and it is known that $p_1 = P(x_1 = 1)$ and $p_2 = P(x_2 = 1)$. Then the
probability that the check is satisfied is

\[ P(x_1 \oplus x_2 = 0) = (1 - p_1)(1 - p_2) + p_1p_2 \]
\[ = 2p_1p_2 - p_1 - p_2 + 1 \]

which can be rewritten as

\[ 2P(x_1 \oplus x_2 = 0) - 1 = (1 - 2p_1)(1 - 2p_2). \]  

(3.3)

Now suppose that \( L + 1 \) bits satisfy an even parity-check constraint \( x_0 \oplus x_1 \oplus x_2 \oplus \cdots \oplus x_L = 0 \), as pictured in the factor graph in Figure 3.8.

![Parity check factor graph](image)

**Figure 3.8:** \( L \) bits that satisfy an even parity check constraint

Then for known probabilities \( \{p_1, p_2, \ldots, p_L\} \) corresponding to the bits \( \{x_1, x_2, \ldots, x_L\} \), it is possible to generalize (3.3) to find the probability distribution on the binary sum \( z_L = x_1 \oplus x_2 \oplus \cdots \oplus x_L \), where \( z_L = z_{L-1} \oplus x_L \)

\[
2P(z_L = 0) - 1 = (1 - 2P(z_{L-1} = 1))(1 - 2p_L)
\]
\[
= (2P(z_{L-1} = 0) - 1)(1 - 2p_L)
\]

where \( p_L = P(x_L = 1) \). Then applying this recursively yields

\[
2P(z_L = 0) - 1 = \prod_{i=1}^{L} (1 - 2p_i). \]  

(3.4)
Since the extrinsic probability \( p_0 \) for \( x_0 \) is given by

\[
p_0 = P(x_1 \oplus x_2 \oplus \cdots \oplus x_L = 1) = 1 - P(x_1 \oplus x_2 \oplus \cdots \oplus x_L = 0),
\]

where having \( x_0 = 1 \) means that the remaining bits must have odd parity, applying equation (3.4) yields

\[
1 - 2p_0 = 2P(x_1 \oplus x_2 \oplus \cdots \oplus x_L = 0) - 1 = \prod_{i=1}^{L} (1 - 2p_i).
\] (3.5)

Translating the probabilities into soft bits \( \chi(p) = 2p - 1 \) gives the expression

\[
\chi(p_0) = -\prod_{i=1}^{L} (-\chi(p_i)).
\]

Looking only at the signs of the soft bits, a positive sign corresponds to \( x = 1 \), while a negative sign corresponds to \( x = 0 \). According to the parity check, there should be an even number of 1's in total. Suppose there are already an even number of positive soft-bits. Then the product \( \prod (-\chi(p_i)) \) is positive, giving a negative value for \( \chi(p_0) \). This corresponds to making \( x_0 \) equal to 0, which makes sense because there are already an even number of 1's. Similarly, if there are an odd number of positive soft-bits, then the sign of the product is negative, so that \( \chi(p_0) \) is positive, corresponding to a 1.

Using this example, it is possible to derive expressions for iterating the probabilities between the \( N \) variable nodes (bits) and the \( M \) function nodes (checks) in a low-density parity check (LDPC) codes, using the message passing algorithm. There is an edge between the \( i \)-th variable node to the \( j \)-th function if and only if there is a 1 in the \((j, i)\)-th entry of the parity check matrix. Let \( \text{Row}[j] \) be the set of the column locations of the 1's in the \( j \)-th row, and \( \text{Col}[i] \) be the set of the row locations of the 1's in the \( i \)-th column. The message from a bit to a check (i.e. variable node
to a function node) is denoted by

\[ q_{ij}(0) = \mu_{x_i \rightarrow f_j}(0) \]
\[ q_{ij}(1) = \mu_{x_i \rightarrow f_j}(1) \]

while the message from a check to a bit (i.e. function node to variable node) is

\[ r_{ij}(0) = \mu_{f_j \rightarrow x_i}(0) \]
\[ r_{ij}(1) = \mu_{f_j \rightarrow x_i}(1). \]

The message-passing algorithm then takes the following form:

- **Step 0. Initialize.** The decoder starts with prior probabilities on the bits, which are represented by \((p_{\text{prior}}^{(0)}, p_{\text{prior}}^{(1)})\).

- **Step 1. Messages from bits to checks.** Applying equation (3.1) gives the following expression:

\[
\begin{align*}
q_{ij}(0) &= c_{ij} p_{i}^{\text{prior}}(0) \prod_{j' \in \text{Col}[i] \setminus \{j\}} r_{ij'}(0) \\
q_{ij}(1) &= c_{ij} p_{i}^{\text{prior}}(1) \prod_{j' \in \text{Col}[i] \setminus \{j\}} r_{ij'}(1) \\
\end{align*}
\] (3.6)

where \(c_{ij}\) is introduced to normalize the messages \(q_{ij}(0) + q_{ij}(1) = 1\).

- **Step 2. Messages from checks to bits.** Applying equation (3.2) gives

\[
\begin{align*}
\begin{align*}
    r_{ij}(0) &= \sum_{\{x_i',\} \text{ for } i' \in \text{Row}[j] \setminus \{i\}} f(x_i = 0, \{x_{i'}\}_{i' \in \text{Row}[j] \setminus \{i\}}) \prod_{i' \in \text{Row}[j] \setminus \{i\}} q_{ij'}(x_{i'}) \\
    r_{ij}(1) &= \sum_{\{x_i',\} \text{ for } i' \in \text{Row}[j] \setminus \{i\}} f(x_i = 1, \{x_{i'}\}_{i' \in \text{Row}[j] \setminus \{i\}}) \prod_{i' \in \text{Row}[j] \setminus \{i\}} q_{ij'}(x_{i'}) \\
\end{align*}
\end{align*}
\]

where \(\{x_{i'}\}_{i' \in \text{Row}[j] \setminus \{i\}}\) is the set of all possible values of the bits involved in the
CHAPTER 3. SOFT ITERATIVE DECODING

$j$-th check, except for the $i$-th bit. The function

$$f(x_1, x_2, \ldots, x_l) = 1 \oplus x_1 \oplus \cdots \oplus x_l$$

selects the combinations that are consistent with the even parity check. Applying equation (3.5) yields the expressions

$$1 - 2r_{ij} (1) = \prod_{i' \in \text{Row}[j] \setminus \{i\}} (1 - 2q_{\nu j}(1))$$

$$\chi (r_{ij}) = - \prod_{i' \in \text{Row}[j] \setminus \{i\}} - (\chi (q_{\nu j}))$$

(3.7)

which gives the messages from checks to bits as

$$r_{ij} (0) = \frac{1}{2} \left( 1 + \prod_{i' \in \text{Row}[j] \setminus \{i\}} (q_{\nu j} (0) - q_{\nu j} (1)) \right)$$

$$r_{ij} (1) = \frac{1}{2} \left( 1 - \prod_{i' \in \text{Row}[j] \setminus \{i\}} (q_{\nu j} (0) - q_{\nu j} (1)) \right).$$

(3.8)

(It is possible to interpret this transformation as a discrete Fourier transform. [Che97].)

- Repeat Steps 1 and 2 until some condition is satisfied (such as a maximum number of iterations)

- Step 3. Compute output. The product of all the probabilities sent to a bit yields the posterior probability,

$$p_i^{\text{posterior}} (0) = c_i \cdot p_i^{\text{prior}} (0) \prod_{j' \in \text{Col}[i]} r_{ij'} (0)$$

$$p_i^{\text{posterior}} (1) = c_i \cdot p_i^{\text{prior}} (1) \prod_{j' \in \text{Col}[i]} r_{ij'} (1)$$

where $c_i$ normalizes the probabilities. The extrinsic probability is given by
leaving out the prior probabilities

\[ p_i^{\text{extrinsic}} = c_i \prod_{j' \in \text{Col}[i]} r_{ij'}(0) \]

\[ p_i^{\text{extrinsic}} = c_i \prod_{j' \in \text{Col}[i]} r_{ij'}(1) \]

where \( c_i \) is another normalizing factor.

### 3.2.2 Message-passing (log-likelihood ratios)

The probabilities for binary variables can be represented in terms of log-likelihood ratios (LLR). Let

\[ \text{LLR} \left( \mu_{f_j \rightarrow x_i} \right) = \text{LLR}(r_{ij}) = \log \left( \frac{r_{ij}(1)}{r_{ij}(0)} \right) \]

represent the message from checks \( f_j \) to bits \( x_i \), and let

\[ \text{LLR} \left( \mu_{x_i \rightarrow f_j} \right) = \text{LLR}(q_{ij}) = \log \left( \frac{q_{ij}(1)}{q_{ij}(0)} \right) \]

represent the messages from bits \( x_i \) to checks \( f_j \). Then the algorithm consists of the following steps:

- **Step 0. Initialize.** The decoder begins with prior log-likelihood ratios for the bits.

  \[ \text{LLR}^{\text{prior}}(x_i) = \log \frac{p_i^{\text{prior}}(1)}{p_i^{\text{prior}}(0)} \]

- **Step 1. Messages from bits to checks.** The product in equation (3.6) becomes a summation:

  \[ \text{LLR}(q_{ij}) = \sum_{j' \in \text{Col}[i] \setminus \{j\}} \text{LLR}(r_{ij'}) + \text{LLR}^{\text{prior}}(x_i) \quad (3.9) \]
• **Step 2. Messages from checks to bits.** Using the relation $\chi(x) = \tanh\left(\frac{1}{2}\text{LLR}(x)\right)$, it is possible to rewrite expression (3.7) in terms of log-likelihood ratios

$$\text{LLR}(r_{ij}) = -2 \tanh^{-1} \left( \prod_{i' \in \text{Row}[j] \setminus \{i\}} - \tanh\left(\frac{1}{2}\text{LLR}(q_{ij})\right) \right). \quad (3.10)$$

• Repeat Steps 1 and 2 until the stop criterion is met.

• **Step 3. Compute output.** The posterior and extrinsic probabilities can be read out when the iterations are complete.

\[
\begin{align*}
\text{LLR}^{\text{posterior}}(x_i) &= \sum_{j' \in \text{Col}[i]} \text{LLR}(r_{ij'}) + \text{LLR}^{\text{prior}}(x_i) \quad (3.11) \\
\text{LLR}^{\text{extrinsic}}(x_i) &= \sum_{j' \in \text{Col}[i]} \text{LLR}(r_{ij'}) \quad (3.12)
\end{align*}
\]

Hence, the decoder iterates log-likelihood ratios between the variable and function nodes using expressions (3.9) and (3.10). This last equation (3.10) can be reformulated in a form which is more suitable for implementation. In particular, the product can be converted into a sum, which is computationally less intensive. Define the function

$$\phi(x) = \log \left( \tanh\left(\frac{1}{2}x\right) \right) = \log \frac{\exp(x) - 1}{\exp(x) + 1}$$

which exists for $x > 0$. As observed by Gallager [Gal63], this function is its own inverse $\phi(x) = \phi^{-1}(x)$. Expression (3.10) then takes the form:

\[
\text{LLR}(r_{ij}) = \phi^{-1} \left( \sum_{i' \in \text{Row}[j] \setminus \{i\}} \phi \left( |\text{LLR}(q_{i'j})| \right) \right) \cdot \prod_{i' \in \text{Row}[j] \setminus \{i\}} \text{sgn}(\text{LLR}(q_{i'j}))( -1)^{|\text{Row}[j]|} \quad (3.13)
\]
Therefore, using equations (3.9) and (3.13), it is possible to implement the message passing algorithm for LDPC codes using only the addition operation and a look-up table for evaluating the function $\phi = \phi^{-1}$.

There is also a natural criterion for deciding when to stop the message-passing algorithm for LDPC codes, which can significantly reduce the average number of iterations: After each iteration, it is possible to make hard-decisions based on the log-likelihood ratios.

\[
\widehat{x}_i = \begin{cases} 
1 & \text{if } \text{LLR}^{\text{posterior}}(x_i) \geq 0 \\
0 & \text{if } \text{LLR}^{\text{posterior}}(x_i) < 0 
\end{cases}
\]

Then it is possible to check if all the parity check constraints have been met, by computing the product of the parity check matrix $A$ and the vector

\[
\widehat{x} = \begin{bmatrix} \widehat{x}_0 & \widehat{x}_1 & \cdots & \widehat{x}_{N-1} \end{bmatrix}^T.
\]

If $A\widehat{x} = 0$, then the iterations are complete. The probability that a randomly chosen word will satisfy all the constraints is $\frac{1}{2^n}$. From simulations, it has been found that the message-passing decoder is unlikely to produce false decodings, especially if the code has reasonably large minimum distance. [Mac99].

### 3.2.3 Approximations

The expression (3.13) can be simplified by observing that the summation is usually dominated by the minimum term, giving the approximation:

\[
\text{LLR}(r_{ij}) \approx \min_{i' \in \text{Row}[j] \setminus \{i\}} |\text{LLR}(q_{i'j})| \cdot \prod_{i' \in \text{Row}[j] \setminus \{i\}} \text{sgn}(\text{LLR}(q_{i'j})) (-1)^{|\text{Row}[j]|} \quad (3.14)
\]
CHAPTER 3. SOFT ITERATIVE DECODING

This approximation provides a method for reduced complexity decoding, as described in [HOP96][FMI99]. Together with

$$\text{LLR}(q_{ij}) = \sum_{j' \in \text{Col}[i] \setminus \{j\}} \text{LLR}(r_{ij'}) + \text{LLR}^{\text{prior}}(x_i)$$  \hspace{1cm} (3.15)

it can be seen that the approximate decoding algorithm for LDPC codes can be implemented only with the minimum function, real additions and some binary arithmetic.

It is interesting to note that this approximate decoder can be applied directly to the AWGN channel without the need for an estimate of the noise variance of the channel. Recall that $\text{LLR}^{\text{prior}}(x_i) = \frac{2}{\sigma^2} y_i$ for the AWGN channel, where $y_i$ is the received sample. By using a normalized log-likelihood ratio (NLLR), so that $\text{NLLR}(x_i) = \frac{2}{\sigma^2} \text{LLR}(x_i)$, it is possible to rewrite the equations as follows:

$$\text{NLLR}(r_{ij}) \approx \min_{i' \in \text{Row}[j] \setminus \{i\}} |\text{NLLR}(q_{i'j})| \cdot \prod_{i' \in \text{Row}[j] \setminus \{i\}} \text{sgn}(\text{NLLR}(q_{i'j})) (-1)^{|\text{Row}[j]|}$$  \hspace{1cm} (3.16)

$$\text{NLLR}(q_{ij}) = \sum_{j' \in \text{Col}[i] \setminus \{j\}} \text{NLLR}(r_{ij'}) + y_i$$

After sufficient iterations, the final bit values are found as follows:

$$\text{NLLR}(q_i) = \sum_{j' \in \text{Col}[i]} \text{NLLR}(r_{ij'}) + y_i$$

$$\hat{x}_i = \text{sgn}(\text{NLLR}(q_i))$$

These equations show that in the case of the approximate decoder, it is possible to omit the estimation of the noise variance, as observed by Fossorier et al. [FMI99], who refer to this decoder as the uniformly most powerful (UMP) iterative decoding algorithm based on belief propagation.

Finally, this expression can be simplified even further by replacing the messages from bits to checks ($q_{ij}$) with pseudo-posterior information ($q_i$), as proposed
in [FM199], which refers to this decoder as the UMP decoder based on the a posteriori probabilities (APP). In other words, let

$$\text{NLLR}(q_i) = \sum_{j' \in \text{Col}[i]} \text{NLLR}(r_{ij'}) + y_i,$$

where the summation is over all of Col[i], and

$$\text{NLLR}(r_{ij}) \approx \min_{i' \in \text{Row}[j] \setminus \{i\}} |\text{NLLR}(q_{i'})| \cdot \prod_{i' \in \text{Row}[j] \setminus \{i\}} \text{sgn} \left( \text{NLLR}(q_{i'}) \right) (-1)^{|\text{Row}[j]|}. \quad (3.17)$$

This expression (3.17) further reduces the number of computations, as compared with (3.16), since only a single output is computed for each bit, but the performance is slightly degraded.

### 3.2.4 Performance on AWGN channel

While binary sequences are naturally represented as $x_i \in \{0, 1\}$, it is more convenient to use a representation $x_i \in \{-1, 1\}$ instead, where $x_i = 2x_i - 1$, for transmitting over a channel. In the case of a channel with additive white Gaussian noise (AWGN), where there is no intersymbol interference (ISI), the channel receiver can estimate the channel log-likelihood ratio:

$$\text{LLR}_{\text{channel}}^{\text{posterior}}(x_i) = \log \frac{P(x_i = 1 \mid y_i)}{P(x_i = -1 \mid y_i)}$$

$$= \log \frac{P(y_i \mid x_i = 1)}{P(y_i \mid x_i = -1)} + \log \frac{P(x_i = 1)}{P(x_i = -1)}$$

$$= \log \frac{\exp \left( -\frac{1}{2\sigma^2} (y_i - 1)^2 \right)}{\exp \left( -\frac{1}{2\sigma^2} (y_i + 1)^2 \right)} + \text{LLR}_{\text{channel}}^{\text{prior}}(x_i)$$

$$= \frac{2}{\sigma^2} y_i + \text{LLR}_{\text{channel}}^{\text{prior}}(x_i)$$
Figure 3.9: Performance of an LDPC decoder after different numbers of iterations

There is normally no prior information for this channel decoder, so that the input to the LDPC or turbo decoder is given by $\text{LLR}^{\text{prior}}(x_i) = \frac{2}{\sqrt{2}} y_i$.

Figure 3.9 shows the bit error rate performance of a LDPC code after different numbers of iterations on a ISI-free channel with additive white Gaussian noise (AWGN). The rate of the LDPC code is $R = \frac{8}{9}$. The measure for the channel noise is given by in terms of the signal to noise ratio (SNR), which is defined as $\frac{E_k}{N_0} = \frac{1}{2R\sigma^2}$, where the power is assumed to be 1, $R$ is the code rate, and $\sigma^2$ is the noise variance. The SNR is dB is given by $10 \log_{10} \left( \frac{E_k}{N_0} \right)$.

3.2.5 Complexity

The complexity of LDPC codes is estimated by counting the number of operations involved in decoding. By considering log-likelihood ratios rather than probabilities, it is possible to convert multiplications into additions using the appropriate transformations. To count the complexity, it helps to consider the simple problem of calculating all $n$ sums of the subsets of $n - 1$ numbers, for a list of $n$ numbers $z_1, z_2, \ldots, z_n$. First,
sum up the first \( n - 1 \) numbers \( \sum_{i=1}^{n-1} z_i \), using \( n - 2 \) additions. Then add the last number for the total sum \( T = \sum_{i=1}^{n} z_i \). Then for \( i = 1, 2, ..., n - 1 \), subtract off each of the numbers to obtain the sums of the subsets \( T - z_i \). This requires a total of \((n - 2) + 1 + (n - 1) = 2(n - 1)\) additions.

For a regular LDPC code, suppose that there are \( t_c \) bits per column in the parity check matrix, and \( t_r \) bits per row in the parity check matrix. To find all the messages from a bit to check, it is required to find the \( t_c \) sums of subsets (of size \( t_c - 1 \)), which uses \( 2(t_c - 1) \) additions. In addition, there is also the prior information, which requires another addition, for a total of \( 2t_c - 1 \) additions. On the other hand, each of the \( M \) checks has \( t_r \) bits connected to it, so using expression (3.13), each can be implemented by \( 4t_r \) table look-ups, and \( 2(t_r - 1) \) additions. Since \( Mt_r = Nt_c \), this is equivalent to \( 4Mt_r = 4Nt_c \) table look-ups and \( 2M(t_r - 1) = 2(Nt_c - M) \) additions. Hence for each iteration of the message-passing algorithm, there are a total of \( 4t_c \) table look-ups per bit, and \( 4t_c - 1 - 2 \frac{M}{N} \approx 4t_c \) addition operations.

On the other hand, for the approximation in equation (3.14), the minimum function can be implemented using compare functions. The minimum of \( n \) numbers can be found in a straightforward manner using \( n - 1 \) comparisons of two numbers, and it is also possible to use tree-based techniques to reduce the number of comparisons, as in [FMI99]. Ignoring binary operations, which are relatively inexpensive compared with real additions and multiplications, the number of operations is then given by \( 2(t_c - 1) \) additions per bit (for messages from bits to checks) and \( \frac{1}{N}M(t_r - 1) = t_c - \frac{M}{N} \approx t_c \) compares per bit (for messages from checks to bits). These results are summarized in Table 3.1 (p. 79).

### 3.3 Forward-Backward Algorithm

For sequences that correspond to paths through a trellis, the Viterbi algorithm finds the maximum likelihood (ML) sequence using dynamic programming. On the other
CHAPTER 3. SOFT ITERATIVE DECODING

hand, the forward-backward algorithm (FBA), also known as the BCFR or Bahl-Jelinek algorithm [BCJR74], is used to find the maximum a \textit{posteriori} (MAP) decisions, rather than the maximum likelihood sequence.

\[ u_{i}^{MAP} = \arg \max_{u_{i}} \Pr(u_{i} \mid y) \]

\[ u_{i}^{ML} = \arg \max_{u} \Pr(u \mid y) \]

Based on prior information for each bit, this algorithm produces posterior probabilities for each bit, and is often referred to as an \textit{a posteriori} probability (APP) decoder. Since the input and output both consist of soft information, this decoder is also referred to as a "soft-in, soft-out" (SISO) decoder. The forward-backward algorithm can be applied to a wide variety of decoding problems, including decoding convolutional codes (as in turbo decoding) and decoding an intersymbol interference channel. In addition, it will be applied in Chapter 4 to the problem of decoding the modulation constraint.

3.3.1 Derivation

The forward-backward algorithm is presented in the context of a time-invariant trellis with a finite number of states, where the paths encode a binary sequence. (It is also possible to generalize the FBA to time-varying and non-binary trellises.) Consider the binary trellis with \( M \) states, as pictured in Figure 3.10. Let \( s_{i} \) and \( s_{i-1} \) be states (denoted by \( 0, 1, \ldots, M - 1 \)) at times \( i \) and \( i + 1 \), respectively. If there is an edge between these states, then the edge is labeled by a bit value \( x(s_{i}, s_{i+1}) \in \{0, 1\} \). In addition, it is assumed that there are two or fewer edges out of each state (in the forward direction), and that if there are two edges, they are distinctly labeled. In other words, \( x(s_{i}, s) = x(s_{i}, s') \) if and only if \( s = s' \).

Finally, let the sequence \( w = \{\ldots, w_{i-1}, w_{i}, w_{i+1}, \ldots\} \) denote a set of observations, where \( w_{i} \) is the current observation at time \( i \), and

\[ w_{i}^{-} = \{\ldots, w_{i-2}, w_{i-1}\} \]

\[ w_{i}^{+} = \{w_{i+1}, w_{i+2}, \ldots\} \]
represent the past and the future observations, respectively. The system satisfies the Markov property so that given a previous state, the future is independent of the past, and vice versa.

\[
P(w_i \mid s_i, w^-_i) = P(w_i \mid s_i)
\]

\[
P(w_i \mid s_{i+1}, w^+_i) = P(w_i \mid s_{i+1})
\]

Figure 3.10: Forward-backward algorithm as message-passing

The probability of a particular edge \((s_i, s_{i+1})\) can then be factored as follows,

\[
P(s_i, s_{i+1}, w^-, w_t, w^+)
\]

\[
= P(s_i, w^-) P(s_{i+1}, w_t, w^+ \mid s_i, w^-)
\]

\[
= P(s_i, w^-) P(s_{i+1} \mid s_i, w^-) P(w_t, w^+ \mid s_{i+1}, s_i, w^-)
\]

\[
= P(s_i, w^-) P(s_{i+1} \mid s_i) P(w_t \mid s_i, s_{i+1}) P(w^+ \mid s_{i+1})
\]

\[
= \alpha_i(s_i) Q_{s_i, s_{i+1}} \gamma_i(w_t, s_i, s_{i+1}) \beta_{i+1}(s_{i+1})
\]

(3.18)

where the following notation is used to represent the terms in this expression.

\[
\alpha_i(s_i) = P(s_i, w^-)
\]

\[
\beta_{i+1}(s_{i+1}) = P(w^+ \mid s_{i+1})
\]
\[ \gamma_i(w_i, s_i, s_{i+1}) = P(w_i \mid s_i, s_{i+1}) \]
\[ Q_{s_i, s_{i+1}} = P(s_{i+1} \mid s_i) \]

The \(\alpha\)'s and \(\beta\)'s correspond to the probability distribution on the states based on the past and future information about the bits, respectively. The two states \(s_i\) and \(s_{i+1}\) specify a particular edge, and the term \(\gamma_i(w_i, s_i, s_{i+1})\) takes into account the observations to specify the likelihood of that edge. Finally, the transition probability \(Q_{s_i, s_{i+1}}\) gives the probabilities of the state transitions on the trellis. For uniform trellises, this is usually constant and can be ignored.

The following recursive formula gives the forward decoding pass,

\[
P(s_{i+1}, w^-, w_i) = \sum_{s_i} P(s_{i+1}, s_i, w^-, w_i)
\]
\[
= \sum_{s_i} P(w_i, s_{i+1} \mid s_i, w^-) P(s_i, w^-)
\]
\[
= \sum_{s_i} P(w_i \mid s_i, s_{i+1}) P(s_{i+1} \mid s_i, w^-) P(s_i, w^-)
\]
\[
= \sum_{s_i} P(w_i \mid s_i, s_{i+1}) P(s_{i+1} \mid s_i) P(s_i, w^-)
\]

which can be rewritten as

\[
\alpha_{i+1}(s_{i+1}) = \sum_{s_i'} \gamma_i(w_i, s_i', s_{i+1}) Q_{s_i', s_{i+1}} \alpha_i(s_i') \quad (3.19)
\]

A similar recursion for the backward decoding pass is

\[
P(w^+, w_i \mid s_i) = \sum_{s_{i+1}} P(w^+, w_i, s_{i+1} \mid s_i)
\]
\[
= \sum_{s_{i+1}} P(w_i \mid s_i, s_{i+1}, w^+) P(s_{i+1}, w^+ \mid s_i)
\]
\[
= \sum_{s_{i+1}} P(w_i \mid s_i, s_{i+1}) P(s_{i+1} \mid s_i) P(w^+ \mid s_{i+1})
\]
which can be summarized as

$$\beta_i(s_i) = \sum_{s_{i+1}} \gamma_i \left( w_i, s_i, s'_{i+1} \right) Q_{s_i, s'_{i+1}} \beta_{i+1}(s'_{i+1}).$$

Finally, the posterior probabilities are obtained using (3.18) as follows

$$P(x_i, w) = \sum_{s_i, s_{i+1}} P(x_i, w, s_i, s_{i+1})$$

$$= \sum_{s_i, s_{i+1}} \alpha_i(s_i) Q_{s_i, s_{i+1}} \gamma_i \left( w_i, s_i, s_{i+1} \right) \beta_{i+1}(s_{i+1})$$

where the summation is over all combinations of \( s_i \) and \( s_{i+1} \) such that \( x(s_i, s_{i+1}) = x_i \). Then normalizing gives the desired posterior probabilities

$$P_{\text{posterior}}(x_i = 1 | w) = \frac{P(x_i = 1, w)}{P(x_i = 0, w) + P(x_i = 1, w)}.$$

The extrinsic information can also be found from the posterior information by subtracting the prior information (in terms of log-likelihood ratios).

**Prior information**

When the FBA decoder receives prior probabilities \( P_{\text{prior}}(x_i) \) on the bits \( x_i \), there are two ways that this prior information can be interpreted. One common approach is to view the prior information as information about the source, so that it is used to weight the state transitions \( Q_{s_i, s_{i+1}} \) appropriately.

The approach that we will take, however, is to view the prior probabilities as part of the observations \( w_i \), since these probabilities are external input to this decoder. This is appropriate since the prior probabilities may correspond to the output from another decoder that processes observed samples. Then the observation variable \( w_i \) can be divided into two parts \( w_i = \{ u_i, v_i \} \), where \( u_i \) is a binary random variable that corresponds to the prior information on \( x_i \), and \( v_i \) corresponds to other observed variables. The distribution on \( u_i \) is given by \( P_{\text{prior}}(x_i) \), so that the conditional probability
is proportional to $P^\text{prior} (x_i)$, so

$$P (u_i \mid s_i, s_{i+1}) = P (u_i \mid x_i) = c'_i P^\text{prior} (x_i).$$

Since this proportionality constant $c'_i$ is the same for all values of $x_i$, then without loss of generality, it is possible to omit this term in the expressions, since it is the relative proportions of the messages that is important. Then it is possible to factor the $\gamma$ expression as follows,

$$\gamma_i (w_i, s_i, s_{i+1}) = P (u_i, v_i \mid s_i, s_{i+1})$$
$$= P (v_i \mid s_i, s_{i+1}, u_i) P (u_i \mid s_i, s_{i+1})$$
$$= P (v_i \mid s_i, s_{i+1}) P^\text{prior} (x_i) \tag{3.22}$$

where $x_i = x (s_i, s_{i+1})$. Also, $v_i$ depends only on the transition $s_i$ to $s_{i+1}$, and not on the prior information $u_i$.

The extrinsic probability can be found by omitting the prior probability in equation (3.21),

$$P^\text{extrinsic} (x_i \mid w) = c'_i \cdot \sum_{s_i, s_{i+1}} \alpha_i (s_i) Q_{s_i, s_{i+1}} P (v_i \mid s_i, s_{i+1}) \beta_{i+1} (s_{i+1}) \tag{3.23}$$

where $c'_i$ is a normalization constant.

**Message-passing algorithm**

This forward-backward algorithm can also be viewed as a form of message-passing, on a linear graph with alternating state nodes, corresponding to the states of the trellis, and function nodes, corresponding to the edges of the trellis. Following the discussion in §3.1.1, the forward and backward messages are proportional to probabilities as follows:

$$\mu_{f_{i-1} \rightarrow s_i} (s_i) = c_i \cdot P (w^- \mid s_i)$$
$$\mu_{f_{i+1} \rightarrow s_{i+1}} (s_{i+1}) = c'_{i+1} \cdot P (s_{i+1}, w^+).$$
CHAPTER 3. SOFT ITERATIVE DECODING

The prior information comes from the variable node \(x_i\), and there are also an observation variable \(u_i\), which ranges over the possible transitions \((s_i, s_{i+1})\), so that there is a separate probability message for each edge. Note that there is a unique bit label \(x_i = x(s_i, s_{i+1})\) for each transition.

\[
\mu_{x_i \rightarrow f_i}(x_i) = P_{\text{prior}}(x_i) \\
\mu_{u_i \rightarrow f_i}(x_i, s_i, s_{i+1}) = P(s_i, s_{i+1} \mid u_i)
\]

Then applying the message-passing algorithm in Section 3.1 gives the following recursions,

\[
\mu_{f_i \rightarrow s_{i+1}}(s_{i+1}) = \sum_{x_i, s_i, s_{i+1}} f(x_i, s_i, s_{i+1}) \mu_{s_i \rightarrow f_i}(s_i) \mu_{x_i \rightarrow f_i}(x_i) \mu_{u_i \rightarrow f_i}(s_i, s_{i+1}) \\
\mu_{f_i \rightarrow s_i}(s_i) = \sum_{x_i, s_i, s_{i+1}} f(x_i, s_i, s_{i+1}) \mu_{s_{i+1} \rightarrow f_i}(s_{i+1}) \mu_{x_i \rightarrow f_i}(x_i) \mu_{u_i \rightarrow f_i}(s_i, s_{i+1})
\]

Then the extrinsic probabilities are obtained as follows

\[
\mu_{f_i \rightarrow x_i}(x_i) = c_i \cdot \sum_{s_i, s_{i+1}} f(x_i, s_i, s_{i+1}) \mu_{s_i \rightarrow f_i}(s_i) \mu_{s_{i+1} \rightarrow f_i}(s_{i+1}) \mu_{u_i \rightarrow f_i}(x_i, s_i, s_{i+1})
\]

where \(c_i\) normalizes the probability.

It is possible to make a comparison between this message-passing algorithm and the forward-backward algorithm using the following glossary

\[
\mu_{f_i \rightarrow s_i}(s_i) = c_i \cdot \alpha_i(s_i) \\
\mu_{f_i+1 \rightarrow s_{i+1}}(s_{i+1}) = c_{i+1} \cdot \beta_{i+1}(s_{i+1}) \\
\mu_{u_i \rightarrow f_i}(x_i, s_i, s_{i+1}) = \frac{P(s_i, s_{i+1})}{P(u_i)} P(u_i \mid s_i, s_{i+1}) \\
f(x_i, s_i, s_{i+1}) = \begin{cases} Q_{x_i, s_{i+1}} & \text{if } x(s_i, s_{i+1}) = x_i \\ 0 & \text{otherwise} \end{cases}
\]

where \(w_i = \{u_i, v_i\}\). It turns out that the two formulations (message-passing and
the forward-backward algorithm) are the same except for constant factors. When
the posterior and extrinsic probabilities are computed, these values are normalized,
so that the resulting probabilities will be the same. Hence the forward-backward
algorithm can be seen to be a direct application of the message-passing algorithm.

3.3.2 Intersymbol interference channel

Consider a system where the decoder receives channel samples \( \{y_i\} \) from an inter-
symbol interference (ISI) channel

\[
y_i = \sum_{j=0}^{\nu} h_j x_{i-j} + n_i,
\]

where \( x_i \in \{-1, 1\} \) is the PAM (pulse amplitude modulation) version of the binary
variable \( x_i \in \{0, 1\} \). The channel polynomial \( h(D) = \sum_{j=0}^{\nu} h_j D^j \) is real-valued, and
without loss of generality, assumed to be monic (\( h_0 = 1 \)). Suppose that the decoder
receives both channel samples and prior probabilities for the bits. Then the FBA can
be used to compute the posterior and extrinsic probabilities based on the trellis for
the intersymbol interference channel.

![Trellis for Intersymbol Interference Channel](image)

Figure 3.11: Trellis for Intersymbol Interference Channel

The trellis for intersymbol interference, as in Figure 3.11, has \( 2^\nu \) states \( s_i = \)
(x_{i-\nu}, ..., x_{i-1}) that record the memory of the channel. There is a transition between 
\( s_i = (\tilde{x}_{i-\nu}, ..., \tilde{x}_{i-1}) \) and \( s_{i+1} = (x_{i-\nu+1}, ..., x_i) \), whenever \( \tilde{x}_{i-j} = x_{i-j} \) for \( j = 1, 2, ..., \nu - 1 \), and the transition is labeled by \( x_i \). Following (3.22), the edge contributes the following probabilities,

\[
\gamma_i (x_i, y_i, s_i, s_{i+1}) = P(y_i \mid s_i, s_{i+1}, x_i) \, P(x_i \mid s_i, s_{i+1})
\]

where the channel sample contributes the following probability, assuming Gaussian noise,

\[
P(y_i \mid s_i, s_{i+1}, x_i) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left( -\frac{1}{2\sigma^2} \left( y_i - \sum_j h_j x_{i-j} \right)^2 \right)
\]

where \( x_{i-j} = 2x_{i-j} - 1 \), and \( x_i = x(s_i, s_{i+1}) \). The other term is the prior probability on \( x_i \),

\[
P(x_i \mid s_i, s_{i+1}) = P_{\text{prior}} (x_i).
\]

Then the forward and backward passes are given by (3.19) and (3.20), and the extrinsic probabilities are obtained by summing out as follows:

\[
p_i^{\text{extrinsic}} (x_i) = c_i \sum_{x(s_i, s_{i+1})=x_i} \alpha_i (s_i) \beta_{i+1} (s_{i+1}) \, P(y_i \mid s_i, s_{i+1}, x_i)
\]

where \( c_i \) is the appropriate normalization.

In cases where the noise statistics are non-linear or data-dependent, it is also possible to adjust this function appropriately. Having defined the FBA algorithm for an ISI channel, it is possible to combine this channel decoder module with an ECC decoder, and iterate soft information between the two decoders. This configuration is referred to as "turbo equalization" in the case of turbo codes. [RMM98]. Finally, it should be mentioned that there are simplified methods of obtaining soft information for the channel decoder, including the soft-output Viterbi algorithm [HH89], and the soft cancellation techniques described in Appendix A.
CHAPTER 3. SOFT ITERATIVE DECODING

3.3.3 Turbo codes

The forward-backward algorithm can also be applied to the decoding of convolutional codes. Consider an example of a rate $\frac{1}{2}$ systematic convolutional code, where the encoder uses a finite set of shift registers to produce parity bits $v_i$ from the input sequence $x_i$ (which is referred to as the systematic bit), so that at each time $i$, the encoder takes in a single user bit $x_i$ and produces the output $w_i = \{x_i, v_i\}$. The state variable $s_i = (x_{i-\nu}, ..., x_{i-1})$ records the memory of the code and can be in one of $2^\nu$ states, corresponding to the possible configurations of the last $\nu$ user bits.

The input to the decoder consists of prior probabilities (based on observations) on both the systematic and parity bits. Then using (3.22), the contribution of each transition is given by

$$\gamma_i (w_i, s_i, s_{i+1}) = P(x_i | s_i, s_{i+1}) P(v_i | s_i, s_{i+1})$$

$$= p_{prior} (x_i) p_{prior} (v_i)$$

Then the forward and backward passes are given by (3.19) and (3.20), and the extrinsic probabilities are obtained by summing out as follows:

$$p_i^{extrinsic} (x_i) = c_i \sum_{x(s_i, s_{i+1})=x} \alpha_i (s_i) \beta_{i+1} (s_{i+1}) p_{prior} (v_i)$$

where $c_i$ is the appropriate normalization.

This soft decoder for convolutional codes is the building block for turbo codes [BGT93]. Three configurations are pictured in Figure 3.12, where $\Pi$ refers to a random permuter (often called an "interleaver"), and $\Pi^{-1}$ is its inverse.

In parallel concatenated turbo codes, a user data sequence $m$ is encoded using two systematic encoders for convolutional codes. The first encoder takes the user data directly and produces a sequence of parity bits $p^{(1)}$, while the second encoder takes a permuted version of the user data and produces the parity bits $p^{(2)}$. The permuter $\Pi$ creates randomness in the code and creates large cycles in the corresponding factor graph. The decoder for this code consists of decoders based on the forward-backward
Figure 3.12: Three configurations for turbo coding
algorithm for the two constituent convolutional codes. Starting with prior probabilities on \( m, p^1, \) and \( p^2, \) the decoder passes information about the data sequence \( m \) between these decoding modules, using the soft information on \( p^1 \) and \( p^2 \) in the decoding.

In parallel concatenation with channel iteration, the extrinsic information from the FBA decoders is summed together (as LLRs) and passed back as prior information to the soft channel decoder, leading to improved performance. This iteration between the channel decoder and the FBA decoders for the convolutional code is known as "turbo equalization." [RMM98].

Another structure is serial concatenation, in which the first convolutional encoder produces the parity \( p^1 \) on the systematic bits \( m \), and the input to a second encoder is a permuted version of bits from both \( m \) and \( p^1 \). It turns out that it is possible to use the intersymbol interference channel as the inner code, in which case, serial concatenation is a form of "turbo equalization," iterating soft information between a single convolutional decoder and the channel decoder.

### 3.3.4 Complexity

We compute the number of operations needed for the forward-backward algorithm, assuming the trellis has a regular structure, with \( M \) states and two edges coming in and out of every state, and all numbers are represented as probabilities. The forward pass in equation (3.19) has 3 terms in each product and there are two terms to add together, requiring a total of \( 4M \) multiplies and \( M \) additions per bit. The same is true of the backward pass in equation (3.20). In addition, the FBA for intersymbol interference requires a computation of the term \( P(y_i | x_i, s_i, s_{i+1}) \), which requires \( 2M \) such computations (which can be implemented using a table look-up) per bit. Finally, to sum out the iteration to obtain extrinsic probabilities as in equation (3.23), it is necessary to take the sum of a product of \( \alpha \)'s and \( \beta \)'s over all \( 2M \) edges corresponding to each bit value, so that the number of operations is \( 4M \) multiplications and \( 2(M-1) \) additions. The total number of operations per bit is roughly \( 12M \) multiplies and \( 4M \) additions per bit. By pre-multiplying \( \gamma_i(w_i, s_i, s_{i+1})Q_{s_i,s_{i+1}} \) and storing these
quantities, it is possible to reduce the number of operations to $10M$ multiplies and $4M$ additions per bit. In addition, for the case of the ISI channel decoder, there are an additional $2M$ table look-ups.

In a practical implementation, it is desirable to avoid real-valued multiplication by converting all the probabilities to the log-probability domain, where $\log(pq) = \log p + \log q$, and $\log(p+q) = \log p + \log(1+p/q)$, where $p > q$. The latter term can be implemented by a look-up table

$$f(p, q) = \log(1+p/q)$$

as a correction term, with a compare function used to decide whether $p > q$. This transformation converts a multiplication into an addition, and an addition into a combination of 1 add, 1 compare and 1 table look-up. Hence the total number of operations then becomes $14M$ adds, $4M$ compares and $4M$ table look-ups per bit per iteration.

### 3.4 Comparison of turbo and LDPC codes

The complexity and performance of LDPC and turbo codes are compared for a magnetic recording system. The simulations of turbo codes are due to Arnon Friedmann and Erozan Kurtas. [FKFM99][FKFM00].

#### 3.4.1 Complexity and performance

The complexity of turbo decoding (using the forward-backward algorithm with log-probabilities) and LDPC decoding is compared in Table 3.1, which gives the number of operations per bit per iteration. Table 3.2 shows that for each iteration, the number of operations required for an LDPC code is less than a turbo code with 4-state convolutional codes, and much less than one using 16-state convolutional codes.

The number of iterations required for a fixed level of performance, however, is greater for LDPC codes than for turbo decoding. The exact number of iterations depends on the block length and can be found empirically. The complexity of LDPC
**Table 3.1:** Comparison of complexity of turbo and LDPC codes, per bit per iteration

<table>
<thead>
<tr>
<th></th>
<th>Additions</th>
<th>Look-ups</th>
<th>Compares</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDPC decoding</td>
<td>$4t_c - 1$</td>
<td>$4t_c$</td>
<td>0</td>
</tr>
<tr>
<td>LDPC min approx</td>
<td>$2t_c - 2$</td>
<td>0</td>
<td>$t_c$</td>
</tr>
<tr>
<td>FBA for conv. code</td>
<td>$14M$</td>
<td>$4M$</td>
<td>$4M$</td>
</tr>
</tbody>
</table>

**Table 3.2:** Some examples of turbo and LDPC decoding complexity, per bit per iteration

<table>
<thead>
<tr>
<th></th>
<th>Additions</th>
<th>Look-ups</th>
<th>Compares</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDPC decoding $t_c = 3$</td>
<td>11</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>LDPC min approx $t_c = 3$</td>
<td>4</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>FBA for 4-state conv. code</td>
<td>56</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>FBA for 16-state conv. code</td>
<td>224</td>
<td>64</td>
<td>64</td>
</tr>
</tbody>
</table>

codes is so much lower than turbo codes that for the same complexity it is possible to run many times more iterations of the LDPC code. The example shown in Figure 3.9 (on p. 66) indicates that the number of iterations required for satisfactory performance is reasonably low, with 8 iterations giving close to the maximal performance.

Figure 3.13 compares the bit error rate performance of a LDPC code with a turbo code (ref. §3.3.3) on an ISI-free channel with additive white Gaussian noise (AWGN).

Some advantages of LDPC codes over turbo codes are the following:

- **Lower complexity** - The number of the operations per iteration is much smaller for LDPC codes than for turbo codes. While it is necessary to use more iterations of the LDPC decoder than of the turbo codes, the overall complexity is usually still lower.

- **Stopping criterion and fewer undetected errors** - It is possible to use the parity checks to detect when a codeword has been reached, sparing unnecessary iterations of the decoder and reducing the average number of iterations per decoding.

- **Parallelizable implementation** - The decoding algorithm of LDPC codes allows us to decode all the bits and parity checks in parallel.
3.4.2 Comparison for magnetic storage channel

There have been numerous papers on the application of turbo codes to magnetic storage, such as [RMM98] and [SFOSW99]. In [FKFM99], LDPC codes were also applied to magnetic storage, and their performance compared with turbo codes. In particular, the data was coded in blocks of $K = 4352$ message bits, corresponding to a sector of 512 user bytes that is slightly expanded due to a rate $\frac{8}{9}$ modulation code. (No modulation code is actually applied in these simulations, although in a practical application of turbo and LDPC codes to magnetic recording systems, the modulation constraint must be considered, as in Chapter 4.) For this comparison, two code rates are considered for the LDPC and turbo codes. In the case of a rate $8/9$ code, the overall block length is $N = 4896$, while in the case of a rate $16/17$ code, the overall block length is $N = 4624$.

The channel impulse response is modeled as a perfectly equalized partial response polynomial EPR4, where the channel output is given by $y(D) = h(D) x(D) + n(D)$
and the channel is \( h(D) = 1 + D - D^2 - D^3 \). This model corresponds to partial response maximum likelihood (PRML), a common technique in which the channel is equalized to a particular polynomial target. In a realistic system, the noise after equalization will most likely be colored due to the mismatch between the channel response and the partial response polynomial, but in this model, the noise \( n(D) \) is assumed to be additive white Gaussian noise. In [FKFM00], LDPC codes are also applied to a channel modeled by an equalized Lorentzian pulse.

A block diagram of a system with an EPR4 channel and LDPC decoder is pictured in Figure 3.14. To decode the channel in a manner compatible with soft iterative decoding, it is necessary to apply a channel decoder using the forward-backward algorithm to decode the intersymbol interference, as in §3.3.2. In analogy to “turbo equalization” [RMM98], soft information is iterated between the LDPC decoder and the channel decoder, which results in a joint decoding of channel and code, but with less much complexity than a maximum-likelihood decoder.

In the block diagram in Figure 3.14, the channel decoder begins by taking the received channel samples (and no prior information on the bits) and producing output probabilities which are then passed as prior information into the decoder for the LDPC code. The LDPC decoder operates on this data using the message-passing algorithm, as described in the next section, generating extrinsic information which can then be passed back to the BCJR channel decoder as prior information.

Figure 3.15 shows the details of message passing between the channel decoder and the LDPC decoder. The extrinsic output of the channel decoder is denoted by
CHAPTER 3. SOFT ITERATIVE DECODING

Figure 3.15: Message passing between the channel decoder and the LDPC decoder

\( p_i^{(B)} \), and this becomes the prior probability \( p_i^1 \) for the LDPC decoder. On the other hand, the output extrinsic probability \( e_i^1 \) from the LDPC decoder is denoted by \( p_i^{(A)} \).

The iteration proceeds as follows: the channel decoder processes the received signal, and passes \( p_i^{(B)} \) to the LDPC decoder, which generates its own extrinsic output \( p_i^{(A)} \). This is passed back as prior information for the channel decoder, which then decodes again. The updated output of the BCJR algorithm is the new \( p_i^{(B)} \), which is taken as the updated prior information for the LDPC decoder. It should be noted that at this stage, to calculate the messages \( \{q_{ij}\} \), it is necessary to use the messages \( r_{ij} \) from the previous iteration. This differs from turbo codes, where each iteration of the forward-backward algorithm for the constituent convolutional codes operates only on the prior information into the decoder, and does not use computed values from the previous iteration.

In Figure 3.16, the intersymbol interference channel is modeled as an ideal EPR4 channel, and a channel decoder is used to generate soft information. In the first approach, the channel decoder is applied only once, and in the second approach, soft information is iterated between the channel decoder and the decoder for the LDPC or turbo code. This second approach is denoted “with channel iteration,” and is equivalent to “turbo equalization.”
Figure 3.16: Performance of turbo and LDPC codes on an ideal EPR4 channel

The parallel concatenated turbo code consists of two concatenated convolutional codes separating by a random permuter (known as an interleaver). Both the configuration with and without channel iteration are considered. A serial concatenated turbo code consists of a single convolutional code, which is concatenated with the channel decoder, so that the decoding involves iteration with the channel. It can be seen that the parallel turbo code with iteration performs slightly worse than the serial turbo code, while the LDPC code with channel iteration performs as well as the serial concatenated turbo code. With both turbo and LDPC codes, iterating back and forth between the two decoders gives better performance than only applying the channel decoder once, giving a 1 dB improvement in performance.

3.4.3 Thermal asperities

A problem with many magnetic read heads is the occurrence of thermal asperities. As a rough approximation, it is possible to model this phenomenon by an extended burst of erased samples, so that a block of received samples (the input to the channel
Figure 3.17: AWGN channel with thermal asperities (rate 8/9)

decoder) is set to 0. This corresponds to the use of some mechanism that detects that the thermal asperity erases a block of samples to avoid overloading the decoder with errors. Simulations were performed in which every sector consistently suffers a burst of erasures of a fixed length, corresponding to the thermal asperity. Figure 3.17 shows the performance on an AWGN channel. Both LDPC codes and parallel concatenated turbo codes are robust to erasures, showing only a modest degradation in performance even for a burst of 80 erasures.

On the other hand, for the EPR4 channel, parallel and LDPC codes perform about the same for thermal asperities, and there is severe degradation in performance for erasures of length 80, as seen in Figure 3.18. Finally, in Figure 3.19, the serially concatenated turbo scheme is shown to perform badly in the presence of thermal asperities, even of length 40. The single convolutional code appears to be more sensitive to erasures than the parallel turbo and LDPC codes.
Figure 3.18: EPR4 channel with thermal asperities, LDPC vs. parallel turbo code

Figure 3.19: EPR4 channel with thermal asperities, LDPC vs. serial turbo code
Chapter 4

Constrained coding for soft decoders

As shown in Chapter 3, soft iterative decoding techniques perform extremely well, and it is desirable to apply them to communication systems such as magnetic and optical recording. In these recording channels, however, the use of constrained coding poses a potential problem for the soft decoder. In this chapter, we consider methods for making the soft iterative decoders compatible with the modulation constraint. Section 4.1 considers some configurations for combined constraint and ECC decoding, including modified concatenation. Section 4.2 discusses the construction of modulation codes so that soft information is available after demodulation. Section 4.3 uses the forward-backward algorithm on the trellis of the constraint graph to decode the sequence using the knowledge of the constraint to improve performance. Finally, Section 4.4 briefly discusses how to apply these techniques when iterating with the channel decoder.

4.1 Soft decoding and constrained coding

Suppose that soft information is available for the channel bits, perhaps as the output of a soft channel decoder. In the standard concatenation of a soft ECC code with a modulation code, it is necessary to use a soft demodulation code, a modulation code
where the demodulation step converts soft channel information into soft information about the demodulated output and passes it to the soft ECC decoder. Constructions of soft demodulation codes are discussed in Section 4.2.

We consider a number of different methods, and focus on the example of the \((0,k)\)-RLL constraint, which is of special interest in magnetic recording, where it is commonly used for timing recovery.

4.1.1 Standard concatenation

As discussed in Section 4.2, it is possible to calculate the post-demodulation soft information for any block modulation code. For more accurate and simpler expressions, however, it is desirable to use modulation codes that are specially constructed to pass soft information. Various methods, such as systematic modulation codes, are discussed in Section 4.2. Soft demodulation codes can be applied to the standard concatenation method to allow the soft ECC decoders to obtain accurate soft information. The primary drawback is the lower rate of these soft demodulation codes, as compared with usual modulation codes that approach the capacity of the constraint.

In the case of the \((0,k)\) constraint, it is possible to construct a systematic modulation code as a rate \(\frac{k}{k+1}\) block code by simply appending a 1 to the existing \(k\) unconstrained bits. The soft demodulation step then simply consists of removing every \((k+1)\)-th probability. Table 4.1 shows that this systematic modulation rate \(\frac{k}{k+1}\) is lower than the capacity of the \((0,k)\) constraint.

<table>
<thead>
<tr>
<th>(k)</th>
<th>(\frac{k}{k+1})</th>
<th>(\text{cap } (0,k))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>0.6942</td>
</tr>
<tr>
<td>2</td>
<td>0.6667</td>
<td>0.8791</td>
</tr>
<tr>
<td>3</td>
<td>0.75</td>
<td>0.9468</td>
</tr>
<tr>
<td>4</td>
<td>0.8</td>
<td>0.9752</td>
</tr>
</tbody>
</table>

Table 4.1: Comparison of systematic modulation and capacity for the \((0,k)\) constraint.
4.1.2 Modified concatenation

It turns out that the modified concatenation technique introduced in Chapter 2 for error-propagation can also facilitate the use of soft decoding in conjunction with a modulation constraint. As seen in Figure 4.1, the order of the modulation code and ECC decoder is reversed, and there is a high rate modulation code \( C_1 \) for the message bits, and a soft demodulation code \( C_2 \) for the parity bits. Soft information about the parity bits can then be calculated via soft demodulation, while the soft information about the message bits can be accessed directly by the ECC decoder. This results in a system in which the complete soft information is available to the soft ECC decoder, at a overall modulation rate that is close to capacity of the constraint. In addition, Section 4.3 presents a soft decoder for the constraint using the forward-backward algorithm. Iterating between the constraint decoder and the ECC decoder can lead to improved performance.

In the case of the \((0, k)\) constraint, it is possible to use a modulation code \( C_1 \) of arbitrary length that achieves a rate close to the capacity of the constraint. For the modulation code \( C_2 \), it is possible to use a systematic modulation code. If the ECC has high rate, then the overall code has a modulation rate that is close to the rate(\(C_1\)).

4.1.3 Bit insertion technique

A variation on modified concatenation is to insert the parity bits into the modulated message sequence in such a way that does not violate the constraint. This bit insertion technique, shown in Figure 4.1 entails choosing a modulation code \( C_1 \) such that inserting random bits into the sequence periodically still meets the target constraint. It is then possible for the decoder to obtain direct reliability for all these bits from the channel decoder. Figure 4.2 illustrates the bit insertion technique, as a comparison with Figure 2.4 (p. 15). This idea has been used by Wijngaarden and Immink [WI98] for reducing error-propagation, and by Anim-Appiah and McLaughlin [AM99a] for using \((0, k)\) modulation codes with turbo codes.

In the case of the \((0, k)\)-RLL constraint, a tighter \((0, k - a)\)-RLL is used for the
Figure 4.1: Configurations for using soft information with a modulation constraint

Figure 4.2: Bit insertion technique
code $C_1$, and parity bits are inserted periodically into the modulated sequence in such a way that does not violate the $(0,k)$ constraint. Starting with a $k_0 = k - a$ constraint, a parity bit is inserted in between sets of $L$ message bits to obtain a sequence which satisfies the $(0,k)$ constraint. Then the number $L$ must satisfy $k - a + \lceil \frac{k-a+1}{a} \rceil \leq k$. The number $L$ is chosen to be its minimum value $L = \lceil \frac{k-a+1}{a} \rceil$ in order to accommodate as many parity bits as possible for the given starting constraint $(0,k-a)$. For a given value of $L$, the insertion rate is $\frac{L}{L+1}$, where $L$ modulated message bits alternate with a single inserted parity bit.

Starting with $m$ message bits, modulating to the $(0,k-a)$ constraint results in approximately $\frac{m}{\text{rate}(0,k-a)}$ bits, where it is assumed that the modulation code has a rate near capacity. Then the number of parity bits out of the ECC is

$$\frac{m}{\text{rate}(0,k-a)} \left( \frac{1}{\text{rate}(\text{ECC})} - 1 \right).$$

The total number of bits that can be inserted is $\frac{m'}{L}$, where $m' = \frac{m}{\text{rate}(0,k-a)}$, so the number of parity bits is limited by $\frac{m'}{L}$, giving a lower bound on the allowable rate of the ECC.

$$\left( \frac{1}{\text{rate}(\text{ECC})} - 1 \right) m' \leq \frac{m'}{L} \quad \text{rate}(\text{ECC}) \geq \frac{L}{L+1}$$

(4.1)

The overall rate of the insertion scheme is given by $\text{rate}(0,k-a) \cdot \text{rate}(\text{ECC})$. Since the overall modulation rate is determined by $\text{rate}(0,k-a)$, it is desirable to choose small values of $a$, such as $a = 1$. On the other hand, the benefit of using larger $a$ is to decrease the lower bound $\frac{L}{L+1}$ on the rate of the ECC.

### 4.1.4 Comparison for $(0,k)$ constraint

Several methods have been presented for incorporating the $(0,k)$-RLL constraint into a system that uses soft decoding:

- Standard concatenation with a systematic modulation code
• Modified concatenation with a systematic modulation code $C_2$

• Bit insertion technique

It should be noted that another construction of $(0,k)$ block codes is given in [AM99b]. When $k = 2^l$ is even, there exists a rate $\frac{l}{l+1}$ block code that simply consists of all $2^l$ words of length $l + 1$ that have odd parity. Then the longest run of 0's is $k$, and this code has minimum distance $d_{\text{min}} = 2$, so that soft modulation can provide some coding gain. We do not consider this code in this analysis, however, due to its lower rate $\frac{l}{l+1} = \frac{k}{k+2}$, which increases the modulation overhead.

We make a comparison between modified concatenation (with a systematic code for $C_2$) and the bit insertion scheme. In modified concatenation, the $m$ user bits are first modulated with a near-capacity code $C_1$ (so $\text{cap}(0,k) \approx \text{rate}(0,k)$). Then the systematic ECC encoder produces $p = \frac{1}{\text{rate}(0,k)} \left( \frac{1}{\text{rate}(\text{ECC})} - 1 \right)$ parity bits. Then applying the systematic modulation code, which has rate $\frac{k}{k+1}$, gives a total of

$$\frac{1}{\text{rate}(0,k)} m + \frac{k+1}{k} p = \left( 1 + \frac{k+1}{k} \left( \frac{1}{\text{rate}(\text{ECC})} - 1 \right) \right) \frac{m}{\text{rate}(0,k)}$$

bits.

In comparison, the bit insertion scheme results in

$$\frac{m}{\text{rate}(\text{ECC}) \cdot \text{rate}(0,k-a)}$$

bits.

Comparing these two quantities, it can be seen that bit insertion requires fewer redundancy bits whenever

$$\frac{m}{\text{rate}(\text{ECC}) \cdot \text{rate}(0,k-a)} \leq \left( 1 + \frac{k+1}{k} \left( \frac{1}{\text{rate}(\text{ECC})} - 1 \right) \right) \frac{m}{\text{rate}(0,k)},$$

which simplifies to

$$\text{rate}(\text{ECC}) \leq 1 - \left( \frac{\text{rate}(0,k)}{\text{rate}(0,k-a)} - 1 \right) k.$$  \hspace{1cm} (4.2)

This expression gives an upper bound on the rate(ECC) for the use of bit-insertion scheme for the $(0,k)$ constraint. For ECC rates exceeding this bound, it is more
efficient to use the systematic modulation code instead of insertion.

Hence for each combination of $k$ and $a$, there is both a lower bound (4.1), given by the feasibility of the insertion code, on the rate of the ECC, and an upper bound (4.2), based on the comparison with systematic modulation codes. Table 4.2 gives the bounds on rate($ECC$) for insertion schemes that satisfy a $(0, k)$ constraint by using a $(0, k - a)$ modulation code for $C_1$. The results in Table 4.2 are plotted in Figure 4.3, where the colored regions show the ranges of value for the rate of the ECC where bit insertion requires less redundancy than systematic modulation.

It is interesting to note that for low values of $k$, the lower bound exceeds the upper bound, which means that the systematic modulation code will always perform better than insertion. In general, the insertion method becomes more appropriate for larger values of $k$, and the use of $a > 1$ allows bit insertion to be used at lower ECC rates.

<table>
<thead>
<tr>
<th>$k$</th>
<th>$a$</th>
<th>$\text{cap}(0, k - a)$</th>
<th>lower</th>
<th>upper</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>0.6942</td>
<td>0.75</td>
<td>0.7690</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0.8791</td>
<td>0.8</td>
<td>0.8797</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0.9468</td>
<td>0.8333</td>
<td>0.9339</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>0.8791</td>
<td>0.6667</td>
<td>0.7817</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0.9752</td>
<td>0.8571</td>
<td>0.9631</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>0.9468</td>
<td>0.75</td>
<td>0.8833</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0.9881</td>
<td>0.875</td>
<td>0.9793</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>0.9752</td>
<td>0.75</td>
<td>0.9361</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0.9942</td>
<td>0.8889</td>
<td>0.9884</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>0.9881</td>
<td>0.8</td>
<td>0.9647</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>0.9752</td>
<td>0.6667</td>
<td>0.9152</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0.9971</td>
<td>0.9</td>
<td>0.9936</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>0.9942</td>
<td>0.8</td>
<td>0.9805</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>0.9881</td>
<td>0.75</td>
<td>0.9538</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0.9993</td>
<td>0.9091</td>
<td>0.9964</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>0.9971</td>
<td>0.8333</td>
<td>0.9893</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>0.9942</td>
<td>0.75</td>
<td>0.9748</td>
</tr>
</tbody>
</table>

Table 4.2: Lower and upper bounds for use of bit insertion
4.1.5 Combined modulation and coding

Taking a broader perspective, it is possible to view modified concatenation as a way of creating a single channel code that combines error-correction and modulation coding. Modified concatenation can be seen as a generalization of the various schemes presented in [Bas68][HHW91][Kau65][PK92][Per95] that combine the \((d, k)\) run-length constraint with an error-control code that corrects single bit errors or bit shifts.

Following the discussion in [MRS99], it is relevant to consider some theoretical bounds on the minimum distance of codes that satisfy a constraint. The Gilbert-Varshamov bound gives a lower bound of the number of codewords in a code with Hamming distance \(d_{\text{min}}\). (Note that this \(d\) is unrelated to \((d, k)\)-RLL constraints).

**Theorem 3** (Gilbert-Varshamov) There exists a code with

\[
M \geq \frac{|\Sigma|^n}{V_{\Sigma^n} (d_{\text{min}} - 1)}
\]
where \( d_{\text{min}} \) is the Hamming distance, \( \Sigma \) is the symbol set, \( n \) is the codeword length, and \( V_{\Sigma^n}(d_{\text{min}} - 1) \) is the volume of a ball of radius \( d_{\text{min}} - 1 \).

This can be generalized to constrained systems as follows. For the constrained system \( S \), define the average volume of a sphere of radius \( r \).

\[
V_X(r) = \frac{1}{|X|} \sum_{w \in X} |B(w; r) \cap X|
\]

It is possible to view modified concatenation as a method for creating a combined code for constrained coding and ECC. There exists an improved Gilbert-Varshamov bound on the rate of a combined code.

**Theorem 4** (Gu and Fuja) There exists a code with

\[
M \geq \frac{|X|}{V_X(d_{\text{min}} - 1)}
\]

where \( d_{\text{min}} \) is the Hamming distance, \( X \) is the set of constrained codewords, \( V_X(d_{\text{min}} - 1) \) is the average number of constrained words in a ball of radius \( d_{\text{min}} - 1 \).

If the rate of the constraint is \( \text{rate}(C) \), and the rate of the code is \( \text{rate}(\text{ECC}) \), then \( |X| = |\Sigma|^{n\cdot\text{rate}(C)} \), and \( M = |\Sigma|^{n\cdot\text{rate}(\text{ECC})\cdot\text{rate}(C)} \), where it is assumed that the combined rate is roughly the product of the rates. Then Theorem 4 can be rewritten as

\[
|\Sigma|^{n\cdot(\text{rate}(\text{ECC}) - 1)\cdot\text{rate}(C)} \geq \frac{1}{V_X(d_{\text{min}} - 1)}
\]

\[
\text{rate}(\text{ECC}) \geq 1 - \frac{1}{n \cdot \text{rate}(C)} \log_2 V_X(d_{\text{min}} - 1)
\]

saying that there exists an ECC whose rate exceeds the quantity on the right.

It should also be noted that while the performance of a maximum likelihood decoder is usually determined by the minimum distance, for certain codes, it is still possible to have very good performance despite a low minimum distance. This is
because the probability that a word gets mistaken for another word is given by

\[
\Pr(\text{misdecode}) \approx \sum_{d=1} \left( \sum_{x \neq x_0} [d(x, x_0) = d] \right) \Pr(\text{misdecode} | \text{distance } d)
\]

where the first part counts the number of neighbors of Hamming distance \(d\), and the second part gives the probability that the decoder makes a mistake and decodes to another codeword of distance \(d\) bits away. The notation \([a = b]\) is an indicator function.

\[
[a = b] = \begin{cases} 
1 & \text{if } a = b \\
0 & \text{if } a \neq b
\end{cases}
\]

If the number of minimum distance pairs (i.e. the “coefficient” in the error expression) is small, it is still possible to achieve good performance with the code.

### 4.2 Soft demodulation codes

First a general expression is given for finding the post-demodulation soft information from an arbitrary block modulation code. In this section, various constructions are presented for *soft demodulation codes*, such as systematic modulation codes. These soft demodulation codes have a rate lower than the capacity of the constraint, since they are constrained to have the special property that the post-demodulation soft information is easily calculated. Since \((d, k)\) constraints and \((0, C/I)\) constraints are commonly used families of RLL codes, explicit constructions for these constraints are presented. These examples may serve as a guide to constructing similar codes for other constraints. Finally, the general problem of constructing soft demodulation codes is considered.
4.2.1 General approach

For a block modulation code, suppose that the block encoder is a 1-1 function binary mapping:

\[ f : \{0, 1\}^K \rightarrow \{0, 1\}^N \]
\[ f : (x_0, x_1, \ldots, x_{K-1}) \mapsto (z_0, z_1, \ldots, z_{N-1}) . \]

In general, this can be a non-linear, non-systematic assignment. The demodulation unit receives probabilities \( q_i = \Pr(z_i = 1) \) on the modulated sequence from a soft channel decoder. It is then possible to translate these probabilities into post-demodulation probabilities \( p_i = \Pr(x_i = 1) \) by summing over all the possible codewords with \( x_i = 1 \), as in the following expression,

\[
\Pr(x_i = 1) = \sum_{\substack{x \in \{0, 1\}^K \\
\quad z_i = 1}} \Pr(f(x)) \\
= \sum_{\substack{z = f(x) \\
\quad x \in \{0, 1\}^K \\
\quad z_i = 1}} \left( \prod_j q_j^{[z_j=1]} (1 - q_j)^{[z_j=0]} \right) \tag{4.3}
\]

for \( i = 0, 1, \ldots, K - 1 \). It is assumed that the probabilities \( q_j \) are independent. This approach is considered in [AM99b].

In general, this is a complicated expression that requires summing up roughly \( 2^{K-1} \) expressions. In addition to the complexity, this approach also can lead to uncertainty in the final answer. In particular, consider a modulation code consisting of \( 2^K \) words of length \( N \), randomly chosen from a uniform distribution. Then suppose the probabilities \( q_j \) have uncertainty \( \epsilon \), so that

\[ q_j = \begin{cases} 
\epsilon & \text{if } z_j = 0 \\
1 - \epsilon & \text{if } z_j = 1 
\end{cases} \]

Suppose that the constrained word \( z \) corresponds to an unconstrained word \( x \). Then
the error in the probability \( p_i \) is lower bounded as follows,

\[
|p_i - x_i| > (1 - \epsilon)^N + (2^{K-1} - 1) \epsilon (1 - \epsilon)^{N-1}.
\]

As can be seen, as the block length increases, the uncertainty in the output probabilities increases.

### 4.2.2 Systematic modulation code

A simple construction is the systematic modulation code, in which the encoder takes the sequence and inserts bits in a predetermined pattern in order to create a constrained sequence. Regardless of the input to the encoder, it is possible to choose the inserted bits so as to meet the constraint. The soft demodulation of this code simply consists of extracting the probabilities corresponding to the systematic bits, and throws away the other probabilities. Some explicit constructions of some systematic modulation codes for the \((d, k)\) constraints are presented. Similar constructions for the \((0, G/I)\) constraint in Appendix B.

**\((d, k)\) constraint**

For \((0, k)\) constraints, a systematic modulation code of rate \( \frac{k}{k+1} \) is obtained by alternating a 1 with \( k \) unconstrained bits. For \((d, k)\) constraints, with \( d \geq 1 \) and \( k \geq 2d+1 \), the rate \( \frac{1}{d+1} \) can be achieved by separating each user bit by exactly \( d \) zeroes. It can be shown that these are the maximum possible rates for these constraints by considering
CHAPTER 4. CONstrained CODING FOR SOFT DECODERS

<table>
<thead>
<tr>
<th>$d \backslash k$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0/1</td>
<td>0/2</td>
<td>0/3</td>
<td>0/4</td>
<td>0/5</td>
<td>0/6</td>
<td>0/7</td>
<td>0/8</td>
<td>0/9</td>
</tr>
<tr>
<td>1</td>
<td>0/1</td>
<td>0/2</td>
<td>0/3</td>
<td>0/4</td>
<td>0/5</td>
<td>0/6</td>
<td>0/7</td>
<td>0/8</td>
<td>0/9</td>
</tr>
<tr>
<td>2</td>
<td>0/0</td>
<td>0/5</td>
<td>0/1</td>
<td>0/2</td>
<td>0/5</td>
<td>0/1</td>
<td>0/2</td>
<td>0/5</td>
<td>0/1</td>
</tr>
<tr>
<td>3</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
</tr>
</tbody>
</table>

Table 4.3: Rates for systematic $(d, k)$ modulation codes

<table>
<thead>
<tr>
<th>$(d, k)$ Constraint</th>
<th>Systematic Mod. Rate</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,2)</td>
<td>$\frac{2}{3} = 0.6667$</td>
<td>0.8791</td>
</tr>
<tr>
<td>(0,4)</td>
<td>$\frac{2}{3} = 0.8$</td>
<td>0.9752</td>
</tr>
<tr>
<td>(0,7)</td>
<td>$\frac{3}{5} = 0.875$</td>
<td>0.9971</td>
</tr>
<tr>
<td>(1,3)</td>
<td>$\frac{1}{3} = 0.5$</td>
<td>0.5515</td>
</tr>
<tr>
<td>(1,7)</td>
<td>$\frac{1}{5} = 0.2$</td>
<td>0.6793</td>
</tr>
<tr>
<td>(2,7)</td>
<td>$\frac{1}{3} = 0.3333$</td>
<td>0.5174</td>
</tr>
<tr>
<td>(3,7)</td>
<td>$\frac{1}{4} = 0.25$</td>
<td>0.4057</td>
</tr>
</tbody>
</table>

Table 4.4: Systematic modulation rate vs. capacity

The worst cases: For the $(0, k)$ constraint, if the user input sequence is all zeroes, then there needs to be a 1 at least every $k + 1$ bits. For the $(d, k)$ constraint with $d > 0$ and $k \geq 2d + 1$, if the user sequence consists of all ones, then the ones must be separated by $d$ zeroes. Table 4.3 lists the maximum systematic modulation rate for small values of $d$ and $k$.

Table 4.4 shows that these modulation code rates are less than the capacity of the constraint, but can be high enough to be considered for practical use as code $C_2$ for the parity symbols in the modified concatenation scheme.

4.2.3 Sliding-block codes

Another approach is to create sliding-block codes with extremely short block lengths (with inputs equal to 1 bit or so). Then it is possible to decode these modulation codes using the forward-backward algorithm on a trellis to obtain post-demodulation soft information. Ashley and Marcus [AM99] describe the use of the state-splitting algorithm for constructing time-varying sliding block modulation codes. Barbero
CHAPTER 4. CONSTRUED CODING FOR SOFT DECODERS

and Yterhus [BY99] also construct such bit-oriented trellis using state-splitting, and describe how to use such bit-oriented runlength trellises for decoding.

The main idea is to have a periodic time-varying modulation code, which cycles through \( l \) encoders, so that at the \( i \)-th stage the encoder has rate \( K_i/N_i \). It is shown in [AM99] that for \( K = K_0 + K_1 + \cdots + K_{l-1} \) and \( N = N_0 + N_1 + \cdots + N_{l-1} \), where the rate \( K/N \) is less than the capacity of the constraint, it is possible to create a time-varying sliding-block modulation code, using a specialized version of the state-splitting algorithm. To apply this method for soft demodulation codes, the input block lengths \( K_i \) are chosen to be very small, either 0 or 1. The resulting construction, however, may involve non-zero memory \( m \) and anticipation \( a \) in the sliding-block code. At each time \( i \), the encoder (which is a finite-state machine) takes \( K_i \) user bits \( u_i = (u_i^{(0)}, u_i^{(1)}, \ldots, u_i^{(K_i-1)}) \) and outputs \( N_i \) coded bits \( x_i = (x_i^{(0)}, x_i^{(1)}, \ldots, x_i^{(N_i-1)}) \).

There is a sliding-block decoder that depends on a sliding-block window whose size is determined by the amount of memory and anticipation.

In general, there are several ways to obtain soft information on the output of this sliding block code from the input. The first is similar to (4.3), and involves summing over all the possible words that are consistent with each bit. The second is to create a trellis corresponding to this decoder and apply the forward-backward algorithm.

- **Sliding window** - Construct a table of all possible values of the \( N_{i-1} + N_i + N_{i+1} \) bits \( (x_{i-1}, x_i, x_{i+1}) \), and then consider which combinations are consistent with each configuration of the \( K_i \) uncoded bits \( u_i \). Then the corresponding probabilities on the constrained words are summed to obtain the probabilities for each of the \( 2^{K_i} \) configurations for \( u_i \).

- **Forward-backward algorithm for demodulation** - At each time \( i \), the number of states in the trellis depends on the number of states in the finite state encoder. The edges in the trellis are labeled by an input label \( u_i \) corresponding to uncoded bits, and an output label \( x_i \). The forward-backward algorithm is applied to this trellis to obtain updated probabilities on the states, giving extrinsic information on the \( K_i \) uncoded bits \( u_i \) for the demodulated output.
CHAPTER 4. CONSTRAINED CODING FOR SOFT DECODERS

This general framework can be used to put our previous approaches into perspective. Systematic modulation codes are time-varying codes where each stage has $K_i = 0$ or 1. In the case of $K_i = 1$ (corresponding to a systematic bit), then $N_i = 1$. In the case of $K_i = 0$ (corresponding to the inserted bits), then $N_i$ can be arbitrary. There may be non-zero memory and anticipation. For any systematic modulation code, then, there also exists a trellis decoder based on the forward-backward algorithm, but this decoder is usually not used because of the excessive complexity and the time-varying structure.

4.2.4 Example of MFM code

The MFM code can be used as an example of soft demodulation. The MFM code is a sliding block modulation code of rate $\frac{1}{2}$ that satisfies the $(1, 3)$ constraint. The encoder for this code makes the following assignments

$$
0 \rightarrow x_0 \\
1 \rightarrow 01
$$

where $x$ is the complement of the previous bit. The input bits are labelled by $u_i$, and the output by $x_i^{(0)}$ and $x_i^{(1)}$. The modulated sequence looks like

$$
\ldots, x_{i-1}^{(0)}, x_{i-1}^{(1)}, x_i^{(0)}, x_i^{(1)}, x_{i+1}^{(0)}, x_{i+1}^{(1)}, \ldots
$$

where

$$
x_i^{(1)} = u_i \\
x_i^{(0)} = \begin{cases} 
1 - u_{i-1} & \text{if } u_i = 0 \\
0 & \text{if } u_i = 1
\end{cases}
$$

Three ways to decode this MFM code are as follows:

- **Systematic modulation code** - As a systematic modulation code, the MFM code is a time-varying code with period 2, with rates $K_0/N_0 = 0/1$ and $K_1/N_1 =$
CHAPTER 4. CONstrained CODING FOR SOFT DECODERS

1/1. The anticipation is zero, and the memory $m_0$ for stage 0 is 1 bit, while the memory $m_1$ for stage 1 is 0 bits. The MFM code can be decoded simply by ignoring every other probability.

- **FBA decoder for trellis** - In the case of the MFM code, the trellis has 3 states, corresponding to trellis outputs of 00, 10 and 01, as pictured in Figure 4.5. The forward-backward algorithm can be applied to obtain soft estimates on the demodulated output.

- **Sliding-block window** - As a sliding-block code, the MFM code is a time-invariant code with $K = 1$ and $N = 2$, with memory of 1 bit. The probability $u_i$ can be estimated from a window of 3 bits, specifically $x_i^{(0)}$, $x_i^{(1)}$, and $x_{i+1}^{(0)}$. This table gives the configurations for these 3 bits that are consistent with the code.

<table>
<thead>
<tr>
<th>$x_i^{(0)}$</th>
<th>$x_i^{(1)}$</th>
<th>$x_{i+1}^{(0)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Then the probabilities can be estimated by

$$P(u_i = 1) = \frac{(1 - p_i^{(0)}) p_i^{(1)} (1 - p_{i+1}^{(0)})}{(1 - p_i^{(1)}) + (1 - p_i^{(0)}) p_i^{(1)} (1 - p_{i+1}^{(0)})}$$

where $p_i^{(a)} = P(x_i^{(a)} = 1)$ for $a = 0, 1$.

As a systematic modulation code, the MFM code is a time-varying code with period 2, with rates $K_0/N_0 = 0/1$ and $K_1/N_1 = 1/1$. The anticipation is zero, and the memory $m_0$ for stage 0 is 1 bit, while the memory $m_1$ for stage 1 is 0 bits.

For reference, these three approaches are compared in Figure 4.6, where the bit error rate performance (after demodulation) is plotted as a function of $\frac{1}{\sigma^2}$ for an
AWGN channel. As expected, the BCJR algorithm performs the best (since it is optimal), followed by the sliding-window decoder, and the systematic demodulation approach.

4.3 Soft constraint decoder

Modified concatenation allows the soft ECC to obtain soft reliability information even when constrained codes are used. This section introduces a soft constraint decoder that updates the probabilities for the modulated message bits using the modulation constraint. This decoder is an application of the forward-backward algorithm on a trellis generated from the constraint graph. This decoder module can be inserted into the soft iterative decoding algorithm, and information is iterated between the ECC and the constraint decoder for improved performance. The construction of the constraint decoder modules is discussed for the case of the \((d, k)\)-RLL constraints. A related result on turbo codes applied to a Markov source appears in [GFV98].

The state transition diagram for the constraint, or constraint graph, can be transformed into a trellis, as shown in Figure 4.7. For the case of the \((d, k)\) constraint, where \(k < \infty\), it is possible to associate each bit \(x_i\) with a state variable \(s_i\) that has \(k + 1\) possible states, with values 0, 1, ..., \(k\), corresponding to the number of consecutive zeros which preceded the current \((i\text{-th})\) bit. For the \((d, \infty)\) constraint, it is
possible to construct a similar trellis with only $d + 1$ states.

The forward-backward algorithm can then be applied to yield a soft-in, soft-out decoder for the constraint. When incorporated into a soft iterative decoding scheme using modified concatenation, this soft constraint decoder yields additional coding gain on top of the gain from the ECC decoder. It is important to note the differences between this soft constraint decoder and the soft demodulation codes discussed earlier: While soft demodulation yields soft information on the demodulated output, the soft constraint decoder updates soft information on the constrained sequence itself. While soft demodulation codes depend on the encoding method, the soft constraint decoder is based on the constraint graph, and hence is independent of the encoding. Finally, in the modified concatenation scheme, soft demodulation is used for the parity bits, while the soft constraint decoder is applied to the modulated message bits. A summary of the differences is listed in Table 4.5.
CHAPTER 4. CONSTRAINED CODING FOR SOFT DECODERS

<table>
<thead>
<tr>
<th>Applied to</th>
<th>Soft demodulation</th>
<th>Soft constraint decoder</th>
</tr>
</thead>
<tbody>
<tr>
<td>parity bits</td>
<td>modulated message bits</td>
<td></td>
</tr>
<tr>
<td>Decoder based on</td>
<td>encoder trellis</td>
<td>constraint trellis</td>
</tr>
<tr>
<td>Soft output for</td>
<td>demodulated parity</td>
<td>modulated message</td>
</tr>
</tbody>
</table>

Table 4.5: Comparison of soft demodulation and the soft constraint decoder

4.3.1 Forward-backward algorithm for the constraint trellis

This decoder for the constraint is another application of the forward-backward algorithm in Section 3.3.

Derivation

For completeness, the details of forward-backward algorithm (ref. Section 3.3) for the constraint decoder are presented. Let $b_i$ denote the current bit, and $b^- = \{..., b_{i-2}, b_{i-1}\}$ and $b^+ = \{b_{i+1}, b_{i+2}, ...\}$ represent the past and the future bits. Let $s_i$ be the state at time $i$, and $s_{i+1}$ be the state at time $i + 1$. Then the joint probability can be factored as follows.

$$P(s_i, s_{i+1}, b^-, b_i, b^+) = P(s_i, b^-) P(s_{i+1} | s_i) P(b_i | s_i, s_{i+1}) P(b^+ | s_{i+1})$$

The following notation can be used to represent the terms in this expression.

$$\alpha_i(s_i) = P(s_i, b^-)$$
$$\beta_{i+1}(s_{i+1}) = P(b^+ | s_{i+1})$$
$$\gamma_i(b_i, s_i, s_{i+1}) = P(b_i | s_i, s_{i+1})$$
$$Q_{s_i, s_{i+1}} = P(s_{i+1} | s_i)$$

The $\alpha$'s and $\beta$'s correspond to the probability distribution on the states based on the past and future information about the bits, respectively. The two states $s_i$ and $s_{i+1}$ specify a particular edge, which has some binary label $x(s_i, s_{i+1})$. Then the term $\gamma_i(b_i, s_i, s_{i+1})$ corresponds to the prior probability that the $i$-th bit is equal to $x(s_i, s_{i+1})$. 
Finally, \( Q_{s_i,s_{i+1}} \) is the state transition probability, which can be determined from the maxentropic distribution of the state probabilities, using Perron-Frobenius theory (ref. [MRS99]). In particular, let \( \mu(s) \) denote the \( s \)-th entry of the (right) eigenvector corresponding to the largest eigenvalue \( \lambda \) of the state transition matrix for the constraint. Then the state transition \( Q_{s_i,s_{i+1}} = P(s_{i+1} | s_i) \) is given by \( \frac{\mu(s_{i+1})}{\mu(s_i)} \) whenever there is a transition from \( s_i \) to \( s_{i+1} \), and is zero otherwise. Note that this derivation assumes the encoding method is efficient and encodes at a rate which is close to the capacity of the constraint.

Recursive formulas can be derived for the \( \alpha \)'s and \( \beta \)'s, giving the forward and backwards decoding passes in the BCJR algorithm,

\[
\alpha_{i+1}(s_{i+1}) = \sum_{s'_i} \alpha_i(s'_i) \gamma_i(b_i, s'_i, s_{i+1}) Q_{s_i,s_{i+1}} \\
\beta_i(s_i) = \sum_{s'_{i+1}} \beta_{i+1}(s'_{i+1}) \gamma_i(b_i, s_i, s'_{i+1}) Q_{s_i,s'_{i+1}}
\]

where the summation is over all states where there exists an edge.

The extrinsic probabilities are obtained by summing over all combinations corresponding to the bit value \( a \), excluding the prior information \( \gamma_i(b_i, s_i, s_{i+1}) = P_{\text{prior}}(b_i) \),

\[
p_i^{\text{extrinsic}}(a) = c_i \cdot \sum_{w(s_i,s_{i+1})=a} \alpha_i(s_i) \beta_{i+1}(s_{i+1}) Q_{s_i,s_{i+1}}
\]

where \( p_i^{\text{extrinsic}}(a) \) is the extrinsic probability that \( x_i = a \), and \( c_i \) is a normalization constant.

\((d,k)\) constraint

An explicit construction for the forward-backward algorithm is given for the \((d,k)\) constraint, where \( k < \infty \). It is possible to associate each bit \( x_i \) with a state variable node \( s_i \) which has \( k+1 \) possible states, with values \( 0, 1, ..., k \), corresponding to the number of consecutive zeros that precede the current \((i\text{-th})\) bit. The trellis diagram for this constraint is shown in Figure 4.7.
The forward-backward algorithm then proceeds as follows, where $Q_{s,s+1}$ are the transition probabilities for the $(d,k)$ constraint given by Perron-Frobenius theory.

$$
\alpha_{i+1}(s) = \begin{cases} 
\sum_{r=d}^{k} \alpha_i(r) \gamma_i(0) Q_{r,s} & \text{for } s = 0 \\
\alpha_i(s-1) \gamma_i(1) Q_{s-1,s} & \text{for } s \neq 0 
\end{cases}
$$

$$
\beta_i(s) = \begin{cases} 
\beta_{i+1}(s+1) \gamma_i(0) Q_{s,s+1} & \text{for } 0 \leq s < d \\
\beta_{i+1}(s+1) \gamma_i(0) Q_{s,s+1} + \beta_{i+1}(0) \gamma_i(1) Q_{s,0} & \text{for } d \leq s < k \\
\beta_{i+1}(0) \gamma_i(1) Q_{s,0} & \text{for } s = k 
\end{cases}
$$

To sum out for the extrinsic information, compute

$$
p_i^{\text{extrinsic}}(0) = c_i \sum_{s=0}^{k-1} \alpha_i(s) \beta_{i+1}(s+1) Q_{s,s+1}
$$

$$
p_i^{\text{extrinsic}}(1) = c_i \sum_{s=d}^{k} \alpha_i(s) \beta_{i+1}(0) Q_{s,0}
$$

where $c_i$ is a normalization constant.

It should be noted that it is possible to set up a similar soft decoder structure for
the \((0, G/I)\) constraints (ref. B), with a soft decoder for the global \((0, G)\) constraint, as well as two \((0, I)\) decoders for the two interleaves.

### 4.3.2 Combined ECC and constraint decoder

As pictured in Figure 4.8, it is possible to incorporate this soft constraint decoder based on the FBA into the overall iterative decoding scheme, with soft information passed between the soft ECC decoder and the constraint decoder. The soft constraint decoder helps the ECC decoder avoid words which violate the constraint by updating the probabilities to reflect knowledge of the constraint. In this way, the decoders cooperate to produce a joint decoding of the constraint and ECC code that approximates the performance of a joint maximum-likelihood decoder but with less complexity. (An alternative approach in [AM99a] is to combine the trellis for the RLL constraint with the trellis for a constituent convolutional code, and perform decoding of the product trellis.)

![Diagram](image)

**Figure 4.8: Iterating with the soft constraint decoder**

The precise configuration of the combined decoding scheme is more easily described using the log-likelihood ratios \(LLR(x_i) = \log \frac{P_{i|1}}{1 - P_{i|1}}\), following the principles for combining decoder modules as outlined in §3.1.3. The channel decoder outputs
CHAPTER 4. CONstrained CODING FOR SOFT DECODERS

a log-likelihood ratio for each modulated message bit, which serves as prior information for the ECC decoder. Meanwhile, the modulated parity bits are demodulated using the $C_2$, and the LLRs for the parity bits are also input as prior information to the ECC decoder. Applying one iteration of the ECC decoder produces extrinsic information which excludes the prior information that was input to the decoder. The extrinsic LLRs from the ECC decoder are then summed with the channel decoder to obtain the prior LLRs for the soft constraint decoder (for the modulated message bits). Similarly, the extrinsic LLRs of the constraint decoder are summed with the channel decoder LLRs to obtain the updated input to the ECC decoder for the next iteration. Finally, summing the output LLRs from all three decoders (channel, constraint and ECC) gives the joint posterior LLRs, which can be sliced to obtain the bit estimates, and then demodulated to obtain the decoded estimate of the user bits.

Simulations were performed with an LDPC code of rate $8/9$ and codeword length of 4896 bits. The code is defined by a regular parity check matrix with exactly $t_c = 4$ ones per column. A maximum of 20 iterations of the message-passing algorithm are performed for the decoding, but the decoding algorithm often stops before the maximum number of iterations is reached. Figure 4.9 compares the performance for various $(d,k)$ constraints. The channel used in the simulation has additive white Gaussian noise of variance $\sigma^2$, and no intersymbol interference. The $x$-axis of the plot is $1/\sigma^2$ in dB (rather than SNR), which provides a fair comparison with the base curve, corresponding to an LDPC code using modified concatenation without the soft constraint decoder. The results of the simulation can be summarized in Table 4.6.

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Capacity</th>
<th>Coding gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, 3)</td>
<td>0.9468</td>
<td>$\approx 0.3$ dB</td>
</tr>
<tr>
<td>(1, 7)</td>
<td>0.6793</td>
<td>$\approx 1.2$ dB</td>
</tr>
<tr>
<td>(1, 3)</td>
<td>0.5515</td>
<td>$\approx 2.1$ dB</td>
</tr>
</tbody>
</table>

Table 4.6: Coding gain from the soft constraint decoder

As the rate of the constrained code, the coding gain increases. An explanation is that the modulation code can be thought of as an error-correcting code so that more redundancy leads to more coding gain. The modulation code does not have
good distance properties, so these gains are significantly smaller than an equivalent ECC, but if a modulation constraint is required for other reasons, as is the case in many storage systems, then the soft constraint decoder provides a method to obtain some additional coding gain that requires only a small amount of added complexity. (In [RWM92], approximately a 2 dB gain is found for soft decoding of the (1,3) constraint in terms of the computational cut-off bound, which appears consistent with our simulation results.)

Figure 4.9: Performance gains from iterating with the soft constraint decoder

Finally, in Figure 4.10, the performance is considered for some parameters that are relevant to optical storage, in particular the optical Compact Disc. The block size of $N = 2336$, the coding rate of 0.8767 and a (2,10)-RLL constraint is used, corresponding to the EFM (Eight-to-Fourteen Modulation) code. The runlength constraints are dictated by physical parameters such as the minimum size of an optical pit. The channel is assumed to have no intersymbol interference, and the noise is AWGN.
CHAPTER 4. CONstrained CODING FOR SOFT DECODERS

Figure 4.10: LDPC and constraint decoder for an optical disc with a (2, 10)-RLL constraint

The usual error-correcting scheme (known as CIRC) in the Compact Disc is replaced by an LDPC code with the same redundancy, resulting in a gain of about 5 dB over the uncoded system. The CIRC is designed for correcting error bursts, not for Gaussian noise, so its performance is not plotted here. By including the soft constraint decoder with the LDPC decoder, there is an additional gain of approximately 2 dB over the LDPC decoder. Since the runlength constraints in optical recording require substantial overhead to implement (nearly half the rate is lost to modulation), the use of the soft constraint decoder to obtain coding gain from the RLL constraint can help to make up for the rate loss due to the RLL constraint. (It is also interesting to note that without the LDPC decoder, it is still possible to implement a soft constraint decoder, in which case the gain is minimum over the uncoded system, as shown in Figure 4.10.)
4.4 Iterating with the channel

The soft constraint decoder can be applied to channels without intersymbol interference (ISI), or where a channel decoder is only applied once for the channel. In a channel with ISI, it is desirable to design the system to incorporate the channel decoding into the iterative decoding algorithm for better performance. In other words, similar to “turbo equalization,” there should be increased performance from iterating soft information between the channel decoder, and the constraint and ECC decoder. If soft information is iterated between the channel and constraint decoders, however, a problem occurs because the channel ISI and constraint memory are aligned in time, resulting in a violation of the independence assumptions in the soft decoding algorithms. Equivalently, there are short cycle in the corresponding factor graph.

Hence, in order to iterate with the channel, it is better to consider a combined decoder for the constraint and channel. In the general case, the combined coder for constraint and channel would have a trellis where the states are product states consisting of a pair of a constraint state and a channel state. There are certain situations, however, when the constraint actually simplifies the decoding trellis. One example is the case of the $d = 1$ constraint, which forbids adjacent 1's in the NRZI sequence, as applied to the EPR4 channel. Incorporating the constraint into the channel decoder can be accomplished simply by removing edges and states from the trellis for the intersymbol interference channel. As shown in Figure 4.11, the forbidden states are removed from the EPR4 trellis, with $\nu = 3$ and $2^\nu = 8$ states, to give the combined trellis, with only 6 states.

In general, this combined channel and constraint trellis does not serve as a constraint demodulator. The constraint graphs is usually much simpler than the encoder graph (resulting from state-splitting). As a result, it is easier to build a decoder that operates directly on the constrained sequence, rather than one that produces information on the unconstrained sequence. As pictured in Figure 4.12, in standard concatenation, it is hence necessary to use a soft constraint demodulator as described in Section 4.2 to convert to soft information on the unconstrained bits. Moreover, for the channel iteration, it is necessary to take the extrinsic output from the ECC
decoder and convert that into soft information about the constrained bits using an additional "soft constraint encoder." While there may be ways to simplify this procedure, this leads to a cumbersome and inaccurate decoding process in general.

On the other hand, in modified concatenation, the output of the combined channel and constraint decoder can be directly used. Modified concatenation alleviates the need to perform soft constraint demodulation on the message bits, and hence is preferable when iterating with the channel decoder.

There are many possibilities for the design of combined channel and constraint decoders, especially for intersymbol interference channels that are equalized to partial response polynomials. Modulation constraints such as maximum transition run (MTR) constraints [KSS99][ISW98] have shown significant improvement for these partial response channels. Soft decoding of these constraints presents a promising area of investigation.
Figure 4.12: Iterating a constraint decoder with the channel
Chapter 5

Array codes

While low density parity check (LDPC) codes and turbo codes perform extremely well, as shown in Chapter 3, the performance is not guaranteed, so that for practical implementations, it is still necessary to use an algebraic error-correcting code such as a Reed-Solomon code, which corrects bursts and corrects a specific number of errors. These algebraic ECCs are generally hard-decision decoders, however, so that they are difficult to incorporate into the soft iterative decoding scheme. (One possibility is to use erasure decoding for Reed-Solomon codes, as in [BHK98].)

In this chapter, a type of algebraic error-correcting code known as an array code is shown to be suitable for decoding as a low-density parity check code. This structured construction has both a soft decoder as well as an algebraic decoder for correcting burst errors. Methods for avoiding short cycles and increase the minimum distance are discussed.

5.1 Array codes

There are many types of array codes [BR93, BFT98, BBV96], but they generally share the common properties of being algebraic error-correcting codes defined on an array that can be used for detecting and correcting error bursts. One construction of array codes [BR93] has an algebraic structure analogous to Reed-Solomon codes, with symbols that lie in rings rather than in Galois fields. The symbols can be very
large (on the order of hundreds or thousands of bits), and the code can correct one symbol error and multiple symbol erasures. The decoder for the array code, if given the locations of the erased symbol, can recover the values of the erased symbols. The algebraic decoders for array codes are not well-suited for correcting random bit errors but can be very useful in situations of long burst errors. Interestingly, this array code also turns out to have a sparse parity-check matrix, which is appropriate for decoding with the message-passing algorithm.

5.1.1 Parity check matrix for array codes

To define an array code, let \( p \) be an odd prime, and choose a \( p \times p \) binary array where each row satisfies an even parity check constraint. In addition, the diagonals (and slope 2 diagonals) also satisfy similar parity check constraints. In general, for any \( r \leq p \), the codewords of the array code consist of a \( p \times p \) binary array \( \{ a_{i,j} \} \) that satisfies the following \( rp \) sets of constraints: for \( 0 \leq k \leq r - 1 \) and \( 0 \leq i \leq p - 1 \),

\[
\bigoplus_j a_{(i+jk),j} = 0
\]

where the notation \( \langle b \rangle_p \) means the unique number between 0 and \( p - 1 \) inclusive such that \( \langle b \rangle_p \equiv b \mod p \). In other words, all the elements on a slope-\( k \) diagonal, which consists of the elements on the line obtained by moving \( k \) vertically for each step to the right (and wrapping around the array appropriately), sum to zero modulo 2. In addition, the last row is automatically set to be 0: for \( 0 \leq j \leq p - 1 \), the last element \( a_{p-1,j} \) is 0. The first \( r \) columns are the parity columns and the message columns are the last \( p - r \) columns.

Example 1 The following array is array code with \( r = 2 \), so that the rows have even parity, as well as the slope-1 diagonals. Note that the last row is all zero by
construction.

\[
A = \begin{bmatrix}
1 & 0 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

To consider the parity check matrix for this code, arrange the columns of the array into a single vector.

\[
A = \begin{bmatrix}
v_0 \\
v_1 \\
\vdots \\
v_{p-1}
\end{bmatrix} \Rightarrow \mathbf{v} = \begin{bmatrix}
v_0 \\
v_1 \\
\vdots \\
v_{p-1}
\end{bmatrix}
\]

Each vector \(v_i\) is a column vector of length \(p\), with the last bit \(v_{i,p-1}\) equal to zero. Then the corresponding parity matrix can be constructed as follows. Choose \(\sigma\) to be the permutation matrix that consists of a single cyclic shift. For example, for \(p = 5\),

\[
\sigma = \begin{bmatrix}
0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0
\end{bmatrix}
\]

Then the following is the parity check matrix for the array code, which has \(p^2\) columns and \(rp\) rows.

\[
\begin{bmatrix}
I & I & I & \cdots & I \\
I & \sigma & \sigma^2 & \cdots & \sigma^{p-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
I & \sigma^{r-1} & \sigma^{(r-1)2} & \cdots & \sigma^{(r-1)(p-1)}
\end{bmatrix}
\begin{bmatrix}
v_0 \\
v_1 \\
v_2 \\
v_{p-1}
\end{bmatrix} = \begin{bmatrix}
0 \\
0 \\
\vdots \\
0
\end{bmatrix}
\]
The number of 1's in each column is $r$ and the number in each row is $p$. It should be noted that the above matrix is not a full rank matrix because the sum of each set of $p$ rows is the all ones vector. Hence the actual rank of this matrix is $r(p - 1)$.

This parity check matrix is sparse, which allows the array code to be decoded as an LDPC code using the message-passing algorithm. On the other hand, while an analogous matrix can be constructed for Reed-Solomon and BCH codes, the corresponding binary parity check matrix would not be sparse. The advantage of these array codes, which share the algebraic structure of Reed-Solomon and BCH codes, is that their sparse matrix allows soft decoding.

5.1.2 Algebraic description of array codes

Taking each of the columns as symbols in a Galois ring, this basic construction actually turns out to have an interpretation as a cyclic code. In particular, consider the two rings $R_{p-1} \subset R_p$.

$$R_{p-1} = GF(2)[x]/(x^{p-1} + \cdots + 1)$$

$$R_p = GF(2)[x]/(x^p - 1)$$

Note that $(x^{p-1} + \cdots + 1)(x - 1) = x^p - 1$.

The two possible representations of an element $b \in R_{p-1}$, as an element of $R_p$, are $\sum_{i=0}^{p-2} b_i x^i$ and $\sum_{i=0}^{p-2} (1 + b_i) x^i + x^{p-1}$. Conversely, it is possible to map elements from $R_p$ to $R_{p-1}$ by using the following ring homomorphism.

$$\mu : R_p \rightarrow R_{p-1}$$

$$\mu : \sum_{i=0}^{p-1} b_i x^i \rightarrow \sum_{i=0}^{p-2} (b_i + b_{p-1}) x^i$$

Recall that the last row of the array code is all zero ($a_{p-1,j} = 0 \ \forall j$), and so each column of the array code corresponds to an element of the ring $R_{p-1}$. It turns out to be more elegant to think of the elements as elements of the extension ring $R_p$. For example, multiplying an element by $x$ corresponds to a cyclic shift of the elements of
the coefficients $a_i$:

$$ x \cdot a = x \sum_{i=0}^{p-1} a_i x^i = \sum_{i=0}^{p-1} a_{<i-1>_p} x^i $$

Hence the parity check equation for the array code can be rewritten in terms of $R_p$, as follows:

$$
\begin{bmatrix}
1 & 1 & 1 & \cdots & 1 \\
1 & x & x^2 & \cdots & x^{p-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & x^{r-1} & x^{(r-1)2} & \cdots & x^{(r-1)(p-1)}
\end{bmatrix}
\begin{bmatrix}
v_0 \\
v_1 \\
v_2 \\
\vdots \\
v_{p-1}
\end{bmatrix}
= 
\begin{bmatrix}
0 \\
0 \\
0 \\
\vdots \\
0
\end{bmatrix}
$$

where $v_i$ are elements of $R_{p-1} \subset R_p$.

This construction gives a cyclic code over the Galois ring $R_p$ that is precisely analogous to the algebraic construction of Reed-Solomon codes and BCH codes. In terms of the ring $R_p$, the code is actually a cyclic code. Consider the set of polynomials with coefficients in $R_p$, and consider the structure defined by

$$ R_p[z]/(z^p - 1) = \left\{ \sum_{i=0}^{p-1} v_i z^i \mid v_i \in R_p \right\} $$

Then the code is defined by

$$ C = \langle (z - 1)(z - x) \cdots (z - x^{r-1}) \rangle \subset R_p[z]/(z^p - 1) $$

where $g(z) = (z - 1)(z - x) \cdots (z - x^{r-1})$ is the generator polynomial. Any element of the code $C$ can be written as $k(z)g(z)$ modulo $z^p - 1$. In other words,

$$ u(z) = \sum_{i=0}^{p-1} v_i z^i \in C \text{ iff } c(z) = k(z)g(z) + h(z)(z^p - 1) $$

for some polynomials $k(z)$ and $h(z)$.

This structure yields a systematic encoder. If the data is put into $v_r, \ldots, v_{p-1}$, then
the corresponding parity symbols can be obtained as follows:

$$\sum_{i=0}^{r-1} v_i z^i = \sum_{i=r}^{p-1} v_i z^i \mod g(z)$$

This can be implemented by a structure which is similar to a linear shift register based on $g(z)$.

It is possible to correct errors and erasure using array codes in a manner similar to BCH and Reed-Solomon codes. Given redundancy $r$, the decoder can correct 1 error and $r - 2$ erasures, or $r$ erasures, as described in Appendix C.. For multiple errors, however, the decoding algorithm does not generalize because of the need for multiplicative inverses, which does not necessarily hold in the Galois ring. (The exception is when $R_{p-1}$ also happens to be a field, in which case it will be possible to decode multiple errors. In that case, $R_{p-1} = GF(2^{p-1})$, but $x$ has order $p - 1$ rather than $2^{p-1} - 1$, and the resulting array code is a very shortened version of a Reed-Solomon code over a very large symbol size.).

5.2 Design of array codes for soft decoding

The message-passing algorithm can be applied to decode an array code using its parity check matrix. Since short cycles and short minimum distance are detrimental to performance, some modification are considered for the array code in order to improve the performance. It turns out that LDPCs constructed from array codes automatically have no cycles of order 4. In addition, there are shortened versions of the array codes that eliminate other short cycles.

5.2.1 Shortened array codes

A standard version of the array code has rate $\frac{r-r}{p}$, and the blocklength is $p(p-1)$. It is possible to gain more flexibility in the block size and the rate by shortening the array code by removing message symbols. [Wic95, Section 4.6]. Suppose that $p - n$
symbols are removed, leaving \( n \) symbols in the code, which are described by

\[
0 \leq \alpha_0 < \alpha_1 < \cdots < \alpha_{n-1} \leq p - 1.
\]

Then the punctured parity check matrix looks like the following:

\[
H = \begin{bmatrix}
I & I & \cdots & I \\
\sigma^{\alpha_0} & \sigma^{\alpha_1} & \cdots & \sigma^{\alpha_{n-1}} \\
\vdots & \vdots & & \vdots \\
\sigma^{(r-1)\alpha_0} & \sigma^{(r-1)\alpha_1} & \cdots & \sigma^{(r-1)\alpha_{n-1}}
\end{bmatrix}
\]

The block length is \( n(p - 1) \) bits (leaving out the last bit of each \( p \)-block), while the number of independent rows is \( r(p - 1) \), giving a rate of \( \frac{n-r}{n} \). By choosing \( r, n \) and \( p \), it is possible to adjust the rate of the code and the block length independently.

For encoding, the systematic algebraic encoder given in §5.1.2 does not work for the shortened code, except in the case that none of the first \( r \) symbols are removed (so \( \alpha_i = i \) for \( 0 \leq i \leq r - 1 \)). An alternative encoding method is provided by solving for the \( r \) parity symbols \( (v_0, v_1, \ldots, v_{r-1}) \) which satisfy this equation, given the message symbols \( (v_r, v_{r+1}, \ldots, v_{n-1}) \).

\[
\begin{bmatrix}
I & \cdots & I \\
\sigma^{\alpha_0} & \cdots & \sigma^{\alpha_{r-1}} \\
\vdots & & \vdots \\
\sigma^{(r-1)\alpha_0} & \cdots & \sigma^{(r-1)\alpha_{r-1}}
\end{bmatrix}
\begin{bmatrix}
v_0 \\
v_1 \\
\vdots \\
v_{r-1}
\end{bmatrix}
= \begin{bmatrix}
I & \cdots & I \\
\sigma^{\alpha_r} & \cdots & \sigma^{\alpha_{n-1}} \\
\vdots & & \vdots \\
\sigma^{(r-1)\alpha_r} & \cdots & \sigma^{(r-1)\alpha_{n-1}}
\end{bmatrix}
\begin{bmatrix}
v_r \\
v_{r+1} \\
\vdots \\
v_{n-1}
\end{bmatrix}
\]

In the case of \( r = 3 \), it is possible to directly invert this matrix on the left-hand size in order to solve for the parity. For larger \( r \), it may be more difficult to compute the inverse of the matrix, and a generator matrix must be computed (e.g. using Gaussian elimination) for this parity check matrix in order to perform the encoding.

For decoding, this array code can be decoded both algebraically and with the message-passing algorithm. In Figure 5.1, an array code with \( n = 50 \), \( p = 97 \) and \( r = 5 \) is decoded as an LDPC code using the message-passing algorithm. The block
length is \( n \frac{n-1}{n} = 4800 \), and the number of parity checks is \( r \frac{n-1}{n} = 480 \), so the rate is \( \frac{n-1}{n} = 0.9 \). In comparison, an LDPC code with the same block length and rate is randomly constructed with \( t_c = 3 \) bits per column. The performance of the two codes is seen to be very close, with the array code only slightly worse than the LDPC code. This shows that despite the regular structure of the array code, it still performs reasonably well using the message-passing algorithm. In addition, as an algebraic decoder, this array code can correct 1 error symbol and up to 3 erased symbols, where the symbols are blocks of \( p - 1 = 96 \) bits each.

5.2.2 Cycle properties of permutation matrices

To understand the cycle properties of array codes, we first consider the cycle properties of matrices constructed from permutation matrices. Consider the following parity
check matrix

\[
H = \begin{bmatrix}
S_{0,0} & S_{0,1} & S_{0,2} & \cdots & S_{0,n-1} \\
S_{1,0} & S_{1,1} & S_{1,2} & \cdots & S_{1,n-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
S_{r-1,0} & S_{r-1,1} & S_{r-1,2} & \cdots & S_{r-1,n-1}
\end{bmatrix}
\]

where \(S_{i,j}\) is a permutation matrix. The cycle structure of the corresponding factor graph can be understood in terms of these permutation matrices using the following theorem. Note that there is a cycle in a permutation \(\pi\) of length \(\ell\) if \(\pi^\ell(i) = i\).

**Theorem 5** For \(2k\) permutation matrices \(S_{i_1,j_1}, S_{i_2,j_2}, S_{i_2,j_3}, \ldots, S_{i_k,j_1}\) where \(i_1 \neq i_2 \neq \cdots \neq i_k \neq i_1\) and \(j_1 \neq j_2 \neq \cdots \neq j_k \neq j_1\), then the possible cycles in the factor graph are \(2k\) times the length of the cycles in the permutation described by

\[
T = S_{i_1,j_1} \left(S_{i_1,j_2}\right)^{-1} S_{i_2,j_2} \left(S_{i_2,j_3}\right)^{-1} \cdots S_{i_k,j_k} \left(S_{i_k,j_1}\right)^{-1}.
\]

In particular, there is a cycle of length \(2k\) in the factor graph corresponding to these matrices if and only if the permutation \(T\) has fixed points.

**Proof:** If a \(p \times p\) permutation matrix \(S\) has a 1 at the \(u\)-th row and \(v\)-th column, then applying \(S\) to a column vector takes the \(v\)-th entry and sends it to the \(u\)-th location. Consider a path of alternating vertical and horizontal line segments of length \(2k\) in the parity check matrix, with the corners being 1's that are located inside the prescribed permutation matrices \(S_{i_1,j_1}, S_{i_1,j_2}, S_{i_2,j_2}, S_{i_2,j_3}, \ldots, S_{i_k,j_1}\).

For the transition from a vertical segment to a horizontal segment at the \((u, v)\)-th location of a permutation matrix, it is going from the \(v\)-th column to the \(u\)-th row, which corresponds to standard multiplication by the permutation matrix. On the other hand, for a horizontal path becoming vertical at \((u, v)\), it goes from the \(u\)-th row to the \(v\)-th column, which corresponds to multiplication by the inverse matrix.

Hence it is possible to follow the course of all paths that pass through this sequence of permutation matrices in this parity check matrix by looking at the product:

\[
T = S_{i_1,j_1} \left(S_{i_1,j_2}\right)^{-1} S_{i_2,j_2} \left(S_{i_2,j_3}\right)^{-1} \cdots S_{i_k,j_k} \left(S_{i_k,j_1}\right)^{-1}
\]
CHAPTER 5. ARRAY CODES

If there is a fixed point in this product $T$, then it is possible to follow the loop and end up at the same point in the parity check matrix, giving a cycle of length $2k$ in the factor graph. Otherwise, if this permutation decomposes into permutation cycles and there is a permutation cycle of order $l$, then it is possible to go through this entire loop $l$ times and end up at the starting point, giving a cycle in the factor graph of length $2kl$. ■

**Corollary 6** The permutation described by $S_{ij}(S_{il})^{-1}S_{kl}(S_{kj})^{-1}$ has no fixed points if and only if there is no cycle of length 4 corresponding to these four matrices.

Hence, to show that a parity check matrix defined from permutation matrices has no cycles of length 4, it is equivalent to show that for all valid $(i, j)$ and $(k, l)$, the permutation $S_{ij}(S_{il})^{-1}S_{kl}(S_{kj})^{-1}$ has no fixed points.

### 5.2.3 Avoiding short cycles

The results on cycle length can be applied to the parity check matrix for the shortened array code.

$$
\begin{bmatrix}
I & I & \cdots & I \\
\sigma^{\alpha_0} & \sigma^{\alpha_1} & \cdots & \sigma^{\alpha_{n-1}} \\
\vdots & \vdots & & \vdots \\
\sigma^{(r-1)\alpha_0} & \sigma^{(r-1)\alpha_1} & \cdots & \sigma^{(r-1)\alpha_{n-1}}
\end{bmatrix}
$$

The presence of cycles are length 4 depends on whether the following permutation has fixed points.

$$
S_{ij}(S_{il})^{-1}S_{kl}(S_{kj})^{-1} = \sigma^{i\alpha_i} \sigma^{-i\alpha_i} \sigma^{(i-k)\alpha_j} \sigma^{-k\alpha_j} = \sigma^{(i-k)(\alpha_j - \alpha_i)}
$$

This has fixed points if and only if $\sigma^{(i-k)(\alpha_j - \alpha_i)} = 1$, which occurs when $(i-k)(\alpha_j - \alpha_i)$ is divisible by the prime $p$. But it has been specified that $i \neq k$, and $j \neq l$, so $\alpha_j \neq \alpha_l$. Moreover, $|i - k| < r$ and $|\alpha_j - \alpha_l| < p$, so that it is impossible for $p$ to divide $(i-k)(\alpha_j - \alpha_l)$. Hence, there are no cycles of order 4 for array codes.
CHAPTER 5. ARRAY CODES

The next possible cycle, of length 6, corresponds to a rectilinear traversal of the parity check matrix, where the corners are 1's that are located in a sequence of six permutation matrices \( S_{i_1,j_1}, S_{i_1,j_2}, S_{i_2,j_2}, S_{i_2,j_3}, S_{i_3,j_3}, S_{i_3,j_1} \). The presence of cycles of length 6 depends on whether the permutation

\[
T = S_{i_1,j_1}(S_{i_1,j_2})^{-1}S_{i_2,j_2}(S_{i_2,j_3})^{-1}S_{i_3,j_3}(S_{i_3,j_1})^{-1}
\]

\[
= \sigma^{i_1\alpha_j_1-i_1\alpha_j_2+i_2\alpha_j_2-i_2\alpha_j_3+i_3\alpha_j_3-i_3\alpha_j_1}
\]

\[
= \sigma^{(i_1-i_2)\alpha_{j_1}+(i_2-i_1)\alpha_{j_2}+(i_3-i_2)\alpha_{j_3}}
\]

has any fixed points. To avoid cycles of length 6, we need to shorten the code and choose the \( \alpha_j \)'s so that it is not possible to have

\[
p \mid (i_1 - i_3)\alpha_{j_1} + (i_2 - i_1)\alpha_{j_2} + (i_3 - i_2)\alpha_{j_3}
\]

for any distinct \( i_1, i_2, i_3 \in [0, r - 1] \) and \( j_1, j_2, j_3 \in [0, n - 1] \). If \( p \) is much larger than \( n \), then there will be much flexibility in the choice of the puncturing pattern, so that it is possible to satisfy this condition.

Similarly, the condition on \( \{\alpha_j\} \) that result in cycles of length 8 is

\[
p \mid (i_1 - i_4)\alpha_{j_1} + (i_2 - i_1)\alpha_{j_2} + (i_3 - i_2)\alpha_{j_3} + (i_4 - i_3)\alpha_{j_4}
\]

where \( i_1 \neq i_2 \neq i_3 \neq i_4 \neq i_1 \) and \( j_1 \neq j_2 \neq j_3 \neq j_4 \neq j_1 \). By shortening the code properly, it is possible to choose \( \alpha_j \) to avoid this condition, and hence avoid cycles of length 8. Generalizing to cycles of length \( 2k \), the condition for a cycle of length \( 2k \) occurs is

\[
p \mid (i_1 - i_k)\alpha_{j_1} + (i_2 - i_1)\alpha_{j_2} + \cdots + (i_k - i_{k-1})\alpha_{j_k},
\]

where adjacent values of the \( i_i \)'s and \( j_i \)'s cannot be identical, i.e. \( i_l \neq i_{l+1} \) and \( i_k \neq i_1 \), and \( j_l \neq j_{l+1} \) and \( j_k \neq j_1 \).
5.2.4 Increasing minimum distance using coded symbols

The performance of the decoder is determined by the minimum distance of the code. Array codes are maximum distance separable (MDS) in terms of the symbols in $R_p$. In other words, they achieve the largest minimum distance that is possible for a code with the same parameters. For $r$ parity symbols, the array code achieves the minimum distance of $r + 1$ symbols.

A good minimum distance in terms of symbols (which we will refer to as symbol-distance), however, does not necessarily translate to good distance in terms of bits (which we call bit-distance). For a randomly designed code, the bit-distance is expected to increase linearly with the block length. In this case, however, it is possible that the bit-distance is not much more than the symbol-distance. Looking at the construction of array code for $r = 3$, for instance, it is possible to find codewords of bit-weight 8, meaning that the minimum bit-distance of this linear code is no more than 8.

One method to increase the bit-distance is to introduce an additional code on each symbol. The use of an error-correcting code to protect the symbol is appropriate especially if the code can correct a small number of errors, and also produce an erasure flag when it is not able to correct the symbol. Such a configuration is well suited for the array code, which can correct only 1 symbol-error but can handle multiple erasures.

Array codes serve as one of the first examples of codes that have the capability to correct error bursts using an algebraic decoder, but can also be incorporated into soft iterative decoding schemes. It is suspected that other soft-decodable algebraic codes can be constructed. These codes have promising application in storage and communication systems as a replacement for hard-decision Reed-Solomon codes.
Appendix A

Soft interference cancellation

In systems with intersymbol interference, an alternative to maximum likelihood sequence detection is to use soft cancellation. Instead of cancelling out with hard decisions, as in Decision Feedback Equalization, it is possible to perform soft cancellations, where probabilistic values are used to partially cancel out the interference. An iterative scheme using soft cancellation has advantages over hard cancellation, since the uncertain decisions are not subtracted off.

A.1 General channel

Consider a general framework for a communications channel

\[ y = Ax + n \]

where \( A \) is the channel matrix, \( x \) is the transmitted sequence, \( y \) is the received samples, and \( n \) is the noise. This can be written as

\[ y_i = \sum_j a_{ij} x_j + n_i \]  \hspace{1cm} (A.1)

where the term \( x_j = 2x_j - 1 \in \{-1, 1\} \) is the transmitted version of the binary number \( x_j \in \{0, 1\} \). From the observed sequence \( y \), it is desired to estimate the probabilities

127
for the input sequence $x$.

First, consider the hypothetical situation where accurate hard-decisions are available for all the bits $x_j$ except for the $k$-th bit. Then define the expression

$$
\tilde{y}_{i,k} = y_i - \sum_{j \neq k} a_{ij} x_j
$$

(A.2)

in which the contributions from the other decisions are subtracted off. Then substituting equation (A.1) into (A.2) gives

$$
\tilde{y}_{i,k} = a_{ik} x_k + n_i.
$$

Assume the noise $n_i$ is Gaussian with variance $\sigma_i^2$, and let $\sigma_{ik}^2 = \sigma_i^2$.

It is possible to estimate the posterior log-likelihood ratio for $x_i$, as in Subsection 3.2.4:

$$
\text{LLR}_{\text{posterior}} (x_k \mid y_i) = \log \frac{P(y_{i,k} \mid x_k = 1) P(x_k = 1)}{P(y_{i,k} \mid x_k = -1) P(x_k = -1)}
$$

$$
= \log \frac{\exp \left( -\frac{1}{2\sigma_{ik}^2} |\tilde{y}_{i,k} - a_{ik}|^2 \right)}{\exp \left( -\frac{1}{2\sigma_{ik}^2} |\tilde{y}_{i,k} + a_{ik}|^2 \right)} + \text{LLR}_{\text{prior}} (x_k)
$$

$$
= \frac{2}{\sigma_{i,k}^2} a_{ik}^* \tilde{y}_{i,k} + \text{LLR}_{\text{prior}} (x_k)
$$

Note that $a_{ik}^*$ is the complex conjugate of $a_{ik}$.

Then summing over all of the observed sequence $y$ gives an estimate for the posterior information based on cancellation

$$
\text{LLR}_{\text{posterior}} (x_k \mid y) = \sum_i \frac{2}{\sigma_{i,k}^2} a_{ik}^* \tilde{y}_{i,k} + \text{LLR}_{\text{prior}} (x_k)
$$

(A.3)

The key idea is to use soft bits to cancel out, rather than hard-decisions, in order to reduce the error-propagation usually associated with interference cancellation schemes. In other words, the hard-decisions $x_j$ are replaced by soft bits, $\chi_j = (\tanh \frac{1}{2} \text{LLR} (x_j))$, which are estimated from the posterior information on the
bit $x_j$.

The uncertainty in this value contributes to the noise, so that the noise estimate should be adjusted accordingly. If a binary variable (with values \{-1, 1\}) has probability $p_j = \Pr(x_j = 1)$, then the average value is $\chi_j = 2p_j - 1$, while the variance is $1 - \chi_j^2 = 4p_j (1 - p_j)$. Then equation (A.2) becomes

$$
\tilde{y}_{i,k} = y_i - \sum_{j \neq k} a_{ij} x_j \\
= a_{ik}^* x_k + \sum_{j \neq k} a_{ij} (x_j - \chi_j) + n_i
$$

so that the noise is

$$
\sum_{j \neq k} a_{ij} (x_j - \chi_j) + n_i
$$

which has variance

$$
\tilde{\sigma}_{i,k}^2 \approx \sum_{j \neq k} |a_{ij}|^2 (1 - \chi_j^2) + \sigma_i^2
$$  \hspace{1cm} (A.4)

Putting this together with equations (A.3) and (A.2), the expression for the posterior LLR is given by

$$
\text{LLR}_{\text{posterior}}(x_k | y) = \sum_i \frac{2}{\tilde{\sigma}_{i,k}^2} a_{ik}^* \left( y_i - \sum_{j \neq k} a_{ij} x_j \right) + \text{LLR}_{\text{prior}}(x_k) \hspace{1cm} (A.5)
$$

where $\tilde{\sigma}_{i,k}^2$ are the revised noise variances that take into account the uncertainty in the cancellation. Then to complete this process, it is possible to estimate an updated value for the soft bit for $x_k$ using this posterior LLR.

$$
\chi_k = \tanh \left( \frac{1}{2} \text{LLR}_{\text{posterior}}(x_k | y) \right) \hspace{1cm} (A.6)
$$

This is an ad hoc method, but for many applications, soft interference cancellation appear to be a robust and low-complexity decoding technique. In addition, since these methods provide soft information on the output, they act as soft-in, soft-out decoders and it is possible to incorporate them with a soft ECC such as a turbo code or
LDPC code. There has been research applying these ideas to different communication systems such as multiuser wireless systems [SLSY98][Moh98] and magnetic recording [WCi99]. Note that it is possible to extend these concepts to complex valued signals, as well as multilevel constellations such as QAM.

### A.2 Application to magnetic recording channel

This section uses a simplified model of the magnetic recording channel as an intersymbol channel with additive white Gaussian noise:

\[ y_i = \sum_j h_j x_{i-j} + n_i \]

There are various techniques for obtaining reliable soft information on the bits. The transmitted signal is \( x_i \in \{-1, 1\} \), corresponding to the binary sequence \( x_i \in \{0, 1\} \). The optimal answer can be obtained from the Viterbi algorithm, while a soft decoder can be obtained from the forward-backward algorithm (a.k.a. the BCJR algorithm). These approaches have high complexity, in general, and while an equalizer can be applied to shorten the channel, as in the case of partial response, the use of an equalizer can cause a mismatch of the channel and the decoder. Alternatively, a low-complexity approach is obtained from Decision-Feedback Equalization (DFE), but this has problems such as error-propagation. The soft cancellation technique has low-complexity like the DFE but without the associated error-propagation. In addition, it also allows for soft information to be computed on the output.

The channel is assumed to be time-invariant, so the matrix \( A \) is Toeplitz, and \( a_{i,j} = h_{i-j} \). Also, all the noise samples \( n_i \) are assumed to have the same variance \( \sigma^2 \), and the noise \( \sigma^2_{i,k} = \frac{\sigma^2}{|a_{i,k}|^2} \) only includes the channel noise (and not the uncertainty in the soft bits \( x_j \)). Then according to (A.5), the posterior LLR for a bit is given in terms of soft bits by
\[
\text{LLR}^{\text{posterior}}(x_k | y) = \sum_i \frac{2}{\sigma^2} |a_{i,k}|^2 \frac{1}{a_{i,k}} \left( y_i - \sum_{j \neq k} a_{ij} x_j \right) + \text{LLR}^{\text{prior}}(x_k)
\]

\[
= \sum_i \frac{2}{\sigma^2} a_{i,k}^* \left( y_i - \sum_{j \neq k} a_{ij} x_j \right) + \text{LLR}^{\text{prior}}(x_k)
\]

\[
= \frac{2}{\sigma^2} \sum_i h_{i-k}^* \left( y_i - \sum_{j \neq k} h_{i-j} x_j \right) + \text{LLR}^{\text{prior}}(x_k)
\]

Convolving with \(h^*(D)\) is equivalent to applying the matched filter. Then it is possible to express the update equation in terms of polynomials as follows

\[
L^{\text{posterior}}(D) = \frac{2}{\sigma^2} (h^*(D^{-*}) y(D) - (h^*(D^{-*}) h(D) - 1) x(D)) + L^{\text{prior}}(D)
\]

where \(L^{\text{posterior}}(D) = \sum (\text{LLR}^{\text{posterior}}(x_k | y)) D^i\), and \(L^{\text{prior}}(D)\) is similarly defined. The iteration is completed by computing the new soft bit estimates using this new approximate posterior LLR,

\[
\chi_k^{\text{new}} = \tanh \left( \frac{1}{2} \text{LLR}^{\text{posterior}}(x_k) \right).
\]

It should be emphasized that this posterior LLR is not the true posterior log-likelihood ratio, but a low-complexity approximation using soft information to cancel off interference.

The similarities between this structure and the two-sided linear canceller [GL81] should be noted. As compared with Decision-Feedback Equalization (DFE), which uses past decisions to cancel out the intersymbol interference, the linear canceller uses tentative decisions from both past and future to cancel out the interference. The suitability of this structure for cancelling ISI and nonlinear distortion using tentative decisions is analyzed in [AS97], and the use of this structure in conjunction with a turbo code appears in [WCi99].
Appendix B

The $(0, G/I)$ constraint

The $(0, G/I)$ constraint is important for magnetic recording, and consists of a global $(0, G)$-RLL constraint as well as a $(0, I)$-RLL constraint on both the even and the odd interleaves. In analogy to the results for $(d, k)$ constraints in Chapter 4, constructions of systematic modulation codes and bit insertion are considered for the $(0, G/I)$ constraint.

B.1 Systematic modulation codes

Systematic modulation codes are presented for the $(0, G/I)$ constraint. First, suppose that $G \geq I$ and $I$ is even. Then following the example of the $(0, k)$ constraint, it is possible to have 1’s alternating with $I$ user bits, giving a code of rate $\frac{I}{I+1}$ that satisfies the constraint $(0, I/I)$, and hence satisfies $(0, G/I)$. Similarly, for $G > I$ and $I$ odd, the ones are alternately separated by $I-1$ and $I+1$ user bits, giving a rate $\frac{I}{I+1}$ code which satisfies the $(0, G/I)$ constraint for $G \geq I+1$. For $G < I$ and $G$ even, the code with 1’s alternating with $G$ user bits satisfies the $(0, G/G)$ constraint with rate $\frac{G}{G+1}$.

The results on systematic modulation codes for the $(0, G/I)$ constraint are summarized in Table B.1, where the blank square $\square$ indicates a place for the user bit. The code may sometimes satisfy a constraint which is tighter than the required $(0, G/I)$ constraint. Finally, Table B.2 lists the maximum rates for a systematic modulation code for some values of $G$ and $I$.  

132
APPENDIX B. THE \((0, G/I)\) CONSTRAINT

<table>
<thead>
<tr>
<th>Condition</th>
<th>Max. rate</th>
<th>Constraint</th>
<th>Example of a modulation pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>(G \geq I, I) even</td>
<td>(I)</td>
<td>((0, I/I))</td>
<td>((0, 4/4), 1\square\square\square1\square\square\square1...)</td>
</tr>
<tr>
<td>(G &gt; I, I) odd</td>
<td>(I+1)</td>
<td>((0, (I+1)/I))</td>
<td>((0, 4/3), 1\square\square\square1\square\square\square1...)</td>
</tr>
<tr>
<td>(G &lt; I, G) even</td>
<td>(\frac{I}{2+1})</td>
<td>((0, G/G))</td>
<td>((0, 4/5), 1\square\square\square1\square\square\square1...)</td>
</tr>
<tr>
<td>(G &lt; I, G) odd</td>
<td>(f(G, I))</td>
<td>((0, G/I))</td>
<td>((0, 3/4), 1\square\square\square1\square\square\square1\square\square\square1...)</td>
</tr>
</tbody>
</table>

Table B.1: Systematic modulation codes for the \((0, G/I)\) constraints

<table>
<thead>
<tr>
<th>(G) (\setminus\ I)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
</tr>
<tr>
<td>2</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
</tr>
<tr>
<td>3</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
</tr>
<tr>
<td>4</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
</tr>
<tr>
<td>5</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
</tr>
<tr>
<td>6</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
</tr>
</tbody>
</table>

Table B.2: Maximum rates for \((0, G/I)\) systematic modulation codes

In the case of \(G \leq I\) and \(G\) odd, the expression for the maximum systematic modulation rate is not straightforward. For example, for the constraint \((0, 3/4)\), it is possible to construct a systematic code of rate 5/7, by alternately using 2 and 3 user bits in between 1's, in the pattern 1\square\square1\square\square1\square\square1.... In general, if there is a run of \(G = 2H + 1\) user bits (which is all zero in the worst case), then this gives \([G/2] = H + 1\) zeroes in one of the two interleaves. This pattern can be continued as long as the accumulated number of continuous zeroes in one interleave does not exceed \(I\). Let the variable \(Z\) represent the number of repetitions of a pattern of \(G\) user bits followed by a 1, and let \(2T\) represent a run of user bits (corresponding to \(T\) user bits in an interleave). Putting these two patterns together, there are \(T + Z(H + 1)\) consecutive user bits in one of the interleaves, which must be \(\leq I\). It turns out that the optimal rate is given by maximizing the following expression

\[
f(G, I) = \max_{T,Z} \frac{T + 2Z(H + 1) - Z}{T + 2Z(H + 1) + 1}
\]
while satisfying the following inequalities for the non-negative variables $Z$ and $T$.

$$T \leq 2H$$

$$T + Z(H + 1) \leq I$$

Then there is the relation $Z = \left\lfloor \frac{I - T}{H + 1} \right\rfloor$ between $Z$ and $T$, and the general solution is as follows. There are only several values of $T$ that need to be considered. Suppose $T_0 \equiv I \mod (H + 1)$, and let $Z_0 = \left\lfloor \frac{I}{H + 1} \right\rfloor$, which gives the rate

$$r_1 = \frac{T_0 + Z_0(2H + 1)}{T_0 + Z_0(2H + 2) + 1}$$

Then for the case $Z = Z_0 - 1$, we have $T = \min \{T_0 + H + 1, 2H\}$. Except when $T_0 \in \{H, H + 1\}$, this gives $T = T_0 + H + 1$, so the rate is given by

$$r_2 = \frac{T_0 + Z_0(2H + 1) - H}{T_0 + Z_0(2H + 2) - H}$$

Finally, there is an additional case of $T = 2H$, where $Z = \left\lfloor \frac{I - 2H}{H + 1} \right\rfloor$

$$r_3 = \frac{2H + Z(2H + 1)}{2H + 2Z(2H + 2) + 1}$$

The maximum of these three rates gives the maximal rate for a systematic $(0, G/I)$ modulation code

$$f(G, I) = \max \{r_1, r_2, r_3\}.$$ 

**B.2 Bit Insertion**

A similar bit insertion technique is also applicable to the $(0, G/I)$ constraint. When $L$ is even, a parity bit is inserted in between every block of $L$ message bits. When $L$ is odd, it is necessary to alternate between blocks of $L - 1$ and $L + 1$ message bits in between parity bits in order to maintain runlength constraint on the interleaves. In terms of the global constraint, the minimum run of message bits is $2 \left\lfloor \frac{L}{2} \right\rfloor$. If
the message is modulated with a \((0, G_0/I_0)\) constraint for code \(C_1\), then inserting a bit every \(L\) bits yields a \(\left(0, G_0 + \left\lfloor \frac{G_0+1}{2} \right\rfloor \right)\) global constraint, and a \(\left(0, I_0 + \left\lceil \frac{I_0+1}{L} \right\rceil \right)\) constraint on each interleave. It is necessary for both inequalities to hold:

\[
G_0 + \left\lfloor \frac{G_0+1}{2} \left\lceil \frac{L}{2} \right\rceil \right\rfloor \leq G
\]

\[
I_0 + \left\lceil \frac{I_0+1}{L} \right\rceil \leq I
\]

Let \(a_G = G - G_0\), and \(a_I = I - I_0\). The two conditions can be rewritten as

\[
2 \left\lceil \frac{L}{2} \right\rceil \geq \left\lfloor \frac{G+1}{a_G} \right\rfloor - 1
\]

\[
L \geq \left\lceil \frac{I+1}{a_I} \right\rceil - 1
\]

so that the minimum possible value for \(L\) is

\[
L = \max \left( 2 \left\lceil \frac{G+1}{a_G} \right\rceil - 1, \left\lceil \frac{I+1}{a_I} \right\rceil - 1 \right).
\]

In the case of \(a_G = a_I = 1\), this simplifies to \(L = \max \left( 2 \left\lceil \frac{G}{2} \right\rceil, I \right)\).

In direct analogy with the \((0, k)\) case, the overall rate can be calculated as

\[
\text{overall rate} = \text{rate}(0, G_0/I_0) \cdot \text{rate}(ECC)
\]

where \(G_0 = G - a_G\) and \(I_0 = I - a_I\), where \(a_G, a_I \geq 1\). The number \(\text{rate}(0, G_0/I_0)\) represents the rate of an modulation code (usually close to the capacity of the constraint). A lower bound on the rate of the ECC is given by a limit on the number of inserted bits:

\[
\text{rate}(ECC) \geq \frac{L}{L+1}
\]  \hspace{1cm} (B.1)

On the other hand, the upper bound is given by comparing the total number of channel bits with bit insertion and with systematic modulation. Bit insertion is more
APPENDIX B. THE \((0, G/I)\) CONSTRAINT

<table>
<thead>
<tr>
<th>((0, G/I))</th>
<th>(a_G, a_I)</th>
<th>(L)</th>
<th>(\text{rate}(0, G_0/I_0))</th>
<th>lower</th>
<th>upper</th>
</tr>
</thead>
<tbody>
<tr>
<td>((0,3/3))</td>
<td>1,1</td>
<td>4</td>
<td>0.8345</td>
<td>0.8000</td>
<td>0.8054</td>
</tr>
<tr>
<td>((0,3/4))</td>
<td>1,1</td>
<td>4</td>
<td>0.8671</td>
<td>0.8000</td>
<td>0.8063</td>
</tr>
<tr>
<td>((0,3/5))</td>
<td>1,1</td>
<td>5</td>
<td>0.8757</td>
<td>0.8333</td>
<td>0.8120</td>
</tr>
<tr>
<td>((0,4/4))</td>
<td>1,1</td>
<td>4</td>
<td>0.9157</td>
<td>0.8000</td>
<td>0.8006</td>
</tr>
<tr>
<td>((0,4/5))</td>
<td>1,1</td>
<td>5</td>
<td>0.9343</td>
<td>0.8333</td>
<td>0.8485</td>
</tr>
<tr>
<td>((0,4/6))</td>
<td>1,1</td>
<td>6</td>
<td>0.9415</td>
<td>0.8571</td>
<td>0.8666</td>
</tr>
<tr>
<td>((0,5/4))</td>
<td>1,1</td>
<td>6</td>
<td>0.9395</td>
<td>0.8571</td>
<td>0.8728</td>
</tr>
<tr>
<td>((0,5/5))</td>
<td>1,1</td>
<td>6</td>
<td>0.9614</td>
<td>0.8571</td>
<td>0.9234</td>
</tr>
<tr>
<td>((0,5/6))</td>
<td>1,2</td>
<td>6</td>
<td>0.9395</td>
<td>0.8571</td>
<td>0.8285</td>
</tr>
<tr>
<td>((0,5/5))</td>
<td>2,1</td>
<td>5</td>
<td>0.9343</td>
<td>0.8333</td>
<td>0.8051</td>
</tr>
<tr>
<td>((0,5/5))</td>
<td>2,2</td>
<td>2</td>
<td>0.9157</td>
<td>0.6667</td>
<td>0.7202</td>
</tr>
<tr>
<td>((0,5/6))</td>
<td>1,1</td>
<td>6</td>
<td>0.9696</td>
<td>0.8571</td>
<td>0.9394</td>
</tr>
<tr>
<td>((0,6/6))</td>
<td>1,1</td>
<td>6</td>
<td>0.9798</td>
<td>0.8571</td>
<td>0.9367</td>
</tr>
<tr>
<td>((0,6/6))</td>
<td>2,2</td>
<td>4</td>
<td>0.9614</td>
<td>0.8000</td>
<td>0.8206</td>
</tr>
<tr>
<td>((0,7/7))</td>
<td>1,1</td>
<td>8</td>
<td>0.9901</td>
<td>0.8889</td>
<td>0.9714</td>
</tr>
<tr>
<td>((0,7/7))</td>
<td>2,2</td>
<td>4</td>
<td>0.9798</td>
<td>0.8000</td>
<td>0.9078</td>
</tr>
</tbody>
</table>

Table B.3: Bounds for bit insertion for \((0, G/I)\) codes

Efficient when

\[
\frac{m}{\text{rate}(ECC) \cdot \text{rate}(0, G_0/I_0)} \leq \left(1 + \frac{1}{\text{sys}(0, G/I)} \left(\frac{1}{\text{rate}(ECC)} - 1 \right)\right) \frac{m}{\text{rate}(0, G/I)},
\]

where \(\text{sys}(0, G/I)\) is the maximum rate for systematic code for the \((0, G/I)\) constraint, and \(\text{rate}(0, G/I)\) is the capacity of the \((0, G/I)\) constraint, as found in [MSW92, Section VII]. This expression simplifies to the following upper bound on \(\text{rate}(ECC)\).

\[
\text{rate}(ECC) \leq \frac{1}{1 - \text{sys}(0, G/I)} \left(1 - \text{sys}(0, G/I) \frac{\text{rate}(0, G/I)}{\text{rate}(0, G_0/I_0)}\right)
\]  \hspace{1cm} (B.2)

Using equations (B.1) and (B.2), it is possible to find lower and upper bounds on \(\text{rate}(ECC)\) for various values of \(G\) and \(I\) for the use of bit insertion (as opposed to modified concatenation with a systematic modulation code), as shown in Table B.3.
Appendix C

Algebraic decoding of array codes

Explicit equations for algebraic decoding are presented for the array codes in Chapter 5, showing the relatively straightforward nature of the decoding.

C.1 Correcting 1 error

Consider the case where \( r = 2 \). To correct a single error, the decoder evaluates the syndromes from the received symbols \( \tilde{v}_k \).

\[
S_0 = \sum_k \tilde{v}_k \\
S_1 = \sum_k \tilde{v}_k x^k
\]

If there are no errors (\( \tilde{v} = v \)), the syndromes are 0. If there is a single column in error, so that \( \tilde{v}_j = v_j + e_j \) for some \( j \), then

\[
S_0 = e_j \\
S_1 = e_j x^j
\]

By comparing these syndromes, it is possible to deduce the column \( j \) since the effect of \( x^j \) is to rotate the vector corresponding to \( e_j \).
We show that it is always possible to deduce a unique value of \( j \) from \( S_0 \) and \( S_1 \): Suppose that it is not unique and there are two possible solutions \( j \) and \( j' = j + k \). In other words, both \( x^j S_0 \) and \( x^{j'} S_0 \) are equal to \( S_1 \), so that \( x^j (1 - x^k) S_0 = 0 \). It turns out for \( k \neq 0 \), elements of the form \( 1 - x^k \) are invertible over \( R_p \) (ref. [BR93]), so that \( S_0 = 0 \), contradicting the assumption that \( S_0 \) is non-zero, corresponding to an error.

### C.2 Correcting 1 error and 1 erasure

For \( r = 3 \), it is possible to correct 1 error and 1 erasure. Suppose the decoder is given the location \( j_1 \) of an erasure. Then the decoder calculate the syndromes

\[
S_i = \sum_k \hat{v}_k x^{ik}
\]

for \( i = 0, 1, 2 \), and obtains the following system of equations over \( R_p \).

\[
\begin{align*}
S_0 &= e_{j_0} + e_{j_1} \\
S_1 &= e_{j_0} x^{j_0} + e_{j_1} x^{j_1} \\
S_2 &= e_{j_0} x^{2j_0} + e_{j_1} x^{2j_1}
\end{align*}
\]

A set of pseudo-syndromes can be obtained as follows

\[
\begin{align*}
S_0^{(1)} &= S_1 - x^{j_1} S_0 = e_{j_0} (x^{j_0} - x^{j_1}) \\
S_1^{(1)} &= S_2 - x^{j_1} S_1 = e_{j_0} (x^{j_0} - x^{j_1}) x^{j_0}
\end{align*}
\]

and then it is possible to apply the decoder for 1 error outlined above, to obtain the location \( j_0 \) and the value \( e^{(1)} \) corresponding to \( e_{j_0} (x^{j_0} - x^{j_1}) \). The inversion of the element \( x^j - x^k \) over \( R_p \) can be implemented with fairly low complexity (ref. [BR93]). Hence it is possible for the decoder to obtain the error location \( j_0 \) as well as the error...
value $e_{j_0}$ and erasure value $e_{j_1}$.

$$e_{j_0} = (x^{j_0} - x^{j_1})^{-1} e^{(1)}$$

$$e_{j_1} = S_0 - e_{j_0}$$

This completes the decoding of 1 error and 1 erasure.

### C.3 Correcting 1 error and r-2 erasures

In general, suppose there are $r$ columns of redundancy in the array code, and the received array $\hat{v}$ has 1 column in error and $s = r - 2$ columns that are erased. The error is in an unknown location $j_0$, and the erased columns are in known positions $j_1, j_2, ..., j_s$. Then the decoder calculates the syndromes

$$S_i = \sum_k x^{ik} \hat{v}_k$$

for $0 \leq i < r$. As a reference, in terms of vectors, this corresponds to $S_i = \sum_k \sigma^{ik} \hat{v}_k$.

To find the location of the error, recursively construct pseudo-syndromes

$$S_i^{(t)} = S_{i+1}^{(t-1)} - x^{j_t} S_i^{(t-1)}$$

for $0 \leq i \leq t$, where the number $t$ goes from 1 to $s$. At the end of this procedure, the two pseudo-syndromes are obtained

$$S_0^{(s)} = S_s - \left( \sum_{t=1}^s x^{j_t} \right) S_{s-1} + \cdots + (-1)^s \left( \prod_{t=1}^s x^{j_t} \right) S_0$$

$$S_1^{(s)} = S_{s+1} - \left( \sum_{t=1}^s x^{j_t} \right) S_s + \cdots + (-1)^s \left( \prod_{t=1}^s x^{j_t} \right) S_1$$

where $s = r - 2$. Applying the decoder for a single error gives the location $j_0$ of the error, and an error value $e^{(s)}$. The original error value is then obtained by multiplying
this value by the appropriate inverses.

\[ e_{j_0} = \left( \prod_{j=1}^{s} (x^{j_0} - x^{j_t}) \right)^{-1} e^{(s)} \]

Next, to calculate the erasure values, the decoder computes

\[ e^{(s-l)}_{j_t} := \left( \prod_{u=l+1}^{s} (x^{j_t} - x^{j_u}) \right) e_{j_t} \]

for \( t < l \), so that \( e^{(s-l)}_{j_t} = (x^{j_t} - x^{j_{l-1}})^{-1} e^{(s-l+1)}_{j_t} \). Then some intermediate expressions are

\[
\begin{align*}
    e^{(s-1)}_{j_1} &= S_0^{(s-1)} - e^{(s-1)}_{j_0} \\
    e^{(s-2)}_{j_2} &= S_0^{(s-2)} - e^{(s-2)}_{j_0} - e^{(s-2)}_{j_1} \\
    &\vdots \\
    e^{(s-l)}_{j_l} &= S_0^{(s-l)} - \sum_{t=0}^{l-1} e^{(s-l)}_{j_t} \\
    &\vdots \\
    e^{(0)}_{j_s} &= S_0 - \sum_{t=0}^{s-1} e^{(0)}_{j_t}
\end{align*}
\]

where \( e^{(0)}_{j_t} = e_{j_t} \) are the desired values. Note that each equation can be written as

\[
\begin{align*}
    e^{(s-l)}_{j_t} &= S_0^{(s-l)} - \sum_{t=0}^{l-1} e^{(s-l)}_{j_t} \\
    &= S_0^{(s-l)} - \sum_{t=0}^{l-1} \left( (x^{j_t} - x^{j_{l-1}})^{-1} e^{(s-l+1)}_{j_t} \right)
\end{align*}
\]

so that this decoder can be implemented by updating the values of \( e^{(s-l)}_{j_t} \) at the \( l \)-th step. The final values for the erasures can be written as follows:
\[ e_{jl} = \left( \prod_{u=l+1}^{s} (x^{jt} - x^{ju}) \right)^{-1} e_{jl}^{(s-l)} \]

\[ = \left( \prod_{u=l+1}^{s} (x^{jt} - x^{ju}) \right)^{-1} \left( S_0^{(s-l)} - \sum_{t=0}^{l-1} \left( \prod_{u=l+1}^{s} (x^{jt} - x^{ju}) \right) e_{jt} \right) \]

\[ = \left( \prod_{u=l+1}^{s} (x^{jt} - x^{ju}) \right)^{-1} S_0^{(s-l)} - \sum_{t=0}^{l-1} \left( \prod_{u=l+1}^{s} \frac{x^{jt} - x^{ju}}{x^{jt} - x^{ju}} \right) e_{jt} \]

for \( 1 \leq l < r \).
Bibliography


