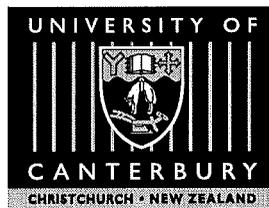


Turbo Codes: Convergence Phenomena & Non-Binary Constructions

A thesis submitted in fulfilment
of the requirements for the Degree of

Doctor of Philosophy

in Electrical and Electronic Engineering
from the University of Canterbury
Christchurch, New Zealand



Andrew Carey Reid
B.E. (Hons I)

June 18, 2002

Abstract

The introduction of turbo codes in 1993 provided a code structure that could approach Shannon limit performance whilst remaining practically decodeable. Much subsequent work has focused on this remarkable structure, attempting to explain its performance and to extend or modify it. This thesis builds on this research providing insights into the convergence behaviour of the iterative decoder for turbo codes and examining the potential of turbo codes constructed from non-binary component codes.

The first chapter of this thesis gives a brief history of coding theory, providing context for the work. Chapter two explains in detail both the turbo encoding and decoding structures considered. Chapter three presents new work on convergence phenomena observed in the iterative decoding process. These results emphasise the dynamic nature of the decoder and allow for both a stopping criteria and ARQ scheme to be proposed. Chapters four and five present the work on non-binary turbo codes. First the problem of choosing good component codes is discussed and an achievability bound on the dominant parameter affecting their performance is derived. Searches for good component codes over a number of small rings are then conducted, and simulation results presented. The new results, and suggestions for further work are summarised in the conclusion of Chapter six.

part 4

Acknowledgments

I would like to thank all of those who have helped make my doctorate possible.

Firstly I would like to thank my wife for her continued support and efforts to buoy me up throughout the course of this work.

I also wish to thank my parents and family for their love and support, for the interest they have shown and encouragement they have given.

Special thanks are also due to my supervisors. Thanks to Aaron Gulliver for his trans-Pacific efforts on my behalf, especially for his insights and time spent checking my work. Thanks to Des Taylor for his direction and invaluable help in preparing work for publication.

Thanks to Michael Fielding and Paul Davidson for being awesome friends, great flatmates and for many evenings talking and drinking. Thanks also to Edward Walsby for his incisive wit and the many hours spent distracted in front of his monitor.

I am much obliged to all my labmates past and present for their companionship and many hilarious moments.

I wish to acknowledge the University of Canterbury and the Marsden Fund for their financial support over the course of this research.

10/10/2010

Contents

1	Introduction	1
1.1	Digital Communications	1
1.2	Coding Theory	3
1.3	Channel Codes	4
1.4	Turbo Codes	5
1.5	Thesis Content	8
1.6	Thesis Contributions	8
2	System Overview	11
2.1	Basic structure	11
2.2	Weights and Distances	12
2.2.1	Basics	12
2.2.2	Linearity and WEFs	13
2.2.3	Code Performance	13
2.3	Mathematics	14
2.3.1	Rings and Fields	14
2.3.2	Arithmetic tables	15
2.3.3	Log-likelihood algebra	15
2.4	Symbol Source	18
2.5	The Encoder	19
2.5.1	Overall Structure	19
2.5.2	Encoder Description	19
2.5.3	Component Encoders	20
2.5.4	Interleaver	21
2.5.5	Frame termination	22
2.6	Transmission and Reception	22
2.6.1	The transmitter	23
2.6.2	The effect of a noisy channel	24
2.6.3	The receiver	24
2.7	The Decoder	26

2.7.1	What is known	26
2.7.2	The iterative decoder	26
2.7.3	The constituent decoders	27
3	Decoder Convergence Study	31
3.1	Introduction	31
3.2	Interleavers	33
3.2.1	The Role of the Interleaver	33
3.2.2	Interleaver Design	34
3.3	Modes of Convergence and Error Events	35
3.3.1	Mode 1 – Quick stable convergence	37
3.3.2	Mode 2 – A small number of bits converge differently	37
3.3.3	Mode 3 – Oscillation	40
3.3.4	Long-Term Performance	43
3.4	Decoder Termination	45
3.5	Selective Repeat ARQ	50
3.6	Conclusion	51
4	Non-binary turbo codes	53
4.1	Introduction	53
4.2	Non-binary Turbo Codes	53
4.3	Choosing ‘good’ component codes	55
4.3.1	What makes a good component code?	56
4.3.2	Recursive codes	56
4.3.3	Causality	57
4.3.4	Further Constraining b_0	58
4.3.5	Primitive Polynomials	58
4.4	Procedure for determining ‘best’	59
4.5	Search algorithm	59
4.5.1	Discussion	60
4.6	Lower bound on z_{\min}	61
4.7	Geometric Uniformity	63
4.8	Conclusion	63
5	Searches and Simulations	65
5.1	Binary Codes	65
5.1.1	The ‘best’ component codes	65
5.1.2	Simulation and bounds	66
5.1.3	Alternative ‘best’ codes	68
5.1.4	Very short frame lengths	71

5.2	Ternary Codes	72
5.3	Quaternary Codes	76
5.3.1	Hamming distance searches	76
5.3.2	Euclidean distance searches	78
5.4	Penternary Codes	80
5.5	Hexernary and Octernary Codes	82
5.6	Observations	83
5.7	Conclusion	83
6	Conclusion	85
6.1	Results	85
6.2	Suggestions for further work	86
A	Simulating the channel	87

Chapter 1

Introduction

This chapter introduces the technological and historical context of the research.

1.1 Digital Communications

Communications engineering encompasses both the theoretical and practical problems associated with transferring information.

Knowledge in this area has led to many of the communication systems which are prominent facets of our society, such as telephones, television, radio and the internet [48]. Each of these systems can involve many different methods of transferring information, such as down a wire using electrical signals, through the atmosphere using electromagnetic signals or through glass fibre using optical signals. Additionally, the information to be transferred can be in either analogue form (such as speech waveforms), or digital form (usually a string of binary digits). The majority of new communications systems are designed and implemented to transfer digital information. These either transfer purely digital data (such as two computers communicating), or they convert analogue information, representing it digitally for transfer, (such as voice transmission on a digital cellphone network). A single link in a simple digital communications system is represented in the block diagram figure of 1.1. The role of each of these blocks is described briefly below. Extended details of system and assumptions specific to this thesis are in Chapter 2.

The information source provides the data for transmission (propaganda for the television news [38], or a word processor document for example). The source encoder takes this information and represents it in the digital (usually binary) alphabet expected by the channel encoder. It also compresses the data stream to remove redundancy ideally resulting in a source of independent, random, equiprobable digits.

The channel encoder now adds redundancy to the data stream prior to transmission. However, in contrast to the redundancy removed by the source encoder, this new

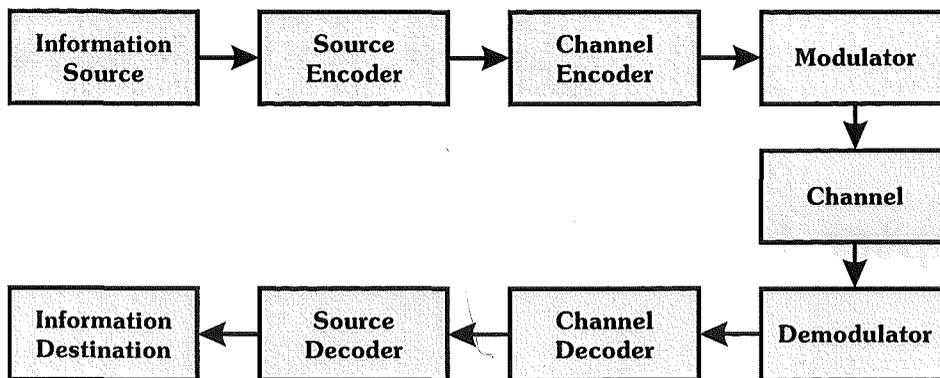


Figure 1.1: Simple block diagram of a digital communications system.

redundancy is added in a known and controlled fashion with the object of allowing errors introduced by the channel to be detected and often corrected by the channel decoder. There are many methods of doing this for different situations, with varying complexity and success. It is the study and extension of a particular method of channel encoding and decoding that is the topic of this thesis.

The modulator takes the encoded stream of digital data and converts it into a form suitable for transmission through the channel. Again, there are many ways of doing this. For example, when modems ‘talk’ over a phone line or digital cell phones transmit through the atmosphere, the modulator must convert the data stream into suitable electromagnetic waves for transmission. Throughout this theses a modulation scheme known as phase shift keying (PSK) is used [36]. In this case electromagnetic signals of a particular frequency are transmitted at with different phase offsets to represent each of the symbols in the code alphabet.

The channel also takes many forms that affect the transmitted signal in different ways. Examples of real channels include the copper wire of telephone lines and computer networks or the atmosphere between a cellphone and a celltower. As the transmitted signal passes through the channel medium, it is distorted. This may be due to thermal noise or interference from other transmitted signals, for example. Theoretical research usually considers an idealised channel or channel model. One of the simplest is the *additive white Gaussian noise* (AWGN) channel, which only allows for additive random (thermal) noise, and has no memory of what has previously passed through the channel. This model is used for the research of this thesis (see section 2.6.2). More complex models attempt to recreate the interference and multiplicative noise observed in most real channels.

In the receiver, the demodulator, channel decoder and source decoder attempt to reverse the steps of their transmitter counterparts. This involves a compromise between accuracy and complexity, but at each stage some kind of ‘best’ estimate of the data stream at the equivalent encoding stage is provided.

The quality of the information passed between the blocks in figure 1.1, can greatly affect overall performance. This is particularly true of the information passed between the demodulator and decoder. In so-called hard decision decoding, the demodulator makes a firm decision about which symbols were transmitted and passes this information to the channel decoder. Essentially, the output of the channel is quantised to represent only the possible transmitted symbols and this causes irreversible loss of information. When more complex soft decision decoding is used, the demodulator is capable of detecting a far wider (possibly continuous) variation in the received signals, and can therefore provide some kind of probability/reliability information along with its decisions. This method recognises that the demodulator can be more certain about some transmitted symbols than others. This process is depicted in figure 1.2. A channel decoder that makes use of this soft information can perform substantially better than one that does not [63].

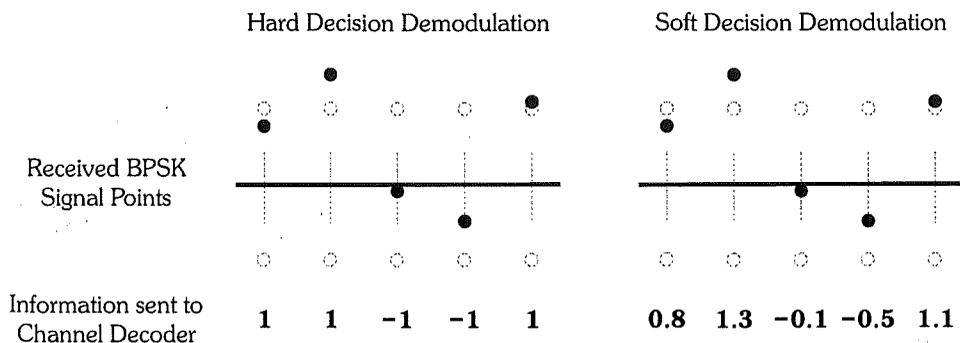


Figure 1.2: The difference between passing hard and soft information.

1.2 Coding Theory

The research presented in this thesis concerns only the channel encoder and decoder. Simple ideal models for the other components of the system are used.

The role of the channel encoder is to add redundancy to the data stream in such a manner that errors introduced by the channel can be detected and hopefully corrected. A simple example of such a scheme is a repetition code [36]. Consider the case where each information symbol is transmitted three times. The decoder then knows that an error has occurred if the three received symbols are not identical. Additionally, for binary symbols, many of the errors can be corrected by a simple majority vote. This code, like all channel codes, cannot detect or correct all possible error patterns, but it will improve the reliability of the system by detecting and correcting the most common error patterns.

When a detectable error is encountered by the channel decoder, there are two choices; the system can request the information be re-sent, hopefully error free, or a best estimate of the transmitted data can be made despite the errors. The first of these methods

involves an *automatic repeat request* scheme (ARQ) [43] in conjunction with a channel code that is designed primarily to detect errors. The second method involves using a *forward error correcting* (FEC) code which is designed to allow both the detection and correction of the most likely errors. Many real systems employ a hybridisation of these two methods.

The improved reliability provided by channel codes comes at a cost. In the repetition code example, each symbol was transmitted three times; so if symbols are transmitted at a fixed rate, it now takes three times as long to transmit the data, or a more complex system that can transmit three symbols at once is required. Alternatively, the symbols could be transmitted three times as fast, but this would require three times the bandwidth.

In 1948, Shannon [52] proved statistically that there was a limit to the increase in reliability possible from using a channel code that added a fixed proportion of redundancy. He showed that every channel has a theoretical maximum capacity, defined as a rate at below which error free data transfer is possible. Conversely, for a fixed rate of transmission, provided the channel introduces no more than a certain amount of noise, error free data transfer is possible. Furthermore, Shannon stated that as we approximate an ideal code more closely, “the transmitted signal approaches a white noise in statistical properties”, and “the required delays at transmitter and receiver increase indefinitely” [53]. This implies that the encoded sequences must become both *long* and *random* in nature for a code to approach capacity. Shannon’s proofs were based on statistical entropy, and non-constructive in nature, however they showed what was possible.

1.3 Channel Codes

There are two basic classes of FEC code, block codes and convolutional codes. Block codes take a fixed number of information symbols, add parity symbols and send the block (codeword) to the modulator, before repeating. The alternative is convolutional codes which operate on a continuous stream of information symbols and splice in parity symbols. In this case, there is no predefined block length.

The earliest practical error correcting codes were Hamming codes introduced in 1947 [34]. These are a class of linear binary block codes which are capable of detecting up to two errors and correcting one. The pre-eminent example is the (7,4) binary Hamming code which takes sets of four binary digits and adds controlled redundancy in the form of three parity bits. So for every four information bits seven code bits are transmitted, and the code rate is said to be 4/7.

More complex algebraic block codes are capable of detecting and correcting many more errors. *Bose-Chaudhuri-Hocquenghem* (BCH) codes, *Reed-Solomon* (RS) codes and the perfect Golay codes are examples of such codes. Of particular interest to this thesis is that fact that some of these codes may be non-binary. For example there exists a

perfect ternary Golay code, and RS codes are generally constructed over an alphabet of 2^n symbols.

The first convolutional codes were proposed in 1955 by Elias [26]. A convolutional encoder takes the form of a finite state machine, a continuous stream of input digits are fed in, and an expanded sequence of encoded symbols is read out. Such codes were first decoded using sequential decoders, but became far more practical with the application of the so-called Viterbi algorithm to decoding [60], which was shown to be a *maximum likelihood* (ML) sequence decoding method [28]. That is, the Viterbi algorithm will determine the most likely (probabilistically) transmitted data sequence given the received symbols. The 1974 *Bahl-Cocke-Jelinek-Raviv* (BCJR) algorithm [3] provides another optimal decoding method. The BCJR algorithm is a *maximum a posteriori* (MAP) algorithm. This is subtly different to ML decoding as a MAP algorithm determines the most likely data symbols individually for each symbol, rather than the most likely sequence.

Non-binary convolutional codes soon followed [16, 56], and their suitability for PSK modulation was noted by Massey and Mittelholzer [44] who suggested that “the ‘natural’ linear codes for q-ary phase modulation are linear codes over the ring of integers modulo q , \mathbb{Z}_q ”. Baldini and Farrell [4] further developed this premise, and identified convolutional codes with good distance properties over \mathbb{Z}_4 , \mathbb{Z}_8 and \mathbb{Z}_{16} suitable for 4-, 8- and 16-PSK modulation respectively. It is this work that suggested the development of non-binary turbo codes for q-ary PSK modulation.

The standard block and convolutional codes mentioned above have been extensively analysed, and the principles of their structure, encoding, decoding, and performance are well understood [63, 13, 61]. Amongst other things this has resulted in an understanding of the properties that lead to both good and bad codes. Of particular interest to this thesis are the concepts of weight enumerators and code polynomials and how these may be related to code performance [37]. (See section 2.2 and chapter 4.) Such basic tools and measures help in the design and analysis of more complex codes such as turbo codes which are not well understood.

1.4 Turbo Codes

Whilst simple block and convolutional codes provide some performance gain over uncoded modulation, they do not allow systems to approach the capacity of the channel whilst remaining decodeable in reasonable time. More complex codes are needed, but they must remain practically decodeable. One method of achieving this is to concatenate simple codes in a serial fashion and then decode them in reverse order [27]. An example of this is the encoding system depicted in figure 1.3 which was used aboard the 1997 Mars Pathfinder space probe [35].

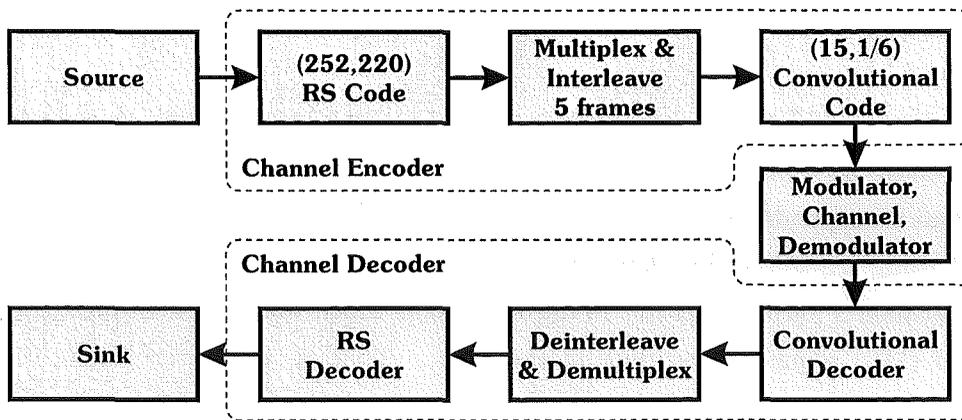


Figure 1.3: Block diagram of the channel coding strategy used for the 1997 Mars pathfinder mission.

Turbo codes are an extension of this idea, where simple codes are used to build up a more complex code, which because of its component code structure, can still be decoded in reasonable time.

The discovery of Turbo codes in 1993 [14] signalled a major breakthrough in channel coding. Turbo codes were the first practical codes that came close to the Shannon limit at moderate *bit error rates* (BERs). Figure 1.4 shows the substantial coding gain achieved by turbo codes over their competitors at similar rates. Not only do they out-perform the earlier codes, they do so at reasonable complexity. For example, the rate 1/2 turbo codes have a similar complexity to the vastly inferior rate 1/2 Planetary Standard code [20].

The performance of, recently re-discovered low density parity check (LDPC) codes originally proposed by Gallager in 1962 [30] is also presented in figure 1.4. These are another class of practically decodeable codes that exhibit the properties required to approach capacity, but do so at much higher rates than turbo codes. In fact LDPC codes that operate within ε dB ($\varepsilon \ll 1$) of capacity have been demonstrated [19, 42].

Since 1993, many papers have been written discussing various aspects of turbo-codes and their variants. This includes, but is not limited to, papers that discuss:

- Why the codes perform as well as they do, *e.g.* [5].
- Different concatenation structures, *e.g.* [7].
- Suboptimal component decoders and decoding strategies, *e.g.* [8].
- Punctured turbo codes, and turbo codes with different numbers and complexity of component codes, *e.g.* [57].
- Performance in fading channels, *e.g.* [33].
- When to stop decoding, *e.g.* [54, 50].

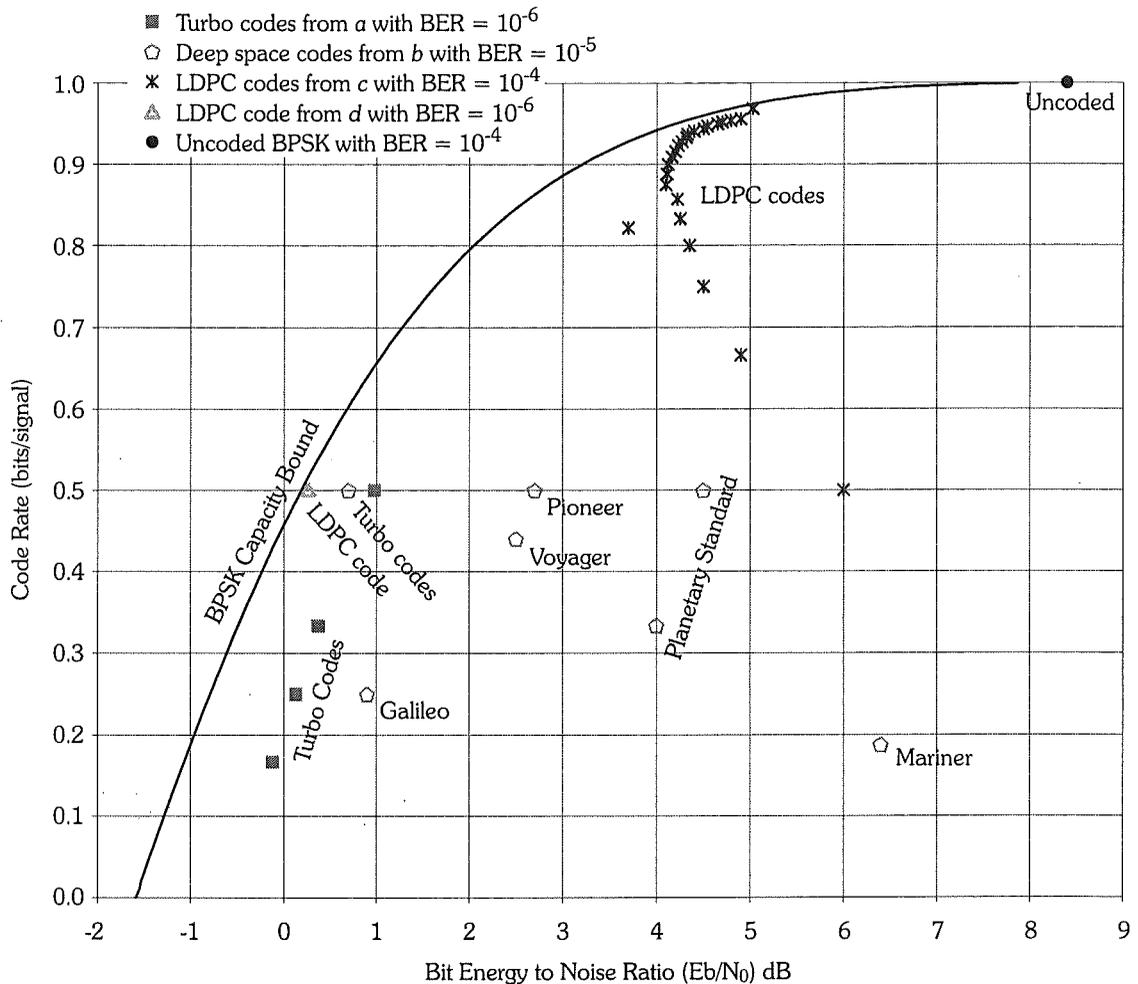


Figure 1.4: Capacity curve for the binary input continuous output channel with additive white Gaussian noise. The performance of various codes are shown, demonstrating the remarkable performance of turbo codes for rates less than $1/2$. The sources cited are a [24], b [20], c [42] and d [19].

- The convergence properties of the decoder and how to model this, *e.g.* [51, 1, 50].
- Extrinsic information and what it means, *e.g.* [32, 17].
- Performance Bounds, *e.g.* [6, 21, 25].

Despite the vast body of information published on turbo-codes since 1993, they are still in many senses enigmatic. In particular lots of optimisation questions remain, such as: How to design the best turbo-codes of a particular complexity? How to best pass extrinsic information between component decoders? What is the best interleaver structure? Can undesirable decoder convergence behaviours be avoided?

This thesis looks at two aspects of turbo code performance. Firstly, the various modes of convergence observed in the turbo decoding of binary turbo codes are shown and classified. This work provides a reference for the various convergence behaviours that should be expected for codes that use iterative decoding. It also allows for the

development of stopping criteria for the decoding process. Secondly, The potential of turbo codes based on non-binary component codes is examined. Previously such codes have been largely overlooked as candidates for providing good overall performance with simple component codes. The results of this work show that non-binary codes are very good candidates when trying to balance complexity with performance.

1.5 Thesis Content

The content of this thesis is divided into three major chapters.

Chapter 2 details the code structure, the encoding and decoding of non binary turbo codes, and much of the background material needed to understand these processes. It is unlikely that any of this chapter represents much new content, especially with regard to the work of Berkmann [12]. However, given the lack of information available on the subject this provides a necessary background resource.

Chapter 3 investigates the convergence phenomena encountered during the course of research toward this thesis. Where it is not explicitly referenced from another source, the chapter is original research. When this work was initially submitted for publication (September 1999) it was not widely accepted that the turbo decoding structure would exhibit oscillatory behaviour in any non-degenerate situation. Significant new works in this area by ten Brink [17] and also by Agrawal and Vardy [1] did not form part of the basis for this portion of research as they post date its completion. They do however provide additional theoretical justification for the results presented.

Chapter 4 examines the potential for turbo codes based on non-binary component codes. This chapter contains a substantial portion of new work. The standard binary results are not new, and the majority of credit for the metric and search criteria must go to Benedetto, Garelo and Montorsi [9]. Additionally, a small number of results from White and Costello [62] are reproduced. In particular, the work on alternate binary codes, short frame lengths and all the non-binary codes except where stated is new, as is the non-binary bound on z_{\min} , the smallest output Hamming weight possible from a weight two input to a component code.

1.6 Thesis Contributions

This thesis contributes new knowledge in the areas of turbo code convergence and non binary turbo codes. The following works have been published/submitted as a result of the research in this thesis:

- A. C. Reid, T. A. Gulliver and D. P. Taylor, "Convergence and errors in turbo-decoding," *Proc. IEEE Int. Symp. on Inform. Theory*, Washington DC, USA, June 2001.

- A. C. Reid, T. A. Gulliver and D. P. Taylor, "Convergence and errors in turbo-decoding," *IEEE Trans. Commun.*, vol. 49, no. 12, pp. 2045–2051, 2001
- A. C. Reid, D. P. Taylor and T. A. Gulliver, "Non-binary turbo codes," to be presented at *IEEE Int. Symp. on Inform. Theory 2002*.
- A. C. Reid, T. A. Gulliver and D. P. Taylor, "Non-binary turbo codes," submitted to *IEEE Trans. Inform. Theory*, October 2001.

Chapter 2

System Overview

This chapter explains the structure and workings of the particular turbo encoder and decoder considered in this thesis. After a brief overview, basic code concepts and non-binary arithmetic are introduced, and then the details of the information source, modulator, channel, demodulator and iterative decoder are established.

2.1 Basic structure

This section briefly introduces the basic turbo code encoding and decoding structures in order to aid the more complete descriptions that follow.

The turbo encoder is constructed from the parallel concatenation of recursive systematic convolutional (RSC) component codes. This thesis considers only rate $1/3$ turbo codes constructed from the concatenation of two identical rate $1/2$ RSC codes. Such an encoder is depicted in figure 2.1. This is a systematic structure. The source symbols are transferred directly to the output accompanied by two sets of parity data from the parity outputs of each constituent encoder. To ensure that the sets of parity data are different (close to independent), the source data is shuffled or interleaved in a random, but known manner before being encoded by the second constituent encoder. The inclusion of this random interleaving stage provides most of the complexity of the turbo code structure.

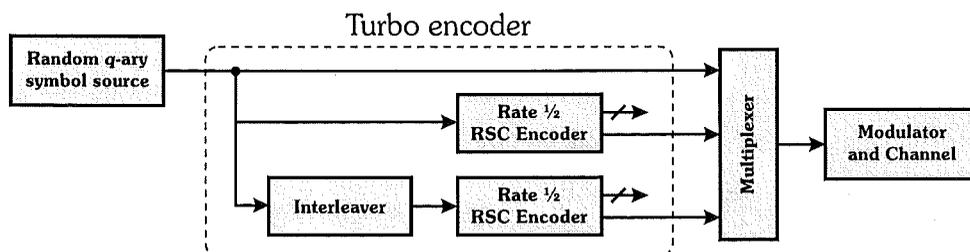


Figure 2.1: Structure of a rate $1/3$ turbo encoder.

At the receiver, after demodulation, an iterative decoding structure is used. This

is depicted in figure 2.2 for one iteration. It is this decoding structure that makes the decoding of turbo codes practical. The first component decoder decodes only the first component code, using the received systematic symbols and the first set of parity symbols. The so called *extrinsic* information from this stage is used to bias the *a priori* input for the second decoder, which decodes the second component code using the received systematic symbols and the second set of parity symbols. The results from this stage are then used to re-bias the first component decoder, and so the process iterates. At any point, the MAP output of the decoder provides an overall decision after the number of iterations that have been performed. This process is fully explained in section 2.7.

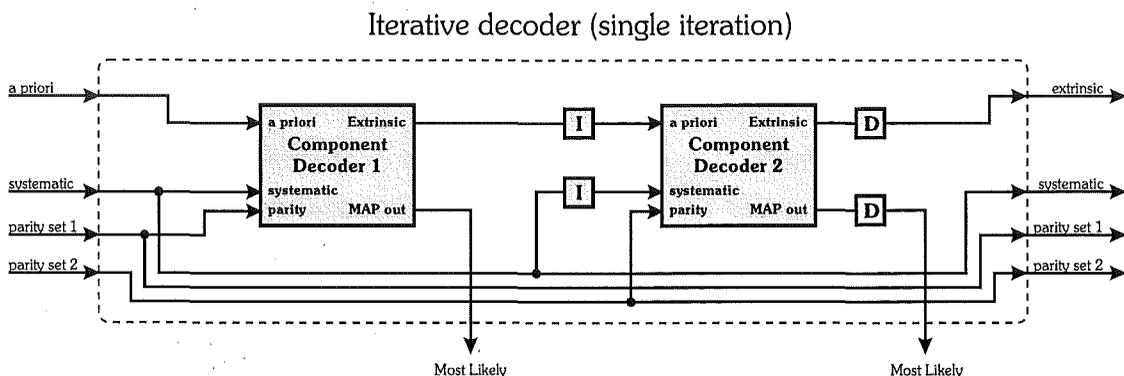


Figure 2.2: Block diagram showing one iteration of the turbo decoder. The ‘I’ and ‘D’ blocks represent interleaving and de-interleaving.

2.2 Weights and Distances

The analysis of turbo codes, the construction of interleavers and methods for choosing good component codes all revolve around considering how the turbo code will respond to low weight inputs as determined by its weight enumerating function (WEF). These concepts need to be defined.

2.2.1 Basics

The Hamming weight of a binary sequence (or codeword or input) is simply the number of 1’s in the sequence. The Hamming distance between two sequences is the number of positions in which the two sequences differ.

For the majority of this thesis only Hamming weights and distances are used. However, in the later portion of chapter 4 where quaternary and higher codes are considered, Euclidean distance is used to measure the distance between two sequences. This measure takes into account the different distances between points in quaternary and higher level signalling constellations. (see section 5.3.)

2.2.2 Linearity and WEFs

Every useful FEC code can be considered as a one-to-one mapping of a set of all possible input sequences of length k onto a set of possible output sequences C of length n where $n > k$. Such a code is said to be linear if the sum of any two output sequences in C is also a member of C .

A consequence of this is that the codeword (output sequence) $\bar{0}$ consisting of all 0's must be a member of every linear code. Additionally, if a codeword is chosen at random, and the distances between it and all other codewords enumerated, this set of distances will be the same no matter which codeword is chosen. Furthermore, if these distances are enumerated with respect to $\bar{0}$, then this is equivalent to just enumerating the weights of all the codewords in C , because the distance between a codeword c and $\bar{0}$ is just the weight of c .

All the codes considered in this thesis are linear. Thus for any linear code, the *weight enumerating function* (WEF) of the code is an enumeration of the distances between codewords, and may be simply determined by enumerating the weights of all codewords.

2.2.3 Free Distance, Nearest Neighbours and Code Performance

The weight enumerating function of a code can be related to its performance. In particular, errors between two codewords are most likely to occur if the codewords are very similar. For this reason, the smallest distance between codewords, known as the minimum distance or free distance of the code is very important.

Also, if there are a lot of alternate codewords (nearest neighbours) close to the transmitted codeword rather than just a few, an error is more likely to occur. Thus the number of nearest neighbours is also very important.

In fact these two parameters dominate the performance of concatenated codes at different *signal to noise ratios* (SNRs) as shown in figure 2.3. The \blacklozenge code has a superior minimum distance c compared to the \bullet code with minimum distance a . However the \blacklozenge code has many more low weight codewords and hence more nearest neighbours than the \bullet code. At lower SNR's in the 'waterfall' region of the performance curve, the number of nearest neighbours dominates code behaviour. (\bullet out performs \blacklozenge .) At high SNR's in the 'floor' region for the performance curve, the free distance of the code dominates performance. (\blacklozenge out performs \bullet .)

Turbo codes typically operate in the 'waterfall' region of the curve and therefore it is the the number of nearest neighbours or low weight codewords that dominate the code's performance. (See section 4.3.)

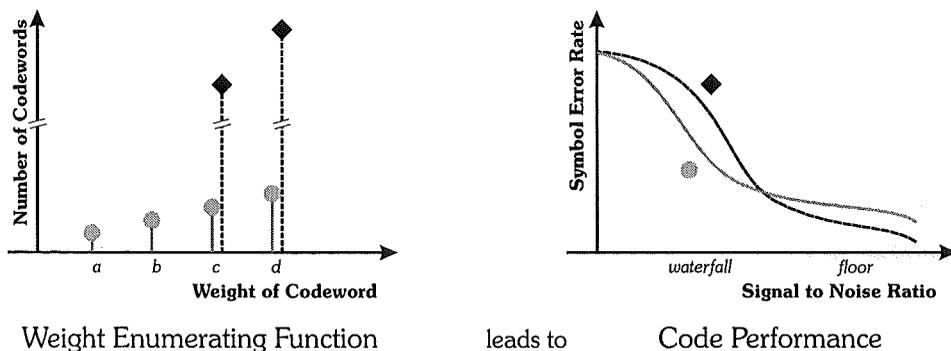


Figure 2.3: Diagram indicating how the minimum distance and number of low weight codewords effect performance.

2.3 Mathematics

2.3.1 Rings and Fields

Binary codes are defined using arithmetic over the Galois field $\text{GF}(2) \equiv \mathbb{Z}_2 \equiv \mathbb{F}_2$ which contains only two elements. Codes over larger sets of elements are possible. In particular, this thesis considers codes over a number of (small) integer rings.

A ring R consists of a set of elements and two operators: addition (+) and multiplication (\times), such that $\forall a, b, c \in R$

$$a + b \in R \quad (2.1)$$

$$a \times b \in R \quad (2.2)$$

$$(a + b) + c = a + (b + c) \quad (2.3)$$

$$a + b = b + a \quad (2.4)$$

$$\exists \text{ an additive identity } 0 \in R : a + 0 = a \quad (2.5)$$

$$\exists \text{ an additive inverse } -a \in R : a + (-a) = 0 \quad (2.6)$$

$$(a \times b) \times c = a \times (b \times c) \quad (2.7)$$

$$a \times (b + c) = (a \times b) + (a \times c) \quad (2.8)$$

There always exists at least one ring of q elements. In most cases, several rings of size q exist. We denote \mathbb{Z}_q as the *integer residue ring* of size q , consisting of the elements $\{0, 1, \dots, q - 1\}$ over which all addition and multiplication is carried out modulo- q .

Fields are a subset of rings which exhibit the following additional properties: $\forall a, b \in F$ a field.

$$a \times b = b \times a \quad (2.9)$$

$$\forall a \neq 0, \exists \text{ an multiplicative identity } 1 \in F : a \times 1 = a \quad (2.10)$$

$$\forall a \neq 0, \exists \text{ an multiplicative inverse } a^{-1} \in F : a \times a^{-1} = 1 \quad (2.11)$$

There exists a maximum of one distinct field of q elements \mathbb{F}_q , and for some q no field of q elements exists (e. g. $q = 6$). Specifically if $q = p^n$, p prime, n an integer, then a field of q elements exists. Additionally if q is prime then $\mathbb{F}_q \equiv \mathbb{Z}_q$.

The particular symbols used to represent elements of a ring or field and their associated binary operators are inconsequential. The integers $\{0, 1, \dots, q-1\}$ and the operators $+$ and \times are used because of their intuitive fit to the required properties.

2.3.2 Arithmetic tables

The construction and decoding of non-binary turbo codes only requires the operations multiplication, addition and subtraction. This means that turbo codes can be constructed over any ring R .

In this thesis, codes are constructed over $\mathbb{F}_2, \mathbb{F}_3, \mathbb{Z}_4, \mathbb{F}_4, \mathbb{F}_5, \mathbb{Z}_6, \mathbb{F}_8$ and also the other possible rings of four elements denoted $\mathbb{F}_2 + u\mathbb{F}_2$ and $\mathbb{F}_2 + v\mathbb{F}_2$.¹ These are all possible rings of two to five elements, and one ring each of six and eight elements. Arithmetic tables for the four rings of four elements are given in table 2.1.

2.3.3 Log-likelihood algebra

The structure of the iterative decoding algorithm used in the decoder depends heavily on the concept of log-likelihoods, and vectors of log-likelihoods.

Notation

The notation $L_{G,ab}(g)$ is used to represent the *log-likelihood ratio*

$$L_{G,ab}(g) = \ln \left(\frac{\mathbb{P}(g = a)}{\mathbb{P}(g = b)} \right) \quad (2.12)$$

where g is a sample of the random variable G . Clearly the sign of $L_{G,ab}(g)$ represents a hard decision as to whether g is more likely to be a or b , while the magnitude represents the strength of that decision. This notation is also used in the time varying case

$$L_{G,ab}(g) = \ln \left(\frac{\mathbb{P}(g(t) = a(t))}{\mathbb{P}(g(t) = b(t))} \right) \quad (2.13)$$

as inclusion of the independent variable t would be too cumbersome. When it is obvious that samples g of a variable G are being considered the additional subscripts are dropped leaving $L_{ab} \equiv L_{G,ab}(g)$.

Additionally the bracketed notation $\{L\}$ is used to represent the set of log-likelihood ratios $L_{i0}, \forall i$, and the bolded notation \bar{L} to represent a time indexed vector of these sets.

¹Specifically, $\mathbb{F}_2 + u\mathbb{F}_2$ consists of the elements $\{0, 1, u, 1 + u\}$ where $u^2 = 0$ and $\mathbb{F}_2 + v\mathbb{F}_2$ consists of the elements $\{0, 1, v, 1 + v\}$ where $v^2 = v$. In table 2.1, integers are chosen for u and v which meet the required properties.

+	0	1	2	3	x	$-x$	×	0	1	2	3
0	0	1	2	3	0	0	0	0	0	0	0
1	1	2	3	0	1	3	1	0	1	2	3
2	2	3	0	1	2	2	2	0	2	0	2
3	3	0	1	2	3	1	3	0	3	2	1

(a) \mathbb{Z}_4 arithmetic tables

+	0	1	2	3	x	$-x$	×	0	1	2	3
0	0	1	2	3	0	0	0	0	0	0	0
1	1	0	3	2	1	1	1	0	1	2	3
2	2	3	0	1	2	2	2	0	2	3	1
3	3	2	1	0	3	3	3	0	3	1	2

(b) \mathbb{F}_4 arithmetic tables

+	0	1	2	3	x	$-x$	×	0	1	2	3
0	0	1	2	3	0	0	0	0	0	0	0
1	1	0	3	2	1	1	1	0	1	2	3
2	2	3	0	1	2	2	2	0	2	0	2
3	3	2	1	0	3	3	3	0	3	2	1

(c) $\mathbb{F}_2 + u\mathbb{F}_2$ arithmetic tables

+	0	1	2	3	x	$-x$	×	0	1	2	3
0	0	1	2	3	0	0	0	0	0	0	0
1	1	0	3	2	1	1	1	0	1	2	3
2	2	3	0	1	2	2	2	0	2	2	0
3	3	2	1	0	3	3	3	0	3	0	3

(d) $\mathbb{F}_2 + v\mathbb{F}_2$ arithmetic tables

Table 2.1: Arithmetic tables for the rings of four elements

For example, consider a quaternary turbo code² with a frame length of 5 systematic symbols. The log-likelihoods associated with the systematic symbols of a particular frame as supplied by the demodulator will be denoted \bar{L}_{sys} with the notation

$$\bar{L}_{\text{sys}} = (\{L\}_{t=0} \{L\}_1 \{L\}_2 \{L\}_3 \{L\}_4) \quad (2.14)$$

while for $j = 0, 1, \dots, 4$

$$\{L\}_{t=j} = \{L_{10}, L_{20}, L_{30}\}_{t=j} \quad (2.15)$$

Conversion

The absolute probabilities

$$p_i = \mathbb{P}(g = i) \quad (2.16)$$

may be calculated from the log-likelihood ratios $\{L_G\}$. First note from (2.12) that

$$\frac{p_i}{p_0} = \exp(L_{i0}), \quad i = 1, 2, \dots, q-1 \quad (2.17)$$

from which it is immediate that

$$\sum_{i=1}^{q-1} p_i = p_0 \sum_{i=1}^{q-1} \exp(L_{i0}) \quad (2.18)$$

where q is the size of the ring. Furthermore, the sum of the probabilities must equal one

$$p_0 + \sum_{i=1}^{q-1} p_i = 1 \quad (2.19)$$

hence

$$\frac{1 - p_0}{p_0} = \sum_{i=1}^{q-1} \exp(L_{i0}) \quad (2.20)$$

so that

$$\Rightarrow p_0 = \frac{1}{1 + \sum \exp(L_{i0})} \quad (2.21)$$

Once p_0 has been determined it is a simple matter to obtain the other p_i via equation (2.17)

$$p_i = p_0 \exp(L_{i0}), \quad i = 1, 2, \dots, q-1. \quad (2.22)$$

²“Quaternary codes” is a general term used to refer to the set of codes whose arithmetic is over any ring of four elements.

Algebra

The iterative decoder adds and subtracts vectors of log-likelihoods \bar{L}_{sys} . This process is performed as follows:

$$\begin{aligned}
 \bar{L}_A + \bar{L}_B &= (\{L\}_{A,t=0}, \{L\}_{A,1}, \dots, \{L\}_{A,n}) + (\{L\}_{B,t=0}, \{L\}_{B,1}, \dots, \{L\}_{B,n}) \\
 &= (\{L\}_{A,t=0} + \{L\}_{B,t=0}, \{L\}_{A,1} + \{L\}_{B,1}, \dots, \{L\}_{A,n} + \{L\}_{B,n}) \\
 &= \left(\left(\begin{array}{c} L_{A,10} \\ L_{A,20} \\ \vdots \\ L_{A,(q-1)0} \end{array} \right)_{t=0} + \left(\begin{array}{c} L_{B,10} \\ L_{B,20} \\ \vdots \\ L_{B,(q-1)0} \end{array} \right)_{t=0}, \dots \right) \\
 &= \left(\left(\begin{array}{c} L_{A,10} + L_{B,10} \\ L_{A,20} + L_{B,20} \\ \vdots \end{array} \right)_{t=0}, \dots \right) \tag{2.23}
 \end{aligned}$$

reducing to the summation of pairs of L_{i0} . In terms of probabilities

$$\begin{aligned}
 L_{A,i0} + L_{B,i0} &= \ln \left(\frac{\mathbb{P}(g=i)}{\mathbb{P}(g=0)} \right)_A + \ln \left(\frac{\mathbb{P}(g=i)}{\mathbb{P}(g=0)} \right)_B \\
 &= \ln \left(\frac{\mathbb{P}_A(g=i)\mathbb{P}_B(g=i)}{\mathbb{P}_A(g=0)\mathbb{P}_B(g=0)} \right) \tag{2.24}
 \end{aligned}$$

Most likely

At some point, it is necessary to turn probabilities into hard decisions by choosing the most likely symbol given a set probabilities or log-likelihood ratios (see section 2.7.2).

If this decision must be made from a set of probabilities p_i with $\sum p_i = 1$, then this is done by choosing the symbol i which has the largest corresponding p_i . If the probabilities are represented by a set of log-likelihood ratios $\{L\}$ then a decision can be made based on the L_{i0} directly. This is done by choosing the symbol i which corresponds to the largest positive L_{i0} , or if no L_{i0} are positive then $i = 0$ is the most likely symbol.

2.4 Symbol Source

An ideal q -ary data source is assumed in this thesis. Such a source provides a continuous stream of random uncorrelated q -ary symbols. This allows the decoder to initially assume equal *a priori* probabilities for each received symbol.

In practice this is a reasonable model for simulation as in any real system data can be compressed by the source encoder so that it closely approximates these assumptions. Computer simulation is not capable of producing truly random data, however the use of a good pseudo random number generator will closely approximate the ideal [49] with the additional advantage of having a known ‘seed’ or initial value which makes individual simulations repeatable.

2.5 The Encoder

2.5.1 Overall Structure

Only rate 1/3 turbo encoders with identical rate 1/2 *recursive systematic convolutional* (RSC) component encoders are considered throughout this thesis. The structure is identical to that used for binary turbo codes with the same parameters except q -ary symbols are used and all arithmetic is over a ring of size q . Such an encoder is depicted in figure 2.4.

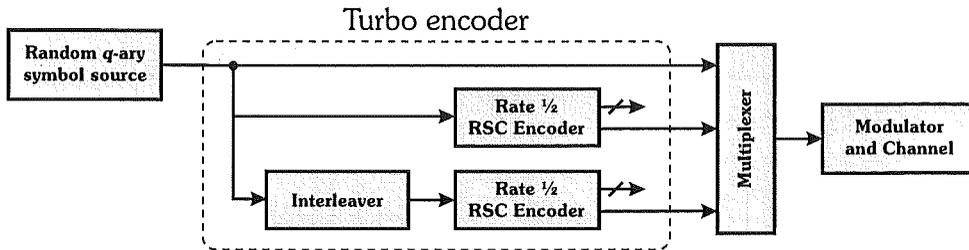


Figure 2.4: Structure of a rate 1/3 turbo encoder

The source provides a random stream of q -ary symbols which are divided up into frames of n symbols. The symbols in each frame are encoded using a rate 1/2 *recursive systematic convolutional* (RSC) encoder to produce one set of parity symbols. The systematic symbols are then randomly permuted or interleaved, and passed through a second RSC encoder to provide a second set of parity symbols. The systematic symbols and both sets of parity symbols are transmitted providing an overall code rate of 1/3.

2.5.2 Encoder Description

The notation used to describe both a particular component encoder and the turbo code constructed from it will be $\langle a_n \dots a_0 | b_n \dots b_0 \rangle$ where the a_i and b_i are coefficients of the numerator and denominator of the non-systematic portion of the component encoder matrix $G(D)$, giving

$$\langle a_n \dots a_0 | b_n \dots b_0 \rangle \triangleq \left[1 \quad \frac{a_n D^n + a_{n-1} D^{n-1} + \dots + a_1 D + a_0}{b_n D^n + b_{n-1} D^{n-1} + \dots + b_1 D + b_0} \right] \quad (2.25)$$

For example a quaternary 16-state component code is defined by

$$\langle 201 | 331 \rangle \triangleq \left[1 \quad \frac{2D^2 + 1}{3D^2 + 3D + 1} \right] \quad (2.26)$$

The systematic portion of the component code will be implied in the notation, with phrases such as ‘the code’ and ‘the encoder’ referring only to the non-systematic part of the code. In this sense we can think of a turbo codeword as consisting of the data symbols plus two sets of parity symbols generated by rate 1 recursive convolutional codes.

This notation does not explicitly state over which particular ring a code is defined, however this will always be clear from the context.

2.5.3 Component Encoders

If we only consider the case where $b_0 = 1$, *i.e.* codes of the form $\langle a_n \dots a_0 | b_n \dots b_1 1 \rangle$, then one block diagram for the non-systematic portion of this code is shown in figure 2.5. This is the structure used by Baldini and Farrell [4], and also by Kerr [41].

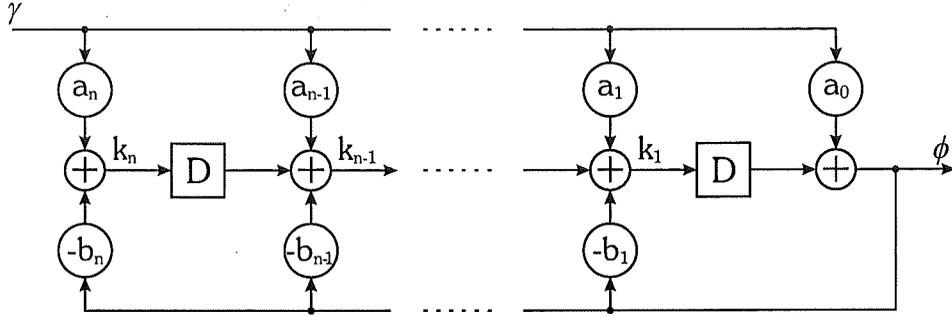


Figure 2.5: Block diagram implementation of the non-systematic portion of the encoder represented by equation 2.25. The circles containing the various α and β represent multiplication by a constant, the circles with + signs addition and the squares with D's, delay elements.

This implementation can be shown to be valid by noting that

$$\phi = k_1 D + a_0 \gamma \quad (2.27)$$

$$k_x = k_{x+1} D + a_x \gamma - b_x \phi \quad x = 1, 2, \dots, n-1 \quad (2.28)$$

$$k_n = a_n \gamma - b_n \phi \quad (2.29)$$

Combining these we see that

$$k_{n-\lambda} = \gamma \sum_{\mu=0}^{\lambda} a_{n-\mu} D^{\lambda-\mu} - \phi \sum_{\mu=0}^{\lambda} b_{n-\mu} D^{\lambda-\mu} \quad 0 < \lambda < n \quad (2.30)$$

$$k_1 = \gamma \sum_{\mu=0}^{n-1} a_{n-\mu} D^{n-1-\mu} - \phi \sum_{\mu=0}^{n-1} b_{n-\mu} D^{n-1-\mu} \quad (2.31)$$

hence

$$\phi = \gamma \sum_{\mu=0}^n a_{n-\mu} D^{n-\mu} - \phi \sum_{\mu=0}^{n-1} b_{n-\mu} D^{n-\mu} \quad (2.32)$$

This gives the required transfer function

$$\begin{aligned} \frac{\phi}{\gamma} &= \frac{\sum_{\mu=0}^n a_{n-\mu} D^{n-\mu}}{\sum_{\mu=0}^{n-1} b_{n-\mu} D^{n-\mu} + 1} \\ &= \frac{a_n D^n + a_{n-1} D^{n-1} + \dots + a_1 D + a_0}{b_n D^n + b_{n-1} D^{n-1} + \dots + b_1 D + 1} \end{aligned} \quad (2.33)$$

2.5.4 Interleaver

The interleaver is simply a permuter that shuffles the order of the systematic bits before they are encoded by the second component encoder. Many interleaver constructions exist. However as Shannon stated [53], to achieve near capacity performance the statistical properties of the transmitted signal must approach white noise. Because of this, interleavers that are random in nature lead to turbo-codes with better performance.

Three different types of pseudo-random interleaver are considered in this thesis.

- *Random*: Interleaver simply shuffles the symbols in the frame in the most random way possible
- *S-Random* [23]: Interleaver shuffles the symbols in the frame as randomly as possible with the constraint that no two symbols that were originally within S positions of each other may remain within S positions of each other.
- *Terminating* [10, 15]: Interleaver shuffles the symbols in the frame as randomly as possible with the constraint that the position of symbols in the frame must remain constant modulo m , where m is the period of the impulse response of the component encoder.³ This terminating interleaver design allows both component encoders to be terminated in a known state (see section 2.5.5).

In the turbo code structure, the role of the interleaver is to attempt to permute data sequences that produce low weight parity output from the first component encoder onto sequences that will produce high weight parity output from the second component encoder (see section 3.2.1).

Both the S-random and terminating interleaver designs choose from a subset of random interleavers with the object of choosing interleavers that appear random, but possess additional useful properties. Unfortunately, the terminating design, despite its nice termination properties, is necessarily bad (see section 3.2.2) because it is less likely to permute low weight producing sequences into high weight producing sequences. However, the S-random interleaver does indeed provide a better than average construction. Because

³For a good component code over a ring of size q with ν memory elements $m = q^\nu - 1$.

of this S-random interleavers are used for all the simulations of chapter 5 where best performance is the primary objective.

2.5.5 Frame termination

The encoding strategy used always starts by placing both component encoders in the 0 state.⁴ As this is known, it can be used by the decoder to provide a slight performance gain over the case where initial states are unknown. A similar situation exists at the end of each frame. Once an entire information frame has passed through the encoder, the states of both component encoders will be a function of their initial states, the information symbols, and for the second encoder, the interleaver. The possibility exists for appending some termination symbols to the information frame to ensure that one or both of the component codes return to a known state which can be assumed by the decoder. This process is called frame termination, and three different termination strategies are considered in this thesis, namely,

- *None*: No termination symbols are added to the information frame, and the best the decoder can do is assume that all possible final states are equally likely.
- *Single* [22]: A convolutional encoder with ν memory elements can always be forced to the 0 state by a state dependent sequence of ν input symbols. This strategy examines the state of the first component encoder after the information frame has passed through, and appends the appropriate zero forcing termination sequence to the information frame. Thus the decoder can assume that the first component encoder finished in the 0 state, however no assumption can be made about the final state of the second component encoder.
- *Double* [10, 15]: The use of a *terminating* interleaver guarantees that the same ν long termination sequence will force both component encoders to finish in the 0 state.

As discussed in the previous section, the terminating interleaver design produces bad interleavers and hence practical application of the double termination method is not feasible. For this reason the performance simulations presented in chapter 5 use the single termination method presented in [22] as some knowledge of the encoders terminating state is better than none.

2.6 Transmission and Reception

The transmitter must take the data produced by the encoder and convert it into a form suitable for transmission through the channel. Conversely the receiver must take the

⁴The 0 state is chosen only for convenience. The important point is that it is a known state.

corrupted signals from the channel and put them in a form suitable for decoding.

2.6.1 The transmitter

The encoder produces a stream of q -ary symbols which the transmitter must modulate and send through the channel.

The modulation considered throughout this thesis is q -ary PSK [36] where q is the size of the ring being used by the code. This makes the mapping of data symbols to constellation points straightforward.⁵ The phase of the carrier takes on one of q possible values, and during each signalling interval of duration T one of q signals

$$s_i(t) = \sqrt{\frac{2E}{T}} \cos\left(2\pi f_c t + \frac{2\pi}{q}i\right), \quad \begin{cases} i = 0, 1, \dots, q-1 \\ f_c = n_c/T, n_c \in \mathbb{Z} \\ 0 \leq t \leq T \end{cases} \quad (2.34)$$

is transmitted, where E is the signal energy per symbol. This results in a two-dimensional signal constellation with

$$\phi_I(t) = \sqrt{\frac{2}{T}} \cos(2\pi f_c t), \quad 0 \leq t \leq T \quad (2.35)$$

$$\phi_Q(t) = \sqrt{\frac{2}{T}} \sin(2\pi f_c t), \quad 0 \leq t \leq T \quad (2.36)$$

as a suitable pair of orthonormal basis functions that determine the inphase I and quadrature Q components of the signal. Using these basis functions to represent the $s_i(t)$ gives

$$\begin{aligned} s_i(t) &= \sqrt{E} \cos\left(\frac{2\pi}{q}i\right) \phi_I(t) - \sqrt{E} \sin\left(\frac{2\pi}{q}i\right) \phi_Q(t), \quad \begin{cases} i = 0, 1, \dots, q-1 \\ 0 \leq t \leq T \end{cases} \\ &= s_{iI} \phi_I(t) - s_{iQ} \phi_Q(t) \end{aligned} \quad (2.37)$$

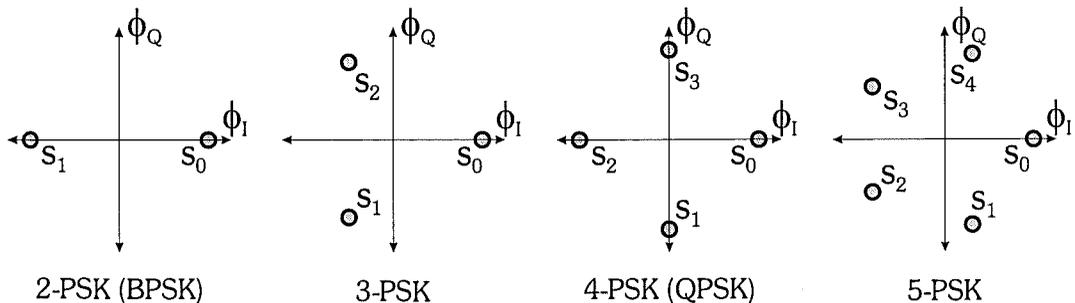


Figure 2.6: q -ary-PSK signal constellations with constant E .

Figure 2.6 shows the resultant signal constellations associated with q -ary-PSK modulation for some small values of q .

⁵Different mappings are possible, and are considered in section 5.3.

2.6.2 The effect of a noisy channel

The channel considered is the ideal *continuous additive white Gaussian noise* (AWGN) channel. This produces noise samples that are independent both of each other and the transmitted signal. Hence we can represent the received signal $x(t)$ as

$$x(t) = s_i(t) + w(t), \quad \begin{cases} i = 0, 1, \dots, M-1 \\ 0 \leq t \leq T \end{cases} \quad (2.38)$$

where $w(t)$ is a sample function from a white Gaussian process $W(t)$ with zero mean and two-sided power spectral density $N_0/2$.

2.6.3 The receiver

The received signal

The receiver correlates the received signal $x(t)$ with each of the basis functions $\phi_j(t)$. The output of correlator j is a sample of the random variable X_j ,

$$\begin{aligned} x_j &= \int_0^T x(t)\phi_j(t) dt \\ &= s_{ij} + n_j, \quad j = I, Q \end{aligned} \quad (2.39)$$

where s_{ij} is the deterministic component from the transmitted signal and

$$n_j = \int_0^T w(t)\phi_j(t) dt \quad (2.40)$$

is the sample of the random variable N_j present due to noise in the output of the correlator.

The details of simulating the effect of this additive noise are presented in appendix A.

Probabilities for the decoder

For each received symbol the decoder requires the set of probabilities p_i for each i that the transmitted signal was $s_i(t)$ given that the received signal is $x(t)$

$$p_i = \mathbb{P}(s(t) = s_i(t)|x(t)) , \quad \forall i \quad (2.41)$$

It turns out to be easier to calculate the set of log-likelihood ratios L_{i0} from which the probabilities p_i can subsequently be determined.^{6,7} Now,

$$L_{i0} = \ln \left(\frac{\mathbb{P}(s(t) = s_i(t)|x(t))}{\mathbb{P}(s(t) = s_0(t)|x(t))} \right) , \quad i = 1, 2, \dots, q-1 \quad (2.42)$$

⁶For notational simplicity we use L_{i0} to represent $L_{S(t)|X(t),s_i(t),s_0(t)}$ in this context.

⁷ $s_0(t)$ is chosen as the reference symbol for these log-likelihoods. This is an arbitrary choice, and choosing any other symbol would provide equivalent results

Using Bayes' rule,

$$\frac{\mathbb{P}(A|B)}{\mathbb{P}(C|B)} = \frac{\mathbb{P}(B|A)}{\mathbb{P}(B|C)} \times \frac{\mathbb{P}(A)}{\mathbb{P}(C)} \quad (2.43)$$

we may write

$$L_{i0} = \ln \left(\frac{\mathbb{P}(x(t)|s(t) = s_i(t))}{\mathbb{P}(x(t)|s(t) = s_0(t))} \right) + \ln \left(\frac{\mathbb{P}(s(t) = s_i)}{\mathbb{P}(s(t) = s_0)} \right) \quad (2.44)$$

Because all symbols are equally likely to be transmitted, the second term (the *a priori* probability) is 0, and hence

$$L_{i0} = \ln \left(\frac{\mathbb{P}(x(t)|s(t) = s_i(t))}{\mathbb{P}(x(t)|s(t) = s_0(t))} \right), \quad i = 1, 2, \dots, q-1 \quad (2.45)$$

The problem now becomes the evaluation of $\mathbb{P}(x(t)|s(t) = s_i(t))$ given the information x_j available at the correlator outputs. First notice that

$$\begin{aligned} \mathbb{P}(x(t)|s(t) = s_i(t)) &= \mathbb{P}(x_I, x_Q | s_I = s_{iI}, s_Q = s_{iQ}), \quad i = 0, \dots, q-1 \\ &= \mathbb{P}(x_I | s_I = s_{iI}) \times \mathbb{P}(x_Q | s_Q = s_{iQ}) \end{aligned} \quad (2.46)$$

using the inphase and quadrature notation of (2.37). The second line is made possible by the fact that in the case of an AWGN channel, the correlator outputs are independent. The probabilities in (2.46) may now be replaced by the probability density functions associated with x_I and x_Q to give

$$\begin{aligned} \mathbb{P}(x(t)|s(t) = s_i(t)) &= \frac{1}{\sigma\sqrt{2\pi}} \exp \left(-\frac{(x_I - s_{iI})^2}{2\sigma^2} \right) \\ &\quad \times \frac{1}{\sigma\sqrt{2\pi}} \exp \left(-\frac{(x_Q - s_{iQ})^2}{2\sigma^2} \right) \\ &= \frac{1}{2\pi\sigma^2} \exp \left(-\frac{(x_I - s_{iI})^2 + (x_Q - s_{iQ})^2}{2\sigma^2} \right) \end{aligned} \quad (2.47)$$

Finally we determine that

$$\begin{aligned} L_{i0} &= \ln \left(\frac{\exp \left(-\frac{(x_I - s_{iI})^2 + (x_Q - s_{iQ})^2}{2\sigma^2} \right)}{\exp \left(-\frac{(x_I - s_{0I})^2 + (x_Q - s_{0Q})^2}{2\sigma^2} \right)} \right) \\ &= \frac{(x_I - s_{0I})^2 + (x_Q - s_{0Q})^2 - (x_I - s_{iI})^2 - (x_Q - s_{iQ})^2}{2\sigma^2} \end{aligned} \quad (2.48)$$

Expressing this functionally we see that

$$L_{i0} = \frac{(\text{distance between } x \text{ and } s_0)^2 - (\text{distance between } x \text{ and } s_i)^2}{2 \times \text{noise variance}} \quad (2.49)$$

The probabilities p_i that were initially sought may now be calculated from the log-likelihood ratios (see section 2.3.3). These probabilities are the information the decoder requires for each symbol received.

2.7 The Decoder

2.7.1 What is known

The decoder necessarily knows the frame length, the component RSC code used, the SNR and the structure of the interleaver. It also assumes that all symbols are equally likely to be transmitted, and that both component encoders started in the zero state. It is possible that it also knows the terminating state of these encoders.

After demodulation, the decoder is supplied with three sets of probabilities; those associated with the systematic symbols \bar{L}_{sys} , and those associated with each of the two sets of parity symbols \bar{L}_{par1} and \bar{L}_{par2} . These probability vectors are made up of sets $\{L\}$ of log likelihood ratios L_{i0} as calculated in section 2.6.3.

2.7.2 The iterative decoder

Despite the apparent simplicity of the turbo encoder, the inclusion of a random interleaver means that the trellis structure of a turbo code is intractably complex. Because of this, the use of traditional one step decoding methods such as the Viterbi algorithm is impossible.

The sub-optimal iterative decoding algorithm is what makes it feasible to decode such complex codes. The basic concept is that the simpler constituent codes are decoded repeatedly using soft-input-soft-output (SISO) constituent decoders. The new information gleaned from one constituent decoder is then used to bias the *a priori* probabilities used by the next constituent decoder. The iterative decoding process is depicted in figure 2.7.

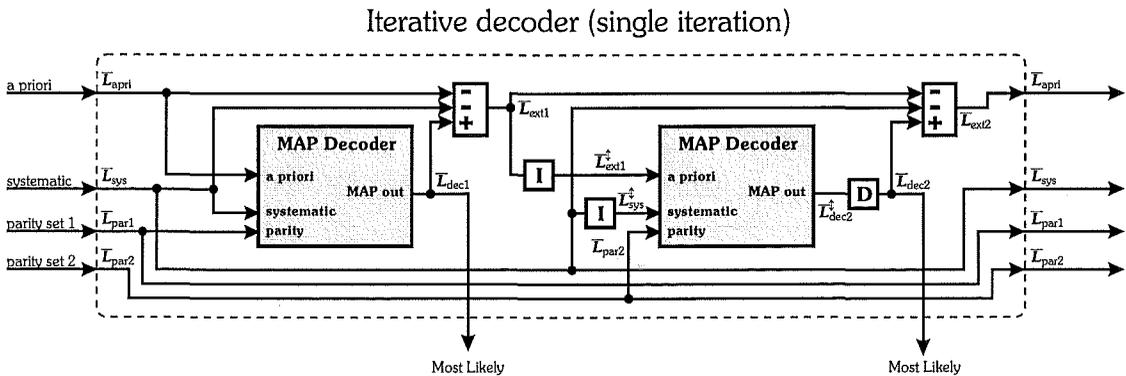


Figure 2.7: Block diagram showing one iteration of the turbo decoder.

The process works as follows:

1. The decoder initially assumes equal *a priori* probabilities \bar{L}_{apri} for each symbol.
2. The SISO component decoder takes the systematic probabilities \bar{L}_{sys} , the first set of parity probabilities \bar{L}_{par1} , the *a priori* estimates of the information symbols \bar{L}_{apri}

and decodes the first component code. This gives an *a posteriori* estimate $\bar{\mathbf{L}}_{\text{dec1}}$ for each information symbol.

3. It can be shown [32] that this *a posteriori* estimate can be viewed as the sum of three terms

$$\bar{\mathbf{L}}_{\text{dec1}} = \bar{\mathbf{L}}_{\text{apri}} + \bar{\mathbf{L}}_{\text{sys}} + \bar{\mathbf{L}}_{\text{ext1}} \quad (2.50)$$

where $\bar{\mathbf{L}}_{\text{ext}}$ is the *extrinsic* or extra information that was gleaned from this particular step in the decoding process. This extrinsic information provides an independent estimate of the information symbols, and is used as the *a priori* information for the second component decoder.

4. The systematic, and extrinsic probabilities are then interleaved (denoted by \updownarrow), and $\bar{\mathbf{L}}_{\text{sys}}^{\updownarrow}$, $\bar{\mathbf{L}}_{\text{par2}}$ and $\bar{\mathbf{L}}_{\text{ext1}}^{\updownarrow}$ are given to the second component decoder which provides a new *a posteriori* estimate of the transmitted symbols so that

$$\bar{\mathbf{L}}_{\text{dec2}} = \bar{\mathbf{L}}_{\text{ext1}}^{\updownarrow} + \bar{\mathbf{L}}_{\text{sys}}^{\updownarrow} + \bar{\mathbf{L}}_{\text{ext2}} \quad (2.51)$$

This new extrinsic information is then deinterleaved and used as the *a priori* estimate when decoding the first component code again.

5. This process repeats (usually for a fixed number of iterations), hopefully providing an improved *a posteriori* estimate of the information symbols with each iteration. When enough iterations have been completed, a hard decision is made about each information symbol from the latest *a posteriori* estimate.⁸

For the iterative process to work, the decomposition of the *a posteriori* log-likelihood ratios that each constituent decoder calculates into the sum of their *a priori*, *systematic* and *extrinsic* parts must be possible

$$\bar{\mathbf{L}}_{\text{dec}} = \bar{\mathbf{L}}_{\text{apri}} + \bar{\mathbf{L}}_{\text{sys}} + \bar{\mathbf{L}}_{\text{ext}} \quad (2.52)$$

This decomposition is shown to be valid in the binary case in [32] and is still valid in this more general situation where vectors of sets of log-likelihood ratios $\bar{\mathbf{L}}$ are considered because its formation ([32] eqn. 44) works independently for each *a posteriori* log-likelihood ratio in the sets.

2.7.3 The constituent decoders

The constituent decoder decodes the systematic data against one set of parity bits at a time. A full iteration of the decoder requires the constituent decoder to be run twice, once on each set of parity bits.

⁸See section 3.4 for a discussion of how many iterations might be required, and section 2.3.3 for a description of how hard decisions are made from soft probability ratios.

The BCJR MAP algorithm [3] is used, and is described in this context below. The BCJR algorithm is optimal in that it finds the most likely transmitted symbol at each time index. This means that each of the component decoders is optimal, but these only decode against one set of parity bits, and the overall iterative decoder is not optimal.

Let the N states of the constituent convolutional encoder be indexed by n , $n = 0, 1, \dots, N - 1$ noting that $N = q^\nu$ where q is the size of the ring used, and ν is the number of memory elements in the constituent encoder. Secondly let the state of the constituent encoder at time t be denoted by S_t , its input by U_t and its output by X_t , where $X_t = (X_{tS}, X_{tP})$ with X_{tS} and X_{tP} representing the systematic and parity components of the output. As the code is an q -ary code we must have $U_t, X_{tS}, X_{tP} \in \{0, 1, \dots, q - 1\}$. Also let a sequence of states from time t to t' be denoted by $\mathbf{S}_t^{t'}$ with corresponding output $\mathbf{X}_t^{t'}$.

The state transitions of the constituent encoder are governed by the transition probabilities

$$k_t(n|n') = \mathbb{P}(S_t = n | S_{t-1} = n') \quad (2.53)$$

These are determined both by the trellis structure of the constituent code (which may disallow some transitions), and by the *a priori* probabilities of the input (which may favour some transitions). If the constituent encoder is forced to end in a known state by the addition of a tail sequence to the input data (see section 2.5.5), then this will limit the allowed transitions near the end of the trellis. If we define $u(n', n)$ to be the input required to move the constituent encoder from state n' to state n (assuming this transition is allowed), then for times t not in the tail section of the trellis we can write

$$k_t(n|n') = \begin{cases} \mathbb{P}(U_t = u(n', n)) & \text{if transition is possible} \\ 0 & \text{otherwise} \end{cases} \quad (2.54)$$

If the trellis is terminated, then in the tail section only one transition is possible from each state, and we can write

$$k_t(n|n') = \begin{cases} 1 & \text{if transition is forced} \\ 0 & \text{otherwise} \end{cases} \quad (2.55)$$

Note that $\mathbb{P}(U_t = u(n', n))$ is simply one of the *a priori* probabilities associated with the input to the constituent code at time t . This information is supplied to the constituent decoder by the main iterative decoding process.

The output of the constituent encoder is determined by the probabilities

$$q_t(X|n', n) = \mathbb{P}(X_t = X | S_{t-1} = n', S_t = n) \quad (2.56)$$

where $X = (X_S, X_P)$ and $X_S, X_P \in \{0, 1, \dots, q - 1\}$. Because the constituent encoders are time invariant, $q_t(X|n', n)$ is time independent and we may write

$$q_t(X|n', n) = \begin{cases} 1 & \text{if } X = \text{output with transition from state } n' \text{ to } n \\ 0 & \text{otherwise} \end{cases} \quad (2.57)$$

Assuming that the constituent encoder starts in state $S_0 = 0$, is presented with an input sequence \mathbf{U}_1^τ , and ends in a possibly known terminal state S_τ , it will produce an output sequence \mathbf{X}_1^τ . At the receiver the constituent decoder will be presented with a sequence \mathbf{Y}_1^τ , a version of \mathbf{X}_1^τ corrupted by AWGN noise in the channel. The objective of the decoder is to examine \mathbf{Y}_1^τ in the light of the *a priori* probabilities $\mathbb{P}(U_t = U)$ and determine the *a posteriori* probabilities

$$\mathbb{P}(U_t = U | \mathbf{Y}_1^\tau), \quad \begin{cases} t = 1, 2, \dots, \tau \\ U = 0, 1, \dots, q-1 \end{cases} \quad (2.58)$$

It turns out that these *a posteriori* probabilities can be calculated from the joint probability

$$\begin{aligned} \sigma_t(n', n) &= \mathbb{P}(S_{t-1} = n', S_t = n, \mathbf{Y}_1^\tau) \\ &= \mathbb{P}(S_{t-1} = n', S_t = n | \mathbf{Y}_1^\tau) \cdot \mathbb{P}(\mathbf{Y}_1^\tau) \end{aligned} \quad (2.59)$$

with

$$\mathbb{P}(U_t = U | \mathbf{Y}_1^\tau) = \frac{1}{\mathbb{P}(\mathbf{Y}_1^\tau)} \times \sum_{\substack{n, n' \\ u(n', n) = U}} \sigma_t(n', n) \quad (2.60)$$

Now for a particular received sequence \mathbf{Y}_1^τ , $\mathbb{P}(\mathbf{Y}_1^\tau)$ will be a constant, and so the first term of (2.60) can be ignored as long as we normalise to ensure that

$$\sum_{U=0}^{q-1} \mathbb{P}(U_t = U | \mathbf{Y}_1^\tau) = 1 \quad (2.61)$$

The problem is then reduced to finding the $\sigma_t(n', n)$. Define

$$\alpha_t(n) = \mathbb{P}(S_t = n, \mathbf{Y}_1^t) \quad (2.62)$$

$$\beta_t(n) = \mathbb{P}(\mathbf{Y}_{t+1}^\tau | S_t = n) \quad (2.63)$$

$$\gamma_t(n', n) = \mathbb{P}(S_t = n, Y_t | S_{t-1} = n') \quad (2.64)$$

It can be shown [3] that

$$\sigma_t(n', n) = \alpha_{t-1}(n') \cdot \gamma_t(n', n) \cdot \beta_t(n) \quad (2.65)$$

and also that $\alpha_t(n)$ can be written as the recursion

$$\alpha_t(n) = \sum_{n'=0}^{N-1} \alpha_{t-1}(n') \cdot \gamma_t(n', n), \quad t = 1, 2, \dots, \tau \quad (2.66)$$

Because it is known that the constituent encoder began in state 0 suitable boundary conditions are

$$\alpha_0(0) = 1, \quad \text{and} \quad \alpha_0(n) = 0, \quad \text{for } n \neq 0 \quad (2.67)$$

Similarly, it can be shown that $\beta_t(n)$ is given by the recursion

$$\beta_t(n) = \sum_{n'=0}^{N-1} \beta_{t+1}(n') \cdot \gamma_{t+1}(n, n'), \quad t = 0, 1, \dots, \tau - 1 \quad (2.68)$$

In this case if the terminal state of the constituent encoder is unknown, suitable boundary conditions are

$$\beta_\tau(n) = \frac{1}{N}, \quad n = 0, 1, \dots, N - 1 \quad (2.69)$$

providing equal probabilities for each state. Conversely if a tail sequence was used and it is known that the constituent encoder ended in state 0 then

$$\beta_\tau(0) = 1, \quad \text{and} \quad \beta_\tau(n) = 0, \quad \text{for } n \neq 0 \quad (2.70)$$

Therefore, σ can be determined from α , β and γ , and α and β can be determined recursively, as shown above, given suitable boundary conditions. Note the the α recursion works from the start of the frame ($t = 0$) while the β recursion works from the end of the frame ($t = \tau$). Because of these backward and forward recursions the entire frame is needed before decoding can begin.

Lastly $\gamma_t(n', n)$ must be determined. It can be shown that

$$\gamma_t(n', n) = \sum_X k_t(n|n') \cdot q_t(X|m', m) \cdot \mathbb{P}(Y_t|X) \quad (2.71)$$

where the summation is over all possible output symbols $X = (X_S, X_P)$. So

$$\mathbb{P}(Y_t|X) = \mathbb{P}(Y_{tS}, Y_{tP}|X_S, X_P) \quad (2.72)$$

Now because the probability of receiving Y_{tS} is only dependent on the transmitted information symbol X_S , and similarly Y_{tP} is dependent only on the transmitted parity symbol X_P , this can be simplified to

$$\mathbb{P}(Y_t|X) = \mathbb{P}(Y_{tS}|X_S) \cdot \mathbb{P}(Y_{tP}|X_P) \quad (2.73)$$

These probabilities $\mathbb{P}(Y_{t\ell}|X_\ell)$, $\ell = S, P$ are in fact some of the probabilities calculated in section 2.6.3 (equation 2.47) and are provided to the constituent decoders by the overall iterative decoding process.

In summary, for each frame, and for each half-iteration, the constituent decoder is provided with *a priori* probabilities for each of the information symbols $\mathbb{P}(U_t = U)$. It is also provided with the transmission probabilities $\mathbb{P}(Y_{t\ell}|X_\ell)$ for the systematic and parity symbols of the particular component code. In return the constituent decoder calculates the maximised *a posteriori* probabilities $\mathbb{P}(U_t = U|\mathbf{Y}_1^T)$.

Chapter 3

Decoder Convergence Study

The iterative turbo decoding algorithm has somewhat elusive convergence properties. In this chapter, the modes of convergence observed in an extensive series of turbo-code simulations are categorised by examining the bit convergence behaviour for each frame. Building on these results, a new method for decoder termination based on average log-likelihood ratios is presented, and is compared with other methods. As a further application, a selective repeat ARQ system using turbo codes is considered.

3.1 Introduction

The work in this chapter was motivated by error rate plots such as that shown in figure 3.1, which appeared to indicate that performing more decoder iterations does not necessarily result in better performance. These results are due to oscillating convergence behaviour in the turbo decoder, and are exacerbated by the very poor interleaver that was used for this particular simulation and the small number of decoder iterations.¹

The convergence properties of the iterative decoding algorithm used for turbo-decoding have been of concern for some time. In 1996, Benedetto and Montorsi [6] considered it an *important theoretical question*. Subsequent research has shown that while the turbo-decoder will usually converge, neither convergence to the correct maximum likelihood (ML) solution, nor convergence to a stable solution can be guaranteed, particularly at low SNRs [45, 51]. It is however clear that good interleaver design can minimize the number of frames for which the decoder converges to an incorrect solution or does not converge at all [23, 2].

Recently, Richardson [51] proved the existence of fixed points using a geometric argument. He showed that the turbo decoding algorithm always possesses fixed points, and gave conditions under which only one fixed point will exist. He also states that even if the decoder possesses stable fixed points, this does not guarantee convergence.

¹The frame depicted in figure 3.11 is from this simulation, and it is frames like this that are responsible for the rising floor seen in figure 3.1.

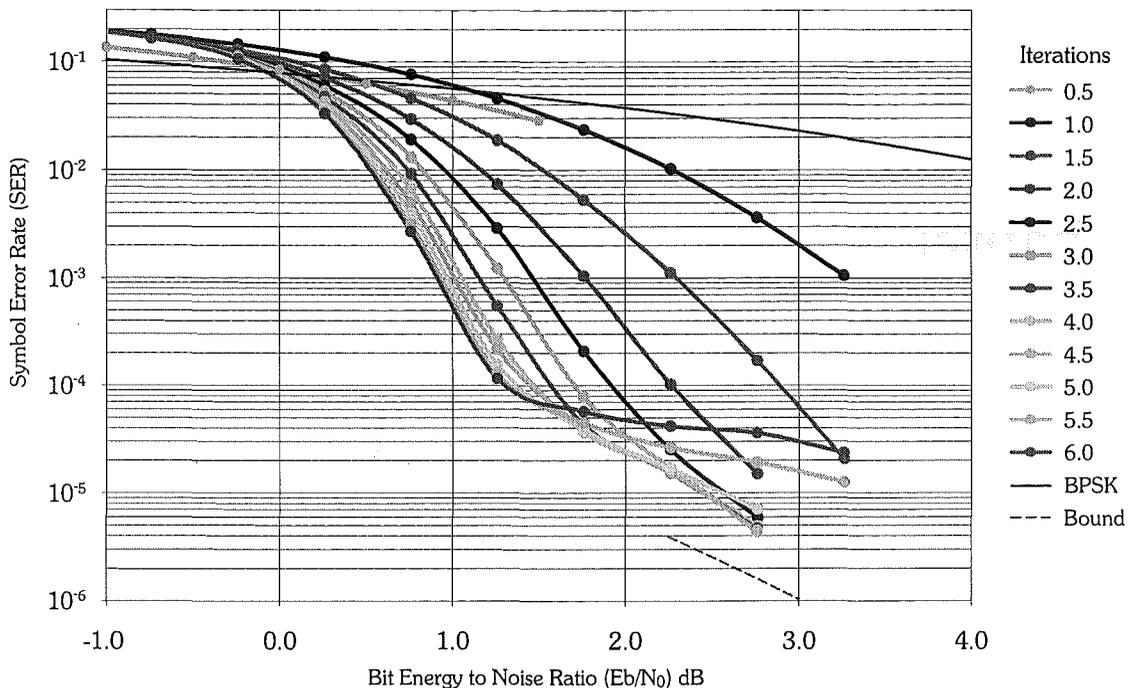


Figure 3.1: Error rate plot which appears to show that more decoder iterations can actually lead to poorer code performance (terminating interleaver). These results are not typical. The bound is a union bound.

These results are very useful, but do not illustrate what occurs when the decoder fails to converge. In this thesis specific information and examples of the dynamic behaviour of the decoder are provided.

Papers subsequent to the work of this chapter enlighten the situation further and reinforce the results presented.

In [1], Agrawal and Vardy examine the fixed points of the turbo decoding algorithm as a function of SNR. They show that at all reasonable SNRs the decoding process will possess so called ‘unequivocal’ fixed points that correspond to a correct or nearly correct decision. However at lower SNRs, and in the waterfall region of the error rate curve these will be ‘shadowed’ by ‘indecisive’ fixed points which represent very incorrect solutions. As the SNR is increased through the waterfall region, such ‘indecisive’ fixed points bifurcate via a number of different methods and eventually disappear. These results indicate that for sufficiently high SNRs, the decoding process will always converge to an ‘unequivocal’ fixed point. For low SNRs, such points will be shadowed, and convergence to an ‘indecisive’ fixed point will occur. In the waterfall region, either process may occur, or alternatively, the bifurcation process may result in stable ‘invariant curves’ to which the decoding process can converge resulting in never ending quasi-periodic oscillation. This view of the turbo decoding process as a dynamical system with fractal like properties was previously suggested by Barbulescu [11].

In [17], ten Brink also considers the convergence properties of turbo decoders, but

rather than considering individual frames, ensemble characteristics are examined. His extrinsic information transfer (EXIT) charts predict the performance of a turbo coded system at different SNRs. These results show that at low SNRs, ensemble convergence to low error solutions is impossible. In the waterfall region, ensemble convergence to low error rates is possible but will take many iterations. As the SNR further increases convergence will be quicker.

The following sections examine the convergence of the classic turbo-decoder with particular focus on those frames that do not decode to the transmitted data. The code considered is a rate 1/3 turbo-code with a rate 1/2, four-state, $\langle 101|111 \rangle$, recursive systematic convolutional (RSC) code used for both constituent encoders. The number of data bits per frame was set at ~ 1000 .² The BCJR algorithm [3] was used in the constituent decoders.

3.2 Interleavers

3.2.1 The Role of the Interleaver

The interleaver lends the turbo code its complexity and the random nature required for good performance. It transforms the concatenation of two simple codes into a very complex code whilst still preserving the ability to implement a feasible decoder. It is because of this that the majority of the gain achieved by turbo codes is attributed to the interleaver [6]. As an ensemble effect, it is possible to view the interleaver as causing the input symbols to be mapped onto somewhat random codewords.

On an individual codeword basis there are particular data frames for which this process does not work well, and it is the prevalence of these frames which determine the code's error performance in the waterfall region. As shown by Benedetto and Montorsi [6], it is specifically the encoders response to weight-two data sequences that dominates the code's performance.

This is explained as follows. Firstly recall that both component codes are recursive, and will therefore by definition have an 'infinite' weight response to a weight-one impulse input. Therefore, the lowest weight (systematic+parity) output possible from an individual component code will be due to short low weight input sequences provided the input weight is at least two. In other words, particular low weight inputs will result in short low weight outputs from the component code. The job of the interleaver is to try and ensure that in such cases the input sequence is shuffled in a fashion which results in a much higher parity weight from the second component encoder. It is when this does not happen that codewords with low overall weight occur. Certain low weight inputs will generate low weight outputs from *both* component codes and therefore a codeword with

²The frames presented all contain 1000 or 1024 data bits, but may be padded to 1002 or 1026 bits by a tail sequence. The resultant frame lengths are between 3000 and 3078 bits in length.

low weight. Such code words are the most susceptible to error, and hence dominate the performance of the turbo code.

The interleaver works by mapping a sequence of weight x onto another random sequence of weight x . The chance that an input sequence which produces a low weight output will map onto another sequence that produces a low weight output is highest for weight two inputs.

This can be demonstrated by considering a turbo code that encodes data frames of n symbols. Now let the number of weight two sequences that produce low output weights be a , and the number of weight three sequences that produce low output weights be b . Furthermore, in a frame of length n , there are $n(n-1)$ possible weight two sequences and $n(n-1)(n-2)$ possible weight three sequences. Therefore the chance of a low output weight producing weight two sequence being mapped onto a similar sequence is $\frac{a}{n(n-1)}$. While for weight three inputs, the chance is only $\frac{b}{n(n-1)(n-2)}$ a decrease by a factor of $\sim \frac{1}{n}$. For even higher weight inputs, this chance will decrease by further factors of $\sim \frac{1}{n}$.³ Hence the chance of two sequences that produce low weight outputs being mapped onto each other by the interleaver is highest for weight two inputs, and therefore, such weight two error events will dominate the turbo codes performance.

3.2.2 Interleaver Design

Choosing a good interleaver limits the number of frames for which the decoder will exhibit undesirable convergence behaviour, but will not eliminate these frames. Three different classes of random or pseudo-random interleaver were used to produce the results reported in this thesis; *terminating*, *random* and *S-random* interleavers (see section 2.5.4).

The different convergence phenomena which lead to the results of figure 3.1 were first observed when studying a frame termination strategy which ensures that both constituent encoders have a known final state [10, 15]. This involves using a constrained or *terminating* random interleaver. Unfortunately such an interleaver is significantly less likely to disturb the weight two information sequences, which as discussed in the previous section have been shown to dominate turbo-code performance [6]. The *terminating* random interleaver design requires that all input symbol positions remain constant modulo m where m is the period of the impulse response of the component encoder. However, it is unfortunately the case that all weight two error patterns that produce finite weight outputs will have the form

$$1 \underbrace{0 \dots 0}_{m-1 \text{ zeros}} \left(\underbrace{0 \dots 0}_{m \text{ zeros}} \right)^x 1 \quad x \text{ an integer } \geq 0 \quad (3.1)$$

³ a and b etc. are in fact likely to be exponentially dependant on each other as there are likely to be more weight three than weight two sequences that produce low output weights. However since only short inputs can produce low weight outputs, if we consider only long n then the factor of $\frac{1}{n}$ given will be very close to correct.

where m is again the period of the impulse response of the component encoder. Some of these weight two error events are depicted in figure 3.2 for the $\langle 101|111 \rangle$ binary component code. Because of this, a *terminating* interleaver will map all weight two inputs that produce finite weight outputs onto each other. Hence such a design constraint is likely to select worse than average random interleavers

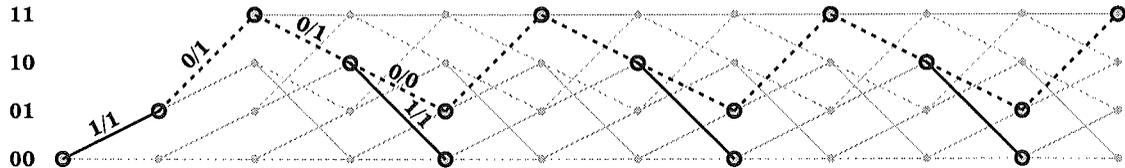


Figure 3.2: Trellis section showing the shortest three weight two error events possible with the $\langle 101|111 \rangle$ code assuming the all 0's path is correct. The solid lines depict transitions that require a 1 input while the dashed lines require a 0 input.

As stated in section 2.5.4, *S-random* interleavers are an alternate constrained design which ensures that no weight two inputs with length less than S are mapped onto each other [23]. This prevents the few lowest weight overall outputs from weight two inputs from ever occurring, and hence represents a better than average random interleaver design. (The *S-random* interleaver used in this work has $S = 21$.)

3.3 Modes of Convergence and Error Events

To display the convergence behaviour of the turbo decoder, and to allow modes of decoder convergence to be categorised *bit-convergence plots* were generated by observing the bit-convergence for various frames. A bit-convergence plot such as that of figure 3.3 presents the log-likelihood ratio [32] for each information bit in a frame versus the number of iterations performed by the decoder. If a single iteration is taken as a single pass through both constituent decoders, then these log-likelihood values are available directly from the decoder after each half-iteration. The bold lines in the plots in this chapter represent the average bit convergence.

Error events are related to these plots, in that after a fixed number of iterations, the decoder makes a hard decision. The log-likelihood ratio equals zero line acts as the decision boundary with bits above chosen as ones, and bits below as zeros.⁴

Examination of the simulation results shows that frame convergence behaviour falls into one of three categories or modes:

- Mode 1: All bits converge quickly and in an almost identical manner to a stable solution. (section 3.3.1, figure 3.3.)

⁴Note that the term “errors” is used to denote differences between the transmitted data and the iteratively decoded data, and that maximum likelihood decoding of the frame may produce a different set of errors. The iterative decoding process is not maximum likelihood.

- Mode 2: All but a small number of bits converge quickly to a stable solution. The remaining bits converge in a markedly different manner. (section 3.3.2, figures 3.4 to 3.6.)
- Mode 3: The bit convergence plot oscillates, often with the log-likelihood ratios for some bits repeatedly crossing the decision boundary. (section 3.3.3, figures 3.7 to 3.12.)

The results presented indicate that these different modes of convergence are usually representative of particular kinds of error events. There will however be exceptions to these associations.

- Mode 1 convergence is what is observed for frames that decode quickly to an error free solution. This is the desirable behaviour.
- Mode 2 convergence is what is observed for frames that converge to a stable solution containing a small number of uncorrectable errors. In particular, it is this mode of convergence that is associated with the low weight error events that dominate turbo-code performance [6].
- Mode 3 convergence is associated with frames in which the number of errors oscillate as more decoder iterations occur.

It is possible to draw some correlation between these error classes and the ‘algorithm anomalies’ observed by McEliece et. al. [45], and also with the results of Agrawal and Vardy’s [1] more recent work on the fixed points in the turbo decoding algorithm. Both of these papers view the the turbo decoder as a dynamical system in which the decoding algorithm, or ‘turbo transformations’ cause the system to converge toward fixed points that represent stable decisions. Using their terminology;

- The turbo transformations may have a unique ‘unequivocal’ fixed point that the iterations converge to, resulting in a turbo decision that agrees with or is close to the *optimal bit decision* (OBD) (The ML solution). This situation will correspond to mode 1 convergence or mode 2 convergence.
- The turbo transformations may have one or more fixed points but the iterations converge to an ‘indecisive’ point that does not correspond to the OBD. It is likely that these situations correspond to mode 2 convergence, or mode 3 convergence, which eventually stabilises.
- The turbo transformations may not converge to a fixed point, instead converging to an stable invariant curve. Clearly this corresponds to mode 3 convergence where the oscillations do not subside.

The following subsections look at sample bit convergence plots for each mode of convergence.

3.3.1 Mode 1 – Quick stable convergence

Figure 3.3 shows the typical bit convergence behaviour for this mode. All errors are corrected. This is the desired behaviour with the decoder making progressively stronger decisions about each bit, until the information it can glean from the received data is exhausted. As the SNR is increased the (average) rise time decreases and the (average) peak log-likelihood ratio increase.

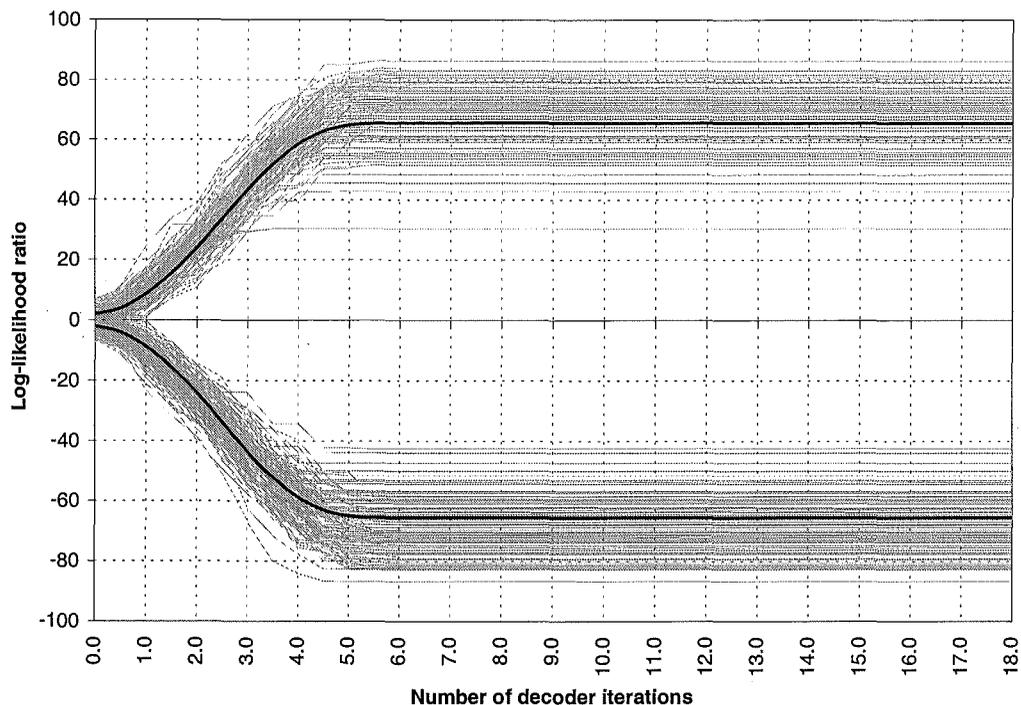


Figure 3.3: Bit convergence of a typical frame in which all errors are corrected. (SNR = 1.77dB, random (unconstrained) interleaver)

3.3.2 Mode 2 – A small number of bits converge differently

Figures 3.4 to 3.6 show the various modes of convergence in this category.

The majority of frames contained in this category are similar to that shown in figure 3.4 with exactly two anomalous bits, representing a weight two error event. Work on the distance spectra of turbo-codes [47], and on performance bounds for turbo-codes [6, 37], has shown that weight two error events dominate the performance. As discussed in section 3.2.1, this is because they can produce a finite low-weight output from the constituent RSC encoders. In general, if a low-weight input causes the first encoder to generate a low-weight output, then the role of the interleaver is to shuffle the input

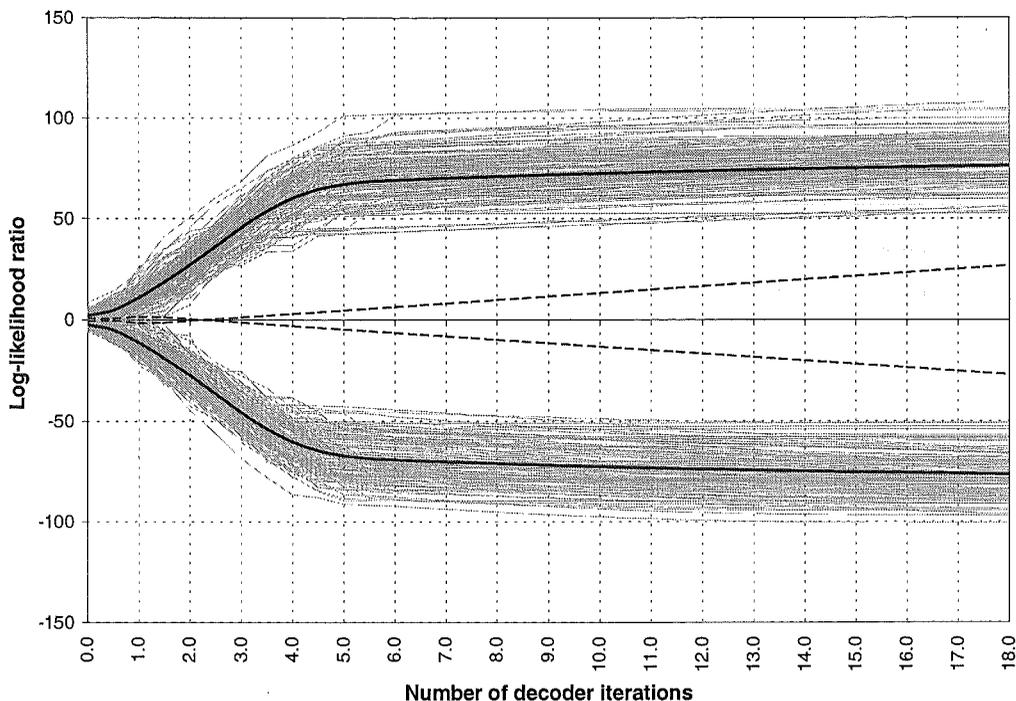


Figure 3.4: Bit convergence of a frame containing a typical weight-two error event. The bits in error are represented by the dashed lines. (SNR = 2.00dB, S-random interleaver)

sequence such that the second encoder produces a high-weight output. If this does not happen, a low-weight codeword is generated that is susceptible to errors.

It is this type of error event that is represented by figure 3.4, with the decoder converging to the wrong codeword. In this case, the bits in error are at positions 759 and 762 in the frame, the correct distance apart to produce a minimum weight codeword from the RSC encoder. The interleaved bit positions are 673 and 744 which still satisfy (3.1) and will produce a lowish weight output. These positions are beyond the range of the S parameter used when designing the interleaver.

What is perhaps most interesting about figure 3.4 is the fact that the bits in error converge in an almost linear manner, very different to the correct bits.

Figure 3.5 shows a more unusual case in which four bits converge differently representing four errors that are never corrected.

Figure 3.6 demonstrates a combination of mode 2 and mode 3 convergence that results in a decision with nine uncorrected errors. Of particular interest is the fact that the inability of the decoder to make a decision about a large number of bits forces the average log-likelihood ratio to remain low until late in the decoding process.

Work on interleaver design has shown that the incidence of error events in this class can be greatly reduced by choosing a good random interleaver such as an *S-random* interleaver [23, 2]. However, figure 3.4 shows clearly that such frames cannot be completely eliminated even with a good interleaver.

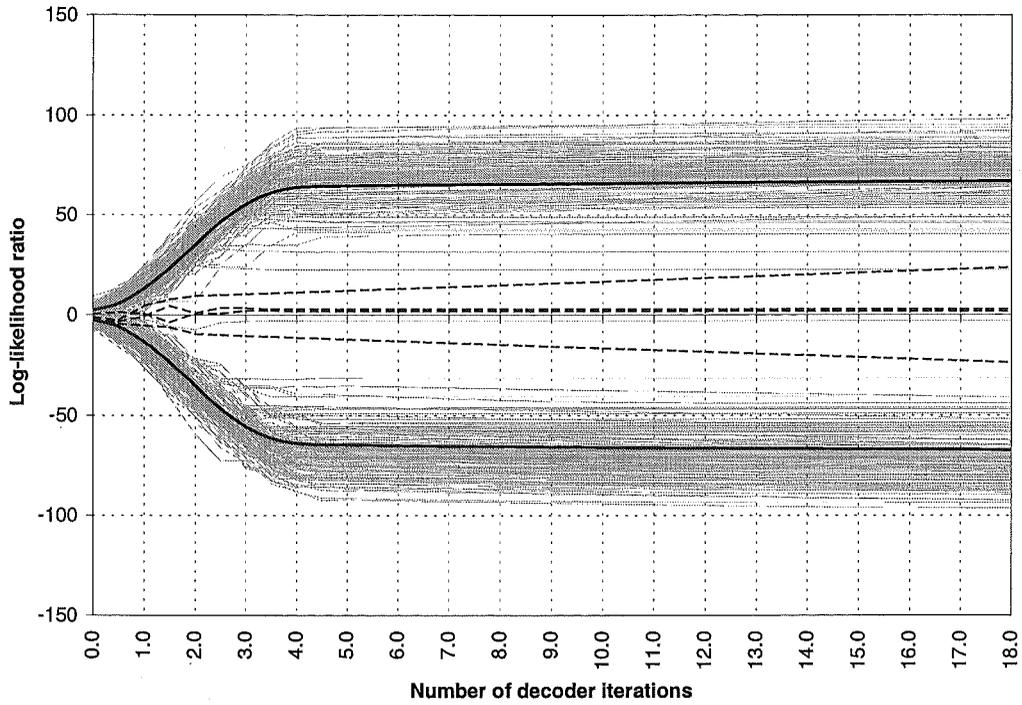


Figure 3.5: Bit convergence of a frame where four errors are never corrected. (SNR = 3.27dB, random interleaver)

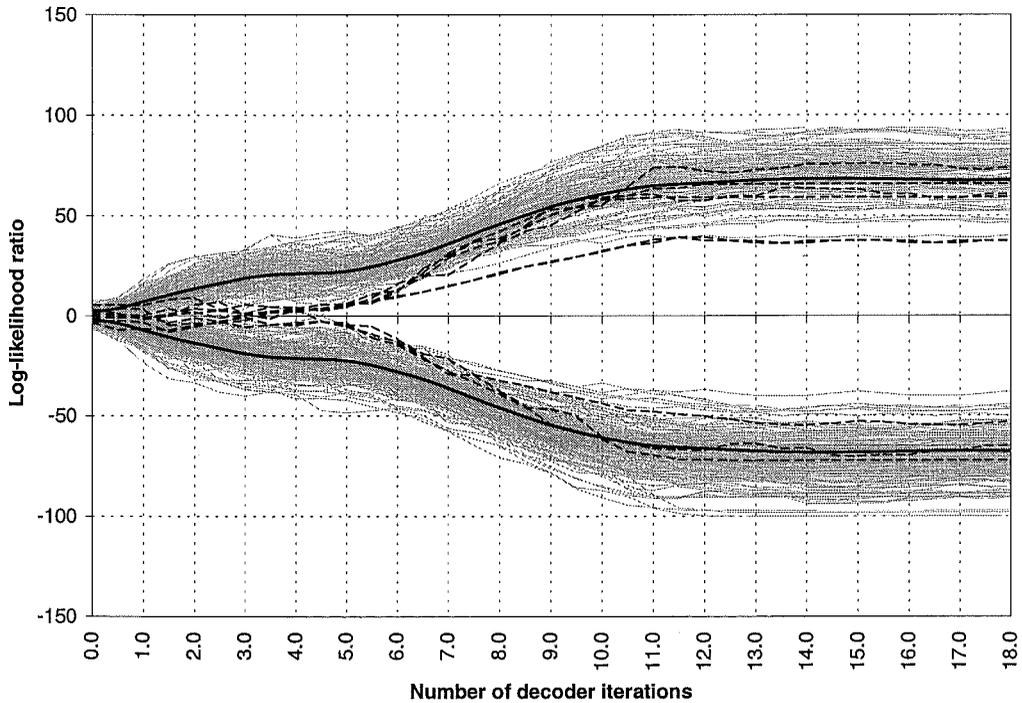


Figure 3.6: Bit convergence of a frame where nine errors (dashed lines) are never corrected. Note the long rise time. (SNR = 1.77dB, terminating interleaver)

3.3.3 Mode 3 – Oscillation

Figures 3.7 - 3.12 represent the various forms of oscillating convergence which were observed. In figure 3.7, the oscillations do not produce any errors within the first 18 iterations.

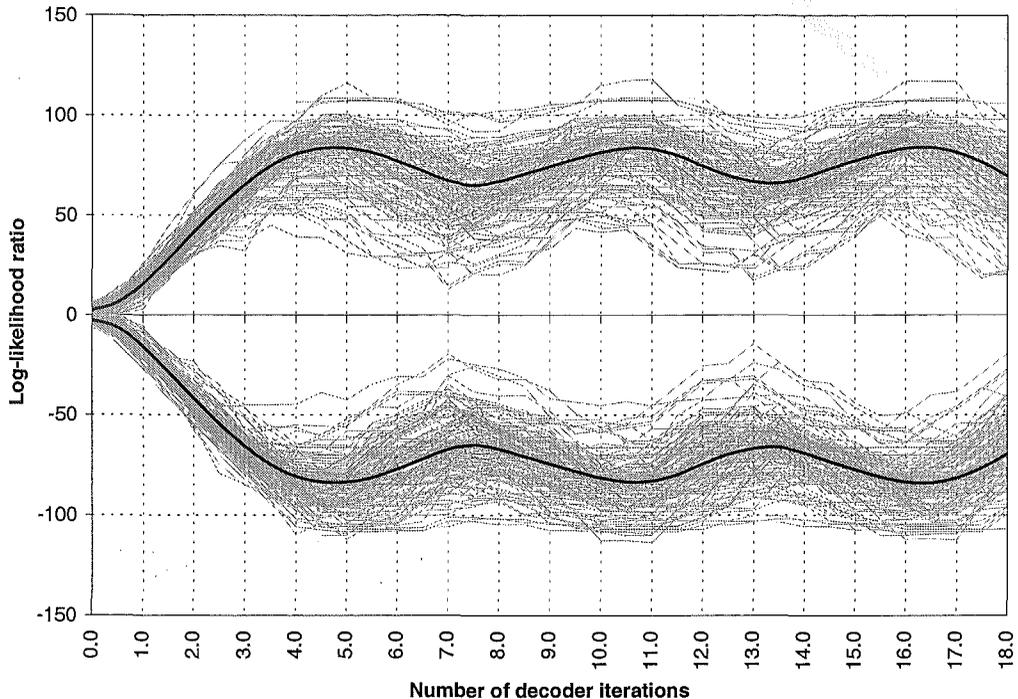


Figure 3.7: Bit convergence oscillates but does not cause any errors in the first 18 iterations. (SNR = 3.27dB, terminating interleaver)

Figure 3.8 demonstrates that oscillation may still occur even with an *S-random* interleaver. The decoder oscillates a number of times before reaching a stable decision after 14 iterations. (With numerous errors and low average log-likelihood ratio.)

Figures 3.9 and 3.10 show very different cases of extreme oscillation that do not die away. In figure 3.9 the ‘eyes’ in the convergence plot contain no errors, but at 6.5 and 13.5 iterations, bits such as the dashed one cross the decision boundary and cause a large number of errors. In contrast, the frame represented by figure 3.10 always contains errors, because in addition to bits like those in figure 3.9, it also contains bits such as the dashed one which are in error most of the time.

Figure 3.11 represents the disturbing scenario in which the decoder appears to quickly converge to a stable error free codeword. However, further iterations cause the decoder to degenerate to a low average log-likelihood ratio result that oscillates and contains errors. Further simulation to 100 iterations (figure 3.13) has shown that the decoder never recovers in this situation.

Figure 3.12 shows the case where two bits oscillate around the zero log-likelihood

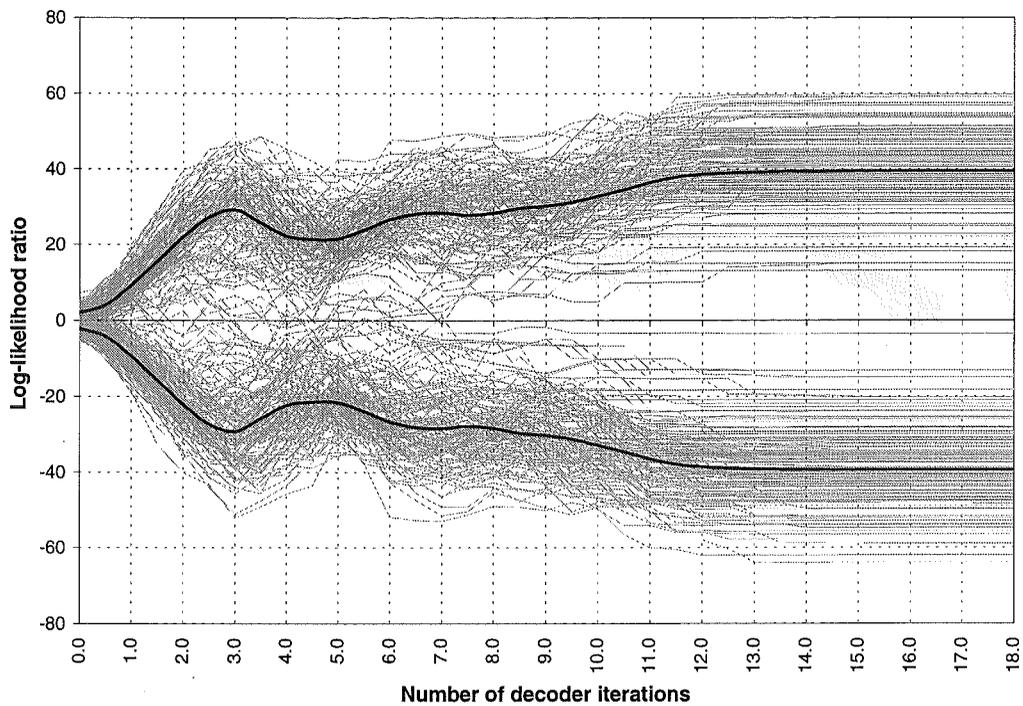


Figure 3.8: Bit convergence oscillates before reaching a stable but error ridden (14 errors) solution after about 14 iterations. (SNR = 2.00dB, S-random interleaver)

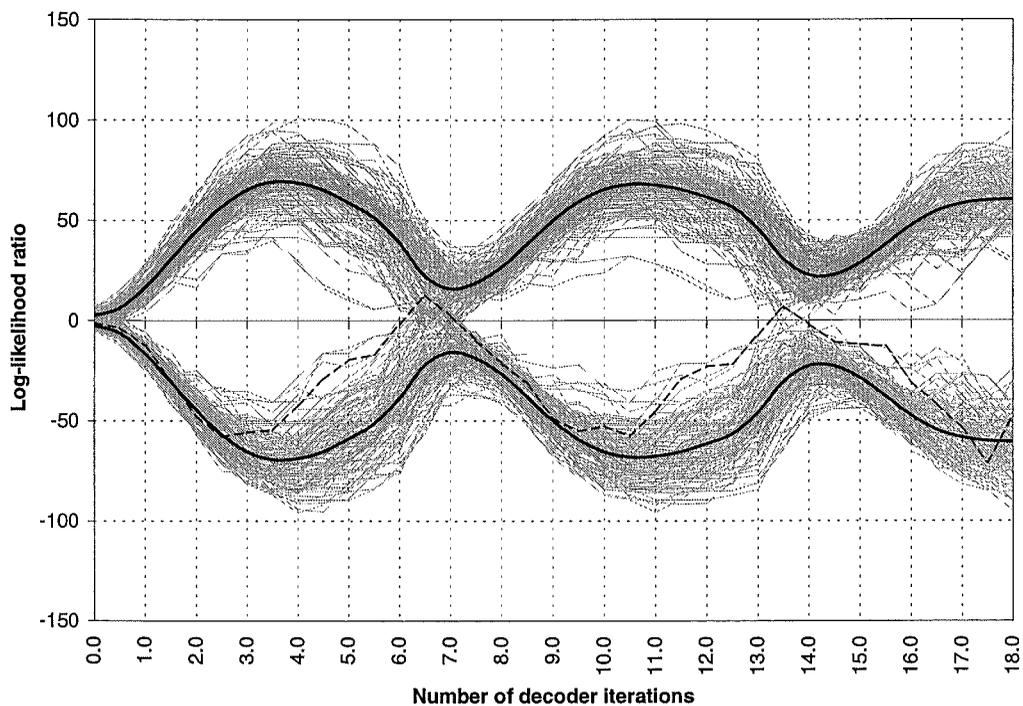


Figure 3.9: Bit convergence of a frame showing extreme oscillation where there are no errors in the 'eyes' of the diagram, but there are many (such as the dashed line) at 6.5 and 13.5 iterations. (SNR = 3.27dB, terminating interleaver)

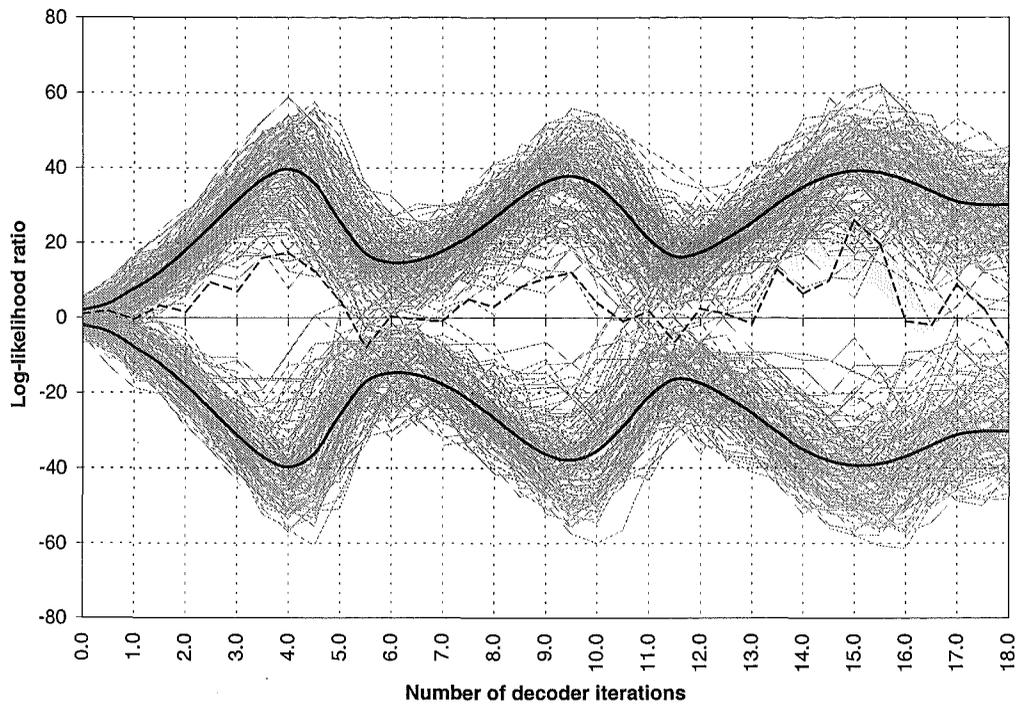


Figure 3.10: Bit convergence of a frame showing extreme oscillation. The dashed line represents a bit that is in error the majority of the time. (SNR = 1.77dB, random interleaver)

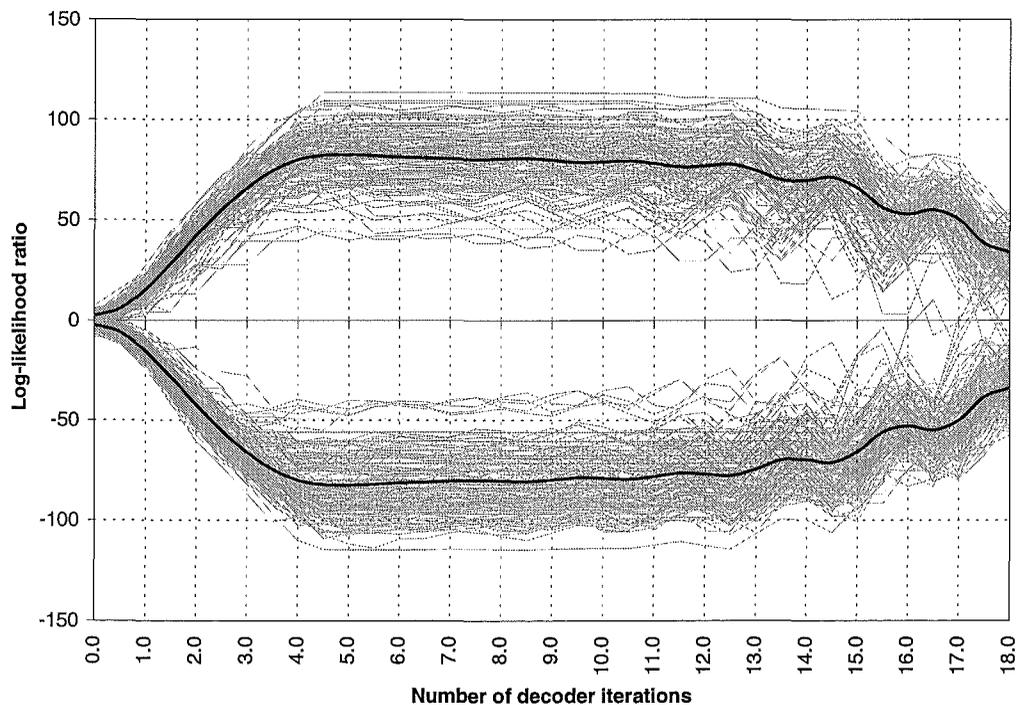


Figure 3.11: Bit convergence of a frame for which the decoder appears to converge to the correct codeword, but then degenerates to a low (average) log-likelihood ratio and many bits in error. (SNR = 3.27dB, terminating interleaver)

ratio decision boundary. These are bits 1018 and 1021 in the frame, the correct distance apart to generate a low output weight from the RSC encoder (see figure 3.2). Therefore this error event is closely related to those examined in Section 3.3.2. All other bits in this frame are decoded correctly.

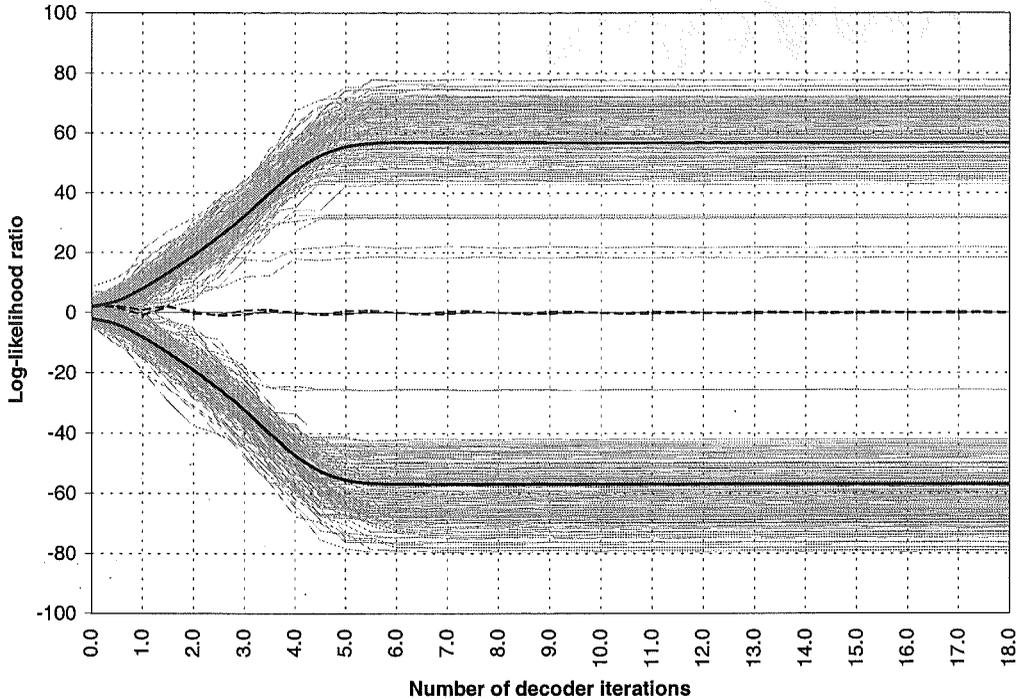


Figure 3.12: Bit convergence of a frame where two bits oscillate about the decision boundary. (SNR = 1.77dB, random interleaver)

The frames appraised in this section show that increasing the number of iterations does not always produce better results. In fact, for an individual frame more iterations can significantly increase the number of errors at the decoder output.

3.3.4 Long-Term Performance

To investigate the long term oscillatory behaviour of the turbo-decoder, the frames represented by figures 3.7, 3.9 and 3.11 were decoded for 100 iterations, and their bit convergence plots examined. In none of these cases did the oscillations show any sign of dying out. This certainly indicates that the oscillatory behaviour predicted in [45, 1] is not just the result of obscure initial conditions, and that convergence cannot be guaranteed in a practical system. Figure 3.13 shows the top half of these convergence plots extended to 100 iterations. Note the quasi-periodic nature of the log-likelihoods for the bits in this frame.

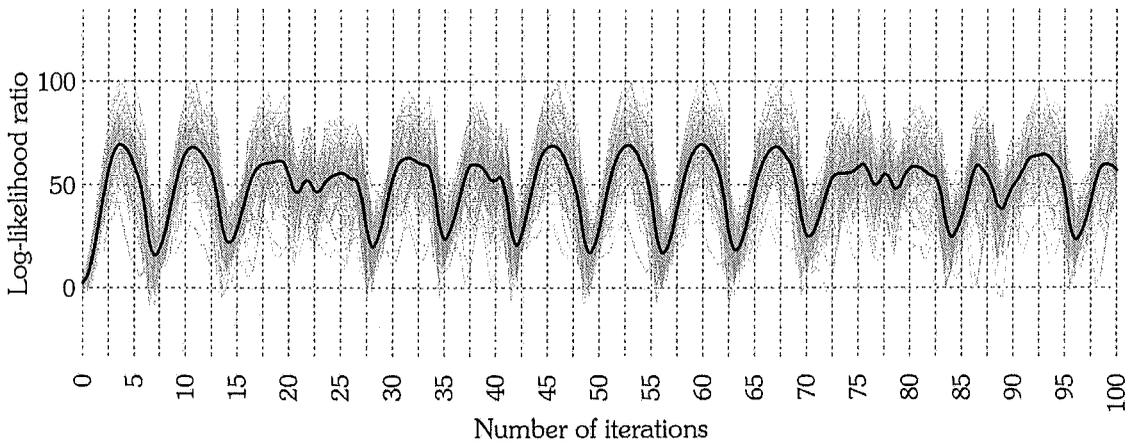
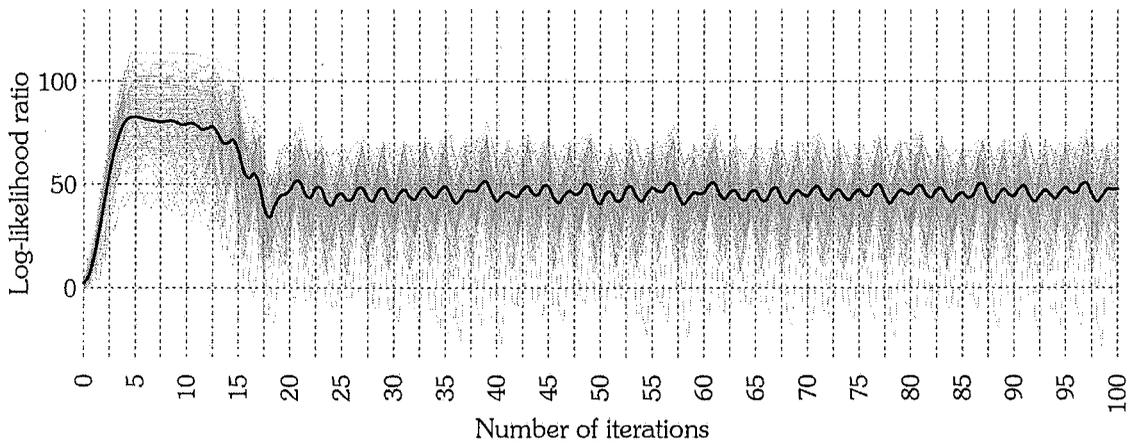
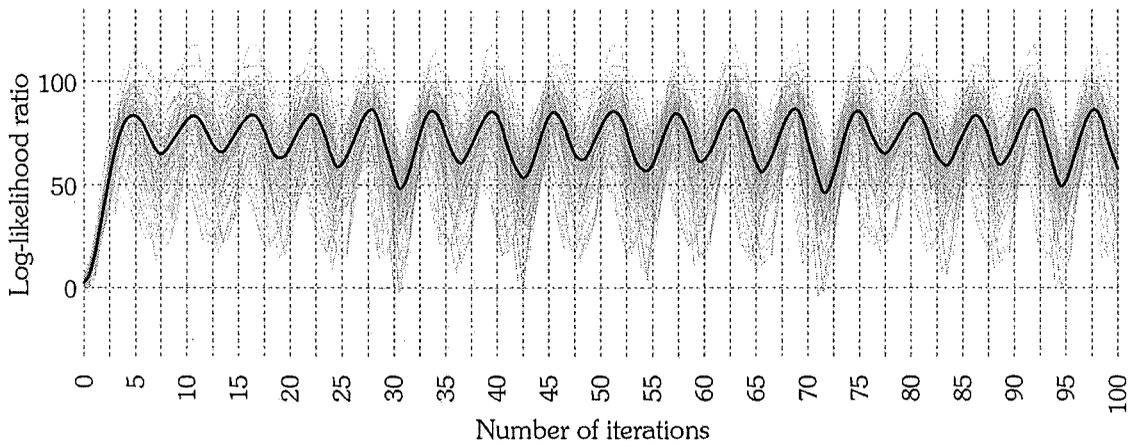


Figure 3.13: One sided bit convergence plots showing the frames of figures 3.7 (top), 3.11 (middle) and 3.9 (bottom) extended to 100 iterations

3.4 Decoder Termination

In a traditional turbo-coded system the decoder processes each frame for a fixed number of iterations. At medium/high SNRs the majority of frames quickly converge to the transmitted data sequence and extra iterations are superfluous. Very few frames require the full number of iterations, so significant savings in processing time (software decoders) or power consumption (hardware decoders) can be realised if decoding for these frames is stopped once sufficient convergence has occurred.

This has been observed by other authors. A decoder stopping criterion that examines the cross-entropy (CE) between decoder outputs from successive iterations is presented in [32] and effectively stops the decoder with minimal performance degradation. This idea is extended in [54] where two much simpler criteria are developed from the CE concept. These *sign-change-ratio* and *hard-decision-aided* criteria perform similarly to the CE criterion. An alternative approach is presented in [55] where the data is first encoded with a simple error detecting code before passing through the turbo-encoder. This means that the turbo-decoder can be stopped when the outer code detects that no errors remain.

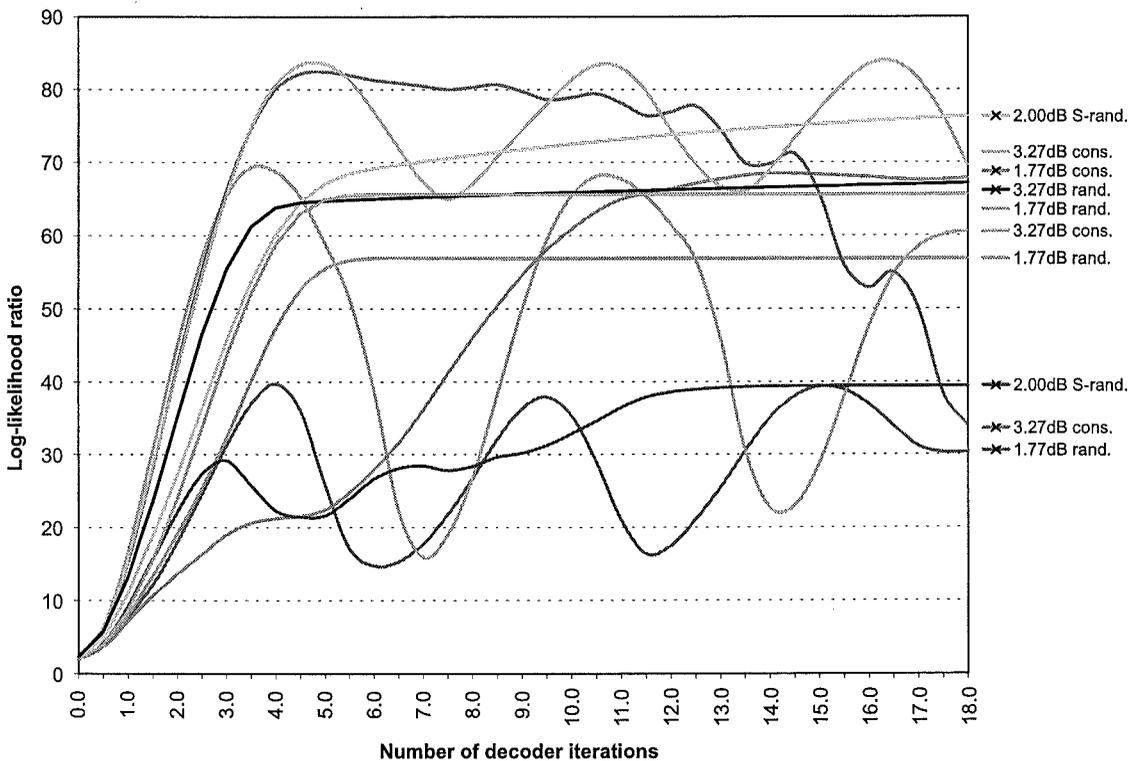


Figure 3.14: Comparison of the average convergence observed for the different example frames presented in this paper. The keys with crosses indicate frames that contained errors after 18 iterations.

Figure 3.14 summarises the various modes of convergence examined in the previous

section. These curves suggest an alternative method of decoder termination based on the average bit convergence. The decoder could be stopped when the average ceases to increase, reaches a set threshold, or does not increase faster than a specified rate.

It should be noted that the average curves calculable by the decoder will differ slightly from the ones presented in figure 3.14 because the transmitted data is unknown and therefore incorrect bits cannot be included as negative terms in the average.

Simulation of these stopping criteria showed that setting a threshold on the average log-likelihood values was the most effective method. Figures 3.15 to 3.17 demonstrate the performance of this method using the code specified in Section 3.1 simulated for 10,000 frames using an S-random interleaver. Each figure presents results for four different stopping criteria:

- Shao et. al.'s *hard-decision-aided* (HDA) criterion [54].
- The proposed average log-likelihood criteria, with the threshold chosen to limit the degradation in *bit-error-rate* (BER) to about 1%.
- The proposed average log-likelihood criteria, with the threshold chosen to limit the degradation in BER to about 10%.
- A hybrid scheme that combines the HDA and 1% threshold schemes.

The use of a hybrid scheme can be justified by observing that the HDA criteria examines the stability of the decoder outputs. This is exactly what the cross entropy measures. It looks at the outputs of the constituent decoders and quantifies the changes. Conversely the threshold stopping criteria only looks at the average strength of the decoder outputs; the magnitude of the log-likelihood average. The hybrid scheme will therefore examine both the strength and stability of the decoder's decisions. A similar stopping criteria was presented in [1] subsequent to the research of this thesis.

The average log-likelihood thresholds used for the 1% and 10% threshold schemes were determined via simulation to give the desired BER degradation. The particular thresholds used for the results presented in figures 3.15 to 3.18 are given for each simulated SNR in table 3.1.

Figure 3.15 shows the raw BER for simulations using each of the stopping criteria, and also a no-termination reference curve for which each frame was decoded for a full 10 iterations. This figure shows that none of the criteria cause significant performance degradation. Figure 3.16 emphasises these small performance differences by showing the increase in BER relative to the 10 iteration reference curve. Lastly, figure 3.17 plots the average number of decoder iterations required by each of the stopping criteria.

These curves clearly show the trade-off between BER performance and the number of iterations required. The HDA criterion gives the worst degradation in performance,

SNR(dB)	1% threshold	10% threshold
0.00	11	7
0.25	16	11
0.50	20	16
0.75	26	21
1.00	33	28
1.25	38	33
1.50	40	36
1.75	45	41
2.00	50	42

Table 3.1: Log-likelihood thresholds for the 1% and 10% threshold stopping criteria.

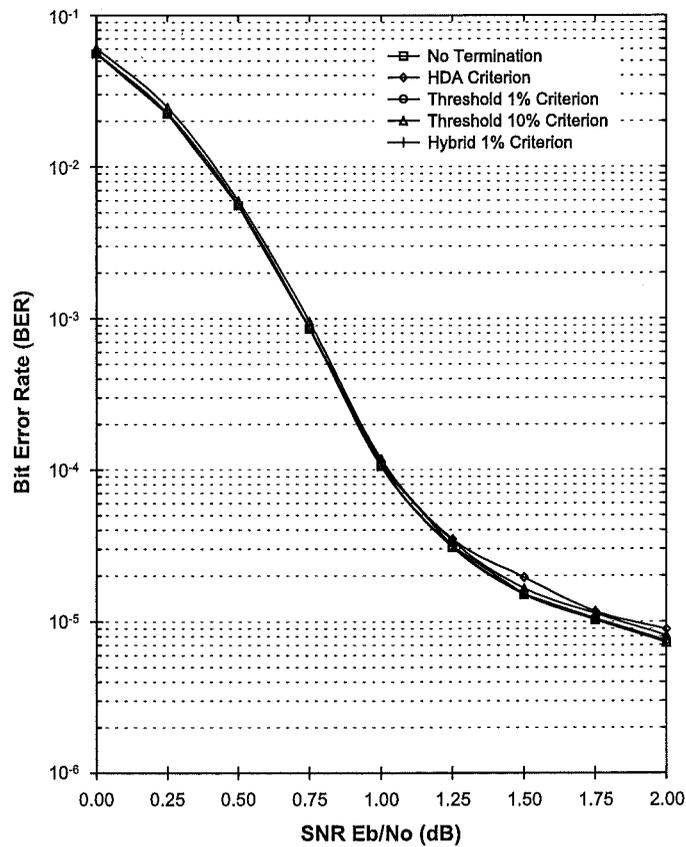


Figure 3.15: The BER performance of a turbocoded system using the HDA, Threshold and Hybrid stopping criteria.

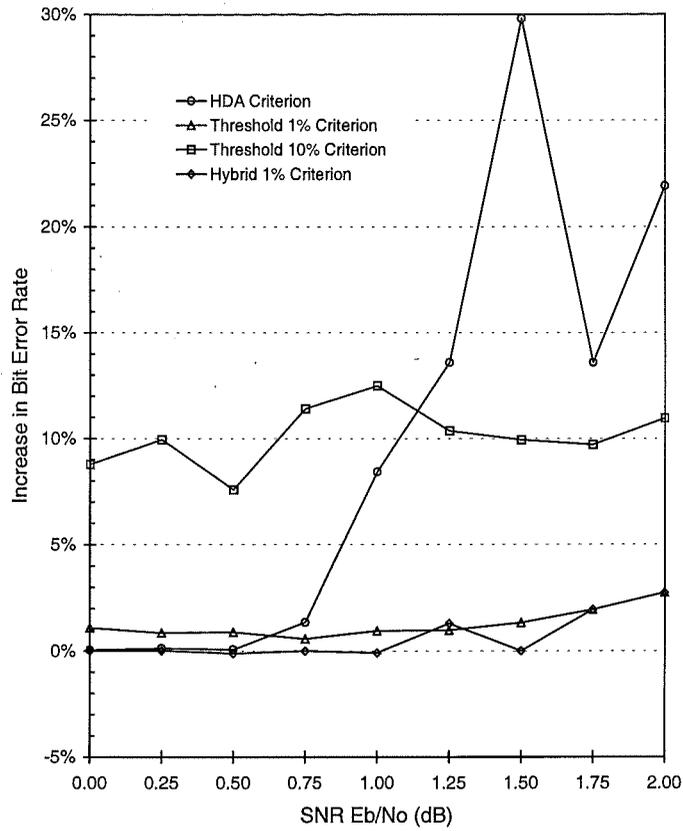


Figure 3.16: The increase in BER rate caused by the various stopping criteria relative to the 10-iterations reference value for every frame performance

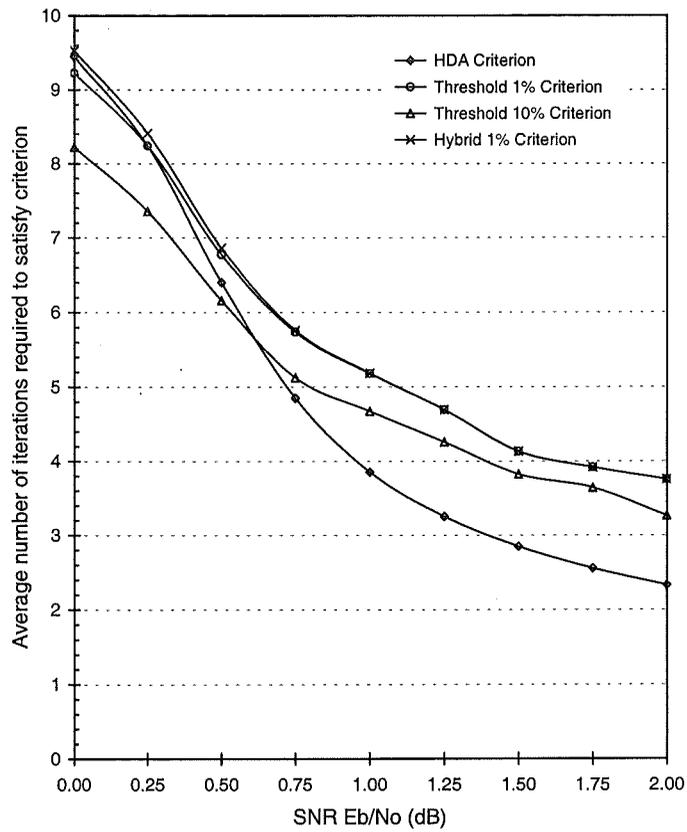


Figure 3.17: The average number of iterations required per frame to satisfy the HDA, Threshold and Hybrid criteria.

but saves the greatest number of iterations. Conversely the hybrid scheme gives very little degradation in performance, but requires the greatest number of iterations.

One difficulty with the threshold criteria is choosing an appropriate threshold. For the results presented in this section appropriate thresholds were determined by trial and error to give the desired results.

3.5 Selective Repeat ARQ

Automatic repeat request (ARQ) schemes using a turbo-code with an outer error detecting code have been proposed previously [46]. However if a significant number of the frames that contain errors can be isolated by the decoder stopping criterion, an ARQ scheme may be applied directly. This idea is applicable to any stopping criteria, by regarding any frame that takes more than x iterations to decode as a decoding failure, and requesting a retransmission of that frame.

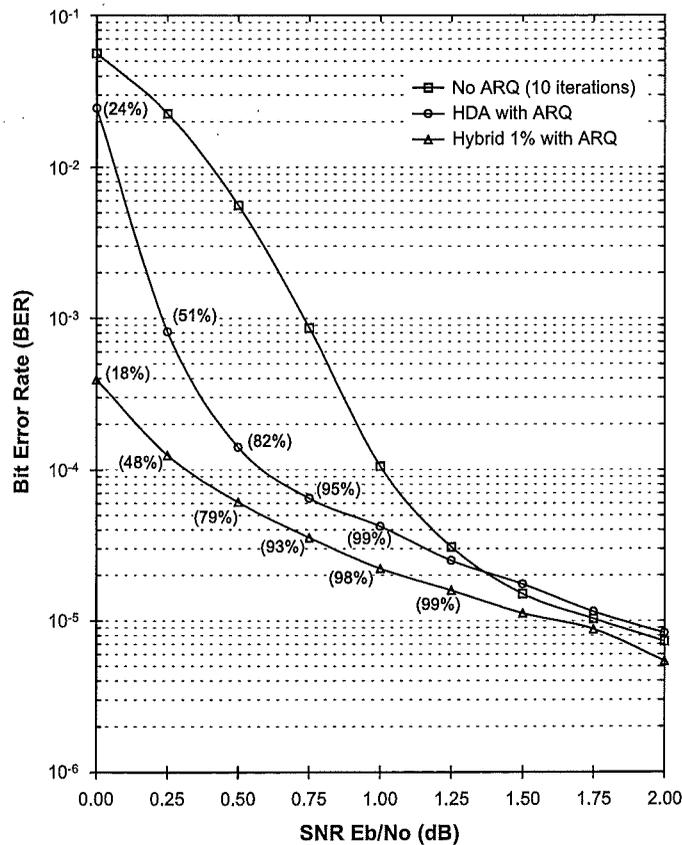


Figure 3.18: The BER performance of a selective-repeat-ARQ turbo-coded system based on the HDA and Hybrid termination criteria. Throughput percentages are next to some of the data points.

Figure 3.18 presents performance curves for selective repeat ARQ systems based on both the HDA and hybrid stopping criteria. Throughput percentages are next to some

points and show the percentage of frames which are not retransmitted.⁵ Once again the code used is that specified in Section 3.1 simulated for 10,000 frames with an S-random interleaver. Any frame that does not satisfy the stopping criteria within 10 iterations is retransmitted.

These results show that considerable performance gains can be made through the use of an ARQ scheme at lower SNRs. For example with $E_b/N_0 = 0.5\text{dB}$, using an ARQ scheme based on the hybrid stopping criterion provides nearly two orders of magnitude improvement in BER at the expense of having to retransmit just over one in four frames.

3.6 Conclusion

Figure 3.14 summarises the various modes of decoder convergence presented in this chapter. This figure emphasises the dynamical behaviour of the decoder, with a number of the curves demonstrating very different forms of oscillation. These curves show that oscillation is not just the result of obscure decoding conditions, but will be present in any real system.

Two distinct type of error events have been observed; oscillating and non-oscillating. Non-oscillating error events appear to be largely an artefact of the interleaver's inability to disrupt all low weight error sequences. These can be viewed as the decoder converging to an 'unequivocal' fixed point that contains only a few errors. Oscillating error events represent the decoder's inability to converge to a fixed point, and it is clear that large unpredictable oscillation is possible. While the choice of interleaver and the available SNR both affect the frequency with which these events occur, both classes of errors were present for all parameter sets simulated.

A stopping criterion based on the average log-likelihood ratio was proposed, and shown to save a significant number of iterations for very little degradation in BER. The potential for a stopping criterion to facilitate a selective repeat ARQ system was also demonstrated, with a large reduction in BER for low SNRs possible at the expense of throughput.

⁵Multiple retransmission was allowed. Therefore, for example a throughput of 25% means that on average each frame had to be retransmitted 4 times.

1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100

Chapter 4

Non-binary turbo codes

The iterative decoding structure and component MAP decoders used for decoding binary concatenated codes can be extended to the non-binary domain. The chapter discusses the problem of choosing good component codes for q -ary turbo codes.

4.1 Introduction

The vast majority of turbo code research considers binary component codes concatenated via a random bit interleaver (*e.g.* [14, 5, 9, 22]). This chapter considers turbo codes constructed from *non*-binary (q -ary) component codes concatenated via a random symbol interleaver and mapped onto the appropriate q -ary-PSK constellation for transmission through the AWGN channel.

First the construction and decoding of non-binary turbo codes is briefly reviewed. This was more fully discussed in chapter 2. Then the parameters that good component codes need to possess are discussed, sets of such codes are found, and simulation results for some exemplary non-binary turbo codes presented.

4.2 Non-binary Turbo Codes

The encoder structure considered is identical to that used for binary turbo codes except that all operations are now performed on symbols from a ring of size q . Such an encoder is depicted in figure 4.1.

The source provides a random stream of q -ary symbols which are divided into frames of n symbols. The symbols in each frame are encoded using a rate $1/2$ *recursive systematic convolutional* (RSC) encoder to produce one set of parity symbols. The systematic symbols are then randomly permuted or interleaved, and passed through a second RSC encoder to provide a second set of parity symbols. The systematic symbols, and both sets of parity symbols are transmitted providing an overall code rate of $1/3$.

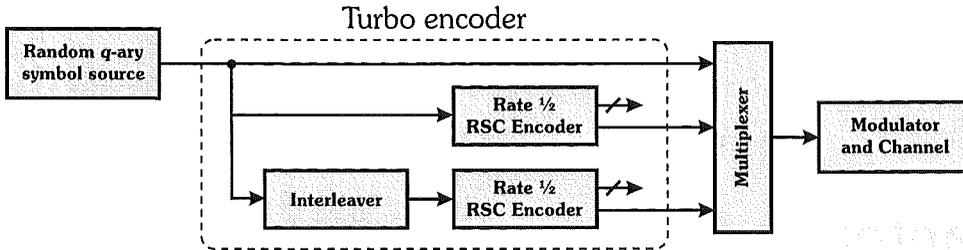


Figure 4.1: Structure of a rate 1/3 turbo encoder

The transmission scheme considered is q -ary-PSK where q is again the size of the ring considered. For example penternary codes constructed over \mathbb{F}_5 are transmitted using 5-PSK. At the receiver, the demodulator calculates soft probabilities $\mathbb{P}(y|x_i)$, $i = 0 \dots q-1$, for each received symbol, where y is the received signal point, and the x_i are each of the possible transmitted constellation points. These probabilities are then passed to the iterative decoder.

In [12], Berkmann describes the application of iterative decoding to the non-binary domain. The block structure of the decoder used is identical to that of the standard binary iterative decoder, one iteration of which is depicted in figure 4.2.

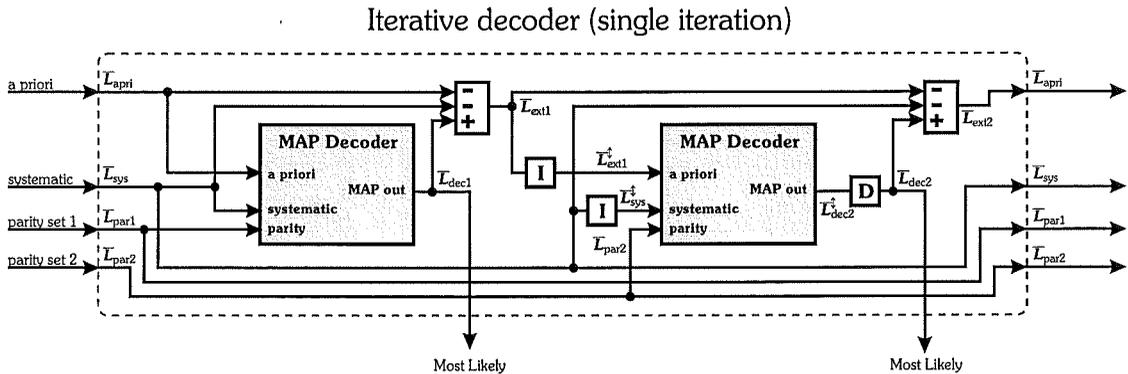


Figure 4.2: Structure of the iterative decoder

In binary iterative decoding, probability information is passed between the constituent decoders in the form of a time-indexed vector of log-likelihood ratios. However, these ratios are inherently binary, so for non-binary codes it becomes necessary to transfer a set of ratios to represent the probabilities associated with each possible symbol value in the frame. For example, to represent the *a priori* probabilities $\mathbb{P}(u_t = u)$ associated with symbol t in a frame of a quaternary code, we use the set of log-likelihood ratios

$$\begin{aligned} \{L\}_{apri,t} &= \{L_{10}, L_{20}, L_{30}\}_t \\ &= \left\{ \ln \left(\frac{\mathbb{P}(u_t = 1)}{\mathbb{P}(u_t = 0)} \right), \ln \left(\frac{\mathbb{P}(u_t = 2)}{\mathbb{P}(u_t = 0)} \right), \ln \left(\frac{\mathbb{P}(u_t = 3)}{\mathbb{P}(u_t = 0)} \right) \right\} \end{aligned} \quad (4.1)$$

It is a simple matter to determine the probabilities $\mathbb{P}(u_t = u)$ from this set of log-

likelihoods.¹

Apart from the necessity to consider sets of ratios for each received symbol (*c.f.* equation 4.1), the iterative process proceeds in an identical fashion to the binary case. The *a priori* probabilities are initially set equal for all symbols. The first component decoder then uses these *a priori* probabilities, and the probabilities associated with the systematic symbols and the first set of parity symbols to provide a soft *maximum a posteriori* (MAP) estimate of the transmitted symbols. From this estimate, we subtract the contribution of the *a priori* and systematic probabilities to obtain the so called *extrinsic* information. The sets of likelihood ratios representing the extrinsic information are then used as *a priori* estimates for the second component decoder which uses the second set of parity data. And so the decoder iterates. After a fixed number of iterations, the iterative process terminates and a hard decision is made based on the MAP estimate of the last component decoder used.

The component decoders simulated implement the BCJR algorithm [3], which maximises the *a posteriori* probabilities of the states and transitions of a Markov source viewed through an additive noise channel. There is nothing inherently binary in this process, so the BCJR algorithm is used without modification. [32] discusses the various MAP decoding algorithms suitable for use in the iterative decoding of binary codes. Any of these that could be adapted to provide sets of probabilities in the form of (4.1) would be suitable for the non-binary situation. Each component decoder takes as its input the *a priori* probabilities for each of the information symbols $\mathbb{P}(u_t = u)$, and the transmission probabilities for both the systematic and parity code symbols $\mathbb{P}(y_{tS}|x_S)$ and $\mathbb{P}(y_{tP}|x_P)$ where x_S and x_P are systematic and parity symbols from the q -ary-PSK transmission constellation, and y_{tS} and y_{tP} are the received systematic and parity signal points. In return the decoder calculates the maximised *a posteriori* probabilities $\mathbb{P}(u_t = u|\mathbf{y}_1^T)$.

4.3 Choosing ‘good’ component codes

For best performance, it is necessary to choose ‘good’ component codes from the set of all possible RSC codes with a particular rate and memory order.

Benedetto and Montorsi [5], and later Benedetto, Garello and Montorsi [9] and also Ho, Pietrobon and Giles [39, 40] addressed this problem in the binary context. [5] demonstrates that the component codes must be recursive for the interleaver to provide significant gain, and that the lowest output parity weight z_{\min} possible from a weight two input is the dominant parameter determining turbo code performance. It is also argued that the denominator polynomial of the function specifying the non-systematic output of a component code should be primitive. These results, and other restrictions on the component encoders, are discussed below.

¹Note that $u_t = 0$ is taken as the reference symbol, but any symbol could be used.

In [5] the procedure for determining the ‘best’ codes is to choose codes that first maximize z_{\min} and then have the best BER performance bound. In [9] a simpler and therefore more easily computed procedure that considers the weight and multiplicity of codewords generated by low weight inputs is used. This latter method (as described in section 4.4) and associated search algorithm are adapted to find good non-binary rate 1/2 RSC component codes for turbo codes using q -ary-PSK modulation.

Ho *et. al.* [39, 40] use an almost identical search method to that presented in [9], however they place additional emphasis on choosing codes with good d_{free} and hence choose different codes as ‘best’ for some parameter sets.

4.3.1 What makes a good component code?

At SNR’s, in the ‘waterfall’ region of the error rate curve, the multiplicity of codewords with both low input and low output weights dominates performance. Specifically the lowest possible output weight due to a weight two input (z_{\min}) is the most important component code parameter and should be maximised.² For SNR’s, in the ‘floor’ portion of the error rate curve, the minimum distance of the turbo code is the most important parameter. Any practical system is likely to operate in the ‘waterfall’ portion of the performance curve.

In section 2.5.3 it is stated that only the codes $\langle a_n \dots a_0 | b_n \dots b_1 1 \rangle$ which have the generator matrix

$$\langle a_n \dots a_0 | b_n \dots b_0 \rangle \triangleq \left[1 \quad \frac{a_n D^n + a_{n-1} D^{n-1} + \dots + a_1 D + a_0}{b_n D^n + b_{n-1} D^{n-1} + \dots + b_1 D + 1} \right] \quad (4.2)$$

will be considered. The restriction to rate 1/2 RSC component codes is done to allow the use of the standard rate 1/3 turbo code structure. The following subsections discuss further non-arbitrary restrictions that can be placed on the code parameters, and other factors that must be considered.

4.3.2 Recursive codes

Only recursive component codes are of interest. Benedetto and Montorsi [5] demonstrate that the component codes must be recursive for the interleaver to provide significant gain. Specifically, no interleaver gain is possible for the weight one error events present in all non-recursive codes, whereas for weight two error events, the interleaver provides a BER gain of $\sim O\left(\frac{1}{N}\right)$ where N is the interleaver length³. Weight two error events are the smallest possible if a recursive component code is used.

A consequence of this is that the lowest output parity weight z_{\min} possible from a weight two input to the component encoder is the dominant parameter determining

²This has the effect of lowering the number of nearest neighbours possible from the overall turbo code.

³The phrase “gain of $\sim O\left(\frac{1}{N}\right)$ ” is slightly misleading, implying a gain of less than one. However in this context it implies a reduction in the BER of $\sim O\left(\frac{1}{N}\right)$ which is a good thing, and hence a gain.

turbo code performance [5]. This is because the interleaver provides more gain for higher weight sequences. In fact codes with less than the largest minimum distance can give the best performance in the waterfall or low SNR region provided that the minimum distance codewords require input sequences of weight greater than two.

Benedetto and Montorsi [9] further state that

“An encoder is *recursive* if, given any sequence $\mathbf{u} \in U^Z$ with Hamming weight $w_H(\mathbf{u}) = 1$, no finite-weight sequences $\mathbf{c} \in C$ exist, such that $(\mathbf{u}, \mathbf{c}) \in \hat{C}_E$.”

In this statement \mathbf{u} is an input sequence, \mathbf{c} is an output sequence and \hat{C}_E represents a particular mapping of input sequences to output sequences by the code. In other words for an input with Hamming weight one, a recursive encoder will not be able to return to the 0-state.

It is of course possible to choose the a_i and b_i in the structure of figure 2.5 such that the encoder has a recursive block structure but in fact has a finite impulse response.⁴ Such encoders may easily be eliminated by checking the impulse response of all possible encoders.

4.3.3 Causality

For an encoder to be causal, its denominator must contain a non-zero constant term (*i.e.* $b_0 \neq 0$).

This statement is true except in the degenerate case where $a_0 = b_0 = 0$, and is easily shown by considering the encoder $\langle a_n \dots a_0 | b_n \dots b_1 \ 0 \rangle$ which relates the input γ and non-systematic output ϕ by

$$\phi(b_n D^n + \dots + b_2 D^2 + b_1 D) = \gamma(a_n D^n + \dots + a_1 D + a_0) . \quad (4.3)$$

Shifting in time so that the left side contains the current output results in

$$\phi(b_n D^{n-1} + \dots + b_2 D + b_1) = \gamma(a_n D^{n-1} + \dots + a_1 + a_0 D^{-1}) \quad (4.4)$$

which is a non-causal system where the present output ϕb_1 requires knowledge of a future input $\gamma a_0 D^{-1}$.

⁴One example is the trivial case where the numerator is a multiple of the denominator. Consider the ternary RSC code $\langle 112|211 \rangle$, this has the generator matrix $\begin{bmatrix} 1 & \frac{D^2+D+2}{2D^2+2D+1} \end{bmatrix} \equiv [1 \ 2]$ which is not recursive.

4.3.4 Further Constraining b_0

When using arithmetic over \mathbb{F}_n only $b_0 = 1$ needs to be considered. This is shown by noting that

$$\begin{aligned} \langle c_n \dots c_0 | d_n \dots d_1 d_0 \rangle &\equiv \left\langle \frac{c_n}{d_0} \dots \frac{c_0}{d_0} \mid \frac{d_n}{d_0} \dots \frac{d_1}{d_0} 1 \right\rangle \\ &\equiv \langle a_n \dots a_0 | b_n \dots b_1 1 \rangle \end{aligned} \quad (4.5)$$

This means that over \mathbb{F}_n every code in the set $\langle c_n \dots c_0 | d_n \dots d_0 \rangle$ is equivalent to a code in the smaller set $\langle a_n \dots a_0 | b_n \dots b_1 1 \rangle$.

This is not true when using rings where $\mathbb{Z}_n \neq \mathbb{F}_n$. In these cases division is not defined, resulting in the enigma of code specifications from which only multiples of the output are available. For example, over \mathbb{Z}_4 , what code does the specification $\langle 102 | 112 \rangle$ represent?⁵ In general the input γ and systematic output ϕ are related by the equation $\phi(b_n D^n + \dots + b_1 D + b_0) = \gamma(a_n D^n + \dots + a_1 D + a_0)$. The output ϕ is available only as ϕb_0 , and so if $b_0 \neq 1$ then the output is not directly available. For this reason the code search is restricted to the cases where $b_0 = 1$ for all rings.

4.3.5 Primitive Polynomials

Is it sufficient to search only primitive feedback polynomials? Benedetto and Montorsi [5] argue convincingly that in the binary case it is. A primitive feedback polynomial maximises the length of the shortest error event possible from a weight two input. This is true in the binary case simply because by definition the smallest x such that $(D^x + 1)$ is a multiple of a binary primitive polynomial of degree ν , is $x = 2^\nu - 1$.

The situation is not so clear in the non-binary case. For fields, it is again true by definition that the smallest x such that $(D^x - 1)$ is a multiple of a q -ary primitive polynomial of degree ν is $x = q^\nu - 1$. However, in general there will be lower degree weight two polynomials $\phi(D^n - \alpha^n)$, $\phi \neq 0$, $\phi \in \mathbb{F}_q$, α a primitive element in $\text{GF}(q^\nu)$ and $n = \frac{q^\nu - 1}{q - 1}$ which contain the same primitive polynomial as a factor. It may be the case that despite this, choosing a primitive feedback polynomial still maximises the length of the shortest error event possible from a weight two input.

This would have two consequences: by maximising the length of the shortest error event due to a weight two input it is easier to maximise z_{\min} the smallest output parity weight possible from a weight two input; also by maximising this length the number of weight two events that can be present in a single frame is minimised.

Because searches will be conducted over both rings and fields, and as the additional burden of searching over all feedback polynomials is not significant, this will be done for completeness.

⁵In this particular case, twice the current parity output 2ϕ is obtained from current input γ and previous inputs and outputs via $2\phi = \gamma(D^2 + 2) - \phi(D^2 + D)$. However ϕ can never be obtained from 2ϕ as division is not defined.

4.4 Procedure for determining ‘best’

The following procedure is an extension of the method used by Benedetto *et al.* [9] to identify good binary component codes.

Consider the *input-output weight enumerating function* (IOWEF) for all possible rate 1/2 RSC component codes with the considered state complexity, constructed over the considered ring. Choose as ‘best’ the codes that first maximise the minimum output weight possible from a weight two input, and second minimise the multiplicity of these low weight outputs. Next consider weight three and higher inputs, and if necessary the second lowest output weight due to weight two inputs, etc.; until a set of one or more ‘best’ codes which are identical under this metric have been identified.

Depending on the size of the ring considered and the corresponding modulation, this procedure may be performed using either Hamming or Euclidean weights.

4.5 Search algorithm

Practical considerations prevent the complete IOWEF of candidate component codes from being determined for any reasonable frame length. However, as we are only interested in the portion of the IOWEF resulting from low weight inputs, the following method can be used to approximately apply the metric. Again this is largely an adaptation of the algorithm used by Benedetto *et al.* [9] to identify good binary component codes.

Cycle through each input weight i starting with $i = 2$ and continuing until enough information is gained to separate all the possible component codes into sets with identical IOWEFs.

For each i generate all possible input sequences of this weight that have a 1 in the first position⁶ and length less than some reasonable length l (discussed below).

Pass each of these sequences through the component encoder. If the encoder terminates in the zero state, and if the zero state was not re-entered and left again during the sequence, then increment the appropriate term in the IOWEF.

Essentially what this process does is enumerate all the simple error events⁷ of length $\leq l$ that are generated from low weight inputs. This approximate IOWEF is then used to determine the ‘best’ codes.

⁶This forces immediate divergence from the all-zero trellis path.

⁷Simple error events are those that diverge immediately from the zero state, and which can not be constructed from the concatenation of two shorter error events.

In practice it can not be guaranteed that the codes identified have identical IOWEFs past the depths i and l of the search. The strategy used is to increase i and l until it is apparent that searching deeper into the code trellis would not discriminate the codes further.

4.5.1 Discussion

The proposed search algorithm approximates a metric based on the full IOWEF of the component code, by considering the small portion of the IOWEF due to short simple error events that are the result of low weight inputs. This requires some justification:

- Only input sequences up to length l are considered because the longer the length of the input, the more likely it is to produce a high weight output. Specifically, as long as there are no all 0/0 loops in the code's state diagram⁸ then as the sequence gets longer either or both of the input/output weights have to rise, eventually giving a sequence with high input/output weight.⁹
- A worst case reasonable l can be determined if it is assumed that apart from the zero state self loop, the state diagram contains no additional 0/0 loops, and that only simple error events are of interest. In this assumed situation a maximum required search depth can be quantified. If the longest 0/0 path has length v , then at a minimum of every $v + 1$ steps either the systematic or parity weight will increase by one. (See figure 4.3.) Therefore all codewords with a combined systematic and parity weight of n will be identified by searching inputs up to $n(v + 1)$ in length.^{10,11}
- The decision to only consider simple error events, and error events that diverge from the zero path immediately stems from work of Benedetto *et al.* [9] who consider only these events, and state:

“Only single [simple] error events are considered, as they contribute to the error probability upper bound; the sequences made by the concatenation of two or more shorter error events are not included.”

This justification is based on the bounding method in [6] which only assumes that the *weight enumerating function* (WEF) of the convolutional code is known, and attempts to estimate the weight and multiplicity of word in the associated block code from this.

⁸A 0/0 loop is a state sequence that can be followed indefinitely where a zero input always gives a zero output. The 0/0 self loop on the zero state that all linear codes possess is not included.

⁹Any code whose state diagram contains such 0/0 loops must be catastrophic, and by definition the systematic convolutional codes considered can not be catastrophic [37].

¹⁰This argument assumes Hamming weights are used, but an identical argument would work for Euclidian output weights.

¹¹This problem is discussed in [58] where a length L_n is defined that is the maximum length of a single error event of weight n in the code.

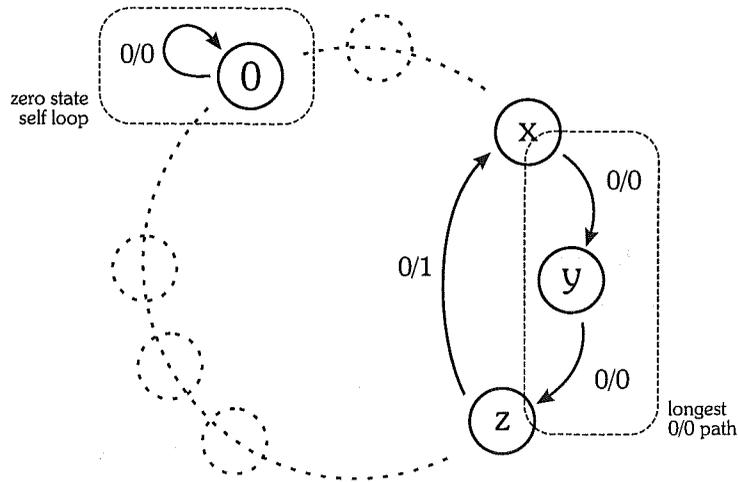


Figure 4.3: Example state diagram showing the 0/0 paths.

The last assumption raises the question of whether considering only simple error events that diverge immediately provides enough information to adequately apply the chosen metric. This is justified by the results presented in section 5.1.4 which consider very short frame lengths where these approximations are likely to be least accurate.

4.6 Lower bound on z_{\min}

The dominant component of the chosen metric is z_{\min} , the minimum output parity weight due to a weight two input. Benedetto and Montorsi [5] prove that in the binary case and considering Hamming distance, there exist rate 1/2 RSC codes with memory order ν that achieve

$$z_{\min} = 2^{\nu-1} + 2 \quad (4.6)$$

This bound can be extended to non-binary codes so that

$$z_{\min} = q^{\nu-1} + 2 \quad (4.7)$$

is achievable for a rate 1/2 convolutional component code with memory ν over \mathbb{F}_q when considering Hamming distance. This fact allows the code search to be simplified so as to examine only codes which achieve this bound. It is also conjectured that this z_{\min} is the maximum achievable as no exceptions to this were observed.

The proof that (4.7) is achievable is a generalisation of that given in [5] for the binary case. Beginning with the definition [13],

Definition:

A *constacyclic* code is any code C over \mathbb{F}_q whose codewords are multiples of a generator polynomial $g(D)$ modulo $D^n - \xi$, where $\xi \in \mathbb{F}_q$ and whose codewords additionally

satisfy the property that if $C_0, C_1, C_2, \dots, C_{n-1}$ is a codeword then its constacyclic shift $\xi C_{n-1}, C_0, C_1, \dots, C_{n-2}$, is also a codeword.

The following theorem can then be proven.

Theorem:

A rate $1/2$ q -ary convolutional encoder defined over \mathbb{F}_q with memory ν and generator matrix $G = [1 \ \frac{k(D)}{d(D)}]$, where $d(D)$ is a primitive polynomial can achieve $z_{\min} = q^{(\nu-1)} + 2$. Indeed where $d(D)$ is primitive of degree ν , some $k(D)$ of the form $k(D) = \gamma_\nu D^\nu + \dots + \gamma_0$, $\gamma_\nu \neq 0$, $\gamma_0 \neq 0$, will result in an encoder that achieves this value of z_{\min} .

Proof:

Let $n = \frac{q^\nu - 1}{q - 1}$.

Assuming $k(D)$ has degree $\deg[k(d)] \leq \nu$, we first prove that all polynomials $k(D)$ with $\deg[k(D)] < \nu$ yield a value of z_{\min} strictly less than the right-hand side of (4.7).

Since $d(D)$ is primitive, it is the generator polynomial of an $(n, n - \nu)$ constacyclic Hamming code G over \mathbb{F}_q [13]. In addition, $\phi(D^n - \alpha^n)$, $\phi \neq 0$, $\phi \in \mathbb{F}_q$, α a primitive element in $\text{GF}(q^\nu)$, must be the shortest weight two inputs that produce finite weight error events, and these error events must be the lowest weight possible from a weight two input. Moreover, the quotient $h(D)$ obtained from the division of $(D^n - \alpha^n)$ by $d(D)$ is the generator polynomial of the dual (n, ν) constacyclic code H . The products $h(D)k(D) = (D^n - \alpha^n)\frac{k(D)}{d(D)}$ are codewords of this code, in which all codewords (except the zero codeword) have the same weight $z = q^{\nu-1}$ which is strictly less than (4.7). This completes the first part of the proof.

To increase the value of z_{\min} , we must increase the degree of $k(D)$ to ν . We now prove that there exist polynomials $k(D)$ of the form $k(D) = \gamma_\nu D^\nu + \dots + \gamma_0$ that can achieve (4.7). Decompose $k(D)$ as $k(D) = D \cdot \gamma_\nu D^{\nu-1} + k_2(D) \triangleq D \cdot k_1(D) + k_2(D)$ and consider that the products $c_1(D) = h(D)k_1(D)$ and $c_2(D) = h(D)k_2(D)$ are codewords of H . Furthermore as $h(D)$ has the form $(D^{n-\nu} + \dots + \eta_0)$, $c_1(D)$ must have the form $(\gamma_\nu D^{n-1} + \dots + \gamma_\nu \eta_0 D^{\nu-1})$ and $c_2(D)$ the form $(\beta_x D^x + \dots + \gamma_0 \eta_0)$, $x \leq D^{n-1}$.

The product $Dc_1(D)$ represents a (non-constacyclic) shift of one position to the left of $c_1(D)$. Since the constant weight code H is constacyclic, and $c_1(D)$ has degree $n - 1$, the word represented by $Dc_1(D)$ coincides with a codeword of the H except for the most significant symbol (γ_ν corresponding to the power D^n) and the least significant symbol (a '0' instead of $\frac{\gamma_\nu}{\alpha^n}$ which would follow from a constacyclic shift of the codeword $c_1(D)$). Thus, summing the words represented by $Dc_1(D)$ and $c_2(D)$ yields, for the powers from D up to D^{n-1} , part of a codeword of H . This portion of the codeword will have weight $q^{\nu-1}$ provided that $\frac{\gamma_\nu}{\alpha^n} + \gamma_0 \eta_0 = 0$. As to the remaining powers, D^n contributes 1 to the weight, and D^0 gives another 1 since $c_2(D)$ has least significant bit equal to $\gamma_0 \eta_0$ whereas $Dc_1(D)$ has this bit equal to '0'.

This completes the proof. ■

4.7 Geometric Uniformity

The use of the search algorithm presented in section 4.5 to conduct the Euclidean distance searches whose results are presented in chapter 5 raises the question of geometric uniformity. In general, a linear code will not preserve its linearity when mapped onto a signal constellation.

The properties of and conditions for geometrically uniform codes are discussed extensively in [29]. If a code is geometrically uniform, then it looks the same from any codeword and all the decision (Voronoi) regions are the same shape. A consequence of this is that the distance profile from every code word is identical and so the WEF of the code can be determined by considering only deviations from the all-0 path.

There are many combinations of linear codes and constellations which are geometrically uniform. For example, all linear binary codes mapped onto BPSK. In the context of this thesis, the codes considered are simple q -ary convolutional codes mapped onto q -ary-PSK modulation. This is not a situation specifically dealt with in the literature, however the work of Ungerboeck [59] may shed some light on the subject. In [59] which discusses the design of trellis codes, systematic binary convolutional codes are mapped onto 8-PSK via a binary partition of the constellation. Furthermore, it is proven that such a mapping preserves geometric uniformity. Perhaps this result can be extended to cover the mapping of q -ary convolutional codes onto q -ary-PSK via a q -ary partition of the constellation.

It should be noted that the geometric uniformity of the complete turbo code need not be considered as the search method presented only examines the properties of the individual component codes. This problem is nonetheless interesting and is discussed in [31] with respect to binary serially-concatenated-codes.

It is unclear whether or not the non-binary component codes considered in this thesis are geometrically uniform when mapped onto q -ary-PSK and considering Euclidean distance. If they are, then the searches presented in 5 are exhaustive. If they are not, then the searches presented are not exhaustive because only error events with respect to the all-0 path are enumerated. Alternative search algorithms that consider distances with respect to all codewords do exist (e.g. [64]), however these have not been pursued.

In either case, the codes identified will be good codes with respect to the search criteria presented in section 4.3. Additionally all the simulation results presented transmit random data not just the all-0 codeword.

4.8 Conclusion

This chapter has extended the search criteria of [9] to develop a procedure for identifying good q -ary RSC component codes for non-binary turbo codes. Additionally, an achiev-

ability bound for the dominant parameter z_{\min} has been determined and proved for all fields \mathbb{F}_q .

Chapter 5

Searches and Simulations

This chapter presents the results of searches for good non-binary component codes for turbo codes. Firstly the binary results of previous authors are emulated and commented on, then tables of good codes over small q -ary rings are presented. Simulation results are provided for many of the simpler codes identified. The domain considered is rate $1/3$ turbo-codes with identical rate $1/2$ RSC component codes transmitted through an AWGN channel using q -ary-PSK modulation.

5.1 Binary Codes

Benedetto, Garelo and Montorsi [9] have published tables of the optimum binary component codes found using the pairwise optimisation criteria discussed in chapter 4. The object of this section is to replicate their results, comment on the effectiveness of the search criteria and look at simulations and bounds that use the chosen component codes.

5.1.1 The ‘best’ component codes

Benedetto *et al.*'s ‘best’ rate $1/2$ binary RSC codes are presented in table 5.1. The d_i, N_i are respectively the minimum output Hamming weight (parity only) and number of words with this output weight resulting from inputs of weight i .¹

code	states	d_2, N_2	d_3, N_3	d_4, N_4	d_5, N_5	d_6, N_6	d_{free}
$\langle 10 11 \rangle$	2	1,1	—	—	—	—	3
$\langle 101 111 \rangle$	4	4,1	2,1	2,1	2,1	2,1	5
$\langle 1111 1011 \rangle$	8	6,1	4,3	2,1	4,9	2,1	6
$\langle 11111 10011 \rangle$	16	10,1	5,3	2,1	5,14	2,1	6
$\langle 100101 110111 \rangle$	32	18,1	7,2	4,1	3,2	4,5	8

Table 5.1: The ‘best’ binary rate $1/2$ RSC component from [9].

¹This format for presenting results differs from that in [9], but the same information is provided.

Table 5.2 presents the equivalent results from the searches conducted during the present work. Again, the d_i, N_i are respectively the minimum output Hamming weight (parity only) and number of words with this output weight resulting from inputs of weight i . In addition a second d_i, N_i pair is shown, this is simply the next lowest parity weight possible from inputs of weight i and its multiplicity.

codes	states	d_2, N_2	d_3, N_3	d_4, N_4	d_5, N_5	d_6, N_6	d_{free}
$\langle 10 11 \rangle$ $\langle 01 11 \rangle$	2	1,1 2,1	—	—	—	—	3
$\langle 101 111 \rangle$	4	4,1 6,1	2,1 4,3	2,1 4,6	2,1 4,10	2,1 4,15	5
$\langle 1111 1011 \rangle$ $\langle 1111 1101 \rangle$	8	6,1 10,1	4,3 6,2	2,1 4,3	4,9 6,29	2,1 4,10	6
$\langle 11111 10011 \rangle$ $\langle 11111 11001 \rangle$	16	10,1 18,1	5,3 7,3	2,1 4,2	5,14 7,40	2,1 4,10	6
$\langle 100101 110111 \rangle$ $\langle 101001 111011 \rangle$	32	18,1 34,1	7,2 11,3	4,1 6,5	3,2 5,6	4,5 6,23	8

Table 5.2: The ‘best’ binary rate 1/2 RSC codes found using the pairwise optimisation criterion.

These codes are, as expected, identical to those identified by Benedetto et. al. [9], as they were chosen in the same manner, by sequentially optimising the d_i, N_i pairs. In all but the 4-state case, an additional *equivalent code* was identified that had the exactly same weight spectrum. These equivalent codes are listed for completeness, and turbo codes constructed from different equivalent codes should have the same performance characteristics.

Dashes ‘—’, are used to indicate a situation specific to the 2-state binary codes where there are not any simple error events with an input weight of 3 or more.

5.1.2 Simulation and bounds

Figure 5.1 presents simulation results for the rate 1/3 turbo codes associated with three of the component codes identified in section 5.1.1. The frames were 1000 data bits in length and terminated in the manner suggested in [22].² An S -random interleaver was used with $S = 21$ [23], and each frame was decoded for 10 full iterations.³ Enough frames were transmitted to ensure that at least 50 frames containing errors were received.

Figure 5.1 also shows the high SNR average upper performance bound for each code calculated using the method described in [21].⁴ The most astonishing thing about this

²This results in total transmitted frame lengths of between 3006 and 3012 symbols depending on the length of the terminating tails.

³One iteration is two passes through the component decoder, once with each set of parity bits.

⁴[6] provides an alternative method for calculating these bounds that appears to give identical results

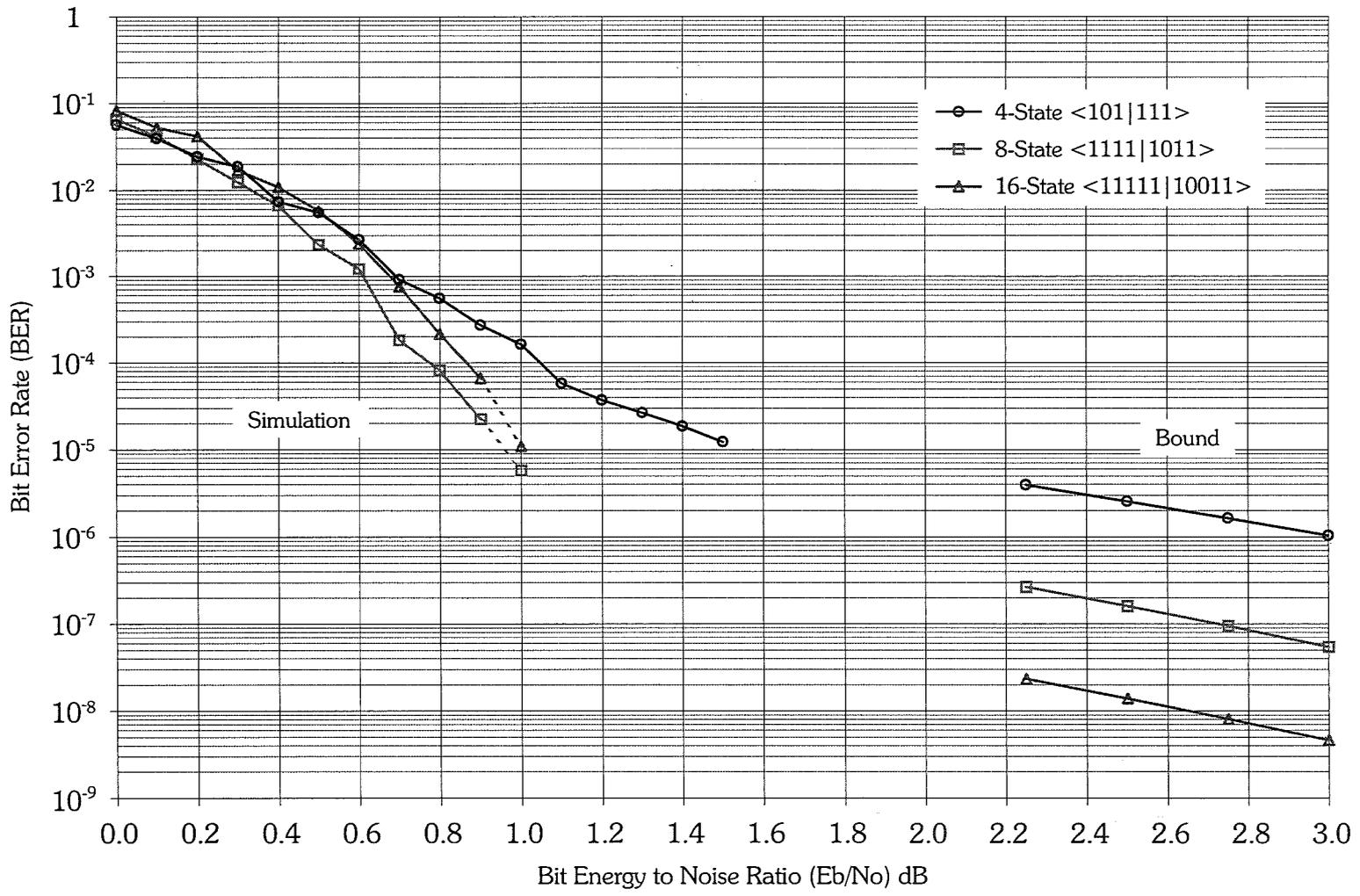


Figure 5.1: Simulated and bounded performance results for turbo codes based on some of the component codes in table 5.2. The turbo codes simulated are rate 1/3 with 1000 data symbols per frame, use an S-random interleaver and were decoded for 10 iterations. The bounds are union bounds. See text for details.

plot is the fact that the more complex 16-state code is out performed by the 8-state code at low SNRs.⁵ Similar results are shown in [39]. Closer investigation shows that these codes have similar *frame error rates* (FERs), but for the 16-state code the frames in error contain more bit errors.⁶ Table 5.3 presents data on the frames in error for the 0.8dB simulation points after 10 iterations. Notice that in particular for the best performing 8-state code the number of errors in a frame is either quite small or very large. This result is expected [1].

The two iteration and five iteration performance have also been checked. In both cases the 8-state code is still better. This rules out the possibility that the 16-state code has the advantage of converging more quickly. In fact it appears that the 16-state code is worse in all cases, although different frame lengths were not checked. One hypothesis is that the 16-state code is stronger but when it fails, it fails more dramatically. However, referring to table 5.3, if this is the case, why does it not have a better FER? Another hypothesis is that the S -random interleaver is partially responsible for this strange result. With $S = 21$, the interleaver will disrupt the four shortest weight two events in the 8-state component code, but only the two shortest in the 16-state code.

Recent work by ten Brink [17] which examines the transfer of extrinsic information between the component decoders back these observations up. His works shows that symmetrical turbo codes⁷ based on more complex component codes will converge more slowly than those based on simpler codes. However after a large number of iterations they will out perform simpler codes for sufficiently high SNRs.

5.1.3 Alternative ‘best’ codes

Pairwise optimisation does not always choose codes that are obviously the best. In table 5.4 two competing 16 state codes are presented along with the previously chosen code.

All three codes have identical enumerating functions in response to weight two error events (the d_2, N_2). However, the first competitor has a better d_{free} and with the exception of the initial $d_3, N_3 = 3, 1$, the second competitor has a superior and more sparse set of distances, although its multiplicities appear high.

Figure 5.2 presents simulation results for these codes at low SNR’s. The simulation parameters were identical to those used for figure 5.2, except that at each SNR exactly the same number of frames were transmitted using identical data and noise sequences for

for SNR’s greater than the computational cutoff rate $R_0 \approx 2.03\text{dB}$. Both these methods calculate an upper union bound averaged over all random interleavers. The use of an S -random interleaver in the simulations should provide better than average performance.

⁵The more complex component codes will always give better performance a high SNRs as the provide a higher d_{free} .

⁶The fact that the 16-state code has very few low weight error events would make it ideal for use in combination with an ARQ scheme. Frames containing errors would be easier to detect reliably. (see section 3.5)

⁷symmetrical turbo codes such as those studied in this thesis have identical component codes.

code	states	number of errors in each of the 50 frames in error	μ	number of frames	BER	FER
$\langle 101 111 \rangle$	4	2, 36, 46, 27, 29, 10, 4, 10, 16, 2, 25, 6, 24, 2, 5, 21, 5, 19, 2, 7, 38, 2, 10, 8, 2, 2, 1, 8, 13, 10, 59, 7, 103, 1, 2, 32, 28, 7, 4, 21, 16, 2, 2, 10, 20, 2, 2, 14, 2, 51	15.5	1367	5.7×10^{-4}	0.0366
$\langle 1111 1011 \rangle$	8	2, 28, 107, 122, 4, 71, 3, 85, 133, 14, 140, 4, 9, 63, 45, 2, 21, 3, 8, 71, 19, 61, 3, 89, 54, 30, 87, 36, 30, 47, 63, 32, 61, 55, 41, 2, 66, 62, 4, 93, 2, 2, 80, 3, 2, 68, 4, 23, 2, 1	41.1	25120	8.2×10^{-5}	0.0020
$\langle 11111 10011 \rangle$	16	125, 21, 67, 44, 36, 156, 89, 84, 74, 8, 165, 72, 18, 58, 157, 125, 78, 80, 79, 133, 8, 117, 88, 124, 30, 63, 79, 6, 144, 62, 70, 112, 150, 96, 161, 96, 170, 29, 156, 110, 4, 67, 145, 84, 67, 110, 131, 61, 123, 74	88.1	20407	2.2×10^{-4}	0.0025
$\langle 100101 110111 \rangle$	32	117, 107, 50, 109, 134, 85, 77, 153, 117, 150, 82, 161, 22, 118, 138, 19, 157, 160, 156, 124, 165, 141, 157, 39, 81, 122, 179, 140, 100, 110, 141, 136, 98, 150, 162, 126, 134, 80, 147, 123, 135, 29, 91, 168, 179, 157, 118, 168, 98, 154	121	6518	9.3×10^{-4}	0.0077

Table 5.3: Statistics on the frames in error for the 0.8 dB simulation points in figure 5.1 and for the related turbo code based on 32 state component codes. (μ is the mean number of errors in frames which contain errors.)

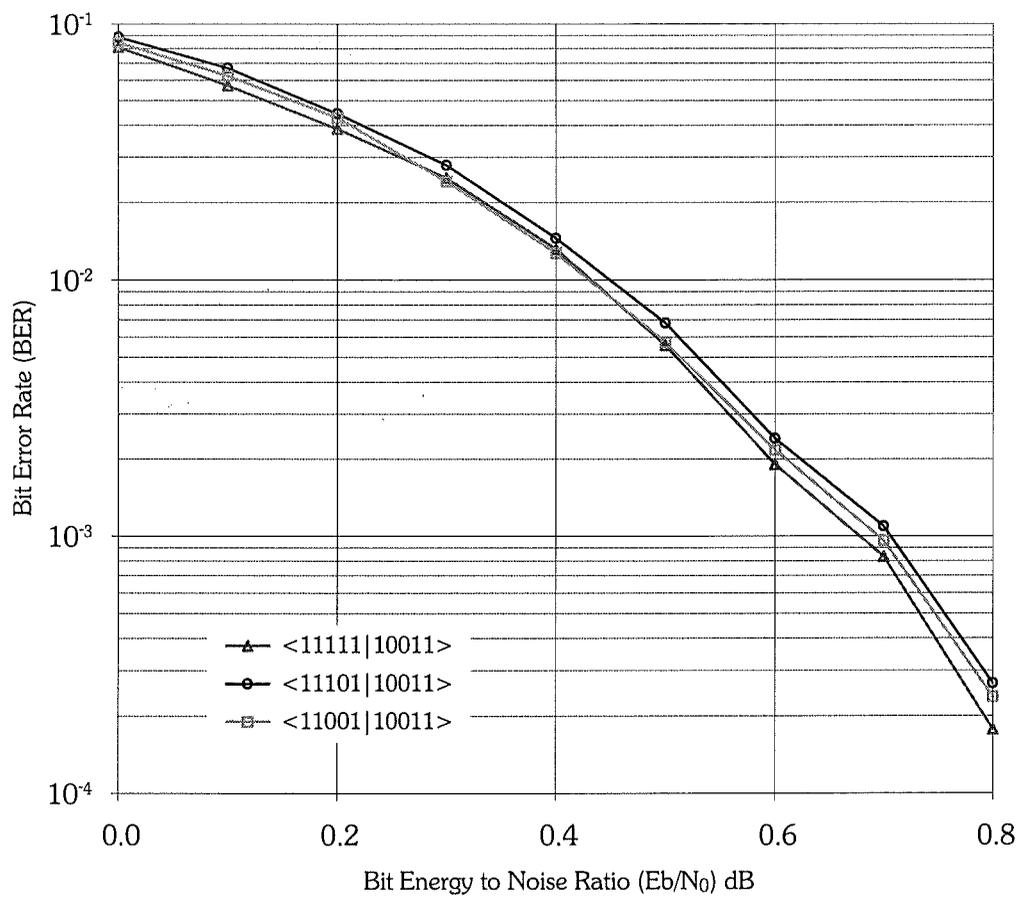


Figure 5.2: Simulation results for competing 16-state component codes. See text for details.

codes	states	d_2, N_2	d_3, N_3	d_4, N_4	d_5, N_5	d_6, N_6	d_{free}
$\langle 11111 10011 \rangle$ $\langle 11111 11001 \rangle$	16	10,1	5,3	2,1	5,14	2,1	6
		18,1	7,3	4,2	7,40	4,10	
		26,1	11,4	6,6	9,86	6,40	
		34,1	13,8				
		42,1	15,7	⋮	⋮	⋮	
		50,1	17,1				
58,1	19,8						
$\langle 11101 10011 \rangle$ $\langle 10111 11001 \rangle$	16	10,1	4,1	4,3	2,1	4,8	7
		18,1	6,2	6,7	4,2	6,42	
		26,1	8,2	8,12	6,19	8,170	
		34,1	10,3				
		42,1	12,6	⋮	⋮	⋮	
		50,1	14,6				
58,1	16,4						
$\langle 11001 10011 \rangle$ $\langle 10011 11001 \rangle$	16	10,1	3,1	4,4	5,14	6,68	6
		18,1	7,4	8,27	9,161	10,906	
		26,1	11,9	12,79	13,605	14,4239	
		34,1	15,10				
		42,1	19,17	⋮	⋮	⋮	
		50,1	23,16				
58,1	27,25						

Table 5.4: The ‘best’ 16-state binary code (top), and two competitors

the three different component codes.⁸ This gives results that are more precise, but less accurate, allowing for easier comparison between the codes.

These results indicate that the search criteria did indeed choose the ‘best’ code for practical error rates ($< 10^{-3}$). The superior distances of the second competitor did however give it a slight advantage at the 0.3dB and 0.4dB points. As expected the better minimum distance of the first competitor did not give it an advantage at low SNRs.

5.1.4 Very short frame lengths

The ‘single error events that diverge immediately’ approximation discussed in section 4.5.1 is necessary to make code searches practical. This is because, with the exception of turbo codes with very short frame lengths, it is impractical to calculate the full WEFs of the component codes.

In this sub-section, the code search of section 5.1.1 is repeated, assuming a frame contains only 20 data bits including the terminating tail sequences. This allows both complex error events and error events that do not diverge immediately to be considered when applying the procedure defined in section 4.4. Table 5.5 presents the ‘best’ codes

⁸The number of frames transmitted at each SNR from 0.0dB up were as follows: 400, 400, 400, 500, 700, 1200, 3000, 6500, 22000. Each simulation point started with identical seeds.

found under these conditions.

codes	states	d_2, N_2	d_3, N_3	d_4, N_4	d_5, N_5	d_6, N_6	d_{free}
$\langle 10 11 \rangle$ $\langle 01 11 \rangle$	2	1,19 2,18	—	2,153 3,272	—	3,680 4,1680	3
$\langle 101 111 \rangle$	4	4,17 6,14	2,18 4,46	2,16 4,82	2,14 4,120	2,12 4,275	5
$\langle 1111 1011 \rangle$ $\langle 1111 1101 \rangle$	8	6,13 10,6	4,46 6,23	2,16 4,41	4,104 6,261	2,12 4,102	6
$\langle 11111 10011 \rangle$ $\langle 11111 11001 \rangle$	16	10,5 —	5,39 7,25	2,15 4,23	5,135 7,223	2,10 4,86	6
$\langle 101001 110111 \rangle$ $\langle 100101 111011 \rangle$	32	—	7,35 9,5	4,10 6,52	3,15 5,72	4,30 6,125	8

Table 5.5: The ‘best’ binary rate 1/2 RSC for frames containing 20 data bits found using the pairwise optimisation criterion but without using the ‘simple error events that diverge immediately’ approximation.

Comparing these codes with those presented in table 5.2 shows that identical codes were chosen as best for 2 to 16 states. The comparison is in a sense invalid for the 32 state codes as a frame length of 20 is so short that no weight two input error events can occur, and hence this cannot be the primary criteria for determining the best codes.⁹ These results provide evidence that the ‘single error events that diverge immediately’ approximation is valid.

5.2 Ternary Codes

Table 5.6 presents the ‘best’ ternary component codes found using the method presented in section 4.5.

As for the binary search results, the bold d_i, N_i pairs in the table are the Hamming weight and multiplicity of the lowest weight parity sequences possible from a weight i information sequence. The second indented d_i, N_i pair are the weight and multiplicity of the next lowest weight parity sequences possible. Furthermore, the notation $\pm(\dots|\dots)$ is used to represent the pair of codes $\langle a_n \dots a_0 | b_n \dots b_0 \rangle$ and $\langle -a_n \dots -a_0 | b_n \dots b_0 \rangle$ when they have identical weight spectra.

For example, if we consider row two of the table which lists the ‘best’ 9-state ternary component codes, we see that there are four codes with identical weight spectra $\langle 12|11 \rangle$, $\langle 21|11 \rangle$, $\langle 11|21 \rangle$ and $\langle 22|21 \rangle$. For each of these component codes the minimum parity weight possible from a weight two input is 5, and there are two different weight two

⁹Given this, a very short turbo code based on these 32 state component codes should have very high interleaver gain – maybe $\sim O(\frac{1}{N^2})$. It would be interesting to see how such a code performs.

inputs that will give this output weight. The next lowest parity weight possible from a weight two input is 8, and again there are two different inputs that will give this parity weight. As expected, all the ‘best’ codes identified achieve $z_{\min} = 3^{\nu-1} + 2$.

codes	states	d_2, N_2	d_3, N_3	d_4, N_4	d_5, N_5	d_6, N_6	d_{free}
$\pm\langle 12 11\rangle$	3	2,2	2,2	2,2	2,2	2,2	4
$\pm\langle 11 21\rangle$	($\nu = 1$)	3,2	3,4	3,6	3,8	3,10	
$\pm\langle 112 211\rangle$	9	5,2	3,6	4,26	2,2	3,18	6
$\pm\langle 122 221\rangle$	($\nu = 2$)	8,2	6,18	7,134	5,144	6,802	
$\pm\langle 1111 1121\rangle$	27 ($\nu = 3$)	11,2 20,2	5,6 6,4	2,2 4,6	4,10 5,22	2,2 4,22	6
$\pm\langle 1111 1211\rangle$							
$\pm\langle 1212 2111\rangle$							
$\pm\langle 1212 2221\rangle$							
$\pm\langle 10202 21121\rangle$	81 ($\nu = 4$)	29,2 56,2	6,2 9,2	6,10 7,2	2,2 3,4	3,4 4,6	7
$\pm\langle 10102 21221\rangle$							
$\pm\langle 10202 22111\rangle$							
$\pm\langle 10102 22211\rangle$							

Table 5.6: The ‘optimum’ ternary rate 1/2 component codes

Simulation results for rate 1/3 turbo codes based on some of these component codes are presented in figure 5.3. The simulation frames contained 1000 data symbols and the parity sequences were terminated using the method described in [22]. The symbol interleaver used had an S-random construction with $S = 21$ [23]. Transmission was over an AWGN channel using 3-PSK modulation (see figure 5.5). Ten decoder iterations were performed, and the simulation was stopped when 50 frames containing errors had been received.

Figure 5.3 graphs *symbol error rate* (SER) against equivalent bit energy to noise ratio (equivalent BER). SER is used instead of bit error rate as each symbol now represents more than one bit of information, and there is no clear way of equating ternary symbol errors with binary bit errors (see section 5.6). The number of binary bits represented by each ternary symbol is however known and an equivalent bit energy to noise density ratio can be determined as

$$\frac{E_b}{N_0} = \frac{E_s}{N_0 \log_2 3}, \quad (5.1)$$

or in the more general case of q -ary-PSK

$$\frac{E_b}{N_0} = \frac{E_s}{N_0 \log_2 q}. \quad (5.2)$$

These simulation results show that as we would expect, a turbo code based on the single memory element component code performs poorly. However the 9-state code generates a turbo code with very respectable performance. In a similar situation to that

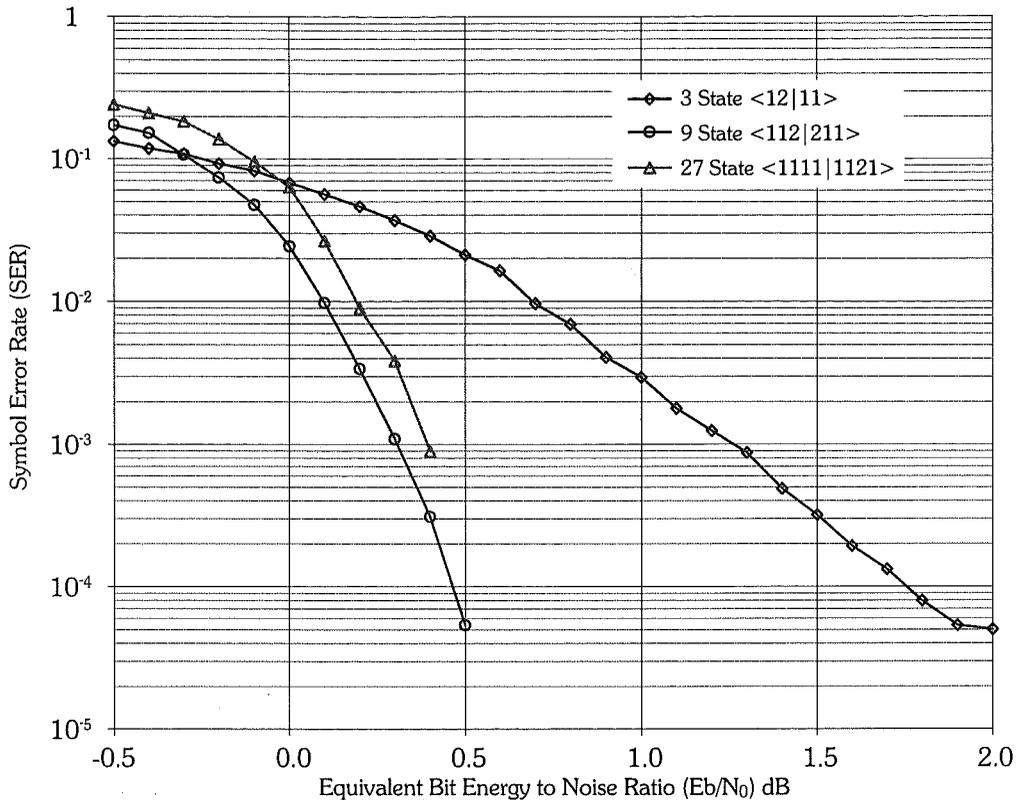


Figure 5.3: Simulation results for ternary turbo codes.

observed with the 8- and 16-state binary codes, the 9-state outperforms the more complex 27-state code for the SERs represented and ten decoder iterations.

The 9-state codes presented in this section are in fact the best component codes identified in this chapter. The weight spectrum of these codes is unique amongst the codes chosen as ‘best’ and possibly related to their very good performance. An extended set of IOWEF terms is presented in table 5.7.

codes	d_2, N_2	d_3, N_3	d_4, N_4	d_5, N_5	d_6, N_6	d_{free}
	5,2	3,6	4,26	2,2	3,18	
$\pm\langle 112 211 \rangle$	8,2	6,18	7,134	5,144	6,802	6
$\pm\langle 122 221 \rangle$	11,2	9,32	10,338	8,914	9,5992	
	14,2	12,46	13,640	11,2974	12,23658	

Table 5.7: Extended table of IOWEF terms for the 9-state component code.

Two aspects are immediately obvious. First, the jump between consecutive d_i for a specific i is always three, and second, the multiplicities N_i are much larger than for the other codes chosen as best.¹⁰ Additionally, despite applying a search procedure which does not even consider overall minimum distance, these 9-state codes have the

¹⁰No other ‘best’ codes have these properties, however the last alternate 16-state binary codes suggested in table 5.4 does. It jumps four between d_i for the same i and has large N_i .

best minimum distance possible.¹¹ This d_{free} event is also the result of a weight 3 input which is far more likely to be disrupted by the interleaver than a weight 2 input. Finally, the state diagrams for these codes are very symmetric as shown in figure 5.4.

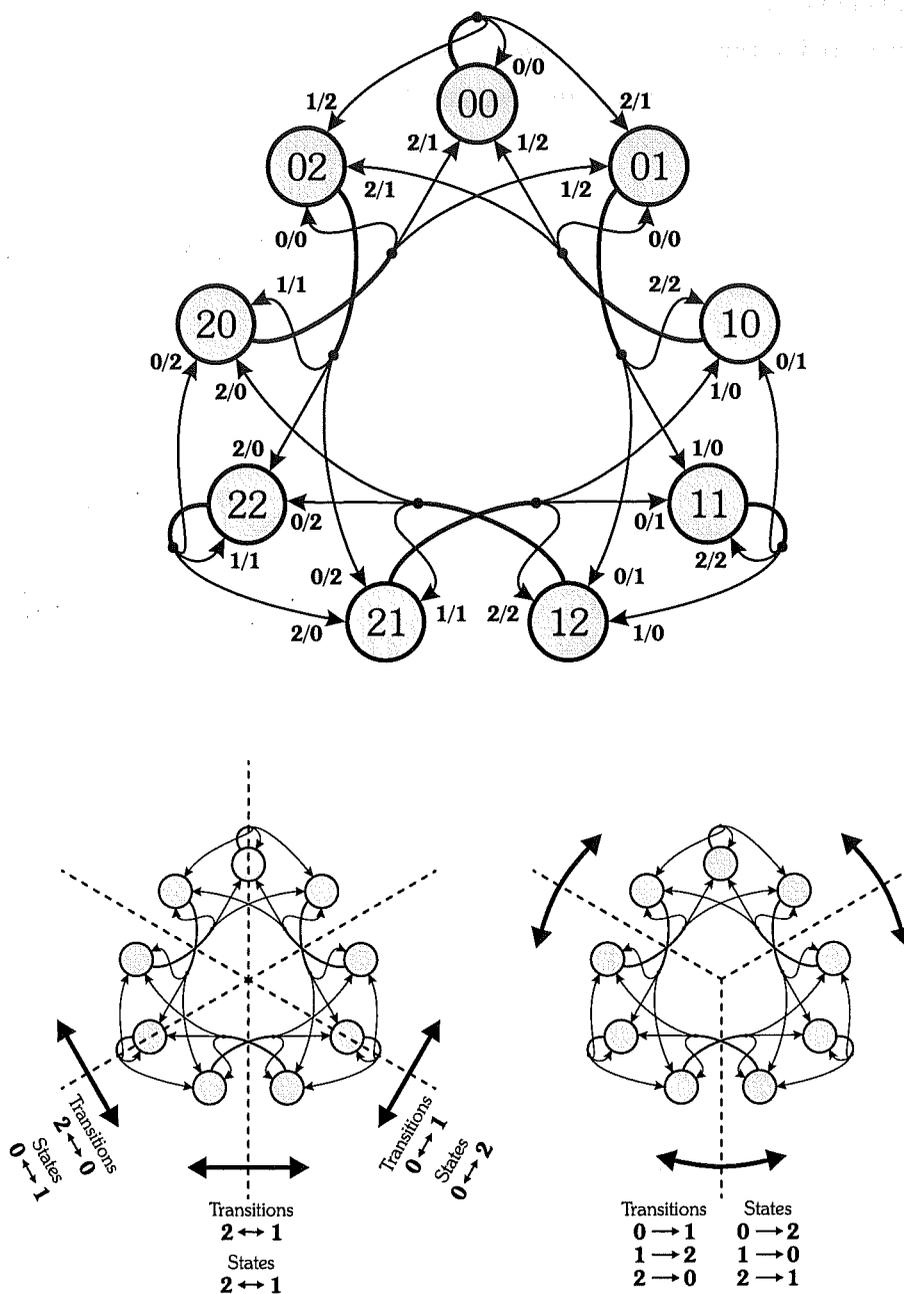


Figure 5.4: State diagram for the 9-state $\langle 112|211 \rangle$ ternary component code with sub diagrams identifying reflective and rotational symmetries.

¹¹This is known as the code searches were exhaustive.

5.3 Quaternary Codes

The extension of the code searches to quaternary arithmetic introduces some new variables. First, there is now the option of different rings/fields for the arithmetic. The Galois extension field $\text{GF}(2^2) \equiv \mathbb{F}_4$, and the other four element rings \mathbb{Z}_4 , $\mathbb{F}_2 + u\mathbb{F}_2$ and $\mathbb{F}_2 + v\mathbb{F}_2$ are all considered in this section. Secondly the modulation scheme is now 4-PSK¹², which means that the distance between any two constellation points is not a constant and so Euclidean distance is no longer equivalent to Hamming distance. The different distances possible in the constellations for 2-PSK, 3-PSK, 4-PSK and 5-PSK are depicted in figure 5.5. For convenience, we define the distance between two closest constellation points to be 1. Finally, different symbol to constellation mappings are possible. These are shown in figure 5.6.

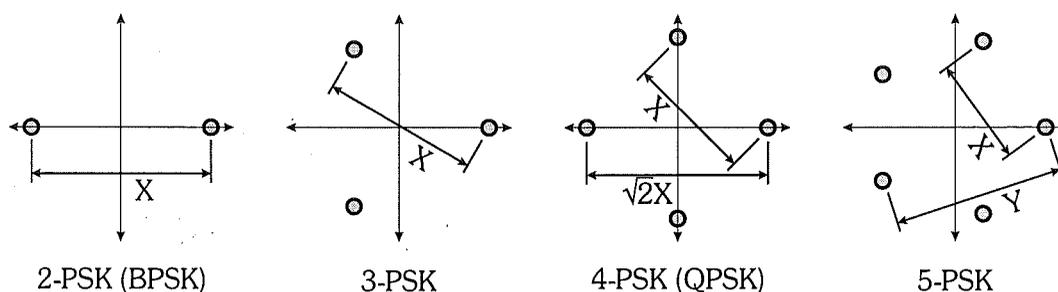


Figure 5.5: Symbol to symbol distances for various n -PSK constellations. In the 5-PSK case, $Y \approx 1.62X$

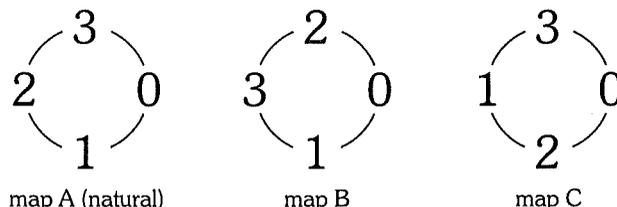


Figure 5.6: The different quaternary symbol to constellation point mappings.

5.3.1 Hamming distance searches

Table 5.8 presents the ‘best’ codes found when considering Hamming distance with the ‘natural’ symbol to constellation mapping over \mathbb{F}_4 . As with the previous searches, the d_i, N_i are the minimum Hamming parity weight, and associated multiplicity for inputs of

¹²4-PSK is often considered as two BPSK systems in quadrature. This is not done in this case as only quaternary information is considered; quaternary data, quaternary encoding, quaternary modulation, quaternary demodulation and quaternary decoding. At no point are dibits of binary information considered.

weight i . The notation $\times\langle\cdots|\cdots\rangle$ is used to represent a set of multiplicatively equivalent codes $\langle\alpha a_n \dots \alpha a_0|b_n \dots b_0\rangle \forall \alpha \in \mathbb{F}_q$ which have identical weight spectra.

codes	states	d_2, N_2	d_3, N_3	d_4, N_4	d_5, N_5	d_{free}
$\times\langle 12 11\rangle \times\langle 13 11\rangle$ $\times\langle 11 21\rangle^\dagger \times\langle 12 21\rangle$ $\times\langle 11 31\rangle \times\langle 13 31\rangle$	4	2,3 3,3	2,3 3,9	2,3 3,15	2,3 3,21	4
$\times\langle 111 121\rangle \times\langle 111 131\rangle$ $\times\langle 123 211\rangle^\dagger \times\langle 123 221\rangle$ $\times\langle 132 311\rangle \times\langle 132 331\rangle$	16	6,3 10,3	3,3 4,12	2,3 4,15	3,9 4,48	6
$\times\langle 1203 2111\rangle \times\langle 1033 2131\rangle$ $\times\langle 1023 2221\rangle \times\langle 1303 2231\rangle$ $\times\langle 1013 2311\rangle \times\langle 1103 2321\rangle$ $\times\langle 1302 3111\rangle \times\langle 1022 3121\rangle$ $\times\langle 1012 3211\rangle \times\langle 1102 3231\rangle$ $\times\langle 1202 3321\rangle \times\langle 1032 3331\rangle$	64	18,3 34,3	5,3 6,3	3,3 4,6	4,6 5,54	7

Table 5.8: The ‘best’ \mathbb{F}_4 rate 1/2 RSC codes found using the pairwise optimisation criterion and Hamming distances. The \dagger identifies codes presented by White and Costello [62].

The sets of 4 and 16 state codes identified include those presented by White and Costello [62], whose aim was to find good component codes for turbo codes that were to be transmitted using FSK modulation. Their 64 state code is not included as it is only a member of the set of ‘best’ codes if the search only considers input weights of four or less. The greater depth of this search results in a small ‘best’ set, which does not include White and Costello’s 64 state code.

Of interest is the fact that each of the identified codes represents a set of three multiplicatively equivalent codes. This is explained by the following lemma

Lemma:

Over any field \mathbb{F}_q , the set of codes $\langle\alpha a_n \dots \alpha a_0|b_n \dots b_0\rangle \forall \alpha \in \mathbb{F}_q$ have identical Hamming weight spectra.

Proof:

Over \mathbb{F}_q division is defined, and hence we can see from the code transfer function that if an input γ applied to the code $\langle a_n \dots a_0|b_n \dots b_0\rangle$ generates the output ϕ , then when the input γ is applied to the codes $\langle\alpha a_n \dots \alpha a_0|b_n \dots b_0\rangle$ the output will be $\frac{\phi}{\alpha}$. Clearly, as we are only considering Hamming weight, the outputs ϕ and $\frac{\phi}{\alpha}$ will be of identical weight. This is true for all inputs γ , and thus the codes $\times\langle a_n \dots a_0|b_n \dots b_0\rangle$ will have the same weight spectra. ■

5.3.2 Euclidean distance searches

The use of 4-PSK means that the errors between different symbols are not equally likely. Because of this, searches that use a Euclidean distance metric provides a set of ‘best’ component codes more suited to the modulation.

With reference to section 4.7, it is possible that the non-binary component codes searched are not geometrically uniform when mapped onto q -ary-PSK and considering Euclidean distance. If this is the case, then the searches presented in this and subsequent sections are not necessarily exhaustive because only error events with respect to the all-0 path are enumerated. However, the codes identified will still be good codes with respect to the search criteria presented in section 4.3.

Table 5.9 presents the results of Euclidean distance searches over \mathbb{F}_4 for the natural symbol to constellation mapping. In this case, the d_i are the lowest, second lowest, and third lowest Euclidean parity weights possible from inputs with Hamming weight i .

codes	states	d_2, N_2	d_3, N_3	d_4, N_4	d_5, N_5
$\times\langle 12 21\rangle$ $\times\langle 13 31\rangle$	4	2.000, 1 2.414, 2 3.000, 2	2.000, 1 2.414, 2 3.000, 2	2.000, 2 2.828, 1 3.000, 4	2.000, 1 2.414, 2 3.000, 10
$\times\langle 101 121\rangle$ $\times\langle 101 131\rangle$	16	6.828, 3 10.828, 1 11.657, 2	2.000, 2 2.828, 1 4.000, 3	2.000, 2 2.828, 1 4.000, 8	2.000, 2 2.828, 1 3.414, 6
$\times\langle 1203 2131\rangle$ $\times\langle 1302 3121\rangle$ $\times\langle 1023 2321\rangle$ $\times\langle 1032 3231\rangle$	64	20.485, 3 38.142, 1 38.971, 2	5.414, 2 6.000, 1 6.243, 1	3.414, 3 4.000, 3 4.828, 6	4.000, 7 4.828, 10 5.414, 28

Table 5.9: The ‘best’ \mathbb{F}_4 rate 1/2 RSC codes found using the pairwise optimisation criterion and Euclidean distances with natural mapping

Of interest is the fact that in the 16 and 64 state cases the codes chosen in table 5.9 are not a subset of those chosen in the Hamming distance search table 5.8. This demonstrates the importance of considering the modulation scheme when choosing component codes.

The results of Euclidean distance searches over each of the three alternative rings \mathbb{Z}_4 , $\mathbb{F}_2 + u\mathbb{F}_2$ and $\mathbb{F}_2 + w\mathbb{F}_2$ are presented in tables 5.10, 5.11 and 5.12. In all of these cases, the codes identified are poorer than those found using \mathbb{F}_4 . Because of the multiplicative equivalence of the ‘best’ codes identified, an identical set of ‘best’ codes was found when any of the different symbol to constellation mappings were used.

Simulation results for rate 1/3 turbo codes based on the best 4 and 16 state codes are presented in figure 5.7. Again, the frames contained 1000 data symbols, an S-random interleaver was used, and the parity sequences were terminated. Transmission was through an AWGN channel using 4-PSK modulation. 10 decoder iterations were performed, and

codes	states	d_2, N_2	d_3, N_3	d_4, N_4	d_5, N_5
$\pm\langle 12 11\rangle$ $\pm\langle 12 31\rangle$ $\pm\langle 21 11\rangle$ $\pm\langle 21 31\rangle$	4	1.414, 1 2.414, 2 2.828, 1	1.000, 2 2.000, 2 2.414, 2	2.000, 2 2.414, 2 3.000, 4	2.000, 4 3.000, 10 3.414, 10
$\pm\langle 123 311\rangle$ $\pm\langle 123 331\rangle$	16	5.657, 1 7.414, 2 8.485, 1	2.828, 1 3.414, 2 4.000, 6	2.828, 1 3.414, 6 4.000, 4	2.000, 2 2.828, 1 3.414, 4
$\pm\langle 1131 1231\rangle$ $\pm\langle 1333 3211\rangle$ $\pm\langle 1311 1321\rangle$ $\pm\langle 1113 3321\rangle$	64	8.485, 1 14.142, 1 15.657, 2	5.657, 3 6.000, 2 6.828, 2	2.828, 1 3.414, 2 4.000, 6	4.000, 4 4.828, 8 5.414, 22

Table 5.10: The ‘best’ \mathbb{Z}_4 rate 1/2 RSC codes found using the pairwise optimisation criterion and Euclidean distances with natural mapping

codes	states	d_2, N_2	d_3, N_3	d_4, N_4	d_5, N_5
$\langle 12 11\rangle \langle 32 11\rangle$ $\langle 12 31\rangle \langle 32 31\rangle$ $\langle 21 11\rangle \langle 23 11\rangle$ $\langle 21 31\rangle \langle 23 31\rangle$	4	1.414, 1 2.414, 2 2.828, 1	1.000, 2 2.000, 2 2.414, 2	2.000, 2 2.414, 2 3.000, 4	2.000, 4 3.000, 10 3.414, 10
$\langle 123 311\rangle \langle 321 311\rangle$ $\langle 123 331\rangle \langle 321 331\rangle$	16	5.657, 1 7.414, 2 8.485, 1	2.828, 1 3.414, 2 4.000, 4	2.828, 1 3.414, 6 4.000, 8	2.000, 2 2.828, 1 3.414, 4
$\langle 1131 1231\rangle \langle 3313 1231\rangle$ $\langle 1333 3211\rangle \langle 3111 3211\rangle$ $\langle 1311 1321\rangle \langle 3133 1321\rangle$ $\langle 1113 3321\rangle \langle 3331 3321\rangle$	64	8.485, 1 14.142, 1 15.657, 2	5.657, 3 6.000, 2 6.828, 2	2.828, 1 3.414, 2 4.000, 6	4.000, 4 4.828, 8 5.414, 22

Table 5.11: The ‘best’ $\mathbb{F}_2 + u\mathbb{F}_2$ rate 1/2 RSC codes found using the pairwise optimisation criterion and Euclidean distances with natural mapping

codes	states	d_2, N_2	d_3, N_3	d_4, N_4	d_5, N_5
$\langle 23 11\rangle$ $\langle 32 11\rangle$	4	1.000, 1 1.414, 1 2.000, 1	1.000, 1 2.000, 2 2.414, 3	2.000, 2 2.414, 1 3.000, 5	2.000, 1 3.000, 5 3.414, 6
$\langle 121 131\rangle$	16	4.000, 1 4.243, 1 5.657, 1	2.000, 1 3.414, 1 4.000, 6	2.000, 1 2.828, 1 3.414, 3	2.000, 1 3.414, 4 4.000, 18
$\langle 1131 1321\rangle$ $\langle 1311 1231\rangle$	64	6.000, 1 7.414, 1 8.485, 1	4.000, 3 4.243, 2 6.000, 2	2.000, 1 4.000, 6 4.828, 1	4.000, 10 4.828, 4 5.414, 24

Table 5.12: The ‘best’ $\mathbb{F}_2 + v\mathbb{F}_2$ rate 1/2 RSC codes found using the pairwise optimisation criterion and Euclidean distances with natural mapping

the simulations were stopped when 50 frames containing errors had been received.

Because binary information can be easily represented using quaternary symbols (two bits per symbol), accurate bit error rate curves are plotted in addition to the symbol error rates. For each simulation point all possible dibit to symbol (constellation) mappings at each SNR were tried. For all but a couple of high SNR points, the standard Gray mapping gave the best performance.

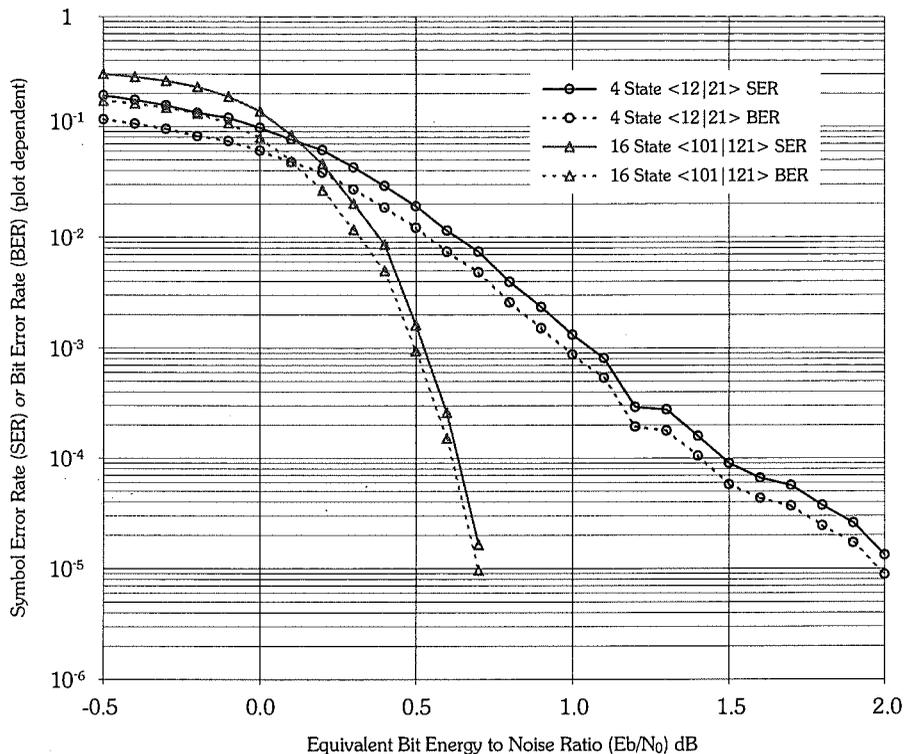


Figure 5.7: Simulation results for quaternary turbo codes.

5.4 Penternary Codes

Five is a prime number and hence $\mathbb{Z}_5 \cong \mathbb{F}_5$, so that we need only consider ring arithmetic. Table 5.13 presents the results of Euclidean distance searches over \mathbb{Z}_5 for the natural symbol to constellation mapping.

Simulation results for rate 1/3 turbo codes based on the best 5 and 25 state codes are presented in figure 5.8. Again the frames contained 1000 data symbols, the parity sequences were terminated and transmission was through an AWGN channel using 5-PSK. Ten decoder iterations were performed, and with the exception of one point, the simulation was stopped when 50 frames containing errors had been received. For the final point on the 25-state curve only 13 frames had been received in error when the simulation stopped.

codes	states	d_2, N_2	d_3, N_3	d_4, N_4	d_5, N_5
$\times\langle 12 11\rangle$	5	2.618, 4	2.000, 2	2.618, 4	2.000, 2
$\times\langle 13 11\rangle$		3.618, 2	3.000, 4	3.000, 2	3.000, 8
$\times\langle 12 41\rangle$		4.236, 2	3.236, 2	3.618, 12	3.236, 2
$\times\langle 13 41\rangle$					
$\times\langle 113 211\rangle$	25	8.236, 2	3.618, 6	4.000, 4	2.000, 2
$\times\langle 143 241\rangle$		10.090, 2	4.236, 6	4.618, 10	3.236, 2
$\times\langle 132 321\rangle$		15.090, 2	5.618, 2	5.236, 24	4.000, 2
$\times\langle 122 331\rangle$					

Table 5.13: The ‘best’ \mathbb{Z}_5 rate 1/2 RSC codes found using the pairwise optimisation criterion and Euclidean distances with natural mapping

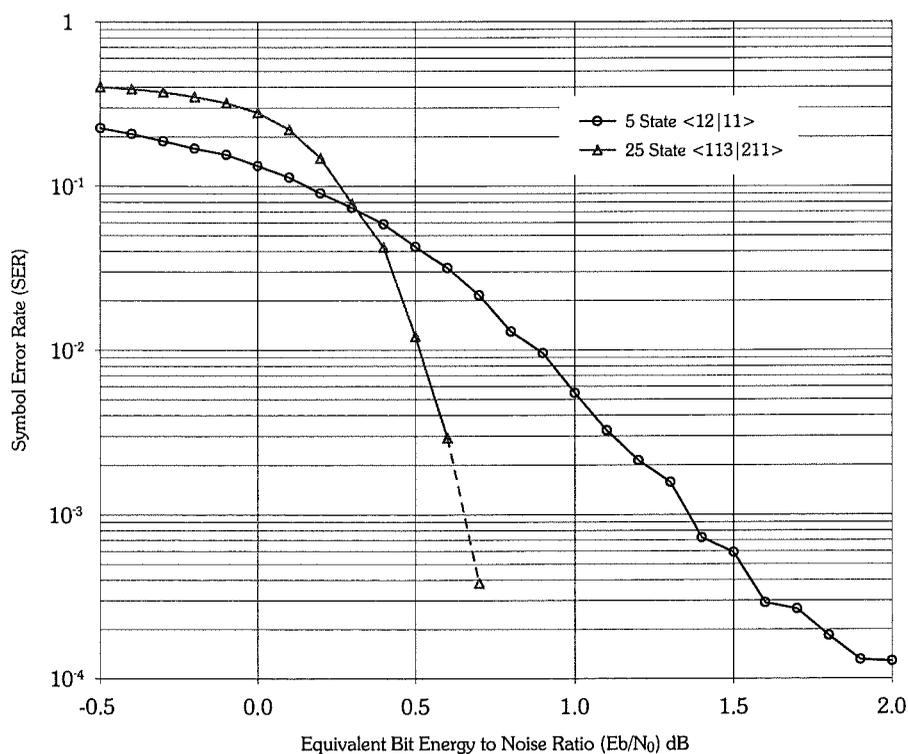


Figure 5.8: Simulation results for pentenary turbo codes.

5.5 Hexernary and Octernary Codes

The last sets of codes presented are for the rings \mathbb{Z}_6 and \mathbb{F}_8 . Table 5.14 lists the ‘best’ 6 and 36 state codes found over \mathbb{Z}_6 when considering Euclidean distance and the natural symbol to constellation mapping for 6-PSK. Tables 5.15 and 5.16 list the results of Hamming and Euclidian distance searches over \mathbb{F}_8 . The Hamming distance results include the codes presented by White and Costello, and we note that as expected they achieve the bound calculated in section 4.6.

codes	states	d_2, N_2	d_3, N_3	d_4, N_4	d_5, N_5
$\pm\langle 21 11 \rangle$	6	2.000, 1	2.000, 2	2.000, 2	2.000, 2
$\pm\langle 25 51 \rangle$		2.732, 2	2.732, 2	2.732, 2	2.732, 2
$\pm\langle 12 11 \rangle$		3.464, 2	3.000, 2	3.000, 2	3.000, 4
$\pm\langle 14 51 \rangle$					
$\pm\langle 145 511 \rangle$	36	8.000, 1	3.732, 2	3.732, 4	2.000, 2
$\pm\langle 125 551 \rangle$		8.660, 2	4.000, 1	4.000, 5	3.464, 2
		12.000, 1	5.000, 4	5.000, 8	4.000, 9

Table 5.14: The ‘best’ \mathbb{Z}_6 rate 1/2 RSC codes found using the pairwise optimisation criterion and Euclidean distances with natural mapping.

codes	states	d_2, N_2	d_3, N_3	d_4, N_4	d_5, N_5	d_{free}
$\times\langle 12 11 \rangle \times\langle 13 11 \rangle \times\langle 14 11 \rangle \times\langle 15 11 \rangle$ $\times\langle 16 11 \rangle \times\langle 17 11 \rangle \times\langle 11 21 \rangle^\dagger \times\langle 12 21 \rangle$ $\times\langle 13 21 \rangle \times\langle 14 21 \rangle \times\langle 15 21 \rangle \times\langle 17 21 \rangle$ $\times\langle 11 31 \rangle \times\langle 12 31 \rangle \times\langle 13 31 \rangle \times\langle 15 31 \rangle$ $\times\langle 16 31 \rangle \times\langle 17 31 \rangle \times\langle 11 41 \rangle \times\langle 12 41 \rangle$ $\times\langle 14 41 \rangle \times\langle 15 41 \rangle \times\langle 16 41 \rangle \times\langle 17 41 \rangle$ $\times\langle 11 51 \rangle \times\langle 12 51 \rangle \times\langle 13 51 \rangle \times\langle 14 51 \rangle$ $\times\langle 15 51 \rangle \times\langle 16 51 \rangle \times\langle 11 61 \rangle \times\langle 13 61 \rangle$ $\times\langle 14 61 \rangle \times\langle 15 61 \rangle \times\langle 16 61 \rangle \times\langle 17 61 \rangle$ $\times\langle 11 71 \rangle \times\langle 12 71 \rangle \times\langle 13 71 \rangle \times\langle 14 71 \rangle$ $\times\langle 16 71 \rangle \times\langle 17 71 \rangle$	8	2, 7 3, 7 4, 7	2, 7 3, 49 4, 91	2, 7 3, 91 4, 427	2, 7 3, 133 4, 1015	4
$\times\langle 171 131 \rangle \times\langle 121 151 \rangle \times\langle 141 161 \rangle$ $\times\langle 176 211 \rangle \times\langle 116 241 \rangle \times\langle 136 251 \rangle$ $\times\langle 154 331 \rangle \times\langle 144 341 \rangle \times\langle 134 371 \rangle$ $\times\langle 123 411 \rangle \times\langle 153 461 \rangle \times\langle 113 471 \rangle$ $\times\langle 157 521 \rangle \times\langle 167 551 \rangle \times\langle 177 571 \rangle$ $\times\langle 122 621 \rangle \times\langle 162 641 \rangle \times\langle 132 661 \rangle^\dagger$ $\times\langle 145 711 \rangle \times\langle 115 721 \rangle \times\langle 165 731 \rangle$	64	10, 7 18, 7 26, 7	3, 7 4, 14 5, 21	3, 14 4, 70 5, 196	3, 35 4, 126 5, 770	6

Table 5.15: The ‘best’ \mathbb{F}_8 rate 1/2 RSC codes found using the pairwise optimisation criterion and Hamming distances with natural mapping. The \dagger identifies codes presented by White and Costello [62].

codes	states	d_2, N_2	d_3, N_3	d_4, N_4	d_5, N_5
$\times\langle 12 51\rangle$	8	2.848, 2	2.848, 1	2.000, 1	2.000, 1
$\times\langle 16 71\rangle$		3.414, 2	3.000, 1	2.848, 1	2.848, 1
		4.262, 1	3.414, 1	3.000, 5	3.000, 1
$\times\langle 112 621\rangle$	64	17.551, 1	3.848, 1	3.848, 1	3.848, 2
$\times\langle 166 241\rangle$		18.399, 2	4.414, 1	4.613, 1	4.000, 2
		18.598, 2	4.696, 1	4.696, 1	4.414, 1

Table 5.16: The ‘best’ \mathbb{F}_q rate 1/2 RSC codes found using the pairwise optimisation criterion and Euclidean distances with natural mapping.

5.6 Observations

Comparison of these codes with binary codes is difficult. How do symbol error rate and bit error rate correspond? Is it fair to compare a simulation using frames of 1000 binary symbols with a simulation using frames of 1000 penternary symbols? The second frame certainly contains a lot more information.

Figures 5.9 and 5.10 attempt to compare codes over different fields. Each frame contains the same volume of binary data, but has a different symbol and hence interleaver length. The binary frames simulated were 2322 bits long using BPSK modulation, while the ternary, quaternary and penternary frames are 1465, 1161 and 1000 symbols long respectively using 3-PSK, QPSK and 5-PSK modulation. As with the previous simulations each code was simulated until a minimum of 50 frames had been received in error. Figure 5.9 presents bit error rates assuming that each symbol error results in one bit error (this is equivalent to presenting symbol error rate), while figure 5.10 assumes that each symbol error results in $\log_2 q$ bit errors. Because the codes over the smaller fields use longer frames, they will benefit more from interleaver gain, and we can assume that this comparison is biased toward the binary codes in particular. Despite this, it is clear that the 9-state ternary code provides exceptional performance.

5.7 Conclusion

Good rate 1/2 component codes for used in non-binary codes have been identified based on the response of these codes to low weight inputs. Additionally an achievability bound for z_{\min} the dominant parameter determining the suitability of an RSC code for use in turbo codes has been determined for all \mathbb{F}_q . Simulation results show that in particular, a turbo code based on 9-state ternary component codes performs very well.

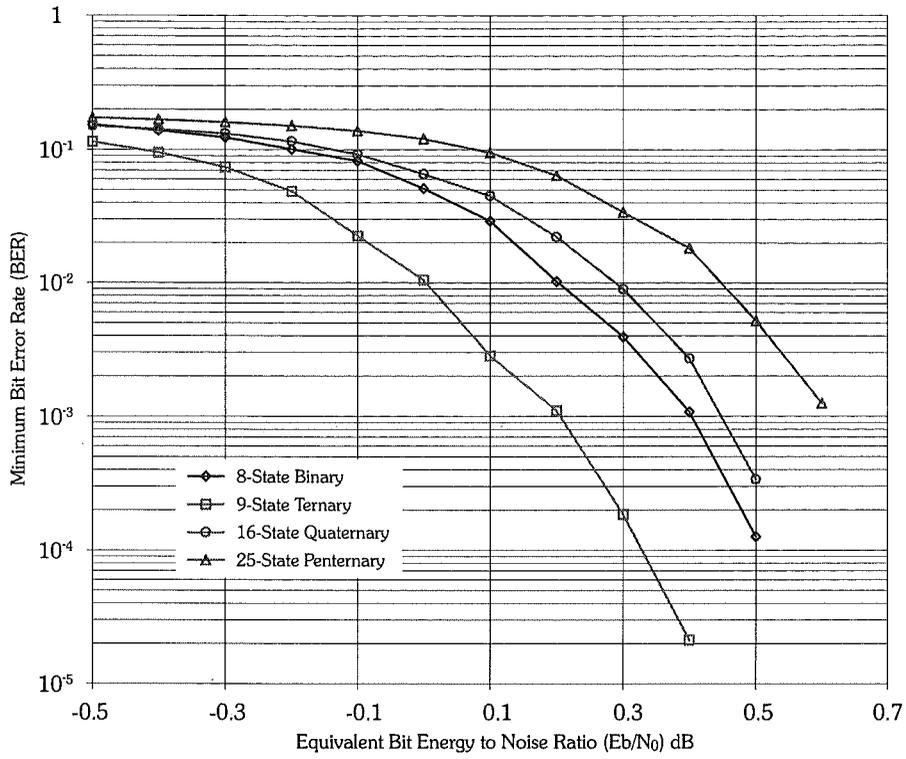


Figure 5.9: Comparison of codes over different fields assuming one bit error per symbol error.

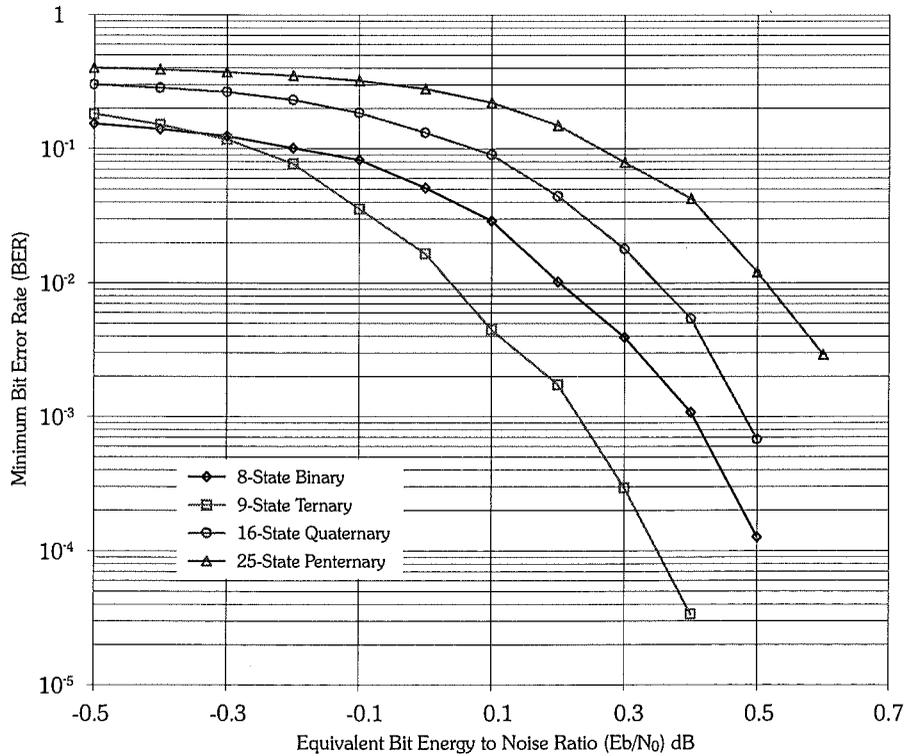


Figure 5.10: Comparison of codes over different fields assuming $\log_2(q)$ bit errors per symbol error.

Chapter 6

Conclusion

6.1 Results

This thesis has involved two largely disjoint pieces of work, although both deal with various aspects of turbo codes.

The work on decoding algorithm convergence demonstrates clearly the dynamic nature of the iterative decoder. Oscillatory behaviour will be present in real turbo-coded systems and is not just an artifact of the interleaver used. In contrast to the theoretical works of McEliece *et. al.* [45] and Richardson [51] or even the recent works of ten Brink [17] and Agrawal *et. al.* [1], the convergence plots presented show the kinds of observable behaviour that can be expected from any turbo code implementation.

It was shown how these convergence results could be used to inform a stopping criterion for the decoder. When hybridised with the HDA method of Shao *et. al.* [54] this provided a criterion that examined both the strength and stability of the decoder's decision and provided very good results. In conjunction with an ARQ scheme reliable communication was possible at very low SNRs without the need for any additional error detection code.

The work on non-binary component codes for turbo codes started by examining the search metric of Benedetto *et. al.* [9] in the binary context. It was demonstrated that this metric was valid by checking its application to very short frame lengths, and by examining competing codes. This metric, and the extension of the achievability bound for z_{\min} to any field provided the basis for a search for good non-binary component codes.

These searches and the accompanying simulations demonstrate the potential of non-binary turbo codes. In particular, the 9-state ternary code was shown to provide exceptional performance given its low complexity.

6.2 Suggestions for further work

Eight years since their introduction, turbo codes and related structures are still one of the best way of approaching capacity for code rates less than $1/2$. Because of this they command much research interest, and there are still many open questions.

Research suggestions that arise directly from this thesis include

- Using convergence plots to examine other parameters in the iterative decoding process. In particular it would be interesting to observe both the *a priori* probability ratios, and the extrinsic information for frames with undesirable convergence behaviours. Can the convergence behaviour be related back to unusual behaviour by either of these parameters?
- A re-examination of the observed phenomena in the light of [17] and [1] may be useful.
- It is mentioned in [6] the the binary achievability bound on z_{\min} is also an upper bound. Can this result be extended to other fields?
- Why does the 9-state ternary code perform so well? Can a capacity bound be derived for 3-PSK modulation to provide an objective measurement of its performance? Also, why do the 8-state binary, and 9-state ternary codes outperform the more complex 16-state and 27-state codes?
- Clearly attempts can be made to extend any binary concatenated code research to non-binary codes as an extension of this thesis. This could include, for example, serial concatenation of non-binary component codes, the performance of non-binary codes in Rayleigh fading channels, or the development of decoders with lower complexity.
- Can the performance of the quaternary codes be improved by bit-interleaving as opposed to symbol-interleaving?
- Are the Euclidean distance code geometrically uniform or not?

Appendix A

Simulating the channel

Because we can simulate the system at baseband, and because our noise samples are independent, the channel is trivially simulated by adding samples n_j of random variables N_j to both the inphase I and quadrature Q components of each transmitted constellation point.

Statistics of inphase and quadrature noise

First it is necessary to determine the statistics of the N_j s given that our noise function $w(t)$ is a sample of the white Gaussian process $W(t)$. This can be done by examining the statistics associated with the correlator outputs. (refer to sections 2.6.2 and 2.6.3)

Let $X(t)$ denote the random process of which $x(t)$ is a sample, and let X_j denote the random variable of which x_j is a sample. Now because we are using an AWGN channel, $X(t)$ is a Gaussian process. It follows that the x_j s must be Gaussian random variables characterised completely by their mean and variance. These can be determined to be [36]

$$\begin{aligned}\mu_{X_j} &= s_{ij} \\ \sigma_{X_j}^2 &= \frac{N_0}{2}, \quad \forall j\end{aligned}\tag{A.1}$$

So each correlator output has a noise variance $N_0/2$ equal to the power spectral density of the noise process $W(t)$.

Consequently the random variables N_j are also Gaussian and are defined by their statistics:

$$\begin{aligned}\mu_{N_j} &= 0 \\ \sigma_{N_j}^2 &= \frac{N_0}{2}, \quad j = I, Q\end{aligned}\tag{A.2}$$

Generating noise with the required variance

If `rannum()` is a *C++* function that generates samples of a Gaussian random variable A with mean 0 and variance 1, the next problem is to determine how to transform the output of this function to give us samples of a Gaussian random variable B with mean 0 and variance σ^2 .

This can be achieved by a linear transformation $g(\cdot)$ of the random variable A

$$B = g(A), \quad b = g(a) = \eta a, \quad \eta = \text{a constant} \quad (\text{A.3})$$

Now if $f_A(a)$ and $f_B(b)$ are the probability density functions of A and B respectively then we find

$$f_B(b) = \frac{f_A(a)}{|dg/da|} \Big|_{a=g^{-1}(b)} \quad (\text{A.4})$$

so using the standard expression of the Gaussian probability density function

$$\begin{aligned} f_B(b) &= \frac{\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{a^2}{2}\right)}{\eta} \Big|_{a=b/\eta} \\ &= \frac{1}{\eta\sqrt{2\pi}} \exp\left(-\frac{b^2}{2\eta^2}\right) \end{aligned} \quad (\text{A.5})$$

giving us a Gaussian distribution with mean 0 and variance η^2 . Thus the required transformation function $g(\cdot)$ is

$$g(a) = \sigma a \quad (\text{A.6})$$

where σ is the square root of the desired variance.

This means that the following code correctly simulates the effect of the channel for each frame `x` of constellation points that is transmitted:

```
for( long i = 0; i < x.length; i++ ) {  
    x[i].I +=  $\sigma$  * rannum();  
    x[i].Q +=  $\sigma$  * rannum();  
}
```

Determining variance from signal to noise ratio

The last remaining problem is to determine the variance σ^2 from a specified *signal to noise ratio* (SNR) which in this thesis will refer to the bit energy to noise ratio E_b/N_0 .

Given that the overall code is rate 1/3 and the modulation scheme used is q -PSK, the relationship between symbol energy E_s and bit energy E_b is

$$E_s = \frac{1}{3} \times \log_2(M) \times E_b \quad (\text{A.7})$$

where the first term accounts for the code rate, and the second term is the number of information bits represented by each transmitted symbol.

With M -PSK, all constellation points lie on the unit circle giving $E_s = 1$, and recall that by definition $\sigma^2 = N_0/2$. Then

$$\begin{aligned}\frac{1}{2\sigma^2} &= \frac{E_s}{N_0} \\ &= \frac{1}{3} \times \log_2(M) \times \frac{E_b}{N_0}\end{aligned}\tag{A.8}$$

and hence

$$\sigma^2 = \frac{3}{2} \times \frac{1}{\log_2(M)} \times \frac{N_0}{E_b}\tag{A.9}$$

Finally if SNR is specified in decibels then

$$\sigma^2 = \frac{3}{2} \times \frac{1}{\log_2(M)} \times \frac{1}{10^{(\frac{\text{SNR}_{\text{dB}}}{10})}}\tag{A.10}$$

Bibliography

- [1] D. Agrawal and A. Vardy, "The turbo decoding algorithm and its phase trajectories," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 699–722, 2001.
- [2] K. S. Andrews, C. Heegard and D. Kozen, "Interleaver design methods for turbo-codes," *Proc. IEEE Int. Symp. Inform. Theory*, Cambridge MA, USA, p. 420, 1998.
- [3] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. 20, pp. 284–287, 1974.
- [4] R. Baldini F. and P. G. Farrell, "Coded modulation based on rings of integers modulo- q Part 2:Convolutional codes," *IEE Proc.-Commun.*, vol. 141, no. 3, 1994.
- [5] S. Benedetto and G. Montorsi, "Design of parallel concatenated convolutional codes," *IEEE Trans. Commun.*, vol. 44, no. 5, pp. 591–600, 1996.
- [6] S. Benedetto and G. Montorsi, "Unveiling turbo codes - some results on parallel concatenated coding schemes," *IEEE Trans. Inform. Theory*, vol. 42, no. 2, pp. 409–428, 1996.
- [7] S. Benedetto, D. Divsalar, G. Montorsi and F. Pollara, "Serial concatenation of interleaved codes: performance analysis, design and iterative decoding," *JPL TDA Progress Report.*, vol. 42-126, pp. 1–25, 1996.
- [8] S. Benedetto, D. Divsalar, G. Montorsi and F. Pollara, "Soft-output decoding algorithms in iterative decoding of turbo codes," *JPL TDA Progress Report.*, vol. 42-124, pp. 63–87, 1996.
- [9] S. Benedetto, R. Garello and G. Montorsi, "A search for good convolutional codes to be used in the construction of turbo codes," *IEEE Trans. Commun.*, vol. 46, no. 9, pp. 1101–1105, 1998.
- [10] S. A. Barbluescu and S. S. Pietrobon, "Terminating the trellis of turbo-codes in the same state," *Electron. Lett.*, vol. 31, no. 1, pp. 22–23, 1995.

- [11] S. A. Barbluescu, "Dynamical system perspective on turbo codes," *Electron. Lett.*, vol. 34, no. 8, pp. 754–755, 1998.
- [12] J. Berkmann, "On turbo decoding of nonbinary codes," *IEEE Commun. Lett.*, vol. 2, no. 4, pp. 94–96, 1998.
- [13] E. R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.
- [14] C. Berrou, A. Glavieux and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: turbo-codes," *Proc. IEEE ICC*, Geneva, Switzerland, pp. 1064-1070, 1993.
- [15] W. J. Blackert, E. K. Hall and S. G. Wilson, "Turbo code termination and interleaver design," *Electron. Lett.*, vol. 31, no. 24, pp. 2082–2084, 1995.
- [16] I. F. Blake, "Codes over certain rings," *Information and Control*, vol. 20, pp. 396–404, 1972.
- [17] S. ten Brink, "Convergence behavior of iteratively decoded parallel concatenated codes," *IEEE Trans. Commun.*, vol. 49, no. 10, pp. 1727–1737, 2001.
- [18] R. de Buda, "Some optimal codes have structure," *IEEE J. Sel. Areas in Commun.*, vol. 7, no. 6, pp. 893–899, 1989.
- [19] S.-Y. Chung, G. D. Forney Jr., T. Richardson and R. Urbanke, "On the design of low-density parity check codes within 0.0045dB of the Shannon limit," *IEEE Commun. Lett.*, vol. 5, no. 2, pp. 58–60, 2001
- [20] D. Costello, Jr., J. Hagenauer, H. Imai and S. Wicker, "Application of error-control coding," *IEEE Trans. Inform. Theory*, vol. 44, no. 6, pp. 2531–2560, 1998.
- [21] D. Divsalar, S. Dolinar and F. Pollara, "Transfer function bounds on the performance of turbo codes," *JPL TDA Progress Report.*, vol. 42-122, pp. 44–54, 1995.
- [22] D. Divsalar and F. Pollara, "Turbo codes for deep-space communications," *JPL TDA Progress Report.*, vol. 42-120, pp. 29–39, 1995.
- [23] D. Dolinar and D. Divsalar, "Weight distributions for turbo codes using random and nonrandom permutations," *JPL TDA Progress Report.*, vol. 42-122, pp. 56–64, 1995.
- [24] S. Dolinar, D. Divsalar and F. Pollara, "Code performance as a function of block size," *JPL TDA Progress Report.*, vol. 42-133, pp. 1–23, 1998.
- [25] T. M. Duman and M. Salehi, "New performance bounds for turbo codes," *IEEE Trans. Commun.*, vol. 46, no. 6, pp. 717–723, 1998.

- [26] P. Elias, "Coding for noisy channels," *IRE Convention Record*, Part 4, pp. 37–46, March 1955.
- [27] G. D. Forney Jr., "Concatenated codes," *Sc.D. Thesis, Massachusetts Institute of Technology*, 1965.
- [28] G. D. Forney Jr., "The Viterbi algorithm," *Proc. IEEE*, vol. 61, pp. 268–278, 1973.
- [29] G. D. Forney Jr., "Geometrically Uniform Codes," *IEEE Trans. Inform. Theory*, vol. 37, no. 5, pp. 1241–1260, 1991.
- [30] R. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, pp. 21–28, Jan. 1962.
- [31] R. Garello, G. Montorsi, S. Benedetto, D. Divsalar and F. Pollara, "Labelings and encoders with the uniform bit error property with applications to serially concatenated trellis codes," *IEEE Trans. Inform. Theory*, vol. 48, no. 1, pp. 123–136, 2002.
- [32] J. Hagenauer, E. Offer and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inform. Theory*, vol. 42, no. 2, pp. 429–445, 1996.
- [33] E. Hall and S. Wilson, "Design and analysis of turbo codes on Rayleigh fading channels," *IEEE J. Selec. Areas Commun.*, vol. 16, no. 2, pp. 160–174, 1998.
- [34] R. W. Hamming, "Error detecting and error correcting codes," *Bell Systems Tech. Journal*, vol. 29, pp. 147–160, 1950.
- [35] L. J. Harcke and G. E. Wood, "Laboratory and flight performance of the Mars Pathfinder (15,1/6) convolutionally encoded telemetry link," *JPL TDA Progress Report.*, vol. 42-129, pp. 1–11, 1997.
- [36] S. Haykin, *Communications systems 3rd. ed.* John Wiley & Sons, New York, 1994.
- [37] C. Heegard and S. B. Wicker, *Turbo Coding*, Kluwer Academic Publishers, Boston, 1999.
- [38] E. S. Herman and N. Chomsky, *Manufacturing Consent: the political economy of the mass media*, Pantheon Books, New York, 1988.
- [39] M. S. Ho, S. S. Pietrobon and T. Giles, "Improving the constituent codes of turbo encoders," *Proc. IEEE GLOBECOM*, Sydney, Australia, vol. 6, pp. 3525–3529, 1998.
- [40] M. S. Ho, S. S. Pietrobon and T. Giles, "Optimising the constituent code for turbo encoders," *Proc. IEEE ISIT*, Cambridge, MA, USA, pp. 178, 1998.
- [41] R. W. Kerr, "Convolutional Ring Codes for Fading Channels," *Ph.D. Thesis, University of Victoria*, 1996.

- [42] Y. Kou, S. Lin and M. Fossorier, “Low-density parity-check codes based on finite geometries: a rediscovery and new results,” *IEEE Trans. Inform. Theory*, vol. 47, no. 7, pp. 2711–2736, 2001.
- [43] S. Lin, D. J. Costello Jr. and M. J. Miller, “Automatic-repeat-request error-control schemes” *IEEE Commun. Mag.*, vol. 22, No. 12, 1984.
- [44] J. L. Massey and T. Mittelholzer, “Convolutional codes over rings,” *Proc. Fourth Joint Swedish-USSR Int. Workshop on Inform. Theory*, Gotland, Sweden, August 1989.
- [45] R. J. McEliece, E. R. Rodemich and J.-F. Cheng, “The turbo decision algorithm,” *Proc. 33rd Allerton Conf on Communication, Control and Computing*, Monticello IL, USA, Aug. 1995.
- [46] K. R. Narayanan and G. L. Stüber, “A novel ARQ technique using the turbo coding principle” *IEEE Commun. Lett.*, vol. 1, no. 2, pp. 49–51, 1997
- [47] L. C. Perez, J. Seghers and D. J. Costello Jr., “A distance spectrum interpretation of turbo-codes,” *IEEE Trans. Inform. Theory*, vol. 42, no. 6, pp. 1698–1709, 1996.
- [48] N. Postman, *Technopoly: the surrender of culture to technology*, Vintage Books, New York, 1993.
- [49] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery, *Numerical recipes in C : the art of scientific computing*, Cambridge University Press, Cambridge, 1992.
- [50] A. C. Reid, T. A. Gulliver and D. P. Taylor, “Convergence and errors in turbo decoding” *IEEE Trans. Commun.*, vol. 49, no. 12, pp. 2045–2051, 2001
- [51] T. Richardson, “The geometry of turbo-decoding dynamics,” *IEEE Trans. Inform. Theory*, vol. 46, no. 1, pp. 9–23, 2000.
- [52] C. E. Shannon, “A mathematical theory of communication,” *Bell System Tech. Journal*, vol. 27, pp. 379–423 and 623–656, 1948.
- [53] C. E. Shannon, “Communication in the presence of noise,” *Proc. IRE*, vol. 37, pp. 10–21, 1949.
- [54] R. Shao, S. Lin and M. Fossorier, “Two simple stopping criteria for turbo decoding” *IEEE Trans. Commun.*, vol. 47, no. 8, pp. 1117–1120, 1999
- [55] A. Shitbutani, H. Suda and F. Adachi, “Reducing average number of turbo decoding iterations” *Electron. Lett.*, vol. 35, no. 9, pp. 701–702, 1999
- [56] E. Spiegel, “Codes over Z_m ,” *Information and Control*, vol. 35, pp. 48–51, 1977.

- [57] O. Takeshita, O. Colling, P. Massey and D. Costello Jr., "A note on asymmetric turbo-codes" *IEEE Commun. Lett.*, vol. 3, no. 3, pp. 69–71, 1999
- [58] O. Takeshita, M. Fossorier and D. Costello Jr., "A new technique for computing the weight spectrum of turbo-codes" *IEEE Commun. Lett.*, vol. 3, no. 8, pp. 251–253, 1999
- [59] G. Ungerboeck, "Channel coding with multilevel/phase signals," *IEEE Trans. Inform. Theory*, vol. 28, pp. 55–67, 1982.
- [60] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory*, vol. 13, pp. 260–269, 1967.
- [61] A. Viterbi and J. Omura, *Principles of Digital Communication and Coding*, McGraw-Hill, New York, 1979.
- [62] G. S. White and D. J. Costello, Jr, "Construction and performance of q -ary turbo codes for use with M -ary modulation techniques," *CISS*, 1999.
- [63] J. Wozencraft and I. Jacobs, *Principles of Communication Engineering*, John Wiley & Sons Inc., New York, 1965.
- [64] C. Xu, M. Fossorier, S. Lin, "A search algorithm for self-concatenated turbo TCM codes," *Proc. CISS 2000*, Princeton University, USA, Mar. 2000.