Adaptive Iterative Decoding: Block Turbo Codes and Multilevel Codes

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

Philippa Anne Martin

B.E. (Hons 1)

February 2001

Abstract

New adaptive, iterative approaches to the decoding of block Turbo codes and multilevel codes are developed. Block Turbo codes are considered as they can readily provide high data rates, low decoding complexity and good performance. Multilevel codes are considered as they provide a moderate complexity approach to a high complexity code and can provide codes with good bandwidth efficiency.

The work develops two adaptive sub-optimal soft output decoding algorithms for block Turbo codes. One is based on approximation and the other on the distance properties of the component codes. They can be used with different codes, modulation schemes, channel conditions and in different applications without modification. Both approaches provide improved performance compared to previous approaches on the *additive white Gaussian noise (AWGN)* channel. The approximation based adaptive algorithm is also investigated on the uncorrelated Rayleigh flat fading channel and is shown to improve performance over previous approaches.

Multilevel codes are typically decoded using a *multistage decoder (MSD)* for complexity reasons. Each level passes hard decisions to subsequent levels. If the approximation based adaptive algorithm is used to decode component codes in a traditional MSD it improves performance significantly. Performance can be improved further by passing reliability (extrinsic) information to all previous and subsequent levels using an iterative MSD. A new iterative multistage decoding algorithm for multilevel codes is developed by treating the extrinsic information as a Gaussian random variable. If the adaptive algorithms are used in conjunction with iterative multistage decoding on the AWGN channel, then a significant improvement in performance is obtained compared to results using a traditional MSD.

Acknowledgments

I would like to thank all the people who have helped and supported me throughout my PhD. I would especially like to acknowledge and thank my supervisor Professor Desmond Taylor for his patience, guidance, endless proof reading and support. I am also grateful to Des for providing funding for me to attend the 2^{nd} International Symposium on Turbo Codes and Related Topics in France and Globecom 2000 in the United States of America.

I wish to acknowledge the public good science fund and Marsden fund of New Zealand for their financial support. Also Consultel and the University of Canterbury for their scholarships during my undergraduate and postgraduate studies.

I would like to thank Aaron Gulliver, Robert van Nobelen and Brian Hart for being willing to answer technical questions. I wish to acknowledge and thank Annie Picart and Ramesh Pyndiah for helping me reproduce their results. I am also grateful to Ben Skelton and Anthony Griffin for the LaTeX style files they gave me. I would also like to thank Mike Shurety, Dave van Leeuwen, Florin Predan and Pieter Kikstra for all their help over the years.

My utmost thanks to all my friends who helped keep spirits up. Thanks Bronwyn and John Ward, Rachel Johnston, Darryl Anderson, Cressida Harding, Jon Cherrie, Gerard van Soest, Elisabeth Nock, Torrie Moore and Jonathon Kong. Thanks also to the "comms lab" residents who answered research and latex questions, tried to educate me about rugby and helped keep life cheerful (Monkee's friday's will be a fond memory). Thanks Katharine Holdsworth, Wing Seng Leon, Anthony Griffin, Peter Green, Ben Jones, Ben Skelton, Andrew Reid, Aarne Mammela, Jukka Mannerkoski, Richard Clarke, Mary Nash, Karen Barton and Casey Miller.

I would also like to thank my family for their support, understanding and

encouragement. I would especially like to thank my parents for bringing me up to believe that I could achieve anything that I wanted to.

Most of all I would like to thank David Rankin for his emotional support, ability to make me laugh, and willingness to answer my questions or be a sounding board at all hours. Thank you for helping me through the bad times and celebrating the good times with me.

PHILIPPA ANNE MARTIN

The University of Canterbury
February 2001

Contents

Chapter 1

Introduction

1.1 Introduction

The number and expectations of wireless users and the demand for limited resources such as bandwidth are increasing rapidly. In order to allow voice, real-time video, internet and banking services to be provided on a mobile phone a variety of difficult engineering problems need to be solved [97].

In order to fully utilize the limited channel resources available and approach the capacity [100] of the channel the components of a communication system must be optimized. The main components in a communication system are source and error correction coding/ decoding, modulation/ demodulation, pulse shaping/ matched filtering and equalization as shown in Fig. 1.1. The purpose of each component will be discussed in section 1.2.

Designing a communication system is all about tradeoffs. For example, if the error correction coding/ decoding provides a 3dB performance gain, the gain can be used to double the data throughput, halve the broadcast power, reduce the antenna diameter by 30% or increase the transmission distance by 40% [2]. The most common tradeoffs in transmitter and receiver design are among bandwidth, power, complexity, cost, delay, error performance and data rate.

The goal of this thesis is to develop decoding algorithms with manageable complexity for error correction codes with moderate block length, high code rate and good bandwidth efficiency in order to achieve low *bit error rates (BERs)* close to

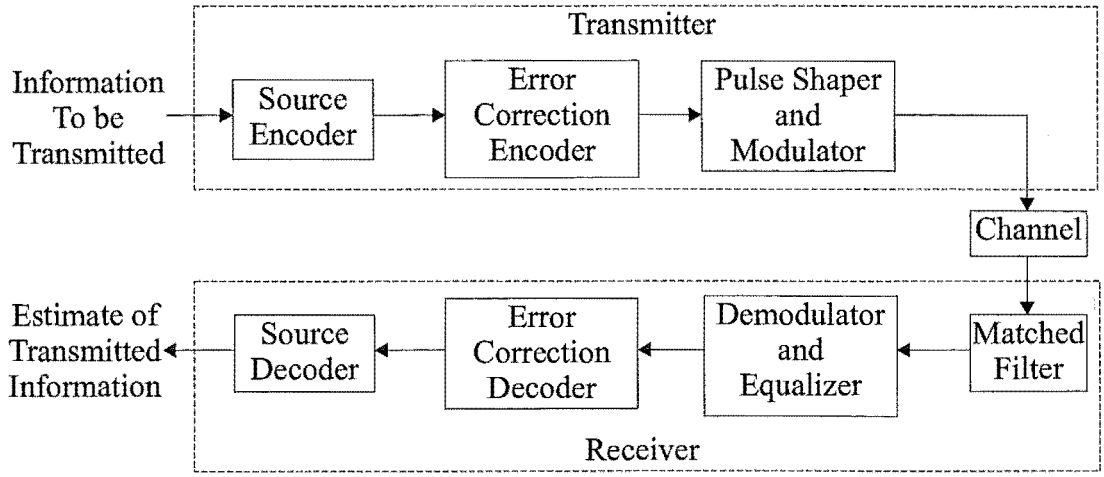


Figure 1.1: A communication system.

capacity[1]. The codes and decoding algorithms will be designed for the *additive white Gaussian noise (AWGN)* channel and the uncorrelated Rayleigh flat fading channel. Multilevel coding will be used to create powerful codes in a moderate complexity, bandwidth efficient manner. An iterative *multistage decoder (MSD)* is developed in this thesis to decode multilevel codes. It is used to provide a good compromise between complexity and performance. In addition two adaptive sub-optimal *soft-input soft-output (SISO)* decoding algorithms based on [90] are developed. These algorithms make the complexity of SISO decoding long block codes feasible, they can adapt to varying channel conditions, treat good and bad blocks differently, and can be used with different codes, modulation schemes and applications. High code rates are achieved by using long block codes in both block Turbo/ product code structures and in multilevel code structures.

1.2 Background

The basic goal of telecommunications engineering is to transmit and receive data over a channel with an error rate below a predefined value. There are many components required in a communication system to achieve this goal. In addition it is desirable to use as little bandwidth, power, time, money and complexity as possible.

The first component in the transmitter is the source encoder as shown in Fig.

[1]To within $2dB$ of capacity on the additive white Gaussian noise channel.

1.1. It is used to reduce redundancies in the binary information to be transmitted [44]. This process is reversed at the end of the receiver by the source decoder. In this thesis it is assumed that the input to the error correction encoder has already been through a source encoder, which has produced equiprobable symbols.

Modulation is used to convert the encoded data to a form more suitable for transmission over a channel. It maps an encoded data stream to a predefined constellation of points. It also shifts the signal to the correct frequency for transmission after pulse shaping. The frequency, phase or amplitude of the carrier wave may be used to transmit the information. Demodulation is the reverse process to modulation.

Two linear modulation schemes are considered in this thesis: M-ary *quadrature amplitude modulation (QAM)* and M-ary *phase shift keying (PSK)*. M-ary QAM transmits information using different amplitudes in both the in-phase and quadrature dimensions. The resulting constellation has M points on a grid. Square M-ary QAM constellations are considered in this thesis. They use $\sqrt{M}$ amplitudes in each dimension as shown in Fig. 1.2 (each dimension transmits a $\sqrt{M}$-ary *amplitude shift keying (ASK)* signal). M-ary QAM trades off power for bandwidth. The M-ary QAM signal can be written as [85]

$$s_m(t) = a_I h(t)\cos(2\pi f_c t) - a_Q h(t)\sin(2\pi f_c t), \quad 0 \le t \le T_s, \quad m = 1, 2, \cdots, M \tag{1.1}$$

where a_I and a_Q are the amplitudes in the inphase and quadrature dimensions respectively, $h(t)$ is the pulse shape, T_s is the symbol period and f_c is the carrier frequency.

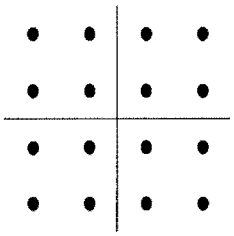


Figure 1.2: M-ary QAM constellation for $M = 16$.

QAM signals transmit information using different amplitudes, whereas, PSK signals transmit information using phase shifts. The PSK constellation points are

equally spaced around a constant energy circle as shown in Fig. 1.3. The M-ary PSK[2] signal can be written as [84]

$$s_m(t) = h(t)\cos\left(\frac{2\pi}{M}(m-1)\right)\cos(2\pi f_c t) - h(t)\sin\left(\frac{2\pi}{M}(m-1)\right)\sin(2\pi f_c t) \quad (1.2)$$

where $m = 1, 2, \cdots, M$ and $0 \leq t \leq T_s$.

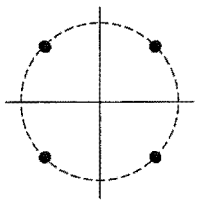


Figure 1.3: M-ary PSK constellation for $M = 4$ (QPSK).

If a sequence of constellation points was transmitted without pulse shaping, then too much bandwidth would be used. Pulse shaping is used to shape the transmitted signal into a band limited form. In an ideal world the minimum amount of bandwidth would be used (a rectangular spectrum). However, this would require the transmission of an infinite duration sinc pulse. Also in practise it is difficult to build filters with a sharp rolloff. Often raised cosine or square root raised cosine pulse shapes are used as they offer a range of tradeoffs between the required bandwidth and the filter rolloff. Modulation in conjunction with pulse shaping allows data to be transmitted at the desired frequency in a bandwidth efficient way.

The resulting signal is then transmitted over a channel. The channel can corrupt the transmitted signal with additive noise and can introduce multiplicative distortion due to multipath fading or *intersymbol interference (ISI)*. Error correction coding/ decoding, matched filtering and equalization are added to the communication system to assist in recovering the transmitted signal as shown in Fig. 1.1. A filter matched to the transmitted pulse shape (the matched filter) is used in the receiver to maximize the peak pulse *signal to noise ratio (SNR)* [47].

Error correction coding/ decoding is required because the channel can cause errors in the received signal. The error correction encoder adds structured redundancy (non-data symbols) to the data to be transmitted. The structured redundancy

[2]Two commonly used PSK constellations are *binary PSK (BPSK)* which consists of $M = 2$ points and *quadrature PSK (QPSK)* which consists of $M = 4$ points. A QPSK constellation consists of two BPSK constellations in quadrature.

is used by the error correction decoder to detect and/ or correct errors in the received demodulated data. The receiver can then reduce the number of errors in the estimate and/ or request that data blocks in error be retransmitted. Error correction coding and decoding will be discussed in more detail in chapter 2.

Error correction coding and modulation can be designed together by using multilevel coding [16, 52, 82, 119]. For complexity reasons multilevel codes are usually decoded using a MSD. Multilevel codes and MSDs will be considered in more detail in chapter 2.

Equalization is used to try to reverse the ISI introduced by the channel. It does this by using the statistics of the received signal and any other knowledge available about the channel (*channel state information(CSI)*).

The design criteria for the error correction coding/ decoding and equalization depend on the channel and how it corrupts the transmitted signal. In this thesis the memoryless AWGN channel and the uncorrelated Rayleigh flat fading channel are considered.

1.2.1 Additive White Gaussian Noise Channel

The AWGN channel is typical for satellite and deep-space communications [69]. On the memoryless AWGN channel the received signal is corrupted by an additive Gaussian random variable with zero mean and two-sided power spectral density $N_0/2$. The noise produces random errors in the received signal. Error correction coding/ decoding is used to detect and/ or correct these random errors. The received signal can be written as a function of time as

$$r(t) = s(t) + g(t) \tag{1.3}$$

where $s(t)$ is the transmitted signal and $g(t)$ is the AWGN signal. When sampled at the symbol rate, the l^{th} received sample (at time $t = lT_s$) is $r_l = s_l + g_l$, where s_l is the l^{th} transmitted symbol and g_l is the l^{th} independent AWGN sample. The probability density function of the l^{th} received sample, r_l, given that s_l was transmitted is given by [69]

$$p(r_l|s = s_l) = \frac{1}{\sqrt{2\pi\sigma_r^2}} \exp\left(\frac{-|r_l - s_l|^2}{2\sigma_r^2}\right) \tag{1.4}$$

where σ_r^2 is the noise variance. This expression will be used by the error correction decoder to determine if s_l was transmitted. As a function of s_l it is known as a likelihood.

The best performance possible is called capacity, it was defined for the AWGN channel by Shannon in [100]. His channel coding theorem states that an arbitrarily small probability of error can be achieved at all rates less than capacity [69]. In [32] a rate 1/2 low density parity check code of block length 10^7 is stated to be within 0.036dB of capacity at a BER of 10^{-6}. In this thesis codes with shorter block lengths ($< 10^4$) and generally with higher code rates will be considered. The capacity of a bandlimited AWGN channel with an input signal that is bandwidth and average power limited is given by [84, 116]

$$\mathcal{C} = B\log_2\left(1+\frac{P_{av}}{BN_0}\right) \quad bits/second \tag{1.5}$$

where B is the channel bandwidth, P_{av} is the average transmitted signal power and $P_{av}/(BN_0)$ is the SNR. As the channel bandwidth tends to infinity the capacity approaches its asymptotic value of [84, 116]

$$\lim_{B\to\infty} \mathcal{C} \approx 1.44\frac{P_{av}}{N_0}. \tag{1.6}$$

1.2.2 Rayleigh Fading Channel

When a radio signal is transmitted it may experience reflection, diffraction, scattering or a combination of these[3]. This results in more than one version of the transmitted signal being received. Each version has travelled a different path and typically has a different phase, delay, Doppler frequency shift and attenuation. This leads to constructive and destructive interference, this effect is called fading [97].

The AWGN channel corrupts the transmitted signal with AWGN. The Rayleigh fading channel corrupts the transmitted signal with multiplicative fading and AWGN. The multiplicative fading is usually modelled as a complex Gaussian random process with zero mean. Its envelope has a Rayleigh distribution, which is given by [104]

$$p(x) = \frac{x}{\sigma_x^2}\exp\left(-\frac{x^2}{2\sigma_x^2}\right) \quad x \geq 0 \tag{1.7}$$

[3]Reflection occurs when the wavelength of the transmitted signal is small compared to the object it hits [97]. Diffraction occurs when the transmitted signal hits an object with sharp edges [97]. Scattering occurs when the wavelength of the signal is large compared to the size of the object [97].

Fig. 1.4 shows that the amplitude of the multiplicative fading can increase or decrease dramatically. This varies the signal strength and means that the channel has a time varying SNR. In deep fades the noise power can exceed the strength of the faded signal. As a result a large percentage of symbols received during a deep fade can be decoded in error. This results in the Rayleigh fading channel suffering from error bursts in addition to the random errors caused by the AWGN. Therefore, error correction codes are required to correct both error bursts and random errors. Most error correction codes are designed to correct random errors. As a result interleaving/ deinterleaving is often used to break up the error bursts[4] before error correction decoding. Equalization and channel estimation techniques [70] are often used to try and reverse some of the channel corruption before error correction decoding. The channel's correlation is exploited during equalization and channel estimation and so any deinterleaving used should be performed after the equalization, but before the error correction decoding.

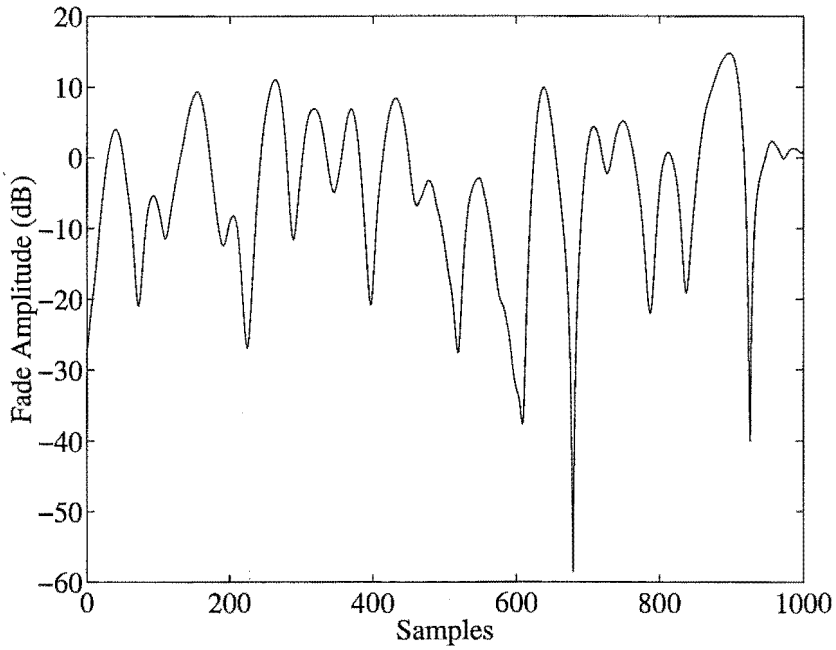


Figure 1.4: Amplitude of a Rayleigh fading variable.

A transmitted symbol will propagate to the receiver using a number of different paths, which may have different delays. The span of path delays is called the

[4]How random the errors are after interleaving/ deinterleaving depends on the length of the error bursts experienced, the interleaver design and the length of the interleaver [116].

maximum excess delay[5], T_m [72, 102]. If the symbol is spread in time to occupying more than a symbol period, $T_m > T_s$, then the channel is called frequency-selective. In this case different frequencies in the transmitted signal will experience different attenuations and phase shifts, and the channel induces ISI [104]. If all versions of a transmitted symbol arrive within a symbol period, $T_m < T_s$, then the channel is called frequency non-selective or flat. The flat fading channel does not induce ISI and all frequencies in the transmitted signal experience the same attenuation and phase shift [104].

A fading channel is also characterised as experiencing slow or fast fading. *Fast fading* occurs when the channel characteristics may vary significantly within the duration of a symbol. *Slow fading* occurs when the channel characteristics vary little during the duration of a symbol. The coherence time, T_0, is used to define the speed of the fading. It is the expected time in which the channel's response is almost invariant [102]. The fading is fast if $T_0 < T_s$ and slow if $T_0 > T_s$ [102].

In chapter 5 the uncorrelated Rayleigh flat fading channel is considered. The received signal from an uncorrelated Rayleigh flat fading channel can be written as [8, 83]

$$r(t) = \gamma(t)s(t) + g(t), \qquad 0 \leq t \leq T_s \tag{1.8}$$

where $\gamma(t) = \gamma_A(t)e^{\mathrm{j}\theta(t)}$ is the multiplicative fading, $\gamma_A(t)$ is the magnitude of the fading, $\mathrm{j} = \sqrt{-1}$ and $\theta(t)$ is the phase of the fading [8]. If the fading is slow enough, then it can be assumed that $\theta(t)$ can be estimated and compensated for [8]. In this case coherent demodulation may be assumed and the received signal can be simplified to [83]

$$r(t) = \gamma_A(t)s(t) + g(t), \qquad 0 \leq t \leq T_s \tag{1.9}$$

Recall that the instantaneous SNR varies on the Rayleigh fading channel. Therefore, the average capacity of the Rayleigh fading channel is considered. When the average SNR, Γ, is greater than two the average capacity is defined as [58, 116]

$$\langle \mathcal{C} \rangle = B log_2(\exp(1)) \exp(-1/\Gamma) \left(-\mathcal{E} + \log(\Gamma) + \frac{1}{\Gamma} \right) \tag{1.10}$$

where $\log(\cdot)$ is the natural logarithm, $\log_2(\cdot)$ is the base-2 logarithm and $\mathcal{E}$ is Euler's constant. Since the instantaneous SNR varies with time, the instantaneous capacity

[5]Only received paths with a magnitude over a given threshold are considered.

can be quite different from the average capacity. As a result the average channel capacity of the Rayleigh fading channel is worse than that of the Gaussian channel. But when the channel bandwidth tends to infinity the channel capacities are asymptotically equal and the average channel capacity becomes [58, 116]

$$\lim_{B\to\infty} \langle C \rangle \approx 1.44 \frac{P_{av}}{N_0} \tag{1.11}$$

The difference between the capacity of the Rayleigh fading channel and the AWGN channel is $\approx 4dB$ for uncoded BPSK [116]. At an error rate of 10^{-4} BPSK performance on the Gaussian channel is only $\approx 4dB$ from capacity, while its performance on the Rayleigh fading channel is $\approx 22dB$ from capacity [116]. Therefore, there is potential for larger coding gains on the Rayleigh fading channel.

1.3 Thesis Content

In Chapter 2 the key background information on error correction coding and multilevel coding is introduced and a literature survey of the area is provided. Chapter 2 introduces the sub-optimal SISO decoding algorithm of [90] and iterative multistage decoding as powerful reduced complexity decoding techniques. It also highlights the need for a flexible adaptive approach to sub-optimal SISO decoding and a new approach to iterative multistage decoding.

In [90] a sub-optimal SISO decoding algorithm is described, which allows block Turbo codes with long block component codes (namely BCH codes) to be iteratively decoded with a manageable decoding complexity. Block Turbo codes can be used in a wide range of applications including *very small aperture terminal (VSAT)* systems (satellite links), wireless *local area networks (LANs)* and broadband wireless internet access links [2]. Block Turbo codes are a good solution for these applications and others, due to their good performance, high code rates, flexible code lengths and range of data rates [2]. The SISO decoder of [90] calculates soft information from the output of a *soft-input list-output (SILO)* decoder. This is done by using two sequences of precomputed parameters. The main restriction of the SISO decoder of [90] is that these sequences need to be optimized using repeated simulations for different codes, modulation schemes, channel conditions and applications for

optimum results. In addition the parameters treat good and bad received blocks the same way, and they do not adapt to varying channel conditions.

In chapter 3 two new adaptive sub-optimal SISO decoding algorithms based on [88, 90] are developed. One approach is based on approximation and the other on the distance properties of the component codes. Both approaches estimate the two sequences of parameters on a block-by-block basis. Therefore, good and bad received blocks can be treated differently, the algorithm can adapt to varying channel conditions and the sequences of parameters do not need to be optimized using repeated simulations for different codes, modulation schemes, channel conditions or applications. The new algorithms will be derived by considering the extrinsic information to be a Gaussian random variable [14, 15, 21], rather than as a priori information [21, 43], as this is found to result in better performance.

Two important design criteria for a communication system are bandwidth efficiency and complexity. Multilevel coding combines error correction coding and modulation to provide a bandwidth efficient system. It also allows complicated codes to be created in a low complexity manner. For complexity reasons they are typically decoded using a MSD, which passes hard decisions to subsequent levels (a traditional MSD). However, by using hard decisions, information is being lost. In addition the component decoders never use the information gained from decoding subsequent levels. Performance can be improved by passing soft information to both previous and subsequent levels [27, 53, 71, 73, 74, 121]. Many of the previous approaches do not pass reliability information back to all previous levels [27, 73, 74] and use interleaving between some of the component encoders and the channel mapping [27, 73, 74, 121]. Interleaving can increase the encoding and decoding delay and so should be avoided when not needed.

Most previous work treats the reliability/ extrinsic information as a priori information (a priori approach) [27, 42, 53, 71, 73, 74]. In chapter 4 a new approach to iterative multistage decoding is derived by treating the extrinsic information as a Gaussian random variable (Gaussian approach). The Gaussian approach is used since it is found to result in better performance when one of the adaptive sub-optimal SISO decoding algorithms of chapter 3 is used. The iterative MSD of chapter 4 passes reliability information to all previous and subsequent levels. It is considered with

and without interleaving.

The decoding algorithm of [79, 90] is not well suited for use in the iterative MSD developed in chapter 4. In an iterative MSD the magnitude of the extrinsic information passed to other levels must be correct. If the algorithm of [79, 90] is to be used, then a joint optimization of the two precomputed sequences of parameters used on every level is required using repeated simulations. This does not allow for varying conditions, and good and bad blocks are treated the same way. In [77, 79] the received signal is mapped so that the input to each level is centred on two possible values[6]. Information about the labelling bits from other levels is not known when this mapping technique is used and this information is required by the iterative MSD.

The adaptive approaches of chapter 3 avoid these restrictions by calculating the sequences of parameters on a block-by-bock basis. The notation for these adaptive approaches is extended in chapter 4 for use in an iterative MSD and with different constellations. The new notation calculates the soft input and output log likelihoods using the constellation points, which avoids the mapping of [79].

One of the adaptive SISO decoding algorithms of chapter 3 is used on the uncorrelated Rayleigh flat fading channel in chapter 5. Simulation results are presented for when ideal or no CSI is available. Finally conclusions and suggestions for future work are discussed in chapter 6.

1.4 Thesis Contributions

This thesis contains original contributions to the fields of block Turbo coding and multilevel coding.

1.4.1 Contributions to Block Turbo Coding

Two adaptive sub-optimal SISO decoding algorithms for block codes are developed in chapter 3. They can be used as component SISO decoders in an iterative decoder for block Turbo codes. One decoding algorithm is based on approximation (approx-

[6]This means that the algorithm must be modified (a new mapping developed) when different constellations (with more than two points) or partitioning strategies are used.

imation approach) and the other on the distance properties of the component codes (distance approach). The notation is extended for use in an iterative MSD and with different constellations in chapter 4. The adaptive approaches are used for signals transmitted on the AWGN channel in chapter 3 and chapter 4. The approximation approach is used for signals transmitted on the uncorrelated Rayleigh flat fading channel in chapter 5. These approaches are more flexible and improve performance over the precomputed approach of [54, 90].

1.4.2 Contributions to Multilevel Coding

In chapter 4 a new approach to iterative multistage decoding is developed that passes extrinsic information to all previous and subsequent code levels. It treats the extrinsic information passed between levels as a Gaussian random variable, rather than as a priori information. The Gaussian approach is found to perform better than the a priori approach in the simulations of chapter 4. The adaptive algorithms improve performance over [79, 90] when a traditional MSD is used. When the approximation based adaptive algorithm is used in the new iterative MSD it performs significantly better than in a traditional MSD.

1.4.3 Publications

Some of the research presented in this thesis has been presented in the following published papers

- P. A. Martin and D. P. Taylor, "On Adaptive Reduced-Complexity Iterative Decoding", Globecom, San Francisco, USA, 2000 [65].
- P. A. Martin and D. P. Taylor, "On Iterative Multistage Decoding", 2^{nd} International Symposium On Turbo Codes and Related Issues, Brest, France, 2000 [66].

and the following submitted papers

- P. A. Martin and D. P. Taylor, "On Multilevel Codes and Iterative Multistage Decoding", Accepted for publication in the IEEE Transactions on Communications, February 2001 [67].

- P. A. Martin and D. P. Taylor, "Distance Based Adaptive Scaling in Sub-Optimal Iterative Decoding", Submitted to the IEEE Transactions on Communications, August 2000 [64].

Chapter 2

Background Information

2.1 Introduction

This chapter covers the relevant background information on error correction coding, error correction decoding, multilevel coding and multistage decoding. It focuses on codes and coded signals transmitted on the *additive white Gaussian noise (AWGN)* channel, but it is noted that they can also be transmitted on fading channels[1] [54, 90].

There are two main categories of error control codes, namely convolutional and block codes [69, 120]. A convolutional code encodes a data stream into a single codeword. A block code splits the data stream into blocks of length k. Each length k block is then encoded. In this thesis block codes will be considered as they can provide good performance at high code rates.

There are many types of block codes [69, 120]. *Bose-Chaudhuri-Hocquenghem (BCH)* codes are one of the most powerful subclasses of block codes [69, 120] and includes all perfect codes [69]. They are cyclic codes which can encode either binary or non-binary data, and can correct multiple errors [69]. In this thesis primitive binary BCH codes are considered. They are binary linear block codes and are described in section 2.2. A *hard-input hard-output (HIHO)* decoder for BCH codes is described in section 2.2. However, performance can be improved by using the reliability information contained in the received signal. *Soft-input hard-output (SIHO)* decoders use this information and are considered in section 2.2. A reduced complexity SIHO

[1]However, the code design criteria for fading channels are different from those for the AWGN channel.

decoder is considered, which allows long BCH codes to be decoded with manageable decoding complexity.

Many concatenated code structures have been developed to construct powerful codes using a number of component codes. Early concatenated structures include product codes [23] and two stage concatenated codes as shown in Fig. 2.1 (the outer code was typically a powerful *Reed Solomon (RS)* code[2]) [29, 69]. These structures were able to provide good performance by using powerful component codes. The concatenated codes were typically decoded in stages as shown in Fig. 2.1 and so the decoding complexity was dependent on the complexity of the component codes (making it manageable). Both of these concatenated structures pass information from the inner decoder to the outer decoder(s). If the inner decoder is a *soft-input soft-output (SISO)* decoder, then the outer decoder can be a soft input decoder and can use the received signal and the information gained by the inner decoder (called extrinsic information) to improve performance.

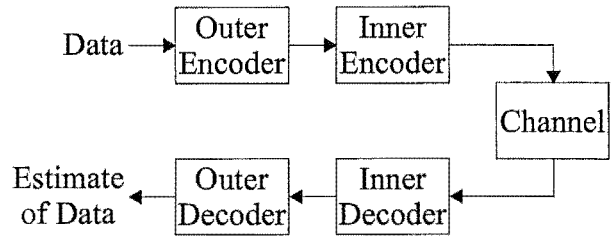


Figure 2.1: Two stage concatenated code.

There has been increased interest in concatenated codes since the invention of Turbo codes [15]. Turbo codes use a random interleaver, π_r, and two weak convolutional codes in a parallel concatenated structure as shown in Fig. 2.2 to provide performance near capacity. All the component codes are decoded using SISO decoders and so extrinsic information can be passed between all the decoders. All the component codes can be decoded more than once to improve performance. This is called iterative decoding and is discussed in more detail in section 2.3.

More recently other concatenated code structures have been decoded using iterative decoding, including serial concatenated [13], hybrid concatenated [22], self concatenated [9, 10] and product code structures [90]. Product codes are also known

[2]RS codes are a subclass of non-binary BCH codes [69].

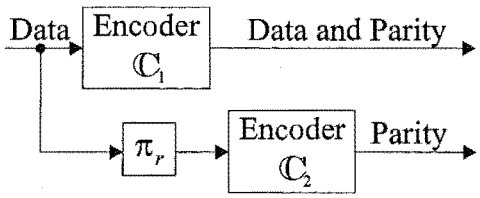


Figure 2.2: Encoder for a Turbo code.

as block Turbo codes [90]. They will be referred to as block Turbo codes in this work. Block Turbo codes can also be constructed using other concatenation structures. However, in this thesis the term is just used for the product code structure. Serial and parallel concatenated structures [11, 12] are shown to be closely related to product/ block Turbo codes [23] in appendix A. In this thesis block Turbo codes are used as they can provide good performance at high code rates. They are described in section 2.3. A sub-optimal reduced-complexity SISO decoder is described in section 2.3 which allows long block codes to be used as component codes in a block Turbo code with manageable decoding complexity.

In this thesis BCH codes and block Turbo codes (with BCH component codes) are used as component codes in multilevel codes. Multilevel coding combines coding and modulation to create powerful codes in a moderate complexity, bandwidth efficient manner. Multilevel codes are described in section 2.4. They can be decoded in a staged manner using a *multistage decoder (MSD)* or an iterative MSD as described in section 2.4.

2.2 Primitive Binary BCH Codes

Primitive binary BCH codes are used in this thesis and form a powerful subclass of linear cyclic block codes. An encoder and two types of decoders for these codes are described in this section. The first is a HIHO decoder. The disadvantage of this type of decoder is that it does not use any reliability information from the received signal, but typically provides a simple decoding procedure. The second type is a SIHO decoder. It uses reliability information from the received signal to improve performance. The cost is increased decoding complexity. For long block

codes the complexity can be prohibitive and so a reduced-complexity, sub-optimal SIHO decoder is also described.

The BCH encoder adds redundancy to a block of data in a structured way. The BCH decoder uses the structured redundancy to detect and/ or correct errors in the received signal. A (n, k) binary block code has 2^k codewords each with n encoded bits, $k \leq n$ of which are information/ data bits. The $n - k$ redundant bits added by the encoder are called parity bits. The rate of the code is then k/n.

The distance properties of a code are important in determining its performance. The Hamming distance between two codewords is the number of positions in which they differ. The weight of a binary codeword is the number of ones it contains. The Hamming distance between two binary codewords equals the weight of the modulo-2 sum of the two codewords. Since the sum of any two codewords in a linear code produces another codeword [63], the minimum Hamming distance, $d_{H,min}$, of the binary code equals the minimum non-zero weight of any codeword in the code. It determines the number of errors that can be corrected[3], $t \leq \lfloor (d_{H,min} - 1)/2 \rfloor$ [69], or detected, $l \leq d_{H,min} - 1$ [120]. If $d_{H,min}$ is an even number, then any combination of up to $t = (d_{H,min}/2) - 1$ errors can be corrected and it can be detected if $l = d_{H,min}/2$ errors are received.

The BCH codes used in this thesis will usually be extended by adding a parity bit to the beginning of the unextended BCH codewords to ensure even parity. These will be called extended BCH codes. The BCH codes considered in this thesis have odd values of $d_{H,min}$ and extending them increases the minimum Hamming distance by one.

2.2.1 Encoder

Primitive binary BCH codes are cyclic codes constructed using a primitive element, α, from an extension field $GF(2^m)$ of $GF(2)$, where $GF(q)$ represents a Galois field with q elements [69]. A table of generator polynomials for primitive BCH codes is given in [69, 120]. A length n systematic[4] binary BCH codeword can be generated

[3] $\lfloor a \rfloor$ is the closest integer to a of equal or lesser value [120].

[4] If the information bits appear unaltered in the encoded word, then the code is called systematic [69].

according to the polynomial equation [69]

$$\boldsymbol{c}(x) = [x^{n-k}\boldsymbol{m}(x) mod \boldsymbol{g}(x)] + x^{n-k}\boldsymbol{m}(x) \tag{2.1}$$

where $\boldsymbol{g}(x)$ is the generator polynomial, $\boldsymbol{c}(x) = c_0 + c_1 x + \cdots + c_{n-1}x^{n-1}$ is the polynomial representing the codeword and $\boldsymbol{m}(x)$ is the polynomial representing the data bits. The first term in (2.1) provides the parity check bits and the second term the data bits.

The extended BCH codeword, $\boldsymbol{c}'(x)$, can be generated from the unextended length n BCH codeword, $\boldsymbol{c}(x)$, according to the polynomial equation

$$\boldsymbol{c}'(x) = \left(\sum_{l=0}^{n-1} c_l\right) \bmod 2 + x\boldsymbol{c}(x) \tag{2.2}$$

Here, the overall parity bit is placed at the beginning of the codeword and results in a length $n' = n + 1$ codeword.

2.2.2 Hard-Input Hard-Output Decoder

The most widely used HIHO decoder for BCH codes is the Berlekamp-Massey algorithm [69, 120]. In this thesis single, double and triple error correcting primitive binary BCH codes are used. For small numbers of errors (such as those considered here) Peterson's direct method provides a simple method for decoding BCH codes. It will be used in this thesis to correct up to t errors. This subsection summarizes Peterson's direct method of HIHO decoding binary BCH codes using the descriptions in [69, 120]. Unextended binary BCH codes are assumed in this section.

The hard input to the decoder is the hard (binary) decision on the output from the demodulator. The hard input polynomial equals $\boldsymbol{y}(x) = \boldsymbol{c}(x) \oplus \boldsymbol{e}(x) = y_0 + y_1 x + \cdots + y_{n-1}x^{n-1}$, where $\boldsymbol{e}(x)$ is the polynomial representing the received error sequence and $\oplus$ represents modulo-2 addition. The first step is to see if there are errors in $\boldsymbol{y}(x)$, by calculating the syndromes. The k^{th} syndrome is given by [69, 120]

$$\mathcal{S}_k = \boldsymbol{y}(\alpha^k) = y_0\alpha^{k(0)} + y_1\alpha^{k(1)} + \cdots + y_{n-1}\alpha^{k(n-1)}, \qquad k = 1, 3, \cdots, 2t \tag{2.3}$$

where all x in $\boldsymbol{y}(x)$ have been replaced with the k^{th} root of the generator polynomial (which is the k^{th} power of the primitive element), α^k. Every codeword produces

syndromes of zero and so $\mathcal{S}_k = \boldsymbol{e}(\alpha^k)$. Therefore, non-zero syndromes are obtained only when there are errors in the hard input, $\boldsymbol{y}(x)$. If the code can correct errors (meaning $t \geq 1$), then an error correction decoder can be used to find the error locations.

The location of the i^{th} error, which occurs in the j^{th} position is denoted $X_i = \alpha^j$. Assuming $\boldsymbol{y}(x)$ has v errors, the k^{th} syndrome can be written as [69, 120]

$$\mathcal{S}_k = \boldsymbol{e}(\alpha^k) = \sum_{i=1}^{v} (X_i)^k, \qquad k = 1, 3, \cdots, 2t \tag{2.4}$$

The syndromes can be calculated using $\boldsymbol{y}(x)$ as in (2.3) and the error locations can be found by using the syndromes, due to the relationship in (2.4). The error locator polynomial is used to find the error locations and is defined as [69, 120]

$$\boldsymbol{\sigma}(x) = \prod_{i=1}^{v} (x + X_i) = x^v + \sigma_1 x^{v-1} + \cdots + \sigma_v \tag{2.5}$$

The roots of $\boldsymbol{\sigma}(x)$ are the error locations.

Newton's identities allow the calculation of the error locator coefficients, σ_i, from the syndromes. Assuming $v = t$ errors, the identities can be written as a set of t equations with t unknowns as [69, 120]

$$\begin{gathered} \mathcal{S}_1 + \sigma_1 = 0 \\ \mathcal{S}_3 + \sigma_1 \mathcal{S}_2 + \cdots + \sigma_2 \mathcal{S}_1 + \sigma_3 = 0 \\ \vdots \\ \mathcal{S}_{2t-1} + \sigma_1 \mathcal{S}_{2t-2} + \cdots + \sigma_t \mathcal{S}_{t-1} = 0 \end{gathered} \tag{2.6}$$

Peterson's method uses the direct solution to this set of simultaneous equations to correct a given number of errors. The error locator coefficient for single error correction is

$$\sigma_1 = \mathcal{S}_1 = \boldsymbol{y}(\alpha) = \boldsymbol{e}(\alpha) \tag{2.7}$$

The coefficients for double error correction are

$$\sigma_1 = \mathcal{S}_1 \tag{2.8}$$

$$\sigma_2 = \frac{\mathcal{S}_3 + \mathcal{S}_1^3}{\mathcal{S}_1} \tag{2.9}$$

and for triple error correction are

$$\sigma_1 = \mathcal{S}_1 \tag{2.10}$$

$$\sigma_2 = \frac{\mathcal{S}_1^2\mathcal{S}_3 + \mathcal{S}_5}{\mathcal{S}_1^3 + \mathcal{S}_3} \tag{2.11}$$

$$\sigma_3 = \mathcal{S}_1^3 + \mathcal{S}_3 + \mathcal{S}_1\sigma_2 \tag{2.12}$$

The equations quickly get more complicated for larger numbers of errors [69, 120].

When $t > 1$ and at least one non-zero syndrome has been calculated, a test is done to see if there are t or $t-1$ errors in $\boldsymbol{y}(x)$. If there are t or $t-1$ errors in $\boldsymbol{y}(x)$, then [69, 120]

$$\det(\mathcal{A}_s) = \det \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ \mathcal{S}_2 & \mathcal{S}_1 & 1 & 0 & 0 & \cdots & 0 \\ \mathcal{S}_4 & \mathcal{S}_3 & \mathcal{S}_2 & \mathcal{S}_1 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathcal{S}_{2t-2} & \mathcal{S}_{2t-3} & \mathcal{S}_{2t-4} & \mathcal{S}_{2t-5} & \mathcal{S}_{2t-6} & \cdots & \mathcal{S}_{t-1} \end{bmatrix} \neq 0 \tag{2.13}$$

and the direct solution for t-error correction is used. If $\sigma_t = 0$, then $t-1$ errors will be corrected and $\boldsymbol{\sigma}(x)$ has degree $t-1$. If $\det(\mathcal{A}_s) = 0$ and $t > 2$, then the two bottom rows and the two rightmost columns of $\mathcal{A}_s$ are removed to produce $\mathcal{A}'_s$ [69]. If $det(\mathcal{A}'_s) \neq 0$ then $t-2$ or $t-3$ errors are corrected, otherwise the matrix is reduced again. This continues until the determinant of the matrix is non-zero. This means that for $t = 1$, if $\mathcal{S}_1 \neq 0$ one error is corrected, otherwise no errors are corrected. For $t = 2$, if $\mathcal{S}_1 \neq 0$ one or two errors are corrected (using the two error correction formulae), otherwise no errors are corrected. For $t = 3$, if $((\mathcal{S}_1)^3 + \mathcal{S}_3) \neq 0$ then two or three errors are corrected (using the three error correction formulae). Otherwise, if $\mathcal{S}_1 \neq 0$, then one error is corrected, otherwise no errors are corrected.

The locations of the errors can be found using $\boldsymbol{\sigma}(x)$ by setting x equal to each position in the codeword in turn. The error-locator polynomial, $\boldsymbol{\sigma}(x)$, equals zero when x equals an error location. If the correct number of distinct roots are found in the appropriate field, then $\boldsymbol{y}(x)$ is corrected in these positions, otherwise a decoding failure is declared. The final step is to check that the decoded vector is a codeword.

In summary the decoding steps are

1. Calculate the syndromes using (2.3) and terminate decoding if all syndromes equal zero.

2. Use (2.13) to determine the number of errors to correct.

3. Calculate the coefficients of $\boldsymbol{\sigma}(x)$ using Peterson's direct solutions to (2.6).

4. Find the roots of $\boldsymbol{\sigma}(x)$. If the correct number of distinct roots in the appropriate field are found, then correct $\boldsymbol{y}(x)$ by changing the bits in the error locations, otherwise declare a decoding failure.

5. Check that the decoded vector is a codeword, otherwise declare a decoding failure[5].

Peterson's direct method does not always find a codeword. A decoding failures occurs when no codeword is within Hamming distance t of $\boldsymbol{y}(x)$. In this case $\boldsymbol{\sigma}(x)$ may have repeated roots or have roots outside the field the BCH code was constructed on. A decoding error occurs when the wrong codeword is within Hamming distance t of $\boldsymbol{y}(x)$.

An extended BCH code was defined as a BCH code with an overall parity bit to ensure even parity. In later chapters the unextended BCH code is decoded using Peterson's algorithm and then the decoded overall parity bit is set equal to the modulo-2 sum of all bits in the decoded unextended BCH codeword.

The problem with using a HIHO decoding algorithm is that no reliability information from the soft output of the demodulator, $\boldsymbol{R}$, is used by the decoder. Performance could be improved by using a SIHO decoding algorithm.

2.2.3 Soft-Input Hard-Output Decoders

In this subsection two SIHO decoding algorithms are considered. First consider a *maximum likelihood (ML)* SIHO decoder for a BPSK signal transmitted on the AWGN channel. The ML decoder calculates the squared Euclidean distance between the soft input and every possible codeword[6]. The squared Euclidean distance between the soft input, $\boldsymbol{\lambda}$, and a codeword, $\boldsymbol{C}$, is defined as

$$d_E^2(\boldsymbol{\lambda}, \boldsymbol{C}) = |\boldsymbol{\lambda} - \boldsymbol{C}|^2 \tag{2.14}$$

[5]This step was not actually used in simulations.

[6]The bits in the codeword are mapped from $\{0, 1\}$ to $\{-1, +1\}$.

where $\boldsymbol{\lambda} = \boldsymbol{R}$ is the soft output from the demodulator. The codeword at the minimum squared Euclidean distance from the soft input is selected as the decoder's decision (hard output). The problem with this decoding algorithm is that all 2^k possible codewords are considered and in this thesis $21 \leq k \leq 113$, which makes the algorithm prohibitively complex.

If $n-k < k$ then ML decoding using the dual code can be a less complicated solution [43]. Dual codes are defined in appendix B. The information vector for the dual code has length $n-k$ and so the dual code has 2^{n-k} possible codewords [69]. Most of the codes considered in this thesis are long block codes and even ML decoding their dual codes can be prohibitively complex. In such cases reduced-complexity approaches are required.

A sub-optimal reduced-complexity approach to SIHO decoding is to use a list based decoder [62]. A list based decoder uses the soft input to choose a set of possible error sequences. This set is used to produce a list of possible transmitted codewords from which to choose the decision codeword (hard output). Unless the list contains all possible codewords, it is a sub-optimal decoder and should have a *bit error rate (BER)* between that of a HIHO decoder and a ML SIHO decoder. A variety of different approaches to selecting/ generating the list of possible codewords have been proposed [19, 26, 30, 33, 34, 35, 36, 40, 46, 55, 56, 99, 105, 106, 109, 110]. Some of these list based decoders use a HIHO decoder to generate the list, while others use an encoder.

The second algorithm in [19] is used in this thesis. It will be called the Chase algorithm/ decoder. The Chase decoder generates a set of error patterns, which are added modulo-2 to the hard decision on the soft input to the decoder to produce a set of test sequences, $\{\boldsymbol{T}\}$. Each test sequence is decoded using a HIHO decoder. This results in a list of possible codewords. The codeword in the list which is closest to the soft input in terms of squared Euclidean distance is the decision codeword, $\boldsymbol{D}$. The remaining codewords in the list are the set of competing codewords, $\{\boldsymbol{C}\}$. Therefore, this decoder can produce a hard or list output and so can be used as a SIHO or *soft-input list-output (SILO)* decoder. The structure of the Chase decoder is shown in Fig. 2.3.

Only a small subset of possible error patterns and therefore, codewords are

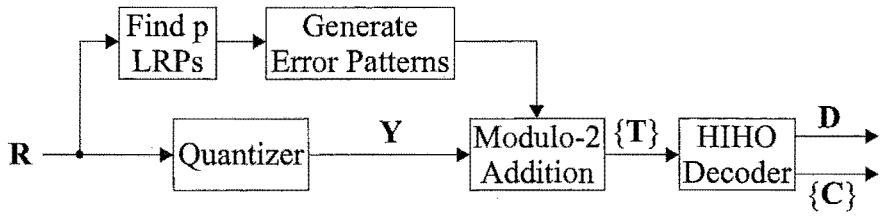


Figure 2.3: Structure of the Chase decoder.

considered. The set of error patterns consists of all sequences containing any combination of ones in the $p = \lfloor \frac{d_{H,min}}{2} \rfloor$ *least reliable positions (LRPs)* in the soft input to the decoder. This results in 2^p error patterns and test sequences. In [90] the $p \geq \lfloor \frac{d_{H,min}}{2} \rfloor$ LRPs are considered and $\leq 2^p$ of the test sequences produced are actually used [77, 79, 89, 90]. In [35] a method is introduced which avoids having to decode all test sequences by looking at the Hamming weights of the codewords already found by the Chase decoder.

If an extended BCH code is used, then the Chase decoder decodes the unextended BCH code. Therefore, only positions in the unextended code are used in the search for the p LRPs. The decoded overall parity bit equals the modulo-2 sum of all bits in the decoded unextended BCH codeword. The decoded extended codeword at the minimum squared Euclidean distance from the soft input is the decision codeword, $\boldsymbol{D}$.

2.3 Block Turbo Codes

Block Turbo codes are a form of concatenated code. They are used in this thesis to provide good performance at high code rates. The block Turbo codes considered in this thesis are product codes. As in [90] the term block Turbo code will be used instead of product code. However, there are some block Turbo code structures which can not be thought of as product codes. For example, some serially concatenated codes can not be thought of as product codes (as mentioned in appendix A).

Block Turbo codes can be created with any number of dimensions/ component codes. A V-dimensional block Turbo code [23] encodes a $k_1 \times k_2 \times \cdots \times k_V$ V-dimensional block of data. In this thesis two-dimensional block Turbo codes with

systematic linear block component codes are considered. The structure of a two-dimensional block Turbo code matrix is shown in Fig. 2.4. The rows are encoded by a $(n_1, k_1, d_{H,1})$ code, $\mathbb{C}_1$, where n_1 is the length of a codeword in $\mathbb{C}_1$, k_1 is the number of data bits and $d_{H,1}$ is the minimum Hamming distance of the code. The columns are encoded by a $(n_2, k_2, d_{H,2})$ code, $\mathbb{C}_2$, where n_2 is the length of a codeword in $\mathbb{C}_2$, k_2 is the number of data bits and $d_{H,2}$ is the minimum Hamming distance of the code. First the k_2 rows of data are encoded using $\mathbb{C}_1$. Then the n_1 columns of data and parity from the row encoding are encoded using $\mathbb{C}_2$. Due to its structure, all rows are codewords of $\mathbb{C}_1$ and all columns are codewords of $\mathbb{C}_2$. Therefore, either the rows or the columns can be encoded or decoded first.

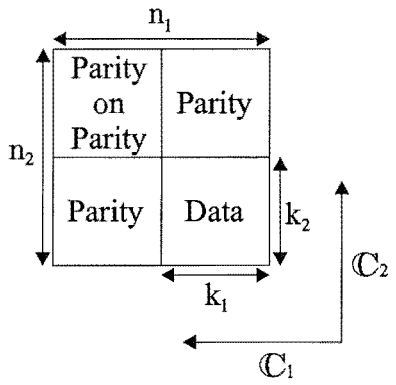


Figure 2.4: Two dimensional block Turbo code with component codes $\mathbb{C}_1$ and $\mathbb{C}_2$.

A two-dimensional block Turbo code has rate

$$\mathcal{R} = \mathcal{R}_1 \times \mathcal{R}_2 \tag{2.15}$$

where $\mathcal{R}_1$ is the code rate of $\mathbb{C}_1$ and $\mathcal{R}_2$ is that of $\mathbb{C}_2$. The minimum Hamming distance of the block Turbo code is given by

$$d_{H,min} = d_{H,1} \times d_{H,2} \tag{2.16}$$

Block Turbo codes have inherent block interleaving between the encoders due to the encoding of rows and then columns (or columns and then rows). This 'interleaver' can be viewed as a rows in columns out (or columns in rows out) block interleaver, π_b. Therefore, the encoder structure of a two-dimensional block Turbo code may be represented by Fig. 2.5.

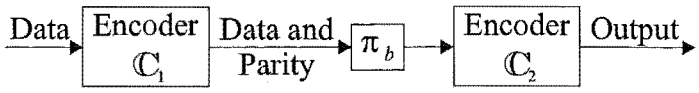


Figure 2.5: Encoder structure of a two-dimensional block Turbo code.

A decoder for a two-dimensional block Turbo code is shown in Fig. 2.6, where $\mathbb{D}_2$ is the decoder for $\mathbb{C}_2$ and $\mathbb{D}_1$ is the decoder for $\mathbb{C}_1$. The columns (or rows) of the block Turbo code matrix are decoded first, then the rows (or columns) are decoded. Therefore, there is an inherent block deinterleaver, π_b^{-1}, between the decoders as shown in Fig. 2.6. Recall that in section 2.2 a HIHO decoding algorithm was described. This algorithm did not fully utilize the received reliability information. Therefore, SIHO decoding algorithms were considered. If $\mathbb{D}_2$ is a SIHO decoder, then it produces no soft information to pass to $\mathbb{D}_1$, so $\mathbb{D}_1$ is a HIHO decoder. However, if $\mathbb{D}_2$ is a SISO decoder, then it can pass soft information to $\mathbb{D}_1$ and $\mathbb{D}_1$ can be a soft input decoder. Now the soft received signal and the reliability information gained by $\mathbb{D}_2$ can be used by $\mathbb{D}_1$ to improve performance. The reliability information gained about the transmitted signal from a decoding is called extrinsic information.

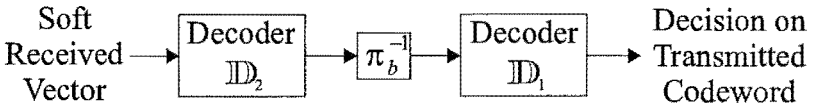


Figure 2.6: Decoding structure for a two-dimensional block Turbo code.

If $\mathbb{D}_2$ and $\mathbb{D}_1$ in Fig. 2.6 are both SISO decoders, then $\mathbb{D}_2$ can pass extrinsic information to $\mathbb{D}_1$ and $\mathbb{D}_1$ can pass extrinsic information back to $\mathbb{D}_2$. This means that the block Turbo code can be decoded a second time. By decoding all component codes several times (each time passing extrinsic information between the decoders) a significant improvement in performance is obtained. This is called iterative decoding and has been increasingly popular since the introduction of Turbo codes [15]. The iterative decoding structure for a two-dimensional block Turbo code is shown in Fig. 2.7. One iteration involves decoding each component code once. The iterative decoder of Fig. 2.7 can also be considered as a pipeline of SISO decoders, where two

decoders are added to the pipeline for each iteration[7].

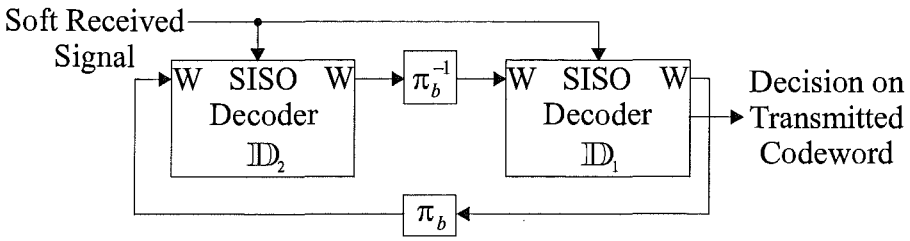


Figure 2.7: Iterative decoding structure for a block Turbo code. $\boldsymbol{W}$ represents the extrinsic information for all encoded bits.

The soft input to each SISO decoder is calculated using the soft received signal, any a priori information available prior to decoding and the extrinsic information from the last decoding of all other codes. Extrinsic information from $\mathbb{D}_1$ / $\mathbb{D}_2$ is not used in later decodings of $\mathbb{D}_1$ / $\mathbb{D}_2$ [101] as it is desirable for the soft received signal and the extrinsic information to provide independent information about the transmitted data. Interleaving between decoders can also be used to decrease the correlation (block Turbo codes have inherent block interleaving between the decoders). Both decoders still indirectly use the same information. Therefore, the extrinsic information from each decoder, the soft received signal and the a priori information will become increasingly correlated in later iterations. Eventually this results in each additional iteration only providing a marginal improvement [43]. In general the second iteration provides the largest gain, with diminishing returns from subsequent iterations.

Since all rows and columns in the block Turbo code matrix are codewords, extrinsic information can be calculated for all transmitted bits (data, parity and parity on parity bits) by every component decoder [90]. The extrinsic information may be treated as a Gaussian random variable [14, 15, 21, 75] or as a priori information [21, 43] when calculating the soft input to a decoder. First consider the case of treating the extrinsic information as a priori information. The input *log likelihood*

[7]In practice only two decoders are used in an iterative structure as shown in Fig. 2.7.

ratio (LLR) to the q^{th} decoder in the pipeline is written as

$$\begin{aligned}\lambda_j(q) &= \log\left(\frac{P(e_j=+1|r_j)}{P(e_j=-1|r_j)}\right) \\ &= \log\left(\frac{p(r_j|e_j=+1)}{p(r_j|e_j=-1)}\right) + \log\left(\frac{P(e_j=+1)}{P(e_j=-1)}\right)\end{aligned} \tag{2.17}$$

where e_j is the j^{th} transmitted bit and r_j is the j^{th} received symbol[8]. This will be called the soft input in the present work. The second term in (2.17) is called the a priori information. If the extrinsic information is assumed to be independent of the received signal and is used as the a priori information then the soft input becomes

$$\lambda_j(q) = \log\left(\frac{p(r_j|e_j=+1)}{p(r_j|e_j=-1)}\right) + w_j(q-1) \tag{2.18}$$

where $w_j(q-1)$ is the extrinsic information on the j^{th} transmitted bit from the previous decoder. If a priori information is available before decoding, then it is included in the soft input to each decoder in the pipeline [43]. The output LLR from the q^{th} decoder in the pipeline, also referred to as the soft output, is defined as

$$\Lambda_j(q) = \log\left(\frac{P(e_j=+1|\boldsymbol{R})}{P(e_j=-1|\boldsymbol{R})}\right) \tag{2.19}$$

where $\boldsymbol{R} = (r_1, \cdots, r_n)$ is the received vector and n is the length of a codeword. The extrinsic information from the q^{th} decoding stage is calculated using

$$w_j(q) = \Lambda_j(q) - \lambda_j(q) \tag{2.20}$$

If the extrinsic information is treated as a Gaussian random variable then the soft input and output LLRs are conditioned on the extrinsic information in addition to the received signal. The soft input can then be written as

$$\lambda_j(q) = \log\left(\frac{P(e_j=+1|r_j, w_j(q-1))}{P(e_j=-1|r_j, w_j(q-1))}\right) \tag{2.21}$$

If the received signal and extrinsic information are assumed to be independent, then the soft input can be written as

$$\lambda_j(q) = \log\left(\frac{p(r_j|e_j=+1)}{p(r_j|e_j=-1)}\right) + \log\left(\frac{p(w_j(q-1)|e_j=+1)}{p(w_j(q-1)|e_j=-1)}\right) + \log\left(\frac{P(e_j=+1)}{P(e_j=-1)}\right) \tag{2.22}$$

[8]One sample per symbol is assumed in this thesis.

When the extrinsic information is treated as a Gaussian random variable the soft output can be written as

$$\Lambda_j(q) = \log\left(\frac{P(e_j = +1|\boldsymbol{R}, \boldsymbol{W}(q-1))}{P(e_j = -1|\boldsymbol{R}, \boldsymbol{W}(q-1))}\right) \tag{2.23}$$

where $\boldsymbol{W}(q-1) = (w_1(q-1), \cdots, w_n(q-1))$ is the vector of extrinsic information from the previous decoder. The extrinsic information is calculated using (2.20). In this case it is often assumed that no a priori information is available (meaning equiprobable symbols and codewords). This approach will be considered in more detail in chapter 3.

Two of the most commonly used SISO decoding algorithms in iterative decoders [43] are the BCJR decoding algorithm [6] and the *soft output Viterbi algorithm (SOVA)* [41]. They are both trellis based decoding algorithms. In this thesis BCH codes (with $n \geq 32$) are used as component codes in block Turbo codes. However, other codes could be used. Unfortunately, due to the trellis complexity of some of these codes, iteratively decoding them using these SISO decoding algorithms can be prohibitively complex. If $n - k < k$, then SISO decoding a code using its dual can reduce the decoding complexity [43]. However, for some of the long block codes considered in this thesis, even SISO decoding using their dual codes can be prohibitively complex. In these cases a reduced complexity approach is required.

2.3.1 Non-adaptive Reduced-Complexity SISO Decoder

In this subsection the sub-optimal reduced-complexity SISO decoding algorithm of [54, 77, 79, 80, 86, 87, 88, 89, 90] is described. This will be referred to as the non-adaptive SISO decoder or the precomputed approach. In [88, 90] a low complexity method was described for calculating soft information using the output of a SILO decoder. As a result a wider range of long block codes can be used as component codes in block Turbo codes with manageable iterative decoding complexity. Another approach to iteratively decoding block Turbo codes is discussed in [34], but will not be considered in great detail here. Both of these approaches are list based, but have different approaches to selecting the list and estimating the soft output. In [34] the SILO order-i reprocessing algorithm is used. The soft input is ordered in terms of reliability and the k *most reliable positions (MRPs)* form a basis. An equivalent

code is developed with these k positions as the information positions. The hard decision on the k MRPs is encoded. Then error patterns are created and encoded, and a soft output is calculated.

Consider a block Turbo code transmitted using BPSK on the AWGN channel. The elements of the transmitted codeword, $\boldsymbol{E}=(e_1,\cdots,e_n)$, are mapped from $\{0,1\}$ to $\{-1,+1\}$, giving the sampled received word $\boldsymbol{R}=(r_1,\cdots,r_n)=\boldsymbol{E}+\boldsymbol{G}$, where n is the length of a component codeword and $\boldsymbol{G}$ represents the AWGN. Transmitting block Turbo codes using higher order modulation schemes such as QAM has been considered in [78, 79] using a pragmatic approach[9] and in [78, 79] using a multilevel code.

As in [90], a two-dimensional block Turbo code with extended BCH component codes is transmitted. The iterative decoder will be considered as a pipeline of SISO decoders. The q^{th} decoding stage in the pipeline is shown in Fig. 2.8. There are two decodings per iteration, a row and a column decoding.

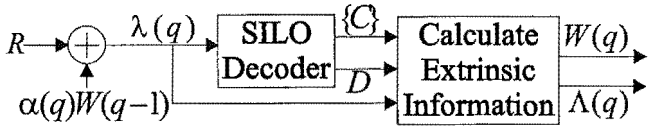


Figure 2.8: The q^{th} decoding stage.

The soft output of the q^{th} decoder corresponding to the j^{th} bit position is defined in [88] as

$$\begin{aligned}\Lambda'_j(q) &= \log\left(\frac{P(e_j=+1|\boldsymbol{\lambda}(q))}{P(e_j=-1|\boldsymbol{\lambda}(q))}\right)\\ &= \log\left(\frac{\sum_{\{\boldsymbol{C}^a|e_j=+1\}}P(\boldsymbol{C}^a|\boldsymbol{\lambda}(q))}{\sum_{\{\boldsymbol{C}^b|e_j=-1\}}P(\boldsymbol{C}^b|\boldsymbol{\lambda}(q))}\right)\end{aligned} \tag{2.24}$$

where $\boldsymbol{\lambda}(q)=(\lambda_1(q),\cdots,\lambda_n(q))$ is the vector of soft inputs to the q^{th} decoder for the current codeword. $\{\boldsymbol{C}^a|e_j=+1\}$ and $\{\boldsymbol{C}^b|e_j=-1\}$ represent the sets of all codewords, $\boldsymbol{C}^a$ and $\boldsymbol{C}^b$, with $e_j=+1$ and $e_j=-1$ respectively. The sub-optimal decoder approximates this by considering only a subset of all possible codewords.

[9]The pragmatic approach maps the bits in the block Turbo code to the QAM constellation using gray mapping. When gray mapping is used the labels of the constellation points at minimum Euclidean distance from each other only differ in one position.

The subset of codewords considered is generated by the SILO Chase decoder[10] described in section 2.2. However, other SILO decoders could be used, as shown in [105].

Equiprobable codewords are assumed and the normalized soft output is obtained by dividing through by $2/\sigma^2_{\boldsymbol{\lambda}(q)}$, where $\sigma^2_{\boldsymbol{\lambda}(q)}$ is the variance of $\boldsymbol{\lambda}(q)$. Using the approximation $\log(e^x + e^y) \approx \max(x, y)$ [43] and some algebra, the soft output can be approximated by

$$\Lambda_j(q) = d_j \left(\frac{|\boldsymbol{\lambda}(q) - \boldsymbol{C}|^2 - |\boldsymbol{\lambda}(q) - \boldsymbol{D}|^2}{4} \right) \approx \Lambda'_j(q) \tag{2.25}$$

where $\boldsymbol{D} = (d_1, \cdots, d_n)$ is the codeword from the Chase decoder which is closest to the soft input in terms of squared Euclidean distance and the competing codeword, $\boldsymbol{C} = (c_1, \cdots, c_n)$, is the closest codeword in $\{\boldsymbol{C}\}$ with $c_j \neq d_j$. After some algebra the soft output of (2.25) can be expressed as

$$\Lambda_j(q) = d_j \sum_{l=1}^{n} \frac{\lambda_l(q)}{2}(d_l - c_l) = d_j \sum_{l=1, d_l \neq c_l}^{n} d_l \lambda_l(q) \tag{2.26}$$

The soft input in the j^{th} bit position to the q^{th} decoder in the pipeline is defined as [88, 90]

$$\lambda_j(q) = r_j + \alpha(q) w_j(q-1) \tag{2.27}$$

where $\boldsymbol{\alpha} = [0, 0.2, 0.3, 0.5, 0.7, 0.9, 1, 1]$ for a 4-iteration simulation [90]. The sequence $\boldsymbol{\alpha}$ is used to scale the extrinsic information to account for the difference in standard deviation between the received signal and extrinsic information [90].

Using (2.20) and (2.26) the unnormalised extrinsic information for the j^{th} bit position from the q^{th} decoder is given by

$$w_j(q) = \Lambda_j(q) - \lambda_j(q) = d_j \sum_{l=1, l \neq j, d_l \neq c_l}^{n} d_l \lambda_l(q) \tag{2.28}$$

All extrinsic information calculated using (2.28) in the current block Turbo code block is normalized/ divided by the arithmetic mean of the absolute value of all extrinsic information calculated using (2.28) in the current block Turbo code block. This will be called the normalized extrinsic information. This normalization is used to reduce the dependence of $\alpha(q)$ on the specific block Turbo code [90] and modulation [79] used.

[10]The $p \geq \lfloor d_{H,min}/2 \rfloor$ LRPs are used to produce 2^p test sequences [90]. In [79, 80, 89] less than 2^p of these test sequences are actually used.

The extrinsic information can be written [79, 87] in the form

$$w_j(q) = d_j\beta(q) \tag{2.29}$$

Comparing this to (2.28) $\beta(q)$ for the j^{th} position emanating from the q^{th} decoder can be defined as

$$\beta_j(q) = \sum_{l=1, l\neq j, d_l \neq c_l}^{n} d_l\lambda_l(q) \tag{2.30}$$

To calculate this directly a competing codeword, $\boldsymbol{C}$, is required from the list of codewords produced by the Chase decoder, $\{\boldsymbol{C}\}$. Unfortunately, competing codewords may not be found for all bit positions if a computationally manageable number of error patterns are used in the Chase decoder. If no competing codeword is available for the j^{th} bit position, then [90] uses a precomputed set of values for $\beta(q)$ defined as $\boldsymbol{\beta}$=[0.2, 0.4, 0.6,0.8, 1, 1, 1, 1] for a 4-iteration simulation. By allowing a position not to have a competing codeword the number of error patterns required in the Chase decoder (and therefore the complexity) is reduced. It should be noted that if an extended block code is used, then the overall parity bit uses the precomputed value of $\beta(q)$.

A drawback to using the precomputed sequences $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ is that they need to be re-optimized using repeated simulations if used in a new application or with a different code or modulation scheme for optimum results. Also they do not adapt to varying channel conditions, and good and bad received blocks are treated the same. This can be avoided if methods to adaptively calculate or estimate $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are developed.

2.3.2 Adaptive Reduced-Complexity SISO Decoders

In this subsection some adaptive approaches to calculating and estimating $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are summarized. In [21] $\boldsymbol{\alpha}$ was derived for SISO decoders using the BCJR algorithm or the SOVA by treating the extrinsic information as a Gaussian random variable. It can be written as

$$\alpha(q) = \mu_{\boldsymbol{W}(q-1)} \frac{\sigma_{\boldsymbol{R}}^2}{\sigma_{\boldsymbol{W}(q-1)}^2} \tag{2.31}$$

where $\sigma_{\boldsymbol{R}}^2$ is the variance of the Gaussian noise, and $\mu_{\boldsymbol{W}(q-1)}$ is the mean and $\sigma_{\boldsymbol{W}(q-1)}^2$ the variance of $\boldsymbol{W}(q-1)$ for the current data block.

In [80] $\boldsymbol{\alpha}$ was derived for the non-adaptive SISO decoder of the previous subsection by treating the extrinsic information as a Gaussian random variable and is given by

$$\alpha(q) = \frac{\sigma^2_{\boldsymbol{R}}}{\sigma^2_{\boldsymbol{W}(q-1)}} \tag{2.32}$$

where $\sigma^2_{\boldsymbol{W}(q-1)}$ is the variance of the matrix of extrinsic information from the current block Turbo code block and $\sigma^2_{\boldsymbol{R}}$ is the variance of the Gaussian noise [80]. However, this did not perform as well as the precomputed approach of [90] and so (2.32) was multiplied by the decoding stage number, q, to give

$$\alpha(q) = \frac{q\sigma^2_{\boldsymbol{R}}}{\sigma^2_{\boldsymbol{W}(q-1)}} \tag{2.33}$$

A potential problem with the $\alpha(q)$ of (2.32) and (2.33) is that as the decoder tends to a decision, they tend to infinity. This is due to $\sigma^2_{\boldsymbol{W}(q-1)}$ tending to zero. This problem can be avoided by using a termination criterion. In [80] decoding was terminated when $\sigma^2_{\boldsymbol{W}(q-1)} < 0.1$ and $\alpha(q) > 100$.

In [80] the following precomputed values of $\beta(q)$ were used

$$\beta(q) = \frac{q}{Q_{max}}, \qquad q = 1, \cdots, Q_{max} \tag{2.34}$$

where Q_{max} is the maximum number of decodings performed. This value of $\beta(q)$ is not adaptive as it is the same for all *signal to noise ratios (SNRs)* and for all received blocks. However, it does not need to be optimized using repeated simulations. The performance when using (2.33) and (2.34) is better[11] than when the precomputed values of [90] are used. The most recent approaches to adaptively estimating $\beta(q)$ are discussed in [1, 26].

In [80] performance was improved by using an adaptively estimated $\alpha(q)$ with a linear scaling function. This shows the promise of an adaptive approach. In chapter 3, $\boldsymbol{\alpha}$ is derived for a sub-optimal reduced-complexity SISO decoder and two new approaches to adaptively estimating $\boldsymbol{\beta}$ are developed. One approach to adaptively estimating $\boldsymbol{\beta}$ is based on approximation and the other is based on the distance properties of the component codes.

[11] The performance is better than [90] when the BER is less than 10^{-2} in the example given in [80].

2.4 Multilevel Coding and Multistage Decoding

Multilevel coding [16, 52, 82, 119] is a technique which allows a complex code to be created as a hierarchy of simple component codes. Multilevel coding combines error correction coding and modulation by using the component codes to choose the transmitted constellation points, usually by means of a partitioning process. Any type of code can be used as a component code [82], including the block Turbo codes described in section 2.3 [79]. They are normally decoded using a MSD as decoding the entire multilevel code is usually prohibitively complex. Performance can be improved by using an iterative MSD.

In this section the design and construction of multilevel codes are described. Then an MSD and several iterative MSDs are described.

2.4.1 Multilevel Encoder, Code Design and Partitioning

An L-level multilevel code consists of L independent component codes. The encoded output from each level is used to choose the constellation point to be transmitted as shown in Fig. 2.9. Partitioning is often used to define the mapping from the encoder outputs to the signal constellation.

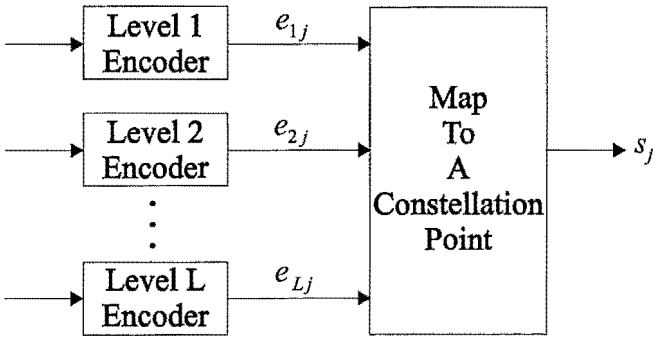


Figure 2.9: L-level multilevel encoder.

The partitioning procedure starts with a constellation of M points. This constellation is partitioned into m_1 equally sized sets of M_1 points on the first level. Each subset of M_1 points is labelled by $\log_2(m_1)$ bits. The l^{th} level partitions each subset of M_{l-1} points chosen on level $l-1$ into m_l equally sized subsets of M_l points.

Each subset of M_l points is labelled by $\sum_{i=1}^{l} \log_2(m_i)$ bits. Different numbers of bits can be used to partition the subsets of points on different levels [16, 82].

A variety of different partitioning strategies have been suggested in [17, 27, 115, 119]. The most commonly used partitioning scheme is Ungerboeck set partitioning [52, 115] which tries to maximize the minimum intra-subset Euclidean distance [119] and results in increasing Euclidean distance at each partition level. It will be used in this thesis. An example of Ungerboeck set partitioning is given in Fig. 2.10 for 16-QAM, $L = 3$, $m_1 = 4$, $m_2 = 2$ and $m_3 = 2$.

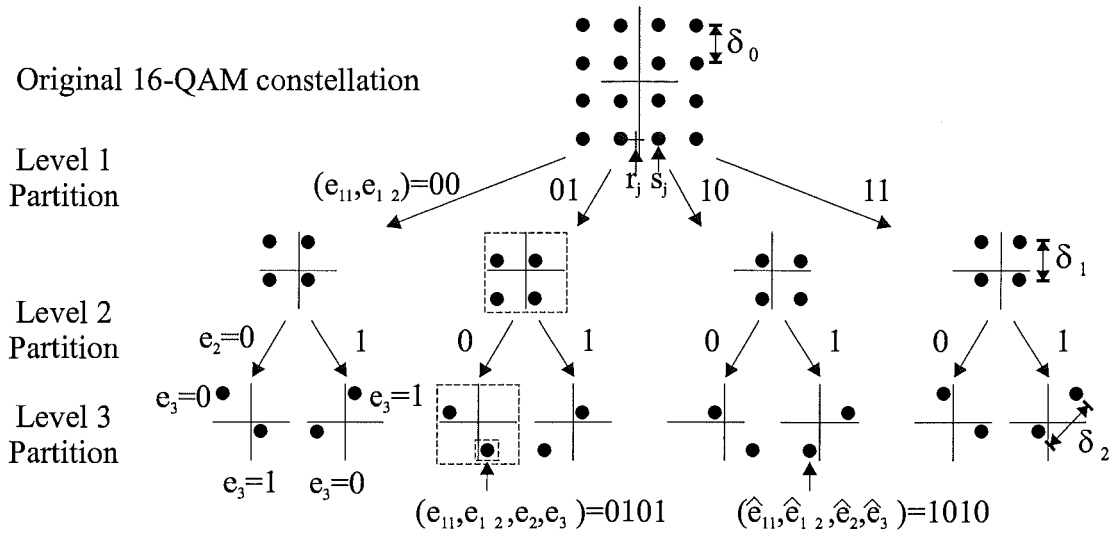


Figure 2.10: Ungerboeck set partitioning of 16-QAM for a $L = 3$ level multilevel code with $m_1 = 4$, $m_2 = 2$ and $m_3 = 2$. The j^{th} bit in the level i label is given by e_{ij} (j is excluded if only one bit is used on level i). The level 3 partition results in a single point being chosen. If $e_3 = 0$ then the point on the left of the vertical axis is chosen, otherwise $e_3 = 1$ and a point to the right is chosen.

The minimum squared Euclidean distance of a multilevel code is given by [48]

$$d^2_{E,min} = \min(d_{H,1}\delta_0^2, d_{H,2}\delta_1^2, \cdots, d_{H,L}\delta_{L-1}^2) \tag{2.35}$$

where $d_{H,l}$ ($l = 1, \cdots, L$) is the minimum Hamming distance of the level l component code and δ_i^2 ($i = 0, 1, \cdots, L-1$) is the minimum squared Euclidean distance between the subsets on level $i + 1$ (and is called the minimum intra-subset distance) as shown in Fig. 2.10. Ungerboeck set partitioning results in $\delta_0^2 \leq \delta_1^2 \leq \cdots \leq \delta_{L-1}^2$ [48]. Therefore, the most powerful code is required on the first level. At high

SNR performance is dominated by the level with the minimum squared Euclidean distance, $d_{H,i}\delta_{i-1}^2$.

Many criteria for choosing the component codes in a multilevel code have been proposed. Some are based on distance [16, 52, 82, 119], capacity (and rate) [28, 50, 57, 119], cutoff rate [119] or the coding exponent [119]. The capacity based approach chooses the code rates of the component codes to be equal to the equivalent capacity[12] of that partition level [119]. Multilevel coding with multistage decoding approaches capacity if the rate of each component code is chosen to equal the equivalent capacity at that partition level [119].

Traditionally the balanced distance decoding rule was used to select the component codes in a multilevel code [119]. The component codes are chosen so that the minimum squared Euclidean distances on all levels are equal, resulting in

$$d_{E,min}^2 = d_{H,1}\delta_0^2 = d_{H,2}\delta_1^2 = \cdots = d_{H,L}\delta_{L-1}^2 \tag{2.36}$$

This design rule is used in this thesis as in the work of [79].

Now that the partitioning and component code design has been discussed, the multilevel encoder shown in Fig. 2.9 can be explained in more detail. The "map to a constellation point" block is performed in a similar way to the partitioning, but only one subset of constellation points is retained at each level. Consider the $L = 3$ level multilevel code of Fig. 2.10. First $log_2(m_1) = 2$ encoded bits from level 1, $(e_{11}, e_{12}) = 01$, are used to choose a subset of $M_1 = 4$ constellation points. Then $log_2(m_2) = 1$ encoded bit from level 2, $e_2 = 0$, is used to choose a subset of $M_2 = 2$ constellation points. Finally, $log_2(m_3) = 1$ encoded bit from level 3, $e_3 = 1$, is used to choose the transmitted constellation point. The transmitted constellation point is labelled by $(e_{11}, e_{12}, e_2, e_3) = 0101$ as shown in Fig. 2.10.

2.4.2 Multistage Decoder

Multilevel codes are usually decoded in a sequential manner using a MSD as shown in Fig. 2.11. This is used when decoding the overall multilevel code is prohibitively complex. The MSD decodes the component codes in the multilevel code in the same

[12]Equivalent capacity is the capacity of a given level within a multilevel code, when each level is regarded as an equivalent subchannel [119].

order as they were encoded. First the level-1 code is decoded to choose a subset of M_1 constellation points for the level-2 decoder. Then the level-2 code is decoded to choose a subset of M_2 constellation points for the level-3 decoder. This pattern continues on all subsequent levels. The subset of points is chosen by the decoder on each level by using hard decisions on the constellation point label from previous decoders and by making a hard decision on its part of the constellation point label. The hard decision is passed to all subsequent decoders/ levels. If there are errors in a level's decision, then these are passed to subsequent levels causing error propagation.

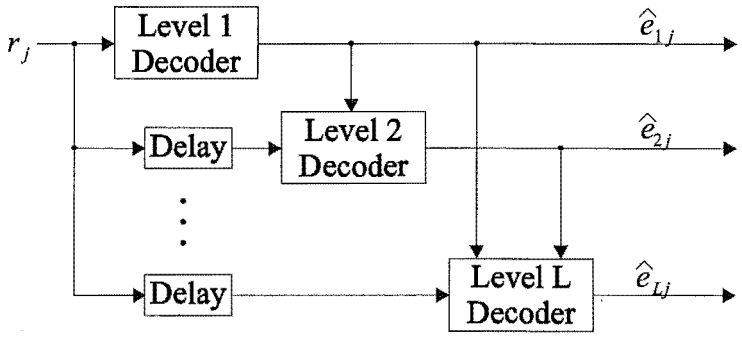


Figure 2.11: L-level multistage decoder, where $(\hat{e}_{1j}, \cdots, \hat{e}_{Lj})$ are the set of estimates of the j^{th} transmitted bit from all the component decoders (assuming binary partitions on all levels).

Consider the $L = 3$ level example of Fig. 2.10. The decoding process is performed in a similar way to encoding. The level 1 decoder considers the entire $(M = 16)$-QAM constellation. Then $log_2(m_1) = 2$ decoded bits (including parity bits) from the level 1 decoder, $(\hat{e}_{11}, \hat{e}_{12}) = 10$, are used to choose a subset of $M_1 = 4$ constellation points. The level 2 decoder operates as if this subset of M_1 constellation points is the entire constellation. The $log_2(m_2) = 1$ decoded bit from the level 2 decoder, $\hat{e}_2 = 1$, is used to choose a subset of $M_2 = 2$ constellation points for the level 3 decoder. The level 3 decoder operates as if this subset of M_2 constellation points is the entire constellation. Finally, $log_2(m_3) = 1$ decoded bit from the level 3 decoder, $\hat{e}_3 = 0$, is used to choose the decoded constellation point. The transmitted constellation point is labelled by $(e_{11}, e_{12}, e_2, e_3) = 0101$ as shown in Fig. 2.10, but the decoded constellation point is labelled by $(\hat{e}_{11}, \hat{e}_{12}, \hat{e}_2, \hat{e}_3) = 1010$. Although the decoded constellation point is at the minimum Euclidean distance, δ_0, from the

transmitted constellation point, s_j, it is the maximum Hamming distance away from it, which means all decoded bits are incorrect. The error on level-1 resulted in the actual transmitted constellation point being excluded from the subsets of points considered on later levels and so later levels were decoded in error. This effect is called error propagation.

2.4.3 Iterative Multistage Decoder

In an attempt to minimize the error propagation effect, soft information can be passed to subsequent levels instead of hard decisions. Performance can also be improved by passing soft information to previous levels and iterating. This is called iterative multistage decoding.

An early approach to iterative multistage decoding is given in [121]. It uses interleaving between the encoded bits from each level. Reliability information from previous levels is used in the metric calculations and the trellis-based algorithm of [31] is used. More than one pass through the MSD is used. After the first pass through the MSD (first iteration) the level 1 decoder looks for the codeword which maximizes $p(r|u^1, \hat{u}^2, \hat{u}^3, \cdots)$, where u^i represents the information bits on level i. No definition of $\hat{u}^i$ is given, it is either a hard decision on the level-i information bits or reliability information on them. This scheme provides a significant performance improvement over traditional multistage decoding.

Another approach to iterative multistage decoding is given in [27]. An interleaver is used after the encoder (but before the mapping to constellation points) on every level other than the first level. The SOVA is used to decode the component codes. The soft (reliability) information produced is passed to all subsequent levels, but only to some previous levels (specifically from level 2 to level 1). It is treated as a priori information. Anti-Ungerboeck mapping[13] is used, which results in the final (third) level requiring the strongest code.

A similar approach is used in [73, 74], where the iterative MSD is used for a hierarchical television terrestrial broadcast systems (and multi-resolution modulation). A six level multilevel code is used with interleaving on every second level. Soft information is passed to all subsequent levels, but only every second level passes soft

[13] Anti-Ungerboeck partitioning is designed for decreasing distance between subsets, δ_i^2 [27].

information back to the previous level[14]. Again the reliability information is treated as a priori information.

In [42] a multilevel code is decoded using an analog non-linear network. Soft information from a level is used by other levels as a priori information. This scheme does not have conventional iterations.

The most recent work on iterative multistage decoding is discussed in [53, 71]. In [53] the reliability information is used as a priori information. The iterative MSD uses belief propagation to update the bit-wise metric. Parallel and serial decoding approaches are proposed. A priori information from all other levels is used. In [71] the soft output from the component decoders on all other levels are used as a priori information in the soft output demodulator. The three level simulations in [71] only used iterative multistage decoding between level one and two. Level three was decoded once, at the end of decoding.

In [77] block Turbo codes are used in a multilevel code, decoded with an iterative MSD. The iterative MSD of [77] passes hard decisions to all subsequent levels and back to level one. This hard iterative MSD was found to perform better than the traditional hard MSD at low SNRs in [77], but the slope of the curve is not as steep and so it should cross the curve of the traditional approach somewhere below 10^{-5}.

In summary the iterative MSDs of [27, 73, 74, 121] use interleaving between levels to combat error propagation. In addition the approaches of [27, 42, 53, 71, 73, 74] use the reliability information as a priori information. The reliability information is passed to all subsequent levels and some previous levels in [27, 73, 74]. All of these papers show the promise of soft iterative multistage decoding.

In chapter 4 of this thesis a new approach to iterative multistage decoding is developed which treats the reliability (extrinsic) information as a Gaussian random variable rather than as a priori information. In addition the extrinsic information is sent to all previous and subsequent decoding levels. The new iterative MSD is investigated with and without interleaving between levels.

Trellis based SISO decoding algorithms are used in [27, 73, 74, 121] to provide reliability information. These algorithms can be prohibitively complex for decoding

[14]Level m passes soft information back to level $m-1$, where m is an even integer. Level m does not pass the soft information back to any other levels, but passes it to all subsequent levels.

long block codes (as considered in this thesis). Alternative SISO decoding algorithms based on list decoding are considered in section 2.3, chapter 3 and chapter 4 of this thesis.

2.5 Summary

This chapter has presented relevant background information on error correction coding and multilevel coding. BCH codes were discussed in section 2.2 and were used in block Turbo codes in section 2.3. Finally multilevel codes and multistage decoding were introduced in section 2.4. A brief literature review of these areas has also been included.

Chapter 3

Adaptive Iterative Decoding of Block Turbo Codes

3.1 Introduction

Soft-input soft-output (SISO) decoding long block codes using a trellis can be prohibitively complex[1]. This is especially true if they are used as component codes in a concatenated coding scheme such as the block Turbo codes discussed in this chapter. In such cases reduced-complexity SISO decoding algorithms are required. There is a large number of reduced complexity decoding algorithms available. This chapter will consider sub-optimal list based decoding algorithms. There are many sub-optimal soft input decoding algorithms which can produce a list of possible codewords and a decision codeword [19, 26, 30, 33, 34, 35, 36, 40, 46, 55, 56, 99, 105, 106, 109, 110]. If these *soft-input list-output (SILO)* decoding algorithms are to be used in an iterative decoding context they must produce soft information. One technique for doing this is discussed in [54, 88, 90] and was described in section 2.3. The structure of this decoder is shown in Fig. 2.8.

Recall that in [54, 88, 90] two sequences of precomputed parameters, α and β, are used when calculating and using the extrinsic information[2]. These sequences need to be reoptimised for different codes or applications using repeated simulations

[1]The SISO decoding of low-density parity check codes is not trellis based.

[2]The sequence α scales the extrinsic information to compensate for the difference in variance between the received signal and the extrinsic information. The sequence β is used in estimating the extrinsic information when it cannot be calculated directly.

for optimum performance and do not adapt with varying channel conditions. The goal of this chapter is to develop adaptive approaches to either calculating or estimating α and β, which do not need to be optimized using repeated simulations and can be used with any code or in different applications. In section 3.2 the theoretical value for α is derived. This α is adaptively estimated in simulations on a block by block basis. It does not have to be reoptimised using repeated simulations and can adapt to varying channel conditions.

In section 3.3 two approaches to adaptively estimating β are developed, one based on approximation and a second based on the distance properties of the component codes. Both approaches are adaptively estimated for each bit, codeword or block Turbo code block and so can adapt to varying channel conditions. These approaches to estimating β do not need to be reoptimised using repeated simulations.

By adaptively estimating α and β instead of using the precomputed values of [54, 88, 90], performance can be improved. The cost is a small increase in algorithm complexity. Both approaches avoid the need to reoptimise α and β using repeated simulations for different codes or applications and they can adapt to varying channel conditions.

The block Turbo codes considered are transmitted on the *additive white Gaussian noise (AWGN)* channel[3] using BPSK/ QPSK. Simulation results are presented in section 3.4 and conclusions are drawn in section 3.5.

3.2 Derivation of α

In this section α is derived by treating the extrinsic information as a Gaussian random variable. The *log-likelihood ratio (LLR)* is conditioned on both the received signal and extrinsic information from the previous decoding stage for the current component codeword. The soft output from the q^{th} decoder for the j^{th} bit position is given by the LLR

$$\begin{aligned}\Lambda'_j(q) &= \log\left(\frac{P(e_j=+1|\boldsymbol{R},\boldsymbol{W}(q-1))}{P(e_j=-1|\boldsymbol{R},\boldsymbol{W}(q-1))}\right)\\ &= \log\left(\frac{\sum_{\{\boldsymbol{C}^a|e_j=+1\}}P(\boldsymbol{C}^a|\boldsymbol{R},\boldsymbol{W}(q-1))}{\sum_{\{\boldsymbol{C}^b|e_j=-1\}}P(\boldsymbol{C}^b|\boldsymbol{R},\boldsymbol{W}(q-1))}\right)\end{aligned} \tag{3.1}$$

[3]The AWGN samples are assumed to be independent.

where e_j is the j^{th} bit[4] in a codeword, $\boldsymbol{W}(q-1) = (w_1(q-1), \cdots, w_n(q-1))$ is the extrinsic information vector from the previous decoding for the current codeword, $\boldsymbol{R} = (r_1, \cdots, r_n)$ is the received vector from the memoryless AWGN channel and n is the length of a component codeword. $\{\boldsymbol{C}^a | e_j = +1\}$ and $\{\boldsymbol{C}^b | e_j = -1\}$ represent the sets of all codewords, $\boldsymbol{C}^a$ and $\boldsymbol{C}^b$, with $e_j = +1$ and $e_j = -1$ respectively. This can be simplified by assuming equiprobable codewords and that the received vector is independent of the extrinsic information. In addition it is assumed that the extrinsic information can be modelled as a Gaussian random variable as in [14]. Then (3.1) can be written as

$$\Lambda'_j(q) = \log\left(\frac{\sum_{\{\boldsymbol{C}^a | e_j=+1\}} p(\boldsymbol{R}|\boldsymbol{C}^a)p(\boldsymbol{W}(q-1)|\boldsymbol{C}^a)}{\sum_{\{\boldsymbol{C}^b | e_j=-1\}} p(\boldsymbol{R}|\boldsymbol{C}^b)p(\boldsymbol{W}(q-1)|\boldsymbol{C}^b)}\right) \tag{3.2}$$

The sub-optimal decoder approximates this by considering only a subset of all possible codewords. The subset of codewords is chosen by a SILO decoder. The subset of codewords in set $\{\boldsymbol{C}^a | e_j = +1\}$ or $\{\boldsymbol{C}^b | e_j = -1\}$ is denoted $\{\boldsymbol{C}^a | e_j = +1\}_C$ or $\{\boldsymbol{C}^b | e_j = -1\}_C$ respectively. Using the approximation $\log(e^x + e^y) \approx \max(x, y)$ [43] the soft output of (3.2) may be approximated as

$$\begin{aligned}\Lambda'_j(q) \approx &\max_{\{\boldsymbol{C}^a | e_j=+1\}_C} \{\log(p(\boldsymbol{R}|\boldsymbol{C}^a)p(\boldsymbol{W}(q-1)|\boldsymbol{C}^a))\} \\ - &\max_{\{\boldsymbol{C}^b | e_j=-1\}_C} \{\log(p(\boldsymbol{R}|\boldsymbol{C}^b)p(\boldsymbol{W}(q-1)|\boldsymbol{C}^b))\} = \Lambda''_j(q)\end{aligned} \tag{3.3}$$

Denote $\boldsymbol{D}$ as the codeword found in (3.3) which has the maximum metric and $\boldsymbol{C}$ as the other codeword found in (3.3) (it has $c_j \neq d_j$). Now (3.3) can be written as

$$\begin{aligned}\Lambda''_j(q) = d_j \Bigg(&\frac{|\boldsymbol{R} - \mu_{\boldsymbol{R}|\boldsymbol{C}}|^2}{2\sigma^2_{\boldsymbol{R}|\boldsymbol{C}}} - \frac{|\boldsymbol{R} - \mu_{\boldsymbol{R}|\boldsymbol{D}}|^2}{2\sigma^2_{\boldsymbol{R}|\boldsymbol{D}}} \\ &+ \frac{|\boldsymbol{W}(q-1) - \mu_{\boldsymbol{W}(q-1)|\boldsymbol{C}}|^2}{2\sigma^2_{\boldsymbol{W}(q-1)|\boldsymbol{C}}} - \frac{|\boldsymbol{W}(q-1) - \mu_{\boldsymbol{W}(q-1)|\boldsymbol{D}}|^2}{2\sigma^2_{\boldsymbol{W}(q-1)|\boldsymbol{D}}} \Bigg)\end{aligned} \tag{3.4}$$

where $\mu_{\boldsymbol{W}(q-1)|\boldsymbol{D}}$ is the mean and $\sigma^2_{\boldsymbol{W}(q-1)|\boldsymbol{D}}$ the variance of $\boldsymbol{W}(q-1)$ conditioned on $\boldsymbol{D}$, and $\mu_{\boldsymbol{R}|\boldsymbol{D}}$ is the mean and $\sigma^2_{\boldsymbol{R}|\boldsymbol{D}}$ the variance of $\boldsymbol{R}$ conditioned on $\boldsymbol{D}$. Similarly $\mu_{\boldsymbol{W}(q-1)|\boldsymbol{C}}$ is the mean and $\sigma^2_{\boldsymbol{W}(q-1)|\boldsymbol{C}}$ the variance of $\boldsymbol{W}(q-1)$ conditioned on $\boldsymbol{C}$, and $\mu_{\boldsymbol{R}|\boldsymbol{C}}$ is the mean and $\sigma^2_{\boldsymbol{R}|\boldsymbol{C}}$ the variance of $\boldsymbol{R}$ conditioned on $\boldsymbol{C}$. In practice the

[4]For the purposes of computing likelihoods all bits are mapped from $\{0, 1\}$ to $\{-1, +1\}$.

following is used

$$\Lambda_j''(q) = d_j \frac{|\boldsymbol{R}-\boldsymbol{C}|^2 - |\boldsymbol{R}-\boldsymbol{D}|^2}{2\sigma_{\boldsymbol{R}}^2} + d_j \frac{|\boldsymbol{W}(q-1) - \mu_{\boldsymbol{W}(q-1)}\boldsymbol{C}|^2 - |\boldsymbol{W}(q-1) - \mu_{\boldsymbol{W}(q-1)}\boldsymbol{D}|^2}{2\sigma_{\boldsymbol{W}(q-1)}^2} \tag{3.5}$$

where $\mu_{\boldsymbol{W}(q-1)}$ is the mean and $\sigma_{\boldsymbol{W}(q-1)}^2$ the variance of the absolute value of $\boldsymbol{W}(q-1)$, and $\sigma_{\boldsymbol{R}}^2$ is the variance of the absolute value of $\boldsymbol{R}$ as used in [14]. A small improvement may be possible if the variance of the AWGN noise, $\sigma_{\boldsymbol{R}}^2$, is known exactly. The variances, $\sigma_{\boldsymbol{W}(q-1)}^2$ and $\sigma_{\boldsymbol{R}}^2$, and the mean, $\mu_{\boldsymbol{W}(q-1)}$, are estimated over a block Turbo code block after each component code is decoded, using the sample variances and sample mean respectively. The sample variance of the extrinsic information can be written as

$$\sigma_{\boldsymbol{W}(q-1)}^2 = \frac{1}{N-1}\sum_{l=1}^{N}\left(w_l(q-1) - \mu_{\boldsymbol{W}(q-1)}\right)^2 \tag{3.6}$$

where

$$\mu_{\boldsymbol{W}(q-1)} = \frac{1}{N}\sum_{l=1}^{N}|w_l| \tag{3.7}$$

and N is the number of bits in a block Turbo code block. The extrinsic information used here has not been normalized by its mean.

The soft output simplifies to

$$\Lambda_j''(q) = d_j \sum_{l=1, d_l \neq c_l}^{n} d_l \left(\frac{2}{\sigma_{\boldsymbol{R}}^2} r_l + \frac{2\mu_{\boldsymbol{W}(q-1)}}{\sigma_{\boldsymbol{W}(q-1)}^2} w_l(q-1)\right) \tag{3.8}$$

As in [90] $\Lambda_j''(q)$ is normalized. Here (3.8) is divided by $2/\sigma_{\boldsymbol{R}}^2$ giving a normalized soft output of

$$\Lambda_j(q) = d_j \sum_{l=1, d_l \neq c_l}^{n} d_l \left(r_l + \mu_{\boldsymbol{W}(q-1)} \frac{\sigma_{\boldsymbol{R}}^2}{\sigma_{\boldsymbol{W}(q-1)}^2} w_l(q-1)\right) \tag{3.9}$$

Now consider the LLR for the soft input to the q^{th} decoder, which is defined as

$$\lambda_j'(q) = \log\left(\frac{P(e_j = +1 | r_j, w_j(q-1))}{P(e_j = -1 | r_j, w_j(q-1))}\right) \tag{3.10}$$

Assuming equiprobable bit values, this can be written as

$$\lambda_j'(q) = \log\left(\frac{p(r_j | e_j = +1) p(w_j(q-1) | e_j = +1)}{p(r_j | e_j = -1) p(w_j(q-1) | e_j = -1)}\right) \tag{3.11}$$

and in practise will be calculated using

$$\lambda'_j(q) = \frac{2}{\sigma^2_{\boldsymbol{R}}} r_j + \frac{2\mu_{\boldsymbol{W}(q-1)}}{\sigma^2_{\boldsymbol{W}(q-1)}} w_j(q-1) \tag{3.12}$$

Now divide by $2/\sigma^2_{\boldsymbol{R}}$ to obtain

$$\lambda_j(q) = r_j + \mu_{\boldsymbol{W}(q-1)} \frac{\sigma^2_{\boldsymbol{R}}}{\sigma^2_{\boldsymbol{W}(q-1)}} w_j(q-1) \tag{3.13}$$

Comparing (3.9) and (3.13), the soft output can be written as a sum of soft inputs as

$$\Lambda_j(q) = d_j \sum_{l=1}^{n} \frac{\lambda_l(q)}{2}(d_l - c_l) = d_j \sum_{l=1, d_l \neq c_l}^{n} d_l \lambda_l(q) \tag{3.14}$$

In [90] the soft input is given as

$$\lambda_j(q) = r_j + \alpha(q) w_j(q-1) \tag{3.15}$$

Comparing (3.13) and (3.15) the value of $\alpha(q)$ can be recognized as

$$\alpha(q) = \mu_{\boldsymbol{W}(q-1)} \frac{\sigma^2_{\boldsymbol{R}}}{\sigma^2_{\boldsymbol{W}(q-1)}} \tag{3.16}$$

This so-called theoretical value of $\boldsymbol{\alpha}$ is estimated for each decoder in the pipeline for every block Turbo code block. Therefore, it will be referred to as the adaptively estimated $\boldsymbol{\alpha}$. It will be so used with both the adaptive approaches to estimating β developed in section 3.3. It can also be used in Turbo decoders as shown in [21, 75] and could be used in SISO order-i reprocessing [34] or in the algorithm of [26]. The $\alpha(q)$ derived here is the same as the initial adaptive $\alpha(q)$ of [80], (2.32), when the extrinsic information is normalized to have a mean absolute value of one.

3.2.1 Scaling and Termination Criteria

This adaptively estimated $\boldsymbol{\alpha}$ and that of [80] tend to infinity as the decoder tends to a decision, since $\sigma^2_{\boldsymbol{W}(q-1)} \to 0$. This can cause decoder overflow problems which can result in a large number of errors being introduced several iterations after the decoder has converged to a decision. This can be avoided by scaling the soft input. To avoid estimating $\alpha(q)$ when $\sigma^2_{\boldsymbol{W}(q-1)} \approx 0$ the soft input is scaled by $\chi(q) = \sigma^2_{\boldsymbol{W}(q-1)}/(\mu_{\boldsymbol{W}(q-1)})^2$ when $\sigma^2_{\boldsymbol{W}(q-1)}/(2\mu_{\boldsymbol{W}(q-1)}) < 1$, giving

$$\lambda'_j(q) = \chi(q)\lambda_j(q) = \chi(q) r_j + \frac{\sigma^2_{\boldsymbol{R}}}{\mu_{\boldsymbol{W}(q-1)}} w_j(q-1) \tag{3.17}$$

The extrinsic information produced using $\lambda'_j(q)$, denoted $w'_j(q)$, is inherently scaled by $\chi(q)$. The mean and variance of the scaled extrinsic information in terms of the mean and variance of the unscaled extrinsic information are given by $\mu_{\boldsymbol{W}'(q)} = \chi(q)\mu_{\boldsymbol{W}(q)}$ and $\sigma^2_{\boldsymbol{W}'(q)} = \chi(q)^2\sigma^2_{\boldsymbol{W}(q)}$ respectively. Therefore, $\alpha(q+1)w_j(q) = \alpha'(q+1)w'_j(q)$. As a result $\alpha'(q+1)$ removes the scaling when $w'_j(q)$ is used in the soft input to the next decoding stage. This means that the scaling factor $\chi(q)$ is only used during the q^{th} decoding stage.

Alternatively, decoding can be terminated when the signs of the soft ($\boldsymbol{\Lambda}$) and hard ($\boldsymbol{D}$) outputs from the last row and column decoding are equal in all positions of the block Turbo code block.

3.3 Adaptive Approaches to Estimating β

In this section two adaptive approaches to estimating β are discussed. But first β is defined. The extrinsic information and hence β can be written in terms of a LLR. The normalized soft output of (3.14) approximates (3.1) divided by $2/\sigma^2_{\boldsymbol{R}}$, meaning

$$\Lambda_j(q) \approx \frac{\sigma^2_{\boldsymbol{R}}}{2} \log\left(\frac{P(e_j = +1|\boldsymbol{R}, \boldsymbol{W}(q-1))}{P(e_j = -1|\boldsymbol{R}, \boldsymbol{W}(q-1))}\right) \tag{3.18}$$

and the soft input of (3.13) estimates (3.10) divided by $2/\sigma^2_{\boldsymbol{R}}$, meaning

$$\lambda_j(q) = \frac{\sigma^2_{\boldsymbol{R}}}{2} \log\left(\frac{P(e_j = +1|r_j, w_j(q-1))}{P(e_j = -1|r_j, w_j(q-1))}\right). \tag{3.19}$$

The extrinsic information for the j^{th} bit position from the q^{th} decoder is given by

$$w_j(q) = \Lambda_j(q) - \lambda_j(q) \tag{3.20}$$

or [79, 87]

$$w_j(q) = d_j\beta(q) \tag{3.21}$$

Using (3.18), (3.19), (3.20) and (3.21) $\beta(q)$ for the j^{th} position from the q^{th} decoder can be approximated by

$$\begin{aligned}\beta_j(q) &\approx \frac{\sigma^2_{\boldsymbol{R}}}{2} \log\left(\frac{P(e_j = d_j|\boldsymbol{R}, \boldsymbol{W}(q-1))}{P(e_j \neq d_j|\boldsymbol{R}, \boldsymbol{W}(q-1))} \frac{P(e_j \neq d_j|r_j, w_j(q-1))}{P(e_j = d_j|r_j, w_j(q-1))}\right) \\ &\approx \frac{\sigma^2_{\boldsymbol{R}}}{2} \log\left(\frac{p(\boldsymbol{R}^{-j}, \boldsymbol{W}^{-j}(q-1)|e_j = d_j, r_j, w_j(q-1))}{p(\boldsymbol{R}^{-j}, \boldsymbol{W}^{-j}(q-1)|e_j \neq d_j, r_j, w_j(q-1))}\right)\end{aligned} \tag{3.22}$$

where $\boldsymbol{R}^{-j} = (r_1, \cdots, r_{j-1}, r_{j+1}, \cdots, r_n)$ and $\boldsymbol{W}^{-j}(q-1) = (w_1(q-1), \cdots, w_{j-1}(q-1), w_{j+1}(q-1), \cdots, w_n(q-1))$. Assuming the received signal and extrinsic information are independent this can be written as

$$\beta_j(q) \approx \frac{\sigma_{\boldsymbol{R}}^2}{2} \log \left(\frac{p(\boldsymbol{R}^{-j}|e_j = d_j, r_j) p(\boldsymbol{W}^{-j}(q-1)|e_j = d_j, w_j(q-1))}{p(\boldsymbol{R}^{-j}|e_j \neq d_j, r_j) p(\boldsymbol{W}^{-j}(q-1)|e_j \neq d_j, w_j(q-1))} \right) \tag{3.23}$$

This is not explicitly dependent on a competing codeword being found. However, no easy way to calculate or estimate it has been found. In addition this definition is only as good as the approximation used to calculate the soft output, namely $\log(e^x + e^y) \approx \max(x, y)$.

But $\beta_j(q)$ can also be written in terms of the soft output calculated using $\log(e^x + e^y) \approx \max(x, y)$. By using (3.3), (3.11), (3.20) and (3.21) it can be written as

$$\begin{aligned} \beta_j(q) = \frac{\sigma_{\boldsymbol{R}}^2}{2} \log \Bigg(& \frac{p(\boldsymbol{R}^{-j}, \boldsymbol{D}^{-j}|e_j = d_j, r_j)}{p(\boldsymbol{R}^{-j}, \boldsymbol{C}^{-j}|e_j \neq d_j, r_j)} \\ & \frac{p(\boldsymbol{W}^{-j}(q-1), \boldsymbol{D}^{-j}|e_j = d_j, w_j(q-1))}{p(\boldsymbol{W}^{-j}(q-1), \boldsymbol{C}^{-j}|e_j \neq d_j, w_j(q-1))} \Bigg) \end{aligned} \tag{3.24}$$

The problem with this definition is that a competing codeword, $\boldsymbol{C}$, is not always found. No easy way to estimate this when no $\boldsymbol{C}$ is found, has been developed.

A simpler definition which also uses the approximation of the soft output can be defined using (3.14), (3.20) and (3.21) as

$$\beta_j(q) = \sum_{l=1, l \neq j, d_l \neq c_l}^{n} d_l \lambda_l(q) \tag{3.25}$$

This can only be calculated when the SILO decoder has found a competing codeword, $\boldsymbol{C}$, for the j^{th} position ($c_j \neq d_j$). The extrinsic information can then be calculated using (3.21). When no competing codeword is available for the j^{th} position, the positions where $c_j \neq d_j$ are not known. In this case an estimate of $\beta_j(q)$ is required. An estimate based on approximation and one based on the distance properties of the component codes are now developed. Although a competitor can always be found if the number of test sequences is large enough, this can result in a high complexity solution, which is not the aim of the present work.

3.3.1 Approximation Based Approach

In this subsection an approximation based approach to adaptively estimating β is developed. There are 4 cases to consider:

1. A competing codeword, $\boldsymbol{C}$, with $c_j \neq d_j$ has been found by the SILO decoder.
2. A competing codeword has been found by the SILO decoder, but $c_j = d_j$.
3. No competing codewords have been found by the SILO decoder. But the decision codeword, $\boldsymbol{D}$, has been found by the SILO decoder.
4. No codewords have been found by the SILO decoder, not even $\boldsymbol{D}$.

In the first case no estimate is required and the extrinsic information can be calculated using the value of $\beta_j(q)$ defined in (3.25). When more than one competing codeword is found for the j^{th} position the competing codeword at the minimum squared Euclidean distance from the soft input is used.

In the second case $\beta_j(q)$ has been calculated using (3.25) for some positions in the current word[5]. If no competing codeword is found for the j^{th} position then the reliability of that bit should be relatively high as all decoded codewords from the SILO decoder agree in that position. However, only a small subset of all possible error patterns, and therefore, codewords are considered by the SILO decoder and so the transmitted codeword may not be included and may have a different value in the j^{th} position. It was found in simulations that when using iterative decoding, it is better to underestimate than overestimate the extrinsic information[6] and hence $\beta_j(q)$, especially in early iterations and at low *signal to noise ratios (SNRs).* In these situations some decodings will correct errors while others will create errors. In order to minimize the number and magnitude of the errors created, the following conservative estimate of the $\beta_j(q)$ of (3.25) is used in case 2,

$$\beta_j(q) = \min_l(\beta_l(q) > 0) \tag{3.26}$$

[5]If a precomputed $\beta_j(q)$ is used as in [79] then the new information about the current codeword from the latest decoding stage is not fully utilized, as only its sign is used.

[6]Overestimation may cause significantly slower average convergence in decoding. If the estimates of $\beta_j(q)$ are too large, then decoding a received block Turbo code block with a large number of errors or badly positioned errors can increase the number of errors and significantly degrade the overall bit error rate.

The minimum is over all positive values of $\beta_l(q)$ calculated using (3.25) for the current component codeword. When $\boldsymbol{D}$ is very reliable this allows $\beta_j(q)$ to be larger than the precomputed values of [79], and therefore, it can correct errors in this component code block more rapidly. When $\boldsymbol{D}$ is not very reliable this allows $\beta_j(q)$ to take a smaller value than the precomputed values of [79] and therefore, it is less likely to create errors. Also the errors will have a smaller value of extrinsic information which makes them easier to correct in later iterations.

In the third case (3.25) cannot be used to calculate $\beta_j(q)$ for any position in the current codeword. When no competing codewords are found, the SILO decoder believes it has found the correct codeword. However, it may actually have found a codeword at the minimum Hamming distance of the code or greater away from the transmitted codeword. The best new information available on whether the decoded bit is correct are the values of $\beta_l(q)$ calculated using (3.25) in other received words in the current block Turbo code block. Since it is important not to overestimate the extrinsic information when using iterative decoding, the following estimate of the $\beta_j(q)$ of (3.25) is used for case 3,

$$\beta_j(q) = \text{mean}_l(\beta_l(q)) \tag{3.27}$$

The mean is taken over all values of $\beta_l(q)$ calculated using (3.25) for any position in the current block Turbo code block. If no $\beta_l(q)$ are able to be calculated using (3.25) for any position in the current block Turbo code block, then the mean value of $\beta_l(q)$ calculated using (3.25) in all block Turbo code blocks to date is used. If this cannot be calculated then the mean of the absolute value of the soft input over the entire block Turbo code block is used.

In the fourth case the extrinsic information can be estimated using (3.27) or $\beta_j(q) = 0$. This case is rare. It does not occur when the unextended BCH component codes are perfect codes (as defined in appendix B). In a perfect code all hard received vectors are at a Hamming distance of t or less from a codeword, where t is the error correction capability of the code. The algebraic decoder can correct t or fewer errors and so at least one codeword can always be found. This means that there is a radius t error correction sphere around each codeword, within which all vectors are decoded to the codeword at its center. In the case of a perfect code all possible binary vectors of length n are in one of the disjoint spheres.

In summary, $\beta_j(q)$ can be written as

$$\beta_j(q) = \begin{cases} \sum_{l=1,l\neq j,d_l\neq c_l}^{n} d_l\lambda_l(q), & \boldsymbol{C} \neq \boldsymbol{D} \text{ found with } c_j \neq d_j \\ \min_l(\beta_l(q) > 0), & \boldsymbol{C} \neq \boldsymbol{D} \text{ found, but } c_j = d_j \\ \text{mean}_l(\beta_l(q)), & \boldsymbol{D} \text{ found, but no } \boldsymbol{C} \neq \boldsymbol{D} \\ \text{mean}_l(\beta_l(q)) \text{ or } 0, & \text{No } \boldsymbol{C} \text{ or } \boldsymbol{D} \text{ found} \end{cases} \tag{3.28}$$

The extrinsic information is calculated using this in (3.21). If an extended block code is being decoded the overall parity bit is treated as if there is no competing codeword.

The importance of a good estimate of $\beta_j(q)$ can be seen by looking at the percentage of positions where the extrinsic information is calculated using an estimate. The following results were obtained using the SILO Chase decoder [19, 90] described in section 2.2. Fig. 3.1 shows the results for the $(64, 57, 4)^2$ block Turbo code with $p = 4$ (and all 16 test sequences being used). The unextended BCH component codes in this block Turbo code, $(63, 57, 3)$, are perfect codes (as defined in appendix B) and so case 4 should never occur. The percentage of extrinsic information cal-

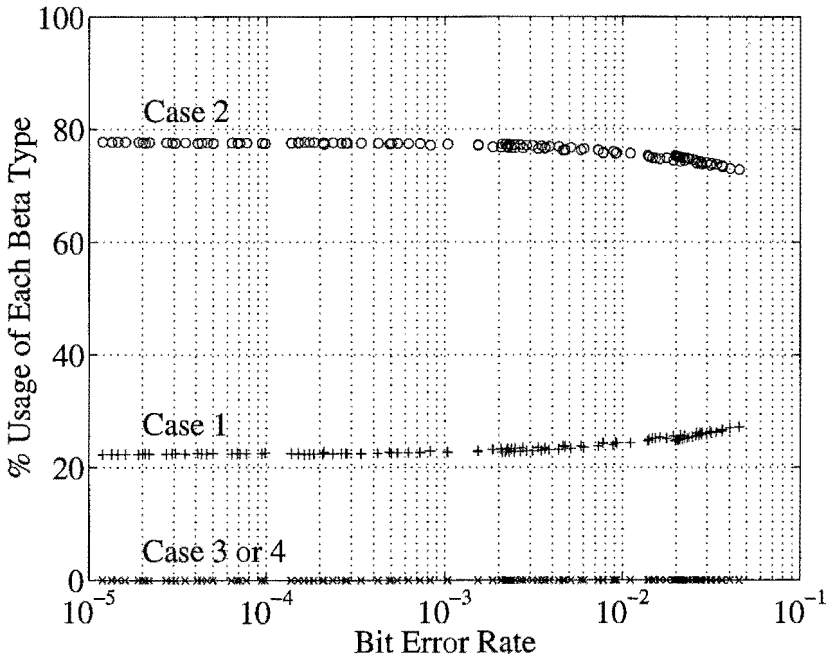


Figure 3.1: Percentage of positions using each type of $\beta_j(q)$ for different bit error rates after decoding. In case 1 $\boldsymbol{C} \neq \boldsymbol{D}$ is found with $c_j \neq d_j$. In case 2 $\boldsymbol{C} \neq \boldsymbol{D}$ is found, but $c_j = d_j$. In case 3 only $\boldsymbol{D}$ is found. In case 4 no codewords are found. Results are for the $(64, 57, 4)^2$ block Turbo code, 2^p test sequences and $p = 4$.

culated using each type of $\beta_j(q)$ is closely related to the *bit error rate (BER)* after decoding, and therefore, the different types of $\beta_j(q)$ are plotted against the BER after decoding rather than against iteration number and SNR. This figure shows that 70-80% of the extrinsic information is calculated using the case 2 $\beta_j(q)$ estimate of (3.26) and the remainder use the case 1 value of (3.25). Since the unextended BCH code is a perfect code and $p = 4$ (and all 2^p test sequences are used), either 8, 12 or 16 distinct codewords are found as shown in Fig. 3.2. Therefore, case 3 and 4 never occur and the estimate of (3.27) is never used. When the BER is high more distinct codewords are found on average and fewer positions use the case 2 estimate of (3.26).

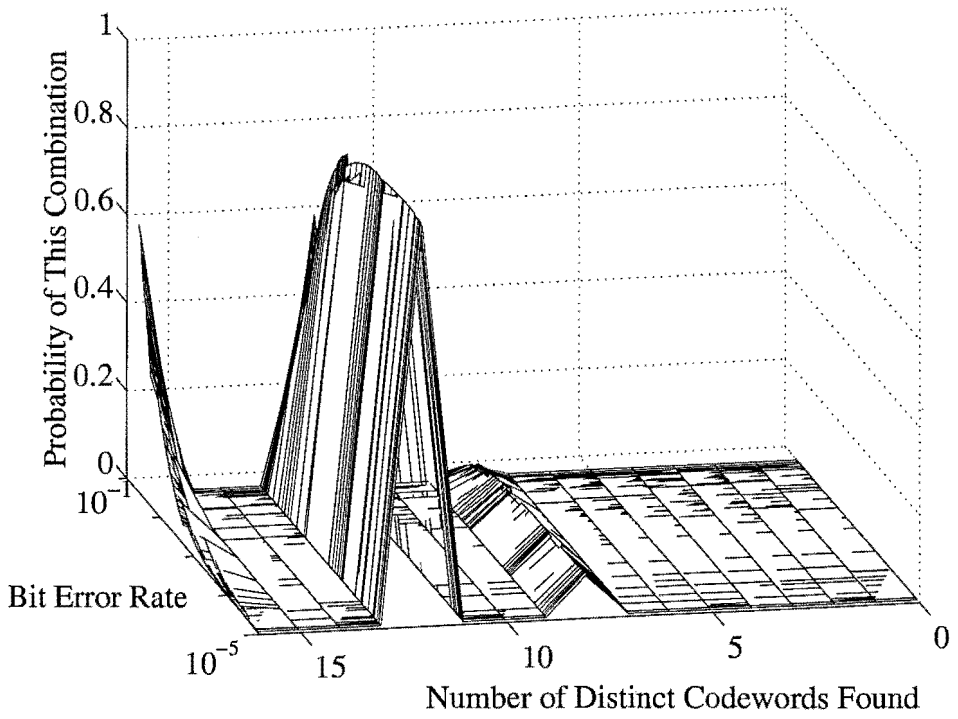


Figure 3.2: Probability of the Chase decoder finding a given number of distinct codewords for a given bit error rate after decoding. Results are for the $(64, 57, 4)^2$ block Turbo code, 2^p test sequences and $p = 4$.

When the unextended component codes are quasi-perfect codes (as defined in appendix B), if p is not large enough, occasionally only one or even no codewords will be found. This is because some hard received vectors are outside all the radius t error correction spheres that surround the codewords. If no codewords are found (case 4) then the hard decision on the soft input is used as the hard output. An example of

this is the $(64, 51, 6)^2$ block Turbo code. The number of distinct codewords found at different BERs is shown in Fig. 3.3 for $p = 4$ and 2^p test sequences. No codewords were found less than $0 - 0.005\%$ of the time. For this code and set of error patterns 74-85% of the extrinsic information is calculated using the case 2 estimate of (3.26), 0-4% using the case 3 estimate of (3.27) or the case 4 estimate, and the remainder using the case 1 value of (3.25) as shown in Fig. 3.4. If p is increased to 5 and all 2^p test sequences are used, then at least 2 distinct codewords are always found so case 3 and 4 never occur and no extrinsic information is calculated using (3.27) as shown in Fig. 3.5. In this case 57-71% of the extrinsic information is calculated using the case 2 estimate of (3.26) and the rest using the case 1 value of (3.25) as shown in Fig. 3.6. By increasing the number of test sequences used the percentage of extrinsic information calculated using (3.25) has increased. The high percentage of positions requiring an estimate of $\beta_j(q)$ (especially for small values of p and a small number of test sequences) highlights the need for a good estimate.

When the unextended component codes are neither perfect or quasi perfect the situation is even more severe. This code will have hard received vectors which do not fall into any of the radius t error correction spheres that surround codewords. As a result decoding failures will occur and in some cases no codewords will be found. An example of this is the $(64, 45, 8)$ extended BCH code, which is a three error correcting code. If $p = 5$ and 32 test sequences are used then at high BERs less than 10 distinct codewords are usually found and at low BERs less than 5 as shown in Fig. 3.7. In addition 0-1.52% of the time no codewords are found by the Chase decoder. This has a dramatic effect on the percentage of positions using each type of $\beta_j(q)$ as shown in Fig. 3.8. In this case 55-70% of the positions use the case 2 estimate of (3.26), 8-35% of the positions use the case 3 estimate of (3.27) or the case 4 estimate and only 10-25% of the positions use the case 1 value of (3.25). Case 1 and 2 are used more at high BERs as more codewords are found. The estimates use the case 1 values and so they will not be quite as good an estimate when fewer case 1 estimates are found. This may mean that a higher number of test sequences are required to obtain good performance and therefore, more complexity results. The previous 1 and 2 error correcting codes may be a better option than this code because of this.

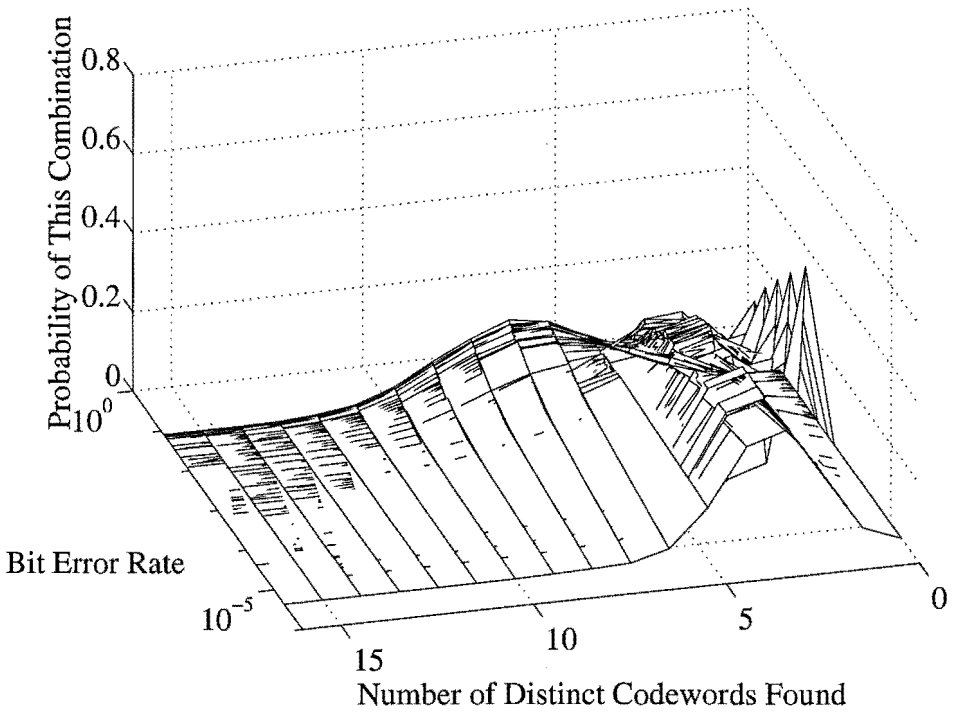


Figure 3.3: Probability of the Chase decoder finding a given number of distinct codewords for a given bit error rate after decoding. Results are for the $(64, 51, 6)^2$ block Turbo code, 2^p test sequences and $p = 4$.

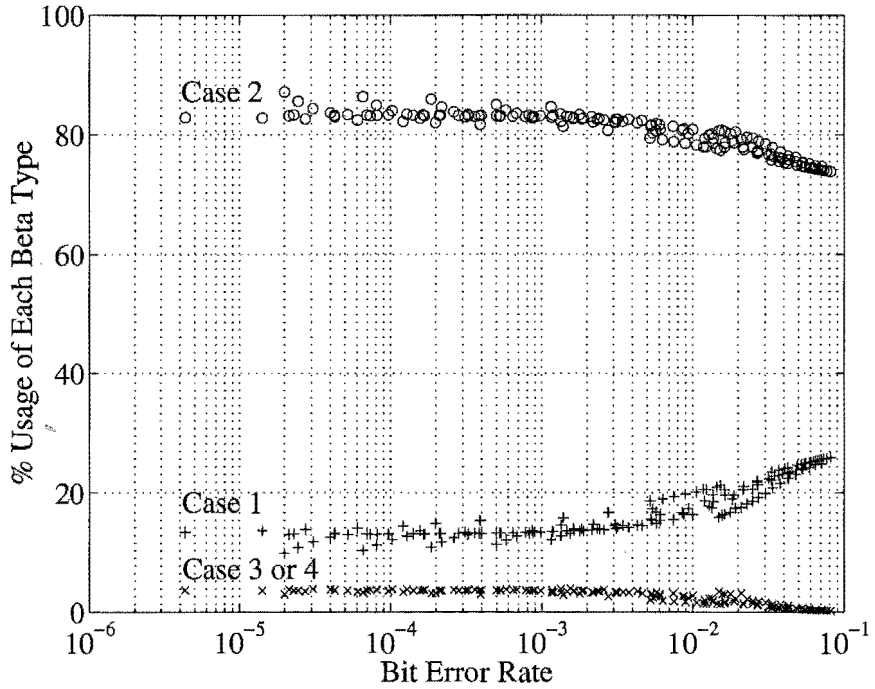


Figure 3.4: Percentage of positions using each type of $\beta_j(q)$ for different bit error rates after decoding. In case 1 $\boldsymbol{C} \neq \boldsymbol{D}$ is found with $c_j \neq d_j$. In case 2 $\boldsymbol{C} \neq \boldsymbol{D}$ is found, but $c_j = d_j$. In case 3 only $\boldsymbol{D}$ is found. In case 4 no codewords are found. Results are for the $(64, 51, 6)^2$ block Turbo code, 2^p test sequences and $p = 4$.

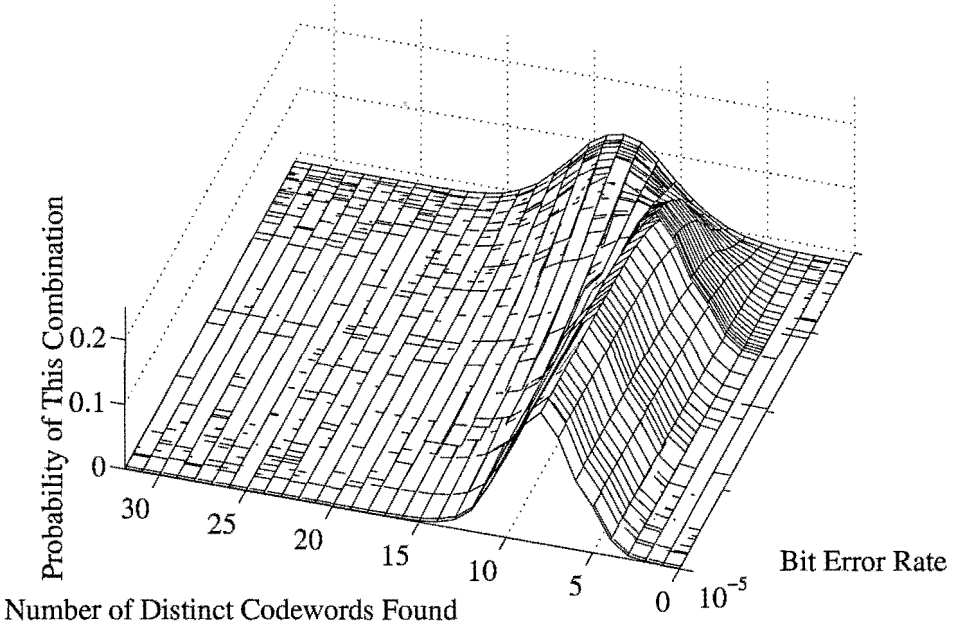


Figure 3.5: Probability of the Chase decoder finding a given number of distinct codewords for a given bit error rate after decoding. Results are for the $(64, 51, 6)^2$ block Turbo code, 2^p test sequences and $p = 5$.

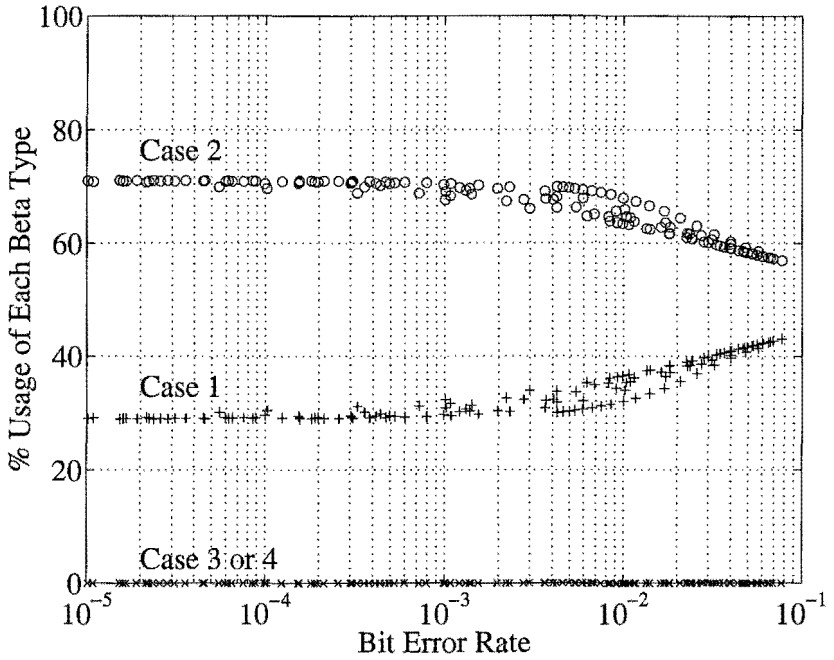


Figure 3.6: Percentage of positions using each type of $\beta_j(q)$ for different bit error rates after decoding. In case 1 $\boldsymbol{C} \neq \boldsymbol{D}$ is found with $c_j \neq d_j$. In case 2 $\boldsymbol{C} \neq \boldsymbol{D}$ is found, but $c_j = d_j$. In case 3 only $\boldsymbol{D}$ is found. In case 4 no codewords are found. Results are for the $(64, 51, 6)^2$ block Turbo code, 2^p test sequences and $p = 5$.

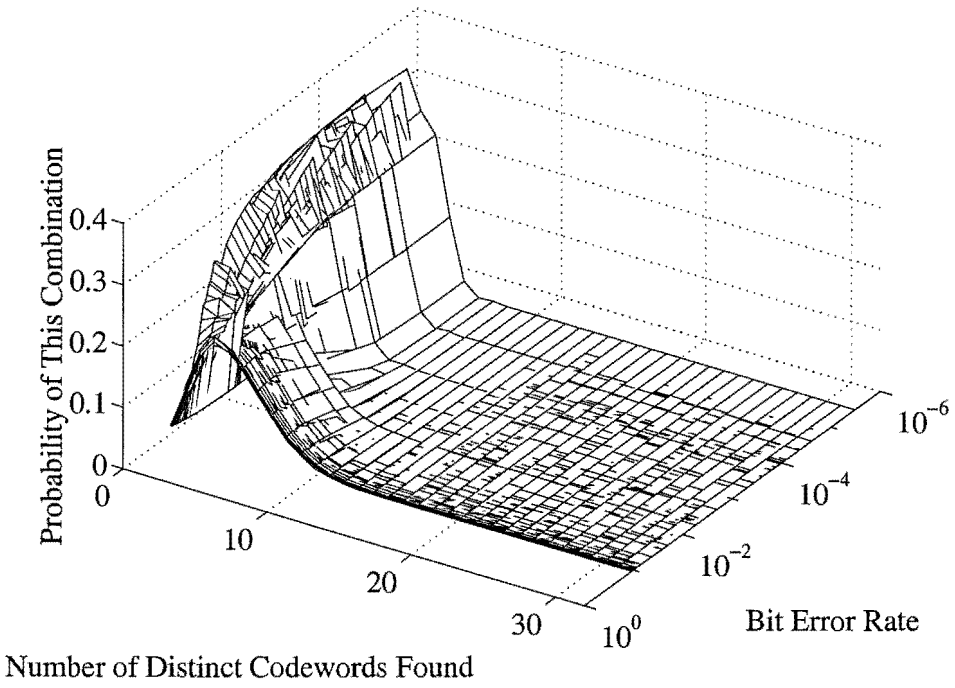


Figure 3.7: Probability of the Chase decoder finding a given number of distinct codewords for a given bit error rate after decoding. Results are for the $(64, 45, 8)^2$ block Turbo code on the AWGN channel. All 2^p test sequences are used and $p = 5$.

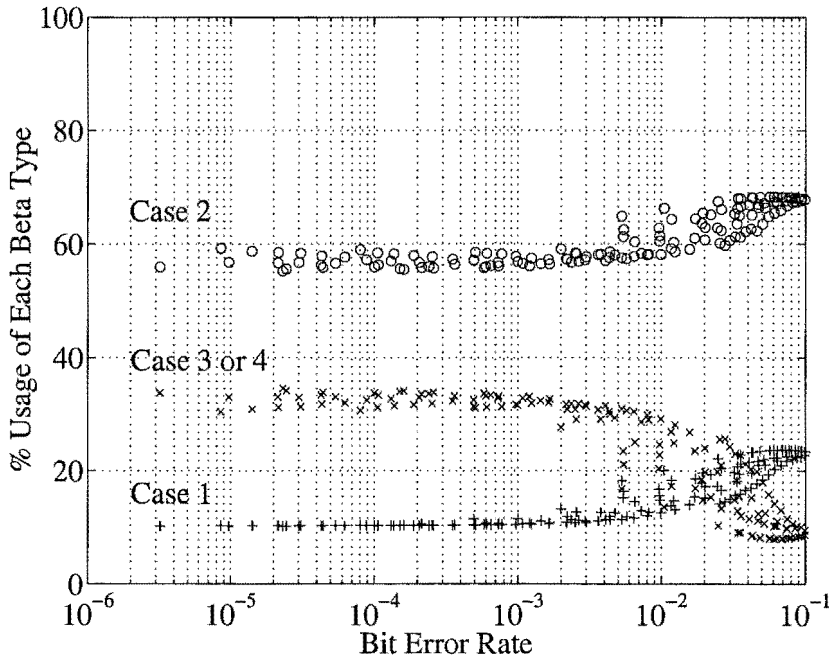


Figure 3.8: Percentage of positions using each type of $\beta_j(q)$ for different bit error rates after decoding. In case 1 $\boldsymbol{C} \neq \boldsymbol{D}$ is found with $c_j \neq d_j$. In case 2 $\boldsymbol{C} \neq \boldsymbol{D}$ is found, but $c_j = d_j$. In case 3 only $\boldsymbol{D}$ is found. In case 4 no codewords are found. Results are for the $(64, 45, 8)^2$ block Turbo code on the AWGN channel. All 2^p test sequences are used and $p = 5$.

3.3.2 Distance Based Approach

When no competing codeword $\boldsymbol{C}$ (with $c_j \neq d_j$) is found for the j^{th} position, $\beta_j(q)$ cannot be calculated directly using (3.25) as the locations where $d_l \neq c_l$ are not known and an estimate is required. This section develops an adaptive estimate of $\beta_j(q)$, denoted $\hat{\beta}_j(q)$, using arguments based on the distance properties of the component codes. First the error patterns used by the SILO decoder are considered. In this chapter the SILO Chase decoder will be used (which was described in section 2.2). Recall that the Chase decoder creates binary error patterns by using combinations of ones in the $p \geq \lfloor d_{H,min}/2 \rfloor$ *least reliable positions*[7] *(LRPs)* and placing zeros elsewhere, where $d_{H,min}$ is the minimum Hamming distance of the code. The LRPs are labelled in order of increasing reliability as $\boldsymbol{x} = (x_1, \cdots, x_p)$, where x_l is the index of the l^{th} LRP in the vector of soft inputs, $\boldsymbol{\lambda}(q)$.

In the AWGN channel the most likely error patterns are those containing the fewest ones in the LRPs and zeros elsewhere. Therefore, for $p = 5$ the $2^{p-1} = 16$ error patterns in Table 3.1 were chosen. Alternatively, all 2^p binary combinations in the p LRPs can be used. Define p' as the number of LRPs, $(x_1, \cdots, x_{p'})$, for which all $2^{p'}$ binary combinations are included in the set of error patterns. If $p' < p$ then some additional error patterns are used which consist of combinations of ones in the p LRPs, $(x_1, \cdots, x_p)$. By inspecting Table 3.1 it can be seen that for this set of error patterns $p' = 3$, since all binary combinations are found in positions (x_1, x_2, x_3) with zeros elsewhere, where $1 \leq x_l \leq n$ for all l.

As in [19] test sequences are created by adding the error pattern to the hard (binary) decision on the soft input to the q^{th} decoder, $\boldsymbol{Y}(q) = (y_1(q), \cdots, y_n(q))$, using modulo 2 addition. If an extended block code is used then the codeword consists of an $n-1$ bit unextended block code with an overall parity bit appended to the beginning of the codeword to ensure even parity. Only the unextended block code is used in the search for the p LRPs, so in this case $2 \leq x_l \leq n$ for $0 \geq l \geq p$. Again the overall parity bit will be treated as having no competing codeword.

The Hamming distance between codewords $\boldsymbol{C}$ and $\boldsymbol{D}$ is $d_H(\boldsymbol{C}, \boldsymbol{D}) \geq d_{H,min}$, where $d_{H,min}$ is the minimum Hamming distance of the component code. In (3.25) the difference between $\boldsymbol{C}$ and $\boldsymbol{D}$ in the j^{th} position is not included in the sum to

[7]The LRP is the position in the soft input with the smallest absolute value.

Table 3.1

Error pattern values in the $p = 5$ least reliable positions, $(x_1, \cdots, x_{p=5})$. Error pattern values equal zero in all other positions.

x_1	x_2	x_3	x_4	x_5	x_1	x_2	x_3	x_4	x_5
0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	0	0	0	0	1
0	1	0	0	0	1	0	0	1	0
1	1	0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0	1	0
1	0	1	0	0	0	1	0	0	1
0	1	1	0	0	0	0	1	1	0
1	1	1	0	0	1	1	0	1	0

calculate $\beta_j(q)$. Therefore, at least $d_{H,min} - 1$ other positions are needed to calculate $\hat{\beta}_j(q)$.

If no $\boldsymbol{C}$ is found for the j^{th} position, then an estimate to reinforce the decoders decision in that position, d_j, is desired implying $\hat{\beta}_j(q) > 0$. Since simulations found it better to underestimate $\beta_j(q)$, the smallest positive value of $\beta_j(q)$ is desired. It is not known where $\boldsymbol{C}$ and $\boldsymbol{D}$ differ, so a set of possible combinations of positions is considered[8], denoted $\{\xi\}$. It must be ensured that $\{\xi\}$ includes enough LRPs to always produce a positive value of $\hat{\beta}_j(q)$. Therefore, the maximum number of positions where $d_l\lambda_l(q)$ is negative must be found (due to the definition of $\beta_j(q)$ in (3.25)). A negative value occurs in the l^{th} position when the Chase decoder's decision differs from the hard decision on the soft input in that position, $d_l \neq y_l(q)$. The error patterns can change at most $weight(p)$ of the LRPs in $\boldsymbol{Y}(q)$ before decoding, where $weight(p)$ is the maximum weight of the error patterns used by the Chase decoder. The structure of the Chase decoder is shown in Fig. 2.3 and contains a *hard-input hard-output (HIHO)* decoder for the unextended component code. The HIHO decoder used in this thesis[9] can change a further t positions in $\boldsymbol{Y}(q)$. If an extended code is used then the overall parity bit can also change. Therefore, the

[8]For complexity reasons the set is restricted, so the smallest estimate may not be included. Since the parity equations relating to the codes are not used by the set, the combinations of positions may not relate to codewords. This means that the estimate can also be too small.

[9]Peterson's direct method for decoding BCH codes is used to decode t or fewer errors in this thesis (it was described in section 2.2). In some rare cases an error locator polynomial to correct more than t errors can be found [120]. A small improvement in performance may be possible if these error patterns were corrected (when using a non-perfect code).

number of positions where $d_l\lambda_l(q) < 0$ for an extended code equals the Hamming distance between $\boldsymbol{D}$ and $\boldsymbol{Y}(q)$ and is defined as

$$d_H(\boldsymbol{D}, \boldsymbol{Y}(q)) \leq weight(p) + t + 1 \tag{3.29}$$

This is used to determine the number of positions, N_s, which must be considered to ensure a positive estimate will be found. The value of N_s also depends on the set $\{\xi\}$ used. The set $\{\xi\}$ is restricted to including only combinations of the N_s LRPs. The new estimate can be written as[10]

$$\hat{\beta}_j(q) = \min_{\{\xi\}} \left\{ b = \left(\sum_{l=1, l\in\xi}^{n} d_l\lambda_l(q) \right) : b > 0 \right\} \tag{3.30}$$

The set must be chosen carefully as it can affect decoding performance. If no $\boldsymbol{D}$ is found, then $\boldsymbol{D}$ is set equal to $\boldsymbol{Y}(q)$ and (3.30) is used for all positions. Alternatively, $\hat{\beta}_j(q)$ can be set to zero or the case 3 estimate from the approximation approach, (3.27), could be used. Two approaches to the set $\{\xi\}$ will now be discussed. At least $d_{H,min} - 1$ positions need to be summed. Both sets will use combinations of $d_{H,min} - 1$ positions.

Set One

The first set of combinations, $\{\xi\}$, used in the simulations of section 3.4 is now discussed. An extended codeword is assumed, but the set could easily be modified for use with an unextended codeword.

If no $\boldsymbol{C}$ is found by the decoder for a given position, then there are $t + 1$ or more positions which differ between $\boldsymbol{Y}(q)$ and $\boldsymbol{C}$, outside the set of test sequences and therefore, outside the p' LRPs. Otherwise, a $\boldsymbol{C}$ would have been found. In this derivation it is assumed that in these positions[11] $d_l \neq c_l$. This means $t + 1$ or more of the $d_{H,min} - 1$ positions to be summed will be outside the p' LRPs. Since it was found to be preferable to underestimate $\beta_j(q)$ in simulations, it is assumed that when $\boldsymbol{C}$ is not found the j^{th} position is outside the p' LRPs[12]. Now t or more of the $d_{H,min} - 1$ positions used to calculate $\hat{\beta}_j(q)$ are outside the p' LRPs. These

[10] Any combination of positions, ξ, which produces a negative estimate is discarded.
[11] In some cases $d_l = c_l$ in the positions outside the p' LRPs where $c_l \neq y_l(q)$.
[12] This is true for $n - p'$ positions in the codeword.

assumptions mean that only one estimate is calculated for each decision codeword, $\boldsymbol{D}$.

If all values of $d_l\lambda_l(q)$ for the p' LRPs are positive, then positive values are required in a maximum of t positions outside the p' LRPs if $p' \geq d_{H,min} - t - 1$. In this case to ensure a positive $\hat{\beta}_j(q)$ at least

$$N_s = p' + t + max(d_H(\boldsymbol{D}, \boldsymbol{Y}(q))) \tag{3.31}$$

LRPs must be considered, where from (3.29) $max(d_H(\boldsymbol{D}, \boldsymbol{Y}(q))) = weight(p)+t+1$. If $p' < d_{H,min} - t - 1$ then more than $d_{H,min} - 1 - p' > t$ positions outside the p' LRPs need to be positive to ensure a positive estimate. Therefore to cover all cases (3.31) becomes

$$N_s = max(p' + t, d_{H,min} - 1) + max(d_H(\boldsymbol{D}, \boldsymbol{Y}(q))) \tag{3.32}$$

The overall parity position is not included in the search for the N_s LRPs. In order to reduce complexity $\boldsymbol{Y}(q)$ and $\boldsymbol{D}$ are used to find a sufficient value of N_s to ensure that at least one combination will produce a positive value of $\hat{\beta}_j(q)$, this value will be called N'_s. The first step in calculating N'_s is to find the number of positions in the p' LRPs where $y_l(q) \neq d_l$, denoted $N_{p'}$. If $N_{p'} > p' - (d_{H,min} - 1 - t)$ then N'_s is chosen to include $t + N_{p'} - (p' - (d_{H,min} - 1 - t))$ positions outside the p' LRPs where $y_l(q) = d_l$. Otherwise, N'_s is chosen to include t positions outside the p' LRPs where $y_l(q) = d_l$. This gives $p' + t \leq N'_s \leq N_s$. The overall parity position is now appended to the N'_s LRPs, giving $N'_s = N'_s + 1$.

The set of combinations, $\{\xi\}$, then consists of all combinations of $t \leq b \leq \min\{N'_s - p', d_{H,min} - 1\}$ positions in the N'_s LRPs which are outside the p' LRPs and $a = d_{H,min} - 1 - b$ positions inside the p' LRPs. This means $0 \leq a \leq \min\{p', d_{H,min} - 1 - t\}$ and $a + b = d_{H,min} - 1$.

Set Two

The second set of combinations, $\{\xi\}$, is easier to explain and has a lower decoding complexity. It simply sums all combinations of $d_{H,min} - 1$ positions in the $N'_s \leq N_s$ LRPs, where

$$N_s = d_{H,min} - 1 + max(d_H(\boldsymbol{Y}(q), \boldsymbol{D})) \tag{3.33}$$

and N_s' is the number of LRPs required to ensure that there are $d_{H,min} - 1$ positions where $y_l(q)d_l \geq 0$. The overall parity is then added to the set of N_s' LRPs. The minimum positive value obtained using (3.30) is then used as the estimate $\hat{\beta}_j(q)$. As with set 1 only one estimate is used for each decision codeword, $\boldsymbol{D}$.

3.4 Simulation Results

This section presents simulation results for a variety of block Turbo codes with extended BCH component codes transmitted using BPSK/ QPSK on the memoryless AWGN channel. The adaptively estimated $\boldsymbol{\alpha}$ of section 3.2 will be used with both the approximation and distance based approaches to estimating $\boldsymbol{\beta}$ of section 3.3. First the simulation results for the approximation based approach to estimating $\boldsymbol{\beta}$ are presented and then those for the distance based approach are presented.

The SILO Chase decoder uses a set of 2^p test sequences for $p = 4$ unless otherwise stated. The BER of a data bit or *frame error rate (FER)* of a data block is displayed against the SNR, which is equal to E_b/N_0, where E_b is the energy used to transmit a data bit and N_0 is the 2-sided baseband power spectral density of the noise. Unless otherwise stated the BER or FER after 1, 2, 3, 4, 6 and 10 iterations is shown and the performance improves after each iteration, but with diminishing returns.

The simulation results presented in this chapter were obtained after finding 200 block Turbo code blocks in error after the final iteration for a given E_b/N_0. Due to time constraints and computer system crashes, some points shown were obtained after fewer blocks in error. Unless otherwise stated at least 100 blocks in error were used. There are n codewords per block.

3.4.1 Approximation Based Approach to Estimating $\boldsymbol{\beta}$

This subsection presents simulation results for the adaptively estimated $\alpha(q)$ of section 3.2 and the approximation based adaptively estimated $\beta_j(q)$ developed in section 3.3. This will be called the approximation approach. The first set of simulations compare the performance of the approximation approach to that of the precomputed (non-adaptive) approach of [90] (described in section 2.3).

Results for the $(32, 21, 6)^2$ block Turbo code are shown in Fig. 3.9. As can be seen the approximation approach provides a significant improvement over the results in [90] in early iterations for a small increase in complexity. The increase in complexity is due to the need to estimate the variances $\sigma_{\boldsymbol{R}}^2$ and $\sigma_{\boldsymbol{W}}^2$, and the mean $\mu_{\boldsymbol{W}}$. Results for the $(64, 51, 6)^2$ block Turbo code are given in Fig. 3.10. Again a significant improvement in performance is achieved in early iterations and after 6 iterations of the approximation approach the performance is slightly better than that of [90] after 10 iterations (a saving of 4 iterations). Results for the $(128, 113, 6)^2$ block Turbo code are given in Fig. 3.11. Again there is a significant improvement in performance in early iterations and now after 4 iterations the approximation approach performs almost as well as the approach of [90] after 10 iterations. The effect of changing the length of the code for a fixed value of $d_{H,min}$ after 1 and 10 iterations is shown in Fig. 3.12. As can be seen the performance gain of the approximation approach increases as n decreases in early iterations, but increases as n increases in later iterations.

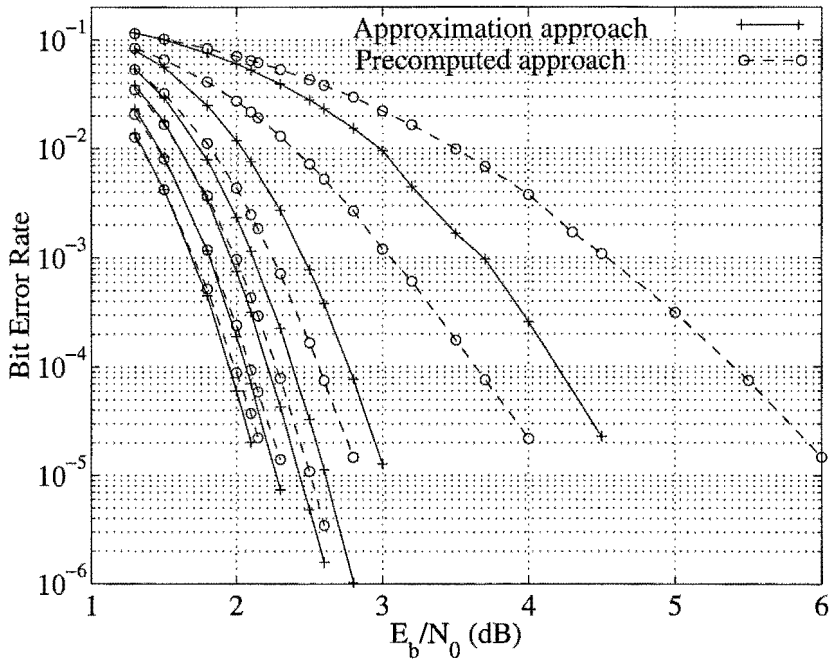


Figure 3.9: Performance of the $(32, 21, 6)^2$ block Turbo code using the precomputed and approximation approaches, $p = 4$ and 2^p test sequences.

The effect of changing $d_{H,min}$ (and therefore t) when $p = d_{H,min}/2$ (with all 2^p

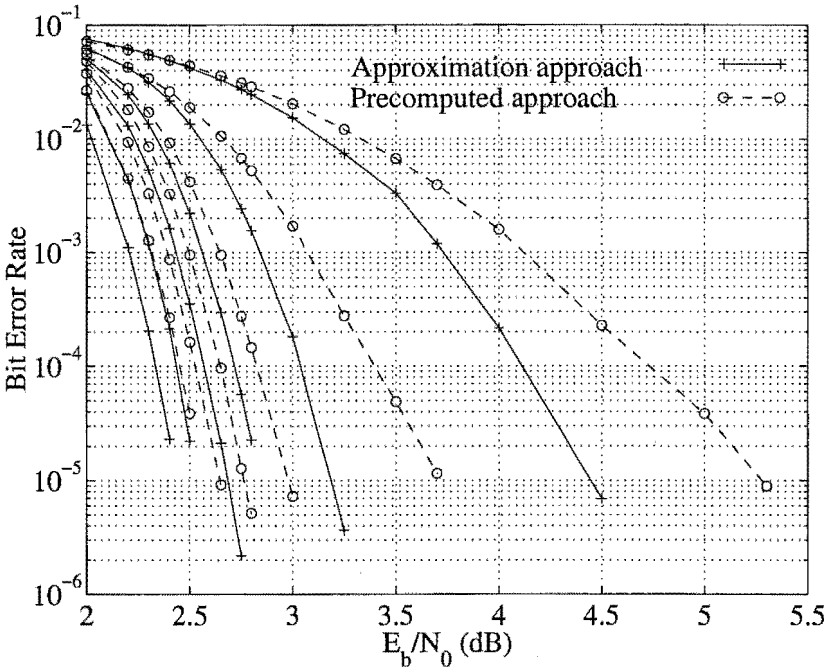


Figure 3.10: Performance of the $(64, 51, 6)^2$ block Turbo code using the precomputed or approximation approach, $p = 4$ and 2^p test sequences.

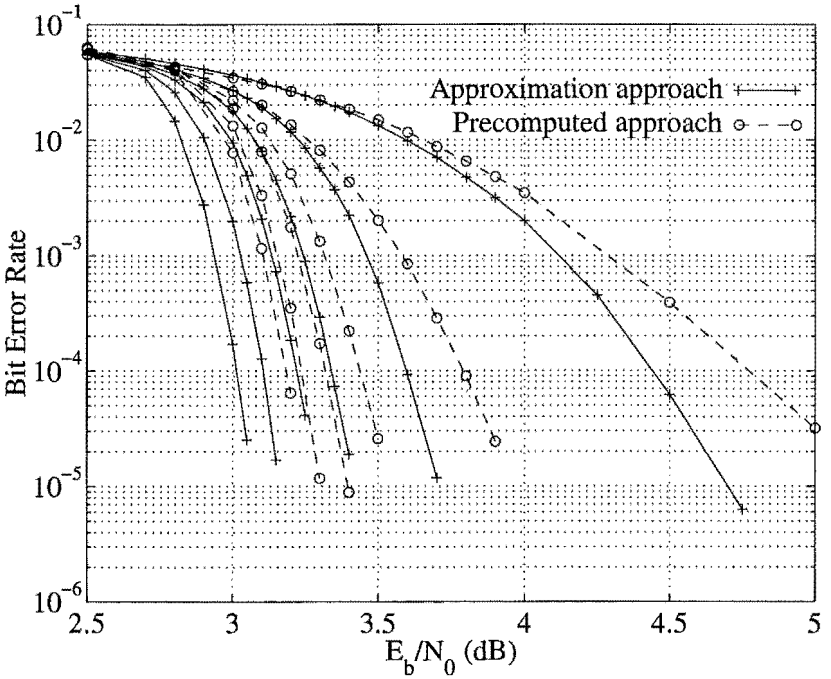


Figure 3.11: Performance of the $(128, 113, 6)^2$ block Turbo code using the precomputed or approximation approach, $p = 4$ and 2^p test sequences.

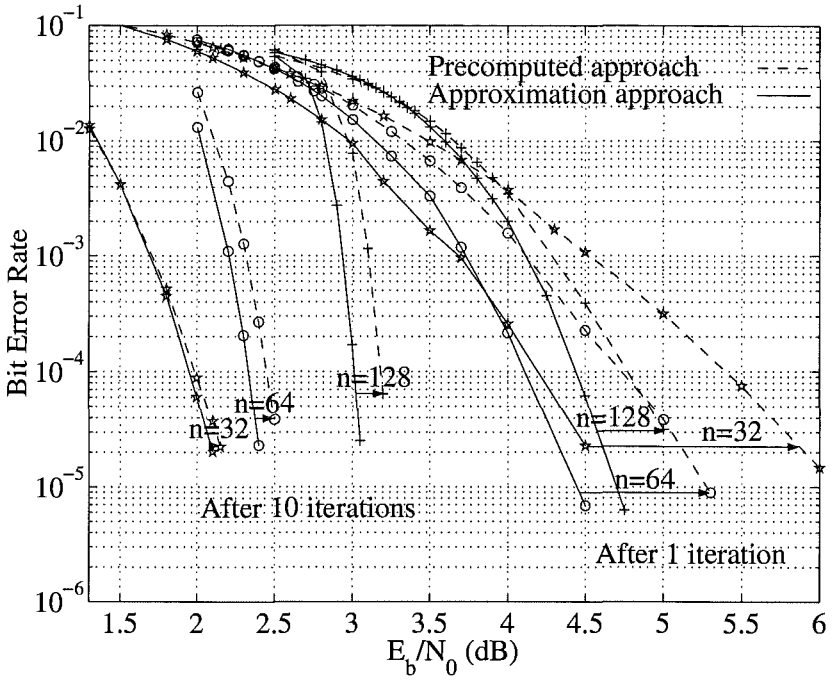


Figure 3.12: Performance after 1 and 10 iterations of the $(32, 21, 6)^2$ ($n = 32$), $(64, 51, 6)^2$ ($n = 64$) and $(128, 113, 6)^2$ ($n = 128$) block Turbo codes using the precomputed or approximation approach, $p = 4$ and 2^p test sequences.

test sequences being used)[13] and a fixed block Turbo code block length of $N = 64^2$ is used is shown in Fig. 3.13. Complexity increases for larger values of $d_{H,min}$ and larger numbers of test sequences. The $(64, 51, 6)^2$ block Turbo code performs better than the $(64, 45, 8)^2$ block Turbo code. The lack of distinct codewords found by the $(64, 45, 8)^2$ block Turbo code (as shown in Fig 3.7) appears to degrade the performance of the code. To improve performance the number of codewords (and therefore test sequences) used would have to be further increased. Taking into account the performance in Fig. 3.13, the complexity and the rates of the codes, the $(64, 51, 6)^2$ block Turbo code would be a better code to use than the $(64, 45, 8)^2$ block Turbo code.

The effect of using different values of p and numbers of test sequences is shown in Fig. 3.14[14] for the $(64, 51, 6)^2$ block Turbo code and in Fig. 3.15 for the $(64, 57, 4)^2$ block Turbo code. As can be seen performance improves when more test sequences

[13] $p = d_{H,min}/2$ is the value used in the original Chase decoder of [19].

[14] Only 47 (out of 143533) and 54 (out of 105151) block Turbo code blocks were observed in error at 2.3 dB for $p = 6$ and 32 test sequences, and $p = 5$ and 32 test sequences respectively. Only 61 (out of 126761) blocks were found in error at 2.4 dB for $p = 5$ and 16 test sequences.

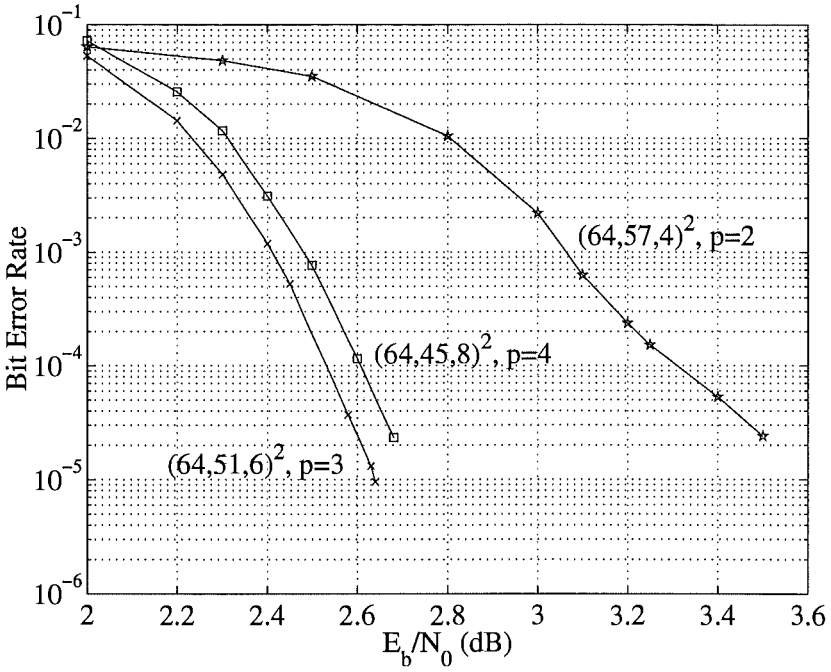


Figure 3.13: Performance of different length $N = 64^2$ block Turbo codes using the approximation approach. Results after 10 iterations for $p = d_{H,min}/2$ and 2^p test sequences are shown.

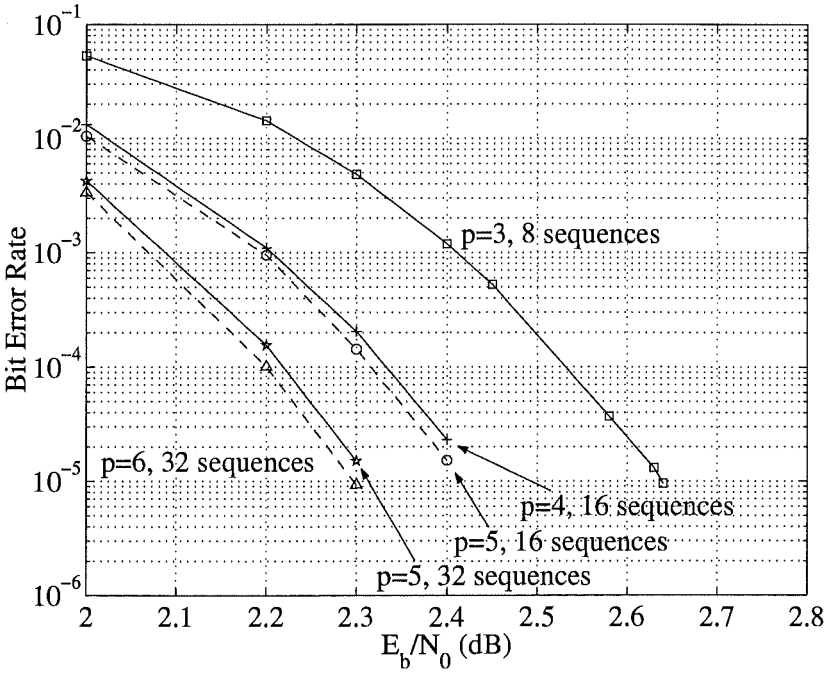


Figure 3.14: Performance of the $(64, 51, 6)^2$ block Turbo code using the approximation approach. Results after 10 iterations for different values of p and numbers of test sequences are shown.

are used, but with diminishing returns. This improvement is due to an increase in the average number of distinct codewords found by the Chase decoder as discussed in section 3.3. An improvement can also be achieved by using $p+1$ positions and 2^p test sequences, instead of p positions and 2^p test sequences. The error patterns of Table 3.1 are used to create the test sequences when $p = 5$ and 16 test sequences are used.

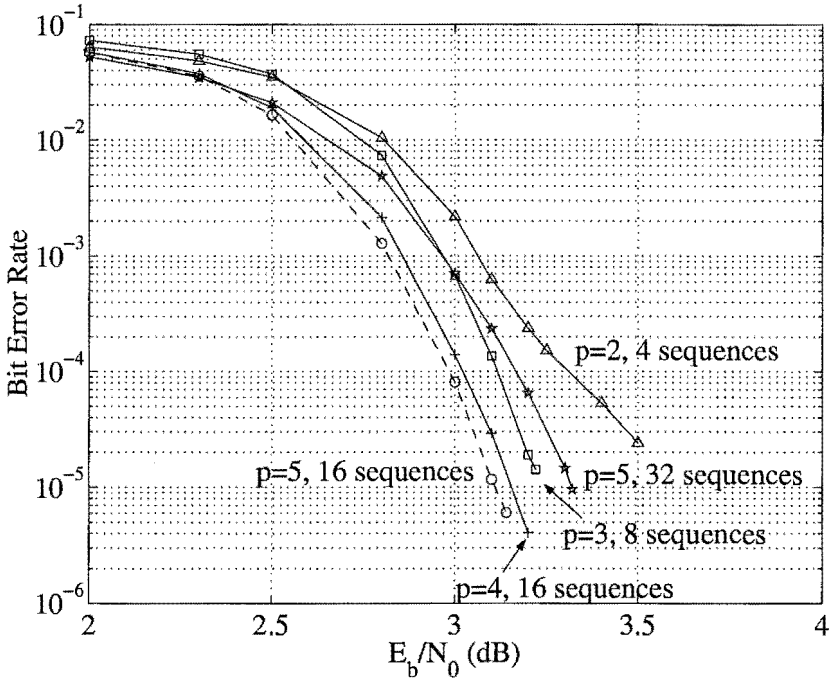


Figure 3.15: Performance of the $(64, 57, 4)^2$ block Turbo code using the approximation approach. Results after 10 iterations for different values of p and numbers of test sequences are shown.

An exception to the observations above can be seen in Fig. 3.15 for $p = 5$ and 32 test sequences. In this case the performance is actually worse than if fewer test sequences or LRPs had been used. Note that the maximum Hamming distance between test sequences is $p = 5 = d_{H,min} + 1$ in this case. The reason for the poor performance appears to be that the set of codewords produced by the Chase decoder may contain codewords that are a long distance from the soft input vector in terms of squared Euclidean distance. This is because even though the error patterns only alter the LRPs, the HIHO decoder does not use reliability information and may change t positions which are very reliable. The overall parity may also be changed by the Chase decoder and may be very reliable. This means that the closest codeword

with $c_j \neq d_j$ may not be considered; instead a codeword a long way from the soft input vector may be used. This results in the extrinsic information calculated using $\boldsymbol{C}$ being too large in some positions. This can be advantageous in some positions, but overall seriously degrades performance as shown in Fig. 3.15.

This problem would also be encountered when using other SILO decoders such as [99]. A solution to this problem is to only use competing codewords within a specified squared Euclidean distance of the soft input vector. This squared Euclidean distance will be called the maximum allowed value.

The squared Euclidean distance between the soft input vector, $\boldsymbol{\lambda}(q)$, and a competing codeword $\boldsymbol{C}$ is given by

$$d_E^2(\boldsymbol{\lambda}(q), \boldsymbol{C}) = |\boldsymbol{\lambda}(q) - \boldsymbol{C}|^2 = \sum_{l=1}^{n} \left(\lambda_l(q)^2 - 2\lambda_l(q)c_l + c_l^2\right) \tag{3.34}$$

which can be written as

$$d_E^2(\boldsymbol{\lambda}(q), \boldsymbol{C}) = \sum_{l=1}^{n} \left(\lambda_l(q)^2 + d_l^2\right) - 2 \sum_{l=1, c_l = d_l}^{n} (\lambda_l(q)d_l) + 2 \sum_{l=1, c_l \neq d_l}^{n} (\lambda_l(q)d_l) \tag{3.35}$$

since $c_l^2 = d_l^2 = 1$. Similarly the squared Euclidean distance between the soft input vector and the decision codeword, $\boldsymbol{D}$, is given by

$$d_E^2(\boldsymbol{\lambda}(q), \boldsymbol{D}) = \sum_{l=1}^{n} \left(\lambda_l(q)^2 + d_l^2\right) - 2 \sum_{l=1, c_l = d_l}^{n} (\lambda_l(q)d_l) - 2 \sum_{l=1, c_l \neq d_l}^{n} (\lambda_l(q)d_l) \tag{3.36}$$

Comparing (3.35) and (3.36) $d_E^2(\boldsymbol{\lambda}(q), \boldsymbol{C})$ can be written as

$$d_E^2(\boldsymbol{\lambda}(q), \boldsymbol{C}) = d_E^2(\boldsymbol{\lambda}(q), \boldsymbol{D}) + 4 \sum_{l=1, c_l \neq d_l}^{n} (\lambda_l(q)d_l) \tag{3.37}$$

From the definition of $\boldsymbol{D}$

$$d_E^2(\boldsymbol{\lambda}(q), \boldsymbol{C}) \geq d_E^2(\boldsymbol{\lambda}(q), \boldsymbol{D}) \tag{3.38}$$

and the second term in (3.37) must satisfy

$$4 \sum_{l=1, c_l \neq d_l}^{n} (\lambda_l(q)d_l) \geq 0 \tag{3.39}$$

A maximum allowed value of $d_E^2(\boldsymbol{\lambda}(q), \boldsymbol{C})$ can be developed based on (3.37). The first term in (3.37) can be calculated exactly and the second term can be estimated to set an upper limit on the value of $d_E^2(\boldsymbol{\lambda}(q), \boldsymbol{C})$ allowed. Any codewords

with a greater squared Euclidean distance will not be used to calculate the extrinsic information. This definition ensures that if $\boldsymbol{D}$ is found it is retained.

It is important not to make the maximum allowed value of $d_E^2(\boldsymbol{\lambda}(q), \boldsymbol{C})$ too small or all codewords found apart from $\boldsymbol{D}$ may be eliminated. In an attempt to reduce this problem the two closest codewords to the soft input vector will automatically be retained (one of which is $\boldsymbol{D}$). This ensures that when two or more codewords are found by the Chase decoder $\beta_j(q)$ can be calculated using (3.25) for some positions.

If the maximum allowed value is too large then it will allow codewords that are too far from the soft input vector. However, if it is too small it can exclude too many codewords and valid competing codewords may be excluded. The following maximum allowed value was developed keeping this in mind. Initially the maximum allowed value is set equal to $d_E^2(\boldsymbol{\lambda}(q), \boldsymbol{D})$. Looking at (3.39) at least $d_{H,min}$ positions must be added to the maximum allowed value and the maximum value is achieved when all the positions summed have $c_l \neq (d_l = y_l(q))$ (meaning $\lambda_l(q)d_l \geq 0$). Outside the p LRPs, $c_l \neq (d_l = y_l(q))$ can only be true in a maximum of $t+1$ positions, since the Chase decoder can change at most $t+1$ positions (including the overall parity position) outside the p LRPs. The overall parity position is often altered (due to the decoding approach). Therefore, if $\lambda_l(q)d_l \geq 0$ in the overall parity position, then it is added to the maximum allowed value and at least $v = d_{H,min} - 1$ other positions must be found. Otherwise, the overall parity position is not used and $v = d_{H,min}$ positions must be found.

The most reliable position where $\lambda_l(q)d_l < 0$ is now found, as it gives an indication of how reliable the positions where $c_l \neq (d_l = y_l(q))$ may be. The values of $\lambda_l(q)d_l$ in the t more reliable positions than this are added to the maximum allowed value. Now at least $v = v - t$ other positions must be found. Now the $q \leq v$ most reliable positions in the p LRPs with $d_l\lambda_l(q) > 0$ are added to the maximum allowed value. Alternatively, all positive values in the p LRPs could be added to the sum (and included in the value of q). At least $v = v - q$ other positions must be found. If $v > 0$ then the v LRPs with $d_l\lambda_l(q) < 0$ are added to the maximum allowed value.

Results for the $(64, 57, 4)^2$ block Turbo code using $p = 5$ and 32 test sequences

are shown in Fig. 3.16 with and without the maximum allowed value (limit) being used. It can be seen that by limiting $d_E^2(\boldsymbol{\lambda}(q), \boldsymbol{C})$ the performance has been improved dramatically. The $(64, 57, 4)^2$ product code now performs better for $p = 5$ and 32 test sequences, than for $p = 5$ and 16 test sequences, as was originally expected. This result indicates that the distant competing codewords found by the Chase decoder were degrading the overall performance of the code.

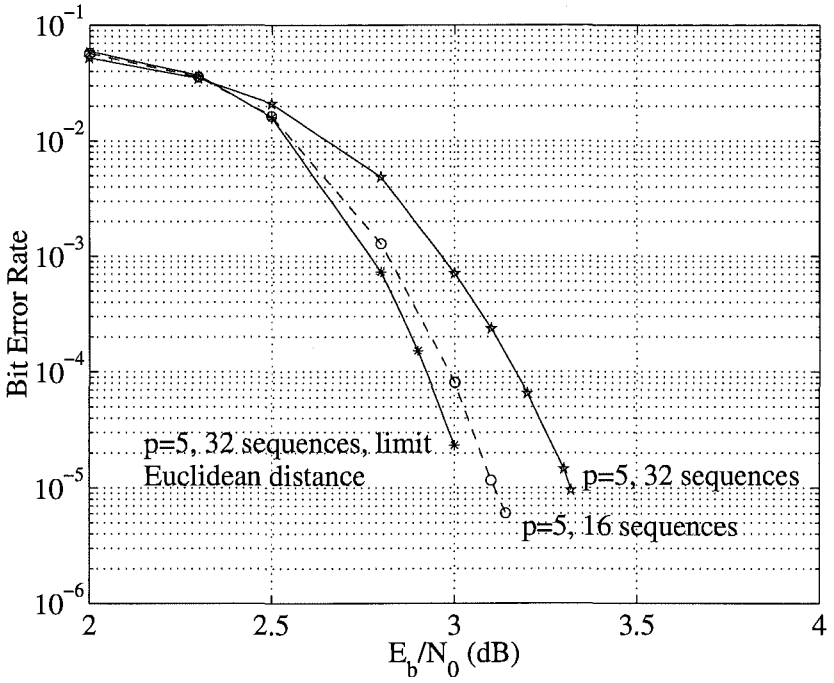


Figure 3.16: Performance of the $(64, 57, 4)^2$ block Turbo code after 10 iterations using the approximation approach. The effect of using a maximum allowed value of $d_E^2(\boldsymbol{\lambda}(q), \boldsymbol{C})$ is shown.

The percentage of positions using each type of β is shown in Fig. 3.17 for the $(64, 57, 4)^2$ product code with $p = 5$ and 32 test sequences, when the limit is not used. In this case $53 - 59\%$ of the extrinsic information is calculated using the case 2 estimate of (3.26) and the remaining $41 - 47\%$ is calculated using the case 1 value of (3.25). The percentage of positions using each type of β when the maximum allowed value is applied is shown in Fig. 3.18. Now $67 - 83\%$ of the extrinsic information is calculated using the case 2 estimate of (3.26) and the remaining $17 - 33\%$ is calculated using the case 1 value of (3.25). Therefore, by applying the maximum allowed value more positions use an estimate of β instead of (3.25). The number of positions calculated using the exact values of β is reduced by 20%, but results

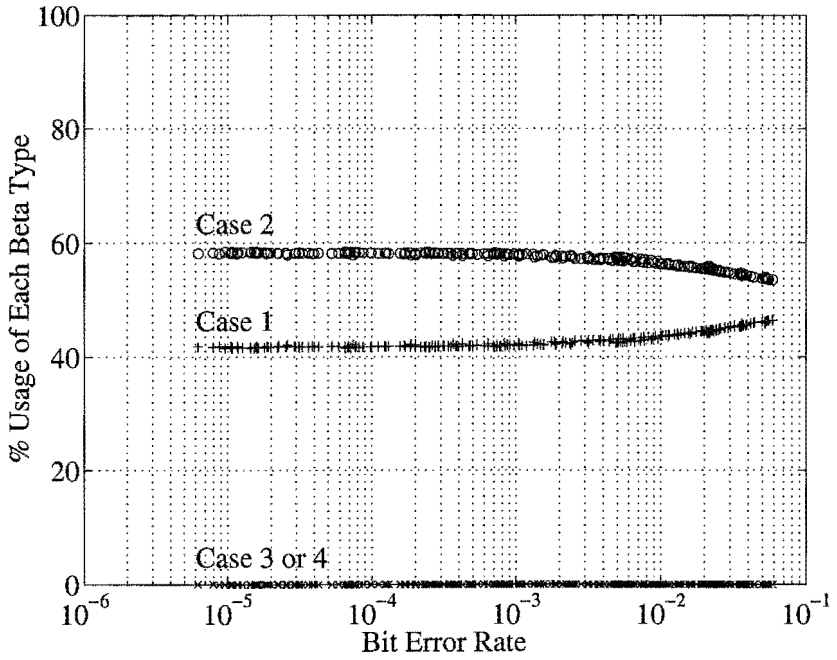


Figure 3.17: Percentage of positions using each type of $\beta_j(q)$ for different bit error rates after decoding. In case 1 $\boldsymbol{C} \neq \boldsymbol{D}$ is found with $c_j \neq d_j$. In case 2 $\boldsymbol{C} \neq \boldsymbol{D}$ is found, but $c_j = d_j$. In case 3 only $\boldsymbol{D}$ is found. In case 4 no codewords are found. Results are for the $(64, 57, 4)^2$ block Turbo code, 2^p test sequences and $p = 5$.

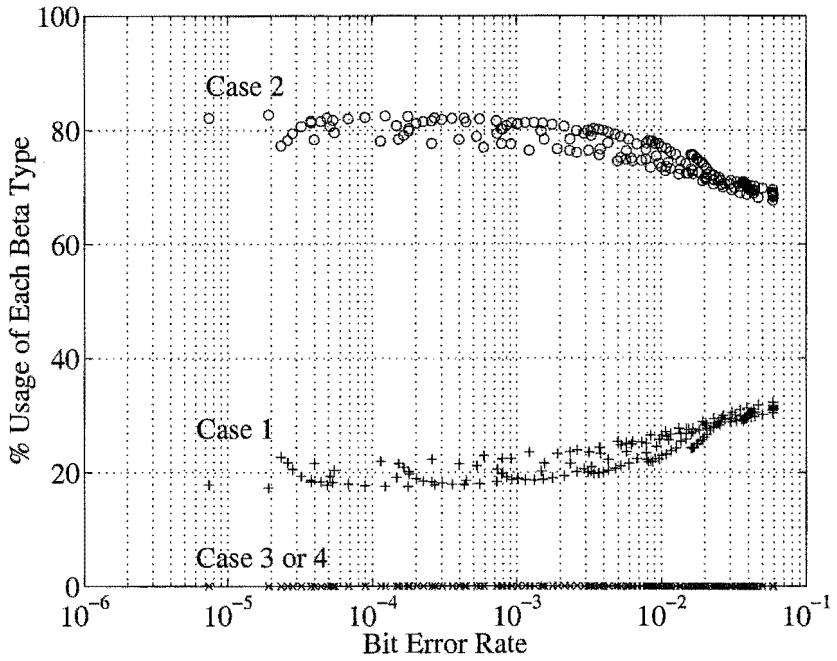


Figure 3.18: Percentage of positions using each type of $\beta_j(q)$ for different bit error rates after decoding. In case 1 $\boldsymbol{C} \neq \boldsymbol{D}$ is found with $c_j \neq d_j$. In case 2 $\boldsymbol{C} \neq \boldsymbol{D}$ is found, but $c_j = d_j$. In case 3 only $\boldsymbol{D}$ is found. In case 4 no codewords are found. Results are for the $(64, 57, 4)^2$ block Turbo code when the maximum allowed value is used. All 2^p test sequences are used and $p = 5$.

in improved performance. This indicates that it is better to use an estimate than to use a distant competing codeword. It also shows that good performance can be obtained by estimating the extrinsic information in a large percentage of positions, if the estimates are good. The key is to find a few close codewords.

The case 2 estimate of $\beta_j(q)$ developed in the approximation approach, (3.26), is a conservative estimate. Fig. 3.19 and Fig. 3.20 show the effect of multiplying (3.26) by various constants when the $(64, 51, 6)^2$ block Turbo code is transmitted. In this case the estimate developed in this chapter, (3.26), performs the best (meaning a scaling of 1). Fig. 3.21 and Fig. 3.22 show the effect of multiplying (3.26) by various constants when the $(64, 57, 4)^2$ block Turbo code is transmitted. In this case multiplying (3.26) by 1.5 performs the best, but only by 0.05dB at a BER of 10^{-5} after 10 iterations. If this "best" estimate (the x 1.5 curve) is overestimated by a factor of 2 (the x 3 curve), then it performs worse than underestimating by a factor of 3 (the x 0.5 curve). This demonstrates how the simulations showed it better to underestimate $\beta_j(q)$ rather than overestimate it. This also shows that if a scaler was used it would have to be optimised for different codes and applications, which is not desirable. The goal of this chapter was to avoid optimising the parameters by repeated simulations. In addition the gain achieved is small.

The effectiveness of the adaptively estimated $\boldsymbol{\alpha}$ can also be investigated by multiplying it by different constants. This also measures how good the overall approach to $\boldsymbol{\beta}$ is, as $\boldsymbol{\alpha}$ scales all the extrinsic information. Simulation results for $\boldsymbol{\alpha}$ multiplied by a constant are given in Fig. 3.23 and Fig. 3.24 for the $(64, 57, 4)^2$ block Turbo code, $p = 4$ and 16 test sequences. The best performance is obtained when no scaling is used (the x 1 curves). This illustrates the effectiveness of the adaptively estimated $\boldsymbol{\alpha}$ developed.

Two techniques for using the extrinsic information were described in section 2.3. In this work the extrinsic information is treated as a Gaussian random variable (Gaussian approach). The other approach treats the extrinsic information as a priori information (a priori approach), which has the effect of making $\alpha(q) = \sigma_R^2/2$ for all decoding stages. In [21] the a priori approach was found to provide better performance than the Gaussian approach when decoding a rate 1/2 Turbo code

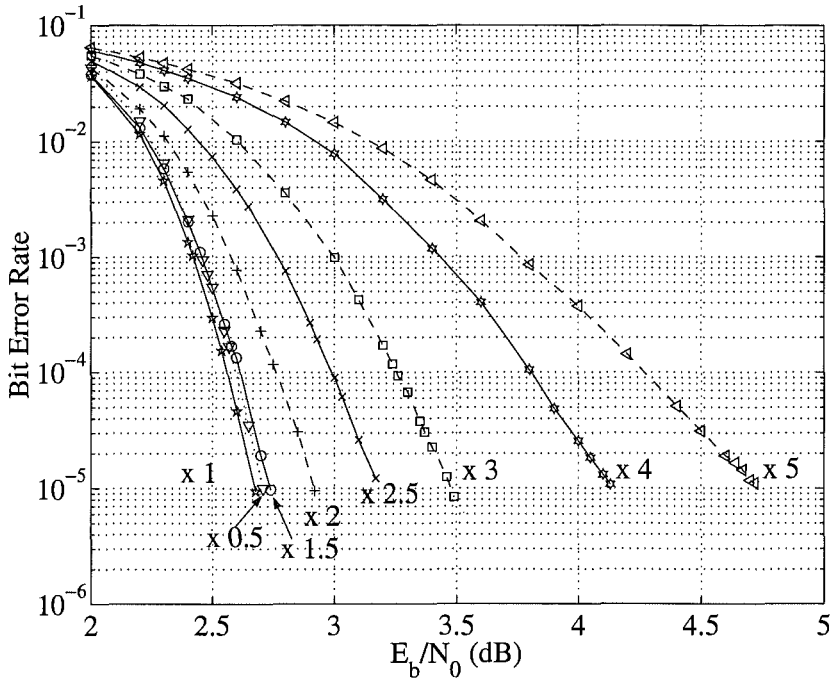


Figure 3.19: Performance of the $(64, 51, 6)^2$ block Turbo code using the approximation approach after 4 iterations for $p = 5$ and 16 test sequences. The case 2 β estimates are multiplied by the values indicated.

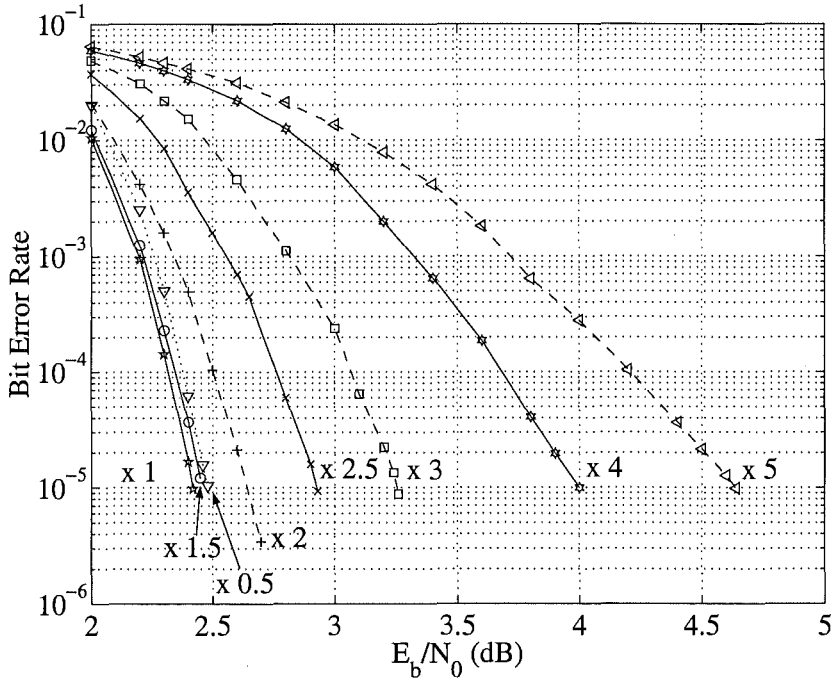


Figure 3.20: Performance of the $(64, 51, 6)^2$ block Turbo code using the approximation approach after 10 iterations for $p = 5$ and 16 test sequences. The case 2 β estimates are multiplied by the values indicated.

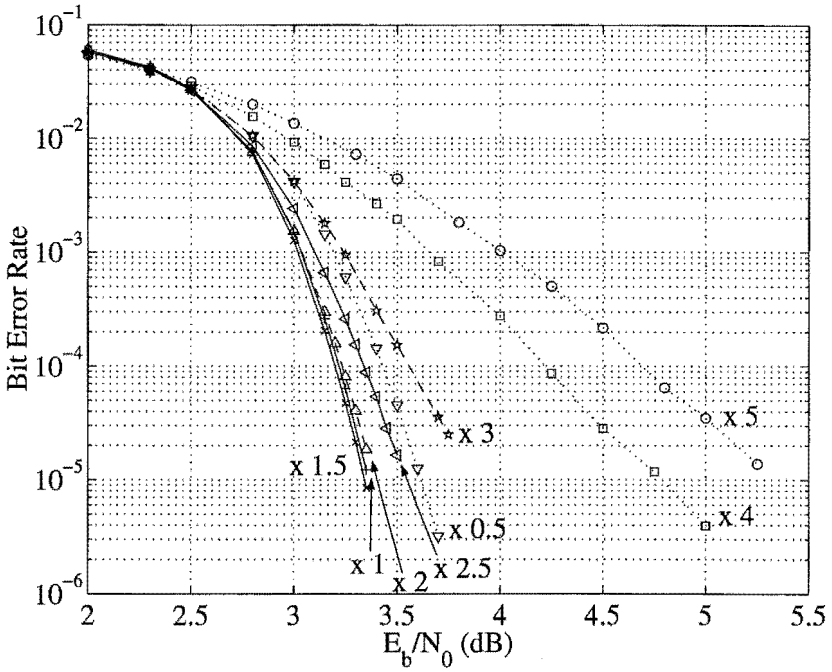


Figure 3.21: Performance of the $(64, 57, 4)^2$ block Turbo code using the approximation approach after 4 iterations for $p = 4$ and 16 test sequences. The case 2 β estimates are multiplied by the values indicated.

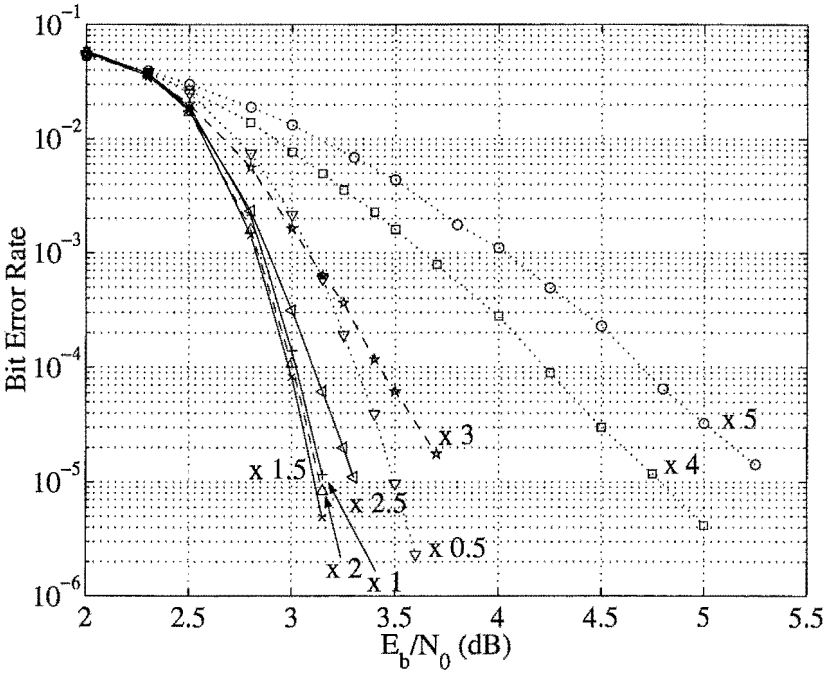


Figure 3.22: Performance of the $(64, 57, 4)^2$ block Turbo code using the approximation approach after 10 iterations for $p = 4$ and 16 test sequences. The case 2 β estimates are multiplied by the values indicated.

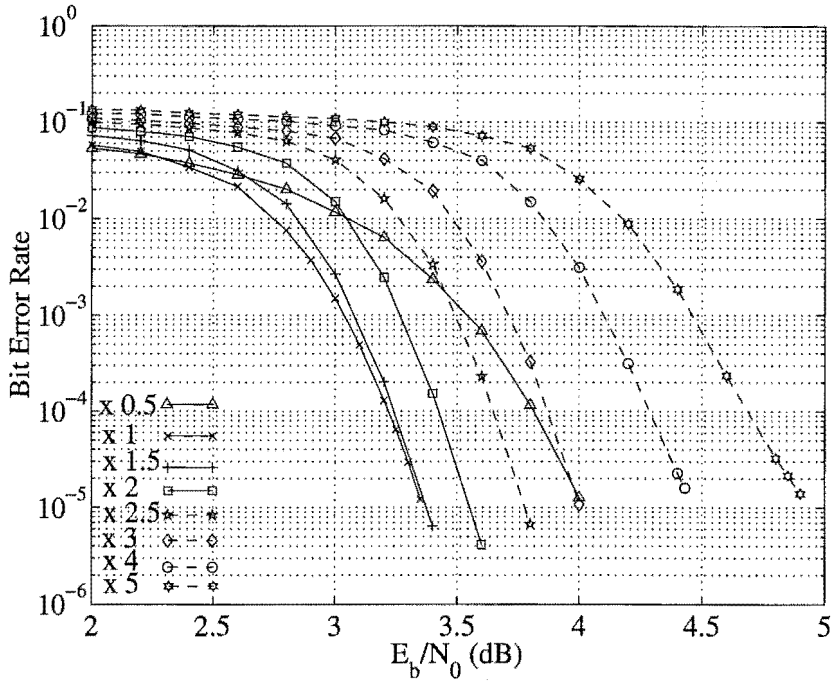


Figure 3.23: Performance of the $(64, 57, 4)^2$ block Turbo code using the approximation approach after 4 iterations for $p = 4$ and 16 test sequences. Different multipliers are used to scale $\boldsymbol{\alpha}$.

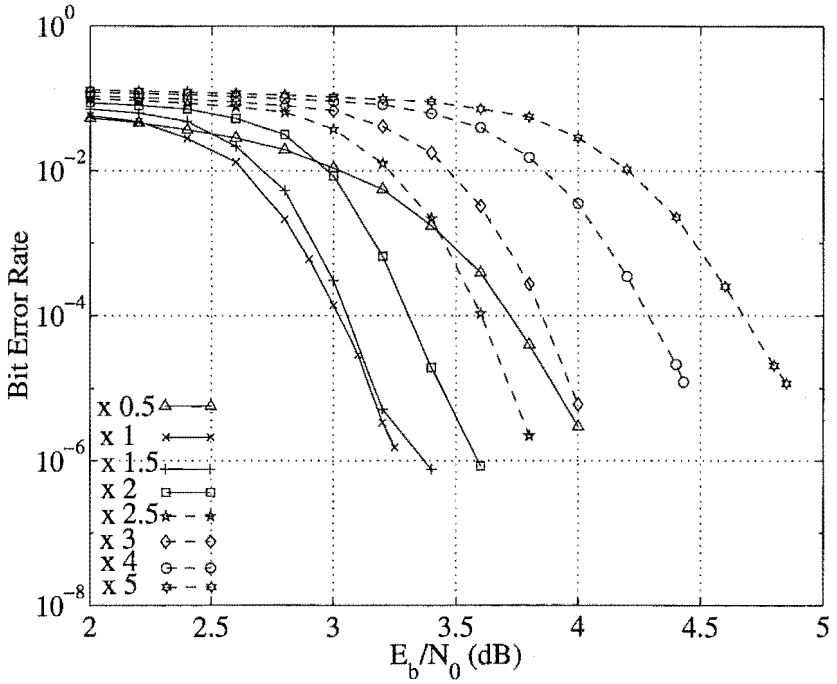


Figure 3.24: Performance of the $(64, 57, 4)^2$ block Turbo code using the approximation approach after 10 iterations for $p = 4$ and 16 test sequences. Different multipliers are used to scale $\boldsymbol{\alpha}$.

using the BCJR algorithm and *soft output Viterbi algorithm (SOVA)*[15]. In Fig. 3.25 it can be seen that the Gaussian approach to calculating the extrinsic information performs better than the a priori approach when the approximation approach of this chapter is used.

It was stated earlier that the variance and mean of the extrinsic information, $\boldsymbol{W}(q)$, are calculated using all positions in $\boldsymbol{W}(q)$. Fig. 3.26 shows results when the variance and mean of the extrinsic information are calculated using all positions and when just those positions calculated using a competing codeword, $\boldsymbol{C}$, are used. Results are for the $(64, 57, 4)^2$ block Turbo code using the approximation approach, $p = 4$ and 16 test sequences. Performance is very similar in both cases and so all positions will continue to be used.

In Fig. 3.27 the effect of decoding the component codes in different orders is shown. In Fig. 3.27 it can be seen that having the more powerful code[16] first is advantageous in early iterations, but both block Turbo codes will tend to the same solution after sufficient iterations. The advantage on early iterations may be due to the weaker code having to cope with fewer errors and the stronger code providing more reliable extrinsic information to the second decoder.

In appendix C bounds on the BER for block Turbo codes transmitted using BPSK on the AWGN are described [112, 113, 114] for *maximum likelihood (ML)* decoding. The performance of the sub-optimal decoding algorithms developed in this chapter can only tend to these bounds as the number of iterations and number of test sequences increases. However, these bounds give some indication of the possible performance at higher SNRs and lower BERs than those simulated. In Fig. 3.28[17] the best performance obtained for various block Turbo codes considered in this chapter after 10 iterations are shown against their ML bounds. The $(32, 21, 6)^2$ and $(128, 113, 6)^2$ block Turbo code simulations are for $p = 4$ and 16 test sequences. The $(64, 57, 4)^2$ block Turbo code simulation is for $p = 5$ and 32 test sequences (when the maximum allowed value is used). The $(64, 51, 6)^2$ block Turbo code simulation is for $p = 6$ and 32 test sequences. As expected these bounds are bad at low SNRs. By

[15]The SOVA performs best when a precomputed sequence scales the extrinsic information [21].

[16]Has a larger minimum Hamming distance and has better BER performance when simulated in a block Turbo code with identical row and column component codes.

[17]Only 47 (out of 143533) block Turbo code blocks were found in error at 2.3 dB for the $(64, 51, 6)^2$ simulation.

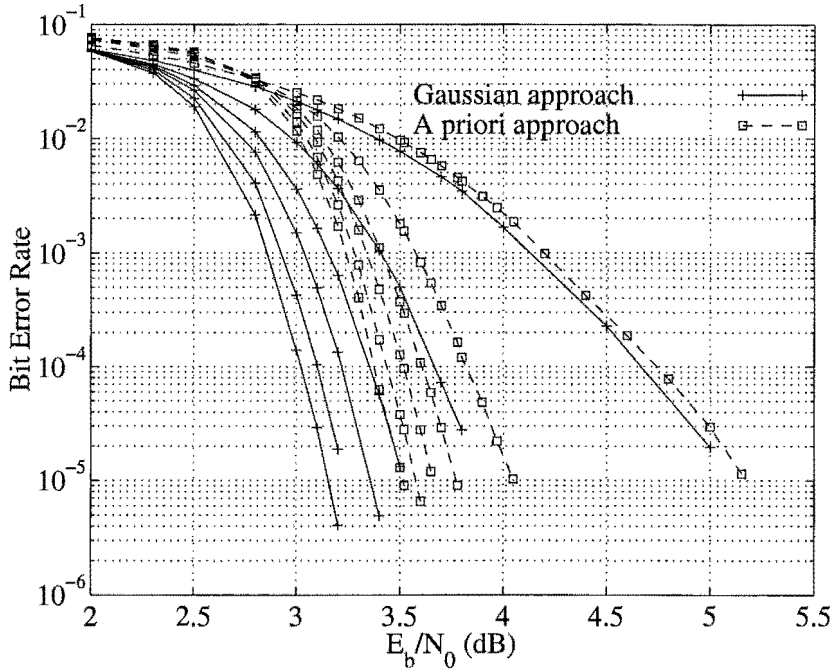


Figure 3.25: Performance of the $(64, 57, 4)^2$ block Turbo code using the approximation approach, $p = 4$ and 16 test sequences. The performance when the extrinsic information is treated as a Gaussian random variable or as a priori information is shown.

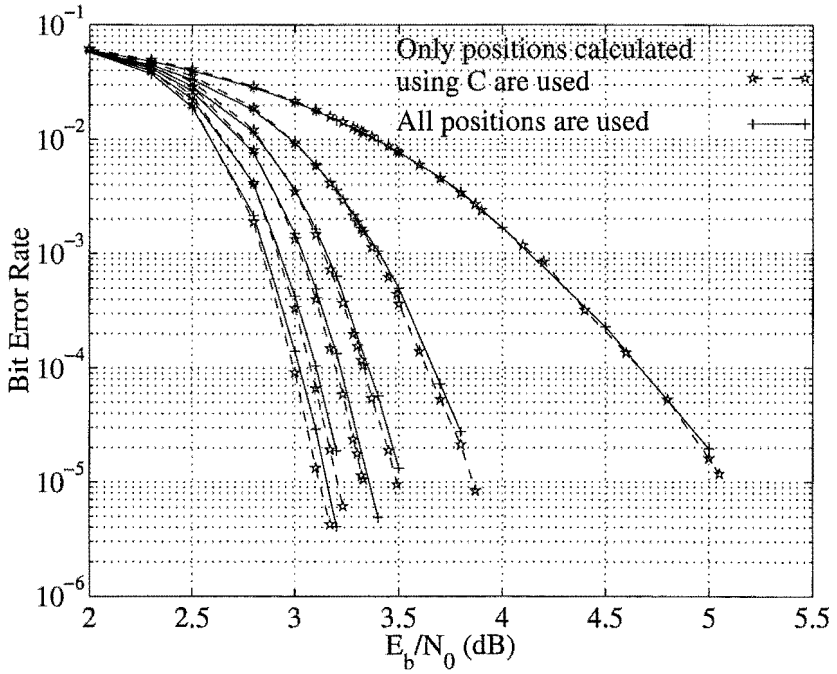


Figure 3.26: Performance of the $(64, 57, 4)^2$ block Turbo code using the approximation approach, $p = 4$ and 16 test sequences. The variance and mean of the extrinsic information are calculated using all positions or just those calculated using $\boldsymbol{C}$.

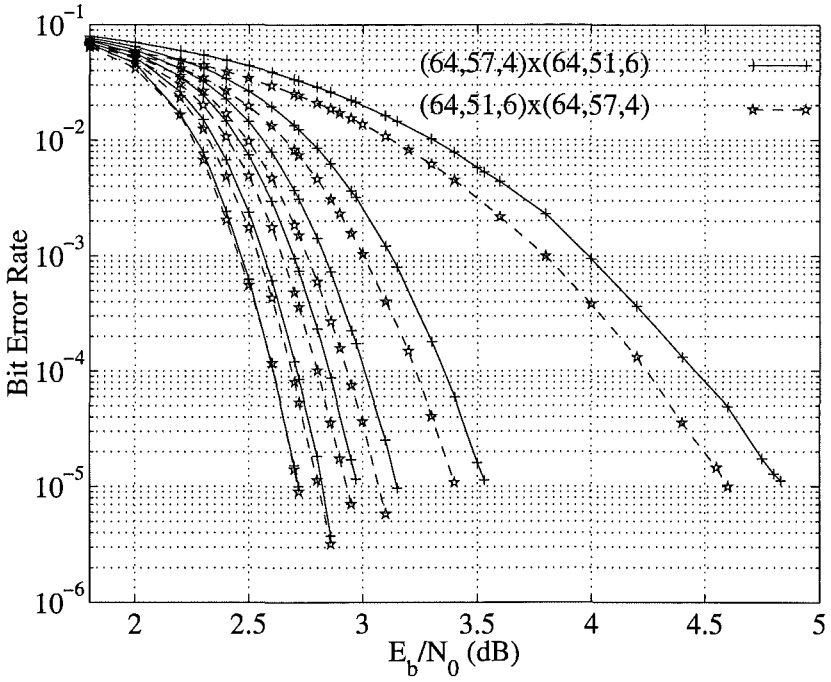


Figure 3.27: Performance of the $(64, 57, 4) \times (64, 51, 6)$ block Turbo code against the $(64, 51, 6) \times (64, 57, 4)$ block Turbo code using the approximation approach, $p = 5$ and 16 test sequences.

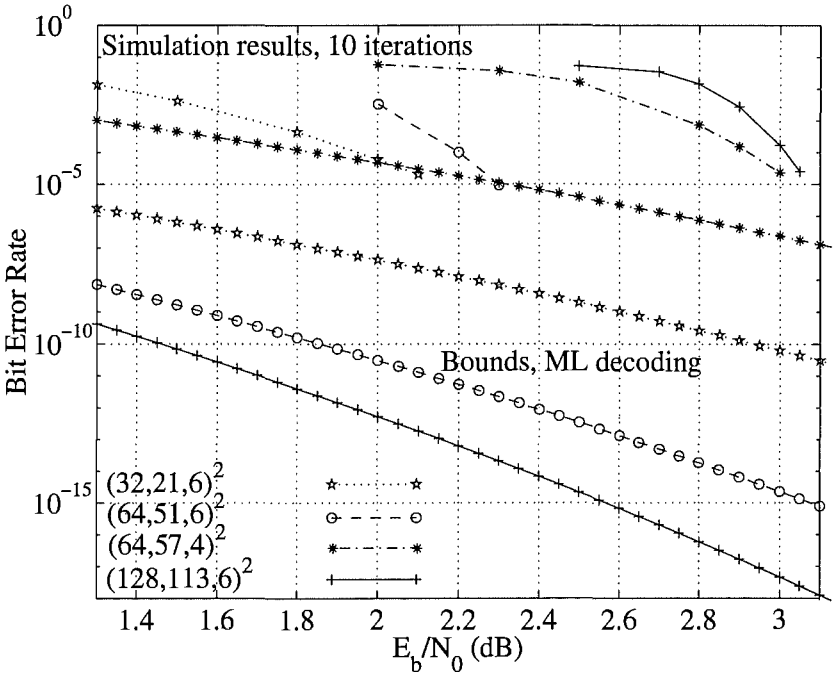


Figure 3.28: Performance after 10 iterations of the $(32, 21, 6)^2$, $(64, 57, 4)^2$, $(64, 51, 6)^2$ and $(128, 113, 6)^2$ block Turbo codes using the approximation approach against their ML bounds for BPSK/ QPSK on the AWGN channel.

comparing the simulation results and bound for the $(64, 57, 4)^2$ block Turbo code, it can be seen that performance will be dominated by the minimum Hamming distance codewords at BERs below 10^{-7}.

The simulation results in Fig. 3.28 are now compared to capacity (for signals transmitted using QPSK on the AWGN channel). The $(32, 21, 6)^2$ block Turbo code has a rate of 0.43 (or 0.86 data bits per QPSK symbol) and is approximately $2.36dB$ from capacity at a BER of 10^{-5} [51]. The $(64, 57, 4)^2$ block Turbo code has a rate of 0.793 (or 1.586 data bits per QPSK symbol) and is approximately $1.04dB$ from capacity at a BER of 10^{-5} [51]. The $(64, 51, 6)^2$ block Turbo code has a rate of 0.635 (or 1.27 data bits per QPSK symbol) and is approximately $1.39dB$ from capacity at a BER of 10^{-5} [51]. Finally, the $(128, 113, 6)^2$ block Turbo code has a rate of 0.779 (or 1.56 data bits per QPSK symbol) and is approximately $1.17dB$ from capacity at a BER of 10^{-5} [51]. The two highest rate block Turbo codes transmit ≈ 1.5 data bits per QPSK symbol and are a little over $1dB$ from capacity at a BER of 10^{-5}. This has been achieved using a moderate complexity adaptive sub-optimal SISO decoder and iterative decoding. Performance could be further improved by using more test sequences or by using a better SILO decoder.

All the simulation results presented in this chapter show the BER of the data bits. However, in some applications the FER is important. A frame is said to be in error if at least one data bit is in error. The FER of the $(64, 57, 4)^2$ block Turbo code using the precomputed and approximation approaches, $p = 4$ and 2^p test sequences is shown in Fig. 3.29. One block Turbo code block is transmitted per frame (meaning 3249 data bits). As can be seen the approximation approach has a better FER than the precomputed approach. The curves for the approximation approach are significantly steeper indicating that the performance improvement over the precomputed approach will increase as the FER decreases. It was also found in simulations that this code preforms better than the precomputed approach in terms of BER.

3.4.2 Distance Based Approach to Estimating β

This subsection presents simulation results for the adaptively estimated α of section 3.2 and the distance based approach to β of section 3.3. This will be called the

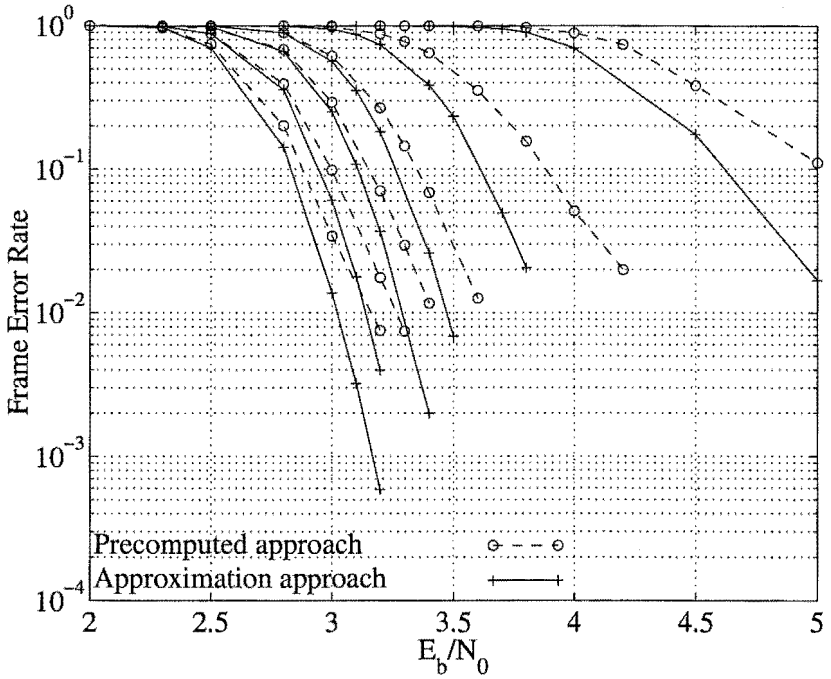


Figure 3.29: Frame error rate of the $(64, 57, 4)^2$ block Turbo code using the precomputed and approximation approaches, $p = 4$ and 2^p test sequences.

distance approach. All simulations use 2^p test sequences and $p = 4$, which means $p' = 4$ and $weight(p) = 4$. First the $(64, 57, 4)^2$ block Turbo code is transmitted. The component codes have $t = 1$ and $d_{H,min} = 4$. Therefore, using (3.29) and (3.32) it is found that $d_H(\boldsymbol{D}, \boldsymbol{Y}(q)) \leq 6$ and $N_s = 11$ for set 1, and using (3.33) it is found that $N_s = 9$ for set 2. The resulting simulated error performance is shown in Fig. 3.30 for set 1. An improvement in performance is obtained after each iteration compared to both the approximation approach and the precomputed approach of [90]. After 10 iterations an improvement of $\approx 0.05dB$ is achieved over the approximation approach at a BER of 3×10^{-5} and an improvement of $\approx 0.13dB$ over the precomputed approach of [90] at a BER of 2×10^{-5}. The set 1 and 2 results are compared in Fig. 3.31. The performance is similar for both sets, but set 2 has a lower complexity.

Now the $(64, 51, 6)^2$ block Turbo code is transmitted. The component codes have $t = 2$ and $d_{H,min} = 6$. Therefore, using (3.29) and (3.32) it is found that $d_H(\boldsymbol{D}, \boldsymbol{Y}(q)) \leq 7$ and $N_s = 13$ for set 1, and using (3.33) it is found that $N_s =$

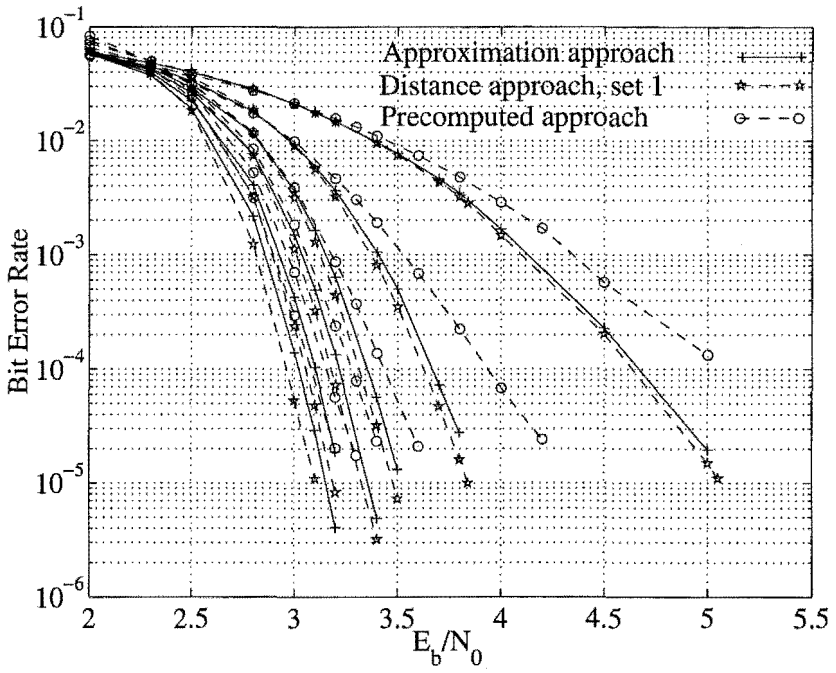


Figure 3.30: Results for the $(64, 57, 4)^2$ block Turbo code using the approximation approach, distance approach (set 1) and the precomputed approach of [90] are shown. All 2^p test sequences are used for $p = 4$.

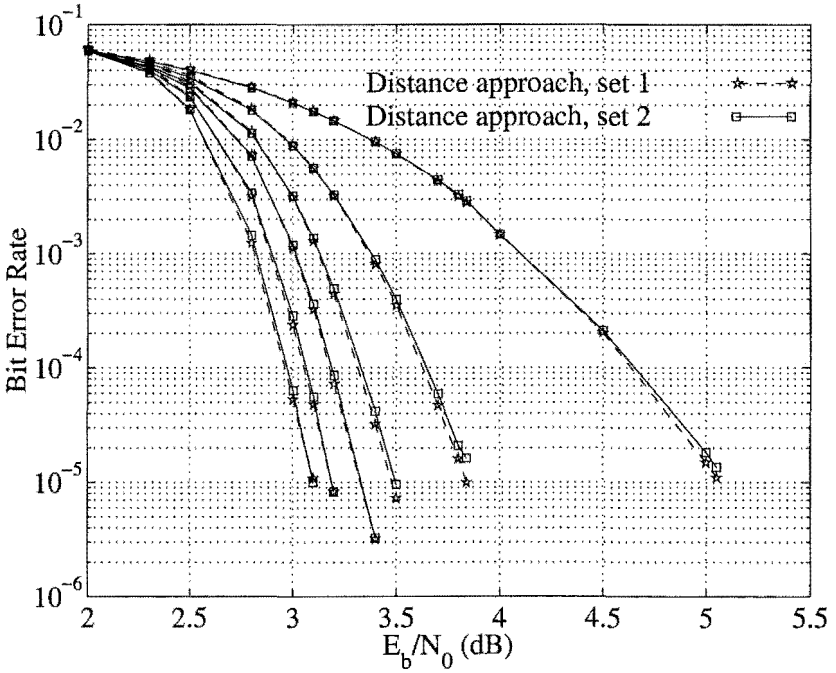


Figure 3.31: Results for the $(64, 57, 4)^2$ block Turbo code using the distance approach with set 1 and 2 are shown. All 2^p test sequences are used for $p = 4$.

12 for set 2. Simulation results are shown for set 1 in Fig. 3.32[18]. In this case the distance approach performs $\approx 0.1dB$ worse than the approximation approach after one iteration, but $\approx 0.7dB$ better than the precomputed approach of [90] at a BER of $\approx 10^{-5}$. After two iterations it performs $\approx 0.07dB$ worse than the approximation based approach, but $\approx 0.5dB$ better than the precomputed approach of [90] at a BER of $\approx 10^{-5}$. In subsequent iterations the distance approach performs approximately the same as the precomputed approach of [90]. For this code slightly better performance is obtained in later iterations if $N_s' - 1$ LRPs are used when $d_0 y_0(q) = +1$ in the overall parity position. However, the performance is still worse than the approximation approach to estimating $\beta_j(q)$. The set 1 and 2 results are compared in Fig. 3.33[19]. The performance is similar for both sets, but set 2 has a lower complexity.

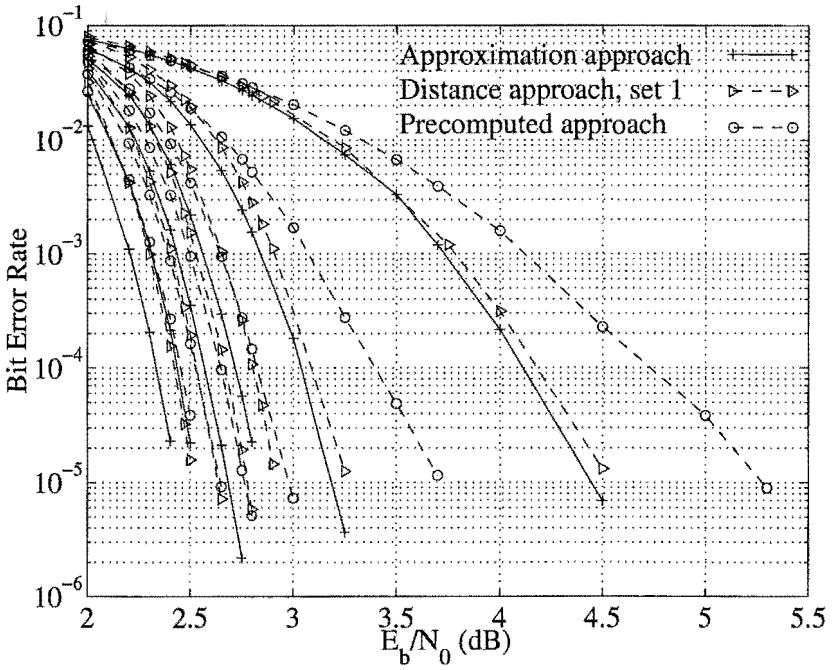


Figure 3.32: Results for the $(64, 51, 6)^2$ block Turbo code using the approximation approach, distance approach (set 1) and the precomputed approach of [90] are shown. All 2^p test sequences are used for $p = 4$.

Although the distance approach improves performance for the $(64, 57, 4)^2$

[18]Only 80 (out of 157211) block Turbo code blocks were found in error at 2.65 dB after 6 iterations for the distance approach (set 1).

[19]Only 80 (out of 157211) block Turbo code blocks were found in error at 2.65 dB after 6 iterations for the distance approach (set 1).

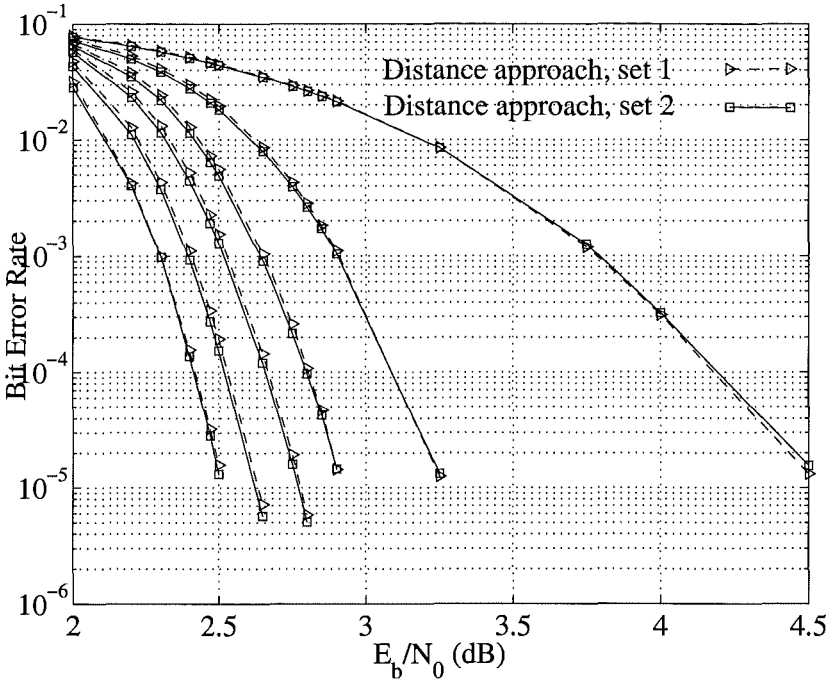


Figure 3.33: Results for the $(64, 51, 6)^2$ block Turbo code using the distance approach with set 1 and 2 are shown. All 2^p test sequences are used for $p = 4$.

product code compared to the approximation approach, it does not perform as well for the $(64, 51, 6)^2$ product code. This lack of code flexibility means the approximation approach is a more reliable and flexible approach and so it will be used in the remainder of this thesis. It is possible that the distance approach could perform better than the approximation based approach for the $(64, 51, 6)^2$ block Turbo code if a different set of combinations, $\{\xi\}$, was used. The difference in performance may be due to the fact that the $(63, 57, 3)$ unextended BCH component code is a perfect code, while the $(63, 51, 5)$ unextended BCH component code is a quasi perfect code.

In many applications the FER is important. In Fig. 3.34 the FER when using the approximation approach and the distance approach (with set 1) are compared for the $(64, 57, 4)^2$ block Turbo code (using $p = 4$ and 2^p test sequences). One block Turbo code block is transmitted per frame (meaning 3249 data bits). A frame is said to be in error if at least one data bit is in error. As can be seen the distance approach works better in terms of the BER (Fig. 3.30) and FER for this code.

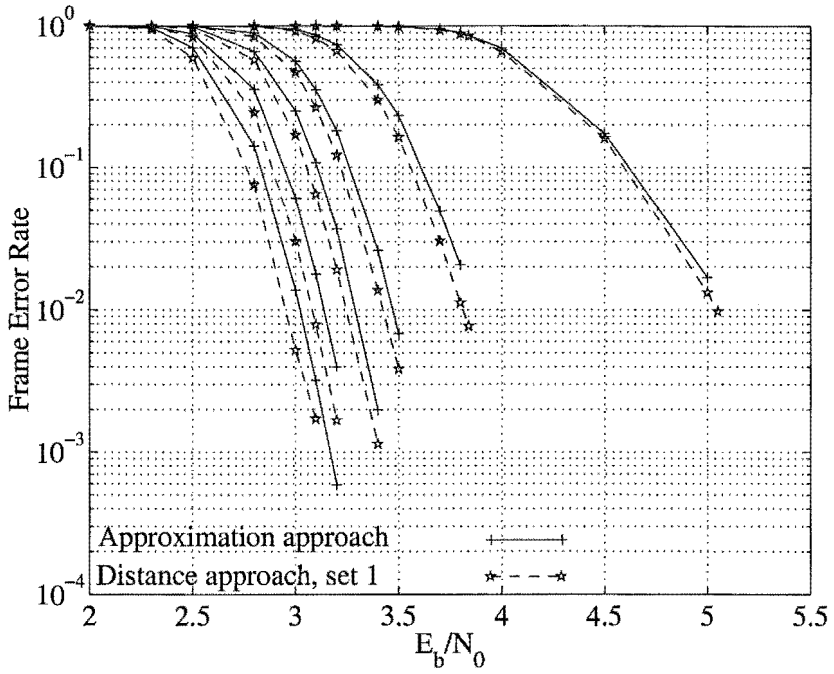


Figure 3.34: Frame error rate of the $(64, 57, 4)^2$ block Turbo code using the approximation approach, the distance approach with set 1, $p = 4$ and 2^p test sequences.

3.5 Conclusions

In [90] a method to calculate soft information from a SILO decoder was developed and used to iteratively decode block Turbo codes. Two precomputed sequences, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, were used which needed to be reoptimised using repeated simulations for different codes and applications. In this chapter $\boldsymbol{\alpha}$ has been derived by considering the extrinsic information to be a Gaussian random variable. It is adaptively estimated in simulations and so does not need to be reoptimised using repeated simulations. Since it is calculated on a block by block basis, it can adapt to varying conditions.

In this chapter $\beta_j(q)$ was written as a sum of soft inputs, but this could only be calculated when a competing codeword was available. Two adaptive approaches to estimating $\beta_j(q)$ when no competing codewords are available were developed. The first approach was based on approximation and performs well with a variety of codes. This approach to adaptively estimating $\beta_j(q)$ improves the performance on the AWGN channel by $\approx 0.5 - 1.4$dB after one iteration and $\approx 0.25 - 1.1$dB at a BER of 10^{-5} after two iterations depending on the block Turbo code transmitted. This improvement is achieved with only a small increase in complexity. It can also

mean fewer iterations are required to achieve a given BER or FER.

A second estimate, $\hat{\beta}_j(q)$, of $\beta_j(q)$ has also been developed based on the distance properties of the component codes. This estimate was obtained by considering the problem to be that of finding the positions where $c_j \neq d_j$. It performs better than the approximation based approach to estimating $\beta_j(q)$ for the $(64, 57, 4)^2$ block Turbo code, but worse for the $(64, 51, 6)^2$ block Turbo code. In both cases it performs better than or approximately the same as the precomputed approach of [90]. The set of combinations of positions the distance based $\beta_j(q)$ sums affects the performance. There is still work to be done on finding the best set of combinations. Therefore, the approximation approach to adaptively estimating $\beta_j(q)$ is more flexible and will be used in the remainder of this thesis. Both approaches can be used with other SILO decoders. In fact using other SILO decoders (such as that in [26]) with these adaptive approaches may further improve performance over [90].

In conclusion, this chapter has developed approaches to adaptively estimating $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ that perform better than (or as well as) the precomputed approach of [90] for a variety of codes and channel conditions. The approximation approach obtained performance within $1.04dB$ of capacity after 10 iterations at a BER of 10^{-5} for the $(64, 57, 4)^2$ block Turbo code on the AWGN channel. This code transmitted 1.586 data bits per QPSK symbol and transmitted 2048 QPSK symbols per code block (equating to 3249 data bits). Therefore, a moderate complexity decoding algorithm has been developed for a moderate block length, high code rate block Turbo code. For one code it has obtained a low BER (10^{-5}) when only $1.04dB$ from capacity.

Chapter 4

Multilevel Coding and Iterative Multistage Decoding

4.1 Introduction

Multilevel coding [16, 52, 82, 119] was introduced in chapter 2 as a technique which allows a complicated code to be created as a hierarchy of simple component codes. Multilevel coding combines error correction coding and modulation by using the component codes to choose the transmitted constellation point. They are usually decoded in a sequential manner using a *multistage decoder (MSD)*, since decoding the overall multilevel code can be prohibitively complex. Traditionally hard decisions are passed to subsequent levels to choose a subset of constellations points for the next level's decoder.

Significant improvement can be made by passing reliability information rather than hard decisions to previous and subsequent levels, and by making more than one pass through the MSD (iterative multistage decoding) [27, 71, 73, 74, 121]. Many of the previous approaches do not pass reliability information back to all previous levels [27, 73, 74]. In addition previous approaches have treated the reliability (extrinsic) information as a priori information rather than as a Gaussian random variable [27, 42, 53, 71, 73, 74]. Interleaving has been used between some of the component encoders and the channel mapping in [27, 73, 74, 121]. Interleaving can increase the encoding and decoding delay, and so should be avoided when not needed.

In chapter 3 performance was improved by treating the extrinsic information

as a Gaussian random variable, rather than as a priori information. Therefore, in this chapter a new iterative MSD is derived by treating the extrinsic information as a Gaussian random variable rather than as a priori information. This results in a different soft input to and soft output from the component decoders. Unlike [27, 73, 74] the iterative MSD developed here passes extrinsic information back to all previous decoders[1]. In addition the iterative MSD is considered with and without interleaving and different options for passing the extrinsic information are considered.

In [27, 73, 74, 121] trellis based decoding algorithms are used to decode the component codes. However, soft decision decoding of long block codes using a trellis can be prohibitively complex, especially when they are viewed as component codes in a concatenated code such as a Block Turbo code or in a multilevel code. Therefore, sub-optimal SISO decoding algorithms are considered when using long block codes.

The non-adaptive sub-optimal SISO decoding algorithm of [79, 90] (described in section 2.3) has some constraints when used to decode the component codes in an iterative MSD. Recall that in [90] two sequences of precomputed scaling parameters, α and β, are used when calculating and using the extrinsic information[2]. These sequences need to be reoptimised for different codes or applications using repeated simulations for optimum performance and do not change with varying channel conditions. In an iterative MSD the magnitude of the extrinsic information passed to other levels must be correct. If the algorithm of [79, 90] is to be used, then a joint optimization of the α and β sequences used on every level is required (using repeated simulations). This does not allow for varying conditions, and good and bad blocks are treated the same way. Another restriction can be seen when transmitting a QAM signal. In [77, 79] the received signal is mapped so that the input to each level is centred on two possible values[3]. Information about the labelling bits from other levels is not produced when this technique is used, this information is required

[1] Extrinsic information may be considered as the information gained from a decoding. Therefore, if a level is uncoded, extrinsic information is not available for that level and so cannot be sent to other levels.

[2] The sequence α scales the extrinsic information to compensate for the difference in variance between the received signal and the extrinsic information. The sequence β is used in estimating the extrinsic information when it cannot be calculated directly.

[3] This means that the algorithm must be modified (a new mapping developed) when different constellations (with more than two points) or partitioning strategies are used.

by the iterative MSD discussed in section 4.2.

In chapter 3 two adaptive algorithms were developed based on [79, 90]. These algorithms adaptively estimate the sequences α and β. This chapter extends the notation of these algorithms so that they can be used in an iterative MSD and with different constellations. It is not necessary to change the definitions of α and β, just their notation. The adaptive approaches eliminate the constraints of [79, 90]. Since α and β are adaptively estimated, they do not require optimization using repeated simulations. These estimates can also adapt to varying conditions, and treat good and bad blocks differently. The mapping is avoided by calculating the soft output using constellation points, which provides the required information on the labelling bits from other levels. As a result these algorithms can be used with a variety of codes, modulation schemes, conditions and applications (such as iterative multistage decoding) without modification.

This chapter develops decoding algorithms for the *additive white Gaussian noise (AWGN)* channel[4]. In section 4.2 an alternative iterative MSD to [27, 71, 73, 74, 121] is developed. Section 4.3 extends the notation of the adaptive decoding algorithms of chapter 3 for use with different constellations and in the iterative MSD of section 4.2. Simulations are discussed and results given in section 4.4. A performance bound is discussed in section 4.5 and conclusions are drawn in section 4.6.

4.2 Iterative Multistage Decoder

Multilevel codes encode a hierarchy of component codes on sequences of signal set partitions. Each component code labels a signal set partition and collectively they choose constellation points for transmission. They are normally decoded by a MSD as in [16, 82].

A traditional MSD [16, 82] passes hard decisions from the decoder on the current level to decoders on all subsequent levels. However, better performance can be achieved by passing reliability information to both previous and subsequent levels in the MSD as shown in [27, 71, 73, 74]. In this section a new approach to iterative

[4]Independent noise samples are assumed.

multistage decoding is derived that treats the extrinsic information as a Gaussian random variable (Gaussian approach). The soft input and output derived here can be used with any SISO decoding algorithm such as the BCJR algorithm [6], the *soft output Viterbi algorithm (SOVA)* [41], the non-adaptive sub-optimal decoding algorithm of [79, 90] or the adaptive algorithms developed in chapter 3 (with the extended notation of section 4.3).

This section considers the general case of an L-level multilevel code with a V_i-stage concatenated code[5] on level i transmitted on the memoryless AWGN channel. The iterative decoding of a level can be considered as a pipeline of decoders[6] that spans all iterations of the iterative MSD. One stage in the pipeline is shown in Fig. 4.1. The number of completed decoding stages in the pipeline on level i is denoted q_i. If each level's code is decoded once before decoding the next level, then there are V_i decodings on level i per iteration of the MSD. To keep the notation simple binary partitions are assumed on each level of the signal set during the derivation.

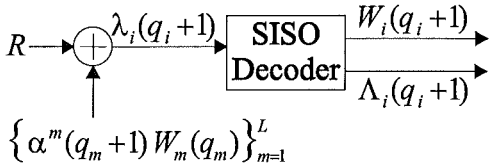


Figure 4.1: One decoding stage in the level-i pipeline of the iterative MSD, where $\{\boldsymbol{\alpha}^m(q_m+1)\boldsymbol{W}_m(q_m)\}_{m=1}^L$ is the set of scaled extrinsic information from the last completed decoding on each of the L levels of the multilevel code. Extrinsic information from level i is only included in the set if $V_i > 1$.

The soft input *log-likelihood ratio (LLR)* to the current, $(q_i+1)^{th}$, decoder in the level i pipeline is defined as

$$\lambda'_{ij}(q_i+1) = \log\left(\frac{P(e_{ij}=+1|r_j,\{w_{mj}(q_m)\}_{m=1}^L)}{P(e_{ij}=-1|r_j,\{w_{mj}(q_m)\}_{m=1}^L)}\right) \tag{4.1}$$

where e_{ij} is the j^{th} bit[7] in the level-i codeword, r_j is the j^{th} received demodulated symbol, $w_{mj}(q_m)$ is the extrinsic information for the j^{th} position in the level-m codeword from the last, $(q_m)^{th}$, decoder in the level-m pipeline and $\{w_{mj}(q_m)\}_{m=1}^L$

[5] A V_i-stage concatenated code consists of V_i component encoders. These may be in a parallel, serial, product/ block Turbo or hybrid structure.

[6] In practice only V_i decoders are used on level i in an iterative or recursive structure.

[7] For the purposes of computing likelihoods all bits are mapped from $\{0,1\}$ to $\{-1,+1\}$.

is the set of extrinsic information from all levels for the j^{th} transmitted symbol. Note that extrinsic information for levels 1 to $(i-1)$ is from the current iteration of the MSD, while levels $(i+1)$ to L must use extrinsic information obtained during the previous iteration of the MSD. Assuming $V_i > 1$, the first decoder on level i uses extrinsic information for level i from the previous iteration of the MSD, but subsequent decoders on level i use extrinsic information for level i from the current iteration. Otherwise extrinsic information for level i is not included in the set. By conditioning the probabilities of (4.1) on the extrinsic information from all levels the decoders in the level-i pipeline can use reliability information from all levels to make their decisions. The structure of the proposed iterative MSD is shown in Fig. 4.2.

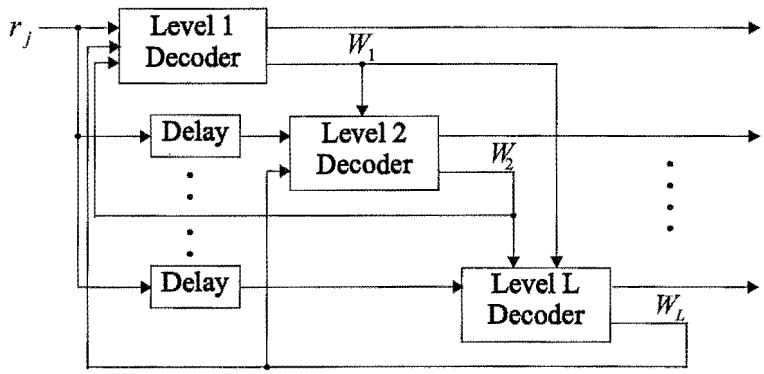


Figure 4.2: L-level iterative multistage decoder.

The received signal and the extrinsic information from all levels are assumed to be independent Gaussian random variables and the constellation points and encoded bits from all levels are assumed to be equiprobable. Using the approximation $\log(e^x + e^y) \approx \max(x, y)$ [43] the soft input of (4.1) can be approximated by

$$\lambda'_{ij}(q_i+1) \approx \lambda_{ij}(q_i+1) = \max_{\{s_j^a | e_{ij}=+1\}} \left(\log \left(p(r_j|s_j^a) \prod_{m=1}^{L} p(w_{mj}(q_m)|e_{mj}) \right) \right) - \max_{\{s_j^a | e_{ij}=-1\}} \left(\log \left(p(r_j|s_j^a) \prod_{m=1}^{L} p(w_{mj}(q_m)|e_{mj}) \right) \right) \quad (4.2)$$

where $\{s_j^a | e_{ij} = +1\}$ denotes the set of constellation points with the level-i label $e_{ij} = +1$ and $\{s_j^a | e_{ij} = -1\}$ the set with $e_{ij} = -1$.

The constellation point in the set $\{s_j^a | e_{ij} = +1\}$ found by the maximum function in (4.2) is denoted s_j^{i+} and that found in the set $\{s_j^a | e_{ij} = -1\}$ is denoted s_j^{i-} . The soft input of (4.2) can then be written as

$$\lambda_{ij}(q_i + 1) = \frac{|r_j - s_j^{i-}|^2}{2\sigma^2_{r_j|s_j^{i-}}} - \frac{|r_j - s_j^{i+}|^2}{2\sigma^2_{r_j|s_j^{i+}}}$$
$$+ \sum_{m=1}^{L} \left(\frac{|w_{mj}(q_m) - \mu_{w_{mj}(q_m)|e_{mj}^{i-}}|^2}{2\sigma^2_{w_{mj}(q_m)|e_{mj}^{i-}}} - \frac{|w_{mj}(q_m) - \mu_{w_{mj}(q_m)|e_{mj}^{i+}}|^2}{2\sigma^2_{w_{mj}(q_m)|e_{mj}^{i+}}} \right) \quad (4.3)$$

where e_{mj}^{i+} is the bit on level-m labelling s_j^{i+} and similarly e_{mj}^{i-} denotes the bit on level-m labelling s_j^{i-}. Also $\sigma^2_{r_j|s_j^{i\pm}}$ is the variance of r_j given $s_j^{i\pm}$ was transmitted, and $\mu_{w_{mj}(q_m)|e_{mj}^{i\pm}}$ is the mean and $\sigma^2_{w_{mj}(q_m)|e_{mj}^{i\pm}}$ the variance of $w_{mj}(q_m)$ given $e_{mj}^{i\pm}$ was transmitted. In practise the following is used

$$\lambda_{ij}(q_i + 1) = \frac{|r_j - s_j^{i-}|^2 - |r_j - s_j^{i+}|^2}{2\sigma^2_{\boldsymbol{R}}}$$
$$+ \sum_{m=1}^{L} \left(\frac{|w_{mj}(q_m) - \mu_{\boldsymbol{W}_m(q_m)} e_{mj}^{i-}|^2 - |w_{mj}(q_m) - \mu_{\boldsymbol{W}_m(q_m)} e_{mj}^{i+}|^2}{2\sigma^2_{\boldsymbol{W}_m(q_m)}} \right) \quad (4.4)$$

which simplifies to

$$\lambda_{ij}(q_i + 1) = \frac{|r_j - s_j^{i-}|^2 - |r_j - s_j^{i+}|^2}{2\sigma^2_{\boldsymbol{R}}} + \sum_{m=1}^{L} \left(\frac{\mu_{\boldsymbol{W}_m(q_m)}}{\sigma^2_{\boldsymbol{W}_m(q_m)}} w_{mj}(q_m)(e_{mj}^{i+} - e_{mj}^{i-}) \right) \quad (4.5)$$

where $\sigma^2_{\boldsymbol{R}}$ is the variance of the received signal/ noise and is either estimated once per received multilevel code block or is (assumed) known. $\mu_{\boldsymbol{W}_m(q_m)}$ is the mean and $\sigma^2_{\boldsymbol{W}_m(q_m)}$ is the variance of the absolute value of the extrinsic information from the $(q_m)^{th}$ decoding on level-m as used in [14]. The ratio of the extrinsic information's mean to its variance scales the extrinsic information so it can be used by all other decoders in the MSD. The mean of the extrinsic information, $\mu_{\boldsymbol{W}_m(q_m)}$, is required since the extrinsic information may not be centred about ± 1 and both e_{mj}^{i+} and e_{mj}^{i-} take the values ± 1. The variance of the extrinsic information is estimated over a component code block after each component decoding using the sample variance. The sample variance of the absolute value of the extrinsic information from the $(q_m)^{th}$ decoder in the level-m pipeline is obtained similarly to [14] as

$$\sigma^2_{\boldsymbol{W}_m(q_m)} = \frac{1}{N_m - 1} \sum_{l=1}^{N_m} (|w_{ml}(q_m)| - \mu_{\boldsymbol{W}_m(q_m)})^2 \quad (4.6)$$

where N_m is the length of the level-m code block and $\mu_{\boldsymbol{W}_m(q_m)}$ is the sample mean of the extrinsic information from the $(q_m)^{th}$ decoder in the level-m pipeline and is given by [14]

$$\mu_{\boldsymbol{W}_m(q_m)} = \frac{1}{N_m} \sum_{l=1}^{N_m} |w_{ml}(q_m)| \tag{4.7}$$

Now consider the soft output LLR from the current, $(q_i+1)^{th}$, decoder in the level-i pipeline, which is defined as

$$\Lambda'_{ij}(q_i+1) = \log\left(\frac{P(e_{ij}=+1|\boldsymbol{R}, \{\boldsymbol{W}_m(q_m)\}_{m=1}^{L})}{P(e_{ij}=-1|\boldsymbol{R}, \{\boldsymbol{W}_m(q_m)\}_{m=1}^{L})}\right) \tag{4.8}$$

where $\boldsymbol{W}_m(q_m)$ is the vector of extrinsic information from the $(q_m)^{th}$ decoder in the level-m pipeline, $\{\boldsymbol{W}_m(q_m)\}_{m=1}^{L}$ denotes the set of extrinsic information vectors from all levels corresponding to their last decoding and $\boldsymbol{R}$ is the demodulated received signal vector. The vectors have the same length as the level-i codeword being decoded. The soft output can also be written as

$$\Lambda'_{ij}(q_i+1) = \log\left(\frac{\sum_{\{\boldsymbol{S}^a|e_{ij}=+1\}} P(\boldsymbol{S}^a|\boldsymbol{R}, \{\boldsymbol{W}_m(q_m)\}_{m=1}^{L})}{\sum_{\{\boldsymbol{S}^b|e_{ij}=-1\}} P(\boldsymbol{S}^b|\boldsymbol{R}, \{\boldsymbol{W}_m(q_m)\}_{m=1}^{L})}\right) \tag{4.9}$$

where $\{\boldsymbol{S}^a|e_{ij}=+1\}$ denotes the set of sequences of constellation points with the j^{th} bit of the level-i label equal to $e_{ij}=+1$ and $\{\boldsymbol{S}^b|e_{ij}=-1\}$ denotes the set with $e_{ij}=-1$.

All sequences of constellation points and all codewords on each level are assumed to be equiprobable. In addition the received signal vector and the extrinsic information vectors from all levels are assumed to be independent Gaussian random variables. Using the approximation $\log(e^X+e^Y) \approx \max(X,Y)$ [43] the soft output of (4.9) can then be approximated by

$$\begin{aligned}\Lambda'_{ij}(q_i+1) \approx \Lambda_{ij}(q_i+1) = &\max_{\{\boldsymbol{S}^a|e_{ij}=+1\}} \left(\log\left(p(\boldsymbol{R}|\boldsymbol{S}^a)\prod_{m=1}^{L} p(\boldsymbol{W}_m(q_m)|\boldsymbol{E}_m^a)\right)\right) \\ &- \max_{\{\boldsymbol{S}^b|e_{ij}=-1\}} \left(\log\left(p(\boldsymbol{R}|\boldsymbol{S}^b)\prod_{m=1}^{L} p(\boldsymbol{W}_m(q_m)|\boldsymbol{E}_m^b)\right)\right)\end{aligned} \tag{4.10}$$

where $\boldsymbol{E}_m^a$ and $\boldsymbol{E}_m^b$ are the vectors of bits from level-m used in the labels that choose the sets of constellation points $\boldsymbol{S}^a$ and $\boldsymbol{S}^b$, and may not be level-m codewords. The sequence of constellation points in the set $\{\boldsymbol{S}^a|e_{ij}=+1\}$ found by the maximum function in (4.10) is denoted $\boldsymbol{S}^{ij+} = (s_1^{ij+}, \cdots, s_n^{ij+})$ and that in the set $\{\boldsymbol{S}^b|e_{ij}=-1\}$

is denoted $\boldsymbol{S}^{ij-} = (s_1^{ij-}, \cdots, s_n^{ij-})$, where n is the length of the level-i codeword being decoded. Also $\boldsymbol{E}_m^{ij+} = (e_{m1}^{ij+}, \cdots, e_{mn}^{ij+})$ is the vector of bits from level m in the label used to choose $\boldsymbol{S}^{ij+}$ and $\boldsymbol{E}_m^{ij-} = (e_{m1}^{ij-}, \cdots, e_{mn}^{ij-})$ that used to choose $\boldsymbol{S}^{ij-}$. Since the received signal and extrinsic information from all levels are considered to be Gaussian random variables (4.10) can be written in the form

$$\Lambda_{ij}(q_i+1) = \frac{|\boldsymbol{R}-\boldsymbol{S}^{ij-}|^2}{2\sigma^2_{\boldsymbol{R}|\boldsymbol{S}^{ij-}}} - \frac{|\boldsymbol{R}-\boldsymbol{S}^{ij+}|^2}{2\sigma^2_{\boldsymbol{R}|\boldsymbol{S}^{ij+}}} \tag{4.11}$$
$$+\sum_{m=1}^{L}\left(\frac{|\boldsymbol{W}_m(q_m)-\mu_{\boldsymbol{W}_m(q_m)|\boldsymbol{E}_m^{ij-}}|^2}{2\sigma^2_{\boldsymbol{W}_m(q_m)|\boldsymbol{E}_m^{ij-}}} - \frac{|\boldsymbol{W}_m(q_m)-\mu_{\boldsymbol{W}_m(q_m)|\boldsymbol{E}_m^{ij+}}|^2}{2\sigma^2_{\boldsymbol{W}_m(q_m)|\boldsymbol{E}_m^{ij+}}}\right)$$

where $\sigma^2_{\boldsymbol{R}|\boldsymbol{S}^{ij\pm}}$ is the variance of $\boldsymbol{R}$ given $\boldsymbol{S}^{ij\pm}$ was transmitted, and $\mu_{\boldsymbol{W}_m(q_m)|\boldsymbol{E}_m^{ij\pm}}$ is the mean and $\sigma^2_{\boldsymbol{W}_m(q_m)|\boldsymbol{E}_m^{ij\pm}}$ is the variance of $\boldsymbol{W}_m(q_m)$ given $\boldsymbol{E}_m^{ij\pm}$ was transmitted. In practise the following is used

$$\begin{aligned}\Lambda_{ij}(q_i+1) &= \frac{|\boldsymbol{R}-\boldsymbol{S}^{ij-}|^2 - |\boldsymbol{R}-\boldsymbol{S}^{ij+}|^2}{2\sigma^2_{\boldsymbol{R}}} \\ &+ \sum_{m=1}^{L}\frac{|\boldsymbol{W}_m(q_m)-\mu_{\boldsymbol{W}_m(q_m)}\boldsymbol{E}_m^{ij-}|^2 - |\boldsymbol{W}_m(q_m)-\mu_{\boldsymbol{W}_m(q_m)}\boldsymbol{E}_m^{ij+}|^2}{2\sigma^2_{\boldsymbol{W}_m(q_m)}}\end{aligned} \tag{4.12}$$

where $\sigma^2_{\boldsymbol{R}}$ is the variance of the AWGN and is (assumed) known or is estimated once for each received multilevel code block. $\mu_{\boldsymbol{W}_m(q_m)}$ is the mean and $\sigma^2_{\boldsymbol{W}_m(q_m)}$ is the variance of the absolute value of $\boldsymbol{W}_m(q_m)$ and these are estimated using (4.7) and (4.6) respectively. Note that they are estimated over a component code block after each decoding.

The soft output simplifies to

$$\Lambda_{ij}(q_i+1) = \sum_{l=1}^{n}\left(\frac{|r_l-s_l^{ij-}|^2 - |r_l-s_l^{ij+}|^2}{2\sigma^2_{\boldsymbol{R}}} + \sum_{m=1}^{L}\frac{\mu_{\boldsymbol{W}_m(q_m)}}{\sigma^2_{\boldsymbol{W}_m(q_m)}}w_{ml}(q_m)(e_{ml}^{ij+}-e_{ml}^{ij-})\right) \tag{4.13}$$

If level i is not encoded with a concatenated code then extrinsic information from the previous decoding of level i may not be used in the current decoding of level i and so must be excluded from (4.5) and (4.13). If the level-m partition is not a binary (2 way) partition[8], then the extrinsic information for the extra bits from level m in the constellation point label is included in (4.5) and (4.13).

From (4.5) and (4.13) the soft output from the $(q_i+1)^{th}$ decoder in the level-i

[8]Multilevel codes allow multi-way partitions of different sizes at each level [16, 82].

pipeline can be written as

$$\Lambda_{ij}(q_i+1) = \sum_{l=1, e_{il}^{ij+} \neq e_{il}^{ij-}}^{n} e_{il}^{ij+} \lambda_{il}(q_i+1) \tag{4.14}$$

The extrinsic information gained by the $(q_i+1)^{th}$ decoder in the level-i pipeline for the j^{th} bit is defined as

$$w_{ij}(q_i+1) = \Lambda_{ij}(q_i+1) - \lambda_{ij}(q_i+1) = \sum_{l=1, l \neq j, e_{il}^{ij+} \neq e_{il}^{ij-}}^{n} e_{il}^{ij+} \lambda_{il}(q_i+1) \tag{4.15}$$

Two ways of calculating the extrinsic information passed to other levels are now considered. The first option is to use the extrinsic information from the final component decoder on level i. This has been assumed in the derivation above for ease of notation. The second option is to calculate the overall extrinsic information produced by the level-i decoder. This is equal to the soft output from the final decoder on level i minus the soft input to the first decoder on level i. Options 1 and 2 are the same when a single code is used on level i.

In a practical system, waiting for all levels to decode before decoding level 1 again can result in an unacceptable delay. This problem can be reduced by decoding the codes on several levels at the same time. The first level has the most complicated code when using Ungerboeck set partitioning [115] and so should take the longest to decode, so several other levels may be able to be decoded in the time it takes to decode level 1. The decoder on each level can use the most recent extrinsic information available on other levels by calculating the soft input of (4.5) for each component codeword just before being decoded. This may mean that later component decoders on a level may use different extrinsic information from other levels than earlier decoders. This should result in good performance with fewer complete decodings of the multilevel code being required and hence less delay. This is due to more reliable extrinsic information being used earlier by some of the component decoders.

4.3 Adaptive Component Decoder

In this section adaptive sub-optimal SISO decoders for the component codes in the multilevel code will be discussed. In chapter 3 two adaptive decoding algorithms

were developed based on the algorithm of [79, 90]. They do not require optimization using repeated simulations, can adapt to different codes and conditions, and treat good and bad blocks differently. This section extends the notation of the adaptive algorithms developed in chapter 3 for use with different constellations and with the iterative MSD of section 4.2. These decoding algorithms use a *soft-input list-output (SILO)* decoder in the center. There are a large number of decoding algorithms which could be used in the SILO decoders [19, 26, 30, 33, 34, 35, 36, 40, 46, 55, 56, 99, 105, 106, 109, 110]. In simulations the Chase decoder [19] as described in section 2.2 will be used. The structure of one stage of the adaptive SISO component decoder on level i is shown in Fig. 4.3. The soft input to the adaptive sub-optimal soft output decoder is given by (4.5). The sub-optimal decoder approximates the soft output of (4.13) by considering only a subset of the possible sequences of constellation points when choosing $\boldsymbol{S}^{ij+}$ and $\boldsymbol{S}^{ij-}$ in (4.10). This subset is chosen from the list of possible level-i codewords from the SILO decoder in the level i SISO component decoder, namely $\{\boldsymbol{C}\}$ and $\boldsymbol{D}$, as shown in Fig. 4.3. For complexity reasons other levels are not constrained to being codewords when choosing $\boldsymbol{S}^{ij+}$ and $\boldsymbol{S}^{ij-}$.

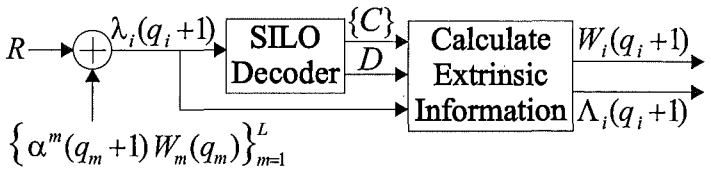


Figure 4.3: One decoding stage in the level-i pipeline, where $\{\boldsymbol{\alpha}^m(q_m + 1)\boldsymbol{W}_m(q_m)\}_{m=1}^L$ is the set of scaled extrinsic information from the last completed decoding on each of the L levels of the multilevel code. Extrinsic information from level-i is only included in the set if $V_i > 1$.

Denote $\boldsymbol{S}^{\boldsymbol{D}}$ as the sequence of constellation points from the larger of the two maximum functions in (4.10) for level i and $\boldsymbol{S}^{\boldsymbol{C}}$ as the other sequence of constellation points produced by the maximums (meaning $\boldsymbol{S}^{\boldsymbol{D}} = \boldsymbol{S}^{ij+}$ or $\boldsymbol{S}^{ij-}$ and $\boldsymbol{S}^{\boldsymbol{C}} = \boldsymbol{S}^{ij-}$ or $\boldsymbol{S}^{ij+}$) provided both $\boldsymbol{S}^{ij+}$ and $\boldsymbol{S}^{ij-}$ are available. The level-i codeword labelling $\boldsymbol{S}^{\boldsymbol{D}}$ is denoted $\boldsymbol{D} = (d_1, \cdots, d_n)$ and the level-i codeword labelling $\boldsymbol{S}^{\boldsymbol{C}}$ is denoted $\boldsymbol{C} = (c_1, \cdots, c_n)$. By definition $c_j \neq d_j$. Then (4.15) can be written as

$$w_{ij}(q_i + 1) = d_j \sum_{l=1, l\neq j, d_l \neq c_l}^{n} d_l \lambda_{il}(q_i + 1) \tag{4.16}$$

This can be written as [79]

$$w_{ij}(q_i+1) = d_j \beta_j^i(q_i+1) \tag{4.17}$$

Using (4.16) and (4.17) $\beta_j^i(q_i+1)$ is given by

$$\beta_j^i(q_i+1) = \sum_{l=1, l\neq j, d_l \neq c_l}^{n} d_l \lambda_{il}(q_i+1) \tag{4.18}$$

Since only a subset of all possible codewords are produced by the Chase algorithm, it may not always be able to find $\boldsymbol{D}$ and competing codewords, $\boldsymbol{C}$, for every position in the codeword. In this case $\beta_j^i(q_i+1)$ cannot be calculated using (4.18) and an estimate is required.

The notation of the approximation based approach to adaptively estimating β developed in chapter 3 is now extended for use in multilevel codes. The value of β for the j^{th} position from the $(q_i+1)^{th}$ decoder in the level-i pipeline can be written as

$$\beta_j^i(q_i+1) = \begin{cases} \sum_{l=1, l\neq j, d_l \neq c_l}^{n} d_l \lambda_{il}(q_i+1), & \boldsymbol{C} \neq \boldsymbol{D} \text{ found with } c_j \neq d_j. \\ \min_l(\beta_l^i(q_i+1) > 0), & \boldsymbol{C} \neq \boldsymbol{D} \text{ found, but } c_j = d_j. \\ \text{mean}_l(\beta_l^i(q_i+1)), & \text{No } \boldsymbol{C} \neq \boldsymbol{D} \text{ found.} \\ \text{mean}_l(\beta_l^i(q_i+1)) \text{ or } 0, & \text{No } \boldsymbol{C} \text{ or } \boldsymbol{D} \text{ found.} \end{cases} \tag{4.19}$$

The minimum is over all positive values of $\beta_l^i(q_i+1)$ calculated using (4.18) for the current received component codeword. The mean is taken over all values of $\beta_l^i(q_i+1)$ calculated using (4.18) in the current block of transmitted codewords. If no $\beta_l^i(q_i+1)$ are able to be calculated using (4.18) for any received codeword in the current block of transmitted codewords, then the mean is taken over all transmitted blocks decoded to date. If this cannot be calculated then the mean of the absolute value of the soft input to the current decoder can be used, instead of $\text{mean}_l(\beta_l^i(q_i+1))$. Using (4.19) the extrinsic information can now be calculated for all positions using (4.17). If an extended block code is being decoded the overall parity bit is treated as if there is no competing codeword.

The notation for the distance based approach to adaptively estimating $\beta_j^i(q_i+1)$ is now extended for a multilevel code. The level-i set of combinations of positions to sum, $\{\xi_i\}$, is restricted to including only combinations of the N_s *least reliable*

positions (LRPs), where N_s is calculated for each component code in the multilevel code. The new estimate can be written as[9]

$$\hat{\beta}_{ij}(q_i+1) = \min_{\{\xi_i\}} \left\{ b = \left(\sum_{l=1, l\in\xi_i}^{n} d_l \lambda_{il}(q_i+1) \right) : b > 0 \right\} \tag{4.20}$$

The set must be chosen carefully as it can affect decoding performance. If no $\boldsymbol{D}$ is found, then $\boldsymbol{D}$ is set equal to the hard decision on the input to the component decoder, $\boldsymbol{Y}(q_i+1)$. In this case $\hat{\beta}_{ij}(q_i+1)$ is used in all positions. Alternatively, when no $\boldsymbol{D}$ is found $\hat{\beta}_{ij}(q_i+1)$ could be set equal to zero or the case 3 $\beta_{ij}(q_i+1)$ estimate of the approximation approach in (4.19).

The notation for the adaptively estimated $\boldsymbol{\alpha}$ of chapter 3 is now extended for use in multilevel codes. In [79] the soft input to level 1 was written in the form

$$\lambda_{1j}(q_1+1) = r_j + \alpha(q_1+1) w_{1j}(q_1) \tag{4.21}$$

Comparing this to (4.5) and taking into account the use of an iterative MSD, $\alpha_j^i(q_i+1)$ can be defined as

$$\alpha_j^i(q_i+1) = \mu_{\boldsymbol{W}_i(q_i)} \frac{\sigma_{\boldsymbol{R}}^2}{\sigma_{\boldsymbol{W}_i(q_i)}^2} \tag{4.22}$$

and so the soft input to the $(q_i+1)^{th}$ decoder in the level-i pipeline becomes

$$\frac{\sigma_{\boldsymbol{R}}^2}{2}\lambda_{ij}(q_i+1) = \frac{|r_j - s_j^{i-}|^2 - |r_j - s_j^{i+}|^2}{4} + \sum_{m=1}^{L} \alpha_j^m(q_m+1) w_{mj}(q_m) \frac{(e_{mj}^{i+} - e_{mj}^{i-})}{2}. \tag{4.23}$$

The $\alpha_j^i(q_i+1)$ of (4.22) and that of [80] tend to infinity as the decoder tends to a decision, since $\sigma_{\boldsymbol{W}_i(q_i)}^2 \to 0$. The scaling developed in chapter 3 to avoid this problem is now extended for use in an iterative MSD. To avoid estimating $\alpha_j^i(q_i+1)$ when $\sigma_{\boldsymbol{W}_i(q_i)}^2 \approx 0$, the soft input is scaled by $\chi_i(q_i+1) = \sigma_{\boldsymbol{W}_i(q_i)}^2/(\mu_{\boldsymbol{W}_i(q_i)})^2$ if $\sigma_{\boldsymbol{W}_i(q_i)}^2/(2\mu_{\boldsymbol{W}_i(q_i)}) < 1$. The total scaling to the soft input on level i is

$$\chi_i^T = \prod_{m=1, (m\neq i \text{ if } V_i=1)}^{L} \chi_m(q_m+1) \tag{4.24}$$

The scaled soft input to level i is

$$\lambda'_{ij}(q_i+1) = \chi_i^T \lambda_{ij}(q_i+1) \tag{4.25}$$

[9]Any combination of positions, ξ_i, which produces a negative estimate is discarded.

The extrinsic information produced using $\lambda'_{ij}(q_i+1)$, denoted $w'_{ij}(q_i+1)$, is inherently scaled by χ_i^T. Since $\mu_{\boldsymbol{W}'_i(q_i)} = \chi_i^T \mu_{\boldsymbol{W}_i(q_i)}$ and $\sigma^2_{\boldsymbol{W}'_i(q_i)} = (\chi_i^T)^2 \sigma^2_{\boldsymbol{W}_i(q_i)}$, $\alpha_i(q_i+2)$ removes the scaling when $w'_{ij}(q_i+1)$ is used in the soft input to the next decoding stage of the iterative MSD, meaning $\alpha'_i(q_i+2)w'_{ij}(q_i+1) = \alpha_i(q_i+2)w_{ij}(q_i+1)$. Otherwise χ_i^T would have to include the scaling from all previous decoding stages. If the minimum absolute value of the extrinsic information from level m is larger than the maximum value of $\frac{|r_j - s_j^{i-}|^2 - |r_j - s_j^{i+}|^2}{2\sigma^2_{\boldsymbol{R}}}$ for all j, then the hard decision on level m is used by all other levels. In this case the extrinsic information from level m and the scaling for level m, $\chi_m(q_m+1)$, are removed from the soft input to other levels. The possible constellation points used to calculate the soft input are then chosen according to the level m decision.

4.4 Simulation Results

This section presents simulation results for multilevel codes with 2 or 3 levels using the iterative MSD of section 4.2, the adaptively estimated $\boldsymbol{\alpha}$ of section 4.3 and the two approaches to adaptively estimating $\boldsymbol{\beta}$ of section 4.3. All results presented plot the *bit error rate (BER)* of the data bits against the average energy used by the multilevel code to transmit a data bit[10], E_b, divided by the 2 sided baseband power spectral density of the noise, N_0. Iterations 1, 2, 3, 4, 6 and 10 are displayed and performance improves after each iteration unless otherwise stated. The noise variance, $\sigma^2_{\boldsymbol{R}}$, is (assumed) known in this section and all simulations transmit 16 QAM on the AWGN channel. The adaptive algorithms are used with the iterative MSD, and the precomputed approach is used with a traditional MSD (which passes hard decisions to subsequent levels) unless otherwise stated.

The simulation results presented in this chapter were obtained after finding at least 100 blocks (of 2048 16-QAM symbols each) in error on each level after the final iteration for a given E_b/N_0. Due to time constraints and computer system crashes some points were obtained after fewer blocks in error. Unless otherwise stated at least 80 blocks in error were observed.

From chapter 1, 16-QAM consists of two orthogonal 4-ASK signals. In [79]

[10]This will be used even when the performance of individual levels is shown.

each 4-ASK constellation point is chosen by two binary partitions (using a level-1 and a level-2 bit) according to Ungerboeck set partitioning as shown in Fig. 4.4. Unless otherwise stated the 2 level simulations use this partitioning. The mapping to consecutive 4-ASK constellations is independent and independent noise is assumed. Therefore, simulations of a 4-ASK constellation with a 2-way partition can be used to obtain results for two 4-way partitions of 16-QAM.

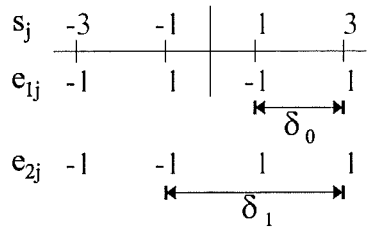


Figure 4.4: Mapping level 1 and 2 coded bits onto 4-ASK.

Unless otherwise stated the 2 level simulations transmit a 2-dimensional $(64, 57, 4)^2$ block Turbo code on level 1 composed of $(64, 57, 4)$ extended BCH component codes. The overall minimum squared Euclidean distance of the level 1 block Turbo code is $d^2_{E,1} = 16\delta_0^2$, where δ_0^2 is the minimum squared Euclidean distance between constellation points before partitioning. Ungerboeck set partitioning increases the squared Euclidean distance on level 2 so a weaker code can be used. Sixty four $(64, 57, 4)$ extended BCH codewords are transmitted on level 2 per transmitted block with a minimum squared Euclidean distance of $d^2_{E,2} = 16\delta_0^2$. This multilevel code has been designed for equal squared Euclidean distance on all levels (the balanced distance design described in section 2.4). Unless otherwise stated the $p = 5$ LRPs are considered, producing $2^p = 32$ error patterns, however only 16 of these are actually used (to reduce complexity). The error patterns retained are given in Table 3.1. In addition the adaptively estimated α is used with the approximation based approach to adaptively estimating β (the approximation approach), no interleaving is used between levels and the extrinsic information is passed according to option[11] 1 unless otherwise stated.

Simulation results are presented in Fig. 4.5 for a two level code using the

[11]Option 1 passes extrinsic information from the final component decoder on each level to other levels.

- non-adaptive sub-optimal SISO decoder of [79] (described in section 2.3) in a traditional MSD (hard decisions are passed to subsequent levels), called SIM1;
- approximation approach of section 4.3 in a traditional MSD, called SIM2;
- approximation approach of section 4.3 in the iterative MSD of section 4.2, called SIM3.

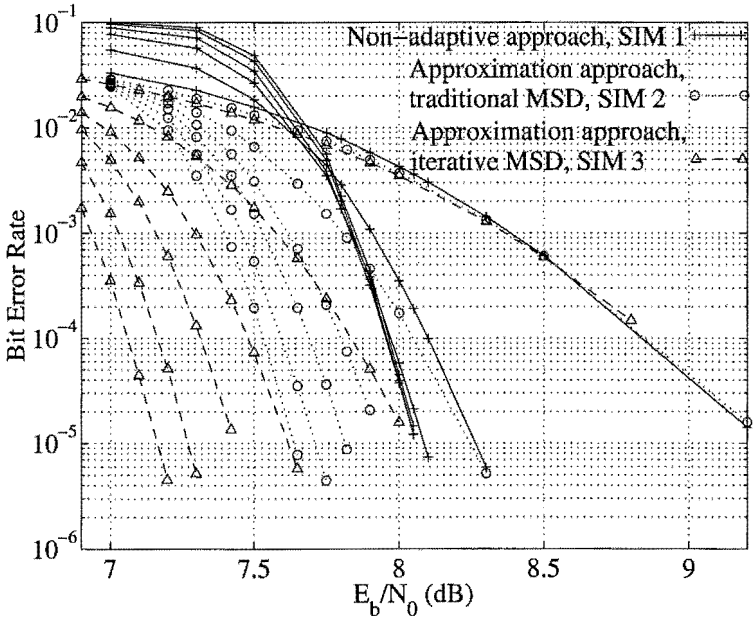


Figure 4.5: Performance of a two-level multilevel code. A $(64, 57, 4)^2$ block Turbo code is transmitted on level 1 and a $(64, 57, 4)$ extended BCH code on level 2.

In Fig. 4.5 the performance improves with each iteration except in SIM1, where iteration 4 performs the best at low BERs. At high BERs the performance of SIM1 gets worse after each iteration. The results for SIM1 are similar to those obtained in [79]. As can be seen the adaptive approach in a traditional MSD (SIM2) improves performance by ≈ 0.24dB at a BER of 10^{-5} after 4 iterations and ≈ 0.43dB at a BER of 10^{-5} after 10 iterations over SIM1. The only difference between SIM1 and SIM2 is the use of adaptively estimated sequences $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, which causes a small increase in complexity due to the variance and mean of the extrinsic information being calculated for each decoding stage.

When the approximation approach of section 4.3 is used in the iterative MSD of section 4.2 (SIM3) the performance improves over SIM1 by $\approx 0.62dB$ at a BER of

10^{-5} after 4 iterations and $\approx$ 0.9dB at a BER of 10^{-5} after 10 iterations. There is an increase in complexity as level 2 is decoded once every iteration instead of only once (after the final iteration of the level 1 decoder) and due to the increased complexity of the adaptive algorithm. Fig. 4.5 clearly shows that a significant improvement can be made by adaptively estimating α and β and by using the new iterative MSD. This verifies that the approximation based adaptive algorithm of chapter 3 can be used in different applications without modification, especially with the extended notation developed in this chapter. Level 2 performs significantly better than level 1 in the simulated range of *signal to noise ratios (SNRs)* as shown in Fig. 4.6.

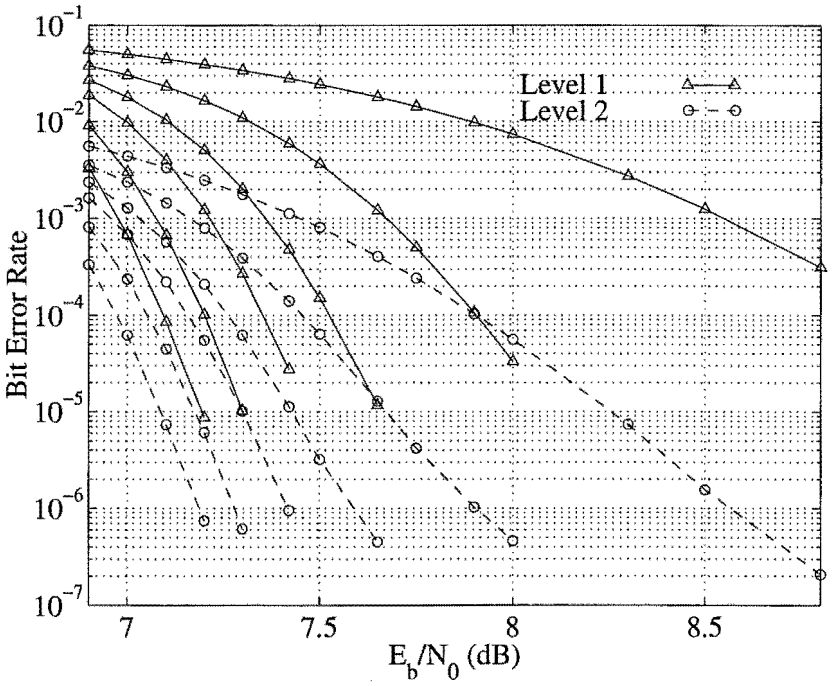


Figure 4.6: Performance of each level of a two-level multilevel code using the approximation approach. A $(64, 57, 4)^2$ block Turbo code is transmitted on level 1 and a $(64, 57, 4)$ extended BCH code on level 2.

All of these simulations passed extrinsic information between levels according to option 1. Now the performance of option[12] 2 is considered. The performance obtained using option 1 and 2 is compared in Fig. 4.7. It shows that option 1 performs better in later iterations.

Using a random interleaver after the encoder on level 2 was found to have a

[12]Option 2 passes the overall extrinsic information from a level (final decoders soft output minus first decoders soft input) to other levels.

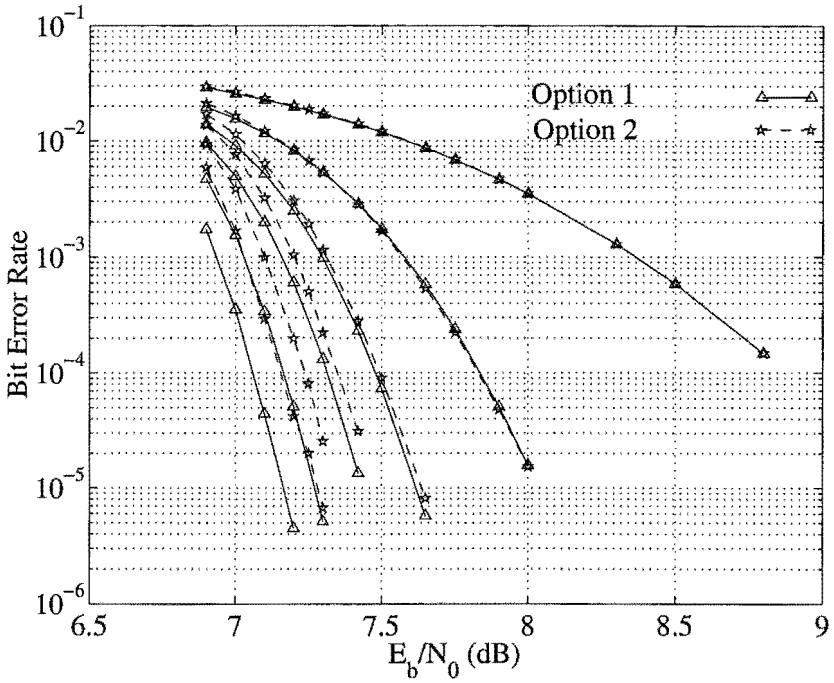


Figure 4.7: Performance of a two-level multilevel code using the approximation approach. A $(64, 57, 4)^2$ block Turbo code is transmitted on level 1 and a $(64, 57, 4)$ extended BCH code on level 2. The performance obtained by using option 1 and 2 is compared.

negligible effect on the overall performance when using option 1 as shown in Fig. 4.8. This is due to the effective interleaving between level 1 and 2 created by using column extrinsic information from level 1 to decode the rows on level 2. It slightly improved the level 2 performance which has a floor at low BERs as shown in Fig. 4.9. The random interleaver is only having a non-negligible effect at BERs below 10^{-5} on level 2. In this 2 level case extrinsic information from the column decoding of level 1 is passed to level 2, which decodes rows. This means that the level 2 code cannot be decoded until the entire level 1 block Turbo code has been decoded. Therefore, in this case the random interleaver does not add much decoding delay. Option 2 performed better with interleaving, but still performed worse than option 1 on later iterations. Option 1 will be used in all remaining simulations as it performs better with and without interleaving in later iterations.

A key difference between the iterative MSD developed in this chapter and previous approaches is that the extrinsic information is treated as a Gaussian random variable, rather than as a priori information. Effectively the a priori approach uses

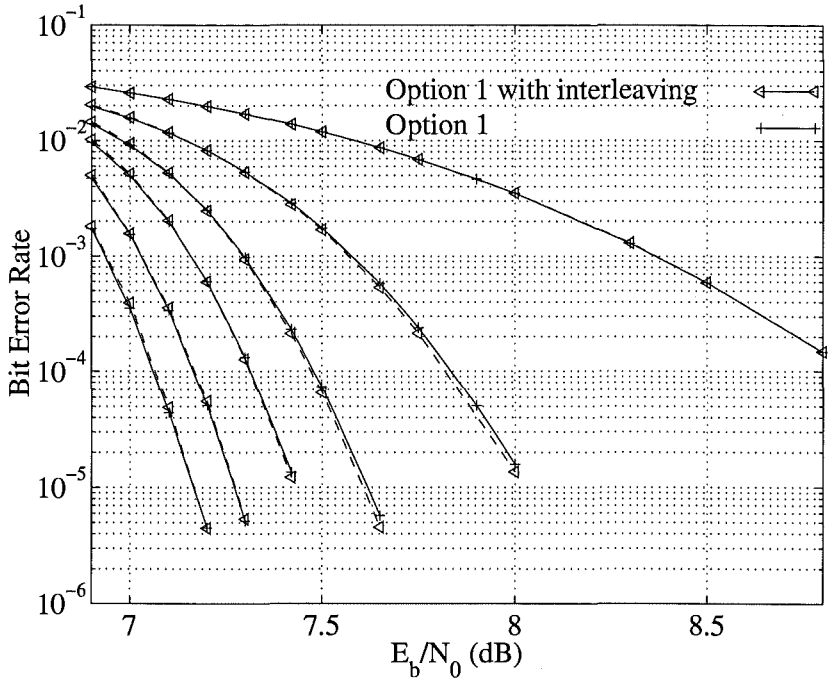


Figure 4.8: Performance of a two-level multilevel code using the approximation approach. A $(64, 57, 4)^2$ block Turbo code is transmitted on level 1 and a $(64, 57, 4)$ extended BCH code on level 2. The effect of using an interleaver on level 2 is shown.

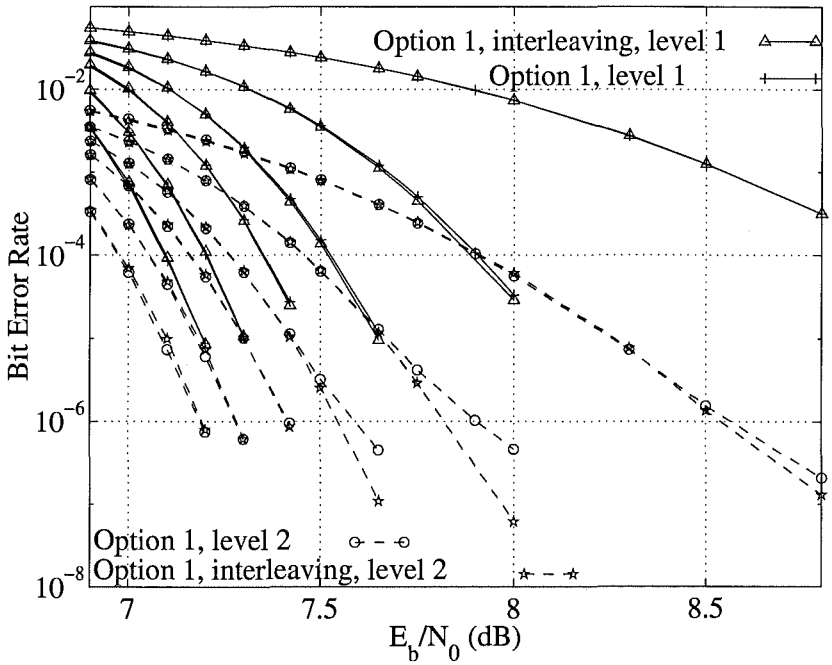


Figure 4.9: Performance on each level of a two-level multilevel code using option 1 and the approximation approach. A $(64, 57, 4)^2$ block Turbo code is transmitted on level 1 and a $(64, 57, 4)$ extended BCH code on level 2. The effect of using an interleaver on level 2 is shown.

$\alpha(q) = \sigma_R^2/2$ for all decoding stages. Fig. 4.10 shows that a significant improvement in performance is obtained by treating the extrinsic information as a Gaussian random variable (the Gaussian approach) rather than using it as a priori information (a priori approach). After four iterations of the Gaussian approach the performance is slightly better than that of the a priori approach after 10 iterations.

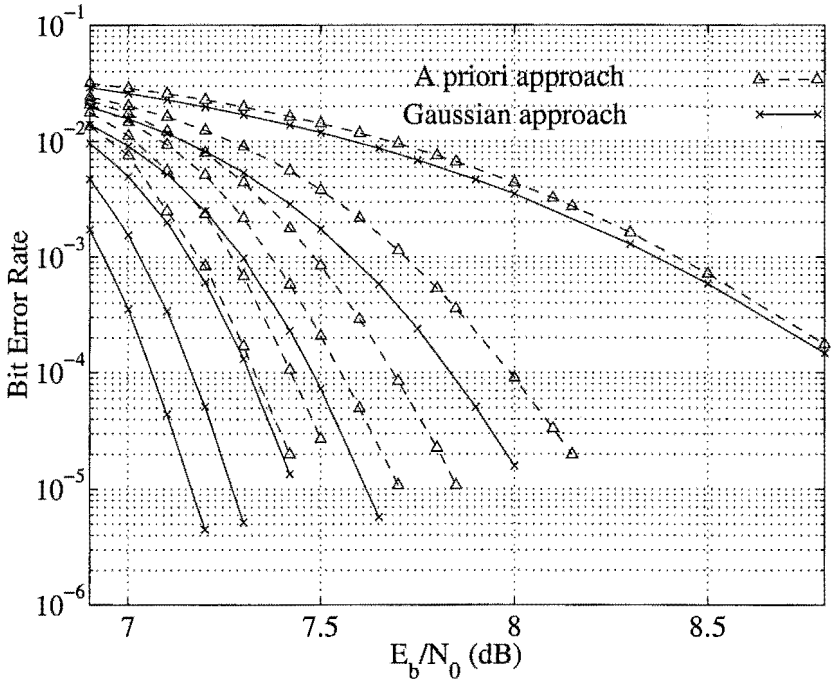


Figure 4.10: Performance of a two-level multilevel code using the approximation approach. The extrinsic information is treated as a priori information or as a Gaussian random variable. A $(64, 57, 4)^2$ block Turbo code is transmitted on level 1 and a $(64, 57, 4)$ extended BCH code on level 2.

Now different ways of passing hard and soft information between levels are compared. All use the approximation approach, no interleaving and option 1. The simulations are

- SIM2, hard information is passed from level 1 to 2 only;
- SIM4, hard information is passed from level 1 to 2 and level 2 to 1;
- SIM5, soft information is passed from level 1 to 2 only;
- SIM6, hard information is passed from level 1 to 2 and soft information is passed from level 2 to 1;

- SIM3, soft information is passed from level 1 to 2 and level 2 to 1.

The overall performance after 10 iterations is shown in Fig. 4.11. When hard information is used information is lost and a degradation in performance occurs. Error propagation problems are more severe when hard rather than soft decisions are passed, as reliability information about the decisions are discarded. As a result the worst performance is obtained when hard decisions are passed in both directions, SIM4. The performance of SIM4 could be improved by using a random interleaver on level 2. The best performance is obtained when passing soft decisions in both directions, SIM3. If hard decisions are passed from level 1 to 2 and soft decisions are passed from level 2 to 1, SIM6, the performance improves dramatically over only passing hard decisions from level 1 to 2, SIM2, at low SNRs. This is because the more reliable second level assists the level 1 decoder by passing it soft information to help choose the transmitted constellation point. The performance is not as good as passing soft decisions in both directions, due to error propagation and the information loss from passing hard decisions from level-1 to 2.

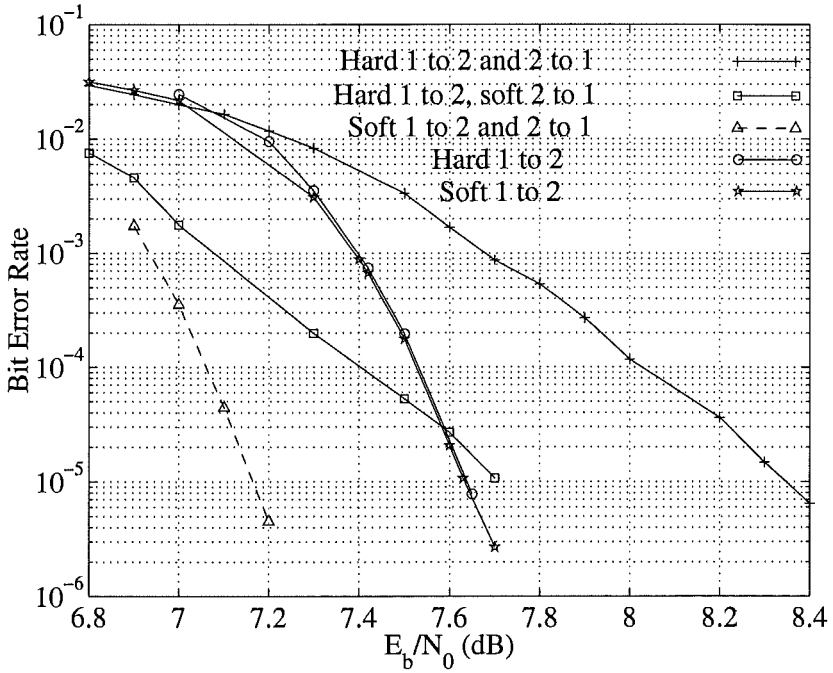


Figure 4.11: Performance of a two-level multilevel code after 10 iterations, using different methods of passing hard and soft information and the approximation approach. A $(64, 57, 4)^2$ block Turbo code is transmitted on level 1 and a $(64, 57, 4)$ extended BCH code on level 2.

In Fig. 4.11 the overall performance when hard decisions are passed from level 1 to 2, SIM2, is almost identical to the performance when soft information is passed from level 1 to 2, SIM5. This is because the level-1 code is dominating the overall performance and it is treated the same in both simulations. The performance of these simulations on level 1 and 2 after 1 and 2 iterations is shown in Fig. 4.12. Fig. 4.12 shows that even though passing hard and soft information from level 1 to 2 results in approximately the same overall performance, the performance on level 2 is better when soft decisions are passed from level 1 to 2. On both levels the best performance is obtained by passing soft information in both directions as shown in Fig. 4.12.

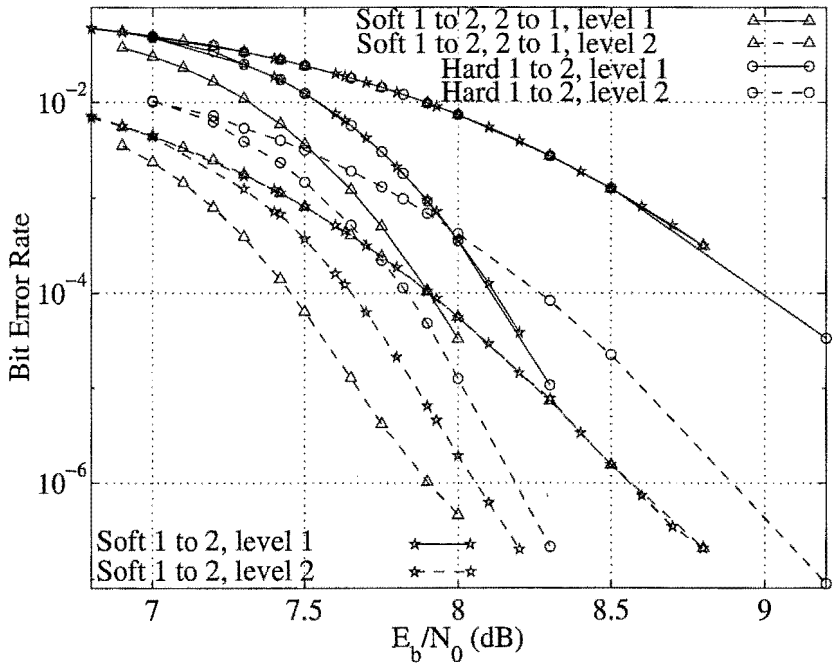


Figure 4.12: Performance on each level of a two-level multilevel code after one and two iterations, using different methods of passing hard and soft information and the approximation approach. A $(64, 57, 4)^2$ block Turbo code is transmitted on level 1 and a $(64, 57, 4)$ extended BCH code on level 2.

Now code design options will be considered. In chapter 3 decoding the more powerful code[13] in a block Turbo code first, resulted in better performance in early iterations, but approximately the same performance after 6 iterations. Consider a

[13]The code with the larger minimum Hamming distance and the code which performs the best when used as both the row and column code in a block Turbo code.

two level multilevel code with the $(64, 57, 4) \times (64, 51, 6)$ asymmetric block Turbo code on level 1 and the $(64, 57, 4)$ extended BCH code on level 1. The level 1 block Turbo code will be written as $(64, 57, 4) \times (64, 51, 6)$ when the $(64, 57, 4)$ extended BCH code is decoded first and as $(64, 51, 6) \times (64, 57, 4)$ when the $(64, 51, 6)$ extended BCH code is decoded first. Fig. 4.13[14] shows results when either the $(64, 57, 4) \times (64, 51, 6)$ or $(64, 51, 6) \times (64, 57, 4)$ block Turbo code is transmitted on level 1. Again option 1 is used with no interleaving and the approximation approach. As in chapter 3 the performance is better in early iterations if the $(64, 51, 6)$ extended BCH code is decoded first, but in Fig. 4.13 later iterations perform better if the $(64, 57, 4)$ extended BCH code is decoded first. The performance of decoding the $(64, 57, 4)$ code first after 6 iterations is the same as that for decoding the $(64, 51, 6)$ first after 10 iterations. Therefore, in this case it is better to decode the most powerful component code on a level last. This is probably because the extrinsic information from the second (more powerful) decoder is sent to the second level. In both cases the performance is better on level 2 than on level 1 over the simulated range of BERs, even though the level 1 code has a larger minimum squared Euclidean distance. The simulation with the $(64, 57, 4) \times (64, 51, 6)$ block Turbo code on level 1 performs better than that with the $(64, 51, 6) \times (64, 57, 4)$ block Turbo code on level 1 after all iterations on level 2, but only performs better on level 1 after 3 iterations.

The level 2 code outperforms the level 1 code significantly when the distance on level 1 and 2 is balanced as shown in Fig. 4.6 (SIM3). Therefore, the level 1 code could be made more powerful when using the same level 2 code[15]. Fig. 4.14 compares the performance on each level after 10 iterations when using the $(64, 57, 4)$ extended BCH code on level 2 ($d^2_{E,2} = 16\delta_0^2$) and one of the following block Turbo codes on level 1:

- $(64, 57, 4)^2$ block Turbo code with $d^2_{E,1} = 16\delta_0^2$ and resulting in an overall multilevel code rate of 0.84 (3.37 *data bits per 16-QAM symbol (BPS)*).

- $(64, 57, 4) \times (64, 51, 6)$ block Turbo code with $d^2_{E,1} = 24\delta_0^2$ and resulting in an overall multilevel code rate of 0.8 (3.2 BPS).

[14]Only 71 (out of 328574) blocks of 64 codewords were found in error on level 2 at 7.2 dB after 3 iterations of the $(64, 57)$ then $(64, 51)$ simulation.

[15]Alternatively the level 2 code could be made less powerful.

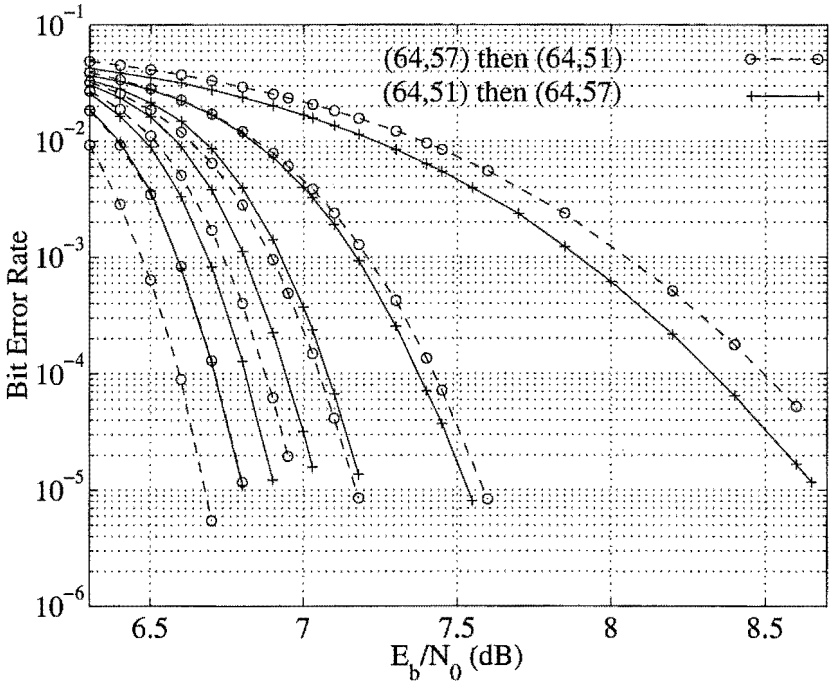


Figure 4.13: Performance of a two-level multilevel code using the approximation approach. A $(64, 57, 4) \times (64, 51, 6)$ block Turbo code is transmitted on level 1 and a $(64, 57, 4)$ extended BCH code on level 2. The importance of the order of decoding the level 1 code is investigated.

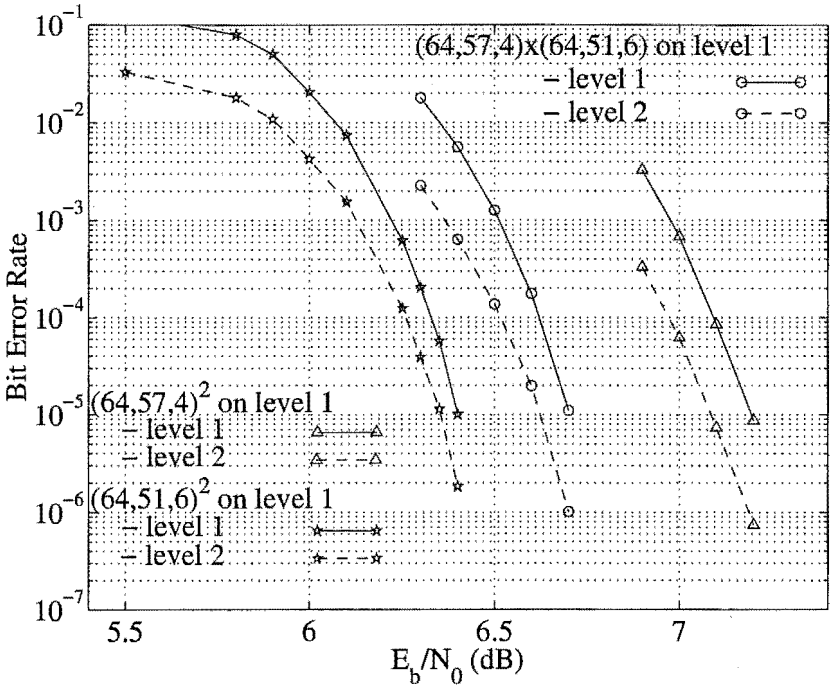


Figure 4.14: Performance on each level after 10 iterations of a two-level multilevel code using the approximation approach. The $(64, 57, 4)^2$, $(64, 51, 6)^2$ or $(64, 57, 4) \times (64, 51, 6)$ block Turbo code is transmitted on level 1 and a $(64, 57, 4)$ extended BCH code on level 2.

- $(64, 51, 6)^2$ block Turbo code with $d^2_{E,1} = 36\delta_0^2$ and resulting in an overall multilevel code rate of 0.76 (3.05 BPS).

Performance improves as the overall minimum distance on level 1 increases (and the rate decreases). Although the level 2 performance is better in all 3 cases the difference between the performance of the levels decreases as the level 1 component code's minimum Euclidean distance increases. The level 1 and 2 curves are tending together as the BER decreases when either the $(64, 51, 6)^2$ or $(64, 57, 4) \times (64, 51, 6)$ block Turbo code is transmitted on level 1.

In Fig. 4.14 it was seen that performance (BER) could be improved by increasing the distance of the level 1 code. However, this decreases the rate of the multilevel code. Now the performance of these codes will be compared to capacity. All simulations use the $(64, 57, 4)$ extended BCH code on level 2. The performance obtained when using different block Turbo codes on level 1 is now given:

- If the $(64, 57, 4)^2$ block Turbo code is used on level 1, then after 10 iterations at a BER of 10^{-5} the performance is $\approx 1.53dB$ from capacity (when transmitting 3.37 BPS) [51].

- If the $(64, 57, 4) \times (64, 51, 6)$ block Turbo code is used on level 1, then after 10 iterations at a BER of 10^{-5} the performance is $\approx 1.59dB$ from capacity (when transmitting 3.2 BPS) [51].

- If the $(64, 51, 6)^2$ block Turbo code is used on level 1, then after 10 iterations at a BER of 10^{-5} the performance is $\approx 1.71dB$ from capacity (when transmitting 3.05 BPS) [51].

Therefore, even though the BER performance improves as the distance of the level 1 code is increased the performance is actually getting further from capacity due to the code rate of the multilevel code. The performance could be improved by increasing the length of the component codes to $n = 128$, designing the code using capacity arguments [119], increasing the number of test sequences considered or possibly by using a different SILO decoder.

In [79] a pragmatic approach to transmitting block Turbo codes using higher order modulations was considered. The $(64, 57, 4)^2$ block Turbo code was transmitted using 16-QAM by mapping the encoded bits to the constellation points using

gray mapping. The pragmatic approach with the approximation algorithm is called SIM7. It transmits 3.17 BPS and has a block length of 4096 encoded bits. The 2 level code of SIM3 has rate 3.367 BPS and a block length of 8192 encoded bits. When SIM3 is compared to the pragmatic approach of SIM7, the pragmatic approach performs $\approx 0.65dB$ better at 10^{-5} after 10 iterations. However, the simulations transmit different numbers of BPS and so their performance should be compared to capacity. At a BER of 10^{-5} after 10 iterations SIM3 is $\approx 1.53dB$ from capacity, while SIM7 is $\approx 1.5dB$ from capacity as shown in Fig. 4.15 [51]. However, by extrapolating the curves to 10^{-6} the pragmatic approach (SIM7) is about $\approx 1.65dB$ from capacity and the two level approach (SIM3) is $\approx 1.63dB$ from capacity. Therefore, at low BERs the two level approach performs better. Performance closer to capacity could be obtained for both approaches by using more test sequences or possibly by using a different SILO decoder.

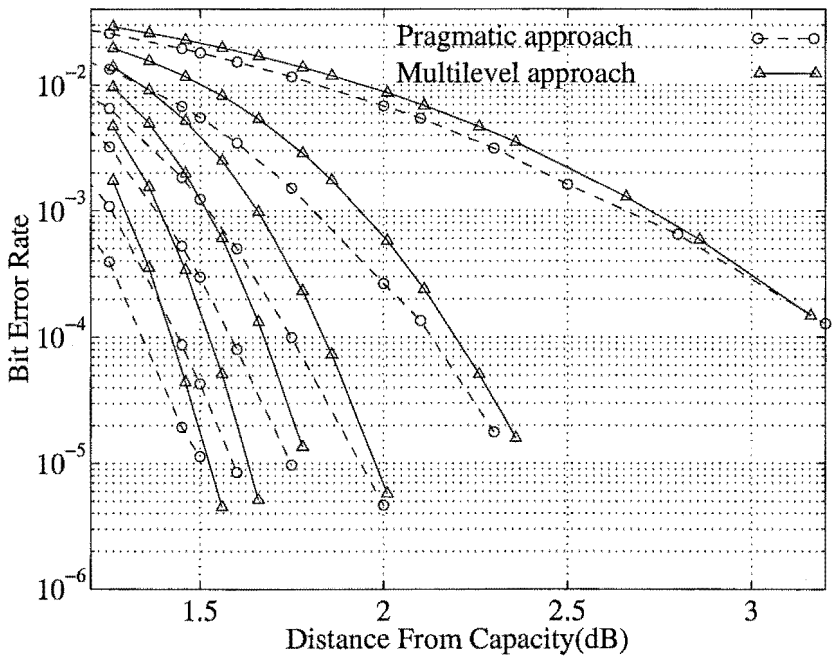


Figure 4.15: Performance of a two-level multilevel code or the pragmatic approach compared to capacity when the approximation algorithm is used. A $(64, 57, 4)^2$ block Turbo code is transmitted on level 1 and a $(64, 57, 4)$ extended BCH code on level 2 in the 2 level case. The pragmatic approach transmits the $(64, 57, 4)^2$ block Turbo code.

In chapter 3 it was found that scaling the minimum based $\beta_j(q)$ estimate (case 2 in (4.19)) by a constant did not improve performance very much if at all and the

constant needed to be re-optimised for different code designs. Fig. 4.16 and Fig. 4.17 show the effect of scaling the minimum based $\beta_j^i(q_i+1)$ by a constant after 10 and 4 iterations respectively. The $(64,57,4)^2$ block Turbo code is transmitted on level 1 and a $(64,57,4)$ BCH code on level 2. After 10 iterations the best performance is obtained by multiplying by 2 ($\times 2$ curve). It is approximately $0.1dB$ better than not scaling (the $\times 1$ curve) at 10^{-5} after 10 iterations. If this multiplier ($\times 2$) is reduced by a factor of 2 (the $\times 1$ curve) it performs better than if the multiplier is increased by a factor of 1.5 (the $\times 3$ curve). Once again this shows that it is better to underestimate than overestimate $\beta_j^i(q_i+1)$. This was also found to be the case when investigating other possible approaches to estimating $\beta_j^i(q_i+1)$.

In chapter 3 a distance based approach to adaptively estimating $\boldsymbol{\beta}$ was developed and two sets of combinations of positions to sum were suggested. This form of adaptively estimated $\boldsymbol{\beta}$, (4.20), is used with the adaptively estimated $\boldsymbol{\alpha}$ of section 4.3 (called the distance approach). The performance of the sets from chapter 3 when used in a two level code with the $(64,57,4)^2$ block Turbo code on level 1 and a $(64,57,4)$ extended BCH code on level 2 are shown in Fig. 4.18. Set 1 performs slightly better than set 2, but has a higher complexity. Unlike the chapter 3 simulations, these simulations have $p' < p$. The performance of the distance approach using set 1 is compared to the approximation approach in Fig. 4.19. Both approaches perform approximately the same. This figure also shows that the distance approach works for $p' < p$.

The iterative MSD developed in section 4.2 can be used for any number of levels. The results for a 3 level code using the approximation approach are shown in Fig. 4.20 for both an iterative MSD and a traditional MSD. Ungerboeck set partitioning of 16-QAM is used [115] as shown in Fig. 2.10. A 4-way partition of 16-QAM is used on level 1 and is encoded using the $(64,51,6)^2$ block Turbo code, giving a minimum squared Euclidean distance of $36\delta_0^2$ on level 1. Level 2 and 3 use 2-way partitions. Level 2 is encoded using the $(128,106,8)$ extended BCH code and there are 16 codewords per transmitted block. This gives a minimum distance of $32\delta_0^2$ on level 2. Level 3 is encoded using the $(64,57,4)$ extended BCH code with 32 codewords per transmitted block. This gives a minimum distance of $32\delta_0^2$ on level 3 and a total of 2048 16-QAM symbols per transmitted block. In the 3 level case

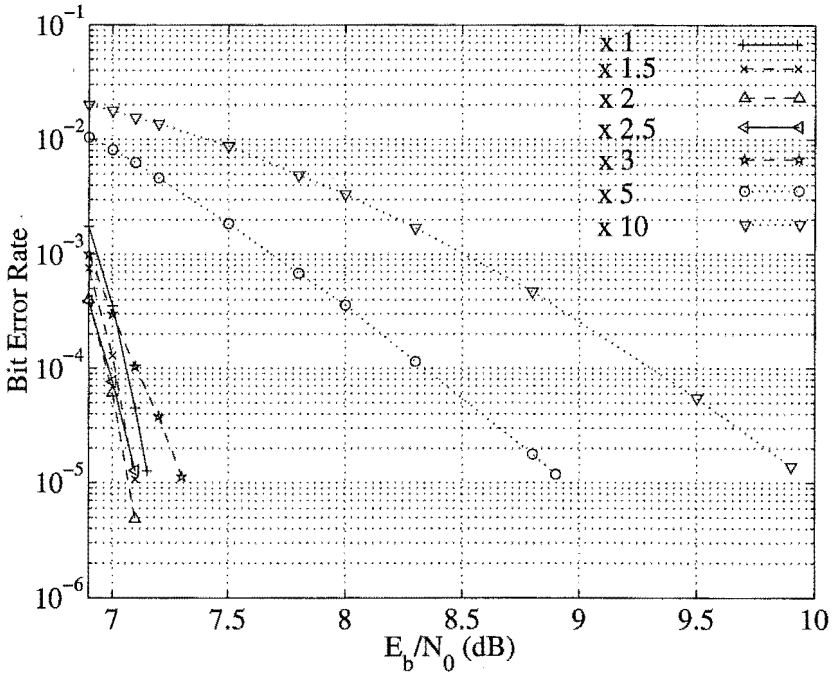


Figure 4.16: Performance after 10 iterations of a two-level multilevel code using the approximation approach, when the minimum based $\beta_j^i(q_i+1)$ is multiplied by a constant. A $(64, 57, 4)^2$ block Turbo code is transmitted on level 1 and a $(64, 57, 4)$ extended BCH code on level 2.

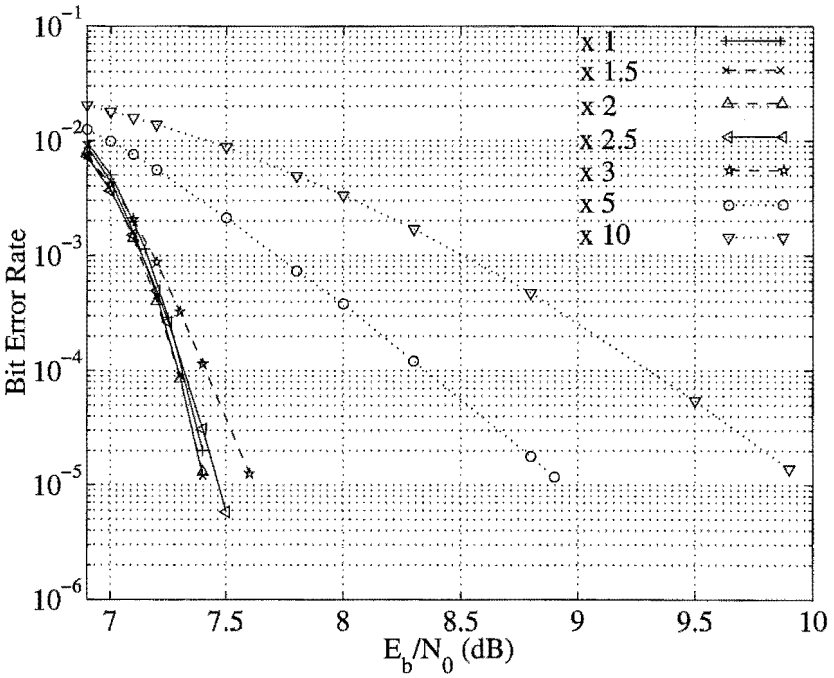


Figure 4.17: Performance after 4 iterations of a two-level multilevel code using the approximation approach, when the minimum based $\beta_j^i(q_i+1)$ is multiplied by a constant. A $(64, 57, 4)^2$ block Turbo code is transmitted on level 1 and a $(64, 57, 4)$ extended BCH code on level 2.

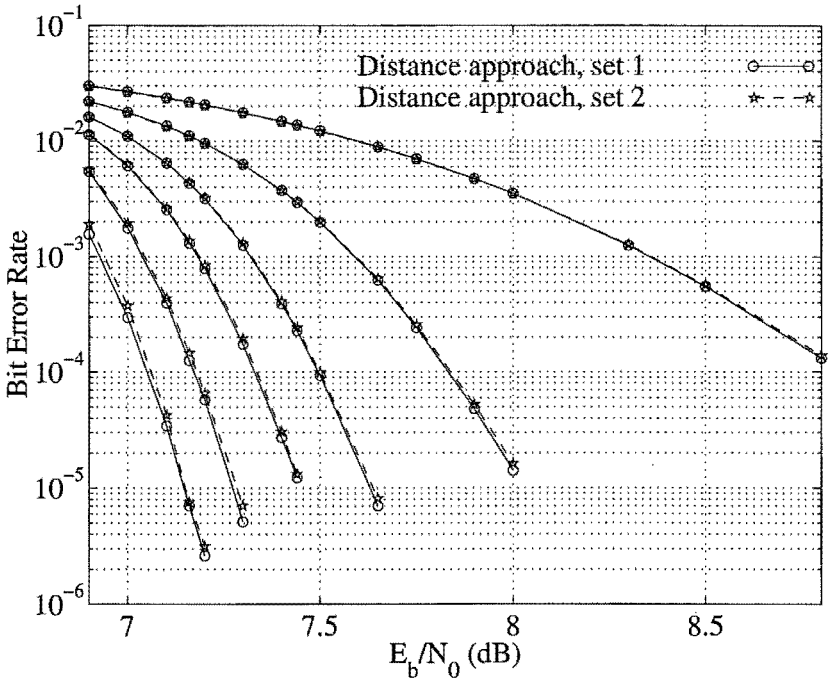


Figure 4.18: Performance of a two-level multilevel code using the distance approach and set 1 or 2. A $(64, 57, 4)^2$ block Turbo code is transmitted on level 1 and a $(64, 57, 4)$ BCH code on level 2.

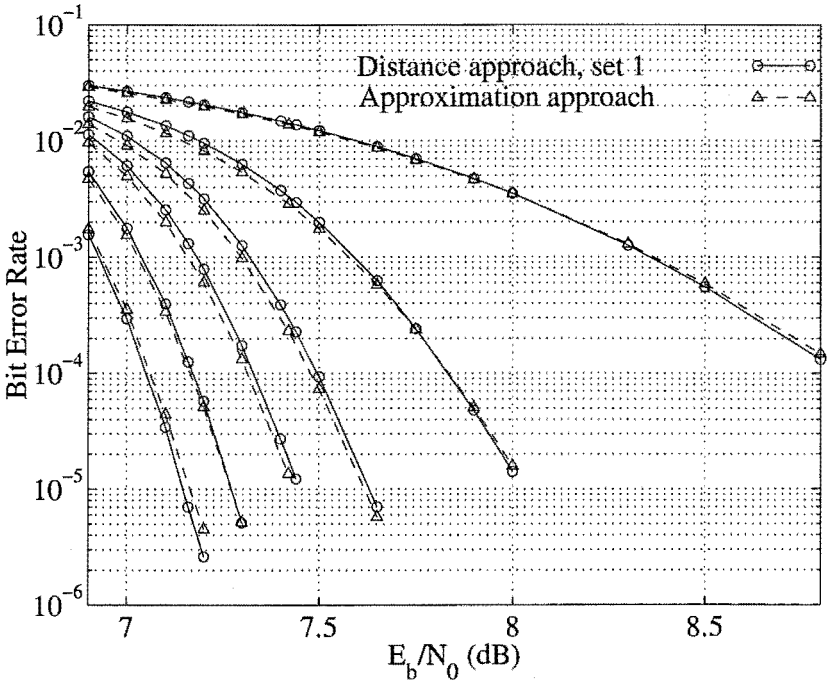


Figure 4.19: Performance of a two-level multilevel code using the distance approach and set 1 against that using the approximation approach. A $(64, 57, 4)^2$ block Turbo code is transmitted on level 1 and a $(64, 57, 4)$ BCH code on level 2.

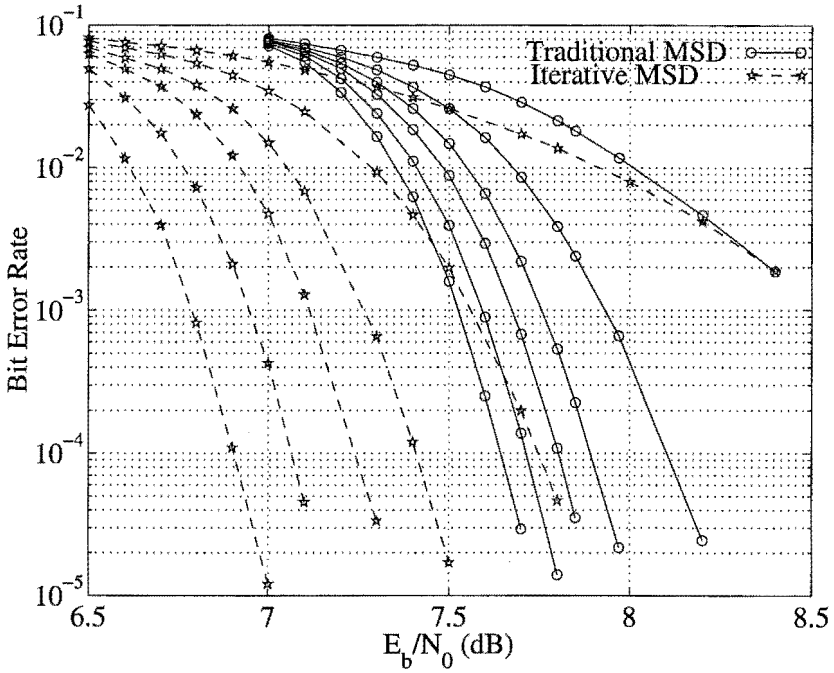


Figure 4.20: Performance of the overall three-level multilevel code using the approximation approach. The $(64, 51, 6)^2$ block Turbo code is transmitted on level 1, the $(128, 106, 8)$ extended BCH code on level 2 and the $(64, 57, 4)$ extended BCH code on level 3. The code is simulated when an iterative MSD and a traditional (hard) MSD are used.

the 16-QAM constellation must be considered rather than 4-ASK (as in the 2 level case), due to the 2-way partitions on level 2 and 3 (the in phase and quadrature 4-ASK constellations can be labelled by the same encoded bit). The level 2 code uses all 32 error patterns for $p = 5$, but the level 1 and 3 codes use only 16 of them as given in Table 3.1.

When the 3 level code (using the approximation approach) is decoded using an iterative MSD it performs 0.75dB better at a BER of 10^{-5} than if a traditional MSD is used after 10 iterations. The performance of each level of the 3 level code is shown in Fig. 4.21 after 1, 2, 4 and 10 iterations. As in the 2 level case (SIM3 and Fig. 4.6) the later levels perform better. This makes these multilevel codes useful for unequal error protection applications.

Most previous iterative MSDs used interleavers on later levels [27, 73, 74, 121]. In Fig. 4.8 little improvement in the overall performance of a two-level code was obtained when interleaving was used on level 2. In Fig. 4.22 the performance of the

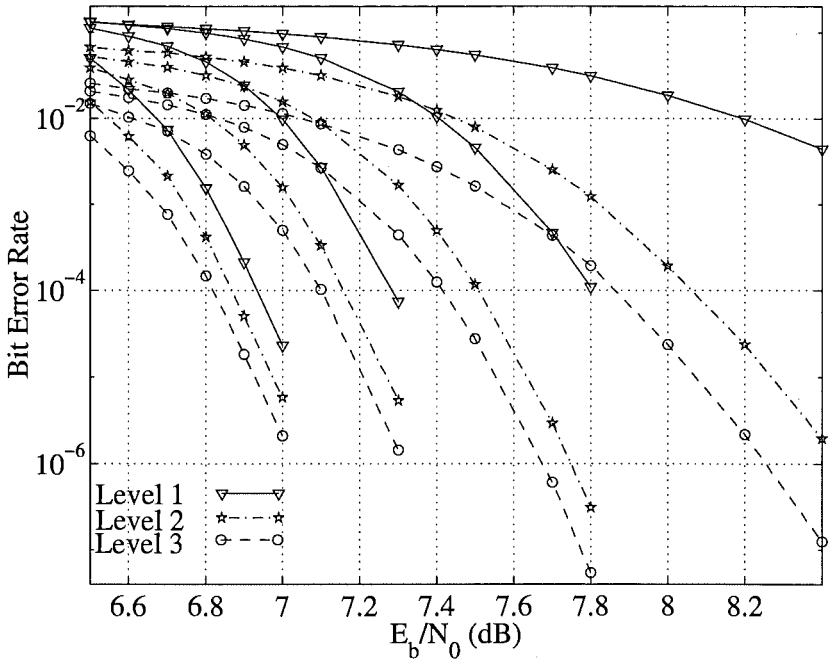


Figure 4.21: Performance on each level of a three-level multilevel code using the approximation approach and an iterative multistage decoder. The $(64, 51, 6)^2$ block Turbo code is transmitted on level 1, the $(128, 106, 8)$ extended BCH code on level 2 and the $(64, 57, 4)$ extended BCH code on level 3. Iteration 1, 2, 4 and 10 are displayed.

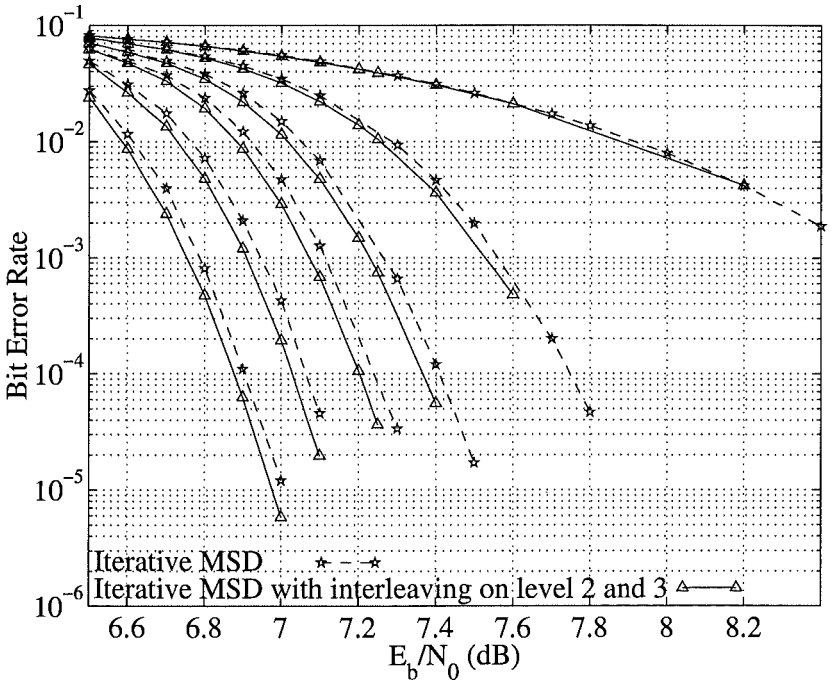


Figure 4.22: Performance of the overall three-level multilevel code using the approximation approach and an iterative multistage decoder. The $(64, 51, 6)^2$ block Turbo code is transmitted on level 1, the $(128, 106, 8)$ extended BCH code on level 2 and the $(64, 57, 4)$ extended BCH code on level 3. Simulation results with and without random interleaving on level 2 and 3 are shown.

3 level code using an iterative MSD when no interleaving is used and when random interleavers are used on both level 2 and 3 (after their encoders) is shown. A small improvement in the overall performance is obtained by using interleaving on level 2 and 3. The level 3 code performs significantly better when interleaving is used, especially in early iterations as shown in Fig. 4.23. The level 1 and 2 codes perform better after the first iteration as shown in Fig. 4.24 and Fig. 4.25 respectively, this may be due to the level 3 code. When interleaving is used the 3 level code is $\approx 2.48dB$ from capacity at 10^{-5} after 10 iterations [51]. Performance closer to capacity may be obtained by replacing the level 2 code. In chapter 3 another 3 error correcting code was not found to perform very well for its complexity. Performance closer to capacity could also be possible if the component codes were chosen using capacity arguments rather than distance arguments [119], more error patterns were used or another SILO decoder was used.

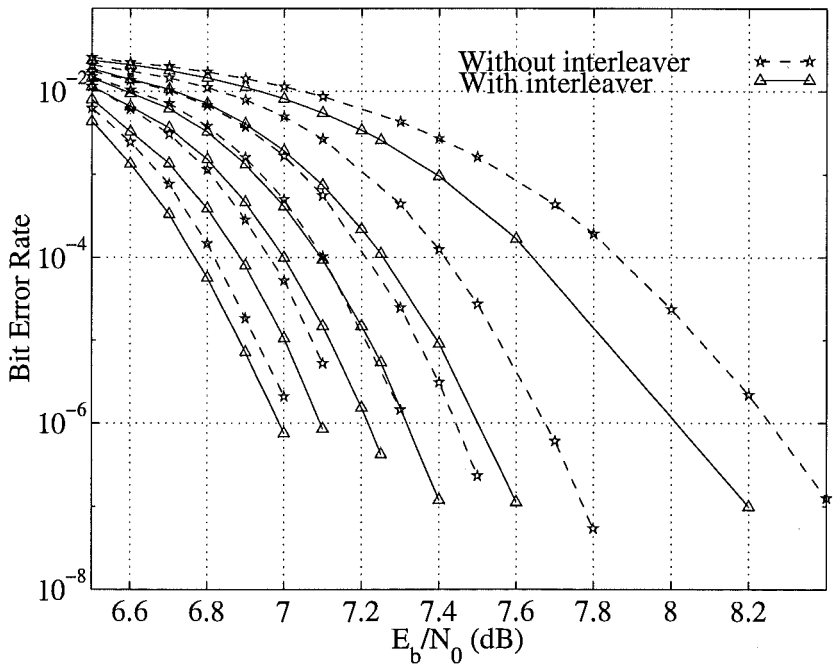


Figure 4.23: Performance on level 3 of a three-level multilevel code using the approximation approach and an iterative multistage decoder. The $(64, 51, 6)^2$ block Turbo code is transmitted on level 1, the $(128, 106, 8)$ extended BCH code on level 2 and the $(64, 57, 4)$ extended BCH code on level 3. Simulation results with and without random interleaving on level 2 and 3 are shown.

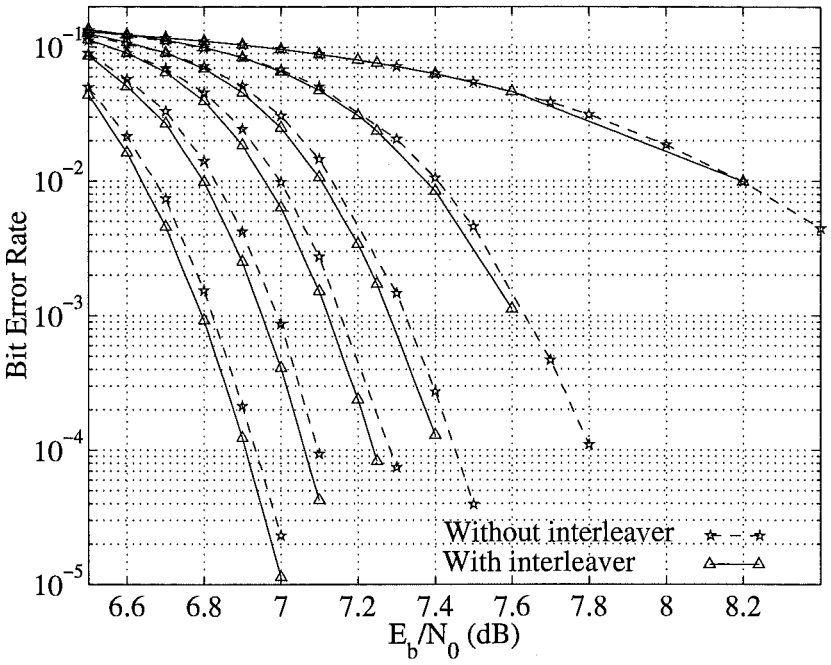


Figure 4.24: Performance on level 1 of a three-level multilevel code using the approximation approach and an iterative multistage decoder. The $(64, 51, 6)^2$ block Turbo code is transmitted on level 1, the $(128, 106, 8)$ extended BCH code on level 2 and the $(64, 57, 4)$ extended BCH code on level 3. Simulation results with and without random interleaving on level 2 and 3 are shown.

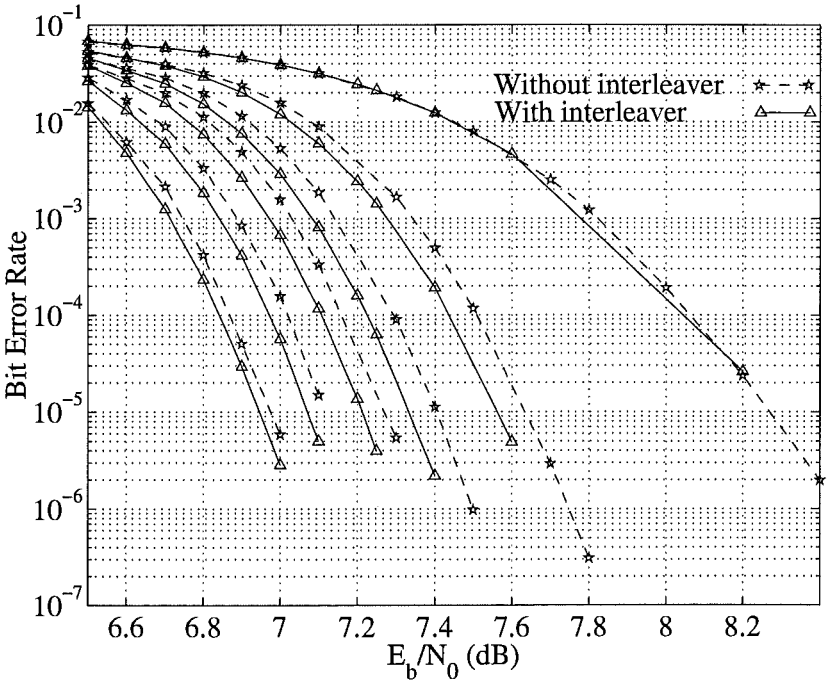


Figure 4.25: Performance on level 2 of a three-level multilevel code using the approximation approach and an iterative multistage decoder. The $(64, 51, 6)^2$ block Turbo code is transmitted on level 1, the $(128, 106, 8)$ extended BCH code on level 2 and the $(64, 57, 4)$ extended BCH code on level 3. Simulation results with and without random interleaving on level 2 and 3 are shown.

4.5 Performance Bound

This section considers a bound on the probability of bit errors for an L-level multilevel code, $\mathbb{C}$, with component codes, $(\mathbb{C}_1, \cdots, \mathbb{C}_L)$, after ML decoding on the AWGN channel. The probability of bit error of an L-level multilevel code is often bounded by[16] [48, 82]

$$P_e(\mathbb{C}) \leq \max\{P_e(\mathbb{C}_1), \cdots, P_e(\mathbb{C}_L)\} \tag{4.26}$$

In the code designs and simulations of section 4.4 the level one block Turbo code, $\mathbb{C}_1$, dominates performance. Therefore, $P_e(\mathbb{C}) \leq P_e(\mathbb{C}_1)$. In appendix C the bound from [114] for block Turbo codes transmitted using BPSK on the AWGN channel is given for *maximum likelihood (ML)* decoding. This bound can be used to bound the level one performance [114] as

$$P_e(\mathbb{C}) \leq P_e(\mathbb{C}_1) \leq \sum_{w=d_{H,1}}^{n_1} \frac{w}{n_1} A_w(\mathbb{C}_1) Q\left(\sqrt{\frac{d^2_{E,w}(\boldsymbol{C}, \boldsymbol{D})}{2N_0}}\right) \tag{4.27}$$

where $d_{H,1}$ is the minimum Hamming distance between codewords in $\mathbb{C}_1$, n_1 is the length of a level one codeword (a block Turbo codeword), $A_w(\mathbb{C}_1)$ is the number of codewords of weight w in $\mathbb{C}_1$ and $Q(\cdot)$ is the Q-function [47]. Consider two codewords, $\boldsymbol{C}$ and $\boldsymbol{D}$, at Hamming distance $d_H(\boldsymbol{C}, \boldsymbol{D}) = w$ from each other. The squared Euclidean distance between constellation point sequences labelled by these codewords is denoted $d^2_{E,w}(\boldsymbol{C}, \boldsymbol{D})$. When binary partitions are used (as in the two-level simulations using 4-ASK in section 4.4) it is easy to see that $d^2_{E,w}(\boldsymbol{C}, \boldsymbol{D}) = w\delta_0^2$, where δ_0^2 is the minimum squared Euclidean distance between constellation points (see Fig. 4.4).

The bound of (4.27) can be expressed in terms of the SNR, E_b/N_0, where E_b is the average energy used by the multilevel code to transmit one data bit. The average energy used to transmit a constellation point is $E_s = \mathcal{R}\log_2(M)E_b$, where $\mathcal{R}$ is the overall multilevel code rate and $\log_2(M)$ is the number of encoded bits used to label each of the M constellation points [48]. Therefore, the bound of (4.27) becomes

$$P_e(\mathbb{C}) \leq P_e(\mathbb{C}_1) \leq \sum_{w=d_{H,1}}^{n_1} \frac{w}{n_1} A_w(\mathbb{C}_1) Q\left(\sqrt{\frac{w\delta_0^2}{2N_0}\frac{\mathcal{R}\log_2(M)E_b}{E_s}}\right) \tag{4.28}$$

[16] A number of other bounds for multilevel codes are presented in [119].

This is approximated by considering only the minimum Hamming distance codewords.

In this thesis sub-optimal SISO decoders are used and so the performance will only tend to ML as the number of iterations and number of error patterns considered increases. By passing soft decisions to subsequent levels the error propagation problem is reduced. Also performance gets closer to the performance of ML decoding the overall multilevel code by using an iterative MSD. Eventually, if the soft information is very reliable it acts like a hard decision. In this case when it is passed back to previous levels the minimum squared Euclidean distance on previous levels has effectively been increased (assuming Ungerboeck set partitioning).

Consider transmitting a multilevel code, $\mathbb{C}$, with the $(64, 57, 4)^2$ block Turbo code on level 1, $\mathbb{C}_1$, and the $(64, 57, 4)$ extended BCH code on level 2 (using 64 codewords per block), $\mathbb{C}_2$, using 4-ASK (with 2 binary partitions). The overall multilevel code bound of (4.26) and (4.28) for this multilevel code is shown in Fig. 4.26. The

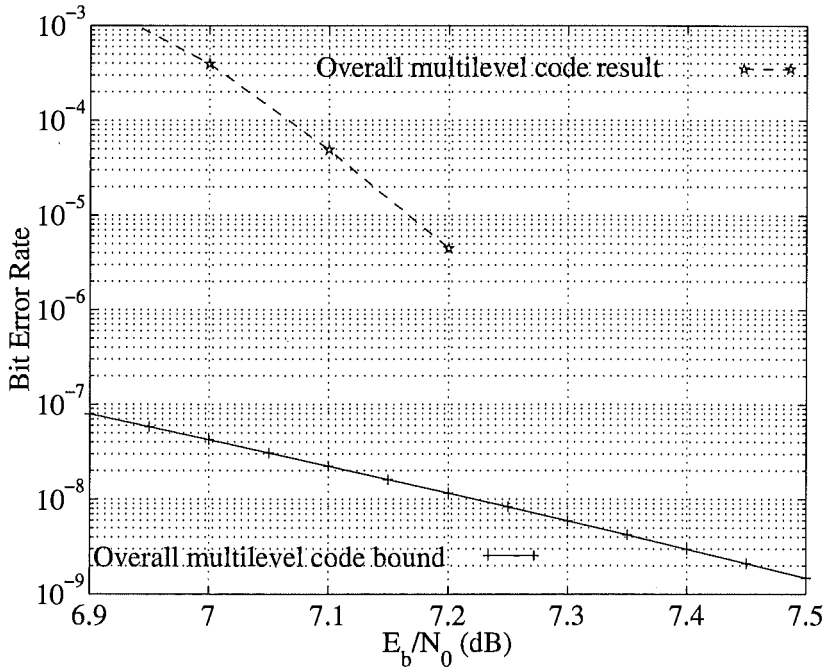


Figure 4.26: Performance bound for a multilevel code with the $(64, 57, 4)^2$ block Turbo code on level 1 and the $(64, 57, 4)$ extended BCH code on level 2 when transmitted on the AWGN channel using 16-QAM. Simulations results are shown after 10 iterations of an iterative MSD when the approximation approach is used and interleaving is used on level 2.

simulated results (of the data BER) for this code using the approximation approach, interleaving on level 2 and an iterative MSD are also shown. As can be seen the bound is not good at low SNR as only the minimum Hamming distance codewords are considered. However, the bound indicates that the minimum Hamming distance codewords will dominate the overall performance at BERs below 10^{-9}. This "error floor" effect is typical of Turbo coding schemes. The weight enumerators for the component codes were calculated using formulae from appendix C.

4.6 Conclusions

A new approach to iterative multistage decoding was derived and simulated in this chapter. A significant improvement in performance is achieved if an iterative MSD is used compared to a traditional (hard) MSD. As in chapter 3 using the extrinsic information as a Gaussian random variable rather than as a priori information improves performance dramatically. A variety of codes and design tradeoffs have been considered in this chapter. Only a slight improvement in overall performance is achieved if interleaving is used on later levels of the iterative MSD. The iterative MSD developed could be used with a variety of component codes, code structures, modulation schemes, SISO decoders, partitioning schemes and design rules (for example the equivalent capacity approach).

This chapter has also extended the notation of the adaptive SISO decoding algorithms developed in chapter 3 for use in an iterative MSD and with different modulation schemes. These approaches avoided the restrictions of the algorithm of [79, 90] by adaptively estimating the sequences of parameters and calculating the soft input and output using constellation points. The (SILO) Chase decoder was used by these algorithms, but other SILO decoders could be used.

The two level multilevel code with the $(64, 57, 4)^2$ block Turbo code on level 1 and the $(64, 57, 4)$ extended BCH code on level two was the closest multilevel code in this chapter to capacity. This code, the approximation based adaptive decoding algorithm and the iterative MSD achieved the goal of this thesis: manageable complexity, moderate block length (2048 symbols), high code rate (0.84), low BER (10^{-5} when only $1.53dB$ from capacity) and good bandwidth efficiency (3.37 data

bits per two-dimensional symbol).

Chapter 5

Block Turbo Codes on the Uncorrelated Rayleigh Flat Fading Channel

5.1 Introduction

This chapter considers the adaptive iterative decoding of block Turbo codes transmitted on the uncorrelated Rayleigh flat fading channel. Block Turbo codes have also been considered on the Rayleigh fading channel in [54, 77, 90]. The soft input and output to the adaptive decoders of chapter 3 are derived for QPSK on the uncorrelated Rayleigh flat fading channel in section 5.2. Then simulation results are presented in section 5.3 and conclusions are drawn in section 5.4.

5.2 Adaptive Iterative Decoding

In this section the soft input to and soft output from the component *soft-input soft-output (SISO)* decoders for a block Turbo code on the uncorrelated Rayleigh flat fading channel are derived. Recall from chapter 1 that in the case of coherent demodulation the received signal can be simplified to [83]

$$r(t) = \gamma_A(t)s(t) + g(t), \qquad 0 \leq t \leq T_s \tag{5.1}$$

where $\gamma_A(t)$ is the magnitude of the fading[1], $s(t)$ is the transmitted signal, $g(t)$ is the *additive white Gaussian noise (AWGN)* signal and T_s is the symbol period. When infinite interleaving is used with the flat fading channel the multiplicative fading is uncorrelated. The i^{th} fading sample, γ_i, of $\gamma_A(t)$ can then be written as [54]

$$\gamma_i = \sqrt{(x_{I,i})^2 + (x_{Q,i})^2} \tag{5.2}$$

where $x_{I,i}$ and $x_{Q,i}$ are the i^{th} samples in independent Gaussian random variables with zero mean. If the variance of both independent Gaussian random variables is set equal to 1/2, then $\gamma_A(t)$ will have an average energy of one and the fading will not alter the average *signal to noise ratio (SNR)*. The performance of QPSK can be simulated using BPSK if two BPSK symbols (corresponding to the inphase and quadrature components of the transmitted symbol) are multiplied by the same fading sample. Assuming BPSK is simulated to obtain the QPSK performance the i^{th} received sample can be written as

$$r_i = \gamma_i s_i + g_i \tag{5.3}$$

where $s_i = \pm 1$ is the i^{th} transmitted BPSK symbol and g_i is the i^{th} independent AWGN sample.

5.2.1 Without Channel State Information

This subsection considers the case when no *channel state information (CSI)* is available. As in the AWGN channel the probabilities in the *log-likelihood ratio (LLR)* are conditioned on both the received signal and extrinsic information from the previous decoding stage for the current component codeword. The soft output from the q^{th} decoder for the j^{th} bit position is given by the LLR

$$\begin{aligned} \Lambda'_j(q) &= \log\left(\frac{P(e_j = +1|\boldsymbol{R}, \boldsymbol{W}(q-1))}{P(e_j = -1|\boldsymbol{R}, \boldsymbol{W}(q-1))}\right) \\ &= \log\left(\frac{\sum_{\{\boldsymbol{C}^a|e_j=+1\}} P(\boldsymbol{C}^a|\boldsymbol{R}, \boldsymbol{W}(q-1))}{\sum_{\{\boldsymbol{C}^b|e_j=-1\}} P(\boldsymbol{C}^b|\boldsymbol{R}, \boldsymbol{W}(q-1))}\right) \end{aligned} \tag{5.4}$$

where e_j is the j^{th} bit[2] in a codeword, $\boldsymbol{W}(q-1) = (w_1(q-1), \cdots, w_n(q-1))$ is the extrinsic information vector from the previous decoding for the current codeword,

[1]The phase distortion has already been compensated for.

[2]For the purposes of computing likelihoods all bits are mapped from $\{0, 1\}$ to $\{-1, +1\}$.

$\boldsymbol{R} = (r_1, \cdots, r_n)$ is the received vector corresponding to the current codeword and n is the length of a codeword. $\{\boldsymbol{C}^a | e_j = +1\}$ and $\{\boldsymbol{C}^b | e_j = -1\}$ represent the sets of all codewords, $\boldsymbol{C}^a$ and $\boldsymbol{C}^b$, with $e_j = +1$ and $e_j = -1$ respectively. This can be simplified by assuming equiprobable codewords. In addition the received signal and extrinsic information are assumed to be independent. Then (5.4) can be written as

$$\Lambda'_j(q) = \log\left(\frac{\sum_{\{\boldsymbol{C}^a|e_j=+1\}} p(\boldsymbol{R}|\boldsymbol{C}^a)p(\boldsymbol{W}(q-1)|\boldsymbol{C}^a)}{\sum_{\{\boldsymbol{C}^b|e_j=-1\}} p(\boldsymbol{R}|\boldsymbol{C}^b)p(\boldsymbol{W}(q-1)|\boldsymbol{C}^b)}\right) \tag{5.5}$$

The sub-optimal SISO decoder approximates (5.5) by considering only a subset of all possible codewords. The subset of codewords is chosen by a *soft-input list-output (SILO)* decoder, as in chapter 3 the (SILO) Chase decoder will be used. The structure of the SISO decoder is shown in Fig. 2.8 and the structure of the Chase decoder is shown in Fig. 2.3. The subset of codewords chosen by the Chase decoder in set $\{\boldsymbol{C}^a | e_j = +1\}$ or $\{\boldsymbol{C}^b | e_j = -1\}$ is denoted $\{\boldsymbol{C}^a | e_j = +1\}_C$ or $\{\boldsymbol{C}^b | e_j = -1\}_C$ respectively. It is assumed that the extrinsic information [14] and received signal can be modelled as Gaussian random variables[3]. Using the approximation $\log(e^x + e^y) \approx \max(x, y)$ [43] the soft output of (5.5) may then be approximated as

$$\begin{aligned}\Lambda'_j(q) \approx & \max_{\{\boldsymbol{C}^a|e_j=+1\}_C} \log(p(\boldsymbol{R}|\boldsymbol{C}^a)p(\boldsymbol{W}(q-1)|\boldsymbol{C}^a)) \\ - & \max_{\{\boldsymbol{C}^b|e_j=-1\}_C} \log(p(\boldsymbol{R}|\boldsymbol{C}^b)p(\boldsymbol{W}(q-1)|\boldsymbol{C}^b)) = \Lambda''_j(q)\end{aligned} \tag{5.6}$$

Denote $\boldsymbol{D}$ as the codeword chosen in (5.6) which has the largest metric and $\boldsymbol{C}$ as the other codeword chosen in (5.6), it will have $c_j \neq d_j$. Now (5.6) can be written as

$$\begin{aligned}\Lambda''_j(q) = d_j \Bigg(& \frac{|\boldsymbol{R} - \mu_{\boldsymbol{R}|\boldsymbol{C}}|^2}{2\sigma^2_{\boldsymbol{R}|\boldsymbol{C}}} - \frac{|\boldsymbol{R} - \mu_{\boldsymbol{R}|\boldsymbol{D}}|^2}{2\sigma^2_{\boldsymbol{R}|\boldsymbol{D}}} \\ & + \frac{|\boldsymbol{W}(q-1) - \mu_{\boldsymbol{W}(q-1)|\boldsymbol{C}}|^2}{2\sigma^2_{\boldsymbol{W}(q-1)|\boldsymbol{C}}} - \frac{|\boldsymbol{W}(q-1) - \mu_{\boldsymbol{W}(q-1)|\boldsymbol{D}}|^2}{2\sigma^2_{\boldsymbol{W}(q-1)|\boldsymbol{D}}} \Bigg)\end{aligned} \tag{5.7}$$

where $\mu_{\boldsymbol{W}(q-1)|\boldsymbol{D}}$ is the mean and $\sigma^2_{\boldsymbol{W}(q-1)|\boldsymbol{D}}$ the variance of $\boldsymbol{W}(q-1)$ conditioned on $\boldsymbol{D}$, and $\mu_{\boldsymbol{R}|\boldsymbol{D}}$ is the mean and $\sigma^2_{\boldsymbol{R}|\boldsymbol{D}}$ the variance of $\boldsymbol{R}$ conditioned on $\boldsymbol{D}$. Similarly

[3]However, the received signal, $r(t)$, is actually a Rayleigh random variable, $\gamma_A(t)$, times the transmitted signal, $s(t)$, plus a Gaussian random variable, $g(t)$. The fading varies the instantaneous SNR, so this is equivalent to considering the case when the average SNR occurs.

$\mu_{\boldsymbol{W}(q-1)|\boldsymbol{C}}$ is the mean and $\sigma^2_{\boldsymbol{W}(q-1)|\boldsymbol{C}}$ the variance of $\boldsymbol{W}(q-1)$ conditioned on $\boldsymbol{C}$, and $\mu_{\boldsymbol{R}|\boldsymbol{C}}$ is the mean and $\sigma^2_{\boldsymbol{R}|\boldsymbol{C}}$ the variance of $\boldsymbol{R}$ conditioned on $\boldsymbol{C}$. The variance of the received signal is equal to the variance of the fading plus the variance of the AWGN. The mean of the received signal given the transmitted signal is not simply the transmitted signal as in the AWGN channel case and so must be retained in the equations. In practice the following is used

$$\Lambda''_j(q) = d_j \frac{|\boldsymbol{R} - \mu_{\boldsymbol{R}}\boldsymbol{C}|^2 - |\boldsymbol{R} - \mu_{\boldsymbol{R}}\boldsymbol{D}|^2}{2\sigma^2_{\boldsymbol{R}}} + d_j \frac{|\boldsymbol{W}(q-1) - \mu_{\boldsymbol{W}(q-1)}\boldsymbol{C}|^2 - |\boldsymbol{W}(q-1) - \mu_{\boldsymbol{W}(q-1)}\boldsymbol{D}|^2}{2\sigma^2_{\boldsymbol{W}(q-1)}} \tag{5.8}$$

where $\mu_{\boldsymbol{W}(q-1)}$ is the mean and $\sigma^2_{\boldsymbol{W}(q-1)}$ the variance of the absolute value of $\boldsymbol{W}(q-1)$, and $\mu_{\boldsymbol{R}}$ is the mean and $\sigma^2_{\boldsymbol{R}}$ the variance of the absolute value of $\boldsymbol{R}$ as used in [14]. The variance and mean of the extrinsic information are estimated over a block Turbo code block after each component code is decoded using the sample variance and mean (which were defined in chapter 3). The variance and mean of the received signal are estimated over a block Turbo code block prior to decoding using the sample variance and mean.

The soft output simplifies to

$$\Lambda''_j(q) = d_j \sum_{l=1, d_l \neq c_l}^{n} d_l \left(\frac{2\mu_{\boldsymbol{R}}}{\sigma^2_{\boldsymbol{R}}} r_l + \frac{2\mu_{\boldsymbol{W}(q-1)}}{\sigma^2_{\boldsymbol{W}(q-1)}} w_l(q-1) \right) \tag{5.9}$$

Now (5.9) is divided by $2\mu_{\boldsymbol{R}}/\sigma^2_{\boldsymbol{R}}$ giving a normalized soft output of

$$\Lambda_j(q) = d_j \sum_{l=1, d_l \neq c_l}^{n} d_l \left(r_l + \frac{\mu_{\boldsymbol{W}(q-1)}}{\mu_{\boldsymbol{R}}} \frac{\sigma^2_{\boldsymbol{R}}}{\sigma^2_{\boldsymbol{W}(q-1)}} w_l(q-1) \right) \tag{5.10}$$

Now consider the LLR for the soft input to the q^{th} decoder, which is defined as

$$\lambda'_j(q) = \log \left(\frac{P(e_j = +1 | r_j, w_j(q-1))}{P(e_j = -1 | r_j, w_j(q-1))} \right) \tag{5.11}$$

Assuming equiprobable bit values, this can be written as

$$\lambda'_j(q) = \log \left(\frac{p(r_j | e_j = +1) p(w_j(q-1) | e_j = +1)}{p(r_j | e_j = -1) p(w_j(q-1) | e_j = -1)} \right) \tag{5.12}$$

and in practice will be calculated using

$$\lambda'_j(q) = \frac{2\mu_{\boldsymbol{R}}}{\sigma^2_{\boldsymbol{R}}} r_j + \frac{2\mu_{\boldsymbol{W}(q-1)}}{\sigma^2_{\boldsymbol{W}(q-1)}} w_j(q-1) \tag{5.13}$$

Now divide by $2\mu_{\boldsymbol{R}}/\sigma^2_{\boldsymbol{R}}$ to obtain

$$\lambda_j(q) = r_j + \frac{\mu_{\boldsymbol{W}(q-1)}}{\mu_{\boldsymbol{R}}} \frac{\sigma^2_{\boldsymbol{R}}}{\sigma^2_{\boldsymbol{W}(q-1)}} w_j(q-1) \tag{5.14}$$

Comparing (5.10) and (5.14), the soft output can be written as a sum of soft inputs as

$$\Lambda_j(q) = d_j \sum_{l=1}^{n} \frac{\lambda_l(q)}{2}(d_l - c_l) = d_j \sum_{l=1, d_l \neq c_l}^{n} d_l \lambda_l(q) \tag{5.15}$$

In [90] the soft input is given as

$$\lambda_j(q) = r_j + \alpha(q) w_j(q-1) \tag{5.16}$$

Thus, by comparing (5.14) and (5.16) $\alpha(q)$ can be defined as

$$\alpha(q) = \frac{\mu_{\boldsymbol{W}(q-1)}}{\mu_{\boldsymbol{R}}} \frac{\sigma^2_{\boldsymbol{R}}}{\sigma^2_{\boldsymbol{W}(q-1)}} \tag{5.17}$$

which is the same as the AWGN case, except that the mean of the received signal has to be included in this case. The adaptive approaches to estimating $\boldsymbol{\beta}$ from chapter 3 can be used with this $\boldsymbol{\alpha}$ without modification.

5.2.2 With Channel State Information

This subsection considers the case when CSI is available. The CSI may not be ideal, but in the simulations of section 5.3 ideal CSI is assumed. The CSI vector is denoted $\boldsymbol{\gamma} = (\gamma_1, \cdots, \gamma_n)$.

The probabilities in the LLR are now conditioned on the received signal, the extrinsic information from the previous decoding stage and the CSI. The soft output from the q^{th} decoder for the j^{th} bit position is given by the LLR

$$\begin{aligned} \Lambda'_j(q) &= \log\left(\frac{P(e_j = +1|\boldsymbol{R}, \boldsymbol{W}(q-1), \boldsymbol{\gamma})}{P(e_j = -1|\boldsymbol{R}, \boldsymbol{W}(q-1), \boldsymbol{\gamma})}\right) \\ &= \log\left(\frac{\sum_{\{\boldsymbol{C}^a|e_j=+1\}} P(\boldsymbol{C}^a|\boldsymbol{R}, \boldsymbol{W}(q-1), \boldsymbol{\gamma})}{\sum_{\{\boldsymbol{C}^b|e_j=-1\}} P(\boldsymbol{C}^b|\boldsymbol{R}, \boldsymbol{W}(q-1), \boldsymbol{\gamma})}\right) \end{aligned} \tag{5.18}$$

This can be simplified by assuming equiprobable codewords. In addition both the received signal and the CSI are assumed to be independent of the extrinsic information. It is assumed that the extrinsic information [14] and the received signal

(conditioned on the CSI and a codeword) can be modelled as Gaussian random variables. Then (5.18) can be written as

$$\Lambda'_j(q) = \log\left(\frac{\sum_{\{\boldsymbol{C}^a|e_j=+1\}} p(\boldsymbol{R}|\boldsymbol{C}^a,\boldsymbol{\gamma})p(\boldsymbol{W}(q-1)|\boldsymbol{C}^a)}{\sum_{\{\boldsymbol{C}^b|e_j=-1\}} p(\boldsymbol{R}|\boldsymbol{C}^b,\boldsymbol{\gamma})p(\boldsymbol{W}(q-1)|\boldsymbol{C}^b)}\right) \tag{5.19}$$

If the extrinsic information is used to estimate the CSI, then the CSI is denoted $\boldsymbol{\gamma}(q) = (\gamma_1(q), \cdots, \gamma_n(q))$. In this case the extrinsic information and the CSI are not assumed to be independent and the soft output of (5.19) can be written as

$$\Lambda'_j(q) = \log\left(\frac{\sum_{\{\boldsymbol{C}^a|e_j=+1\}} p(\boldsymbol{R}|\boldsymbol{C}^a,\boldsymbol{\gamma}(q))p(\boldsymbol{W}(q-1)|\boldsymbol{C}^a,\boldsymbol{\gamma}(q))}{\sum_{\{\boldsymbol{C}^b|e_j=-1\}} p(\boldsymbol{R}|\boldsymbol{C}^b,\boldsymbol{\gamma}(q))p(\boldsymbol{W}(q-1)|\boldsymbol{C}^b,\boldsymbol{\gamma}(q))}\right) \tag{5.20}$$

This case will not be considered further in this chapter.

The sub-optimal SISO decoder approximates (5.19) by considering only a subset of all possible codewords. Using the approximation $\log(e^x + e^y) \approx \max(x, y)$ [43] the soft output of (5.19) may be approximated as

$$\begin{aligned}\Lambda'_j(q) \approx &\max_{\{\boldsymbol{C}^a|e_j=+1\}_C} \log(p(\boldsymbol{R}|\boldsymbol{C}^a,\boldsymbol{\gamma})p(\boldsymbol{W}(q-1)|\boldsymbol{C}^a)) \\ &- \max_{\{\boldsymbol{C}^b|e_j=-1\}_C} \log(p(\boldsymbol{R}|\boldsymbol{C}^b,\boldsymbol{\gamma})p(\boldsymbol{W}(q-1)|\boldsymbol{C}^b)) = \Lambda''_j(q)\end{aligned} \tag{5.21}$$

Again denote $\boldsymbol{D}$ as the codeword chosen in (5.21) with the largest metric and $\boldsymbol{C}$ as the other codeword chosen in (5.21), it has $c_j \neq d_j$. Now (5.21) can be written as

$$\begin{aligned}\Lambda''_j(q) = d_j\Bigg(&\frac{|\boldsymbol{R}-\mu_{\boldsymbol{R}|\boldsymbol{C},\boldsymbol{\gamma}}|^2}{2\sigma^2_{\boldsymbol{R}|\boldsymbol{C},\boldsymbol{\gamma}}} - \frac{|\boldsymbol{R}-\mu_{\boldsymbol{R}|\boldsymbol{D},\boldsymbol{\gamma}}|^2}{2\sigma^2_{\boldsymbol{R}|\boldsymbol{D},\boldsymbol{\gamma}}} \\ &+\frac{|\boldsymbol{W}(q-1)-\mu_{\boldsymbol{W}(q-1)|\boldsymbol{C}}|^2}{2\sigma^2_{\boldsymbol{W}(q-1)|\boldsymbol{C}}} - \frac{|\boldsymbol{W}(q-1)-\mu_{\boldsymbol{W}(q-1)|\boldsymbol{D}}|^2}{2\sigma^2_{\boldsymbol{W}(q-1)|\boldsymbol{D}}}\Bigg)\end{aligned} \tag{5.22}$$

where $\mu_{\boldsymbol{W}(q-1)|\boldsymbol{D}}$ is the mean and $\sigma^2_{\boldsymbol{W}(q-1)|\boldsymbol{D}}$ the variance of $\boldsymbol{W}(q-1)$ conditioned on $\boldsymbol{D}$, and $\mu_{\boldsymbol{R}|\boldsymbol{D},\boldsymbol{\gamma}}$ is the mean and $\sigma^2_{\boldsymbol{R}|\boldsymbol{D},\boldsymbol{\gamma}}$ the variance of $\boldsymbol{R}$ conditioned on $\boldsymbol{D}$ and $\boldsymbol{\gamma}$. Similarly $\mu_{\boldsymbol{W}(q-1)|\boldsymbol{C}}$ is the mean and $\sigma^2_{\boldsymbol{W}(q-1)|\boldsymbol{C}}$ the variance of $\boldsymbol{W}(q-1)$ conditioned on $\boldsymbol{C}$, and $\mu_{\boldsymbol{R}|\boldsymbol{C},\boldsymbol{\gamma}}$ is the mean and $\sigma^2_{\boldsymbol{R}|\boldsymbol{C},\boldsymbol{\gamma}}$ the variance of $\boldsymbol{R}$ conditioned on $\boldsymbol{C}$ and $\boldsymbol{\gamma}$. In practice the following is used

$$\begin{aligned}\Lambda''_j(q) = d_j&\frac{|\boldsymbol{R}-\boldsymbol{\gamma}\boldsymbol{C}|^2 - |\boldsymbol{R}-\boldsymbol{\gamma}\boldsymbol{D}|^2}{2\sigma^2_{\boldsymbol{R}}} \\ +d_j&\frac{|\boldsymbol{W}(q-1)-\mu_{\boldsymbol{W}(q-1)}\boldsymbol{C}|^2 - |\boldsymbol{W}(q-1)-\mu_{\boldsymbol{W}(q-1)}\boldsymbol{D}|^2}{2\sigma^2_{\boldsymbol{W}(q-1)}}\end{aligned} \tag{5.23}$$

where $\mu_{\boldsymbol{W}(q-1)}$ is the mean and $\sigma^2_{\boldsymbol{W}(q-1)}$ the variance of the absolute value of $\boldsymbol{W}(q-1)$ as used in [14], and $\sigma^2_{\boldsymbol{R}}$ is the variance of the AWGN. The variance $\sigma^2_{\boldsymbol{W}(q-1)}$ and the mean $\mu_{\boldsymbol{W}(q-1)}$ are estimated over a block Turbo code block after each decoding stage using the sample variance and mean. The variance of the AWGN, $\sigma^2_{\boldsymbol{R}}$, is (assumed) known when CSI is available.

Since the fading samples are uncorrelated (independent) the soft output simplifies to

$$\Lambda''_j(q) = d_j \sum_{l=1, d_l \neq c_l}^{n} d_l \left(\frac{2}{\sigma^2_{\boldsymbol{R}}} \gamma_l r_l + \frac{2\mu_{\boldsymbol{W}(q-1)}}{\sigma^2_{\boldsymbol{W}(q-1)}} w_l(q-1) \right) \tag{5.24}$$

Now (5.24) is divided by $2/\sigma^2_{\boldsymbol{R}}$ giving a normalized soft output of

$$\Lambda_j(q) = d_j \sum_{l=1, d_l \neq c_l}^{n} d_l \left(\gamma_l r_l + \mu_{\boldsymbol{W}(q-1)} \frac{\sigma^2_{\boldsymbol{R}}}{\sigma^2_{\boldsymbol{W}(q-1)}} w_l(q-1) \right) \tag{5.25}$$

Consider the LLR for the soft input to the q^{th} decoder, which is defined as

$$\lambda'_j(q) = \log \left(\frac{P(e_j = +1 | r_j, w_j(q-1), \gamma_j)}{P(e_j = -1 | r_j, w_j(q-1), \gamma_j)} \right) \tag{5.26}$$

Assuming equiprobable bit values, this can be written as

$$\lambda'_j(q) = \log \left(\frac{p(r_j | e_j = +1, \gamma_j) p(w_j(q-1) | e_j = +1)}{p(r_j | e_j = -1, \gamma_j) p(w_j(q-1) | e_j = -1)} \right) \tag{5.27}$$

and in practice will be calculated using

$$\lambda'_j(q) = \frac{2}{\sigma^2_{\boldsymbol{R}}} \gamma_j r_j + \frac{2\mu_{\boldsymbol{W}(q-1)}}{\sigma^2_{\boldsymbol{W}(q-1)}} w_j(q-1) \tag{5.28}$$

Now divide by $2/\sigma^2_{\boldsymbol{R}}$ to obtain

$$\lambda_j(q) = \gamma_j r_j + \mu_{\boldsymbol{W}(q-1)} \frac{\sigma^2_{\boldsymbol{R}}}{\sigma^2_{\boldsymbol{W}(q-1)}} w_j(q-1) \tag{5.29}$$

Comparing (5.25) and (5.29), the soft output can be written as a sum of soft inputs as

$$\Lambda_j(q) = d_j \sum_{l=1}^{n} \frac{\lambda_l(q)}{2} (d_l - c_l) = d_j \sum_{l=1, d_l \neq c_l}^{n} d_l \lambda_l(q) \tag{5.30}$$

The soft input can be written as

$$\lambda_j(q) = \gamma_j r_j + \alpha(q) w_j(q-1) \tag{5.31}$$

where (by comparing (5.29) and (5.31))

$$\alpha(q) = \mu_{\boldsymbol{W}(q-1)} \frac{\sigma^2_{\boldsymbol{R}}}{\sigma^2_{\boldsymbol{W}(q-1)}} \tag{5.32}$$

which is the same as the AWGN case. The adaptive approaches to estimating $\boldsymbol{\beta}$ from chapter 3 can be used with this $\boldsymbol{\alpha}$ without modification.

5.3 Simulation Results

This section presents simulation results using the adaptively estimated $\boldsymbol{\alpha}$ of section 5.2 and the approximation based adaptively estimated $\boldsymbol{\beta}$ of chapter 3. This will be called the approximation approach. The $(64, 57, 4)^2$ block Turbo code is transmitted using QPSK on the uncorrelated Rayleigh flat fading channel. The Chase decoder uses a set of 16 test sequences (for $p = 4$). The approximation approach is compared to the precomputed approach of [54] (described in section 2.3). In [54] $\boldsymbol{\alpha} = [0, 0.3, 0.35, 0.5, 0.7, 0.9, 1.2, 1.5]$ and $\boldsymbol{\beta} = [0.2, 0.4, 0.6, 0.8, 1, 1, 1, 1]$ for a four iteration simulation. All positions in the extrinsic information are used in the estimation of $\mu_{\boldsymbol{W}(q-1)}$ and $\sigma^2_{\boldsymbol{W}(q-1)}$. The *bit error rate (BER)* of the data bits is displayed.

All simulation results presented in this chapter were obtained after finding at least 200 block Turbo code blocks in error after the final iteration for a given E_b/N_0.

5.3.1 Without Channel State Information

This subsection considers the case when no CSI is available. The simulation results are shown in Fig. 5.1 for 1, 2, 3 and 4 iterations, where E_b is the average energy used to transmit a data bit and N_0 is the 2-sided power spectral density of the noise (baseband). Performance improves with each iteration, but with diminishing returns. The approximation approach improves performance over the precomputed approach of [54] after the first iteration.

5.3.2 With Channel State Information

This subsection considers the case when CSI is available. Recall from chapter 3 that the Chase decoder can find codewords a long (Euclidean) distance from the soft input to the decoder. This occurs because even though the error patterns used by the Chase decoder only alter the *least reliable positions (LRPs)*, the *hard-input hard-output (HIHO)* decoder in the Chase decoder does not use reliability information and may change t positions which are very reliable, where t is the error correction capability of the component code. The overall parity bit may also be changed by the Chase decoder and may be very reliable. This means that the closest codeword

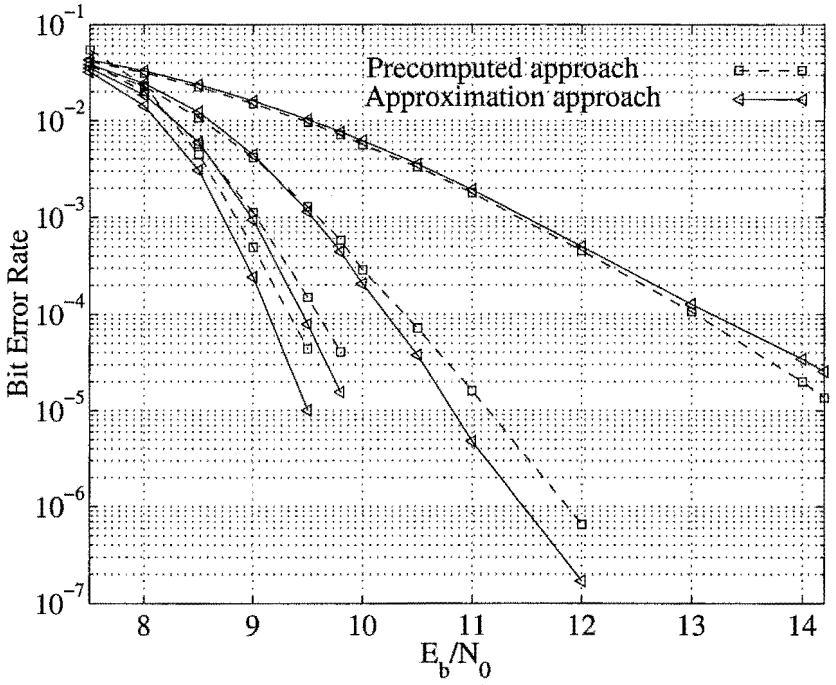


Figure 5.1: Performance of the $(64, 57, 4)^2$ block Turbo code using the precomputed and approximation approaches, $p = 4$, 2^p test sequences, no CSI and QPSK on the uncorrelated Rayleigh flat fading channel. The first four iterations are displayed.

with $c_j \neq d_j$ may not be considered; instead a competing codeword, $\boldsymbol{C}$, a long way from the soft input vector may be used. This results in the extrinsic information calculated using $\boldsymbol{C}$ being too large in some positions. This can be advantageous in some positions, but overall seriously degrades performance. This problem will also be encountered when using other SILO decoders such as that of [99]. This problem is worse when CSI is used as the range of values in the soft input to the decoder is larger. A solution to this problem is to only use competing codewords within a specified squared Euclidean distance of the soft input vector. This distance will be called the maximum allowed value.

In chapter 3, (3.37), it was found that the squared Euclidean distance between a competing codeword, $\boldsymbol{C}$, and the soft input vector, $\boldsymbol{\lambda}(q)$, is given by

$$d_E^2(\boldsymbol{\lambda}(q), \boldsymbol{C}) = d_E^2(\boldsymbol{\lambda}(q), \boldsymbol{D}) + 4 \sum_{l=1, c_l \neq d_l}^{n} (\lambda_l(q) d_l) \tag{5.33}$$

A maximum allowed value of $d_E^2(\boldsymbol{\lambda}(q), \boldsymbol{C})$ can be developed based on this definition. The first term in (5.33) can be calculated exactly and the second term can be used to set an upper limit on the maximum allowed value of $d_E^2(\boldsymbol{\lambda}(q), \boldsymbol{C})$. Any codewords

with a greater squared Euclidean distance will not be used to calculate the extrinsic information.

It is important not to make the maximum allowed value of $d_E^2(\boldsymbol{\lambda}(q), \boldsymbol{C})$ too small or all codewords found apart from $\boldsymbol{D}$ may be eliminated. In an attempt to reduce this problem, when two or more codewords are found by the Chase decoder, the two closest codewords to the soft input vector will automatically be retained (one of which is $\boldsymbol{D}$). This ensures that when two or more codewords are found $\beta_j(q)$ can be calculated exactly for some positions, which means a better estimate can be used by the approximation approach.

If the maximum allowed value is too large, then it will allow codewords that are too far from the soft input vector. But if it is too small then valid competing codewords will be excluded. The following maximum allowed value was developed keeping this in mind. Initially the maximum allowed value is set equal to $d_E^2(\boldsymbol{\lambda}(q), \boldsymbol{D})$. Looking at (5.33) at least $d_{H,min}$ positions must be added to the maximum allowed value, where $d_{H,min}$ is the minimum Hamming distance of the code (the minimum number of positions where $c_l \neq d_l$). The maximum value is achieved when all the positions summed have $c_l \neq (d_l = y_l(q))$ (meaning $\lambda_l(q)d_l \geq 0$), where $y_l(q)$ is the sign of $\lambda_l(q)$. Outside the p LRPs $c_l \neq (d_l = y_l(q))$ can only be true in a maximum of $t+1$ positions, since $\boldsymbol{C}$ can change at most $t+1$ positions (including the overall parity position). Since the overall parity position can be very reliable it is not included in the sum. However, $\lambda_l(q)d_l$ in t positions outside the p LRPs are added to the maximum allowed value. As an estimate, t times the mean of the absolute value of the soft input is used to represent these positions in the maximum allowed value. Now $\lambda_l(q)d_l$ in at least $v = d_{H,min} - t$ other positions must be added to the maximum allowed value. Up to $q \leq v$ of the most reliable positions with $\lambda_j(q)d_j > 0$ in the p LRPs are now added to the maximum allowed value, then $v = v - q$ other positions must be summed. If $v > 0$, then the v LRPs where $\lambda_j(q)d_j > 0$ outside the p LRPs are also summed. This is used so that the maximum allowed value will not be too small.

The simulation results are shown in Fig. 5.2 for the approximation approach using the maximum allowed value described above and that for the precomputed

approach[4] of [54] when CSI is available. Performance improves with each iteration, but with diminishing returns. The approximation approach improves performance in iteration 1, 3 and 4. After 3 iterations the approximation approach is slightly better than the precomputed approach after 4 iterations. Performance may be improved further by using a different maximum allowed value, more test sequences or a different SILO decoder to choose the subset of codewords considered.

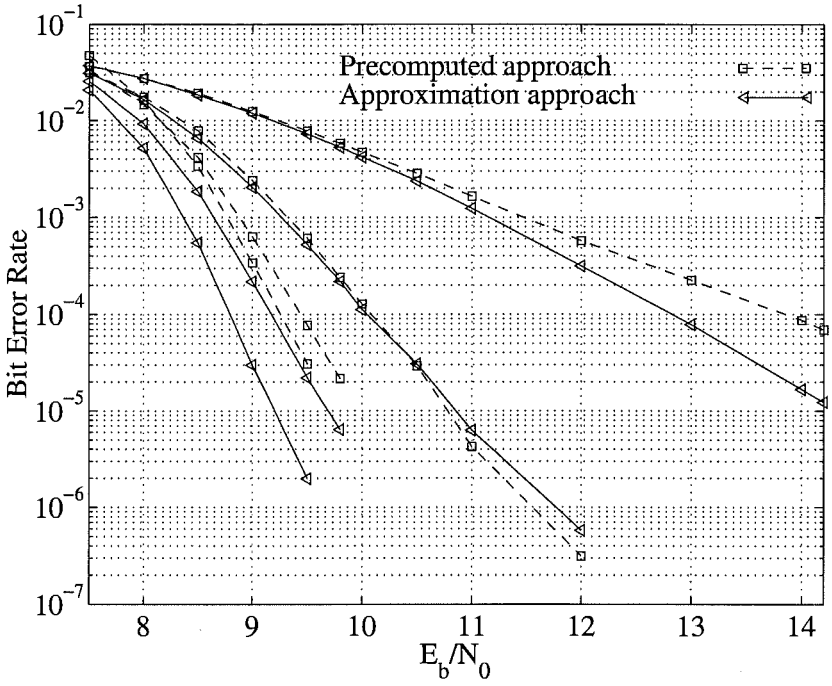


Figure 5.2: Performance of the $(64, 57, 4)^2$ block Turbo code using the precomputed approach and the approximation approach (with maximum allowed value), $p = 4$, 2^p test sequences, CSI and QPSK on the uncorrelated Rayleigh flat fading channel. The first four iterations are displayed.

In Fig. 5.3 the simulation results for the $(64, 57, 4)^2$ block Turbo code using the approximation approach are shown when ideal CSI and no CSI is available. As can be seen in every iteration (other than the second iteration below a BER of 10^{-5}) performance is improved by using CSI. But the gain is less than $0.35dB$.

In chapter 1 it was stated that there is potential for larger coding gains in the fading channel than the AWGN channel. In Fig. 5.4 the performance on the Rayleigh flat fading channels when ideal CSI is used is compared to the uncoded

[4]This approach does not use the maximum allowed value. This approach normalizes the absolute value of the extrinsic information, so that it has a mean absolute value of one.

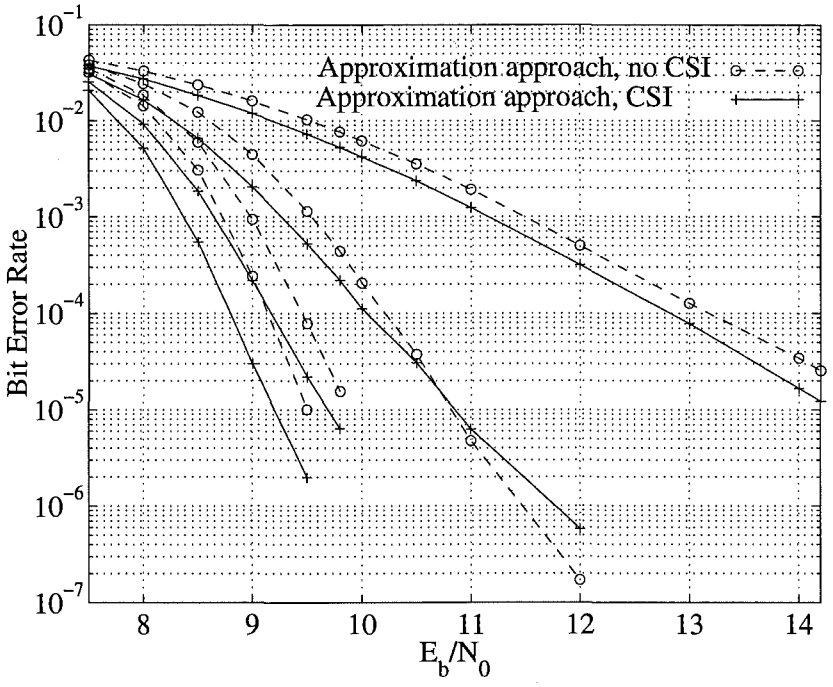


Figure 5.3: Performance of the $(64, 57, 4)^2$ block Turbo code using the approximation approach (with maximum allowed value), $p = 4$, 2^p test sequences and QPSK on the uncorrelated Rayleigh flat fading channel. The first four iterations are displayed. Both ideal and no CSI cases are considered.

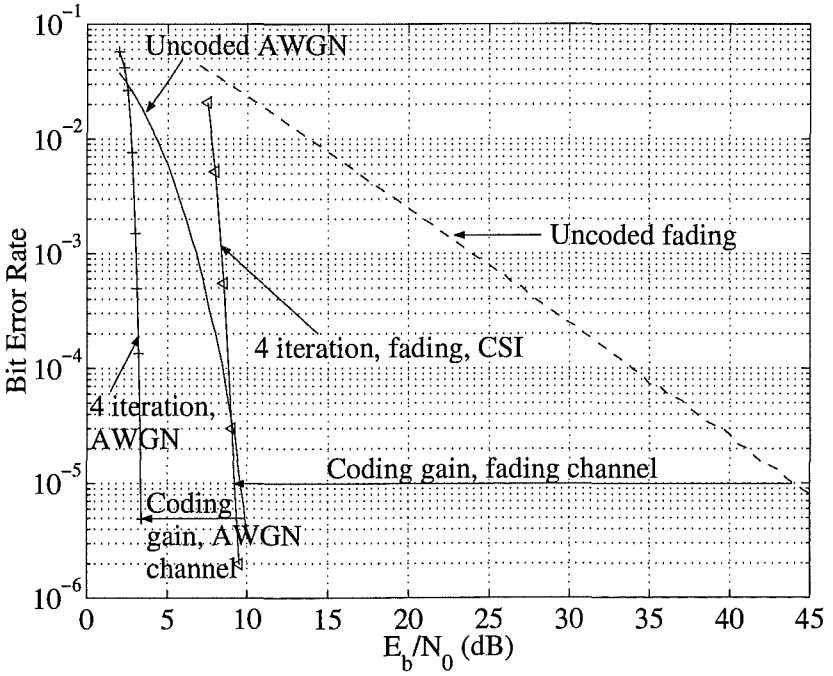


Figure 5.4: Performance of the $(64, 57, 4)^2$ block Turbo code using the approximation approach (with maximum allowed value on the fading channel), $p = 4$, 2^p test sequences on the uncorrelated Rayleigh flat fading channel and the AWGN channel. The performance after four iterations is displayed against the uncoded performance on the AWGN channel and Rayleigh flat fading channel (with CSI).

performance. As can be seen there is a coding gain of $\approx 35dB$ at a BER of 10^{-5}. The performance is also shown for the AWGN channel. The coding gain on the AWGN channel is $\approx 6dB$ at a BER of 10^{-5}. The coding and iterative decoding have managed to improve the slope of the fading curve dramatically. It now has a similar slope to the results for the AWGN channel.

5.4 Conclusions

In this chapter $\boldsymbol{\alpha}$ was derived for the uncorrelated Rayleigh flat fading channel with and without CSI. The approximation based approach to estimating $\boldsymbol{\beta}$ was simulated with this $\boldsymbol{\alpha}$ (which is adaptively estimated) on the uncorrelated Rayleigh flat fading channel with and without CSI. This algorithm improves performance over the precomputed approach of [54] in both cases.

Chapter 6

Conclusions and Suggestions for Future Work

6.1 Introduction

This thesis made contributions to the fields of block Turbo coding and multilevel coding. The main contributions to block Turbo coding are the two adaptive *soft-input soft-output (SISO)* decoding algorithms developed for block codes. These algorithms enable moderate complexity iterative decoding of block Turbo codes with long component codes. The main contribution to multilevel coding is the development and evaluation of an iterative *multistage decoder (MSD)*. In this chapter conclusions on the work presented in this thesis are given. Then suggestions for future work are presented.

6.2 Conclusions

In chapter 3 two adaptive SISO decoding algorithms for block codes were developed, based on [54, 90]. These adaptive algorithms scale the extrinsic information by an adaptively estimated sequence of parameters, $\boldsymbol{\alpha}$. In addition they estimate the extrinsic information when it cannot be calculated directly, by using adaptive estimates of a sequence $\boldsymbol{\beta}$. One algorithm uses an approximation based approach to estimating $\boldsymbol{\beta}$ and the other estimates $\boldsymbol{\beta}$ using the distance properties of the component codes.

In chapter 3, $\boldsymbol{\alpha}$ was derived for the *additive white Gaussian noise (AWGN)*

channel by treating the extrinsic information as a Gaussian random variable (the Gaussian approach). When the Gaussian approach is used α takes into account the different mean and variance of the received signal and extrinsic information. The components in α are adaptively estimated on a block-by-block basis. As a result they can adapt to varying channel conditions, and treat good and bad blocks differently, unlike the precomputed approach to α of [54, 90].

In addition, two adaptive approaches to estimating β were developed in chapter 3 for use with the adaptively estimated α. One approach is based on approximation (used in the approximation algorithm), and the other is based on the distance properties of the component codes (used in the distance algorithm). Both approaches estimate β for each codeword or code block. This means that the estimated value of β (and therefore, the extrinsic information) can be small, corresponding to a low reliability, when a block or codeword is received with lots of errors. Conversely it can be large, corresponding to a high reliability, when a block or codeword is received with no or very few errors (a good block/ codeword). As a result both approaches adapt to varying channel conditions, and treat good and bad blocks differently, unlike the precomputed approach of [54, 90].

The performance of the two adaptive algorithms is discussed below:

- The approximation algorithm provided a significant gain over the precomputed approach of [54, 90] for a variety of codes using BPSK/ QPSK on the AWGN channel. The $(64, 57, 4)^2$ block Turbo code, decoded using the approximation algorithm, is within 1.04dB of capacity at a *bit error rate (BER)* of 10^{-5} after 10 iterations. Simulations found that treating the extrinsic information as a Gaussian random variable resulted in better performance than using it as a priori information (a priori approach). When the a priori approach is used the mean and variance of the extrinsic information are not taken into account.

- The distance algorithm performed[1] better than the approximation algorithm when perfect codes were used in a block Turbo code, but not when quasi-perfect codes[2] were used in a block Turbo code.

[1]Unless otherwise stated performance refers to the BER of the data bits.

[2]The component codes in the block Turbo codes were extended BCH codes. The corresponding unextended BCH codes were either perfect or quasi-perfect codes.

In chapter 4 the bandwidth efficiency was improved by using higher order modulations and a multilevel code. An iterative MSD was developed for the AWGN channel to provide a good compromise between performance and complexity. The iterative MSD passes reliability/ extrinsic information to all previous and subsequent levels. The extrinsic information is treated as a Gaussian random variable when calculating the soft input to each level. As a result the different mean and variance of the extrinsic information and the received signal are taken into account (as shown in chapters 3, 4 and 5).

Each level of the iterative MSD is decoded using SISO decoders so that extrinsic information can be passed to other levels. The two adaptive SISO decoding algorithms of chapter 3 are used in the iterative MSD with extended notation[3]. These algorithms allow long block codes and block Turbo codes to be used as component codes in multilevel codes with a manageable decoding complexity. In simulations a block Turbo code was used on level one and extended BCH component codes were used on subsequent levels. The results for simulations transmitting 16-QAM on the AWGN channel are given below:

- The approximation algorithm provided a significant improvement over the algorithm of [54, 79, 90] in a traditional MSD. But an even larger improvement was obtained by using the approximation algorithm in the iterative MSD of chapter 4. Treating the extrinsic information as a Gaussian random variable resulted in better performance than treating it as a priori information when the iterative MSD and the approximation algorithm were used. In this case a two level multilevel code with the $(64, 57, 4)^2$ block Turbo code on level 1 and the $(64, 57, 4)$ extended BCH code on level two was within 1.53dB of capacity at a BER of 10^{-5} after 10 iterations. This code transmitted 3.37 data bits per 16-QAM symbol and 2048 16-QAM symbols per block. Therefore, on the AWGN channel this code, the approximation algorithm and the iterative MSD achieved: manageable complexity, moderate block length, high code rate, low BER and good bandwidth efficiency.

- The performance of the distance algorithm was very similar to that of the

[3]The extended notation developed in chapter 4 allows the algorithms to be used with different modulations and with the iterative MSD of chapter 4.

approximation algorithm when used in the iterative MSD.

Different component codes and decoding structures were also investigated. Simulations considered order of decoding, passing combinations of hard and soft information between levels, and tradeoffs between the BER and code rate. This work showed the potential of multilevel codes decoded using the iterative MSD for unequal error protection applications. It also showed that the best performance was obtained when soft/ reliability information was passed among all levels.

In chapter 5 the approximation algorithm was used to iteratively decode a block Turbo code transmitted on the uncorrelated Rayleigh flat fading channel using QPSK. In simulations it was assumed that the phase had been compensated prior to decoding. Performance was improved over the precomputed approach of [54] with and without ideal *channel state information (CSI)*. Chapter 5 showed the potential of block Turbo codes in a fading environment. Simulation results obtained for the $(64, 57, 4)^2$ block Turbo code after 10 iterations, using ideal CSI, were approximately 6dB from those obtained on the AWGN channel at a BER of 10^{-5}.

In conclusion this thesis has developed a robust, high performance, moderate complexity SISO decoder for block codes and a high performance iterative MSD. Both performed well for a variety of codes on the AWGN channel. In addition block Turbo codes decoded using the robust approximation based SISO decoder have been shown to perform well on the uncorrelated Rayleigh flat fading channel.

6.3 Future Work

In chapter 3 two adaptive SISO decoding algorithms for block codes were developed. Although these algorithms perform better than the non-adaptive SISO decoding algorithm of [54, 90] further improvements can be made. In this section some suggestions for future work on adaptive SISO decoding algorithms for block codes are given. Suggestions for future work on channel estimation, multilevel codes and iterative multistage decoding are also presented.

The adaptive SISO decoding algorithms of chapter 3 and the non-adaptive SISO decoding algorithm of [54, 90] reduce the decoding complexity by considering a subset of paths through the decoding trellis. Equalization can also be performed

using a trellis. Therefore, the complexity reduction ideas used in these decoding algorithms can also be used in the equalizer. Suggestions for future work on equalization using these concepts are also given in this section.

6.3.1 Error Correction Coding/ Decoding

There are many possible extensions to the error correction coding/ decoding schemes presented in this thesis. One of the simplest extensions is to use the SISO decoding algorithms developed in chapters 3 and 4 to decode different concatenated codes and block codes. The iterative MSD developed in chapter 4 could also be used to decode different component codes. Furthermore, different SISO decoding algorithms could also be used in the iterative MSD. This subsection also gives suggestions for future work on

- *soft-input list-output (SILO)* decoding algorithms,
- decoding strategies for multilevel codes,
- estimating β,
- restricting which codewords are used to calculate the extrinsic information,
- and fading channels.

List Based Decoding Algorithms

The Chase decoder [19] (as described in section 2.2) is used in the adaptive SISO decoding algorithms developed in chapters 3, 4 and 5. It is a simple, low complexity SILO decoder that can provide good performance[4]. However, performance may be improved by using a different set of error patterns. The distance properties of the component codes could be useful in choosing these error patterns.

Performance may also be improved by using a different SILO decoder. In [19, 26, 30, 33, 34, 35, 36, 40, 46, 55, 56, 99, 105, 106, 109, 110] a number of algorithms for SILO decoders are described. Some have fixed complexity, while others vary the complexity depending on the reliability of the soft input. Additional SILO decoders

[4]It was used in this work so that performance could be compared to the non-adaptive algorithm of [54, 90] (described in section 2.3).

could be created by combining techniques from one or more of these algorithms. This could result in better performance and should be investigated.

Decoding Strategies for Multilevel Codes

In section 4.2 it was suggested that the iterative MSD could be implemented using parallel decoding techniques (meaning more than one level could be decoded at the same time). This could reduce decoding delay and should be considered for practical implementations of the iterative MSD of chapter 4.

It would also be interesting to investigate where the decoding complexity is best spent in an iterative MSD. Consider a multilevel code with a concatenated code on level one. The simulations of chapter 4 decoded all levels in the multilevel code once during each iteration. However, better performance for a given complexity may be obtained if the first level is decoded more than once during an iteration of the iterative MSD. For example, each level could be decoded once on the first iteration of the iterative MSD, but on each subsequent iteration of the iterative MSD the level one code could be decoded several times. For multilevel codes with more than two levels the iterative MSD could decode all levels on the first iteration and then iterate between the two lowest levels on subsequent iterations. Then all levels could be decoded once on each of the final two iterations.

Probability Approach to β

In chapter 3 expressions for β were developed using the original soft input and output LLRs. These expressions were given in[5] (3.22) and (3.23), but no low-complexity way to compute them was found and it remains an open problem. If (3.22) or (3.23) could be calculated (or an expected value found) in a low complexity manner, without knowledge of a competing codeword, then they could be used to calculate β when no competing codeword is found.

[5] One problem with these expressions is that they do not use the max-log-MAP approximation, $\log(e^x + e^y) \approx \max(x, y)$, which is used to calculate the soft output in simulations.

Distance Approach to Adaptively Estimating β

In chapter 3 an approach to adaptively estimating β based on the distance properties of the component codes was developed (the distance approach). It was found to improve performance over the approximation approach when perfect codes were used in a block Turbo code, but not when quasi-perfect codes were used in a block Turbo code[6]. The estimate for the q^{th} decoding stage, $\hat{\beta}(q)$, was calculated by summing the hard output times the soft input in a combination of positions. In chapter 3 a set of combinations in the *least reliable positions (LRPs)* was used to produce a set of possible estimates. Two approaches to choosing the set were developed based on the distance properties of the component code. However, the performance may be improved if the set is determined using the parity check equations and distance properties of the component code. In chapter 3 the minimum positive estimate in the set was used to calculate the extrinsic information, (3.30). However, there may be a better way of choosing the estimate. For example, if the combinations in the set were ordered according to decreasing probability, then the first positive estimate calculated could be used. This would result in a reduction in the average complexity.

Restricting the Competing Codewords Used

In chapters 3 and 5 performance was improved, in some cases, by restricting which codewords from the SILO decoder were used to calculate the extrinsic information. Only codewords, $\boldsymbol{C}$, within a given squared Euclidean distance of the soft input[7], $\boldsymbol{\lambda}(q)$, were used. This squared Euclidean distance was called the maximum allowed value of $d_E^2(\boldsymbol{C}, \boldsymbol{\lambda}(q))$. It was calculated on a codeword-by-codeword basis as the reliability of the soft input vectors could vary significantly. Different ways of calculating this were developed in each chapter as different channels were considered (namely the memoryless AWGN channel and uncorrelated Rayleigh flat fading channel with ideal CSI). In both cases it was better not to use the maximum allowed value if distant codewords were not significantly degrading performance. Therefore, an important open problem is to develop an expression for the maximum allowed

[6]The component codes in the block Turbo codes were extended BCH codes. The corresponding unextended BCH codes were either perfect or quasi-perfect codes.

[7]For performance reasons at least two codewords were always retained (when found) regardless of how far they were from the soft input in terms of squared Euclidean distance.

value of $d_E^2(\boldsymbol{C}, \boldsymbol{\lambda}(q))$, which can be used with a variety of codes, SILO decoders, test sequences and channels, whether or not distant competing codewords are adversely affecting performance.

Fading Channels

In chapter 5 the approximation algorithm was used to decode component codes in a block Turbo code transmitted using QPSK on the uncorrelated Rayleigh flat fading channel. Suggestions for future work in this area include using the iterative MSD of chapter 4 to decode a multilevel code transmitted using a higher order modulation on the uncorrelated Rayleigh flat fading channel. In addition, the adaptive SISO decoding algorithms of chapters 3 and 4, and the iterative MSD of chapter 4 could be extended for use on the correlated Rayleigh flat and frequency-selective fading channels. Decoders should be developed for these channels when there is ideal, imperfect or no CSI available. In most practical applications an (imperfect) estimate of the CSI is used. One area for future work is to develop a method of using the extrinsic information from the decoder in the channel state estimator. In this case the soft output LLR is given by (5.20). Developing a SISO decoder to calculate (5.20) is another area for future work.

When signals are transmitted over fading channels interleaving is often used to reduce the correlation between symbols and to break up error bursts. Unfortunately, interleaving can introduce significant encoding and decoding delays. Therefore, it is desirable to minimize the size of the interleaver used. The size of interleaver required for symbols to be considered as uncorrelated for a given fade rate was determined in [116, 117]. Since block Turbo codes contain inherent interleaving this information can be used to choose the size of the block Turbo code used on a given channel.

6.3.2 Equalization

In this subsection ideas for future work on equalization are given. These ideas have not been implemented yet and some of the exact details are yet to be determined. Before the ideas are outlined a very brief literature review and some background information is required. For more information on equalization and a more extensive literature review see [48, 70, 108].

Background

Equalization is used to help recover signals transmitted over channels which suffer from *intersymbol interference (ISI)* such as the frequency selective Rayleigh fading channel. Optimum performance for coded systems can be obtained by performing combined equalization and error correction decoding. Since the channel can be modelled as a convolutional code, equalization can be performed using a trellis [70]. Therefore, both equalization and decoding can be performed using a combined trellis, which uses the constraints of the channel and the error correction code [118]. Unfortunately, the number of states required for equalization or error correction decoding alone can be prohibitive. Therefore, a combined trellis is often prohibitively complex and reduced complexity approaches such as [3, 4, 5, 38, 48, 49, 59, 60, 76, 78, 95, 96, 98, 103, 107, 111, 122, 123] are required.

One approach to combined equalization and decoding is to use the equalizer (or channel decoder/ detector) as a decoding stage in an iterative decoder [38, 59, 60, 78, 96, 98]. Interleaving may be used between the equalizer and error correction decoders to reduce correlation between the extrinsic information from different decoding stages [38]. Many of these approaches use trellis based *maximum a posteriori probability (MAP)* decoders and equalizers, which can be prohibitively complex. Therefore, reduced complexity equalizers [18, 20, 24, 25, 37, 39, 70, 91, 92, 108, 118] and/ or decoders [90] may be required. In [48, 49], a reduced complexity approach based on iterative decoding and decision feedback equalization [39, 48, 70, 91, 108] was developed. The structure of this *combined equalizer and decoder (CED)* is shown in Fig. 6.1. The first iteration of the CED is performed like a conventional *decision feedback equalizer (DFE)* followed by an error correction decoder. But on subsequent

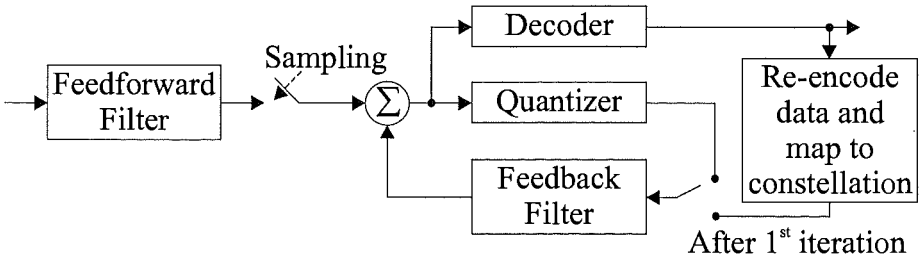


Figure 6.1: Combined equalizer and error correction decoder of [48].

iterations hard decisions from the error correction decoder are fed into the feedback filter.

Suggestions for Future Work on Combined Equalization and Decoding

Some ideas for future work on combined equalization and error correction decoding using iterative decoding will now be presented. These ideas are based on the concept used in [54, 90] (and chapters 3, 4 and 5) to perform reduced complexity error correction decoding.

The proposed CED uses DFEs and is based on [4, 5, 48, 49, 122]. On the first iteration of the CED the structure is the same as a traditional DFE followed by an error correction decoder as in [48, 49]. The structure after the first iteration of the CED is shown in Fig. 6.2. This can be considered as a bank of DFEs[8] with a common feedforward filter[9]. Hard and/ or soft information from the error correction decoder is used to choose a set of paths/ constellation point sequences through the ISI/ equalization trellis. Each path uses a separate feedback filter. But, the number of filters required may be reduced using techniques from [3, 122]. A hard sequence is fed into each feedback filter. However, better performance may be possible by using soft sequences [5, 48]. In [68] a sliding block algorithm is described, which may be useful in developing the proposed scheme.

The bank of DFEs produces a set of soft outputs, only one of which is sent to the error correction decoder (called the soft output for the most "reliable" path). There are a number of ways the most reliable path from the bank of DFEs could be chosen. If a quantizer is placed on the output of each DFE stage, then the path with the minimum difference between the absolute value of the quantizers input and its output could be chosen as the most reliable path. A simple code could be used to reduce the number of paths to choose from, as in [122]. This code could also be a component code in the error correction code.

The soft output from the DFE corresponding to the most reliable path/ constellation point sequence is fed to the error correction decoder. But performance could be improved if the equalizer also passes extrinsic information to the error cor-

[8] A bank of DFEs is also used in [122]. The proposed structure differs from [122] in the way the inputs to the filters are chosen and because it is used within an iterative decoder.

[9] A training sequence is used to initialize the filters, then they are adaptively updated as in [48].

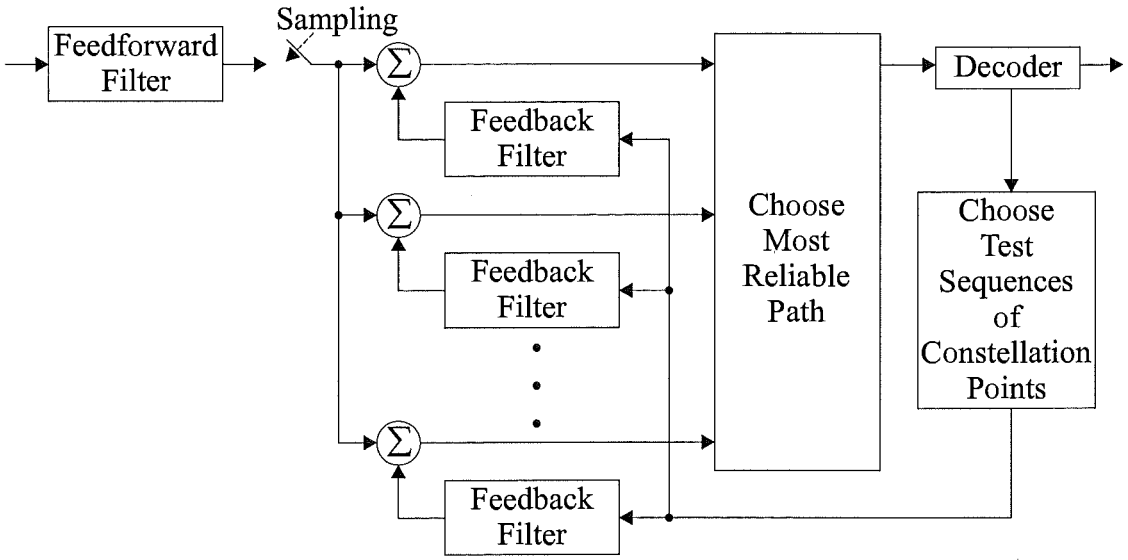


Figure 6.2: Proposed combined equalizer and error correction decoder after the first iteration.

rection decoder. Extrinsic information may be calculated using the set of DFE soft outputs in a similar way to the SISO algorithms for error correction decoding of chapters 3 and 4, or that of [90]. Consider the case of BPSK. The hard decision on the most reliable path is denoted $\boldsymbol{S}^D$ and a path with the other constellation point in the j^{th} received symbol position is denoted $\boldsymbol{S}^C$. The extrinsic information could be approximated by using these two sequences as follows

$$w_j(q) = \Lambda_j(q) - \lambda_j(q) = d_j \log\left(\frac{p(\boldsymbol{R}, \boldsymbol{\gamma}|\boldsymbol{S}^D)}{p(\boldsymbol{R}, \boldsymbol{\gamma}|\boldsymbol{S}^C)}\right) - \lambda_j(q) \tag{6.1}$$

where d_j is the bit labelling the j^{th} constellation point in $\boldsymbol{S}^D$, $\boldsymbol{R}$ is the received signal, $\Lambda_j(q)$ is the soft output from and $\lambda_j(q)$ is the soft input to the q^{th} decoding stage for the j^{th} symbol, and $\boldsymbol{\gamma}$ is the CSI. This concept would need to be extended to a non-binary approach for higher order modulations. Interleaving may be required between the equalizer and error correction decoder to reduce error propagation[10] and the correlation between the extrinsic information from the equalizer and the decoder [38].

Now consider the test sequences of constellation points fed into the bank of feedback filters. They could be chosen in a number of different ways including the following:

[10]DFEs are prone to producing error bursts.

1. The hard and/ or soft output from the error correction decoder is used to choose the test sequences. This could be done in a similar manner to the Chase decoder. First all test sequences are set equal to the sequence of constellation points chosen by the hard output from the error correction decoder. Then each test sequence uses a different combination of constellation points in the p_r LRPs[11]. The soft output or extrinsic information is used to find the p_r LRPs. In the case of a multilevel code the p_r LRPs are chosen using the reliability information[12] from all levels.

2. Use the set of codewords found by the SILO or trellis based[13] decoder to choose the test sequences of constellation points. In the case of a multilevel code all levels can produce a hard decision or a list of possible codewords. The codewords from all levels could be used to choose a set of test sequences of constellation points.

The CED of Fig. 6.2 uses a bank of DFEs. However, other equalizers could be used in the CED. In [44] a *maximum likelihood sequence estimation (MLSE)* predictor receiver is developed using the Cholesky decomposition[14]. The trellis used in this approach can be reduced to the set of paths/ constellation point sequences selected by the error correction decoder to provide a sub-optimal reduced-complexity approach. A DFE or a one path approach to the MLSE predictor receiver could be used on the first iteration. Tentative decisions from the error correction decoder could be used to update this one path equalizer. A possible structure for this approach is shown in Fig. 6.3.

6.3.3 Summary

This section has presented a range of ideas to further improve the performance and flexibility of the decoders presented in this thesis. Suggestions for further extensions to fading channels and channel state estimation were also given. Finally, ideas

[11]The value of p_r could be chosen using the reliability information.

[12]The soft output or extrinsic information could be used.

[13]If a trellis based error correction decoder is used then a given number of the most reliable paths through the trellis could be used.

[14]The impulse response of the channel would be assumed known if this option was used.

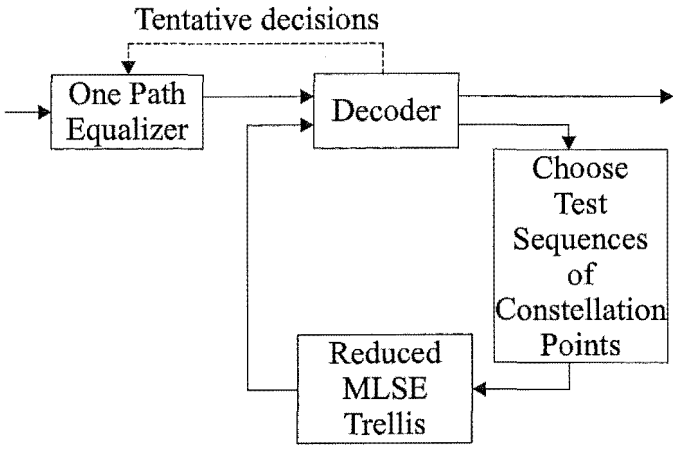


Figure 6.3: Alternative combined equalizer and error correction decoder.

for future work on reduced complexity combined equalization and decoding were presented.

Appendix A

Block Turbo Codes

A.1 Introduction

Since the introduction of Turbo codes in 1993 [15] there has been much interest in serial and parallel concatenated coding schemes [11, 12]. Both of these schemes can be shown to be closely related to block Turbo/ product codes [23].

This appendix considers the structural relationships between both serial and parallel concatenated codes, and block Turbo codes for systematic binary linear block component codes[1]. The properties of V-dimensional block Turbo codes are discussed in section A.2. Section A.3 considers V-stage *serial concatenated codes (SCCs)* and their relationship to block Turbo codes. Section A.4 considers V-stage *parallel concatenated codes (PCCs)* (or Turbo codes) and their relationship to block Turbo codes. Finally conclusions are given in section A.5.

A.2 Block Turbo Codes

A V-dimensional block Turbo code encodes a $k_1 \times k_2 \times \cdots \times k_V$ V-dimensional block of data. The first dimension is encoded using code $\mathbb{C}_1$. Then the data and parity from encoding the first dimension are encoded using code $\mathbb{C}_2$. The m^{th} dimension encodes the data and all parity bits from the previous encodings/ dimensions using code $\mathbb{C}_m$. This pattern continues in all subsequent dimensions. The structure of a

[1]Binary linear block component codes are assumed, however, convolutional and non-binary component codes could be used.

two-dimensional block Turbo code is shown in Fig. A.1 and can be extended to more dimensions if desired. Due to its structure, all rows of the code matrix are codewords of $\mathbb{C}_1$ and all columns are codewords of $\mathbb{C}_2$. The same is true in all V-dimensions. Therefore, any dimension can be encoded or decoded first.

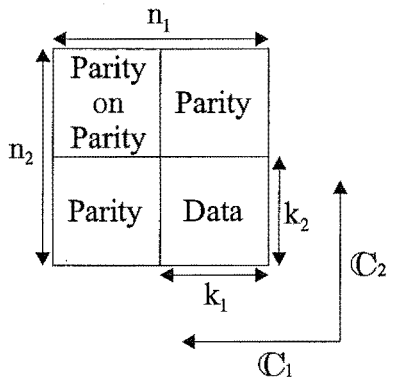


Figure A.1: Two-dimensional block Turbo code with component codes $\mathbb{C}_1$ and $\mathbb{C}_2$.

When the *parity on parity (PoP)* bits are transmitted a V-dimensional block Turbo code has rate

$$\mathcal{R} = \prod_{l=1}^{V} \mathcal{R}_l \tag{A.1}$$

where $\mathcal{R}_l$ is the code rate of component code $\mathbb{C}_l$. The minimum Hamming distance of the block Turbo code is given by

$$d_{H,min} = \prod_{l=1}^{V} d_{H,l} \tag{A.2}$$

where $d_{H,l}$ is the minimum Hamming distance of component code $\mathbb{C}_l$.

In some cases, the PoP bits are not transmitted. The code rate is then [43]

$$\mathcal{R} = \frac{1}{\frac{n_1}{k_1} + \frac{(n_2-k_2)}{k_2} + \frac{(n_3-k_3)}{k_3} + \cdots + \frac{(n_V-k_V)}{k_V}} \tag{A.3}$$

where n_l is the length of codewords in $\mathbb{C}_l$ and k_l is the number of information bits in a codeword. In this case the minimum Hamming distance of the block Turbo code (without PoP bits) is bounded by

$$d_{H,min} \leq \prod_{l=1}^{V} d_{H,l} \tag{A.4}$$

Block Turbo codes have inherent block interleaving between the encoders. In the two-dimensional case this is due to encoding rows and then columns (or columns and then rows). This 'interleaver' can be seen as a rows in columns out (or columns in rows out) block interleaver, $\pi_T = \pi_b$. Therefore, the structure of a 2-dimensional block Turbo code when PoP bits are transmitted is represented by Fig. A.2a. Figure A.2b illustrates the encoder for a block Turbo code when the PoP bits are not transmitted (or encoded).

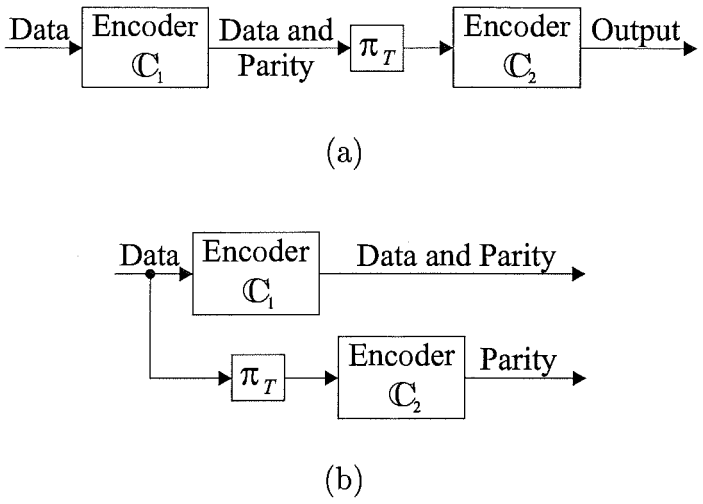


Figure A.2: a) Encoder structure of a two-dimensional block Turbo code when the PoP bits are transmitted and the encoder structure of a two-stage serial concatenated code. b) Encoder structure of a two-dimensional block Turbo code when the PoP bits are not transmitted and the encoder structure of a two-stage parallel concatenated code.

A further variation is to use a random interleaver, π_r, between encoding each dimension of the block Turbo code in addition to the inherent block interleaver, π_b, so the resulting interleaver is $\pi_T = \pi_b \pi_r$ [93, 94]. Then some uncorrectable error patterns may be broken up. The second code now provides parity bits on the randomly interleaved data and parity bits from the first encoding. When random interleaving is used and the PoP bits are transmitted the minimum Hamming distance of (A.2) is no longer guaranteed. In addition the output is now dependent on which code was encoded first. So the last code encoded is the first code decoded when the full block Turbo code matrix is transmitted. In the two-dimensional case all the rows and columns are now not guaranteed to be codewords. Unless otherwise stated only

the inherent block interleaving is used.

One of the advantages of having/ sending the PoP bits is that all rows and columns are codewords. This means that extrinsic information can be calculated for all transmitted bits (data, parity and PoP bits) by every decoder as shown in [90] and Fig. A.3a. The iterative decoding of block Turbo codes when the PoP bits are not transmitted is considered in [43] for both block and convolutional component codes. In this case all decoders can calculate extrinsic information for the data bits, but not for all the parity bits. The passing of soft information in the iterative decoder of a block Turbo code which transmits the PoP bits and uses a random interleaver is shown in Fig. A.3b. The iterative decoder for a block Turbo code when the PoP bits are not transmitted is shown in Fig. A.3c.

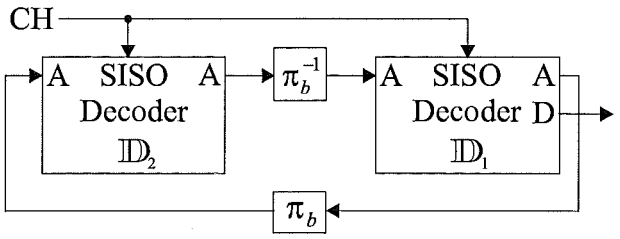
A.3 Serial Concatenated Codes

A SCC consists of a chain of $V \geq 2$ component encoders separated by interleavers. The encoder for a two-stage SCC is shown in Fig. A.2a. A block or random interleavers could be used. Each encoder adds redundancy to the output of the previous encoder. The rate of a V-stage SCC is given by (A.1).

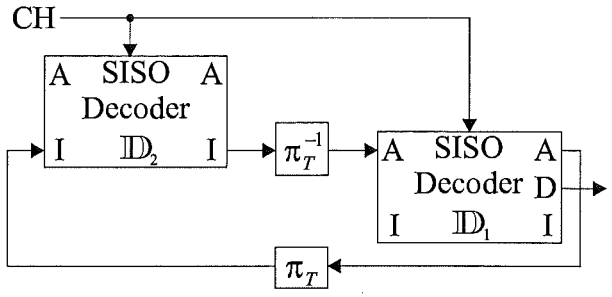
SCCs are decoded in the reverse order to which they are encoded. The iterative decoder for a SCC is shown in Fig. A.3b. Extrinsic information can only be calculated by every decoder for those bits common to all codes (the output from $\mathbb{C}_1$).

A block Turbo code with PoP bits transmitted is structurally equivalent to a SCC with a block interleaver between the component codes as they then produce the same codewords. This assumes the size of the interleaver is determined by the component codes of the block Turbo code. As expected the code rates are the same in the two cases. The fact that block Turbo codes are SCCs is stated in [90] and implied in [61] for block Turbo codes using only the inherent block interleaving.

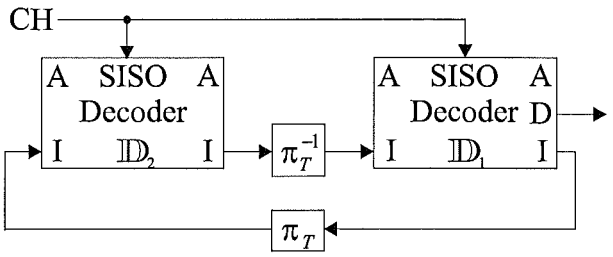
If a random interleaver is used between dimensions of the block Turbo code the resulting code is the same as a SCC with a random interleaver of equal length. In order to obtain the same codeword the random interleaver of the SCC would consist of the inherent block interleaver from the block Turbo code followed by the



(a)



(b)



(c)

Figure A.3: a) Iterative decoding structure for a two-dimensional block Turbo code with PoP bits and block interleaver π_b. b) Iterative decoding structure for a two-dimensional block Turbo code that transmits all encoded bits and has random interleaver $\pi_T = \pi_b\pi_r$. Iterative decoding structure for a two-stage SCC [12]. c) Iterative decoding structure for a two-dimensional block Turbo code when the PoP bits are not transmitted. Random interleaving may be used. Iterative decoding structure for a two-stage PCC [12]. *I* and *A* represent the extrinsic information for the input bits to the corresponding encoder and for all encoded bits from the corresponding encoder respectively. *CH* represents the soft information from the demodulator.

random interleaver used by the block Turbo code.

If only block interleaving is used, block Turbo codes have an advantage over

SCCs in their decoding. Any component code can be decoded first by a block Turbo code (when only the inherent block interleaving is used and the PoP bits are transmitted). In addition, iterative decoding of these block Turbo codes can obtain extrinsic information on all encoded bits from all decoders [90] as indicated in Fig. A.3a. When decoding SCCs, the component codes are decoded in reverse order to what they were encoded in and not all decoders can calculate extrinsic information for all encoded bits. This is shown in the iterative decoding structure for SCCs in Fig. A.3b. It is also the decoder of a block Turbo code with random interleaving when the full block Turbo code matrix is transmitted.

One advantage of SCCs is their flexibility in the choice of interleaver size. The interleaver size of a block Turbo code is determined by the size of the component codes within the block Turbo code. The interleaver has n_1 columns and k_2 rows (or n_2 rows and k_1 columns) in the two-dimensional case (see Fig. A.1). The interleaver length for a SCC is only limited to being a multiple of the output length of $\mathbb{C}_1$, n_1, and the input length of $\mathbb{C}_2$, k_2, for the two code case. In the V-stage/ V-dimensional case the interleaver size for the block Turbo code is restricted by the component codes and the SCC must have a V-dimensional input to produce the same codeword. This means a V-stage SCC can have a much shorter block length than a V-dimensional block Turbo code. A V-dimensional block Turbo code with random interleaving of the data bits is described in [93, 94].

A.4 Parallel Concatenated Codes

PCCs do not re-encode the parity bits from any of the encoders. A PCC encodes the data using the first code and then encodes interleaved versions of the data with the other encoders. Block or random interleavers can be used. The two code case is shown in Fig. A.2b. The data bits from only one encoder are transmitted, but the parity bits from all encoders are transmitted. Any component code can be encoded or decoded first.

The rate of a PCC is given by (A.3) when both data and parity bits from encoder $\mathbb{C}_1$ are transmitted and only parity bits from the other encoders are transmitted. Again iterative decoding can be used, with extrinsic information only being

calculated by every decoder for the data bits as shown in Fig. A.3c.

A block Turbo code which does not transmit PoP bits is structurally a PCC with V-dimensional block interleavers between the component codes. Again a random interleaver can be used between encodings of the block Turbo code and an equivalent random interleaver can be found for the PCC that takes the inherent block interleaving of the block Turbo code into account. In this case there is no decoding advantage for the block Turbo code compared to the PCC. Both have the iterative decoder shown in Fig. A.3c. The connection between block Turbo codes without PoP and PCCs using a block interleaver is stated in [11] and implied in [81]. This relationship can be seen by noting that the encoder for a block Turbo code (Fig. A.2c) is the same as that for a PCC. The rate of the PCC and the block Turbo code when no PoP bits are transmitted is also the same as expected.

One advantage of PCCs is their flexibility in the choice of interleaver size. The interleaver size of a block Turbo code when no PoP bits are transmitted is determined by the input lengths of its component codes. The interleaver has k_1 columns and k_2 rows in the two dimensional case (see Fig. A.1). The interleaver length for a PCC is only limited to being a multiple of the input length of $\mathbb{C}_1$, k_1, and the input length of $\mathbb{C}_2$, k_2, for the two code case. In the V-stage/ V-dimensional case the PCC must have a V-dimensional input to produce the same codeword.

A.5 Discussion and Conclusions

There is clearly a close structural connection between block Turbo codes and both SCCs and PCCs. A block Turbo code can be seen as a subclass of both of these concatenated coding structures, with different restrictions on interleaver size. However, given an appropriate interleaver size both SCCs and PCCs may be viewed as subclasses of block Turbo codes.

A block Turbo code that transmits the PoP bits and only uses an inherent block interleaver, π_b, has advantages over SCCs. The block Turbo code has the advantage that it can encode or decode any code first and all decoders can calculate extrinsic information for all encoded bits. But a SCC is a block Turbo code if an appropriately sized interleaver is used. Therefore, if an appropriately sized block

interleaver is used, then the SCC can encode or decode any code first, and all decoders can calculate extrinsic information for all encoded bits.

Iterative decoding can be used successfully on all these code structures [13, 15, 90]. Iterative decoding can also be used successfully on block Turbo codes with random interleavers as shown in [93, 94]. To allow the iterative decoding of block codes with a large trellis, reduced complexity algorithms need to be used. Reduced complexity decoding algorithms can be used with great success if soft information can be calculated as shown in [90], and chapters 3, 4 and 5 of this thesis. This allows more complex component codes to be used in the concatenated structures.

Appendix B

Binary Linear Block Codes

B.1 Introduction

Binary linear block codes are introduced in section 2.2 as a method of allowing received errors to be detected and/ or corrected. In this appendix some additional information on binary linear block codes is presented. The properties of perfect and quasi-perfect codes are considered in section B.2. Encoding and decoding a binary linear block code using generator matrices are considered in section B.3 and section B.4 respectively. Dual codes are defined in section B.4.

B.2 Perfect and Quasi-Perfect Codes

The codewords in a code, $\mathbb{C}$, of length n can be considered as points in an n-dimensional space. Each codeword has a decoding sphere centred on it. Any received binary vector in a decoding sphere will be decoded to the codeword at its center. Problems arise if the decoding spheres overlap as any received signal in more than one sphere could be decoded to more than one codeword. The properties of the code determine the radius of the decoding spheres, whether all received vectors are in a sphere and whether the spheres are disjoint. Now the decoding spheres of perfect and quasi-perfect codes will be discussed.

Consider decoding spheres of radius t around each of the 2^k codewords in $\mathbb{C}$, where t is the error correction capability of the code and k is the number of data bits in each codeword. In the case of a perfect code all possible binary vectors of

length n are contained in the decoding spheres [63]. Around each codeword there are [69]

$$\binom{n}{1} + \cdots + \binom{n}{t} \tag{B.1}$$

possible error sequences[1] (meaning all possible binary vectors of weight $\leq t$ and length n). Therefore, in each sphere of radius t there are

$$1 + \binom{n}{1} + \cdots + \binom{n}{t} \tag{B.2}$$

possible binary vectors of length n (including the centre codeword). The decoding spheres of a perfect code do not overlap and contain all 2^n possible binary vectors. There are 2^k possible codewords and therefore, 2^k spheres. Therefore, in the case of a perfect code the number of vectors in each sphere times the number of spheres equals the number of possible length n vectors [69], meaning

$$\left[1 + \binom{n}{1} + \cdots + \binom{n}{t}\right] 2^k = 2^n \tag{B.3}$$

This means that perfect codes can only correct error sequences with t or fewer errors [63], but they can correct all of these error sequences. Single error correcting binary BCH codes are perfect codes [69], while double error correcting binary BCH codes are quasi-perfect codes [63].

A quasi-perfect code can correct all error sequences with t or fewer errors and some sequences containing $t+1$ errors [63]. Spheres of radius $t+1$ around the codewords contain all possible vectors of length n, but the spheres may overlap [63]. This means that a received binary vector may be contained in more than one sphere, with a different error sequence for each sphere it is in. In this case the error sequences can only be used to detect errors, not correct them. This is why only some error sequences of weight $t+1$ can be corrected.

B.3 Encoder

A generator polynomial or generator matrix can be used to encode an information vector. This section summarizes the encoding of systematic linear block codes using a generator matrix, based on the description in [69]. If the information bits appear unaltered in the encoded word then the code is called systematic [69]. Non-systematic

[1]Error sequences are binary vectors of length n representing a given pattern of errors.

codes are usually avoided since they require additional encoding and decoding operations and an equivalent systematic code can always be found [69]. An equivalent code only differs in the order in which the bits appear in the codeword.

Define $\boldsymbol{g}(x)$ as the generator polynomial of a systematic linear block code. A systematic generator matrix can be created by encoding all possible information sequences containing a single one using $\boldsymbol{g}(x)$. Each of these encoded words is a row in the generator matrix. By ordering them appropriately the systematic generator matrix can be written as

$$\boldsymbol{G} = [\boldsymbol{P}\boldsymbol{I}_k] \tag{B.4}$$

where $\boldsymbol{P}$ is a $k \times (n-k)$ matrix of parity bits and $\boldsymbol{I}_k$ is the $k \times k$ identity matrix. The generator matrix has no redundant rows and so can not be reduced. It can create all 2^k possible codewords. A codeword, $\boldsymbol{C}$, is generated by multiplying the $1 \times k$ data/ information vector, $\boldsymbol{m}$, by the generator matrix, $\boldsymbol{G}$, to obtain

$$\boldsymbol{C} = \boldsymbol{m}\boldsymbol{G} \tag{B.5}$$

The encoder leaves the information bits unaltered at the end of the codeword and places the parity bits at the beginning of the codeword.

B.4 Decoder and Dual Codes

A *hard-input hard-output (HIHO)* decoder for binary systematic linear block codes is now discussed based on the discussion in [69]. First the parity check matrix, $\boldsymbol{H}$, of the code is defined as [69]

$$\boldsymbol{H} = [\boldsymbol{I}_{n-k}\boldsymbol{P}^T] \tag{B.6}$$

where the $(n-k) \times k$ matrix $\boldsymbol{P}^T$ is the transpose of the submatrix of parity bits $\boldsymbol{P}$ in the generator matrix and $\boldsymbol{I}_{n-k}$ is the $(n-k) \times (n-k)$ identity matrix. Since [69]

$$\boldsymbol{G}\boldsymbol{H}^T = [\boldsymbol{P}\boldsymbol{I}_k] \begin{bmatrix} \boldsymbol{I}_{n-k} \\ \boldsymbol{P} \end{bmatrix} = \boldsymbol{P} + \boldsymbol{P} = \boldsymbol{0} \tag{B.7}$$

and

$$\boldsymbol{C} = \boldsymbol{m}\boldsymbol{G} \tag{B.8}$$

the syndrome vector, $\boldsymbol{S}$, of a codeword is given by

$$\boldsymbol{S} = \boldsymbol{C}\boldsymbol{H}^T = \boldsymbol{m}\boldsymbol{G}\boldsymbol{H}^T = \boldsymbol{0} \tag{B.9}$$

If the hard input vector, $\boldsymbol{Y}$, has received errors, $\boldsymbol{e}$, then

$$\boldsymbol{S} = \boldsymbol{Y}\boldsymbol{H}^T = (\boldsymbol{C} + \boldsymbol{e})\boldsymbol{H}^T = \boldsymbol{0} + \boldsymbol{e}\boldsymbol{H}^T \tag{B.10}$$

and a non-zero syndrome is produced.

Another code can be generated and decoded using $\boldsymbol{H}$ and $\boldsymbol{G}$. The code generated by encoding $\boldsymbol{G}$ is denoted $\mathbb{C}_G$ and has k information bits and 2^k possible codewords. The dual code to $\mathbb{C}_G$ is defined as the code created by using the parity check matrix of $\mathbb{C}_G$, $\boldsymbol{H}$, as the generator matrix of the dual code, and the generator matrix of $\mathbb{C}_G$, $\boldsymbol{G}$, as the parity check matrix of the dual code [69]. If $\mathbb{C}_G$ is a cyclic code, then so is its dual. The dual code has $n-k$ information bits and 2^{n-k} possible codewords [69]. Therefore, an (n, k) linear block code has a $(n, n-k)$ dual code [69]. $\mathbb{C}_G$ can be decoded using its dual code [7, 43, 45]. If $n-k < k$, then decoding the dual code can be simpler [43].

Appendix C

Performance Bounds for Block Turbo Codes

C.1 Extended BCH Component Codes

In this appendix the bound on the probability of bit error for block Turbo/ product codes with extended BCH component codes of [112, 114] is discussed. This section will summarize the derivation given in [112, 114] for this union bound. Consider transmitting a linear binary block code $\mathbb{C}$ over the *additive white Gaussian noise (AWGN)* channel. The Gaussian noise has a mean value of zero and a two-sided power spectral density (passband) of $N_0/2$. The union bound can be written as

$$P(\boldsymbol{D} \neq \boldsymbol{C}|\boldsymbol{C}) \leq \sum_{\boldsymbol{D}\in\mathbb{C},\boldsymbol{C}\neq\boldsymbol{D}} P_e(\boldsymbol{C} \rightarrow \boldsymbol{D}) \tag{C.1}$$

where $\boldsymbol{C}$ is the transmitted codeword, $\boldsymbol{D}$ is the decoded word from a *maximum likelihood (ML)* decoder and the pairwise error probability is given by

$$P_e(\boldsymbol{C} \rightarrow \boldsymbol{D}) = Q\left(\sqrt{\frac{d_E^2(\boldsymbol{C},\boldsymbol{D})}{2N_0}}\right) \tag{C.2}$$

where $Q(\cdot)$ is the Q-function[1] and $d_E^2(\boldsymbol{C},\boldsymbol{D})$ is the squared Euclidean distance between the sequence of constellation points labelled by $\boldsymbol{C}$ and that labelled by $\boldsymbol{D}$.

[1]The Q-function is defined as [47]

$$Q(v) = \frac{1}{\sqrt{2\pi}}\int_v^\infty \exp\left(\frac{-x^2}{2}\right) = \frac{1}{2}\mathrm{erfc}\left(\frac{v}{\sqrt{2}}\right) \tag{C.3}$$

where erfc$(\cdot)$ is the complementary error function.

For BPSK $d_E^2(\boldsymbol{C},\boldsymbol{D}) = 4E_s d_H(\boldsymbol{C},\boldsymbol{D})$, where $d_H(\boldsymbol{C},\boldsymbol{D})$ is the Hamming distance between codewords $\boldsymbol{C}$ and $\boldsymbol{D}$. $E_s = \mathcal{R}E_b log_2(M)$ is the average energy used to transmit a symbol, $\mathcal{R}$ is the code rate, E_b is the average energy used to transmit a data bit and $log_2(M)$ is the number of encoded bits transmitted per symbol [48].

Since $\mathbb{C}$ is a linear code it is distance invariant[2] [114]. Now for any codeword in $\mathbb{C}$ (C.1) can be written as

$$P(\boldsymbol{D} \neq \boldsymbol{C}|\boldsymbol{C}) \leq \sum_{w=d_{H,min}(\mathbb{C})}^{n} A_w(\mathbb{C}) Q\left(\sqrt{\frac{2E_s w}{N_0}}\right) \tag{C.4}$$

where $d_{H,min}(\mathbb{C})$ is the minimum Hamming distance between two codewords in $\mathbb{C}$, $A_w(\mathbb{C})$ is the number of codewords of weight w in $\mathbb{C}$ and n is the length of a codeword.

If the decoded codeword is w bits from the transmitted codeword, then there will be w errors in the n decoded bits. Therefore, probability of bit error after ML decoding can be written as

$$P_e(\mathbb{C}) \leq \sum_{w=d_{H,min}(\mathbb{C})}^{n} \frac{w}{n} A_w(\mathbb{C}) Q\left(\sqrt{\frac{2E_s w}{N_0}}\right) \tag{C.5}$$

As w increases in value, $Q\left(\sqrt{\frac{2E_s w}{N_0}}\right)$ decreases in value exponentially. Therefore, the dominant components in (C.5) are those with small values of w. At low *signal to noise ratios (SNRs)* the number of codewords at higher weights affects performance, but at high SNRs (C.5) should provide a good bound on the *bit error rate (BER)*.

In order to calculate the probability of bit error of (C.5) the coefficients, $A_w(\mathbb{C})$, of the weight enumerator, $\boldsymbol{A}(x)$, need to be known. For a large code, for example the $(64, 57, 4)^2$ block Turbo code, the weight enumerator is time consuming to calculate due to there being 2^{3249} possible codewords. The component codewords in the block Turbo code are a lot shorter and have only 2^{57} possible codewords, therefore, it is easier to calculate their weight enumerators.

Denote the block Turbo code by $\mathbb{C}_p$, its row code by $\mathbb{C}_r$ (with length n_r and minimum Hamming distance $d_{H,r}$) and its column code by $\mathbb{C}_c$ (with length n_c and minimum Hamming distance $d_{H,c}$). In [113] it is shown that in general the entire weight enumerator of a block Turbo code is not just dependent on the weight

[2]Each codeword in a distance invariant code, $\mathbb{C}$, has $A_w(\mathbb{C})$ codewords at Hamming distance w from it, for all w.

enumerators of the component codes. However, for block Turbo codes with linear component codes the number of codewords with weight [112, 114]

$$w < d_{H,r}d_{H,c} + \max\left(d_{H,r}\lceil\frac{d_{H,c}}{q}\rceil, d_{H,c}\lceil\frac{d_{H,r}}{q}\rceil\right) \tag{C.6}$$

can be determined using the weight enumerators of the component codes. This achieves equality if $q = 2$ and both $d_{H,r}$ and $d_{H,c}$ are odd, where q is the size of the Galois field (q=2 since binary codes are considered). The coefficients of the block Turbo code's weight enumerator (or the number of codewords in $\mathbb{C}_p$ with weight[3] w) in this range of w are given by

$$A_w(\mathbb{C}_p) = \frac{1}{q-1}\sum_{i|w} A_i(\mathbb{C}_c)A_{w/i}(\mathbb{C}_r) \tag{C.7}$$

where the sum is over all divisors, i, of w, $A_i(\mathbb{C}_c)$ is the number of codewords in $\mathbb{C}_c$ of weight i and $A_{w/i}(\mathbb{C}_r)$ is the number of codewords in $\mathbb{C}_r$ of weight w/i [112].

The weight enumerator of an (n, k) single-error correcting[4] primitive binary BCH code is given by [120]

$$\boldsymbol{A}(x) = \frac{1}{n+1}\left[(1+x)^n + n(1-x)(1-x^2)^{(n-1)/2}\right] \tag{C.8}$$

The coefficient of the x^w term, $A_w(\mathbb{C})$, is the number of codewords of weight w in $\mathbb{C}$.

For a length n extended single error correcting BCH code (extended by an overall parity bit to ensure even parity) the weight enumerator is given by [69]

$$\boldsymbol{A}(x) = \frac{1}{2n}\left[(1+x)^n + (1-x)^n + 2(n-1)(1-x^2)^{n/2}\right] \tag{C.9}$$

The weight distributions are only known for some BCH codes, including all double and triple error correcting primitive binary BCH codes. The weight enumerator of a code can be determined using the weight enumerator of its dual code[5]. Tables of weight enumerators for the dual codes of these codes are given in [120]. If $\boldsymbol{A}(x)$ is the weight enumerator of a (n, k) code and $\boldsymbol{B}(x)$ is that of its $(n, n-k)$ dual code then [69, 120]

$$\boldsymbol{B}(x) = 2^{-k}(1+x)^n\boldsymbol{A}\left[\frac{1-x}{1+x}\right] \tag{C.10}$$

[3]The weight of a binary codeword is the number of ones it contains.

[4]n is the length of a codeword and k is the number of data bits in the codeword.

[5]Dual codes are defined in appendix B.

or [69]

$$\boldsymbol{A}(x) = 2^{k-n}(1+x)^n \boldsymbol{B}\left[\frac{1-x}{1+x}\right] = \sum_{i=0}^{n} A_i x^i \tag{C.11}$$

The weight enumerator of a length $n+1$ extended two or three error-correcting BCH code, $\boldsymbol{A}_e(x) = A_{e,0}+A_{e,1}x+\cdots+A_{e,n+1}x^{n+1}$, can be calculated using the weight enumerator of the length n unextended code,$\boldsymbol{A}_u(x) = A_{u,0} + A_{u,1}x + \cdots + A_{u,n}x^n$, using

$$\boldsymbol{A}_e(x) = \sum_{i=0}^{n} A_{u,i} x^{2\lceil i/2 \rceil} \tag{C.12}$$

meaning the extended code only has even weight codewords and so the weight of every odd weight unextended codeword is increased by one.

Appendix D

Glossary of Abbreviations

AHA = Advanced Hardware Architectures.

ASK = amplitude shift keying.

AWGN = additive white Gaussian noise.

BCH = Bose-Chaudhuri-Hocquenghem.

BCJR = Bahl-Cocke-Jelinek-Raviv.

BER(s) = bit error rate(s) (of data bits).

BPS = data bits per 16-QAM symbol.

BPSK = binary phase shift keying.

CED = combined equalizer and decoder.

CSI = channel state information.

dB = decibel.

DFE = decision feedback equalizer.

FER(s) = frame error rate(s) (of data bits).

HIHO = hard-input hard-output.

ISI = intersymbol interference.

LAN(s) = local area network(s).

LLR(s) = log-likelihood ratio(s).

LRP(s) = least reliable position(s).

MAP = maximum a posteriori probability.

ML = maximum likelihood.

MLSE = maximum likelihood sequence estimation.

MRPs = most reliable positions.

MSD(s) = multistage decoder(s).

PCC(s) = parallel concatenated code(s).

PoP = parity on parity bits.

PSK = phase shift keying.

QAM = quadrature amplitude modulation.

QPSK = quadrature phase shift keying.

RS = Reed Solomon.

SCC(s) = serial concatenated code(s).

SIHO = soft-input hard-output.

SILO = soft-input list-output.

SISO = soft-input soft-output.

SNR(s) = signal to noise ratio(s).

SOVA = soft output Viterbi algorithm.

VSAT = very small aperture terminal.

Appendix E

Glossary of Symbols

Parameters:

- a_I, a_Q = amplitudes of a QAM constellation point in the in-phase and quadrature dimensions respectively.
- $A_w(\mathbb{C})$ = the number of codewords of weight w in code $\mathbb{C}$.
- $\boldsymbol{A}(x)$ = weight enumerator.
- $\boldsymbol{\mathcal{A}}_s$ = syndrome matrix.
- B = bandwidth.
- $\boldsymbol{B}(x)$ = weight enumerator of dual code.
- $\boldsymbol{C} = (c_1, \cdots, c_n)$ = competing codeword.
- $\boldsymbol{C}^{-j} = (c_1, \cdots, c_{j-1}, c_{j+1}, \cdots, c_n)$ = competing codeword excluding the j^{th} position.
- $\{\boldsymbol{C}\}$ = set of possible competing codewords found by the list based decoder.
- $\{\boldsymbol{C}^a | e_j = +1\}$, $\{\boldsymbol{C}^b | e_j = -1\}$ = the sets of codewords, $\boldsymbol{C}^a$ and $\boldsymbol{C}^b$, with $e_j = +1$ and $e_j = -1$ respectively.
- $\{\boldsymbol{C}^a | e_j = +1\}_C$, $\{\boldsymbol{C}^b | e_j = -1\}_C$ = the subsets of codewords, $\boldsymbol{C}^a$ and $\boldsymbol{C}^b$, chosen by the Chase decoder with $e_j = +1$ and $e_j = -1$ respectively.
- $\mathcal{C}$ = capacity.
- $\mathbb{C}$ = block code.
- $\boldsymbol{c}(x) = (c_0 + c_1 x + \cdots + c_{n-1} x^{n-1})$ = codeword polynomial.
- $\boldsymbol{c}'(x)$ = polynomial for an extended BCH codeword.
- $\boldsymbol{D} = (d_1, \cdots, d_n)$ = decision codeword from the list based decoder.
- $\boldsymbol{D}^{-j} = (d_1, \cdots, d_{j-1}, d_{j+1}, \cdots, d_n)$ = decision codeword from the list based de-

coder excluding the j^{th} position.

- $\mathbb{D}$ = decoder for code $\mathbb{C}$.
- d_H = Hamming distance.
- $d_{H,i}$, $d_{H,min}$ = minimum Hamming distance of code $\mathbb{C}_i$, $\mathbb{C}$.
- d_E^2 = squared Euclidean distance.
- $\boldsymbol{E} = (e_1, \cdots, e_n)$ = transmitted codeword.
- $e_{mj}^{i\pm}$ = bit on level m labelling $s_j^{i\pm}$.
- $\mathcal{E}$ = Euler's constant.
- $\boldsymbol{e}$ = error vector.
- $\boldsymbol{e}(x)$ = polynomial representing the received errors.
- $\boldsymbol{E}_m^a$, $\boldsymbol{E}_m^b$ = the vectors of bits from level m used in the label that chooses $\boldsymbol{S}^a$ or $\boldsymbol{S}^b$ respectively.
- $\boldsymbol{E}_m^{ij\pm} = (e_{m1}^{ij\pm}, \cdots, e_{mn}^{ij\pm})$ = the vector of bits from level m in the label used to choose $\boldsymbol{S}^{ij\pm}$.
- E_b = energy used to transmit a data bit.
- E_s = symbol energy.
- f_c = carrier frequency.
- f_0 = coherence bandwidth.
- $g(t)$ = *additive white Gaussian noise (AWGN)* signal as a function of time.
- g_l = AWGN sample.
- $\boldsymbol{g}(x)$ = generator polynomial.
- $\boldsymbol{G}$ = generator matrix in appendix B, otherwise, it is the AWGN vector.
- $GF(q)$ = Galois field with q elements.
- $\boldsymbol{H}$ = parity check matrix.
- $h(t)$ = pulse shape.
- $\boldsymbol{I}_l$ = $l \times l$ identity matrix.
- j = imaginary component = $\sqrt{-1}$.
- k = information/ data length in codeword.
- l = error detection capability of the code.
- L = number of levels in the multilevel code.
- M = number of constellation points.
- $\boldsymbol{m}$ = data vector.

- $\boldsymbol{m}(x)$ = data polynomial.
- n = codeword length.
- N = total number of encoded bits transmitted per block/ frame.
- N_l = number of bits transmitted by level l per transmitted block/ frame.
- N_m = length of level m code block.
- N_0 = two-sided power spectral density of the additive white Gaussian noise (N_0 for baseband signals, $N_0/2$ for passband signals).
- N'_p = number of positions in the p' *least reliable positions (LRPs)*, where $y_l(q) \neq d_l$.
- N_s = maximum number of LRPs used by the distance approach.
- N'_s = number of LRPs used by the distance approach.
- p = number of LRPs considered by the Chase decoder.
- p' = number of LRPs for which the error patterns contain all possible binary combinations, $(x_1, \cdots, x_{p'})$.
- p_r = number of LRPs used to develop test sequences of constellation points for the proposed combined equalizer and decoder.
- P_{av} = average signal power.
- $\boldsymbol{P}$ = matrix of parity bits in the generator matrix.
- P_e = probability of bit error after maximum likelihood decoding.
- $P_e(\boldsymbol{C} \rightarrow \boldsymbol{D})$ = pairwise error probability.
- p_r = number of least reliable constellation points found in one of the proposed equalizers.
- q = decoding stage number.
- q_i = number of completed decoding stages on level i.
- Q_{max} = maximum number of decoding stages to be completed.
- $r(t)$ = received signal as a function of time.
- $\boldsymbol{R} = (r_1, \cdots, r_n)$ = soft received signal (with real or complex values).
- $\boldsymbol{R}^{-j} = (r_1, \cdots, r_{j-1}, r_{j+1}, \cdots, r_n)$ = soft received signal excluding the j^{th} position.
- $\mathcal{R}$ = code rate.
- $s(t)$ = transmitted signal as a function of time.
- s_l = constellation/ transmitted point.

- $\{s_j^a | e_{ij} = \pm 1\}$ = set of constellation points with the level i label $e_{ij} = \pm 1$.
- $\{\boldsymbol{S}^a | e_{ij} = +1\}$, $\{\boldsymbol{S}^b | e_{ij} = -1\}$ = sets of sequences of constellation points, $\boldsymbol{S}^a$ and $\boldsymbol{S}^b$, with the j^{th} bit in the level i label equal to $e_{ij} = +1$ or $e_{ij} = -1$ respectively.
- $\mathcal{S}_k = k^{th}$ syndrome.
- $\boldsymbol{\mathcal{S}}$ = syndrome vector.
- $s_j^{i\pm}$ = constellation point with the largest soft input metric and $e_{ij} = \pm 1$.
- $\boldsymbol{S}^{ij\pm} = (s_1^{ij\pm}, \cdots, s_n^{ij\pm})$ = sequence of constellation points with maximum metric and $e_{ij} = \pm 1$.
- $\boldsymbol{S}^{\boldsymbol{C}}$ = second most likely sequence of constellation points. It equals $\boldsymbol{S}^{ij+}$ or $\boldsymbol{S}^{ij-}$.
- $\boldsymbol{S}^{\boldsymbol{D}}$ = most likely sequence of constellation points. It equals $\boldsymbol{S}^{ij+}$ or $\boldsymbol{S}^{ij-}$.
- t = time index in chapter 1.
- t = error correcting capability of the code.
- $\{\boldsymbol{T}\}$ = set of test sequences.
- T_0 = coherence time.
- T_m = maximum excess delay.
- T_s = symbol period.
- v = number of errors received.
- V_i = number of component codes on level-i.
- w = weight of a codeword in section 4.5 and appendix C.
- $\{w_{mj}(q_m)\}_{m=1}^L$ = the set of extrinsic information from all levels for the j^{th} transmitted symbol.
- $\boldsymbol{W}(q) = (w_1(q), \cdots, w_n(q))$ = extrinsic information from the q^{th} decoder.
- $\boldsymbol{W}^{-j}(q) = (w_1(q), \cdots, w_{j-1}(q), w_{j+1}(q), \cdots, w_n(q))$ = extrinsic information from the q^{th} decoder excluding the j^{th} position.
- $\{\boldsymbol{W}_m(q_m)\}_{m=1}^L$ = the set of extrinsic information vectors from all levels.
- $weight(p)$ = the maximum weight of an error pattern.
- $\boldsymbol{x} = (x_1, \cdots, x_p)$ = the indices for the p LRPs in the soft input (smallest absolute values)
- X_i = the location of the i^{th} received error.
- $\boldsymbol{y}(x) = (y_0 + y_1 x + \cdots, y_{n-1} x^{n-1})$ = hard input polynomial (binary coefficients).
- $\boldsymbol{Y}(q) = (y_1(q), \cdots, y_n(q))$ = hard (binary) decision on the soft input vector to

the q^{th} decoder.

- $\boldsymbol{\alpha}$, $\alpha(q)$ = used to scale the extrinsic information.
- α^k = k^{th} root of the generator polynomial and a k^{th} power of a primitive element, α, of an extension field in section 2.2.
- $\boldsymbol{\beta}$, $\beta(q)$ = used to calculate the extrinsic information.
- $\hat{\beta}_j(q)$ = distance based estimate of $\beta_j(q)$.
- $\gamma(t)$ = multiplicative fading.
- $\gamma_A(t)$, γ_A = magnitude of multiplicative fading, with sample γ_i.
- $\boldsymbol{\gamma} = (\gamma_1, \cdots, \gamma_n)$ = channel state information vector.
- Γ = average signal to noise ratio.
- δ = intra subset Euclidean distance.
- $\theta(t)$ = phase of multiplicative fading variable.
- $\boldsymbol{\lambda}(q) = (\lambda_1(q), \cdots, \lambda_n(q))$ = soft input vector to q^{th} decoder.
- $\boldsymbol{\Lambda}(q) = (\Lambda_1(q), \cdots, \Lambda_n(q))$ = soft output vector from q^{th} decoder.
- $\{\xi\}$ = set of combinations of positions used by the distance based approach to estimating $\beta_j(q)$ and $\{\xi_i\}$ is the set for level-i in a multilevel code.
- π = 3.14 to 2 decimal places.
- π_r = random interleaver.
- π_T, π_T^{-1} = total interleaver and deinterleaver respectively.
- π_b, π_b^{-1} = block interleaver (rows in columns out or vice versa) and deinterleaver respectively.
- $\boldsymbol{\sigma}(x) = x^v + \sigma_1 x^{v-1} + \cdots + \sigma_v$ = error locator polynomial.
- $\chi(q)$ = used to scale the soft input to avoid decoder overflow problems.
- χ_i^T = the total scaling to the soft input to level i. This scaling is used to avoid decoder overflow problems.

Functions:

- $det(\cdot)$ = determinant.
- $\exp(\cdot)$ = exponential.

- $\min\{\cdot\}$ = minimum value.
- $\langle\cdot\rangle$ = average.
- $\lfloor x \rfloor$ = closest integer to x of equal or lesser value.
- $\oplus$ = modulo 2 addition.
- μ_x = mean of x.
- σ_x^2 = variance of x.
- $p(\cdot)$ = probability density function.
- $P(\cdot)$ = probability.
- $erfc(\cdot)$ = complementary error function.
- $Q(\cdot)$ = Q-function.
- $\log(\cdot)$ = natural logarithm, unless otherwise stated.
- $\log_2(\cdot)$ = base 2 logarithm.

Bibliography

[1] P. ADDE AND R. PYNDIAH, "Recent Simplifications and Improvements in Block Turbo Codes", 2^{nd} *International Symposium on Turbo Codes and Related Topics*, pp. 133–136, 2000.

[2] ADVANCED HARDWARE ARCHITECTURES (AHA) INC., AHA website http://www.aha.com.

[3] R. ANAND AND K. RAMCHANDRAN, "Application of Continuous Error Detection for Joint Equalization and Coding for ISI Channels", *ICC*, 2000.

[4] S. ARIYAVISITAKUL AND YE. (G.) LI, "Joint Coding and Decision Feedback Equalization for Broadband Wireless Channels", *VTC*, vol. 3, pp. 2256–2261, 1998.

[5] S. L. ARIYAVISITAKUL AND YE. (G.) LI, "Joint Coding and Decision Feedback Equalization for Broadband Wireless Channels", *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 9, pp. 1670–1678, Dec. 1998.

[6] L. R. BAHL, J. COCKE, F. JELINEK, AND J. RAVIV, "Optimal Decoding of Linear Codes for Minimising Symbol Error Rate", *IEEE Transactions on Information Theory*, pp. 284–287, Mar. 1974.

[7] G. BATTAIL, M. C. DECOUVELAERE, AND P. GODLEWSKI, "Replication Decoding", *IEEE Transactions on Information Theory*, vol. 25, no. 3, pp. 332–345, May 1979.

[8] S. BENEDETTO AND E. BIGLIERI, *Principles of Digital Transmission with Wireless Applications*, Kluwer Academic/ Plenum Publishers, 1999.

[9] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Self-Concatenated Codes with Self-Iterative Decoding for Power and Bandwidth Efficiency", *ISIT*, p. 177, 1998.

[10] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Self-Concatenated Trellis Coded Modulation with Self-Iterative Decoding", *Globecom*, pp. 585–591, 1998.

[11] S. Benedetto and G. Montorsi, "Unveiling Turbo Codes: Some Results on Parallel Concatenated Coding Schemes", *IEEE Transactions on Information Theory*, vol. 42, no. 2, pp. 409–428, Mar. 1996.

[12] S. Benedetto, G. Montorsi, D. Divsalar, and F. Pollara, "Serial Concatenation of Interleaved Codes: Performance Analysis, Design and Iterative Decoding", *TDA Progress Report*, vol. 42-126, Aug. 1996.

[13] S. Benedetto, G. Montorsi, D. Divsalar, and F. Pollara, "A Soft-Input, Soft-Output Maximum A Posteriori (MAP) Module to Decode Parallel and Serial Concatenated Codes", *TDA Progress Report*, vol. 42-127, pp. 1–20, Nov. 1996.

[14] C. Berrou and A. Glavieux, "Near Optimum Error Correcting Coding and Decoding: Turbo Codes", *IEEE Transactions on Communications*, vol. 44, no. 10, pp. 1261–1271, Oct. 1996.

[15] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon Limit Error Correcting Coding and Decoding: Turbo-Codes (1)", *ICC*, vol. 2 of 3, pp. 1064–1070, 1993.

[16] A. R. Calderbank, "Multilevel Codes and Multistage Decoding", *IEEE Transactions on Communications*, vol. 37, no. 3, pp. 222–229, Mar. 1989.

[17] A. R. Calderbank and N. Seshadri, "Multilevel Codes for Unequal Error Protection", *IEEE Transactions on Information Theory*, vol. 39, no. 4, pp. 1234–1248, 1993.

[18] A. M. Chan and G. W. Wornell, "A Class of Block-Iterative Equalizers For Intersymbol Interference Channels", *ICC*, 2000.

[19] D. CHASE, "A Class of Algorithms for Decoding Block Codes with Channel Measurement Information", *IEEE Transactions on Information Theory*, vol. 18, no. 1, pp. 170–182, Jan. 1972.

[20] PIERRE R. CHEVILLAT AND EVANGELOS ELEFTHERIOU, "Decoding of Trellis-Encoded Signals in the Presence of Intersymbol Interference and Noise", *IEEE Transactions on Communications*, vol. 37, no. 7, pp. 669–676, jul 1989.

[21] G. COLAVOLPE, G. FERRARI, AND R. RAHELI, "Extrinsic Information in Turbo Decoding: A Unified View", *Globecom*, pp. 505–509, 1999.

[22] D. DIVSALAR AND F. POLLARA, "Hybrid Concatenated Codes and Iterative Decoding", *TDA Progress Report (JPL)*, vol. 42-130, Aug. 15 1997.

[23] P. ELIAS, "Error Free Coding", *IRE Transactions on Inform. Theory*, vol. PGIT-4, pp. 29–37, Sept. 1954.

[24] M. V. EYUBOGLU AND S. U. H. QURESHI, "Reduced-State Sequence Estimation with Set Partitioning and Decision Feedback", *IEEE Transactions on Communications*, vol. 36, no. 1, pp. 13–20, Jan. 1988.

[25] M. V. EYUBOGLU AND S. U. H. QURESHI, "Reduced-State Sequence Estimation for Coded Modulation on Intersymbol Interference Channels", *IEEE Journal on Selected Areas of Communiations*, vol. 7, no. 6, pp. 989–995, Aug. 1989.

[26] J. FANG, F. BUDA, AND E. LEMOIS, "Turbo Product Code: A Well Suitable Solution To Wireless Packet Transmission For Very Low Error Rates", 2^{nd} *International Symposium on Turbo Codes and Related Topics*, pp. 101–111, 2000.

[27] K. FAZEL AND L. PAPKE, "Combined Multilevel Turbo-Code with 8PSK Modulation", *Globecom*, pp. 649–653, 1995.

[28] R. FISCHER, J. HUBER, AND U. WACHSMANN, "Multilevel Coding: Aspects from Information Theory", *Globecom*, pp. 26–30, 1996.

[29] G. D. FORNEY JR., *Concatenated Codes*, MIT Press, 1966.

[30] G. D. FORNEY JR., "Generalized Minimum Distance Decoding", *IEEE Transactions on Information Theory*, vol. 12, no. 2, pp. 125–131, Apr. 1966.

[31] G. D. FORNEY JR., "The Viterbi Algorithm", *IEEE Proceedings*, vol. 61, no. 3, pp. 268–278, Mar. 1973.

[32] G. D. FORNEY JR., "Codes on Graphs: News and Views", 2^{nd} *International Symposium on Turbo Codes and Related Topics*, pp. 9–16, 2000.

[33] M. P. C. FOSSORIER AND S. LIN, "Soft-Decision Decoding of Linear Block Codes Based on Ordered Statistics", *IEEE Transactions on Information Theory*, vol. 41, no. 5, pp. 1379–1396, Sept. 1995.

[34] M. P. C. FOSSORIER AND S. LIN, "Soft-Input Soft-Output Decoding of Linear Block Codes Based on Ordered Statistics", *Globecom*, 1998.

[35] S. FRAGIACOMO, C. MATRAKIDIS, AND J. O'REILLY, "Novel Near Maximum Likelihood Soft Decision Decoding Algorithm for Linear Block Codes", *IEE Proc. on Communications*, vol. 146, no. 5, pp. 265–270, Oct. 1999.

[36] D. GAZELLE AND J. SNYDERS, "Reliability-Based Code-Search Algorithms for Maximum-Likelihood Decoding of Block Codes", *IEEE Transactions on Information Theory*, vol. 43, no. 1, pp. 239–249, Jan. 1997.

[37] W. H. GERSTACKER, R. R. MULLER, AND J. B. HUBER, "Iterative Equalization with Adaptive Soft Feedback", *IEEE Transactions on Communications*, vol. 48, no. 9, pp. 1462–1466, Sept. 2000.

[38] M. J. GERTSMAN AND J. H. LODGE, "Symbol-by-Symbol MAP Demodulation of CPM and PSK Signals on Rayleigh Flat-Fading Channels", *IEEE Transactions on Communications*, vol. 45, no. 7, pp. 788–799, July 1997.

[39] R. D. GITLIN, J. F. HAYES, AND S. B. WEINSTEIN, *Data Communications Principles*, Plenum Press, 1992.

[40] C. M. HACKETT, "An Efficient Algorithm for Soft-Decision Decoding of the (24,12) Extended Golay Code", *IEEE Transactions on Communications*, vol. 29, no. 6, pp. 909–911, June 1981.

[41] J. HAGENAUER AND P. HOEHER, "A Viterbi Algorithm with Soft-Decision Outputs and its Applications", *Globecom*, vol. 3, pp. 1680–1686, 1989.

[42] JOACHIM HAGENAUER, ELKE OFFER, CYRIL MEASSON, AND MATTHIAS MOERZ, "Decoding and Equalization with Analog Non-Linear Networks", *European Transactions on Telecommunications*, 1999.

[43] J. HAGENAUER, E. OFFER, AND L. PAPKE, "Iterative Decoding of Binary Block and Convolutional Codes", *IEEE Transactions on Information Theory*, vol. 42, no. 2, pp. 429–445, Mar. 1996.

[44] B. HART, *MLSE Diversity Receiver Structures*, PhD thesis, University of Canterbury, New Zealand, 1996.

[45] C. P. HARTMANN AND L. D. RUDOLPH, "An Optimum Symbol-by-Symbol Decoding Rule for Linear Codes", *IEEE Transactions on Information Theory*, vol. 22, no. 5, pp. 514–517, Sept. 1976.

[46] A. A. HASSAN AND W. E. STARK, "On Decoding Concatenated Codes", *IEEE Transactions on Information Theory*, vol. 36, no. 3, pp. 677–683, May 1990.

[47] S. HAYKIN, *Communication Systems*, John Wiley & Sons, Inc., 3 edition, 1994.

[48] K. O. HOLDSWORTH, *Coding and Equalisation for Fixed-Access Wireless Systems*, PhD thesis, University of Canterbury, New Zealand, 2000.

[49] K. O. HOLDSWORTH, D. P. TAYLOR, AND R. T. PULLMAN, "On Combined Equalization and Decoding of Multilevel Coded Modulation", *ICC*, vol. 3, pp. 1687–1691, 2000.

[50] J. HUBER AND U. WACHSMANN, "Capacities of Equivalent Channels in Multilevel Coding Schemes", *Electronics Letters*, vol. 30, no. 7, pp. 557–558, 31 March 1994.

[51] A. W. HUNT, "Hyper-codes: High-performance Low-complexity Error-correcting Codes", Master's thesis, Carleton University, Canada, 1998.

[52] H. IMAI AND S. HIRAKAWA, "A New Multilevel Coding Method Using Error-Correcting Codes", *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 371–377, May 1977.

[53] M. ISAKA AND H. IMAI, "Design and Iterative Decoding of Multilevel Modulation Codes", 2^{nd} *International Symposium on Turbo Codes and Rel. Topics*, pp. 193–196, 2000.

[54] S. JACQ, *Decodage Iteratif des Codes Produits: 'Turbo-Codes en Bloc', et Evaluation de leurs Performances pour des Modulations MDP et MAQ sur Canal de Gauss et de Rayleigh*, PhD thesis, l'Universite de Limoges, France, 1996.

[55] T. KANEKO, T. NISHIJIMA, AND S. HIRASAWA, "An Improvement of Soft-Decision Maximum-Likelihood Decoding Algorithm Using Hard-Decision Bounded-Distance Decoding", *IEEE Transactions on Information Theory*, vol. 43, no. 4, pp. 1314–1319, July 1997.

[56] T. KANEKO, T. NISHIJIMA, H. INAZUMI, AND S. HIRASAWA, "An Efficient Maximum-Likelihood-Decoding Algorithm for Linear Block Codes with Algebraic Decoder", *IEEE Transactions on Information Theory*, vol. 40, no. 2, pp. 320–327, Mar. 1994.

[57] Y. KOFMAN, E. ZEHAVI, AND S. SHAMAI (SHITZ), "Performance Analysis of a Multilevel Coded Modulation System", *IEEE Transactions on Communications*, vol. 42, no. 2/3/4, pp. 299–312, Feb./ Mar./ Apr. 1994.

[58] W. C. Y. LEE, "Estimate of Channel Capacity in Rayleigh Fading Environment", *IEEE Transactions on Vehicular Technology*, vol. 39, no. 3, pp. 187–189, Aug. 1990.

[59] Y. LI AND B. CHEN, "Hybrid Equalization for Multipath Fading Channels with Intersymbol Interference", *VTC*, vol. 1, pp. 309–313, 1999.

[60] Y. LI AND W. H. MOW, "Iterative Decoding of Serially Concatenated Convolutional Codes Over Multipath Intersymbol-Interference Channels", *ICC*, vol. 2, pp. 947–951, 1999.

[61] J. LODGE, R. YOUNG, P. HOEHER, AND J. HAGENAUER, "Separable MAP 'Filters' for the Decoding of Product and Concatenated Codes", *ICC*, pp. 1740–1745, 1993.

[62] R. LUCAS, M. BOSSERT, AND M. BREITBACH, "On Iterative Soft-Decision Decoding of Linear Binary Block Codes and Product Codes", *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 2, pp. 276–296, Feb. 1998.

[63] F. J. MACWILLIAMS AND N. J. A. SLOANE, *The Theory of Error-Correcting Codes*, vol. 16, Elsevier Science BV, 1996.

[64] P. A. MARTIN AND D P. TAYLOR, "Distance Based Adaptive Scaling in Sub-Optimal Iterative Decoding", *Submitted to IEEE Transactions on Communications*, Aug. 2000.

[65] P. A. MARTIN AND D. P. TAYLOR, "On Adaptive Reduced-Complexity Iterative Decoding", *Globecom*, Nov.-Dec. 2000.

[66] P. A. MARTIN AND D. P. TAYLOR, "On Iterative Multistage Decoding", 2^{nd} *International Symposium on Turbo Codes and Related Topics*, 2000.

[67] P. A. MARTIN AND D. P. TAYLOR, "On Multilevel Codes and Iterative Multistage Decoding", *Accepted for publication in the IEEE Transactions on Communications*, Feb. 2001.

[68] D. W. MATOLAK AND S. G. WILSON, "Detection for a Statistically Known, Time-Varying Dispersive Channel", *IEEE Transactions on Communications*, vol. 44, no. 12, pp. 1673–1683, Dec. 1996.

[69] A. M. MICHELSON AND A. H. LEVESQUE, *Error-Control Techniques for Digital Communication*, John Wiley and Sons, 1985.

[70] A. F. MOLISCH, Ed., *Wideband Wireless Digital Communications*, Prentice Hall, 2001.

[71] K. R. NARAYANAN AND J. LI, "Bandwidth Efficient Low Density Parity Check Coding using Multi Level Coding and Iterative Multi Stage Decoding",

2^{nd} *International Symposium on Turbo Codes and Related Topics*, pp. 165–168, 2000.

[72] K. PAHLAVAN AND A. H. LEVESQUE, *Wireless Information Networks*, John-Wiley and Sons Inc., 1995.

[73] L. PAPKE AND K. FAZEL, "Different Decoding Algoriths for Combined Concatenated Coding and Multiresolutional Modulation", *ICC*, pp. 1249–1254, 1994.

[74] L. PAPKE AND K. FAZEL, "Combined Multilevel Turbo-Code with MR-Modulation", *ICC*, pp. 668–672, 1995.

[75] L. PAPKE, P. ROBERTSON, AND E. VILLEBRUN, "Improved Decoding with the SOVA in a Parallel Concatenated (Turbo-code) Scheme", *ICC*, pp. 102–106, 1996.

[76] B. PENTHER, D. CASTELAIN, AND H. KUBO, "A Modified Turbo-Detector for Long Delay Spread Channels", 2^{nd} *International Symposium on Turbo Codes and Related Topics*, pp. 295–298, 2000.

[77] A. PICART, *Concatenation de Codes et Decodage Iteratif Application des Turbo-Codes Produits aux Transmissions a Forte Efficacite Spectrale*, PhD thesis, L'Universite de Rennes, France, 1998.

[78] A. PICART, P. DIDIER, AND A. GLAVIEUX, "Turbo-Detection: A New Approach to Combat Channel Frequency Selectivity", *ICC*, vol. 3, pp. 1498–1502, 1997.

[79] A. PICART AND R. PYNDIAH, "Performance of Turbo-Decoded Product Codes used in Multilevel Coding", *ICC*, pp. 107–111, 1996.

[80] A. PICART AND R. PYNDIAH, "Adapted Iterative Decoding of Product Codes", *Globecom*, pp. 2357–2362, 1999.

[81] L. PING, S. CHAN, AND K. L. YEUNG, "Iterative Decoding of Multi-Dimensional Concatenated Single Parity Check Codes", *ICC*, pp. 131–135, 1998.

[82] G. J. POTTIE AND D. P. TAYLOR, "Multilevel Codes Based on Partitioning", *IEEE Transactions on Information Theory*, vol. 35, no. 1, pp. 87–98, Jan. 1989.

[83] J. G. PROAKIS, *Digital Communications*, McGraw-Hill, 1983.

[84] J. G. PROAKIS, *Digital Communications*, McGraw-Hill, 2nd edition, 1989.

[85] J. G. PROAKIS, *Digital Communications*, McGraw-Hill, 3rd edition, 1995.

[86] R. PYNDIAH, "Iterative Decoding of Product Codes: Block Turbo Codes", *International Sym. on Turbo Codes and Related Topics*, 1997.

[87] R. PYNDIAH, P. COMBELLES, AND P. ADDE, "A Very Low Complexity Block Turbo Decoder for Product Codes", *Globecom*, pp. 101–105, 1996.

[88] R. PYNDIAH, A. GLAVIEUX, A. PICART, AND S. JACQ, "Near Optimum Decoding of Product Codes", *Globecom*, pp. 339–343, 1994.

[89] R. PYNDIAH, A. PICART, AND A. GLAVIEUX, "Performance of Block Turbo Coded 16-QAM and 64-QAM Modulations", *Globecom*, pp. 1039–1043, 1995.

[90] R. M. PYNDIAH, "Near Optimum Decoding of Product Codes: Block Turbo Codes", *IEEE Transactions on Communications*, vol. 46, no. 8, pp. 1003–1010, Aug. 1998.

[91] S. U. H. QURESHI, "Adaptive Equalization", *Proceedings of the IEEE*, vol. 73, no. 9, pp. 1349–1385, Sept. 1985.

[92] R. RAHELI, A. POLYDOROS, AND C.-K. TZOU, "Per-Survivor Processing: A General Approach to MLSE in Uncertain Environments", *IEEE Transactions on Communications*, vol. 43, no. 2/3/4, pp. 354–364, Feb./ Mar./ Apr. 1995.

[93] D. RANKIN AND A. GULLIVER, "Randomly Interleaved SPC Product Codes", *ISIT*, p. 88, 2000.

[94] D. RANKIN AND T. A. GULLIVER, "Randomly Interleaved Single Parity Check Product Codes", *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pp. 420–423, 1999.

[95] D. RAPHAELI AND A. SAGUY, "Linear Equalizers for Turbo Equalization a New Optimization Criterion for Determining the Equalizer Taps", 2^{nd} *International Symposium on Turbo Codes and Related Topics*, pp. 371–374, 2000.

[96] D. RAPHAELI AND Y. ZARAI, "Combined Turbo Equalization and Turbo Decoding", *Globecom*, vol. 2, pp. 639–643, 1997.

[97] T. S. RAPPAPORT, *Wireless Communications - Principles and Practise*, Prentice Hall, 1996.

[98] M. C. REED AND C. B. SCHLEGEL, "An Iterative Receiver for the Partial Response Channel", *ISIT*, p. 63, 1998.

[99] M. RICE, "A Geometric Approach to Incomplete Soft Decision Block Decoding", *IEEE Transactions on Communications*, vol. 43, no. 2/3/4, pp. 1383–1391, 1995.

[100] C. E. SHANNON, "Communication in the Presence of Noise", *Proc. IRE*, vol. 37, no. 1, pp. 10–21, Jan. 1949.

[101] B. SKLAR, "A Primer on Turbo Code Concepts", *IEEE Communications Magazine*, pp. 94–102, Dec. 1997.

[102] B. SKLAR, "Rayleigh Fading Channels in Mobile Digital Communication Systems Part 1: Characterization", *IEEE Communications Magazine*, pp. 136–146, Sept. 1997.

[103] PAUL STRAUCH, CARLO LUSCHI, MAGNUS SANDELL, AND RAN YAN, "Turbo Equalization for an 8-PSK Modulation Scheme in a Mobile TDMA Communication System", *Vehicular Technology Conference*, vol. 3, pp. 1605–1609, 1999.

[104] G. L. STUBER, *Principles of Mobile Communication*, Kluwer Academic Publishers, 1996.

[105] P. SWEENEY AND S. WESEMEYER, "Iterative Soft-Decision Decoding of Linear Block Codes", *IEE Proceedings on Communications*, vol. 147, no. 3, pp. 133–136, June 2000.

[106] H. TANAKA AND K. KAKIGAHARA, "Simplified Correlation Decoding by Selecting Possible Codewords Using Erasure Information", *IEEE Transactions on Information Theory*, vol. 29, no. 5, pp. 743–748, Sept. 1983.

[107] S. TANTIKOVIT, A. U. H. SHEIKH, AND M. Z. WANG, "Code-Aided Adaptive Equaliser for Mobile Communication Systems", *Electronic Letters*, vol. 34, no. 17, pp. 1638–1640, 20 Aug. 1998.

[108] D. P. TAYLOR, G. M. VITETTA, B. D. HART, AND A. MAMMELA, "Wireless Channel Equalisation", *European Transactions on Telecommunications*, vol. 9, no. 2, Mar.-Apr. 1998.

[109] N. N. TENDOLKAR AND C. R. P. HARTMANN, "Generalization of Chase Algorithms for Soft Decision Decoding of Binary Linear Codes", *IEEE Transactions on Information Theory*, vol. 30, no. 5, pp. 714–721, Sept. 1984.

[110] W. H. THESLING AND F. XIONG, "Pragmatic Approach to Soft-Decision Decoding of Linear Block Codes", *IEE Proc. on Communications*, vol. 142, no. 1, pp. 40–47, Feb. 1995.

[111] M. TOGEL, W. PUSCH, AND H. WEINRICHTER, "Combined Serially Concatenated Codes and Turbo-Equalization", 2^{nd} *International Symposium on Turbo Codes and Related Topics*, pp. 375–378, 2000.

[112] L. TOLHUIZEN, S. BAGGEN, AND E. HEKSTRA-NOWACKA, "Union Bounds on the Performance of Product Codes", *ISIT*, p. 267, 1998.

[113] L. M. G. M. TOLHUIZEN AND C. P. M. J. BAGGEN, "On the Weight Enumerator of Product Codes", *Discrete Mathematics*, vol. 106/107, pp. 483–488, 1992.

[114] L. M. G. M. TOLHUIZEN AND C. P. M. J. BAGGEN, "Union Bounds on the Performance of Binary Product Codes", *Internal Report Philips Research (Nat.Lab. Report 7015)*, Apr. 1999.

[115] G. UNGERBOECK, "Channel Coding with Multilevel / Phase Signals", *IEEE Transactions on Information theory*, vol. 28, no. 1, pp. 55–67, 1982.

[116] R. VAN NOBELEN, *Coding for the Rayleigh Fading Channel*, PhD thesis, University of Canterbury, New Zealand, 1996.

[117] R. VAN NOBELEN AND D. P. TAYLOR, "Analysis of the Pairwise Error Probability of Noninterleaved Codes on the Rayleigh-Fading Channel", *IEEE Transactions on Communications*, vol. 44, no. 4, pp. 456–463, Apr. 1996.

[118] G. M. VITETTA AND D. P. TAYLOR, "Maximum Likelihood Decoding of Uncoded and Coded PSK Signal Sequences Transmitted over Rayleigh Flat-Fading Channels", *IEEE Transactions on Communications*, vol. 43, no. 11, pp. 2750–2758, Nov. 1995.

[119] U. WACHSMANN, R. F. H. FISCHER, AND J. B. HUBER, "Multilevel Codes: Theoretical Concepts and Practical Design Rules", *IEEE Transactions on Information Theory*, vol. 45, no. 5, pp. 1361–1391, July 1999.

[120] S. B. WICKER, *Error Control Systems for Digital Communication and Storage*, Prentice Hall, 1995.

[121] T. WOERZ AND J. HAGENAUER, "Iterative Decoding for Multilevel Codes using Reliability Information", *Globecom*, pp. 1779–1784, 1992.

[122] D. YELLIN, A. VARDY, AND O. AMRANI, "Joint Equalization and Coding for Intersymbol Interference Channels", *IEEE Transactions on Information Theory*, vol. 43, no. 2, pp. 409–425, Mar. 1997.

[123] K. ZHOU, J. G. PROAKIS, AND F. LING, "Decision-Feedback Equalization of Time-Dispersive Channels with Coded Modulation", *IEEE Transactions on Communications*, vol. 38, no. 1, pp. 18–24, Jan. 1990.