INFORMATION TO USERS

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

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

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

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

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

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

University of Alberta

New Stopping Criteria and Error Detection in Turbo Decoding

by

Fengqin Zhai 

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Master of Science

Department of Electrical and Computer Engineering

Edmonton, Alberta

Spring 2001

University of Alberta

Library Release Form

Name of Author: Fengqin Zhai

Title of Thesis: New Stopping Criteria and Error Detection for Turbo Decoding

Degree: Master of Science

Year this Degree Granted: 2001

Fg Zhai

Fengqin Zhai
Apt. 606, Vanier House, Michener Park
Edmonton, Alberta
T6H 4N1

Date: Jan. 10, 2001

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **New Stopping Criteria and Error Detection for Turbo Decoding** submitted by **Fengqin Zhai** in partial fulfillment of the requirements for the degree of **Master of Science.**

Dr. Ivan J. Fair

Dr. Witold A. Krzymien

Dr. Ioanis Nikolaidis

Date: January 8, 2001

ABSTRACT

This thesis introduces three new stopping criteria in conjunction with error detection techniques for turbo decoding. The approaches are based on monitoring the mean of the absolute values of the log-likelihood ratio of the decoded bits over a frame. From simulation, it is found that this mean value increases as the number of errors in a frame decreases. The simple mean estimate (ME) criterion has been established to determine if the iterative decoding process is to be stopped. In addition, the mean-sign-change (MSC) criterion and MSC-CRC criterion, in which MSC is concatenated with an external short cyclic redundancy check (CRC), are proposed to improve the performance of early stopping and error detection further. The proposed approaches offer good performance and can be easily implemented in a practical turbo decoder.

The thesis begins with an overview of error control coding, highlighting turbo codes, which mark one of the most important developments in coding theory. Details of turbo coding are then presented. Following an introduction to challenges with iterative decoding and discussions of several early stopping criteria, new approaches for combining early stopping and error detection are proposed and analyzed. These results show that the proposed schemes provide simple and efficient methods to stop the iterative decoding process without appreciably degrading performance and to check for errors in the decoded frames without introducing redundancy or very little redundancy.

ACKNOWLEDGMENTS

I would like to express my thanks to the following people and institutions for their contributions to this work:

- Dr. I. J. Fair, for supervising this project, providing financial support, offering an opportunity to work in the field of error control coding, and encouraging me to develop to my full potential;
- the members of the examining committee, for taking the time to review this work;
- TR*Labs*, the University of Alberta and Dalhousie University, for financial assistance;
- TR*Labs* and its staffs and students, for providing a favorable environment to carry out this research;
- my parents and parents-in-law, for their love, understanding and support throughout my graduate studies;

and, especially my husband, Yan Xin, for his love, encouragement, support and valuable technical discussion.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Error control codes are used in a wide variety of digital communication systems to increase the reliability of communication. In 1993, Berrou, Glavieux and Thitimajshima proposed a new class of error control codes called turbo codes [1] whose performance on the additive white Gaussian noise (AWGN) channel has been shown to perform within a few tenths of a dB of the capacity limit. The key principles of turbo codes are (i) encoding using parallel concatenated recursive systematic convolutional (RSC) constituent codes separated with an interleaver, and (ii) decoding using iterative maximum *a posteriori* (MAP) decoders employing soft information. Since turbo decoding proceeds in an iterative fashion, once iterations fail to improve the accuracy of decoding, the iterative process should be terminated by a stopping criterion. This will reduce decoding delay. Also, when turbo codes are used for data transmission, the decoded sequence must be examined to determine whether or not errors remain in the decoded frames.

This thesis introduces new stopping criteria in conjunction with error detection techniques for turbo decoding. The approaches are based on monitoring $M_{|L|}$, the mean of the absolute values of the component decoder log-likelihood ratio (LLR) over a frame. From simulation, it is found that $M_{|L|}$ increases as the number of errors in a frame decreases. By observing successive values of $M_{|L|}$, simple methods have been established to determine if all errors have been corrected or if errors remain in the frame.

1.1 Error Control Coding, Turbo Coding and Objective of This Thesis

Compared to analog communication systems, digital systems have many advantages [2]:

- Digital signals are easy to regenerate
- Digital circuits are less subject to interference
- Digital circuits are more reliable and can be produced at lower cost
- Signal processing functions are a natural fit with digital techniques

A communication system is a means of transporting information from a source to a sink. A system is "digital" if it uses a sequence of symbols from a finite alphabet to represent the information. The transmission of data in digital form allows for the use of a number of powerful

signal processing techniques, including error control coding. Figure 1.1 shows a block diagram of a basic digital communication system [3]. The upper blocks indicate the signal transformations from the source to the transmitter. The lower blocks indicate the signal transformations from the receiver to the sink; these essentially reverse the signal processing steps performed by the upper blocks. The data source may represent any of a number of sources of information. The modulator maps the information symbols onto signals that can be efficiently transmitted over the physical channel. The physical channel distorts the transmitted signal and introduces noise.

Between the data source and modulator, three distinct types of encoding procedures can be implemented. A source encoder performs analog-to-digital (A/D) conversion (for an analog source) and removes redundant or unneeded information. Encryption generates secrecy codes that prevent unauthorized users from understanding messages and from injecting false messages into the system. The channel encoder adds structured redundancy to the sequence that allows for the detection and/or correction of errors at the receiver. This type of coding, called error control coding, is the subject of this thesis.

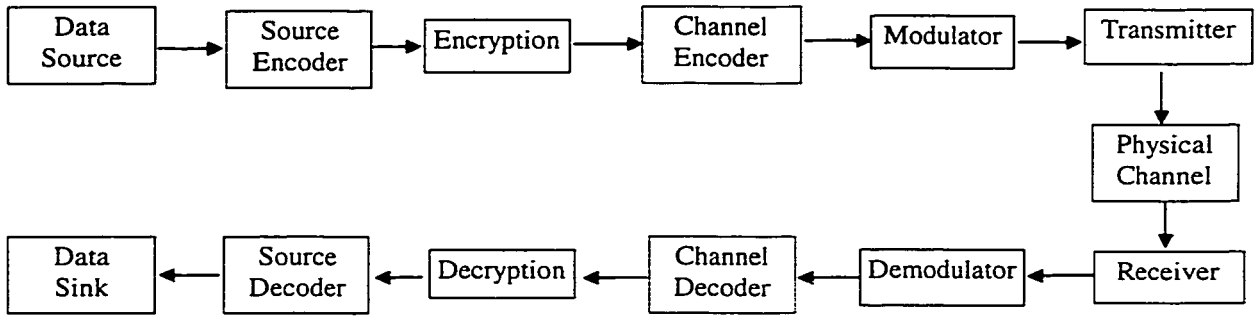


Figure 1.1: Basic digital communication system.

Using powerful error control coding is a very helpful approach to obtaining sufficient quality of transmission in digital communication systems. A fundamental theorem of information theory, originally proved by Claude Shannon, states that any channel's capacity is a function of the type of channel and the per-bit signal-to-noise ratio (SNR). According to Shannon's theorem, essentially error-free transmission can be achieved if, and only if, the information rate is less than the channel capacity. Shannon's theorem sets a lower bound on the amount of redundancy required for errorless communication, or a limit to system accuracy under other system constraints, but does not state how to approach this limit in practical communication systems. It does, however, suggest that data bits should be encoded as groups instead of as individual bits.

The goal of coding theory has always been to approach the Shannon limit with tolerable encoding and decoding complexity. The results achieved so far show that it is relatively easy to operate at signal-to-noise ratios of E_b / N_0 up to the cutoff rate limit, a few dB short of the capacity limit. The cutoff rate R_0 provides a simpler way to obtain the word error bound. For source words of length k_0 and code words of length n_0, the code rate is $R = k_0 / n_0$, and the word error probability is bounded from above by the quantity $2^{-n_0 E(R)}$ for R less than the channel capacity C. The function $E(R)$, which only depends on the code rate, is called the reliability function of the channel. $E(R)$ is a concave ($\cup$), decreasing, nonnegative function of R for $0 \le R \le C$. The cutoff rate R_0 is the rate at which the tangent to $E(R)$ of slope -1 intercepts the R axis [4], so that for $R \le R_0$, $E(R) \ge R_0 - R$, and hence word error probability $P_w(e) \le 2^{-n_0(R_0 - R)}$.

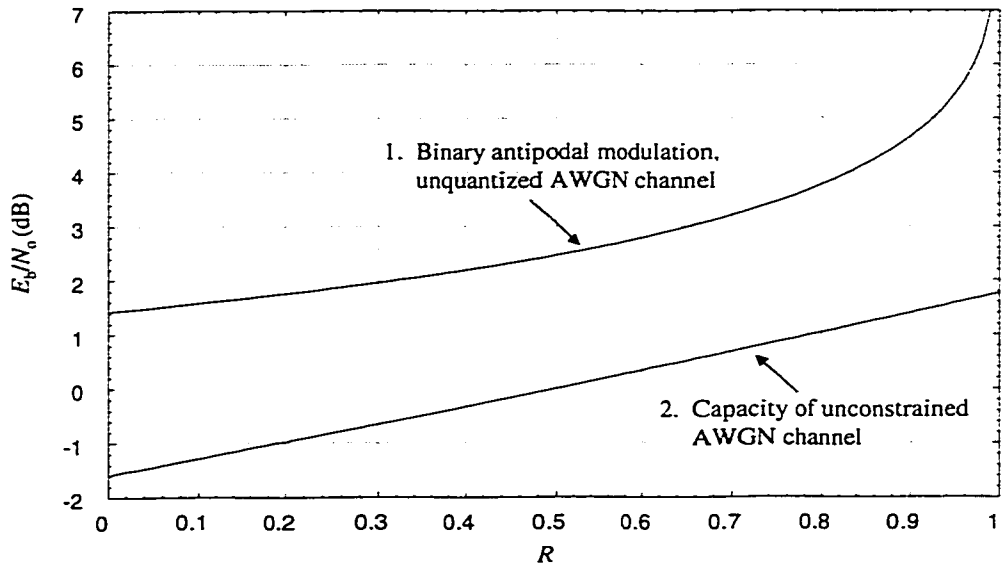


Figure 1.2: The required signal-to-noise ratio based on cutoff rate for binary antipodal modulation over AWGN channel (curve 1), and the minimum required signal-to-noise ratio vs. code rate for unconstrained AWGN channel (curve 2).

Figure 1.2 shows the gap between the cutoff rate limit and the channel capacity limit. The curves show, for a given code rate R, the possible value (curve 1, previously thought to be the practical limit) and the minimum possible value (curve 2, in theory) of E_b / N_0 for which the word error probability can still be driven to zero. For a rate 1/2 code and unquantized demodulation on a binary input additive white Gaussian noise (AWGN) channel, the cutoff rate limit is 2.5 dB, as opposed the capacity limit which is 0 dB. Generally the task becomes very complex to achieve reliable communication between these two values [5]. But Berrou, Glavieux,

and Thitimajshima developed a simple approach to design and implement a code to reach the region between the cutoff rate limit and capacity limit.

The most important development in coding theory since Ungerboeck introduced trellis codes in 1982 [6] has been the announcement of "turbo codes." Turbo codes can achieve very low error rates while operating at less than 1 dB above the capacity limit. In contrast, most current systems using conventional codes operate at 3 to 6 dB away from this bound. Uncoded systems are typically 10 dB or more away. The performance of turbo codes is so good that the initial reaction to the coding and decoding method was skepticism, but in 1994 Robertson [7] and in 1995 Divsalar and Pollara [8] were able to reproduce Berrou's results. The introduction of turbo codes has lead to a new way of constructing good codes and decoding them with low complexity, and has permanently changed the way that error control codes are looked at.

Concatenation is an attractive scheme to obtain high coding gains with moderate decoding complexity. Classically, concatenation is used for cascading a block code (the outer code, typically a Reed-Solomon code) and a convolutional code (the inner code) in a serial scheme. The novel encoder of turbo codes consists of the parallel concatenation of RSC codes with pseudo-random interleavers.

Turbo decoding proceeds in an iterative fashion. Generally, the iterative decoding process is automatically stopped when a specified maximum number of iterations is reached. However, simulations show that different frames need a different number of iterations to converge under the same channel conditions. Once the iterative process has converged, additional iterations are not helpful and iteration should be stopped. This "early stopping" will reduce the average decoding time. In addition, in data transmission systems, it is common to check for errors after decoding. In practice, this could be implemented by using a powerful cyclic redundancy check (CRC) to determine whether a frame of data has been completely corrected. But the using a powerful CRC adds additional redundancy and reduces the code rate. The objective of this thesis is to design new and efficient techniques to combine early stopping and error detection without introducing redundancy or introducing less redundancy than standard CRC techniques.

1.2 Organization of Thesis

This thesis is structured as follows. Further details of turbo coding are presented in Chapter 2. After introducing the BCJR algorithm that is the basis for turbo decoding, this chapter describes modifications to this algorithm. These include the modifications that are necessary to decode RSC

codes, Berrou's turbo coding scheme, and the improved decoding structure first described by Robertson. The concepts of weight and distance of codes, which are relevant to the performance of codes and interleaver design, are introduced. Interleavers and termination of trellises, which are important issues in the design of turbo codes, are also discussed in this chapter. Finally a simulation platform for turbo coding is described.

Following an introduction to challenges with iterative decoding, several early stopping criteria are overviewed in Chapter 3. In Chapter 4, new approaches for combining early stopping and error detection are proposed and analyzed. The performance of the new approaches is reported in Chapter 5. The bit error rate (BER), average number of iterations, frame error rate (FER), missed detection rate (MDR) and false alarm rate (FAR) are compared with corresponding values for other approaches. These results show that the proposed schemes provide simple and efficient methods to stop the iterative decoding process without appreciably degrading performance and to check for errors without introducing redundancy or introducing a minimal amount of redundancy. Chapter 6 concludes the thesis by summarizing the concepts, algorithms and performance of the new approaches of early stopping and error detection for turbo decoding and providing suggestions for further research.

1.3 Notation

The following symbol representations and measures of performance are used throughout this thesis. Other notation is introduced where required.

R:	Code rate; $R = k_0 / n_0$. In general, a convolutional encoder with k_0 inputs and n_0 outputs is said to have code rate k_0 / n_0.
K:	Constraint length of convolutional codes.
ν:	Memory of convolutional encoder; $\nu = K - 1$.
N_1:	Frame length of source sequence corresponding to convolutional codes (including RSC codes) with termination; frame length for turbo codes with single termination.
N_2:	Frame length of source sequence corresponding to joint termination in turbo codes.
N:	Frame length of encoded sequence after inserting terminating bits in source sequence. $N = N_1 + \nu$ or $N = N_2 + 2\nu$ depending on the terminating scheme.

$\overline{u}$: The source information frame; $\overline{u} = (u_1, u_2, \cdots, u_k, \cdots)$.

M: The total number of encoder states. $M = 2^{k_0 \nu}$.

m: Encoder states, where $m = 0, 1, ..., M-1$ in decimal.

$G(g_1, g_2)$: Generators for a rate 1/2 convolutional encoder, where g_1 and g_2 denotes two feedforward connection patterns associated with the generation of parity bits. Generally g_1 and g_2 are octal numbers (e.g. $G(7,5)$). For an RSC encoder, g_1 denotes the feedback generator and g_2 denotes the feed-forward generator.

$G(g_1, g_2, g_c)$: g_1 and g_2 have the same meaning as those of $G(g_1, g_2)$; g_c is the generator for a cyclic redundancy code (CRC) check.

E_b / N_0: Ratio of bit energy to noise power spectral density.

E_s / N_0: Ratio of channel symbol energy to noise power spectral density. $E_s / N_0 = R E_b / N_0$.

CHAPTER 2

TURBO CODING

Turbo codes are a new class of error control codes, originally based on the use of two RSC codes in parallel [1]. The novelty of turbo codes was the use of RSC codes, a random interleaver and iterative decoding using soft information. In 1996, Divsalar and Pollara developed multiple turbo codes, which consist of more than two RSC codes and more than one interleaver [11]. Because turbo codes are parallel concatenated convolutional codes, they are also called PCCCs. Using the same functional blocks, namely convolutional encoders and interleavers, serially concatenated convolutional codes (SCCCs) were proposed in 1996 [12]. In this thesis, only PCCCs with two component RSC codes and an interleaver are considered.

Elias first introduced convolutional codes in 1955 [13]. In this thesis, the term "convolutional codes" is used to denote conventional non-recursive convolutional codes, not including RSC codes. Since 1955, several decoding algorithms have been developed for convolutional codes; four algorithms are highlighted here. The earliest decoding technique for convolutional codes was the sequential decoding algorithm, originally proposed by Wozencraft in 1957 [14], and subsequently modified by Fano[15] and Jelinek [16, 17]. This algorithm provides fast but suboptimal performance. In 1963 Massey [18] presented the threshold decoding algorithm for both block and convolutional codes, which is generally less powerful than the sequential algorithm for convolutional codes, although it is optimal for several classes of block codes.

In 1967, a third decoding approach, the Viterbi decoding algorithm, was proposed and analyzed by Viterbi [19]. The Viterbi algorithm (VA) essentially performs maximum likelihood sequence decoding and limits decoding computational complexity by taking advantage of the special structure of the code trellis diagram. The VA has achieved widespread use, and is still the technique used in most practical systems.

The VA is an optimal decoding approach that minimizes probability of sequence error for convolutional codes. But the VA is not able to directly evaluate the *a posteriori* probability (APP) for each decoded bit. The APP is used to make the optimum decision in a digital communication system. For a digital communication system in which the events X_i, $i = 0,1,...,n$, represent the possible transmitted messages in a given time interval, $\Pr(X_i)$ represents their *a priori* probabilities (which consist of transmitted message probabilities), and Y represents the received

signal (which consists of transmitted message corrupted by noise), $\Pr(X_i|Y)$ is called the *a posteriori* probability of X_i conditioned on having observed the received signal Y. An algorithm, which generates the APP information, was proposed by Bahl, Cocke, Jelinek and Raviv in 1974 and is now called the BCJR algorithm [20]. This algorithm minimizes the bit error rate (BER) in decoding linear block and convolutional codes and produces the APP for each decoded bit. Since it minimizes BER, the BCJR algorithm theoretically should result in better performance than the VA algorithm. The BCJR algorithm implements a MAP decoding which is optimum, while the VA algorithm performs maximum likelihood (ML) decoding. Generally, there is no knowledge available about the a-priori probabilities. The a-priori probabilities are usually assumed to be equal. The ML decoding and the MAP decoding are equivalent with this assumption [2, p.739]. Thus, in most applications of interest, the performance of the BCJR algorithm and the VA algorithm for decoding convolutional codes is effectively identical. As a result of its higher complexity, the BCJR algorithm did not attract much attention until the invention of turbo codes by Berrou *et al.* in 1993.

In this chapter, several issues are discussed that are relevant to turbo coding, including the encoder, encoder trellis termination, interleavers, estimation of symbol energy and noise variance, the decoder and decoding algorithms. The simulation platform used for evaluating the performance of the whole system is also described. This chapter commences with a detailed description of the BCJR algorithm for decoding convolutional codes.

2.1 BCJR Algorithm

The BCJR algorithm performs two basic recursions on the sequence of received channel outputs. From the results of these recursions, it computes the transitions in the code trellis and the probability of the encoded data. It can be used to decode convolutional codes and block codes. A block encoder is equivalent to a time-varying Markov source whereas a convolutional encoder can be modelled as a stationary Markov source. In subsection 2.1.1, the general BCJR algorithm is introduced, and then in subsection 2.1.2 the BCJR algorithm is applied to convolutional codes.

2.1.1 General BCJR Algorithm

The notation in Figure 2.1 will be used to describe a transmission system that uses channel coding to correct errors, in particular, a system that uses the BCJR decoding algorithm.

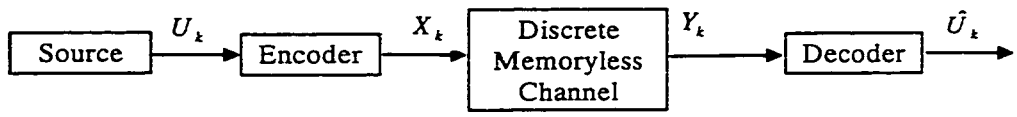


Figure 2.1: Schematic diagram of transmission system.

A. Encoding

The source generates a random binary bit sequence $\{U_k\}$ (the index k will always refer to time throughout this thesis). The encoder is assumed to be a discrete-time finite-state Markov process (Markov chain). The encoder has M distinct states and generates the encoded code sequence $\{X_k\}$ where the symbols are drawn from a finite discrete alphabet X. The encoder divides the continuous input sequence into frames of size N. Figure 2.2 shows the state transition from state m' to m with probability $q_k(X \mid m', m)$ and output X. The following notation will be used when describing the encoder:

S_k : State of the encoder at time k

X_k : Output of the encoder at time k

$S_k^{k'}$: Sequence of states from time k to k', where $S_k^{k'} = (S_k, S_{k+1}, ..., S_{k'})$

$X_k^{k'}$: Output sequence corresponding to $S_k^{k'}$, where $X_k^{k'} = (X_k, X_{k-1}, ...X_{k'})$

$\Pr(\cdot \mid \cdot)$: Conditional probability

$p_k(m \mid m')$: The state transition probability of the encoder, where

$$p_k(m \mid m') = \Pr\{S_k = m \mid S_{k-1} = m'\}.$$

$q_k(X \mid m', m)$: Output probability, where $q_k(X \mid m', m) = \Pr\{X_k = X \mid S_{k-1} = m'; S_k = m\}$.

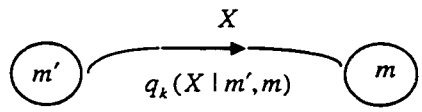


Figure 2.2: State transition diagram.

B. Channel

Assume the encoder starts in the initial state $S_0 = 0$, and produces an output sequence X_1^N ending in the terminal state $S_N = 0$. Let the sequence $X_1^N = (X_1, X_2, ..., X_N)$ be the input to a discrete memoryless channel (DMC) whose output is the sequence $Y_1^N = (Y_1, Y_2, ..., Y_N)$. Define the

transition probabilities of the DMC as $R(Y_j \mid X_j) = \Pr\{Y_j \mid X_j\}$ where $1 \le j \le N$, so that for all $1 \le k \le N$ the conditional probability of the received sequence is:

$$\Pr\{Y_1^k \mid X_1^k\} = \prod_{j=1}^{k} R(Y_j \mid X_j). \tag{2.1}$$

C. Decoding

The following probability formulas and Markov properties are used frequently in the derivation of the BCJR algorithm.

- Conditional probability:

$$\Pr(A \mid B) = \Pr(AB)/\Pr(B) \quad \text{or} \quad \Pr(AB) = \Pr(A \mid B)\Pr(B), \tag{2.2}$$

$$\Pr(AB \mid C) = \Pr(A \mid BC)\Pr(B \mid C). \tag{2.3}$$

- Introduction of a new random variable S:

$$\Pr\{A\} = \sum_{m=1}^{M} \Pr\{A; S = m\}. \tag{2.4}$$

- Markov chain [21, p. 137]:

  Consider a stochastic process $\{S_k, k = 0,1,2,...\}$ that takes on finite number of possible values. If $S_k = m$, then the process is said to be in state m at time k. Suppose that whenever the process is in state m', there is a fixed probability $p_k(m \mid m')$ that it will next be in state m. That is, suppose that:

$$\Pr\{S_k = m \mid S_{k-1} = m', ..., S_1 = m_1, S_0 = m_0\} = \Pr\{S_k = m \mid S_{k-1} = m'\} = p_k(m \mid m') \tag{2.5}$$

  for all states $m_0, m_1, ..., m', m$ and all $k \ge 0$. Such a stochastic process is known as Markov chain. The distinctive property of a Markov chain is that the conditional distribution of any future state S_k, given the past states $S_0, S_1, ..., S_{k-2}$ and the present state S_{k-1}, is independent of the past states and depends only on the present state S_{k-1}.

- The output of the Markov chain, or the combination of output and process state, is called an event. A Markov property is that if S_{k-1} is known, the events after time $k-1$ do not depend on the output of the chain from times up to and including time $k-1$.

To decode the received symbol sequence, the decoder examines a frame of received data Y_1^N, and estimates the *a posteriori* probabilities (APP) of the encoder states, $\Pr\{S_k = m \mid Y_1^N\}$, given the frame of observed channel outputs:

$$\Pr\{S_k = m | Y_1^N\} = \Pr\{S_k = m; Y_1^N\} / \Pr\{Y_1^N\}, \quad k = 1,2,...,N. \tag{2.6}$$

The decoder is trying to maximize $\Pr\{S_k = m | Y_1^N\}$. In (2.6), since $\Pr\{Y_1^N\}$ is easily obtained and fixed given the received data Y_1^N (see relation (2.17)), the goal becomes to maximize $\Pr\{S_k = m; Y_1^N\}$. In the following, this joint probability is derived first:

$$\lambda_k(m) = \Pr\{S_k = m; Y_1^N\}, \quad k = 1,2,...,N. \tag{2.7}$$

To assist in the calculation of $\lambda_k(m)$, the following probability functions are defined:

$$\alpha'_k(m) = \Pr\{S_k = m; Y_1^k\}, \quad k = 1,2,...,N, \tag{2.8}$$

$$\beta'_k(m) = \Pr\{Y_{k+1}^N \mid S_k = m\}, \quad k = 1,2,...,N-1, \tag{2.9}$$

$$\gamma_k(m',m) = \Pr\{Y_k; S_k = m \mid S_{k-1} = m'\}, \quad k = 1,2,...,N. \tag{2.10}$$

As shown in relation (A1.1) of Appendix 1, the values $\gamma_k(m',m)$ are dependent on state transition probabilities $p_k(m \mid m')$, probabilities $q_k(X \mid m',m)$ that indicate if paths between states m' and m exist, and the block transition probabilities $R(Y_k \mid X)$ from the channel.

$\lambda_k(m)$ can be obtained by the formula $\Pr\{AB\} = \Pr\{A \mid B\}\Pr\{B\}$ and the Markov property that if S_k is known, events after time k are not dependent on Y_1^k.

$$\begin{aligned}
\lambda_k(m) &= \Pr\{S_k = m; Y_1^N\} \\
&= \Pr\{(S_k = m; Y_1^k); Y_{k+1}^N\} \\
&= \Pr\{S_k = m; Y_1^k\} \cdot \Pr\{Y_{k+1}^N \mid (S_k = m; Y_1^k)\} \\
&= \Pr\{S_k = m; Y_1^k\} \cdot \Pr\{Y_{k+1}^N \mid S_k = m\} \\
&= \alpha'_k(m) \cdot \beta'_k(m), \quad k = 1,2,...,N.
\end{aligned} \tag{2.11}$$

Probabilities $\alpha'_k(m)$ and $\beta'_k(m)$ can be recursively calculated from the probability $\gamma_k(m',m)$. For these calculations, relevant expressions are given below. For the detailed derivation, please refer to Appendix 1.

$$\alpha'_k = \sum_{m'=0}^{M-1} \alpha'_{k-1}(m') \cdot \gamma_k(m',m), \quad k = 1,2,...,N, \tag{2.12}$$

$$\beta'_k(m) = \sum_{m'=0}^{M-1} \beta'_{k+1}(m') \cdot \gamma_{k+1}(m,m'), \quad k = N-1,...,2,1, \tag{2.13}$$

where the boundary conditions are:

$$\alpha'_0(0) = 1, \text{ and } \alpha'_0(m) = 0, \text{ for all } m \neq 0,$$

$$\beta'_N(0) = 1, \text{ and } \beta'_N(m) = 0, \text{ for all } m \neq 0. \tag{2.14}$$

These boundary conditions are derived from the assumption that the process starts and ends in the zero state.

Given these expressions, the general BCJR decoding algorithm can be mechanized in the following three-step procedure:

(1) Initialize $\alpha'_0(m)$ and $\beta'_N(m)$, $m = 0,1,...,M-1$.

(2) As soon as Y_k is received, compute $\gamma_k(m',m)$ and $\alpha'_k(m)$. Store the $\alpha'_k(m)$ values for all k and m for subsequent calculation of $\lambda_k(m)$.

(3) After the complete sequence Y_1^N has been received, calculate $\beta'_k(m)$ in reverse order starting from time N. Once $\beta'_k(m)$ has been computed, evaluate $\lambda_k(m)$. Given values of $\lambda_k(m)$, decode the most probable state $S_k = m$ by choosing the largest $\lambda_k(m)$ value, $m = 0,1,...,M-1$.

2.1.2 BCJR Algorithm for Convolutional Codes

In this subsection, the application of the BCJR algorithm to the decoding of convolutional codes is discussed. The convolutional codes considered are binary codes with rate k_0/n_0, and overall memory $k_0\nu$. The encoded sequence is assumed to pass through a DMC.

A. Encoding

For convenience of notation, define:

u: An input bit

x: An encoded bit

$U_k = (u_{1,k}, u_{2,k}, ..., u_{k_0,k})$: Input word to encoder at time k

$X_k = (x_{1,k}, x_{2,k}, ..., x_{n_0,k})$: Output word of encoder at time k

$S_k = (s_{1,k}, s_{2,k}, ..., s_{k_0\nu,k}) = (U_k, U_{k-1}, ..., U_{k-\nu+1})$: State value denoted by input bit sequence at time k.

The input to the encoder is assumed to consist of equally likely and independent symbols (an assumption used throughout all analysis in this thesis). The encoder is ensured to start from the

zero state by clearing the encoding registers prior to accepting the input data sequence. It is terminated in the zero state at time N by padding $k_0\nu$ zeros to end of the length $N_1 = N - k_0\nu$ input data sequence.

Figure 2.3 shows a shift-register based encoder structure for a rate 1/2 convolutional code with memory $\nu = 2$, and generator $G(7,5)$ in octal. A trellis diagram is shown in Figure 2.4 with $N_1 = 6$ and $N = 8$. In this figure, solid lines correspond to paths taken when input bit is a 0, and the dashed lines correspond to paths taken when the input bit is a 1. State values are determined by the contents of the encoding shift register.

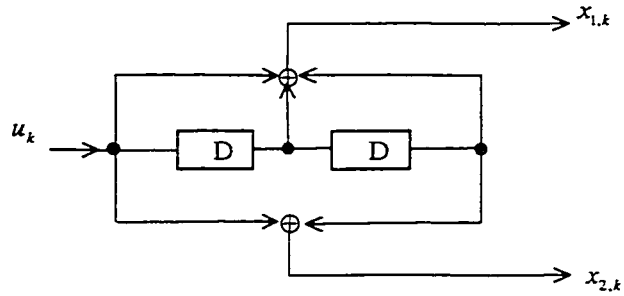


Figure 2.3: Rate 1/2 convolutional code encoder with generator $G(7,5)$.

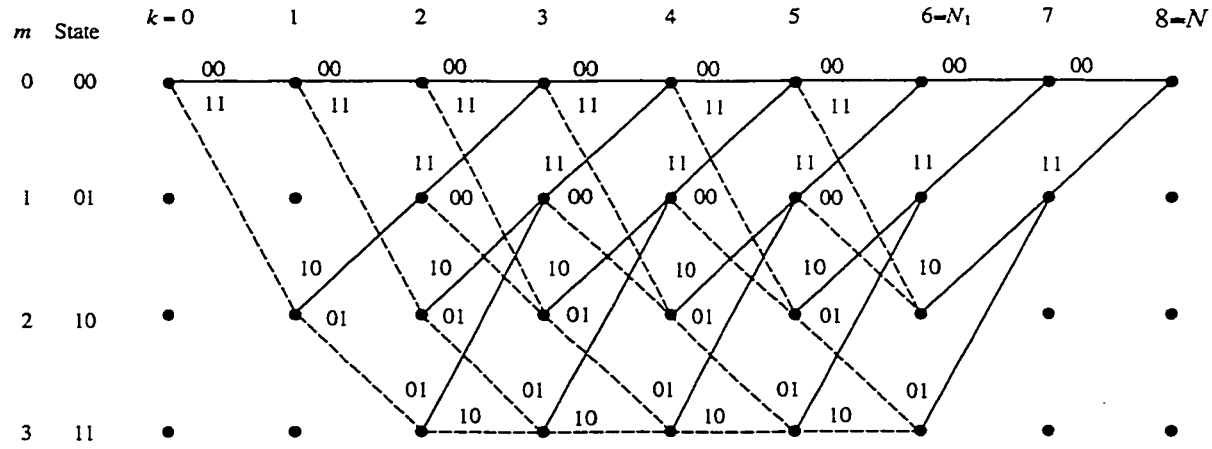


Figure 2.4: Trellis diagram for rate 1/2 convolutional code with generator G (7,5).

B. Channel

Assume that the channel is a DMC. Define symbol transition probabilities to be $r(y_{j,k} \mid x_{j,k})$. Because the memoryless property of the channel, the block transition probability is:

$$R(Y_k \mid X_k) = \prod_{j=1}^{n_0} r(y_{j,k} \mid x_{j,k}) \ , \quad j = 1,2,...,n_0. \tag{2.15}$$

C. Decoding with BCJR

The computation of $\gamma_k(m',m)$ is critical for decoding convolutional codes with the BCJR algorithm. Where a path does not exist between states, the state transition probability is zero. Where paths do exist between states, the state transition probabilities are divided into two cases. First, for $k = 0,1,...,N_1 - 1$, the state transition probability, $p_k(m \mid m')$, is determined by the input statistics. Since all input sequences are assumed to be equally likely, and there are 2^{k_0} possible transitions out of each state, $p_k(m \mid m') = 2^{-k_0}$ for each of these transitions. For example, in Figure 2.4, because $k_0 = 1$, there are two possible transitions out of each state that is connected, and hence $p_k(m \mid m') = 0.5$ for each point of these states for $k = 0,1,...,N_1 - 1$. For $k = N_1,...,N-1$, since there is only one transition out of each state, $p_k(m \mid m') = 1$ for each connected pair of states.

The value of $q_k(X \mid m',m)$ expresses the probability of output X given the state m' and m. According to the trellis diagram, $q_k(X \mid m',m) = 0$ if a transition does not exist between state m' and m, and $q_k(X \mid m',m) \neq 0$ if a transition exists between state m' and m, where X denotes the branch values on the trellis diagram and can take 2^{n_0} different values. For example, in Figure 2.4, because $n_0 = 2$, there are four branch values ($X = 00,01,10,11$). In relation (A1.1) of appendix 1, for the calculation of $\gamma_k(m',m)$, given state m and m', there is a summation over all possible output symbols X. If there is more than one transition and output value, given state m' and m, $q_k(X \mid m',m) \neq 1$. But in convolutional codes, because X is a deterministic function of the state transition, when m and m' are given, only one X symbol is true. Therefore $q_k(X \mid m',m) = 1$ for one symbol value X if the transition does exist, and no summation is needed in evaluation of $\gamma_k(m',m)$ for convolutional codes.

Consider now the application of the BCJR algorithm to decoding the received symbol sequence. First, note that:

$$\begin{aligned} S_k &= (s_{1,k}, s_{2,k}, \ldots, s_{k_0 v,k}) \\ &= (U_k, U_{k-1}, \ldots, U_{k-v+1}) \\ &= (u_{1,k}, u_{2,k}, \ldots, u_{k_0,k}; u_{1,k-1}, u_{2,k-1}, \ldots, u_{k_0,k-1}; \ldots; u_{1,k-v+1}, u_{2,k-v+1}, \ldots, u_{k_0,k-v+1}) . \end{aligned}$$

Decoding the state values is equivalent to decoding the input bit sequence according to:

$$s_{j,k} = u_{j,k}, \quad j = 1, 2, \ldots, k_0 . \tag{2.16}$$

Using this relationship, evaluation of $\lambda_k(m)$ is sufficient to obtain an estimate of the input information bits, as shown by the following:

$$\begin{aligned} \Pr\{u_{j,k} = 0 | Y_1^N\} &= \Pr\{u_{j,k} = 0; Y_1^N\} / \Pr(Y_1^N) \\ &= \Pr\{s_{j,k} = 0; Y_1^N\} / \Pr(Y_1^N) \\ &= \frac{1}{\Pr\{Y_1^N\}} \sum_{s_k \in A_{j,k}} \Pr\{S_k = m; Y_1^N\} \\ &= \frac{1}{\Pr\{Y_1^N\}} \sum_{s_k \in A_{j,k}} \lambda_k(m) \end{aligned}$$

where $A_{j,k}$ is the set of states S_k such that $s_{j,k} = 0$. For a given Y_1^N, $\Pr\{Y_1^N\}$ is a constant and is equal to $\lambda_N(0)$ since:

$$\begin{aligned} \lambda_N(0) &= \Pr\{S_N = 0; Y_1^N\} \\ &= \Pr\{S_N = 0\} \cdot \Pr\{Y_1^N\} \\ &= 1 \cdot \Pr\{Y_1^N\} \\ &= \Pr\{Y_1^N\}. \end{aligned} \tag{2.17}$$

Since $\lambda_N(0)$ is evaluated as part of the BCJR algorithm, it is straightforward to evaluate:

$$\Pr\{u_{j,k} = 0 | Y_1^N\} = \frac{1}{\lambda_N(0)} \sum_{s_k \in A_{j,k}} \lambda_k(m) \tag{2.18}$$

and decode $u_{j,k} = 0$ if $\Pr\{u_{j,k} = 0 | Y_1^N\} \geq 0.5$, and otherwise decode $u_{j,k} = 1$. Note that the probability $\Pr\{u_{j,k} = 0 | Y_1^N\}$ is the APP for the input bit $u_{j,k}$. Note also that $\Pr\{u_{j,k} = 0 | Y_1^N\}$ is a soft value (a real number). This is an important characteristic required for iterative decoding.

To demonstrate summation of $\lambda_k(m)$ in formula (2.18), consider the two examples given below. Consider first the case with $k_0 = 1$ and $\nu = 3$. The number of states is $2^{k_0 \nu} = 8$ (Note that when $k_0 = 1$, u_k is used instead of $u_{1,k}$ for simplicity.)

State m	(u_k, u_{k-1}, u_{k-2})	
0	0 0 0	} Sum $\lambda_k(m)$ for these states when decoding u_k.
1	0 0 1	
2	0 1 0	
3	0 1 1	
4	1 0 0	
5	1 0 1	
6	1 1 0	
7	1 1 1	

For $1/n_0$ convolutional codes, the final step in decoding can always be performed by summing the probabilities $\lambda_k(m)$ of the upper $M/2$ states:

$$\Pr\{u_k = 0 \mid Y_1^N\} = \frac{1}{\lambda_N(0)} \sum_{m=0}^{\frac{M}{2}-1} \lambda_k(m) \tag{2.19}$$

When $k_0 = 2$ and $\nu = 2$, the number of states is $2^{k_0 \nu} = 16$:

State m	$(u_{1,k}, u_{2,k}, u_{1,k-1}, u_{2,k-1})$	
0	0 0 0 0	} Sum for $u_{1,k}$.
1	0 0 0 1	
2	0 0 1 0	
3	0 0 1 1	
4	0 1 0 0	
5	0 1 0 1	
6	0 1 1 0	
7	0 1 1 1	
8	1 0 0 0	} Sum for $u_{1,k}$.
9	1 0 0 1	
10	1 0 1 0	
11	1 0 1 1	
12	1 1 0 0	
13	1 1 0 1	
14	1 1 1 0	
15	1 1 1 1	

Sum for $u_{2,k}$.

When $k_0 > 1$, the first input bit can always be decoded by adding the values of $\lambda_k(m)$ for the upper $M/2$ states, but for the other input bits, the $\lambda_k(m)$ values must be selected by determining the states whose associated input bits have value zero.

The BCJR decoding process for convolutional codes is now summarized using matrix notation for values of $\alpha'_k(m)$, $\beta'_k(m)$ and $\gamma_k(m',m)$:

(1) Initialize $\alpha'_0(m) \rightarrow [\alpha'_0]_{1\times M} = [100,...,0]$, and $\beta'_N(m) \rightarrow [\beta'_N]_{N\times 1} = [100,...,0]^T$, where T denotes matrix transpose.

(2) As soon as Y_k is received, compute $\gamma_k(m',m) \rightarrow [\gamma_k]_{M\times M}$ and $\alpha'_k(m) \rightarrow [\alpha'_k]_{1\times M}$ using $[\alpha'_k]_{1\times M} = [\alpha'_{k-1}]_{1\times M}[\gamma_k]_{M\times M}$ with initial conditions $[\alpha'_0]_{1\times M}$. Store the $[\alpha'_k]_{1\times M}$ in a two-dimension matrix denoted by $[\alpha'_{mk}]_{M\times(N+1)}$. That is, $[\alpha'_{mk}]_{M\times(N+1)} = \left[[\alpha'_0]^T,...,[\alpha'_N]^T\right]$. The shape of this matrix matches the distribution of states in the trellis diagram.

(3) After the complete sequence Y_1^N has been received, calculate $\beta'_k(m) \rightarrow [\beta'_k]_{M\times 1}$ using $[\beta'_k]_{M\times 1} = [\gamma_{k+1}]_{M\times M}[\beta'_{k+1}]_{M\times 1}$ with initial conditions $[\beta'_N]_{M\times 1}$, in the reverse time order. If the $[\gamma_k]_{M\times M}$ values were not stored in step 2, they must be recalculated, also in the reverse time order. When $[\beta'_k]_{M\times 1}$ is available, $[\alpha'_k]_{1\times M}$ values can be recalled from $[\alpha'_{mk}]_{M\times(N+1)}$ and $\lambda_k(m)$ can be obtained from $\lambda_k(m) = \alpha'_k(m)\cdot\beta'_k(m)$, where this multiplication is component-by-component multiplication. After $\lambda_k(m)$ is obtained, use relation (2.19) to obtain the APP of the input bits. If the APP is larger than or equal to 0.5, the decoded bit is judged to be 0, otherwise the decoded bit is judged to be 1. Note that although $\lambda_k(m)$ is available for $k = 0,1,...,N$, $\lambda_0(m)$ cannot be used for decoding since $\alpha'_0(m)$ has no relationship to the input bit values.

To confirm the validity of the BCJR algorithm as outlined above, Figure 2.5 shows BER performance against signal to noise ratio for the VA and BCJR algorithms. Clearly, the performance of these algorithms is, for practical purposes, identical. The BCJR algorithm is more complicated than the VA. However it generates APP values that are used for the decoding of turbo codes.

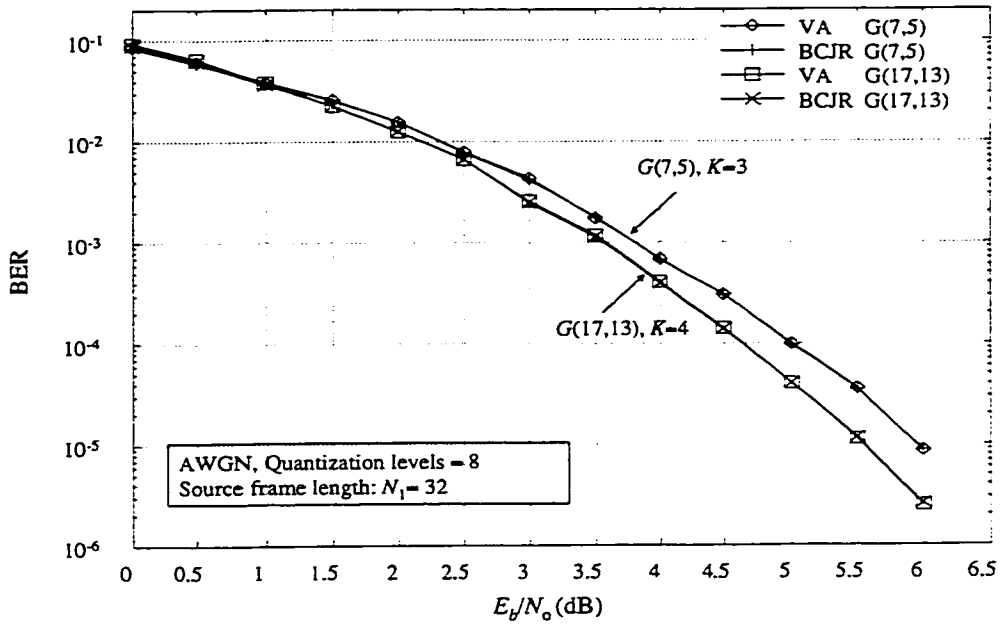


Figure 2.5: BER performance vs. E_b/N_0 for VA and BCJR algorithm.

Determination of the $\gamma_k(m',m)$ values is very important to the BCJR decoding process. The following example details the calculation of $\gamma_k(m',m)$ when $k=3$. Let the system parameters be:

- Input: Binary bit sequences.
- Convolutional encoder: $k_0=1, n_0=2, \nu=2, M=4$ and $G(7,5)$.
- Channel: Gaussian symmetric DMC.
- Demodulation: Four quantization levels with symbol transition probabilities as given in Table 2.1.

Table 2.1: Symbol transition probability $r(y_{j,k} \mid x_{j,k})$

$x_{j,k}$ \ $y_{j,k}$	0	1	2	3
0	0.5	0.3	0.15	0.05
1	0.05	0.15	0.3	0.5

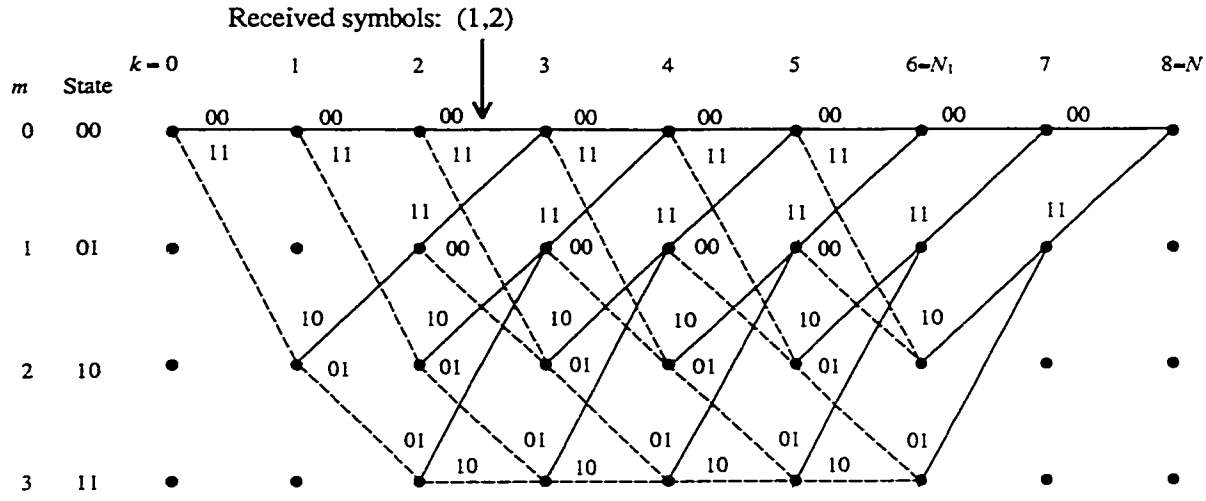


Figure 2.6: Trellis diagram for example calculation of $\gamma_3(m',m)$.

Consider calculation of $\gamma_3(m',m)$ values when the received symbols in this interval are (1,2). In matrix form, these values comprise the matrix $[\gamma_3]_{4\times4}$. Some entries of this matrix can immediately be set to zero by observing the states in the trellis diagram between which transitions do not exist:

$$[\gamma_3]_{4\times4} = \begin{bmatrix} & 0 & & 0 \\ & 0 & & 0 \\ 0 & & 0 & \\ 0 & & 0 & \end{bmatrix}.$$

With knowledge of the received symbols, the coded symbol values (also called branch values) on the trellis diagram, and the symbol transition probabilities in Table 2.1, the non-zero values for matrix $[\gamma_3]_{4\times4}$ can be calculated. For example, consider the branch with coded symbols $(0,0)$. If the received symbols are $(1,2)$, $\gamma_3(0,0) = \gamma_3(1,2) = 0.5 \cdot r(1|0) \cdot r(2|0) = 0.5 \cdot 0.3 \cdot 0.15 = 0.0225$, where 0.5 is the value of $p_3(m|m')$ since there are two possible transitions out of each state for $k < N_1$. For the branch with coded symbols $(0,1)$, $\gamma_3(2,3) = \gamma_3(3,1) = 0.5 \cdot r(1|0) \cdot r(2|1)$ $= 0.5 \cdot 0.3 \cdot 0.3 = 0.045$. Similarly, the other non-zero entries for matrix $[\gamma_3]_{4\times4}$ can be obtained:

$$[\gamma_3]_{4\times4} = \begin{bmatrix} 0.0225 & 0 & 0.0225 & 0 \\ 0.0225 & 0 & 0.0225 & 0 \\ 0 & 0.01125 & 0 & 0.045 \\ 0 & 0.045 & 0 & 0.01125 \end{bmatrix}.$$

Note that the number of different non-zero values is determined by the number of coded symbols. Although eight non-zero values appear in the matrix $[\gamma_3]_{4\times4}$, actually only four calculations are required since there are only four different coded symbols (00, 01, 10, 11).

2.2 Modified BCJR Algorithm (M-BCJR)

For conventional convolutional codes, the BCJR algorithm and the VA can be used to decode the received bit sequences; for recursive systematic convolutional codes, the VA can be directly applied, but the BCJR algorithm has to be modified. The reason for the modification is that the relationship between the states and input bits cannot be constructed directly. This section first introduces the structure of RSC codes and then presents the modified BCJR algorithm [1] in detail.

2.2.1 Recursive Systematic Convolutional Codes

It is well known that the BER performance of a conventional systematic convolutional (SC) code is worse than that of a conventional non-systematic convolutional (NSC) code with the same constraint length at large signal to noise ratios (SNR's) [1]. But at low SNR's, the performance of SC codes is slightly better. The BER performance of an RSC code that integrates the properties of NSC and SC codes can be better than the equivalent NSC code for code rates greater than or equal to 2/3 [20]. Figure 2.7 shows BER performance for SC, RSC and NSC codes with constraint length 3 and rate 1/2. From this figure, it can be seen that the performance of the RSC code is slightly worse than that of the NSC code from medium to high SNR's for code rate 1/2.

Figure 2.8 compares BER performance for RSC and NSC codes with different constraint lengths and rate 1/2. From this figure, it can be seen that the performance of the two codes becomes closer when the constraint length is larger.

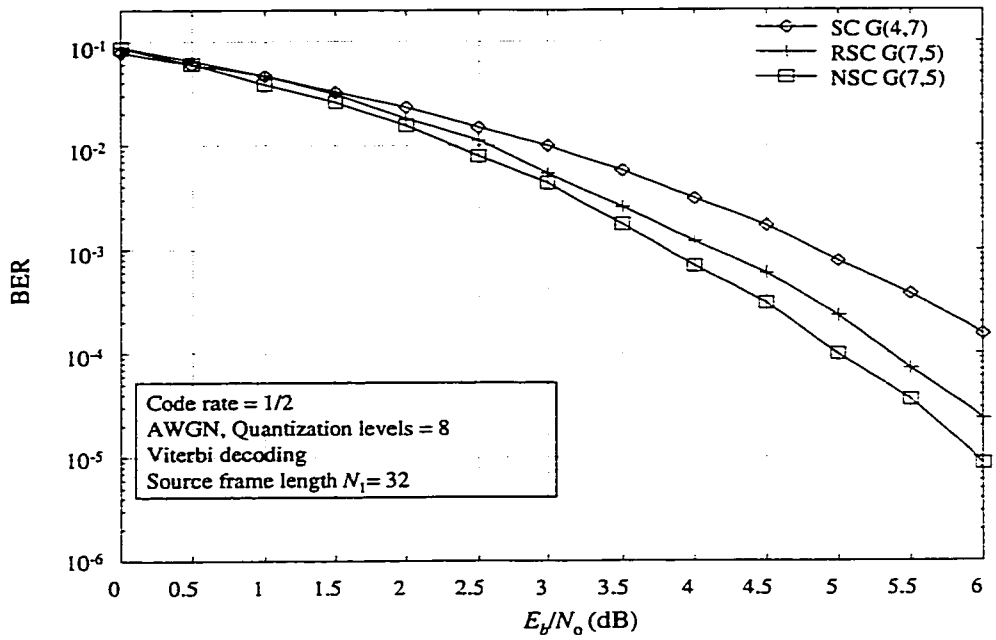


Figure 2.7: BER performance vs. E_b/N_o for SC, RSC and NSC codes with constraint length K=3.

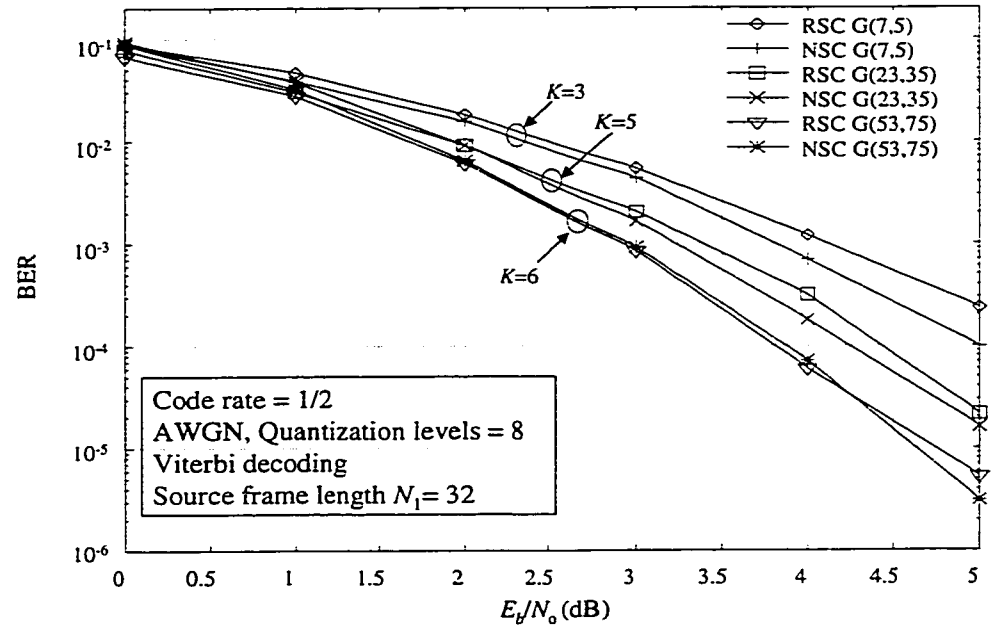


Figure 2.8: BER performance vs. E_b/N_o for RSC and NSC codes with different constraint length K.

A binary RSC code can be obtained from an NSC code by using a feedback loop and setting one of the two outputs equal to the input bit. Figure 2.9 shows a rate 1/2 NSC encoder and Figure 2.10 illuminates the corresponding RSC encoder, which has two output bits $x_{s,k}$ and $x_{p,k}$, where

$x_{s,k}$ is the systematic bit that is equal to u_k, and $x_{p,k}$ is the parity bit. Note that the minimum free distance of these two codes is identical in the trellis diagram, but the mapping from input bit sequence to encoded symbol sequence differs. The states of the RSC encoder are dependent on a_k, and not directly on u_k, therefore when summing the values of $\lambda_k(m)$ in the BCJR algorithm, only a_k can be estimated. What is needed, however, is the estimation of u_k. This is the reason that the BCJR algorithm has to be modified: to accommodate the mapping from u_k to a_k. The modified BCJR algorithm (M-BCJR) can be used to decode the values of the input bits.

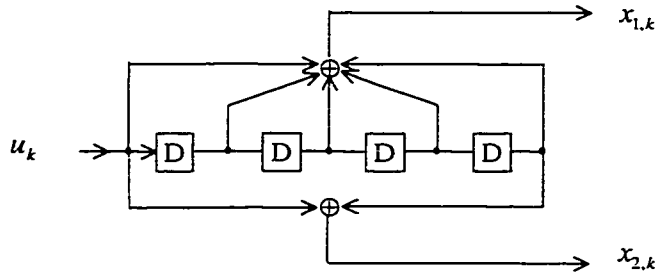


Figure 2.9: Non-systematic convolutional code encoder with code rate 1/2 and generator $G(37,21)$.

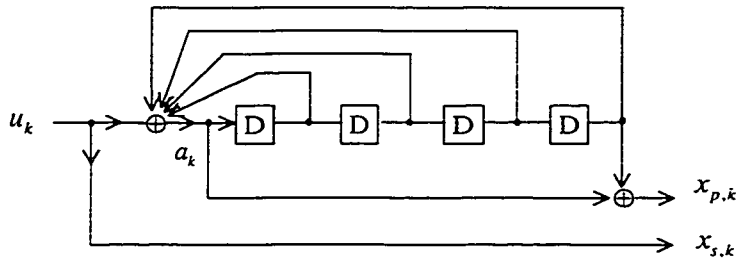


Figure 2.10: Recursive systematic convolutional code encoder with code rate 1/2 and generator $G(37,21)$.

2.2.2 M-BCJR Algorithm for RSC Codes

Note that the derivation of the relationships below is limited to rate $1/n_0$ RSC codes for simplicity, because rate $1/n_0$ RSC codes are the component codes used for turbo codes. If a high rate code is desired, the parity bits can be punctured.

A. Encoding

Define the symbols:

u_k : Input to encoder at time k.

$X_k = (x_{s,k}, x_{1p,k}, ..., x_{(n_0-1)p,k})$: Output at time k.

$S_k = (s_{1,k}, s_{2,k}, ..., s_{v,k}) = (a_k, a_{k-1}, ..., a_{k-v+1})$: State at time k.

$S_0 = 0$: Initial state at time 0.

$S_N = 0$: Final state at time N.

The RSC encoder is terminated at time N. Since the encoder is recursive, it is not sufficient to set the last v information bits to zeros in order to drive the encoder to the zero state; setting the last v bits of a_k to zeros will drive the encoder to return to the zero state. The last v input information bits u_k required to ensure that the bits a_k are zeros can be calculated from the contents of the registers. Figure 2.11 shows a trellis termination structure for a rate 1/2 RSC code [8]. For the first N_1 bits, the switch is in position 1, and $x_{s,k} = u_k$, $k = 1, 2, ..., N_1$. Then the switch is moved to position 2, and $x_{s,k}$, $k = N_1 + 1, ..., N$, are obtained from the feedback path of the registers. During this period, a_k is always zero and this drives the state of encoder to zero at time N.

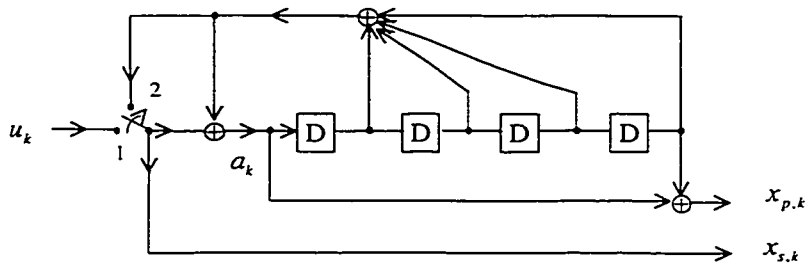


Figure 2.11: RSC encoder with trellis termination.

B. Channel

Assume that the channel is the additive white Gaussian noise (AWGN) memoryless channel. If the received signal is not quantized, the channel is called the discrete-input continuous-output Gaussian memoryless channel and if the received signal is quantized, the channel is called the discrete-input discrete-output Gaussian memoryless channel. For the discrete-input discrete-output Gaussian memoryless channel, define:

- Symbol transition probability $r(y_{j,k} \mid x_{j,k})$.
- Block transition probability $R(Y_k \mid X_k) = \prod_{j=1}^{n_0} r(y_{j,k} \mid x_{j,k})$.

C. Decoding with M-BCJR

The decoder examines Y_1^N and generates the APP associated with binary information bit u_k from the probability $\lambda_k^i(m)$ defined by:

$$\lambda_k^i(m) = \Pr\{u_k = i, S_k = m \mid Y_1^N\},\ i = 0,1 \text{ and } m = 0,1,...,M-1, \tag{2.20}$$

and thus, the APP of the decoded information bit u_k is:

$$\Pr\{u_k = i \mid Y_1^N\} = \sum_{m=0}^{M-1} \Pr\{u_k = i, S_k = m \mid Y_1^N\} = \sum_{m=0}^{M-1} \lambda_k^i(m). \tag{2.21}$$

The logarithm of the likelihood ratio (LLR) associated with the decoded bit u_k is equal to:

$$L(u_k) = \log \frac{\Pr\{u_k = 1 \mid Y_1^N\}}{\Pr\{u_k = 0 \mid Y_1^N\}} = \log \frac{\sum_{m=0}^{M-1} \lambda_k^1(m)}{\sum_{m=0}^{M-1} \lambda_k^0(m)}. \tag{2.22}$$

Given this LLR value, the decoder can make the decision:

$$\hat{u}_k = 1, \text{ if } \quad L(u_k) \geq 0,$$
$$\hat{u}_k = 0, \text{ if } \quad L(u_k) < 0. \tag{2.23}$$

In order to calculate the probability $\lambda_k^i(m)$, the probability functions $\alpha_k^i(m)$, $\alpha_k(m)$, $\beta_k(m)$, $\gamma_k^i(m',m)$ and $\gamma_k(m',m)$ must be defined. The definitions of these probabilities are somewhat different from those in the BCJR algorithm, the major difference being augmentation with the variable u_k. Define:

$$\alpha_k^i(m) = \Pr\{u_k = i; S_k = m \mid Y_1^k\} \tag{2.24}$$

$$\alpha_k(m) = \sum_{i=0}^{1} \alpha_k^i(m) = \Pr\{S_k = m \mid Y_1^k\} \tag{2.25}$$

$$\beta_k(m) = \frac{\Pr\{Y_{k+1}^N \mid S_k = m\}}{\Pr\{Y_{k+1}^N \mid Y_1^k\}} \tag{2.26}$$

$$\gamma_k^i(m',m) = \Pr\{u_k = i; Y_k; S_k = m \mid S_{k-1} = m'\} \tag{2.27}$$

$$\gamma_k(m',m) = \sum_{i=0}^{1} \gamma_k^i(m';m) = \Pr\{Y_k; S_k = m \mid S_{k-1} = m'\} \tag{2.28}$$

From these definitions, the following relationships can be constructed:

$$\hat{u}_k \Leftarrow L(u_k) \Leftarrow \lambda_k^i(m) \Leftarrow \begin{cases} \alpha_k^i(m) \Leftarrow \begin{cases} \alpha_{k-1}(m') \\ \gamma_k^i(m',m) \end{cases} \Leftarrow \begin{cases} Y_k \\ \text{Channel transition probability} \\ \text{Initial condition for } \alpha_0(m) \\ \text{Trellis diagram} \end{cases} \\ \beta_k(m) \Leftarrow \begin{cases} \alpha_k(m') \\ \gamma_k(m',m) \\ \beta_{k+1}(m) \end{cases} \Leftarrow \begin{cases} Y_k \\ \text{Channel transition probability} \\ \text{Initial condition for } \beta_N(m) \\ \text{Trellis diagram} \\ \text{Stored values of } \alpha_k(m) \end{cases} \end{cases}$$

Computation of these terms is outlined below. To streamline the presentation, some detailed derivations are deferred to Appendix 2.

(1) $\lambda_k^i(m)$

$$\lambda_k^i(m) = \Pr\{u_k = i; S_k = m \mid Y_1^N\}$$

$$= \frac{\Pr\{u_k = i; S_k = m; Y_1^N\}}{\Pr\{Y_1^N\}}$$

$$= \frac{\Pr\{(u_k = i; S_k = m; Y_1^k); Y_{k+1}^N\}}{\Pr\{Y_1^k; Y_{k+1}^N\}}$$

Pr(AB)=Pr(A)Pr(B|A)⇒

$$= \frac{\Pr\{u_k = i; S_k = m; Y_1^k\} \Pr\{Y_{k+1}^N \mid u_k = i; S_k = m; Y_1^k\}}{\Pr\{Y_1^k\} \Pr\{Y_{k+1}^N \mid Y_1^k\}}$$

Simplify the second term in the numerator by Markov property such that after time k

events are dependent on state S_k only. ⇒

$$= \frac{\Pr\{u_k = i; S_k = m; Y_1^k\} \Pr\{Y_{k+1}^N \mid S_k = m\}}{\Pr\{Y_1^k\} \Pr\{Y_{k+1}^N \mid Y_1^k\}}$$

$$= \frac{\Pr\{u_k = i; S_k = m; Y_1^k\}}{\Pr\{Y_1^k\}} \cdot \frac{\Pr\{Y_{k+1}^N \mid S_k = m\}}{\Pr\{Y_{k+1}^N \mid Y_1^k\}}$$

$$= \Pr\{u_k = i; S_k = m \mid Y_1^k\} \cdot \frac{\Pr\{Y_{k+1}^N \mid S_k = m\}}{\Pr\{Y_{k+1}^N \mid Y_1^k\}}$$

From (2.24) and (2.26) ⇒

$$= \alpha_k^i(m) \cdot \beta_k(m). \qquad (2.29)$$

(2) $\gamma_k^i(m',m)$

Evaluation of the branch transition probability $\gamma_k^i(m',m)$ is critical to calculation of the other quantities. As derived in Appendix 2:

$$\gamma_k^i(m',m) = \Pr\{Y_k \mid u_k = i; S_k = m; S_{k-1} = m'\} \cdot q_k(u_k \mid m',m) \cdot p_k(m \mid m'), \tag{2.30}$$

and thus $\gamma_k^i(m',m)$ is determined by the channel distribution and the encoder trellis diagram.

(3) $\alpha_k^i(m)$ and $\beta_k(m)$

Probabilities $\alpha_k^i(m)$ and $\beta_k(m)$ can be recursively calculated from the probability $\gamma_k^i(m',m)$. Please see Appendix 2 for the complete derivation.

$$\alpha_k^i(m) = \frac{\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^i(m',m)}{\sum_{m=0}^{M-1} \sum_{m'=0}^{M-1} \sum_{i=0}^{1} \alpha_{k-1}(m') \cdot \gamma_k^i(m',m)} \tag{2.31}$$

$$\alpha_k(m) = \frac{\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k(m',m)}{\sum_{m=0}^{M-1} \sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k(m',m)} \tag{2.32}$$

$$\beta_k(m) = \frac{\sum_{m'=0}^{M-1} \sum_{i=0}^{1} \cdot \beta_{k+1}(m') \cdot \gamma_{k+1}^i(m,m')}{\sum_{m=0}^{M-1} \sum_{m'=0}^{M-1} \sum_{i=0}^{1} \alpha_k(m) \cdot \gamma_{k+1}^i(m,m')} \tag{2.33}$$

$$\beta_k(m) = \frac{\sum_{m'=0}^{M-1} \gamma_{k+1}(m,m') \cdot \beta_{k+1}(m')}{\sum_{m'=0}^{M-1} \sum_{m=0}^{M-1} \alpha_k(m) \cdot \gamma_{k+1}(m,m')} \tag{2.34}$$

(4) $L(u_k)$

Consider calculation of the LLR $L(u_k)$. From relation (2.22):

$$L(u_k) = \log_e \frac{\Pr\{u_k = 1 \mid Y_1^N\}}{\Pr\{u_k = 0 \mid Y_1^N\}}$$

$$= \log_e \frac{\sum_{m=0}^{M-1} \lambda_k^1(m)}{\sum_{m=0}^{M-1} \lambda_k^0(m)}$$

$\lambda_k^i(m) = \alpha_k^i(m) \cdot \beta_k(m) \Rightarrow$

$$= \log_e \frac{\sum_{m=0}^{M-1} \alpha_k^1(m) \cdot \beta_k(m)}{\sum_{m=0}^{M-1} \alpha_k^0(m) \cdot \beta_k(m)}$$

In (2.31), the denominator has the same value when $i=1$ or $i=0$.
Let Δ denote this value. $\Rightarrow$

$$= \log_e \frac{\sum_{m=0}^{M-1} \frac{\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^1(m',m)}{\Delta} \cdot \beta_k(m)}{\sum_{m=0}^{M-1} \frac{\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^0(m',m)}{\Delta} \cdot \beta_k(m)}$$

$$= \log_e \frac{\sum_{m=0}^{M-1} \sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^1(m',m) \cdot \beta_k(m)}{\sum_{m=0}^{M-1} \sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^0(m',m) \cdot \beta_k(m)} \tag{2.35}$$

$$= \log_e \sum_{m=0}^{M-1} \sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^1(m',m) \cdot \beta_k(m)$$

$$- \log_e \sum_{m=0}^{M-1} \sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^0(m',m) \cdot \beta_k(m) \tag{2.36}$$

Comparing the calculation of $L(u_k)$ using relation (2.22) and (2.35), it is noticed that memory can be saved by use of relation (2.35), because only values of $\alpha_k(m)$ are stored, not $\alpha_k^0(m)$ and $\alpha_k^1(m)$, and it is not necessary to introduce the parameter $\lambda_k^i(m)$. In addition, relation (2.22) involves multiplication of probabilities with very small values, so use of relation (2.36) is highly recommended for avoiding floating point calculation errors.

The process of decoding RSC codes using the M-BCJR algorithm is now summarized:

(1) Initialize $\alpha_0(m)$ and $\beta_N(m)$, $m = 0,1,...,M-1$.

(2) As soon as Y_k is received, calculate $\gamma_k(m',m)$ and $\alpha_k(m)$. For all k and m, store the values of $\alpha_k(m)$ for subsequent calculation of $\beta_k(m)$ and $L(u_k)$.

(3) After the complete sequence Y_1^N has been received, in the reverse time order from time N, calculate $\gamma_k^i(m';m)$ and $\beta_k(m)$, obtain $L(u_k)$, and make a decision based on the sign of $L(u_k)$.

Now the decoding process is illuminated using matrix expressions.

A. Forward Process

(1) Initialize $\alpha_0(m) \rightarrow [\alpha_0]_{1\times M} = [100,...,0]$.

(2) After obtaining the demodulator output at time k $(k = 1,2,...,N)$ starting from time $k = 1$, calculate $\gamma_k(m',m) \rightarrow [\gamma_k]_{M\times M}$ using the demodulator output values, trellis diagram and symbol transition probabilities if the received signals are quantized, or channel distribution function if the received signals are not quantized.

(3) Calculate $\alpha_k(m) \rightarrow [\alpha_k]_{1\times M}$ using $[\alpha_{kh}]_{1\times M} = [\alpha_{k-1}]_{1\times M}[\gamma_k]_{M\times M}$, $[\alpha_k]_{1\times M} = \frac{1}{S_{\alpha_{kh}}} \cdot [\alpha_{kh}]_{1\times M}$, where $S_{\alpha_{kh}}$ = summation over all entries of matrix $[\alpha_{kh}]_{1\times M}$.

(4) Store the $[\alpha_k]_{1\times M}$ to a two-dimensional matrix $[\alpha_{mk}]_{M\times(N+1)}$, $[\alpha_{mk}]_{M\times(N+1)} = \left[[\alpha_0]^T, [\alpha_1]^T, ..., [\alpha_N]^T\right]$.

B. Backward process

After finishing the calculation of $[\alpha_{mk}]_{M\times N}$, in the inverse time order starting from time N, $\gamma_k^i(m',m)$, $\beta_k(m)$ and $L(u_k)$ are calculated, and a decision regarding the value of the information symbol can be made immediately.

(1) Initialize $\beta_N(m) \rightarrow [\beta_N]_{M\times 1} = [100,...,0]^T$.

(2) Fetch demodulator output values for time k $(k = 1,2,...,N)$ starting from time $k = N$, and then calculate $\gamma_k^i(m',m) \rightarrow [\gamma_k^i]_{M\times M}$ with $i = 0$ and 1, and $[\gamma_k]_{M\times M} = [\gamma_k^0]_{M\times M} + [\gamma_k^1]_{M\times M}$.

(3) Calculate $\beta_k(m) \rightarrow [\beta_k]_{M\times 1}$ with $[\beta_k]_{M\times 1} = \frac{1}{S_{\alpha_{k\beta}}} \cdot [\beta_{kh}]_{M\times 1}$ for time k $(k = 1,...,N-1)$ starting from time $k = N-1$, where $S_{\alpha_{k\beta}}$ is the summation over all entries of matrix $[\alpha_{k\beta}]_{1\times M}$ with $[\alpha_{k\beta}]_{1\times M} = [\alpha_k]_{1\times M}[\gamma_{k+1}]_{M\times M}$ by recalling $[\alpha_k]_{1\times M}$ from $[\alpha_{mk}]_{M\times(N+1)}$, and $[\beta_{kh}]_{M\times 1} = [\gamma_{k+1}]_{M\times M}[\beta_{k+1}]_{M\times 1}$.

(4) Calculate $L(u_k)$ for time k $(k = 1,2,...,N)$ starting from time $k = N$, using $L(u_k) = \log_e[\alpha_{k-1}]_{1\times M}[\gamma_k^1]_{M\times M}[\beta_k]_{M\times 1} - \log_e[\alpha_{k-1}]_{1\times M}[\gamma_k^0]_{M\times M}[\beta_k]_{M\times 1}$, and make a decision based on the sign of $L(u_k)$.

Due to its importance in the decoding process, an example for calculation of $\gamma_k^i(m',m)$ is now given. Consider the following system parameters:

- Encoder input: binary bit sequences with equal probabilities.
- Encoder: RSC code with $k_0 = 1, n_0 = 2, \nu = 2, M = 4, G(7,5)$.
- Channel: memoryless AWGN channel.
- $E_b / N_0 = 4$ dB.
- BPSK modulation: logic one $\rightarrow$ +1, logic zero $\rightarrow$ -1.
- Demodulator output: unquantized.
- Decoding algorithm: M-BCJR.
- Decoder input at time $k = 3$: $Y_3 = (y_{s,3}, y_{p,3}) = (1.2, -0.5)$.

Consider calculation of $\gamma_3^i(m',m) \rightarrow [\gamma_3^i]_{M\times M}$. First set to zero entries of matrix $[\gamma_3]_{4\times 4}$, $[\gamma_3^0]_{4\times 4}$ and $[\gamma_3^1]_{4\times 4}$, which correspond to states between which there are no transitions. According to the trellis diagram given in Figure 2.12:

$$[\gamma_3]_{4\times 4} = \begin{bmatrix} & 0 & & 0 \\ & 0 & & 0 \\ 0 & & 0 & \\ 0 & & 0 & \end{bmatrix}, \quad [\gamma_3^0]_{4\times 4} = \begin{bmatrix} & 0 & 0 & 0 \\ 0 & 0 & & 0 \\ 0 & 0 & 0 & \\ 0 & & 0 & 0 \end{bmatrix}, \quad [\gamma_3^1]_{4\times 4} = \begin{bmatrix} 0 & 0 & & 0 \\ & 0 & 0 & 0 \\ 0 & & 0 & 0 \\ 0 & 0 & 0 & \end{bmatrix}.$$

When $k=3$, demodulator output is (1.2,-0.5).

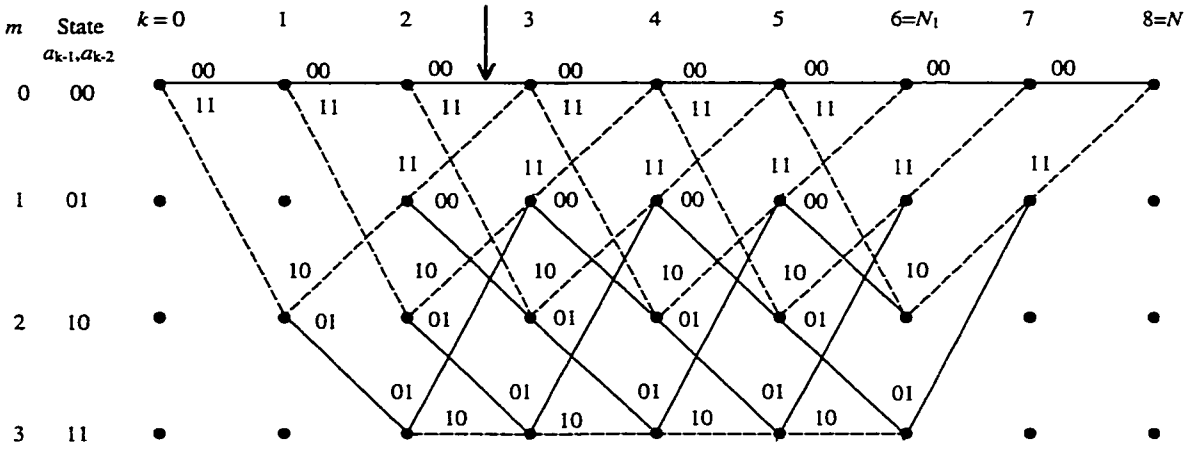


Figure 2.12: Trellis diagram for example calculation of $\gamma_3^i(m',m)$ using the M-BCJR algorithm.

Now, use the demodulator output values and different branch values on the trellis diagram at time $k=3$ to obtain the non-zero values for these matrices. The channel variance σ^2 or N_0, the single-sided power spectral density of white noise, is required for the calculation of the probability density function.

Since $E_b/N_0 = 4$ dB, the code rate $R = k_0/n_0 = 1/2$ (note that strictly speaking, with terminating bits, the code rate is slightly less than 1/2, however the value of 1/2 is used because the reduction of the code rate is small for larger frame size and smaller memory of code generators, which are common in turbo codes), and the average symbol energy is assumed to be $E_s = 1.0$:

$$E_s/N_0 = R \cdot E_b/N_0,$$

$$E_b/N_0 = 10^{\frac{4}{10}} = 2.512,$$

$$\frac{1.0}{N_0} = \frac{1}{2} \cdot 2.512 \Rightarrow N_0 = 0.796 \Rightarrow \sigma^2 = \frac{N_0}{2} = 0.398.$$

Now relation (A2.5) is used to compute the probability $\Pr\{Y_k \mid u_k = i; S_k = m; S_{k-1} = m'\}$ $= e^{-\frac{1}{N_0}((y_{s,k}-b_s(i,m',m))^2+(y_{p,k}-b_p(i,m',m))^2)}$. Consider evaluation of $\left[\gamma_3^0\right]_{4\times4}$. Since $\left[\gamma_3^0\right]_{4\times4} =$

$\begin{bmatrix} & 0 & 0 & 0 \\ 0 & 0 & & 0 \\ 0 & 0 & 0 & \\ 0 & & 0 & 0 \end{bmatrix}$, calculation of only 4 non-zero values is needed. These values correspond to the branches whose corresponding input bit is zero. The first such branch is from state 0 to state 0 whose encoded output is two logic zeros. Since logic zeros are transmitted as –1's, and since the demodulated values were 1.2 and –0.5:

$$\Pr\{Y_3 \mid u_3 = 0; S_3 = 0; S_2 = 0\} = e^{-\frac{1}{0.796}[(1.2-(-1))^2+(-0.5-(-1))^2]} = 0.00167 .$$

Since $\gamma_k^i(m',m) = \Pr\{Y_k \mid u_k = i; S_k = m; S_{k-1} = m'\} \cdot q_k(u_k \mid m',m) \cdot p_k(m \mid m')$, then:

$$\gamma_3^0(0,0) = 0.00167 * 1.0 * 0.5 = 0.000835 .$$

Consider now the transition from state 2 to state 3, whose encoded output is logic zero and logic one. Since these symbols are transmitted as –1 and +1 respectively:

$$\Pr\{Y_3 \mid u_3 = 0; S_3 = 3; S_2 = 2\} = e^{-\frac{1}{0.796}[(1.2-(-1))^2+(-0.5-1)^2]} = 0.000135 ,$$

$$\gamma_3^0(2,3) = 0.000135 * 1.0 * 0.5 = 0.000068 .$$

Similarly, $\gamma_3^0(1,2) = 0.000835$ and $\gamma_3^0(3,1) = 0.000068$. Therefore:

$$\left[\gamma_3^0\right]_{4\times4} = \begin{bmatrix} 0.000835 & 0 & 0 & 0 \\ 0 & 0 & 0.000835 & 0 \\ 0 & 0 & 0 & 0.000068 \\ 0 & 0.000068 & 0 & 0 \end{bmatrix}.$$

To calculate $\left[\gamma_3^1\right]_{4\times4}$, focus on the branches whose corresponding input bit is one. Doing so according to the above procedure yields:

$$\left[\gamma_3^1\right]_{4\times4} = \begin{bmatrix} 0 & 0 & 0.028155 & 0 \\ 0.028155 & 0 & 0 & 0 \\ 0 & 0.347334 & 0 & 0 \\ 0 & 0 & 0 & 0.347334 \end{bmatrix}.$$

Finally, summing $\left[\gamma_3^0\right]_{4\times4}$ and $\left[\gamma_3^1\right]_{4\times4}$ yields:

$$[\gamma_3]_{4\times4}=\begin{bmatrix}0.000835 & 0 & 0.028155 & 0\\ 0.028155 & 0 & 0.000835 & 0\\ 0 & 0.347334 & 0 & 0.000068\\ 0 & 0.000068 & 0 & 0.347334\end{bmatrix}.$$

It is possible to obtain $[\gamma_k]_{M\times M}$ directly, not from the summation of $\left[\gamma_k^0\right]_{M\times M}$ and $\left[\gamma_k^1\right]_{M\times M}$. However, for the decision purpose when $[\beta_k]_{M\times1}$ is calculated, it is necessary to first obtain $\left[\gamma_k^0\right]_{M\times M}$ and $\left[\gamma_k^1\right]_{M\times M}$.

Also, note that for the given decoder considered above, although there are eight non-zero values in $[\gamma_k]_{M\times M}$, only four different values are obtained because there are only four branch values. Similarly, there are only two different non-zero values for $\left[\gamma_k^0\right]_{M\times M}$ or $\left[\gamma_k^1\right]_{M\times M}$, because there are only two different branch values corresponding to the given input bit. For instance, given the input bit 0, the branch values must be $0x$, where x is either 0 or 1. Therefore only two different branch values are produced for $\left[\gamma_k^0\right]_{M\times M}$, although four non-zero values are needed to be determined. If this feature is used, half of the calculations can be saved for the computation of $[\gamma_k]_{M\times M}$ and $\left[\gamma_k^i\right]_{M\times M}$. Generally, for $k_0=1$, there are $2^1\cdot2^\nu=2^{\nu+1}=2^K$ branches, and there are 2^{n_0} different branch values. If $K>n_0\geq2$, the number of computations required during evaluation of $[\gamma_k]_{M\times M}$ can be reduced by up to $1-\frac{2^{n_0}}{2^K}=1-2^{-(K-n_0)}$. Table 2.2 shows the reduced computation load of $[\gamma_k]_{M\times M}$ for $k_0=1, n_0=2$ using this feature.

Table 2.2: Saved computation load for $k_0=1, n_0=2$

K	Reduced computation load
3	50%
4	75%
5	87.5%

2.3 Berrou's Turbo Coding

The amazing performance of turbo codes is a result of the encoder using parallel concatenation of RSC codes with random interleaving, and the associated decoder using iterative decoding. In this section, the encoder structure is introduced, and the decoder structure is described in detail.

2.3.1 Turbo Encoder

Figure 2.13 shows the block diagram of the turbo encoder, which consists of two identical RSC encoders with a random interleaver. The output of the two RSC encoders can be either punctured or not punctured. The RSC codes are of rate 1/2.

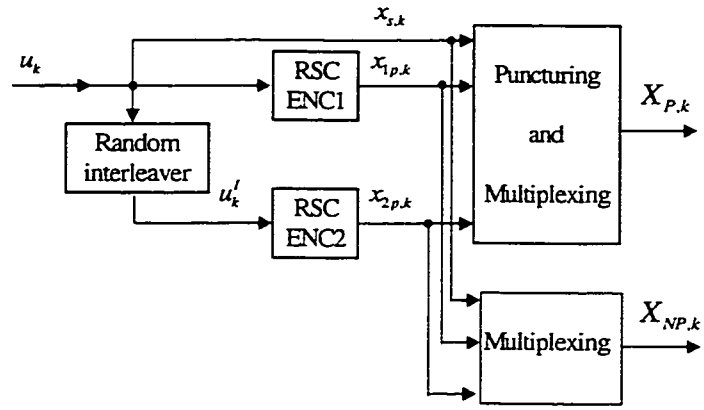
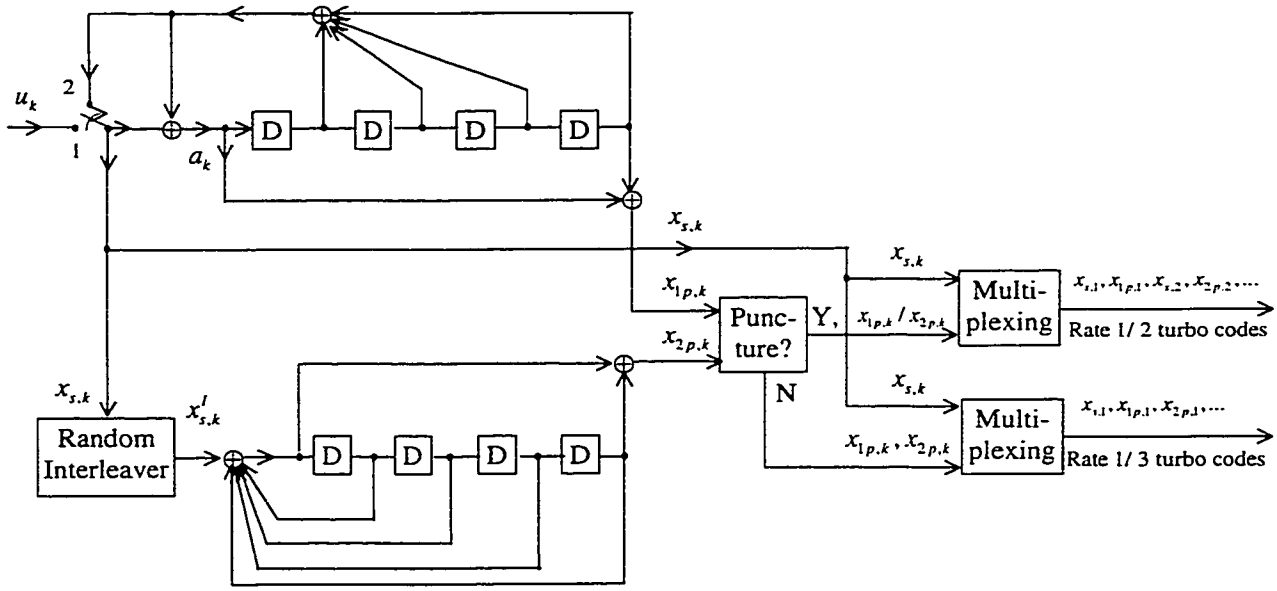


Figure 2.13: Block diagram of turbo code encoder with punctured or non-punctured outputs.

The input of the first RSC encoder is the bit sequence $\overline{u} = (u_1, ..., u_k, ...u_N)$. The interleaver permutes this sequence to form the sequence $\overline{u}^I$, which is the input sequence of the second RSC encoder. At time k, $x_{s,k}$ is the systematic component of the turbo encoder, $x_{1p,k}$ is the parity bit generated by the first RSC encoder, and $x_{2p,k}$ is the parity bit generated by the second RSC encoder. The parity bits can be punctured according to a desired puncturing matrix to obtain a higher code rate. No puncturing yields a rate 1/3 turbo code, where $X_{NP} = \{X_{NP,k}\} = \{x_{s,k}, x_{1p,k}, x_{2p,k}\} = (x_{s,1}, x_{1p,1}, x_{2p,1}, ..., x_{s,k}, x_{1p,k}, x_{2p,k}, ..., x_{s,N}, x_{1p,N}, x_{2p,N})$. If parity bits are punctured according to the typical puncturing matrix $P = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$ with the puncturing period 2, the result is a rate 1/2 turbo code. In Figure 2.13, $X_P = \{X_{P,k}\} = \{x_{s,k}, x_{1p,k} \text{ or } x_{2p,k}\}$

$= (x_{s,1}, x_{1p,1}, E, x_{s,2}, E, x_{2p,2}, \ldots) = (x_{s,1}, x_{1p,1}, x_{s,2}, x_{2p,2}, x_{s,3}, x_{1p,3}, x_{s,4}, x_{2p,4}, \ldots)$, where the E's mark the locations of deleted bits.

A detailed turbo encoder structure is shown in Figure 2.14. Both RSC constituent encoders are assumed to start from the all zero state. The upper constituent encoder is forced to terminate, but the lower one is not terminated. These different operations will affect the initialization of $\beta_k(m)$ in the decoding algorithm. Because of the interleaver, the encoder is frame-oriented. For example, if the frame length is $N = 65536$ and RSC code memory is $\nu = 4$, the length of the information bit sequence $\bar{u}$ will be $N_1 = 65532$ $(N = N_1 + \nu)$ per frame. For time $1 \le k \le N_1$, the switch is in position 1 to accept the user data. For $N_1 < k \le N$, the switch is in position 2, so that the $x_{s,k}$ is obtained according to the contents of the shift register, not $\bar{u}$, so the upper constituent encoder will be properly terminated. After the whole $\bar{x}_s$ sequence of one frame is obtained, this sequence is passed through the interleaver to obtain $\bar{x}_s^I$, the permuted version of $\bar{x}_s$.



Note: Only first RSC encoder is terminated.

Figure 2.14: Turbo encoder structure with generator $G(37,21)$.

The systematic component of the second RSC encoder is not transmitted, because the turbo decoder will use the same interleaver as the one that is used by the encoder to recover this information.

Assume that the channel is AWGN with zero mean and variance σ^2, and that BPSK modulation is used.

2.3.2. Turbo Decoder

A. General structure

It is the feedback structure of the iterative turbo decoder, shown in Figure 2.15, which resembles the turbo engine principle that gives turbo-codes their name. This figure shows the decoding structure originally proposed by Berrou *et al.* in 1993. After the encoder output is transmitted through the noisy channel, the receiver receives the disturbed signal $Y_k' = (y_{s,k}, y_{1p,k}, y_{2p,k})$ at time k. If the parity bits are punctured, zeros are inserted into the punctured positions.

The iterative turbo decoder consists of two MAP decoders using Berrou's M-BCJR algorithm, an interleaver (the same as the one used for the turbo encoder), a corresponding deinterleaver, and the estimation and normalization blocks. In general, the greater the number of iterations, the fewer the number of decoded bits in error. After finishing a specified number of iterations, the decision block makes hard decisions to output an estimate of the information bits. Note that the two constituent decoders in Figure 2.15 are not identical. Decoder 1 has three inputs $(z_{s,k}, y_{s,k}, y_{1p,k})$, and decoder 2 has only two inputs $(\tilde{L}_1^N(u_k^I), y_{k,2p})$, where $z_{s,k}$ and $\tilde{L}_1^N(u_k^I)$ are generated in the decoding process. In the decoding scheme developed by Robertson, discussed in the next section, these two decoders are identical, each has three inputs, and the M-BCJR decoding algorithm is slightly different from Berrou's.

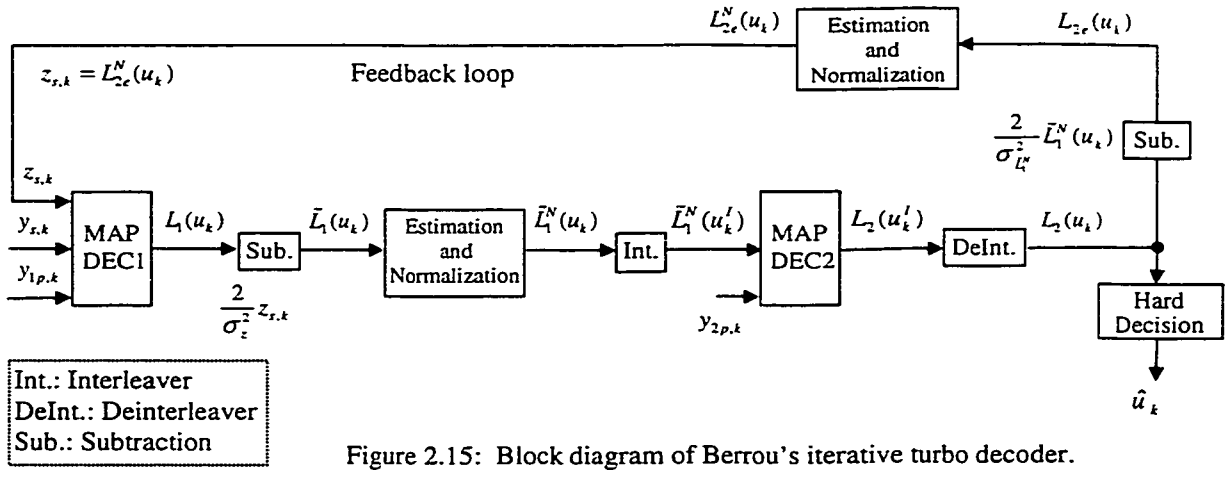


Figure 2.15: Block diagram of Berrou's iterative turbo decoder.

In Berrou's turbo decoder the output ($L_1(u_k)$ or $L_2(u_k)$) of each constituent decoder includes two components: one is the information ($z_{s,k}$ or $\tilde{L}_1^N(u_k)$) generated by the previous decoder, and the other is the information ($\tilde{L}_1(u_k)$ or $L_{2e}(u_k)$) generated by the decoder itself, which is passed

to the next decoder. Random variables $\tilde{L}_1(u_k)$ and $L_{2e}(u_k)$ have soft values, and the distributions of them can be assumed to be Gaussian distributed. However, the mean values of them are generally not ± 1 (say ± 20) (see relations (2.46-2.49)). It is necessary to normalize these two random variables to be used with the channel outputs $(y_{s,k}, y_{1p,k}, y_{2p,k})$ for turbo decoding.

B. Extrinsic information from the MAP decoder

In the following the extrinsic information is derived from the LLR $L(u_k)$. For purposes of generality, assume the MAP decoder has input $Y_k(y_{s,k}, y_{p,k})$ and output LLR $L(u_k)$. From relation (2.35), the LLR $L(u_k)$ is:

$$L(u_k) = \log_e \frac{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^1(m',m) \cdot \beta_k(m)}{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^0(m',m) \cdot \beta_k(m)}.$$

Since the encoder is systematic $(x_{s,k} = u_k)$, in the expression for $\gamma_k^i(m',m)\,(i=0,1)$ the transition probability $\Pr\{y_{s,k} \mid u_k = i; S_k = m; S_{k-1} = m'\}$ is independent of state values S_k and S_{k-1}. Therefore:

$$\begin{aligned}
\gamma_k^i(m',m) &= \Pr\{Y_k \mid u_k = i; S_k = m; S_{k-1} = m'\} \cdot q_k(u_k \mid m',m) \cdot p_k(m \mid m') \\
&= \Pr\{y_{s,k} \mid u_k = i; S_k = m; S_{k-1} = m'\} \cdot \Pr\{y_{p,k} \mid u_k = i; S_k = m; S_{k-1} = m'\} \\
&\quad \cdot q_k(u_k \mid m',m) \cdot p_k(m \mid m') \\
&= \Pr\{y_{s,k} \mid u_k = i\} \cdot \Pr\{y_{p,k} \mid u_k = i; S_k = m; S_{k-1} = m'\} \cdot q_k(u_k \mid m',m) \cdot p_k(m \mid m') \\
&= \Pr\{y_{s,k} \mid u_k = i\} \cdot \gamma_k^i(y_{p,k}, m', m)\,.
\end{aligned}$$

In the following derivation, note that the "$\log_e$" denotes the natural logarithm, i.e., base e, so:

$$L(u_k) = \log_e \frac{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^1(m',m) \cdot \beta_k(m)}{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^0(m',m) \cdot \beta_k(m)}$$

$$= \log_e \frac{\Pr\{y_{s,k} \mid u_k = 1\} \cdot \sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^1(y_{p,k}, m', m) \cdot \beta_k(m)}{\Pr\{y_{s,k} \mid u_k = 0\} \cdot \sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^0(y_{p,k}, m', m) \cdot \beta_k(m)}$$

$$= \log_e \frac{\Pr\{y_{s,k} \mid u_k = 1\}}{\Pr\{y_{s,k} \mid u_k = 0\}} + \log_e \frac{\sum_{m=0}^{M-1} \sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^1(y_{p,k}, m', m) \cdot \beta_k(m)}{\sum_{m=0}^{M-1} \sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^0(y_{p,k}, m', m) \cdot \beta_k(m)}. \tag{2.37}$$

The channel is assumed to be AWGN with zero mean and variance σ^2, and that BPSK modulation is used. In order to calculate the first component in relation (2.37), consider integrating the probability density function $f(\cdot \mid \cdot)$ over a very small range ε of integration.

$$\log_e \frac{\Pr\{y_{s,k} \mid u_k = 1\}}{\Pr\{y_{s,k} \mid u_k = 0\}} = \log_e \frac{f(y_{s,k} \mid u_k = 1) \cdot \varepsilon}{f(y_{s,k} \mid u_k = 0) \cdot \varepsilon}$$

$$= \log_e \frac{f(y_{s,k} \mid u_k = 1)}{f(y_{s,k} \mid u_k = 0)}$$

$$= \log_e \frac{\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{y_{s,k} - b_s(1)}{\sigma}\right)^2}}{\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{y_{s,k} - b_s(0)}{\sigma}\right)^2}}$$

$$= \log_e \frac{\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{y_{s,k} - 1}{\sigma}\right)^2}}{\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{y_{s,k} + 1}{\sigma}\right)^2}}$$

$$= -\frac{1}{2}\left(\frac{y_{s,k} - 1}{\sigma}\right)^2 + \frac{1}{2}\left(\frac{y_{s,k} + 1}{\sigma}\right)^2$$

$$= \frac{2}{\sigma^2} y_{s,k}. \tag{2.38}$$

Introducing relation (2.38) to (2.37) yields:

$$L(u_k) = \frac{2}{\sigma^2} y_{s,k} + \log_e \frac{\sum_{m=0}^{M-1} \sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^1(y_{p,k}, m', m) \cdot \beta_k(m)}{\sum_{m=0}^{M-1} \sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^0(y_{p,k}, m', m) \cdot \beta_k(m)}$$

$$= \frac{2}{\sigma^2} y_{s,k} + L_e(u_k), \tag{2.39}$$

$$L_e(u_k) = L(u_k)\big|_{y_{s,k}=0} = \log_e \frac{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^1(y_{p,k}, m', m) \cdot \beta_k(m)}{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^0(y_{p,k}, m', m) \cdot \beta_k(m)}. \quad (2.40)$$

$L_e(u_k)$ is called extrinsic information and it is a function of the parity introduced by the encoder. It is also a soft estimate of u_k. $L(u_k)$ is a better soft estimate of u_k, but it includes information that has been used by the constituent decoder itself. It is not proper to transfer $L(u_k)$ to next decoder. In turbo decoding, due to the presence of the interleaving between decoder 1 and decoder 2, the extrinsic information generated by decoder 2 is only weakly correlated with the inputs of decoder 1, so after normalization, it can be fed back to the decoder 1 to help the decoding of decoder 1. As shown in Figure 2.15, the normalized extrinsic information is denoted $z_{s,k}$.

The interface between the two constituent decoders, and the decoding algorithm is described in the following subsection in detail.

2.3.3 Interface between Two MAP Decoders and Berrou's Decoding Algorithm

A. MAP decoder 1

Interface

- Input:
  - $y_{s,k}, y_{1p,k}, z_{s,k}$ (For the first iteration, $z_{s,k} = 0$.).
  - σ^2, variance of the AWGN channel.
  - σ_z^2, variance of the meta-channel corresponding to $z_{s,k}$.
  - Encoding information.
- Output: $L_1(u_k)$.

Decoding algorithm

- Use the M-BCJR algorithm but since $z_{s,k}$ is a feedback component, the algorithm must be modified slightly.

- Assume that $z_{s,k}$ is a Gaussian variable with variance $\sigma_z^2 \neq \sigma^2$, where σ^2 is the channel variance of the AWGN.
- The calculation of the branch transition probability $\gamma_k^i(m',m)$ for the two decoders is different, because of the different number of inputs. In decoder 1, data input is $Y_k = (y_{s,k}, y_{1p,k}, z_{s,k})$, so in $\gamma_k^i(m',m)$, the probability $\Pr\{Y_k \mid u_k = i; S_k = m; S_{k-1} = m'\}$ is:

$$\Pr\{Y_k \mid u_k = i; S_k = m; S_{k-1} = m'\} = e^{-(y_{s,k} - b_s(i,m',m))^2/2\sigma^2} \cdot e^{-(y_{1p,k} - b_{1p}(i,m',m))^2/2\sigma^2} \cdot e^{-(z_{s,k} - b_z(i,m',m))^2/2\sigma_z^2}. \quad (2.41)$$

  Note that the calculation corresponding to $z_{s,k}$ in relation (2.41) will use the same bipolar branch values as the calculation for the systematic part, but also uses a variance other than the channel variance.
- The LLR $L_1(u_k)$ generated by decoder 1 is equal to:

$$L_1(u_k) = \log_e \frac{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^1(m',m) \cdot \beta_k(m)}{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^0(m',m) \cdot \beta_k(m)} \quad (2.42)$$

$$= \frac{2}{\sigma_z^2} z_{s,k} + \log_e \frac{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^1(y_{s,k}, y_{1p,k}, m', m) \cdot \beta_k(m)}{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^0(y_{s,k}, y_{1p,k}, m', m) \cdot \beta_k(m)}. \quad (2.43)$$

  In this instance, $\alpha_k(m)$ and $\beta_k(m)$ are initialized as follows, because the first encoder is terminated.

$$\alpha_0(0) = 1 \ , \ \alpha_0(m) = 0 \ \ \forall m \neq 0$$
$$\beta_N(0) = 1 \ , \ \beta_N(m) = 0 \ \ \forall m \neq 0 \quad (2.44)$$

B. Process after decoder 1 and before decoder 2

- The first component in relation (2.43) is discarded in order to avoid the $z_{s,k}$ generated by decoder 2 in the previous iteration being fed back to decoder 2. Subtracting this component yields:

$$\tilde{L}_1(u_k) = L_1(u_k) - \frac{2}{\sigma_z^2} z_{s,k} \quad (2.45)$$

- Assuming that $\tilde{L}_1^N(u_k)$, the normalized version of $\tilde{L}_1(u_k)$, is the output of the meta-channel, where the meta-channel consists of the serial concatenation of the AWGN channel and constituent decoders, the decrement of the variance of the meta-channel $\sigma^2_{\tilde{L}_1^N}$ compared to the variance of the channel shows the gain of decoding. The following estimation formulas [22] can be used to obtain $\tilde{L}_1^N(u_k)$ and the associated variance of the meta-channel:

$$m_{1\tilde{L}_1|} = \frac{1}{N} \cdot \sum_{k=1}^{N} |\tilde{L}_1(u_k)| \tag{2.46}$$

$$\sigma^2_{|\tilde{L}_1|} = \frac{1}{N} \cdot \sum_{k=1}^{N} |L_1(u_k)|^2 - (m_{1\tilde{L}_1|})^2 \tag{2.47}$$

$$\tilde{L}_1^N(u_k) = \tilde{L}_1(u_k) / m_{1\tilde{L}_1|} \tag{2.48}$$

$$\sigma^2_{\tilde{L}_1^N} = \sigma^2_{|\tilde{L}_1|} / m^2_{1\tilde{L}_1|}. \tag{2.49}$$

- Obtain $\tilde{L}_1^N(u_k^I)$, the interleaved version of $\tilde{L}_1^N(u_k)$, by using the same interleaver as the one used by the encoder. Note that the variance of the meta-channel corresponding to $\tilde{L}_1^N(u_k^I)$ is equal to that corresponding to $\tilde{L}_1^N(u_k)$.

C. MAP decoder 2

Interface

- Input:
  - $\tilde{L}_1^N(u_k^I), y_{2p,k}$.
  - σ^2, variance of channel.
  - $\sigma^2_{\tilde{L}_1^N}$, variance of the meta-channel corresponding to $\tilde{L}_1^N(u_k^I)$.
  - Encoding information.
- Output: $L_2(u_k^I)$.

Decoding algorithm

- Use the M-BCJR for decoding, but note that two different variances are used.
- For the calculation of branch transition probability $\gamma_k^i(m', m)$ with $Y_k = (\tilde{L}_1^N(u_k^I), y_{2p,k})$, the probability $\Pr\{Y_k \mid u_k = i; S_k = m; S_{k-1} = m'\}$ is calculated by:

$$\Pr\{Y_k \mid u_k = i; S_k = m; S_{k-1} = m'\} = e^{-(\tilde{L}_1^N(u_k^I) - b_s(i,m',m))^2 / 2\sigma^2_{\tilde{L}_1^N}} \cdot e^{-(y_{2p,k} - b_p(i,m',m))^2 / 2\sigma^2}.$$

- The LLR $L_2(u_k^I)$ generated by decoder 2 is equal to:

$$L_2(u_k^I) = \log_e \frac{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^i(m',m) \cdot \beta_k(m)}{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^j(m',m) \cdot \beta_k(m)}$$

$$= \frac{2}{\sigma^2_{\tilde{L}_1^N}} \tilde{L}_1^N(u_k^I) + \log_e \frac{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^1(y_{2p,k},m',m) \cdot \beta_k(m)}{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^0(y_{2p,k},m',m) \cdot \beta_k(m)}$$

$$= \frac{2}{\sigma^2_{\tilde{L}_1^N}} \tilde{L}_1^N(u_k^I) + \tilde{L}_2(u_k^I) \tag{2.50}$$

- $\alpha_k(m)$ and $\beta_k(m)$ are initialized as follows. Because the second encoder is not terminated, it is assumed that the encoding process can stop at any state with equal probability.

$$\alpha_0(0) = 1\ ,\ \alpha_0(m) = 0\ \ \forall m \neq 0\ .$$

$$\beta_N(m) = 1/M\ \ \forall m\text{, where } M \text{ denotes the number of states.} \tag{2.51}$$

D. Process after decoder 2 and before decoder 1

- Obtain $L_2(u_k)$, the deinterleaved version of $L_2(u_k^I)$:

$$L_2(u_k) = \frac{2}{\sigma^2_{\tilde{L}_1^N}} \tilde{L}_1^N(u_k) + \tilde{L}_2(u_k) \tag{2.52}$$

- If iteration is complete, the decision can be made for this frame by:

$$\hat{u}_k = 1, \quad \text{if} \quad L_2(u_k) \geq 0$$

$$\hat{u}_k = 0, \quad \text{if} \quad L_2(u_k) < 0\ . \tag{2.53}$$

- If iteration is not complete, $\tilde{L}_2(u_k)$ can be obtained by subtracting the term corresponding $\tilde{L}_1^N(u_k)$ by:

$$\tilde{L}_2(u_k) = L_2(u_k) - \frac{2}{\sigma^2_{\tilde{L}_1^N}} \tilde{L}_1^N(u_k)\ . \tag{2.54}$$

Let $L_{2e}(u_k) = \tilde{L}_2(u_k)$, where $L_{2e}(u_k)$ denotes the extrinsic information generated by decoder 2. Note that $\tilde{L}_1(u_k)$ cannot be called extrinsic information, because $\tilde{L}_1(u_k)$ includes the systematic component $y_{s,k}$.

- Also assuming that $z_{s,k} = L_{2e}^{N}(u_k)$ is the normalized version of $L_{2e}(u_k)$ and therefore is the output of the meta-channel, $z_{s,k}$ and the variance of the meta-channel with output $z_{s,k}$ can then be obtained:

$$m_{|L_{2e}|} = \frac{1}{N} \cdot \sum_{k=1}^{N} |L_{2e}(u_k)|$$

$$\sigma^2_{|L_{2e}|} = \frac{1}{N} \cdot \sum_{k=1}^{N} |L_{2e}(u_k)|^2 - (m_{|L_{2e}|})^2$$

$$z_{s,k} = L_{2e}(u_k) / m_{|L_{2e}|} \tag{2.55}$$

$$\sigma_z^2 = \sigma^2_{|L_{2e}|} / (m_{|L_{2e}|})^2 \tag{2.56}$$

The whole decoding process is now summarized in Figure 2.16. Before commencing decoding, obtain $Y_k' = (y_{s,k}, y_{1p,k}, y_{2p,k})$, channel information, and turbo encoding information, and then follow the procedures outlined in the block diagram of Figure 2.16.

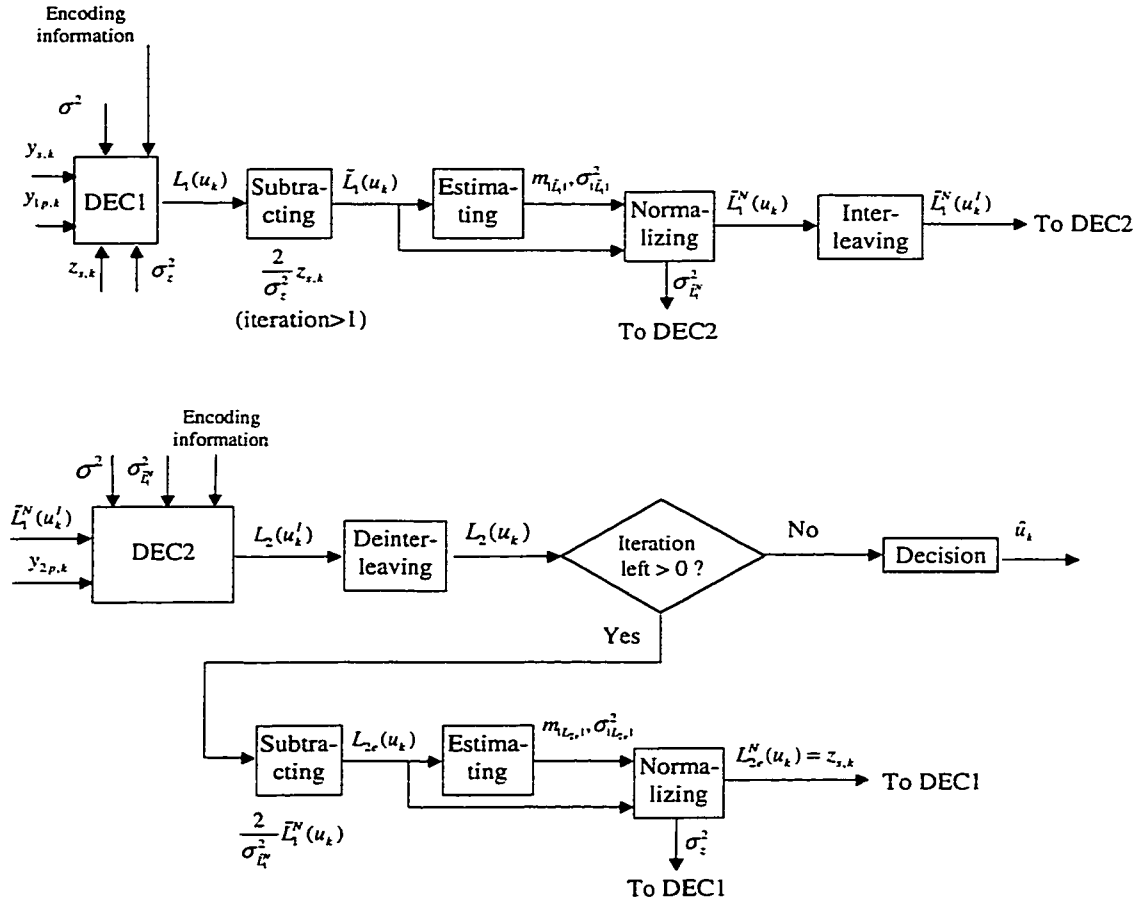


Figure 2.16: Implementation of turbo decoding by Berrou's algorithm.

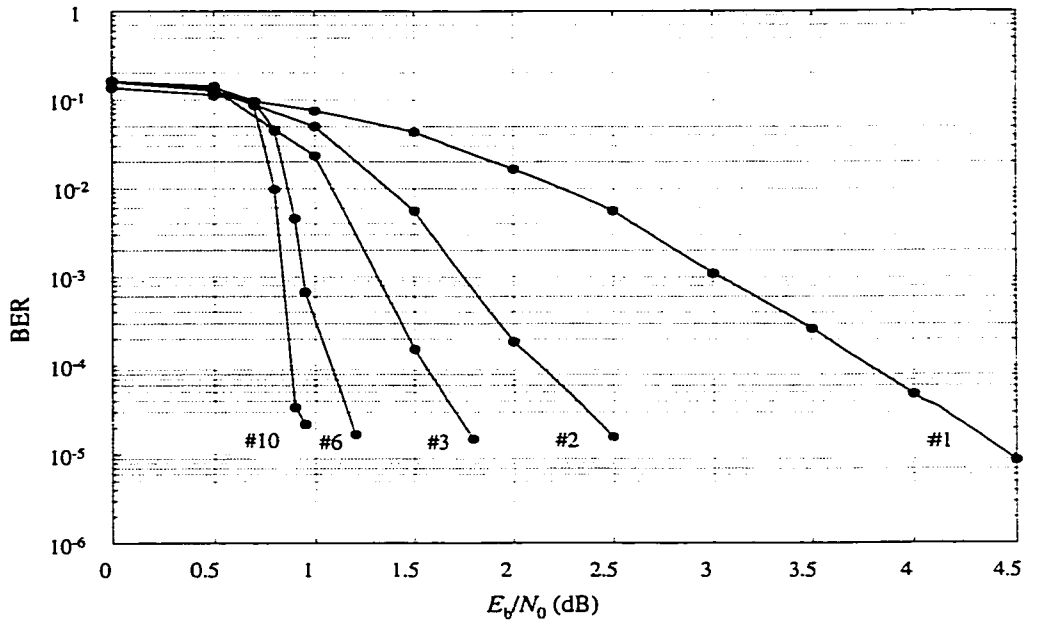


Figure 2.17: BER for transmission through AWGN channel with code rate 1/2, $G(37,21)$, Berrou's interleaver, $N = 65536$, terminating 1st encoder, and using Berrou's decoding algorithm.

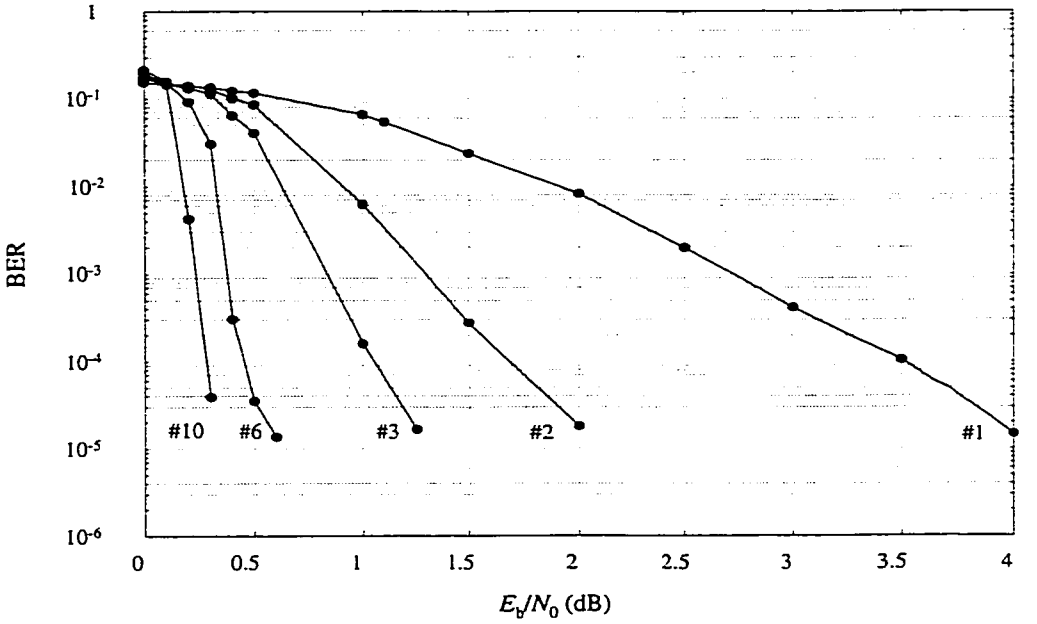


Figure 2.18: BER for transmission through AWGN channel with code rate 1/3, $G(37,21)$, Berrou's interleaver, $N = 65536$, terminating 1st encoder, and using Berrou's decoding algorithm.

Figure 2.17, Figure 2.18 and Figure 2.19 show simulation results of BER performance obtained using Berrou's turbo decoding algorithm and Berrou's interleaver discussed in section 2.7. These simulations are for $G(37,21)$, with a frame length $N = 65536$ and with code rates 1/2 and 1/3. Clearly, in general, the performance improves as the number of iterations increases.

However, it should be noted that at very low E_b / N_o, performance becomes worse with more iterations.

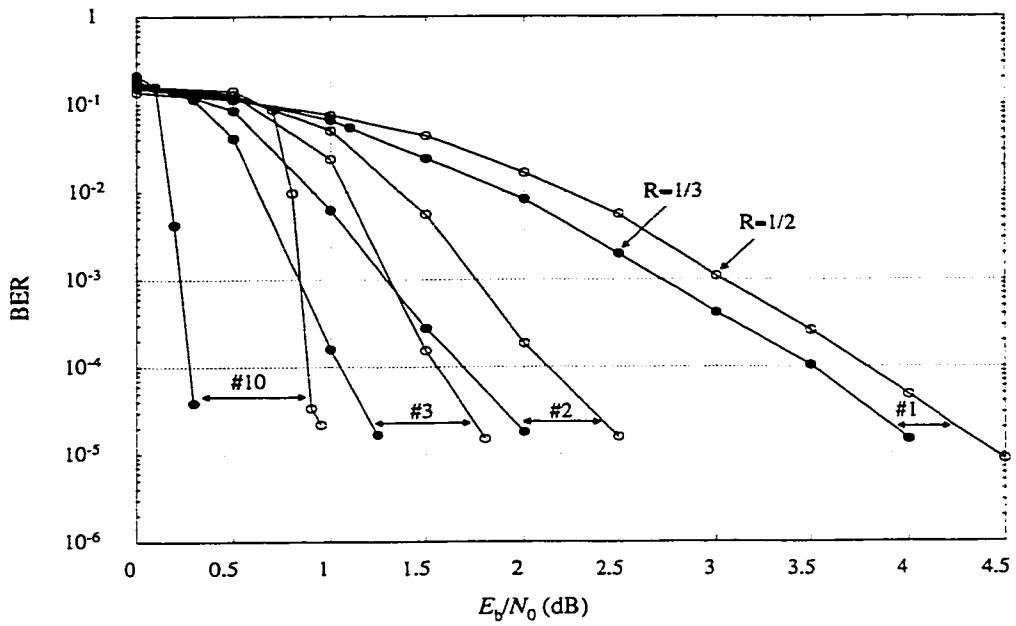


Figure 2.19: Comparison of the for BER for rate 1/2 and 1/3 turbo codes using Berrou's decoding algorithm with $G(37,21)$, Berrou's interleaver, $N = 65536$, and terminating 1st encoder.

2.4 Robertson's Turbo Decoding

As discussed in the previous section, Berrou's turbo decoding algorithm requires an additional variable $z_{s,k}$, which is an independent observation of $x_{s,k}$. Also, the first MAP decoder has three data inputs while the second one has two, and it is necessary to normalize the LLR output or extrinsic information, which requires estimation of the mean and variance.

Robertson's turbo decoding algorithm avoids some of these difficulties. In Robertson's turbo decoding, there is same number of data inputs to each constituent MAP decoder, and it is not necessary to estimate the mean and variance of the LLR values from the decoders. Extrinsic information can be obtained directly from the LLR output of the constituent decoders, and converted to a-priori probabilities to be used by the other constituent decoder [7][23]. Figure 2.20 shows the block diagram of Robertson's iterative turbo decoder. It is clear from this figure that the two constituent decoders now are identical.

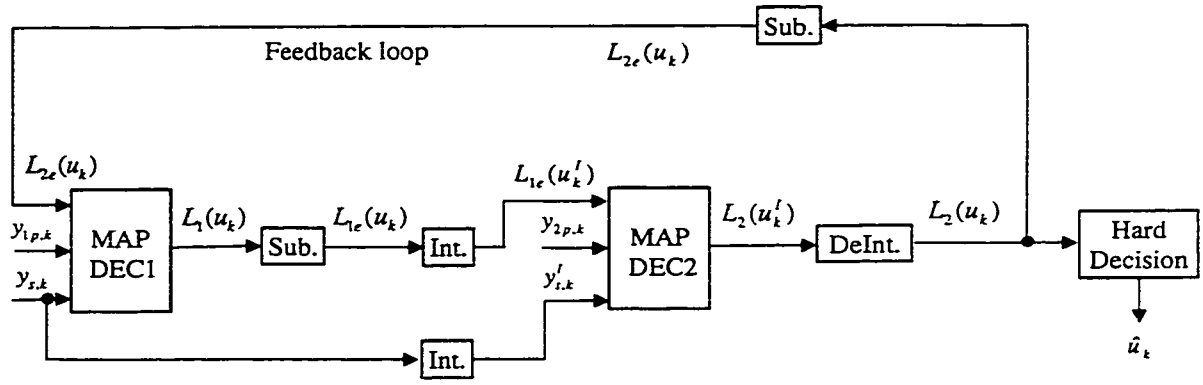


Figure 2.20: Block diagram of Robertson's iterative turbo decoder.

2.4.1 The Key Points of Robertson's Decoding Algorithm

Let the turbo encoder structure and channel model be the same as those described in Section 2.3. The following formulas then apply:

$$\alpha_k(m) = \frac{\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k(m',m)}{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k(m',m)}$$

$$\beta_k(m) = \frac{\sum_{m'=0}^{M-1}\sum_{i=0}^{1} \beta_{k+1}(m') \cdot \gamma_{k+1}^i(m,m')}{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1}\sum_{i=0}^{1} \alpha_k(m) \cdot \gamma_{k+1}^i(m,m')}$$

$$L(u_k) = \log_e \frac{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^i(m',m) \cdot \beta_k(m)}{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^i(m',m) \cdot \beta_k(m)}.$$

A. $\gamma_k^i(m',m)$

In Robertson's decoding algorithm the calculation of the branch transition probabilities $\gamma_k^i(m',m)$ is different from Berrou's calculation. For convenience, the relation for $\gamma_k^i(m',m)$ is again given below.

$$\gamma_k^i(m',m) = \Pr\{Y_k \mid u_k = i; S_k = m; S_{k-1} = m'\} \cdot q_k(u_k \mid m',m) \cdot p_k(m \mid m')$$

$$q_k(u_k \mid m',m) = \Pr\{u_k = i \mid S_k = m; S_{k-1} = m'\}$$

$$p_k(m \mid m') = \Pr\{S_k = m \mid S_{k-1} = m'\}$$

The difference in the calculation of $\gamma_k^i(m',m)$ stems from the method to handle the term $p_k(m \mid m')$. In Berrou's method, the probability $p_k(m \mid m')$ are governed by the input sequence statistics. For the binary input case, if all input sequences are equally likely for $1 \le k \le N_1$, then since there are 2^{-k_0} possible transitions out of each state, $p_k(m \mid m') = 2^{-k_0}$ for each of the transitions. For example, for $1/n_0$ codes, $p_k(m \mid m') = 0.5$, $1 \le k \le N_1$. For $N_1 < k \le N$, the termination period, $p_k(m \mid m') = 1.0$ for the one path that is known to exist out of each state. Therefore, in Berrou's method, this component is fixed. In Robertson's method, however, for $1 \le k \le N$, this component is updated according to the extrinsic information.

The probability $p_k(m \mid m')$ depends directly on the *a-priori* probability of the information bit u_k. In turbo decoding, at the first iteration for decoder 1, $p_k(m \mid m')$ can be set to 1/2 (Assuming $k_0 = 1$, there are two transitions out of each state, one associated the input bit "0" and the other associated one input bit "1". It is straightforward to extend this assumption for $k_0 > 1$.), because decoder 1 has no information regarding the likelihood of the input bit being "0" or "1". After obtaining the LLR output of decoder 1, decoder 2 will have some information regarding which probability is larger at time k. As the number of iterations increases, both decoders will have a better sense of which probability is larger.

By taking the natural logarithm of the likelihood ratio of the *a-priori* probability of information bit u_k [5]:

$$L(u_k) = \log_e\left[\frac{\Pr(u_k = 1)}{\Pr(u_k = 0)}\right] \tag{2.57}$$

and considering:

$$\Pr(u_k = 1) + \Pr(u_k = 0) = 1 \tag{2.58}$$

the following results can be obtained:

$$e^{L(u_k)} = \frac{\Pr(u_k = 1)}{\Pr(u_k = 0)} = \frac{\Pr(u_k = 1)}{1 - \Pr(u_k = 1)} \tag{2.59}$$

$$\Pr(u_k = 1) = \frac{e^{L(u_k)}}{1 + e^{L(u_k)}} \tag{2.60}$$

$$\Pr(u_k = 0) = 1 - \frac{e^{L(u_k)}}{1 + e^{L(u_k)}} = \frac{1}{1 + e^{L(u_k)}}. \tag{2.61}$$

The *a priori* probabilities of information bit u_k given by the previous decoder are used in:

$$\Pr\{S_k = m \mid S_{k-1} = m'\} = \begin{cases} \Pr(u_k = 1), \text{ if } \Pr\{u_k = 1 \mid S_k = m; S_{k-1} = m'\} = 1; \\ \Pr(u_k = 0), \text{ if } \Pr\{u_k = 0 \mid S_k = m; S_{k-1} = m'\} = 1. \end{cases} \tag{2.62}$$

Note that the extrinsic information $L_e(u_k)$, where $L_e(u_k)$ denotes either $L_{1e}(u_k)$ from constituent decoder 1 or $L_{2e}(u_k)$ from constituent decoder 2, is also the soft estimate of the information bit u_k. To develop the iterative decoding algorithm, it is better to use $L_e(u_k)$ than $L(u_k)$ to obtain the estimate of the *a priori* probability. Relation (2.62) above and relation (A2.1) in Appendix 2 yield:

$$p_k(m \mid m') = \begin{cases} \dfrac{e^{L_e(u_k)}}{1 + e^{L_e(u_k)}}, \text{ if } q_k(1 \mid m', m) = 1; \\ \dfrac{1}{1 + e^{L_e(u_k)}}, \text{ if } q_k(0 \mid m', m) = 1. \end{cases} \tag{2.63}$$

Because the two constituent decoders are identical, in the following only the detailed calculation of $L_1(u_k)$ associated with decoder 1 is given.

B. $L_1(u_k)$

$$L_1(u_k) = \log_e \frac{\sum_{m=0}^{M-1} \sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^1(m', m) \cdot \beta_k(m)}{\sum_{m=0}^{M-1} \sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^0(m', m) \cdot \beta_k(m)} \tag{2.64}$$

where

$$\begin{aligned} \gamma_k^1(m', m) &= \gamma_k^i(m', m)\big|_{u_k = i = 1} \\ &= \Pr\{y_{s,k}, y_{1p,k} \mid u_k = 1; S_k = m; S_{k-1} = m'\} \cdot q_k(1 \mid m', m) \cdot p_k(m \mid m') \\ &= \Pr\{y_{s,k} \mid u_k = 1\} \cdot \Pr\{y_{1p,k} \mid u_k = 1; S_k = m; S_{k-1} = m'\} \cdot q_k(1 \mid m', m) \cdot \frac{e^{L_{2e}(u_k)}}{1 + e^{L_{2e}(u_k)}} \\ &= \Pr\{y_{s,k} \mid u_k = 1\} \cdot \frac{e^{L_{2e}(u_k)}}{1 + e^{L_{2e}(u_k)}} \cdot [\Pr\{y_{1p,k} \mid u_k = 1; S_k = m; S_{k-1} = m'\} \cdot q_k(1 \mid m', m)] \\ &= \Pr\{y_{s,k} \mid u_k = 1\} \cdot \frac{e^{L_{2e}(u_k)}}{1 + e^{L_{2e}(u_k)}} \cdot \gamma_k^1(y_{1p,k}, m', m) \end{aligned} \tag{2.65}$$

Similarly:

$$\gamma_k^0(m',m) = \Pr\{y_{s,k} \mid u_k = 0\} \cdot \frac{1}{1+e^{L_{2e}(u_k)}} \cdot \gamma_k^0(y_{1p,k}, m', m) \tag{2.66}$$

Now, introducing relations (2.65) and (2.66) to (2.64) yields:

$$L_1(u_k) = \log_e \frac{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot [\Pr\{y_{s,k} \mid u_k = 1\} \cdot \frac{e^{L_{2e}(u_k)}}{1+e^{L_{2e}(u_k)}} \cdot \gamma_k^1(y_{1p,k}, m', m)] \cdot \beta_k(m)}{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot [\Pr\{y_{s,k} \mid u_k = 0\} \cdot \frac{1}{1+e^{L_{2e}(u_k)}} \cdot \gamma_k^0(y_{1p,k}, m', m)] \cdot \beta_k(m)}$$

$$= \log_e \frac{\Pr\{y_{s,k} \mid u_k = 1\} \cdot \frac{e^{L_{2e}(u_k)}}{1+e^{L_{2e}(u_k)}} \sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^1(y_{1p,k}, m', m) \cdot \beta_k(m)}{\Pr\{y_{s,k} \mid u_k = 0\} \cdot \frac{1}{1+e^{L_{2e}(u_k)}} \sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^0(y_{1p,k}, m', m) \cdot \beta_k(m)}$$

$$= \log_e \frac{\Pr\{y_{s,k} \mid u_k = 1\}}{\Pr\{y_{s,k} \mid u_k = 0\}} + \log_e(e^{L_{2e}(u_k)}) + \log_e \frac{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^1(y_{1p,k}, m', m) \cdot \beta_k(m)}{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^0(y_{1p,k}, m', m) \cdot \beta_k(m)}$$

$$= \frac{2}{\sigma^2} y_{s,k} + L_{2e}(u_k) + \log_e \frac{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^1(y_{1p,k}, m', m) \cdot \beta_k(m)}{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^0(y_{1p,k}, m', m) \cdot \beta_k(m)}$$

$$= \frac{2}{\sigma^2} y_{s,k} + L_{2e}(u_k) + L_{1e}(u_k) \tag{2.67}$$

where

$$L_{1e}(u_k) = \log_e \frac{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^1(y_{1p,k}, m', m) \cdot \beta_k(m)}{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^0(y_{1p,k}, m', m) \cdot \beta_k(m)}.$$

In relation (2.67), the first component is the systematic term (AWGN channel is assumed), the second one is the *a priori* term previously generated by constituent decoder 2, and the last one is the new extrinsic information called $L_{1e}(u_k)$, just generated by the decoder 1.

C. $L_2(u_k)$

Since the structure of constituent decoder 2 is identical to that of constituent decoder 1, and since the systematic input to decoder 2 is interleaved, the following result is obtained:

$$L_2(u_k^I) = \frac{2}{\sigma^2} y_{s,k}^I + L_{1e}(u_k^I) + L_{2e}(u_k^I),$$

where

$$L_{2e}(u_k^I) = \log_e \frac{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^1(y_{2p,k}, m', m) \cdot \beta_k(m)}{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^0(y_{2p,k}, m', m) \cdot \beta_k(m)}. \quad (2.68)$$

After deinterleaving

$$L_2(u_k) = \frac{2}{\sigma^2} y_{s,k} + L_{1e}(u_k) + L_{2e}(u_k). \quad (2.69)$$

Next the interface between the two constituent decoders and Robertson's turbo decoding algorithm is outlined in detail.

2.4.2 Interface between Two MAP Decoders and Robertson's Decoding Algorithm

A. MAP decoder 1

Interface

- Input:
  - $y_{s,k}, y_{1p,k}, L_{2e}(u_k)$, (For the first iteration, $L_{2e}(u_k) = 0$).
  - σ^2, variance of channel.
  - Encoding information.
- Output: $L_1(u_k)$.

Decoding algorithm

- Use the M-BCJR algorithm to calculate $\alpha_k(m)$ and $\beta_k(m)$. Assuming that the first encoder is terminated, these values are initialized as:

$$\alpha_0(0) = 1,\ \alpha_0(m) = 0 \quad \forall m \neq 0,$$

$$\beta_N(0) = 1,\ \beta_N(m) = 0 \quad \forall m \neq 0.$$

- The branch transition probability $\gamma_k^i(m', m)$ can be obtained by

$\gamma_k^i(m', m) = \Pr\{Y_k \mid u_k = i; S_k = m; S_{k-1} = m'\} \cdot q_k(u_k \mid m', m) \cdot p_k(m \mid m')$, where:

- $\Pr\{Y_k \mid u_k = i; S_k = m; S_{k-1} = m'\} = e^{\frac{1}{2\sigma^2}((y_{s,k} - b_s(i,m',m))^2 + (y_{1p,k} - b_p(i,m',m))^2)}$.
- $q_k(u_k \mid m', m)$ is either 0 or 1 depending on the trellis diagram.
- $$p_k(m \mid m') = \begin{cases} \frac{e^{L_{2e}(u_k)}}{1+e^{L_{2e}(u_k)}}, & \text{if } q_k(1 \mid m', m) = 1; \\ \frac{1}{1+e^{L_{2e}(u_k)}}, & \text{if } q_k(0 \mid m', m) = 1. \end{cases}$$

  When iteration = 1, $L_{2e}(u_k) = 0$, so $p_k(m \mid m') = 0.5$ for either input bit value.

- $L_1(u_k) = \log_e \sum_{m=0}^{M-1} \sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^1(m', m) \cdot \beta_k(m) - \log_e \sum_{m=0}^{M-1} \sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^0(m', m) \cdot \beta_k(m)$.

B. Process after decoder 1 and before decoder 2

- Subtract the appropriate quantities from $L_1(u_k)$ to obtain the extrinsic information:

  $$L_{1e}(u_k) = L_1(u_k) - [\frac{2}{\sigma^2} y_{s,k} + L_{2e}(u_k)].$$

- Let $L_{1e}(u_k)$ pass through the interleaver to generate input $L_{1e}(u_k^I)$ for decoder 2.

C. Process before decoder 2

- Let $y_{s,k}$ pass through the interleaver to obtain $y_{s,k}^I$. Note that this term does not appear in Berrou's decoding algorithm.

D. MAP decoder 2

Interface

- Input:
  - $y_{s,k}^I, y_{2p,k}, L_{1e}(u_k^I)$
  - σ^2, variance of channel.
  - Encoding information to set up the trellis diagram.
- Output: $L_2(u_k^I)$.

Decoding algorithm

- Use the M-BCJR algorithm to calculate $\alpha_k(m)$ and $\beta_k(m)$ which, assuming that the second encoder is not terminated, are initialized as follows.

  $$\alpha_0(0) = 1, \ \alpha_0(m) = 0 \ \ \forall m \neq 0$$

$$\beta_N(m) = 1/M \quad \forall m.$$

Note that Robertson's original algorithm did not initialize $\beta_N(m)$ as outlined above [7]. Instead, these values were initialized by the last forward recursion step: $\beta_N(m) = \alpha_N(m)$.

- The branch transition probability $\gamma_k^i(m',m)$ can be obtained by

  $\gamma_k^i(m',m) = \Pr\{Y_k \mid u_k = i; S_k = m; S_{k-1} = m'\} \cdot q_k(u_k \mid m',m) \cdot p_k(m \mid m')$, where:

  - $\Pr\{Y_k \mid u_k = i; S_k = m; S_{k-1} = m'\} = e^{-\frac{1}{2\sigma^2}((y_{s,k}^i - b_s(i,m',m))^2 + (y_{2p,k} - b_p(i,m',m))^2)}$
  - $q_k(u_k \mid m',m)$ is either 0 or 1 dependent on the trellis diagram.
  - $p_k(m \mid m') = \begin{cases} \dfrac{e^{L_{1e}(u_k)}}{1+e^{L_{1e}(u_k)}}, & \text{if } q_k(1 \mid m',m) = 1; \\ \dfrac{1}{1+e^{L_{1e}(u_k)}}, & \text{if } q_k(0 \mid m',m) = 1. \end{cases}$ for any iterations.

- Obtain $L_2(u_k')$

  $$L_2(u_k') = \log_e \sum_{m=0}^{M-1} \sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^1(m',m) \cdot \beta_k(m) - \log_e \sum_{m=0}^{M-1} \sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^0(m',m) \cdot \beta_k(m).$$

E. Process after decoder 2 and before decoder 1

- Obtain $L_2(u_k)$, the deinterleaved version of $L_2(u_k')$.
- If the required iterations are finished, a decision can be made for data bit values in this frame. If not, $L_{2e}(u_k)$ can be obtained by $L_{2e}(u_k) = L_2(u_k) - [\frac{2}{\sigma^2} y_{s,k} + L_{1e}(u_k)]$, and the entire process can be repeated.

Figure 2.21 illuminates the entire decoding process according to Robertson's decoding method. It can be seen that Robertson's decoding algorithm is simpler than Berrou's. Figure 2.22, Figure 2.23 and Figure 2.24 show simulated BER performance using Robertson's decoding algorithm. The simulation conditions are G(37,21), single termination, Berrou's interleaver, frame length $N = 65536$ with code rates 1/2 and 1/3. From comparison of these curves with the results for Berrou's algorithm, it can be seen that at very low signal to noise ratios the performance does not become worse with more iterations. This indicates that Robertson's decoding algorithm is superior to Berrou's algorithm at low signal to noise ratios.

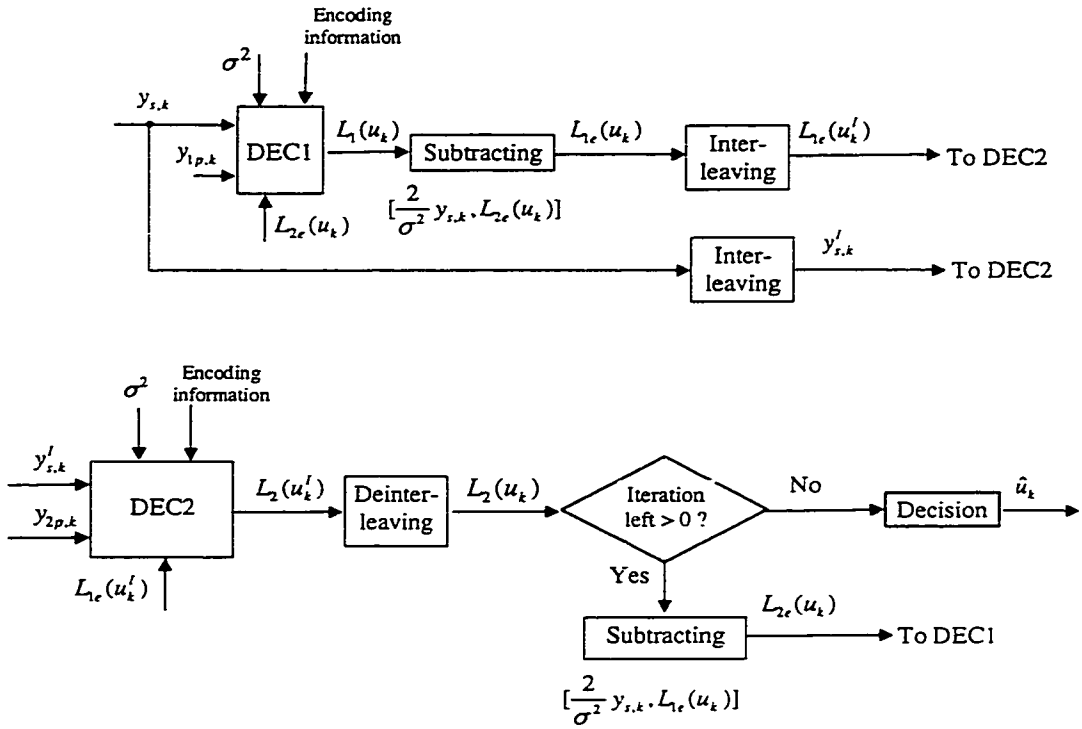


Figure 2.21: Implementation of turbo decoding with Robertson's algorithm.

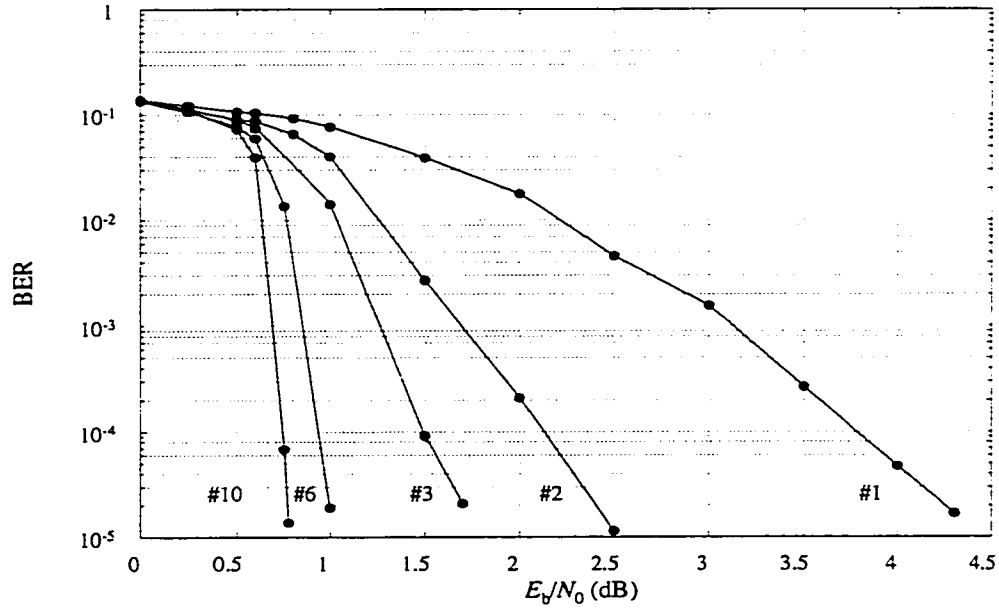


Figure 2.22: BER for transmission through AWGN channel with code rate 1/2, $G(37,21)$, Berrou's interleaver, $N = 65536$, terminating 1st encoder, and using Robertson's decoding algorithm.

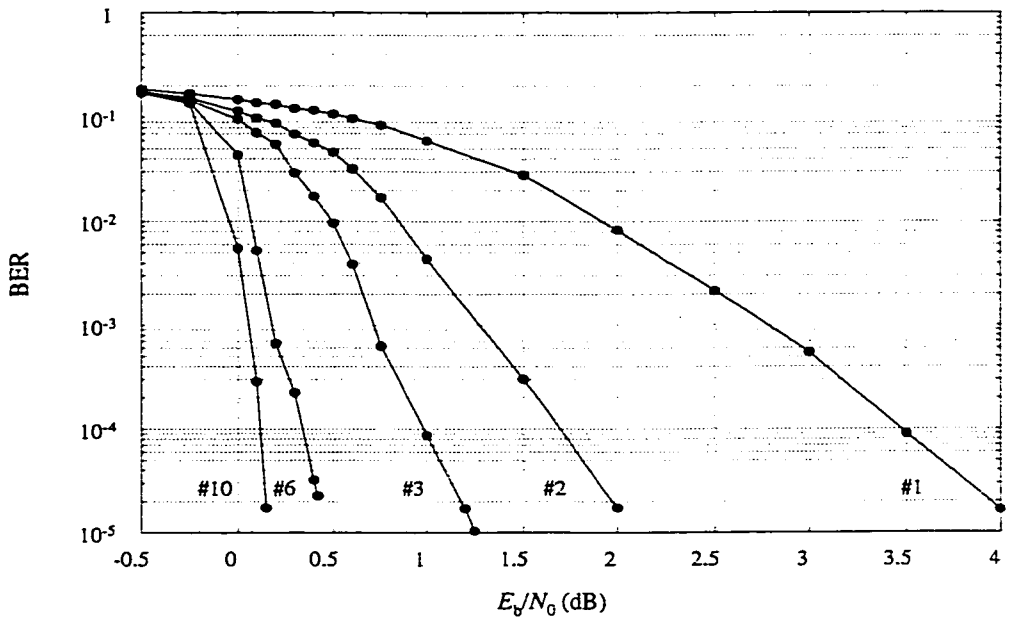


Figure 2.23: BER for transmission through AWGN channel with code rate 1/3, $G(37,21)$, Berrou's interleaver, $N = 65536$, terminating 1st encoder, and using Robertson's decoding algorithm.

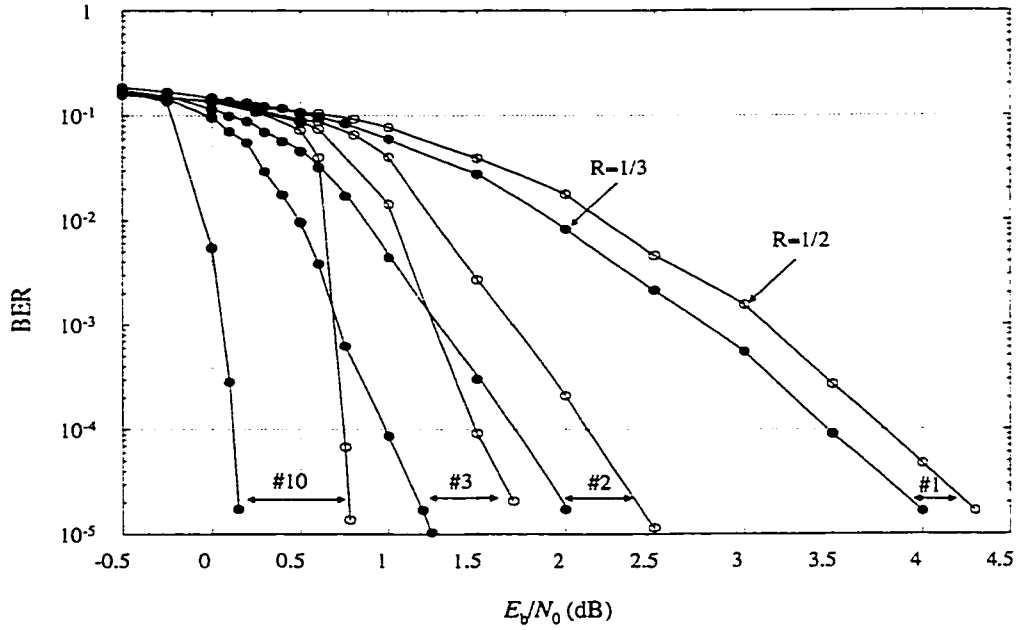


Figure 2.24: Comparison of the BER for rate 1/2 and 1/3 turbo codes using Robertson's decoding algorithm with G(37,21), Berrou's interleaver, $N = 65536$, and terminating 1st encoder.

The BER performance of turbo codes changes with code parameters including encoding generators, memory of shift registers, code rate, interleaver type, and interleaver size. Note that in the decoding algorithm, the channel variance has to be estimated. In the next section, practical methods to estimate symbol energy and the variance of the AWGN channel are introduced.

2.5 Estimating Symbol Energy and Noise Variance

Assume that transmission is with binary phase-shift-keying (BPSK) over an additive white Gaussian noise (AWGN) channel and that coherent demodulation is used. During the k-th symbol interval the received data can be represented as:

$$y_k = \pm\sqrt{E_s} + n_k$$

where E_s is the received symbol energy and n_k is a Gaussian random variable having zero mean and variance $\sigma^2 = N_0/2$, where $N_0/2$ is the two-sided noise spectral density of the channel. The MAP decoding algorithm used during decoding of turbo codes requires knowledge of both received symbol energy and noise variance. The performance of turbo decoding will degrade with poor estimation of signal energy and noise variance [24].

If the energy of the received symbols is known, only estimation of noise variance or E_s/N_0 is required. However, this is rarely the case, even in systems that include an automatic gain control (AGC) at the front-end of the demodulator to compensate for signal attenuation through the channel. The AGC maintains the output (signal plus noise) at a constant level, and the output of the demodulator can be written:

$$y_{k,A} = A(\pm a\sqrt{E_{s,T}} + n_k) = \pm\sqrt{E_{s,A}} + n_{k,A}$$

where a and A denote the channel attenuation and the gain of the AGC respectively, $E_{s,T}$ is the energy with which each symbol is transmitted and therefore is known, and the variance of $n_{k,A}$ is $\sigma_A^2 = A^2\sigma^2$. Both a and A are unknown but are assumed to be constant over the duration of a frame. In this instance, the turbo decoder requires the knowledge of $\sqrt{E_{s,A}}$ and σ_A^2. Even though the mean value of $|y_{k,A}|$ is controlled by these systems, determination of $\sqrt{E_{s,A}}$ is not straightforward.

Simple methods for estimating both the received symbol energy and noise variance are proposed in this section. For convenience, these quantities are denoted as E_s and σ^2, where it is understood they represent $E_{s,A}$ and σ_A^2 in systems with AGC. Summers and Wilson [24] have reported a technique to estimate the signal-to-noise ratio (SNR), E_b/N_0. Their approach can be extended to estimate both E_s and σ^2 over a frame of received data. Following the introduction

of the conventional noise variance estimate, the method for estimating SNR is reviewed, and the new methods for estimating E_s and σ^2 are then discussed.

2.5.1 Conventional Noise Variance Estimate

The variance of a random variable w can be written:

$$\mathrm{Var}(w) = E[w^2] - \left(E[w]\right)^2 . \tag{2.70}$$

The distribution of y_k is the sum of two Gaussian distributions with means $\pm\sqrt{E_s}$ and variance σ^2, scaled by the factor one-half. Note that because the distribution of y_k is symmetrical about zero, the mean of y_k is zero and the variance of y_k is $E_s + \sigma^2$ (see relation (2.78)). Due to the fact that symbols in the demodulated sequence have mean values $\pm\sqrt{E_s}$, the simplest method of estimating the channel variance is by considering the absolute value of y_k. This yields a channel model that approximates a Gaussian channel with mean $\sqrt{E_s}$ and variance σ^2. From Figure A2.1, it can be seen that if the variance of the AWGN channel is very small, the overlap of the two Gaussian curves is small, and therefore this approximation is acceptable. However, with larger channel variance, the error introduced by this approximation is large.

This approximation can be expressed as:

$$\sigma^2 \approx \mathrm{Var}(|\, y_k \,|) = E[y_k^2] - \left(E[|\, y_k \,|]\right)^2 . \tag{2.71}$$

To develop an estimate for $\mathrm{Var}(|\, y_k \,|)$, the expectations are replaced with time averages, denoted by $\langle\cdot\rangle$, over the sequence length N', where $N' = N/R$. This yields:

$$\sigma^2 \approx \mathrm{Var}(|\, y_k \,|) \approx \frac{1}{N'}\sum_{k=1}^{N'} y_k^2 - \left(\frac{1}{N'}\sum_{k=1}^{N'} |\, y_k \,|\right)^2 = \langle y_k^2 \rangle - \langle |\, y_k \,| \rangle^2 . \tag{2.72}$$

In [22], Berrou used this approach to estimate the variance of the meta-channel (see relations 2.46 and 2.47).

Reed [25] also investigated this approach and improved it by assuming that the channel is stationary, and by incorporating the previously estimated values into the calculation. In his approach, for the f-th frame of data $\{y_{k,f}\}$ provided at the turbo decoder, the mean of $|\, y_{k,f} \,|$ is estimated by its time average:

$$\hat{m}_f = \langle | y_{k,f} | \rangle . \tag{2.73}$$

The variance of the AWGN channel for the f-th frame of data is also estimated as:

$$\hat{\sigma}_f^2 = \langle y_{k,f}^2 \rangle - \hat{m}_f^2 . \tag{2.74}$$

For the subsequent frame of data, the variance is calculated frame-wise as in relation (2.74) and then averaged using the single pole auto-recursive formula:

$$\hat{\sigma}_{f+1}^2 = \xi \hat{\sigma}_{f+1}^2 + (1-\xi)\hat{\sigma}_f^2 \tag{2.75}$$

where $0 < \xi \leq 1$. The value of ξ determines the memory of the variances of the previous frames. If ξ is 1, the estimator is self-contained on a frame level. The mean is also averaged using:

$$\hat{m}_{f+1} = \xi \hat{m}_{f+1} + (1-\xi)\hat{m}_f . \tag{2.76}$$

For subsequent frames, this method provides a less variable estimate if the frame size is very short. For example, if ξ is fixed for each frame, the variance for the 4-th frame is $\hat{\sigma}_4^2 = \xi\hat{\sigma}_4^2 + \xi(1-\xi)\hat{\sigma}_3^2 + \xi(1-\xi)^2\hat{\sigma}_2^2 + (1-\xi)^3\hat{\sigma}_1^2$. Also in this example, let $\xi = 0.2$, and then $\hat{\sigma}_4^2 = 0.2\hat{\sigma}_4^2 + 0.16\hat{\sigma}_3^2 + 0.128\hat{\sigma}_2^2 + 0.512\hat{\sigma}_1^2$. Note that when ξ is fixed for each frame, the coefficients (i.e. 0.2, 0.16, 0.128, 0.516) that weight the estimated variance of each frame are not identical. It can be shown that the following modified single pole auto-recursive formula provides identical coefficients:

$$\hat{\sigma}_f = \xi_f \hat{\sigma}_f^2 + (1-\xi_f)\hat{\sigma}_{f-1}^2$$

$$\xi_f = \frac{1}{f}, \quad f \geq 2 \tag{2.77}$$

where f is the frame number. Relation (2.77) yields:

$$f = 2, \quad \hat{\sigma}_2^2 = \frac{1}{2}\hat{\sigma}_2^2 + \frac{1}{2}\hat{\sigma}_1^2 = \frac{1}{2}(\hat{\sigma}_2^2 + \hat{\sigma}_1^2)$$

$$f = 3, \quad \hat{\sigma}_3^2 = \frac{1}{3}\hat{\sigma}_3^2 + \frac{2}{3}\hat{\sigma}_2^2 = \frac{1}{3}(\hat{\sigma}_3^2 + \hat{\sigma}_2^2 + \hat{\sigma}_1^2)$$

$$f = 4, \quad \hat{\sigma}_4^2 = \frac{1}{4}\hat{\sigma}_4^2 + \frac{3}{4}\hat{\sigma}_3^2 = \frac{1}{4}(\hat{\sigma}_4^2 + \hat{\sigma}_3^2 + \hat{\sigma}_2^2 + \hat{\sigma}_1^2)$$

$$\vdots$$

Therefore, in order to obtain the average over several variances, only the variance from the previous frame is stored. Note that the frame number has to be counted for this modified method.

2.5.2 SNR Estimate [24]

Summers and Wilson proposed an approach for estimating E_b/N_0 using a second-order polynomial approximation. Consider the mean of the absolute value of y_k. It is not difficult to show that (see Appendix 3 for a complete derivation):

$$E[y_k^2] = E_s + \sigma^2 . \tag{2.78}$$

$$E[|\, y_k \,|] = \sigma\sqrt{\frac{2}{\pi}}e^{-(E_s/2\sigma^2)} + \sqrt{E_s}\left[\operatorname{erf}\left(\sqrt{\frac{E_s}{2\sigma^2}}\right)\right] \tag{2.79}$$

where $\operatorname{erf}(x) = 1 - \frac{2}{\sqrt{\pi}}\int_x^\infty e^{-u^2}du$ is the error function. From relations (2.78) and (2.79), the following ratio can then be constructed:

$$\begin{aligned}\frac{E[y_k^2]}{(E[|\, y_k \,|])^2} &= \frac{E_s + \sigma^2}{\left(\sigma\sqrt{\frac{2}{\pi}}e^{-(E_s/2\sigma^2)} + \sqrt{E_s}\left[\operatorname{erf}\left(\sqrt{\frac{E_s}{2\sigma^2}}\right)\right]\right)^2} \\ &= \frac{1 + \frac{E_s}{\sigma^2}}{\left(\sqrt{\frac{2}{\pi}}e^{-(E_s/2\sigma^2)} + \sqrt{\frac{E_s}{\sigma^2}}\left[\operatorname{erf}\left(\sqrt{\frac{E_s}{2\sigma^2}}\right)\right]\right)^2} \\ &= q\left(\frac{E_s}{\sigma^2}\right) = q(r) = f(r(\mathrm{dB}))\end{aligned}$$

where $r = \frac{E_s}{\sigma^2} = \frac{2E_s}{N_0} = \frac{2RE_b}{N_0}$, $r(\mathrm{dB}) = 10\log_{10} r$. Note that in $q(r)$, r is not in dB, but in $f(r(\mathrm{dB}))$, r is in dB. Define the variable z to replace the ratio of the expectations:

$$z = E[y_k^2]/(E[|\, y_k \,|])^2 .$$

Thus, $z = f(r(\mathrm{dB}))$ provides a means to estimate $r(\mathrm{dB})$. However, the complexity of the function $f(r(\mathrm{dB}))$ precludes a closed-form solution for $r(\mathrm{dB}) = f^{-1}(z)$. This difficulty may be alleviated by first evaluating the function on a point-wise basis over the range of interest for $r(\mathrm{dB})$, and then using a simple polynomial function to approximate the relationship between $r(\mathrm{dB})$ and z.

For $E_b / N_0 = 0$ through 6 dB, and code rates 1/2 and 1/3, Summers and Wilson obtained a simple second-order polynomial to approximate the function $r(\text{dB}) = f^{-1}(z)$:

$$r(\text{dB}) = -34.0516z^2 + 65.9548z - 23.6184 \ . \tag{2.80}$$

To develop an estimate for z, expectations are replaced by the corresponding time average based on one frame of data:

$$z \approx \langle y_k^2 \rangle / \langle | y_k | \rangle^2 \ . \tag{2.81}$$

Therefore, to obtain an estimate of E_b / N_0 from each frame, calculate z using (2.81), compute $r(\text{dB})$ using (2.80), and finally obtain the estimate of SNR from:

$$\frac{E_b}{N_0}(\text{dB}) = 10\log_{10}\frac{1}{2R} + r\,(\text{dB}). \tag{2.82}$$

Summers and Wilson focus attention on estimating E_b / N_0. Their approach can also be used to estimate the channel variance, if E_s is known. From $r(\text{dB})$, r is obtained by:

$$r = 10^{\frac{r(\text{dB})}{10}} \tag{2.83}$$

and then the channel variance can be estimated as $\hat{\sigma}^2 = \frac{E_s}{r}$. However, if E_s is known, using relation (2.78) to estimate the channel variance is simpler.

If E_s is unknown, the approach of Summers and Wilson can also be extended to estimate the symbol energy and channel variance as described in the next subsection.

2.5.3 New Approaches for Estimating Symbol Energy and Noise Variance

To obtain estimates for $\sqrt{E_s}$ and σ^2, equations (2.78) and (2.79) should be considered together. Due to the complexity of equation (2.79), it appears that a simple direct solution for these two values does not exist. However, $\sqrt{E_s}$ and σ^2 can be estimated by applying Summers and Wilson's result that $\frac{E_s}{\sigma^2} = r$ in either equation (2.78) or equation (2.79).

(1) Approach 1

Combining $\frac{E_s}{\sigma^2} = r$ with equation (2.78) yields:

$$\sigma^2 = \frac{\langle y_k^2 \rangle}{r+1} \tag{2.84}$$

$$\sqrt{E_s} = \sqrt{\frac{r}{r+1}\langle y_k^2 \rangle} \tag{2.85}$$

where $E[y_k^2]$ in equation (2.78) is replaced by the time average $\langle y_k^2 \rangle$. Therefore, to obtain estimates of $\sqrt{E_s}$ and σ^2 for each frame, calculate z using (2.81), compute r(dB) using (2.80), obtain r using (2.83), and then determine these two values using (2.84) and (2.85).

(2) Approach 2

Equation (2.79) yields:

$$\begin{aligned}
\frac{E[|\, y_k \,|]}{\sqrt{E_s}} &= \frac{\sigma}{\sqrt{E_s}}\sqrt{\frac{2}{\pi}}e^{-(E_s/2\sigma^2)} + \operatorname{erf}\left(\sqrt{\frac{E_s}{2\sigma^2}}\right) \\
&= \frac{\sqrt{\frac{N_0}{2}}}{\sqrt{E_s}}\sqrt{\frac{2}{\pi}}e^{-(E_s/N_0)} + \operatorname{erf}\left(\sqrt{\frac{E_s}{N_0}}\right) \\
&= \frac{1}{\sqrt{\pi\frac{E_s}{N_0}}}e^{-(E_s/N_0)} + \operatorname{erf}\left(\sqrt{\frac{E_s}{N_0}}\right) \\
&= \frac{1}{\sqrt{\pi\frac{r}{2}}}e^{-(r/2)} + \operatorname{erf}\left(\sqrt{\frac{r}{2}}\right) \\
&= h\left(\frac{r}{2}\right) = g\left(r(\text{dB})\right)
\end{aligned} \tag{2.86}$$

Because $g(r(\text{dB}))$ is somewhat complicated, another fitting function can be used to simplify the calculation. It is found that the relationship between $g(r(\text{dB}))$ and r(dB) can be closely approximated by an exponential function. Assume that the range of interest for E_b/N_0 is 0 through 6 dB, and consider code rates with 1/2 and 1/3. The range of $2E_s/N_0$ is dependent on

the range of E_b / N_0 and the code rate. Given E_b / N_0 of 0 to 6 dB, since $2E_s / N_0 = 2RE_b / N_0$, for the code rate 1/2, $2E_s / N_0$ is 0 to 6 dB, and for the code rate 1/3, $2E_s / N_0$ is -1.77 to 4.23 dB. Considering these two ranges for $2E_s / N_0$ together, the range for $2E_s / N_0$ of -2 through 6 dB is considered here. As shown in Figure 2.25, the following exponential function $f_g(r(\text{dB}))$ is found to closely approximate the function $g(r(\text{dB}))$:

$$f_g(r(\text{dB})) = 1 + 0.3048e^{-0.35(r(\text{dB})+2)}.$$

Using this fitting function instead of $g(r(\text{dB}))$ in relation (2.87) yields

$$\sqrt{\hat{E}_s} = \frac{\langle | y_k | \rangle}{f_g(r(\text{dB}))} \tag{2.87}$$

where $r(\text{dB})$ is obtained from (2.80). The noise variance can then be estimated from relation (2.78) as:

$$\hat{\sigma}^2 = \langle y_k^2 \rangle - \hat{E}_s. \tag{2.88}$$

Therefore, for each frame of received data, calculate z based on (2.81), estimate $r(\text{dB})$ using (2.80), and then compute the estimated values of the symbol energy and the variance σ^2 using relations (2.87) and (2.88).

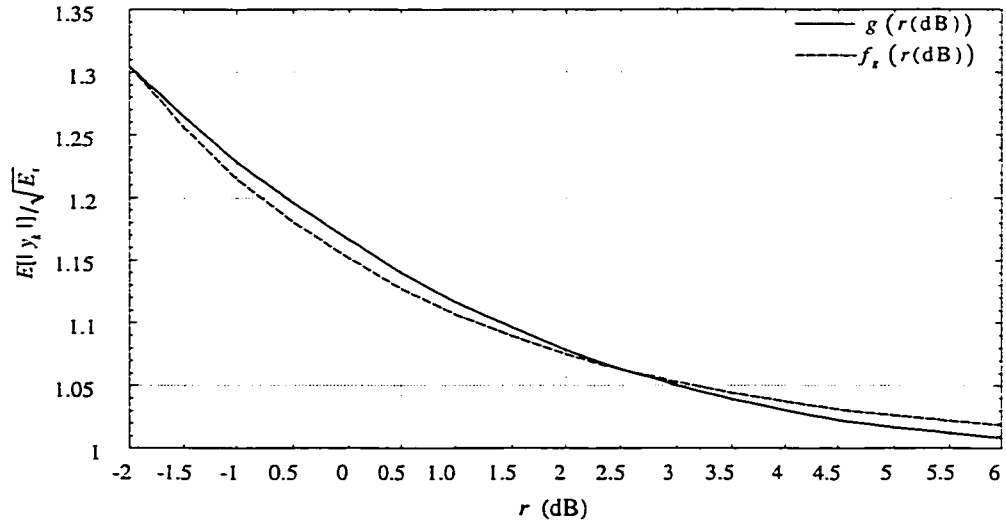


Figure 2.25: Relationship between ratio $E[|y_k|]/\sqrt{E_s}$ and $r(\text{dB})$ using the true relationship $g(r(\text{dB}))$ and the exponential approximation $f_g(r(\text{dB}))$.

Figure 2.26 shows simulation results for estimated values of $\sqrt{E_s}$ and σ^2 versus SNR with a frame size of 1024 and code rate 1/3. The estimation results were averaged over 10 000 frames. In this figure, curves of the true $\sqrt{E_s}$ and true σ^2 are also given for purpose of comparison. Clearly, the estimation is very accurate for both approaches, and on average, approach 2 is slightly more accurate than approach 1. Also it was found that these results were consistent from one frame to the next. The standard deviations for the simulation results were 0.034 for $\sqrt{\hat{E}_s}$ and 0.051 for $\hat{\sigma}^2$ (approach 1), and 0.035 for $\sqrt{\hat{E}_s}$ and 0.053 for $\hat{\sigma}^2$ (approach 2).

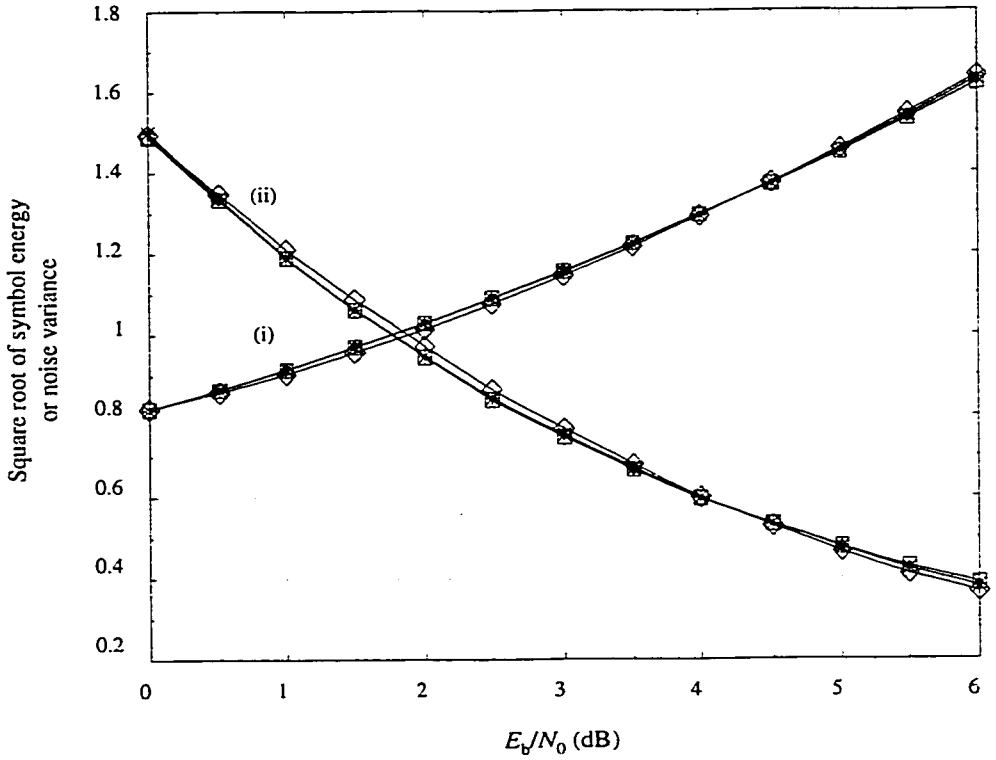


Figure 2.26: True and estimated values, square root of symbol energy and noise variance.

(i) Square root of symbol energy with fixed noise variance 1.0
(ii) Noise variance with fixed symbol energy 1.0
✳ true values
◇ estimated values with approach 1
□ estimated values with approach 2

Figure 2.27 compares the performance of the conventional approach and approach 2 for variance estimate. The estimated results are obtained from a single frame of received data where

the frame size is 65536 and the code rate is 1/2. With fixed symbol energy $E_s = 1$, E_b / N_0 is varied from 0 dB to 6 dB, and the variance changes accordingly. For the conventional method, ξ is set to one. Clearly, the performance using approach 2 is very good for this large frame, while the variance is underestimated by the conventional method over the range of interest of E_b / N_0. At low E_b / N_0, the error in estimation is especially large. However, as $E_b / N_0 > 5$ dB, the conventional method provides an estimate that is close to the true variance.

Note that Berrou used the conventional method to estimate the variance of meta-channel. The simulation results plotted in Figure 2.17 and Figure 2.18 show that at very low E_b / N_0, the BER increases as the number of iterations increases using Berrou's decoding algorithm. To solve this problem, Berrou suggested that the extrinsic information z_k be divided by a value larger than one. Doing this, equivalently, forces the estimated variance to become larger, closer to the true variance.

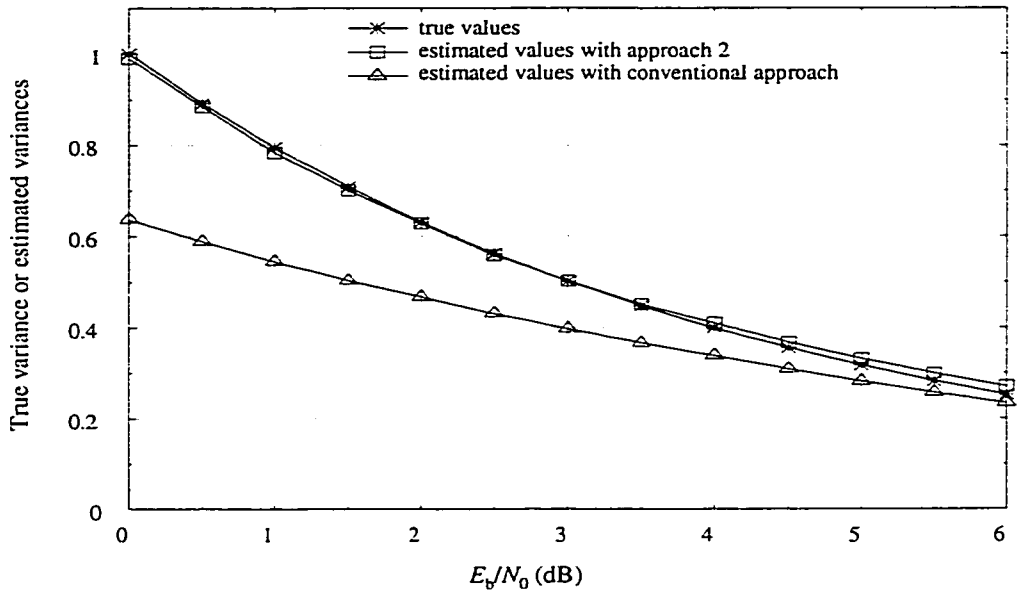


Figure 2.27: The estimated variances or true variances vs. E_b/N_0 for a single frame of size 65536.

Approach 2 has very good performance for estimating the symbol energy and variance of the AWGN channel and it is simple to implement in practical turbo decoding as well as other applications. Therefore, it is an attractive approach for estimating these two values.

2.6 Average Performance of Turbo Codes

Not all error patterns can be corrected by a decoder. The error correction capability of a code depends on its geometry. The minimum distance and weight distribution of a code are used to estimate its performance.

2.6.1 Weight and Distance of Binary Vectors

The Hamming distance between two binary code vectors $\overline{u}$ and $\overline{v}$, denoted $d(\overline{u},\overline{v})$, is defined to be the number of bits in which they differ. For example, if $\overline{u}=10001000$ and $\overline{v}=10101010$, $d(\overline{u},\overline{v})=2$. In all linear codes, the element-wise modulo-2 sum of vectors $\overline{u}$ and $\overline{v}$ yields another code vector. In this example, $\overline{z}=\overline{u}+\overline{v}=00100010$ whose weight is two. Therefore, the Hamming distance between two code vectors is equal to the Hamming weight of their sum, where the Hamming weight $w(\overline{z})$ of a vector $\overline{z}$ is defined to be the number of ones in the vector. Also, the Hamming weight of a code vector is equal to its Hamming distance from the all-zero vector.

The minimum distance, $d_{\min}$, between all pairs of code vectors provides a measure of the code's minimum capability for error correction. Because of the relationship between the distance and weight of the code, it is convenient to examine the weight of each code vector (excluding the all-zero vector) in the code subspace instead of the distance between code vectors; the minimum weight corresponds to the minimum distance. Equivalently, $d_{\min}$ corresponds to the minimum distance between the all-zero code vector and all other code vectors.

The term minimum distance is from the world of block coding. In convolutional coding, because the codes are also linear, there is no loss in generality in defining $d_{\min}$ to be the minimum distance between each of the codeword sequences and the all-zero sequence. In these codes, the Viterbi or MAP decoders use the entire received codeword to decode a single bit. Assuming that the all-zero input sequence is transmitted, paths of interest are paths of any length that diverge from and remerge onto the zero state and do not return to the zero state anywhere in between. Sequences corresponding to these paths, called self-terminating sequences, can be regarded as error events. The number of ones in the sequence denotes the number of errors in that error event. If information about the weight distribution of these sequences is available, a bound on the performance of a code can be estimated.

For convolutional codes, the term minimum free distance, denoted d_{free}, is usually used instead of minimum distance. The minimum free distance is the distance between all code vectors in the set of all arbitrarily long paths that diverge from and remerge onto the all-zero sequence. To obtain d_{free}, the minimum weight of valid code sequences other than the all-zero sequence must be evaluated.

2.6.2 Performance Evaluation Based on Weight Enumerating Function

The weight distribution of an (n_0, k_0) block code C is a series of coefficients $B_0, B_1, ..., B_i, ..., B_{n_0}$, where B_i is the number of code words in C of weight i [3, pp. 90]. The weight distribution $(B_0, B_1, ..., B_i, ..., B_{n_0})$ for a code is often written as a polynomial:

$$B^C(H) = \sum_{i=0}^{n_0} B_i H^i . \tag{2.89}$$

This representation is often called the weight enumerating function (WEF).

The performance analysis of turbo codes introduced here is limited to turbo encoder structures with two identical constituent codes at rate 1/2, terminating both constituent encoders, and overall code rate 1/3 [9]. For the more general case, refer to [29]. A turbo code can be regarded as a long block code. For N much larger than the memory of the constituent codes, the performance of a turbo code is almost identical to that of the equivalent parallel concatenated block code (PCBC) C_P [9]. For constituent codes of memory ν, frame size N, and joint termination, the size of equivalent block code is $(3N, N - 2\nu)$.

The random interleaver in turbo codes causes difficulty in analyzing the performance of these codes. Benedetto and Montorsi evaluated the average performance of turbo codes by separating weight contributions from the input bits and parity check bits, and introducing an abstract interleaver of length N called the uniform interleaver, defined as a probabilistic device that maps a given input sequence of length N and weight w into all distinct $\binom{N}{w}$ permutations of it with equal probability $1/\binom{N}{w}$ [9], [26] – [29].

For turbo codes, the two sequences that enter the two encoders have the same weight, because the interleaver only changes the order of input sequence, but does not change the weight of the input sequence. As with other error correcting codes, the WEF of turbo codes is needed to

evaluate decoding performance. Define a special weight enumerating function of the constituent codes called the input redundancy weight enumerating function (IRWEF) [27]:

$$A^{C_P}(W,Z) \triangleq \sum_{i,j} A_{i,j}^{C_P} W^i Z^j \tag{2.90}$$

where $A_{i,j}^{C_P}$ denotes the number of codewords generated by an input information word of weight i whose parity check bits have weight j, so that the overall weight is $i+j$. In this definition, $i=0,1,2,...$, but j and $A_{i,j}^{C_P}$ are dependent on i. The IRWEF explicitly separates the weight contributions from the information bits and the parity check bits.

Consider the conditional weight enumerating function $A_w^{C_P}(Z)$ of the equivalent PCBC where parity check bits have weight j corresponding to the input sequence of weight w. From the IRWEF, it is straightforward to obtain:

$$A_w^{C_P}(Z) = \sum_j A_{w,j}^{C_P} Z^j = \frac{1}{w!} \frac{\partial^w A^{C_P}(W,Z)}{\partial W^w}\bigg|_{W=0}$$

Based on this result, $A^{C_P}(W,Z)$ can also be written as:

$$A^{C_P}(W,Z) = \sum_w W^w A_w^{C_P}(Z)\,. \tag{2.91}$$

The $A_w^{C_P}(Z)$ can be used with the union bound to compute an upper bound to the bit error probability $P_b(e)$ for ML soft decoding of the code over a channel with AWGN in the form [9]:

$$P_b(e) \le \sum_{w=1}^{N} \frac{w}{N} W^w A_w^{C_P}(Z)\bigg|_{W=Z=e^{-RE_b/N_0}} \tag{2.92}$$

Let $A_w^C(Z)$ denote the conditional weight enumerating function of the two $(2N, N-\nu)$ constituent codes. It is straightforward to evaluate the conditional weight enumerating functions of the equivalent PCBC using these constituent codes if the uniform interleaver is also used. In this instance, the conditional weight enumerating functions are the product, suitably normalized, of the two conditional weight enumerating functions of the constituent codes. The relationship between these two terms is:

$$A_w^{C_P}(Z) = \frac{\left[A_w^C(Z)\right]^2}{\binom{N}{w}} \tag{2.93}$$

where $A_w^C(Z)$ can be derived from the standard transfer function of the convolutional codes [28].

Note that the codewords of the constituent codes are concatenations of error events. Let $A_{w,n}(Z)=\sum_j A_{wjn}Z^j$ denote the parity check enumerating function of the sequences of the convolutional code generated by concatenating n events with total information weight w. A_{wjn} is the number of codewords with parity check weight j and the number of concatenated error events n given the information bit weight w. For $N \gg \nu$, the $A_w^C(Z)$ can be approximated by [9]:

$$A_w^C(Z) \approx \sum_{n=1}^{n_{\max}} \binom{N}{n} A_{w,n}(Z) \tag{2.94}$$

where $n_{\max}$, the largest number of error events generated by a weight w information sequence, is a function of w which depends on the geometry of the encoder.

For large N, inserting (2.94) into (2.93) and using the asymptotic approximation for the binomial coefficient $\binom{N}{n} \approx \frac{N^n}{n!}$ yields [9]:

$$A_w^{C_P}(Z) \approx \frac{w!}{(n_{\max}!)^2} N^{2n_{\max}-w} \left[A_{w,n_{\max}}(Z)\right]^2. \tag{2.95}$$

Inserting (2.95) into (2.92), the following asymptotic (in N) bound for the bit error probability is obtained:

$$P_b(e) \lesssim \sum_{w=w_{\min}}^{N} w \cdot \frac{w!}{(n_{\max}!)^2} \cdot N^{2n_{\max}-w-1} \cdot W^w \cdot \left[A_{w,n_{\max}}(Z)\right]^2 \Big|_{W=Z=e^{-RE_b/N_0}} \tag{2.96}$$

where the sum for w starts from $w_{\min}$, which denotes the minimum information weight in the error events of the constituent code.

For non-recursive convolutional constituent encoders, using $w_{\min}=1$, $n_{\max}=w$ and $A_{w,w}(Z)=\left(A_{1,1}(Z)\right)^w$ [9] in (2.96) yields:

$$P_b(e) \lesssim \sum_{w=1}^{N} \frac{N^{w-1}}{(w-1)!} \cdot W^w \cdot \left[A_{1,1}(Z)\right]^{2w} \Big|_{W=Z=e^{-RE_b/N_0}}. \tag{2.97}$$

From (2.97), it can be seen that, in the most likely error event ($w=1$), the bit error probability is independent of N, so that no interleaver gain is obtained in this case.

On the other hand, in the case of a recursive constituent code, $w_{\min}=2$ and $n_{\max}=\lfloor w/2 \rfloor$, where $\lfloor x \rfloor$ denotes the integer part of x. By considering the terms in the sum in (2.96) with odd and even w separately, it can be shown that the terms with odd w depend on N as N^{-2}, and the terms with even w depend on N as N^{-1}, so that the terms with odd w are negligible [9]. Also note that when $w=2$, $n_{\max}=1$ and $A_{2,1}(Z)=\dfrac{Z^{z_{\min}}}{1-Z^{z_{\min}-2}}$, where $z_{\min}$ is the minimum weight of the parity check bits in error events generated by information sequences with $w=2$ [9]. Using these results in (2.96) yields the following asymptotic expression of the upper bound:

$$P_b(e) \lesssim \sum_{k=1}^{\lfloor N/2 \rfloor} 2k \cdot \binom{2k}{k} \cdot N^{-1} \cdot \frac{(H^{2+2z_{\min}})^k}{(1-H^{z_{\min}-2})^{2k}}\Bigg|_{H=e^{-RE_b/N_0}} \tag{2.98}$$

where $w=2k$ and $W=Z=H$.

Relation (2.98) leads to two important conclusions. First, the interleaver gain for the bit error rate probability of turbo codes employing recursive constituent codes is $1/N$. Second, $z_{\min}$ of a constituent code influences the performance of turbo codes. It can be seen that increasing $z_{\min}$ can decrease $P_b(e)$.

Define the lowest exponent of H in the numerator in (2.98) as the effective free distance of the turbo codes:

$$d_{free,eff}=2+2z_{\min} \ . \tag{2.99}$$

This is the minimum weight of coded sequences generated by input sequences of weight 2. It plays a role similar to that of the free distance for convolutional codes [9]. From the above discussions, it can be seen that the RSC codes should be chosen to maximize $z_{\min}$ and, hence, $d_{free,eff}$. For rate $1/2$ RSC constituent codes with memory ν, let its feedback polynomial be $d(D)$ and its feed forward polynomial be $n(D)$. It is has been suggested to choose $d(D)$ as a primitive polynomial [9]. An irreducible polynomial $p(D) \in GF(2)[D]$ of degree m is said to be primitive if the smallest positive integer q for which $p(D)$ divides D^q+1 is $q=2^m-1$ [3, p.

40]. If $d(D)$ is any primitive polynomial of degree ν and $n(D)$ is any monic polynomial of degree ν except $d(D)$, the following value for $z_{\min}$ can be obtained [9]:

$$z_{\min} = 2^{\nu-1} + 2 . \tag{2.100}$$

The above results are obtained based on the uniform interleaver. For practical interleaver design, one of the considerations is to avoid generating low weight output sequences simultaneously for both constituent encoders with weight 2 input sequences. For turbo codes where the feedback polynomial $d(D)$ is chosen to be a primitive polynomial of degree ν, $d(D)$ divides the weight 2 input sequence $D^{2^{\nu}-1}+1$. Moreover, $d(D)$ also divides $D^{k(2^{\nu}-1)}+1$ [9], where k is a positive integer. For example, the feedback polynomial $d(D) = D^2 + D + 1$ ($\nu = 2$) divides the weight 2 input sequence $i(D) = D^3 + 1$.

Polynomials are often represented as binary sequences (e.g., $D^3 + 1 \leftrightarrow 1001$). Let $l = k(2^{\nu} - 1)$ where l denotes the distance that separates the two 1s of weight 2 input sequences. The weight 2 sequences with these characteristics are self-terminating and generate low weight codewords. It would be advantageous to design an interleaver to destroy these patterns and increase the weight of the encoded sequences as much as possible. The self-terminating weight-2 input sequences tend to dominate the performance characteristics of the turbo codes for a moderate bit to high signal-to-noise ratio as the random interleaver size N becomes large [30].

2.7 Interleaver Choices

Interleaving is used during both encoding and decoding of turbo codes, and plays a key role in the pseudo-random nature and consequently the high performance of turbo codes. Thus the study and design of interleavers has become a subject of high priority in this area. The majority of these efforts have been directed at ensuring good randomness, spread-correlation, or free distance properties, while ensuring that the interleaver is simple to generate quickly. Next, descriptions for the following interleavers are presented: Berrou's interleaver, the prime interleaver (PIL), and spread-random (SR) interleaver.

2.7.1 Berrou's Interleaver Scheme [22]

A. Interleaver

In the original proposal for turbo codes, the interleaver consists of a square matrix with $N = L \times L$ entries, where N is the frame length and L must be a power of 2. The input sequence is written to the square matrix row by row, and the output sequence is read out diagonally. This non-uniform interleaving can be described by

$$\begin{aligned} i_r &= (L/2+1)\cdot(i_w + j_w) \quad \text{mod } L \\ \xi &= (i_w + j_w) \quad \text{mod } 8 \\ j_r &= P(\xi)\cdot(j_w + 1) - 1 \quad \text{mod } L \end{aligned} \tag{2.101}$$

where

- i_w and j_w are the addresses of the rows and columns for writing, and i_r and j_r are the addresses of the rows and columns for reading.
- "mod" denotes modular calculation.
- $P(\xi)$ is a number, relatively prime with L, which is a function of the line address $(i_w + j_w)$ mod 8. $P(\xi) = \{17,37,19,29,41,23,13,7), \xi = 0,1,...,7$.

Note that a multiplying factor $(L/2+1)$ is used to prevent neighbouring data written on two consecutive rows from remaining neighbours after reading. Note also that reading is performed diagonally in order to avoid possible relationships between L and the period of puncturing, if any. Next an example is given to show how reading is performed.

Consider the following conditions:

- Frame length $N = 64 = 4^4$.
- $L = 8 = 2^3$.
- Input sequence = {0,1,2,...,63}. Note that integers instead of binary digits ("0" or "1") are used in order to clearly demonstrate how the interleaver works.

i_w, j_w, i_r and j_r take the values from 0 to 7. The input sequence is written in one 8 by 8 matrix $[A] = [A_{i_w, j_w}]$ row by row. It can be observed that there are totally 15 left diagonals. For easy observation, matrix $[A']$ is introduced, which denotes the results obtained after interleaving,

where $[A^I]=[A_{i_r,j_r}]$. The position of the original input sequence in this matrix is demonstrated below. The output sequence is read out from $[A^I]$ row by row.

$$[A]=\begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 \\ 24 & 25 & 26 & 27 & 28 & 29 & 30 & 31 \\ 32 & 33 & 34 & 35 & 36 & 37 & 38 & 39 \\ 40 & 41 & 42 & 43 & 44 & 45 & 46 & \boxed{47} \\ \boxed{48} & 49 & 50 & 51 & 52 & 53 & 54 & 55 \\ 56 & 57 & 58 & 59 & 60 & 61 & 62 & 63 \end{bmatrix}$$

, after interleaving, $[A^I]=$

$$\begin{bmatrix} 0 & 57 & 50 & 43 & 36 & 29 & 22 & 15 \\ 62 & 5 & 12 & 19 & 26 & 33 & 40 & 55 \\ 2 & 45 & 16 & 59 & 38 & 9 & 52 & 31 \\ 14 & 21 & 28 & 35 & 42 & 49 & 56 & 7 \\ 32 & 25 & 18 & 11 & 4 & 61 & 54 & \boxed{47} \\ 44 & 1 & 30 & 51 & 8 & 37 & 58 & 23 \\ 20 & 41 & 6 & 27 & \boxed{48} & 13 & 34 & 63 \\ 60 & 17 & 46 & 3 & 24 & 53 & 10 & 39 \end{bmatrix}.$$

It can be seen that digits on the 0^{th} and 8^{th} left diagonal in $[A]$ are moved to the 0^{th} row in $[A^I]$. Also it can be seen that neighbouring data on two consecutive rows are separated by a significant distance by this interleaver.

B. Interleaving and deinterleaving implementation

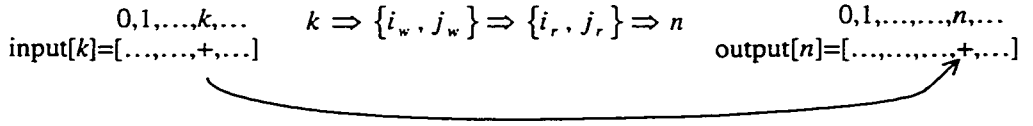
It is not necessary to use a two dimensional matrix for the interleaving operation. A row matrix can be used. The following steps show how to move a digit at position k, denoted by symbol "+", to its correct position n in the output sequence.

- $0,1,2,\ldots,k,\ldots$

  input[k] = [..., ..., +, ...].
- $i_w=\left\lfloor \frac{k}{L} \right\rfloor$ and $j_w = k \bmod L$.
- Calculate (i_r, j_r) from (i_w, j_w).
- $n = i_r \cdot L + j_r$;
- $0,1,2,\ldots,\ldots,n,\ldots$

  output[n] = [... , ..., ..., +,...].

So after interleaving:



There does not appear to be a simple calculation for determining addresses in the deinterleaving process as there is in the interleaving process. However, deinterleaving can be implemented using a mapping address row matrix that can be generated by the interleaving process.

$$\begin{matrix} & 0,1, \ldots, n, \ldots \\ \text{MappingAddress}[n] = [& \ldots, \ldots, k, \ldots]. \end{matrix}$$

The mapping address row matrix stores the original position of symbol "+" of the input sequence, so that when the output sequence and mapping address matrix are known, the symbol "+" can be moved back to its original position in the input sequence, allowing the deinterleaving process to be performed.

2.7.2 Prime Interleaver [31]

Shibutani, Suda and Adachi proposed the prime interleaver (PIL) in 1999. PIL is currently proposed for the UMTS-3GPP channel coding standard (UMTS: Universal Mobile Telecommunications System, and 3GPP: 3rd Generation Partnership Project). The key futures of PIL are low complexity, good pseudo-random pattern, and a wide range of interleaving sizes N, from 240 to 8200.

PIL is based on the construction of a rectangular mother interleaver with L_{row} rows and L_{col} columns, and pruning this mother interleaver if it is larger than the required size. The input sequence is written into the buffer row by row. Permutations are performed on an intra-row and then inter-row basis, and after interleaving the output sequence is read out column by column.

Let i_w, j_w, i_r and j_r denote the same quantities as in the discussion of Berrou's interleaver above. Also let $n = L_{row} \cdot j_r + i_r$ (read column by column) and $k = L_{col} \cdot i_w + j_w$ (write row by row). What is required is the mapping address MappingAddress[n] = k. The mapping process in PIL consists of three stages. The three stages are now discussed in detail.

(1) First, the input sequence is written into one L_{row} by L_{col} buffer memory row by row.

- L_{row} is either 10 or 20. L_{row}=10 for interleaver size N from 481 to 530 (case 1), and L_{row}=20 for every other N (case 2).
- L_{col} is chosen based on the minimum prime number P larger than or equal to N / L_{row}. The minimum P is selected because of the limitation of slight pruning. Heavy pruning can be detrimental to the performance of the interleaver, so only mild pruning is used.

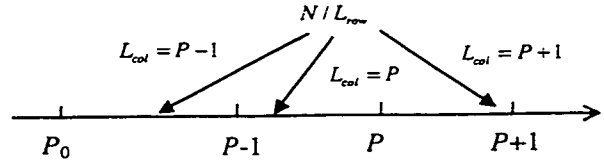
The Table 2.3 shows the allowed primes P and associated primitive roots g_0 (see stage 2).

Table 2.3: Table of P allowed and associated primitive root g_0

P	g_0	P	g_0	P	g_0	P	g_0	P	g_0
13	2	73	5	151	6	233	3	317	2
17	3	79	3	157	5	239	7	331	3
19	2	83	2	163	2	241	7	337	10
23	5	89	3	167	5	251	6	347	2
29	2	97	5	173	2	257	3	349	2
31	3	101	2	179	2	263	5	353	3
37	2	103	5	181	2	269	2	359	7
41	6	107	2	191	19	271	6	367	6
43	3	109	6	193	5	277	5	373	2
47	5	113	3	197	2	281	3	379	2
53	2	127	3	199	3	283	3	383	5
59	2	131	2	211	2	293	2	389	2
61	2	137	3	223	3	307	5	397	5
67	2	139	2	227	2	311	17	401	3
71	7	149	2	229	6	313	10	409	21

1) In case 1 ($L_{row}=10$), $L_{col}=P$, $N/L_{row} \le P$ (minimum prime available).

2) In case 2 ($L_{row}=20$), L_{col} is chosen to be either a prime number or adjacent to a prime number according to the following figure.



In this figure, P_0 and P are two consecutive primes. The value of L_{col} is given by:

- $L_{col}=P-1$, if $P_0 < N/L_{row} \le P-1$.
- $L_{col}=P$, if $P-1 < N/L_{row} \le P$.
- $L_{col}=P+1$, if $P < N/L_{row} \le P+1$.

(2) In the second stage, intra-row permutation is performed as follows:

- Select the primitive root g_0 from Table 2.3. g_0 is the smallest positive integer available such that $g_0^i \bmod P$ does not generate repetitive values, where $i=0,1,2,...P-2$. Note that $g_0^0 \bmod P=1$, and $g_0^{P-1} \bmod P=1$ (if $g_0 < P$), so i is limited to $P-2$. Also note that $g_0^i \bmod P \neq 0$ for all i.

- Construct the base mapping sequence $c(i)$ for intra-row permutation. $c(i)$ is obtained by $c(i) = (g_0 \cdot c(i-1)) \bmod P$, where $i = 0,1,2,...P-2$, and $c(0) = 1$. Here the recursion calculation is actually $g_0^i \bmod P$. The benefit of performing the recursion calculation is to avoid g_0^i being too large to be handled by the program. An advantage of generating the base mapping sequence first is a savings in calculations. The base mapping sequence is actually a permutation of the sequence (1,2,..., P-1).
- Select the integer set $\{q_{i_r}\}(i_r = 1,2,...,L_{row}-1)$ such that g.c.d. $\{q_{i_r}, P-1\} = 1$, where g.c.d. is the greatest common divisor and q_0 is set to 1. An easy way to form the set $\{q_{i_r}\}$ is to choose the minimum prime set such that g.c.d. $\{q_{i_r}, P-1\} = 1$ where $q_{i_r} > 6$ and $q_{i_r} > q_{i_r-1}$. Furthermore, the set $\{q_{i_r}\}$ is permuted to form a new set $\{p_{i_w}\}$ according to the inter-row permutation pattern defined in the third stage. Primes 2, 3, and 5 are not selected because they ofen result in g.c.d. $\{q_{i_r}, P-1\} \neq 1$. For example, consider $q_{i_r} = 2$. From Table 2.3, it can be seen that $13 \le P \le 409$, so $P-1$ is an even number and g.c.d. $\{2, P-1\} = 2$. Although primes larger than 6 can be chosen, depending on the value of P, some prime cannot be selected. For example, let $N = 1280$, so $L_{row} = 20$, $1280/20 = 64$. Then $P = 67$ and $P-1 = 66$. Therefore prime 11 cannot be chosen. Therefore $\{q_{i_r}\} = \{1, 7, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83\}$. As discussed below, the corresponding permutation pattern in the third stage is $PIP_A(i_r) = i_w = \{19, 9, 14, 4, 0, 2, 5, 7, 12, 18, 10, 8, 13, 17, 3, 1, 16, 6, 15, 11\}$ and $\{p_{i_w}\} = \{19, 67, 23, 61, 17, 29, 73, 31, 47, 7, 43, 83, 37, 53, 13, 79, 71, 59, 41, 1\}$.
- Perform the i_w-th row intra-row permutation. Let $c_{i_w}(j_r)$ be the input bit position of the j_r-th output after permutation of the i_w-th row. Given i_w, obtain p_{i_w}, and use the relation $j_w = c_{i_w}(j_r) = c((j_r \cdot p_{i_w}) \bmod (P-1))$ to obtain the input bit position $c_{i_w}(j_r)$ for $j_r = 0$, 1, 2,..., P-2 . To perform this evaluation, it is easier to first calculate $((j_r \cdot p_{i_w}) \bmod (P-1))$, and then obtain the value of $c_{i_w}(j_r)$ from the base mapping sequence. Depending on the value of L_{col}, the calculation of $c_{i_w}(j_r)$ is modified as follows.
  1) If $L_{col} = P$, a value for $c_{i_w}(P-1)$ is required. Set $c_{i_w}(P-1) = 0$. This is reasonable because the base mapping sequence only stores the values for $j_r = 0$, 1, 2,..., P-2 and

none of these values equal zero. For $L_{col} = P$, the sequence $\{c_{i_w}(j_r)\}$ is actually a permutation of the sequence (0, 1, 2,..., P-1).

2) If $L_{col} = P+1$, values for $c_{i_w}(P-1)$ and $c_{i_w}(P)$ are required. Set $c_{i_w}(P-1)=0$ and set $c_{i_w}(P)=P$. These assignments are reasonable because the base mapping sequence does not include these values and these are the only values available. For $L_{row} = P+1$, the sequence $\{c_{i_w}(j_r)\}$ is actually a permutation of sequence (0, 1, 2,..., P-1, P).

3) If $L_{col} = P-1$, the sequence $\{c_{i_w}(j_r)\}$ can be obtained, and is actually a permutation of sequence (1, 2,..., P-1). However, j_r = 0, 1, 2,..., P-2. The simplest solution to resolving this difference is to substract 1 from every entry of the sequence such that $c'_{i_w}(j_r)=c_{i_w}(j_r)-1$. The resulting sequence $\{c'_{i_w}(j_r)\}$ is a permutation of sequence (0, 1, 2,..., P-2) as required.

(3) In the third stage, inter-row permutation is performed and then the buffer is read out column by column. The inter-row permutation is based on the following $PIP(i_r)$ patterns, where the value of $PIP(i_r)$ denotes the original row position of the i_w-th permuted row. There are three types of $PIP(i_r)$ patterns based on the size of the interleaver.

- Case A: use PIP_A for all values of N except the values for case B and case C. $PIP_A = \{19,9,14,4,0,2,5,7,12,18,10,8,13,17,3,1,16,6,15,11\}$.
- Case B: use PIP_B for $N=2281$ to 2480, and 3161 to 3210. $PIP_B = \{19,9,14,4,0,2,5,7,12,18,16,13,17,15,3,1,6,11,8,10\}$.
- Case C: use PIP_C for $N=481$ to 530. $PIP_C = \{9,8,7,6,5,4,3,2,1,0\}$.

If the mother interleaver is larger than the required size, the unneeded positions are pruned. In Figure 2.28, the flowchart illustrates the process to generate the MappingAddress[n] = k. When the conditions $k < N$ and $n < N$ cannot be satisfied at the same time, the value k is not saved and the value n does not increase. In this way, the value k is pruned.

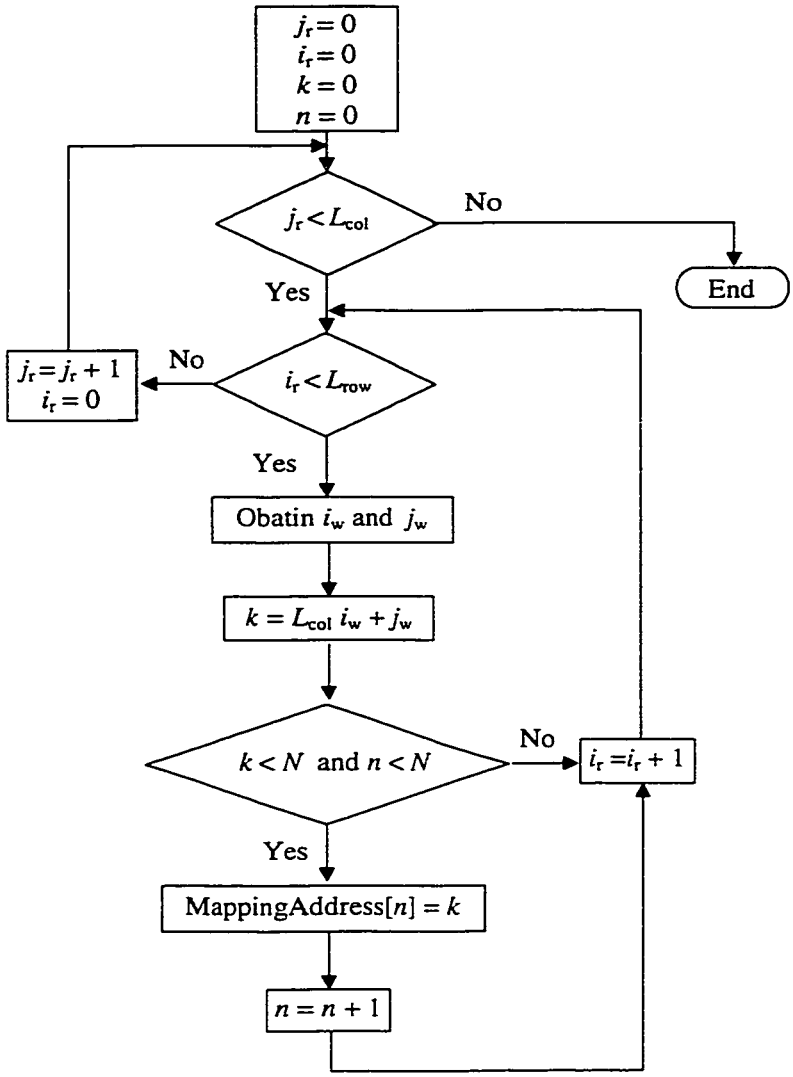
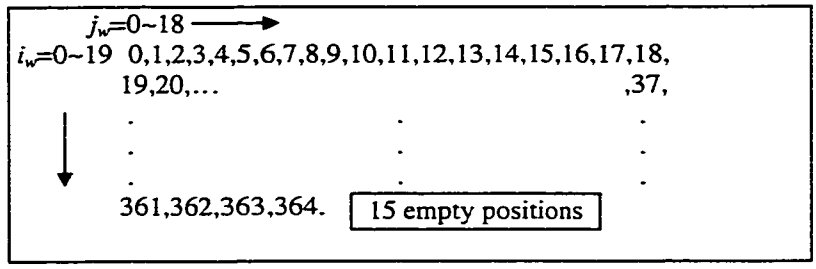


Figure 2.28: The flowchart to show the process of generating mapping address for PIL interleaver.

An example is now given to illustrate the process of generating the PIL interleaver for the case when $N = 365$.

(1) From the value of N, it follows that $L_{row} = 20$ and $N / L_{row} = 365/20 = 18.25$. Select $P_0 = 17, P = 19$. Because $(19-1) < 18.25 < 19$, $L_{col} = P = 19$. Therefore the input sequence is written into the buffer memory of size $L_{row} \times L_{col} = 20 \times 19$ row by row. Because the mother interleaver size is $20 \times 19 = 380$ which is greater than $N = 365$, this interleaver needs to be pruned. Denote the positions of the input sequence as input[k] = k, $k = 0, 1, 2, \ldots, 364$, which are written into the buffer shown as below. Note that the permutation is actually the permutation of the positions.



(2) Intra-row permutation is performed.

- Since P = 19, from Table 2.3 the value $g_0 = 2$ is obtained. Construct the base mapping sequence $c(i)$. $c(i) = \{1,2,4,8,16,13,7,14,9,18,17,15,11,3,6,12,5,10\}$ for i = 0, 1,...,17. Note that $P-2=17$.
- Select the integer set $\{q_{i_r}\}$, and then obtain $\{p_{i_w}\}$. $\{q_{i_r}\} = \{1,7,11,13,17,19,23,29,31,37,41, 43,47,53,59,61,67,71,73,79\}$, $i_r = 0,1,...,19$. By using the permutation $PIP_A = \{19,9,14,4,0,2,5,7,12,18,10,8,13,17,3,1,16,6,15,11\}$, $\{p_{i_w}\}$ can be evaluated yielding $\{p_{i_w}\} = (17,61,19,59,13,23,71,29,43,7,41,79,31,47,11,73, 67,53,37,1\}$.
- Perform the i_w-th row intra-row permutation using the formula $c_{i_w}(j_r) = c((j_r \cdot p_{i_w}) \bmod (P-1))$. Because $L_{col} = P = 19$, set $c_{i_w}(P-1) = c_{i_w}(18) = 0$. Table 2.4 below shows the process clearly.

(3) Inter-row permutation is implemented. Since $N = 365$, select pattern PIP_A for the inter-row permutation. Table 2.5 shows the result after this permutation, where the straight line marks the positions for pruning. The size of the mother interleaver is 380, however the size of the required interleaver is 365. There are 15 positions are pruned to satisfy this requirement in Table 2.5. Note that reading is performed column by column starting from $j_r = 0$. To see the process shown in Figure 2.28 clearly, define variables n_m and k_m to generate MappingAddress[n_m] = k_m for the mother interleaver. For the required interleaver, MappingAddress[n] = k is needed. Let np denote the number of positions pruned during the process. For generating the required interleaver, $n = n_m - np$, and $k = k_m$ if $k_m < N$. Now the inteleaver required is generated using the relationships $n_m = 20j_r + i_r$ and $k_m = 19i_w + j_w$ where i_r, j_r, i_w and j_w are given in Table 2.5. Some of these values are listed below. In this

list the underline indicates the pruned positions. The first pruned position is $j_r = 4$ and $i_r = 0$ corresponding to $i_w = 19$ and $j_w = 4$. Because $k_m = 365$ which is not less than $N = 365$, it is pruned.

(i_r, j_r): (0,0), (1,0), (2,0), (3,0), (4,0),..., (0,2), (1,2),... ,(19,2), (0,3), (1,3),...

(i_w, j_w): (19,1), (9,1), (14,1), (4,1), (0,1),...,(19,4), (9,6),... ,(11,6), (19,8), (9,8),...

n_m:		0,	1,	2,	3,	4, ...,	40,	41,...,	59,	60,	61,...							
k_m:		362,	172,	267,	77,	1, ... ,	365,	177,...,	215,	369,	160,...							
np:	0	0,	0,	0,	0,	0, ... ,	1,	1, ...,	1,	2,	2,...							
n:		0,	1,	2,	3,	4, ...,	40,	40, ...,	58,	59,	59,...							
k:		362,	172,	267,	77,	1, ... ,	__,	177, ...,	215,	__,	59,...							

MappingAddress[n]=(362,172,267,77,1,...,19,304,114,285,209), n =0,1,2,...,364. Note that in Table 2.5 for $i_r = 0$, the value of i_w is 19, which is the largest. Therefore, the 15 pruned positions are at this row when j_w is greater than 3. Also note that for program implementation, it is not necessary to introduce variables n_m, k_m and np (see Figure 2.28).

Table 2.4: Obtain j_w from (p_{i_w}, i_w, j_r) in the intra-row process

p_{i_w}	j_r / j_w / i_w	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
17	0	1	10	5	12	6	3	11	15	17	18	9	14	7	13	16	8	4	2	0
61	1	1	14	6	8	17	10	7	3	4	18	5	13	11	2	9	12	16	15	0
19	2	1	2	4	8	16	13	7	14	9	18	17	15	11	3	6	12	5	10	0
59	3	1	13	17	12	4	14	11	10	16	18	6	2	7	15	5	8	9	3	0
13	4	1	3	9	8	5	15	7	2	6	18	16	10	11	14	4	12	17	13	0
23	5	1	13	17	12	4	14	11	10	16	18	6	2	7	15	5	8	9	3	0
71	6	1	10	5	12	6	3	11	15	17	18	9	14	7	13	16	8	4	2	0
29	7	1	15	16	12	9	2	11	13	5	18	4	3	7	10	17	8	6	14	0
43	8	1	14	6	8	17	10	7	3	4	18	5	13	11	2	9	12	16	15	0
7	9	1	14	6	8	17	10	7	3	4	18	5	13	11	2	9	12	16	15	0
41	10	1	13	17	12	4	14	11	10	16	18	6	2	7	15	5	8	9	3	0
79	11	1	14	6	8	17	10	7	3	4	18	5	13	11	2	9	12	16	15	0
31	12	1	3	9	8	5	15	7	2	6	18	16	10	11	14	4	12	17	13	0
47	13	1	15	16	12	9	2	11	13	5	18	4	3	7	10	17	8	6	14	0
11	14	1	15	16	12	9	2	11	13	5	18	4	3	7	10	17	8	6	14	0
73	15	1	2	4	8	16	13	7	14	9	18	17	15	11	3	6	12	5	10	0
67	16	1	3	9	8	5	15	7	2	6	18	16	10	11	14	4	12	17	13	0
53	17	1	10	5	12	6	3	11	15	17	18	9	14	7	13	16	8	4	2	0
37	18	1	2	4	8	16	13	7	14	9	18	17	15	11	3	6	12	5	10	0
1	19	1	2	4	8	16	13	7	14	9	18	17	15	11	3	6	12	5	10	0

Table 2.5: Inter-row permutation

i_w / $PIP_A(i_r)$	j_r / j_w / i_r	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
19	0	1	2	~~4~~	~~8~~	~~16~~	~~13~~	~~7~~	~~14~~	~~9~~	~~18~~	~~17~~	~~15~~	~~11~~	3	~~6~~	~~12~~	~~5~~	~~10~~	0
9	1	1	14	6	8	17	10	7	3	4	18	5	13	11	2	9	12	16	15	0
14	2	1	15	16	12	9	2	11	13	5	18	4	3	7	10	17	8	6	14	0
4	3	1	3	9	8	5	15	7	2	6	18	16	10	11	14	4	12	17	13	0
0	4	1	10	5	12	6	3	11	15	17	18	9	14	7	13	16	8	4	2	0
2	5	1	2	4	8	16	13	7	14	9	18	17	15	11	3	6	12	5	10	0
5	6	1	13	17	12	4	14	11	10	16	18	6	2	7	15	5	8	9	3	0
7	7	1	15	16	12	9	2	11	13	5	18	4	3	7	10	17	8	6	14	0
12	8	1	3	9	8	5	15	7	2	6	18	16	10	11	14	4	12	17	13	0
18	9	1	2	4	8	16	13	7	14	9	18	17	15	11	3	6	12	5	10	0
10	10	1	13	17	12	4	14	11	10	16	18	6	2	7	15	5	8	9	3	0
8	11	1	14	6	8	17	10	7	3	4	18	5	13	11	2	9	12	16	15	0
13	12	1	15	16	12	9	2	11	13	5	18	4	3	7	10	17	8	6	14	0
17	13	1	10	5	12	6	3	11	15	17	18	9	14	7	13	16	8	4	2	0
3	14	1	13	17	12	4	14	11	10	16	18	6	2	7	15	5	8	9	3	0
1	15	1	14	6	8	17	10	7	3	4	18	5	13	11	2	9	12	16	15	0
16	16	1	3	9	8	5	15	7	2	6	18	16	10	11	14	4	12	17	13	0
6	17	1	10	5	12	6	3	11	15	17	18	9	14	7	13	16	8	4	2	0
15	18	1	2	4	8	16	13	7	14	9	18	17	15	11	3	6	12	5	10	0
11	19	1	14	6	8	17	10	7	3	4	18	5	13	11	2	9	12	16	15	0

2.7.3 Spread Random Interleaver

The spread random inteleaver is based on the random generation of N integers from 1 to N with an S-parameter constraint [8][30]. When a random integer is generated, it is compared to the S_1 most recently selected integers. If the currently generated integer is within a distance of S_2 from at least one of previous S_1 integers, it is rejected and a new integer is produced again until the previous condition is satisfied. This process is repeated until all N integers have been extracted. The search time for this algorithm increases with S_1 and S_2, and it is not guaranteed to obtain an interleaver successfully. It is suggested that if the two constituent codes are equal, selecting $S = S_1 = S_2 < \lfloor \sqrt{N/2} \rfloor = \lfloor 0.707\sqrt{N} \rfloor$ usually produces a solution in a reasonable time. For example, if $N = 512 = 2^9$, $S = \sqrt{N/2} = 16$. Note that for a square block interleaver that is a nonrandom interleaver, $S = \sqrt{N} - 1$, and for a purely random interleaver $S = 1$.

2.8 Termination Methods

In practice, there are two approaches for implementing convolutional code encoding and decoding: continuous and frame-oriented coding. Continuous encoding is used to encode the entire information data sequence without stopping; frame-oriented encoding breaks the encoded sequence into independent frames so that it is possible to build parallel decoders that work on different received frames. In frame-oriented convolutional codes, forcing the trellis into a known state periodically makes the encoder appear like a block encoder in that it allows the decoder to operate on each individual frame independently.

Techniques for state enforcement include trellis truncation, trellis termination and tail biting.

- *Trellis truncation*: The starting state for each frame is reset to zero, and no operation is performed at the end of a frame so that the trellis is left unterminated. The code rate of the equivalent encoder is the same as that of continuous convolutional codes, but the decoder has poorer performance near the end of each decoded sequence.
- *Trellis termination*: The starting state for each frame is reset to zero, and by inserting additional input terminating bits to the frame (note that termination positions depend on the specific termination schemes), the ending state is driven back to zero. For conventional convolutional codes, the state is terminated to zero by appending ν zeros to the end of each frame. Trellis termination improves decoding performance near the end of trellis, but reduces the code rate. When the frame length is long, the code rate reduction is small.
- *Tail biting*: In this technique, the starting and ending encoder states of each frame are constrained to be identical; that is, each frame of the encoded sequence starts from the state at which it will eventually end. The advantages of this technique are no performance degradation over trellis truncation and no code rate reduction compared to trellis termination. The disadvantage is that the decoding process is more complex. A full explanation of tail biting is given by Ma and Wolf [32].

A turbo encoder, being based on two constituent RSC encoders, is inherently a type of convolutional encoder. In addition, the presence of the block random interleaver forces the general turbo encoder to be frame-oriented. Therefore, the three techniques for trellis state enforcement described above are used in turbo codes. In this subsection, general trellis termination methods for turbo codes, including tail termination and joint termination, are discussed.

2.8.1 Tail Termination Scheme

Two additional problems exist for the trellis termination of turbo codes as compared to the case of conventional convolutional codes. Firstly, since the constituent encoders are recursive, it is not sufficient to append ν zeros to the information bit sequence to flush the encoder to zero state. Secondly, due to the interleaving between the constituent encoders, the bits for terminating the first encoder do not terminate the second constituent encoder. Solutions to these problems are as follows.

To terminate an RSC encoder, terminating bits whose values are determined by the encoder structure and information bit sequence can be inserted into the termination positions. A simple way to obtain the termination bits for an RSC encoder is reported by Divsalar and Pollara [8], and was depicted earlier in Figure 2.11. Their method is called tail termination. Based on this method, three schemes have been developed for the trellis termination of turbo codes:

- *Single termination*: Only the first constituent encoder is terminated, and the terminating bits are also passed into the interleaver, which makes a slightly longer interleaver length N. The resulting code rate is $R_1 = \frac{N-\nu}{3N} < \frac{1}{3}$. All sequences for decoding have the same length N which results in a simple decoding process, but because the second encoder is left open, all initial values for backward recursion of $\beta_N(m)$ in the M-BCJR algorithm for the second decoder are set to $1/M$ since all states are assumed to have the same possibility of being the ending state.
- *Dual termination with one individual tail*: The two constituent encoders are terminated individually, but the terminating bits for the second RSC encoder are not transmitted. The resulting code rate is $R_2 = \frac{N-\nu}{3N} = R_1$. The interleaver has a shorter length $N-\nu$. When decoding, the $L_{2e}(u_k)$, $L_{1e}(u_k^I)$ and $y_{s,k}^I$ have shorter effective data lengths $N-\nu$, because the last ν values for them have to be set to zeros.
- *Dual termination with two individual tails*: The two constituent encoders are terminated individually, and the terminating bits for both encoders are transmitted. The resulting code rate is $R_3 = \frac{N-\nu}{3N+\nu} < R_1$. When decoding, the extrinsic information, $L_{2e}(u_k)$ and $L_{1e}(u_k^I)$ have shorter effective lengths $N-\nu$.

2.8.2 Joint Termination Scheme

When no restrictions are imposed on the interleaver design, it is impossible to drive both constituent encoders to the zero state by only using ν terminating bits. But if 2ν terminating bits are used, it is possible to terminate both constituent encoders in the zero state (assuming that two constituent encoders have same memory ν). This scheme is called joint termination [33][34]. The basic idea is that by inserting appropriate termination bits, which are evaluated from the source bit values, into 2ν independent positions, both final states of the two constituent encoders can be driven to zero.

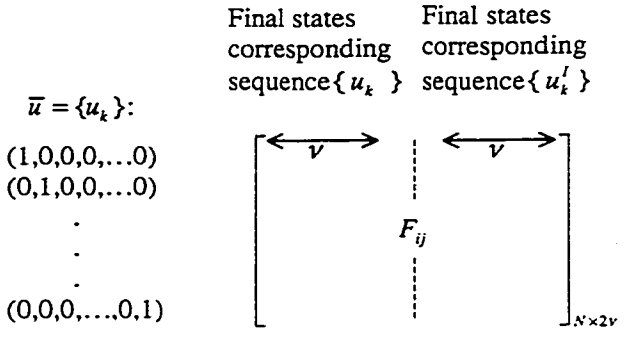
Both constituent encoders are terminated and all terminating bits are transmitted in schemes of the joint termination and dual termination with two individual tails. The comparison of two schemes is provided in Table 2.6. In this table, the complexity of encoder and decoder caused by the termination schemes is evaluated.

Table 2.6: Comparison of the joint termination and dual termination with two individual tails

Termination Scheme	Code Rate	Int. Size	ENC Complexity	DEC Complexity
Joint termination	$(N-2\nu)/3N$	N	High	Low
Dual termination with two individual tails	$(N-\nu)/(3N+\nu)$	$N-\nu$	Low	High

A. Generator matrix for final states

Because RSC codes are linear, the generator matrix for final states of the turbo encoder, denoted by $\boldsymbol{F}$, can be constructed by the base input sequences that contain only a single one. $\boldsymbol{F}$ is an N by 2ν matrix, in which the first ν columns of the i-th row stores the final state of the base input sequence $\{u_k\}$ $(k=0,1,...,N-1)$ that contains a single one at position i, and the last ν columns stores the final state of input sequence $\{u_k^I\}$ that is the interleaved version of $\{u_k\}$. The generation of $\boldsymbol{F}$ is illustrated below.



The final states for turbo encoding of any input source sequence can be obtained by multiplying the input sequence with F .

Note that every single one causes a periodic infinite state transition sequence in an RSC encoder. For example, in Figure 2.12, if the base input sequence is 1,0,0,0, ... , the state sequence is 10,11,01,10,11,01,.... This figure shows the trellis diagram for the RSC codes with generator $G(7,5)$, $\nu = 2$. Since this encoder has $M = 4$ states, there are $M - 1 = 3$ states other than the zero state to construct the state transition sequence. Also in F , the first ν columns show the final states with period $M - 1$, so if the first $M - 1$ final states are known, it is simple to obtain the other state vectors. However, the states stored in the last ν columns of F are not periodic because of the interleaver. Although the length of the base input sequence is very long, the number of states is relatively small. To determine the final states, define the effective length of the base input sequence as $N_{eff} = N - N_0$, where N_0 is the number of zeros before the single one in the base input sequence, and also define $M - 1$ effective base input sequences u_{eff} as 1, 10, 100,.... Let $r_{N_{eff},M-1}$ denote the remainder of $N_{eff}/(M-1)$. If $r_{N_{eff},M-1}$ is not zero, u_{eff} is a sequence with a single one followed by $r_{N_{eff},M-1} - 1$ zeros. If $r_{N_{eff},M-1}$ is zero, u_{eff} is a sequence with a single one followed by $M - 2$ zeros. The following table can be constructed. Note that the final states are not given in the table because they are dependent on the generator of the RSC encoder.

Table 2.7: Determination of the final states

$r_{N_{eff},M-1}$	u_{eff}	Final State
1	1	
2	10	
⋮	⋮	
$M-2$	$1\underbrace{0\cdots0}_{M-3}$	
0	$1\underbrace{0\cdots0}_{M-2}$	

Based on Table 2.7, it is straightforward to obtain matrix F. For example, consider $N=10$, $M=4$, $G(7,5)$, $u_k=0,010,000,000$, and $u_k^I=1,000,000,000$. For u_k, $N_{eff}=8$, $r_{N_{eff},M-1}=2$, $u_{eff}=10$, and the final state is 11. For u_k^I, $N_{eff}=10$, $r_{N_{eff},M-1}=1$, $u_{eff}=1$, and the final state is 10.

B. Independent positions

The rows of matrix F are directly related to positions in the input sequence with length N. Choose $2v$ rows from F to construct a square matrix F_s. If these $2v$ row vectors are linearly independent, the corresponding positions are called independent positions. These positions have to be determined by search techniques because of the interleaver. There are many independent positions; the effort choosing different independent positions is still unknown.

F_s is a $2v$ by $2v$ square matrix. The following statements are equivalent [35, pp 207]:

- The row vectors of n are linearly independent.
- F_s has rank $2v$.
- F_s is invertible.
- F_s is row equivalent to I_{2v}, where I_{2v} denotes a $2v$ by $2v$ identity matrix. Matrices that can be obtained from one another by a finite sequence of elementary row operations are said to be row equivalent.

To determine if the row vectors of F_s are linearly independent, the rank of F_s is calculated. In order to obtain the rank of any m by n matrix A, the following definitions, theorems and procedures are described [35].

(1) The common dimension of the row space and column space of a matrix A is called the rank of A and is denoted by rank(A).

(2) The row space and column space of A have the same dimension.

(3) The subspace spanned by the row vectors of A is called the row space of A, and the subspace spanned by the column vectors is called the column space.

(4) The nonzero row vectors in any row-echelon form of a matrix A form a basis for the row space, and the number of these nonzero row vectors is the dimension of the row space.

(5) Elementary row operations include: (i) multiplying a row through by a nonzero constant, (ii) interchanging two rows, and (iii) adding a multiple of one row to another row.

(6) The procedure that produces a row-echelon form of a matrix using elementary row operations is called Gaussian elimination. To be of this form, a matrix must have the properties: (i) if a row does not consist entirely of zeros, then the first non zero number in the row is a 1, called a leading 1, (ii) if there are any rows that consist entirely of zeros, then they are grouped together at the bottom of the matrix, (iii) in any two successive rows that do not consist entirely of zeros, the leading 1 in the lower row occurs father to the right than the leading 1 in the higher row. For example, the row-echelon form of matrix $A = \begin{bmatrix} 1 & 1 & 2 & 9 \\ 2 & 4 & -3 & 1 \\ 3 & 6 & -5 & 0 \\ 4 & 8 & -6 & 2 \end{bmatrix}$ is $\begin{bmatrix} 1 & 1 & 2 & 9 \\ 0 & 1 & -\frac{7}{2} & -\frac{17}{2} \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 0 \end{bmatrix}$. The rank of A is 3, because there are three nonzero row vectors in the row-echelon form of the matrix A.

Therefore, in order to calculate the rank of F_s, reduce it to its row-echelon form and count the number of the nonzero vectors. Note that F_s consists of binary numbers and that the elementary row operations are done using modulo-2 addition. If the rank of F_s is not $2v$, choose another row from F to replace a row of the original F_s to construct a new F_s, and calculate the rank again. Repeat this process until the rank of F_s is $2v$. The corresponding positions are then independent.

C. Termination bits

To simplify notation, define:

- Input sequence $\overline{u}_{s0} = \{u_{s0,k}\}$, $k = 0,1,...,N-1$, that consists of the source sequence with zeros inserted into $2v$ independent positions.
- Input sequence $\overline{u}_{0t} = \{u_{0t,k}\}$, $k = 0,1,...,N-1$, that consists of terminating bits inserted into $2v$ independent positions with the values of bits in all other positions set to zero.
- Input sequence $\overline{u}_{st} = \overline{u}_{s0} + \overline{u}_{0t}$, that is the sequence put to the turbo encoder that results in an encoded sequence where both constituent encoders are terminated in the zero state.
- Vector $\overline{u}_t = (u_{t,0},...,u_{t,2v-1})$, which contains $2v$ terminating bits.

- Square matrix F_{ir}, which is constructed by choosing $2v$ independent rows of F.
- Final state vector $\overline{s}_f = \{s_{f,0},...s_{f,v-1},s_{f,v},...s_{f,2v-1}\}$ corresponding to $\overline{u}_{st}$.
- Final state vector $\overline{s}_{f_t} = \{s_{f_t,0},...s_{f_t,v-1},s_{f_t,v},...s_{f_t,2v-1}\}$ corresponding to $\overline{u}_{s0}$.

The final state after encoding sequence $\overline{u}_{st}$ can be forced to zero, if the final states for the sequences $\overline{u}_{s0}$ and $\overline{u}_{0t}$ are identical, because in GF(2), these state values are canceled by modulo-2 addition. The final state can then be written as follows:

$$\begin{aligned}\overline{s}_f &= \overline{u}_{st}F \\ &= (\overline{u}_{s0} + \overline{u}_{0t})F \\ &= \overline{u}_{s0}F + \overline{u}_{0t}F \\ &= \overline{u}_{s0}F + \overline{u}_t F_{ir} \\ &= \overline{s}_{f_t} + \overline{u}_t F_{ir}.\end{aligned}$$

Imposing $\overline{s}_f = 0$ yields:

$$\overline{u}_t = \overline{s}_{f_t} F_{ir}^{-1} \tag{2.102}$$

where all arithmetic is from GF(2).

The terminating positions are independent of the source sequence, so these positions can be determined during system design. Generally the termination bits need to be distributed through the frame somewhat, but it is possible to adjust the interleaver design slightly to make the last $2v$ positions be terminating positions. If so, it is easy for practical implementation to discard the terminating bits after decoding.

The following outlines the procedure for implementing joint termination in practice:

- Pre-compute and store the set of terminating positions and the matrix F_{ir}^{-1}.
- Insert the source information sequence into the non-terminating positions to construct $\overline{u}_{s0}$.
- Obtain $\overline{s}_{f_t}$ by passing $\overline{u}_{s0}$ to the first constituent encoder and the interleaved version of $\overline{u}_{s0}$ to the second constituent encoder.
- Calculate the vector of terminating bits $\overline{u}_t$ using relation 2.102.
- Obtain the sequence $\overline{u}_{st}$ by inserting the terminating bits into the terminating positions in the sequence $\overline{u}_{s0}$.

- Apply the sequence $\overline{u}_{st}$ to the turbo encoder to generate the encoded sequence.

As an example of joint termination, consider the case when the turbo encoder consists of two identical constituent encoders with generator G(7,5), $N=10$, and the permutation table is $\pi=\begin{bmatrix}0123456789\\2104935768\end{bmatrix}$. In this table, the first row denotes the position k before permutation, and the second row denotes the permuted position n. For example, by permutation, position 0 is moved to position 2, position 1 stays at position 1, the last position 9 is moved to position 4 etc. This permutation table can be translated into the mapping address, MappingAddress[k] $=\{n\}$ $=(2,1,0,4,9,3,5,7,6,8)$ or MappingAddress[n]$=\{k\}=(2,1,0,5,3,6,8,7,9,4)$. Table 2.8 shows the generation of $\boldsymbol{F}$.

Table 2.8: The generation of $\boldsymbol{F}$

Uninterleaved bit position k	Uninterleaved base sequence u_k	The final state of 1st encoder	The final state of 2nd encoder	Interleaved base sequence $u_n=u_k^I$	Interleaved bit position n
0	1000000000	10	11	0010000000	2
1	0100000000	01	01	0100000000	1
2	0010000000	11	10	1000000000	0
3	0001000000	10	01	0000100000	4
4	0000100000	01	10	0000000001	9
5	0000010000	11	10	0001000000	3
6	0000001000	10	11	0000010000	5
7	0000000100	01	01	0000000100	7
8	0000000010	11	10	0000001000	6
9	0000000001	10	11	0000000010	8

From this table, it cab be seen that the sequence of final states of the first constituent encoder has period 3. From Table 2.8, it is straightforward to obtain the matrix $\boldsymbol{F}$:

$$\boldsymbol{F}=\begin{bmatrix}1&0&1&1\\0&1&0&1\\1&1&1&0\\1&0&0&1\\0&1&1&0\\1&1&1&0\\1&0&1&1\\0&1&0&1\\1&1&1&0\\1&0&1&1\end{bmatrix}$$

Let $\overline{r}_k$ ($k=0,1,2,...,9$) denote the rows of $\boldsymbol{F}$, and $k=3,4,8,9$ rows are chosen to check if they are linearly independent vectors. If yes, these rows will be chosen to construct $\boldsymbol{F}_{ir}$.

$$\begin{aligned} 1\ 0\ 0\ 1 &= \overline{r}_3 \\ 0\ 1\ 1\ 0 &= \overline{r}_4 \\ 1\ 1\ 1\ 0 &= \overline{r}_8 \\ 1\ 0\ 1\ 1 &= \overline{r}_9 \end{aligned}$$

It is straightforward to show that these vectors are linearly independent by the fact that all length-4 weight-1 vectors can be constructed as follows using elementary row operations:

$$\begin{aligned} 1\ 0\ 0\ 0 &= \overline{r}_4 + \overline{r}_8 \\ 0\ 1\ 0\ 0 &= \overline{r}_3 + \overline{r}_4 + \overline{r}_9 \\ 0\ 0\ 1\ 0 &= \overline{r}_3 + \overline{r}_9 \\ 0\ 0\ 0\ 1 &= \overline{r}_3 + \overline{r}_4 + \overline{r}_8 \end{aligned}$$

This shows that $\boldsymbol{F}_{ir}$ is row equivalent to the identity matrix I_4. Also reducing $\boldsymbol{F}_{ir}$ to its row-echelon form $\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ demonstrates that the rank of $\boldsymbol{F}_{ir}$ is 4.

So $\boldsymbol{F}_{ir}$ and $\boldsymbol{F}_{ir}^{-1}$ are obtained:

$$\boldsymbol{F}_{ir} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}, \quad \boldsymbol{F}_{ir}^{-1} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}.$$

Let the source data sequence be $\overline{u}=100100$, and the terminating positions be 3, 4, 8 and 9 as determined from above. Then, $\overline{u}_{s0}$ = 100<u>00</u>10<u>00</u>, where the underline marks the terminating positions, $\overline{s}_{f_s} = \overline{u}_{s0}\boldsymbol{F}$ =0101, $\overline{u}_t = \overline{s}_{f_s}\boldsymbol{F}_{ir}^{-1}$ =0011, $\overline{u}_{st}$ = 100<u>00</u>10<u>11</u>, and $\overline{s}_f = \overline{u}_{st}\boldsymbol{F}$ = 0000. Therefore, by the joint termination method, the final states of both encoders are forced to the zero state. Note that in practice, $\overline{s}_{f_s}$ is obtained by passing sequence $\overline{u}_{s0}$ to the first constituent encoder, and $\overline{u}_{s0}^{I}$ to the second constituent encoder. The positions of termination bits are fixed during the design phase, and the values of the termination bits are determined from the source sequence.

A comparison of the BER and FER for turbo codes using single termination (terminating only the 1st encoder) and joint termination is shown in Figure 2.29 and Figure 2.30. The interleaver is spread random and the size is not large, only 256. The simulation results show that terminating the entire trellis rather than just the first trellis does not significantly enhance performance for low values of SNR.

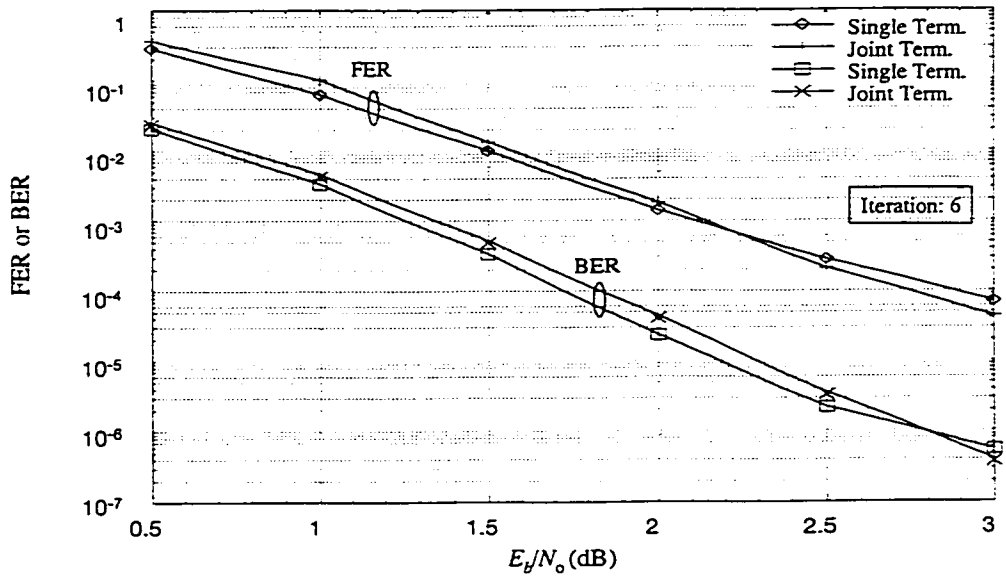


Figure 2.29: Comparison of FER and BER performance for single termination and joint termination with 6 iterations, spread random interleaver $N = 256$, R = 1/3, $G(7,5)$, and terminating positions 213, 234, 244 and 252 for joint termination.

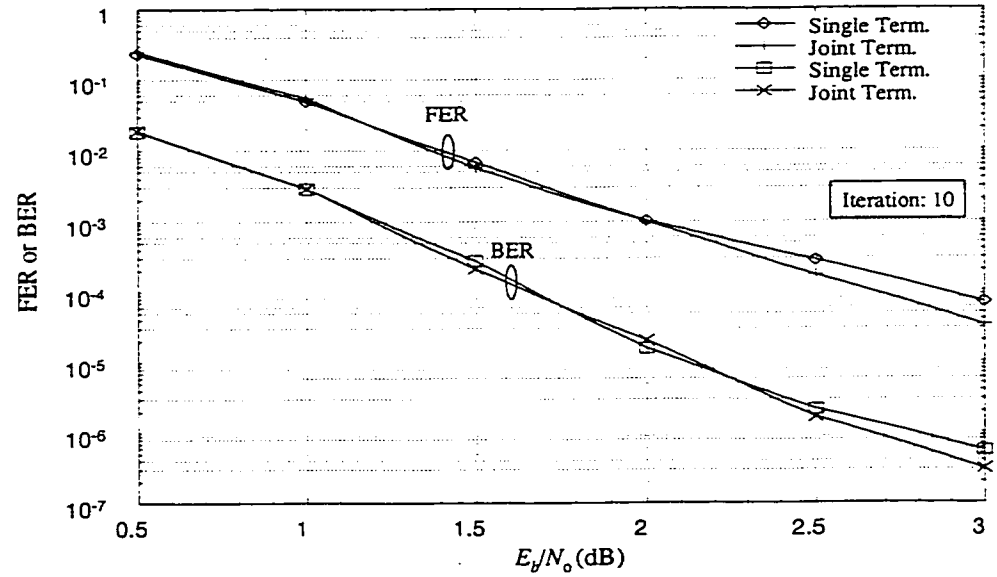


Figure 2.30: Comparison of FER and BER performance for single termination and joint termination with 10 iterations, spread random interleaver $N = 256$, R = 1/3, $G(7,5)$, and terminating positions 213, 234, 244 and 252 for joint termination.

2.9 Simulation Platform for Turbo Coding

Simulation of turbo codes and new techniques of early stopping and error detection for turbo decoding has provided validation of analysis and provided performance results for these techniques. All simulation and analysis was performed under the Borland C++ Builder 3 environment (professional version) [36] on the IBM PC family of computers.

C++ Builder 3 is Borland's rapid application development (RAD) product for writing C++ applications. With C++ Builder 3, C++ Win32 console applications or Win32 graphical user interface (GUI) programs can be created easily. For simulation of turbo codes, all programs were created as console applications. A Win32 console application is a 32-bit program that runs in a DOS box under Windows 95, 98, 2000 or Windows NT.

2.9.1 Simulation Structure and Projects Groups

There are two ways to construct programs for simulation of an entire communication link. First, a complex program could be written to simulate all operations between data source and data sink with specified configurations and channel parameters. This program implemention operates like a real time simulation. The source generates a frame of data, and passes it to encoder, channel and finally to the decoder, and then repeats. The program could be controlled by the number of errors collected (say 100). If the specified number of errors is collected, the program could stop or continue by moving to another channel parameter. Alternatively, a number of short programs could be written with each simulating only a part of the transmission system. Information could be transferred between these programs through a series of data files. Encoder and channel parameters could be varied easily without the necessity of generating source data again. Code performance could be examined at any point of the transmission link by recording the appropriate information. Due to its flexibility, simplicity of expansion, and ease of use, the second structure was implemented for simulations in this research.

In addition, the second structure is easy to use by creating what is known as a project group within C++ Builder 3. A project is a collection of files that work together to create a standalone executable file, and a project group is a collection of projects. A project group is used to manage a group of projects that work together to form a complete simulation system. For example, a source project was created to yield random binary source data. The source project named "Source" consists of many files, but only three files are of interest here, including "source.cpp" which is a C++ program, "source.dat" which is the random binary sequence data generated by

"source.cpp", and "seed.dat" which is a random integer to guarantee randomness of the source data.

For simulating the overall transmission link, several projects need to be created. Generally, a project group consists of the projects to simulate portions of the transmission link including the data source, encoder, channel, and decoder. In this thesis, the projects for source and channel do not vary significantly. However, the projects for encoder and decoder have more variation depending on the different codes and techniques used in the decoder. After the projects were compiled successfully, the final executable files were made. Then, the appropriate program was run by activating the specified project.

2.9.2 Project and Project Groups for Turbo Codes

To simulate and compare the performance of turbo codes and other encoding and decoding schemes, separate projects were created and several project groups of special interest were constructed. The projects are listed in Table 2.9. Appendix 4 gives the program specifics for the simulation. Note that project names listed in Table 2.9 are more readable versions of the file names than those given in Appendix 4. For example, the program "NSCCodes" discussed in Appendix 4 is listed as "NSC codes" in Table 2.9.

Table 2.9: The projects for evaluating turbo codes

Portions of transmission link	**Projects**
Data Source	Source
Encoder	NSC codes
	RSC codes
	Turbo codes
	Turbo with CRC codes
Channel	AWGN channel
Decoder	VA
	BCJR algorithm
	M-BCJR algorithm
	Berrou's decoding algorithm
	Robertson's decoding algorithm
	Algorithm with early stopping
	Algorithm with early stopping and error detection
	Algorithm with early stopping error detection and CRC check

Based on the projects listed in Table 2.9, the project groups listed in Table 2.10 were constructed.

Table 2.10: Project groups and the constituent projects for evaluating turbo codes

Project groups	**Constituent projects**
NSC 1	Source → NSC codes → AWGN channel → VA
NSC 2	Source → NSC codes → AWGN channel → BCJR algorithm
RSC 1	Source → RSC codes → AWGN channel →VA
RSC 2	Source → RSC codes → AWGN channel → M-BCJR algorithm
Turbo codes 1	Source → Turbo codes → AWGN channel → Berrou's decoding algorithm
Turbo codes 2	Source →Turbo codes → AWGN channel → Robertson's decoding algorithm
Early stopping	Source → Turbo codes → AWGN channel → Algorithm with early stopping
Early stopping and error detection	Source → Turbo codes → AWGN channel → Algorithm with early stopping and error detection.
Stopping, error detection and CRC	Source → Turbo with CRC codes → AWGN channel → Algorithm with early stopping , error detection and CRC check.

CHAPTER 3

APPROACHES FOR EARLY STOPPING

Because the turbo decoder works in an iterative fashion, excellent performance is obtained at the cost of delay. Once iterations fail to improve the accuracy of decoding, the iterative process should be terminated by a stopping criterion. This will reduce decoding delay. In this chapter, following introduction of the challenges associated with iterative decoding, several stopping criteria are briefly described. In general, stopping criteria can be applied after each half iteration or each full iteration (see Figure 3.1). It is convention to report the average number of iterations for a stopping criterion. Let i denote the iteration number and let i_r be the value used for the calculation of average number of iterations. If the stopping criterion is applied after decoder 1, $i_r = i - 0.5$, otherwise $i_r = i$.

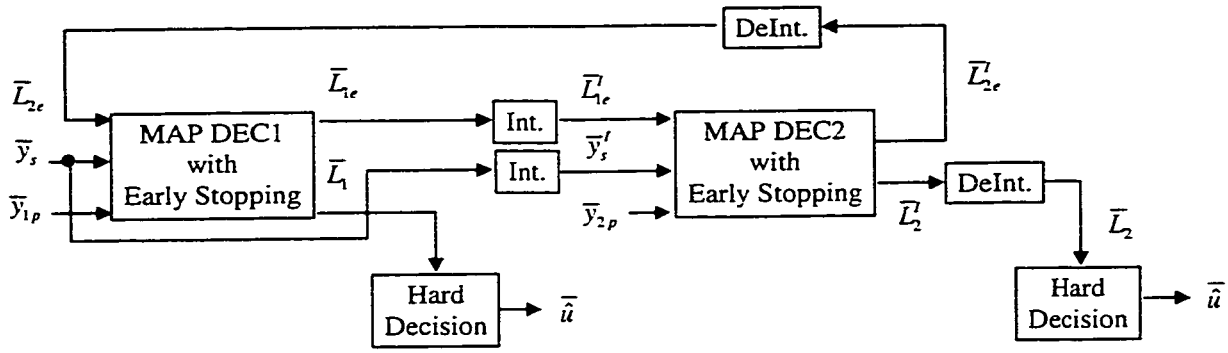


Figure 3.1: Structure of turbo decoder with early stopping

3.1 Challenges with Iterative Decoding

From the BER curves for turbo codes, it can be seen that with each iteration, the required E_b / N_o to obtain a specified BER decreases. But it can also be observed that the improvement in E_b / N_o becomes smaller with each iteration. For example, when the turbo code BER curves in Figure 2.22 are examined at BER = 10^{-4}, it is found that the required E_b / N_o decreases 1.65 dB from iteration 1 to 2, but from iterations 6 to 10, it decreases only an additional 0.2 dB. This is because with each iteration the extrinsic sequences $\bar{L}_{1e}$ and $\bar{L}_{2e}$ become more correlated, and this correlation decreases the improvement in performance [37].

The question then arises as to how to determine when additional iterations will provide little or no increase in BER performance. Techniques of doing so have been called early stopping criteria, and several attempts have been made at designing good early stopping criteria to terminate the decoding process without significantly impacting the performance of the turbo decoder [5][7][31][38]. In addition, the computational complexity of turbo decoding can be high. Decoding with early stopping criteria leads to iterative decoders that are more computationally efficient than fixed-complexity iterative decoders.

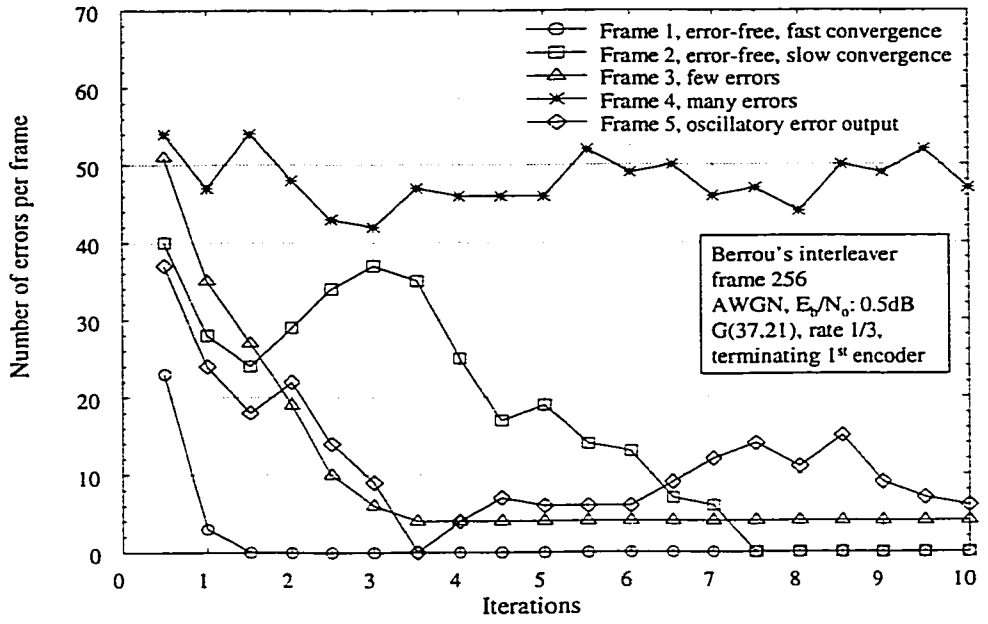


Figure 3.2: Typical turbo decoding performance

Figure 3.2 shows turbo decoding performance in terms of number of output errors as a function of the number of iterations for five typical frames under the same channel conditions. Based on the investigation of 2,000 frames, five general classes regarding the number of output errors with iterations are obtained. The five frames in Figure 3.2 are typical representatives of these classes. Note that the number of iterations is assumed to be large enough for the investigation. For the selection of the frame size of 256, first, the low frame error rate is relatively easier to be simulated using a shorter frame size within a reasonable period of time; second, in CDMA2000 high rate packet data air interface specification (HDR standard proposed by QUALCOMM), frame size of 256 is one of the five reverse traffic channel payload sizes (i.e. 256, 512, 1024, 2048 and 4096) [39, p. 11-4].

These five curves in Figure 3.2 illustrate that large differences exist in the convergence and error performance of different frames. In Frame 1, the errors are corrected quickly. Correction of

all errors in Frame 2 requires more iterations, but after 8 iterations all errors are corrected. With a maximum of ten iterations, errors in Frames 3 and 4 are not corrected, nor does it appear that additional iterations would help in these instances. Frame 5 exhibits oscillation in the decoding performance. Although all errors are corrected after the first few iterations, errors appear again after a number of additional iterations. The most probable reason for this situation is the accumulation of numerical errors in the decoding algorithm if iterative decoding continues too long. Clearly, the best performance can be obtained if the decoding process can be stopped at the proper time. In this thesis it is demonstrated that it is possible to design good stopping criteria to achieve this performance, using less time and complexity than fixed-complexity iterative decoding.

3.2 Early Stopping Criteria

In this section, several previously published stopping criteria are introduced, including variance estimate (VE) [7], cross entropy (CE) [5], sign-change-ratio (SCR) and hard-decision-aided (HDA) [38], and turbo-CRC [31].

3.2.1 Variance Estimate

In Berrou's original turbo decoding algorithm, the turbo decoder used received values from the AWGN channel and values from the meta-channel (i.e. $\tilde{L}_1^N(u_k)$ and $z_{s,k}$), where the meta-channel consists of the concatenation of the AWGN channel and constituent decoder. That is, the constituent decoders are looked upon as a part of the meta-channel. In order to use the values from the meta-channel, the variance of the meta-channel has to be estimated. The variance of the meta-channel changes with each half iteration. Robertson improved the turbo decoding algorithm, so that it was not necessary to estimate the meta-channel. But Robertson suggested that the variance of the meta-channel could still be calculated in order to decide whether or not the iterative process should be carried on. If the variance of the meta-channel is low, further iterations are likely to have no effect on decreasing the number of errors in that frame. This scheme is called variance estimate, and is the first early stopping criterion proposed for turbo decoding [7]. In this subsection, VE is described in detail.

Assume the channel model of the meta-channel is AWGN with zero mean and variance σ_m^2, where $x \in \{-1,+1\}$ is transmitted and z is the received value. During turbo decoding, the LLR

$L(x)$, the conditional LLR of x given observation z, is available after each constituent decoder. It is straightforward to show that if x assumes values ± 1 with equal probability, then:

$$\begin{aligned} L(x) &= \log_e \frac{\Pr\{x=+1 \mid z\}}{\Pr\{x=-1 \mid z\}} \\ &= \log_e \frac{\Pr\{z \mid x=+1\}}{\Pr\{z \mid x=-1\}} \\ &= \log_e \frac{f(z \mid x=+1)}{f(z \mid x=-1)} \\ &= \frac{2}{\sigma_m^2} z . \end{aligned} \tag{3.1}$$

Therefore, there exists a linear relationship between LLR $L(x)$ and the associated meta-channel observation z:

$$z = \frac{\sigma_m^2}{2} \cdot L(x) \tag{3.2}$$

where $f(\cdot \mid \cdot)$ denotes the conditional probability density function (PDF). The PDF of x is:

$$f_x(b) = 0.5\delta(b-1) + 0.5\delta(b+1), \quad b \in \pm 1 \tag{3.3}$$

where $\delta(\cdot)$ is the unit impulse function and the random variable (RV) x takes on the values of b. By assuming that the noise of the meta-channel n is normally distributed:

$$f_n(b) = \frac{1}{\sqrt{2\pi}\sigma_m} e^{-b^2/2\sigma_m^2}, \quad -\infty < b < \infty . \tag{3.4}$$

Because $z = x + n$, and x and n are independent, the PDF of z is the convolution of the PDF of x and the PDF of n [21, p. 54]:

$$f_z(b) = \frac{1}{2} \frac{1}{\sqrt{2\pi}\sigma_m} (e^{-(b-1)^2/2\sigma_m^2} + e^{-(b+1)^2/2\sigma_m^2}), \quad -\infty < b < \infty . \tag{3.5}$$

From relation (3.5), it can be observed that the distribution of z is the superposition of two normal distributions. An important fact about normal random variables is that if w is normally distributed with mean μ and variance σ^2, then the RV $v = \alpha w$, where α is a constant, is normally distributed with mean $\alpha\mu$ and variance $\alpha^2\sigma^2$ [21, p. 35]. Then, by the relations (3.1), (3.5) and $\alpha = \frac{2}{\sigma_m^2}$, the PDF of $L(x)$ can be obtained as:

$$f_L(b) = \frac{1}{2}\frac{1}{\sqrt{2\pi}\alpha\sigma_m}(e^{-(b-\alpha)^2/2\alpha^2\sigma_m^2} + e^{-(b+\alpha)^2/2\alpha^2\sigma_m^2}), \quad -\infty < b < \infty. \tag{3.6}$$

Now consider the second moment of RV $L(x)$, denoted m_{L^2}. If the RV w is normally distributed with mean μ and variance σ^2 then the second moment of w is [21, p. 62]:

$$m_{w^2} = E(w^2) = \mu^2 + \sigma^2, \tag{3.7}$$

where "E" denotes the expectation of a RV. From relation (3.6) and (3.7), m_{L^2} is:

$$\begin{aligned} m_{L^2} &= \frac{1}{2}(\alpha^2 + \alpha^2\sigma_m^2) + \frac{1}{2}((-\alpha)^2 + \alpha^2\sigma_m^2) \\ &= \alpha^2(1+\sigma_m^2) \\ &= \frac{4}{\sigma_m^4}(1+\sigma_m^2). \end{aligned}$$

Rearranging this expression yields a quadratic equation in σ_m^2 as $m_{L^2}(\sigma_m^2)^2 - 4\sigma_m^2 - 4 = 0$, with the positive solution:

$$\sigma_m^2 = \frac{2 + 2\sqrt{1+m_{L^2}}}{m_{L^2}}. \tag{3.8}$$

Therefore, to estimate σ_m^2, an estimation of m_{L^2} is required. The following formula can be used to estimate m_{L^2} of $L(x)$ in a frame [24.]. Note that $L_k(x)$ is the value of $L(x)$ at time k:

$$m_{L^2} = \frac{1}{N}\sum_{k=1}^{N} L_k^2(x) \tag{3.9}$$

In turbo decoding, $L_k(x)$ is the LLR output of a constituent decoder. If m_{L^2} is very large, $\sigma_m^2 \approx 2/\sqrt{m_{L^2}}$. Figure 3.3 shows the variance estimate of the meta-channel based on the second moment of the LLR output of each constituent decoder for the five frames, where error characteristics were previously shown in Figure 3.2.

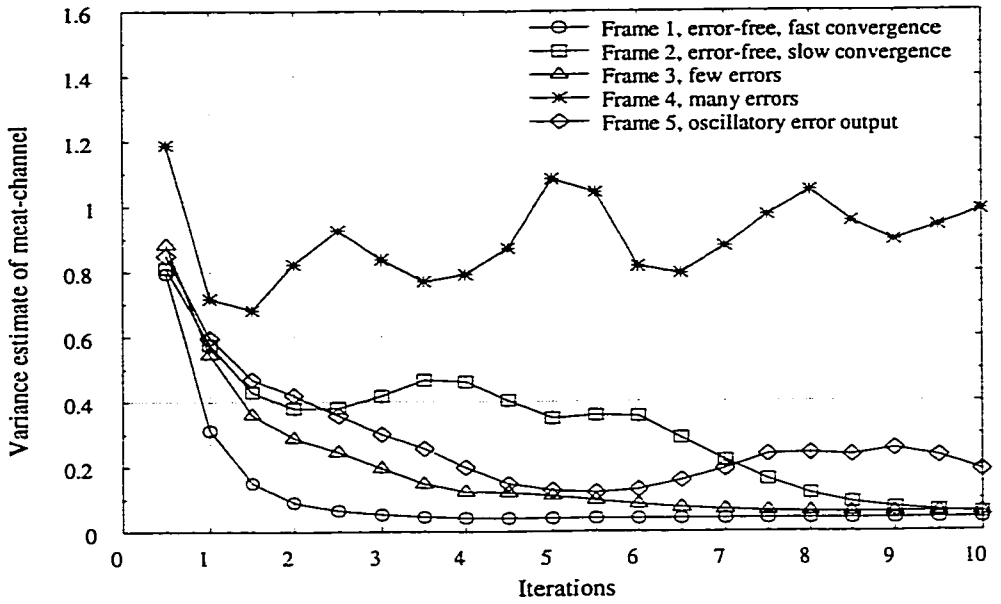


Figure 3.3: Variance estimate of meta-channel based on the second moment of the LLR for five typical frames (see Figure 3.2 for the corresponding number of errors).

Now for each iteration, an estimate of the overall accuracy of the decoded bit sequence is available. Assuming that E_b is normalized to 1 and a correlator is employed at the receiver, the relationship between the variance σ_m^2 and E_b / N_o for AWGN channel is:

$$E_b / N_o = 1 / N_o = 1 / 2\sigma_m^2 . \tag{3.10}$$

For example, a variance of 0.055 is equivalent to an E_b / N_o of 9.59 dB. The probability of bit error is [2, pp. 87]:

$$P_B = \int_{1/\sigma_m}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}} du = Q(1/\sigma_m) \tag{3.11}$$

where $Q(\cdot)$ is the complementary error function. For $\sigma_m^2 = 0.055$, $P_B = 1.0 \times 10^{-5}$.

It has been recommended that the threshold to stop iterations be chosen for an equivalent channel SNR of either 10 dB or 12.5 dB depending on the BER range of interest [7]. Therefore, to implement VE for early stopping, specify the threshold of σ_m^2 and the maximum number of iterations. During the iterative process, after each constituent decoder, estimate m_{L^2} using relation (3.9), calculate σ_m^2 using relation (3.8), and determine if the threshold has been met. If so, stop the iterative process. If not, continue the iterative process. The iterative process will stop after the maximum number of iterations for those frames that never satisfy the threshold.

3.2.2 Cross Entropy

Cross entropy is a useful criterion for stopping an iterative decoder [5]. The cross entropy of two probability distributions $P_1(\overline{u})$ and $P_2(\overline{u})$ is defined as $E_{P_2}\left[\log_e \frac{P_2(\overline{u})}{P_1(\overline{u})}\right]$, where $\overline{u}=(u_1,u_2,...,u_N)$ and E_{P_2} denotes the expectation over the distribution P_2. Cross entropy is a measure of the closeness of two probability distributions. Assuming statistical independence of $u_1,u_2,...,u_N$ with identical distribution p_1, then $P_1(\overline{u})=\prod_{k=1}^{N} p_1(u_k)$. Similarly, it is assumed that $P_2(\overline{u})=\prod_{k=1}^{N} p_2(u_k)$. Therefore:

$$\begin{aligned} E_{P_2}\left[\log_e \frac{P_2(\overline{u})}{P_1(\overline{u})}\right] &= E_{P_2}\left[\log_e \prod_{k=1}^{N} \frac{p_2(u_k)}{p_1(u_k)}\right] \\ &= E_{P_2}\left[\sum_{k=1}^{N} \log_e \frac{p_2(u_k)}{p_1(u_k)}\right] \\ &= \sum_{k=1}^{N} E_{p_2}\left[\log_e \frac{p_2(u_k)}{p_1(u_k)}\right]. \end{aligned} \tag{3.12}$$

For the i-th iteration of the turbo decoder at time k :

$$L_1^{(i)}(u_k) = L_{2e}^{(i-1)}(u_k) + \frac{2}{\sigma^2} y_{s,k} + L_{1e}^{(i)}(u_k), \quad i \geq 1, \tag{3.13}$$

$$L_2^{(i)}(u_k) = L_{1e}^{(i)}(u_k) + \frac{2}{\sigma^2} y_{s,k} + L_{2e}^{(i)}(u_k), \quad i \geq 1, \tag{3.14}$$

$$\Delta L_{2e}^{(i)}(u_k) = L_{2e}^{(i)}(u_k) - L_{2e}^{(i-1)}(u_k) = L_2^{(i)}(u_k) - L_1^{(i)}(u_k), \quad i \geq 1, \tag{3.15}$$

$$\Delta L_{1e}^{(i)}(u_k) = L_{1e}^{(i)}(u_k) - L_{1e}^{(i-1)}(u_k) = L_1^{(i)}(u_k) - L_2^{(i-1)}(u_k), \quad i \geq 2, \tag{3.16}$$

and if a decision is made from $L_2^{(i)}(u_k)$, then:

$$u_k^{(i)} = \text{sign}(L_2^{(i)}(u_k)).$$

The probability distribution at the output of the m-th constituent decoder, where $m=1$ or 2, is given by:

$$p_m^{(i)}(u_k = +1) = \frac{e^{L_m^{(i)}(u_k)}}{1+e^{L_m^{(i)}(u_k)}}$$

$$p_m^{(i)}(u_k=-1)=\frac{1}{1+e^{L_m^{(i)}(u_k)}}. \tag{3.17}$$

For the CE stopping criterion at the i-th iteration, the closeness of $P_1^{(i)}$ and $P_2^{(i)}$ must be evaluated, where $P_1^{(i)}=\prod_{k=1}^{N} p_1^{(i)}(u_k)$ and $P_2^{(i)}=\prod_{k=1}^{N} p_2^{(i)}(u_k)$. To do so, first consider calculation of $E_{p_2}[\log_e \frac{p_2(u_k)}{p_1(u_k)}]$:

$$\begin{aligned}
E_{p_2^{(i)}}[\log_e \frac{p_2^{(i)}(u_k)}{p_1^{(i)}(u_k)}] &= p_2^{(i)}(u_k=+1)\log_e \frac{p_2^{(i)}(u_k=+1)}{p_1^{(i)}(u_k=+1)} + p_2^{(i)}(u_k=-1)\log_e \frac{p_2^{(i)}(u_k=-1)}{p_1^{(i)}(u_k=-1)} \\
&= \frac{e^{L_2^{(i)}(u_k)}}{1+e^{L_2^{(i)}(u_k)}}(\log_e \frac{e^{L_2^{(i)}(u_k)}}{1+e^{L_2^{(i)}(u_k)}} - \log_e \frac{e^{L_1^{(i)}(u_k)}}{1+e^{L_1^{(i)}(u_k)}}) \\
&\quad + \frac{1}{1+e^{L_2^{(i)}(u_k)}}(\log_e \frac{1}{1+e^{L_2^{(i)}(u_k)}} - \log_e \frac{1}{1+e^{L_1^{(i)}(u_k)}}) \\
&= \frac{e^{L_2^{(i)}(u_k)}}{1+e^{L_2^{(i)}(u_k)}}(\Delta L_{2e}^{(i)}(u_k) + \log_e \frac{1+e^{L_1^{(i)}(u_k)}}{1+e^{L_2^{(i)}(u_k)}}) + \frac{1}{1+e^{L_2^{(i)}(u_k)}}(\log_e \frac{1+e^{L_1^{(i)}(u_k)}}{1+e^{L_2^{(i)}(u_k)}}) \\
&= \frac{e^{L_2^{(i)}(u_k)}}{1+e^{L_2^{(i)}(u_k)}}\Delta L_{2e}^{(i)}(u_k) + \log_e \frac{1+e^{L_1^{(i)}(u_k)}}{1+e^{L_2^{(i)}(u_k)}} \\
&= -\Delta L_{2e}^{(i)}(u_k)\frac{1}{1+e^{L_2^{(i)}(u_k)}} + \log_e \frac{1+e^{-L_1^{(i)}(u_k)}}{1+e^{-L_2^{(i)}(u_k)}}.
\end{aligned} \tag{3.18}$$

When the decisions regarding input bit values no longer change, i.e., $u_k^{(i)}=\text{sign}(L_1^{(i)}(u_k))=\text{sign}(L_2^{(i)}(u_k))$, $u_k^{(i)}$ can be introduced to relation (3.18) to obtain:

$$E_{p_2^{(i)}}[\log_e \frac{p_2^{(i)}(u_k)}{p_1^{(i)}(u_k)}] = -\Delta L_{2e}^{(i)}(u_k)\frac{1}{1+e^{u_k^{(i)}|L_2^{(i)}(u_k)|}} + \log_e \frac{1+e^{-u_k^{(i)}|L_1^{(i)}(u_k)|}}{1+e^{-u_k^{(i)}|L_2^{(i)}(u_k)|}}.$$

If $u_k^{(i)}=+1$, then:

$$E_{p_2^{(i)}}[\log_e \frac{p_2^{(i)}(u_k)}{p_1^{(i)}(u_k)}] = -u_k^{(i)}\Delta L_{2e}^{(i)}(u_k)\frac{1}{1+e^{|L_2^{(i)}(u_k)|}} + \log_e \frac{1+e^{-|L_1^{(i)}(u_k)|}}{1+e^{-|L_2^{(i)}(u_k)|}}. \tag{3.19}$$

If $u_k^{(i)}=-1$, noting that $|L_2^{(i)}(u_k)|-|L_1^{(i)}(u_k)|=u_k^{(i)}\Delta L_{2e}^{(i)}(u_k)$ results in:

$$E_{p_2^{(i)}}[\log_e \frac{p_2^{(i)}(u_k)}{p_1^{(i)}(u_k)}] = -\Delta L_{2e}^{(i)}(u_k)\frac{1}{1+e^{-|L_2^{(i)}(u_k)|}} + \log_e \frac{1+e^{|L_1^{(i)}(u_k)|}}{1+e^{|L_2^{(i)}(u_k)|}}$$

$$= -\Delta L_{2e}^{(i)}(u_k)\frac{1}{1+e^{-|L_2^{(i)}(u_k)|}} + \log_e \frac{e^{-|L_1^{(i)}(u_k)|}e^{-|L_2^{(i)}(u_k)|}(1+e^{|L_1^{(i)}(u_k)|})}{e^{-|L_1^{(i)}(u_k)|}e^{-|L_2^{(i)}(u_k)|}(1+e^{|L_2^{(i)}(u_k)|})}$$

$$= -\Delta L_{2e}^{(i)}(u_k)\frac{e^{|L_2^{(i)}(u_k)|}}{1+e^{|L_2^{(i)}(u_k)|}} + (|L_1^{(i)}(u_k)| - |L_2^{(i)}(u_k)|) + \log_e \frac{(1+e^{-|L_1^{(i)}(u_k)|})}{(1+e^{-|L_2^{(i)}(u_k)|})}$$

$$= u_k^{(i)}\Delta L_{2e}^{(i)}(u_k)\frac{e^{|L_2^{(i)}(u_k)|}}{1+e^{|L_2^{(i)}(u_k)|}} - u_k^{(i)}\Delta L_{2e}^{(i)}(u_k) + \log_e \frac{(1+e^{-|L_1^{(i)}(u_k)|})}{(1+e^{-|L_2^{(i)}(u_k)|})}$$

$$= -u_k^{(i)}\Delta L_{2e}^{(i)}(u_k)\frac{1}{1+e^{|L_2^{(i)}(u_k)|}} + \log_e \frac{1+e^{-|L_1^{(i)}(u_k)|}}{1+e^{-|L_2^{(i)}(u_k)|}}. \qquad (3.20)$$

From relations (3.19) and (3.20), it is clear that for either value of $u_k^{(i)}$:

$$E_{p_2^{(i)}}[\log_e \frac{p_2^{(i)}(u_k)}{p_1^{(i)}(u_k)}] = -u_k^{(i)}\Delta L_{2e}^{(i)}(u_k)\frac{1}{1+e^{|L_2^{(i)}(u_k)|}} + \log_e \frac{1+e^{-|L_1^{(i)}(u_k)|}}{1+e^{-|L_2^{(i)}(u_k)|}}. \qquad (3.21)$$

Further, if the value of $|L_m^{(i)}|$ is very large, $e^{-|L_m^{(i)}(u_k)|}$ is very small, and the fact that $\log_e(1+x) = x$ for very small x can be used. Using this in relation (3.21) yields:

$$E_{p_2^{(i)}}[\log_e \frac{p_2^{(i)}(u_k)}{p_1^{(i)}(u_k)}] \approx -u_k^{(i)}\Delta L_{2e}^{(i)}(u_k)\frac{1}{1+e^{|L_2^{(i)}(u_k)|}} + e^{-|L_1^{(i)}(u_k)|} - e^{-|L_2^{(i)}(u_k)|}$$

$$\approx -u_k^{(i)}\Delta L_{2e}^{(i)}(u_k)\frac{1}{1+e^{|L_1^{(i)}(u_k)|+u_k^{(i)}\Delta L_{2e}^{(i)}(u_k)}} + e^{-|L_1^{(i)}(u_k)|} - e^{-|L_1^{(i)}(u_k)|-u_k^{(i)}\Delta L_{2e}^{(i)}(u_k)}$$

$$\approx e^{-|L_1^{(i)}(u_k)|}(\frac{-u_k^{(i)}\Delta L_{2e}^{(i)}(u_k)}{e^{-|L_1^{(i)}(u_k)|}+e^{u_k^{(i)}\Delta L_{2e}^{(i)}(u_k)}} + 1 - e^{-u_k^{(i)}\Delta L_{2e}^{(i)}(u_k)})$$

$e^{-|L_1^{(i)}(u_k)|} \approx 0 \Rightarrow$ $$\approx e^{-|L_1^{(i)}(u_k)|}(1 - u_k^{(i)}\Delta L_{2e}^{(i)}(u_k)e^{-u_k^{(i)}\Delta L_{2e}^{(i)}(u_k)} - e^{-u_k^{(i)}\Delta L_{2e}^{(i)}(u_k)})$$

$$\approx e^{-|L_1^{(i)}(u_k)|}(1 - e^{-u_k^{(i)}\Delta L_{2e}^{(i)}(u_k)}(1+u_k^{(i)}\Delta L_{2e}^{(i)}(u_k))). \qquad (3.22)$$

It is well known that the exponential series for e^x is $e^x = 1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \cdots$. Based on this series, it is straightforward to show that when x is negative and $|x| < 1$, $e^x \approx 1 + x$. For example, if $x = -0.2$, then $e^x = 0.819$, and $1 + x = 0.8$. In relation (3.22), if at the i-th iteration $\Delta L_{2e}^{(i)}(u_k)$ holds the same sign as $u_k^{(i)}$ and has a magnitude less than one, the first two terms of the exponential series dominate its value and:

$$E_{p_2^{(i)}}[\log_e \frac{p_2^{(i)}(u_k)}{p_1^{(i)}(u_k)}] \approx e^{-|L_1^{(i)}(u_k)|}(1 - (1 - u_k^{(i)}\Delta L_{2e}^{(i)}(u_k))(1 + u_k^{(i)}\Delta L_{2e}^{(i)}(u_k)))$$

$$\approx (\Delta L_{2e}^{(i)}(u_k))^2 e^{-|L_1^{(i)}(u_k)|}. \qquad (3.23)$$

Combining this result with relation (3.12), the CE for the turbo decoder at the i-th iteration after the second constituent decoder can be approximated as:

$$T_2^{(i)} \approx \sum_{k=1}^{N} (\Delta L_{2e}^{(i)}(u_k))^2 e^{-|L_1^{(i)}(u_k)|}, \quad i \geq 1. \tag{3.24}$$

Similarly, after decoder 1 during iteration i, the CE $T_1^{(i)}$ is:

$$T_1^{(i)} \approx \sum_{k=1}^{N} (\Delta L_{1e}^{(i)}(u_k))^2 e^{-|L_2^{(i)}(u_k)|}, \quad i \geq 2. \tag{3.25}$$

Simulation results have shown that when $T_m^{(i)} / T_2^{(1)} = 10^{-2} \sim 10^{-4}$ $(m = 1,2)$, iterative decoding should be stopped [5]. $T_m^{(i)} / T_2^{(1)}$ is called the normalized CE. The normalized CE for the 1st constituent encoder is:

$$T_1^{(i)} / T_2^{(1)} \approx \frac{1}{T_2^{(1)}} \sum_{k=1}^{N} (\Delta L_{1e}^{(i)}(u_k))^2 e^{-|L_2^{(i)}(u_k)|}, \quad i \geq 2. \tag{3.26}$$

The normalized CE for the 2nd constituent encoder is:

$$T_2^{(i)} / T_2^{(1)} \approx \frac{1}{T_2^{(1)}} \sum_{k=1}^{N} (\Delta L_{2e}^{(i)}(u_k))^2 e^{-|L_1^{(i)}(u_k)|}, \quad i \geq 1. \tag{3.27}$$

Figure 3.4 shows normalized CE values for the five frames considered in Figures 3.2 and 3.3.

Therefore, to implement CE for early stopping, specify the threshold of normalized CE and the maximum number of iterations. At the 1st iteration after the 2nd constituent decoder calculate $T_2^{(1)}$ using (3.24). Then during subsequent iterations, examine the normalized CE after the 1st constituent decoder using relation (3.26) or after the 2nd constituent decoder using relation (3.27). Compare the normalized CE with the specified threshold. If the threshold is satisfied at either decoder, stop the iterative process and made a decision from the sign of the LLRs of that constituent decoder. If the threshold is not satisfied, continue the iterative process. For the CE stopping criterion, it can be seen that the minimum number of iterations for decoding is 1.5 owing to evaluation of $T_2^{(1)}$.

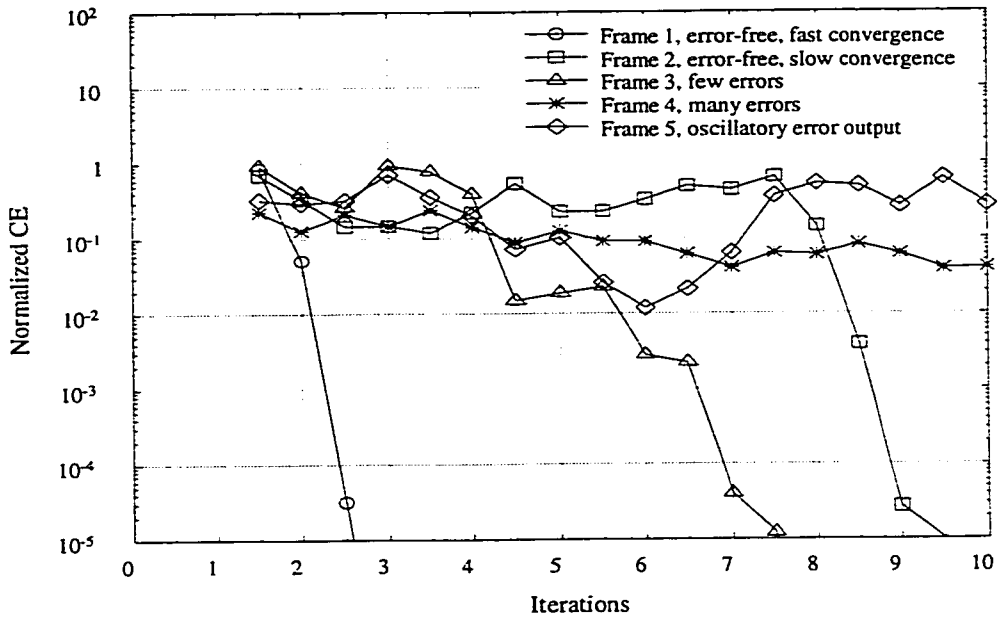


Figure 3.4: Normalized cross entropy for five typical frames (see Figure 3.2 for the corresponding number of errors).

3.2.3 Sign-Change-Ratio and Hard-Decision-Aided

Two simple stopping criteria based on the concept of CE have been proposed: sign-change-ratio (SCR) and hard-decision-aided (HDA) [38]. SCR considers the sign change between extrinsic information $\bar{L}_{me}^{(i-1)}$ and $\bar{L}_{me}^{(i)}$ $(m=1,2)$ from iteration $i-1$ to i $(i \geq 2)$. Sign changes of the LLR $\bar{L}_{m}^{(i-1)}$ and $\bar{L}_{m}^{(i)}$ $(m=1,2)$ from iteration $i-1$ to i $(i \geq 2)$ are the basis of the HDA technique. Both are described below. Suppose that the iterative decoding process converges, and at iteration i the decoding process can be terminated. Then, four assumptions are made regarding the LLR and extrinsic values at the outputs of two constituent decoders at iteration i [38]:

(1) Hard decisions of the information bits based on the LLR values do not change anymore, i.e., $\text{sign}\left(L_1^{(i)}(u_k)\right)=\text{sign}\left(L_2^{(i)}(u_k)\right)=u_k^{(i)}=\pm 1$.

(2) The magnitudes of LLR values are very large.

(3) $\text{sign}\left(\Delta L_{me}^{(i)}(u_k)\right)=\text{sign}(u_k^{(i)})$.

(4) $|\Delta L_{me}^{(i)}(u_k)|<1$.

A. SCR

Consider the calculation of $T_2^{(i)}$ given by (3.24). In the iterative decoding process, the values of LLR and extrinsic information are different for each bit. From simulation, it can be shown that when $|\Delta L_{2e}^{(i)}(u_k)|<1$, if there are no sign changes between $L_{2e}^{(i-1)}(u_k)$ and $L_{2e}^{(i)}(u_k)$, the value of $\Delta L_{2e}^{(i)}(u_k)$ is negligible when compared to the value of $\Delta L_{2e}^{(i)}(u_k)$ for positions where there are sign changes [38]. That is, when $k \in A_k \triangleq \left\{k : \text{sign}\left(L_{2e}^{(i)}(u_k)\right) = \text{sign}\left(L_{2e}^{(i-1)}(u_k)\right)\right\}$, $\Delta L_{2e}^{(i)}(u_k) \approx 0$, where A_k denotes the set of position k such that $L_{2e}^{(i-1)}(u_k)$ and $L_{2e}^{(i)}(u_k)$ have the same sign. Therefore, in a frame, the terms for calculation of $T_2^{(i)}$ can be separated into two groups. Let $T_2^{(i)}(1)$ denote those terms that contribute little to $T_2^{(i)}$, and let $T_2^{(i)}(2)$ denote those bits which have sign changes and therefore have greater contribution to the value of $T_2^{(i)}$. Then:

$$T_2^{(i)} \approx T_2^{(i)}(1) + T_2^{(i)}(2) \approx \sum_{k \in A_k} \frac{\left(\Delta L_{2e}^{(i)}(u_k)\right)^2}{e^{|L_1^{(i)}(u_k)|}} + \sum_{k \notin A_k} \frac{\left(\Delta L_{2e}^{(i)}(u_k)\right)^2}{e^{|L_1^{(i)}(u_k)|}}, \quad i \geq 2. \tag{3.28}$$

Furthermore, the average value of $|L_1^{(i)}(u_k)|$ in $T_2^{(i)}(1)$ is much larger than the corresponding value for $T_2^{(i)}(2)$. Therefore, $T_2^{(i)}(1)$ is negligible compared with $T_2^{(i)}(2)$, and:

$$T_2^{(i)} \approx \sum_{k \notin A_k} \frac{\left(\Delta L_{2e}^{(i)}(u_k)\right)^2}{e^{|L_1^{(i)}(u_k)|}} \approx C_{L_{2e}}^{(i)} \delta^{(i)}, \quad i \geq 2 \tag{3.29}$$

where $\delta^{(i)}$ denotes the average value of $\left(\Delta L_{2e}^{(i)}(u_k)\right)^2 / e^{|L_1^{(i)}(u_k)|}$ for $k \notin A_k$, and $C_{L_{2e}}^{(i)}$ denotes the number of sign changes of $\bar{L}_{2e}^{(i)}$ in a frame from iteration $i-1$ to i ($i \geq 2$).

Relation (3.29) shows that $T_2^{(i)}$ varies directly as $C_{L_{2e}}^{(i)}$. This relationship provides a stopping criterion based on the number of sign changes $C_{L_{2e}}^{(i)}$. The ratio $C_{L_{2e}}^{(i)} / N$ is called the sign-change-ratio, and when SCR drops to $0.005 \sim 0.03$, iterative decoding can be stopped with about the same performance degradation as when the CE criterion is used [38]. This criterion also holds for $\bar{L}_{1e}^{(i)}$ ($i \geq 2$). Figure 3.5 shows SCR values with iterative decoding for the five typical frames considered previously.

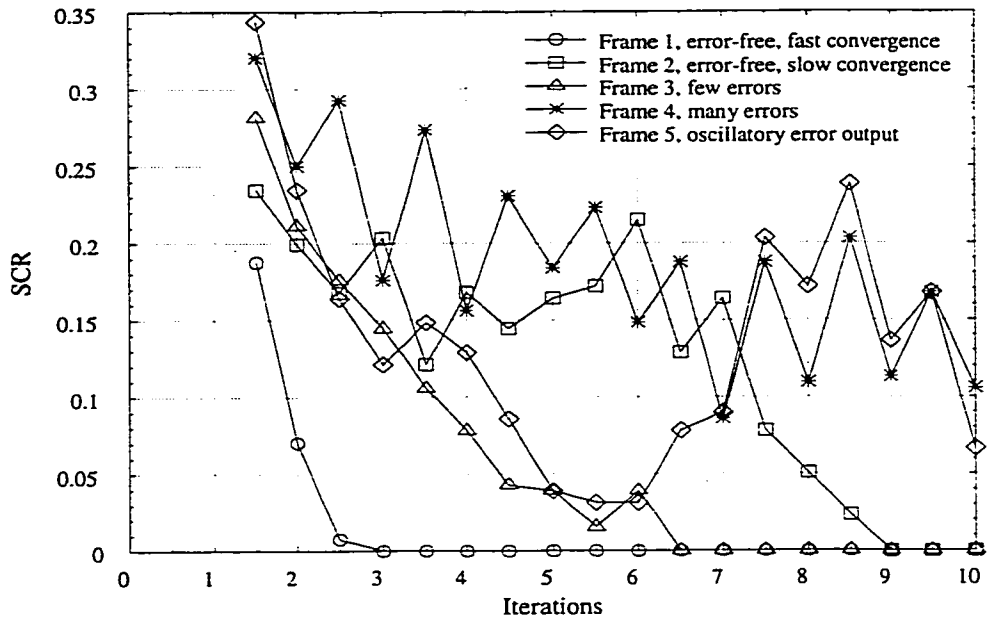


Figure 3.5: Sign-change-ratio with different iterations for five typical frames (see Figure 3.2 for the corresponding number of errors).

B. HDA

Decisions regarding the values of the information bits are based on the sign of the LLR values. The HDA stopping criterion is based on comparison of these values from one iteration to the next. At iteration $i-1 (i \geq 2)$, store the hard decision values based on $\overline{L}_2^{(i-1)}$. One iteration later, compare these values to the hard decision values corresponding to $\overline{L}_2^{(i)}$. If they agree with each other for the entire frame, the iterative process can be stopped at iteration i. This criterion also holds for $\overline{L}_1^{(i)}$. In the following, the connection between CE and HDA is described.

Consider the CE between $P_2^{(i-1)}$ and $P_2^{(i)}$, where $P_2^{(i-1)} = \prod_{k=1}^{N} p_2^{(i-1)}(u_k)$ and $P_2^{(i)} = \prod_{k=1}^{N} p_2^{(i)}(u_k)$.

The CE $T_{L_2}^{(i)}$ is:

$$T_{L_2}^{(i)} = \sum_{k=1}^{N} E_{p_2^{(i)}} \left[\log_e \frac{p_2^{(i)}(u_k)}{p_2^{(i-1)}(u_k)} \right]. \tag{3.30}$$

Assume that no sign changes occur between $\overline{L}_2^{(i-1)}$ and $\overline{L}_2^{(i)}$ from iteration $i-1$ to i, which means that $u_k^{(i)} = \text{sign}\left(L_2^{(i-1)}(u_k)\right) = \text{sign}\left(L_2^{(i)}(u_k)\right)$ and let $\Delta L_2^{(i)}(u_k) = L_2^{(i)}(u_k) - L_2^{(i-1)}(u_k)$, so that $u_k^{(i)} \Delta L_2^{(i)}(u_k) = |L_2^{(i)}(u_k)| - |L_2^{(i-1)}(u_k)|$. Then:

$$T_{L_2}^{(i)} \approx \sum_{k=1}^{N} \frac{e^{L_2^{(i)}(u_k)}}{1+e^{L_2^{(i)}(u_k)}} \Delta L_2^{(i)}(u_k) + \log_e \frac{1+e^{L_2^{(i-1)}(u_k)}}{1+e^{L_2^{(i)}(u_k)}}. \tag{3.31}$$

Assume that the magnitude of $L_2^{(i)}(u_k)$ is very large. If $u_k^{(i)} = +1$, then (3.31) becomes:

$$T_{L_2}^{(i)} \approx \sum_{k=1}^{N} u_k^{(i)} \Delta L_2^{(i)}(u_k) + \log_e \frac{1+e^{|L_2^{(i-1)}(u_k)|}}{1+e^{|L_2^{(i)}(u_k)|}}. \tag{3.32}$$

If $u_k^{(i)} = -1$, from (3.31), it follows that:

$$\begin{aligned}
T_{L_2}^{(i)} &\approx \sum_{k=1}^{N} \frac{e^{-|L_2^{(i)}(u_k)|}}{1+e^{-|L_2^{(i)}(u_k)|}} \Delta L_2^{(i)}(u_k) + \log_e \frac{1+e^{-|L_2^{(i-1)}(u_k)|}}{1+e^{-|L_2^{(i)}(u_k)|}} \\
&\approx \sum_{k=1}^{N} \frac{e^{-|L_2^{(i)}(u_k)|}}{1+e^{-|L_2^{(i)}(u_k)|}} \Delta L_2^{(i)}(u_k) + \log_e \frac{e^{|L_2^{(i)}(u_k)|}\left(1+e^{|L_2^{(i-1)}(u_k)|}\right)}{e^{|L_2^{(i-1)}(u_k)|}\left(1+e^{|L_2^{(i)}(u_k)|}\right)} \\
&\approx \sum_{k=1}^{N} \frac{e^{-|L_2^{(i)}(u_k)|}}{1+e^{-|L_2^{(i)}(u_k)|}} \Delta L_2^{(i)}(u_k) + \left(|L_2^{(i)}(u_k)| - |L_2^{(i-1)}(u_k)|\right) + \log_e \frac{1+e^{|L_2^{(i-1)}(u_k)|}}{1+e^{|L_2^{(i)}(u_k)|}} \\
&\approx \sum_{k=1}^{N} -u_k^{(i)} \frac{e^{-|L_2^{(i)}(u_k)|}}{1+e^{-|L_2^{(i)}(u_k)|}} \Delta L_2^{(i)}(u_k) + u_k^{(i)} \Delta L_2^{(i)}(u_k) + \log_e \frac{1+e^{|L_2^{(i-1)}(u_k)|}}{1+e^{|L_2^{(i)}(u_k)|}} \\
&\approx \sum_{k=1}^{N} u_k^{(i)} \Delta L_2^{(i)}(u_k) \frac{1}{1+e^{-|L_2^{(i)}(u_k)|}} + \log_e \frac{1+e^{|L_2^{(i-1)}(u_k)|}}{1+e^{|L_2^{(i)}(u_k)|}} \\
&\approx \sum_{k=1}^{N} u_k^{(i)} \Delta L_2^{(i)}(u_k) + \log_e \frac{1+e^{|L_2^{(i-1)}(u_k)|}}{1+e^{|L_2^{(i)}(u_k)|}}.
\end{aligned} \tag{3.33}$$

Therefore, considering (3.32) and (3.33), for any $u_k^{(i)}$ it is clear that:

$$T_{L_2}^{(i)} \approx \sum_{k=1}^{N} u_k^{(i)} \Delta L_2^{(i)}(u_k) + \log_e \frac{1+e^{|L_2^{(i-1)}(u_k)|}}{1+e^{|L_2^{(i)}(u_k)|}}. \tag{3.34}$$

Simulation shows that when no sign changes occur between $L_2^{(i-1)}(u_k)$ and $L_2^{(i)}(u_k)$, and the magnitude of $L_2^{(i)}(u_k)$ is large, $\Delta L_2^{(i)}(u_k)$ is very small [38]. This causes $T_{L_2}^{(i)}$ to be small, which indicates that the iterative process can be stopped. Figure 3.6 shows the number of sign changes of LLR ($\overline{L}_1^{(i)}$ or $\overline{L}_2^{(i)}$) for five typical frames. Generally, for any given interleaver size, simulation

shows that when compared to CE and SCR, HDA can save more iterations at low to medium values of SNR, but that it is not as efficient at high values of SNR [38].

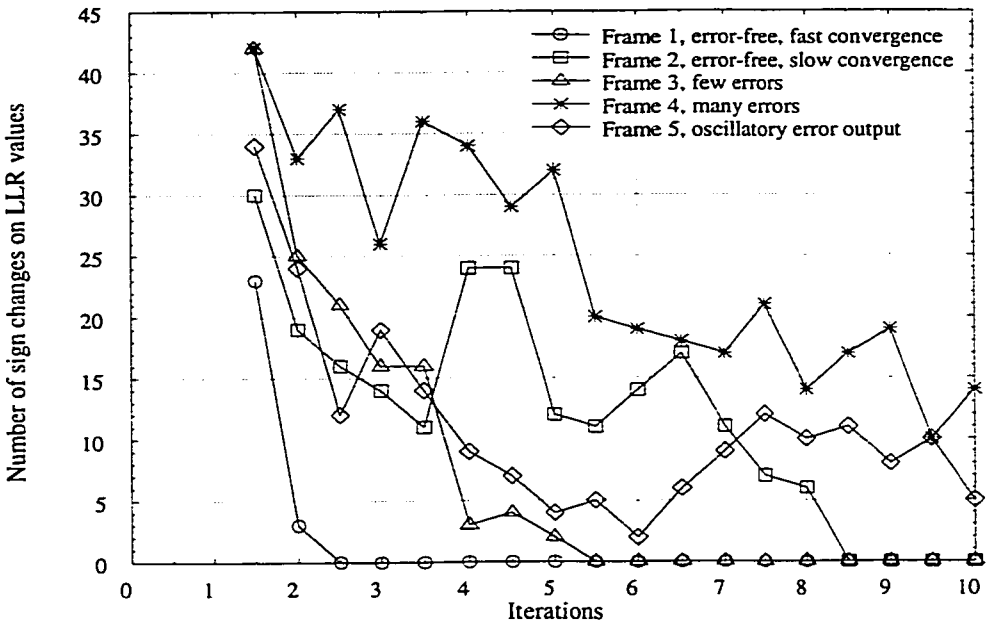


Figure 3.6: Number of sign changes of LLR for five typical frames. (see Figure 3.2 for the corresponding number of errors).

3.2.4 Turbo-CRC

The idea of the turbo-CRC technique is that by incorporating a CRC into turbo codes, a CRC check can be used to reduce the average number of iterations of decoding [31]. The CRC encoder is introduced before the turbo encoder. After each constituent decoder, hard decisions are made and a CRC error check is performed. If no errors are found, the iterative process can be stopped.

In [31], the information data is divided into a sequence of 5104-bit frames. Each frame of data is first encoded using a 16-bit CRC code and then encoded by turbo codes with $G(13,15)$. The resulting performance is very good but 16 redundant CRC parity bits, which reduce the effective code rate, have to be introduced in each frame.

CHAPTER 4

NEW APPROACHES FOR COMBINING EARLY STOPPING WITH ERROR DETECTION

When turbo codes are used for data transmission, the decoded sequence must be examined to determine whether or not the frames contain errors. In an automatic-repeat-request (ARQ) data transmission system, if any errors are detected, a retransmission request for whole frame will be triggered. Buckley and Wicker [40] [41] have introduced an error estimation scheme for turbo decoders in which a neural network attempts to determine the presence of errors using the CE of the component decoders as input. In this chapter, new, simple and efficient methods to stop the iterative decoding process and detect errors are presented. When the frame is thought to be error-free, the iterative decoding process is stopped.

4.1 Error Detection

The flowchart in Figure 4.1 demonstrates what can happen when an error detection mechanism is used. In this example there are 1000 frames of data, 100 of which are not decoded correctly.

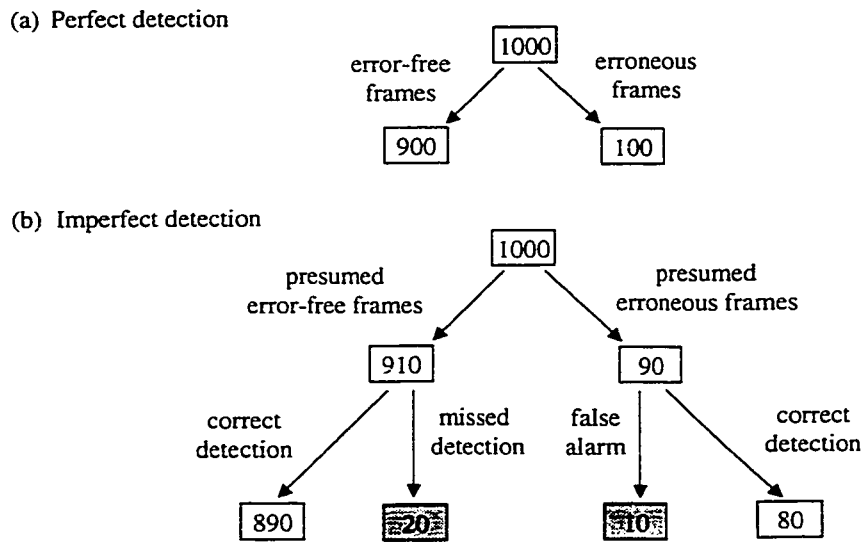


Figure 4.1: Flowchart to illustrate error detection.

If error detection is perfect, the 900 error-free frames and 100 erroneous frames will be correctly identified. But if an imperfect detection approach is used, some invalid frames might be judged to be without errors (20 frames with missed detection) and some correct frames might be considered invalid (10 frames with false alarm). The goal of error detection is to reduce the occurrence of missed detection and false alarm to acceptable levels.

4.2 Error Detection by Use of a Neural Network with CE

Buckley and Wicker showed that neural networks can be trained to use cross entropy of turbo codes [5] to detect errors in the decoded frame of data. The neural networks they designed to do so include a future error detecting network (FEDN-CE), a decoder error detecting network (DEDN-CE) and a "backup" DEDN (DEDNpost-CE). The objective of the FEDN-CE is to determine whether or not the frame is likely to be accurately decoded in future iterations, and if not, terminating the decoding process early to minimize decoder complexity. The CE values of the first two iterations are provided to the FEDN. The DEDN-CE is designed to maximize the reliability. The DEDN-CE is provided the CE values from the first two iterations and the last two iterations before the iterative decoding process is stopped. The DEDNpost-CE checks all frames that are accepted by the FEDN-CE, providing enhanced reliability. Like the DEDN-CE, the input values for the DEDNpost-CE are the CE values from the first two iterations and the last two iterations. Design of their networks requires specification of a parameter K that can be viewed as a relative emphasis of test frames with errors during network training. Selection of this parameter is crucial to the performance of their systems [40].

Table 4.1 summarizes the error detection performance obtained by Buckley and Wicker for their neural networks for turbo codes [40, 41], where MDR and FAR denote missed detection rate and false alarm rate respectively. These results are averaged for E_b / N_0 values of 0 through 0.8 dB. The simulations were carried out for the AWGN channel, frame length 1023, code rate 1/3, $G(31,33)$ and $G(31, 27)$ constituent codes, a maximum of 10 iterations, and CE threshold 0.01; the interleaver type was not mentioned.

Table 4.1: Performance of neural networks for the error detection

Network	K	MDR	FAR	Average errors per missed detection frame
DEDN-CE	20	4.8×10^{-4}	1.3×10^{-2}	8.2
FEDN-CE	0.05	0.68	7.7×10^{-3}	26.2
DEDNpost-CE	20	8.6×10^{-4}	7.1×10^{-3}	26.2

4.3 Mean Estimate

In this section, the relationship between the mean of the absolute LLR values and the number of errors as iterations proceed, is discussed. The mean estimate criterion for stopping and error detection is then introduced.

4.3.1 The Number of Errors and the Mean of Absolute LLR Values

Figure 4.2 shows typical LLR values for ten frames of data at the output of a component decoder after ten iterations. It can immediately be seen that there exist large differences in average absolute LLR values between frames. This difference can be used to determine when to stop the decoding process and detect errors. The new approaches are based on monitoring $M_{|L|}$, the mean of the absolute values of the constituent decoder LLR values. From simulation it can be shown that as iterations proceed, $M_{|L|}$ increases as the number of errors in a frame decreases. How the mean $M_{|L|}$ and corresponding errors of five typical frames change with iterations is shown in Figures 4.3 and 4.4. Note that Figure 3.2 is redrawn as Figure 4.3 here for convenience to compare with other stopping criteria. From these figures, it can be seen that in general, the mean of the absolute LLR values is inversely proportional to the number of errors in the frames. This mean value will increase as the number of errors decreases, it will decrease as the number of errors increase, and it will remain small if many errors continue to exist. A more rigorous foundation for these observations is given in the next subsection.

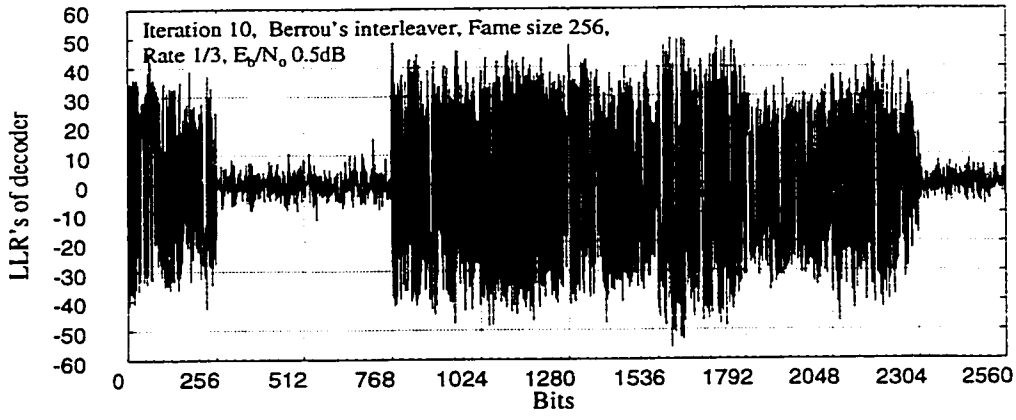


Figure 4.2: LLR values for ten frames under identical channel conditions.

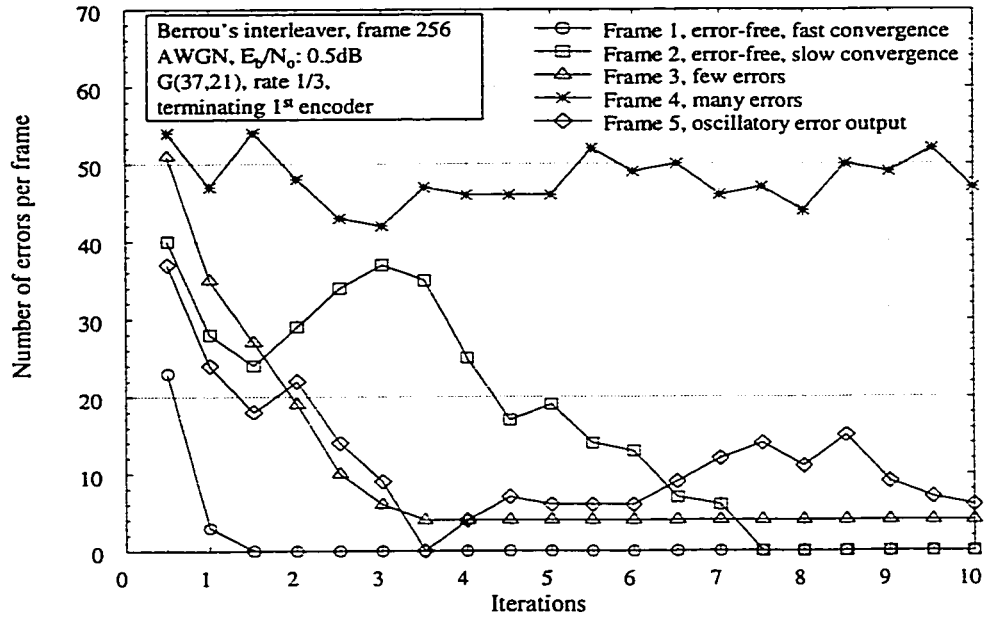


Figure 4.3: The number of errors with iterations for 5 typical frames.

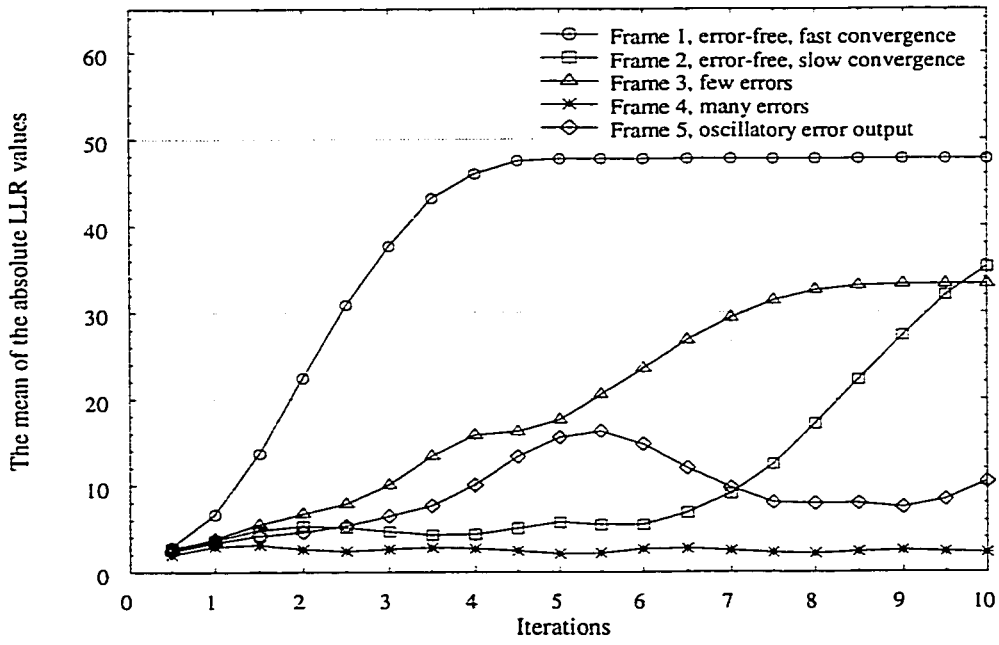


Figure 4.4: The mean of the absolute LLR values with iterations for 5 typical frames.

4.3.2 The Mean of Absolute LLR Values and the Variance of the Meta-Channel

Recall that the meta-channel for turbo decoding was introduced in the discussion of Berrou's decoding algorithm (subsection 2.3.3) and Robertson's variance estimate criterion (subsection 3.2.1). The meta-channel is modeled as AWGN with zero mean and variance σ_m^2, where $x \in \{-1,+1\}$ is transmitted and z is the received value. The LLR $L(x)$, given observation z, is available after each constituent decoder during turbo decoding. From relation (3.2), considering the mean absolute values yields:

$$E[|L(x)|] = \frac{2}{\sigma_m^2} E[|z|]. \tag{4.1}$$

Also from relation (2.79), it is clear that:

$$E[|z|] = \sqrt{2/\pi}\,\sigma_m e^{-1/(2\sigma_m^2)} + \operatorname{erf}\left(1/\sqrt{2\sigma_m^2}\right). \tag{4.2}$$

Through iteration, turbo decoding decreases the variance σ_m^2. Figure 4.5 plots the relationship between $E[|z|]$ with σ_m^2 for values of σ_m^2 in the range 0.01 to 1.2.

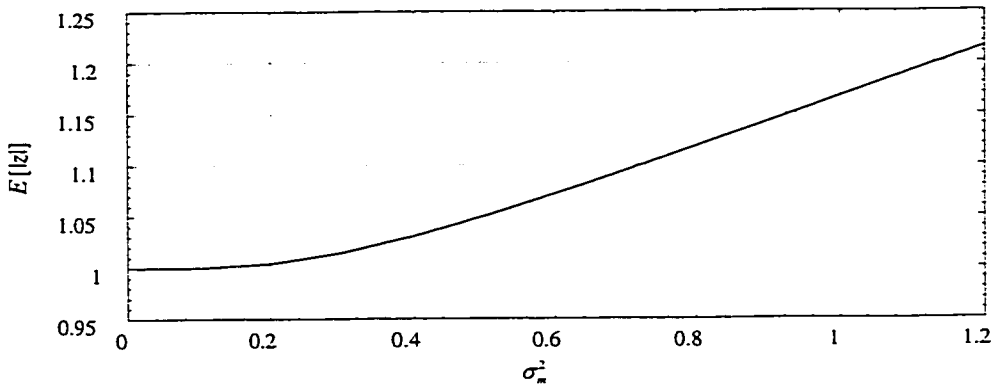


Figure 4.5: Relationship between the mean of the absolute value of z and σ_m^2.

Relation (4.1) yields:

$$\sigma_m^2 = E[|z|] \frac{2}{E[|L(x)|]}. \tag{4.3}$$

From Figure 4.5 it can be seen that when σ_m^2 is less than 0.2, $E[|z|] \approx 1.0$, and therefore:

$$\sigma_m^2 \approx \frac{2}{E[|L(x)|]}, \text{ for } E[|L(x)|] > 10. \tag{4.4}$$

It is known that σ_m^2 can be calculated from the second moment of $L(x)$ according to relation (3.8). Also relation (4.4) suggests σ_m^2 can also be approximated from $E[|L(x)|]$. Figure 4.6 shows the estimate of σ_m^2 from $M_{L^2} = E[L^2(x)]$ and $M_{|L|} = E[|L(x)|]$ for five typical frames. In these simulations, expectations are replaced by the time average within a frame. From this figure, it can be seen that the values estimated from both M_{L^2} and $M_{|L|}$ approach equality when σ_m^2 is small, but that the estimated values from $M_{|L|}$ are slightly less than the values from M_{L^2} for larger values of σ_m^2. This occurs because $E[|z|]$ is greater than 1.0 for large σ_m^2, and therefore the accuracy of the approximation of (4.4) decreases. Also because $M_{|L|}$ is inversely proportional to the number of errors in a frame, σ_m^2 is proportional to this number of errors.

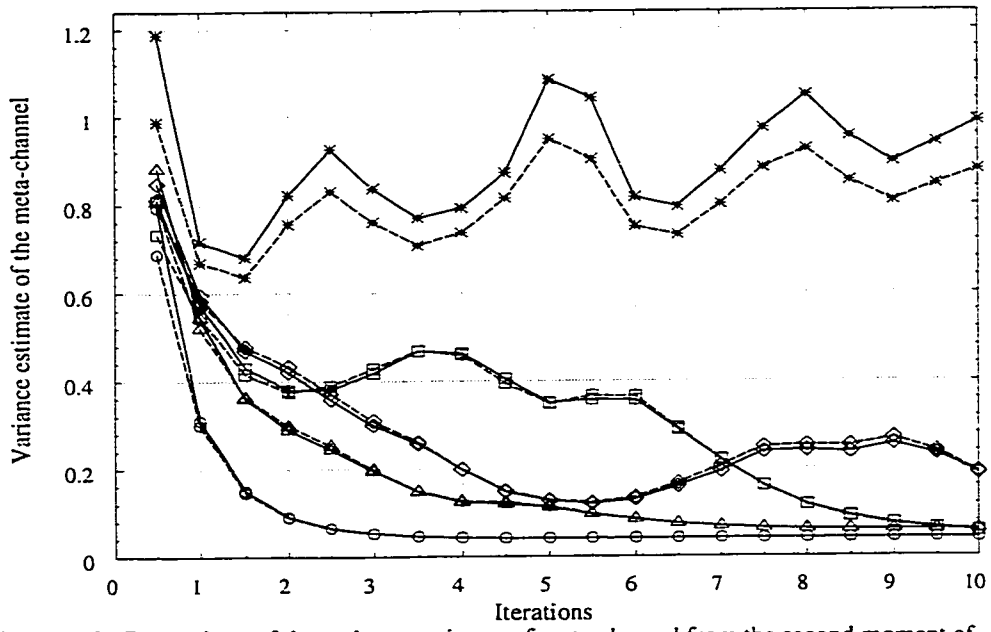


Figure 4.6: Comparison of the variance estimate of meta-channel from the second moment of the LLR values and the mean of the absolute LLR values for five typical frames (see Figure 4.3 for the corresponding number of the errors).

- ○ Frame 1, error-free, fast convergence
- □ Frame 2, error-free, slow convergence
- △ Frame 3, few errors
- ✳ Frame 4, many errors
- ◇ Frame 5, oscillatory error output
- —— M_{L^2}
- —— $M_{|L|}$

4.3.3 New Mean Estimate Criterion (ME)

Given the relationship between $M_{|L|}$ and number of errors in decoded frames, a new stopping criterion and error detection technique called mean estimate is proposed. This technique involves continuously monitoring $M_{|L|}$ and, based on this value, making a judgement as to whether or not the most recently decoded frame contains errors. $M_{|L|}$ is calculated from the LLR values after constituent decoder j during iteration i according to:

$$M_{|L|} = \frac{1}{N}\sum_{k=1}^{N} | L_j^{(i)}(u_k) | . \tag{4.5}$$

First, specify MAX_{it}, the maximum number of iterations allowed, and a threshold Th. The ME criterion is as follows:

> Compare $M_{|L|}$ to the threshold Th. If $M_{|L|} > Th$, stop the iterative process and consider the frame to be error-free, otherwise continue iterating. If $M_{|L|} \le Th$ for MAX_{it}, consider the frame to be erroneous.

Although the ME approach can be implemented easily, the threshold has to be determined from simulation, and optimum values change with E_b / N_o, frame size and code rate. From relations (3.13) and (3.14), it can be seen that the magnitude of LLR's is related to the variance σ^2. When E_b / N_o increases, σ^2 decreases, and therefore the magnitude of the LLR's will increase. Figure 4.7 shows how the optimum threshold for percentage of correct detection varies with E_b / N_o, where percentage of correct detection is defined as:

$$\text{Percentage of correct detection} = (1 - \text{MDR} - \text{FAR}) \times 100\% .$$

As shown in the next chapter, this easily implemented scheme offers good performance.

4.4 New Mean-Sign-Change Criterion (MSC)

In order to decrease the average number of iterations and further enhance the performance of error detection, both the mean of the absolute LLR values and the number of sign changes of LLR values can be considered simultaneously. The sign changes considered are different from those in HDA. In HDA, signs are compared between $\bar{L}_1$ from iteration i to $i+1$ (SCL_{11}) or between $\bar{L}_2$ from iteration i to $i+1$ (SCL_{22}). In this new criterion, the sign changes between $\bar{L}_1$ and $\bar{L}_2$ (SCL_{12}) are compared for two concatenated MAP decoding processes. Simulations show that

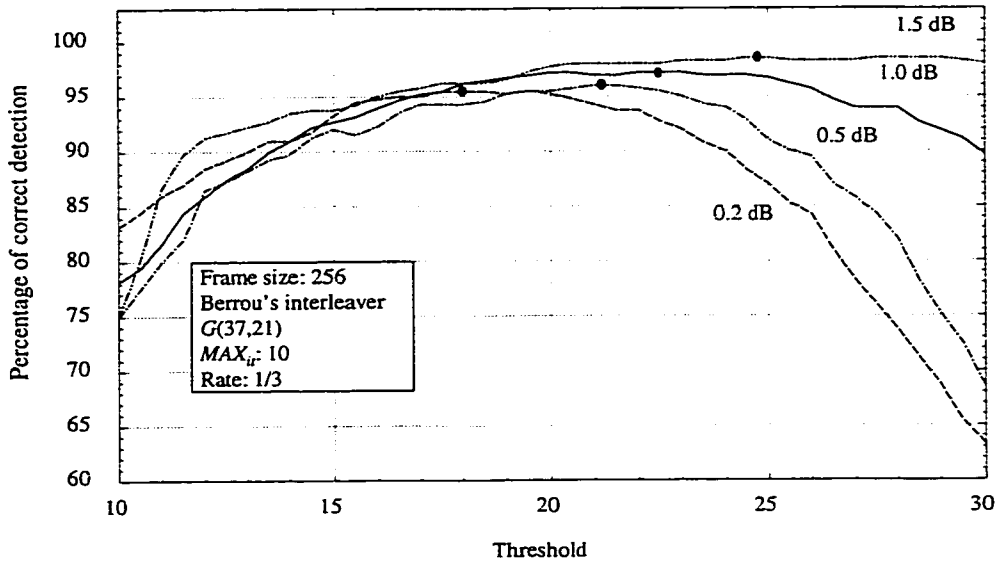


Figure 4.7: The optimum thresholds for different E_b/N_0.

using SCL_{12} can save iterations and provide better error detection. The new Mean-Sign-Change (MSC) criterion improves the performance of decoding and error detection over ME at the cost of increased complexity. For this approach, establish a base threshold, Th_b, and a limit threshold, Th_l. Set $Th_l = f \cdot Th_b$, where f is a factor greater than 1.

Having evaluated M_{1U} of the output of a component decoder, proceed with the MSC criterion as follows:

(1) If $M_{1U} > Th_l$, and if SCL_{12} is zero, stop the process and consider the frame to be error-free. If SCL_{12} is not zero for MAX_{it}, also consider the frame to be error-free.

(2) If $Th_b < M_{1U} \leq Th_l$, consider SCL_{12}. If SCL_{12} is zero, terminate the decoding process, and consider the frame to be error-free. If SCL_{12} is not zero for MAX_{it}, consider the frame to be erroneous.

(3) If $M_{1U} \leq Th_b$ up until $MAX_{it} - 0.5$, stop the decoding process after decoder 1, and consider the frame to be erroneous.

Possible scenarios for the MSC criterion are illustrated in Figure 4.8. As shown in the next chapter, this criterion results in improved performance when compared to the ME technique, at the cost of additional complexity.

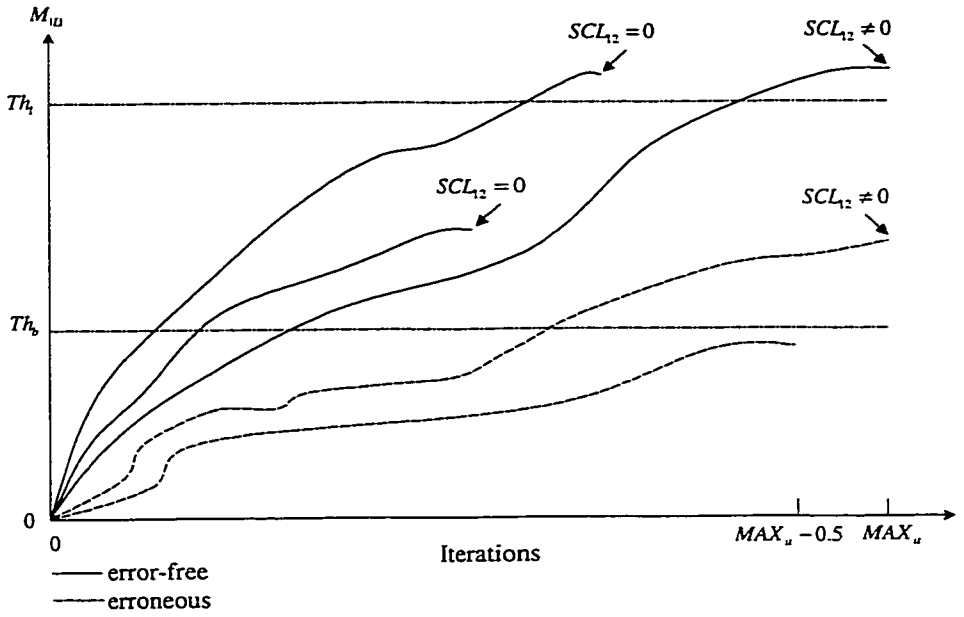


Figure 4.8: Illustration of the MSC criterion

4.5 MSC with a Short CRC

In this section, it is shown that the systematic component of a terminated recursive systematic convolutional encoder provides a built-in CRC, and that this characteristic can be used to improve the performance of the MSC criterion. In addition, by concatenating an external short CRC with the MSC criterion the error detection performance can be improved even further.

4.5.1 Use of Built-in CRC

It is well known that the recursive structure of an RSC encoder implements division over the ring of polynomials in GF(2). If an RSC encoder is terminated, its final state is all-zero, indicating that the remainder from division of the systematic sequence by the feedback polynomial is zero. This is also the premise of a cyclic redundancy check: the transmitted sequence is a multiple of the CRC polynomial [42]. This reveals that in the case of turbo coding with a terminated RSC encoder, there is an implicit CRC. Therefore, the remainder R_t of the estimated data and terminating bits divided by the feedback polynomial can be used as an indication of the accuracy of the decoded sequence.

For CRC error detection, the order of the polynomial usually is high. Three polynomials that have become international standards include CRC-12 $= D^{12} + D^{11} + D^3 + D^2 + D + 1$, CRC-16 $= D^{16} + D^{15} + D^2 + 1$, and CRC-CCITT $= D^{16} + D^{12} + D^5 + 1$ [43]. However, the order of the generator used in turbo coding is low, generally $\nu = 2$, 3, or 4. Therefore, the built-in CRC is susceptible to some patterns of errors that will cause missed detection, and cannot be used on a stand-alone basis. But it can be shown that the built-in CRC property of the turbo codes can be used in conjunction with the MSC criterion to further improve the performance of the MSC. Note that if only the first RSC encoder is terminated, R_t can be considered only after the first constituent decoder or after the deinterleaver that follows the second constituent decoder.

To simplify the decision mechanism, a simplified version of the MSC criterion outlined above is considered here. This mechanism uses only a base threshold Th_b instead of both this threshold and a limit threshold Th_l. After having evaluated $M_{1,l}$ of the output of a constituent decoder, the simplified MSC criterion is implemented as follows:

(1) If $M_{1,l} > Th_b$, consider SCL_{12}. If SCL_{12} is zero, terminate the decoding process, and consider the frame to be error-free. If SCL_{12} is not zero for MAX_{it}, consider the frame erroneous.

(2) If $M_{1,l} \leq Th_b$ up until $MAX_{it} - 0.5$, stop the decoding process after decoder 1, and consider the frame to be erroneous.

Two approaches are proposed to incorporate the built-in CRC with this simplified MSC criterion:

- *Approach 1*: If $M_{1,l} > Th_b$ and $SCL_{12} = 0$, consider R_t. If R_t is zero, terminate the decoding process, and consider the frame to be error-free. If R_t is not zero for MAX_{it}, consider the frame erroneous.
- *Approach 2*: If $M_{1,l} > Th_b$, consider R_t and SCL_{12}. If $R_t = 0$ and $SCL_{12} = 0$, terminate the decoding process, and consider the frame to be error-free. If $R_t = 0$, but SCL_{12} is not zero for MAX_{it}, also consider the frame error-free.

4.5.2 New MSC-CRC Criterion

The MSC criterion provides an easy way to detect a frame with a large number of errors; its weakness is that since it is based on an average measure of the entire frame, it may not detect a frame with very few errors. Inclusion of an external CRC prior to turbo encoding can permit

detection of frames with very few errors at the expense of additional redundancy. Note that the concept of using an external CRC for error detection with turbo decoding is not new [31], however, to date only use of standard, high-order CRC polynomials has been proposed. By exploiting knowledge of the kinds of error patterns that often miss being detected by the MSC criterion, it is shown that polynomials of much lower degree can be used for CRC detection, thereby lowering the amount of redundancy introduced into the transmitted sequence.

It is well known that the decoded error patterns most likely to occur at moderate and high SNR's with turbo codes using RSC constituent codes with a primitive feedback polynomial of degree ν are weight-2 patterns where the errors are separated by a distance $l = n(2^{\nu} - 1)$, where n is any positive integer. In order to detect decoded frames containing these highly probable error patterns, it is proposed to select an external CRC code designed specifically to detect patterns of two errors. It can be shown that choosing CRC polynomials to be low degree (not equal to ν) and primitive is sufficient for detecting errors when combined with the MSC criterion. For convenience, define:

$E_2(D)$: weight-2 error polynomial

$d(D)$: feedback polynomial of turbo codes

$C_{RC}(D)$: generator polynomial of CRC codes.

In turbo decoding, if $E_2(D)$ is divisible by $d(D)$ for both constituent decoders and l is not large, the errors will not be corrected, and also the errors cannot be detected without introducing additional redundancy. Since the likelihood of these error patterns appearing in the decoded sequence decreases with increasing l, it is likely to use a low degree $C_{RC}(D)$ in conjunction with $d(D)$ to detect the most likely weight-2 error patterns. For example, let $d(D) = 7 \leftrightarrow 111$ and $C_{RC}(D) = 45 \leftrightarrow 100101$. Therefore, the only undetectable error patterns are $E_2(D)$ with distance which is a multiple of 93 ($(2^2 - 1)(2^5 - 1) = 93$) because only in these cases do $d(D)$ and $C_{RC}(D)$ both divide $E_2(D)$.

As shown in the next chapter, for the proposed two approaches using the built-in CRC, approach 2 provides better error detection performance than approach 1. Therefore, approach 2 is used in the MSC-CRC criterion. Let R_C be the remainder associated with the short CRC. The MSC-CRC criterion works as follows:

(1) If $M_{1U} > Th_b$, consider SCL_{12}, R_t and R_C. If $SCL_{12} = 0$, $R_t = 0$ and $R_C = 0$, terminate the decoding process, and consider the frame to be error-free. If $R_t = 0$ and $R_C = 0$, but SCL_{12} is not zero for MAX_{it}, also consider the frame error-free.

(2) If $M_{1U} \leq Th_b$ up until $MAX_{it} - 0.5$, stop the decoding process after decoder 1, and consider the frame to be erroneous.

The MSC-CRC is a simple and efficient method for early stopping and, as shown in the next chapter, provides excellent performance for error detection in iterative turbo decoding.

CHAPTER 5

PERFORMANCE RESULTS

This chapter reports the performance of the new algorithms introduced in the previous chapter. The performance of these new algorithms is summarized in terms of BER, FER, average number of iterations, MDR, FAR, and the average number of errors per missed detection frame. The early stopping performance of the new algorithms is compared with the HDA and CE techniques; their error detection performance is compared with that of the neural network approaches. The performance of the new algorithms is investigated by changing the parameters of codes including constraint length, generators, frame sizes, code rate, interleavers and the maximum number of iterations allowed. Note that only the first constituent encoder was terminated for all simulations that were carried out in this chapter.

5.1 Simulation Groups and Corresponding Thresholds

The performance of the new algorithms is investigated using the four groups of simulation conditions shown in Table 5.1. The names of simulation groups are derived from the type of interleaver, the frame size and the memory (PIL900-2 uses the PIL interleaver, frame size 900 and memory $v=2$). Frame sizes of approximately 1000 were selected when obtaining performance for FER, MDR, FAR and average number of errors per missed detection frame with medium and high E_b / N_0. Shorter frame sizes were used when collecting the data for comparison of the performance within a reasonable period of time. In these simulations, two different interleavers, PIL and SR (spread random), were used.

Each simulation group was designed for a special purpose. The first group was designed to compare the performance of the new algorithms with the performance of other early stopping criteria including CE and HDA, and to compare it to the case when a fixed number of iterations, in this case six (FixIt6), are used. Their error detection performance is also compared with the neural network approaches using the results obtained from the second simulation group. The third and fourth group of simulations were designed to show the influence of code generators and memory. The thresholds used for early stopping and error detection are provided in Tables 5.2 through 5.5. The base thresholds used for MSC-CRC are identical to those for MSC. For MSC, the factor $f=3$ is used for PIL900-2 and SR1024-4, and $f=5$ for SR256-2 and SR256-3.

Table 5.1: Simulation groups and the corresponding parameters

Simulation group	Interleaver	Frame size	Code rate	MAX_{it}	ν	Code generator	Criteria
1	PIL	900	1/2	6	2	G(7,5)	FixIt6, CE, HDA, ME, and MSC
						G(7,5,15)	MSC-CRC
2	SR	1024	1/3	10	4	G(31,33)	ME and MSC
						G(31,33,45)	MSC-CRC
3	SR	256	1/3	10	2	G(7,5)	ME and MSC
						G(7,5,15)	MSC-CRC
4	SR	256	1/3	10	3	G(13,15)	ME and MSC
						G(13,15,31)	MSC-CRC

Table 5.2: Thresholds for PIL900-2

E_b/N_0 (dB) / Th. / Stopping criteria	0.0	0.5	1.0	1.5	2.0	2.5	3.0
ME	12	14	18	26	32	38	40
MSC	7	8	9	12	16	17	19
CE	10^{-3}		10^{-4}				

Table 5.3: Thresholds for SR1024-4

E_b/N_0 (dB) / Th. / Stopping criteria	0.0	0.2	0.3	0.4	0.6	0.8
ME	22	26	28	30	34	36
MSC	7	9	10	12	17	19

Table 5.4: Thresholds for SR256-2

E_b/N_0 (dB) / Th. / Stopping criteria	0.0	0.5	1.0	1.5	2.0	2.5
ME	18	20	24	28	33	36
MSC	16	18	20	22	24	26

Table 5.5: Thresholds for SR256-3

E_b/N_0 (dB) / Th. / Stopping criteria	0.0	0.5	1.0	1.5	2.0
ME	18	22	25	30	36
MSC	14	15	17	18	20

5.2 Performance with PIL900-2

In this section, the performance of the new algorithms is simulated for parameters from the simulation group PIL900-2 in Table 5.1. The performance of the new algorithms for early

stopping and error detection is shown in Figures 5.1 through 5.6; the performance comparison of the new algorithms with other stopping criteria is provided in Figures 5.7 through 5.9.

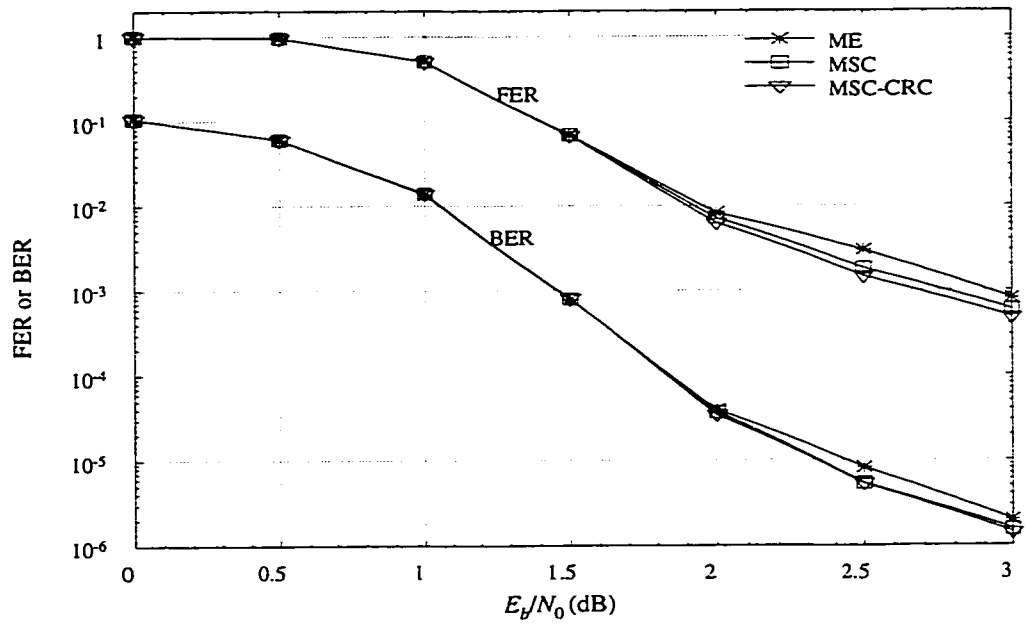


Figure 5.1: FER and BER vs. E_b/N_0 for the new algorithms in simulation group PIL900-2.

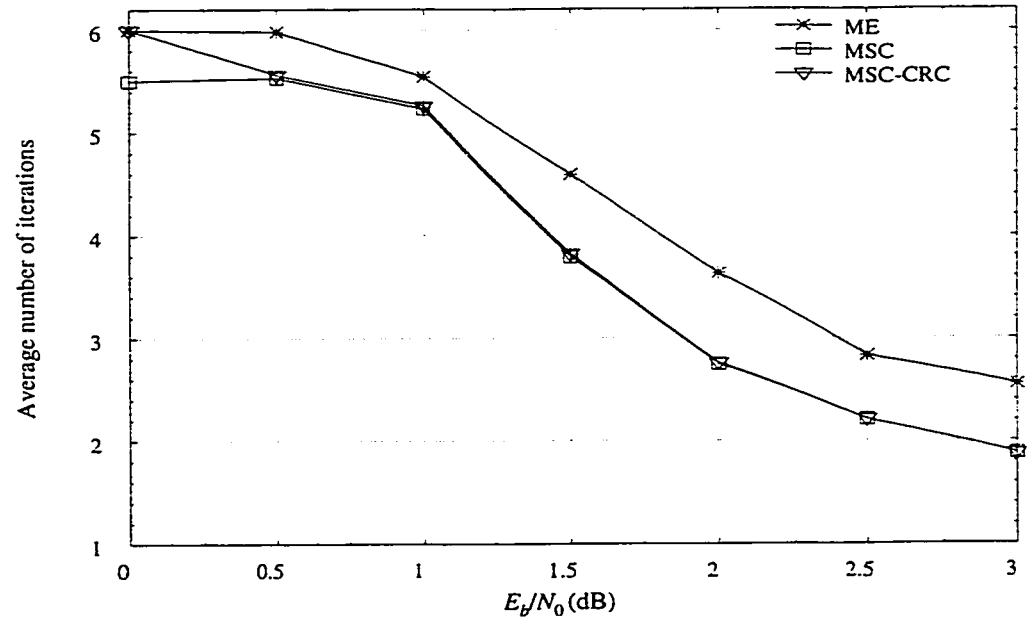


Figure 5.2: Average number of iterations vs. E_b/N_0 for the new algorithms in simulation group PIL900-2.

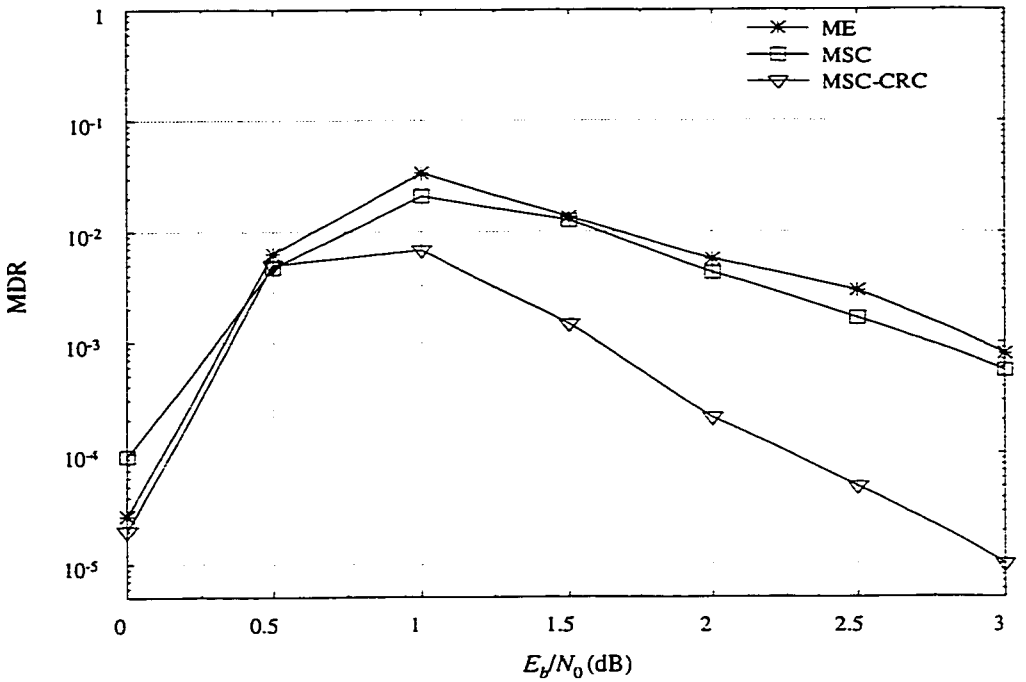


Figure 5.3: MDR vs. E_b/N_0 for the new algorithms in simulation group PIL900-2.

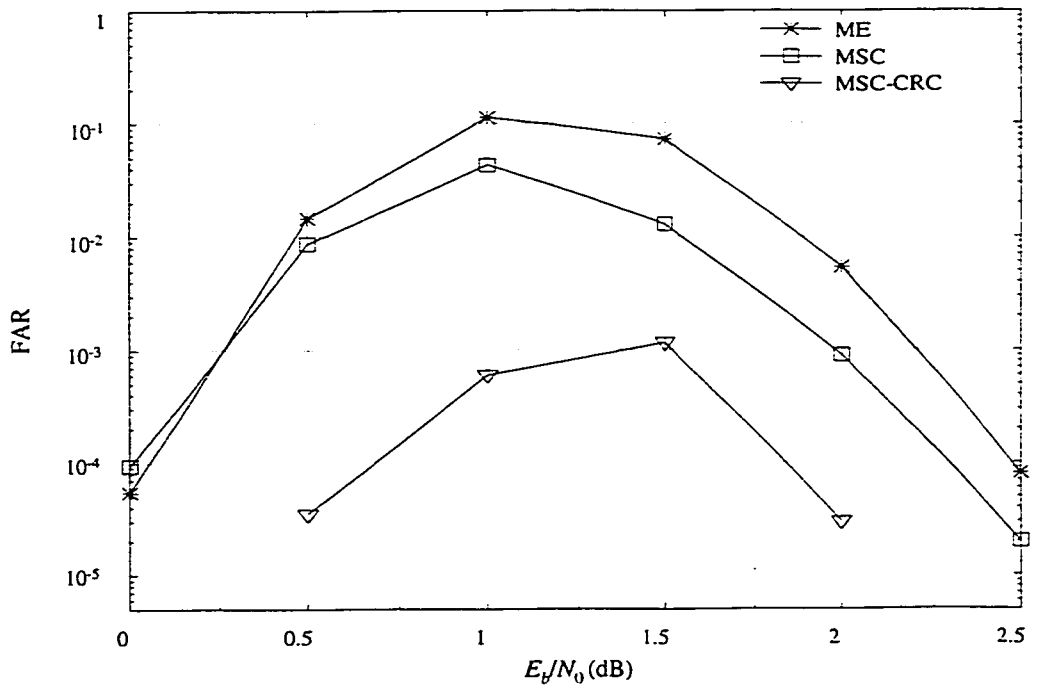


Figure 5.4: FAR vs. E_b/N_0 for the new algorithms in simulation group PIL900-2.

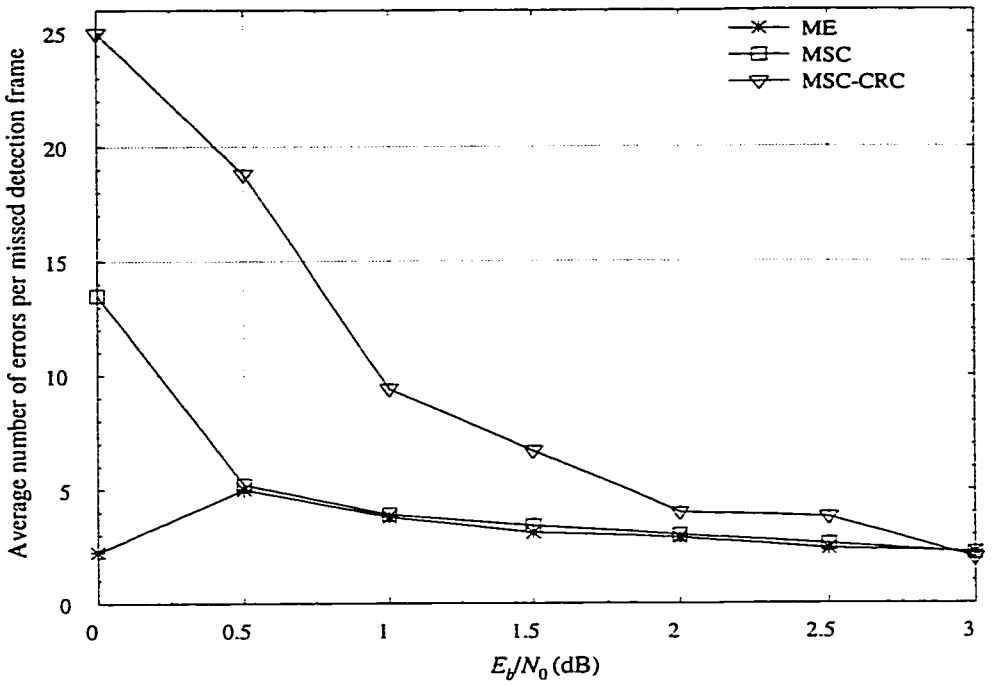


Figure 5.5: Average number of errors per missed detection frame vs. E_b/N_0 for the new algorithms in simulation group PIL900-2.

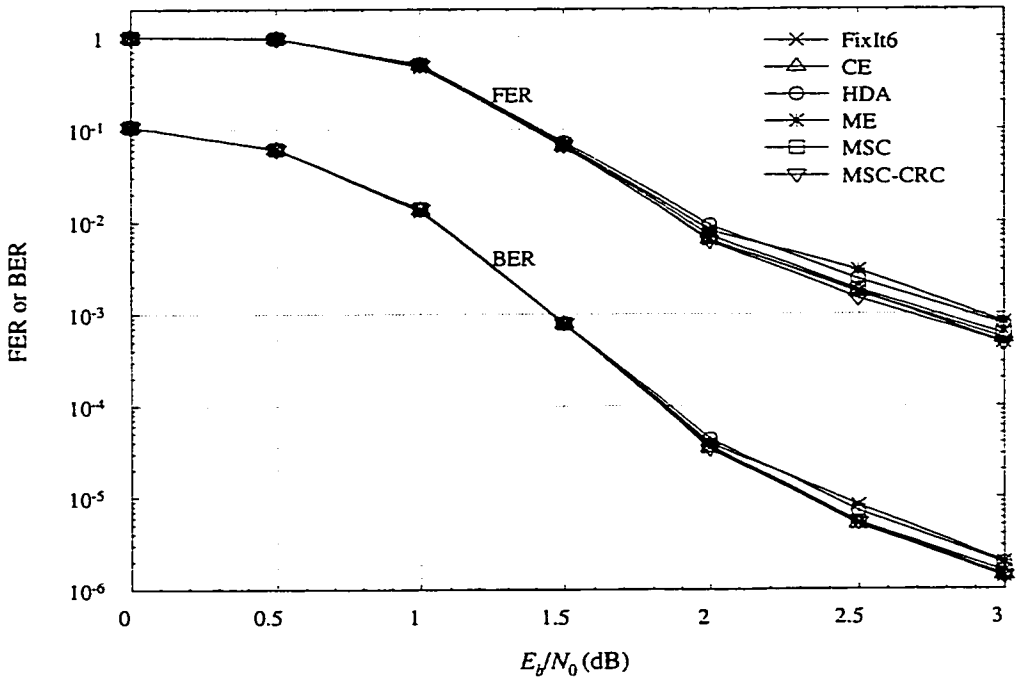


Figure 5.6: Comparison of the FER and BER for the new algorithms and other stopping criteria in simulation group PIL900-2.

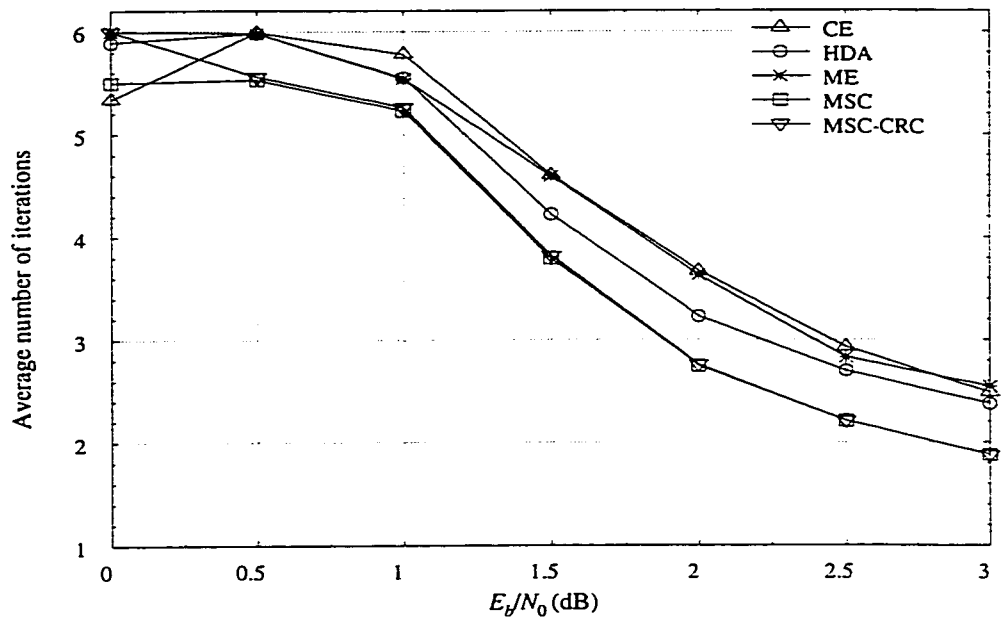


Figure 5.7: Comparison of the average number of iterations for the new algorithms and other stopping criteria in simulation group PIL900-2.

From Figures 5.1 and 5.2, regarding the early stopping performance of the three new algorithms, it can be seen that there is no difference in BER and FER when $E_b / N_0 \leq 1.5\text{dB}$. MSC-CRC criterion gives slightly better performance when $E_b / N_0 > 1.5\text{dB}$, and MSC saves the most number of iterations. Also the performance of MSC-CRC is very similar to that of MSC.

From Figures 5.3 and 5.4, regarding the error detection performance, it is clear that MSC-CRC results in the best MDR and FAR. It is interesting that the ME criterion provides MDR performance very similar to that of the MSC-CRC technique at low E_b / N_0 (say 0.0 dB). This occurs since the mean of the absolute LLR values are very small and thus it is easier to detect an erroneous frame with the simple ME criterion. Note that the curves of MDR and FAR have convex shapes ($\cap$). This means that at low or high E_b / N_0, the performance in terms of MDR and FAR is better than the case with intermediate values of E_b / N_0. Given the high BER and FER for low E_b / N_0, however, the good results for MDR and FAR at these low signal-to-noise ratios are not of much interest. Note that FAR results were not generated for $E_b / N_0 = 0.0$ dB and 2.5 dB in Figure 5.4, because the FAR was too low to simulate within a reasonable period of time.

Figure 5.5 depicts the average number of errors per missed detection frame. MSC-CRC gives the largest values, however this is to be expected since MSC-CRC technique is designed to detect frames with very few errors that cannot be detected by the ME and the MSC techniques.

Figures 5.6 and 5.7 compare the performance of the new algorithms with the performance of other stopping criteria. Note that performance with six fixed iterations is also given for purpose of comparison. In terms of BER and FER, at low to intermediate E_b / N_0, all early stopping criteria provide the same performance as FixIt6; at higher E_b / N_0, MSC, MSC-CRC and CE give the same performance as FixIt6. Figure 5.7 illustrates that MSC saves the most iterations of all simulated stopping criteria, however MSC-CRC saves nearly the same number of iterations as MSC except at very low E_b / N_0. Also the performance of ME is very similar to that of CE except at 0.0 dB.

5.3 Performance with SR1024-4

The performance of the new algorithms with simulation parameters taken from group SR1024-4 in Table 5.1 is investigated in this section. The performance of the new algorithms for early stopping and error detection is shown in Figures 5.8 to 5.12, and the performance comparison of the new algorithms and the neural network techniques is provided in Figures 5.13 to 5.15. Note that performance of the neural networks with the CE for error detection in turbo codes was not simulated in this thesis. The results obtained by Buckley and Wicker (see Table 4.1) for error detection with the neural networks was used for purposes of comparison. Their simulations of neural networks with CE (see Section 4.2) were performed with parameters very similar to those of SR1024-4.

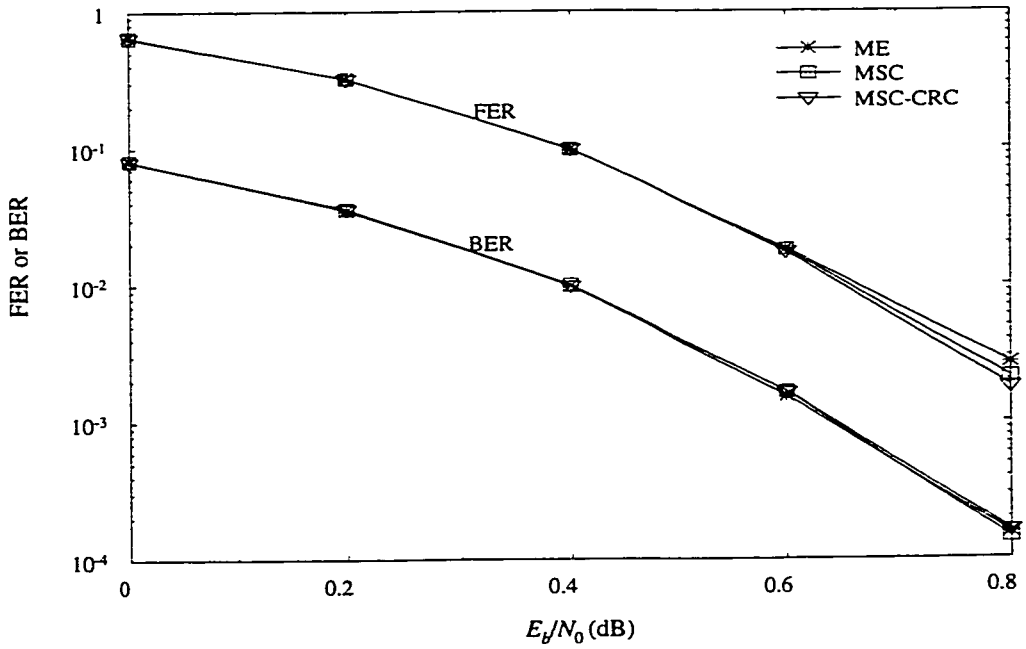


Figure 5.8: FER and BER vs. E_b/N_0 for simulation group SR1024-4.

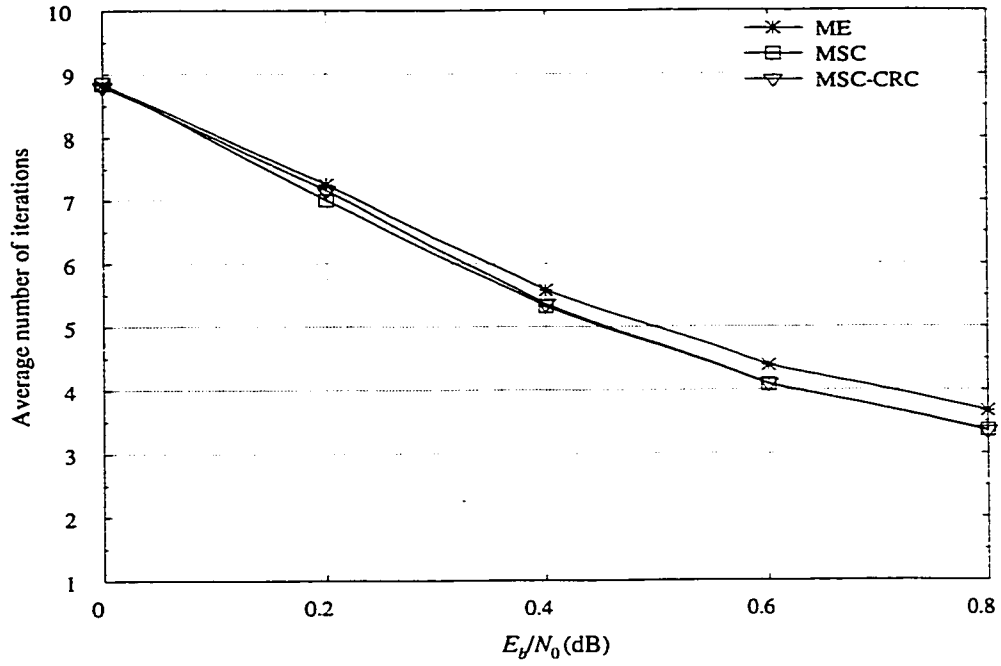


Figure 5.9: Average number of iterations vs. E_b/N_0 for simulation group SR1024-4.

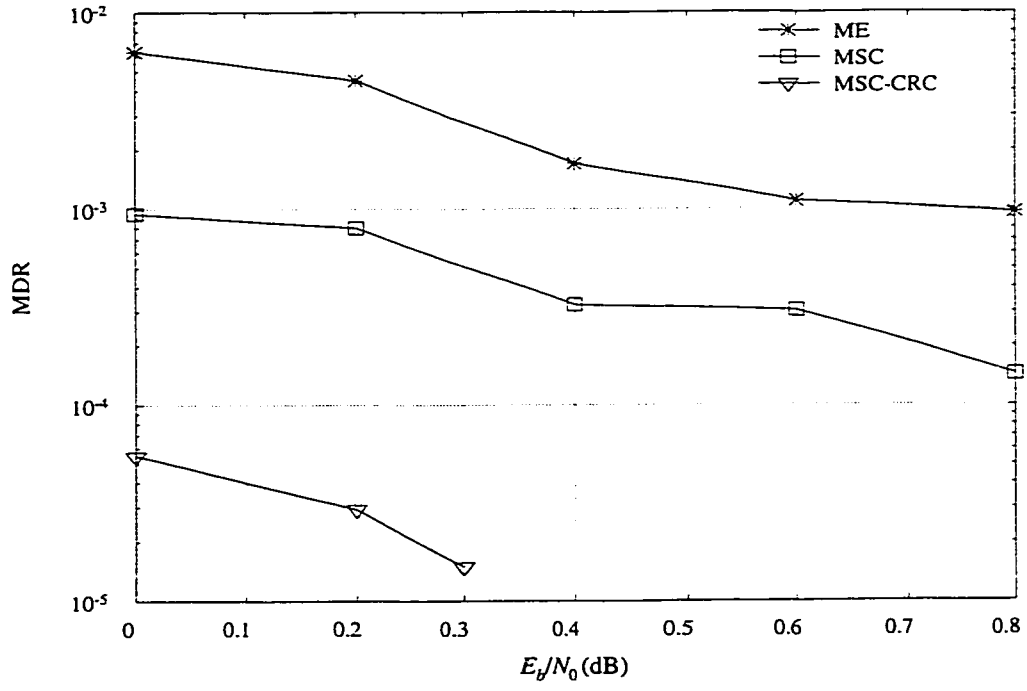


Figure 5.10: MDR vs. E_b/N_0 for simulation group SR1024-4.

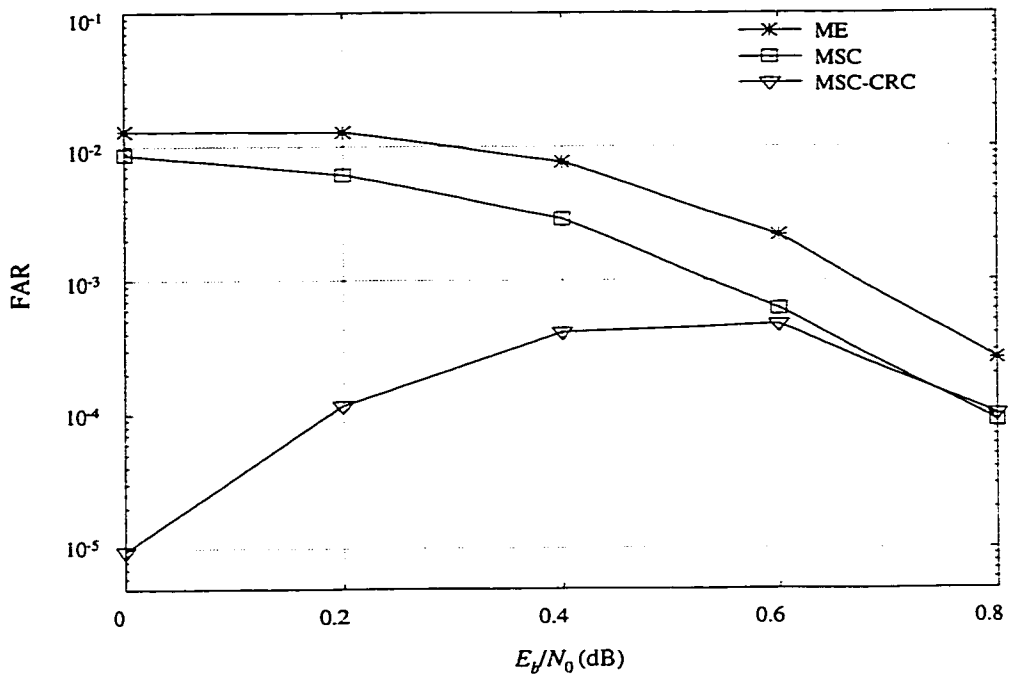


Figure 5.11: FAR vs. E_b/N_0 for simulation group SR1024-4.

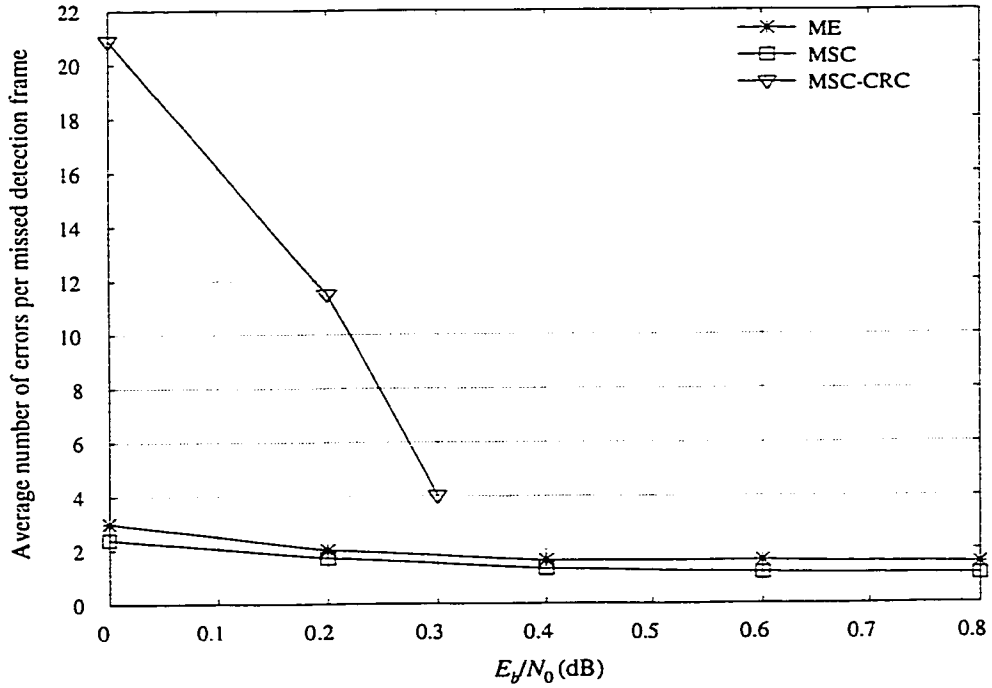


Figure 5.12: Average number of errors per missed detection frame vs. E_b/N_0 for simulation group SR1024-4.

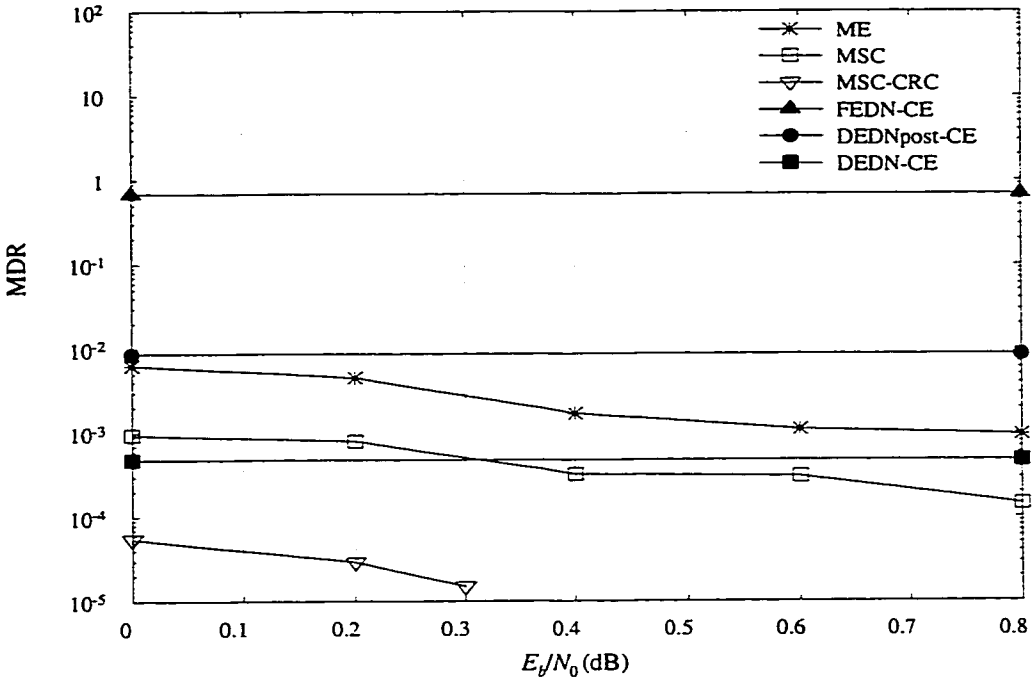


Figure 5.13: Comparison of MDR for simulation group SR1024-4 and neural network techniques.

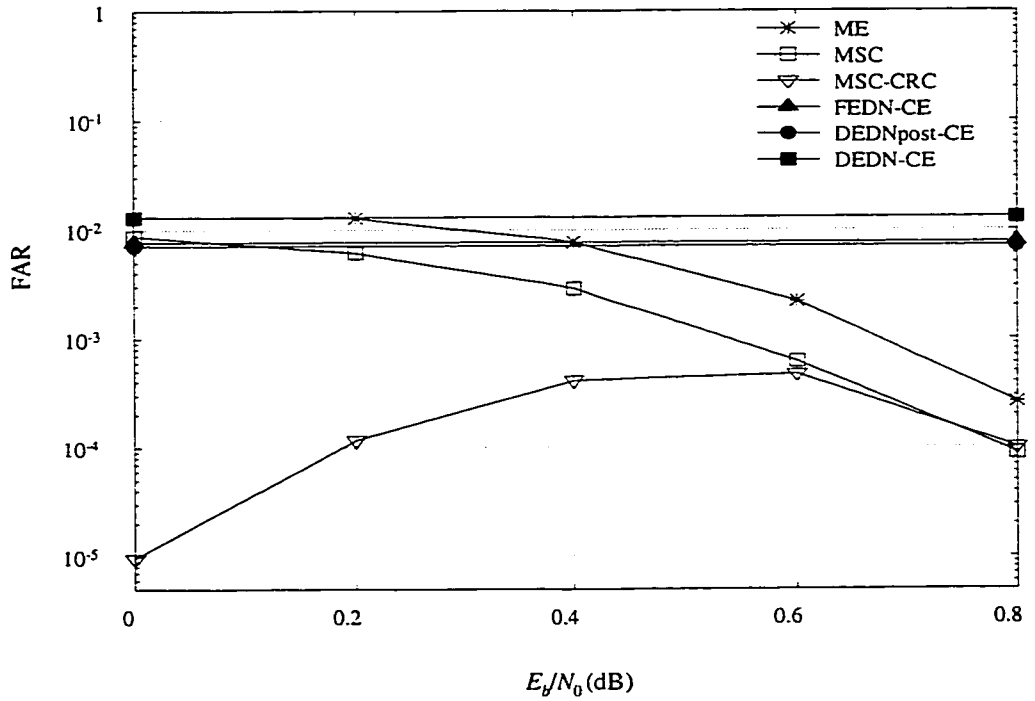


Figure 5.14: Comparison of FAR for simulation group SR1024-4 and neural network techniques.

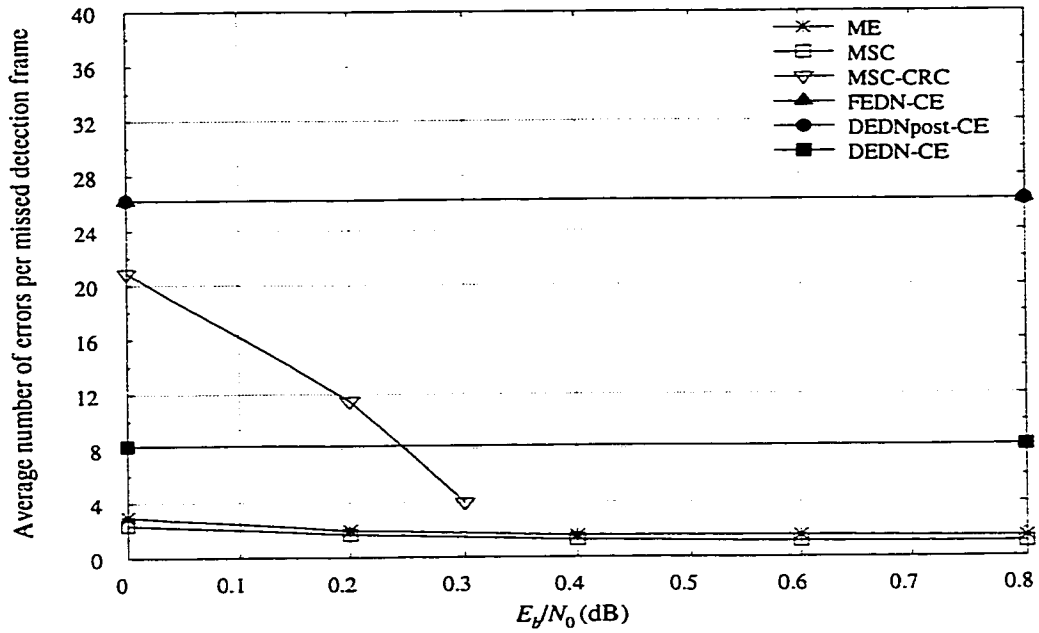


Figure 5.15: Comparison of average number of errors per missed detection frame for simulation group SR1024-4 and neural network techniques.

The following observations can be made from Figures 5.8 through 5.12. In terms of BER and FER, there is nearly no difference between the three new algorithms, except that the MSC-CRC gives slightly better FER performance at the highest value of E_b / N_0 simulated. Also it can be observed that MSC and MSC-CRC save the most iterations. In terms of MDR and FAR, the MSC-CRC technique gives the best result. Note that the MDR vs. E_b / N_0 curve for MSC-CRC was only generated for 0.0 dB, 0.2dB and 0.3 dB, since when $E_b / N_0 > 0.3\,\text{dB}$, the MDR was too low to generate a reliable value within a reasonable period of time.

Figures 5.13 to 5.15 compare the error detection performance of the new algorithms to that of neural networks with CE. When presenting their results, Buckley and Wicker presented only averaged results for E_b / N_0 values of 0 through 0.8 dB [40, 41]. Therefore, in these figures, their averaged results are plotted as straight lines. It should be emphasized, however, that the neural networks could show performance trends not visible in these averaged results. Regarding MDR, ME is better than FEDN-CE and DEDNpost-CE, however it is worse than DEDN-CE. On average, MSC is better than all three neural network schemes and MSC-CRC provides much better performance than do the neural networks. Regarding FAR, on average the ME and the MSC techniques provide better performance than the neural networks, and MSC-CRC provides significantly superior performance. Regarding the average number of errors per missed detection frame, ME and MSC offer much better results than the neural networks. Although MSC-CRC is

worse than DEDN-CE at the lower E_b / N_0, it can be seen that it tends toward better performance at higher E_b / N_0.

5.4 Performance Comparison

In this section, the performance of the new algorithms is investigated for varying coding parameters. For purpose of comparison, three comparison pairs of the simulation groups are constructed: (a) SR256-2 and SR256-3, (b) SR256-3 and SR1024-4, and (c) PIL900-2 and SR1024-4. The first pair of codes are identical, except for the different generators and memory ν. The second pair differ in their generators, memory and frame size. For the third pair, all parameters are different. In the following, a performance comparison of these pairs is presented.

A. FER and BER

The performance of the FER and BER for the three pairs is compared in Figure 5.16 (a) – (c).

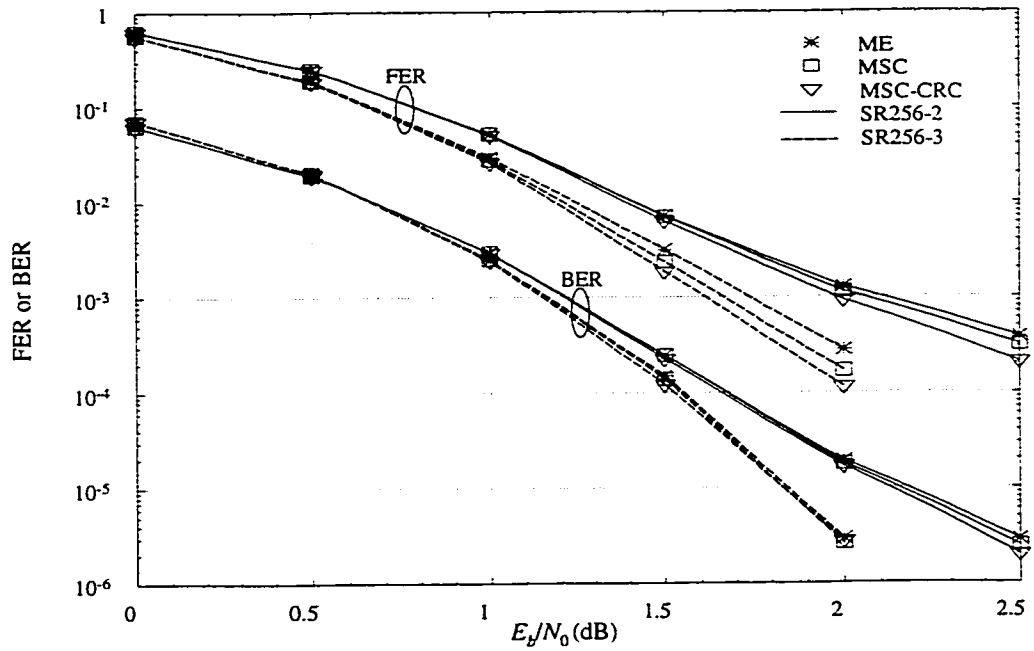


Figure 5.16 (a): Comparison of the FER and BER for simulation groups SR256-2 and SR256-3.

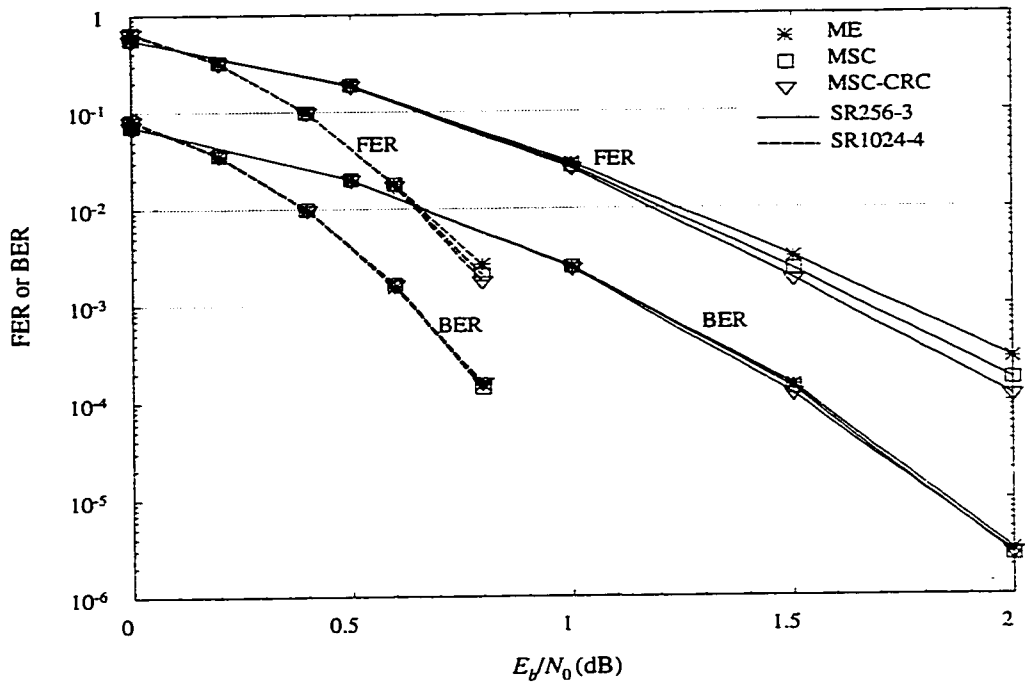


Figure 5.16 (b): Comparison of the FER and BER for simulation groups SR256-3 and SR1024-4.

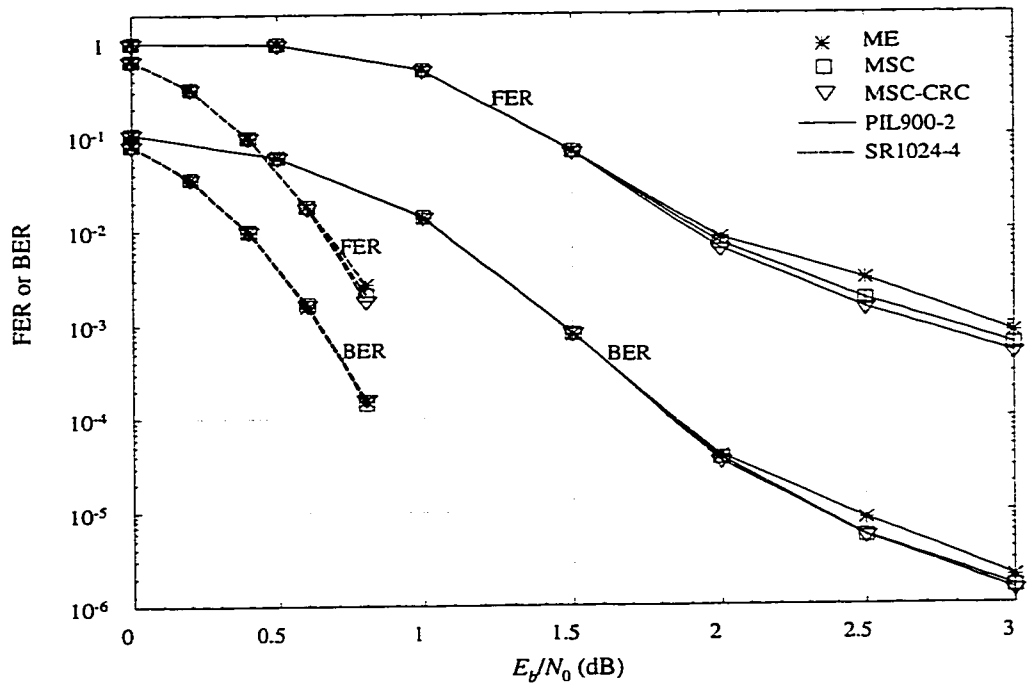


Figure 5.16 (c): Comparison of the FER and BER for simulation groups PIL900-2 and SR1024-4.

B. The average number of iterations

The performance of the average number of iterations for the three pairs is compared in Figure 5.17 (a) – (c).

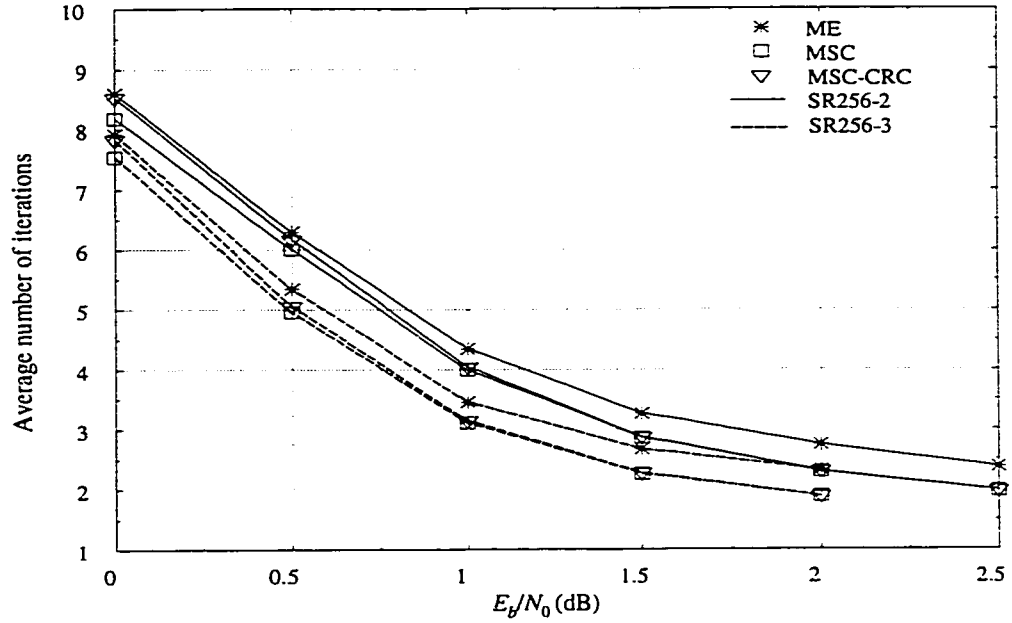


Figure 5.17 (a): Comparison of the average number of iterations for simulation groups SR256-2 and SR256-3.

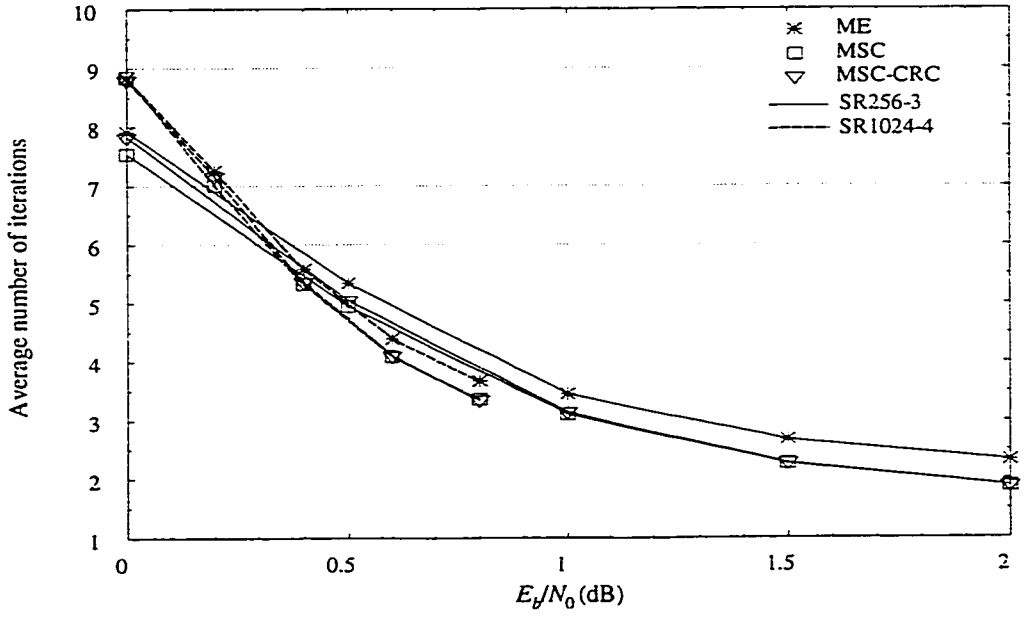


Figure 5.17 (b): Comparison of the average number of iterations for simulation groups SR256-3 and SR1024-4.

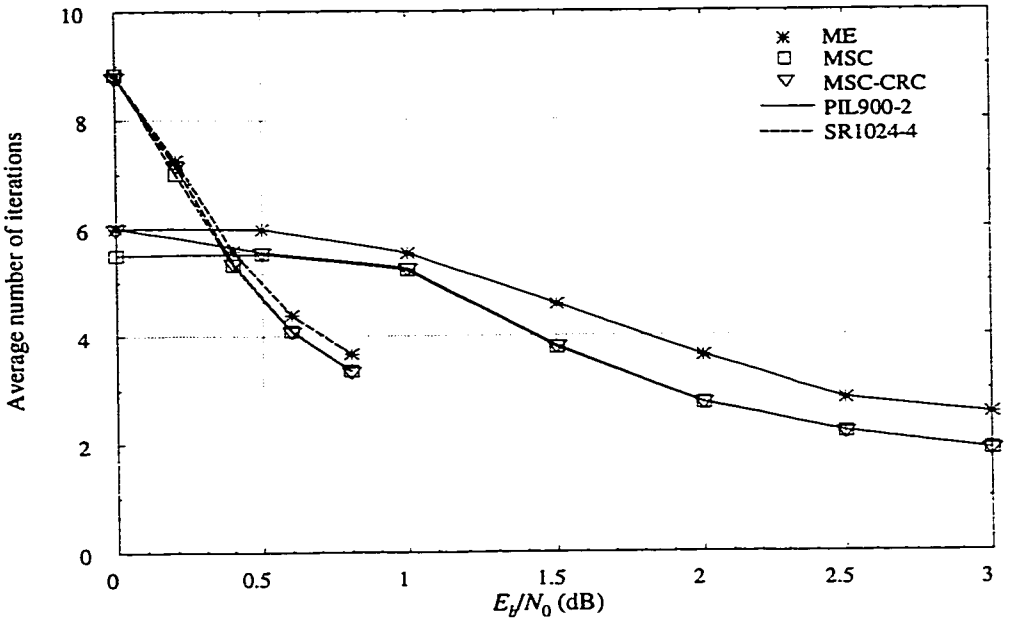


Figure 5.17 (c): Comparison of the average number of iterations for simulation groups PIL900-2 and SR1024-4.

C. MDR

The performance of the MDR for the three pairs is compared in Figure 5.18 (a) – (c).

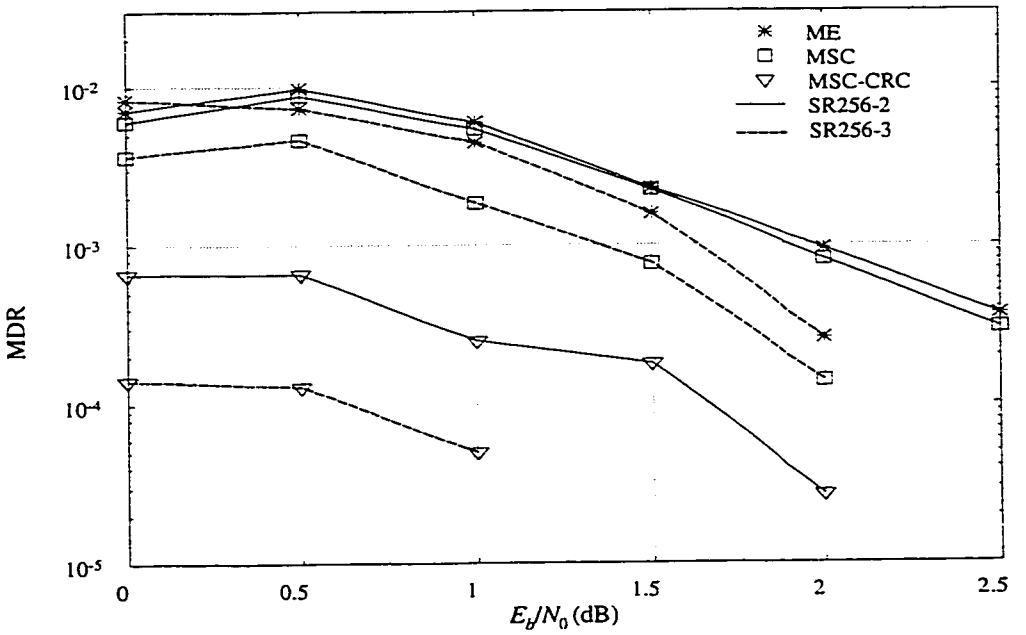


Figure 5.18 (a): Comparison of the MDR for simulation groups SR256-2 and SR256-3.

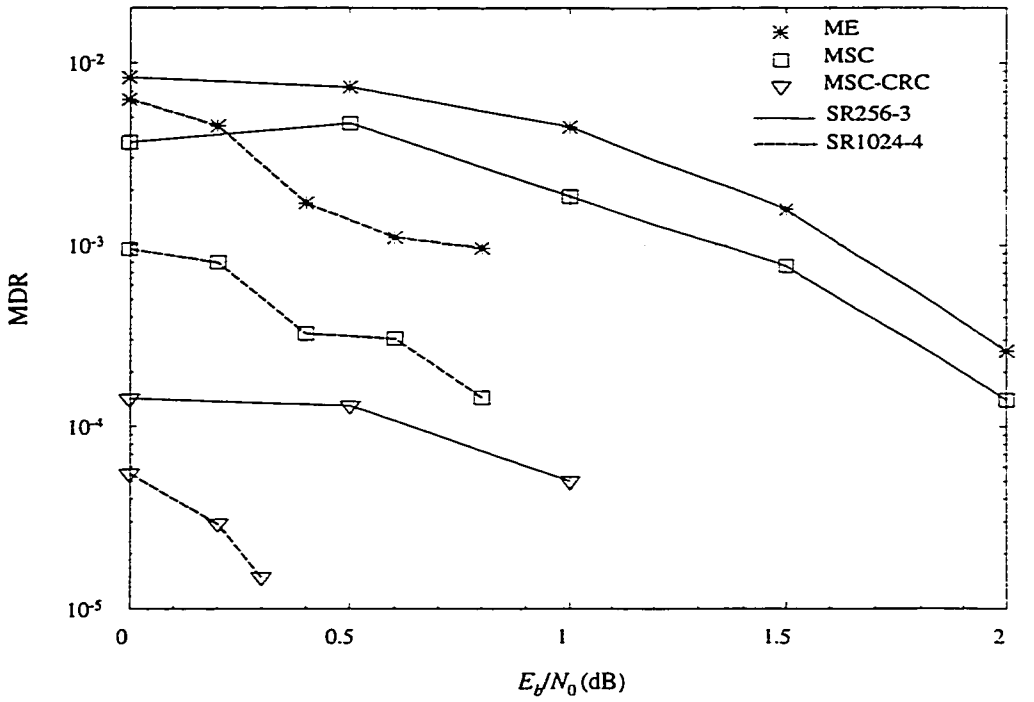


Figure 5.18 (b): Comparison of the MDR for simulation groups SR256-3 and SR1024-4.

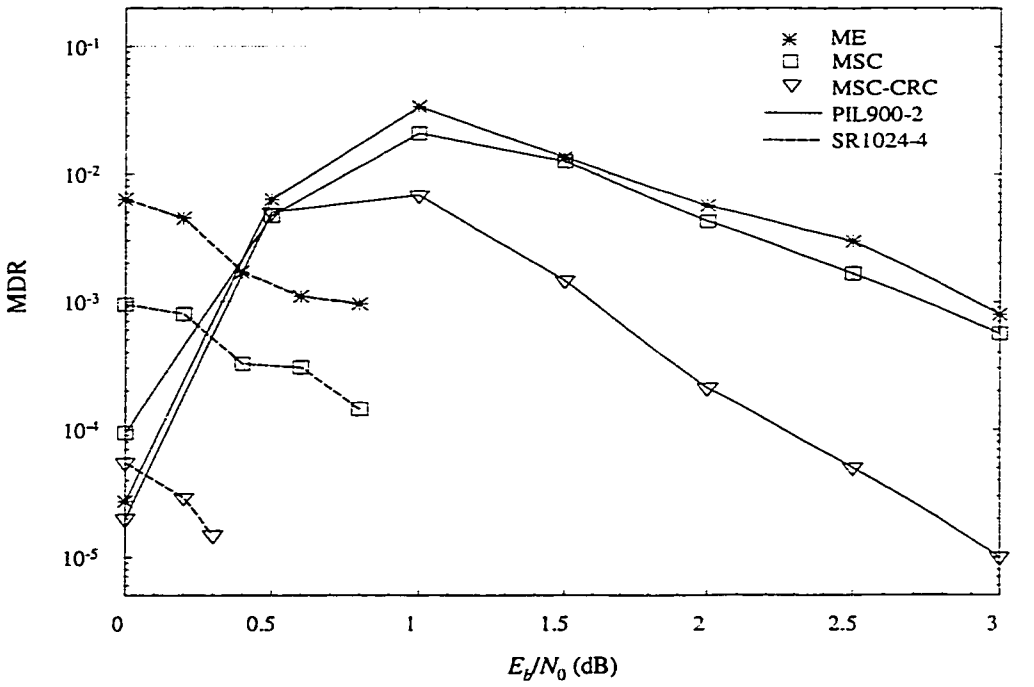


Figure 5.18 (c): Comparison of the MDR for simulation groups PIL900-2 and SR1024-4.

D. FAR

The performance of the FAR for the three pairs is compared in Figure 5.19 (a) – (c).

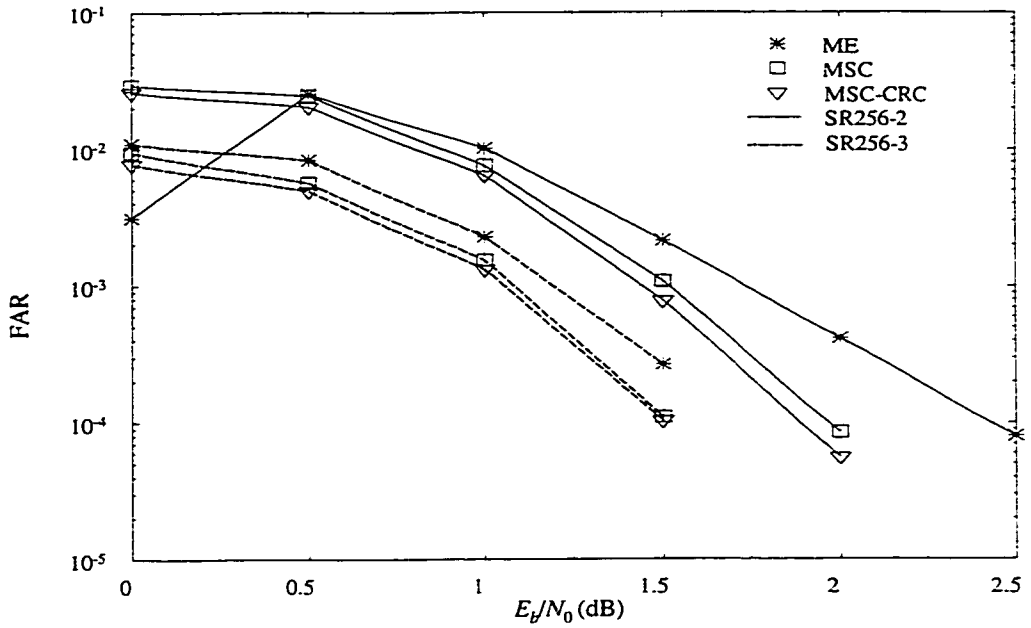


Figure 5.19 (a): Comparison of the FAR for simulation groups SR256-2 and SR256-3.

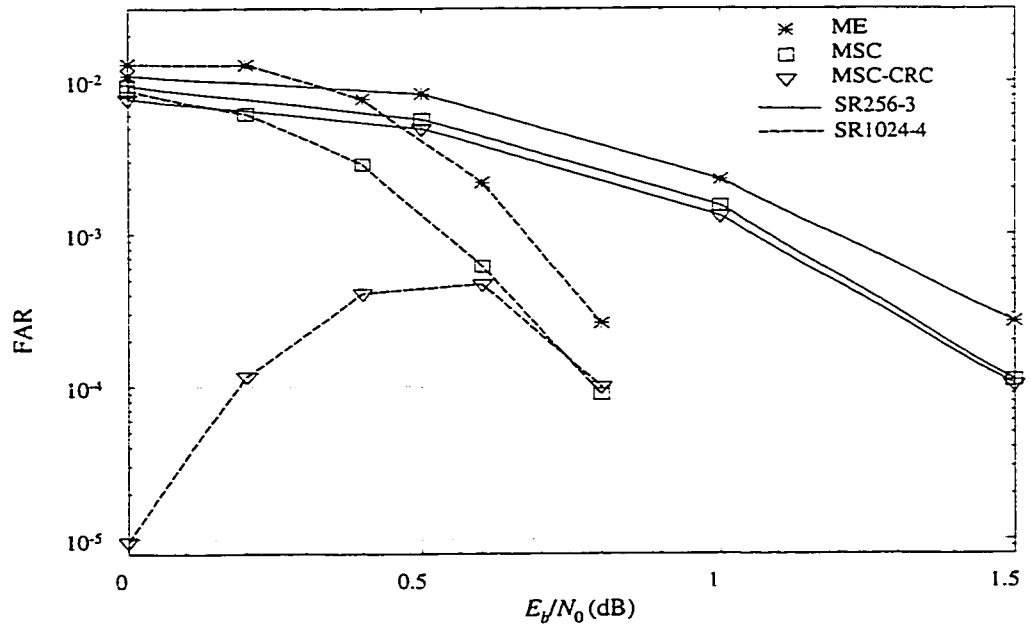


Figure 5.19 (b): Comparison of the FAR for simulation groups SR256-3 and SR1024-4.

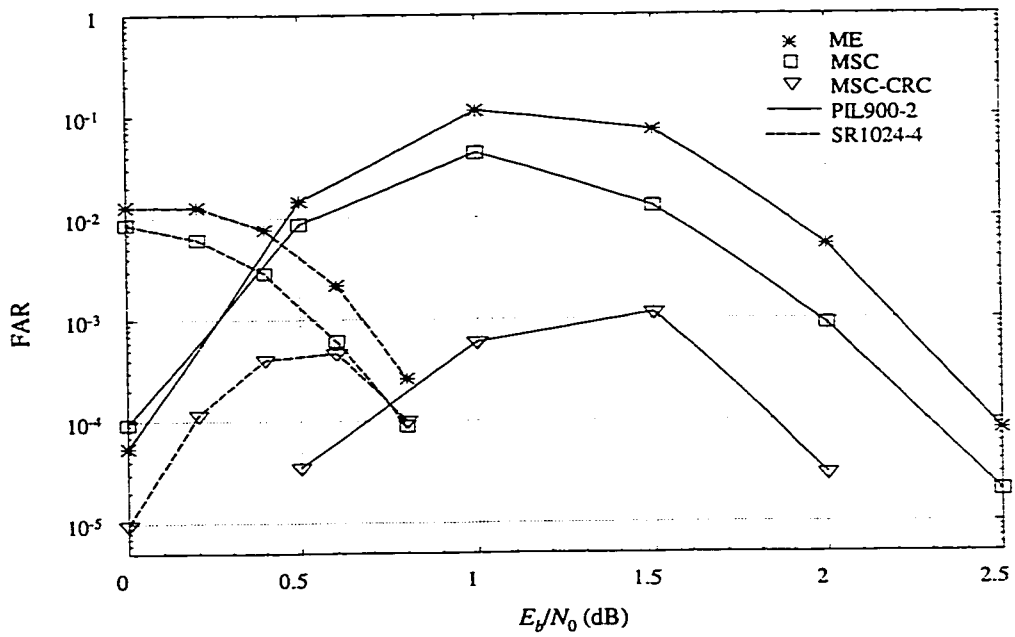


Figure 5.19 (c): Comparison of the FAR for simulation groups PIL900-2 and SR1024-4.

E. The average number of errors per missed detection frame

The performance of the average number of errors per missed detection frame for the three pairs is compared in Figure 5.20 (a) – (c).

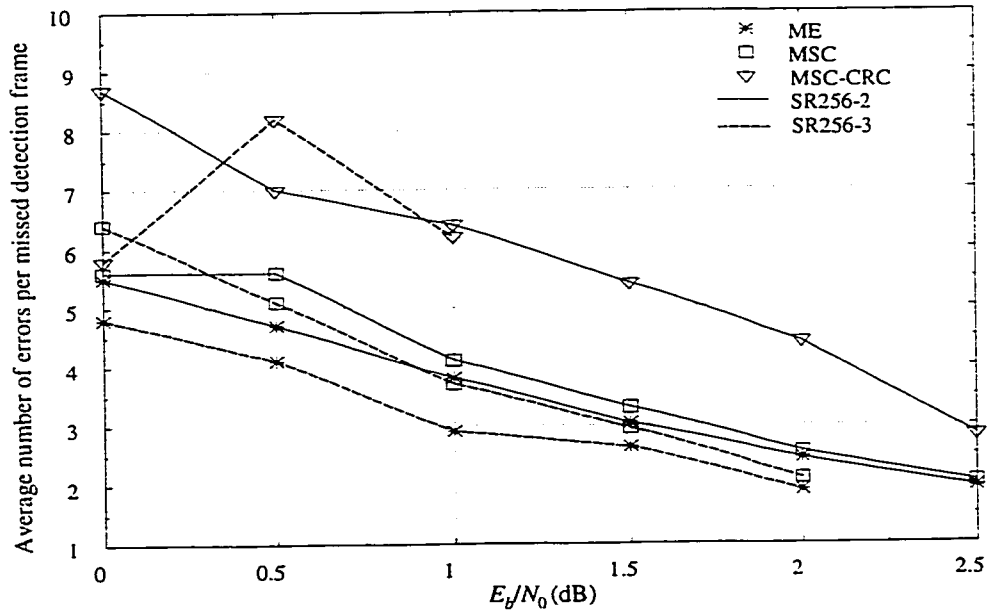


Figure 5.20 (a): Comparison of the average number of errors per missed detection frame for simulation groups SR256-2 and SR256-3.

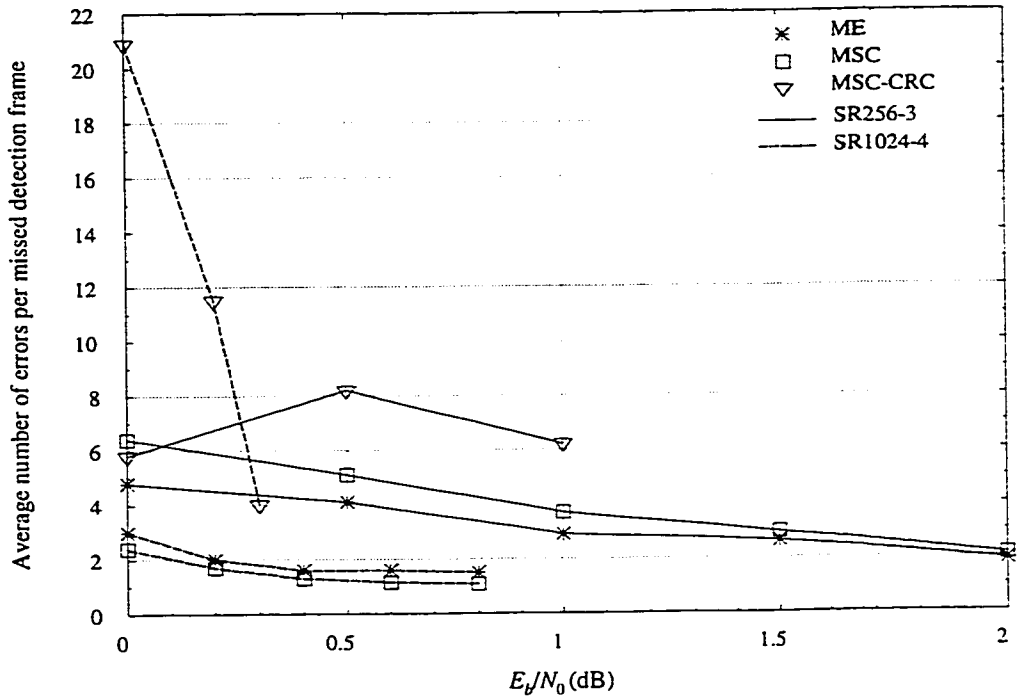


Figure 5.20 (b): Comparison of the average number of errors per missed detection frames for simulation groups SR256-3 and SR1024-4.

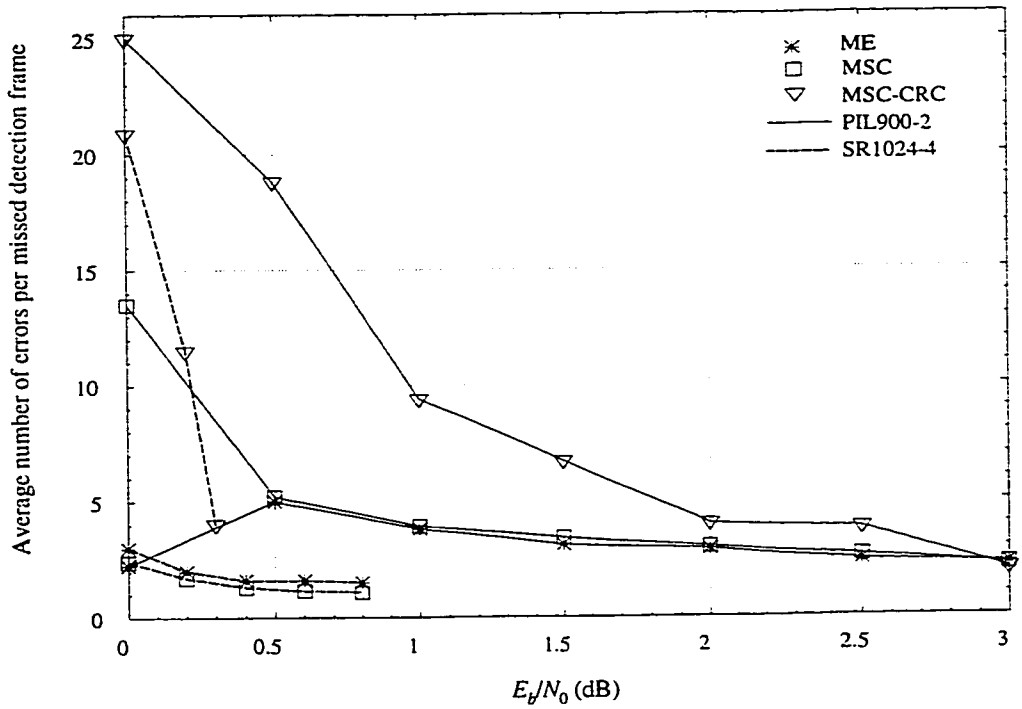


Figure 5.20 (c): Comparison of the average number of errors per missed detection frames for simulation groups PIL900-2 and SR1024-4.

From these figures, generally it is clear that for all three new algorithms at all but the lowest values of E_b / N_0, increasing the memory of the generators, frame sizes, and MAX_{it}, and decreasing the code rate offers better performance. Given the very high FER and BER results at the lowest values of E_b / N_0, however, differences in terms of other performance measures are irrelevant, therefore this conclusion holds for all E_b / N_0 of interest.

From these figures, the reduction in the required E_b / N_0, between the two codes in the simulation pair, to achieve a specified values of the FER, BER, MDR and FAR is obtained to show the influence of increasing the difference of code parameters for the three comparison pairs. The results are illustrated in Table 5.6. Note that since the performance in terms of the FER and BER for the three new algorithms is very similar, the reduction in the required E_b / N_0 in terms of FER and BER is given without considering the difference for the three new algorithms.

Table 5.6: Reduction in the required E_b / N_0

Reduction in E_b/N_0 (dB) — Comparison pairs / Specified rates and algorithms		1st (SR256-3 / SR256-2)	2nd (SR1024-4 / SR256-3)	3rd (SR1024-4 / PIL900-2)
FER (1×10^{-2})		0.21	0.57	1.30
BER (1×10^{-3})		0.06	0.53	0.84
MDR	ME (1×10^{-3})	0.37	0.83	2.11
	MSC (5×10^{-4})	0.63	1.35	2.70
	MSC-CRC (5×10^{-5})	0.84	0.98	2.47
FAR (4×10^{-4})	ME	0.60	0.65	1.56
	MSC	0.44	0.62	1.47
	MSC-CRC	0.40	0.63	1.00

From Table 5.6, it can be observed that for the values in each row, the reduction in the required E_b / N_0 between the two codes in the simulation pair increases from the first through the third comparison pairs. The difference of the code parameters also increases according to this order. The third comparison pair yields the largest reduction in the required E_b / N_0.

The reductions for the average number of errors per missed detection frame and the average number of iterations, between the two codes in the simulation pair, are presented in the Tables 5.7 and 5.8 for the three comparison pairs. The results are taken from Figures 5.20 (a) – (c) and 5.17 (a) – (c) for the specified values of E_b / N_0.

Table 5.7: Reduction of the average number of errors per missed detection frame

Reduction of errors / Comparison pairs / Algorithms	1st (SR256-3 / SR256-2)	2nd (SR1024-4 / SR256-3)	3rd (SR1024-4 / PIL900-2)
ME (0.5 dB)	0.60	2.49	3.25
MSC (0.5 dB)	0.49	3.87	3.9
MSC-CRC (0.3dB)	0.64	3.31	17.37

Table 5.8: Reduction of the average number of iterations when $E_b / N_0 = 0.8\,\text{dB}$

Reduction of iterations / Comparison pairs / Algorithms	1st (SR256-3 / SR256-2)	2nd (SR1024-4 / SR256-3)	3rd (SR1024-4 / PIL900-2)
ME	0.94	0.53	2.02
MSC	0.95	0.51	2.02
MSC-CRC	1.00	0.54	2.03

Table 5.7 demonstrates trends similar to those in Table 5.6; the reduction in the average number of errors per missed detection frame increases from the first through the third comparison pairs for the values in each row. Note that for the third comparison pair MSC-CRC yields a very large reduction at $E_b / N_0 = 0.3\,\text{dB}$.

In Table 5.8, the reduction of the average number of iterations is given for the three new algorithms when $E_b / N_0 = 0.8\,\text{dB}$. Note that when comparing the first two columns, the difference of the parameters for the second pair is larger, but the reduction for the average number of iterations is smaller. The reason is that increasing the frame size results in an increase of the average number of iterations [38]. Therefore, for simulation group SR1024-4 in the second comparison pair, the average number of iterations decreases due to the larger memory generators, but not as much as it would have decreased if the frame size had remained small. For the third comparison pair, the difference in code memory results in the reduction of the average number of iterations being larger than with other comparison pairs. Also the small difference of the frame sizes, only 124, does not significantly affect this reduction.

5.4 Performance Summary

The main points of the preceding discussion regarding the performance of the three new algorithms can be summarized as follows:

(1) In terms of FER and BER, the three new algorithms offer similar performance at low values of E_b / N_0 and MSC-CRC provides slightly better performance at larger E_b / N_0. In terms of average number of iterations, among the three algorithms the MSC technique saves the most number of iterations, while the MSC-CRC approach provides almost the same performance as MSC at larger values of E_b / N_0. In terms of MDR and FAR, MSC-CRC gives the best results except at the lowest values of E_b / N_0, and MSC provides better performance than ME. Regarding the average number of errors per missed detection frame, MSC-CRC results in the largest values. ME and MSC may miss detection of frames with very few errors.

(2) The FER and BER performance of the three new algorithms is very similar to that of the other CE and HDA early stopping techniques and the performance obtained when a sufficient number of fixed iterations are used. In terms of the average number of iterations, the performance of the ME is similar to that of CE. MSC, which gives very similar performance to MSC-CRC at medium and higher values of E_b / N_0, provides the best result among all simulated criteria.

(3) When compared to the error detection performance of the neural network techniques, ME provides better MDR results than FEDN-CE and DEDNpost-CE. The performance of MSC is also better, and the performance of MSC-CRC much better, than the performance of all three neural network approaches. In terms of FAR, on average ME and MSC provide better performance than that of the neural networks, and MSC-CRC provides significantly superior performance. For the average number of errors per missed detection frame, ME and MSC offer much better results than that of the neural networks. When compared to the DEDN-CE, the MSC-CRC is appears worse at the lowest values of E_b / N_0, however it can be seen that MSC-CRC tends toward better performance at higher E_b / N_0.

(4) Generally, at all but the lowest values of E_b / N_0, increasing the memory of the generators, frame sizes, and MAX_{it}, and decreasing the code rate offers better

performance in terms of FER, BER, MDR, FAR and the average number of errors per missed detection frame. The average number of iterations will decrease with an increase in memory and MAX_{it}, and a decrease in code rate, but it will increase with an increase in the frame size.

CHAPTER 6

CONCLUSION

This thesis has introduced new stopping criteria in conjunction with error detection techniques for turbo decoding. This chapter summarizes the development of these techniques and offers suggestions for further work.

6.1 Thesis Summary

Following the introduction of error control coding and turbo coding in Chapter 1, Chapter 2 presented further details regarding turbo coding. Since the BCJR algorithm is the basis for understanding turbo decoding, it was introduced first. The modifications to this algorithm were then described. These included the modifications that are needed to decode RSC codes, Berrou's turbo coding scheme, and the improved decoding structure proposed by Robertson. The estimates of the symbol energy and noise variance in practical turbo coding were discussed. The concepts of weight and distance of codes, which are relevant to the average performance of codes and interleaver design, were introduced. Interleavers and termination of trellises, which are important issues in the design of turbo codes, were also discussed in that chapter. Finally a simulation platform for turbo coding was described. It is apparent from this chapter that turbo codes provide excellent performance with a cost of increasing complexity.

Following an introduction to challenges with iterative decoding, several early stopping criteria were overviewed in Chapter 3. It can be seen that the early stopping criteria developed to date focus on stopping the iterative process early to save the number of iterations without degrading BER and FER. However, these criteria are not typically used to determine if all errors are corrected or if some still remain in the frame. In Chapter 4, the ME, MSC, and MSC-CRC approaches for combining early stopping and error detection were proposed and analyzed. The relationship between the mean of the absolute LLR values and the variance of the meta-channel for decoding was discussed. It was shown that this mean value over a frame is inversely proportional to the number of errors, and the variance of the meta-channel is proportional to the number of the errors. Therefore, this mean can be used to construct a criterion called ME to stop the iterative decoding process and detect errors. In order to reduce the number of iterations further, the number of sign changes of LLR values can be considered with this mean

simultaneously to obtain the MSC criterion. The ME and the MSC techniques provide simple ways to detect a frame with a large number of errors, although they may not detect a frame with very few errors. This provided the motivation to include an external CRC prior to turbo encoding to detect a frame with very few errors. In Chapter 4 it was shown that polynomials of a much lower degree than conventional polynomials can be used for CRC detection. This results in the MSC-CRC criterion.

The performance of the new approaches was reported in Chapter 5. The frame error rate, bit error rate, average number of iterations, missed detection rate, false alarm rate, and average number of errors per missed detection frame were compared with corresponding values for other approaches. These results showed that the proposed schemes provide simple and efficient methods to stop the iterative decoding process without appreciably degrading performance, and also check for errors without introducing any redundancy or, in the case of MSC-CRC, introducing less redundancy than other techniques including standard CRC polynomials. The BER and FER performance of the three newly proposed criteria are very similar. The error detection performance of the MSC-CRC technique is better than MSC, which is better than ME.

6.2 Suggestions for Further Work

This thesis is the first detailed investigation of the relationships between the number of errors in a frame with the mean of the absolute LLR values, the sign changes of the LLR values, and the remainders of built-in CRC and external short CRC codes. Work remains regarding the suitability of these techniques when they are used with other decoding algorithms, scheme of the single threshold, and criterion of saving the number of iterations still further.

The following provides suggestions for further research.

(1) Throughout the simulations of the new techniques, only the MAP algorithm for turbo decoding was investigated. Although the MAP algorithm is optimum for decoding performance and it is not difficult to implement in simulation with software, it is likely to be considered complex for implementation in a real system mainly because of the numerical representation of the real numbers and the mixed multiplications of these values. The Log-MAP algorithm [44], which works in the logarithmic domain and has equivalent performance of the MAP algorithm, is a transformation of the MAP algorithm. This algorithm decreases the complexity of the algorithm for practical implementation of turbo decoding by converting multiplications to additions. Although in the Log-MAP algorithm, the intermediate variables

used to obtain the LLR values are calculated in the logarithmic domain, the LLR values will be approximately equal in both decoding algorithms. Therefore, the new techniques developed in this thesis should be applicable to the Log-MAP algorithm. However, further simulation is required to verify these techniques in the logarithmic domain.

(2) In this thesis, simulations of the new techniques were carried out to search the best thresholds of M_{LLI} for different E_b / N_0. Although in practical turbo decoding, knowledge of E_b / N_0 will be available through estimation, it would be convenient to use a single threshold over the range of SNR values of interest. Therefore, the performance of the new techniques using a single threshold should be investigated.

(3) In order to save the most number of iterations, the error-free frame should be stopped at the instant when all the errors in the frame have just been corrected. In general for a potential error-free frame, the sign changes SCL_{12} in the MSC criterion decrease to zero earlier than the sign changes SCL_{11} and SCL_{22}, and for an erroneous frame, the value of M_{LLI} usually remains low as iterations proceed. Table 6.1 illustrates the relevant information for an error-free frame and an erroneous frame where $MAX_{it} = 10$ and $Th_b = 15$. From this table, it can be seen that for this error-free frame the error number decreases to zero at iteration 3.5. However, the iterative process will be stopped at iteration 4.0 using the MSC criterion. It means that there is still potentially 0.5 iteration that could be saved. It does not appear that the MSC criterion can be modified to save this half iteration. However, if MAX_{it} is large enough, the MSC-CRC criterion could be changed as follows

> If $M_{LLI} > Th_b$, consider SCL_{12}, R_t and R_C. If $SCL_{12} \le 2$, $R_t = 0$ and $R_C = 0$, terminate the decoding process, and consider the frame to be error-free. If $R_t = 0$ and $R_C = 0$, but SCL_{12} is not zero for MAX_{it}, also consider the frame error-free.

For the erroneous frame, in order to save iterations, the M_{LLI} values could be checked at iteration $\lceil 0.5MAX_{it} \rceil$. If $M_{LLI} < 0.2Th_b$ at the half MAX_{it}, it may be impossible for M_{LLI} of this frame to be greater than Th_b at iteration MAX_{it}. Therefore, the iterative decoding could be stopped at $\lceil 0.5MAX_{it} \rceil$ for this frame. Also this table illustrates that M_{LLI} is very important to determine if many errors remain in the frame. If M_{LLI} is very small, there are many uncorrected errors in the frame. Clearly, however, just the fact that the number of the sign

changes of the LLR values is zero cannot be taken as an indication that no errors exist in the frame.

Table 6.1: Relevant information for an error-free frame and an erroneous frame

Error-free frame					Erroneous frame				
Iteration	Error num.	M_{LLI}	SCL_{12}	SCL_{11} or SCL_{22}	Iteration	Error num.	M_{LLI}	SCL_{12}	SCL_{11} or SCL_{22}
0.5	31	2.93	N/A	N/A	0.5	51	[illegible]	N/A	N/A
1.0	15	5.40	30	N/A	1.0	46	[illegible]	23	N/A
1.5	6	8.43	13	33	1.5	41	[illegible]	13	20
2.0	6	11.80	4	17	2.0	39	[illegible]	12	21
2.5	2	[illegible]	4	4	2.5	38	[illegible]	9	11
3.0	2	19.07	2	4	3.0	41	[illegible]	9	14
3.5	0	20.41	2	2	3.5	41	[illegible]	6	9
4.0	0	20.63	0	2	4.0	41	[illegible]	2	6
4.5	0	20.88	0	0	4.5	39	[illegible]	2	4
5.0	0	21.93	0	0	5.0	41	[illegible]	2	4
5.5	0	23.37	0	0	5.5	41	2.22	2	2
6.0	0	24.84	0	0	6.0	41	2.22	0	2
6.5	0	26.22	0	0	6.5	41	2.24	0	0
7.0	0	27.58	0	0	7.0	41	2.24	0	0
7.5	0	28.94	0	0	7.5	41	2.24	0	0
8.0	0	29.91	0	0	8.0	40	2.24	1	1
8.5	0	30.66	0	0	8.5	40	2.23	0	1
9.0	0	31.02	0	0	9.0	41	2.23	1	1
9.5	0	31.20	0	0	9.5	41	2.23	0	1
10.0	0	31.33	0	0	10.0	41	2.23	0	0

Bibliography

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error correcting coding and decoding: Turbo-codes (1)," in *Proc. ICC'93*, Geneva, Switzerland, May 1993, pp. 1064-1070.

[2] B. Sklar, *Digital communications, fundamental and applications*, Prentice Hall, Englewood Cliffs, NJ, 1988.

[3] S. B. Wicker, *Error control systems for digital communication and storage*, Prentice Hall, Englewood Cliffs, NJ, 1995.

[4] S. Benedetto and E. Biglieri, *Principles of digital transmission with wireless applications*, Kluwer Academic/Plenum Publishers, New York, NY, 1999.

[5] Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. on Inform. Theory*, vol. 42, no. 2, pp. 429 – 445, Mar. 1996.

[6] G. Ungerboeck, "Channel coding with multilevel/phase signals," *IEEE Trans. on Inform. Theory*, vol. 28, no. 1, pp. 55 – 67, Jan. 1982.

[7] P. Robertson, "Illuminating the structure of decoders for parallel concatenated recursive systematic (turbo) codes," in *Proc. GLOBECOM'94*, San Francisco, CA, Nov. 1994, pp. 1298 – 1303.

[8] D. Divsalar and F. Pollara, "Turbo codes for PCS applications," in *Proc. ICC'95*, Seattle, WA, June 1995, pp. 54 – 59.

[9] S. Benedetto and G. Montorsi, "Design of parallel concatenated convolutional codes," *IEEE Trans. on Commun.*, vol. 44, no. 5, pp. 591 – 600, May 1996.

[10] J. Hagenauer and P. Hoeher, "A Veiterbi algorithm with soft-decision outputs and its applications," in *Proc. CLOBECOM'89*, Dallas, TX, Nov. 1989, pp. 1680 –1686.

[11] D. Divsalar and F. Pollara, "Multiple turbo codes," in *Proc. MILCOM'96*, McLean, VA, Oct. 1996, pp. 279 – 285.

[12] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial concatenation of interleaved codes: performance analysis, design, and iterative decoding," JPL TDA Progress Report 42-126, pp. 1 – 26, Aug. 15, 1996.

[13] P. Elias, "Coding for noise channels," *IRE Conv. Record*, Part 4, pp. 37 – 47, 1955.

[14] J. M. Wozencraft, "Sequential decoding for reliable communications," *IRE Natl. Conv. Rec.*, vol. 5, Part 2, pp. 11 – 25, 1957.

[15] R. M. Fano, "A heuristic discussion of probabilistic decoding," *IEEE Trans. on Inform. Theory*, vol. IT-9, pp. 64 – 74, Apr. 1963.

[16] F. Jelinek, "A fast sequential decoding algorithm using a stack," *IBM Journal of Research and Development*, vol. 13, pp. 675 – 685, Nov. 1969.

[17] F. Jelinek, "An upper bound on moments of sequential decoding effort," *IEEE Trans. on Inform. Theory*, vol. IT-15, pp. 140 – 149, Jan. 1969.

[18] J. L. Massey, *Threshold decoding*, MIT Press, Cambridge MA, 1963.

[19] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. on Inform. Theory*, vol. IT-13, pp. 260 – 269, Apr. 1967.

[20] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 248 – 287, Mar. 1974.

[21] S. M. Ross, *Introduction to probability models*, Fifth Edition, Academic Press, London, UK, 1993.

[22] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: turbo-codes," *IEEE Trans. Commun.*, vol. 44, no.10, pp. 1261 – 1271, Oct. 1996.

[23] P. Robertson, "Improving decoder and code structure of parallel concatenated recursive systematic (turbo) codes," in *Proc. ICUPC '94*, San Diego, CA, Sept. 1994, pp. 183 – 187.

[24] T. A. Summers and S. G. Wilson, "SNR mismatch and online estimation in turbo decoding," *IEEE Trans. Commun.* vol. 46, no. 4, pp. 421 – 423, Apr. 1998.

[25] M. C. Reed, "Iterative receiver techniques for coded multiple access communication systems," Ph.D. Dissertation, The University of South Australia, Oct. 1999.

[26] S. Benedetto and G. Montorsi, "Average performance of parallel concatenated block codes," *Electronics Letters*, vol. 31, no. 3, pp. 156 – 158, 2nd Feb. 1995.

[27] S. Benedetto and G. Montorsi, "Performance evaluation of turbo codes," *Electronics Letters*, vol. 31, no. 3, pp. 163 – 165, 2nd Feb. 1995.

[28] S. Benedetto and G. Montorsi, "Design guidelines of parallel concatenated convolutional codes," in *Proc. GLOBECOM'95*, Singapore, Nov. 1995, pp. 2273 – 2277.

[29] S. Benedetto and G. Montorsi, "Unveiling turbo codes: some results on parallel concatenated coding schemes," *IEEE Trans. on Inform. Theory*, vol. 42, no. 2, pp. 409 – 428, Mar. 1996.

[30] D. Divsalar and F. Pollara, "On the design of turbo codes," JPL TDA Progress Report 42-123, pp. 99 – 121, Nov. 15, 1995.

[31] A. Shibutani, H. Suda and F. Adachi, "Complexity reduction of turbo decoding," in *Proc. VTC'99-Fall*, Amsterdam, The Netherlands, Sept. 1999, pp. 1570 – 1574.

[32] H. H. Ma and J. K. Wolf, "On tail biting convolutional codes," *IEEE Trans. Commun,.* vol. COM-34, no. 2, pp. 104 – 111, Feb. 1986.

[33] O. Joerssen and H. Meyr, "Terminating the trellis of turbo-codes," *Electronics Letters*, vol. 30, no. 16, pp. 1285 – 1286, 4th Aug. 1994.

[34] P. Guinand and J. Lodge, "Trellis termination for turbo codes," *Proc. 17th Biennial Symp. on Communications*, Queen's University, Kingston, ON, Canada, May 1994, pp. 389 – 392.

[35] H. Anton and C. Rorres, *Elementary linear algebra, application version*, Sixth Edition, John Wiley & Sons, Inc., USA, 1991.

[36] K. Reisdorph, *Sams' teach yourself, Borland C++ Builder 3 in 14 days*, SAMS PUBLISHING, Indianapolis, IN, 1998.

[37] P. Jung, "Comparison of turbo-code decoders applied to short frame transmission systems," *IEEE J. Select. Areas Commun.*, vol. 14, no. 3, pp. 530 – 537, 1996.

[38] R. Y. Shao, S. Lin, and M. P. C. Fossorier, "Two simple stopping criteria for turbo decoding," *IEEE Trans. Commun.*, vol. 47, no. 8, pp. 1117 – 1120, Aug. 1999.

[39] "CDMA2000 high rate packet data air interface specification," 3rd generation partnership project 2, 3GPP2 specifications and documentation, Sept., 2000.

[40] M. E. Buckley and S. B. Wicker, "A neural network for predicting decoder error in turbo decoders," *IEEE Commun. Letters*, vol. 3, no.5, pp. 145 – 147, May 1999.

[41] M. E. Buckley and S. B. Wicker, "The design and performance of a neural network for predicting turbo decoding error with application to hybrid ARQ protocols," *IEEE Trans. Commun.*, vol. 48, no. 4, pp. 566 – 576, Apr. 2000.

[42] R. E. Blahut, *Theory and practice of error control codes*, Addison-Wesley Publishing Company, Reading, Massachusetts, May 1984.

[43] A. S. Tanenbaum, *Computer networks*, third edition, Prentice Hall, Upper Saddle River, New Jersey, 1996.

[44] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," in *Proc. ICC'95*, Seattle, WA, June 1995, pp. 1009 – 1013.

APPENDIX 1

EVALUATION OF PROBABILITIES $\gamma_k(m',m)$, $\alpha'_k(m)$ AND $\beta'_k(m)$ FOR THE BCJR ALGORITHM

This appendix presents detailed derivation of probabilities $\gamma_k(m',m)$, $\alpha'_k(m)$ and $\beta'_k(m)$ for the BCJR algorithm.

A1.1 $\gamma_k(m',m)$

$$\gamma_k(m',m) = \Pr\{Y_k; S_k = m \mid S_{k-1} = m'\}$$

Introducing $X_k \Rightarrow$

$$= \sum_X \Pr\{(Y_k; X_k = X); S_k = m \mid S_{k-1} = m'\}$$

Pr(AB|C)=Pr(B|C)Pr(A|BC) $\Rightarrow$

$$= \sum_X \Pr\{S_k = m \mid S_{k-1} = m'\} \cdot \Pr\{Y_k; X_k = X \mid S_k = m, S_{k-1} = m'\}$$

Using Pr(AB|C)=Pr(B|C)Pr(A|BC) to the second term $\Rightarrow$

$$= \sum_X \Pr\{S_k = m \mid S_{k-1} = m'\} \cdot \Pr\{X_k = X \mid S_k = m; S_{k-1} = m'\} \cdot \Pr\{Y_k \mid X_k = X; S_k = m, S_{k-1} = m'\}$$

Considering DMC property to simplify the condition in the third term $\Rightarrow$

$$= \sum_X \Pr\{S_k = m \mid S_{k-1} = m'\} \cdot \Pr\{X_k = X \mid S_k = m; S_{k-1} = m'\} \cdot \Pr\{Y_k \mid X_k = X\}$$

$$= \sum_X p_k(m \mid m') \cdot q_k(X \mid m', m) \cdot R(Y_k \mid X). \qquad \text{(A1.1)}$$

where the summation in (A1.1.1) is over all possible output symbols X.

A1.2 $\alpha'_k(m)$

For time $k = 1, 2, ..., N$:

$$\alpha'_k(m) = \Pr\{S_k = m; Y_1^k\}$$

Introducing $S_{k-1} \Rightarrow$

$$= \sum_{m'=0}^{M-1} \Pr\{S_{k-1} = m'; S_k = m; Y_1^k\}$$

$$= \sum_{m'=0}^{M-1} \Pr\{S_{k-1} = m'; S_k = m; Y_1^{k-1}; Y_k\}$$

$$= \sum_{m'=0}^{M-1} \Pr\{(S_{k-1} = m'; Y_1^{k-1}); (Y_k; S_k = m)\}$$

Pr(AB)=Pr(A)Pr(B|A) ⇒

$$= \sum_{m'=0}^{M-1} \Pr\{(S_{k-1} = m'; Y_1^{k-1})\} \cdot \Pr\{(Y_k; S_k = m) \mid (S_{k-1} = m'; Y_1^{k-1})\}$$

Using Markov property ⇒

$$= \sum_{m'=0}^{M-1} \Pr\{S_{k-1} = m'; Y_1^{k-1}\} \cdot \Pr\{Y_k; S_k = m \mid S_{k-1} = m'\}$$

$$= \sum_{m'=0}^{M-1} \alpha'_{k-1}(m') \cdot \gamma_k(m', m) . \tag{A1.2}$$

For $k = 0$, the boundary conditions are

$$\alpha'_0(0) = 1 \text{ and } \alpha'_0(m) = 0, \text{ for all } m \neq 0. \tag{A1.3}$$

These boundary conditions are suitable since the process is assumed to start from the zero state at time $k = 0$. For the calculation of probability $\alpha'_k(m)$, begin with the boundary conditions and knowledge of the values of $\gamma_k(m', m)$, and then $\alpha'_k(m)$ can be obtained recursively for any time k. The calculation of $\alpha'_k(m)$ can be looked at as a forward recursion.

A1.3 $\beta'_k(m)$

For time $k = 1, 2, ..., N-1$:

$$\beta'_k(m) = \Pr\{Y_{k+1}^N \mid S_k = m\}$$

Introducing S_{k+1} ⇒

$$= \sum_{m'=0}^{M-1} \Pr\{S_{k+1} = m'; Y_{k+1}^N \mid S_k = m\}$$

$$= \sum_{m'=0}^{M-1} \Pr\{S_{k+1} = m'; Y_{k+1}; Y_{k+2}^N \mid S_k = m\}$$

$$= \sum_{m'=0}^{M-1} \Pr\{Y_{k+2}^N; (Y_{k+1}; S_{k+1} = m') \mid S_k = m\}$$

Pr(AB|C)=Pr(A|BC)Pr(B|C) ⇒

$$= \sum_{m'=0}^{M-1} \Pr\{Y_{k+2}^N \mid Y_{k+1}; S_{k+1} = m'; S_k = m\} \cdot \Pr\{Y_{k+1}; S_{k+1} = m' \mid S_k = m\}$$

Using Markov property ⇒

$$= \sum_{m'=0}^{M-1} \Pr\{Y_{k+2}^N \mid S_{k+1} = m'\} \cdot \Pr\{Y_{k+1}; S_{k+1} = m' \mid S_k = m\}$$

$$= \sum_{m'=0}^{M-1} \beta'_{k+1}(m') \cdot \gamma_{k+1}(m, m') . \tag{A1.4}$$

Because the encoder is assumed to end in the zero state at time $k = N$, the appropriate boundary conditions for $\beta'_N(m)$ are:

$$\beta'_N(0) = 1, \text{ and } \beta'_N(m) = 0, \text{ for all } m \neq 0. \tag{A1.5}$$

For the calculation of probability $\beta'_k(m)$, begin with the boundary conditions and knowledge of the values of $\gamma_{k+1}(m,m')$, and then $\beta'_k(m)$ can be obtained recursively for any allowed time k. The calculation of $\beta'_k(m)$ can be looked at as a backward recursion.

APPENDIX 2

EVALUATION OF PROBABILITIES $\gamma_k^i(m',m)$, $\alpha_k^i(m)$ AND $\beta_k(m)$ FOR THE M-BCJR ALGORITHM

A2.1 $\gamma_k^i(m,m')$

$$\begin{aligned}
\gamma_k^i(m',m) &= \Pr\{u_k = i; Y_k; S_k = m \mid S_{k-1} = m'\} \\
&= \Pr\{(u_k = i; Y_k); S_k = m \mid S_{k-1} = m'\} \\
\text{Pr(AB|C)=Pr(A|BC)Pr(B|C)}\Rightarrow \quad &= \Pr\{u_k = i; Y_k \mid S_k = m; S_{k-1} = m'\} \cdot \Pr\{S_k = m \mid S_{k-1} = m'\} \\
&= \Pr\{Y_k; u_k = i \mid S_k = m; S_{k-1} = m'\} \cdot \Pr\{S_k = m \mid S_{k-1} = m'\} \\
\text{Pr(AB|C)=Pr(A|BC)Pr(B|C)}\Rightarrow \quad &= \Pr\{Y_k \mid u_k = i; S_k = m; S_{k-1} = m'\} \cdot \Pr\{u_k = i \mid S_k = m; S_{k-1} = m'\} \\
&\quad \cdot \Pr\{S_k = m \mid S_{k-1} = m'\} \\
&= \Pr\{Y_k \mid u_k = i; S_k = m; S_{k-1} = m'\} \cdot q_k(u_k \mid m',m) \cdot p_k(m \mid m') \qquad \text{(A2.1)}
\end{aligned}$$

where the trellis transition probabilities are $q_k(u_k \mid m',m) = \Pr\{u_k = i \mid S_k = m; S_{k-1} = m'\}$ and the state transition probabilities are $p_k(m \mid m') = \Pr\{S_k = m \mid S_{k-1} = m'\}$. Therefore the branch transition probability $\gamma_k^i(m',m)$ can be determined from the transition probabilities of the channel and the transition probabilities of the encoder trellis.

The value of $q_k(u_k \mid m',m)$ is either one or zero depending on whether input bit i is associated with the transition from state m' to m according to the trellis diagram. The probabilities $p_k(m \mid m')$ are also generated in part by the transitions in the trellis diagram. If a state transition does not exist between state m' and m, $p_k(m \mid m') = 0$. If the state transition between them does exist, the value of $p_k(m \mid m')$ is governed by the input sequence statistics. Generally, if all input sequences are equally likely for $0 \le k < N_1$, $p_k(m \mid m') = 2^{-k_0}$, since there are 2^{-k_0} possible transitions out of each state. For $1/n_0$ codes, $p_k(m \mid m') = 0.5$. During the trellis termination period for $N_1 \le k < N$, $p_k(m \mid m') = 1.0$ when it is nonzero, because there is only one path out of each state.

If $q_k(u_k \mid m', m) = 0$, it is not necessary to consider the calculation of $\Pr\{Y_k \mid u_k = i; S_k = m; S_{k-1} = m'\}$. If $q_k(u_k \mid m', m) = 1$, this term must be evaluated. This evaluation requires knowledge of the branch values X_k that weight the corresponding branch in the trellis.

Two cases exist for the calculation of probability $\Pr\{Y_k \mid u_k = i; S_k = m; S_{k-1} = m'\}$, depending on if the received signal is quantized or not. If the received signal is quantized, the discrete symbol transition probabilities are used, and the method is similar to the BCJR algorithm. However, in the M-BCJR case, the probability $\Pr\{Y_k \mid u_k = i; S_k = m; S_{k-1} = m'\}$ is related to the input bit u_k, so there are two γ_k: one is γ_k^0 for $u_k = 0$, and the other is γ_k^1 for $u_k = 1$. In matrix notation, the matrix $[\gamma_k]$ will be separated into two matrices, $[\gamma_k^0]$ and $[\gamma_k^1]$.

A demodulator that does not quantize the received signal is said to perform perfect soft decisions. The channel investigated here is the AWGN channel, where after modulation each transmited symbol is added to an independent noise sample with zero mean and variance σ^2. Figure A2.1 shows conditional Gaussian distributions with mean +1 and mean −1. This diagram assumes binary phase shift keying (BPSK) where the modulator maps an encoded logic one to +1 and a logic zero to -1 respectively. That is, the transmitted signal has values $2x_k - 1$, where x_k is the logic value of the encoded bit.

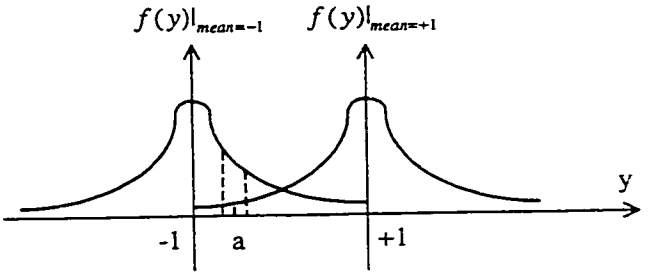


Figure A2.1: Conditional probability density function with Gaussian distribution.

Calculation of the probability $\Pr\{Y_k \mid u_k = i; S_k = m; S_{k-1} = m'\}$ is simplified if the value of the probability density function is used instead of the value of the probability. This is feasible for the following reasons.

Let Y be a continuous random variable with probability density function $f(y)$, let a be the possible value in the value set, and let ε be a very small interval. Then [21, pp. 32]:

$$\Pr\{a-\frac{\varepsilon}{2}\le Y\le a+\frac{\varepsilon}{2}\}=\int_{a-\varepsilon/2}^{a+\varepsilon/2} f(y)dy\approx \varepsilon f(a). \tag{A2.2}$$

In other words, the probability that Y will be contained in an interval of length ε around the point a is approximately $\varepsilon f(a)$.

At time k, assuming BPSK modulation, the receiver obtains the sequence:

$$\begin{aligned} Y_k &= (y_{s,k}, y_{1p,k},\ldots, y_{(n_0-1)p,k}) \\ &= (2x_{s,k}-1+n_{s,k}, 2x_{1p,k}-1+n_{1p,k},\ldots, 2x_{(n_0-1)p,k}-1+n_{(n_0-1)p,k}). \end{aligned}$$

Conditionally to $(u_k=i; S_k=m; S_{k-1}=m')$, $(y_{s,k}, y_{1p,k},\ldots, y_{(n_0-1)p,k})$ are uncorrelated Gaussian random variables and therefore:

$$\begin{aligned} \Pr\{Y_k \mid u_k=i; S_k=m; S_{k-1}=m'\} &= \Pr\{Y_k \mid \ldots\} \\ &= \Pr\{y_{s,k}\mid\ldots\}\cdot\Pr\{y_{1p,k}\mid\ldots\}\cdots\Pr\{y_{(n_0-1)p,k}\mid\ldots\} \\ &= \{f(y_{s,k}\mid\ldots)\cdot\varepsilon\}\cdot\{f(y_{1p,k}\mid\ldots)\cdot\varepsilon\}\cdots\{f(y_{(n_0-1)p,k}\mid\ldots)\cdot\varepsilon\} \\ &= f(y_{s,k}\mid\ldots)\cdot f(y_{1p,k}\mid\ldots)\cdots\{f(y_{(n_0-1)p,k}\mid\ldots)\cdot\varepsilon^{n_0} \end{aligned} \tag{A2.3}$$

$$\begin{aligned} \gamma_k^i(m',m) &= \Pr\{Y_k \mid u_k=i; S_k=m; S_{k-1}=m'\}\cdot q_k(u_k\mid m',m)\cdot p_k(m\mid m') \\ &= f(y_{s,k}\mid\ldots)\cdot f(y_{1p,k}\mid\ldots)\cdots\{f(y_{(n_0-1)p,k}\mid\ldots)\cdot\varepsilon^{n_0}\cdot q_k(u_k\mid m',m)\cdot p_k(m\mid m') \end{aligned} \tag{A2.4}$$

Note that in the formulas for $\alpha_k^i(m)$ and $\beta_k(m)$ (see A2.8 and A2.14), shown below for reference, $\gamma_k^i(m',m)$ is in both the numerator and denominator, so the ε^{n_0} terms cancel. Therefore for the actual calculation of $\Pr\{Y_k \mid u_k=i; S_k=m; S_{k-1}=m'\}$ in $\gamma_k^i(m',m)$, the probability density function can be used instead of the probability.

$$\alpha_k^i(m)=\frac{\sum_{m'=0}^{M-1}\alpha_{k-1}(m')\cdot\gamma_k^i(m',m)}{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1}\sum_{i=0}^{1}\alpha_{k-1}(m')\cdot\gamma_k^i(m',m)},$$

$$\beta_k(m)=\frac{\sum_{m'=0}^{M-1}\sum_{i=0}^{1}\cdot\beta_{k+1}(m')\cdot\gamma_k^i(m,m')}{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1}\sum_{i=0}^{1}\alpha_k(m)\cdot\gamma_k^i(m,m')}.$$

If the channel model is AWGN with zero mean and variance σ^2 (note $\sigma^2=N_0/2$ when the mean is zero), the calculation of $\Pr\{Y_k \mid u_k=i; S_k=m; S_{k-1}=m'\}$ for $\gamma_k^i(m',m)$ can be simplified

even further [7]. For example, let $n_0 = 2$ so that $Y_k = (y_{s,k}, y_{p,k})$ at time k. Let $b_s(i,m',m)$ and $b_p(i,m',m)$ denote the bipolar branch values (branch value logic zero $\rightarrow$ bipolar branch value -1, branch value logic one $\rightarrow$ bipolar branch value +1), which are the modulator outputs associated with the branch from state m' to m at time k. Then:

$$\begin{aligned}
\Pr\{Y_k \mid u_k = i; S_k = m; S_{k-1} = m'\} &= \Pr\{y_{s,k} \mid u_k = i; S_k = m; S_{k-1} = m'\} \\
&\quad \cdot \Pr\{y_{p,k} \mid u_k = i; S_k = m; S_{k-1} = m'\} \\
&= f(y_{s,k} \mid u_k = i; S_k = m; S_{k-1} = m') \\
&\quad \cdot f(y_{p,k} \mid u_k = i; S_k = m; S_{k-1} = m') \cdot \varepsilon^2 \\
&= \frac{1}{\sqrt{2\pi\sigma}} e^{-(y_{s,k} - b_s(i,m',m))^2/2\sigma^2} \cdot \frac{1}{\sqrt{2\pi\sigma}} e^{-(y_{p,k} - b_p(i,m',m))^2/2\sigma^2} \cdot \varepsilon^2 \\
&= k_e e^{-(y_{s,k} - b_s(i,m',m))^2/2\sigma^2} \cdot e^{-(y_{p,k} - b_p(i,m',m))^2/2\sigma^2} \\
&= k_e e^{-(y_{s,k} - b_s(i,m',m))^2/N_0} \cdot e^{-(y_{p,k} - b_p(i,m',m))^2/N_0} \qquad \text{(A2.5)}
\end{aligned}$$

where k_e denotes $\frac{1}{\sqrt{2\pi\sigma}} \cdot \frac{1}{\sqrt{2\pi\sigma}} \cdot \varepsilon^2$, which is common to all probabilities $\Pr\{Y_k \mid u_k = i; S_k = m; S_{k-1} = m'\}$ and can be eliminated for the calculation of $\alpha_k^i(m)$ and $\beta_k(m)$. Equivalently, k_e can be set to 1 in relation (A2.5).

A2.2 $\alpha_k^i(m)$

$$\begin{aligned}
\alpha_k^i(m) &= \Pr\{u_k = i; S_k = m \mid Y_1^k\} \\
&= \frac{\Pr\{u_k = i; S_k = m; Y_1^k\}}{\Pr\{Y_1^k\}} \\
&= \frac{\Pr\{(u_k = i; S_k = m; Y_k); Y_1^{k-1}\}}{\Pr\{Y_1^{k-1}; Y_k\}} \\
&= \frac{\Pr\{(u_k = i; S_k = m; Y_k) \mid Y_1^{k-1}\} \cdot \Pr\{Y_1^{k-1}\}}{\Pr\{Y_k \mid Y_1^{k-1}\} \cdot \Pr\{Y_1^{k-1}\}} \\
&= \frac{\Pr\{u_k = i; S_k = m; Y_k \mid Y_1^{k-1}\}}{\Pr\{Y_k \mid Y_1^{k-1}\}}
\end{aligned}$$

Introduce u_k and S_k in

the denominator $\Rightarrow$

$$= \frac{\Pr\{u_k = i; S_k = m; Y_k \mid Y_1^{k-1}\}}{\sum_{m=0}^{M-1}\sum_{i=0}^{1} \Pr\{u_k = i; S_k = m; Y_k \mid Y_1^{k-1}\}} \tag{A2.6}$$

Note that there exists the same term in the numerator and denominator in relation (A2.6). In order to obtain a recursive relation for calculation of $\alpha_k^i(m)$, this term is computed by introducing variables u_{k-1} and S_{k-1}.

$$\Pr\{u_k = i; S_k = m; Y_k \mid Y_1^{k-1}\} = \sum_{m'=0}^{M-1}\sum_{j=0}^{1} \Pr\{u_{k-1} = j; S_{k-1} = m'; u_k = i; S_k = m; Y_k \mid Y_1^{k-1}\}$$

$$= \sum_{m'=0}^{M-1}\sum_{j=0}^{1} \frac{\Pr\{u_{k-1} = j; S_{k-1} = m'; u_k = i; S_k = m; Y_k; Y_1^{k-1}\}}{\Pr\{Y_1^{k-1}\}}$$

$$= \sum_{m'=0}^{M-1}\sum_{j=0}^{1} \frac{\Pr\{(u_{k-1} = j; S_{k-1} = m'; Y_1^{k-1}); (u_k = i; S_k = m; Y_k)\}}{\Pr\{Y_1^{k-1}\}}$$

$$= \sum_{m'=0}^{M-1}\sum_{j=0}^{1} \frac{\Pr\{u_{k-1} = j; S_{k-1} = m'; Y_1^{k-1}\}}{\Pr\{Y_1^{k-1}\}} \cdot \Pr\{u_k = i; S_k = m; Y_k \mid u_{k-1} = j; S_{k-1} = m'; Y_1^{k-1}\}$$

Markov property $\Rightarrow$

$$= \sum_{m'=0}^{M-1}\sum_{j=0}^{1} \frac{\Pr\{u_{k-1} = j; S_{k-1} = m'; Y_1^{k-1}\}}{\Pr\{Y_1^{k-1}\}} \cdot \Pr\{u_k = i; S_k = m; Y_k \mid S_{k-1} = m'\}$$

$$= \sum_{m'=0}^{M-1}\sum_{j=0}^{1} \alpha_{k-1}^j(m') \cdot \gamma_k^i(m', m)$$

$$= \sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^i(m', m) \tag{A2.7}$$

By introducing relation (A2.7) to (A2.6), a recursive formula for $\alpha_k^i(m)$ can be obtained:

$$\alpha_k^i(m) = \frac{\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k^i(m', m)}{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1}\sum_{i=0}^{1} \alpha_{k-1}(m') \cdot \gamma_k^i(m', m)} \tag{A2.8}$$

$$\alpha_k(m) = \frac{\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k(m', m)}{\sum_{m=0}^{M-1}\sum_{m'=0}^{M-1} \alpha_{k-1}(m') \cdot \gamma_k(m', m)} \tag{A2.9}$$

A2.3 $\beta_k(m)$

Note that $\beta_k(m) = \dfrac{\Pr\{Y_{k+1}^N \mid S_k = m\}}{\Pr\{Y_{k+1}^N \mid Y_1^k\}}$, and consider first the numerator. Introducing u_{k+1} and S_{k+1} yields:

$$\Pr\{Y_{k+1}^N \mid S_k = m\} = \sum_{m'=0}^{M-1}\sum_{i=0}^{1} \Pr\{u_{k+1} = i; S_{k+1} = m'; Y_{k+1}^N \mid S_k = m\}$$

$$= \sum_{m'=0}^{M-1}\sum_{i=0}^{1} \Pr\{u_{k+1} = i; S_{k+1} = m'; Y_{k+2}^N; Y_{k+1} \mid S_k = m\}$$

$$= \sum_{m'=0}^{M-1}\sum_{i=0}^{1} \Pr\{Y_{k+2}^N; (u_{k+1} = i; S_{k+1} = m'; Y_{k+1}) \mid S_k = m\}$$

Pr (A|B|C)=Pr(A|B|C)Pr(B|C) ⇒

$$= \sum_{m'=0}^{M-1}\sum_{i=0}^{1} \Pr\{Y_{k+2}^N \mid (u_{k+1} = i; S_{k+1} = m'; Y_{k+1}); S_k = m\}$$

$$\cdot \Pr\{u_{k+1} = i; S_{k+1} = m'; Y_{k+1} \mid S_k = m\}$$

Markov property ⇒

$$= \sum_{m'=0}^{M-1}\sum_{i=0}^{1} \Pr\{Y_{k+2}^N \mid S_{k+1} = m'\}$$

$$\cdot \Pr\{u_{k+1} = i; S_{k+1} = m'; Y_{k+1} \mid S_k = m\} \qquad \text{(A2.10)}$$

Consider the denominator of $\beta_k(m)$:

$$\Pr\{Y_{k+1}^N \mid Y_1^k\} = \Pr\{Y_{k+2}^N; Y_{k+1} \mid Y_1^k\}$$

Pr (A|B|C)=Pr(A|B|C)Pr(B|C) ⇒

$$= \Pr\{Y_{k+2}^N \mid Y_{k+1}; Y_1^k\} \cdot \Pr\{Y_{k+1} \mid Y_1^k\}$$

$$= \Pr\{Y_{k+2}^N \mid Y_1^{k+1}\} \cdot \Pr\{Y_{k+1} \mid Y_1^k\} \qquad \text{(A2.11)}$$

By use of relations (A2.10) and (A2.11), $\beta_k(m)$ becomes:

$$\beta_k(m) = \frac{\sum_{m'=0}^{M-1}\sum_{i=0}^{1} \Pr\{Y_{k+2}^N \mid S_{k+1} = m'\} \cdot \Pr\{u_{k+1} = i; S_{k+1} = m'; Y_{k+1} \mid S_k = m\}}{\Pr\{Y_{k+2}^N \mid Y_1^{k+1}\} \cdot \Pr\{Y_{k+1} \mid Y_1^k\}}$$

$$= \frac{\sum_{m'=0}^{M-1}\sum_{i=0}^{1} \dfrac{\Pr\{Y_{k+2}^N \mid S_{k+1} = m'\}}{\Pr\{Y_{k+2}^N \mid Y_1^{k+1}\}} \cdot \Pr\{u_{k+1} = i; S_{k+1} = m'; Y_{k+1} \mid S_k = m\}}{\Pr\{Y_{k+1} \mid Y_1^k\}}$$

$$= \frac{\sum_{m'=0}^{M-1}\sum_{i=0}^{1} \beta_{k+1}(m') \cdot \gamma_{k+1}^i(m, m')}{\Pr\{Y_{k+1} \mid Y_1^k\}}. \qquad \text{(A2.12)}$$

The denominator of relation (A2.12) is:

$$\Pr\{Y_{k+1} \mid Y_1^k\} = \sum_{m'=0}^{M-1} \sum_{i=0}^{1} \Pr\{u_{k+1} = i; S_{k+1} = m'; Y_{k+1} \mid Y_1^k\}$$

Using relation (A2.7) $\Rightarrow$

$$= \sum_{m'=0}^{M-1} \sum_{i=0}^{1} \{\sum_{m=0}^{M-1} \alpha_k(m) \cdot \gamma_{k+1}^i(m,m')\}$$

$$= \sum_{m'=0}^{M-1} \sum_{m=0}^{M-1} \alpha_k(m) \cdot \gamma_{k+1}(m,m') . \tag{A2.13}$$

By introducing relation (A2.13) to (A2.12), a recursive relation for the calculation of $\beta_k(m)$ is obtained:

$$\beta_k(m) = \frac{\sum_{m'=0}^{M-1} \sum_{i=0}^{1} \cdot \beta_{k+1}(m') \cdot \gamma_{k+1}^i(m,m')}{\sum_{m=0}^{M-1} \sum_{m'=0}^{M-1} \sum_{i=0}^{1} \alpha_k(m) \cdot \gamma_{k+1}^i(m,m')} , \tag{A2.14}$$

$$\beta_k(m) = \frac{\sum_{m'=0}^{M-1} \gamma_{k+1}(m,m') \cdot \beta_{k+1}(m')}{\sum_{m'=0}^{M-1} \sum_{m=0}^{M-1} \alpha_k(m) \cdot \gamma_{k+1}(m,m')} . \tag{A2.15}$$

APPENDIX 3

PROOFS OF $E[|\, y_k \,|]$ AND $E[y_k^2]$

This appendix provides the proofs for relations (2.78) and (2.79) in subsection 2.5.2. In order to simplify the description let $\mu = \sqrt{E_s}$.

A3.1 $E[|\, y_k \,|]$

The probability density function of the y_k is given by:

$$f(y_k) = \frac{1}{2}\left(\frac{1}{\sqrt{2\pi}\sigma} e^{-(y_k-\mu)^2/2\sigma^2} + \frac{1}{\sqrt{2\pi}\sigma} e^{-(y_k+\mu)^2/2\sigma^2} \right). \tag{A3.1}$$

$$E[|\, y_k \,|] = \int_{-\infty}^{\infty} |\, y_k \,| f(y_k)\, dy_k$$

$$= 2\int_0^{\infty} |\, y_k \,| f(y_k)\, dy_k$$

$$= 2\int_0^{\infty} y_k\, f(y_k)\, dy_k$$

$$= \frac{1}{\sqrt{2\pi}\sigma}\left\{ \int_0^{\infty} (y_k-\mu) e^{-(y_k-\mu)^2/2\sigma^2} d(y_k-\mu) + \int_0^{\infty} \mu e^{-(y_k-\mu)^2/2\sigma^2} dy_k \right.$$

$$\left. + \int_0^{\infty} (y_k+\mu) e^{-(y_k+\mu)^2/2\sigma^2} d(y_k+\mu) - \int_0^{\infty} \mu e^{-(y_k+\mu)^2/2\sigma^2} dy_k \right\}$$

$$= \frac{1}{\sqrt{2\pi}\sigma}\left\{ \sigma^2 \int_0^{\infty} e^{-(y_k-\mu)^2/2\sigma^2} d\left(\frac{(y_k-\mu)^2}{2\sigma^2}\right) + \sqrt{2}\sigma\mu \int_0^{\infty} e^{-(y_k-\mu)^2/2\sigma^2} d\left(\frac{(y_k-\mu)}{\sqrt{2}\sigma}\right) \right.$$

$$\left. + \sigma^2 \int_0^{\infty} e^{-(y_k+\mu)^2/2\sigma^2} d\left(\frac{(y_k+\mu)^2}{2\sigma^2}\right) - \sqrt{2}\sigma\mu \int_0^{\infty} e^{-(y_k+\mu)^2/2\sigma^2} d\left(\frac{(y_k+\mu)}{\sqrt{2}\sigma}\right) \right\}$$

$u_1 = \frac{y_k-\mu}{\sqrt{2}\sigma}$,

$u_2 = \frac{y_k+\mu}{\sqrt{2}\sigma} \Rightarrow$

$$= \frac{\sigma}{\sqrt{2\pi}}\left\{ -e^{-(y_k-\mu)^2/2\sigma^2} - e^{-(y_k+\mu)^2/2\sigma^2} \right\}\Big|_0^{\infty} + \frac{\mu}{\sqrt{\pi}}\left\{ \int_{\mu/\sqrt{2}\sigma}^{\infty} e^{-u_1^2} du_1 - \int_{\mu/\sqrt{2}\sigma}^{\infty} e^{-u_2^2} du_2 \right\}$$

$$= \sqrt{\frac{2}{\pi}}\sigma e^{-\mu^2/2\sigma^2} + \frac{\mu}{2}\left\{ \frac{2}{\sqrt{\pi}} \int_{\mu/\sqrt{2}\sigma}^{\infty} e^{-u_1^2} du_1 - \frac{2}{\sqrt{\pi}} \int_{\mu/\sqrt{2}\sigma}^{\infty} e^{-u_2^2} du_2 \right\}$$

$erfc(x) = \frac{2}{\sqrt{\pi}} \int_x^{\infty} e^{-u^2} du \Rightarrow$

$$= \sqrt{\frac{2}{\pi}}\sigma e^{-\mu^2/2\sigma^2} + \frac{\mu}{2}\left\{ erfc\left(-\frac{\mu}{\sqrt{2}\sigma}\right) - erfc\left(\frac{\mu}{\sqrt{2}\sigma}\right) \right\}$$

$$erf(x) = 1 - erfc(x) \Rightarrow \quad = \sqrt{\frac{2}{\pi}}\sigma e^{-\mu^2/2\sigma^2} + \frac{\mu}{2}\left\{1 - erf\left(-\frac{\mu}{\sqrt{2}\sigma}\right) - 1 + erf\left(\frac{\mu}{\sqrt{2}\sigma}\right)\right\}$$

$$erf(-x) = -erf(x) \Rightarrow \quad = \sqrt{\frac{2}{\pi}}\sigma e^{-\mu^2/2\sigma^2} + \frac{\mu}{2}\left\{erf\left(\frac{\mu}{\sqrt{2}\sigma}\right) + erf\left(\frac{\mu}{\sqrt{2}\sigma}\right)\right\}$$

$$= \sqrt{\frac{2}{\pi}}\sigma e^{-\mu^2/2\sigma^2} + \mu erf\left(\frac{\mu}{\sqrt{2}\sigma}\right)$$

$$= \sqrt{\frac{2}{\pi}}\sigma e^{-E_s/2\sigma^2} + \sqrt{E_s}\, erf\left(\sqrt{\frac{E_s}{2\sigma^2}}\right).$$

where $erfc(x)$ is the complementary error function and $erf(x)$ is the error function.

A3.2 $E[y_k^2]$

If the random variable w is normally distributed with mean μ and variance σ^2, then the second moment of w is [21, pp62]:

$$E[w^2] = \int_{-\infty}^{\infty} w^2 \frac{1}{\sqrt{2\pi}\sigma} e^{-(w-\mu)^2/2\sigma^2} dw$$

$$= \mu^2 + \sigma^2. \tag{A3.2}$$

This result is used to derive $E[y_k^2]$ as follows.

$$E[y_k^2] = \int_{-\infty}^{\infty} y_k^2 f(y_k)\, dy_k$$

$$= \int_{-\infty}^{\infty} y_k^2 \left\{\frac{1}{2}\left(\frac{1}{\sqrt{2\pi}\sigma} e^{-(y_k-\mu)^2/2\sigma^2} + \frac{1}{\sqrt{2\pi}\sigma} e^{-(y_k+\mu)^2/2\sigma^2}\right)\right\} dy_k$$

$$= \frac{1}{2}\left\{\int_{-\infty}^{\infty} y_k^2 \frac{1}{\sqrt{2\pi}\sigma} e^{-(y_k-\mu)^2/2\sigma^2} dy_k + \int_{-\infty}^{\infty} y_k^2 \frac{1}{\sqrt{2\pi}\sigma} e^{-(y_k+\mu)^2/2\sigma^2} dy_k\right\}$$

$$= \frac{1}{2}\left\{\left(\mu^2 + \sigma^2\right) + \left((-\mu)^2 + \sigma^2\right)\right\}$$

$$= \mu^2 + \sigma^2$$

$$= E_s + \sigma^2.$$

APPENDIX 4

PROGRAM SPECIFICS FOR SIMULATION OF TURBO CODES

This appendix gives a brief discussion of each program involved in the simulation of turbo codes. More complete descriptions are available in the header information of the specific programs, and also throughout the program segments. Note that parameters specified by the user are recorded in the data files in addition to the data, where the data is the source information bit stream, encoded codewords, codewords distorted by the channel, and decoded information bits. The structure of the data file consists of the parameter information and data. Generally when a new data file (except "source.dat") is created, the parameter message of the input data file is recorded again. For example, "decoded.dat" also includes source, encoder, channel and decoder parameter information.

A4.1 Source

Source bit stream generation:

Program: source.cpp

Input: seed.dat, tool.h

Output: source.dat, log.dat

Function: Generates the random source bit stream and writes the data file to "source.dat". The user is asked to specify the number of source bits to be generated and the probabilities of occurrence of the symbols. "seed.dat" is a random integer. It is updated every time after generating "source.dat", so the next time, a new random source bit sequence is produced. "tool.h" includes several commonly used functions for calculation and file input that are not available in the C++ Builder 3 software package. "log.dat" provides information regarding when the data file is updated.

A4.2 Encoder

NSC encoded code streams:

Program: NSCCodes.cpp

Input: source.dat, seed.dat, tool.h

Output: encoded.dat, log.dat

Function: Performs the encoding for code rate $1/n_0$ NSC codes by reading in one bit source words and generating n_0-bit codewords, and writing the encoded data to "encoded.dat". The user is asked to specify the number of vectors (n_0), the values of the vectors (the vectors can be typed in either binary or octal), the constraint length and the frame length. Encoding is frame-oriented. If the source data cannot be divided into an integer number of frames, the program can generate extra random data padding to the last frame such that the last frame is full. The reason for generating random data and not just padding with zeros is to maintain the original probabilities of occurrence of symbols specified by the user.

RSC encoded code streams:

Program: RSCCodes.cpp

Input: source.dat, seed.dat, tool.h

Output: encoded.dat, log.dat

Function: The process is as same as NSC encoding except that RSC codes are implemented.

Turbo code streams:

Program: TurboCodes.cpp

Input: source.dat, seed.dat, tool.h

Output: encoded.dat, log.dat

Function: Generates punctured (code rate 1/2) or unpunctured (code rate 1/3) turbo codes. The default number of vectors is two; the user is asked to specify the values of these vectors. Also the user is asked to specify whether or not the code is punctured, the constraint length, the frame length, and the type of interleaver. There are three choices of interleavers: Berrou, PIL and spread random. There are two termination schemes including single termination and joint termination. If joint termination is specified, the information corresponding to termination positions is written to a data file for use in decoding. There are many choices for termination positions, but the program searches the termination positions starting from the last row of $\boldsymbol{F}$ and chooses the first available result. The information regarding source and encoder parameters and the encoded words are written into "encoded.dat".

Turbo code streams with CRC:

Program: TurboCodesCRC.cpp

Input: source.dat, seed.dat, tool.h

Output: encoded.dat, log.dat

Function: The program is designed to integrate short CRC codes with turbo codes to improve the performance of error detection. The user is required to specify the CRC generator in addition to parameters that are needed for turbo encoding.

A4.3 Channel

Discrete AWGN:

Program: AWGNChannel.cpp

Input: encoded.dat, seed.dat, tool.h

Output: demoded.dat, log.dat

Function: This program generates samples of additive white Gaussian noise. The program integrates three functions together for simplicity. First, binary phase shift keying modulation is assumed, so each logic "1" is transferred to "+1" and each logic "0" is transferred to "-1". Second, a Gaussian noise sample is generated according to the SNR specified by the user and added to the modulated data. Third, the user is asked to specify the demodulation style that determines if channel transition probabilities are required. If so, they are included in the data file "demoded.dat". Demodulation choices include hard or soft decision. If the user chooses soft decision, the user will be asked if perfect soft decision is selected. If so, perfect soft decision data (no quantization) will be written to "demoded.dat" directly. If not, then quantized soft decision has been selected, so the user is asked to specify the number of quantization levels. For hard decision, where the default is two quantization levels, or quantized soft decision, which implies the discrete Gaussian memoryless channel, the transition probabilities are generated and written to data files, and the output data are quantized values denoted with integer values. This program has strong connection with decoding programs. In this thesis, for the VA and BCJR decoding programs in which the Gaussian channel will be quantized, the transition probabilities are required for decoding. For the M-BCJR algorithm and turbo decoding algorithm, it is not necessary to quantize the channel, so transition probabilities are not required.

A4.4 Decoder

In the following programs, decoding is frame-oriented. As each frame is decoded, the termination bits are discarded, the decoded sequence is compared with the source sequence, and relevant error information is collected. After all data has been decoded, error information is written to "ErrorRate.dat" or "ErrorRateAll.dat". The decoding programs obtain the relevant information from the input data file automatically, and display a message on the monitor of the computer which includes the type of encoder, the trellis diagram (input, branch values and state transitions), constraint length, generators, code rate, frame length, E_b / N_0 for the channel, the time when the decoding process is half completed, and the time when decoding starts and ends. Note that for the Viterbi decoding algorithm, the state transitions of the trellis are shown for previous states, and for other decoding algorithms, the state transitions of the trellis are shown for future states.

Viterbi decoding algorithm:

Program: VA.cpp

Input: demoded.dat, source.dat, tool.h

Output: ErrorRate.dat, log.dat

Function: Decodes the received codewords using the Viterbi decoding algorithm. Writes the bit error rate and frame error rate to "ErrorRate.dat".

BCJR decoding algorithm:

Program: BCJR.cpp

Input: demoded.dat, source.dat, tool.h

Output: ErrorRate.dat, log.dat

Function: Decodes the received codewords using the BCJR decoding algorithm. Writes the bit error rate and frame error rate to "ErrorRate.dat".

M-BCJR decoding algorithm for RSC codes:

Program: M_BCJR.cpp

Input: demoded.dat, source.dat, tool.h

Output: ErrorRate.dat, log.dat

Function: Decodes the received codewords corrupted by Gaussian noise using the M-BCJR decoding algorithm. Note that the encoder should be an RSC encoder, and the channel

is not quantized. The decoder needs the encoded symbol energy $\sqrt{E_s}$ and the variance of the channel σ^2 for decoding. For simplicity, perfect estimation is assumed. The decoder uses $\sqrt{E_s}=1$ and the value of E_b/N_0 recorded in "demoded.dat" for decoding (note that from $\sqrt{E_s}$, E_b/N_0 and code rate R, the variance of the channel can be obtained). This assumption is also made for turbo coding programs. Writes the bit error rate and frame error rate to "ErrorRate.dat".

Berrou's decoding algorithm for turbo codes:

Program: BerrouTurbo.cpp

Input: demoded.dat, source.dat, tool.h

Output: ErrorRate.dat, log.dat

Function: Decodes the received turbo codewords corrupted by Gaussian noise using Berrou's decoding algorithm. The user is asked to specify the number of iterations. The greater the number of iterations, the more time required for decoding. The interleaver style is obtained from "demoded.dat" automatically. Writes the bit error rate and frame error rate to "ErrorRate.dat".

Robertson's decoding algorithm for turbo codes:

Program: RobertsonTurbo.cpp

Input: demoded.dat, source.dat, tool.h

Output: ErrorRate.dat, log.dat

Function: Decodes the received turbo codewords corrupted by Gaussian noise using Robertson's decoding algorithm. Writes the bit error rate and frame error rate to "ErrorRate.dat".

Decoding algorithm with early stopping:

Program: EarlyStoppingHSC.cpp

Input: demoded.dat, source.dat, tool.h

Output: ErrorRate.dat, log.dat

Function: Decodes the received turbo codewords using early stopping criteria including HDA, SCR and CE. The user is required to indicate the maximum number of iterations and specify which stopping criterion is to be used. After selecting the stopping criterion, the user may be required to specify a threshold because for SCR and CE, the threshold may change for different channel conditions for better performance. In addition, the

average number of iterations is evaluated. Writes the bit error rate, frame error rate and the average number of iterations to "ErrorRate.dat".

Decoding algorithm with early stopping error detection:

Program: StoppingErrorDetection.cpp

Input: demoded.dat, tool.h, source.dat

Output: ErrorRateAll.dat, log.dat

Function: Decodes the received turbo codewords using ME and MSC. Writes the bit error rate, frame error rate, missed detection rate, false alarm rate, average number of errors per missed detection and the average number of iterations to "ErrorRateAll.dat".

Decoding algorithm with early stopping error detection and CRC check:

Program: StoppingErrorDetectionCRC.cpp

Input: demoded.dat, source.dat, tool.h

Output: ErrorRateAll.dat, log.dat

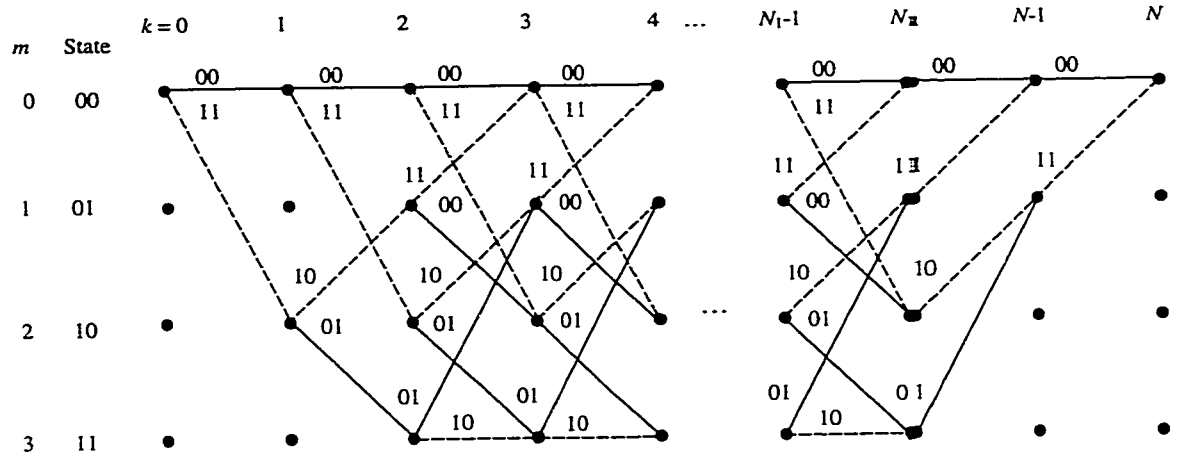
Function: Decodes the received turbo codewords using MSC-CRC. Writes the bit error rate, frame error rate, missed detection rate, false alarm rate, average number of errors per missed detection and the average number of iterations to "ErrorRateAll.dat".

APPENDIX 5

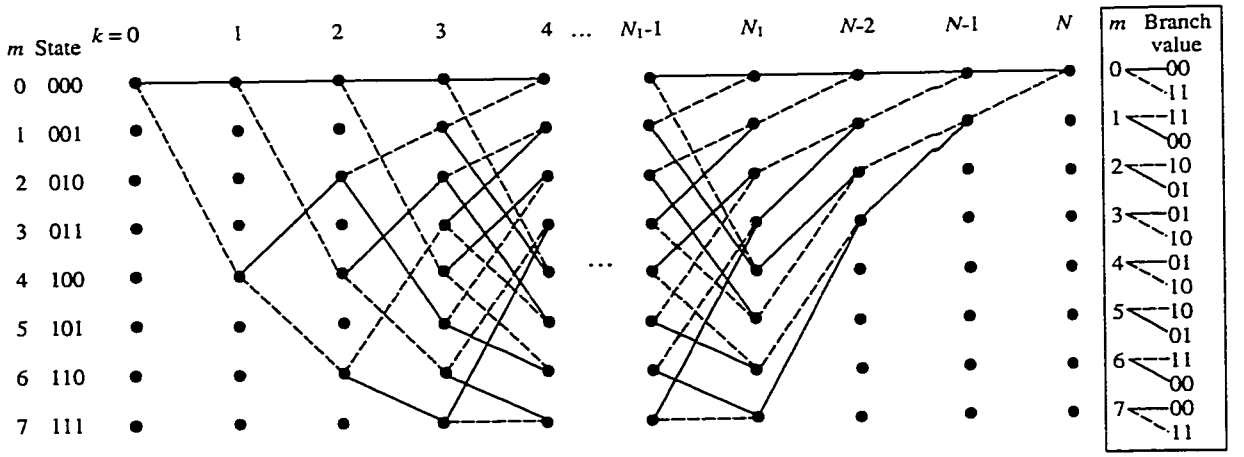
TRELLIS DIAGRAMS FOR GENERAL USE OF RSC CODES

This appendix presents three trellis diagrams of RSC codes with generators $G(7,5)$, $G(13,15)$, $G(37,21)$, and $G(31,33)$ that are used in this thesis. In these trellis diagrams, solid lines correspond to paths taken when input bit is a 0, and dashed lines correspond to paths taken when the input bit is a 1. Note that for the trellis diagrams with generators $G(13,15)$, $G(37,21)$, and G(31,33) the branch values are separately placed to the right of the trellis diagrams.

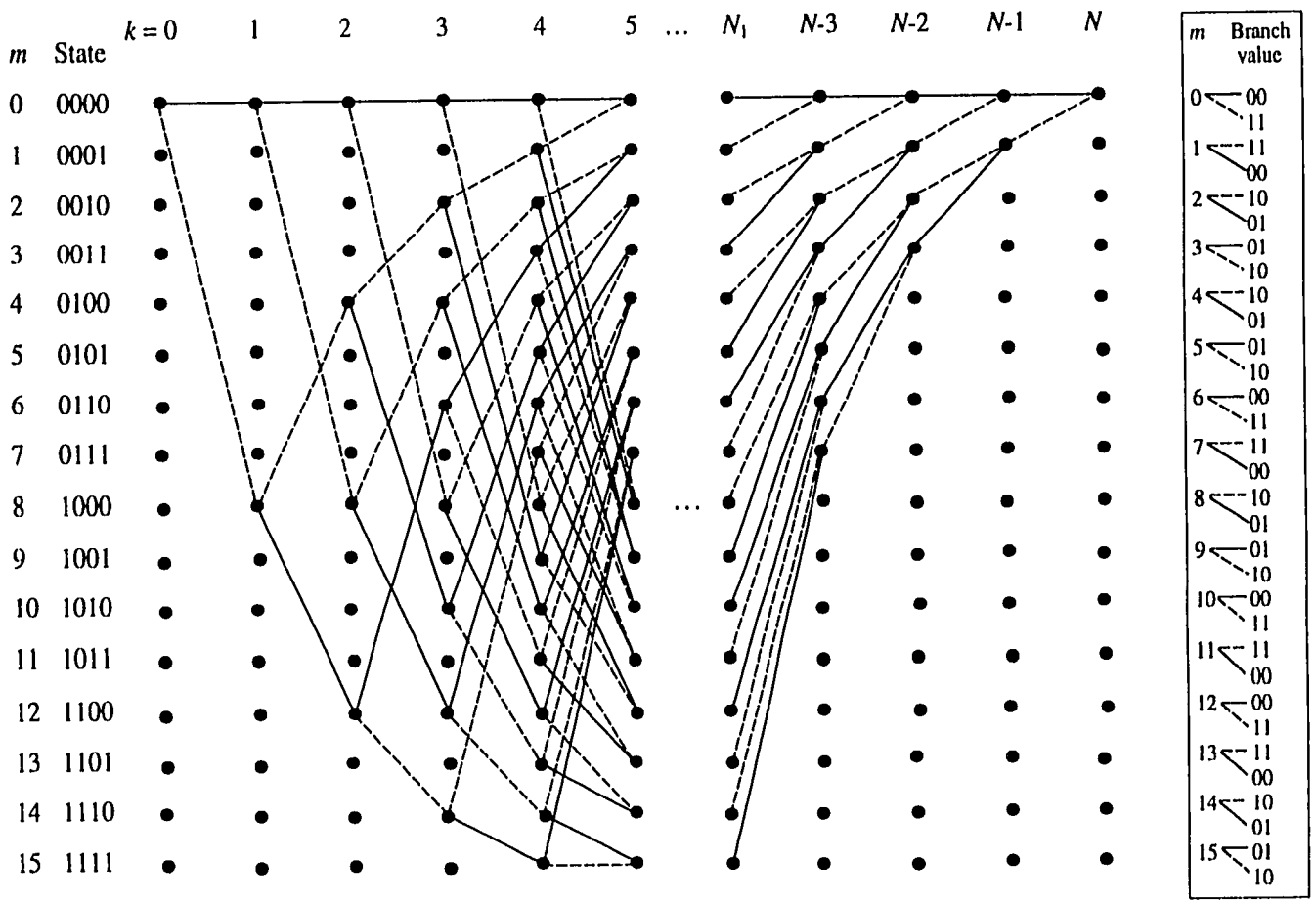
A5.1 $G(7,5)$



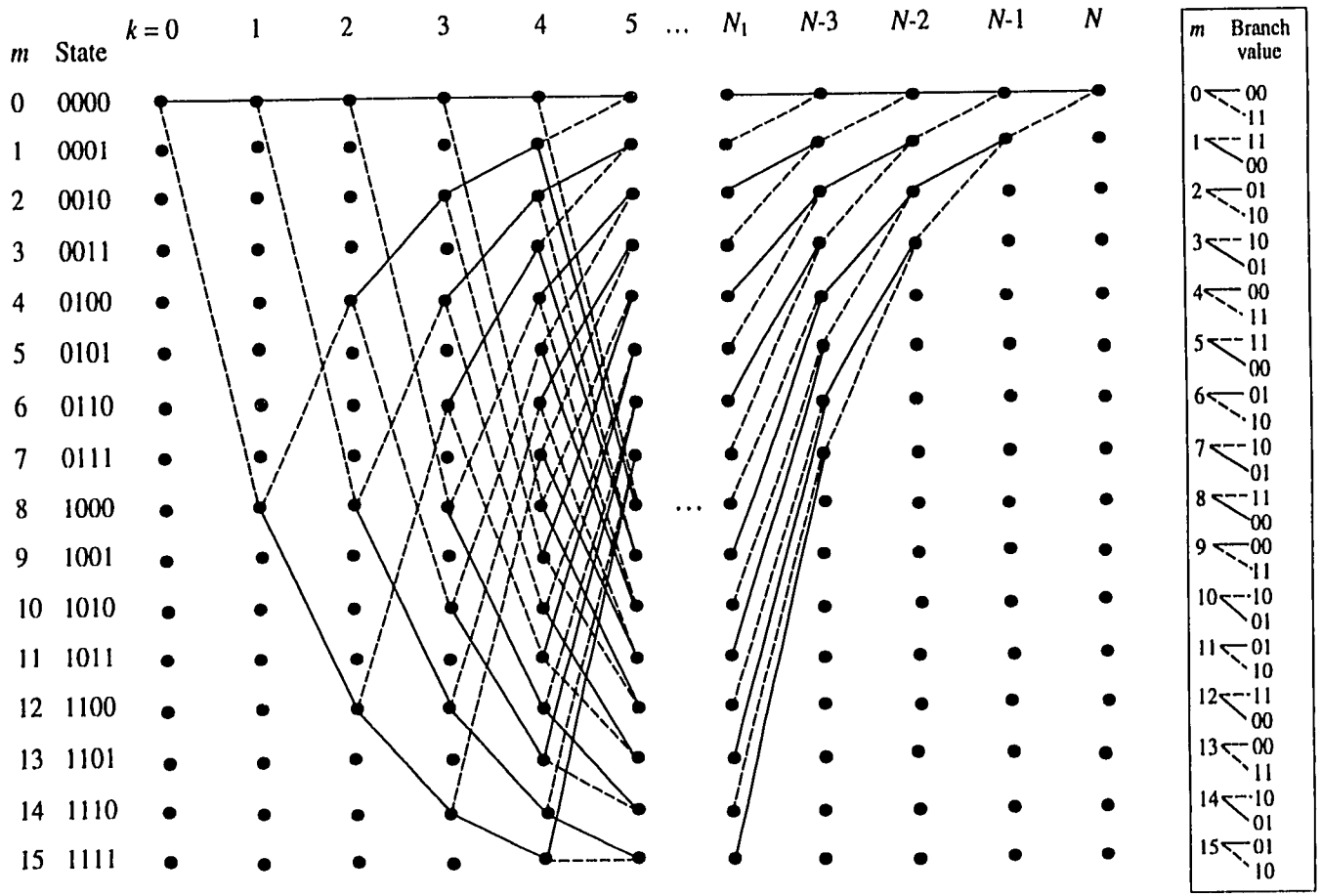
A5.2 G(13,15)



A5.3 *G*(37,21)


k = 0
1
2
3
4
5
...
N1
N-3
N-2
N-1
N
m
State
0 0000
1 0001
2 0010
3 0011
4 0100
5 0101
6 0110
7 0111
8 1000
9 1001
10 1010
11 1011
12 1100
13 1101
14 1110
15 1111
m
Branch value
0 00 11
1 11 00
2 10 01
3 01 10
4 10 01
5 01 10
6 00 11
7 11 00
8 10 01
9 01 10
10 00 11
11 11 00
12 00 11
13 11 00
14 10 01
15 01 10

A5.4 $G(31,33)$


m State
k = 0
1
2
3
4
5
...
N1
N-3
N-2
N-1
N
0 0000
1 0001
2 0010
3 0011
4 0100
5 0101
6 0110
7 0111
8 1000
9 1001
10 1010
11 1011
12 1100
13 1101
14 1110
15 1111
m Branch value
0 00 11
1 11 00
2 01 10
3 10 01
4 00 11
5 11 00
6 01 10
7 10 01
8 11 00
9 00 11
10 10 01
11 01 10
12 11 00
13 00 11
14 10 01
15 01 10

APPENDIX 6

LIST OF ABBREVIATIONS

3-GPP — 3rd Generation Partnership Project

The 3rd Generation Partnership Project is developing technical specifications for IMT-2000, the International Telecommunication Union's (ITU) framework for third-generation standards. 3GPP is a global co-operation between six Organizational Partners (ARIB, CWTS, ETSI, T1, TTA and TTC) who are recognised as being the world's major standardization bodies from Japan, China, Europe, USA and Korea.

Note:

ARIB	Association of Radio Industries and Businesses, Japan
CWTS	China Wireless Telecommunication Standards organization
ETSI	the European Telecommunications Standards Institute
T1	committee T1 (standards committee T1 telecommunications, USA)
TTA	Telecommunication Technology Association, Korea
TTC	Telecommunication Technology Committee, Japan

A/D — Analog-to-Digital

The A/D conversion is the process of changing continuously varying data into digital quantities that represent the magnitude of the data at the moment the conversion is made. The most common use is to change analog signals into digital signals that can be used in data communications. The digital-to-analog (D/A) conversion is the inverse process of the A/D.

AGC — Automatic Gain Control

The AGC is an adaptive system that operates over a wide dynamic range while maintaining the output signal at a constant (signal plus noise) level. The AGC is designed specifically for modem applications.

APP — *A Posteriori* Probability

The probability $\Pr(u_k = i \mid Y_1^N)$ is an APP for u_k having the value of element i after Y_1^N is received, where Y_1^N is a frame of the demodulated coded sequence with frame size N, and u_k is the k-th symbol in the information sequence.

ARQ — Automatic Repeat-reQuest

ARQ is an error control scheme for data transmission in which the receiver detects transmission errors in a message and automatically requests a retransmission from the transmitter. Usually, when the transmitter receives the request, the transmitter retransmits the message until it is either correctly received or until the error persists beyond a predetermined number of retransmissions.

AWGN — Additive White Gaussian Noise

An uncorrelated Gaussian (normally distributed) noise process that is independent of the transmitted signals, corrupts the signal through addition, and has flat power spectral density for all frequencies. The adjective "white" is used in the sense that white light contains equal amounts of all frequencies within the visible band of electromagnetic radiation.

BCJR — Bahl, Cocke, Jelinek and Raviv

The BCIR algorithm minimizes the symbol error probability for linear codes. The BCJR algorithm generates the APPs of the states and transitions on the encoder trellis.

BER — Bit Error Rate

BER is the probability that a message bit is incorrect. The number of erroneous bits is averaged over the entire number of bits transmitted.

BPSK — Binary Phase Shift Keying

BPSK is a binary modulation format in which the data is modulated onto the carrier by varying the phase of the carrier by $\pm\pi$ radians.

CE — Cross Entropy

The CE is a criterion for stopping an iterative decoder based on the closeness of the two distributions that are obtained from the outputs of two constituent decoders.

CRC — Cyclic Redundancy Check

The CRC is an error detection scheme that appends parity bits generated by polynomial encoding of digital signals to the end of the data sequence, and uses these parity bits to detect the presence of errors in the received digital signal.

CRC-12 $D^{12}+D^{11}+D^3+D^2+D+1$

An international standard for CRC error detection.

CRC-16 $D^{16}+D^{15}+D^2+1$

An international standard for CRC error detection.

CRC-CCITT $D^{16}+D^{12}+D^5+1$

An international standard for CRC error detection.

DMC Discrete Memoryless Channel

The DMC is characterized by a discrete input alphabet, a discrete output alphabet, and a set of conditional probabilities for all output symbols given the input symbols. The channel noise affects each input symbol independently of all other input symbols.

DEC DECoder

A decoder is a device that attempts to restore the received, encoded signal to the original message.

DEDN-CE Decoder Error Detecting Network using CE

The DEDN-CE is a neural network trained to detect errors in the decoded frames of data for turbo codes. The DEDN-CE maximizes the reliability by basing retransmission requests on a highly accurate determination of whether errors are present in the decoded data.

DEDNpost-CE backup Decoder Error Detecting Network using CE

The DEDNpost-CE is a neural network trained to detect errors in a decoded frame of data for turbo codes. The DEDNpost-CE checks all frames that are accepted by the FEDN-CE, providing enhanced reliability.

ENC ENCoder

An encoder is a device that transforms a source signal into an encoded signal for transmission.

FAR False Alarm Rate

FAR is the probability that an error-free frame is judged to be erroneous by imperfect detection schemes. In order to obtain the FAR, the number of false alarmed frames is averaged over the entire number of frames transmitted.

FEDN-CE — Future Error Detecting Network using CE

The FEDN-CE is a neural network trained to detect errors in a decoded frame of data for turbo codes. The FEDN-CE minimizes decoder complexity by determining whether or not the frame is likely to be accurately decoded in future iterations, and if not, terminates decoding process early.

FER — Frame Error Rate

FER is the probability that a decoded frame of data contains errors. In order to obtain the FER, the number of erroneous frames is averaged over the entire number of frames transmitted.

GF(2) — A Galois Field of order 2

Finite fields were discovered by Evariste Galois and are thus known as Galois fields. A Galois field of order q is usually denoted GF(q). The simplest Galois field is GF(2). GF(2) can be represented by the two element set {0,1} where addition is defined as an exclusive-OR operation and multiplication is the binary AND.

HDA — Hard-Decision-Aided

HDA is a criterion for stopping iterative decoding based on the number of sign changes of the LLR values for a constituent decoder from two consecutive iterations.

IRWEF — Input Redundancy Weight Enumerating Function

The IRWEF is a special enumerating function for constituent codes in turbo codes. The IRWEF explicitly separates the weight contributions of the information bits and the parity check bits.

LLR — Log-Likelihood Ratio

LLR is a useful metric evaluated by taking the logarithm of the ratio of two probabilities. The LLR of u_k, the k-th symbol in the information sequence, is $L(u_k) = \log \frac{\Pr(u_k = +1)}{\Pr(u_k = -1)}$, where $u_k \in \{+1, -1\}$. After Y_1^N, a frame of coded sequence with frame length N, is received, the LLR of the APPs of u_k is $L(u_k \mid Y_1^N) = \log \frac{\Pr(u_k = +1 \mid Y_1^N)}{\Pr(u_k = -1 \mid Y_1^N)}$.

MAP — Maximum *A Posteriori*

The MAP criterion is a detection criterion that leads to the selection of i that maximizes the probability $\Pr(i \mid Y)$ for some received Y. It is also called minimum error criterion, since on average this criterion yields the minimum number of incorrect decisions. Note that this criterion is optimum only when all types of errors are equally harmful or costly. When some of the error types are more costly than others, a criterion that incorporates relative cost of errors should be employed.

M-BCJR — Modified BCJR

The M-BCJR algorithm is obtained by modifying the BCJR algorithm for decoding an RSC code.

MDR — Missed Detection Rate

The MDR is the probability that an erroneous frame is detected to be error-free by imperfect detection schemes. In order to obtain the MDR, the number of missed detection frames is averaged over the entire number of frames transmitted.

ME — Mean Estimate

ME is a criterion for early stopping and error detection in turbo decoding. In this criterion, the mean of the absolute LLR values is examined to determine whether the iterative decoding process should be stopped and whether the decoded frame is error-free.

ML — Maximum Likelihood

The ML is a detection criterion that leads to the selection of i that maximize the probability $\Pr(Y \mid i)$ for some received Y. If the source, denoted by i, is equiprobable, then ML is equivalent to the MAP.

MSC — Mean-Sign-Change

MSC is a criterion for early stopping and error detection in turbo decoding. In this criterion, the mean of the absolute LLR values and the number of sign changes of LLR values are considered to determine whether the iterative decoding process should be stopped and whether the decoded frame is error-free.

MSC-CRC	Mean-Sign-Change and Cyclic Redundancy Check MSC-CRC is a criterion for early stopping and error detection in turbo decoding. In this criterion, low order primitive CRC polynomials are combined with the MSC criterion to improve the error detection performance of MSC.
NSC	Non-Systematic Convolutional An NSC code is one in which the input data is not visible in the output code word.
PCBC	Parallel Concatenated Block Code A PCBC is obtained by concatenating two systematic block codes that are linked through an interleaver.
PCCCs	Parallel Concatenated Convolutional Codes PCCCs are obtained through parallel concatenation of convolutional encoders with interleavers. In this thesis, PCCCs are referred to as turbo codes.
PIL	Prime Interleaver PIL is based on the construction of a rectangular mother interleaver and pruning this mother interleaver if it is larger than the required size. The key features of PIL are low complexity, good pseudo-random nature, and a wide range of interleaving sizes from 240 to 8200.
RSC	Recursive Systematic Convolutional An RSC code is obtained from an NSC code by using a feedback loop and setting one of the output binary sequences equal to the input sequence.
SC	Systematic Convolutional An SC code is one in which the input data is visible in the output code word.
SCCCs	Serially Concatenated Convolutional Codes The SCCCs are generated through serial concatenation of convolutional encoders with interleavers.
SCR	Sign-Change-Ratio SCR is a criterion for early stopping in turbo decoders. SCR considers the sign changes in extrinsic information values from a constituent decoder over two consecutive iterations.

SNR	Signal-to-Noise Ratio SNR is a ratio of average signal power to average noise power.
SR	Spread Random The SR interleaver is based on the random generation of N integers from 1 to N with a minimum constraint on the amount of spreading between adjacent positions.
UMTS	Universal Mobile Telecommunications System UMTS is part of the International Telecommunications Union's "IMT-2000" vision of a global family of third-generation (3G) mobile communications systems. UMTS will play a key role in creating the future mass market for high-quality wireless multimedia communications
VA	Viterbi Algorithm The VA is a maximum likelihood decoding method that minimizes the probability of word errors for convolutional codes.
VE	Variance Estimate VE is the first early stopping criterion proposed for turbo decoding. The meta-channel variance is estimated to determine whether the iterative decoding process should be stopped.
WEF	Weight Enumerating Function The WEF is a representation of the weight distribution for a block code.