TURBO CODE IN CDMA SYSTEM

SUN JIANHUA
WANG QI

SCHOOL OF ELECTRICAL & ELECTRONIC ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY
1997/98

NANYANG TECHNOLOGICAL UNIVERSITY

SCHOOL OF ELECTRICAL AND ELECTRIONIC ENGINEERING

Submitted by:

Sun Jianhua

Wang Qi

Report

on

Final Year Project

Project 5045

Turbo Code in CDMA System

A Final Year Project presented to the Nanyang Technological University in partial fulfillment of the requirements for the degree of Bachelor of Engineering

March 1998

Abstract

The working principle of turbo code was first proposed by Berrou *et al.* in 1993[1]. Near Shannon limit error correcting capacity was achieved by application of turbo code, astonishing the whole coding community. However, some important details required to reproduce the simulation results were omitted in the original paper. This project confirms the results claimed by Berrou *et al.*, except that the requirement for stability factor for large number of iterations and estimation of variance of soft output from a component decoder addressed in [1] is found to be unnecessary with slight modification of the turbo decoder. The project also investigates the effect of various factors on the performance of turbo code, like interleaver type, frame size, encoder constraint length and approximation methods used. To simplify computation required for turbo decoding, various approximation methods are studied and proved to have comparable performance. A simple method for estimating channel noise variance which is important to the performance of turbo decoder but seldom mentioned in literature is proposed. It is found to work perfectly well for long sequence. The feasibility of turbo code in CDMA system is studied. It is shown that the performance of turbo code for short frames as required in CDMA system is not so impressive as that for long sequence proposed for deep space communications. However, turbo code with constraint length $K = 5$ can achieve coding gain of about 1 dB over conventional coding schemes with $K = 9$.

Acknowledgments

The final year project students would like to express their deep gratitude to the project supervisors **Dr. Mow Wai Ho** and **Dr. Li Kwok Hung** for their continuous guidance and help during the past 10 months. They shared with the students their wide knowledge on modern communication engineering and software development by giving lots of useful advice and suggestions on the technical problems encountered during the project.

Last but not least, the students would like to thank all the staff in the Workstation Resource Lab for providing technical support to conduct computer simulation.

Contribution Page

		Sun Jianhua	Wang Qi
Chapter 1	Introduction	⋆	
Chapter 2	Consideration and Hypothesis in Software Implementation	⋆	⋆
Chapter 3	Maximum A Posteriori Component Decoder		⋆
Chapter 4	Turbo Decoder	⋆	
Chapter 5	Factors Affecting Performance of Turbo Code	⋆	
Chapter 6	Application of Turbo Code to CDMA System		⋆
Chapter 7	Conclusion & Recommendations		⋆

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Communication is defined as the transmission of information from one point to another through a succession of processes. When a signal is transmitted through a channel, it is distorted because of channel imperfection. Furthermore, noise and interference from other users sharing the same channel are superimposed on the desired signal, making it a corrupted version of the original signal at the receiving end. To ensure an acceptable level of reliability and quality for the information being transmitted, we can either increase the signal power or use larger bandwidth. It may not be practical to raise the signal power in real life, the option left to improve the data quality is the application of *error control coding.* Coding is a process of introducing *redundancy* to the original data bits, which on one hand increases transmission bandwidth and system complexity, on the other hand helps the receiver to retain the original data from the corrupted version with higher reliability (lower bit error rate). A typical communication system is depicted in Figure 1.1.

Channel coding theorem states that if a source generates information at a rate less than the channel capacity, it is possible to achieve zero error transmission with proper coding schemes. However, this theorem simply asserts the existence of such a coding technique without specifying how to find it. Currently, there are many different

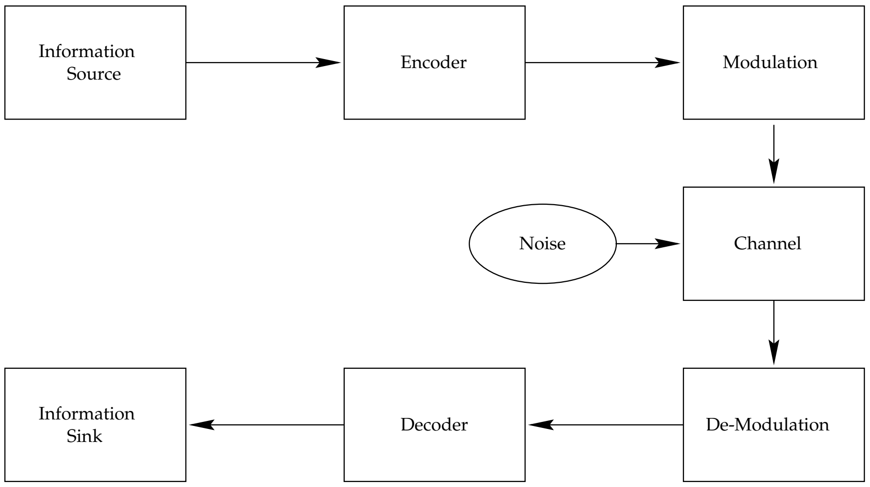


Figure 1.1: A typical communication system

error correcting codes which can be classified into two main categories, block codes and convolutional codes. The major difference between these two is the presence or absence of memory (shift register) in the encoder. Output bits from an encoder are redundancies introduced and termed as parity bits. If input bits are left unaltered by the encoder, they are called systematic bits.

Two famous algorithms for decoding convolutional codes are Viterbi algorithm and BCJR algorithm [1]. These two algorithms can provide satisfactory error correcting performance, but are still far from Shannon limit. One breakthrough in coding theory in recent years is the announcement of turbo code which can achieve performance very near the Shannon limit by iterative decoding. The idea of turbo code is actually a combination of some existing concepts, like *Recursive Systematic Convolutional* (RSC) encoding, *soft-in/soft-out* decoder, interleaver and iterative decoding. Turbo code is expected to play an important role in near future communications because of its extraordinary performance.

1.2 Scope

This final year project covers the following topics:

[1]The algorithm is usually referred to in recent literature as the "Bahl algorithm", BCJR is used in this report to credit all the authors: L. R. Bahl, J. Cocke, F. Jenlinek and J. Raviv.

1. Study of working principle of BCJR algorithm.
2. Implementation of MAP decoder using BCJR algorithm both in Matlab and C.
3. Use of approximation methods and log operations to reduce calculation burden.
4. Investigation of the theory of turbo code.
5. Implementation of turbo decoder in C.
6. Conduct simulation for evaluation of turbo code.
7. Study the performance of turbo decoder for different cases.
8. Investigate the feasibility of applying turbo code in CDMA system.

1.3 Organization of the Report

The report is organized as follows. Chapter 1 gives a brief introduction on the final year project. Chapter 2 illustrates the channel model applied and the Gaussian noise generator used throughout this project. In Chapter 3, equations are used to explain the working principle of BCJR MAP decoder and how it is implemented in log domain with various approximation methods to reduce computation effort. Chapter 4 provides an insight into the working principle of turbo code. The effect of varying some parameter values with simulation results is shown in the following chapter. Chapter 6 studies the feasibility of applying turbo code in CDMA system. Finally, conclusion and recommendations are made in Chapter 7. The flowcharts for the implementation of the BCJR MAP decoder and turbo decoder are given in the appendix.

Chapter 2

Consideration and Hypothesis in Software Implementation

2.1 Program Language and Data Representation

C is chosen as the programming language in this project, because it can produce compact code with high execution speed. For algorithm discussed in Section 3.4, variables are declared as *double*, while constants are *int* (integer). After simplification in Section 3.5, variables can be down-graded to *float*. Of course, the precision of data representation in software simulation cannot be realized in hardware implementation. Quantization of finite bit representation will lead to minor degradation in *bit error rate* (BER) performance of the decoder. We do not consider this quantization effect in this project. All simulation is carried out in HP workstation. The following table summarizes the basic data types recognized by the C-compiler in UNIX.

	Bytes	Range and Precision
long int	4	-2147483648 to +2147483647
float	4	$\pm 1.3\text{E} \pm 38$, 7-digit precision
double	8	$\pm 1.8\text{E} \pm 308$, 15-digit precision

Table 2.1: Basic C data types

2.2 Communication Channel, Modulation and E_b/N_0

Every practical communication system experiences interference from random noise. The noise comes from different sources: inherent thermal noise inside the components that make up the system itself and electromagnetic interference from other communication processes. Due to presence of noise, the received information is random in nature. In order to study the communication processes and provide reliable transmission, the random noise arising from the channel must be modeled. Gaussian random process is the most commonly used model. It has the largest differential entropy attainable by any random variable for a finite variance σ^2[17]. The probability density function (p.d.f) of Gaussian random variable is given by:

$$f_x(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\{-\frac{(x-\mu_x)^2}{2\sigma_x^2}\}, \quad -\infty < x < \infty, \tag{2.1}$$

The Gaussian random noise arising from communication channel is assumed to be additive. The received symbols are the summation of original data and noise. In the remaining sections, such channel is called *Additive White Gaussian Noise* (AWGN) channel.

Information source is assumed to be binary and equi-probable. The output bits d_k (k is time index) from the source has the property of:

$$d_k \in [0, 1] \tag{2.2}$$

$$Pr(d_k = 1) = Pr(d_k = 0) = 0.5. \tag{2.3}$$

Binary Phase Shift Keying (BPSK) modulation is assumed. With this modulation, output d_k is mapped to X_k by

$$X_k = 2d_k - 1. \tag{2.4}$$

When X_k is transmitted along the channel, noise is superimposed on it and the noisy version is given by

$$x_k = X_k + n_k, \tag{2.5}$$

where n_k is a Gaussian random noise.

Ratio between energy per bit to single sided noise power density (E_b/N_0) is a key parameter that indicates how effective a coding scheme makes use of transmitting power. With AWGN channel, BPSK modulation and coherent detection, E_b/N_0 can be calculated as follows:

$$E_b/N_0 = \frac{S^2}{2r\sigma^2}, \tag{2.6}$$

where r is code rate, which is defined in Section 3.1, S is the signal strength and σ^2 is the variance of AWGN noise. As discussed in Section 3.3, convolutional decoding needs termination bits, which is equal to the number of shift registers M in convolutional encoder. When the termination tail bits are appended to the data sequence, the code rate r is changed as follows:

$$r' = r\frac{L}{L+M} \tag{2.7}$$

where r' is the new code rate, L is the data sequence length. This affects the calculation of E_b/N_0. Normally, L is much larger than M for large block sequence. Thus the error introduced is negligible. In this project, unless otherwise stated, the tail bits are ignored in the calculation of E_b/N_0 using Equation (2.6).

For uncoded transmission, with BPSK modulation, coherent detection and code rate $r = 1$, the bit error rate P_e is given by

$$P_e = Q\left(\sqrt{\frac{2E_b}{N_o}}\right) \tag{2.8}$$

where Q is the *Complementary Error Function* defined as

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^{\infty} \exp\left(-\frac{u^2}{2}\right) \mathrm{d}u \tag{2.9}$$

2.3 Uniformly Distributed Random Number Generator

How to generate an uniformly distributed random number over a certain range is actually an academic issue to be explored. Here, we invoke the simple standard function **rand()** in C library to generate uniformly distributed random numbers. **rand()** returns the number between zero and RAND_MAX (32767). The following equation:

$$u = \frac{\text{rand()}}{\text{RAND\_MAX}}$$

ensures that u is uniformly distributed and ranged between zero and one. To enhance the randomness, **rand()** will be initialised by calling the current time and key counter in the program during execution.

2.4 Gaussian Random Number Generator

The Gaussian random number (noise) generator can be implemented in several ways. Two methods are considered in this project.

Central Limit Theorem This algorithm is an intuitive way of generating Gaussian random numbers. A simple proof can be found in [17].

C-1 Generate N independent random variables $\{u_1, ..., u_N\}$ uniformly distributed between zero and one.

C-2 Compute $\overline{u} = \frac{\sum_{i=1}^{N} u_i}{N}$.

C-3 Compute $v = \overline{u} - \frac{N}{2}$.

C-4 Compute $v = \sigma v + \mu$ to make v normally distributed with mean μ and variance σ^2.

Polar Method for Normal Deviates This algorithm and its proof can be found in [16].

P-1 Generate two independent random variable, u_1 and u_2, uniformly distributed between zero and one.

P-2 Compute $v_1 = 2u_1 - 1$ and $v_2 = 2u_2 - 1$ to make v_1 and v_2 uniformly distributed between -1 and $+1$.

P-3 Compute $S = v_1^2 + v_2^2$.

P-4 If $S \geq 1$, return to Step 1.

P-5 Compute

$$x_1 = v_1\sqrt{\frac{-\ln S}{S}}, \qquad x_2 = v_2\sqrt{\frac{-\ln S}{S}}.$$

P-6 Both x_1 and x_2 are normally distributed variable with mean zero and variance one.

P-7 Compute $x_1 = \sigma x_1 + \mu$ and $x_2 = \sigma x_2 + \mu$ to make x_1 and x_2 normally distributed with mean μ and variance σ^2.

Simulation has been conducted to verify the performance of these methods. All the Gaussian noise needed in the project has a mean of zero. For various variance σ^2, a sequence of Pseudo-Gaussian random variable has been generated with length L. For the two methods with different L, the mean and variance of the sequence are examined in Table 2.2. The factor N in Central Limit Theorem is taken as 12. The performance deviations between these two methods is not so great and Polar Method for Normal Deviates is slightly better than Central Limit Theorem. The length of the variable sequence L affects the output statistical property of the Gaussian noise generator. As L gets greater, the random variable distribution is more like Gaussian distribution. The distribution plots of the Pseudo-Gaussian noise by the two method

Method	Seq. Length	σ Specified	σ Simulated	Mean Simulated
Central Limit Theorem	$L = 1,000$	2.00	2.079	-0.1219
		1.60	1.614	-0.122
		1.20	1.247	0.010
		0.80	0.840	-0.007
	$L = 10,000$	2.00	1.979	0.013
		1.60	1.593	0.004
		1.20	1.200	0.005
		0.80	0.796	0.008
Polar Method for Normal Deviates	$L = 1000$	2.00	2.021	-0.027
		1.60	1.618	-0.011
		1.20	1.192	0.075
		0.80	0.830	-0.061
	$L = 10,000$	2.00	1.997	0.009
		1.60	1.678	0.020
		1.20	1.217	-0.013
		0.80	0.813	-0.002

Table 2.2: Performances of two Gaussian random variable generation methods

are depicted in Figure 2.1. Center Limit Theorem is used in this project because of its simpler computation.

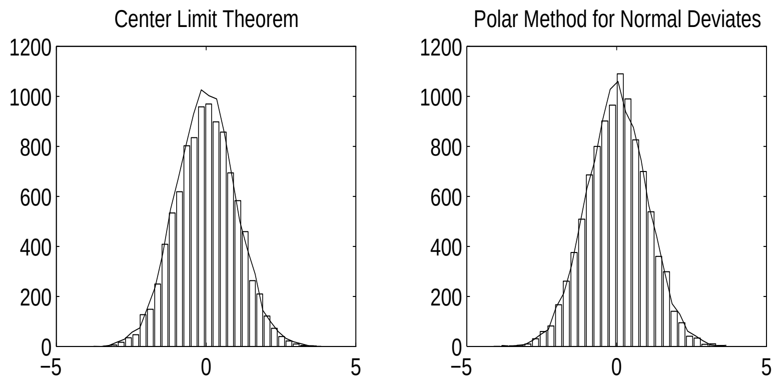


Figure 2.1: Distribution of the two Gaussian noise generators

Chapter 3

Maximum A Posteriori Component Decoder

In digital communication, convolutional error correcting or channel coding is a common technique used to combat channel noise. There are two convolutional decoding algorithms which offer good performance: Viterbi algorithm and BCJR algorithm. Both algorithms are maximum likelihood decoding method, while Viterbi algorithm minimizes the probability of sequence error and BCJR algorithm minimizes the probability of bit error. This chapter is devoted to the *Maximum A Posteriori* (MAP) decoder which makes use of BCJR algorithm. Basic concepts for convolutional code, trellis diagram and Viterbi algorithm will be described briefly.

3.1 Convolutional Encoder and Trellis

A convolutional encoder can be viewed as a *finite-state machine* that consists of an M-stage shift register with prescribed connections to n modulo-2 adders. A convolutional encoder is uniquely defined if the following parameters are specified:

1. Constraint length K, which is equal to $M + 1$ by definition,
2. Code rate r, which is the ratio between number of information bits to be encoded and number of output bits from encoder.

3. Generator polynomial, which is equivalent to unit-delay transform of the impulse response. The generator polynomial is generally written in octal form.

The following facts can be deduced from the parameters given above:

1. The encoder has $S = 2^M$ states.
2. The encoder needs M tail bits to return to all-zero state.

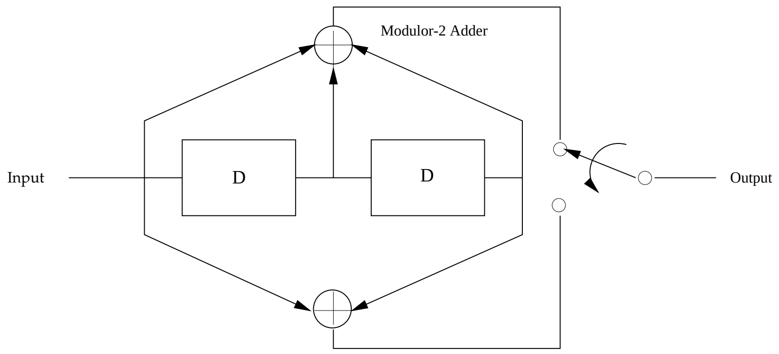


Figure 3.1: A simple convolutional encoder

Figure 3.1 shows a simple convolutional encoder with $n = 2$ and $M = 2$. Hence the encoder has constraint length $K = 3$, code rate $r = 1/2$ and generator polynomial of [7,5].

3.2 NSC and RSC Encoders

Convolutional encoder can be further classified into *Non-Systematic Convolutional* (NSC) encoder and *Systematic Convolutional* (NC) encoder by the way of using the information bits. A convolutional encoder is a SC encoder if the information bits are directly used as output bits. Otherwise, it is a NSC encoder. In family of SC encoder, there is a sub-class encoder called *Recursive Systematic Convolutional* (RSC) encoder. The recursiveness of the RSC encoder comes from the fact that the state of the encoder are fed-back and combined with the information bits by modulo-2 adder. The resultant bits are the input to be shifted into the M-stage shift register during the encoding process. Figure 3.2 depicts a [7,5] RSC encoder.

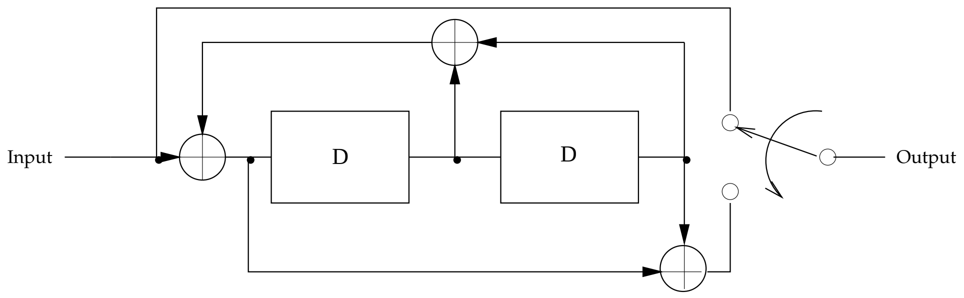


Figure 3.2: A simple RSC encoder

RSC encoder is chosen as the basic building block of turbo code due to the following reasons:

1. RSC code outperforms classical NSC at high code rate [1].
2. RSC decoder extracts the extrinsic information which can be used by another decoder in the next decoding stage in a turbo decoder. This will be further clarified in Chapter 4.

Therefore, effort will be focused on RSC encoder in this project.

3.3 Trellis and State Diagram

Three graphical forms can be used to summarize the structure properties of a convolutional encoder: *code tree*, *trellis* and *state diagram*. Among the three techniques, trellis is of great importance because of its effectiveness, when describing a convolutional encoder.

Figure 3.3 is a trellis diagram for the RSC encoder in Figure 3.2. The dots in Figure 3.3 represent the different states of the encoder. Time progresses from left to right along the x-axis. Most of the decoding algorithms used by convolutional decoder require that the encoder starts from a known state and ends at a known state. For simplicity, both known states are chosen to be all-zero state. Therefore, it is obvious that the trellis contains $L + K$ levels, where L is the length of the information bit sequence and K is the constraint length ($K = 3$ in Figure 3.3). Each level corresponds to a time instant. If a sequence of data bits is applied to a convolutional encoder, the

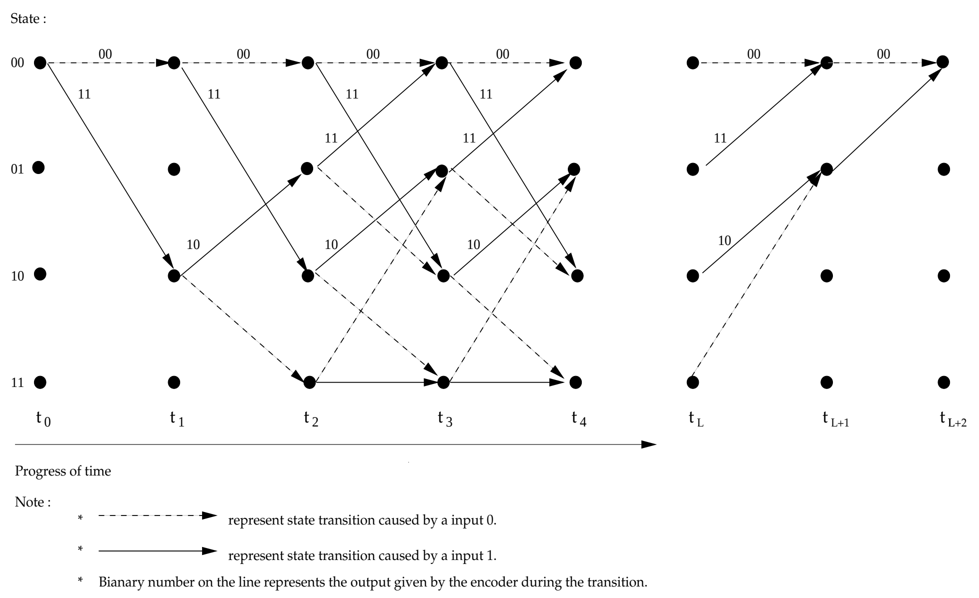
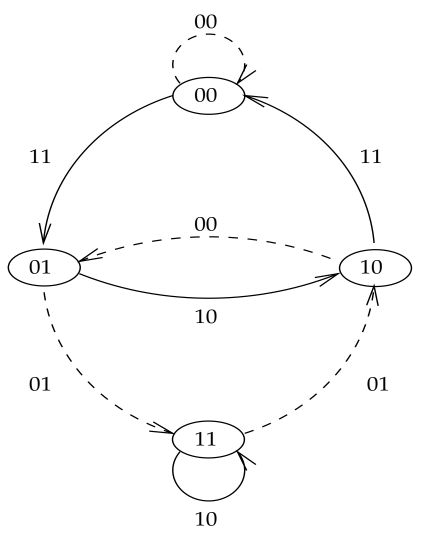

State :
00
01
10
11
00
11
10
t0
t1
t2
t3
t4
tL
tL+1
tL+2
Progress of time
Note :
* represent state transition caused by a input 0.
* represent state transition caused by a input 1.
* Bianary number on the line represents the output given by the encoder during the transition.

Figure 3.3: A trellis diagram of RSC encoder used in Figure 3.2

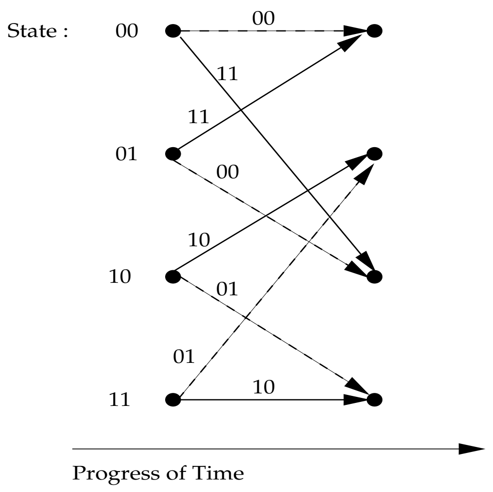
state of the encoder will make a trip along the valid paths in the trellis. The encoder finishes departuring from the initial zero state in the first $K - 1$ levels. It returns to end state in the last $K - 1$ levels. It is clearly shown that encoder cannot enter all states in the starting and ending portion of the trellis. In the center portion, all states are reachable. Figure 3.4 shows the state diagram of RSC encoder in Figure 3.2. It is clear that state diagram is equivalent to a repeating portion of the trellis diagram. However, trellis diagram provides more information about the state of the encoder changing with time for given input sequence. Hence, trellis diagram is used in various decoding algorithms, such as Viterbi Algorithm, to give intuitive understanding.

To further clarify the operation of a RSC encoder, suppose that a sequence of data [0,1,0,1,0,0] is applied to the RSC encoder in Figure 3.2. The state of the RSC encoder will follow the bold line in the trellis diagram in Figure 3.5 and output sequence [00,11,01,10,01,11] is produced.

After passing through a noisy channel, output bit sequence will be corrupted by random noise. Using the trellis diagram, the most intuitive decoding method coming into the mind is to find the most probable path among all possible paths based on the noisy observation. Viterbi algorithm exactly follows this doctrine. However, Viterbi algorithm cannot give *soft* description about every decoded bit. An algorithm is said



(a) State diagram.



(b) Portion of trellis diagram.

Figure 3.4: State diagram and repeating portion of the trellis diagram of the RSC encoder shown in Figure 3.2

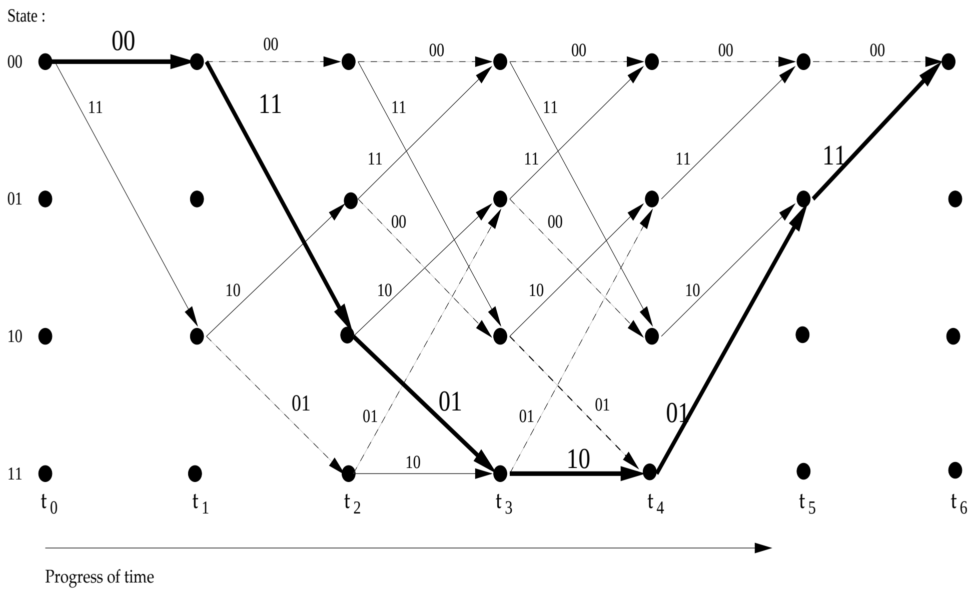


Figure 3.5: An example of input $[0, 1, 0, 1, 0, 0]$ applying to the RSC encoder shown in Figure 3.2

to be soft decoding algorithm, when it provides a *measure of confidence* along with the decision based on the test statistic being above or below a threshold level. In another words, soft decoding algorithm gives not only the final decision but also the estimation about the probability of one particular bit being 0 (or 1) in the decoded sequence.

Hence, we switch to the BCJR algorithm.

3.4 MAP Decoder Based on BCJR Algorithm

BCJR algorithm was proposed as early as in the seventies. However, BCJR algorithm was neglected because it outperforms Viterbi algorithm by only a small margin, yet it presents a much higher complexity and needs a great amount of computation power. Only until the report of turbo code by Berrou *et al.* [1] in 1993 did BCJR algorithm regain the attention. BCJR algorithm can produce the soft output or *A Posteriori Probabilities* (APP) of each bit in a decoded sequence. This soft output can provide the extrinsic information that is exchanged between the MAP component decoders to help each other to combat the noise interference in the turbo decoder.

3.4.1 Working Principle

BCJR algorithm was derived by Bahl *et al.* [2]. A simplified BCJR algorithm was proposed by Pietrobon in [3]. In this chapter, only the results from the two papers will be detailed. The MAP decoder based on BCJR algorithm[1] is basically a *soft-in/soft-out* decoder.

For a discrete memoryless AWGN channel, BPSK modulation and a code with rate $r = 1/2$, the observation of channel output at MAP decoder input will be:

$$R_1^N = (R_1, R_2, ..., R_k, ..., R_N), \tag{3.1}$$

[1]this will be simply called MAP decoder in the remaining sections

where $R_k = (x_k, y_k)$ is the received data at time k,

$$\begin{aligned} x_k &= (2d_k - 1) + m_k, && (3.2) \\ y_k &= (2c_k - 1) + n_k. && (3.3) \end{aligned}$$

d_k is the systematic bit sequence and c_k is the parity bit sequence from RSC encoder. m_k and n_k are two independent normally distributed random noise with variance, $\sigma^2 = N_0/2$, where N_0 is the single sided noise power spectral density.

Based on the observation R_1^N and the channel information σ, the MAP decoder produces the *Logarithm of Likelihood Ratio* (LLR):

$$L(d_k) = \ln \left\{ \frac{Pr(d_k = 1|R_1^N)}{Pr(d_k = 0|R_1^N)} \right\} \tag{3.4}$$

where $Pr(d_k = i|R_1^N)$ $(i = 0, 1)$, is the *a posteriori* probability of the information bit d_k. Figure 3.6 shows schematically a MAP decoder as a soft-in/soft-out device. MAP decoder can be viewed as a soft-in/soft-out device, which needs input $(R_1^N, L'(d_k))$ and produces output $L(d_k)$. With the input $L'(d_k)$ as *a prior* probability, MAP decoder works as if it had some knowledge on the probability of every bit being 0 (or 1). If the prediction given by $L'(d_k)$ is correct, MAP decoder will definitely yield better results in term of lower bit error rate. However, $L'(d_k)$ generally comes from another MAP decoder and it may have wrong bias, this can be tackled by iterative decoding method in a turbo decoder as discussed in Chapter 4. When a MAP decoder works alone, it is only a soft-out device. This is because the information source is normally assumed to have such characteristic that $Pr(d_k = 1) = Pr(d_k = 0) = 0.5$. In another words, at any instant of time, it is equi-probable for the information source to emit a bit 1 or bit 0. Therefore, $L'(d_k)$ for a work-alone MAP decoder is initialized to be zero.

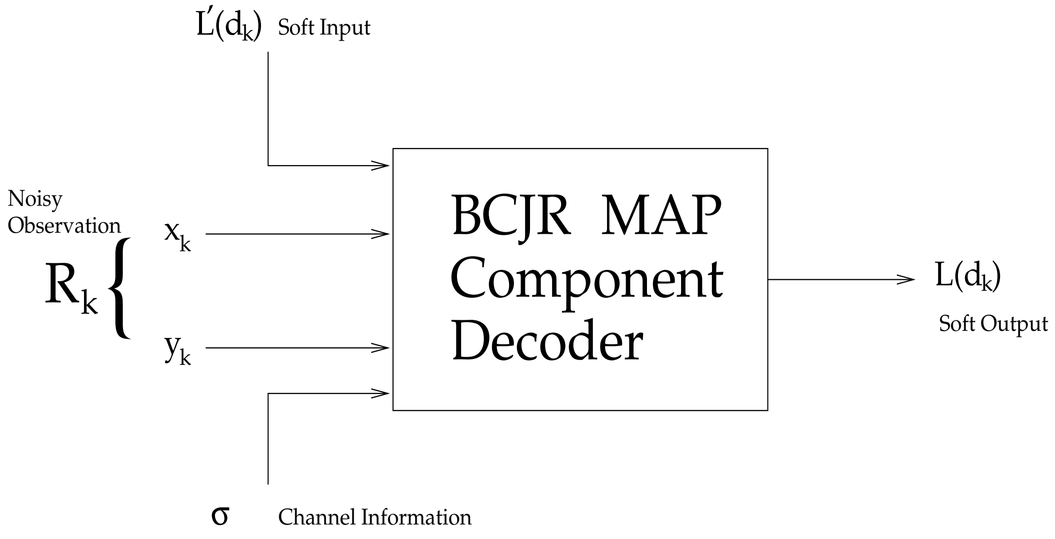


Figure 3.6: Schematic diagram for a soft-in/soft-out BCJR MAP component decoder

3.4.2 α, β, γ and Their Simplification

It is shown in [2] that LLR produced by BCJR MAP decoder can be expressed by:

$$L(d_k) = \ln\left\{\frac{\sum_{m=0}^{2^M-1}[\alpha_k^1(m)\beta_k^1(m)]}{\sum_{m=0}^{2^M-1}[\alpha_k^0(m)\beta_k^0(m)]}\right\}. \tag{3.5}$$

$\alpha_k^i(m)$, $\beta_k^i(m)$ $(i=0,1)$ are two important parameter to MAP decoder and they can be calculated using recursive formulas given in [2]:

$$\alpha_k^i(m) = \sum_{m'}\sum_{j=0}^{1}\alpha_{k-1}^j(m')\gamma_{i,j}(R_k,m,m') \tag{3.6}$$

$$\beta_k^i(m) = \sum_{m'}\sum_{j=0}^{1}\beta_{k+1}^j(m')\gamma_{j,i}(R_{k+1},m',m), \tag{3.7}$$

where γ is called branch metric. γ is related to the properties of trellis structure, information source and noisy channel by:

$$\begin{aligned}\gamma_{i,j}(R_k,m,m') &= Pr(S_k=m|d_{k-1}=j,S_{k-1}=m') && (3.8)\\ &\times\; Pr(d_k=i) && (3.9)\\ &\times\; Pr(R_k|d_k=i,S_k=m) && (3.10)\end{aligned}$$

and

$$\begin{aligned}\gamma_{j,i}(R_{k+1}, m', m) &= Pr(S_{k+1} = m' | d_k = i, S_k = m) \\ &\times Pr(d_{k+1} = j) \\ &\times Pr(R_{k+1} | d_{k+1} = j, S_{k+1} = m').\end{aligned}$$

Notation follows the conventions: k is the time index, m is the state index, i and j are the input alphabet (0 or 1), the summations in Equation (3.6) and (3.7) are over all possible 2^M states. By investigating the expression of $\gamma_{i,j}(R_k, m, m')$, the simplification can be done based on the intuitive meanings of each term. This is first proposed in [3].

The first term (3.8) gives:

$$Pr(S_k = m | d_{k-1} = j, S_{k-1} = m') = \begin{cases} 1, & \text{if } m' = S_b^j(m), \\ 0, & \text{otherwise,} \end{cases} \tag{3.11}$$

where $S_b^j(m)$ is the previous state given the path defined by $S_k = m$ and $d_{k-1} = j$. From Figure 3.4, it is clear that RSC encoder is a deterministic state machine. The next state of the encoder will be completely determined if previous state and input bit are specified. In another words, the probability of getting to S_k will be 1, if there is an transition between S_{k-1} and S_k given input as d_{k-1}.

The second term (3.9) reflects the fact that MAP decoder is a soft-in device. $L'(d_k)$ in Figure 3.6 directly contributes to the value of the term (3.9) of $\gamma_{i,j}(R_k, m, m')$. By re-arranging the definition of LLR in Section (3.4.1), the term (3.9) will become:

$$\begin{aligned} L'(d_k) &= \ln\left\{\frac{Pr(d_k = 1)}{Pr(d_k = 0)}\right\} \\ \Longrightarrow \frac{Pr(d_k = 1)}{Pr(d_k = 0)} &= \exp\{L'(d_k)\} \\ \Longrightarrow Pr(d_k = i) &= \frac{\exp\{i \cdot L'(d_k)\}}{1 + \exp\{L'(d_k)\}} \qquad i \in [0, 1] \end{aligned} \tag{3.12}$$

For the second step in above derivations to be true, $Pr(d_k = 0) + Pr(d_k = 1) = 1$ must hold, which is obvious. Actually, the calculations in Equation (3.12) are only necessary when MAP decoder is integrated into turbo decoder. In addition, there is a way to approximate complex operations in Equation (3.12), which will be discussed in Chapter 4. For a binary and equi-probable information source, $L'(d_k) \equiv 0$, Equation (3.12) is only:

$$Pr(d_k = i) = \frac{1}{2} \qquad i \in [0, 1] \tag{3.13}$$

Channel information σ and observation R_k will decide the value of the third term (3.10). Since the AWGN channel with mean zero and variance σ^2 is used, the term (3.10) can be expressed as:

$$\begin{aligned} Pr(R_k|d_k = i, S_k = m) = (\frac{1}{\sqrt{2\pi}\sigma})^2 & \exp\{-\frac{1}{2\sigma^2}[x_k - (2d_k - 1)]^2\}\mathrm{d}x_k \\ & \times \exp\{-\frac{1}{2\sigma^2}[y_k - (2c_k - 1)]^2\}\mathrm{d}y_k \end{aligned} \tag{3.14}$$

Take a close look at the exponent of Equation (3.14). Resolve and recombine the terms ($-\frac{1}{2\sigma^2}$ is ignored because of irrelevence):

$$\begin{aligned} [x_k - (2d_k - 1)]^2 + [y_k - (2c_k - 1)]^2 &= (2d_k - 1)^2 + (2c_k - 1)^2 && (3.15) \\ &+ x_k^2 + y_k^2 + 2(x_k + y_k) && (3.16) \\ &+ 4(x_k d_k + y_k c_k) && (3.17) \end{aligned}$$

Since BPSK modulation is used, $2d_k - 1$ and $2c_k - 1$ are either -1 or $+1$, the first term (3.15) of the above equation has a constant value of 2. The second term (3.16) is constant for the same time index k. Only the last term (3.17) is a function of the input bit d_k and the encoder state $S_k = m$. Based on these understanding, Equation (3.14) can be simplified to:

$$Pr(R_k|d_k = i, S_k = m) = K_k \exp(\frac{2}{\sigma^2}(x_k i + y_k c_k^i(m)), \tag{3.18}$$

where c_k is written as $c_k^i(m)$ to emphasize that c_k is a function of the d_k and $S_k = m$. Since K_k will not affect the value of LLR in Equation (3.4), a new parameter is defined as:

$$\delta_i(R_k, m) = \exp(\frac{2}{\sigma^2}(x_k i + y_k c_k^i(m)) \tag{3.19}$$

With results from Equation (3.11), (3.13) and (3.19), $\gamma_{i,j}(R_k, m, m')$ can be simplified to:

$$\gamma_{i,j}(R_k, m, m') = \begin{cases} \delta_i(R_k, m), & m' = S_b^j(m); \\ 0, & \text{otherwise.} \end{cases} \tag{3.20}$$

Since between most of the states (if K is large), there is no transition. The invalid terms in summation over all 2^M states in Equation (3.6) can be eliminated using $S_b^j(m)$. After manipulations, Equation (3.6) becomes:

$$\alpha_k^i(m) = \delta_i(R_k, m) \sum_{j=0}^{1} \alpha_{k-1}^j(S_b^j(m)). \tag{3.21}$$

A graphic representation of Equation (3.21) is shown in Figure 3.7 and the calculations for $\alpha_k^i(00)$ and $\alpha_k^i(01)$ for RSC encoder in Figure 3.2 is portrayed in Figure 3.8.

Branch matrix $\gamma_{j,i}(R_{k+1}, m', m)$ for $\beta_k^i(m)$ can be simplified using the similar procedure. However, $S_f^i(m)$ is used instead of $S_b^j(m)$. $S_f^i(m)$ is the next state given the path defined by $S_k = m$ and $d_k = i$. This is within expectation: Equation (3.6) used to calculate $\alpha_k^i(m)$ is forward-recursive in time domain, while Equation (3.7) of $\beta_k^i(m)$ is backward-recursive in nature. With the definition of $S_f^i(m)$, $\gamma_{j,i}(R_{k+1}, m', m)$ becomes:

$$\gamma_{j,i}(R_{k+1}, m', m) = \begin{cases} \delta_j(R_{k+1}, m'), & m' = S_f^j(m); \\ 0, & \text{otherwise.} \end{cases} \tag{3.22}$$

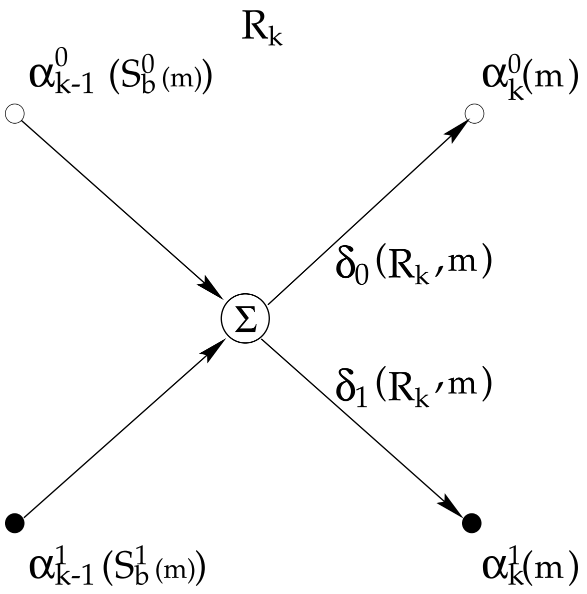


Figure 3.7: Graphic representation of Equation (3.21) for $\alpha^i_k(m)$

Therefore, with results in (3.22), Equation (3.7) becomes:

$$\beta^i_k(m) = \sum_{j=0}^{1} \beta^j_{k+1}(S^i_f(m))\delta_j(R_{k+1}, S^i_f(m)). \tag{3.23}$$

Similarly, a graphic representation of Equation (3.23) is shown in Figure 3.9 and the calculations for $\beta^i_k(01)$ and $\beta^i_k(11)$ for RSC encoder in Figure 3.2 is portrayed in Figure 3.10.

From the above simple recursive equations (3.21) for $\alpha^i_k(m)$ and (3.23) for $\beta^i_k(m)$, and Equation (3.19) for the branch matrix γ, a MAP decoder is ready to be implemented.

3.4.3 Implementation of BCJR Algorithm

With the definition of α, β and δ in Section 3.4.2, the sequence of decoding is as follows:

1. Starting from time $k = 1$, $\delta_i(R_k, m)$ can be calculated using Equation (3.19) for every received noisy output (x_k, y_k) from the AWGN channel. The value of

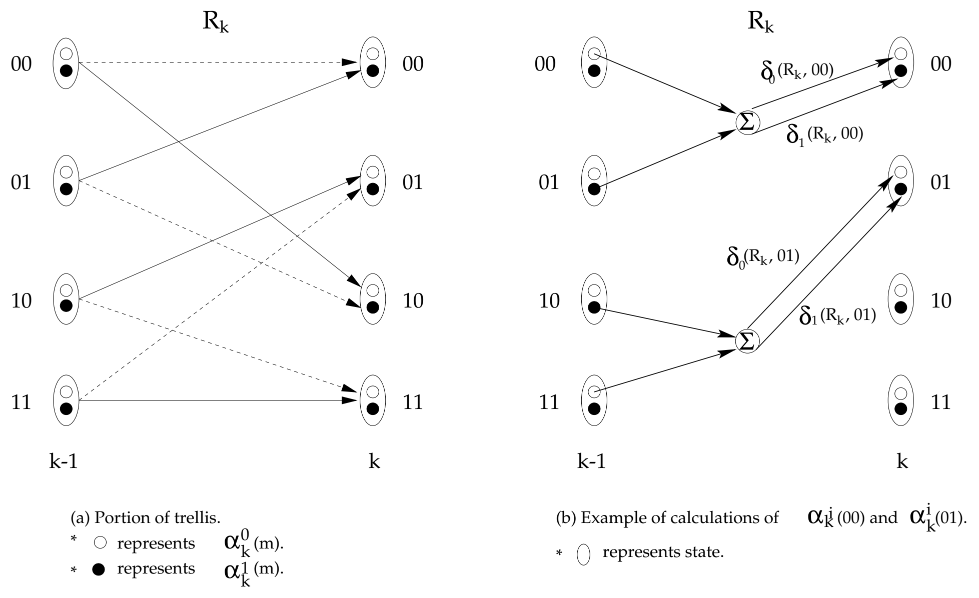


Figure 3.8: Calculation of $\alpha_k^i(00)$ and $\alpha_k^i(01)$ for RSC encoder in Figure 3.21

δ is stored in a *double*[2] array with size of $2 \times 2^M \times N$, where 2 is the number of possible output alphabet $[0, 1]$ from the information source, 2^M is the total number of states of the RSC encoder and N is the length of the noisy observation R_1^N.

2. Initialization should be done to α as follows:

$$\alpha_1^i(m) = \begin{cases} \delta_i(R_1, 0), & \text{if } m = 0; \\ 0, & \text{otherwise,} \end{cases} \tag{3.24}$$

for $i = 0, 1$. Using Equation (3.21), $\alpha_k^i(m)$ can be calculated forward-recursively from $k = 2$ to $k = N$. If for every time index k, $\alpha_{\max}$ and $\alpha_{\min}$ are found to be the maximum and minimum value of all $\alpha_k^i(m)$. At that instant of time, $\alpha_k^i(m)$ can be normalized by dividing every $\alpha_k^i(m)$ by factor $\overline{\alpha}$, where $\overline{\alpha} = \sqrt{\alpha_{\max}\alpha_{\min}}$. A *double* array with size of $2 \times 2^M \times N$ is needed for the storage of α.

[2]Standard data type in C detailed in Table 2.1.

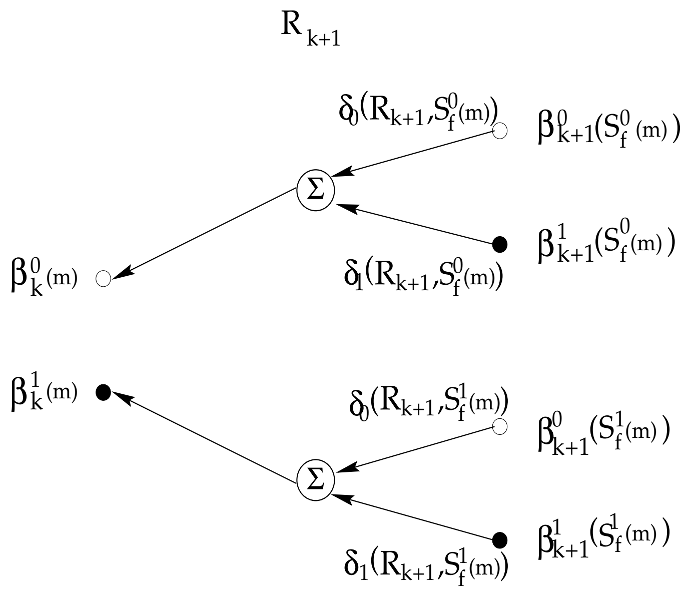

R_{k+1}
$\delta_0(R_{k+1}, S_f^0(m))$
$\beta_{k+1}^0(S_f^0(m))$
Σ
$\beta_{k+1}^1(S_f^0(m))$
$\delta_1(R_{k+1}, S_f^0(m))$
$\beta_k^0(m)$
$\beta_k^1(m)$
$\delta_0(R_{k+1}, S_f^1(m))$
$\beta_{k+1}^0(S_f^1(m))$
Σ
$\beta_{k+1}^1(S_f^1(m))$
$\delta_1(R_{k+1}, S_f^1(m))$

Figure 3.9: Graphic representation of Equation (3.23) for $\beta_k^i(m)$

3. Initialization to β is as follows:

$$\beta_N^i(m) = \begin{cases} 1, & \text{if } m = S_b^i(0); \\ 0, & \text{otherwise,} \end{cases} \tag{3.25}$$

for $i = 0, 1$. Using Equation (3.23), $\beta_k^i(m)$ can be calculated backward-recursively from k=$N-1$ to $k = 1$. With similar definition of $\beta_{\max}$ and $\beta_{\min}$, at time k, normalization to β is done by dividing every $\beta_k^i(m)$ by factor $\overline{\beta}$, where $\overline{\beta} = \sqrt{\beta_{\max}\beta_{\min}}$. an *double* array with size of $2 \times 2^M \times N$ is needed for the storage of β.

4. Finally, LLR $L(d_k)$ ($k \in [1, N]$) of the noisy input sequence R_k can be calculated by substituting $\alpha_k^i(m)$ and $\beta_k^i(m)$ into Equation (3.4). The MAP decoder can make hard-decision about the bit sequence from the information source by comparing the $L(d_k)$ with threshold value 0.

Table 3.1 summarizes the estimation of computation load needed by each variable. It is apparent that the total computation needed by BCJR algorithm grows exponentially with constraint length K and linearly with the data block length N. The major drawback of this algorithm is that recursive Equation (3.21) and (3.23)

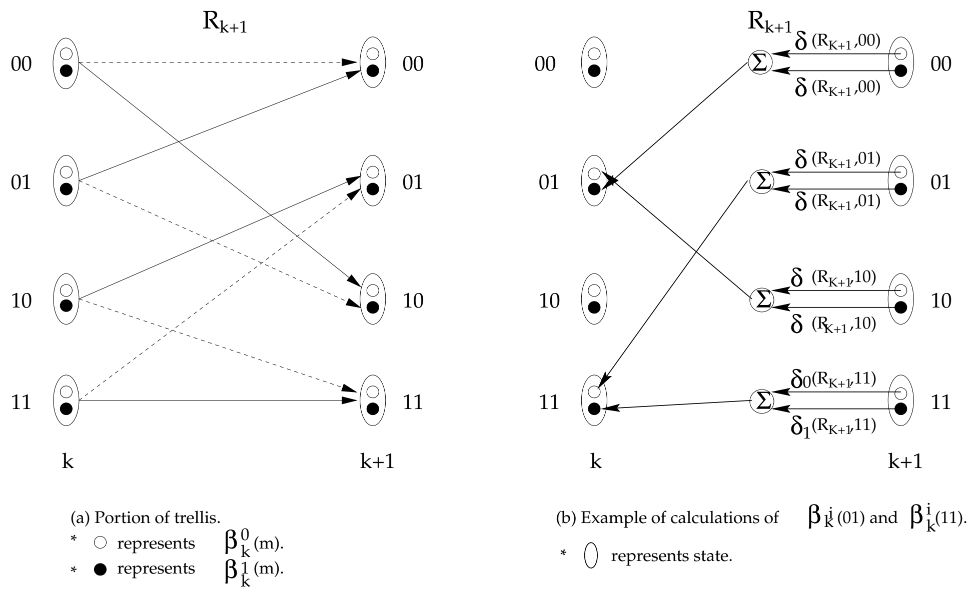


Figure 3.10: Calculation of $\beta_k^i(01)$ and $\beta_k^i(11)$ for RSC encoder in Figure 3.23

used to calculate $\alpha_k^i(m)$ and $\beta_k^i(m)$ are in multiplication form and Equation (3.19) for $\delta_k(R_k, m)$ is in exponential form. This creates complexity and difficulty in hardware implementation.

BCJR algorithm also has numerical representation problem. In Equation (3.19), a exponent with large magnitude can make δ very small. After integrating δ into the recursive formulas, the value of $\alpha_k^i(m)$ and $\beta_k^i(m)$ can become extremely small even for a short sequence of data. In the simulation we conducted for output sequence of several hundred bits from the AWGN channel with no prescribed normalization, α and β can easily go beyond what standard data type *double* ($\pm 1.8\mathrm{E} \pm 308$, 15-digit precision) in C can cope with. The overflow problem will be even worse at hardware implementation stage, because the number of bits used to represent floating number is very limited.

To overcome the disadvantages above, a sub-optimal alternative version of BCJR algorithm is investigated in the next section.

Variable	Addition	Multiplication	Exponential
$\delta_i(R_k, m)$	$2 \times 2^M \times N$	$3 \times 2 \times 2^M \times N$	$2 \times 2^M \times N$
$\alpha_i(R_k, m)$	$2 \times 2^M \times (N-1)$	$2 \times 2^M \times (N-1)$	0
$\beta_i(R_k, m)$	$2 \times 2^M \times (N-1)$	$2 \times 2^M \times (N-1)$	0
$L(d_k)$	$2 \times (2^M - 1) \times N$	$2 \times (2^M + 1) \times N$	N

Table 3.1: Estimation of Computation Load for BCJR Algorithm

3.5 MAP Decoder Based on BCJR Algorithm in Log Domain

3.5.1 BCJR Algorithm in Log Domain

The extremely intensive computation required by BCJR algorithm is due to multiplication recursive Equation (3.21), (3.23) and exponential operation in Equation (3.19). The logarithm function is used to convert the original BCJR algorithm into an additive form because of its monotonicity.

Prior to the conversion, several new definitions are given as :

$$\begin{aligned} A_k^i(m) &= \ln[\alpha_k^i(m)] & (3.26) \\ B_k^i(m) &= \ln[\beta_k^i(m)] & (3.27) \\ D_i(R_k, m) &= \ln[\delta_i(R_k, m)]. & (3.28) \end{aligned}$$

Knowing these definitions, take logarithm on both side of Equation (3.21), (3.23) and (3.19) and the following can be obtained :

$$\begin{aligned} A_k^i(m) &= D_k^i(m) + \biguplus_{j=0}^{1} A_{k-1}^j(S_b^j(m)) & (3.29) \\ B_k^i(m) &= \biguplus_{j=0}^{1} B_{k+1}^i(S_f^j(m)) + D_i^(R_k, m) & (3.30) \\ D_i(R_k, m) &= \frac{2}{\sigma^2}(x_k i + y_k c_k^i(m)). & (3.31) \end{aligned}$$

The conversion of original BCJR algorithm to log domain introduces a new operator

$\uplus$, which has the meaning of :

$$x \uplus y = \ln(e^x + e^y) \tag{3.32}$$

$$\begin{aligned} \biguplus_{i=1}^{N} f(i) &= f(0) \uplus f(1) \uplus \cdots \uplus f(N) \\ &= \ln\left\{\sum_{i=1}^{N} \exp(f(i))\right\}. \end{aligned} \tag{3.33}$$

3.5.2 Implementation of BCJR Algorithm in Log Domain

With the new definitions in Equation (3.29), (3.30) and (3.31), the decoding method can be implemented as follows:

1. Starting from time $k = 1$, $D_i(R_k, m)$ can be calculated using Equation (3.31) for every received noisy output (x_k, y_k) from the AWGN channel. The value of D is stored in a *float*[3] array with size of $2 \times 2^M \times N$.

2. Initialization should be done to A as follows:

$$A_1^i(m) = \begin{cases} D_i(R_1, 0), & \text{if } m = 0; \\ -\infty, & \text{otherwise,} \end{cases} \tag{3.34}$$

   for $i = 0, 1$. Using Equation (3.29), $A_k^i(m)$ can be calculated forward-recursively from $k = 2$ to $k = N$. At each time k, A is normalized by subtracting factor $\overline{A} = \frac{A_{\max}+A_{\min}}{2}$, where $A_{\max}$ and $A_{\min}$ are the maximum and minimum value among all $A_k^i(m)$ at that time instant. This is to prevent the problem of overflow. A *float* array with size of $2 \times 2^M \times N$ is needed for the storage of A.

3. Initialization to B is as follows:

$$B_N^i(m) = \begin{cases} 0, & \text{if } m = S_b^i(0); \\ -\infty, & \text{otherwise,} \end{cases} \tag{3.35}$$

[3]Standard data type in C detailed in Table 2.1.

Variable	Addition	Multiplication	$\uplus$
$D_i(R_k, m)$	$2 \times 2^M \times N$	$3 \times 2 \times 2^M \times N$	0
$A_i(R_k, m)$	$2 \times 2^M \times (N-1)$	0	$2 \times 2^M \times (N-1)$
$B_i(R_k, m)$	$2 \times 2^M \times (N-1)$	0	$2 \times 2^M \times (N-1)$
$L(d_k)$	$2 \times (2^{M+1} - 1) \times N$	$2 \times (2^M - 1) \times N$	$N \times 2 \times (2^M - 1)$

Table 3.2: Estimation of Computation Load for BCJR Algorithm in Log Domain

for $i = 0, 1$. Using Equation (3.30), $B_k^i(m)$ can be calculated backward-recursively from k=N−1 to $k = 1$. At each time k, normalization of B is done by subtracting every $B_k^i(m)$ by factor $\overline{B}$, which has similar definition with its counterpart $\overline{A}$. A *float* array with size of $2 \times 2^M \times N$ is needed for the storage of B.

4. LLR $L(d_k)$ ($k \in [1, N]$) of the noisy input sequence R_k can be calculated using $A_k^i(m)$, $B_k^i(m)$, Equation (3.4) and (3.33). Hard-decisions can be made to decode depending on the sign of $L(d_k)$.

It should be noted that data type used in original BCJR algorithm discussed in Section 3.4.3, which is *double*, has been converted to *float*. Based on simulation results, negative infinity ($-\infty$) in the initialization of A and B is set to be $-5,000.00$. A interesting point is acquired by us for the value of negative number used to represent $-\infty$. If negative number with very large magnitude is used, error will occur because data precision of *float* in C is only 7-digit. $-5,000.00$ is more than enough in practical situations, since $e^{-5000.00}$ has magnitude of 10^{-2170}.

By comparing Table 3.2 with 3.1, it is clear that the exponential and multiplication operations are minimized to a very small number. The new operation $\uplus$ is introduced and it is a composite of the undesired operations, exponential and logarithm. Reasonable approximation should be made for $\uplus$ in order to reduce the computation required.

3.5.3 Various Approximation for Operator $\uplus$

Take a close look at the operator $\uplus$:

$$\begin{aligned} x \uplus y &= \ln(e^x + e^y) \\ &= \ln[e^x(1 + e^{-|x-y|})], \quad \text{assume } x \geq y \\ &= \mathrm{Max}(x, y) + \ln(1 + e^{-|x-y|}). \end{aligned} \tag{3.36}$$

Plot out $\ln(1+e^{-z})$, where $z \geqslant 0$, in Figure 3.11. $\ln(1+e^{-z})$ is a fast-decaying function.

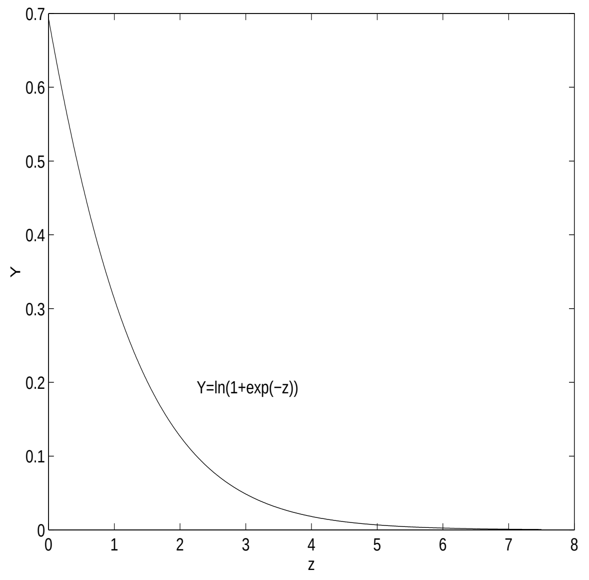


Figure 3.11: Plot of function $Y = \ln(1 + e^{-z}), z \geqslant 0$

The curve of the function at $z \in [0, 2]$ is almost a straight line and it quickly converges to z-axis at $z \in [2, 5]$. Knowing the characteristic of $\ln(1 + e^{-z}), z \geqslant 0$, the operator $\uplus$ can be approximated using several techniques.

Linear Approximation This simple technique is found in [4]. The idea is to use a straight line to approximate the complicated logarithm function. The approximation

Range	Round-up Value	Range	Round-up Value
$0 \leqslant z < 0.5$	0.693	$0.5 \leqslant z < 1$	0.474
$1 \leqslant z < 1.5$	0.313	$1.5 \leqslant z < 2$	0.201
$2 \leqslant z < 2.5$	0.127	$2.5 \leqslant z < 3$	0.079
$3 \leqslant z < 3.5$	0.049	$3.5 \leqslant z < 4$	0.030
$4 \leqslant z < 4.5$	0.018	$4.5 \leqslant z$	0.0

Table 3.3: Lookup table used for approximation of $\uplus$

is given by:

$$\ln(1+e^{-z}) \approx \begin{cases} -az+b, & 0 < z < b/a, \\ 0, & \text{otherwise} \end{cases} \tag{3.37}$$

where $b = \ln(2)$. The value for b is chosen from the fact that $y = \ln(1+e^{-z})$ intersects Y-axis at $[0, \ln(2)]$. Some simulation is needed for the value of a, which can be easily implemented. To save time, a has been chosen to be 0.3 as stated in [4]. With this method, operator $\uplus$ becomes:

$$x \uplus y \approx \begin{cases} \text{Max}(x,y) - 0.3|x-y| + \ln(2), & 0 < |x-y| < b/a, \\ \text{Max}(x,y), & \text{otherwise} \end{cases} \tag{3.38}$$

Lookup Table This method is to sample the continuous curve of $\ln(1+e^{-z})$ at several equally spaced points. If the value of z falls in certain interval, $\ln(1+e^{-z})$ will be rounded up to the prescribed value stored in the lookup table. If the curve for $\ln(1+e^{-z})$ is sampled uniformly with step of 0.5 at interval [0,5], a simple lookup table can be constructed, which is shown in Table 3.3.

Maximum approximation If $\ln(1+e^{-|x-y|})$ is totally ignored in Equation (3.36), the operator $\uplus$ is simply a Max. function

$$x \uplus y \approx \text{Max}(x,y).$$

The degradation of this approximation is of course the greatest. However, maximum approximation makes the implementation of BCJR MAP component decoder a rather easy task. Figure 3.12 summarizes the different approximation methods.

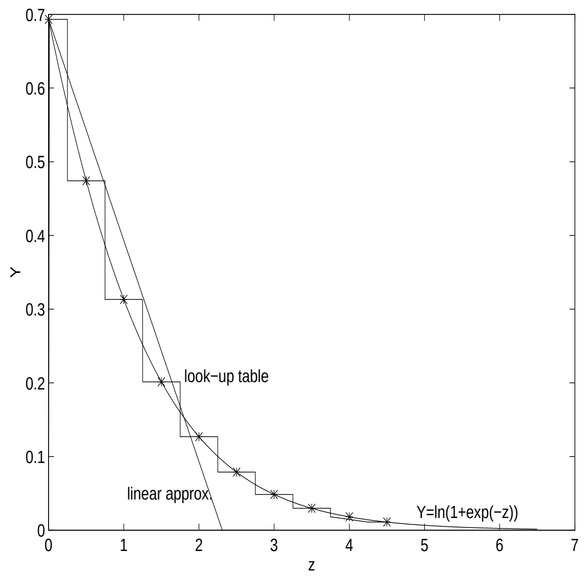


Figure 3.12: Summary plot for different approximation methods

3.5.4 the problem of Results and Analysis

To wrap up the discussion in above sections, we have a look at the simulation results. Here, several nomenclatures are used to ease the explanation. We will use xxx-MAPn to denote the different BCJR MAP component decoder. The number n denotes the constraint length and the prefix xxx represents the approximation method used for the $\uplus$ operator (Lin = linear approximation, Max = maximum approximation and Per = No approximation). The basic RS encoder used in this project is MAP5 with generator polynomial [37,21], which appears in the original paper published by Berrou *et al.* [1]. The data block length used in the simulation is $10,000$. At least 200 bit errors are accumulated for high E_b/N_0 (dB) in the range of [2.0,3.5], while at least 2,000 bit errors are accumulated for low E_b/N_0 (dB) in the range of [1.0,2.0].

Figure 3.13 summarizes the performance of each approximation method at low E_b/N_0. It is apparent that the BER degradation for the various approximation

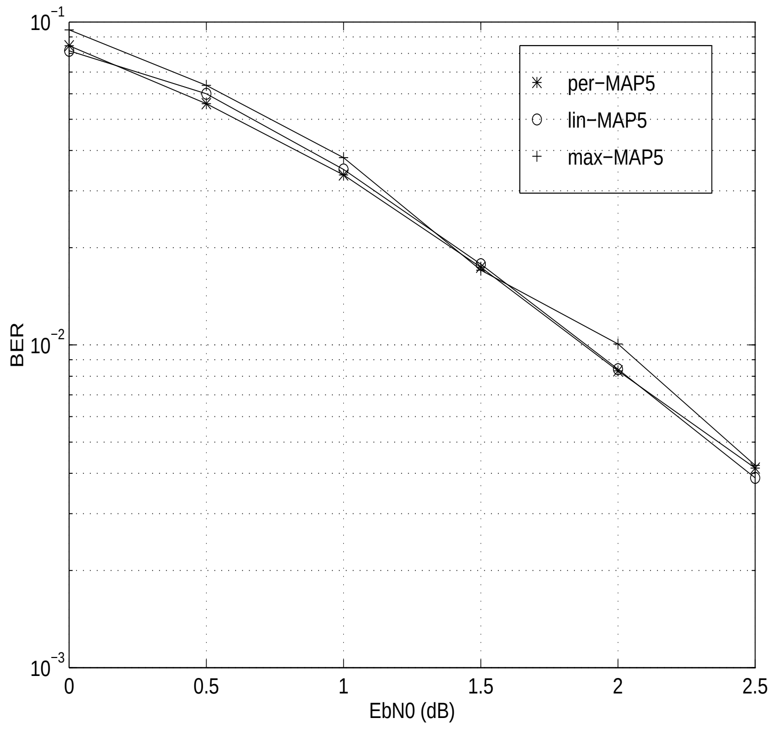


Figure 3.13: BER performance of different approximation method for MAP5 code with generator polynomial of [27,31]

method is quite insignificant. With no surprise, Max-MAP5 give the poorest performance among these three. Per-MAP5 is only superior to lin-MAP5 by only 0.1dB, which is also claimed in [4]. At high E_b/N_0 range, the difference of BER performance will be even trivia, because the correct decision differs more distinctly from the incorrect decision. In another words, $|x - y|$ in Equation (3.38) is more likely greater than b/a specified. The amount of computation saved by max-MAP5 and lin-MAP5 is tremendous. Max-MAP5 should claim its distinct advantage when come to hard-ware implementation. Most of operation done in max-MAP5 is addition and comparison, which is just what most micro-processors can handle effectively. In the later stage of the development, the MAP decoder implemented will use linear approximation.

Constraint length K is also a key parameter that influences the BER performance and the decoding delay introduced by a RSC decoder. Figure 3.14 depicts the BER perform from the three code: lin-MAP3, lin-MAP4 and lin-MAP5. The generator polynomial used are [5,7] for lin-MAP3 and [13, 15] for lin-MAP4. From Figure

3.14, lin-MAP5 ranks No.1 among the three codes. The coding gain provided by lin-MAP4 is comparable with that of lin-MAP5, while lin-MAP3 gives slightly worse performance. From Table 3.2, the computation load of lin-MAP5 is approximately 2 times of lin-MAP4 and 4 times of lin-MAP3. When integrating MAP component decoders into turbo decoder, compromise should be made between the BER performance and the implementation complexity.

Up to now, a soft-in/soft-out MAP decoder is ready for the implementation of a turbo decoder.

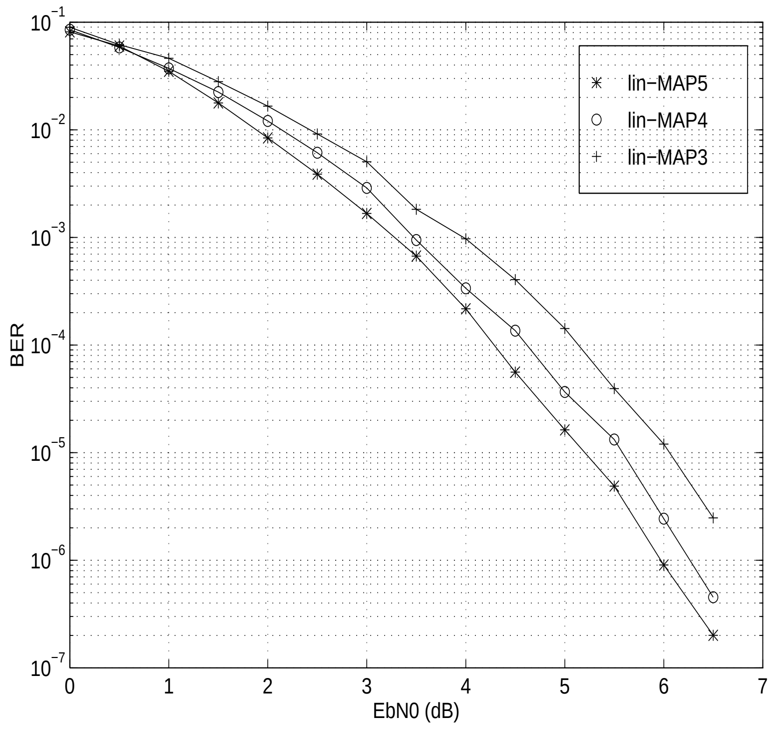


Figure 3.14: BER performance comparison between MAP decoders with different constraint lengths

Chapter 4

Turbo Decoder

The working principle of turbo decoding was first proposed by Berrou *et al.* in 1993[1]. Results near Shannon limit with moderate coding complexity were claimed, astonishing the whole coding community. However, some details required to reproduce the simulation results like how to deal with the soft output of a MAP decoder and feed it to the decoder at next decoding stage were omitted in [1]. To provide an insight into turbo code, this chapter analyzes some core concepts of turbo code, such as extrinsic information, non-uniform interleaver, iterative decoding, etc. Simulation results with same conditions as those in [1] are also presented, confirming the near Shannon limit performance claimed.

4.1 Turbo Encoder

Turbo codes make use of a pair of *parallel concatenated recursive systematic convolutional* (RSC) encoders interconnected by an interleaver as shown in Figure 4.1.

If the encoder is not recursive, certain pattern of low-weight codewords fed to the first encoder will always appear again in the second one regardless the choice of interleaver. This results in pairing low-weight codewords from one encoder with low-weight codewords from the other one, which is undesirable [14]. This is probably one of the reasons for choosing recursive encoder.

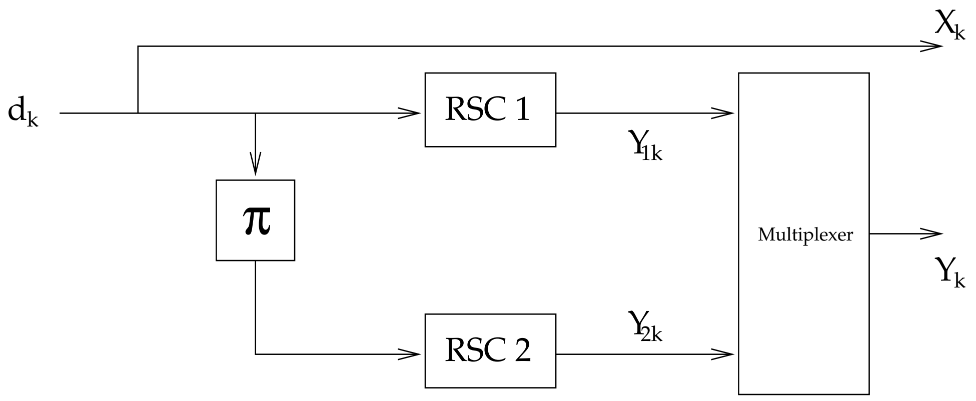


Figure 4.1: A typical turbo encoder

Because of the BCJR algorithm applied and the application of interleaver, the encoding is frame oriented. RSC1 operates on the data sequence d_k directly, where k is the time index, producing a sequence of parity information denoted by Y_{1k}. The second encoder RSC2, which may or may not be identical to RSC1, takes the interleaved version of d_k as input. Its parity sequence Y_{2k} together with Y_{1k} from the first encoder and the uncoded data bit d_k which is also termed as systematic bit and denoted by X_k are transmitted for turbo codes with code rate 1/3. To achieve higher code rate of 1/2, the two parity bits are multiplexed according to the following rule

$$Y_k = \begin{cases} Y_{1k}, & k \text{ is even} \\ Y_{2k}, & k \text{ is odd} \end{cases} \tag{4.1}$$

The sequences X_k and Y_k are then modulated using *Binary Phase Shift Keying* (BPSK) and transmitted through the channel where additive white Gaussian noise is superimposed on the signal. We denote the noisy sequences by x_k and y_k respectively.

4.2 Turbo Decoding

4.2.1 Turbo Decoder Structure

The structure of a turbo decoder consists of two *series* MAP component decoders interconnected by interleavers and de-interleavers. These two component decoders are not necessary to be identical, depending on the RSC encoders employed. The configuration of a typical turbo decoder is shown in Figure 4.2. For the sake of simplicity, different delays introduced by the two MAP component decoders, interleaver and de-interleaver are not shown in this figure.

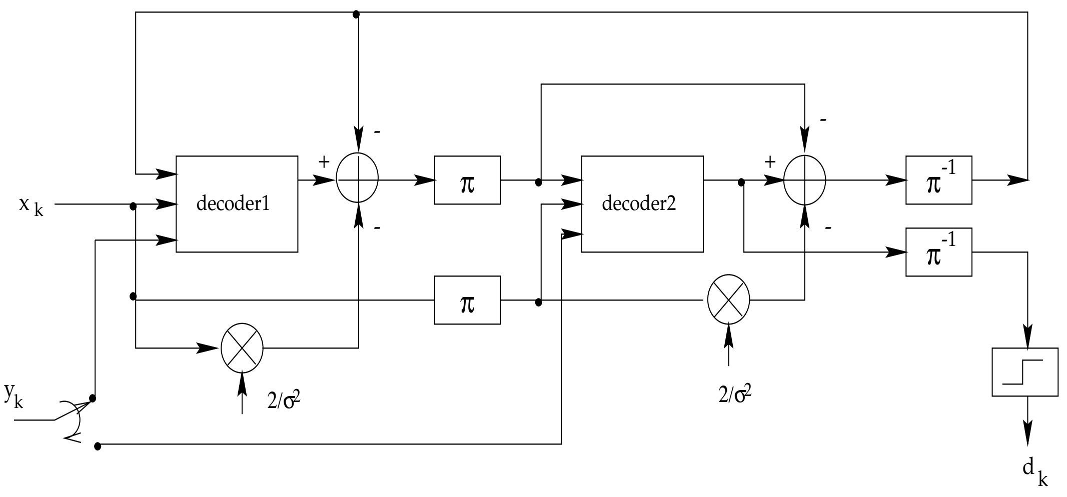


Figure 4.2: A typical turbo decoder

From Figure 3.6, each MAP component decoder has 4 inputs, namely, the systematic bit, the parity bit, the *a priori* information about the data bit and the noise variance of the channel. Unless otherwise stated, the decoder is assumed to have perfect knowledge on the channel noise, hence the fourth input (noise variance) is hidden in Figure 4.2.

The systematic bit sequence to the second component decoder is simply the interleaved version of the noisy systematic bit to the first one. As discussed in Section 4.1, the parity bit sequence is punctured to achieve code rate higher than 1/3. Two sequences can be reconstructed by demultiplexing the punctured sequence and filling the empty elements with zeros.

$$\begin{aligned} y_{1k} &= \{0, y2, 0, y4, ...\} \\ y_{2k} &= \{y1, 0, y3, 0, ...\} \end{aligned} \tag{4.2}$$

From Equation (3.19), by setting the empty elements to zero, that particular parity bit y_k has no contribution to the value of $\delta_i(R_k, m)$. This is true for the case of BPSK where logic one and logic zero are represented by "$+1$" and "-1" respectively. Furthermore, the channel noise is assumed to have zero mean. In cases other than BPSK, the empty elements should be set to value with equal distance to all possible modulated values. The systematic bit is now the only factor determining the transition probability, as shown in the following equation.

$$\delta_i(R_k, m) = \exp\left(\frac{2}{\sigma^2}(x_k i)\right) \tag{4.3}$$

One important feature of turbo code is that *part* of the soft output from one component decoder is used by the other one in subsequent stage to iteratively improve the error correcting performance. From the figure, it is clearly indicated that some kind of special treatment has to be performed on the *a posteriori* information before it can be fed to the other component decoder as *a priori* probability. Note that σ^2 in the figure is the channel noise variance. More details regarding the feed back loop will be given in the following sections.

The soft output from the second component decoder after certain number of iterations is de-interleaved and passed to a decision making unit. Data sequence consisting of $+1$ and -1 is then recovered, depending on the sign of the LLR. Of course, we can also make decision based on the soft output directly drawn from the first component decoder. The number of iterations for this case is $n + 1/2$, where n is an integer.

4.2.2 Extrinsic Information

The soft output of one MAP component decoder (LLR) at any time is composed of three components, namely, a weighted version of the noisy systematic bits, the *a priori* probability from the previous stage and some added information which is termed as extrinsic information. This idea is best illustrated by equations mathematically.

Substitute Equation (3.6, 3.7) into Equation (3.4), we have

$$L(d_k) = \ln \frac{\sum_{m=0}^{2^M-1}[\sum_{m'} \sum_{j=0}^{1} \alpha_{k-1}^{j}(m')\gamma_{1,j}(R_k, m, m')}{\sum_{m=0}^{2^M-1}[\sum_{m'} \sum_{j=0}^{1} \alpha_{k-1}^{j}(m')\gamma_{0,j}(R_k, m, m')} \times \frac{\sum_{m'} \sum_{j=0}^{1} \beta_{k+1}^{j}(m')\gamma_{j,1}(R_{k+1}, m', m)]}{\sum_{m'} \sum_{j=0}^{1} \beta_{k+1}^{j}(m')\gamma_{j,0}(R_{k+1}, m', m)]} \quad (4.4)$$

As demonstrated in Section 3.4.2, $\gamma_{i,j}(R_k, m, m')$ can be decomposed into three terms. The second term $Pr(d_k = i)$ is dependent on the *a priori* probability only, hence it can be factorized in both the numerator and the denominator and brought to the front. The third term can be rearranged as

$$\begin{aligned} Pr(R_k|d_k = i, S_k = m) &= Pr(x_k, y_k|d_k = i, S_k = m) \\ &= Pr(y_k|d_k = i, S_k = m) \times Pr(x_k|y_k, d_k = i, S_k = m) \\ &= Pr(y_k|d_k = i, S_k = m) \times Pr(x_k|d_k = i) \quad (4.5) \end{aligned}$$

The derivation above makes use of the fact that the systematic bit x_k is independent of the parity bit y_k or the state S_k. Then Equation (4.4) can be simplified as,

$$L(d_k) = \ln \frac{Pr(d_k = 1)}{Pr(d_k = 0)} + \ln \frac{Pr(x_k|d_k = 1)}{Pr(x_k|d_k = 0)} + W_k \quad (4.6)$$

where W_k is the so-called extrinsic information[1] which is the redundant information introduced by the component decoder. It is given by

$$\begin{aligned} W_k &= \ln \frac{\sum_{m=0}^{2^M-1}[\sum_{m'} \sum_{j=0}^{1} \alpha_{k-1}^{j}(m')Pr(S_k = m|d_{k-1} = j, S_{k-1} = m')}{\sum_{m=0}^{2^M-1}[\sum_{m'} \sum_{j=0}^{1} \alpha_{k-1}^{j}(m')Pr(S_k = m|d_{k-1} = j, S_{k-1} = m')} \\ &\times \frac{Pr(y_k|d_k = 1, S_k = m) \sum_{m'} \sum_{j=0}^{1} \beta_{k+1}^{j}(m')\gamma_{j,1}(R_{k+1}, m', m)]}{Pr(y_k|d_k = 0, S_k = m) \sum_{m'} \sum_{j=0}^{1} \beta_{k+1}^{j}(m')\gamma_{j,0}(R_{k+1}, m', m)]} \quad (4.7) \end{aligned}$$

It is noted that W_k is independent of the systematic bit at time k, x_k, instead, it is a function of parity bit y_k. It generally has the same sign as the data bit d_k, hence can be used to improve the error correcting capacity.

Assume AWGN channel, from the probability density function (pdf) of Gaussian distribution,

$$Pr(x_k|d_k = i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x_k - d_k)^2}{2\sigma^2}\right) dx_k \tag{4.8}$$

Substitute [4.8] into [4.6] and note that d_k which is the data bit without noise can be of value 1 or -1, simplifying the equation yields,

$$\begin{aligned} L(d_k) &= \ln \frac{Pr(d_k = 1)}{Pr(d_k = 0)} + \ln \frac{\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x_k-1)^2}{2\sigma^2}\right) dx_k}{\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x_k+1)^2}{2\sigma^2}\right) dx_k} + W_k \\ &= \ln \frac{Pr(d_k = 1)}{Pr(d_k = 0)} + \frac{2}{\sigma^2} x_k + W_k \end{aligned} \tag{4.9}$$

The extrinsic information is the *incremental* information through current decoding process. It can be taken as a channel input or *a priori* information by the other component decoder. The former case requires the estimation of variance[1]. In our project, it is used as *a priori* information.

4.2.3 Iterative Decoding

The important feature of turbo code is that *part* of the soft output of one component decoder is taken as the soft input by the other decoder to iteratively improve the error correcting capacity. Each component decoder has three inputs, the systematic bit, the de-multiplexed parity bit and the *a priori* probability generated by previous stage.

The same information should not be circulated during the iterative process, i.e. the *a posteriori* information generated by one decoder should not be passed to the same decoder as *a priori* information. We have to assure that the *a priori* information used by one decoder is independent of the other information used by that decoder. As the result, only the extrinsic information is fed to the next stage and taken as *a priori* probability. This is also the motivation of using LLR instead of probability, as it is more difficult to extract the extrinsic information when using probability. The extrinsic information, from Equation (4.9), can be derived as,

$$W_k = L(d_k) - \ln \frac{Pr(d_k = 1)}{Pr(d_k = 0)} - \frac{2}{\sigma^2} x_k \tag{4.10}$$

Initially, without any knowledge of the data bits, the probability of a data bit being 0 or 1 is reasonably assumed to be 0.5. After subsequent decoding process, this *a priori* information is replaced by W_k and continuously improved, yielding more accurate estimation.

The soft output of a MAP decoder is mainly dependent on the branch transition probability $\delta_i(R_k, m)$ which is a function of the channel noise and *a priori* information of data bits. Here we show the modifications to be made in the calculation of $\delta_i(R_k, m)$ in order to take the extrinsic information into account. The *a priori* information is no longer a constant of 0.5 which can be taken away without loss of generality for a single decoder, instead, it is now a variable with continuously improved accuracy. Equation (3.19) now becomes,

$$\delta_i(R_k, m) = Pr(d_k = i) \times \exp(\frac{2}{\sigma^2}(x_k i + y_k c_k^i(m)) \tag{4.11}$$

Converted to log domain,

$$D_i(R_k, m) = \frac{2}{\sigma^2}\left(x_k i + y_k c_k^i(m)\right) + \ln Pr(d_k = i) \tag{4.12}$$

The LLR value has to be converted to the logarithm of the probability to accomplish this modification. By the definition of LLR,

$$\begin{aligned} &\ln \frac{Pr(d_k = 1)}{Pr(d_k = 0)} = LLR_k \\ \Longrightarrow &\begin{cases} \ln Pr(d_k = 1) = -\ln[1 + \exp(-LLR_k)] \\ \ln Pr(d_k = 0) = -\ln[1 + \exp(LLR_k)] \end{cases} \end{aligned} \tag{4.13}$$

where LLR_k stands for *a posteriori* information for data bit d_k.

Exponential operations are performed in this equation. In order to prevent problem of *overflow* (*underflow*), and to accelerate the simulation process, linear approximation is employed instead of accurate calculation.

$$\ln Pr(d_k = 1) = \begin{cases} LLR_k, & LLR_k < -10 \\ 0, & LLR_k > 10 \\ -\ln[1 + \exp(-LLR_k)], & \text{otherwise} \end{cases} \tag{4.14}$$

It can be easily verified that the error introduced because of this approximation is smaller than 4.54×10^{-5} which is negligible. A plot of the exact value together with the approximate value is shown in Figure 4.3. Similarly,

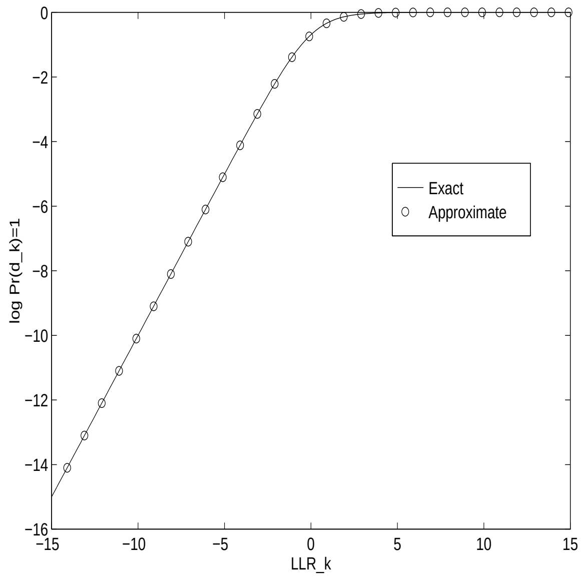


Figure 4.3: Exact value and approximate value for LLR conversion

$$\ln Pr(d_k = 0) = \begin{cases} 0, & LLR_k < -10 \\ -LLR_k, & LLR_k > 10 \\ -\ln[1 + \exp(LLR_k)], & \text{otherwise} \end{cases} \tag{4.15}$$

4.3 Interleaver

In digital communications, we often assume that the channel errors are completely independent from one signaling interval to the next, or in other word, the channel is memoryless. Unfortunately, it may not be the case in real life. To tackle this problem, interleavers (de-interleavers) are used to randomize the burst errors, breaking the correlation between symbols.

Interleavers are capable of re-indexing low-weight input sequence to produce codewords with higher weight. Hence they are used in turbo code *mainly* to increase the minimum distance of a codeword and to reduce the number of low-weight codewords [14]. Some sequences with patterns giving the shortest distance can be broken by interleaving process.

As demonstrated in Section 4.2.2, the extrinsic information generated by component decoder 2 is a function of the *a priori* probability which is dependent on the observation x_k, y_{1k}. In other words, the extrinsic information is correlated with the channel observation to the first component decoder and it is to be fed back to this same component decoder. Some papers argue that the *a priori* information should remain as independent on other inputs as possible. Hence it is important to weaken this correlation, an interleaver is required to be placed between the two component decoders to *partly* serve this purpose[1, 15].

The design of interleavers is still an open problem. There are two common types of interleavers, namely, the block interleaver and the nonuniform random interleaver.

4.3.1 Block Interleaver

Block interleaver[1] is the simplest type of interleaver. However, for short data frames with size less than 200, it can provide comparable performance with random interleaver. Its implementation rule is to write rowly (columnly) and read columnly (rowly), i.e. $b_{i,j} = a_{j,i}$, where $b_{i,j}$ is the interleaved version and $a_{j,i}$ is the original

[1]It will be referred to as *block* in next chapter.

sequence. It is noticed that the following relationship holds for block interleaver,

$$\pi_{m\times n}\left(\pi_{n\times m}\{a_k\}\right) = \{a_k\} \tag{4.16}$$

where $\pi_{m\times n}$ denote the interleaving process with block $m\times n$. Hence, block interleaver can be used as de-interleaver as well, by simply performing even number of interleaving processes to retain the original sequence. A simple example illustrating the working principle is shown in Figure 4.4.

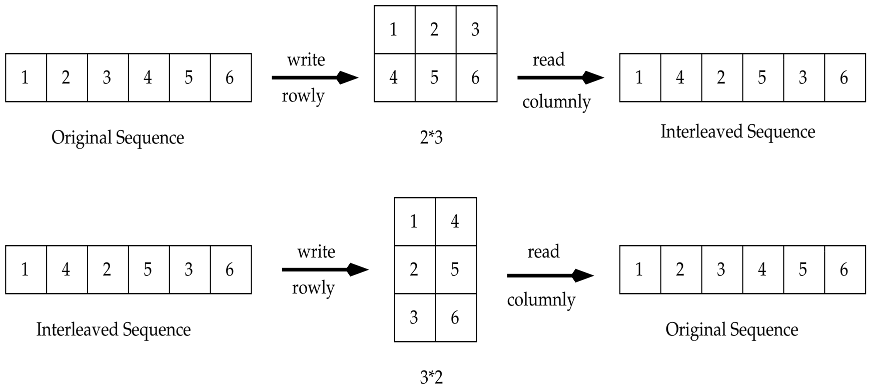


Figure 4.4: Working principle of block interleaver

4.3.2 Random Interleaver

The random interleaver employed in this project is the same as that given in [1][2]. In the interleaver, the input sequence is written row by row and read following the rule given below,

$$\begin{aligned} i_r &= (M/2+1)(i_w + j_w) \qquad \pmod M \\ \psi &= (i_w + j_w) \qquad \pmod 8 \\ j_r &= [P(\psi)\cdot(j_w+1)] - 1 \qquad \pmod M \end{aligned} \tag{4.17}$$

where i_r and j_r indicate the position of the element for reading, i_w and i_w indicate the position of the element for writing and $P(\cdot)$ is a number relatively prime to M

[2]It will be referred to as *Berrou_ran* in next chapter.

which is the size of the interleaver. They are given as follows,

$$\begin{aligned} P(0) = 17; \quad P(1) = 37; \quad P(2) = 19; \quad P(3) = 29; \\ P(4) = 41; \quad P(5) = 23; \quad P(6) = 13; \quad P(7) = 7. \end{aligned} \tag{4.18}$$

For random interleavers, Equation (4.16) no longer holds. As modular operations are involved in the interleaving process, it is necessary to find dividend from the divider and remainder in order to retain the original sequence from its interleaved version (de-interleave). Unfortunately, the relationship is not one to one. To solve this problem, a look up table holding the position relationship between the original sequence and its interleaved version is established during the first interleaving process. This look up table is then used in subsequent interleaving and de-interleaving processes. Memory required increases as a result. On the other hand, the amount of calculation is reduced.

However, the application of look up table is not the unique way to de-interleave a sequence. Although Equation (4.16) no longer holds for random interleaver, interleaving process still can be employed for purpose of de-interleaving. Suppose k is the position of an element in the original sequence and n is its position in the interleaved version. It is easy to obtain n from k by interleaving process. Let k takes values ranged from 1 to N, where N is the sequence length, then n should go through all the values in the same range. Doing the following assignments can retain the original sequence from its interleaved version without a look up table.

$$\begin{aligned} &\pi(k) = n \\ &a_k = b_n, \quad k = 1, 2, ..., N \end{aligned} \tag{4.19}$$

where a_k is the de-interleaved sequence and b_n is the interleaved version. The interleaving and de-interleaving of a 4×4 sequence is illustrated in the following figure,

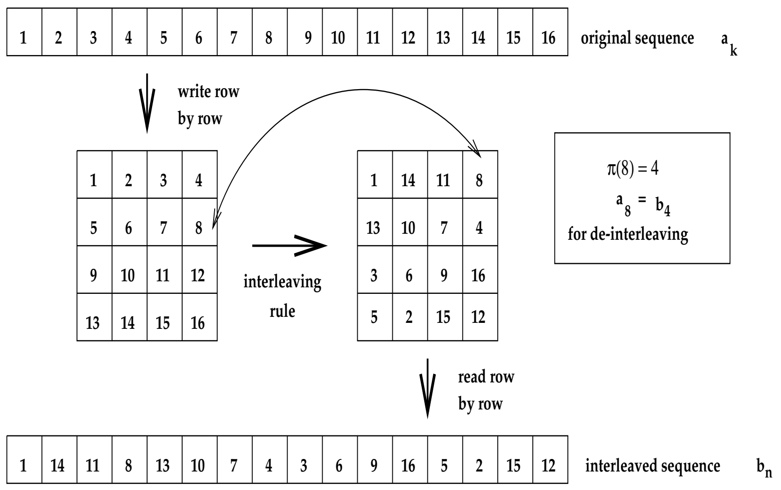


Figure 4.5: Working principle of random interleaver

4.4 Computation Complexity and Memory Requirement

This section studies the memory and computation effort required for turbo code. Actually, they are mainly predetermined by the requirement of the MAP component decoder as turbo decoder is simply two component decoders interconnected by interleaver and de-interleaver.

As depicted in Figure 4.2, each MAP component decoder has three sequences as input, namely, the systematic bit, the parity bit and the *a priori* probability and produces one sequence of soft output (*a posteriori* information). Hence total number of 8 arrays each of which has length of N, where N is the frame size, are required to store input and output data for two component decoders. However, this number can be reduced to 6 by overwriting the the existing *a priori* and *a posteriori* sequences when a new sequence is generated. In a component decoder using MAP algorithm, values for $A_k^i(m)$, $B_k^i(m)$ and $D_k^i(m)$ have to be kept for backward and forward recursive calculations. This leads to the total memory requirement for turbo decoder as given by

$$M_t = \left(6 \times N + 3 \times N \times 2^M \times 2\right) \times B \tag{4.20}$$

Stage	Addition	Multiplication	$\uplus$
Decoder	$10N2^M - 4\ 2^M - 2N$	$8N2^M - 2N$	$(6N-4)2^M - 2N$
Interleaver	$5 \times N$	$9 \times N$	0
De-interleaver	$5 \times N$	$9 \times N$	0
Extrinsic Infor	$4 \times (N-1)$	$6 \times N$	0
Total	$20N2^M - 8\ 2^M + 10N - 4$	$16N2^M + 20N$	$(12N-8)2^M - 4N$

Table 4.1: Estimation of computation load for turbo decoder

where M is the encoder memory and B is the number of bytes used to represent a floating point number in the computer. The memory required to hold other variables is insignificant as compared with M_t given above.

The computation effort required for a turbo decoder during *one* iteration is summarized and shown in Table 4.1.[3] Subtraction is considered as a special type of addition, similarly, division is taken as one form of multiplication. If a look up table is created during the interleaving process and used in subsequent de-interleaving process, no computation is required in the de-interleaver. Otherwise, it has same computation load as interleaver. Note that the computation load shown in the table does not include decision making and assignment operations.

4.5 Results and Analysis

4.5.1 BER Performance of Turbo Decoder

To evaluate the performance of the turbo decoder and to verify the correctness of the algorithm, computer simulation was conducted. Two identical encoders each of which has memory of 4 and generators $G1 = 37$, $G2 = 21$ were used in the simulation. The same random interleaver of size 256×256 was employed to provide a fair comparison, implying that the data frame was of size 65536. The parity bits from the two RSC encoders were punctured to have code rate of 1/2. With all conditions exactly the same as those given in [1], bit error rate versus E_b/N_o in dB was plotted as shown in

[3]The interleaver is the nonuniform interleaver given in [1]. The row *Extrinsic Infor* is the computation effort needed to extract the extrinsic information from the LLR output of one component decoder.

Figure 4.6. In the calculation of BER, at least 100 errors were accumulated for each point. The curve for uncode bit error rate was obtained using Equation (2.8).

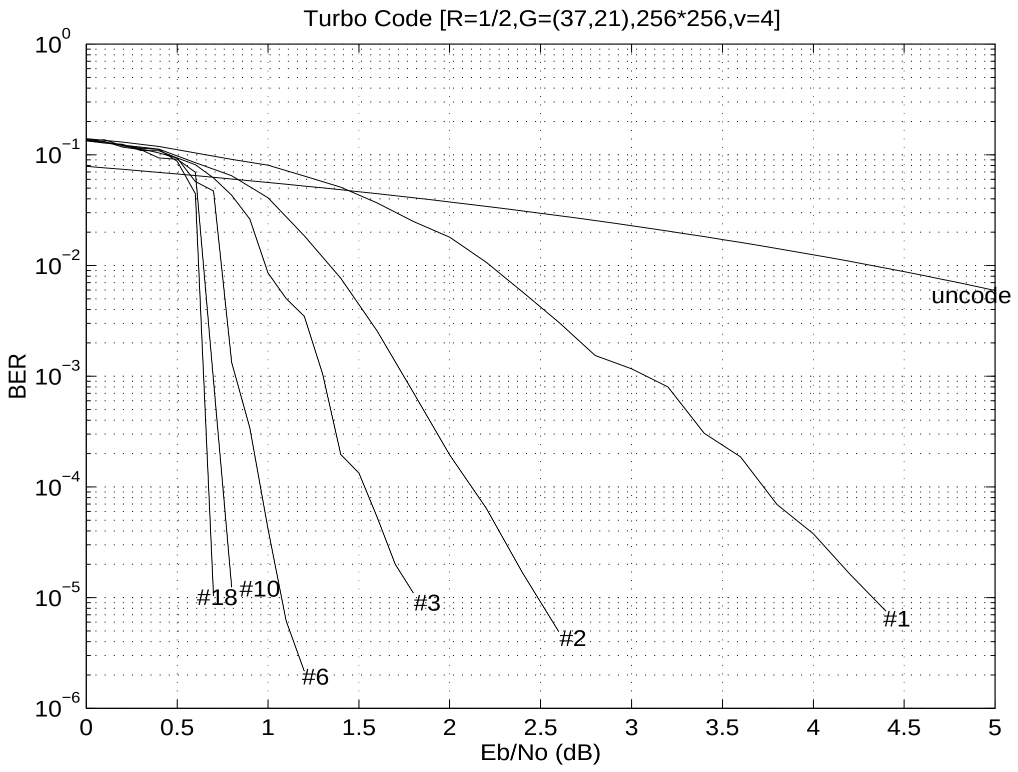


Figure 4.6: BER performance of a turbo decoder

From the figure, the simulation results attained are quite close to those given in [1]. The bit error rate decreases as the number of iterations increases. The improvement for the first three iterations is quite dramatic. If BER of 10^{-5} is chosen as reference point, the coding gain of 2 iterations over 1 iteration is about 1.9 dB. The coding improvement becomes less and less significant for large number of iteration steps (iteration > 6) as compared with its previous iteration step. As an example, the coding gain for iteration step 18 is just $0.1dB$ higher than that for iteration step 10. It is reasonable to stop the iterative process at 6 iterations as carrying out more iterations can not make much difference.

It is shown in the figure that the slope of the curve for iteration step 18 is extremely deep, implying that BER drops dramatically with minor increase in E_b/N_o. The bit error rate at 0.7 dB is reduced to about 10^{-5} which can be considered as error free in real life. The near Shannon limit error correcting capacity as claimed in [1] is confirmed.

In Berrou's paper, the BER is observed to increase for low E_b/N_o during the iterative processes sometimes. This effect is overcome by dividing the extrinsic information by a stability factor in his paper. However, this phenomenon is not observed from our simulation results without applying any correcting factor. As the extrinsic information is taken as *a priori* information instead of a channel input, it is not necessary to estimate the variance of the extrinsic information as stated in [1].

4.5.2 Study of Extrinsic Information from MAP Component Decoder

To study the characteristics of the extrinsic information from the MAP component decoder after certain number of iteration steps, a data sequence consisting of all 1's was transmitted and the distribution of extrinsic information (W_k) from the second component decoder was plotted.

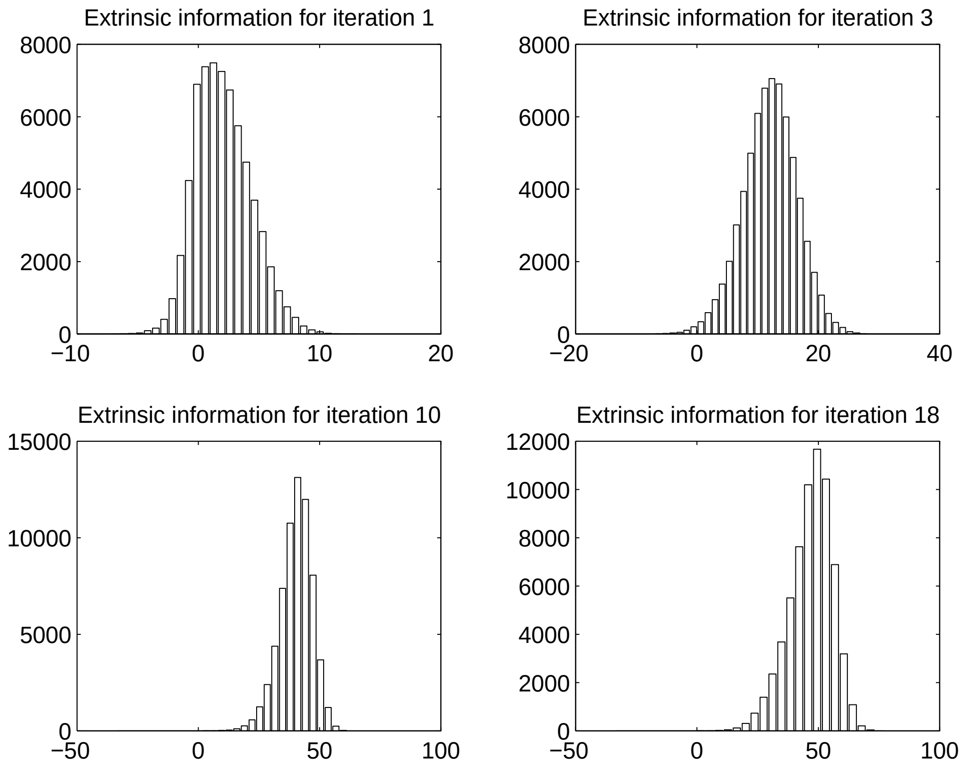


Figure 4.7: Distribution of extrinsic information for different number of iterations

We mentioned that the extrinsic information W_k in general has the same sign as the data bit and helps to improve the error correcting performance. However, from Figure 4.7, it is noticed that for iteration step 1 and 3, there is a small portion of the extrinsic information left negative (the left tail). The area under the curve to the left of the origin indicates the number of error bits if the iterative process is stopped at that step. As more iterative processes are performed, the distribution shifts to the right, eventually, the whole population is distributed on the right half plane (iteration 10 and iteration 18). The absolute value of extrinsic information indicates the reliability of the decoded bit sequence. At iteration step 10, the whole population is already on the right half plane , which implies that performing more iterations after 10 iterations can hardly improve the BER performance further. This is confirmed by the simulation results shown in Figure 4.6 from which we can see the improvement of iteration 18 over iteration 10 for bit error rate of 10^{-5} is only 0.1 dB. As a result, the characteristics of the extrinsic information can be used to determine when to stop the iteration process[6].

Chapter 5

Factors Affecting Performance of Turbo Code

In Chapter 4, we have confirmed the near Shannon limit performance of turbo code. The error correcting capacity of turbo code relies on the performance of its various components, namely, the *soft-in/soft-out* component decoder, the RSC encoder and the interleaver (de-interleaver) employed. This chapter investigates factors affecting the performance of turbo code and the feasibility of making trade-off between system performance and computation complexity.

5.1 Termination of RSC Encoders

As discussed in the previous chapter, the application of BCJR algorithm assumes that the trellis starts and terminates with a known state. The initial state is then used to initialize $A_k^i(m)$ for forward recursion, while the final state is for the initialization of $B_k^i(m)$ for backward recursion. A tail of length equal to the encoder memory has to be appended to the data sequence to force the trellis to a known state (conventionally zero state). For non-recursive code, the input bit will directly appear as one bit of the state and shift forwards, which implies that simply transmitting consecutive zeros with length equal to the memory can force the final state to zero. Because of the feedback loop employed in RSC code, the input bit required to turn the encoder to zero state is a function of the current state.

5.1.1 Various Termination Schemes

There are various termination schemes already proposed in the literature [5]. Termination with zero state can be easily achieved for the first RSC encoder. As the input to the second RSC encoder is the interleaved version of the systematic bits, the order of the sequence is permuted and this new sequence, in general, can not force the trellis to the known state at the same time. Due to the impossibility of terminating these two encoders at the same time, most of the methods proposed in the literature only terminate the first encoder, while leave the second one open or just leave both encoders open.

If an encoder starts and ends with zero state, the initialization for the forward and backward recursion will be[1],

$$A_1^i(0) = D_i(R_1, 0), \quad A_1^i(m \neq 0) = -\infty; \tag{5.1}$$

$$B_N^i(0) = 0, \quad B_N^i(m \neq 0) = -\infty; \tag{5.2}$$

where i is input bit which can take value as 0 or 1, N is the frame length and m is the encoder state.

For any encoder left open, the final state is uncertain, so relationship (5.2) no longer holds. There are two ways of initializing $B_N^i(m)$ in this case. It is reasonable to assume that the probability that the trellis terminates at a particular state is the same as any of the other state if the data bits are equal-probable. This assumption yields the following initialization,

$$\beta_N^i(m) = \frac{1}{2^M} \quad \text{for } m = 1, 2, ..., 2^M \tag{5.3}$$

where M is the encoder memory. Without loss of generality, the β values can be

[1]Note that the equations shown here is in log domain which means $-\infty$ corresponds to 0 and 0 is equivalent to 1 in absolute value.

Method	Dec1 Status	Equ. for Dec1	Dec2 Status	Equ. for Dec2
1	Closed	Equ[5.2]	Open	Equ[5.4]
2	Open	Equ[5.4]	Open	Equ[5.4]
3	Closed	Equ[5.2]	Open	Equ[5.5]
4	Open	Equ[5.5]	Open	Equ[5.5]

Table 5.1: Various termination schemes

normalized to 1 and converted to log domain,

$$B_N^i(m) = 0 \quad \text{for } m = 1, 2, ..., 2^M \tag{5.4}$$

Alternatively, $B_N^i(m)$ can initialized as,

$$B_N^i(m) = A_N^i(m) \quad \text{for } m = 1, 2, ..., 2^M \tag{5.5}$$

5.1.2 Comparison of Simulation Results for Various Termination Schemes

An encoder can be either open or closed. In the case it is open, there are two ways of initializing the value of $B_N^i(m)$, the different combinations provide many possible termination schemes. We just analyze some of them summarized in Table 5.1.

Figure 5.1 shows the BER performance for different termination schemes listed in the table. Three iterations were performed and the tail bits used to terminate RSC encoders were taken into account in the calculation of E_b/N_o. [5] obtained simulation results for turbo code with different termination schemes. Short sequences with block size $N = 48$ were transmitted. The results showed that method 2 gave best performance in term of BER for low to moderate E_b/N_o, whereas method 4 is the second best. However, our simulation results with large frame size of 65536 show that various termination schemes are comparable. Method 3 is the best, however, its gain over other schemes is very minor. It seems that performance of different schemes depends on system parameters like block size, generator polynominal, etc. In general, the results are quite close, hence the selection of termination schemes does not matter much.

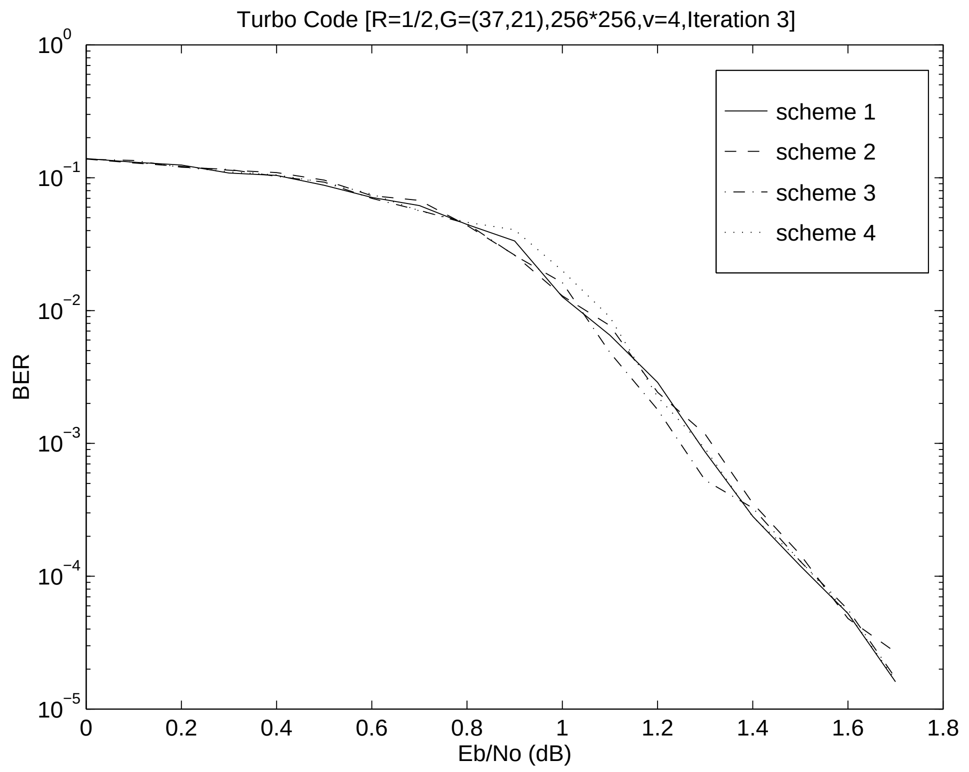


Figure 5.1: BER performance for turbo decoder with various termination schemes

5.2 Effect of Encoder Constraint Length

For the simulation results obtained previously, both encoders employed for constructing the turbo code have memory of 4. It is known that the computation complexity and memory required in turbo code is proportional to 2^{K-1}, where K is the encoder constraint length. In this section, we study the performance degradation due to smaller encoder memory so as to investigate the feasibility of sacrificing performance within tolerable extent for less computation effort.

Instead of using generator polynominal $[37, 21]$, the encoder memory is now reduced to 2 with generator $[7, 5]$. A hybrid case where the first encoder has memory 4 with generator $[37, 21]$ as before, while the second one has memory 2 with generator $[7, 5]$ is also under investigation. Besides the encoder memory, all other conditions remain the same as those in [1]. The simulation results are shown in Figure 5.2. In the figure, $[M_1, M_2, \#p]$ indicates that the first encoder is of memory M_1, whereas M_2 is the memory of the second one and p is the number of iterations performed.

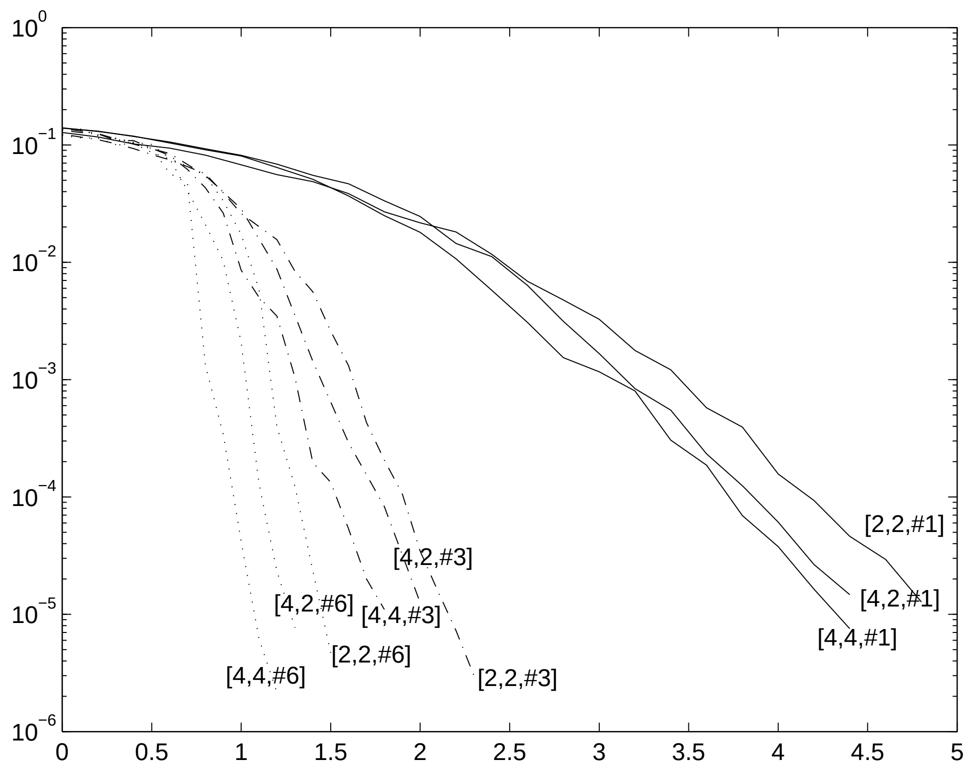


Figure 5.2: BER performance for turbo decoder with different constraint lengths

From the simulation results obtained, with no surprise, the BER performance degrades as the constraint length decreases. However, it is noted that the degradation is quite insignificant. As an example, the performance improvement of $[4, 2, \#3]$ over $[2, 2, \#3]$ is only about 0.15 dB and it is similar for $[4, 4, \#3]$ over $[4, 2, \#3]$. This performance degradation due to the smaller constraint length can be compensated by using more iterations. For example, the performance of $[2, 2, \#6]$ is superior over $[4, 4, \#3]$ by about 0.4 dB. As discussed in previous chapters, the computation effort grows proportionally with the number of iterations performed and exponentially with the encoder constraint length. Thus the amount of computation required for $[4, 4, \#3]$ is about $\frac{2^4\times 3}{2^2\times 6} = 2$ times that for $[2, 2, \#6]$. We can define the turbo decoder complexity C as,

$$C_{[M_1,M_2,\#p]} = (2^{M_1} + 2^{M_2}) \times p \tag{5.6}$$

Note that C is just an approximate indicator for the system complexity, as the real computation load and memory required are not exactly proportional to C given above.

With same complexity C, both encoders with memory 2 is shown to have the best performance. Therefore, turbo decoding with less memory but more iterations is recommended.

5.3 Effect of Approximation Method

In Section 3.5.4, we have compared the performance of a MAP component decoder using linear approximation (lin-MAP5) and maximum approximation (max-MAP5). Here we compare their performance in turbo code. Besides the approximation methods employed, other conditions are kept the same.

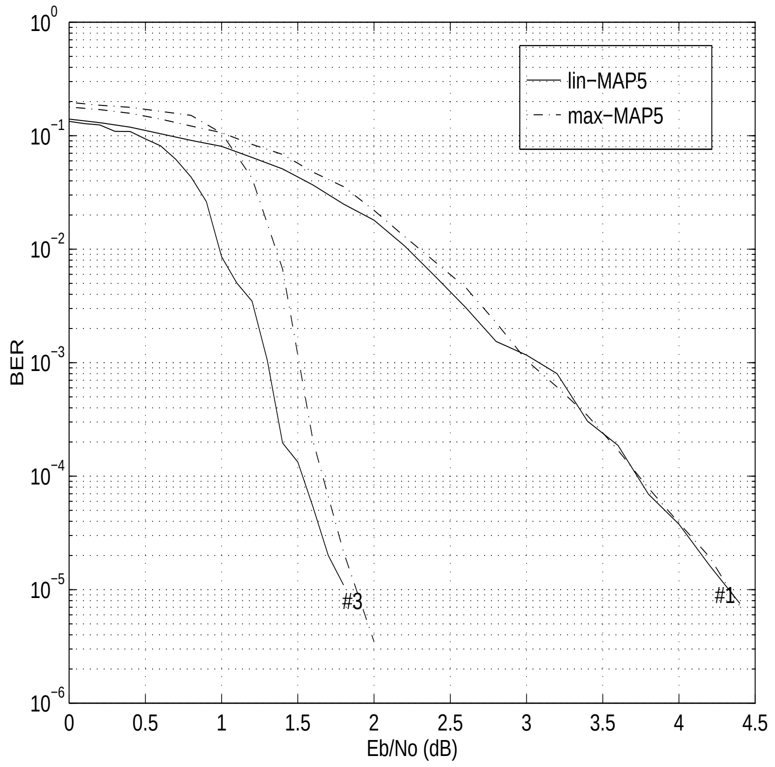


Figure 5.3: BER performance for turbo decoder with different approximation methods to $\uplus$

With no surprise, lin-MAP5 has superior performance over max-MAP5 and the improvement is more significant within the low E_b/N_o range. However, as E_b/N_o increases, the difference between the two methods vanishes. The two curves converge when the BER falls below 1×10^{-5}. It is expected that the operands of $\uplus$ (Section

3.5.3) x and y are not so distinguished from each other at low E_b/N_o. Thus the maximum approximation which selects the larger one and discards the other will introduce more errors. At higher E_b/N_o, the decoder is more certain about which path in the trellis to go, which implies that the difference between x and y is larger. As a result, the accuracy of maximum approximation is almost as good as linear approximation. The choice of approximation methods is dependent on the interested E_b/N_o range. Maximum approximation method is preferred at higher E_b/N_o for simpler computation and comparable performance.

5.4 Effect of Interleaver Type

In Section 4.3, the importance of interleaver (de-interleaver) was discussed. There are different types of interleavers available. In this section, we will evaluate the performance of two different types of interleavers depicted in Section 4.3 by comparing their BER performance at different iteration steps.

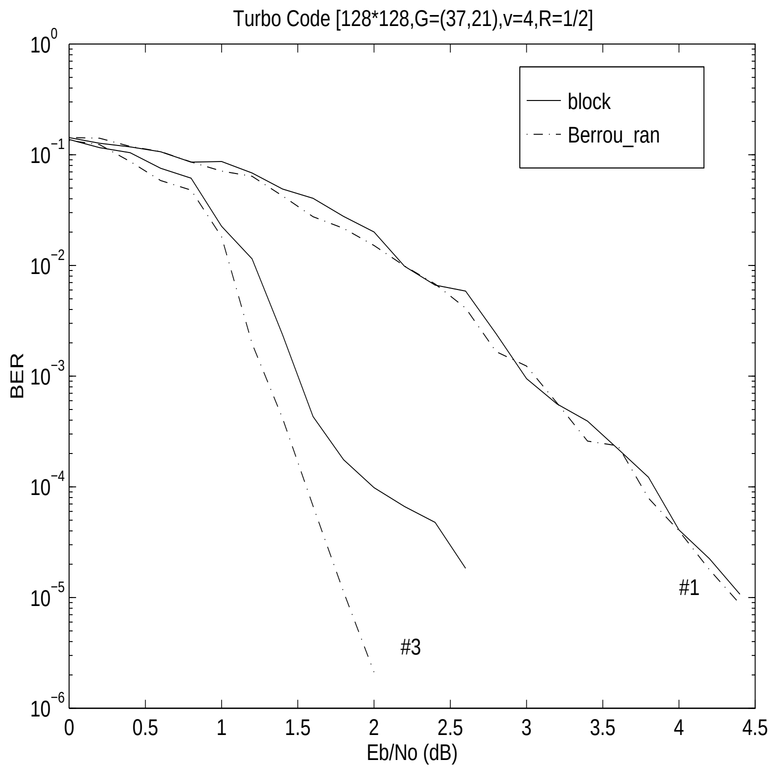


Figure 5.4: BER performance for turbo decoder with different interleavers

From the Figure 5.4, it is apparent that for the first iteration step, both interleavers give comparable performance. This is not surprising as during the first iteration, the interleaver is just used once and the correlation of the *a priori* information with the input bits is weak. Moreover, the first component decoder does not make use of the extrinsic information generated by the second one. Therefore, the interleaver does not play an important role up to this stage. However, for iteration 3, the performance difference is significant as shown in the figure. If a reference point is chosen to be 1×10^{-5}, the coding gain of non-uniform interleaver over block interleaver is about 1 dB. As discussed in Section 4.3, the main function of interleaver in turbo code is to reduce the number of low-weight codewords. That block interleaver is poorer in increasing codeword weight for certain sequences[14] is the main contribution to this performance degradation. Other possible reason is given as follows. As the iterative process goes on, the extrinsic information generated by the second decoder is more correlated with the decoder input. When the correlation is strong and the interleaver fails to suppress this correlation, there will be no further improvement. It is shown that block interleaver is poorer in randomizing the input sequence, hence it is only suitable for the first one or two iterations for long sequence.

5.5 On Line Estimation of Noise Variance

Turbo decoder consists of two MAP component decoders whose performance rely on channel information, namely the variance of the noise in the AWGN channel. For the simulation results shown in previous chapters, perfect knowledge on the channel was assumed. How can the decoder determine the noise variance from the channel observation? This problem of estimating noise variance can be straightforward by transmitting a sequence of known training symbols, however, this will degrade the channel efficiency. In this section, we try to deduce a simple method to estimate the noise variance just based on the channel observation received at the decoder input, i.e. find noise variance from $x_k = d_k + n_k$, where d_k is the data bit with unknown polarity. As the extrinsic information is used as *a priori* information instead of a channel input, it is not necessary to estimate the variance of the extrinsic information.

5.5.1 Estimation of Noise Variance from Channel Observation

By definition, the variance of the channel observation can be calculated as follows,

$$\sigma_{x_k}^2 = \frac{1}{N}\sum_{i=0}^{N}(x_i - \overline{M_{x_k}})^2$$

where N is the sequence length, x_i is the channel observation at time i and $\overline{M_{x_k}}$ is the mean value of the channel observation.

$$\sigma_{x_k}^2 = \frac{1}{N}\sum_{i=0}^{N}(d_i + n_i - \overline{M_{x_k}})^2$$

$$\begin{aligned}\sigma_{x_k}^2 &= \frac{1}{N}\sum_{i=0}^{N}(d_i^2) + \frac{1}{N}\sum_{i=0}^{N}\overline{M_{x_k}}^2 + \frac{1}{N}\sum_{i=0}^{N}n_i^2 \\ &+ \frac{1}{N}\sum_{i=0}^{N}(-2\cdot d_i\overline{M_{x_k}} + 2\cdot d_i n_i - 2\overline{M_{x_k}}n_i) \qquad (5.7)\end{aligned}$$

For large value of N, the latter term in Equation (5.7) can be replaced by its *expected* (mean) value. Assume the noise is Gaussian distributed with zero mean and data bits only take value of 1 or -1 with equal probability, we have,

$$\begin{aligned}\frac{1}{N}\sum_{i=0}^{N}(-2\cdot d_i\overline{M_{x_k}} + 2\cdot d_i n_i - 2\overline{M_{x_k}}n_i) &= E\left[-2\cdot d_i\overline{M_{x_k}} + 2\cdot d_i n_i - 2\overline{M_{x_k}}n_i\right] \\ &= 0 \qquad (5.8)\end{aligned}$$

Substitute Equation (5.8) into Equation (5.7),

$$\sigma_{x_k}^2 = 1 + \overline{M_{x_k}}^2 + \frac{1}{N}\sum_{i=0}^{N}n_i^2 \qquad (5.9)$$

rearrange this equation,

$$\sigma^2 = \sigma_{x_k}^2 - 1 - \overline{M_{x_k}}^2 \tag{5.10}$$

where $\sigma_{x_k}^2$ can be easily acquired at the decoder from the channel observation.

5.5.2 Evaluation of the Estimation Method

The derivation given in the previous section is based on the assumption that the block size N is large such that Equation (5.8) holds. As a result, the accuracy of estimation using the above mentioned method is constraint by value of N. The relationship of estimated noise σ versus the true σ is plotted in Figure 5.5 with the block size N as independent variable.

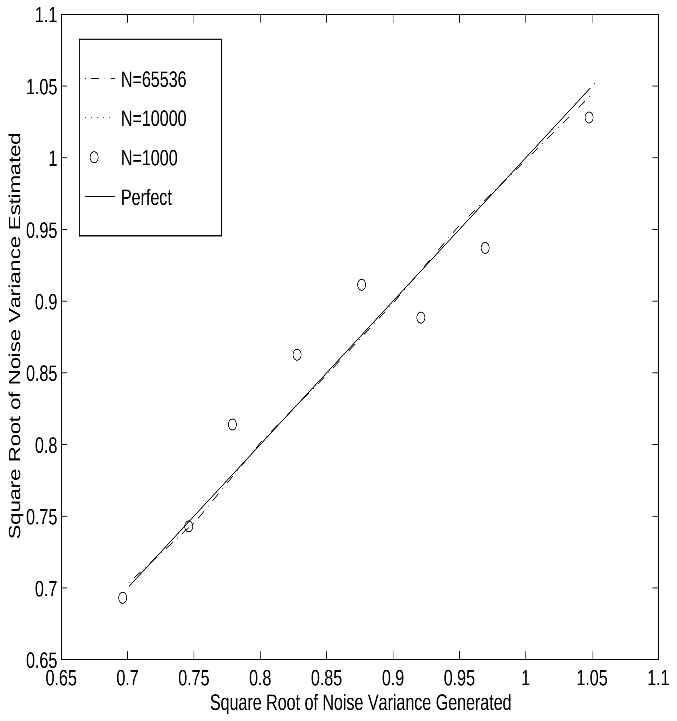


Figure 5.5: Estimated noise variance for different block size

From the curves obtained, it is observed that the simple method works perfectly well for sequences with length larger than 10000. For shorter sequence $N = 1000$, the estimation is not accurate. However, [7] has shown that turbo code can tolerate a mismatch of -2 dB to 6 dB between the estimated E_b/N_o and the actual E_b/N_o without introducing severe performance degradation. Furthermore, for short sequence, we can use both the systematic bit sequence and the parity bit sequence to estimate the noise variance, which implies that the sampled numbers for estimation is $2N$. In case N is sufficiently large, noise variance can be determined from the first half sequences, reducing the delay introduced.

In order to examine the performance of the estimation method stated in the previous section, we assume the turbo decoder has no knowledge on channel information and it uses Equation (5.10) to calculate the noise variance based on the channel observation. A plot of BER versus E_b/N_o together with the curve obtained previously assuming the decoder has perfect channel knowledge is shown in Figure 5.6.

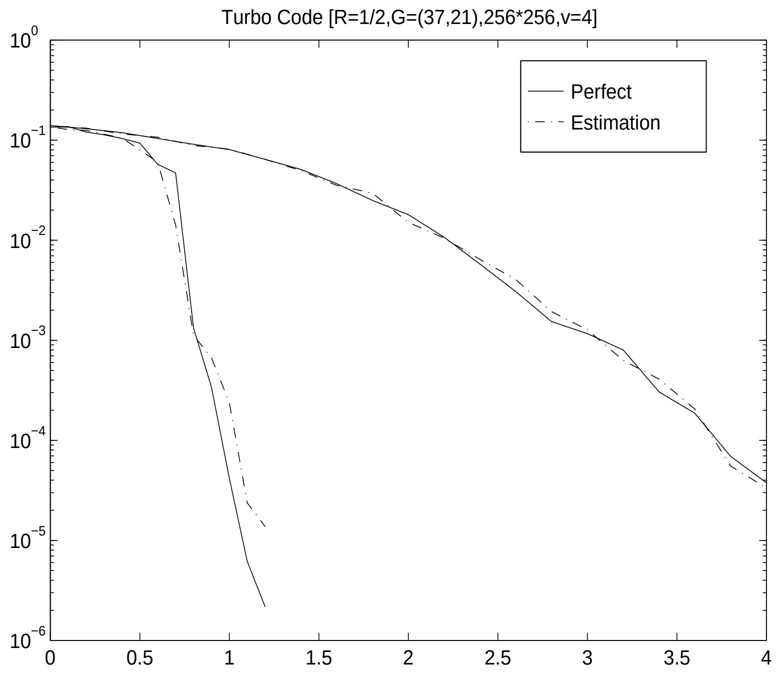


Figure 5.6: BER performance for turbo decoder with on-line noise estimation

It is shown that for the first iteration, turbo code making use of the estimated noise variance performs as good as that assumes perfect knowledge of channel information. For more iterations, the performance degrades slightly. The underlying reason may be the inaccurate information is repeatedly used, accumulating the effect of the small error to a slightly larger value.

Chapter 6

Application of Turbo Code to CDMA System

6.1 Introduction to CDMA System

Over the past several years, demand for mobile communications has increased dramatically. As a result, *Direct Sequence Code Division Multiple Access* (DS-CDMA) becomes a viable alternative to both *Frequency Division Multiple Access* (FDMA) and *Time Division Multiple Access* (TDMA) in wireless cellular communications [8] as it has been shown theoretically to have a system capacity 4-6 times of that for its counterparts.

In DS-CDMA system, the bandwidth of the transmitted signal is spread by a wide band *Pseudo-Noise* (PN) code which is distinct for different users. This idea of spread-spectrum modulation was originally developed for military applications to minimize the effect of intended jammer. It is also used for civilian purpose, like *multiple access* communications where a number of independent users are transmitting information over a shared channel. The processing gain which together with the value of E_b/N_o determines the system capacity of a CDMA system is defined as follows,

$$P = 10 \log \frac{B_{ss}}{B_D} \tag{6.1}$$

where B_{ss} is the spread bandwidth and B_D is the signal bandwidth.

At the receiver, the received signal (product of signal with the PN sequence) is cross-correlated with its corresponding PN sequence. All the noise and signals with mis-matched PN code will be reduced to a insignificant extent as compared with the wanted signal. This idea is best illustrated by the Figure 6.1, The capacity of a

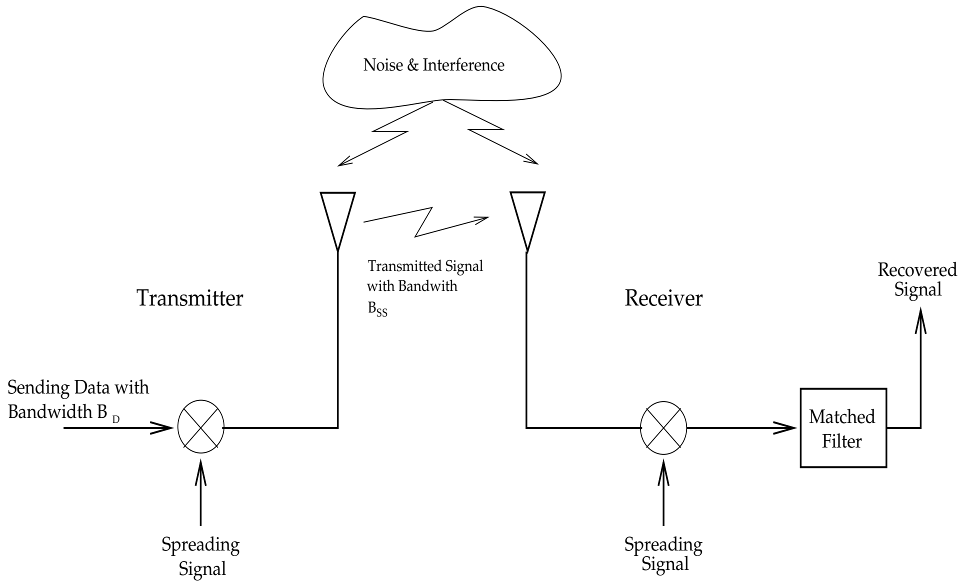


Figure 6.1: Basic Spread Spectrum Technique

single-cell CDMA system is given by [12],

$$M_o \approx \frac{B_{ss}/B_D}{E_b/N_o} \tag{6.2}$$

where M_o is the number of users.

CDMA also makes use of voice activity cycles to increase the capacity by a factor of about 3. This is difficult to be implemented in either FDMA or TDMA because of the time delay associated with reassigning the channel resource during speech process. Some other advantages of CDMA include soft hand off, less fading and easy transition.

The transmission channel from a *Base Station* (BS) to mobiles is called the forward link, while the reverse link refers to that from mobiles to its BS. In this project, we investigate the use of turbo code in cellular DS-CDMA to improve the error correcting performance. It is well known that cellular capacity of DS-CDMA is mainly limited by the reverse link [9, 10]. Hence, only the mobile-to-BS channel is under consideration.

6.2 Turbo Code Application to CDMA System

Chapter 4 has demonstrated the astonishing performance of turbo code, which requires E_b/N_0 of 0.7 dB for BER of 10^{-5} using code rate 1/2 by means of puncturing. This leap towards the ultimate Shannon limit of information transmission comes from the three concepts: recursive systematic convolutional (RSC) code, parallel concatenation with non-uniform interleaving and iterative decoding with minimal feedback correlation. The performance mentioned above needs a interleaver or data block size of 65536 (256×256), which is only feasible in application such as deep space communication, where no stringent requirement is imposed on the system. In mobile communication, the information being transmitted is normally voice data. The delay introduced by the interleaver cannot be too great to make user feel uncomfortable. In the following sections, we will examine whether turbo coding scheme with small interleaver size is applicable in CDMA system for mobile communication.

6.2.1 European JD-CDMA System

Mobile communication standard making bodies suggest that the speech frame should contain less than 200 bits[11]. For example, the pan-European Global System for Mobile communication (GSM) and Digital Cellular System (DCS) 1800 indicate that the speech frames contain 200 bits. Meanwhile, the recently introduced joint detection code division multiple access (JD-CDMA) states frame size of 192 bit in the up-link.

To make explanation easier, the nomenclature is given as TCL$_c$-BL, where TC refers to turbo code, L$_c$ is the constraint length of the RSC encoder and BL stands for uniform block interleaver. For very large data frame size, a random interleaver gives optimum output, while for small data frame size, a uniform block interleaver is good enough. In the following discussion, a 12×16 block interleaver is used to accommodate the block size of 192 bits. The additive BCJR MAP decoder discussed in Section 3.5 is used with the linear approximation for operator $\uplus$. The termination scheme follows the method discussed in Section 5.1: only the first encoder is terminated and second one left open. The two parity sequences are punctured to achieve code rate of 1/2.

Firstly, we examine the performance of the TC5-BL with generator polynomial [37,21] proposed in [1] over the AWGN channel. It is apparent from Figure 6.2 that the

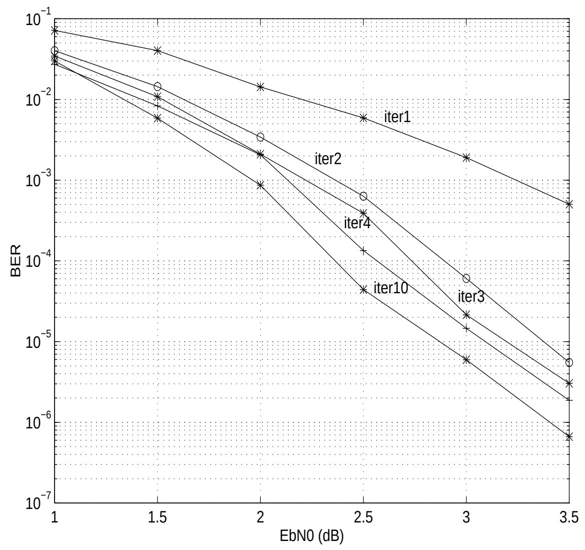


Figure 6.2: BER performance of TC5-BL with interleaver of 12×16

BER performance of the TC5-BL gets better with the increase of the iterations. After circulating in the turbo decoder for many times, the new extrinsic information comes out of a MAP decoder is more and more correlated with the old extrinsic information at the input of this MAP decoder. Hence, further iteration will be fruitless. From Figure 6.2, the BER improvement becomes negligible after 10 iterations. TC5-BL gives BER of 10^{-5} at $E_b/N_0 = 2.8$dB. This is not as impressive as the case in Figure 4.6 that only $E_b/N_0 = 0.7$dB is needed for same BER by using random interleaver with interleaver of (256×256) at code rate of $1/2$. This degradation is due to the short data sequence. For the same bit, the permutation done by the interleaver cannot separate them by a large enough distance in the original sequence and interleaved sequence. The extrinsic information of the same bit in different sequences are most likely to be in error together and cannot help each other to combat noise.

For short frame transmission, Frame Error Rate (FER) is another parameter to judge the performance of the coding scheme. In frame oriented data transmission, Automatic Repeat Request is implemented to ask the sender to re-send the data frame if errors are detected. A system with high FER induce speech echoes. Figure 6.3 portrays the FER performance of TC5-BL.

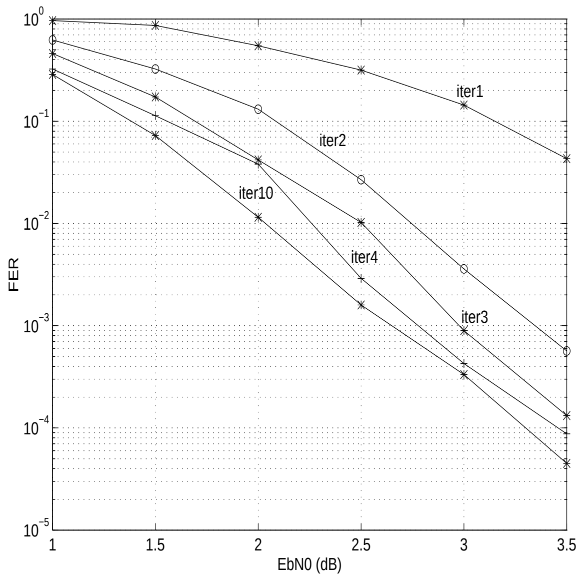


Figure 6.3: FER performance of TC5-BL with interleaver of 12×16

At stage of hardware implementation, all MAP decoder are placed parallel in the chip. The output from turbo decoder can be sampled at any point. For example in Figure 4.2, both LLR at output of decoder1 or decoder2 can be used as final result to do hard-decision. A interesting point from Figure 6.2 is: there are large gaps between the first 4 iterations. This motivates us to investigate the BER performance when sampling LLR from decoder1. Hence, the iteration number is 1.5, 2.5, 3.5 etc.. Figure 6.4 and 6.5 show the BER and FER performance for these iterations. First two iterations contribute most of the improvement. Iteration 2 improve the E_b/N_0 by 1 dB compared as iteration 1 for the same BER. The total gain in E_b/N_0 is around 1 dB. Iteration 2.5 is an attractive selection, because it need quarter of the computation but give BER performance with degradation less than 1dB compared with 10 iteration.

From Table 4.1, the computation load of a turbo decoder grows exponentially with the constraint length and linearly with the number of iterations. It is worthwhile

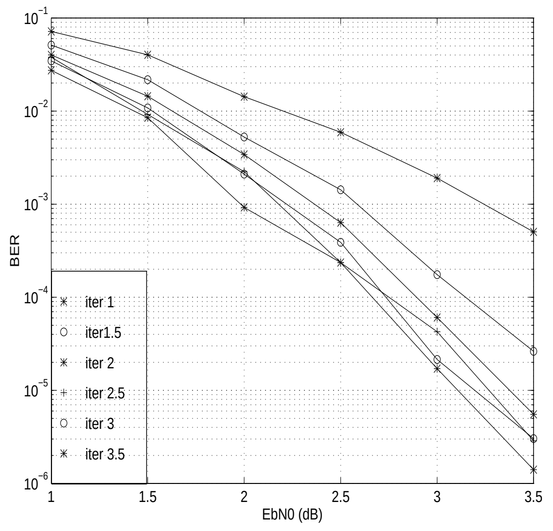


Figure 6.4: BER performance of TC5-BL with interleaver of 12×16 at iteration 1.5, 2.5 and 3.5

to examine the performance of the turbo decoder with shorter constraint length but with more iterations. Therefore, simulations is done to TC3-BL and TC4-BL with generator polynomial of [7,5] and [13,15] respectively. TC5-BL has 16 states and TC4-BL has 8 states. In term of computation, 5 iterations of TC5-BL is equivalent to 10 iterations of TC4-BL. However, the latter gives a superior performance. This is more clearly shown in Figure 6.6, where the performance of two codes are summarized together. Figure 6.7 depicts the performance of TC3-BL (10 iterations) and TC5-BL (2.5 iterations) on ground of same computation effort. From the plot, it is clear that the two code have comparable BER and FER performance. However, TC3-BL, which has only 4 states, is relatively easy to be implemented than TC5-BL. If powerful processing electronic product is available and computation load is out of consideration, TC5-BL is best candidate because of its superior performance. TC3-BL will be the choice where processing delay is an important criterion of the system performance.

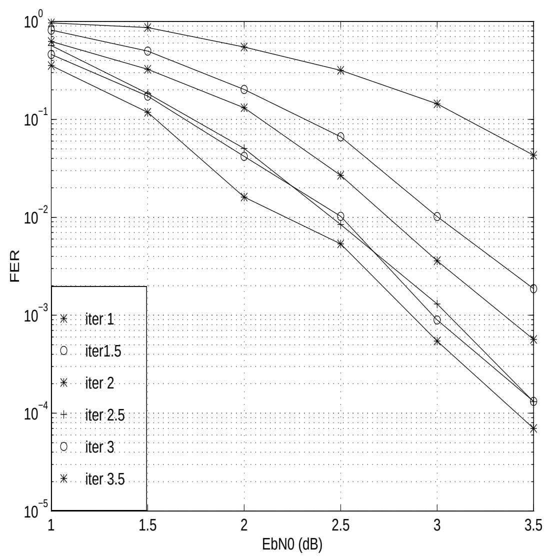


Figure 6.5: FER performance of TC5-BL with interleaver of 12×16 at iteration 1.5, 2.5 and 3.5

6.2.2 QCDMA Reverse Link

Qualcomm Inc. is the first company in North America that proposed the spread spectrum digital cellular system, based on CDMA. The Qualcomm CDMA (QCDMA) features all common techniques that increase the overall capacity of the mobile communication system. It has unique specification based on the design philosophy of Qualcomm Inc. For example, in both forward and reverse link, the information to be transmitted is encoded a convolutional encoder with constraint length $K = 9$ code and code rate of 1/2 and 1/3, respectively[12]. This two highly redundancy error correcting codes, together with interleavers, long Pseudo-Noise (PN) sequences, and two independent I and Q channel BPSK modulation schemes, makes the digital voice performance acceptable at high noise or interuser interference environment. From Section 3.1, we know that the complexity introduces by convolutional encoder and decoder grows exponentially with the constraint length. The encoder used in both forward and reverse link of QCDMA has $2^9 = 512$ states, which is extremely complex and expensive to be implemented. It is meaningful to investigate the possibility of using turbo code of smaller constraint length K to replace the traditional convolutional code with $K = 9$. Mobile communication system capacity is mainly limited by reverse link. Hence, simulation is only conducted based on reverse link of QCDMA.

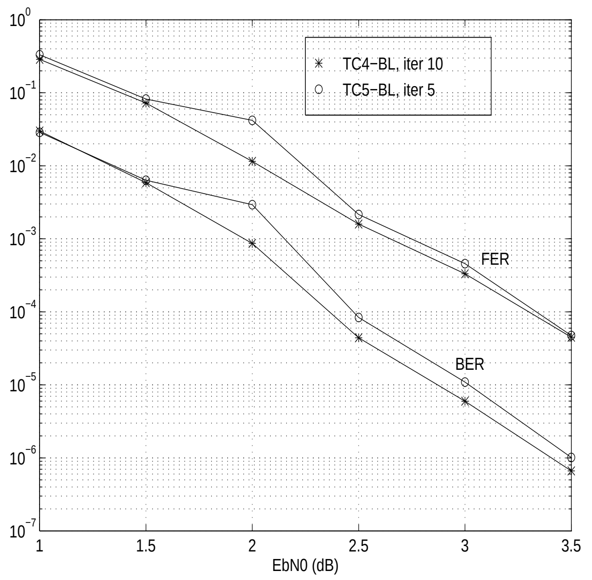


Figure 6.6: BER and FER performance of TC4-BL (iter. 10) and TC5-BL (iter. 5)

Figure 6.8 depicts schematically the structure of the reverse link employed in QCDMA system.

In QCDMA specification, delay introduce by block interleaver in both forward and reverse links is stated to be less than 20ms. The information data at the input of the reverse links is clocked at 28.8 kbps. Therefore, the interleaver block size is 20ms $\times$ 28.8kbps $=$ 576bits. TC5-BL with generator polynomial of [31,27] and 32 $\times$ 18 linear block interleaver , which accommodates 576 bits/block, is used to conduct simulation. Other details, such as tail termination, approximation method, of TC5-BL is exactly same with that discussed in Section 6.2.1. Only difference is that no puncturing is employed. For fair comparison, both parity bit sequences (permuted and unpermuted) are sent over the AWGN channel from encoder to decoder. This results in the code rate $r = 1/3$. Figure 6.9 depicts the BER performance of TC5-BL (6 iterations).

In [12], a plot is given for upper bound on BER of convolution code with $K = 9$ and $r = 1/3$. By comparing the results in that paper and the results from Figure 6.9, Table 6.2.2 is constructed. It is straightforward to conduct simulations to examine the feasibility of application of TC4-BL and TC3-BL in same conditions. Results are obtained, which are similar to those discussed in Section 6.2.1.

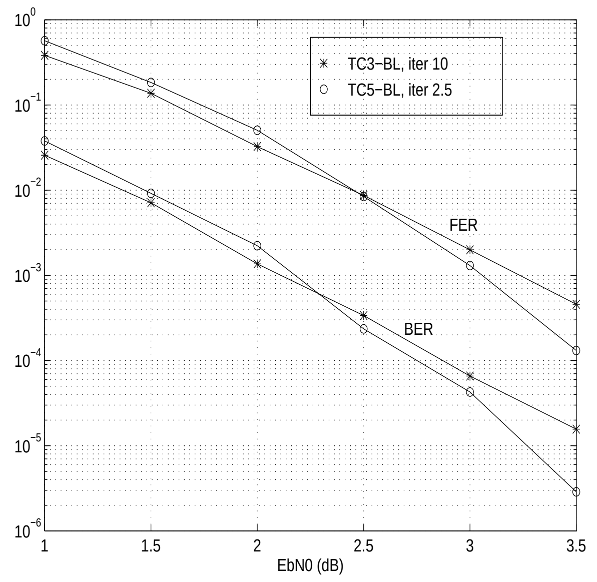


Figure 6.7: BER and FER performance of TC3-BL (iter. 10) and TC5-BL (iter. 2.5)

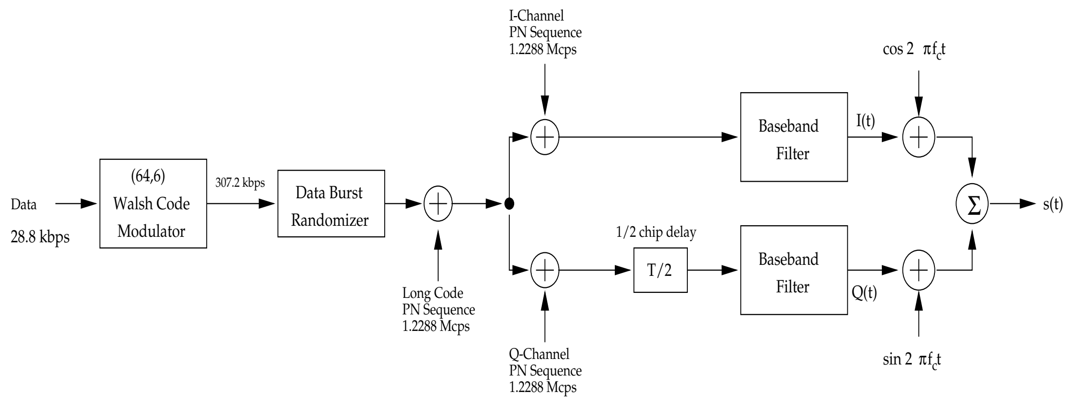


Figure 6.8: QCDMA cellular system reverse link

After going through all the figures and tables, a confident conclusion can be drawn:

1. Turbo code is a promising forward error correction coding scheme in CDMA system for mobile communication system.

2. TC5-BL, TC4-BL, TC3-BL can all give acceptable result in term of BER and FER.

3. Realization of TC5-BL, which gives the best performance, is most expensive in term of complexity and computation required.

4. In practical situation, TC3-BL is the most attractive solution.

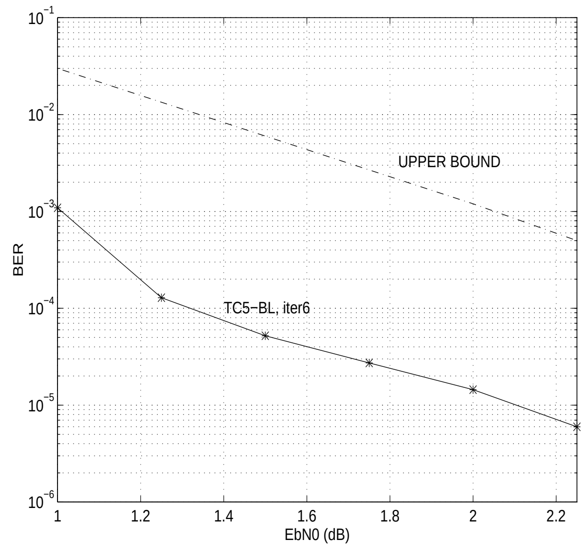


Figure 6.9: BER performance of TC5-BL with interleaver size of 32×18 in QCDMA reverse link

BER	10^{-3}	10^{-4}	10^{-5}
Upper Bound	2.05dB	2.7dB	3.3dB
TC5-BL(iter 6)	1.0dB	1.3dB	2.1dB
Improvement	1.05dB	1.4dB	1.3dB

Table 6.1: Improvement of TC5-BL in QCDMA reverse link compared to upper bound of convolutional code with $K = 9$

Chapter 7

Conclusion and Recommendations

7.1 Conclusion

In this project, the working principles of BCJR MAP decoder, turbo code were studied and implemented in C. Simulation results were obtained for various cases with different parameter values, from which we draw conclusions as follows.

1. The logarithm likehood ratio (LLR) from a MAP component decoder is composed of three terms, namely, the weighted version of the noisy systematic bit, the *a priori* information and the so-called extrinsic information which is used as *a priori* information for subsequent component decoder to iteratively improve the error correcting performance. With the application of turbo principle, near Shannon limit error correction performance can be achieved. E_b/N_o of $0.7dB$ is required to achieve bit error rate of 10^{-5}.

2. The performance of turbo code is a function of various factors, such as sequence length, type of interleaver employed, encoder polynominal and knowledge of channel noise variance. Trade-off can be made between system performance and computation load by changing the encoder constraint length and number of iteration steps.

3. The BCJR algorithm employed in the MAP component decoder can be implemented in log domain. With proper approximation methods, the computation complexity can be reduced without introducing unacceptable performance degradation.

4. Turbo code can be applied in CDMA systems for forward error correction. With encoder constraint length less than 5 instead of the conventional length of 9, turbo code can provide coding gain of more than 1 dB over the upper bound.

7.2 Recommendations

Due to the time constraint and divergent topics related to turbo code, the final year project is unable to cover many details, leaving some interesting fields untouched. The following recommendations are made for further study in turbo code.

1. The performance of turbo code relies on the type of interleaver employed as demonstrated in chapter 5. However, the design of interleaver best suit for turbo code is still an open problem. It is interesting to study the mathematical aspects of a non-uniform interleaver.

2. In all simulations conducted, each floating number is represented by 8 bytes which are sufficient to provide satisfactory precision. However, in hardware implementation, numbers may be represented by less bits, introducing the so-called quantization error. Furthermore, it is possible to use integers to approximate floating numbers to save memory. It is worth of studying the degree of performance degradation due to these errors.

3. The idea of multiple turbo codes was proposed in [13]. However, the rule for combining the *a posteriori* information from two different component decoders is not specified. The structure of the multiple decoders is of importance for further study.

Bibliography

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "*Near Shannon Limit Error Correcting Coding and decoding: Turbo-Codes*", ICC'93, Geneva, Switzerland, pp. 1064-1070, May 1993.

[2] Bahl. L., Cocke, J., Jelinek, F., and Raviv, J., "*Optimal Decoding of Linear Codes for Minimising Symbol Error Rate*", IEEE Trans. Inform. Theory, vol. IT-20, pp. 284-406, May 1974.

[3] S. S. Pietrobon and A. S. Barbulescu, "*A Simplification of the Modified Bahl Algorithm for Systematic Convolutional Codes*", Proceedings of ISITA'94, Sydney, Australia, pp. 1073-1077, November 1994.

[4] S. Benedetto and G. Monorsi, "*Soft-Output Decoding Algorithms in Iterative Decoding of Turbo Codes*", TDA Progress Report, 42-124, February, 1996.

[5] Mark C. Reed, Steven S. Pietrobon, "*Turbo-Code Termination Schemes and a Novel Alternative for Short Frames*", PIMRC'96, Taipei, Taiwan, Oct. 15-18, 1996.

[6] P. Robertson, "*Illuminating the Structure of Code and Decoder of Parallel Concatenated Recursive Systematic (Turbo) codes*". Globecom, pp. 1298-1303, 1994.

[7] Todd A. Summers, Stephen G. Wilson, "*SNR Mismatch and On-Line Estimation in Turbo Decoding*", http://www.ee.virginia.edu/CSL/turbo_codes/tcodes-bib.html.

[8] R. L. Pickholtz, L. B. Milstein and D. L. Schilling, "*Spread Spectrum for Mobile Communications*", IEEE Trans. Veh. Technol., vol. VT-40, no. 2, pp. 313–322, May 1991.

[9] K. H. Li, K. L. Cheah, "*Performance Evaluation of Cellular CDMA Systems over Fading Channels*", Proc. APCC '95, Jakarta, Indonesia, Nov. 1995.

[10] J. M. Holtzman, "*DS/CDMA Successive Interference Cancellation*", Proc. ISSSTA '94, Oulu, Finland, pp. 69–78, July 1994.

[11] P . Jung, "*Comparison of Turbo-Code Decoders Applied to Short Frame Transmission Systems*", IEEE Journal On Selected Area in Communications, VOL 14, NO 3, April, 1996.

[12] Jhong Sam Lee, "*Overview of the Technical Basis of Qualcomm's CDMA Cellular Telephone System Design*", ICCS'94, Singapore, 1994.

[13] D. Divsalar, F. Pollara, "*Multiple Turbo Codes for Deep-Space Communications*", TDA Progress Report, pp 42–121, May 15, 1995.

[14] D. Divsalar, F. Pollara, "*Turbo Codes for Deep-Space Communications*", TDA Progress Report, pp 42–120, Feb 15, 1995.

[15] Komulainen P., Pehkonen K., "*A Low-Complexity Superorthogonal Turbo-Code for CDMA Applications*", pp 369–373, IEEE, 1996.

[16] D. E. Knuth, *The Art of Computer Programming*, vol II, pp. 122–123, 3rd ed, Addison-Wesley, 1997.

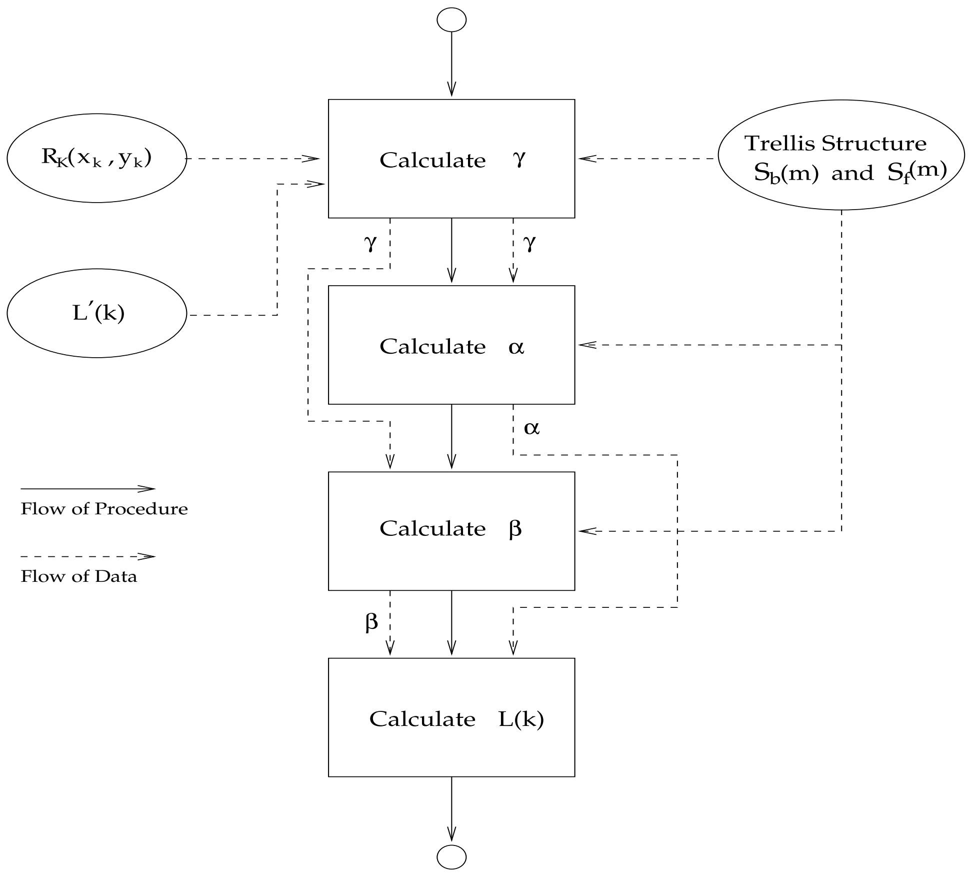
[17] S. Haykin, *Communication System*, 3rd ed. John Wiley & Sons, 1994.

Appendix A

List of Flowchart

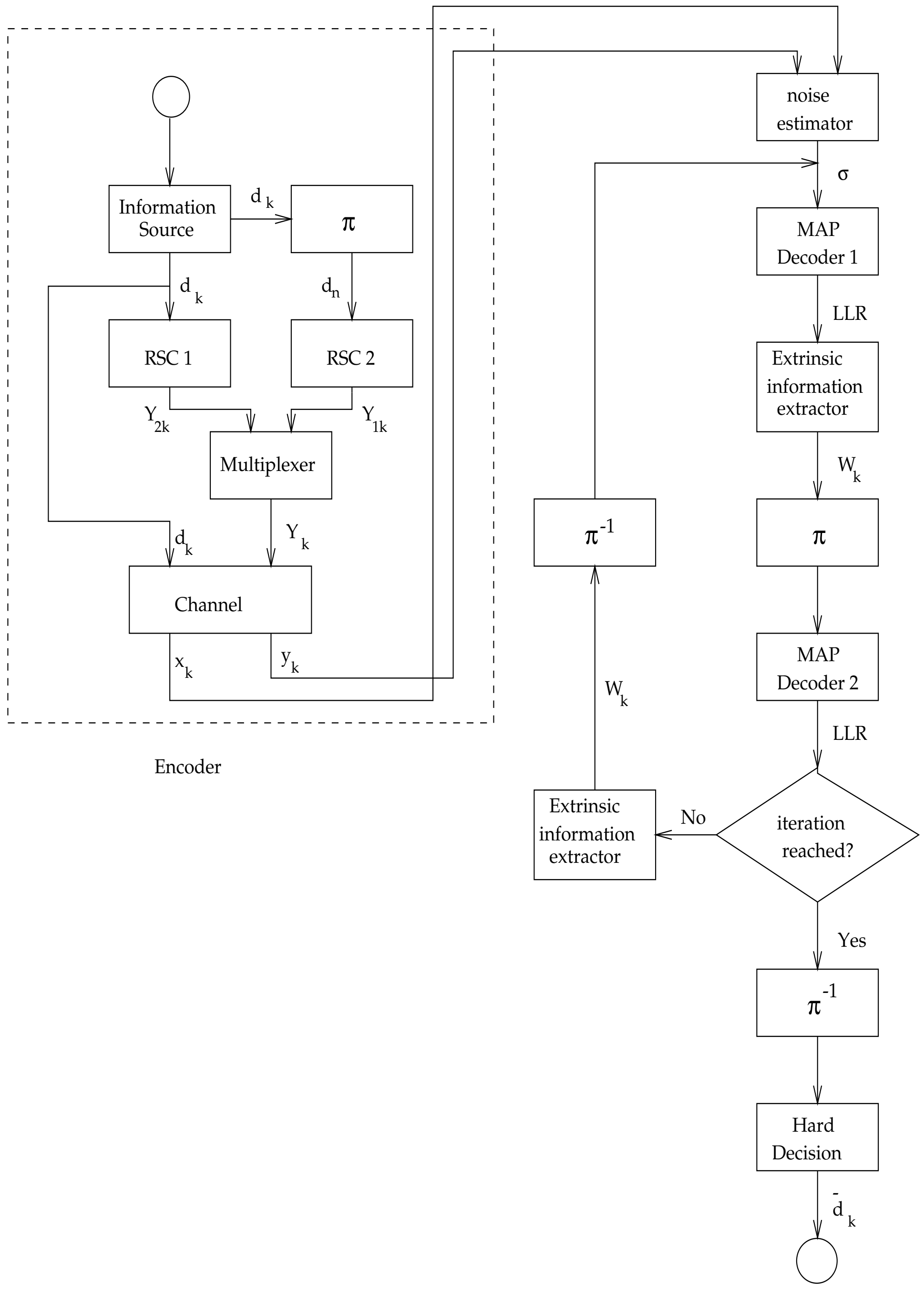
A.1 Flowchart for BCJR MAP Decoder

The symbols follow the convention defined in Section 3.4.2.



A.2 Flowchart for Turbo Decoder

The symbols follow the convention defined in Chapter 4.



TURBO CODE IN CDMA SYSTEM

SUN JIANHUA
WANG QI

SCHOOL OF ELECTRICAL & ELECTRONIC ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY
1997/98