Performance of Turbo Codes under Designed Interleaving and Puncturing Conditions

by

Ali Hussain Abdallah Mugaibel

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

ELECTRICAL ENGINEERING

May, 1999

INFORMATION TO USERS

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

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

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

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

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

UMI®

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

Performance of Turbo Codes under Designed Interleaving and Puncturing Conditions

BY

ALI HUSSAIN ABDALLAH MUGAIBEL

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE
In

ELECTRICAL ENGINEERING

May 1999

UMI Number: 1395620

UMI Microform 1395620
Copyright 1999, by UMI Company. All rights reserved.

This microform edition is protected against unauthorized copying under Title 17, United States Code.

UMI
300 North Zeeb Road
Ann Arbor, MI 48103

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS
DHAHRAN, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **Ali Hussain Abdu-Allah Mugaibel** under the direction of his Thesis Advisor and approved by his Thesis Committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN ELECTRICAL ENGINEERING**

Thesis Committee

Dr. Maan A. Kousa, **Chairman**

Dr. Maamar Bettayeb, **Member**

Dr. Saud A. Al-Semari, **Member**

Dr. Adil Balghonaim, **Member**

Dr. Samir Al-Baiyat
Chairman, Department of Electrical Engineering

Dr. Abdallah M. Al-Shehri
Dean, Graduate Studies

31/5/99
Date

Dedication

All praises goes to Allah,

Yaa Allah,
Have Mercy and,
Accept it as a dedication to you

Yaa Allah,
Show it to me in the last day with the good deeds

Your Ni'ma, I cannot deny, led to this accomplishment

Acknowledgements

My unreserved praises and thankfulness are for Allah, the Most Compassionate, and the Most Merciful. He blesses me with his ever-enduring mercies. May His peace and blessings be upon Prophet Muhammed, and his family.

I acknowledge the support and facilities provided by the Electrical Engineering Department and the Deanship of Graduate Studies at King Fahd University of Petroleum and Minerals.

I would like to express my special thanks and appreciation to my thesis advisor, Dr. Maan Kousa. He is an exceptional person in all advising aspects. He has punctured my mistakes and deinterleaved my scattered ideas. After sufficient number of iterations, and utilizing the decision feedback from the thesis committee members, I ended up this thesis work with low error rate at relatively short time. A set of words cannot do the thanking job. Thank you "Dr. Maan" for your time and effort. I also remain grateful to Dr. Saud Al-Semari for his constructive suggestions and comments. Throughout my learning process, he strengthened my skills and understanding of the communication concepts. I also would like to thank the other thesis committee members, Dr. Maamar Bettayeb and Dr. Adil Balghonaim for their valuable criticisms, helpful suggestion and kind cooperation.

I also want to thank Mr. Bassam Al-Dossary, Mr. Fahad Al-Rashed, Mr. Furaih Al-Rwaili, Mr. Ghassan Al-Regeb and Mr. Ali Sheikhi for what they did. I am indebted to all those friends and brothers who have helped me through their patience and understanding.

The unflinching support of my family made all the heavy tasks achievable. May Allah reward their patience.

Contents

Acknowledgements ..... **i**

Abstract (English) ..... **x**

Abstract (Arabic) ..... **xi**

1 INTRODUCTION **1**

1.1 GENERAL OVERVIEW 1

1.2 TURBO CODES 2

1.2.1 Introduction *2*

1.2.2 Turbo Encoder *3*

1.2.3 Turbo Decoder *6*

1.3 INTERLEAVING 10

1.4 PUNCTURING 14

1.5 THESIS CONTRIBUTION 16

1.5.1 Thesis Organization *18*

2 PERFORMANCE EVALUATION OF TURBO CODES 19

2.1 INTRODUCTION 19

2.2 BOUNDING TECHNIQUES 20

2.3 UNION BOUND FOR TURBO CODES 21

2.3.1 The Constituent Code 21

2.3.2 Transfer Function of Turbo Codes 26

2.4 WEIGHT DISTRIBUTION 30

2.4.1 Free Distance 31

2.5 SIMULATION 32

3 INTERLEAVING EFFECTS ON THE CODE PERFORMANCE 34

3.1 INTRODUCTION 34

3.2 INTERLEAVER OBJECTIVES 35

3.2.1 Scattering of Burst Errors 35

3.2.2 Independent Parity 36

3.2.3 Shaping the Weight Distribution 36

3.3 CLASSES OF INTERLEAVERS 39

3.3.1 Matrix Interleavers 44

3.3.2 Weight-Two Breaker 47

3.4 STUDY ISSUES 55

3.4.1 Interleaver Length 55

3.4.2 Number of Iterations 59

3.4.3 Interleaving (I) / Deinterleaving (I^{-1}) *66*

3.4.4 Cyclically Shifted Interleavers *71*

3.5 ASYMPTOTIC BEHAVIOR OF TURBO CODES 71

4 PUNCTURING EFFECTS ON THE CODE PERFORMANCE 80

4.1 INTRODUCTION 80

4.2 PUNCTURING MATRIX 82

4.3 WEIGHT DISTRIBUTION OF PUNCTURED TCS 83

4.3.1 Puncturing Parity Bits *83*

4.3.2 Puncturing Systematic Bits *85*

4.4 Modified Transfer Function *86*

4.4.1 Method 1 ..... *86*

4.5 COMPARISON BETWEEN THE PROPOSED AND THE AVERAGING TECHNIQUES 90

4.6 PUNCTURING PATTERNS 100

4.6.1 Puncturing Systematic Sequence *100*

4.6.2 Puncturing Parity sequences *107*

4.6.3 Permutations of Columns *108*

4.6.4 Permutations of Parity Rows *111*

4.6.5 Guidelines of a Good Puncturing Matrix *112*

4.7 DECISION DEPTH OF PUNCTURED TC 112

5 SUMMARY, CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK 116

5.1 INTRODUCTION 116

5.2 SUMMARY OF INTERLEAVING RESULTS 117

5.3 SUMMARY OF PUNCTURING RESULTS 119

5.4 SUGGESTIONS FOR FUTURE WORK 120

Nomenclature **121**

Bibliography **124**

List of Tables

4.1 Codewords as a result of weight-two input for the (7,4) Hamming code..... 91

4.2 Codewords as a result of input weight 2 for punctured (7,4) Hamming code... 93

(a) $b_1\ b_3$ punctured conditional weight distribution.....

(b) $b_0\ b_2$ punctured conditional weight distribution.....

4.3 CIWEF using the averaging and the new method 95

4.4 D_m using the averaging and the proposed methods..... 96

List of Figures

1.1 Simplified turbo encoder 4

1.2 (a) Non-recursive non-systematic code 7

(b) Corresponding recursive systematic code .....

1.3 Block diagram of aturbo decoder 8

2.1 Different representations of the (1,5/7) encoder..... 23

2.2 Trellis diagram for the 4-state (1,5/7) convolutional code 25

3.1 (a) Random interleaver 40

(b) Cyclic interleaver .....

(c) (n,m) matrix interleaver .....

3.2 Performance of TCs under different random interleavers 42

3.3 Performance of cyclic interleavers for different cyclic shifts..... 43

3.4 Performance of matrix interleavers for different values of m and n 46

3.5 Theoritical weight distributions 48

3.6 Illustration of an "unfortunate" weight-two mapping 50

3.7 Performance of weight-two breaker 53

3.8 Performance of different interleaver classes 54

3.9 Effects increasing the frame size *k* on the code performance..... 56

3.10 Relation between BER and *k* does not hold for all interleavers 58

3.11 Effect of frame size *k* for different number of iterations..... 60

3.12 Relation between number of iterations and the interleaver 61

3.13 Effect of the number of iterations for different scattering powers 63

3.14 Effect of number of iterations on the W2B and a random interleaver 64

3.15 (a) Effective number of iterations for k=192, (16,12) matrix..... 65

(b) Effective number of iterations for k=768=4*192, (32,24) matrix .....

3.16 Interleaver/deinterleaver performance..... 69

3.17 (a) Decoder I, I^{-1} loop 70

(b) Turbo encoder (with interleaver I^{-1}).....

(c) I and I^{-1} are added to (b) .....

(d) Similarity of (b) to the turbo encoder with (I) as the interleaver.....

3.18 Performance of cyclically shifted interleavers 72

3.19 Two performance regions of TCs..... 73

3.20 Effect of increasing the number of terms in the union bound 75

3.21 Asymptotic performance for convolutional and TC..... 76

3.22 Interleaver effect on the asymptotic behavior (sketch) 79

4.1 Comparison of the two bounding techniques 98

4.2 Puncturing systematic vs. parity bits using the union bound 102

4.3 Puncturing systematic vs. parity bits based on simulation 103

4.4 (a) Modified state diagram (5/7) code 106

(b) Modified state diagram (2/7) code .....

4.5 Performance of TC under different puncturing arrangements of parity sequences

..... 109

4.6 Performance under different permutations on the puncturing matrix 110

4.7 Effect of increasing the decision depth for unpunctured codes 113

4.8 Effect of increasing the decision depth for punctured code 114

THESIS ABSTRACT

Name: Mugaibel, Ali Hussain Abdallah

Title: Performance of Turbo Codes under Designed Interleaving and Puncturing Conditions

Degree: Master of Science

Major Field: Electrical Engineering

Date of Degree: May, 1999

The astonishing performance of turbo codes has attracted many researchers which resulted in a huge amount of literature since their introduction few years ago. As turbo codes did not actually arise form applying a pre-existing theory, most of their outstanding features remain to be explained.

In this thesis, we examine the behavior of the code with respect to two processes: interleaving and puncturing. Different interleaver classes are evaluated. The effect of the interleaver parameters on the code performance as well as the interaction between the interleaver and the other code elements are studied thoroughly.

This thesis also contributes towards the analysis of turbo codes under puncturing. By developing a new more accurate approach of estimating the weight distribution, the thesis arrived at a new tighter upper bound on the bit error rate performance of the code. The developed approach also allows investigating different puncturing patterns. We provide an explanation of why puncturing systematic bits must be avoided. Guidelines for designing a good puncturing matrix are highlighted.

The results and findings of this work are very useful for understanding the behavior of turbo codes and improving their design.

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS,

DHAHRAN, SAUDI ARABIA

خُلاصة الرّسالة

الاســـــــم : علي حسين عبد الله مقيبل

العنـــــــوان : أداء ترميزات التيربو تحت ظروف تخلخل وتعطيب مصممة

التخصـــــص : هندسة كهربائية

التاريـــــخ : مايو ١٩٩٩م

لقد جذب الأداء المذهل لترميزات التيربو (Turbo Codes) كثيراً من الباحثين لدراسة هذا الترميز ، فنتج عن ذلك كم هائل من الأبحاث في الفترة القصيرة الماضية، وبما أن ترميزات التيربو لم تنشأ كنتيجـــة لتطبيـــق نظريات موجودة فإن معظم خصائصها الباهرة ما تزال بحاجة إلى تفسير.

يدرس هذا البحث خصائص الشفرات السريعة تحت تأثير عمليتي : التخلخـــل (Interleaving) و العـــطب (Puncturing). ولقد قيّمت نوعيات مختلفة من المخلخلات، وكذلك درس أثر تغيير صفات المخلخل علـــى أداء الترميز ، بالإضافة إلى دراسة التفاعل بين المخلخل وعناصر الترميز الأخرى.

يساهم هذا البحث في تحليل ترميزات التيربو تحت ظروف العطب، عن طريق تطوير طريقة جديدة ودقيقـة لتقدير توزيع الأوزان (Weight Distribution)، ولقد توصل الباحث إلى تقييم أدق للحد الأعلـــى (Upper Bound) لأداء الترميز بالنسبة الى نسبة الخطأ في البتات (BER). وكذلك مكّنت هذه الطريقة المطورة مـــن بحث أنماط العطب المختلفة. ولقد استطاع الباحث من إيجاد تفسير لظاهرة الأداء السيئ عند عطب البتـات المنتظمة (Systematic Bits)، كما يضع البحث الخطوط العريضة لتصميم مصفوفة تعطيب جيدة.

إن مكتشفات ونتائج هذا البحث مفيدة جداً في فهم وتفسير خصائص وتصرفات ترميزات التيربو وتحسين تصميمها.

درجة الماجستير في العلوم

جامعـــــة الملك فهد للبترول والمعادن

الظهران، المملكة العربية السعودية

مايو ١٩٩٩م

CHAPTER 1

INTRODUCTION

1.1 General Overview

One of the fundamental results of channel coding theory is that the performance of a code is proportional to its length for block codes or to its constraint length for convolutional codes. However, increasing the code length results in increasing the complexity of the decoder. Researchers have been trying to construct long codes that possess enough structure to be decoded easily. Proposals include product codes, concatenated codes, iterated codes and combinations of them. The most successful attempt so far is that of turbo codes proposed by Berrou *et al* [1].

The introduction of turbo codes attracted many researchers and produced an explosive amount of literature. Since turbo codes did not actually result from applying a pre-existing theory, most of their outstanding features remain to be explained [2]. Neither a

good explanation of the code nor an adequate understanding of the relative importance and interaction of the code ingredients (constituent codes, interleaver, ...etc) have appeared yet.

In this introductory chapter, we start by describing turbo codes. Both the encoder and the decoder are reviewed. Being the main emphasis of this thesis, an up-to-date literature survey on interleaving and puncturing and their effects on the code performance is then presented. The chapter ends with stating the thesis contribution and highlighting the thesis organization.

1.2 Turbo Codes

1.2.1 Introduction

In channel coding, redundancy is added to the information sequence in order to make the system more reliable. The channel coding theorem states that even at relatively low signal-to-noise ratios, reliable communication can still be maintained provided that channel capacity limit is observed. However, the theorem tells us nothing about how to design the code that achieves such performance. All what it says is that the code should appear random. Unfortunately random codes are very difficult to decode. It is essential to have some structure in the code to make the decoding feasible. Researchers have been trying to resolve these two seemingly conflicting needs: structure and randomness.

Lately [1], the proposal of **P**arallel-**C**oncatenated **C**onvolutional **C**odes (PCCC), called *turbo codes*, has solved the dilemma of structure and randomness by allowing structure

through concatenation and randomness through interleaving. The introduction of turbo codes has increased the interest in the coding area since these codes provide most of the gain promised by the channel-coding theorem.

Turbo codes have an astonishing performance of bit error rate (BER) at relatively low ratio of bit energy to noise power spectral density (E_b/N_0). To give an idea of how powerful turbo codes are, a BER=10^{-5} was achieved over additive white Gaussian noise (AWGN) channel at E_b/N_0=0.7dB, which is very close to the channel capacity (-0.8 dB) [1]. For a Rayleigh fading channel a BER=10^{-5} was achieved at E_b/N_0=4.3dB which represents a gain of 2.3dB as compared to classical convolutional codes with similar complexity [3]. Recently [4], A BER=10^{-4} was achieved by a rate-1/5 turbo code at 0 dB, which is only 1.2 dB from channel capacity limit.

In the following subsections the structure of both the encoder and the decoder are explained. The modified soft output Viterbi algorithm (SOVA) for decoding proposed by [5] is briefed. Normalized SOVA, an improvement on SOVA suggested by [6], is also highlighted.

1.2.2 Turbo Encoder

In a simplified turbo code, there are two convolutional encoders in parallel. The information bits are scrambled before entering the second encoder as depicted in Figure 1.1. The codeword in a turbo code consists of the input followed by the parity check bits from the first encoder and then the parity bits from the second encoder, i.e. the augmentation $X_2\ X_1\ X_0$.

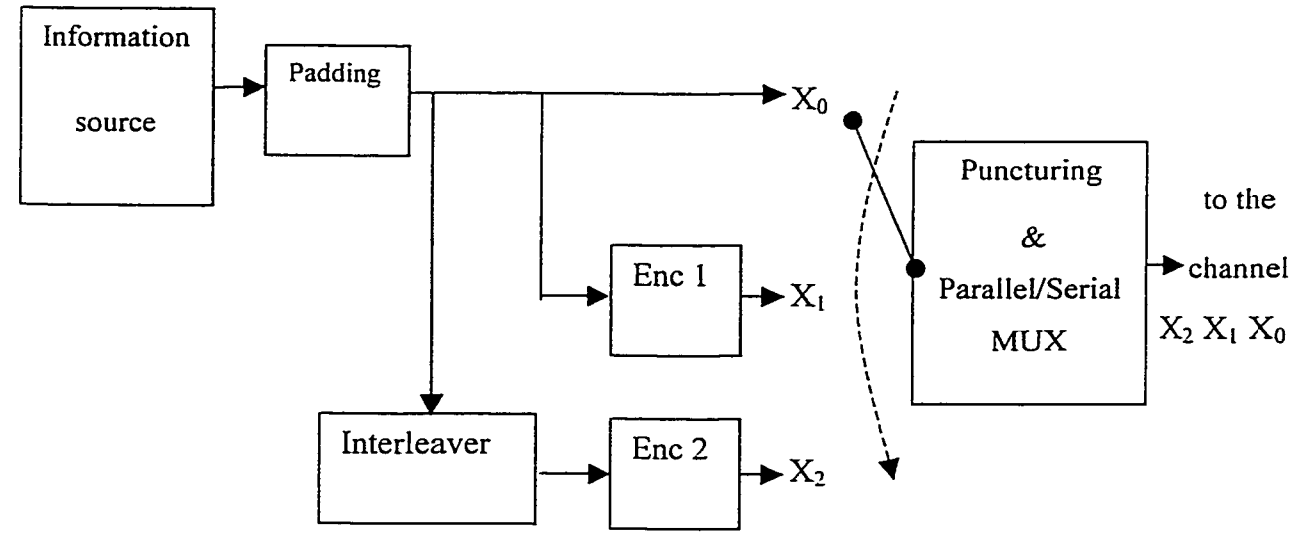


Figure 1.1: Simplified turbo encoder

The simplified turbo code block diagram in Figure 1.1 shows only two branches, other than the systematic branch. In general, a turbo code may have more than two branches [7]. The convolutional code in every branch is called the constituent code (CC). The CCs can have similar or different generator functions. We will concentrate on the most widely used configuration of the two branches having the same CC.

A padding block is shown in the figure to append the proper sequence of bits to terminate all the encoders to the all-zero state. This is essential because turbo codes are used to produce block codes. If we have one encoder then the required tail is a sequence of bits with length equal to the memory order v. The problem of terminating both encoders simultaneously seems to be difficult because of the interleaver. However, it is still possible to do with v tail bits only [8].

In general another interleaver may be placed before the first encoder, but this has not been the practice. Usually a delay line is inserted before the first encoder to account for the interleaver delay and keep the branches properly synchronized.

Puncturing can be introduced to increase the rate of the convolutional code above that resulting from the basic structure of the encoder. Separate sections are devoted for interleaving and puncturing.

If one of the output sequences is exactly the input sequence, the code is said to be *systematic*; otherwise it is *non-systematic*. Clearly, the code in Figure 1.1 is systematic. On the other hand, if the state of the internal shift register depends on the past outputs the code is called *recursive*; otherwise it is *non-recursive*. In turbo codes, Recursive

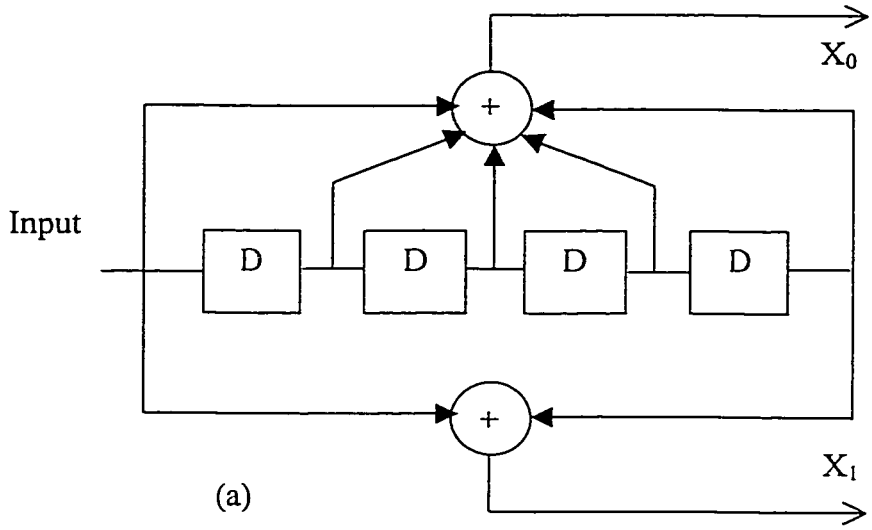
Systematic Convolutional (RSC) codes have been proved to perform better than the non-recursive ones [1, 9].

An RSC encoder can be obtained from a non-systematic non-recursive encoder by setting one of the outputs equal to the input (in case of a single input) and using a feedback. Figure 1.2 illustrates a non-recursive non-systematic convolutional code with its corresponding recursive systematic code. X_0 and X_1 are the output streams. The trellis structure and the free distance (d_{free}) are the same for both codes [1].

1.2.3 Turbo Decoder

The decoder works in an iterative way. Figure 1.3 shows a block diagram of a turbo decoder. The iteration stage is shown with doted lines to differentiate it from the initialization stage. The first decoder will decode the sequence and then pass the hard decision together with a reliability estimate of this decision to the next decoder after proper interleaving. For the first iteration, the decoder relies solely on the estimates coming from the channel. For all subsequent iterations, the decoder incorporates the reliability estimate coming from the second decoder output through the feedback connection. The second decoder will utilize the reliability estimates produced by the first encoder, and thus will have extra information for decoding.

The interleaver in-between is responsible for making the two decisions uncorrelated and the channel between the two decoders will seem to be memoryless due to interleaving. After a certain number of iterations is executed, a hard decision is made and the estimated sequence is delivered to the user. In practice the number of iterations does not exceed 18,



D: delay memory element

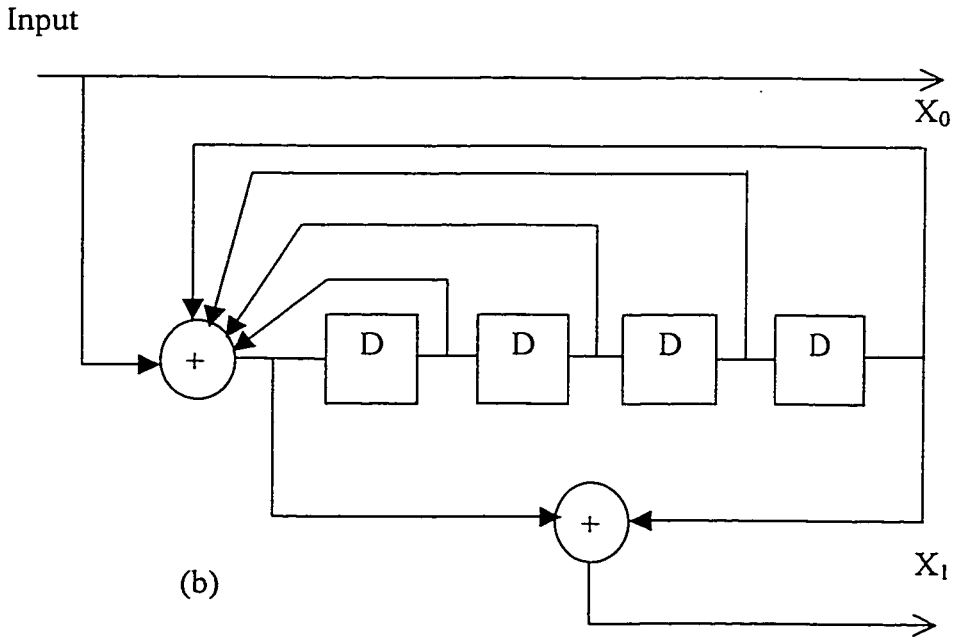


Figure 1.2: (a) Non-recursive non-systematic code

(b) Corresponding recursive systematic code

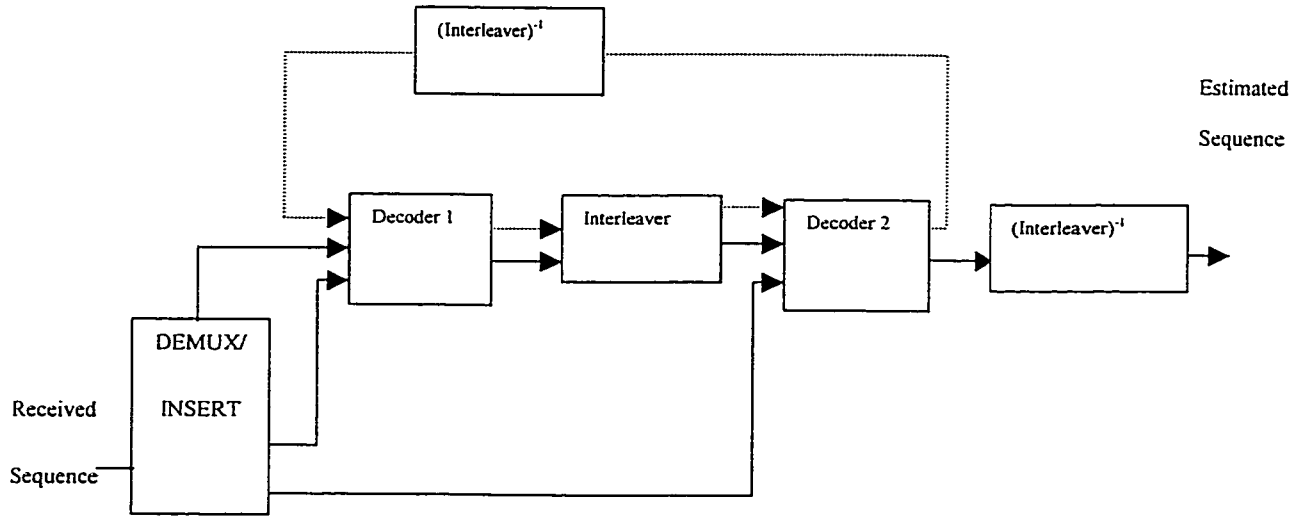


Figure 1.3: Block diagram of a turbo decoder

and in many cases 6 iterations were found to provide satisfactory performance [1]. In fact, the term "turbo" is given for this iterative decoder scheme with reference to the turbo engine principle.

The details of what information to pass to the next decoder or next iteration stage , i.e. the reliability measure, is subject to research. In the following, we describe a widely accepted decoding algorithm, which is the normalized soft output Viterbi algorithm.

Soft Output Viterbi Algorithm (SOVA)

Algorithms used in decoding convolutional codes can be modified to be used in decoding turbo codes. In the original paper on turbo codes [1], a modified Bahl *et al* algorithm was proposed for the decoding stage. This algorithm is based on **M**aximum **A**posteriori **P**robability (MAP). The problem with this algorithm is the inherent complexity and time delay. MAP algorithm was originally developed to minimize the bit-error probability instead of the sequence error probability. The algorithm, although optimal, seems less attractive due to the increased complexity. Using MAP with turbo codes is still debatable.

Viterbi algorithm is an optimal decoding method that minimizes the probability of sequence error for convolutional codes. A modified version of Viterbi algorithm, called SOVA (**S**oft **O**utput **V**iterbi **A**lgorithm), which uses soft outputs was introduced in [3, 5]. SOVA has only twice the complexity of Viterbi algorithm.

The key point in decoding is that every decoder will pass a reliability estimate together with the hard decision. Specifically, the decoder shall deliver for each symbol an estimate of the probability, p`, that this symbol has been incorrectly detected, that is,

$$p` = \text{Prob}\{\text{estimated symbol} \neq \text{sent symbol} \mid \text{received symbol}\} \quad (1.1)$$

The practical feasibility of turbo codes resides in the availability of such simple sub-optimal iterative decoding. Iterative SOVA output approaches maximum likelihood (ML) decoding performance bound by increasing the number of iterations [10]. However, the question of convergence is still waiting for a concrete answer.

An improved decoding with SOVA, called Normalized SOVA, that provides remedy to some problems of SOVA was proposed by [6]. In this work, we implemented the Normalized SOVA decoder.

Decoding details of turbo codes are out of the scope of this introductory chapter. An interested reader is referred to [3,5,6, 11-16] for further details.

1.3 Interleaving

An *interleaver* is a device that rearranges the ordering of a sequence of symbols in a deterministic manner according to a map. Associated with the interleaver is a *deinterleaver* that applies the inverse permutation to restore the original sequence.

Conventionally, interleaving has been used to spread out burst errors that are frequently encountered in many digital systems. For turbo codes, the interleaver has more functions.

According to [2] the most critical part in the design of a turbo code is the interleaver. The two main issues in the interleaver design are the interleaver size and the interleaver map. The size of the interleaver plays an important rule in the trade off between performance

and delay since both of them are directly proportional to the size. On the other hand, the map of the interleaver plays an important role in setting the code performance.

Undoubtfully, the interleaver has the most significant contribution to the superior performance of turbo codes. However, the mechanics by which the interleaver plays its crucial role is not yet well understood. Researchers offered various conjectures in their attempts to explain this issue. Consequently, every conjecture led to a particular line of thinking of how the interleaver should be designed.

According to [2], interleaving is used to feed the encoders with permutations so that the generated parity sequences can be assumed independent. This will exclude a number of interleavers which generate correlated sequences such as cyclic shifts.

For [2] and other researchers [10, 17], the key role of the interleaver is to shape the weight distribution of the code, which ultimately controls its performance. The interleaver in essence maps the parity sequence of the second encoder to that of the first encoder, and hence sets the weight of the complete codeword. So the aim of the designer is to produce (by manipulating the weights of the second parity sequence through interleaver mapping) whole codewords with the overall weights as large as possible [17]. Turbo codes, unlike convolutional codes, make the distribution of the weight more important than the minimum distance [10].

A third conjecture that we add is that it is the interleaver/deinterleaver pair at the decoder that is actually doing the good job. The interleaver/deinterleaver pair has to de-correlate

the decisions of the two constituent decoders for a good performance. As decoded bits manifest the highest correlation, a good interleaver must have a high scattering capability.

In fact it could be that all three features, randomization, weight shaping and scattering contribute to the performance of the code.

An issue that is worth considering in the design of the interleaver is the termination of the trellis of both convolutional encoders. By properly designing the map of the interleaver, it is possible to force the two encoders to the all-zero state with only v bits (where v is the memory length of the convolutional encoder assuming the same convolutional code is used in both encoders). To achieve this task a condition on the interleaver map was proposed in [8] and demonstrated in [18].

There have been many attempts to invoke the effects of the interleaver in the analysis of turbo codes. To overcome the difficulty of representing the interleaver map or the difficulty of enumerating all the permutations, the authors in [10] introduced an abstract interleaver called *uniform interleaver*, defined as follows:

A uniform interleaver of length k is a probabilistic device which maps a given input word of weight w into all distinct C_w^k permutations of it with equal probability of $1/C_w^k$.

The uniform interleaver cannot be used in practice since one is confronted with using a deterministic interleaver. However, it has been shown that for each value of signal-to-noise ratio, the performance obtained with the uniform interleaver is achievable by at least one deterministic interleaver [10].

The concept of uniform interleaving was further used by [9] and [19] in the design and evaluation of turbo codes. In [20] an asymptotic bound on the performance was given as a function of the interleaver length and some other code parameters.

In [10] it was shown that random interleavers offer a level of performance close to that of the uniform interleaver, independent, to a large extent, of the particular interleaver used. It was also shown that the beneficial effect of increasing the interleaver length tends to decrease at large k (interleaver length).

Jung and Naβhan [21] investigated the effects of different interleavers on the error performance for short frames which are typical for speech transmission in mobile radio applications. In contrast to large frame systems, some structured interleavers seem to be suitable for turbo code encoders in short frame transmission systems.

Dolinar and Divsalar [22] compared the difference in performance between random and nonrandom interleavers. The authors discussed a partial separation of the problem of picking good permutations and that of picking good component codes.

Khandani [23] proposed a technique for optimizing the interleaver using Hungarian method (linear sum assignment problem). Results show some improvement with respect to random interleaving.

Researchers are still working to develop design guidelines and to relate the interleaver parameters to the code performance.

1.4 Puncturing

Puncturing is the process of deleting some bits from the codeword according to a puncturing matrix. The puncturing matrix (**P**) consists of zeros and ones, where the zero represents an *omitted* bit while the one represents an *emitted* bit. It is usually used to increase the rate of a given code. Puncturing can be applied to both block and convolutional codes. *

Punctured convolutional codes were first introduced by Cain *et al* [24]. An example of a puncturing matrix that transforms a rate-1/3 code to a rate-1/2 code is given by

$$\mathbf{P} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{1.2}$$

The number of columns implies that the puncturing period p=2. The first row implies that the first branch (the systematic bits in case of turbo codes) is not punctured. The second

---

* In classical coding books, puncturing refers to the process of deleting parity bits while shortening refers to the process of deleting systematic bits. This distinction is common for block codes. Almost all convolutional codes that appear in literature are non-systematic codes, hence the distinction between puncturing and shortening is not applicable, and the former is usually used. Since turbo codes use systematic convolutional codes, one can argue about emphasizing the difference between the two operations. However, all the references on punctured turbo codes did not make this distinction, and therefore we decided to follow the same terminology.

and third rows indicate that one parity bit is selected for transmission from one of the encoders alternatively.

One of the main adv7antages of puncturing is that the same decoder may serve various codes of different rates obtained from puncturing the same mother code and thus allowing the same silicon product to be used in different applications. This technique find a lot of applications in adaptive rate and unequal protection systems [25, 26]. Of course, the decoder needs to know the puncturing table. When the redundant information of a given encoder is not transmitted, the corresponding decoder input is set to zero. This function is performed by the DEMUX/INSERTION block in the turbo decoder (see Figure 1.3). When the code is punctured, the branch metric corresponding to the punctured bits need not be computed.

Determining the best puncturing pattern for turbo codes is still an open problem. By "best" puncturing pattern we mean the puncturing matrix that results in the lowest error rate for the same number of punctured bits. [27] suggests that systematic bits should not be punctured (Berrous puncturing). The paper demonstrated, by simulation, that it is always better to avoid puncturing systematic bits. However, [28] suggested another puncturing pattern, named UKL and concluded that nonsystematic version of the designed punctured code is advantageous at high values of E_b/N_0. This result was confirmed over AWGN and fully interleaved flat Rayleigh fading channels.

One disadvantage of punctured codes is that error events at high rates can be quite large with large spans. The decision depth of punctured codes is generally longer than that of

unpunctured codes. This problem is not severe since in most applications data frames are transmitted with sync words and proper termination of short frames. [29]

Puncturing is a trade off between rate and performance, but, fortunately, punctured codes come with 0.1 or 0.2 dB of the optimum code (as reported for the convolutional codes)[30]. Convolutional codes and their punctured alternatives together with their difference in performance are tabulated [31][32].

In [33] it was observed that punctured turbo codes did not provide significant reduction in the frame error rate (FER) compared to a convolutional code of the same rate. The reason is that puncturing the component encoders to produce high-rate turbo codes drastically reduces the effective free distance of the code.

It is suggested that using unequal puncturing (puncturing the encoder outputs with different puncturing patterns) improves the performance [34]. In [26] Caire shows that the performance is not achieved by considering puncturing alone but the interleaver should be designed jointly. Caire then suggested to define the puncturing pattern on the interleaver map. Using adaptive puncturing for time-varying channels has been addressed in [25].

1.5 Thesis Contribution

The objective of this work is to contribute towards better understanding of turbo codes. The thesis investigates two main ingredients of the code, interleaving and puncturing:

Most researchers have assumed a uniform interleaver. Few researchers considered the cyclic interleaver, while others proposed an interleaver structure that helps in shaping the weight distribution of the code. Matrix interleavers receive much less attention and have

not been examined. We share thought with [21] that matrix interleavers are good candidates for turbo codes with short frames. They already possess the advantage of perfectly randomizing burst errors. In this work the performance of turbo codes with matrix interleaving is evaluated, and compared to the other types of interleavers mentioned above.

Some other issues related to interleaving are also investigated. These issues include the relative effect of interleaver size on the code performance, the number of effective iterations, the interaction between the interleaver and the deinterleaver and their contribution to the code performance and the effect of cyclically shifting an interleaver. The interleaver/deinterleaver issue was not investigated by post research, although it has valuable significance. Of prime concern to us was the interpretation of simulation results in an attempt to deepen the understanding of the interleaver role in the code performance.

This thesis also contributes towards the analysis of turbo codes under puncturing. By modifying the approach in [35], we were able to arrive at a new tighter upper bound on the bit error rate performance of the code. The modified approach also allows the evaluation of relative contribution of systematic branch and parity branches as well as investigating different puncturing patterns. We also provide an explanation for why systematic bits should not be punctured as observed by [1]. Some properties on the puncturing matrix are explored.

1.5.1 Thesis Organization

The next chapter, Chapter 2, is devoted for describing the techniques used in evaluating the performance of turbo codes, simulation and upper bounding. The upper bounds are evaluated using the weight distribution of the code. The procedures for estimating the weight distribution as introduced by the key papers in the field are reviewed.

In chapter 3, interleaving effects on the code performance is studied. The interleaver objectives and the roles it plays in turbo codes are outlined. Different classes of interleavers are defined and compared. A separate section is devoted for some study issues. These issues include interleaver length, number of iterations, interleaver/deinterleaver interaction , and cyclically shifted interleavers. The effects of these issues on the code performance are studied. We conclude Chapter 3 by examining and interpreting the two slope feature that is observed in the performance curves.

The effect of puncturing is discussed in Chapter 4. We start the chapter by deriving our new evaluation technique of the weight distribution of turbo codes. The effect of the new technique in tightening the upper bound is illustrated. The effect of puncturing systematic versus parity bits is discussed in the subsequent sections. Also the relative contribution of the different CCs is studied. Some guidelines on the puncturing matrix design are outlined. The chapter ends with presenting the properties of the puncturing matrix.

The findings and conclusions of this thesis are summarized in Chapter 5, where we also state our suggestions for direct extensions of this work that worth further research.

CHAPTER 2

PERFORMANCE EVALUATION OF TURBO CODES

2.1 Introduction

In this chapter we present the various techniques used for evaluating turbo codes. These techniques will be used in the coming chapters as means for judging the performance of the code under different interleaving and puncturing conditions.

Turbo codes are evaluated in terms of the post-decoding bit error probability P_b. Due to the random structure of the codes, there is no mathematical model for exact evaluation of P_b. Instead, the post-decoding bit error probability is evaluated in terms of upper bound or

by simulation. Both techniques have their limitations. Simulation, by default, yields the exact performance of the code. However, due to the huge amount of the required simulation, simulation is prohibitive to use at relatively large SNR, where one has to resort to bounding techniques. The typical bounding techniques used in evaluating the bit error rate of turbo codes are explained in the following section

2.2 Bounding Techniques

Intensive research was carried out to find performance bounds for turbo codes [9, 10, 36], but the inherent complexity of the code prevented general and tight results.

To analyze the turbo encoder, the effect of the decoder is de-coupled by assuming the availability of an optimum or near optimum decoder. The criteria for a good code is then based on the free distance and the weight distribution of the code. The free distance (d_{free}) is defined as the minimum Hamming weight of all possible codewords. The distribution of the codewords based on the Hamming weight will determine the code performance.

Since turbo codes are linear, they can be bounded by the union bound. The fact that a turbo code is linear is seen by recalling that every CC is linear and the interleaver performs a linear operation. Therefore the overall code is linear. The linearity of turbo codes is demonstrated in [10]. The union bound, though not sufficiently tight, can be used for comparing the performance of different turbo codes. It can also be used to evaluate the effect of changing certain code ingredients. The bound is good at high SNR where neglecting higher weights is less detrimental to the approximate bound.

Another performance bound is proposed by [19]. It is based on Gallager bound. This bound is a generalization of the union bound so it reduces to the union bound for a particular selection of parameters [19]. The bound is found to be tight for a large range of signal-to-noise ratios extending below the channel cutoff rate, which is the region where the standard union bound diverges. However, the Gallager bound requires more computation. The authors in [19] suggest using their bound in parallel with the union bound. If the result of the union bound is satisfactory, then there is no need to perform the calculations for the other bound. In our work we restrict ourselves to the union bound to avoid excess computation which might not end up with a converging result. The evaluation of the union bound requires finding the weight distribution of turbo codes. This is the topic of the next section.

2.3 Union Bound for Turbo Codes

An essential introduction to this section is the definition of some functions and parameters related to the CC. It is more illustrative to define those functions and parameters with reference to a particular CC.

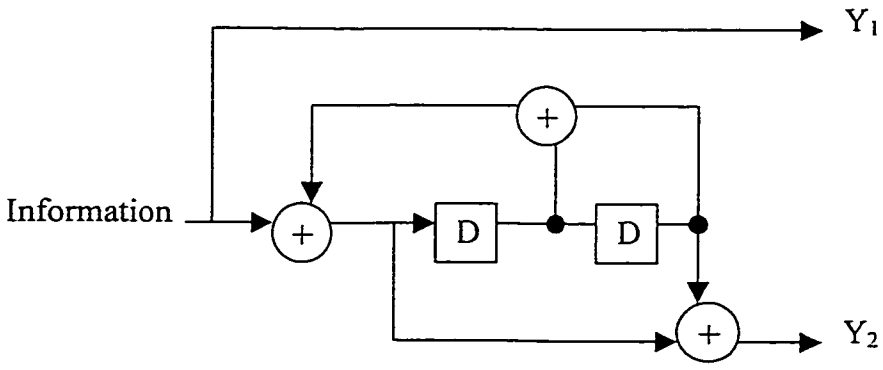
2.3.1 The Constituent Code

The constituent code, CC, is the basic building block of the turbo encoder. In this section we define the particular CC that will be used throughout this work. Many researchers have used the (1,7/5,7/5) or the (1,5/7,5/7) convolutional encoders as the CCs. The three entries describe the three branches of the code. The first entry, 1, indicates that the code is

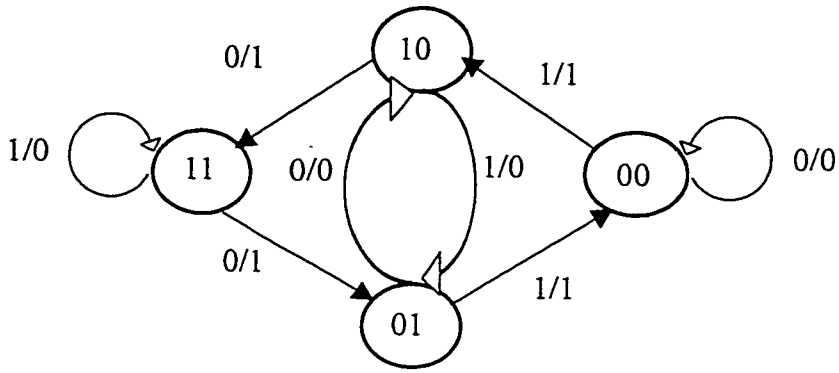
systematic. The second and third entries represent the ratio of the forward polynomial to the feedback polynomial, in octal format. The "5/7" CC is illustrated in Figure 2.1a. Note that the forward polynomial is $1+D^2$ (octal 5), whereas the feedback polynomial is $1+D+D^2$ (octal 7).

Throughout this work the (1,5/7,5/7) encoder will be used because it has an excellent performance compared to other encoders of the same complexity [32]. This is due the primitive feedback polynomial. However, the evaluation techniques discussed here are general, and can be applied to any other encoder.

The encoder of the CC can be represented in many ways. In Figure 2.1, the block diagram, the state diagram and the state transition matrix of the selected code are shown. The state diagram is used to enumerate all paths and their corresponding weights. For the state diagram shown in part (b), labels on the arrows represent the input/output weights of corresponding branch. In fact two output bits are produced for every input bit, but because the code is systematic, the output is taken to be the parity bit only. It is convenient to replace each edge label with the monomial $L^l W^w Z^z$, where L, W, and Z are dummy variables introduced to facilitate the evaluation of the length, input weight and parity weight, respectively, for every codeword. It follows that l is equal to "1" for every branch, while w and z are either 0 or 1 depending on whether the corresponding input and parity bits are 0 or 1, respectively. Using this notation, the information contained in the state diagram can be transformed to the transition matrix $\mathbf{A}(L,W,Z)$ shown in Figure 2.1c.



(a)



(b)

$$\mathbf{A}(L,W,Z)=\begin{pmatrix} L & 0 & LWZ & 0 \\ LWZ & 0 & L & 0 \\ 0 & LW & 0 & LZ \\ 0 & LZ & 0 & LW \end{pmatrix}$$

(c)

Figure 2.1: Different representations of the (1,5/7) encoder

(a) Block diagram (b) State diagram (c) Transition matrix

The *trellis diagram* shown in Figure 2.2 represents a more illustrative encoder description than that gained from the state diagram. In drawing the trellis diagram, a solid line denotes a transition generated by an input bit, "0", while a dashed line denotes a transition generated by an input bit, "1". The nodes of the trellis represent the encoder states. The parity bits appear as labels on the trellis branches

Transfer Function

The transfer function, also called the weight enumerating function is a useful mathematical representation of the CC. It describes all possible paths in the trellis in regard of their lengths, the weight of the systematic sequence (input) and the weight of the parity sequence. Starting from the transition matrix $\mathbf{A}(L,W,Z)$, the transfer function can be found as the (0,0) element of the following matrix [35]

$$(\mathbf{I}+\mathbf{A}(L,W,Z)+\mathbf{A}(L,W,Z)^2+\mathbf{A}(L,W,Z)^3+...)\mathbf{A}(1,W,Z)^v \qquad (2.1)$$

where the last term $\mathbf{A}(1,W,Z)^v$ accounts for the contribution of the padding sequence where no path length is accumulated. If the CC was non-recursive we could have set $W=1$ in the last term, since for non-recursive codes only zeroes are required to terminate the trellis and hence no input weight is accumulated. The first bracket in (2.1) can be evaluated using the relation:

$$\mathbf{I}+\mathbf{A}+\mathbf{A}^2+\mathbf{A}^3+...=(\mathbf{I}-\mathbf{A})^{-1} \qquad (2.2)$$

This approach requires the inversion of a matrix with three variables. If we are interested in the weight distribution for a frame of k bits, then it is sufficient to compute the matrix

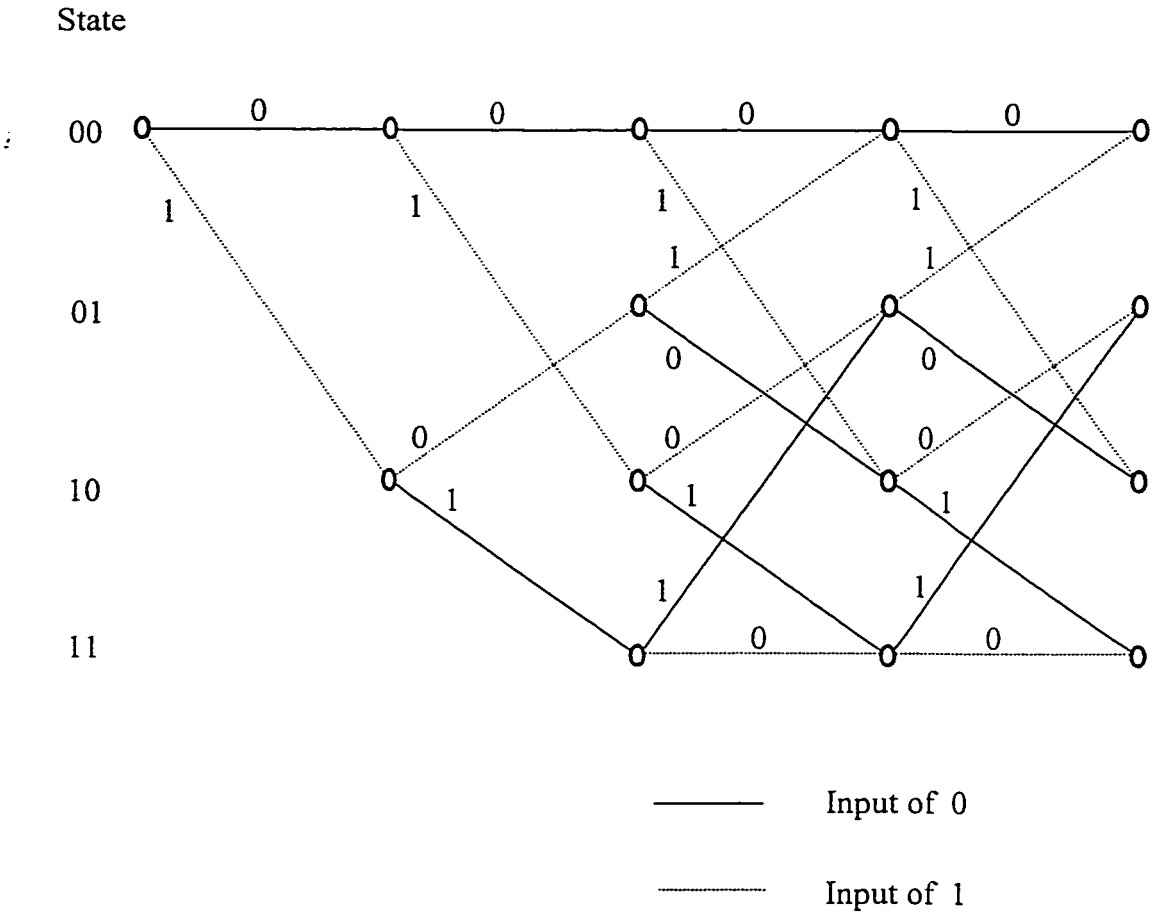


Figure 2.2: Trellis diagram for the 4-state (1,5/7) convolutional code

$$\mathrm{A}(L,W,Z)^{k-v}\,\mathrm{A}(1,W,Z)^{v} \tag{2.3}$$

The power of L is not required for weight enumeration. Therefore the desired transfer function may be found exclusively in the (0,0) element of the matrix

$$\mathrm{A}(W,Z)^{k} \tag{2.4}$$

2.3.2 Transfer Function of Turbo Codes

The transfer function of a turbo code, which will be used to find its weight distribution, is a function of the transfer functions of its constituent codes and the interleaver. [10] presents a good treatment of the subject, so we follow his approach.

To make the analysis tractable we will start by defining some of the repeatedly used terms, namely: *weight enumerating function* (WEF), *input redundancy weight enumerating function* (IRWEF), and *conditional input redundancy weight enumerating function* (CIRWEF) as in [10].

WEF

For an (n_c, k) systematic block code, we define the weight enumerating function (WEF) as the function that shows the distribution of the Hamming distance of the code. It shows the number of codewords with all available Hamming weights. The weight enumerating function $B^C(H)$ is represented mathematically by the summation:

$$B^C(H) = \sum_{i=0}^{n_c} B_i H^i \tag{2.5}$$

Note that H is a dummy variable with its power representing the weight and B_i is the number of codewords with Hamming weight i. For any practical code B_i is an integer number however B_i can be a fraction for some abstract interleavers like the uniform interleaver.

IRWEF

The input-redundancy weight enumerating function makes the contribution of the information to the Hamming weight explicit. It separates the total weight into the information weight and the parity weight. This separation makes it possible to find the bit error probability, whereas the WEF can be used to find the word error probability. The input-redundancy weight enumerating function $A^C(W,Z)$ is represented by the summation:

$$A^C(W,Z) = \sum_{w,j} A_{w,j} W^w Z^j \tag{2.6}$$

Where w is the information weight, j is the parity check weight. Therefore the overall output weight for systematic codes is $w+j$

W and Z: are dummy variables.

It is easily seen that:

$$B^C(H) = A^C(W = H, Z = H) \tag{2.7}$$

CIRWEF

The conditional input-redundancy weight enumerating function (CIRWEF) $A_w^C(Z)$ is similar to the IRWEF except that it is calculated only for a certain input Hamming weight (w). It is represented by the summation:

$$A_w^C(Z) = \sum_J A_{w,J} Z^J \tag{2.8}$$

The CIRWEF and the IRWEF are related as follows:

$$A_w^C(Z) = \frac{1}{w!} \frac{\partial^w A^C(W,Z)}{\partial W^w}\Big|_{W=0} \tag{2.9}$$

$$A^C(W,Z) = \sum_w W^w A_w^C(Z) \tag{2.10}$$

Our objective is to find the IRWEF for the parallel concatenation $A^{C_P}(W,Z)$ from the knowledge of the CCs and the interleaver. For a given interleaver the only way is exhaustive enumeration which is a lengthy process for large k. To overcome the difficulty of representing the interleaver map or the difficulty of enumerating all the permutations the authors in [10] introduced the uniform interleaver, defined in Chapter 1.

Utilizing the uniform interleaver the CIRWEF for the parallel concatenation is given by:

$$A_w^{C_P}(Z) = \frac{A_w^{C_1}(Z) \cdot A_w^{C_2}(Z)}{\binom{k}{w}} \tag{2.11}$$

Where $A_w^{C_1}(Z)$ and $A_w^{C_2}(Z)$ are the IRWEF for the first and second CCs.

The above derivation assumes that the codes are not punctured. In Chapter 4, we present a new method for invoking puncturing in the evaluation of the union bound for TCs.

For a code of rate R_c over AWGN channel, assuming maximum likelihood (ML) soft decoding, the bit error probability is upper bounded by [10]:

$$P_b(e) \le \frac{w}{k} \frac{\partial A^{C_P}(W,Z)}{\partial W}\bigg|_{W=Z=e^{\frac{R_c E_b}{N_0}}} \tag{2.12}$$

$$= \sum_{w=1}^{k} \frac{w}{k} W^w A_w^{C_P}(Z)\bigg|_{W=Z=e^{\frac{R_c E_b}{N_0}}} \tag{2.13}$$

$$= \sum_{m} D_m H^m \bigg|_{H=e^{\frac{R_c E_b}{N_0}}} \tag{2.14}$$

Where

$$D_m = \sum_{J+w=m} \frac{w}{k} A_{w,J} \tag{2.15}$$

Since turbo codes are most likely used with long frames, applying this bound usually requires the truncation of the summation. An important question raised by [10] is whether to truncate the summation at a certain information weight or at a certain overall weight? To answer this question let us examine the two cases. What makes the codewords apart from each other is the overall Hamming weight. Therefore, truncation should be related to the overall weight. Unfortunately, this requires the knowledge of the transfer function of the entire code. However, since the code is systematic the overall Hamming weight is

greater than the information weight. Therefore, if we truncate the summation at a certain information, say w^*, we will guarantee covering overall weights greater than or equal to w^*. For the evaluation of the union bound we evaluated the first 100 terms of D_m unless stated otherwise.

After some manipulation and simplifications the bit error probability can be represented as follows [10]:

$$P_b \approx \frac{1}{2}\sum_m D_m erfc\left(\sqrt{m\frac{R_c E_b}{N_0}}\right) \tag{2.16}$$

Equation (2.16) will be utilized in the subsequent chapters to serve as a mean to evaluate the effects of interleaving and puncturing on the overall performance.

2.4 Weight Distribution

By examining the weight distribution, also refereed to as distance spectrum by some researchers, one can judge the relative performance of different turbo codes. The dominant part of the distribution is the low weight codewords. This is why codewords resulting from low weight input sequences received a lot of attention [9, 10, 20, 22, 31]. In particular, the response of the turbo encoder to weight-two information sequences (frames) is believed, by some authors, to have a significant effect on the code performance. Note that because of padding there is no weight-one sequences, and weight-two sequences are therefore the smallest weight information sequences. In the following we study the contribution of the free distance to the overall performance.

2.4.1 Free Distance

The free distance, d_{free}, is the weight of the minimum Hamming weight codeword that diverges from the all-zero code word and re-merges with it for the first time. The free distance is the dominant part in determining the asymptotic performance of the code.

[11] has observed that the occurrence of minimum distance is quite rare (i.e. $B_{d_{free}}$ in Eq. 2.5 is small) and if it happens then the difference between the competing information Hamming weight is small. This means that for the asymptotic performance, we can truncate the summation after computing up to a certain information weight because we do not expect that a low Hamming weight output codeword will result from a high Hamming weight input sequence.

For a turbo code that is consisting of two convolutional codes, let z_{min} be the minimum parity-check weight and recall that w is the weight of the information sequence which is at least two because of trellis termination. For large SNR's, the performance is determined by the free distance, bounded as:

$$d_{free} \geq 2 + 2z_{\min} \tag{2.17}$$

This simply means that at least $2 \times z_{\min}$ check bits are generated using at least weight-two input sequence.

2.5 Simulation

Simulation is commonly used for estimating the performance of turbo codes. However, simulation requires extensive computation especially at high values of signal-to-noise ratios since the probability of error will be very low.

Simulation of turbo codes is not an easy task. It requires understanding of many concepts like concatenated coding, interleaving, puncturing and iterative decoding. Unfortunately, the details of the encoder and decoder are not clearly presented in the literature because many issues are still to be resolved [11].

For the decoding of turbo codes Bahl and Viterbi algorithms are commonly used. Both algorithms require huge amount of computational and processing time, though not to the same degree. The huge processing time is a consequence of the interleaver and the iterative process.

At any rate, the only way to give accurate estimation of the code performance at low probability of error ($P_b \leq 10^{-6}$) is through simulation. At smaller value of P_b the processing time becomes prohibitive.

Code Parameters Used in Simulation

Unless stated otherwise, we will be using the 5/7 code, unpunctured rate-1/3, normalized SOVA decoding with 6 iterations. The frame size is 192 bits which is suggested for speech transmission in a possible future mobile radio system. For wireless application

delay is limited to 20ms, corresponding to a block length of 192 at a bit rate of 9.6kbps [36].

A library of functions was programmed to simplify the simulation of turbo codes. The newly designed functions were programmed using the C language. Motivation for that are the speed and memory requirements.

CHAPTER 3

INTERLEAVING EFFECTS ON THE CODE PERFORMANCE

3.1 Introduction

Recall that "turbo code" is the name given to parallel concatenation with interleaving [27]. So the interleaver is a fundamental part of turbo codes, which distinguishes it from normal parallel-concatenated codes. According to [2] the most critical part in the design of a turbo code is the interleaver. It is also worth mentioning that the interleaver internally present in turbo codes helps randomizing burst errors and thus can prove beneficial for transmission over fading channels.

This chapter is devoted for studying the effect of interleaving on the performance of turbo codes. It is organized as follows: Section 3.2 outlines the interleaver objectives and its effects on the code performance. In the third section, different types of interleavers are defined and their relative performance is studied. Finally, the effects of some interleaver-related issues are presented and explained. These include interleaver length, number of iterations, interleaver/deinterleaver interaction, and cyclically shifted interleavers. In the last section we use the knowledge gained about weight distribution to explain the two-slope feature of the performance curves of the code.

3.2 Interleaver Objectives

Finding out an optimum interleaver for turbo codes (TCs) is one of the research challenges. Designing a good interleaver for a TC follows from understanding how the interleaver affects the performance of TCs. Such understanding is still far from maturity. There are many conjectures regarding the role of the interleaver in a TC. Our survey results in the following.

3.2.1 Scattering of Burst Errors

Conventionally, interleaving is used to spread out the errors occurring in bursts as a result of the channel. For concatenated coding, burst errors are also expected at the output of the **inner decoder**, applied last and removed first in concatenated coding. This is due to the inherent memory of the encoder that will correlate nearby symbols in time. At the decoder a faulty sample will accumulate in the path metric up to the end of the decision depth.

Though the effect of this faulty sample decreases gradually, it can still affect the nearby decisions.

We believe that this is particularly essential for turbo codes. It is conjectured that the interleaver that maximally scatters the symbols is the desired interleaver because of iterative decoding; the output of every decoder is interleaved or deinterleaved before being fed to the next decoder. If bits are not sufficiently scattered, we may not get the best of the iterative decoding. This issue is further investigated in 3.4.3.

3.2.2 Independent Parity

In TC, the interleaver is required to feed the encoders with different permutations of the information sequence so that the generated parity sequences can be assumed independent. The validity of this assumption is a function of the particular interleaver used. This will exclude a number of interleavers, which generate regular sequences such as interleavers based on cyclic shifting [2].

3.2.3 Shaping the Weight Distribution

Another conjecture is that the interleaver improves the performance of TCs via **shaping the weight distribution** of the code [2, 10, 17]. This is so because the interleaver decides which word of the second encoder is concatenated with the current word of the first encoder, and hence influences the weight of the complete codeword. Therefore the aim of the designer is to produce (by manipulating the weights of the second redundancy part through interleaver mapping) whole codewords with overall weights as large as

possible[17]. This can be achieved by making parity generation as diverse as possible. This results in increasing the minimum distance of the TC. However it should be stated at this time that increasing the minimum distance alone does not necessary lead to a good TC. Turbo codes, unlike convolutional codes, make the distribution of the weight more important than the minimum distance [10]. This is why we express the objective of the interleaver as that of shaping the weight distribution and not only maximizing the minimum distance.

Sub-objectives related to shaping the weight distribution can be summarized as follows:

- Breaking certain structures of low-weight input sequences. Low-input sequences, in particular weight-two input sequences, are the candidates for producing low-weight output sequences. A weight-two sequence, with the two "ones" separated by a certain distance will cause the output to diverge from the all-zero path and remerge with the all-zero path again after a short span. This particular weight-two input sequence is "disliked" because it results in a low-weight output. The objective of the interleaver set by some researchers [23] is that such "disliked" patterns should be broken by the interleaver such that they do not appear at the input of the 2nd encoder. Doing so, prevents producing low-weight sequences at the output of the two encoders simultaneously.

- Of approximately the same effect is to design the interleaver to provide a longer span between the 1's of the low-weight input sequences. When a path leaves the all-zero path it stays for a while before re-emerging with it again, resulting in a large output weight.

- If there are un-terminated constituent encoders, the interleaver should provide a large span form the end edge of the sequence of at least one un-terminated constituent encoder.

Trellis Termination

We conclude this section by few words regarding trellis termination. Trellis termination is essential when convolutional codes are used to encode blocks of bits. Since at the end of every block the state of the encoder will not necessary be zero, we need to append some bits to force the state of the encoder back to the all-zero state. The minimum number of bits required for this purpose equals to the memory order ν. For turbo codes, the problem gets more complicated since we require ν bits for every encoder. If we can make sure that the encoders are at the same state at the end of the information block then we can terminate all the encoders with only ν bits. Due to the presence of the interleaver, it is not possible to guarantee that the encoders will terminate at the same state unless we put some conditions on the interleavers as proposed in [8].

In our study, the interleaver objective of terminating both encoders with the same ν bits will not be considered. This is because this procedure will place a lot of restrictions on the design of the interleaver map. The problem gets more complicated and restricted if the code uses more than one interleaver. Fortunately, degradation in throughput due to terminating each encoder by appropriate ν bits and not using the same ν bits is not significant even for small blocks.

3.3 Classes of Interleavers

Interleavers can be classified in general into *block interleavers* and *convolutional interleavers*. This classification is much similar to the classification of codes. Convolutional interleavers, like convolutional code, are very suitable for continuous stream of information.

In block interleaving, a block of symbols is reordered according to a permutation function. Figure 3.1a illustrate the process for a block of 15 symbols. The variable x represents the original time index in the input sequence while $\pi(x)$ is the time index after interleaving. Since we are using turbo codes for encoding frames of information, we will restrict our self to block interleavers.

For the purpose of understanding the interleaver objectives for turbo codes, we have considered many types of block interleavers, namely: the random interleaver, the uniform interleaver, the cyclic interleaver, the matrix interleaver, the weight-two breaker, and the unity interleaver (no-interleaving). Performance of these interleavers will be presented in detail. The case of no-interleaving will be taken as a reference, which provides a measure of the improvement due to interleaving for different interleavers.

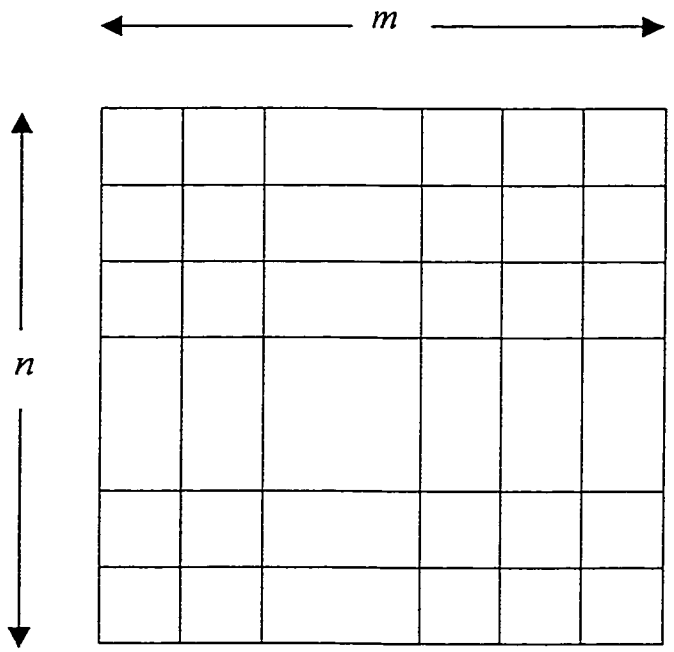
In random interleavers, shown in Figure 3.1a, the map of this interleaver is pseudo-random. In [10] it was shown through a bound that large-frame random interleavers offer performance close to the average ones, independent, to a large extent, of the

x	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi(x)$	4	8	12	7	11	15	10	14	3	13	2	6	1	5	9

(a)

x	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi(x)$	5	6	7	8	9	10	11	12	13	14	15	1	2	3	4

(b)



(c)

Figure 3.1: (a) Random interleaver

(b) Cyclic interleaver

(c) *(n,m)* matrix interleaver

particular interleaver used. Many random interleavers were tested by simulation and the conclusion was consistent. Based on the interleaver objectives, bad interleavers are easily avoided.

The uniform interleaver, defined in Section 1.3, though an abstract interleaver, can be approximated by changing the interleaver map at every frame. Statistically this requires a huge number of simulations in order to converge to the uniform interleaver. However, as will be shown later, there are sets of equally performing interleavers, which make the convergence process relatively fast. This interleaver is very important in testing the applicability of the union bound because it is designed based on the assumption of uniform interleaving. Many pseudo-random interleavers perform near the average (uniform).

Figure 3.2 shows the performance of TCs under different random interleavers. The figure also shows the performance of the uniform interleaver. The results clearly show that random interleavers perform near the average.

A cyclic interleaving is obtained by cyclically shifting the information sequence. If x is the location of the input symbol then the cyclic interleaver of length k will map this bit to the location $\pi(x)$ given by the following relation:

$$\pi(x) = (x + a) \bmod k \tag{3.1}$$

Where a is a constant representing the amount of shifting. Figure 3.1b shows a cyclic interleaver for a=4. Figure 3.3 shows the performance of cyclic interleavers of length 192 for different cyclic shifts a. It is clearly seen that cyclic interleavers are very poor.

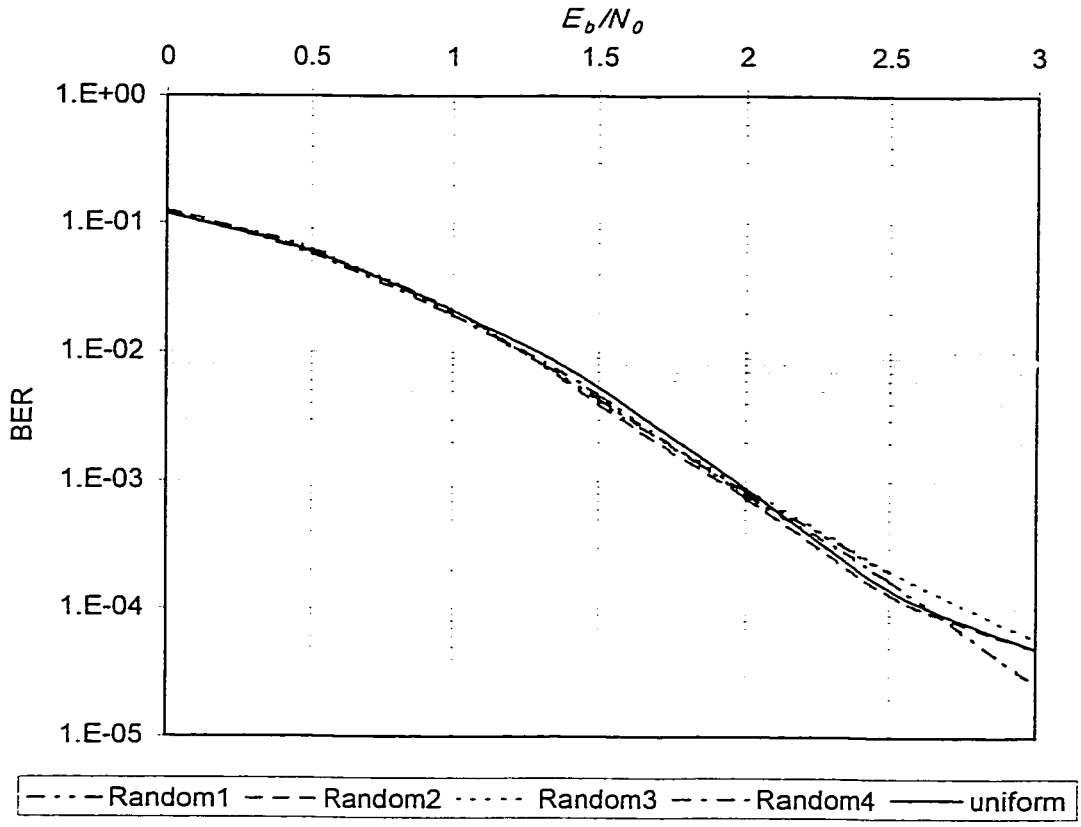


Figure 3.2: Performance of TCs under different random interleavers

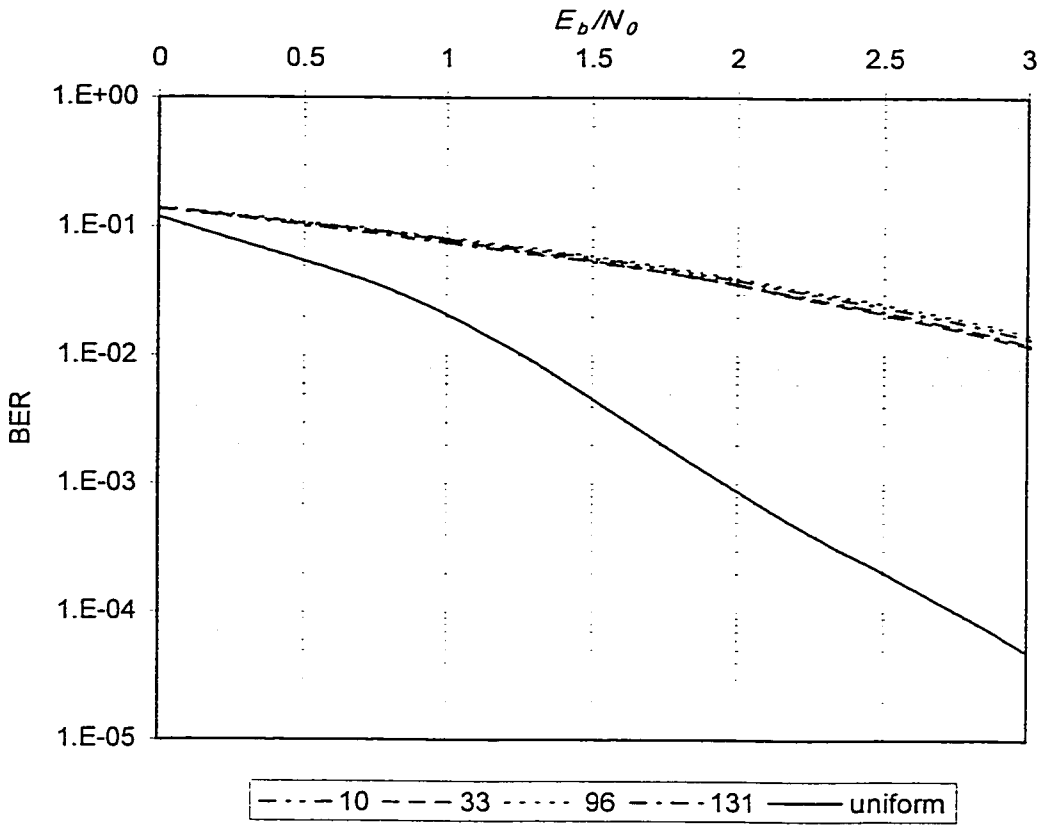


Figure 3.3: Performance of cyclic interleavers for different cyclic shifts (a)

In a very popular type of block interleavers, the *matrix interleaver*, the information stream is written row by row in a matrix of n rows and m columns and read out column by column. The column size n is called the *depth* and the row size m is the *span*. Such an interleaver is completely defined by n and m as illustrated in Figure 3.1c and is thus refereed to as (n,m) *matrix interleaver*. At the deinterleaver information is written column-wise and read out row-wise.

[23] proposes an interleaver that is designed to break certain weight-two frames. We will refer to the interleaver as weight-two breaker. This interleaver is very useful in evaluating the conjuncture that the interleaver is doing its job by breaking weight-two information sequences.

Separate subsections are devoted for studying the marix interleaver and the weight-two breaker.

3.3.1 Matrix Interleavers

The design of an (n,m) matrix interleaver involves the selection of the values of n and m. The map of the interleaver is not changed after that. For matrix interleavers any burst of errors of length $b \leq n$ results in single errors at the deinterleaver output each separated by at least m symbols. By recalling that matrix interleavers are very popular and posses the useful capability of randomizing burst errors, there is enough motivation to consider their use with TC.

Some researchers had categorized matrix interleavers as bad interleavers based on their behavior towards rectangular -shape errors. However, based on the actual performance of matrix interleavers observed by simulation, matrix interleavers are very efficient for short and medium size blocks. We therefore support, with confidence, the statement in [21]:

> "The application of block **(matrix) interleavers** can be regarded as **promising** in the turbo-encoder for the investigated short frame transmission system"

Selecting the Value of m &n.

Based on matrix interleavers, the next objective would be to find the optimum value for the number of rows *n* and the number of columns *m* for a given frame size. Recall that the capability of burst error scattering depends on the values of *n* and *m*. Since both the interleaver and the deinterleaver are utilized iteratively at the decoder, the optimum design is where *m* and *n* are both high in value. Consider an interleaver size of 192. Note that 192 can be factored as follows:

192=2*2*2*2*2*2*3=(192*1),(1*192),(96*2),(2*96),(48*4),(4*48),(32*6),(6*32),
(24*8),(8*24).....

Figure 3.4 shows the performance of a matrix interleaver at different dimensions. An *(n,m)* matrix interleaver, that is close to a square pattern (i.e. *n* and *m* are comparable), performs very well as compared to the theoretical uniform interleaver for short and medium frame lengths.

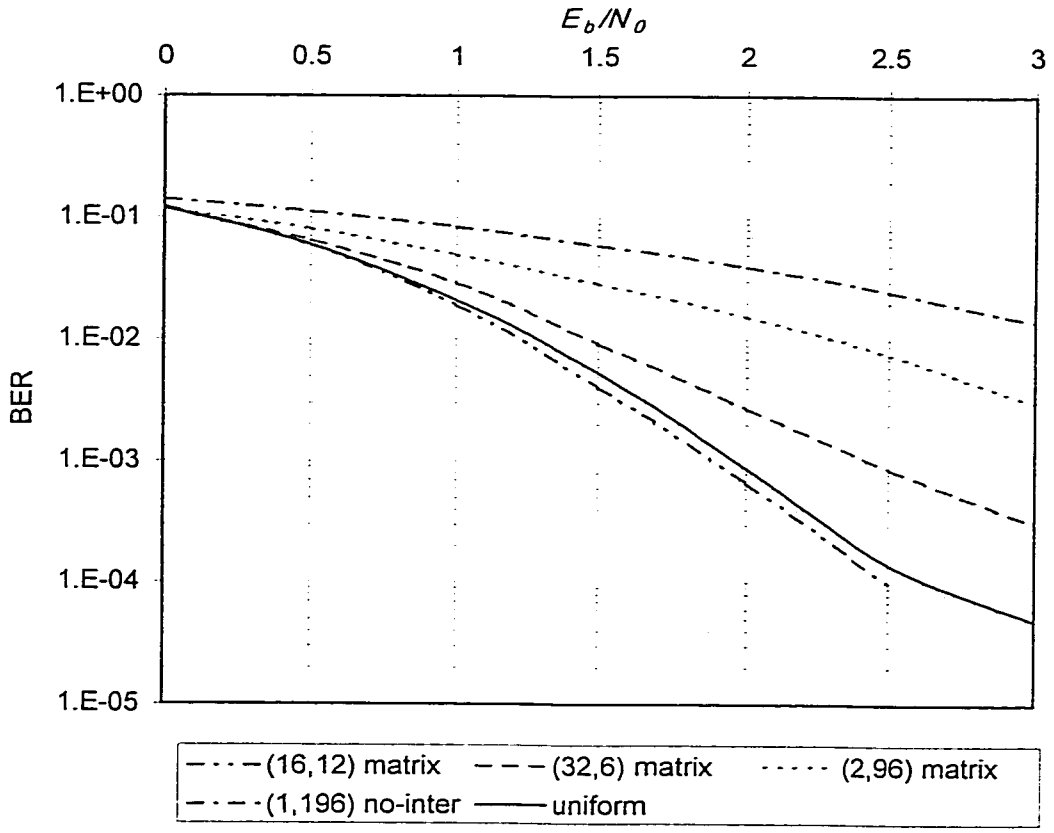


Figure 3.4: Performance of matrix interleavers for different values of m and n.

(i=6,k=192)

In Figure 3.4 some cases are not shown, for example, n=12 and m= 16, since the (12,16) matrix is the deinterleaver for the (16,12) matrix interleaver, their performance is almost the same as will be demonstrated in Section 3.4.3.

We anticipate that for large frames the performance of the matrix interleaver will improve as we approach the square matrix until we reach to a point where the value of m and n is sufficient to break the correlated symbols. This will occur when both m and n are greater than the decision depth.

3.3.2 Weight-Two Breaker

Before explaining the weight-two breaker, we ask the following question:

Do we really need to think only about weight distribution when looking for a good interleaver? First it should be clear that improving (increasing) all codewords weights is not possible. All what we can do is just changing the concentration of the weight distribution a round a certain value. Consider the sketch in Figure 3.5 with horizontal axis as the weight of the codeword and the vertical axis as the probability density function. The weight distribution can look like 1 & 2 but we can never get something like 3 because plot 3 contradicts the linearity of the code. This can be explained as follows: as the number of codewords with high weights increases, the number of codewords with low weight increases also. This is because the linear sum of the high weight codewords is expected to be of low weight.

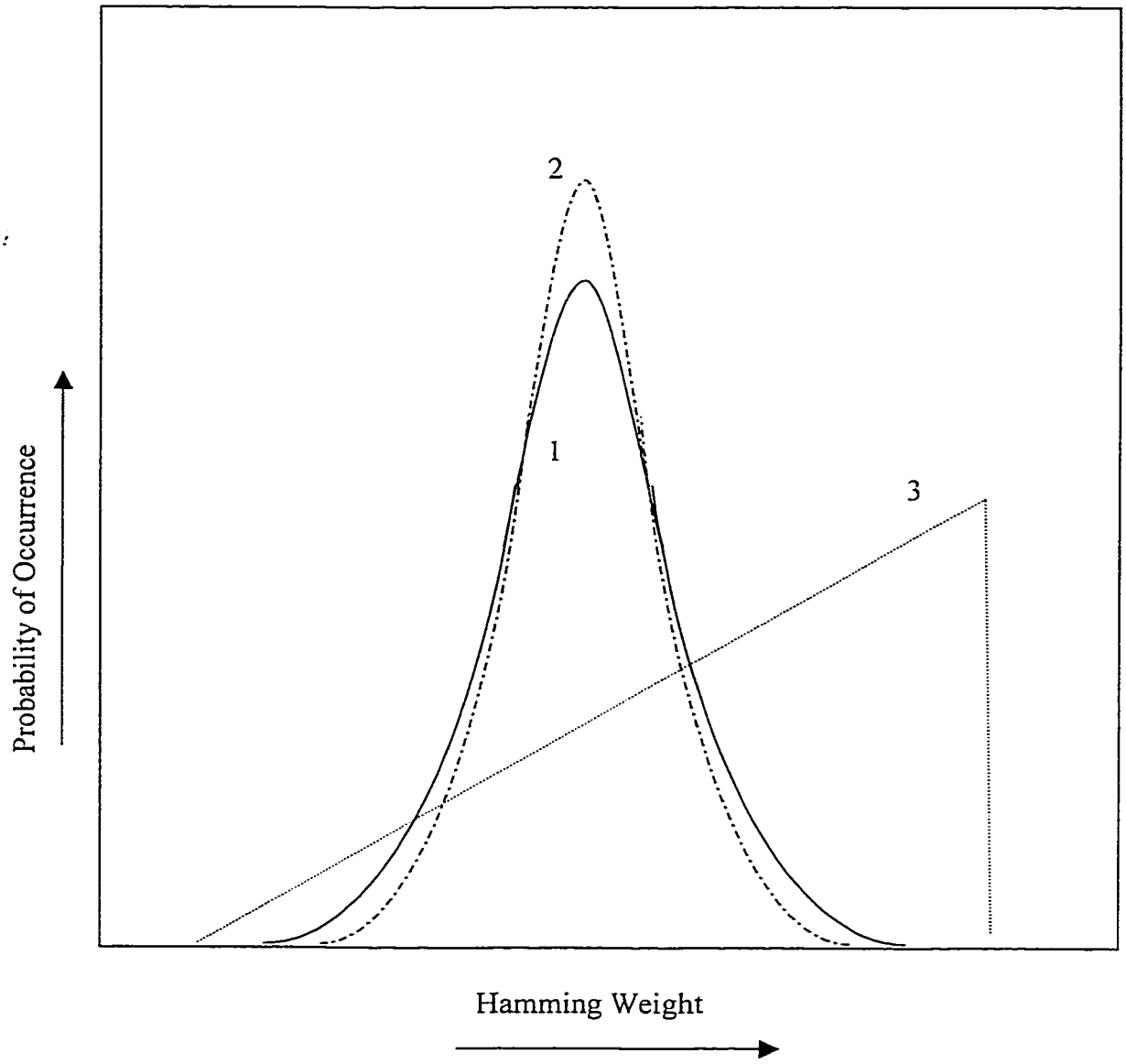


Figure 3.5: Theoretical weight distributions

The above argument does not mean that it is easy to shape the weight distribution. In fact considering the entire distribution seems to be a formidable task. Even if we get an optimal interleaver that gives the required weight distribution, it would be optimal for one value of E_b/N_0, (refer to Eq. 2.16). If the value of E_b/N_0 changed the performance will degrade accordingly. Our only option is then to concentrate on the low weight codewords since they are the most dominant terms in the evaluation of the bound. Our task will be two-fold, to increase the weight of low-weight codewords and reduce their multiplicity, if possible. This is exactly the motivation behind low-weight input frames.

Design of interleavers usually concentrated around the task of breaking weight-two information sequences because they are the lowest-weight sequences[23]. Since the RSC encoder is terminated, weight one blocks do not exist. An RSC can be driven out of the all-zero state and back to it with a minimum of two ones that are suitably spaced. An "unfortunate" mapping occurs if the self-terminating weight two sequence with short lifetime (output Hamming weight) is interleaved to another self-terminating sequence with short lifetime. This event results in a small overall Hamming weight. The case is illustrated in Figure 3.6.

Figure 3.6 shows a weight-two input sequence. Assume that two 1's separated with four 0's result in a self-terminating sequence. The second block is an example of an unfortunate mapping. The encoder will go back to the all-zero state and no weight will be accumulated. This is why the overall Hamming weight is expected to be low.

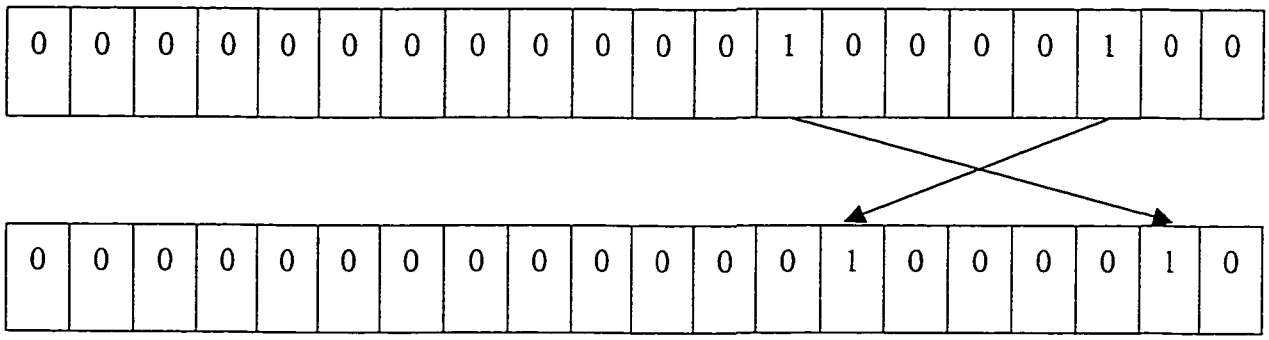


Figure 3.6: Illustration of an 'unfortunate' weight-two mapping

[23] proposed an optimal interleaver with respect to the task of breaking weight two sequences. The author proved the following two statements. First, there exists an interleaver that breaks all weight-two sequences with a span (i.e. number of zeros in between) limited to p_c^2, where p_c is the period of the code (the impulse response of RSCs used in turbo codes is periodic with period $p_c = 2^r - 1$, where r is the constraint length of the code [23]). Second, there does not exist an interleaver which breaks all weight-two sequences of larger span.

In practice $k > p_c^2$ and hence the interleaver will not be able to break all weight-two sequences. The interleaver should be designed such that the span of each unbroken sequence is maximized, thus allowing the Hamming weight of its corresponding parity sequence to accumulate.

The structure of such interleaver is very simple. It is sufficient to partition the input block into sub-blocks of length p_c and apply a cyclic shift of i positions ($i=0,1,2,3,..$) to the elements of the ith sub-block. The effective number of cyclic shifts is (i mod p_c). After p_c sub-blocks, we come back to our original point where no shift is applied. [23] called this interleaver uniform interleaver. To avoid the confusion with the pre-defined uniform interleaver, we will call this interleaver *weight-two breaker* (W2B). [23] finds that this interleaver gives a poor performance which means that it is not sufficient to consider weight-two input sequences only.

In our study, W2B interleaver is implemented so that its performance can be compared to other interleavers. Figure 3.7 shows the performance of the W2B compared to the (12,16) matrix interleaver and the no-interleaving case. Figure 3.7 supports the conclusion that W2B interleaver has a poor performance. In fact, it adds very little to the no-interleaving case. We even found that the performance of W2B with 6 iterations is worse than the performance of (16,12) matrix interleaver with only one iteration. Hence, it is not sufficient to consider the effect of weight-two information sequences only. The performance of a TC is strongly related to the other input sequences with higher Hamming weights. Note that the structure of weight-two breaker has a low spreading capability.

We conclude this section by showing a comparison between the five types of interleavers that have been discussed. For each of the random, cyclic and matrix interleavers, we have shown the best tested one among its group. Figure 3.8 shows that the random and matrix interleavers compete with each other and offer a performance that is comparable to the abstract uniform interleaver, while cyclic and W2B interleavers are very poor.

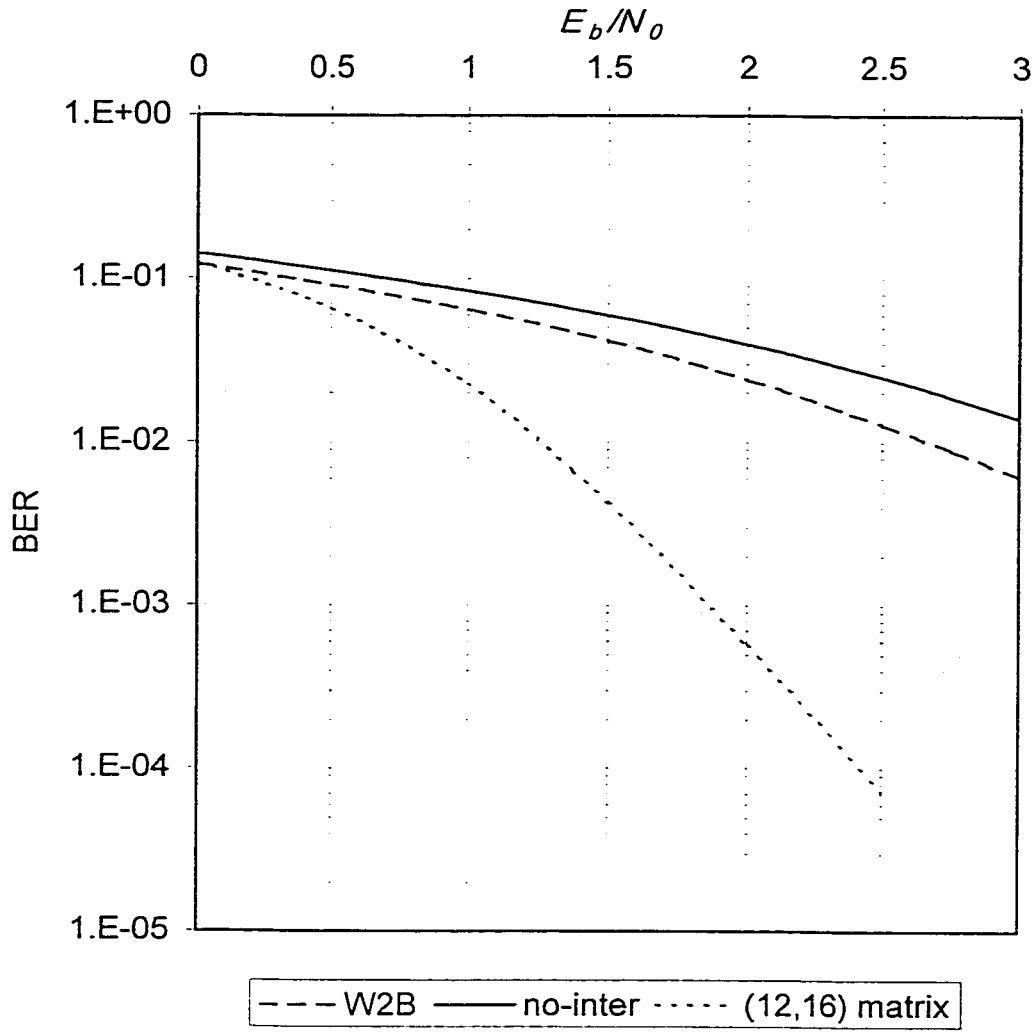


Figure 3.7: Performance of weight-two breaker

(i=6, k=192)

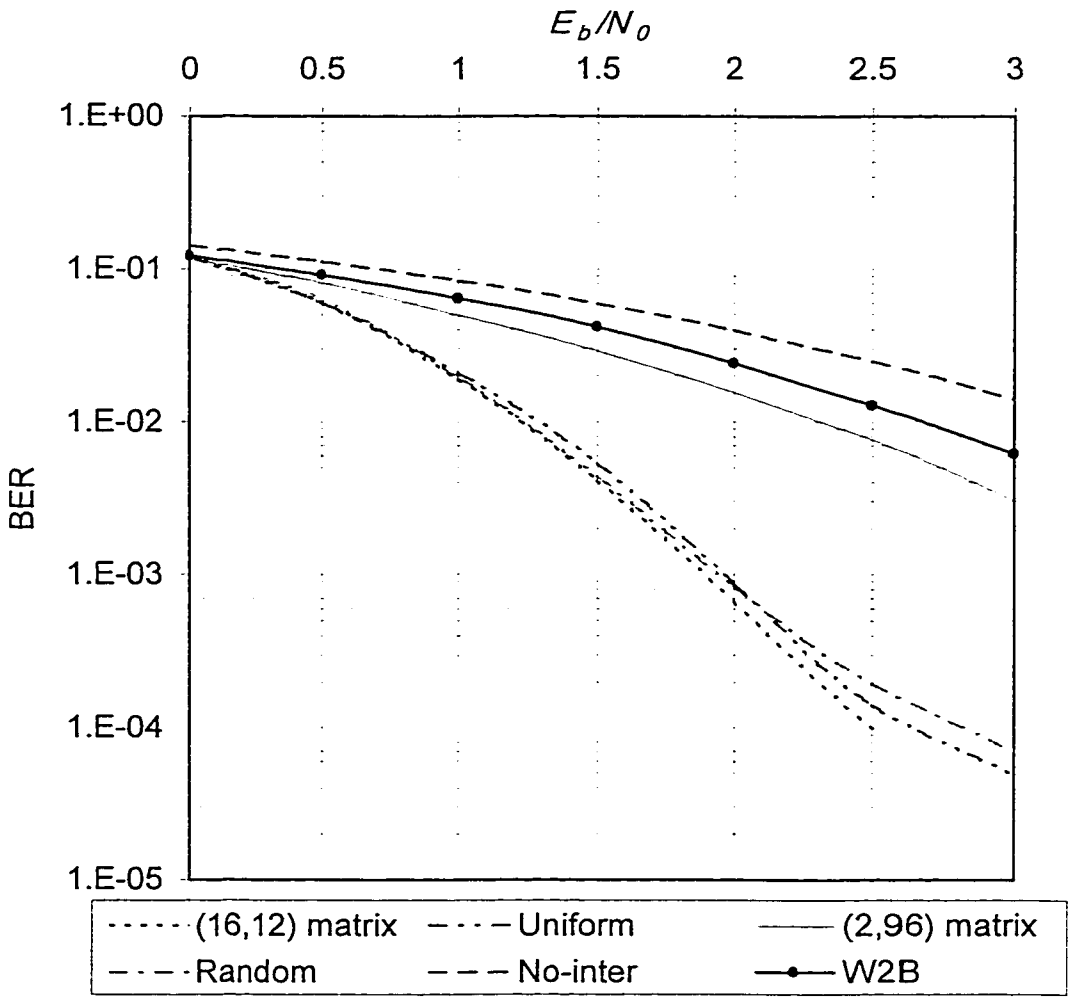


Figure 3.8: Performance of different interleaver classes

(i=6, k=192)

3.4 Study Issues

In this section, some interesting issues related to the interleaver are studied. These issues include the interleaver length, the effect of the number of iterations, the interleaver/deinterleaver contribution, and the cyclically shifted interleavers.

3.4.1 Interleaver Length

Originally [1], turbo codes were proposed with very large frames. However, many applications require shorter frames. Using TCs with very short frames does not seem worthwhile [11]. As we increase the frame length k, the performance improves, then saturates for very large k. Figure 3.9 shows the effects of increasing the size for the case of uniform interleaving.

The effect of interleaver length should be considered in conjunction with the memory span of the CCs. The interleaver length k should be much greater than the decision depth. The interleaver length and the memory of the CC have to be traded off according to the system requirements. Increasing the interleaver length k increases the decoding delay at almost no expense in complexity, whereas, increasing the memory increases the complexity with only a slight increase in the decoding delay.

The previous conclusion, regarding the relation between performance and k, found in the literature was based on the theoretical uniform interleaver. For a uniform interleaver increasing the size improves the performance until saturation is reached. This is because for a uniform interleaver increasing the length makes the probability of

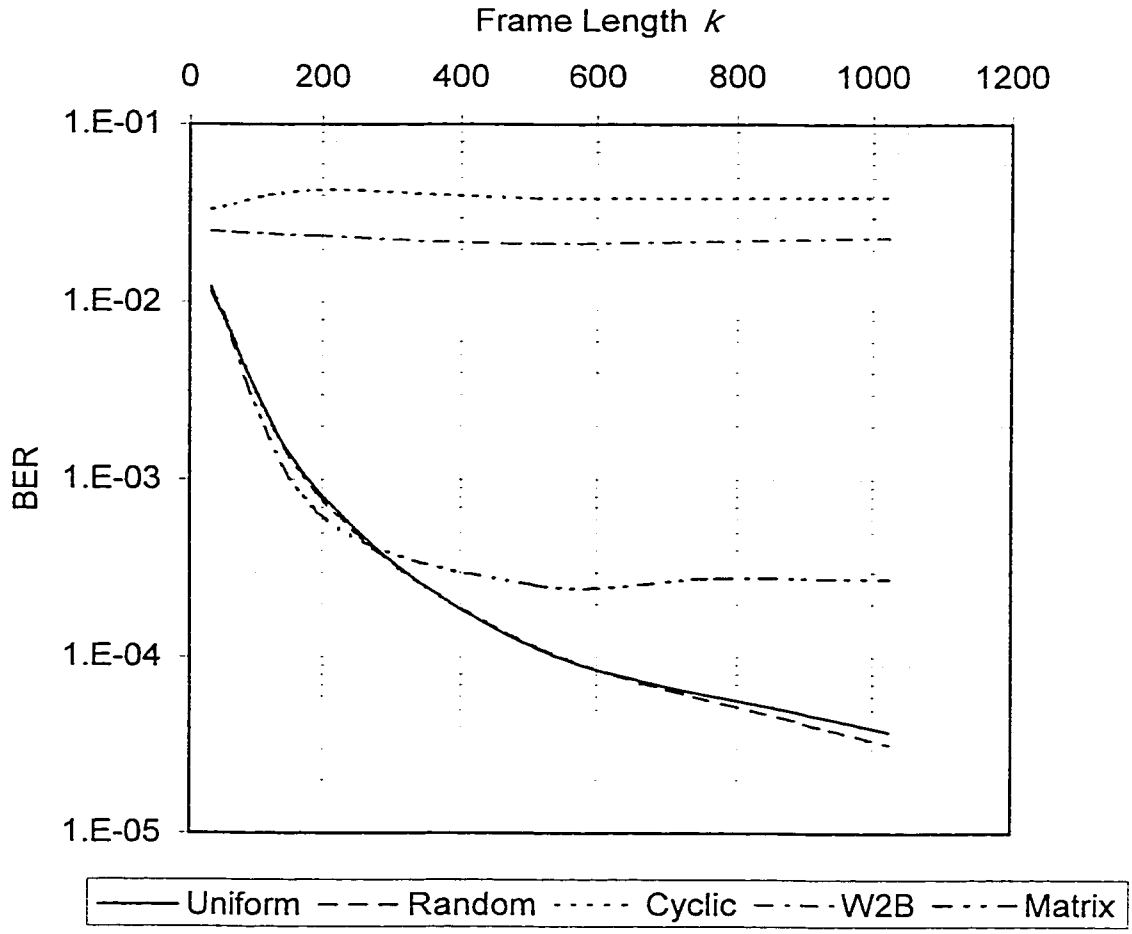


Figure 3.9: Effects increasing the frame size k on the code performance

(i=6, uniform interleaver, E_b/N_0=2 dB)

scattering the estimates at the output of the decoder higher and therefore correlated estimates can be efficiently de-correlated.

Presenting the BER vs. k at a given E_b/N_0 (as in Figure 3.9) did not appear in the literature before. It is very informative as it verifies that the BER decreases as $1/k$ [10].

We would like to add here that this conclusion holds if the permutation function is directly related to the size of the interleaver as in pseudo-random interleavers. However, this conclusion is not true for interleavers based on cyclic shifting, W2B and some other structured interleavers. Consider the case of cyclic interleaver. Doubling the interleaver size can be thought of as augmenting two blocks of the original size, resulting in no improvement. Even worse, The original system might perform better due to trellis termination where we are sure about the middle values.

This point is illustrated in Figure 3.10 which is obtained for two interleavers of length 192 and 512. Both interleavers perform cyclic shifting of 96 bits i.e.

$$\pi(x) = (x + 96) \bmod k \tag{3.2}$$

This clearly proves that the relation between performance and k is coming from the embedded relation between k and the scattering capability of the interleaver.

The fact that there are interleavers that do not perform better with larger frames tells us that not all interleavers improve as $1/k$. One has to consider of the specific interleaver under use.

Another result on the interleaver size is that improvements are proportional to the number of iterations. In other words, the improvement due to increasing the interleaver size will

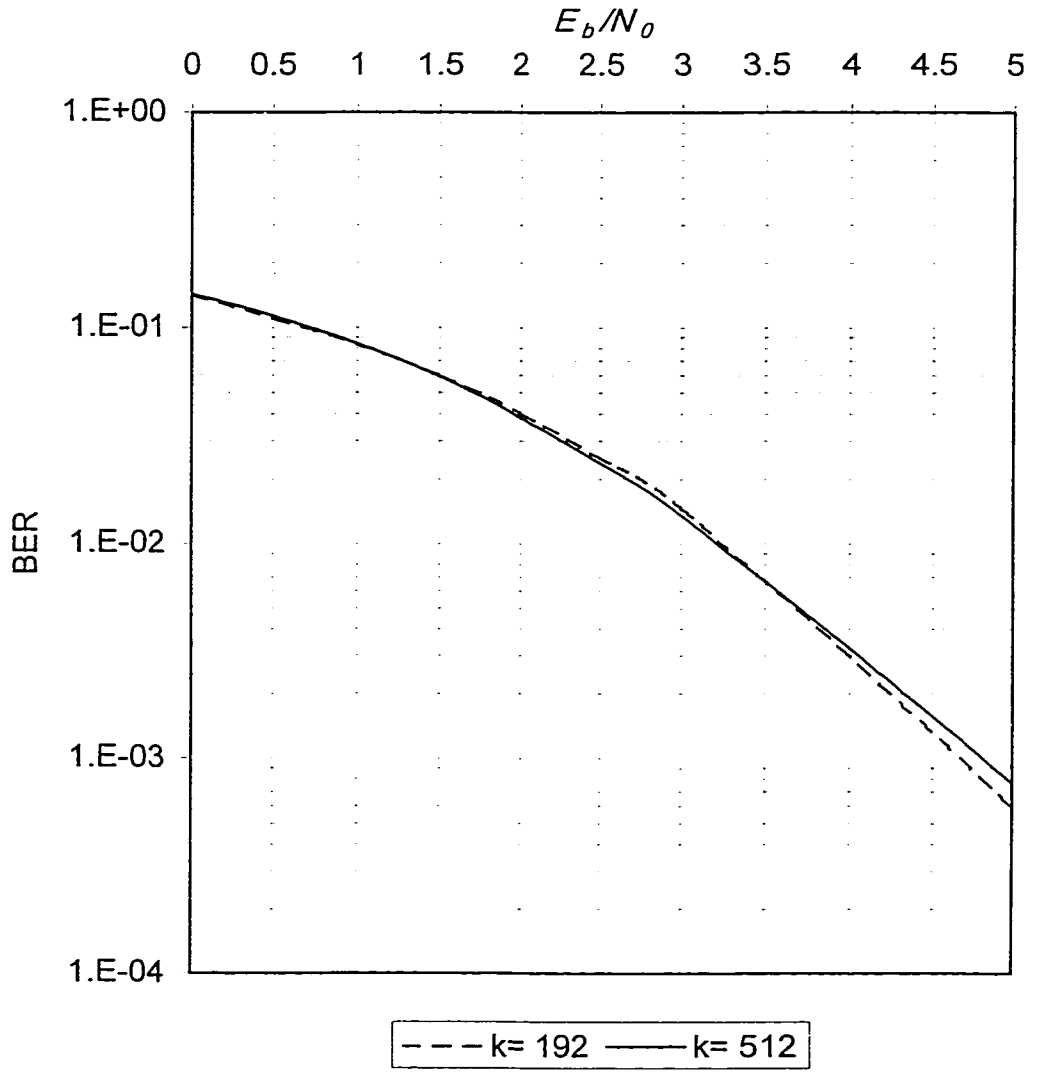


Figure 3.10: Relation between BER and k does not hold for all interleavers

(i=6, cyclic interleaver)

only be achieved for sufficiently large number of decoding iterations. Figure 3.11 shows the performance of the interleaver as a function of the size for different number of iterations. For a single iteration there is eventually no effect of increasing the frame size. As we increase the number of iterations we are increasing the utilization of the interleaver/deinterleaver and hence the benefit of using large frames is more evident. This very important result supports our conjecture that a significant role of the interleaver is played at the decoder. The role of the interleaver at the decoder will be investigated in depth is Section 3.4.3.

In conclusion, increasing the interleaver size improves the code performance provided that the interleaver has a good scattering/randomization capability, and the number of iterations is large enough to exploit the increase in size.

3.4.2 Number of Iterations

A common conclusion by past researchers is that increasing the number of iterations (to some extent) reduces the BER significantly. Some of them reported the value i=6 to be the best compromise between quality and delay because no significant reduction in BER is achieved beyond i=6. Their conclusions were based on the uniform and random interleavers. Hence, we examine these results for other types of interleavers.

First of all, it is emphasized that there is a mutual benefit between the interleaver and the iterative decoder. A good interleaver is not advantageous without sufficient number of iterations, and increasing the iterations does not do much without good interleaving. This mutual benefit is illustrated in Figure 3.12 which shows the performance of TC without

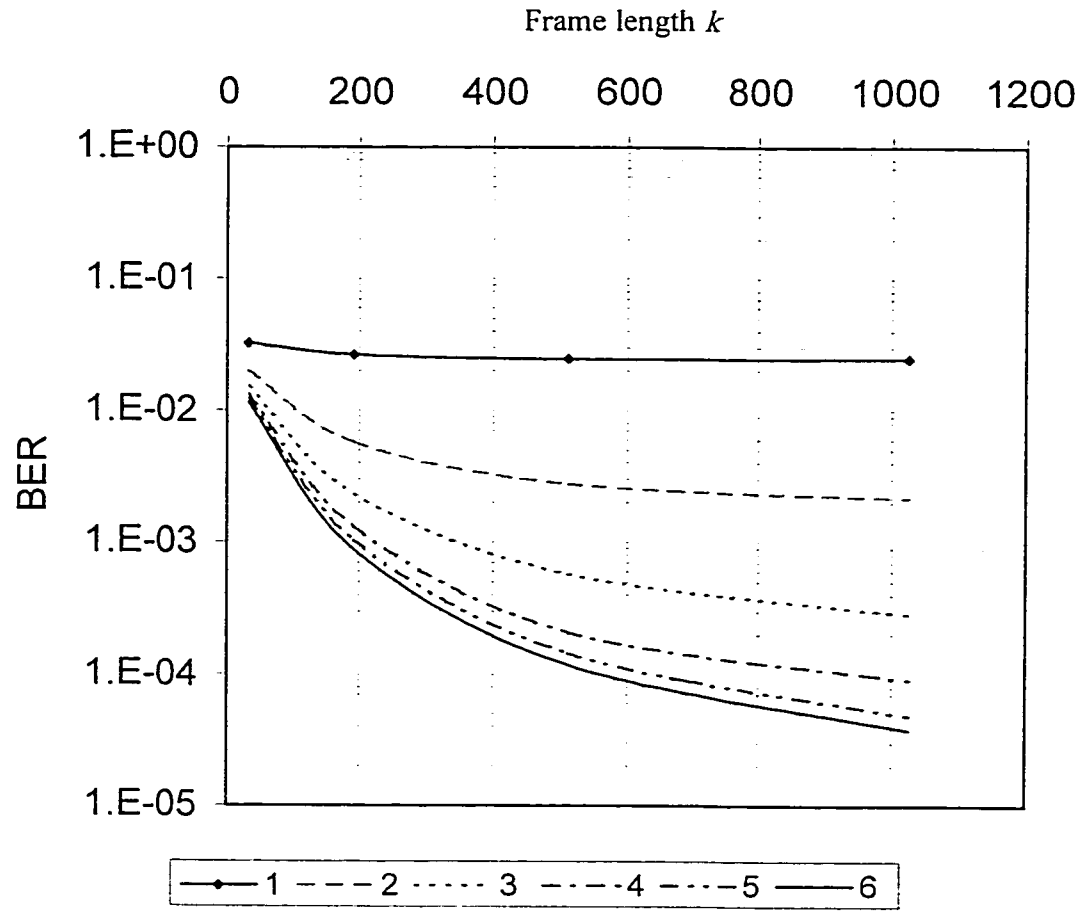


Figure 3.11: Effect of frame size k for different number of iterations.

(i=1,2,3,4,5 & 6, Uniform interleaver, E_b/N_0=2 dB)

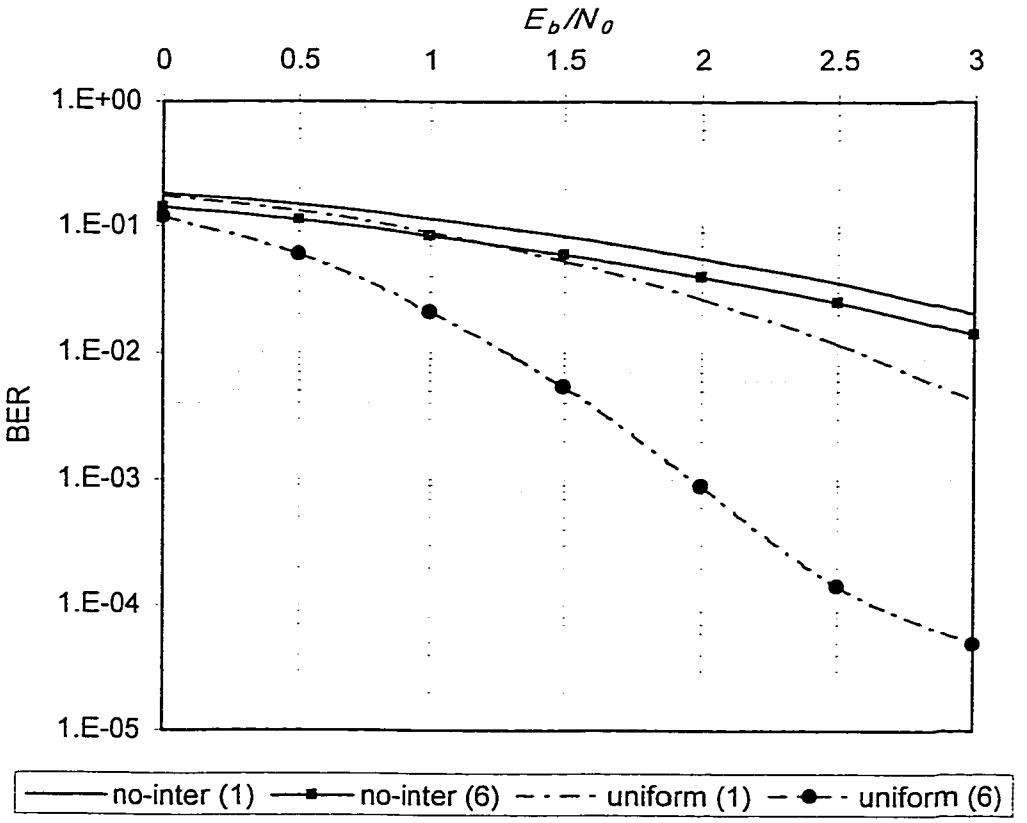


Figure 3.12: Relation between number of iterations and the interleaver

interleaving, and with uniform interleaver for i=1 and 6.

Our point is demonstrated by observing that the performance of uniform(6) is much superior than that of no-inter(6) and uniform(1). This observation threw the clue that the gain obtained from increasing i depends on the interleaver type, or more accurately, is proportional to the scattering/randomization capability of the interleaver.

The matrix interleaver allows us to examine this conjecture because we can design it for different scattering power. Figure 3.13 shows the performance of the matrix interleaver for different dimensions, all for i=1 and 6.

For the completeness of the results, we show the improvement in performance from i=1 to i=6 for the W2B and cyclic interleaver Figure 3.14. The results are quite consistent with our previous conclusion; that is, as these interleavers have low scattering capability, they did not help the code to benefit from the increased iterations.

The last investigation related to this section is the number of effective iterations i^*. Figure 3.15a shows the performance of the TC with (16,12) matrix interleaver for different number of iterations. It is clearly seen that i=6 is a good design choice for matrix interleavers as well.

We increased the interleaver length to see whether it has an effect on i^*. Figure 3.15b show the performance of the code using (32,24) matrix interleaver for different number of iterations. There is still no significant improvement beyond i=6 again, which suggests that increasing k does not affect i^*.

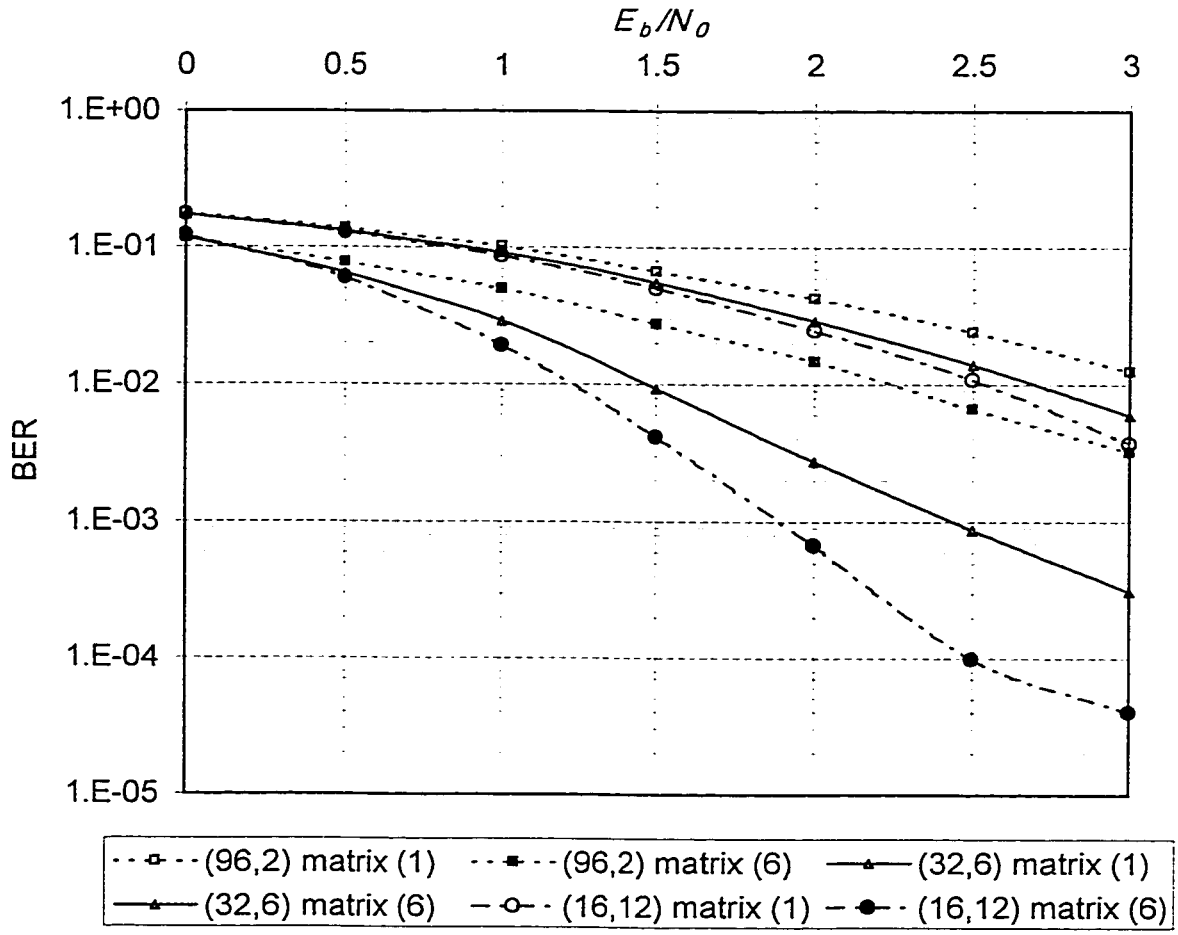


Figure 3.13: Effect of the number of iterations for different scattering powers

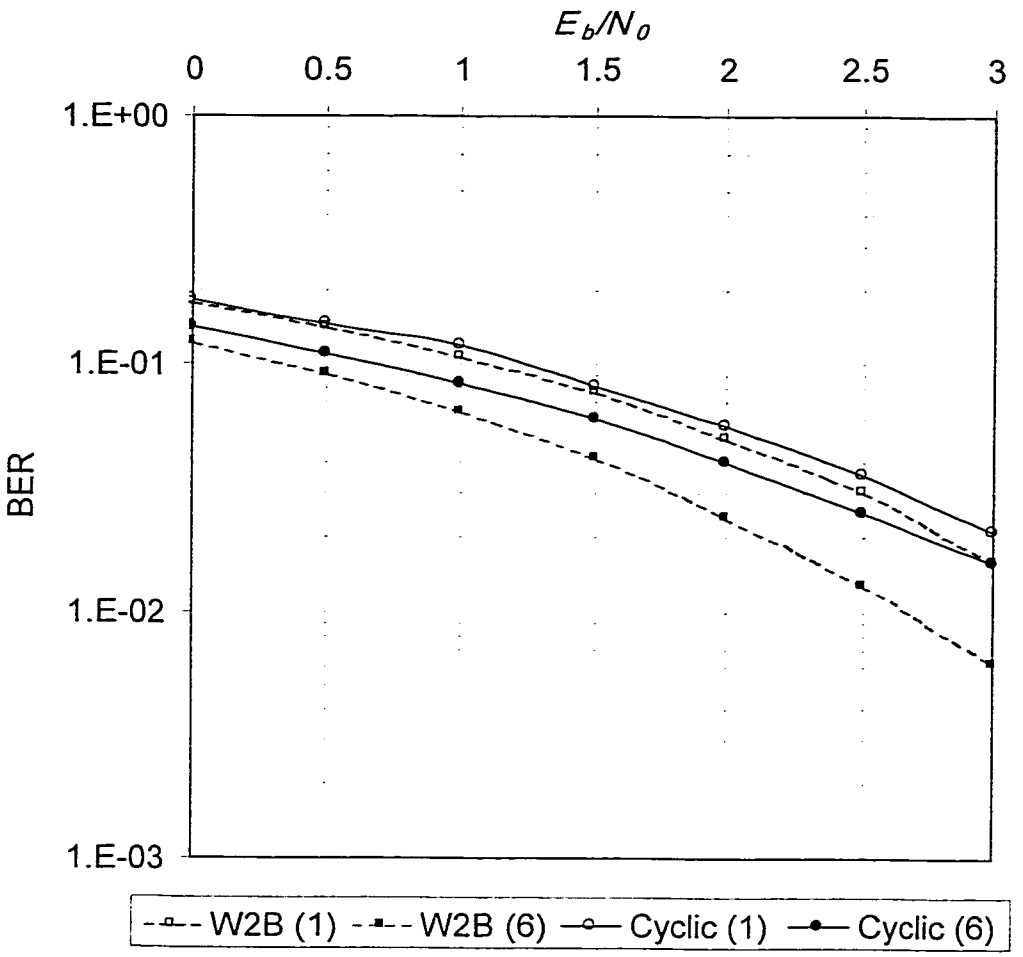


Figure 3.14: Effect of number of iterations on the W2B and a random interleaver

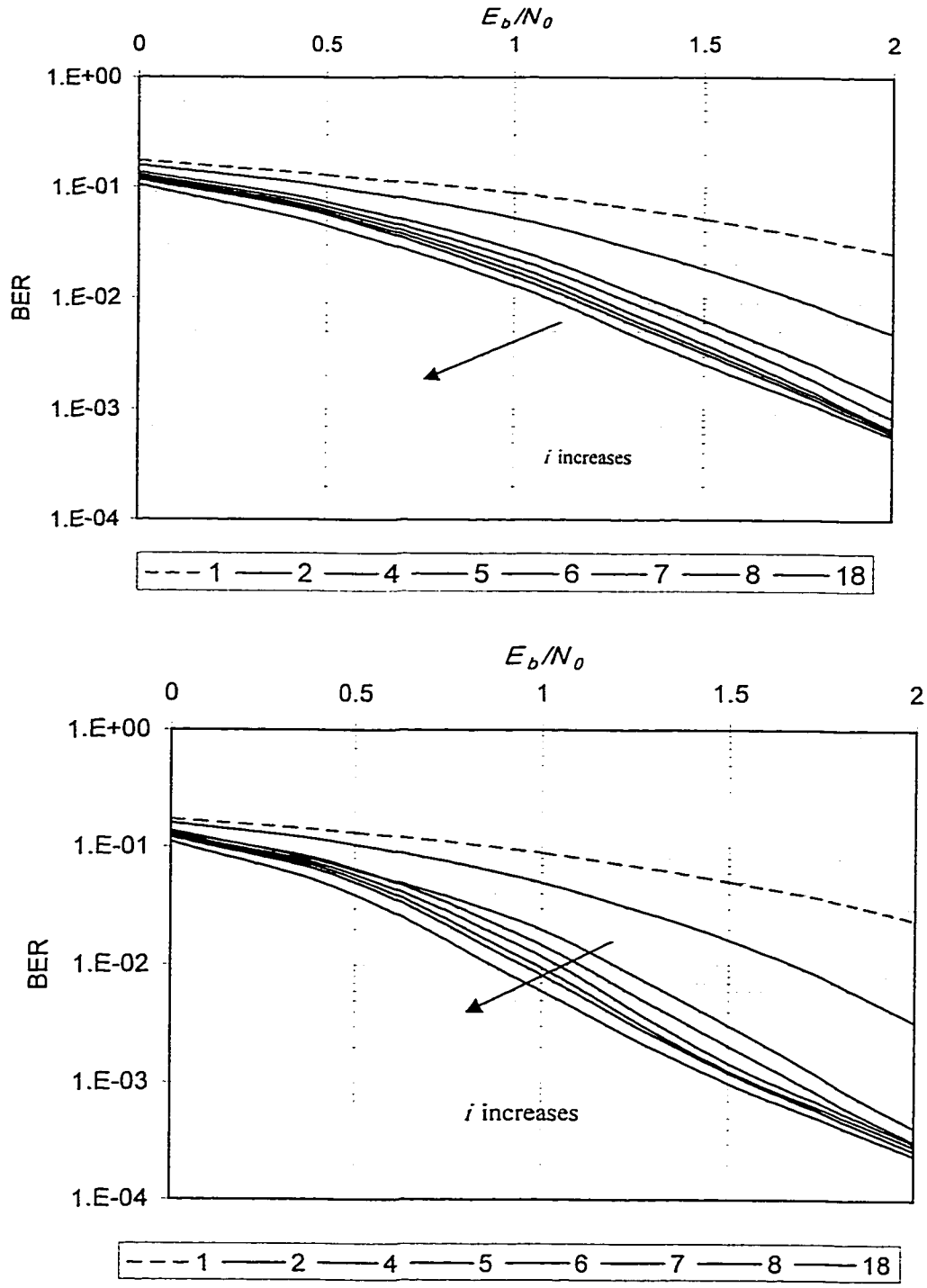


Figure 3.15: (a) Effective number of iterations for k=192, (16,12) matrix

(b) Effective number of iterations for k=768=4*192, (32,24) matrix

Now, let's combine the findings of last section and this section. For good interleavers;

1) Increasing the interleaver size improves the performance.

2) Increasing the number of iterations improves the performance.

3) The effective number of iterations is almost independent of the interleaver (provided it is a good one) map (type) or interleaver size. This implies that the percentage improvement with increasing i is almost constant for any given good interleaver.

3.4.3 Interleaving (I) / Deinterleaving (I^{-1})

All the available literature tries to optimize the interleaver by considering its effect on the encoder (randomizing the codeword and shaping the weight distribution). We have already seen that interleaving plays a significant role in the decoding process, Figure 3.11. As a result, two points have to be understood, namely: the interaction between the interleaver and deinterleaver characteristics, and the role of the I and the I^{-1} and their effects on the performance. Each of these two points will be explained with further details.

Interleaver and deinterleaver characteristics

The interaction between the I and I^{-1} can not be ignored. Consider, for example, the spread which is defined as follows [37]:

An interleaver with permutation function π has spread (s,t) if $|\pi(x)-\pi(y)| \geq t$ whenever $|x-y| < s$. That is, π has spread (s,t) if any two input symbols separated by a distance of at most s are separated at the output by a distance of at least t. Note that if an interleaver has spread (s,t), its inverse (de-interleaver) has spread (t,s). That is designing the spread for the interleaver determines the spread of the deinterleaver. Consider the following example:

Sequence	1	2	3	4	5	6	7	8
I	1	3	5	7	2	4	6	8
I^{-1}	1	5	2	6	3	7	4	8

Notice that the spread for I is 2,4 and for I^{-1} is 4,2. For I , the next bit is at least away by 4 positions. For I^{-1}, the next 4 bits are at least not adjacent to the current bit.

In turbo coding it is desirable to have large spread for both the interleaver and deinterleaver. A large spread helps to counteract burst errors. Burst errors are probable at the output of a Viterbi decoder due to memory correlation. A wrong decision will cause the decoder to be in a wrong state until it gets back to a correct decision. If the decision is taken by the first decoder, we need an interleaver with good spreading capability to spread these errors and make the correction chance higher. If the wrong decision is taken by the second decoder, I^{-1} is required to have a large spread to do the same and make the two decisions uncorrelated.

The role of the I and the I^{-1} and their effects on the performance

The results of Sections 3.4.1 and 3.4.2 showed that a good interleaver alone does not add to the performance of the code without iterative decoding. By adding to that the similarity of the roles played by I and I^{-1} at the decoder, one suggests that designing a code with I^{-1} as the interleaver and doing the corresponding modification at the decoder should not affect the overall performance. This is strongly supported by Figure 3.16 where the performance of TC; under different interleaver types with I and I^{-1} interchanged are shown.

Our observation that I and I^{-1} can be interchanged without any significant effect on the code performance can be explained by the following argument. First consider the decoder. The iterative decoding is depicted in Figure 3.17a. The similar role played by I and I^{-1} is quite evident. It is apparent that, for i iterations, the final output sequence would have been processed by I and I^{-1} for i+1 and i times respectively. For i reasonably large (i=6) the two boxes can be interchanged.

Next, let's consider the encoder. Figure 3.17b is obtained by replacing I in Figure 1.1 with I^{-1}. We will show that the two block diagrams have almost the same encoding effect. Figure 3.17c is identical to 3.17b and is obtained by adding I^{-1} and I at the front of the encoder. Distributing I over the three branches leads to Figure 3.17d. Note that I^{-1} at the front of the encoder and I in the systematic branch add nothing to the performance since they are external to the encoding process. As a result the TC in Figure 3.17d results in the same performance as that in Figure 1.1.

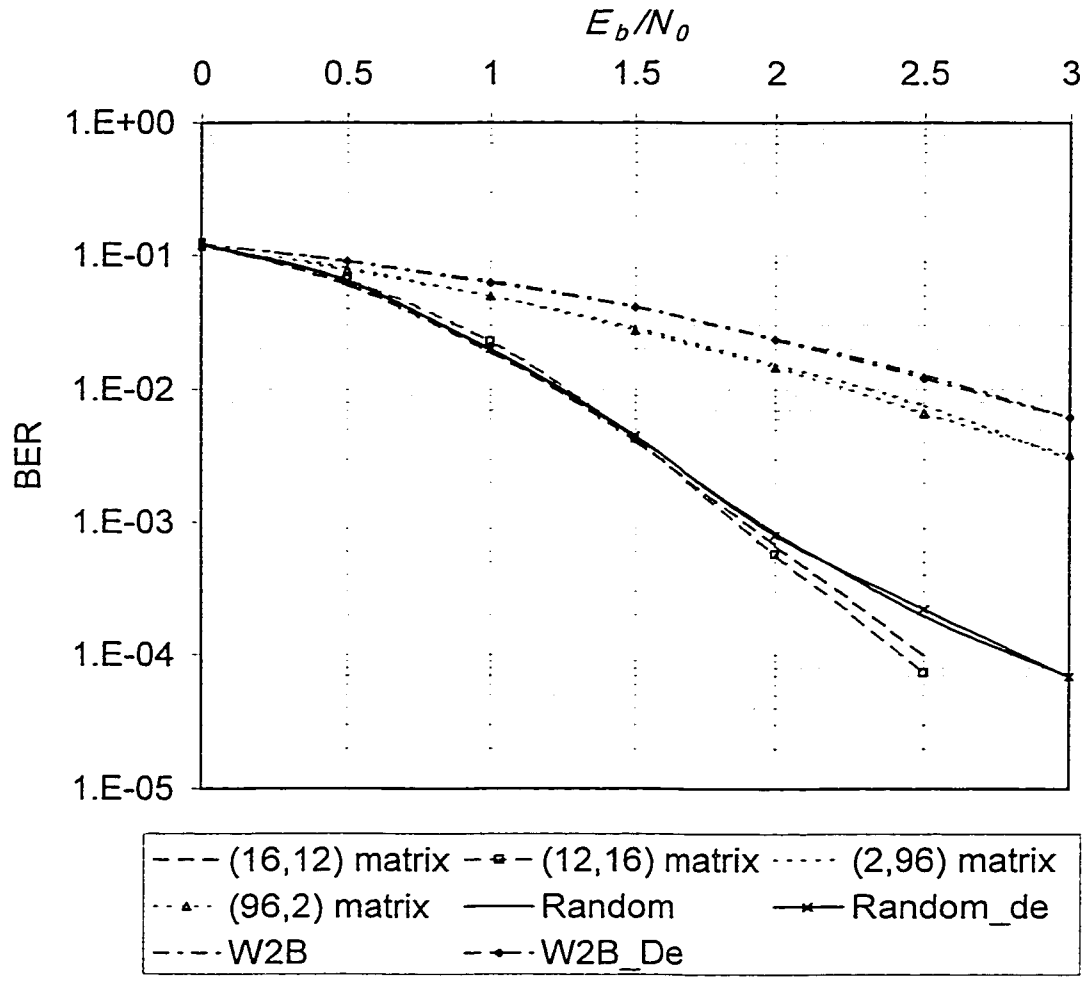


Figure 3.16: Interleaver/ deinterleaver performance

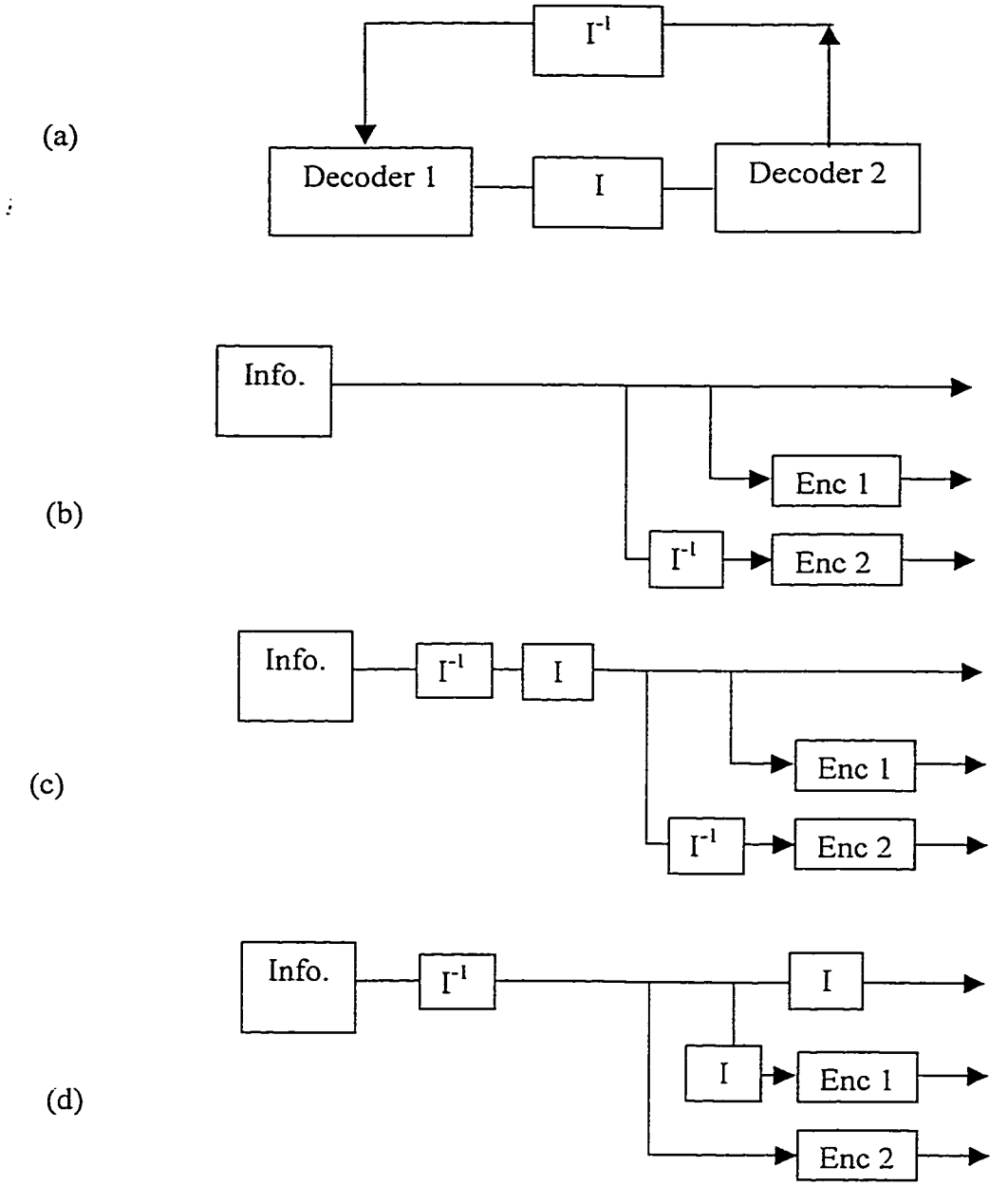


Figure 3.17: (a) Decoder I, I^{-1} loop

(b) Turbo encoder (with interleaver I^{-1})

(c) I and I^{-1} are added to (b)

(d) Similarity of (b) to the turbo encoder with (I) as the interleaver

3.4.4 Cyclically Shifted Interleavers

In this section we study the relative performance of interleavers obtained by a number of cyclic shifts of one interleaver. Figure 3.18 shows two sets of curves, one for a random interleaver and two of its cyclic shifts, and another for the (32,6) matrix interleaver and two of its cyclic shifts. The figure clearly shows that cyclic shifting has minimal effect on the performance.

From this observation we support the following conclusions. First, the effective feature of the interleaver is its spreading capability. Performing any number of cyclic shifts does not modify the spreading feature of the interleaver, and hence results in essentially the same performance. Second, there is no one super interleaver that gives performance far from others. For any interleaver we can form k equivalent interleavers by 1,2,....k cyclic shifts.

Since the performance is not sensitive to interchanging I and I^{-1} (the result of previous subsection) there is a total of $2k$ equivalent interleavers modified from the same interleaver.

3.5 Asymptotic Behavior of Turbo Codes

Figure 3.19 shows a typical performance curve of a TC. Two regions can easily be distinguished: a steep-fall region at low E_b/N_0 (below 2dB in the figure) and a slow-fall region at higher values of E_b/N_0 (asymptotic region). A failure to observe the steep-fall region is a result of not considering sufficient terms in the evaluation of TC. "Turbo codes seem to turn the convolutional design principles of their head; they make error

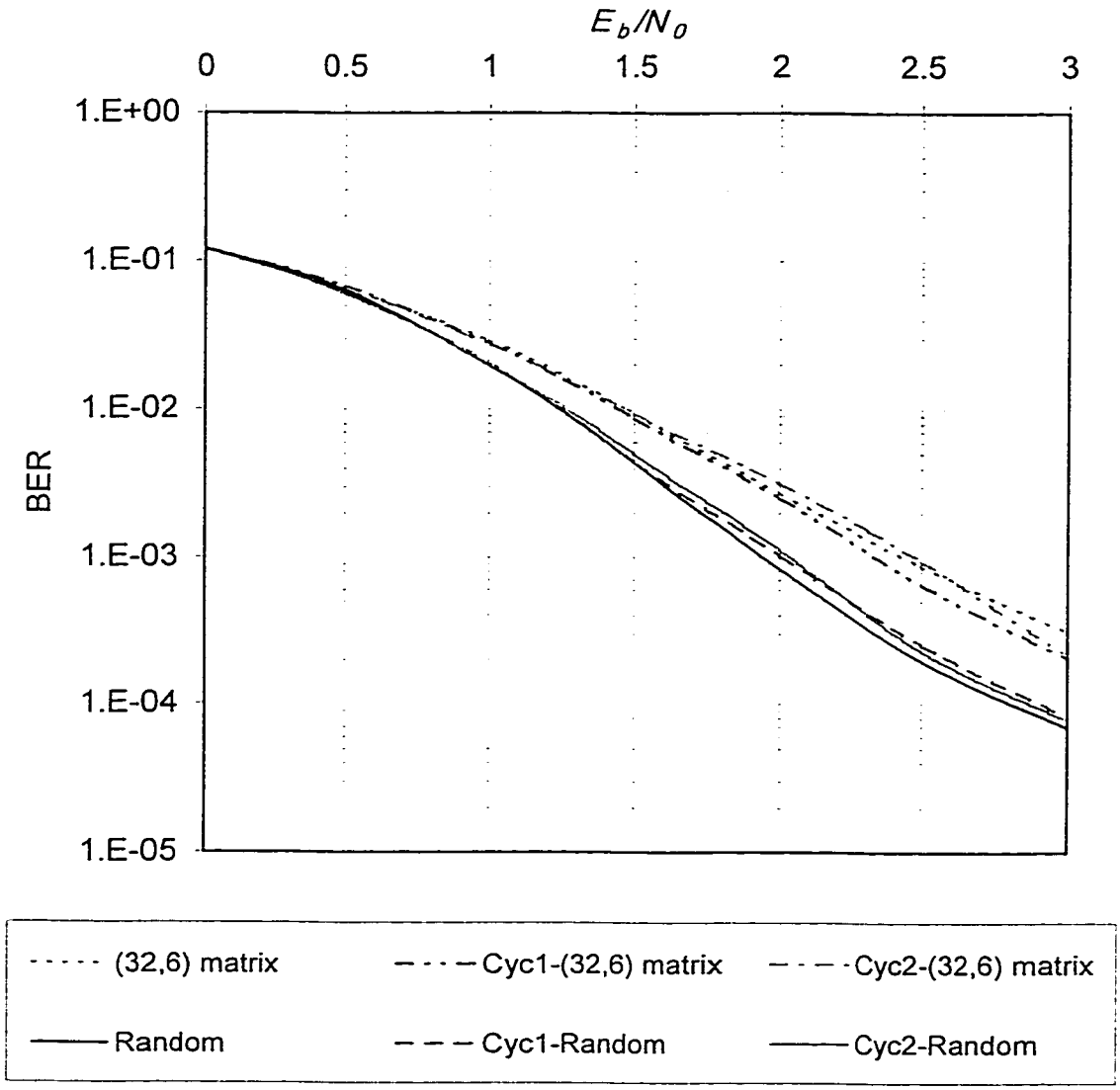


Figure 3.18: Performance of cyclically shifted interleavers

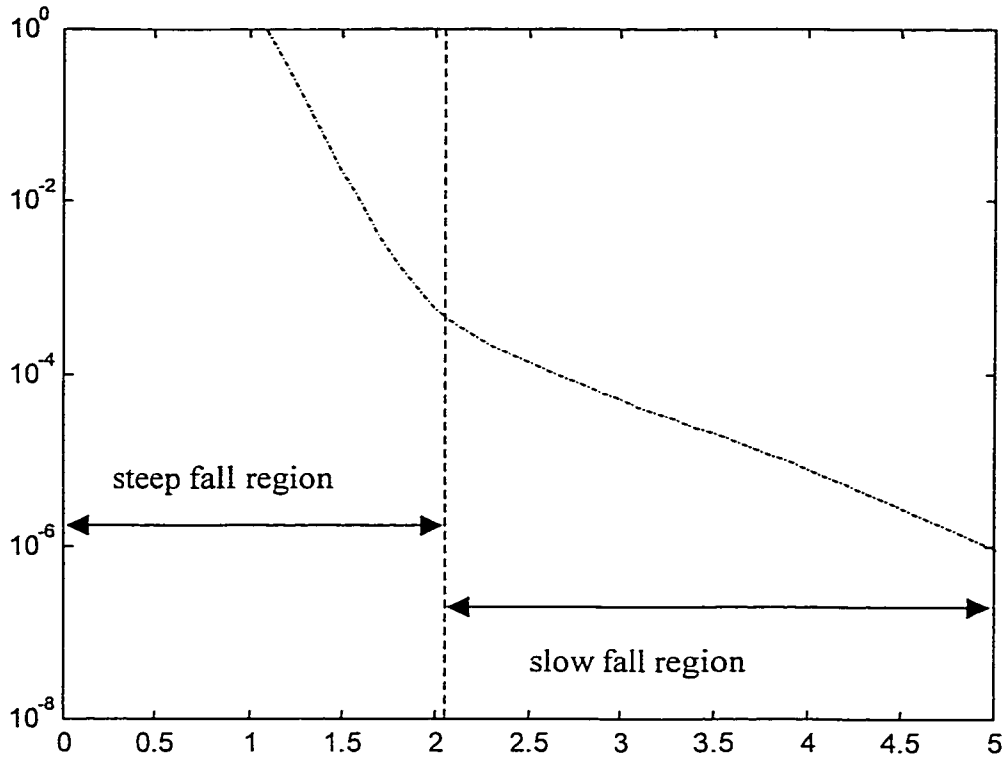


Figure 3.19: Two performance regions of TCs

(union bound, k=192)

coefficients (multiplicity B_i) more important than minimum distance!", Dave Forney [10]. These terms are significant in the low E_b/N_0 region.

Figure 3.20 shows the clear effect of the number of terms on the evaluation of turbo codes at low E_b/N_0.

The above discussion indicates that TC's performs perfectly well at low E_b/N_0, but do not perform as good asymptotically. If we compare TCs to other codes, then we will find that TCs competes the other codes at low E_b/N_0, but as we increase E_b/N_0 there will come a point where the other code will take over.

As an illustration for this last point, the way TCs perform at high SNR is compared with convolutional code with similar complexity. The (2,1,14) convolutional code has the following parameter d_{free}=18, and the asymptotic performance is given by [20]:

$$P_{free} = \frac{3*2}{65536} Q\left(\sqrt{6\frac{E_b}{N_0}}\right) \tag{3.3}$$

For the (1,37/21,37/21) TC with k=65536, the free distance asymptote is [20]:

$$P_{free} = 137 Q\left(\sqrt{18\frac{E_b}{N_0}}\right) \tag{3.4}$$

The objective of presenting these two plots together, Figure 3.21, is not comparison but rather to explore the extreme case. We can see that at some point the two carves will intersect and convolutional codes will outperform TCs.

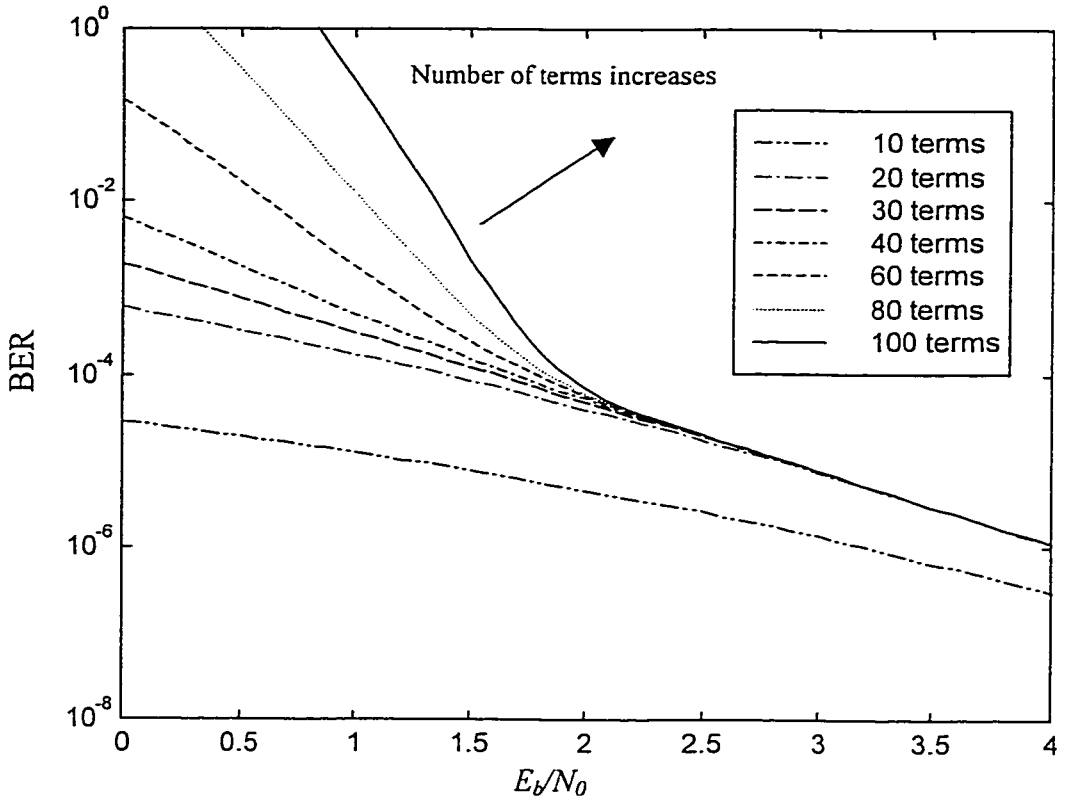


Figure 3.20: Effect of increasing the number of terms in the union bound

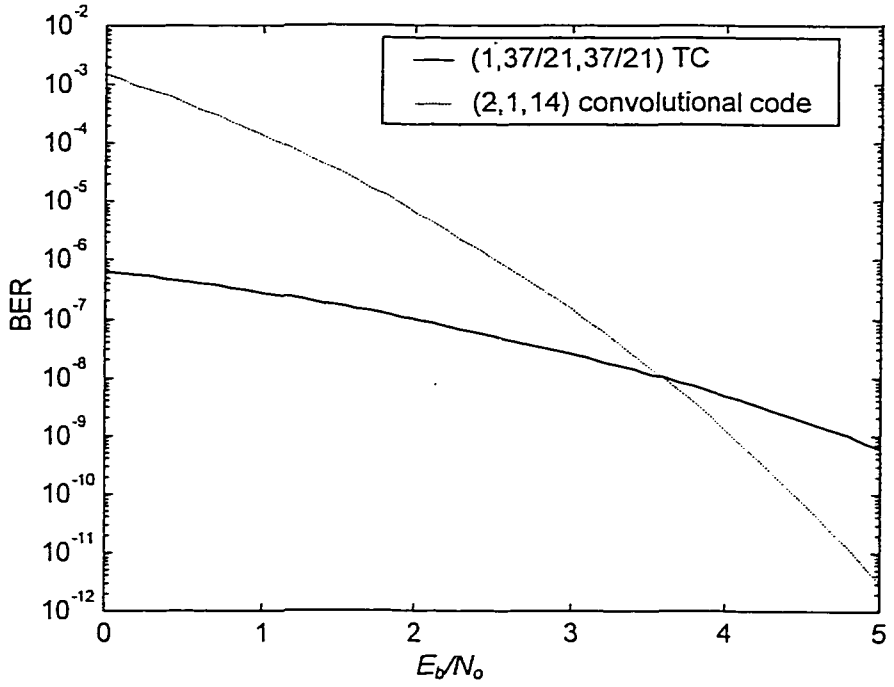


Figure 3.21: Asymptotic performance for convolutional and TC

Personal communication systems, PCS, usually operate around 10^{-3} [36]. So even though the convolutional code asymptotically performs better than the TC. Turbo codes are preferred at the regions at which many systems operate.

What are the factors that affect the asymptotic behavior? And how we improve the asymptotic behavior?

For moderate and high signal-to-noise ratios, it is well known that the free distance in the union bound on the BER performance dominates the bound [38]. So the asymptotic BER is:

$$P_b \approx \frac{B_{free}\widetilde{w}_{free}}{k} Q\left(\sqrt{d_{free}\frac{2RE_b}{N_0}}\right) \tag{3.5}$$

Where B_{free} : is the multiplicity of free distance codewords

$\widetilde{w}_{free}$: average weight of the information sequence causing free distance codeword

From now on we will call this probability P_{free}, *free distance asymptote.* It is shown for some codes and interleaver length k that the slow slope region goes in parallel with the free distance asymptote [20].

To simplify the presentation of Eq. 3.5, the Q function can be approximated to an exponential function at high values of E_b/N_0, utilizing the following relations [39],

$$Q(x) = \frac{1}{2} erfc(\frac{x}{\sqrt{2}}) \tag{3.6}$$

$$erfc(x) < \frac{\exp(-x^2)}{\sqrt{\pi}} \tag{3.7}$$

So the low slope portion can be approximated as a straight line if we take the natural logarithmic value of both sides of Eq 3.5 and we get

$$\log_e P_b < \log_e \frac{B_{free} \widetilde{w}_{free}}{2k\sqrt{\pi}} - Rd_{free} \frac{E_b}{N_0} \tag{3.8}$$

Form Eq. 3.8, it can be concluded that the slow fall observed with TCs is a result of having small free distance and consequently a relatively low free distance asymptote.

The interleaver can play a significant role in determining the asymptotic behavior of the code. In particular, a large d_{free} makes the asymptotic curve steeper, while a smaller multiplicity pushes the turning point further down. Figure 3.22 illustrates the effect of controlling d_{free} and the multiplicity on the asymptotic behavior.

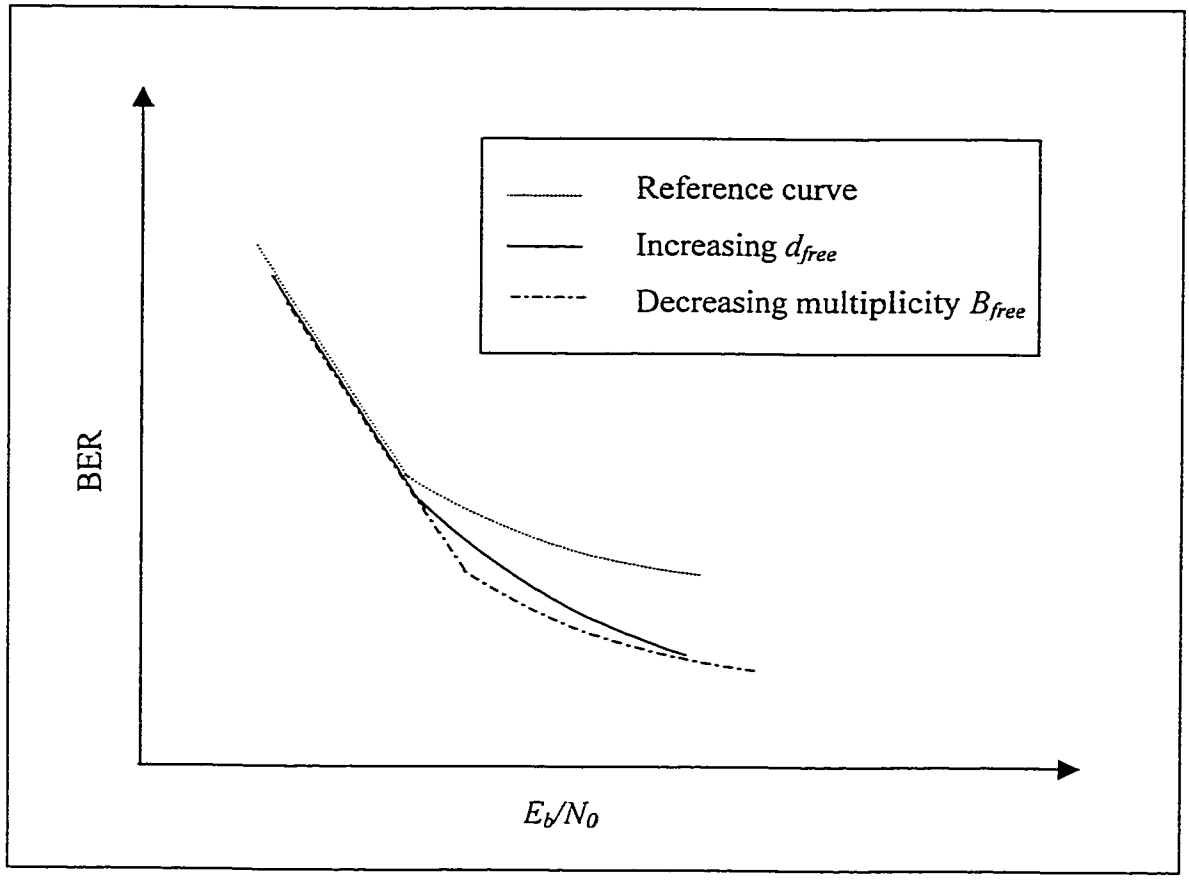


Figure 3.22: Interleaver effect on the asymptotic behavior (sketch)

CHAPTER 4

PUNCTURING EFFECTS ON THE CODE PERFORMANCE

4.1 Introduction

Due to the low rate of TCs, by virtue of having more than one parity branch, there is a genuine reason to puncture them so that to boost their rate. Moreover, the need for adaptive rate error correcting coding arises in many applications. In some applications the data to be transmitted does not have the same level of importance, thus calling for unequal error protection. In other applications the channel may be time varying. Typical examples are the mobile multi-path radio channels, transmission in a jamming environment, or in HF transmission. For these channels, the optimum performance can be obtained by matching the code rate capability to the prevailing channel conditions.

From a practical point of view adapting the code rate should not be done by switching between a set of encoders and decoders, but rather by modifying the encoder and decoder slightly such that the same basic encoder and decoder circuits can be used at all rates. This makes punctured codes a favorite candidate for adaptive-rate systems.

Some communication systems operate under incremental redundancy [40], and thus require a class of codes known as Rate Compatible Codes. Rate compatible means that every lower rate code has a distance, which is at least the distance of the same path within the higher rate code. A rate-compatibility restriction on the puncturing tables ensures that all code bits of high rate codes are used by the lower rate codes. If the higher rate code is not powerful enough for decoding, only the previously punctured bits have to be transmitted [29].

In conclusion, punctured codes offer a perfect solution for adaptive as well as non-adaptive systems. There are excellent papers for the analysis and evaluation of punctured convolutional codes. Extending this work to turbo codes has not been mature yet.

This chapter starts by defining the puncturing matrix. Then the approach in [35] for incorporating the puncturing effect in the evaluation of the weight distribution is reviewed. In the section that follows, we modify the approach for a more accurate evaluation of the weight distribution. Consequently, a tighter upper bound on the code performance is obtained. Also, the new approach facilitates the investigation of some puncturing aspects that are otherwise not possible to study. The results are supported with the new bounding technique and simulation.

4.2 Puncturing Matrix

For a mother (unpunctured) code having N output branches, the puncturing matrix can be represented as follows:

$$\mathbf{P} = \begin{array}{c} \\ \uparrow \\ N \\ \downarrow \end{array} \begin{array}{c} \begin{array}{ccc} \leftarrow & p & \rightarrow \end{array} \\ \begin{pmatrix} p_{11} & \cdots & p_{1p} \\ \cdots & p_{ik} & \cdots \\ p_{N1} & \cdots & p_{Np} \end{pmatrix} \end{array} \tag{4.1}$$

Where every row corresponds to a branch of the encoder.

Note that $p_{ik} \in \{0,1\}$ where 0 implies that the corresponding bit is punctured. The period of the puncturing matrix is p. Of course, a degree of freedom in controlling the code rate can be gained by increasing p. For rate-$1/N$ mother turbo code, the first row corresponds to the systematic branch, the second row corresponds to the first parity branch and so on.

If the rate of the constituent code is $1/N_{cc}$ then the resulting unpunctured turbo code rate is equal to $1/(N(N_{cc}-1)+1)$. If $w(\cdot)$ is the Hamming weight operator, then the rate of the code after puncturing with the puncturing matrix **P** is:

$$R = \frac{p}{w(\mathbf{P})} \tag{4.2}$$

Note that for the unpunctured case $w(\mathbf{P})=Np$ and Equation 4.2 reduces to $1/N$.

4.3 Weight Distribution of Punctured TCs

In order to evaluate the union bound for unpunctured codes, we need to keep track of each input weight and the corresponding overall output weight. For unpunctured codes,

$$\textit{Overall output weight} = \textit{input weight} + \textit{parity weight} \tag{4.3}$$

And hence it is sufficient to know the input weight and the corresponding parity weight. This information is fully contained in the CIWEF.

Unfortunately, Equation 4.3 does not hold for punctured codes. Also it is not easy, in general, to calculate the transfer function of punctured codes [27]. This makes the task of finding the overall output weight and its corresponding input weight a tedious one. Consequently, the evaluation of the union bound for punctured codes requires more computation compared to unpunctured codes.

A novel algorithm for evaluating the weight distribution of TCs was presented by [35], which incorporates the effects of puncturing the parity bits or the systematic bits. This algorithm is based on the transfer function procedure presented in Chapter 2. A brief description of the algorithm follows.

4.3.1 Puncturing Parity Bits

Consider first puncturing the parity bits. Let k, the block size, be a multiple of the puncturing period. For illustration let the puncturing period p=4 and the puncturing

pattern for the first parity branch be 1101. This means that the third symbol in every sub-sequence of four symbols at the output of the first CC is punctured. For this case to compute the block transition matrix $\mathbf{A}(W,Z)^k$, that is required in Eq. 2.11, we compute the *period transition matrix* **B**, which is the transition matrix for one puncturing period given by:

$$\mathbf{B}(W,Z) = \mathbf{A}(W,Z)\mathbf{A}(W,Z)\mathbf{A}(W,1)\mathbf{A}(W,Z) \tag{4.4}$$

Recall that W and Z are the dummy variables that carry the weight of the input and the parity, respectively.

Setting the parity variable Z in the third matrix to "1" indicates that the third bit is not transmitted. In general, for puncturing period p and arbitrary puncturing vector $\mathbf{P}_i$ (where $\mathbf{P}_i$ is a vector corresponding to the ith parity row of the puncturing matrix **P**), we form the period transition matrix **B**:

$$\mathbf{B}(W,Z) = \prod_{j=1}^{p} \mathbf{A}(W, Z^{\mathbf{P}_i(j)}) \tag{4.5}$$

Where $\mathbf{P}_i(j)$ is the jth element of the vector $\mathbf{P}_i$.

The block transition matrix $\mathbf{A}(W,Z)^k$, which is the kth power of $\mathbf{A}(W,Z)$ can then be computed as the $(k/p)^{\text{th}}$ power of $\mathbf{B}(W,Z)$, i.e.

$$\mathbf{A}(W,Z)^k = \mathbf{B}(W,Z)^{k/p} \tag{4.6}$$

The above procedure can be used to calculate the weight distribution for different puncturing patterns of each CC. The resultant matrices are substituted in Eq. 2.11 to find the composite transfer function of the turbo code under uniform interleaving.

It should be noted that if the block length k is not a multiple of the puncturing period p then a boundary effect will occur. We are not considering this case here. Anyhow, this effect can usually be neglected especially at very large k.

4.3.2 Puncturing Systematic Bits

When systematic bits are punctured, we need to find the weight of the information sequence after puncturing.

For an input sequence of length k and Hamming weight w, if M bits in a period of p are punctured then the total number of punctured bits is $(k*M/p)$. The punctured systematic sequence weight y is then bounded as

$$w-(k*M/p) \le y \le w \tag{4.7}$$

The method developed in [35] does not have the capability of finding y for a given w and a particular puncturing pattern. Instead, the authors resort to an averaging technique over the whole frame to find $p(y|w)$. That is they assumed that all patterns of $k*M/p$ punctured bits out of the k bits are possible, and that they are all equiprobable.

Based on these assumptions, The probability of having certain systematic output weight y given input weight w is given by:

$$p(y \mid w) = \frac{C_{w-y}^{k*M/p} * C_{y}^{k-(k*M/p)}}{C_{w}^{k}} \tag{4.8}$$

Where C_d^x is zero if either $x<d$ or $d<0$.

The drawback of this method is clearly that it does not consider the particular puncturing pattern nor relate puncturing to the specific CC used. It only averages out a certain amount of degradation irrespective of the CC or the positions of the punctured bits.

4.4 Modified Transfer Function

Recall that for the evaluation of the code performance, we need to keep a record of the associated input and the output weights. For a punctured TC where puncturing is applied to the systematic sequence, the overall output weight is not equivalent to the sum of the input and parity check weights. To overcome this problem one has to keep a record of the weight of the punctured systematic sequence. We propose two methods of modifying the transfer function to acquire this information.

4.4.1 Method 1

The first method will be to add a new variable and introduce a new function, which we call the *conditional input-redundancy-output weight enumerating function* CIROWEF.

CIROWEF

The conditional input-redundancy-output weight enumerating function keeps track of the weight of the systematic sequence after puncturing (y). It is represented by the following summation:

$$D_w^C(Z,Y) = \sum_{y,j} A_{y,j} Z^j Y^y \tag{4.9}$$

Where Y and Z are dummy variables, and

y : is the weight of the systematic branch after puncturing.

j : is the weight of the parity branch.

One method to evaluate $D_w^C(Z,Y)$ is to utilize the procedure presented for punctured parity bits (Section 4.3.1). We need first to modify the transition matrix of the first CC to account for the variable Y. The variable Y is introduced in the transition matrix by replacing W by WY. For the (1,5/7,5/7) code the modified transition matrix will be

$$\mathbf{A}(W,Z,Y) = \begin{pmatrix} 1 & 0 & WZY & 0 \\ WZY & 0 & 1 & 0 \\ 0 & WY & 0 & Z \\ 0 & Z & 0 & WY \end{pmatrix} \tag{4.10}$$

The procedure for evaluating the CIROWEF is explained with a specific example.

Consider the puncturing matrix

$$\mathbf{P} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \tag{4.11}$$

The first row of **P** implies that the third systematic symbol in every sub-sequence of three symbols is punctured. The second and third rows are for the parity bits from the first and second encoders respectively.

As outlined in Section 4.3.1, we first compute the period transition matrix of each CC. For the puncturing matrix in (4.11) and for the first CC, we get

$$\mathbf{B}^{C_1}(W,Z,Y) = \mathbf{A}(W,Z,Y)\mathbf{A}(W,1,Y)\mathbf{A}(W,Z,1) \tag{4.12}$$

This equation implies that in the first transition both the systematic bit and the parity bit are transmitted, in the second transition the systematic bit is transmitted but the parity bit is punctured, and in the third transition the systematic bit is punctured and the parity bit is transmitted.

The period transition matrix for the second CC does not change and is given by:

$$\mathbf{B}^{C_2}(W,Z) = \mathbf{A}(W,1)\mathbf{A}(W,Z)\mathbf{A}(W,Z) \tag{4.13}$$

Since the systematic bits are sent once, they are included only in the first CC. Setting the parity variable Z to "1" implies that we are not transmitting the corresponding parity bits. Now the matrices $\mathbf{B}^{C_1}$ and $\mathbf{B}^{C_2}$ are substituted in Eq. 4.6 to evaluate the block transition matrix for the corresponding CC. The resulted $\mathbf{A}^{C_1}$ and $\mathbf{A}^{C_2}$ are inserted in Eq. 2.11 and

the overall weight for every codeword of the composite TC is the sum of the powers of Y and Z.

The pre-mentioned method requires the computation of matrices with three variables. Method 2 is a modification of Method 1 that reduces the number of variables to two.

Method 2

In the second method after formulating the period transition matrices $\mathbf{B}^{C_1}$ and $\mathbf{B}^{C_2}$, both variables Z and Y are replaced by the variable H. This simplification is based on the observation that from this point onward the contribution of the punctured systematic and parity branches to the overall output weight need not be distinguished. This will simplify the calculation of the block transition matrix $\mathbf{A}(W,Z,Y)^k$ which will now be a function of two variables W and H, with the power of the variable H carrying the overall output weight, i.e. systematic and parity, for the first CC.

The drawback of the second method is that we have to truncate the power of the variable H at larger value to get the same accuracy as method 1. This is because the power of H carries the overall weight. At any rate, Method 2 possesses the advantage of being simpler to implement.

4.5 Comparison between the Proposed and the Averaging Techniques

In this section, a simple example is worked out to help understanding the differences between the approach in [35] and our modified approach, and how the modified approach leads to a more accurate evaluation of the weight distribution.

Consider the very popular (7,4) Hamming code. All the codewords resulting from an input of weight two (w=2) are shown in Table 4.1. The first four bits are the systematic bits $b_0b_1b_2b_3$. From the table it is clearly seen that for the given Hamming code:

$$A_2^C(Z) = 3Z + 3Z^2 \tag{4.14}$$

For a turbo code built of two (7,4) Hamming codes as the constituent codes, we get,

$$A_2^{C_1}(Z) = A_2^{C_2}(Z) \tag{4.15}$$

Using a uniform interleaver, the CIRWEF for the parallel concatenation is given by:

$$A_2^{C_P}(Z) = \frac{A_2^{C_1}(Z) * A_2^{C_2}(Z)}{6} = W^2(1.5Z^2 + 3Z^3 + 1.5Z^4) \tag{4.16}$$

Note that the sum of the coefficients of Z is 6 which is the total number of codewords with input weight two.

Systematic				Parity			Monomial
b_0	b_1	b_2	b_3	b_4	b_5	b_6	Representation
1	0	1	0	0	0	1	Z
0	1	1	0	1	0	0	Z
0	0	1	1	0	1	0	Z
0	1	0	1	1	1	0	Z^2
1	0	0	1	0	1	1	Z^2
1	1	0	0	1	0	1	Z^2
CIRWEF							$W^2(3Z+3Z^2)$

Table 4.1: Codewords as a result of weight-two input for the (7,4) Hamming code

If the two systematic bits b_1 and b_3 are punctured then the new CIRWEF can be calculated in two ways: the averaging [35] and the proposed methods. Using the averaging technique as in Eq. 4.8, the conditional probabilities are as follows:

$$p(y=0|w=2)=1/6\,,\ p(y=1|w=2)=2/3 \text{ and } p(y=2|w=2)=1/6 \qquad (4.17)$$

After puncturing the CIRWEF is given by:

$$A_2^{C_P}=(1.5Z^2+3Z^3+1.5Z^4)(\frac{1}{6}+\frac{2}{3}W+\frac{1}{6}W^2) \qquad (4.18)$$

For this approach, if the punctured bits were b_0 and b_2 the same result will be obtained. Moreover, the effect of puncturing represented in the second bracket is totally isolated from the first bracket, which is determined by the specific code used. This means that the amount of degradation in performance due to puncturing is independent of the specific code used.

Applying the new method, Table 4.2a illustrates the effect of puncturing bits b_1b_3 on the weight distribution of the code. After puncturing, the CIROWEFs for the two CCs are given by:

$$A_2^{C_1}=2YZ+Z^2+2YZ^2+Y^2Z \qquad (14.19)$$

$$A_2^{C_2}=3Z+3Z^2 \qquad (14.20)$$

In Equations 14.19 and 14.20 the two variables Y and Z can be replaced by H to get:

Systematic				Parity			Monomial
b_0	b_1	b_2	b_3	b_4	b_5	b_6	Representation
1	0	1	0	0	0	1	Y^2Z
0	1	1	0	1	0	0	YZ
0	0	1	1	0	1	0	YZ
0	1	0	1	1	1	0	Z^2
1	0	0	1	0	1	1	YZ^2
1	1	0	0	1	0	1	YZ^2
CIROWEF				$W^2(Y^2Z+2YZ+Z^2+2YZ^2)$			

(a) b_1 b_3 punctured conditional weight distribution

Systematic				Parity			Monomial
b_0	b_1	b_2	b_3	b_4	b_5	b_6	Representation
1	0	1	0	0	0	1	Z
0	1	1	0	1	0	0	YZ
0	0	1	1	0	1	0	Z^2
0	1	0	1	1	1	0	Y^2Z^2
1	0	0	1	0	1	1	YZ^2
1	1	0	0	1	0	1	YZ^2
CIROWEF				$W^2(Z+YZ+Z^2+Y^2Z^2+2YZ^2)$			

(b) b_0 b_2 punctured conditional weight distribution

Table 4.2: Codewords as a result of input weight 2 for punctured (7,4) Hamming code

$$A_2^{C_1} = 2H^2 + H^2 + 2H^3 + H^3 = 3(H^2 + H^3) \tag{14.21}$$

$$A_2^{C_2} = 3H + 3H^2 = 3(H + H^2) \tag{14.22}$$

and the overall CIRWEF is given by:

$$A_2^{Cp} = \frac{A_2^{C_1} * A_2^{C_2}}{6} = \frac{3(H^2 + H^3) * 3(H + H^2)}{6} \tag{14.23}$$

$$A_2^{Cp} = 1.5H^3 + 3H^4 + 1.5H^5 \tag{14.24}$$

Let's compare the weight distribution of the code C_P obtained this way with that obtained in (14.18). For ease of comparison we substitute H for Z and W in (14.18) to get

$$A_2^{C_P} = 0.25H^2 + 1.5H^3 + 2.5H^4 + 1.5H^5 + 0.25H^6 \tag{14.25}$$

The results are tabulated in Table 4.3. From the table the difference between the two methods is quite apparent. The minimum Hamming weight codeword is guaranteed to be not less than three with any interleaver used. This conclusion on the minimum weight codeword cannot be reached by the averaging technique. Therefore, it is clear that the proposed method provides a better estimate of the minimum distance as well as weight distribution of the code.

Now let's apply the two methods to find the weight distribution of the (1,5/7,5/7) TC. The systematic sequence is punctured according to the vector [1 1 0], that is p=3 and M=1. Table 4.4 shows the weight distribution of the code up to the weight m=18 using the

Weight j	0	1	2	3	4	5	6	Sum
Averaging	**0**	**0**	0.25	1.5	2.5	1.5	0.25	**6**
New Method	**0**	**0**	**0**	1.5	3	1.5	**0**	**6**

Table 4.3: CIWEF using the averaging and the new method

Codeword Weight m	D_m [35]	D_m [New]
3	0	0
4	0.00001758369000	0
5	0.00010925460000	0
6	0.00032662980000	0.00049479290000
7	0.00080264170000	0
8	0.00390303700000	0.00975134400000
9	0.01206679000000	0
10	0.01856358000000	0.03666044000000
11	0.02618730000000	0
12	0.03786333000000	0.07284244000000
13	0.04912956000000	0
14	0.07454436000000	0.14234940000000
15	0.10620620000000	0
16	0.17585560000000	0.34564950000000
17	0.28859980000000	0
18	0.51055000000000	1.00283600000000

Table 4.4: D_m using the averaging and the proposed methods

bound in [35] and ours. The accuracy of our approach is quite evident. In particular, it tells us the following:

(i) The minimum distance of the code cannot be less than 6

(ii) Odd-weight codewords do not exist.

This added accuracy in estimating the minimum distance and the weight distribution of the code leads to computing a tighter bound on the BER performance.

Figure 4.1 shows a comparison between the bound using the method in [35] and the bound using our method for the (1,7/5,7/5) TC. We considered two puncturing patterns of the systematic sequence. First, one systematic bit is punctured in a period of two (M=1, p=2). In the second case, one systematic bit is punctured in a period of three (M=1, p=3).

From this figure, the following two observations are evident:

1. The performance at low values of E_b/N_0 is almost equivalent using any of the methods, with the new method providing a slightly tighter bound. The difference in this region can result from two things. The first possible reason is the different number of terms in the union bound. The second reason is the accuracy of the new bound due to the more appropriate evaluation of the weight distribution. The first reason was phased out by allowing a maximum number of terms until the bound converges to a fixed value. Consequently, the little improvement is all due to the

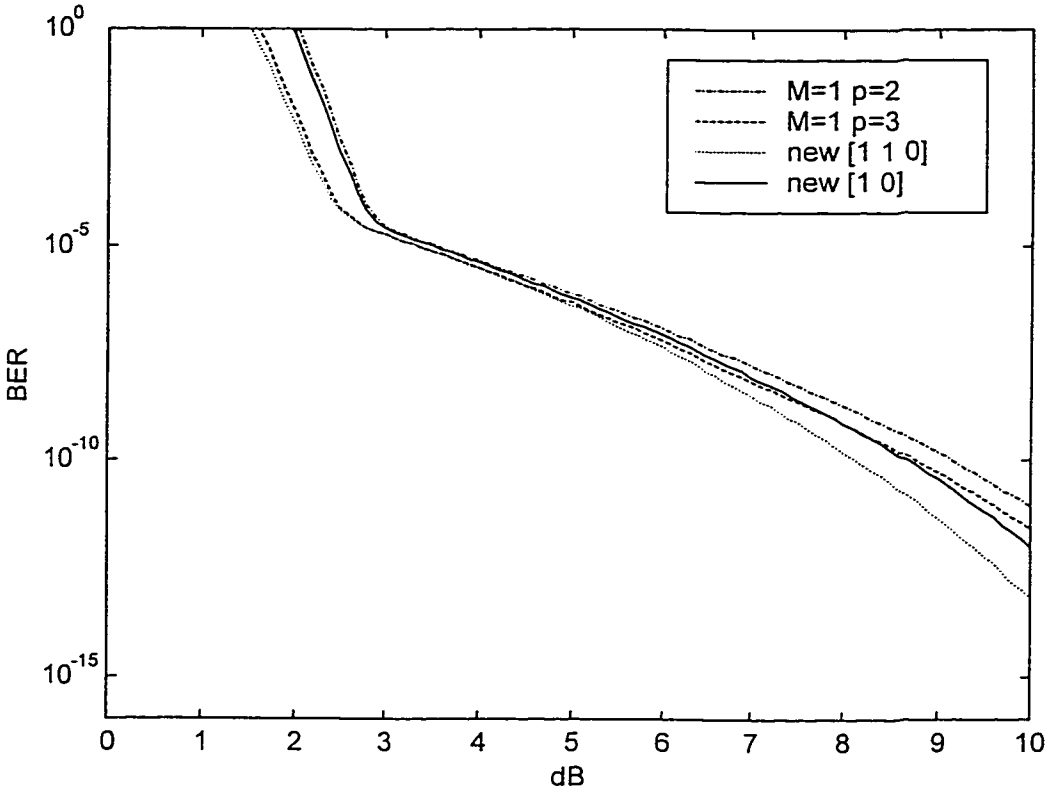


Figure 4.1: Comparison of the two bounding techniques

better accuracy in determining the weights. Although the union bound diverges at low values of E_b/N_0, the added accuracy will help in comparing different codes.

2. At high values of E_b/N_0, the two methods start diverging. Over this range of high E_b/N_0 the code performance is dominated by the minimum distance of the code. The drawback of the averaging method is that it will assume the presence of all possible combinations of the two CCs outputs with different probabilities. So it will join low weight codewords even though that might not exist. Not that this method allows the presence of codewords of weights 4 and 5, while the new method showed that the minimum weight cannot be less than 6.This is why the averaging method is less effective at high E_b/N_0. This also explains why the union bound with uniform interleaving yields an underestimated performance for high E_b/N_0.

There is another big advantage of the new method; it permits the investigation of different puncturing patterns when systematic bits are involved. Consider again the TC built from the (7,4) Hamming code. If the puncturing pattern was [0 1 0 1] instead of [1 0 1 0] a different CIROWEF will be obtained; compare Tables 4.2a and 4.2b. The method in [35] does not have this freedom, and as long as the puncturing patterns has the same p and M the same CIRWEF will be obtained. This flexibility helps also in investigating the effect of puncturing systematic symbols compared to puncturing parity ones. Both investigations are not possible using the approach in [35]. Detailed discussion of the investigations is furnished in the following section.

A final word of comparison between the two methods is related to the computation time and processing requirements. Both methods manipulate matrices with two variables. However, for the proposed method we need to truncate the computation of the block transition matrix of the first CC at a higher power since H holds the overall weight. But on the other hand, no averaging calculation is required. Recall that the averaging calculation, as presented in Eq. 4.8, must be performed for every possible w and y, and hence it cannot be marginalized. From our simulation experience, our method does not require more computation time compared with the averaging method.

4.6 Puncturing Patterns

In this section, different puncturing patterns are studied. In particular we examine:

(i) Puncturing the systematic sequence.

(ii) Puncturing the parity sequences.

(iii) The effects of permuting columns or permuting rows of the puncturing matrix.

We conclude this section by summarizing the properties of a good puncturing matrix

4.6.1 Puncturing Systematic Sequence

It has been reported in [27], based on simulation, that systematic bits should not be punctured. In the following section we evaluate the validity of this conclusion and provide an explanation to it.

Comparison based On the Union Bound

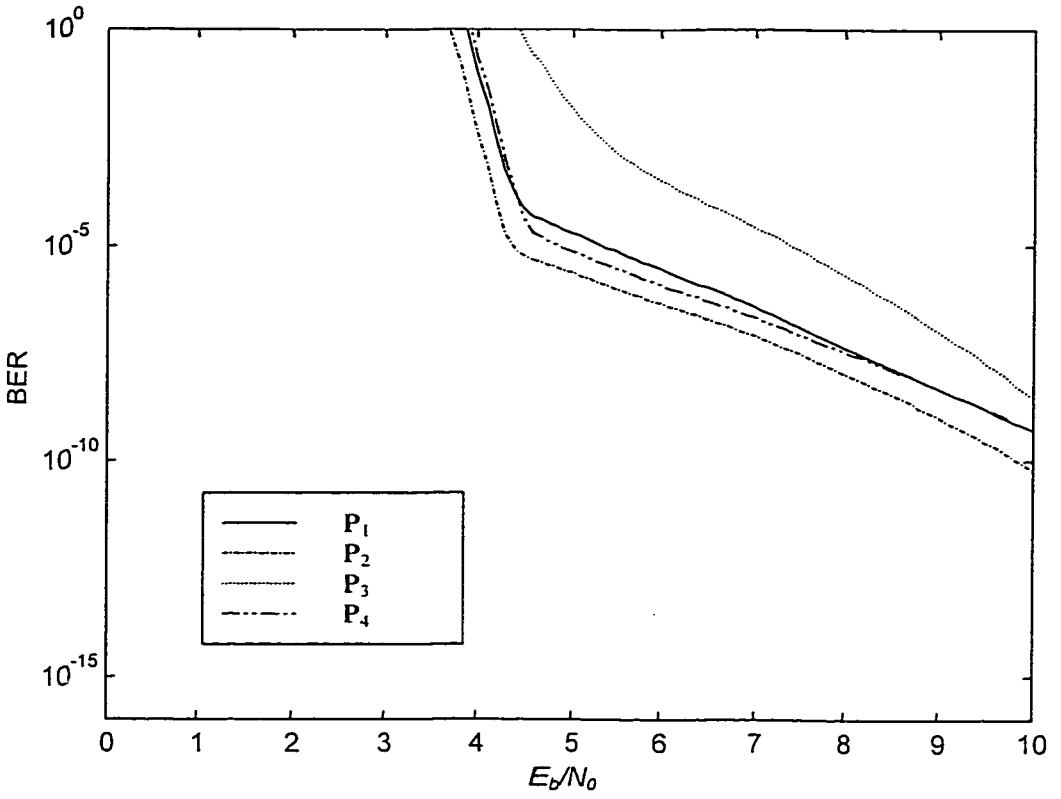
Figure 4.2 shows plots of the performance of the (1,5/7,5/7) code under different puncturing conditions all with a puncturing period p=2. All cases will produce a rate-1/2 code out of the rate-1/3 mother code. The four puncturing matrices are listed in the figure.

Note that for $\mathbf{P}_1$ no systematic bits are punctured, while for $\mathbf{P}_2$ all systematic bits are punctured. $\mathbf{P}_3$, by puncturing the whole parity sequence, transforms the TC into a convolutional code.

From Figure 4.2, it can be seen that the worst case occurs when the code is convolutional ($\mathbf{P}_3$) while the best case occurs when we puncture all systematic bits ($\mathbf{P}_1$). So we can conclude based on the union bound, that for this particular code puncturing systematic bits happens to be better than puncturing parity bits!, which is not consistent with the conclusion of [1]. Let's leave it here and work at the simulation results.

Comparison Based on Simulation

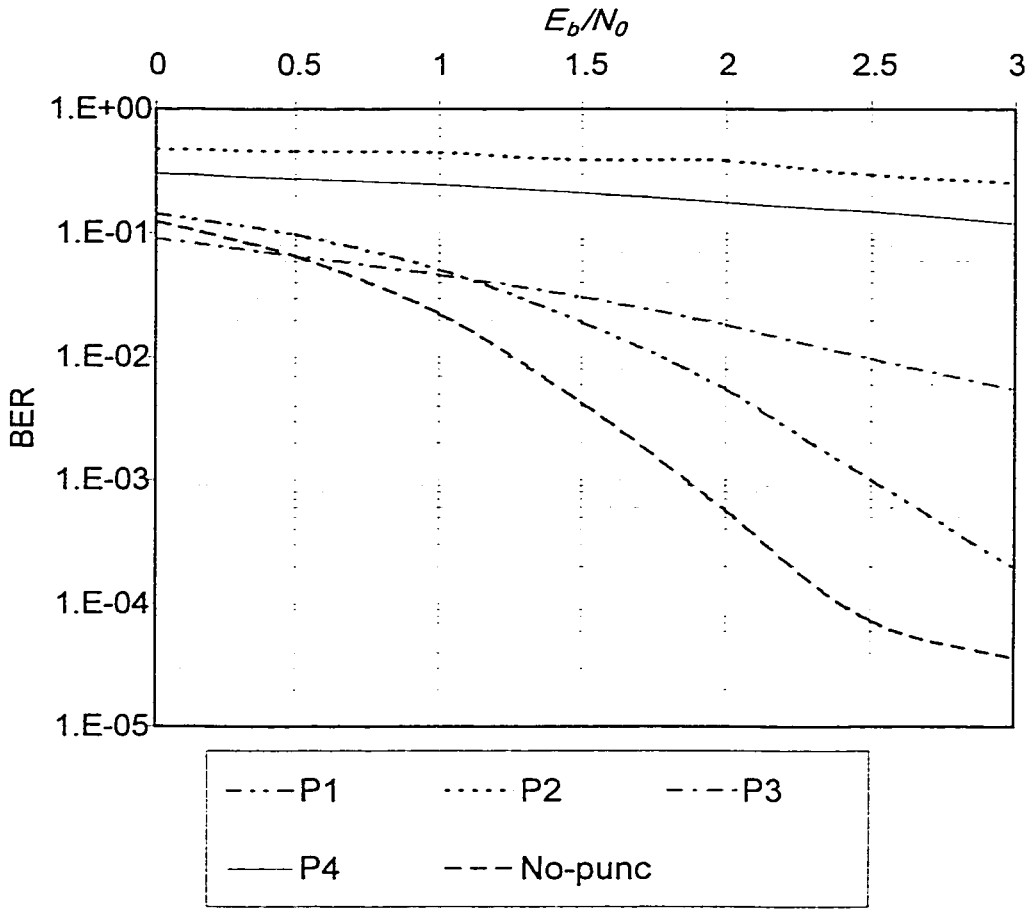
The previous four punctured codes were simulated using a decoder based on SOVA. It was found that irrespective of the specific interleaver used, the code performance degrades sharply as we start puncturing systematic bits! Figure 4.3 shows the four cases with a (12,16) matrix interleaver. Note that the performance under $\mathbf{P}_2$ (all systematic bits are punctured) is the worst, and the performance under $\mathbf{P}_4$ (half of the systematic bits are punctured) is the second worst.



$$\mathbf{P}_1 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \mathbf{P}_2 = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$\mathbf{P}_3 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 0 \end{bmatrix} \qquad \mathbf{P}_4 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}$$

Figure 4.2: Puncturing systematic vs. parity bits using the union bound



$$\mathbf{P}_1 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \mathbf{P}_2 = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$\mathbf{P}_3 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 0 \end{bmatrix} \qquad \mathbf{P}_4 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}$$

Figure 4.3: Puncturing systematic vs. parity bits based on simulation

Why puncturing systematic bits is that damaging? And why this damaging effect was missed by the union bound technique?

Let's attempt answering the first question. Recall that systematic bits are used twice; in the first and second decoder. If the systematic bits are punctured both decoders will be affected and two weak decisions are expected.

More importantly, the presence of the systematic bits gives the code the immunity against being catastrophic. Systematic codes are always non-catastrophic [41]. A catastrophic error is defined as an event whereby a finite number of code symbol errors causes an infinite number of decoded bit errors [41]. If systematic bits are punctured, the code is not guaranteed to be non-catastrophic any more.

Analyzing the behavior of the decoder with punctured systematic bits closely indicates that the second point is very critical. Our simulation results showed the following:

For the case where all systematic bits are punctured (P_2) and at E_b/N_0= 3dB the BER=0.2538. A total of 934 frames were simulated. Out of these, 500 frames were found in error and 434 frames were error free. This means that for erroneous frames, on the average, 50 bits out of 192 bits are in error. So, puncturing systematic bits result in 55% of the frames heavily in error, and 45% of the frames perfectly error free. This is a clear symptom of catastrophic errors.

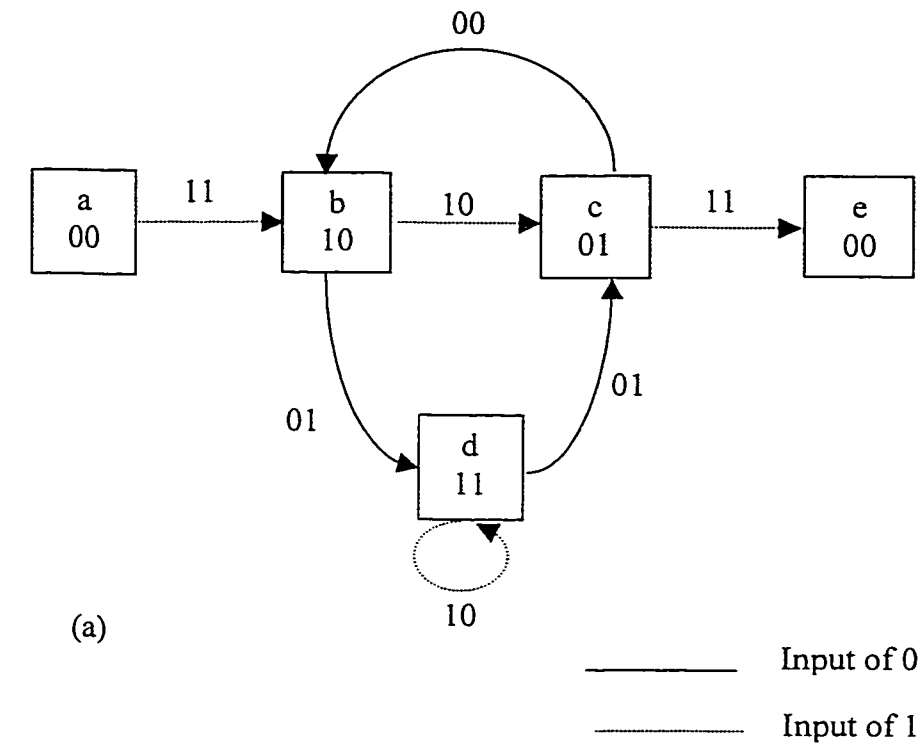
Let's examine closely how puncturing systematic bits affects the immunity to catastrophic errors. According to [41] catastrophic errors occur if, and only if, any closed loop path in the state diagram has zero output weight with non-zero input. To illustrate this, the

modified state diagram of the (5/7) CC is shown in Figure 4.4a. A solid line corresponds to a systematic bit of "0" while a dotted line corresponds to an input of "1". For clarity, the systematic bit and the parity bit are shown as labels of transition branches.

It is clearly seen that the code is non-catastrophic since all loops have non-zero weights. Consider the case where the systematic bits are punctured. Assuming that the all-zero path is the correct path, the incorrect path *a b d d . . . d c e* has exactly 4 ones, no matter how many times we go around the self-loop at node *d*. Thus few channel errors may cause us to choose this incorrect path. An arbitrary large number of errors (depending on the number of times the self-loop is traversed) can result from this wrong decision.

In conclusion, the puncturing **pattern should be selected so as to make the code non-catastrophic**. Optimum codes under no puncturing conditions like the (1,5/7,5/7) might perform very bad under puncturing because of the catastrophic behavior. For example, the sub-optimal (1,2/7,2/7) turbo code performs much better than the optimum (1,5/7,5/7) code when all systematic bits are punctured in both. For instance, our simulation results show that at 3 dB, the BER can get as small as 8.509E-3 using the (1,2/7,2/7) turbo code compared to 2.538E-1 for the (1,5/7,5/7) turbo code. This is because the (1,2/7,2/7) has more immunity to catastrophic errors. The modified state diagram of the (2/7) CC is shown in Figure 4.4b. Even after puncturing the systematic bits, the weight of the loops is proportional to the input weight. For example the Hamming weight is propotional to the number of visits to state *d*.

Now we move to the second question: why does the bounding technique give a contradicting conclusion to that obtained from simulation?



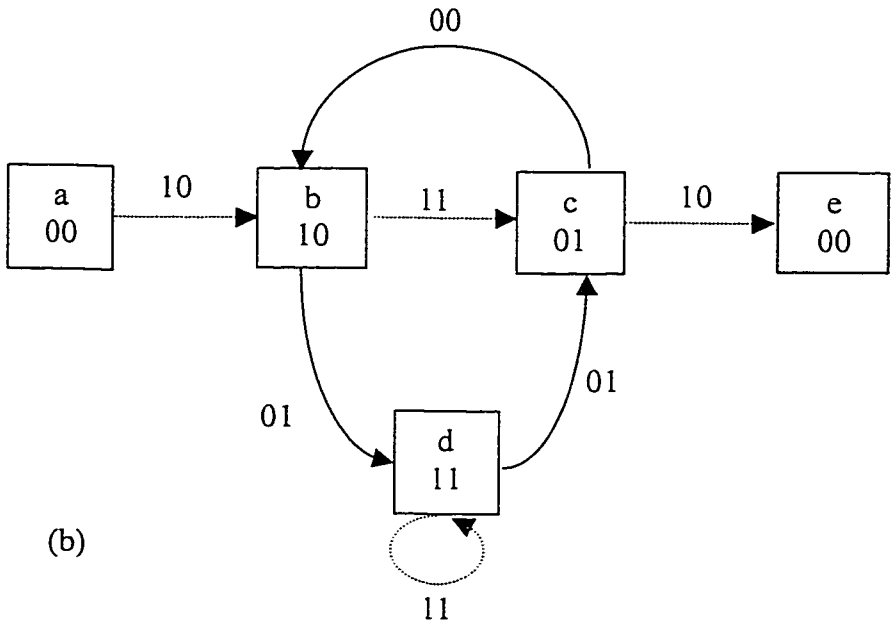


Figure 4.4: (a) Modified state diagram (5/7) code
(b) Modified state diagram (2/7) code

The answer to the question lies in the type of the decoder. Union bound is derived based on ideal ML decoder, while simulation results are obtained using the practical turbo decoder.

The catastrophic behavior is not seen by the ML decoder for turbo codes. This is because the ML decoder considers the entire codeword while the practical turbo decoder splits the codewords into two sequences one for each constituent decoder. The required condition for catastrophic behavior is that one of the constituent encoders is catastrophic. Consider the example when the encoder is looping through the states catastrophically and no Hamming weight is accumulated. The other encoder might produce some "1"s for the interleaved input sequence. This increase in the Hamming weight is seen by the ML decoder to break the catastrophic behavior. However, the real turbo decoder will consider every sequence separately.

Since many CC, which are designed for TCs, are designed to optimize the parity "ones" generation for low-weight input sequences, the union bound, for these codes, is in favor to puncturing the systematic bits while simulation will be in favor to puncturing parity bits irrespective of the specific code used.

4.6.2 Puncturing Parity sequences

The fact that the two parity sequences play identical roles at the decoder suggest that puncturing should be distributed equally between the parity branches. It is also intuitive that the punctured bits in any sequence should be well scattered, as adjacent punctured bits destroy the structure of the code. Both conjectures are tested by simulation.

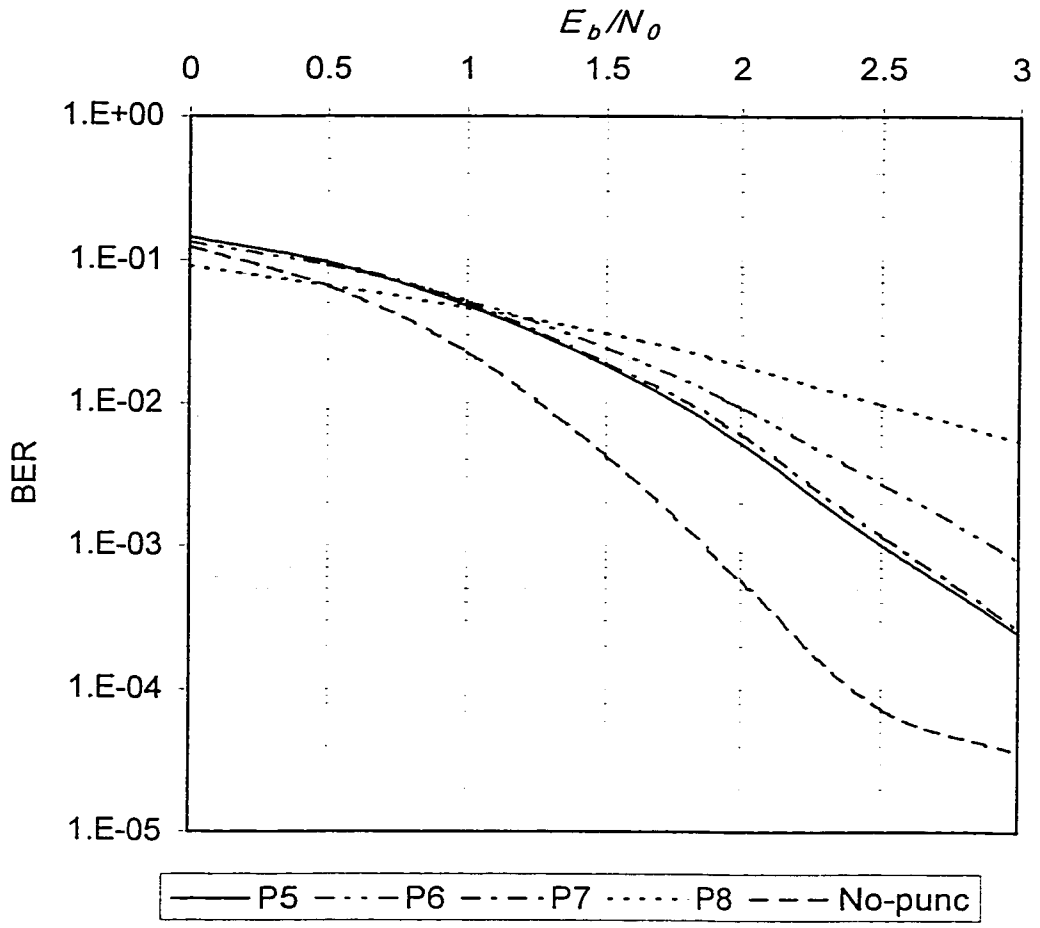
Figure 4.5 shows the performance of TC under different puncturing arrangements of parity sequences. Note that $\mathbf{P}_5$ is best satisfying the above tow conditions, and therefore yields the best performance compared to the other puncturing matrices. In $\mathbf{P}_6$ puncturing is not equally distributed while in $\mathbf{P}_7$ it is equally distributed but not well scattered. As a result the performance of the code under these puncturing patterns is inferior to that of $\mathbf{P}_5$. The puncturing matrix $\mathbf{P}_8$ is the extreme case of puncturing the second parity sequence completely and leaving the first parity sequence intact. In fact this puncturing pattern transforms a TC to a normal convolutional code and hence results in the worst performance.

4.6.3 Permutations of Columns

The procedure for getting the block transition matrix requires computing the period transition matrix. The periodic transition matrix **B** is computed as in Eq. 4.5. Multiplying **B** by itself (k/p) times results in the required block transition matrix. Due to the grouping property of matrix multiplication, cyclic shift will not affect the overall result, assuming the sequence is relatively long. The concept is illustrated as follows:

..**CD**)(**ABCD**)(**ABCD**)(**ABCD**)(**AB**....=...**AB**)(**CDAB**)(**CDBA**)(**CDBA**)(**CD**.... (14.25)

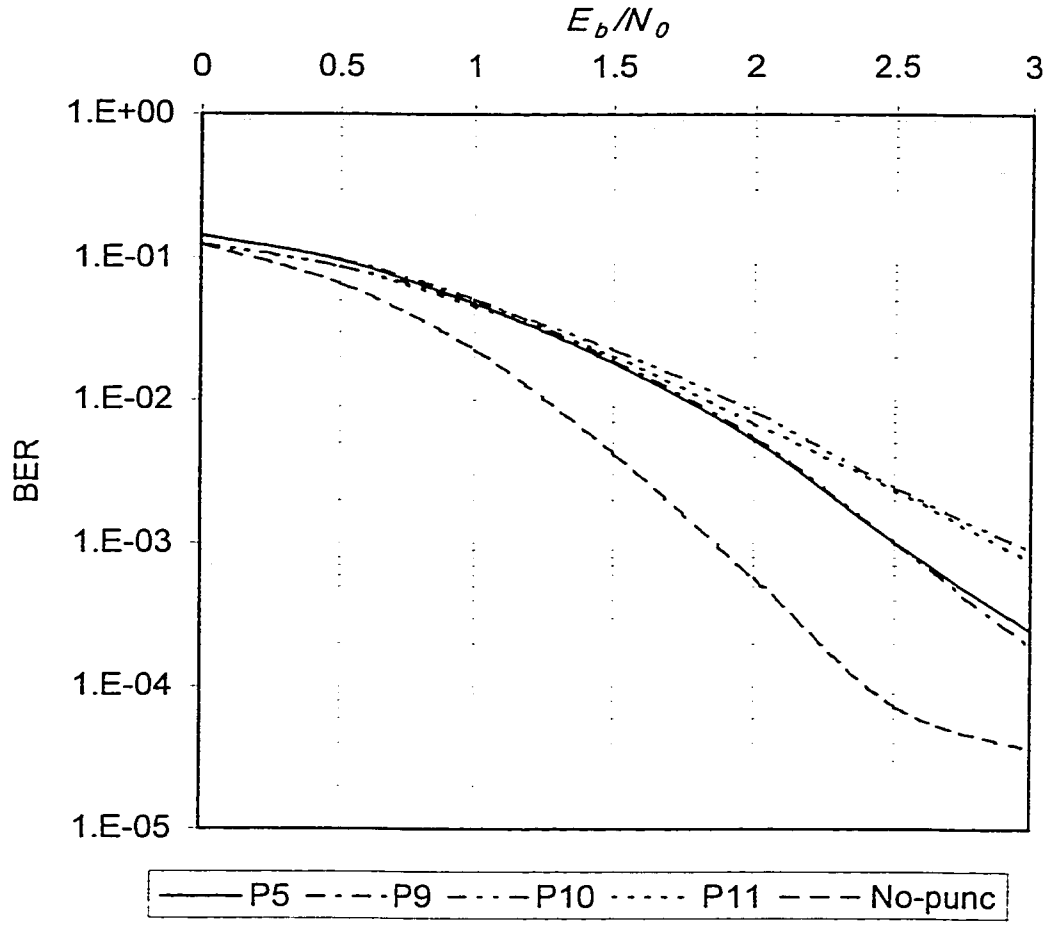
The conclusion can also be arrived at by noting that cyclic shifting of columns does not change the puncturing pattern with respect to the two conditions stated in the previous subsection. The independence of the performance to cyclically shifting the columns of the puncturing matrix is also verified by simulation as shown in Figure 4.6. Note that $\mathbf{P}_5$ and $\mathbf{P}_9$ are columns cyclic shifts of each other and hence they resulted in essentially the same



$$\mathbf{P}_5 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \qquad \mathbf{P}_6 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\mathbf{P}_7 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \qquad \mathbf{P}_8 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 4.5: Performance of TC under different puncturing arrangements of parity sequences.



$$\mathbf{P}_5 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \qquad \mathbf{P}_9 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

$$\mathbf{P}_{10} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \qquad \mathbf{P}_{11} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Figure 4.6: Performance under different permutations on the puncturing matrix

performance. This property has been tested for different interleavers and the behavior was found to be consistent.

Another interpretation of cyclically shifting the columns of the puncturing matrix is the duality with cyclically shifted interleavers. If we alternatively puncture the bits of the two parity branches ($\mathbf{P}_5$ in Figure 4.6), shifting the interleaver map cyclically by one position will interchange the transmitted bits with the punctured bits. Cyclically shifting the puncturing matrix ($\mathbf{P}_9$) will also result in interchanging the transmitted and punctured bits.

It should be noted that permutation of columns other than cyclic shifting might affect the performance. In Figure 4.5, $\mathbf{P}_7$ is obtained from $\mathbf{P}_5$ by interchanging columns 1 and 4. The two patterns yield different performance. Though, the difference is not clear, it will be more pronounceable as the puncturing period increases. This is because for a long puncturing period, the code can behave like a convolutional code by making the punctured bits of one parity branch adjacent to each other. This can be done by permuting the columns.

4.6.4 Permutations of Parity Rows

Now let's discuss the effect of interchanging the pairyt rows of the puncturing matrix,.as in Figure 4.6, $\mathbf{P}_{10}$ vs. $\mathbf{P}_{11}$. Figure 4.6 shows that this has a minimal effect on the performance, which suggests that the roles of parity branches are similar. It is worth mentioning that this property can not be tested using the bounding technique because the uniform interleaver makes the contribution of the CC the same.

4.6.5 Guidelines of a Good Puncturing Matrix

The results of the previous subsections lead to the following set of guidelines for constructing a good puncturing matrix.

1) Puncturing systematic bits should be avoided.

2) Puncturing should be applied equally to parity sequences.

3) Within every parity sequence, punctured bits should be scattered.

4) Columns of the puncturing matrix can be cyclically shifted but not interchanged.

5) Puncturing patterns (rows) of the parity sequences can be interchanged.

In conclusion, the pre mentioned properties on the puncturing matrix serve as useful guidelines in designing the puncturing pattern. However, one has to test the specific code used and avoid the catastrophic behavior.

4.7 Decision Depth of Punctured TC

It is known for convolutional codes that the decision depth must be larger for punctured codes compared to unpunctured codes. As a result of puncturing, metrics corresponding to punctured bits will not add up to the accuracy of the decision.

We will show that this is also true for turbo codes. Figure 4.7 shows the BER performance of the unpunctured TC for decision depths of 3ν, 5ν and10ν where ν is the constraint length of the CC. The figure shows that very little is added by increasing the decision depth from 5ν to 10ν. Figure 4.8 shows the corresponding curves for a punctured

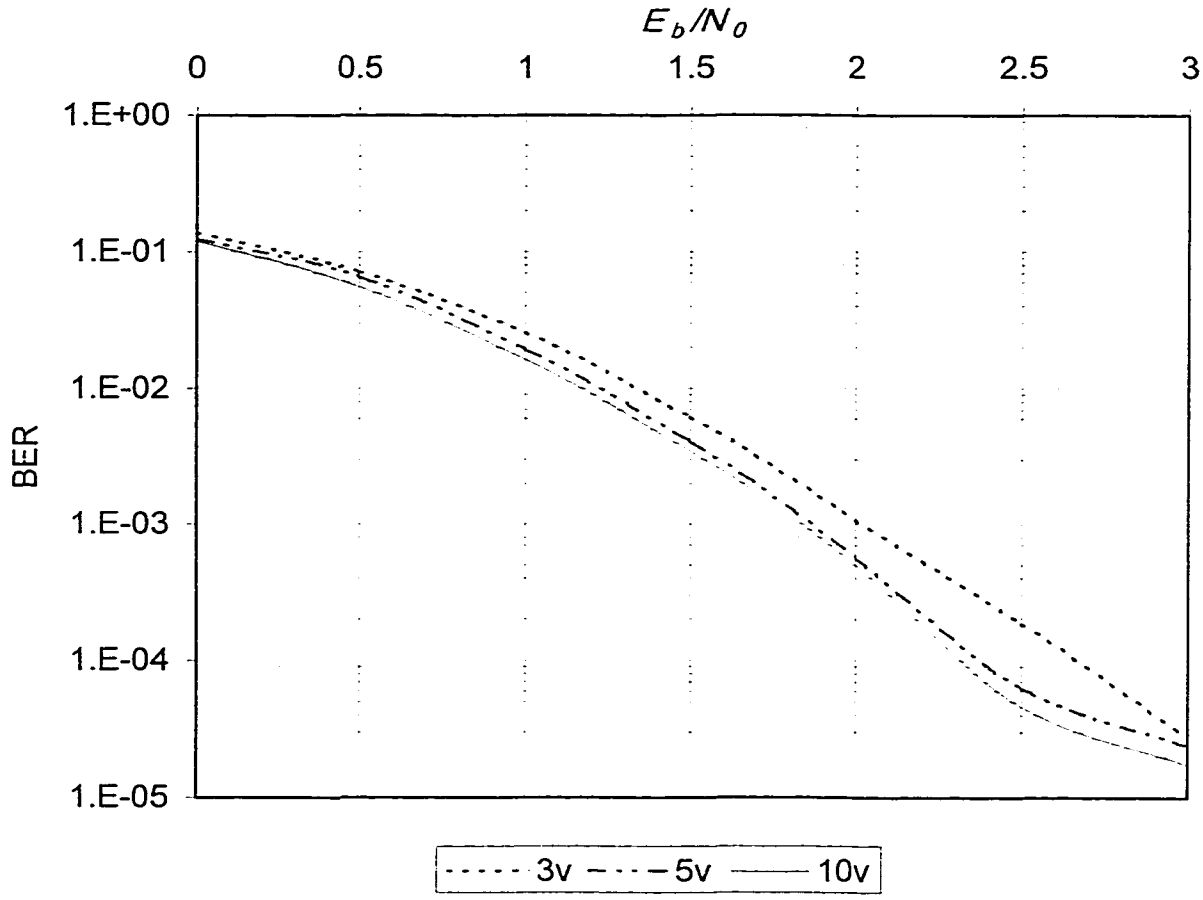


Figure 4.7: Effect of increasing the decision depth for unpunctured codes

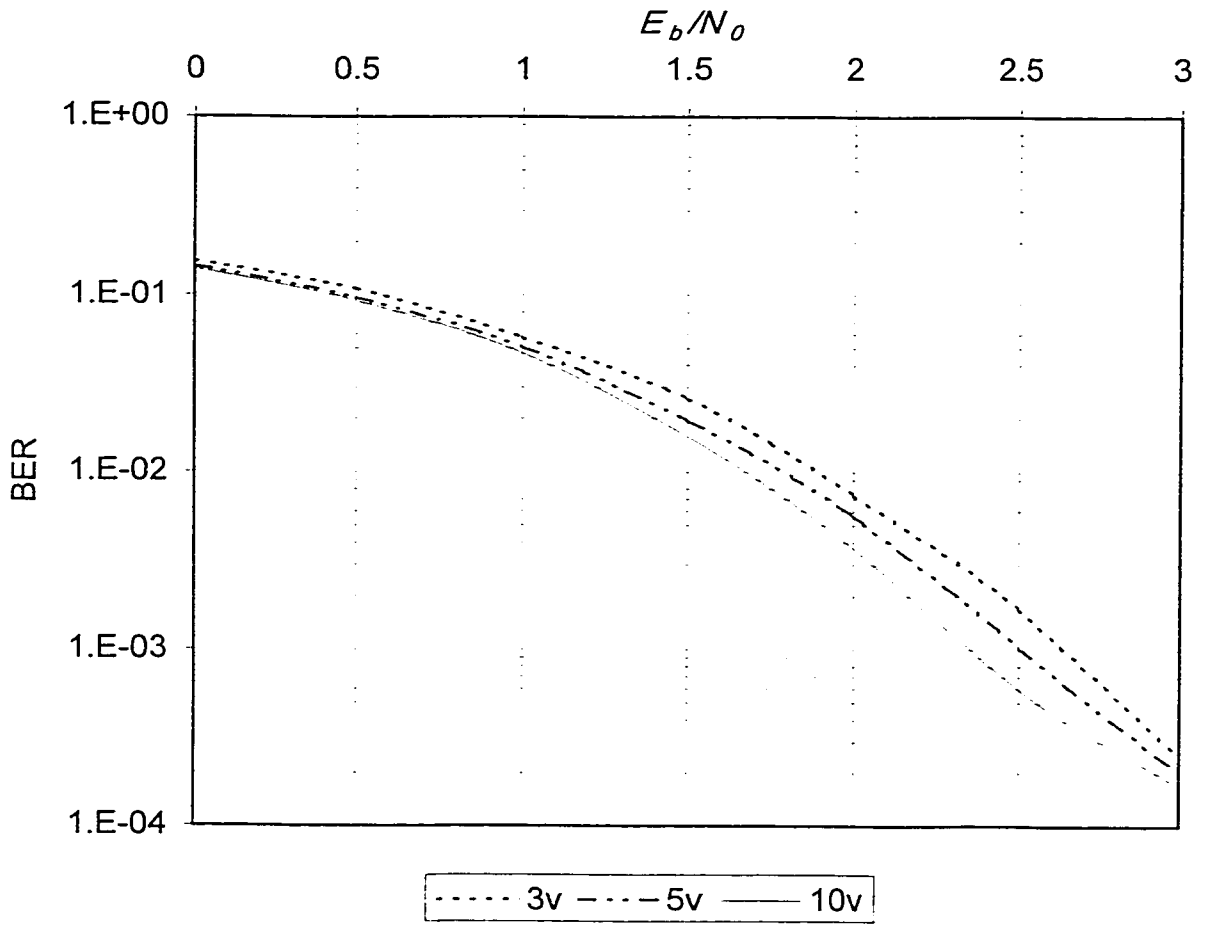


Figure 4.8: Effect of increasing the decision depth for punctured code

TC. Note that there is a more significant improvement in doubling the depth from $5v$ to $10v$.

Hence, the decision depth required for punctured codes is larger than that for unpunctured codes.

CHAPTER 5

SUMMARY, CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK

5.1 Introduction

Before being implemented for real applications, turbo codes have to be well understood. We have in particular studied the effect of various interleaving types and different puncturing patterns on the performance of the code.

Both upper bounding and simulation based on normalized SOVA were used in evaluating the BER over AWGN channel. The obtained results were analyzed and explained.

We believe that this work adds a significant contribution towards better understanding of the code performance. In the following two sections we summarize our findings in regards to interleaving and puncturing. Suggestions for direct extensions of this work are listed in the last section.

5.2 Summary of Interleaving Results

In the following, we summarize the results and findings on the effects of interleaving on the performance of turbo codes, Chapter 3:

- Random interleavers perform near the average.
- The application of matrix interleavers can be regarded as promising in turbo codes with short frames.
- The performance of the code improves for matrix interleavers as both the number of rows and the number of columns increase up to the decision depth.
- Very structured interleavers like the weight-two breaker and cyclic shift interleavers are very poor interleavers
- Not all interleavers improves as $1/k$. One has to consider the specific interleaver under use.
- There is an interaction between the interleaver size and the number of iterations.

- In relation to the above two findings we conclude that increasing the interleaver size improves the code performance provided that the interleaver has a good scattering/randomization capability, and the number of iterations is large enough to exploit the increase in size.

- It is very important to consider the structure of the turbo decoder to get a more accurate evaluation of the code performance.

- The effective number of iterations is almost independent of the interleaver map or interleaver size (provided a good interleaver is used). This implies that the percentage improvement with increasing i is almost the same for any given good interleaver.

- Cyclically shifting the interleaver does not change the performance.

- The performance is not sensitive to interchanging the interleaver and the deinterleaver.

- There is at least a total of $2k$ interleavers modified from the same interleaver which have the same performance (a conclusion from the previous two points). Therefore, there is no single optimum interleaver.

- Properly designing the interleaver can control the asymptotic performance of the code.

5.3 Summary of Puncturing Results

In the following, we summarize the results and findings on the effects of puncturing on the performance of turbo codes, Chapter 4:

- We developed an approach for a more accurate evaluation of the minimum distance and weight distribution of turbo codes. The new approach facilitates the investigation of some puncturing aspects that are otherwise not possible to study.
- The new approach also leads to a tighter upper bound on the BER.
- By examining different puncturing patterns we arrived at the following conclusions.
  - Puncturing systematic bits should be avoided. It was explained how puncturing systematic bits may lead to catastrophic errors. We also explained why such behavior was not observed using the union bound.
  - Puncturing should be applied equally to parity sequences.
  - Within every parity sequence, punctured bits should be scattered.
  - Columns of the puncturing matrix can be cyclically shifted but not interchanged without affecting the code performance.
  - Puncturing patterns of the parity sequences (the rows of the puncturing matrix except the first row) can be interchanged without affecting the code performance.
  - Punctured turbo codes require longer decision depth

5.4 Suggestions for Future Work

Future work may consider the following extensions:

- Testing the performance of turbo codes with matrix interleavers in burst-error channels and compare the performance to random and uniform interleavers.
- Extending the conclusions using different decoding algorithms like Bahl algorithm.
- Evaluating the union bound for specific interleavers. One might consider some structured interleavers like the matrix interleaver.
- The difference between the ML decoder and the practical decoder need to be explained. Does the turbo decoder converge to the ML decoder and under what conditions?

Nomenclature

Abbreviations

AWGN	**A**dditive **W**hite **G**aussian **N**oise
BER	**B**it **E**rror **R**ate
CC	**C**onstituent **C**ode
CIROWEF	**C**onditional **I**nput-**R**edundancy-**O**utput-**W**eight **E**numerating **F**unction
CIRWEF	**C**onditional **I**nput-**R**edundancy **W**eight **E**numerating **F**unction
dB	**d**eci**B**el; Decibel
DEMUX	**DEMU**ltiple**X**er; De-multiplexer
FER	**F**rame **E**rror **R**ate
I	**I**nterleaver
I^{-1}	De-**I**nterleaver
IRWEF	**I**nput-**R**edundancy **W**eight **E**numerating **F**unction
MAP	**M**aximum **A**posteriori **P**robability
ML	**M**aximum **L**ikelihood
PCCC	**P**arallel-**C**oncatenated **C**onvolutional **C**odes
RSC	**R**ecursive **S**ystematic **C**onvolutional
SNR	**S**ignal to **N**oise **R**atio
SOVA	**S**oft **O**utput **V**iterbi **A**lgorithm
TC	**T**urbo **C**ode
UKL	Puncturing pattern suggested by [Jun97]
W2B	**W**eight-two **B**reaker interleaver
WEF	**W**eight **E**numerating **F**unction

Symbols

a	Amount of cyclic shift
$A^C(w,z)$	input redundancy weight enumerating function (IRWEF)
$A_w^C(Z)$	conditional input -redundancy weight enumerating function (CIRWEF)
$A^{C_P}(W,Z)$	IRWEF for the parallel concatenation
B_i	number of codewords with Hamming weight (number of ones) i
B_{free}	number of codewords with Hamming weight (number of ones)=d_{fee}
$B^C(H)$	Weight enumerating function
d_{free}	**free d**istance, weigh of the minimum distance codeword
D_m	Defined in page 29.
$D_w^C(Z)$	conditional input-redundancy-output weight enumerating function (CIROWEF)
D	**D**elay; memory element.
E_b	**E**nergy per bit
H	Dummy variable with its power representing the weight
i^*	Number of effective iterations
j	parity check weight
k	Interleaver length, length of the input sequence to the encoder
m	Number of columns in a matrix interleaver, depth Also used for the overall weight
M	Number of punctured bits in a puncturing period
n	Number of rows in a matrix interleaver, span
n_c	Length of the code word
N_0	**N**oise spectral density
N_{CC}	**N**umber of branches in the **c**onstituent **c**ode
p	Puncturing **p**eriod
p_c	***p***eriod of the code

$\mathbf{P}_i$	i^{th} of the puncturing matrix
$\mathbf{P}$	Puncturing matrix
P_{free}	Asymptotic Probability of error
P_b	bit Error Probability
P_{free}	free distance asymptote
r	Constrain length of the code
R	Code Rate
v	Memory order, i.e. number of memory elements
w	Hamming weight
$w(.)$	Hamming weight operator
$\widetilde{w}_{free}$	average weight of the information sequence causing free distance codeword
W	dummy variable, whose power is the Hamming weigh of the input
X_i	Output of the encoder
y	Weight of systematic branch after puncturing
Y	Dummy variable whose power is y
z_{min}	minimum parity-check weight
Z	dummy variable, whose power is the Hamming weight of the parity
$\pi(x)$	Interleaving function

Bibliography

[1] Berrou, C., Glavieux, A. and Thitimajhima, P., "Near Shannon limit error correcting coding and decoding : turbo-codes," *Proc. Of ICC '93*, Geneva, May 1993, pp. 1064-1070.

[2] Battail, Gérard, "A conceptual framework for understanding turbo codes," *IEEE Journal on Selected Areas in Communications*, vol. 16, No. 2, February 1998, pp. 245-254.

[3] Hagenauer, J. and Papke, L., "Decoding turbo codes with the soft-output Viterbi algorithm (SOVA)," in *Proceedings of International Symposium On Information Theory* (Trondheim, Norway, June 1994), p. 164.

[4] NASA , http://www331.jpl.nasa.gov/public/AllCodesVsSize.GIF

[5] Hagenauer, J., Hoeher, P., "A Viterbi algorithm with soft-decision outputs and its applications," *Proceedings of GLOBECOM '89*, Dallas, Texas, Nov. 1989, pp. 47.11-47.17.

[6] Papke, L. and Robertson, P., "Improved decoding with the SOVA in a parallel concatenated (turbo code) scheme," *IEEE Int. Conference on Communications ICC '96,* 1996, pp. 102-106.

[7] Cheng, J.-F. and McEliece, R. J., "Unit memory Hamming turbo codes," *Proceedings 1995 Proceedings of International Symposium On Information Theory (ISIT)* , 17-22 Sept. 1995 , pp. 33-36.

[8] Blackert, W. J., Hall, E. K., and Wilson, S. G., "Turbo code termination and interleaver conditions," *Electronic Letters*, vol.31, No.24, November 1995, pp. 2082-2084.

[9] Benedetto, Sergio and Montorsi., Guido "Design of parallel concatenated convolutional codes," *IEEE Transactions on Communications*, vol. 44, No. 5, May 1996, pp. 591-600.

[10] Benedetto, Sergio and Montorsi., Guido "Unveiling turbo-codes: some results on parallel concatenated coding schemes," *IEEE Transactions on Information Theory*, vol. 42, No. 2, March 1996, pp. 409-428.

[11] Robertson, Patrick, " Illuminating the structure of code and decoder of parallel concatenated recursive systematic (turbo) codes," *Global Telecommunications Conference, 1994. GLOBECOM '94.* Vol. 3, Dec. 1994, pp. 1298-1303.

[12] Hagenauer, J., "Source-Controlled Channel Decoding," *IEEE Transactions On Communications*, vol.43, No. 9, Sep. 1995, pp. 2449-2457.

[13] Hagenauer, J. and Papke, L. "Iterative decoding of binary block and convolutional codes", *IEEE Transactions On Information Theory*, vol. 42, No. 2, March 1996, pp. 429-445

[14] Anastasopoulos, A. and Polydoros, A., "Soft-decisions per-survivor processing for mobile fading channels," *Vehicular Technology Conference, 1997, IEEE 47th*, vol. 2, pp. 705-709.

[15] Sklar, Bernard, "A primer on turbo code concepts," *IEEE Communications Magazine*, Dec. 1997, pp. 94-102.

[16] McEliece, R. J., Rodemich, E. and Cheng, J.-F. ,"The Turbo Decision Algorithm," *Proceedings of 33rd Allerton Conference On Communications, Control, and Computing, 1995*

[17] Svirid, Yuri V., "Weight distribution of turbo-codes," *Proceedings of 1995 International Symposium on Iinformation Theory*, 17-22 Sep. 1995 p.38.

[18] Valenti, Matthew C., "An introduction to turbo codes," *Not published.*

[19] Duman, T. M. and Salehi, M., "New performance bounds for turbo codes," *IEEE Transactions on Communications,* vol. 46, No. 6, June 1998, pp. 717-723.

[20] Perez, L. C., Seghers, J. and Cotello, D. J., "A distance spectrum interpretation of turbo codes," *IEEE Transactions on Information Theory*, vol. 42, No. 6, Nov. 1996.

[21] Jung, P. and Naβhan, M., "Dependence of the error performance of turbo-codes on the interleaver structure in short frame transmission systems," *Electronics Letters*, Feb. 1994, vol. 30, No. 4, pp.287-288.

[22] Dolinar, S. and Divsalar, D. "Weight distribution for turbo codes using random and nonrandom permutations," *TDA Progress Report 42-122* , Aug. 1995, pp. 56-65.

[23] Khandani, A. K., "Design of turbo-code interleaver using Hungarian method," *Electronics Letters*, vol. 34, No. 1, Jan. 1998, pp. 63-65.

[24] Cain, J. B., Clark, G. C., and J. M. Geist, " Punctured convolutional codes of rate (n-1)/n and simplified maximum likelihood decoding" *IEEE Transactions on Information Theory,* vol. IT-20, Jan. 1979, pp. 79-100.

[25] Chan, W., Geraniotis, E. and Nguyen, V., "An adaptive hybrid FEC/ARQ protocol using turbo codes", *Proceedings of ICUPC 97 - 6th,* vol.2 , 1997, pp.541-545.

[26] G. Caire and E Biglieri, "Parallel concatenated codes with unequal error protection," *IEEE Transactions on Communications* , vol. 46, No. 5, May 1998, pp. 565-567.

[27] Berrou, C. and Glavieux, A.: "Near optimum error correcting coding and decoding: Turbo-Codes", *IEEE Transactions on Communications*, 1996, vol. 44, pp. 1261-1271.

[28] Jung, P. and Plechinger,J. "Performance of rate compatible punctured Turbo-codes for mobile radio applications," *Electronics Lettes*, 1997, vol. 33, No.25, pp. 2102-2103.

[29] Hagenauer, J. , "Rate compatible punctured convolutional codes (RCPC-codes) and their application," *IEEE Transactions on Communications,* Apr. 1988, vol. 36, pp. 389-400.

[30] Sweeney, P. ," Error Control Coding : an Introduction," UK: *Prentice Hall International Ltd.* 1991.

[31] Clark, G. and Cain, J., "Error-Correction Coding for Digital Communications" New York: *Plenum Publishing Corporation* 1988 .

[32] Benedetto, S., Garello, R. and Montorsi, G., "A Search for Good Convolutional Codes to be Used in the Construction of Turbo Codes," *IEEE Transactions on Communication*, vol. 46, No. 9, Sep. 1998, pp. 1101-1105.

[33] Narayanan, K., Stüber, G. and Austin, M., "Physical Layer Design for Packet Data Over IS-136", *Vehicular Technology Conference, 1997, IEEE 47th*, vol. 2, pp.1029-1033.

[34] Mohammadi, A. H. and Khandani, A. K., "Unequal error protection on the turbo-encoder output bits", *Proceedings of ICC'97 - International Conference on Communications*, vol. 2 , 1997, pp. 730-734.

[35] Rowitch, Douglas and Milstein, Laurence, "A Novel Algorithm for the Evaluation of Transfer Function Bounds on the Performance of Turbo Codes", submitted to *IEEE Trans. Inform. Theory,* Aug. 1997.

[36] Divsalar, D. and Pollara, F., "Turbo Codes for PCS Apllications", *TDA Progress Report 42-120,* May. 1995, pp. 54-59. [Bar95] Barbulescu, A. and Pietrobon, S., "Rate Compatible Turbo Codes," *Electronic Letters*, vol. 31, No. 7 , March 1995, pp. 535-536.

[37] Andrew, K. , Heegrad, C. and Kozen, D. , "A Theory of Interleavers," Technical report 97-1634, Computer Science Department, Cornell University, June, 1997.

[38] Lin, Shu and Costello, Jr. , "Error Control Coding: Fundametals and Applications" 1983

[39] Haykin, Simon, "Communication Systems" *John Wiley & Sons*,2nd edition 1983.

[40] Mandelbaum, D. M. "An adaptive-feedback coding scheme using incremental redundancy," *IEEE Transactions Inform. Theory*, vol. IT-20, May 1974, pp. 388-389.

[41] Sklar, Bernard, "Digital Communications: Fundamentals and Applications" *Prentice Hall* 1988.

[42] Benedetto, S. and Montorsi, G., "Role of Recursive Convolutional Codes in Turbo Codes," *Electronics Letters*, vol. 31, No. 11, May 1995, pp. 858-859.

[43] Burket, F. and Caire, G.,"Turbo Decoding with Unequal Error Protection applied to GSM speech," (0-7803-3336-5/96 *IEEE* 1996)

[44] Caire, G. and Lencher, G. ,"Turbo codes with unequal error protection" , *Electronic Letters*, Mar. 1996, vol. 32, No. 7, pp. 629-631.

[45] Divsalar, D. Dolinar, S. and Pollara, F., "Transfer function bounds on the performance of turbo codes" , *TDA Progress Report 42-122*, Aug. 1995, pp. 44-55.

[46] Kuganesan, P. and Ben Letaief, K., "High Speed Data Transmission with Muticode Modulation and Turbo-codes for Wireless Personal Communications" (0-7803-3925-8/97) 1997 *IEEE*

[47] Mugaibel, Ali and Kousa, Maan, "Turbo Codes: Promises and Challenges" *Proceedings of CETEM 1999*, Saudi Aramco, March 1999.

[48] Mugaibel, Ali and Kousa, Maan, "Understanding Turbo Codes", *Proceddings of the 6th Annual IEEE TEM* 1999, Saudi Arabia, April 1999, pp163-167.

[49] Proakis, J., "Digital Communications" *McGraw-Hill* Book Co. , 2nd edition1989.

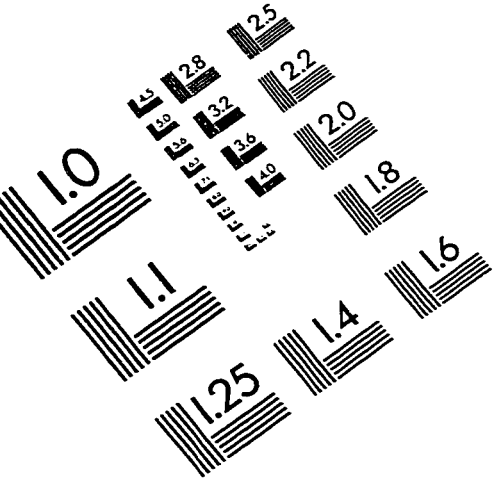
[50] Rowitch, Douglas and Milstein, Laurence , "Rate compatible punctured turbo (RCPT) codes in a hybrid FEC/ARQ system," *Proceeding of GLOBECOM '97,* vol. 4, Nov. 1997. pp.55-59.

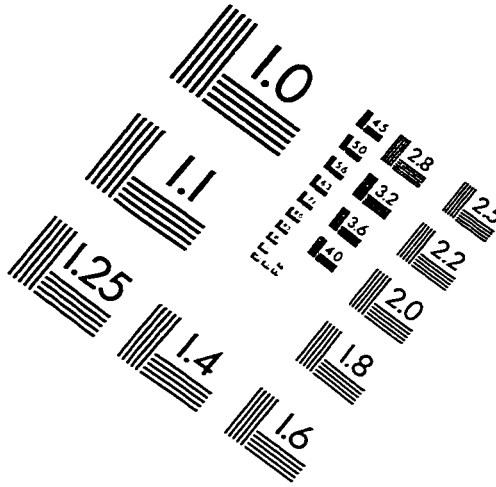
[51] Telatar, E. and Urbanke, R., "On the Ensamble Performance of Turbo Codes" ISIT 1997, Ulm, Germany, June 29- July 4, pp. 105

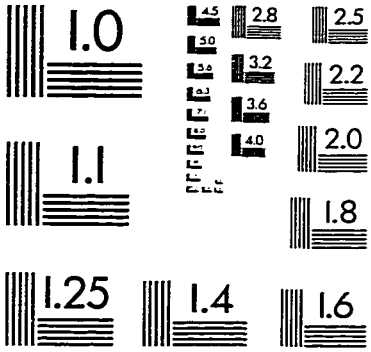
[52] Viterbi, A. J. ,"Convolutional Codes and Their Performance in Communication Systems," IEEE Trans. Commun., Vol. 19, pp.751-772. Oct. 1971.

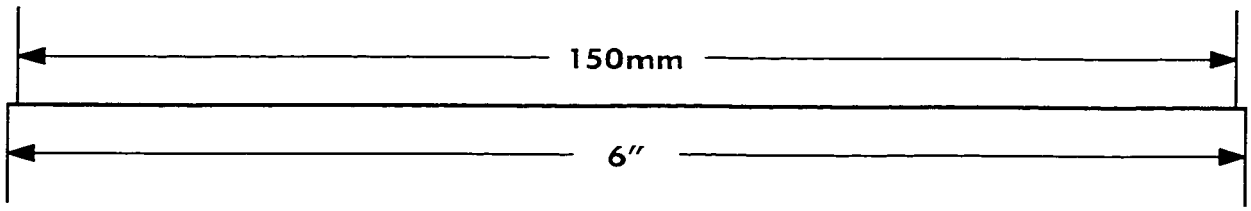
[53] Amberoze, A. , Wade, G. and Tomlinson, M. , "Turbo code tree and code performance," *Electronics Letters*, vol. 34, No. 4, Feb. 1998, pp. 353-354.

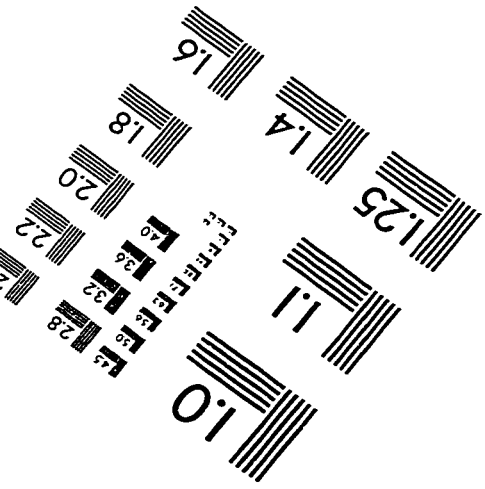
IMAGE EVALUATION TEST TARGET (QA-3)

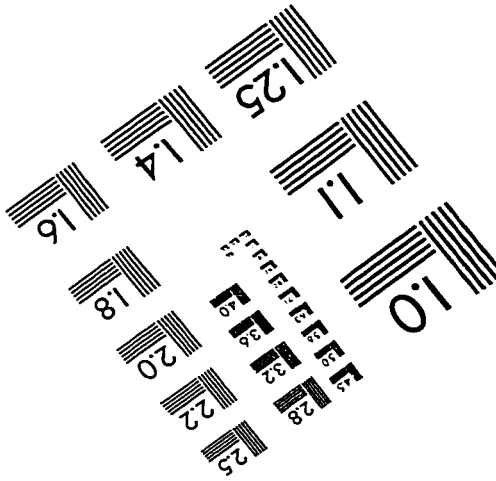












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989

© 1993, Applied Image, Inc., All Rights Reserved