Joint Interleaver Design for Multiple Turbo Codes

Neda Ehtiati

A Thesis
in
The Department
of
Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Applied Science at
Concordia University
Montréal, Québec, Canada

March 2004

Your file Votre référence
ISBN: 0-612-91022-9
Our file Notre référence
ISBN: 0-612-91022-9

ABSTRACT

Joint Interleaver Design for Multiple Turbo Codes

Neda Ehtiati

The outstanding performance of turbo codes at low signal-to-noise ratio is deteriorated in the *error-floor* region, due to the low values of the *free distance* of turbo codes. Interleaver design for turbo codes can improve the minimum distance of the code, by eliminating low-weight input sequences that result in low-weight codewords. Interleaver can also be designed to improve the performance of the sub-optimal iterative decoding methods. While the relating criteria to satisfy these conditions are known, we are not able to find an interleaver that fulfills all these conditions in reasonable time.

In this thesis, we have proposed a *Joint interleaver design methods*, for multiple turbo codes, where we have at least three constituent codes and two interleavers to design. In our proposed methods each interleaver is designed to satisfy some of the conditions and the other conditions are satisfied by the other interleaver. This will decrease the load on one interleaver to satisfy all the conditions at the same time and enable us to satisfy tougher conditions.

It is shown that the proposed *Sequential joint interleaver design* converges very fast and can satisfy very tough conditions, which means better performance at *error-floor* region of the turbo code.

Dedicated to my parents, my sister and my husband

ACKNOWLEDGEMENTS

Firstly, I would like to express my sincere gratitude and thanks to my supervisor Dr. Mohammad Reza Soleymani for his generous help and constant support. I appreciate the way Dr. Soleymani devoted his precious time to discuss with me the details of my research, which was both encouraging and inspiring. His goal has always been to keep the research environment friendly and stress free.

My heartfelt thanks to my parents and my sister, who supported me with their love and understanding. Special thanks to my beloved husbanded who has been involved in every aspect of this research, without his support none of this would have been possible.

I would like to thanks Dr.Hamid Reza Sadjapour and Dr. Daruish Divsalar for their valuable advice at various stages of this research and from Dr. Daruish Divsalar for offering suggestions specially with multiple turbo code simulation and results of density evolution for multiple turbo codes.

I would like to thank Ms. Shirin Esfandiari and Mr. Mohsen Ghotbi and all of my friends at Wireless and Communication Laboratory of their valuable advice and suggestions.

I also extend my thanks to the faculty and staff of Electrical and Computer Engineering department and to my defence committee.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

1.1 Fundamentals

Channel coding has become an indispensable tool in modern communication systems dominated by power and bandwidth constraints. In a digital transmission system, error control is achieved by the use of a channel encoder at the transmitter and a corresponding decoder at the receiver, as depicted in Fig. 1.1. As made clear by Shannon back in 1948 [1], large coding gains (defined as the difference in the signal energy between the uncoded and coded systems required to achieve a given error probability) for a given spectral efficiency can be obtained by encoding the information sequence in long blocks. The largest possible coding gain for a given rate and very long blocks corresponds to the capacity limit derived by Shannon. However, the optimal decoding complexity increases exponentially with the block length, up to a point where decoding becomes physically unmanageable. For about five decades, the goal of coding theorists has been to develop codes that have large equivalent block lengths, yet contain enough structure that practical decoding is possible. From this unsuccessful search stems the folk theorem: *all codes are good, except those that we know how to decode* [2].

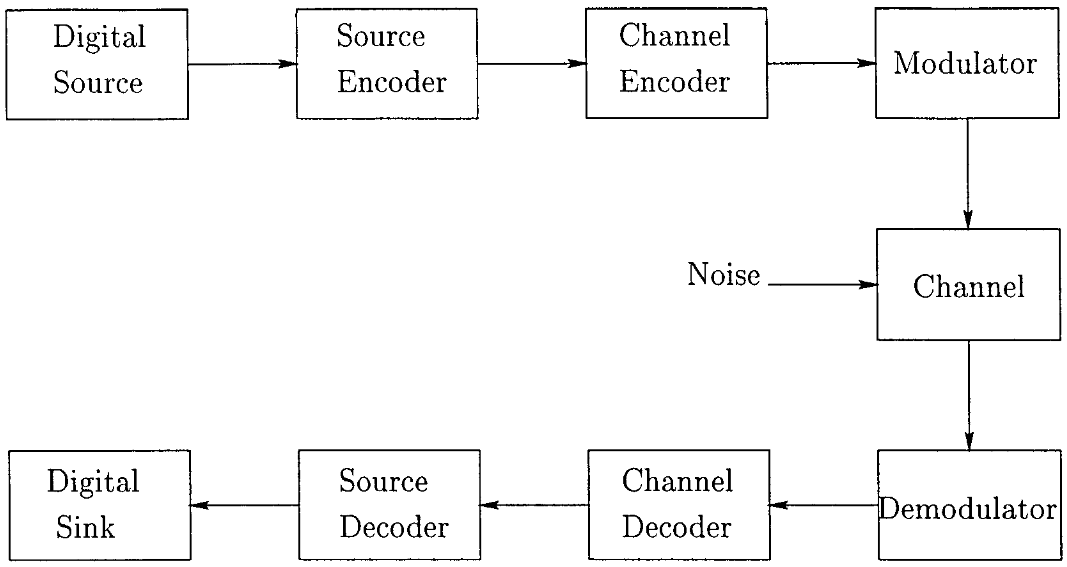


Fig. 1.1: A Typical Digital Communication System

In order to break this barrier methods such as sequential decoding and concatenated coding using Viterbi and Reed-Solomon decoders [3]were introduced. Iterative decoding schemes with "soft-in/soft-out" decoders were proposed in [4], [5]. An important step in the right direction was made by Forney with his thesis work on concatenated codes [6]. Instead of making the code more and more complex in the search for larger gains, he proposed to cascade relatively simple codes (in practise, two codes, inner and outer) to obtain a powerful overall code for which the decoding complexity increased only algebraically with the block size.

Unprecedented results very close to Shannon limit (0.5 dB) were presented in [7]. Berrou et al. introduced for the first time a structure called *turbo code* in which two encoders and one interleaver were organized as the parallel concatenated convolutional code (PCCC) shown in Fig. 1.2.

1.2 Turbo Coding

A Turbo code is constructed as shown in Fig. 1.2. In the general case, a codeword consists of two parts: the uncoded information bits or systematic bits and a set of

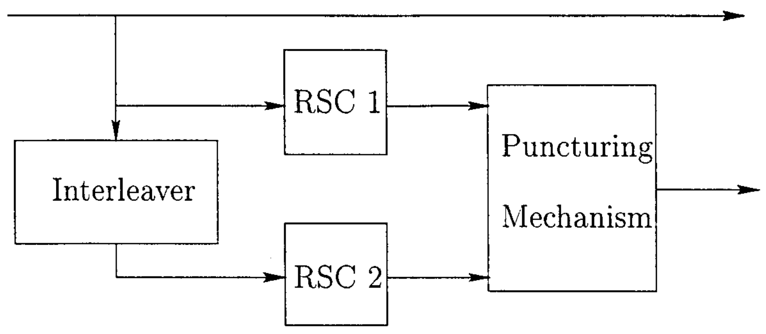


Fig. 1.2: A Typical Turbo Encoder

parity bits. Typically, the encoders used are *Recursive Systematic Convolutional* (RSC) encoders, also in most Turbo codes the encoders used are the same (making the Turbo code symmetric). Two sets of parity bits are used, one which is generated from the non-interleaved data sequence and another generated from an interleaved sequence. The parity bits are usually punctured in order to raise the code rate to desired values. The two encoders are generally terminated (i.e., transformed into block codes) by appending a suitable number of dummy bits at the end of the information word to drive the encoder trellis to a known state, e.g., the zero state. Decoding of turbo codes is performed, by breaking this large complexity by means of a local decoding algorithm working on each individual encoder and by letting each decoder to take advantage of the progress made by the other through the recirculation of an enhanced reliability measure about the information symbols (called *extrinsic* information). This issue raise the need for an algorithm that is capable of accepting "soft" input values (called *a priori* values) and compute "soft" outputs (called *a posteriori* probability).

An algorithm like this was already available [8], waiting since 1974 for practical application. It is the BCJR (acronym from the authors names "Bahl, Cocke, Jelinek and Raviv") algorithm, which computes the *a posteriori* probability (APP) of code and information symbols through two recursions on the code trellis, one in the forward and the other in the backward direction. The computational complexity for

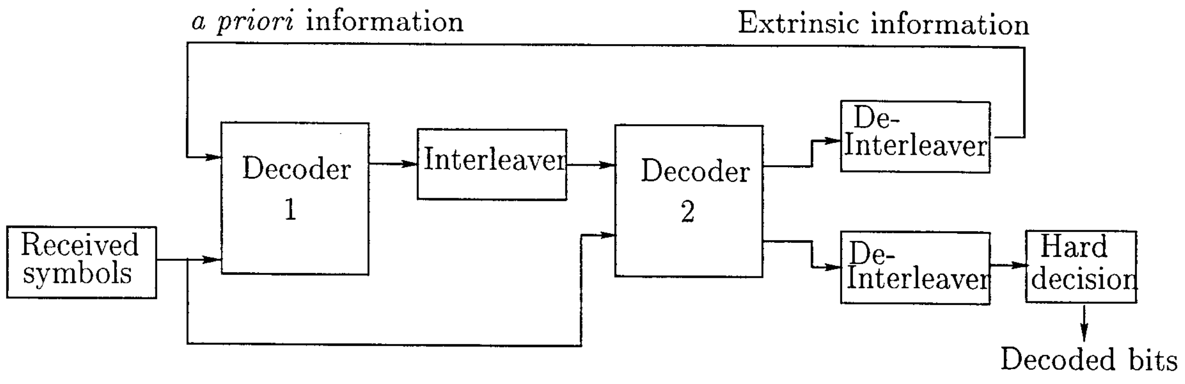


Fig. 1.3: A Typical Turbo Decoder

BCJR algorithm increases linearly with the number of number of trellis states. For the typical two-component Turbo code of Fig. 1.2, the standard iterative decoder is shown in a simplified format in Fig. 1.3.

1.3 Analysis and Design of Turbo Codes

The error probability performance of concatenated codes with interleavers under iterative decoding is represented in general by curves like those depicted in Fig. 1.4. There are three distinctive regions when E_b/N_0 is increased. The first one is the *non convergence region*, where the error probability keeps high nearly constant values. At a certain point, the curves start a rather steep descent to medium-low values of the error probability (the *waterfall region*). Finally, in the third region (the *error floor region*), the slope of the curves decreases significantly and as a result, performance improvement is achieved at the expense of significant additional energy. The waterfall region is dominated by the interleaver gain whereas the error floor region is dictated by the minimum distance of the code. In the first region, the interleaver acts mainly through its size, whereas in the second the kind of interleaver plays a dominant role.

The effect of interleaver on the performance can be examined by studying the

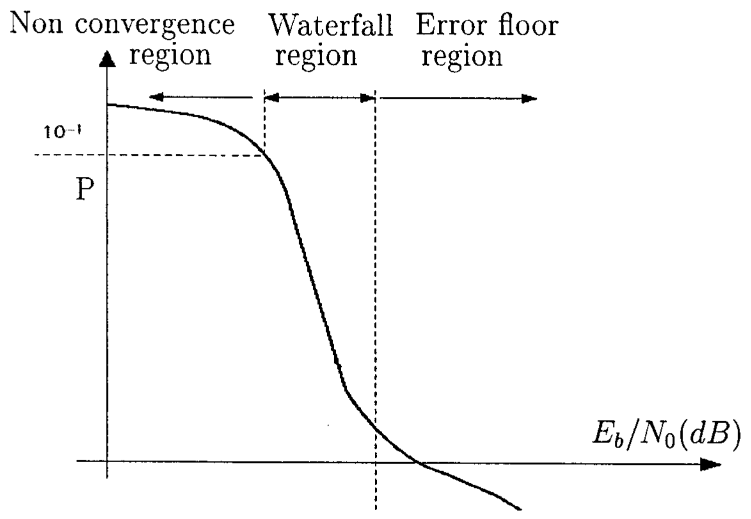


Fig. 1.4: Performance of Turbo Codes

lower bound for error probabilities. The two most important performance measures of a code, the Word Error Probability (WEP), the average probability that the decoder chooses a code word different from the transmitted one, only depends on the code, whereas the Bit Error Probability (BEP), the average probability that the decoder makes a mistake in decoding an information bit, depends on the encoder. A complete description of an (n, k) encoder (k being the size of the input information word and n the code word size) is given by coefficients $A_{w,d}$, which is number of codewords with Hamming weight d, generated by input words of weight w. Upper bounds to the BEP, like the union bound, are based on the input-output coefficients. In particular, for the case of an additive white Gaussian noise channel, binary antipodal modulation and ML soft decoding, the union bound is

$$P_b \leq 1/2 \sum_{w=1}^{k} \sum_{d=d_{min}}^{n} A_{w,d} Q(\sqrt{\frac{2drE_b}{N_0}}) \tag{1.1}$$

where r is the code rate and E_b/N_0 is the signal-to-noise ratio.

Traditionally a good code had been defined as a code with a set of large weights d and in particular, with a large *minimum distance* d_{min} (the lowest Hamming weight of possible codewords excluding the all-zero codeword), which determines the asymptotic performance for large E_b/N_0. So one way to improve the performance at "error-floor" region is to increase the minimum Hamming weight of the code. Looking at Equation (1.1) we see that the bit error probability can be reduced, even in the presence of a small minimum distance, by acting on the coefficients $A_{w,d}$. This is precisely what concatenated codes with interleavers do at "waterfall" region (we will see the details in the following chapters).

To upper bound the BEP for Turbo codes according to Equation (1.1) we need to know its input-output coefficients, assuming the knowledge of those of the constituent encoders. For large interleavers such as those used in practise, computing the input-output coefficients of the overall encoder is a task with overwhelming complexity. We can substitute the interleaver with the theoretical "uniform interleaver", which maps a given input to its all possible combinations with equal probability, in the calculation of BEP [9].

In a concatenated code with interleaver, the role of the interleaver is:

- To improve the code strength by increasing the Hamming weight of the code words. As an example, for the information word that generate low weight codewords at the output of the first encoder. Interleaver should be able to permute such information word, to a sequence that at the output of the second encoder, it generates high weights. In this respect, the interleaver improves the performance, which depends on the code weight spectrum.

- Interleaver can decrease the correlation between the input of two decoders. This, as will be discussed later, will improve the behaviour of the iterative decoding algorithm.

The mathematical analysis in [9] shows that indeed the BEP decreases with the input block size as $1/N$, a phenomenon known as the interleaving gain, provided that both encoders are recursive. We also learn that the most likely error events in turbo codes will be codewords generated by information words of low weight. As a consequence, the choice of the code and interleaver must be aimed at maximizing the weights of the code words generated, by information words with weight $w = 2, 3, 4, \ldots$.

1.4 Multiple Turbo coding

Multiple Turbo Codes (MTC), are parallel concatenation of three or more constituent codes. The advantage of using three or more codes is that the corresponding two or more interleavers have a better chance to break the undesired low-weight input sequences that were not broken by other interleavers. This will improve the performance, as we discussed in the previous section. The disadvantage of multiple turbo codes is that the overall code rate of the system is increased, and in order to get code rate equal to a conventional turbo code we have to puncture more, resulting in weaker constituent codes [10]. Unlike turbo codes where exchange of information between decoders can be done in one way, in the case of MTC, information can be exchanged in different ways between decoders, some of the possible ways to exchange information are shown in Fig. 1.5. Different configurations differ on: first, how many decoders exchange information with a certain decoder and second, when the extrinsic information is updated, i.e., after each decoder finishes its decoding, or after each iteration. In serial(S) method, each decoder receives extrinsic information from only one decoder and this is done after the decoder finishes its operation.

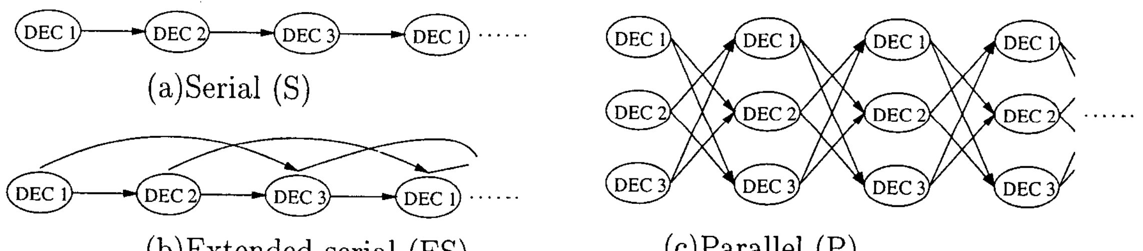


Fig. 1.5: Decoding configurations for Multiple turbo codes

In parallel(P) method, each decoder receives extrinsic information from all other decoders and information is updated after a whole iteration is done. In extended serial(ES) method, each decoder receives extrinsic information from all other decoders and information is updated after each decoder finishes its operation. It is shown in [11] that the extended serial(ES) method has the best performance and serial(S) method has the poorest performance.

1.5 Interleaver Design for Multiple Turbo Codes

As we discussed earlier the outstanding performance of turbo codes at low SNR is deteriorated in *error-floor* region, due to the low values of free distance of turbo codes. As we will see in Chapter 4, improving the minimum distance of the turbo codes, require to eliminate certain patterns that generate low-weight outputs, from the input of encoders. Breaking up the low-weight input sequences and also other criteria for improving the performance of the iterative decoding, can be summarized in certain conditions that need to be satisfied while we are designing the interleaver. While these criteria are known, the fact that holds us from designing a *good* interleaver is convergence issue. Usually for a given length, the interleaver that satisfy all the conditions, will not converge. In these cases we have to loosen the conditions in order that interleaver can converge, which will result in deteriorating the performance.

In case of *Multiple Turbo Codes*, where we have three or more constituent codes and we have at least two interleavers. Each interleaver can be designed to eliminate some of the patterns and other patterns are eliminated by the other interleaver. This will decrease the burden on one interleaver to satisfy all the conditions at the same time and enable us to satisfy tougher conditions.

The joint design of interleaver can be done in two different approaches. In the first approach, we can design the interleavers in parallel, this requires that before designing we decide on which patterns or which conditions will be satisfied by the first interleaver and which ones by the second interleaver. In the second approach we will design the one interleaver first, ignoring the patterns that we can not satisfy and the second interleaver we apply the conditions that the first one have not satisfied.

1.6 Thesis Outline

Chapter 2 introduces the concept of Turbo coding and gives a detailed description of encoding and decoding algorithms. In addition, Multiple Turbo Codes will also be presented, together with different decoding configurations and decoding algorithms. In Chapter 3, we will review the performance bounds for block and convolutional codes, which we will use later to find the performance bounds for turbo codes. We will also analyze the behaviour of turbo codes in different SNR regions and determine the parameter that dominates the performance in each region.

In Chapter 4, the interleaver design criteria will be introduced. These criteria usually aim at improving the distance spectrum of the code or improving the performance of iterative decoding. We also present some existing interleaver design methods. Next we will introduce the simulation results for code-matched interleaver design for 3GPP system as an example of how a proper design of interleaver can effect the performance.

In Chapter 5, we will introduce the idea of joint interleaver design for Multiple turbo codes. The simulation results for Multiple Turbo codes will be given. Parallel and sequential design methods for joint interleaver design will be introduced in detail. Comprehensive comparison between two methods and existing methods will also be given.

Finally, in Chapter 6, we summarize our comments and conclusions and suggest some directions for future work.

Chapter 2

Turbo Codes

Turbo codes, first presented in 1993 [7], have performance close to the Shannon limit. The turbo encoder is built using a parallel concatenation of two constituent codes and proper use of an interleaver between encoders. Decoding is done by using an iterative decoding method such as *maximum a posteriori*(MAP) and exchanging information between component decoders.

In this chapter, we present encoding and decoding of Turbo Codes with two constituent codes in detail. Later, we show how this method can be extended to Turbo codes with more than two component codes i.e. *Multiple Turbo Codes.*

2.1 A Turbo Encoder

A Turbo encoder is formed by parallel concatenation of two or more encoders separated by an interleaver. The first encoder operates directly on the information sequence. Then the information sequence will be passed through an *interleaver*, which has the duty of scrambling the input data. Thus, the second encoder uses the interleaved version of the information sequence. The constituent encoders can be either *block codes* or *convolutional codes.* In this thesis we use turbo encoders with *recursive systematic convolutional* (RCS) encoders as constituent encoders

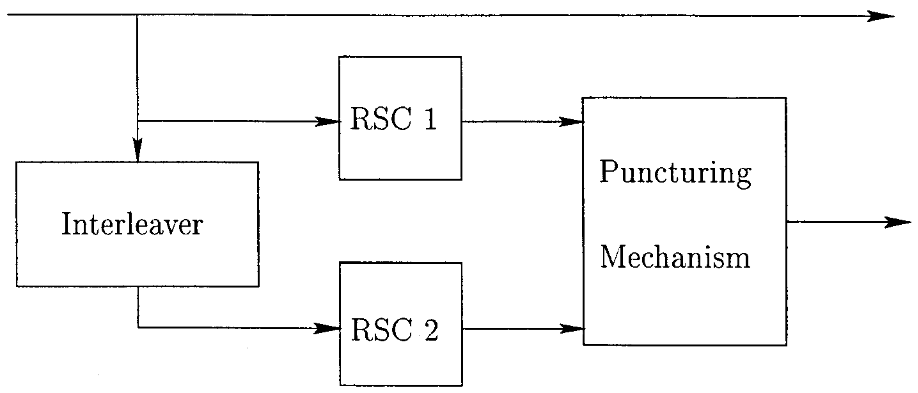


Fig. 2.1: Standard Turbo Encoder with Two RSC Encoders

as depicted in Fig. 2.1. The parity sequence can be punctured in order to get the desired code rate.

2.2 A Turbo Decoder

An iterative Turbo decoder consists of two or more component decoders, connected via an interleaver. Any *soft-in/soft-out* decoder can be used as a component decoder. These decoders accept soft inputs —including *a priori* values— and deliver soft outputs —*a posteriori* information— at their output [12]. In each iteration, the soft output of the previous step is used as an input to the decoder for generating the new soft output. A convolutional encoder with certain memory can be modelled as a finite-state Markov process. The *Maximum a posteriori Probability* (MAP) algorithm is used to estimate the *a posteriori* probabilities of the states and transitions of a Markov source observed through a discrete memoryless channel [8]. In the following we will discuss MAP decoder and how it is used in iterative decoding of the Turbo codes.

2.2.1 Log-Likelihood Algebra

Let U be in $GF(2)$ with the elements $\{+1, -1\}$. The log-likelihood ratio of a binary random variable U, $L_U(u)$, is defined as

$$L_U(u) = ln\frac{P_U(u=+1)}{P_U(u=-1)} \tag{2.1}$$

The log-likelihood ratio $L_U(u)$ is the "soft" value of the random variable U. The sign of the soft value is the hard decision and the magnitude $|L_U(u)|$ is the reliability of this decision.

If the binary random variable U is conditioned on a different random variable or vector Y, then we have a conditioned log-likelihood ratio $L_{U|Y}(u|y)$ with

$$\begin{aligned} L_{U|Y}(u|y) &= \ln\frac{P(u=+1|y)}{P(u=-1|y)} \\ &= \ln\frac{p_{Y|U}(y|u=+1)}{p_{Y|U}(y|u=-1)} + \ln\frac{P_U(u=+1)}{PU(u=-1)} \\ &= L_U(u) + L_{Y|U}(y|u) \end{aligned} \tag{2.2}$$

If we transmit information bit x over a *binary symmetric channel* (BSC) or a Gaussian/fading channel we can calculate the log-likelihood ratio of x conditioned on the matched filter output y

$$\begin{aligned} L(x|y) &= \ln\frac{P(x=+1|y)}{P(x=-1|y)} \\ &= \ln\left(\frac{p(y|x=+1)}{p(y|x=-1)}\cdot\frac{P(x=+1)}{P(x=-1)}\right) \end{aligned} \tag{2.3}$$

we obtain

$$\begin{aligned} L(x|y) &= \ln\frac{\exp(-\frac{E_s}{N_0}(y-a)^2)}{\exp(-\frac{E_s}{N_0}(y+a)^2)} + \ln\frac{P(x=+1)}{P(x=-1)} \\ &= L_c\cdot y + L(x) \end{aligned} \tag{2.4}$$

with $L_c = 4a \cdot \frac{E_s}{N_0}$. For a fading channel, a denotes the fading amplitude whereas for a Gaussian channel we set $a = 1$. L_c is called the reliability value of the channel.

2.2.2 Optimal and Suboptimal Algorithms

BCJR algorithm presented by Bahl et al. in [8], computes *a posteriori* probabilities of code, through two recursions on the code trellis, one in the forward and the other in the backward direction. The estimate result from MAP algorithm is optimal, however, MAP algorithm is not suitable for implementation because of high complexity. In practice usually suboptimal algorithms such as Log-MAP or Max-Log-MAP are used.

2.2.2.1 MAP algorithm

The trellis of a binary feedback convolutional encoder has the structure shown in Fig. 2.2 [12]. Where s_k is the encoder state at time k. u_k is the symbol transmitted from time $k-1$ to k. The trellis states at level $k-1$ and at level k are indexed by the integer s' and s, respectively. The goal of MAP algorithm is to calculate the *a posteriori* values for all bits of information given the received sequence $\mathbf{y}$

$$\begin{aligned} L(\hat{u}_k) &= \ln \frac{P(u_k = +1|\mathbf{y})}{P(u_k = -1|\mathbf{y})} \\ &= \ln \frac{\sum_{u_k=+1}^{(s',s)} p(s', s, \mathbf{y})}{\sum_{u_k=-1}^{(s',s)} p(s', s, \mathbf{y})} \end{aligned} \tag{2.5}$$

The summations are over all existing transitions from state s' to s, with $u_k = +1$ and $u_k = -1$, respectively. Assuming a memoryless channel we can separate $p(s', s, \mathbf{y})$

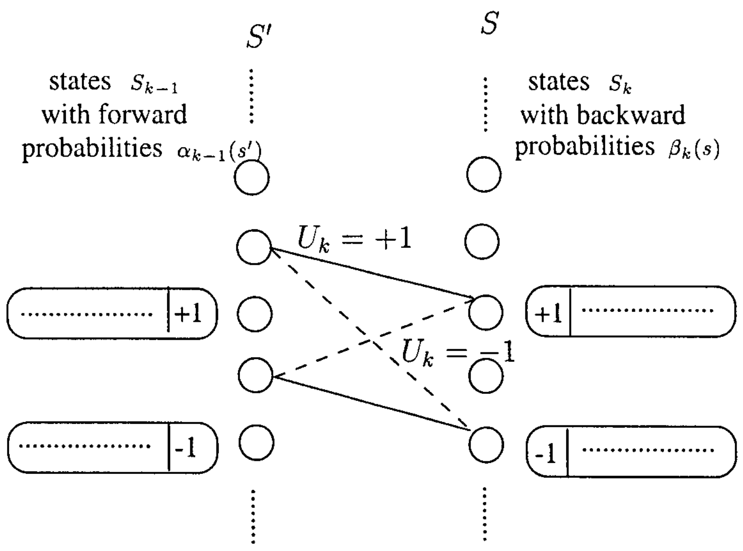

S'
S
states S_{k-1} with forward probabilities $\alpha_{k-1}(s')$
states S_k with backward probabilities $\beta_k(s)$
$U_k = +1$
$U_k = -1$
+1
-1
+1
-1

Fig. 2.2: Trellis Structure of Systematic Convolutional Codes with Feedback Encoders

into three independent probabilities

$$\begin{aligned} p(s', s, \mathbf{y}) &= p(s', \mathbf{y}_{j<k}) \cdot \quad p(s, \mathbf{y}_k \mid s') \quad \cdot p(\mathbf{y}_{j>k} \mid s) \\ &= \underbrace{p(s', \mathbf{y}_{j<k})}_{} \cdot \underbrace{P(s|s') \cdot p(\mathbf{y}_k|s', s)}_{} \cdot \underbrace{p(\mathbf{y}_{j>k} \mid s)}_{} \\ &= \alpha_{k-1}(s') \quad \cdot \quad \gamma_k(s', s) \quad \cdot \quad \beta_k(s) \end{aligned} \tag{2.6}$$

Here $\mathbf{y}_{j<k}$ is the sequence of received symbols $\mathbf{y}_j$ from the beginning of the trellis up to time $k-1$ and $\mathbf{y}_{j>k}$ is the sequence of symbols from time $k+1$ up to the end of the frame.

$\alpha_k(s)$ and $\beta_{k-1}(s')$ can be calculated using the following recursive algorithm:

$$\alpha_k(s) = \sum_{s'} \gamma_k(s', s) \cdot \alpha_{k-1}(s') \tag{2.7}$$

with initial conditions

$$\alpha_0(0) = 1 \quad \textit{and} \quad \alpha_0(s \neq 0) = 0 \tag{2.8}$$

These conditions assume that the encoder has started from state zero.

$$\beta_{k-1}(s') = \sum_{s'} \gamma_k(s', s) \cdot \beta_k(s) \tag{2.9}$$

with boundary conditions

$$\beta_N(0) = 1 \quad \textit{and} \quad \beta_N(s \neq 0) = 0 \tag{2.10}$$

These conditions assume that the encoder has terminated in state zero, which can be done using trellis termination methods. Branch transition probabilities for existing transitions between s' and s are given by

$$\gamma_k(s', s) = P(\mathbf{y}_k | u_k) \cdot P(u_k) \tag{2.11}$$

Using the log-likelihood, the *a priori* probability $P(u_k)$ can be written as [13]

$$\begin{aligned} P(u_k = \pm 1) &= \frac{e^{\pm L(u_k)}}{1 + e^{\pm L(u_k)}} = \left(\frac{e^{\frac{-L(u_k)}{2}}}{1 + e^{-L(u_k)}} \right) \cdot e^{\frac{L(u_k)u_k}{2}} \\ &= A_k \cdot e^{\frac{L(u_k)u_k}{2}} \end{aligned} \tag{2.12}$$

and in a similar way, the conditional probability $p(\mathbf{y}_k|u_k)$ for systematic convolutional codes can be written as [13]

$$\begin{aligned} P(\mathbf{y}_k|u_k) &= P(y_{k,1}|u_k) \cdot \left(\prod_{v=2}^{n} P(y_{k,v}|x_{k,v}) \right) \\ &= B_k \cdot \exp(\frac{1}{2} L_c \cdot y_{k,1} \cdot u_k) \cdot \left(\prod_{v=2}^{n} \exp(\frac{1}{2} L_c \cdot y_{k,v} \cdot x_{k,v}) \right) \\ &= B_k \cdot \exp\left(\frac{1}{2} L_c \cdot y_{k,1} \cdot u_k + \frac{1}{2} \sum_{v=2}^{n} L_c \cdot y_{k,v} \cdot x_{k,v} \right) \end{aligned} \tag{2.13}$$

where $B_k = \left(\frac{P(y_k)\cdot(1+e^{-L(u_k)})\cdot e^{\frac{-L_c\cdot y_k}{2}}}{1+e^{-(L(u_k)+L_c\cdot y_k)}}\right)$ and u_k is the sent information bit and the coded bits are $x_{k,v}$ for $v = 2, \dots, n$ and $y_{k,i}$ for $i = 1, \dots, n$ are received bits.

Therefore, the branch transition operation reduces to the expression

$$\begin{aligned}
\gamma_k(s', s) &= P(\mathbf{y}_k|u_k) \cdot P(u_k) \\
&= B_k \cdot \exp(\frac{1}{2}L_c \cdot y_{k,1} \cdot u_k + \frac{1}{2}\sum_{v=2}^{n} L_c \cdot y_{k,v} \cdot x_{k,v}) \\
&\quad \cdot A_k \cdot \exp((\frac{1}{2}u_k \cdot L(u_k))) \\
&= A_k \cdot B_k \cdot \exp(\frac{1}{2}L_c \cdot y_{k,1} \cdot u_k + \frac{1}{2}\sum_{v=2}^{n} L_c \cdot y_{k,v} \cdot x_{k,v} \\
&\quad + \frac{1}{2}u_k \cdot L(u_k))
\end{aligned} \tag{2.14}$$

Substituting form Equation (2.6) in Equation (2.5) we will have

$$L(\hat{u_k}) = \ln \frac{\sum_{u_k=+1}^{(s',s)} \gamma_k(s', s) \cdot \alpha_{k-1}(s') \cdot \beta_k(s)}{\sum_{u_k=-1}^{(s',s)} \gamma_k(s', s) \cdot \alpha_{k-1}(s') \cdot \beta_k(s)} \tag{2.15}$$

The terms A_k and B_k in Equation (2.14) are equal for all transitions from level $k-1$ to level k and hence will cancel out in the ratio of Equation (2.15). Therefore, the branch transition operation to be used in Equation (2.7) and Equation (2.9), is reduced to the expression

$$\exp(\frac{1}{2}u_k(L_c \cdot y_{k,1} + L(u_k))) \cdot \gamma_k^{(e)}(s', s) \tag{2.16}$$

with

$$\gamma_k^{(e)}(s', s) = \exp(\frac{1}{2}\sum_{v=2}^{n} L_c \cdot y_{k,v} \cdot x_{k,v}) \tag{2.17}$$

Thus, the log-likelihood ratio becomes

$$\begin{aligned} L(\hat{u_k}) &= L_c \cdot y_{k,1} + L(u_k) + \ln \frac{\sum_{u_k=+1}^{(s',s)} \gamma_k^{(e)}(s',s) \cdot \alpha_{k-1}(s') \cdot \beta_k(s)}{\sum_{u_k=-1}^{(s',s)} \gamma_k^{(e)}(s',s) \cdot \alpha_{k-1}(s') \cdot \beta_k(s)} \\ &= L_c \cdot y_{k,1} + L(u_k) + L_e(\hat{u_k}) \end{aligned} \tag{2.18}$$

where

$$L_e(\hat{u_k}) = \ln \frac{\sum_{u_k=+1}^{(s',s)} \gamma_k^{(e)}(s',s) \cdot \alpha_{k-1}(s') \cdot \beta_k(s)}{\sum_{u_k=-1}^{(s',s)} \gamma_k^{(e)}(s',s) \cdot \alpha_{k-1}(s') \cdot \beta_k(s)} \tag{2.19}$$

According to Equation (2.18), the soft output for any random information bit can be represented in three additive terms, where $L_e(\hat{u_k})$ is *extrinsic* information and $L_c \cdot y_{k,1}$ is the channel values and $L(u_k)$ is the *a priori* values. Using Equation (2.18) and Equation (2.19), MAP algorithm can calculate *a posteriori* information for all symbols.

2.2.2.2 Log-MAP and MAX-Log-MAP algorithm

MAP decoding algorithm discussed in Section 2.2.2.1, requires large memory and a large number of operations involving exponentiations and multiplications. This issue makes it very complex for implementation in many communication systems.This algorithm can be simplified using logarithms of $\alpha_k(s)$, $\gamma_k(s',s)$, $\beta_k(s)$ [13]. Using the logarithms we will have

$$\begin{aligned} \overline{\gamma}_k(s',s) &= \ln \gamma_k(s',s) \\ &= \frac{1}{2} L_c \cdot y_{k,1} \cdot u_k + \frac{1}{2} \sum_{v=2}^{n} L_c \cdot y_{k,v} \cdot x_{k,v} + \frac{1}{2} u_k \cdot L(u_k) \end{aligned} \tag{2.20}$$

$$\begin{aligned}\overline{\alpha}_k(s) &= \ln \alpha_k(s) \\ &= \ln\left(\sum_{s'} e^{\overline{\gamma}_k(s',s)} \cdot e^{\overline{\alpha}_{k-1}(s)}\right) \\ &= \ln\left(\sum_{s'} e^{\overline{\gamma}_k(s',s)+\overline{\alpha}_{k-1}(s)}\right) \qquad (2.21)\end{aligned}$$

with the initial conditions

$$\overline{\alpha}_0(0) = 0 \quad and \quad \overline{\alpha}_0(s \neq 0) = -\infty \qquad (2.22)$$

$$\begin{aligned}\overline{\beta}_{k-1}(s') &= \ln \beta_{k-1}(s') \\ &= \ln\left(\sum_{s} e^{\overline{\gamma}_k(s',s)} \cdot e^{\overline{\beta}_k(s')}\right) \\ &= \ln\left(\sum_{s} e^{\overline{\gamma}_k(s',s)+\overline{\beta}_k(s')}\right) \qquad (2.23)\end{aligned}$$

with boundary conditions

$$\overline{\beta}_N(0) = 0 \quad and \quad \overline{\beta}_N(s \neq 0) = -\infty \quad \cdot \qquad (2.24)$$

Therefor, the log-likelihood ratio can be expressed as

$$\begin{aligned}L(\hat{u_k}) &= \ln \frac{\sum_{u_k=+1}^{(s',s)} e^{\overline{\gamma}_k(s',s)} \cdot e^{\overline{\alpha}_{k-1}(s')} \cdot e^{\overline{\beta}_k(s)}}{\sum_{u_k=-1}^{(s',s)} e^{\overline{\gamma}_k(s',s)} \cdot e^{\overline{\alpha}_{k-1}(s')} \cdot e^{\overline{\beta}_k(s)}} \\ &= \ln \frac{\sum_{u_k=+1}^{(s',s)} e^{\overline{\gamma}_k(s',s)+\overline{\alpha}_{k-1}(s')+\overline{\beta}_k(s)}}{\sum_{u_k=-1}^{(s',s)} e^{\overline{\gamma}_k(s',s)+\overline{\alpha}_{k-1}(s')+\overline{\beta}_k(s)}} \\ &= \ln\left(\sum_{u_k=+1}^{(s',s)} e^{\overline{\gamma}_k(s',s)+\overline{\alpha}_{k-1}(s')+\overline{\beta}_k(s)}\right) \\ &\quad - \ln\left(\sum_{u_k=-1}^{(s',s)} e^{\overline{\gamma}_k(s',s)+\overline{\alpha}_{k-1}(s')+\overline{\beta}_k(s)}\right) \qquad (2.25)\end{aligned}$$

It is known that

$$\begin{aligned} E(x,y) &= \ln(e^x + e^y) \\ &= max(x,y) + \ln(1 + e^{-|x-y|}) \end{aligned} \tag{2.26}$$

which can be extended to general form of

$$\begin{aligned} E(x_1, x_2, \ldots, x_k) &= \ln \sum_{i=1}^{k} e^{x_i} \\ &= max(x_i) + \ln \sum_{i=1}^{k} e^{x_i - max(x_i)} \end{aligned} \tag{2.27}$$

So whenever we have to calculate the sum of exponentials we can use Equation (2.27) instead. The Equation (2.27) can be used to simplify the calculation for log-likelihood ratio of Equation (2.25) and forward/backward recursion of Equation (2.21) and Equation (2.23). These calculations can be further simplified if instead of Equation (2.27) we use the following approximation

$$E(x,y) = \ln(e^x + e^y) \approx max(x,y) \tag{2.28}$$

$$E(x_1, x_2, \ldots, x_k) = \ln \sum_{i=1}^{k} e^{x_i} \approx max(x_i) \tag{2.29}$$

This approximation will result in a sub-optimal method which is called MAX-Log-MAP algorithm.

2.2.3 Iterative Decoding Algorithm

Iterative decoders, usually use a "Soft-in/Soft-out" (SISO) decoder, such as a MAP decoders depicted in Fig. 2.3. As we have seen in Section 2.2.2, the output of the

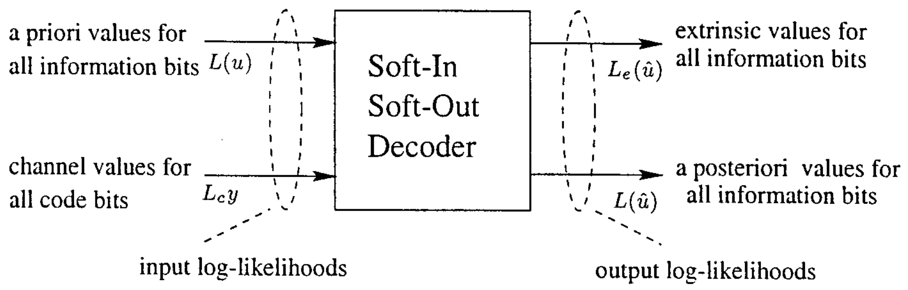

a priori values for all information bits
$L(u)$
channel values for all code bits
$L_c y$
input log-likelihoods
Soft-In Soft-Out Decoder
extrinsic values for all information bits
$L_e(\hat{u})$
a posteriori values for all information bits
$L(\hat{u})$
output log-likelihoods

Fig. 2.3: Soft-in/Soft-out Decoder

MAP decoder will be the soft output $L(\hat{u})$ and extrinsic information $L_e(\hat{u})$ for all information bits. The input to the SISO decoder is *a priori* values $L(u)$ for all information bits and also the channel value $L_c \cdot y$. The output of the SISO decoder is extrinsic information for all information bits. Extrinsic information contains the soft output information from all the other coded bits in the code sequence but does not contain the information in $L(u)$ and $L_c \cdot y$. For systematic codes, the soft output for the information bit u will be represented as the sum of three terms

$$L(\hat{u}) = L_c \cdot y + L(u) + L_e(\hat{u}) \tag{2.30}$$

This shows that there are three independent estimates for the log-likelihood ratio of the information bits: the channel values $L_c \cdot y$, the *a priori* vales $L(u)$ and the values $L_e(\hat{u})$ extrinsic information generated using the code constraint.

The procedure of iterative decoding with two component SISO decoders is shown in Fig. 2.4. For the first iteration, assuming equally likely information bits, the *a priori* information $L(u) = 0$ is initialized to zero, then the extrinsic information at the output of first decoder is

$$L_e^1(\hat{u}) = L^1(\hat{u}) - [L_c \cdot y + L(u)] \tag{2.31}$$

This extrinsic information from first decoder is passed to the second decoder, which uses $L_e^1(\hat{u})$ as the *a priori* value in place of $L(u)$ to compute $L^2(\hat{u})$. Hence, the

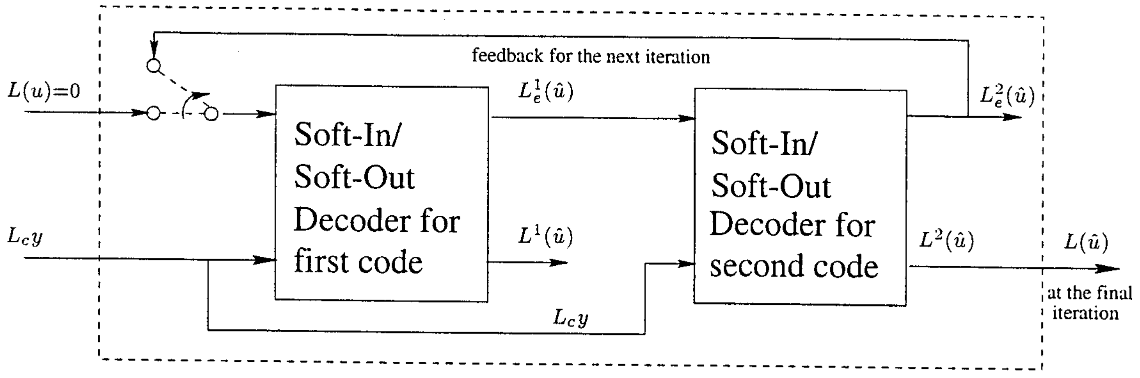


Fig. 2.4: Iterative Turbo Decoding

extrinsic information value computed by the second decoder is

$$L_e^2(\hat{u}) = L^2(\hat{u}) - [L_c \cdot y + L_e^1(\hat{u})] \tag{2.32}$$

This value will be used by first decoder, in the next iteration, as *a priori* value. This procedure is repeated for each iteration, until the improvement through the iterations becomes marginal. The final soft decision output is a combination of the last two extrinsic values with the received values, as follow

$$L(\hat{u}) = L_e^1(\hat{u}) + L_e^2(\hat{u}) + L_c \cdot y \tag{2.33}$$

2.3 Trellis Termination

A turbo encoder operating on each block of information bits can be regarded as a block code. Each block, will start from zero state and will be driven to zero state by means of trellis termination. For a non-recursive encoder, a tail of zero bits would terminate the encoder, whereas for recursive encoders, the required tail bits for trellis termination depends on the state of the encoder.

The circuit in Fig. 2.5 can be used for trellis termination for recursive encoders.

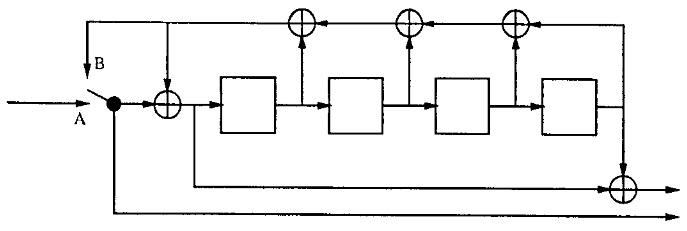


Fig. 2.5: Trellis Termination

For normal operation the switch is in position A, and for the last v bits (v is the memory of the encoder) the switch is in position B.

2.4 Interleaving

As it was shown in Section 2.1, parallel concatenation of codes is possible using an *interleaver* between the two encoders. Interleaver will scramble the input data or in other words change the order of the elements in the sequence. The basic role of the interleaver is to construct a long random code.

Design of interleaver has a remarkable effect on the performance of the code. There are two major criteria involved in the design of the interleaver: first, the distance spectrum properties of the code and second, the correlation between the output of each decoder corresponding to its information and extrinsic input sequences.

As we will see in Chapter 3, performance of the turbo codes specially at error-floor region depends on their distance spectrum. Interleaver has a vital role in the shaping of the distance spectrum of the code. Interleaver can be used to improve the performance of turbo codes by increasing the minimum Hamming distance of the code.

Performance of iterative decoding is dependent on the quality of the extrinsic information, which is the information exchanged between the constituent encoders. When an output is highly correlated to a few positions of input sequence, it becomes very sensitive to channel noise at these positions. The choice of the interleaver affects the degree of correlation between extrinsic inputs and thereby the performance of iterative decoding.

Many interleaving strategies have been proposed including block interleavers, Odd-Even block interleavers, Random and S-random interleavers and Code matched interleavers. We will introduce these methods in Chapter 4 in detail and study their effect on the performance of turbo codes.

2.5 Multiple Turbo Codes

Turbo codes discussed in previous section, mostly consist of two constituent codes. In this section, we discuss *multiple turbo codes* (MTC) where we have three or more constituent codes. In general, the advantage of using three or more constituent codes is that the corresponding two or more interleavers have a better chance of breaking the undesired low weight input sequences that were not broken by other interleavers. The disadvantage is that, for an overall desired code rate, each code must be punctured more, resulting in weaker constituent codes [10].

Encoders for multiple turbo code are similar to those of turbo codes with two constituent codes. As shown in Fig. 2.6, for a multiple turbo code with three constituent codes, we need two interleavers (π_1, π_2) and the output codeword contain the systematic sequence and three sets of parities for each information block. The parity bits can be punctured if needed.

Unlike turbo codes with two constituent codes, where there is one way to exchange information between two decoders, in *multiple turbo codes* there are different configurations for exchanging information between decoders. The choice of proper configurations becomes a design issue and some configurations perform better than others [11]. Different configurations are presented in Section 2.5.1 and in Section 2.5.2 the iterative decoding for multiple turbo codes is discussed.

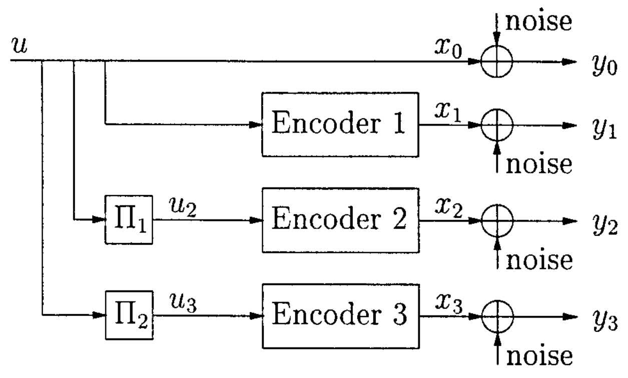


Fig. 2.6: Multiple Turbo Encoder

2.5.1 Decoding Configurations for Multiple turbo Codes

The superior performance of turbo codes, is mainly the result of information exchange between decoders. In turbo codes with two constituent codes there is just one possible configuration for exchanging the information but in *multiple turbo codes* different configurations are possible.

Each decoder exchanges information with some or all of other decoders and extrinsic information can be updated after each decoder finishes one decoding cycle, or after one whole iteration. Possible configurations have been presented in [11], as depicted in Fig. 2.7. Some commonly used configurations are Parallel(P), where decoder receive information from all decoders after each iteration and Extended serial(ES) configuration where each decoder again receives information from all other decoders but this is done after each decoder finishes its operation. Simulation results show that choice of configuration can have a huge effect on the performance. Results in [11] shows that Extended serial (ES), Extended master-slave (EMS) and Parallel (P) configurations have the best and the Extended parallel 1 (EP1) configuration has the worst performance among all configurations.

The choice of configuration also affects the E_b/N_0 threshold value above which the decoding procedure will converge. Let Th_x denote the E_b/N_0 threshold value above which iterative decoding converges for decoding configuration x. In [11] it is

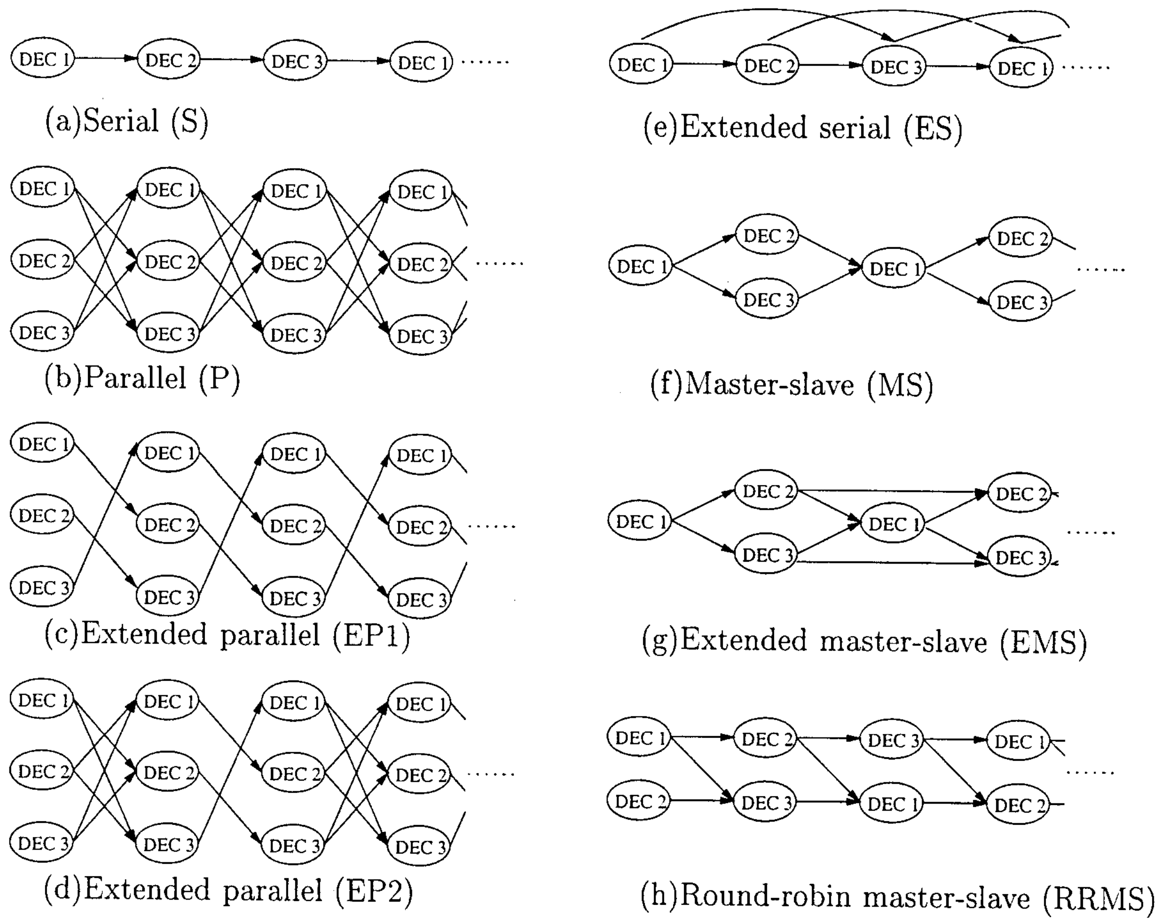

DEC 1
DEC 2
DEC 3
(a)Serial (S)
(b)Parallel (P)
(c)Extended parallel (EP1)
(d)Extended parallel (EP2)
(e)Extended serial (ES)
(f)Master-slave (MS)
(g)Extended master-slave (EMS)
(h)Round-robin master-slave (RRMS)

Fig. 2.7: Decoding Configurations for Multiple Turbo Codes

shown that

$$Th_{ES} = Th_{EMS} = Th_P \leq Th_{EP2}, Th_{MS}, Th_{RRMS} \leq Th_{EP1} = Th_S \qquad (2.34)$$

2.5.2 Iterative Decoding for Multiple Turbo Codes

In this section, we review decoding algorithm for multiple turbo codes. Information sequence is transmitted through an Additive White Gaussian Noise (AWGN) channel with zero-mean, unit variance and $\rho = \sqrt{2rE_b/N_0}$, where E_b/N_0 is the signal to noise ratio (SNR). The combination of the permuter and the encoder can be considered as a block code with input $\mathbf{u}$ and outputs $\mathbf{x}_i$, $i = 1, 2, 3$ ($\mathbf{x}_0 = \mathbf{u}$) and the corresponding received sequences $\mathbf{y}_i$, $i = 0, 1, 2, 3$. As shown in Fig. 2.6.

$$\mathbf{y}_i = \rho(2\mathbf{x}_i - 1) + \mathbf{n_i} \tag{2.35}$$

The optimum bit decision metric on each bit is (for data with uniform a priori probabilities) [14]

$$L_k = \log \frac{\sum_{\mathbf{u}:u_k=1} P(\mathbf{y}_0|\mathbf{u})P(\mathbf{y}_1|\mathbf{u})P(\mathbf{y}_2|\mathbf{u})P(\mathbf{y}_3|\mathbf{u})}{\sum_{\mathbf{u}:u_k=0} P(\mathbf{y}_0|\mathbf{u})P(\mathbf{y}_1|\mathbf{u})P(\mathbf{y}_2|\mathbf{u})P(\mathbf{y}_3|\mathbf{u})} \tag{2.36}$$

In practice, Equation (2.36) can not be computed for large N because the permutations π_2 , π_3 imply that $\mathbf{y}_2$ and $\mathbf{y}_3$ are no longer simple convolutional encoding of $\mathbf{u}$. In order to evaluate Equation (2.36), we will use the following approximation [15]

$$P(\mathbf{u}|\mathbf{y}_i) \simeq \prod_{k=1}^{N} \tilde{P}_i(u_k) \tag{2.37}$$

Note that $P(\mathbf{u}|\mathbf{y}_i)$ is not separable in general. However, for $i = 0$, $P(\mathbf{u}|\mathbf{y}_0)$ is separable. Using Bayes' rule we have

$$\begin{aligned} P(\mathbf{y}_i|\mathbf{u}) &= \frac{P(\mathbf{u}|\mathbf{y}_i)P(\mathbf{y}_i)}{P(\mathbf{u})} \\ &= \frac{\prod_{k=1}^{N} \tilde{P}_i(u_k)P(\mathbf{y}_i)}{P(\mathbf{u})} \end{aligned} \tag{2.38}$$

Defining $\tilde{L}_{ik}$ which is the *a priori* information, we have

$$\tilde{P}_i(u_k) = \frac{e^{u_k \tilde{L}_{ik}}}{1 + e^{u_k \tilde{L}_{ik}}} \tag{2.39}$$

substitute Equation (2.39) in Equation (2.38), we get

$$P(\mathbf{y}_i|\mathbf{u}) = \frac{\prod_{k=1}^{N} \frac{e^{u_k \tilde{L}_{ik}}}{1+e^{u_k \tilde{L}_{ik}}} P(\mathbf{y}_i)}{P(\mathbf{u})} \tag{2.40}$$

If we expand $P(\mathbf{y}_i|\mathbf{u})$ for $i = 0, 2, 3$ in Equation (2.36), using Equation (2.40) the value $P(\mathbf{y}_i)$ for $i = 0, 2, 3$ will cancel out in the ratio and $P(\mathbf{u})$ will also cancel out because we have assumed data with uniform a priori probabilities. Equation (2.36) will simplify to

$$\begin{aligned} L_k &= \log \frac{\sum_{\mathbf{u}:u_k=1} P(\mathbf{y}_1|\mathbf{u}) \prod_j e^{u_j(\tilde{L}_{0j}+\tilde{L}_{2j}+\tilde{L}_{3j})}}{\sum_{\mathbf{u}:u_k=0} P(\mathbf{y}_1|\mathbf{u}) \prod_j e^{u_j(\tilde{L}_{0j}+\tilde{L}_{2j}+\tilde{L}_{3j})}} \\ &= \tilde{L}_{ok} + \tilde{L}_{2k} + \tilde{L}_{3k} + f(\mathbf{y}_1, \tilde{\mathbf{L}}_0, \tilde{\mathbf{L}}_2, \tilde{\mathbf{L}}_3) \end{aligned} \tag{2.41}$$

where

$$f(\mathbf{y}_1, \tilde{\mathbf{L}}_0, \tilde{\mathbf{L}}_2, \tilde{\mathbf{L}}_3) = \log \frac{\sum_{\mathbf{u}:u_k=1} P(\mathbf{y}_1|\mathbf{u}) \prod_{j\neq k} e^{u_j(\tilde{L}_{0j}+\tilde{L}_{2j}+\tilde{L}_{3j})}}{\sum_{\mathbf{u}:u_k=0} P(\mathbf{y}_1|\mathbf{u}) \prod_{j\neq k} j e^{u_j(\tilde{L}_{0j}+\tilde{L}_{2j}+\tilde{L}_{3j})}} \tag{2.42}$$

and $\tilde{L}_{0k} = 2\rho y_k/\sigma^2$.

Similarly for $i = 0, 1, 3$ we have

$$L_k = \tilde{L}_{ok} + \tilde{L}_{1k} + \tilde{L}_{3k} + f(\mathbf{y}_2, \tilde{\mathbf{L}}_0, \tilde{\mathbf{L}}_1, \tilde{\mathbf{L}}_3) \tag{2.43}$$

and for $i = 0, 2, 3$ we have

$$L_k = \tilde{L}_{ok} + \tilde{L}_{1k} + \tilde{L}_{2k} + f(\mathbf{y}_3, \tilde{\mathbf{L}}_0, \tilde{\mathbf{L}}_1, \tilde{\mathbf{L}}_2) \tag{2.44}$$

A solution to satisfy Equations (2.41), (2.43) and (2.44) is

$$\begin{aligned}
\tilde{L}_{1k} &= f(\mathbf{y}_1, \tilde{\mathbf{L}}_0, \tilde{\mathbf{L}}_2, \tilde{\mathbf{L}}_3) \\
\tilde{L}_{2k} &= f(\mathbf{y}_2, \tilde{\mathbf{L}}_0, \tilde{\mathbf{L}}_1, \tilde{\mathbf{L}}_3) \\
\tilde{L}_{3k} &= f(\mathbf{y}_3, \tilde{\mathbf{L}}_0, \tilde{\mathbf{L}}_1, \tilde{\mathbf{L}}_2)
\end{aligned} \tag{2.45}$$

for $k = 1, 2, \cdots, N$. The final decision is then based on

$$L_k = \tilde{L}_{ok} + \tilde{L}_{1k} + \tilde{L}_{2k} + \tilde{L}_{3k} \tag{2.46}$$

which is passed through a hard limiter with zero threshold. The decoder structure is shown in Fig. 2.8, $f(.)$ is computed using the MAP algorithm with initial condition $\tilde{\mathbf{L}}_1^{(0)} = \tilde{\mathbf{L}}_2^{(0)} = \tilde{\mathbf{L}}_3^{(0)} = \tilde{\mathbf{L}}_0$. At the input of each decoder we add the appropriate extrinsic information from other decoders. After the final iteration the hard decision can be made, based on the value of L_k computed using Equation (2.46).

2.6 Summary

In this chapter, the encoder and decoder structure for turbo codes has been described. Iterative decoding method (BCJR Algorithm) for decoding of turbo codes was presented in detail. We also extend, these methods for multiple turbo codes and discuss the different decoding configurations possible for decoding in this case.

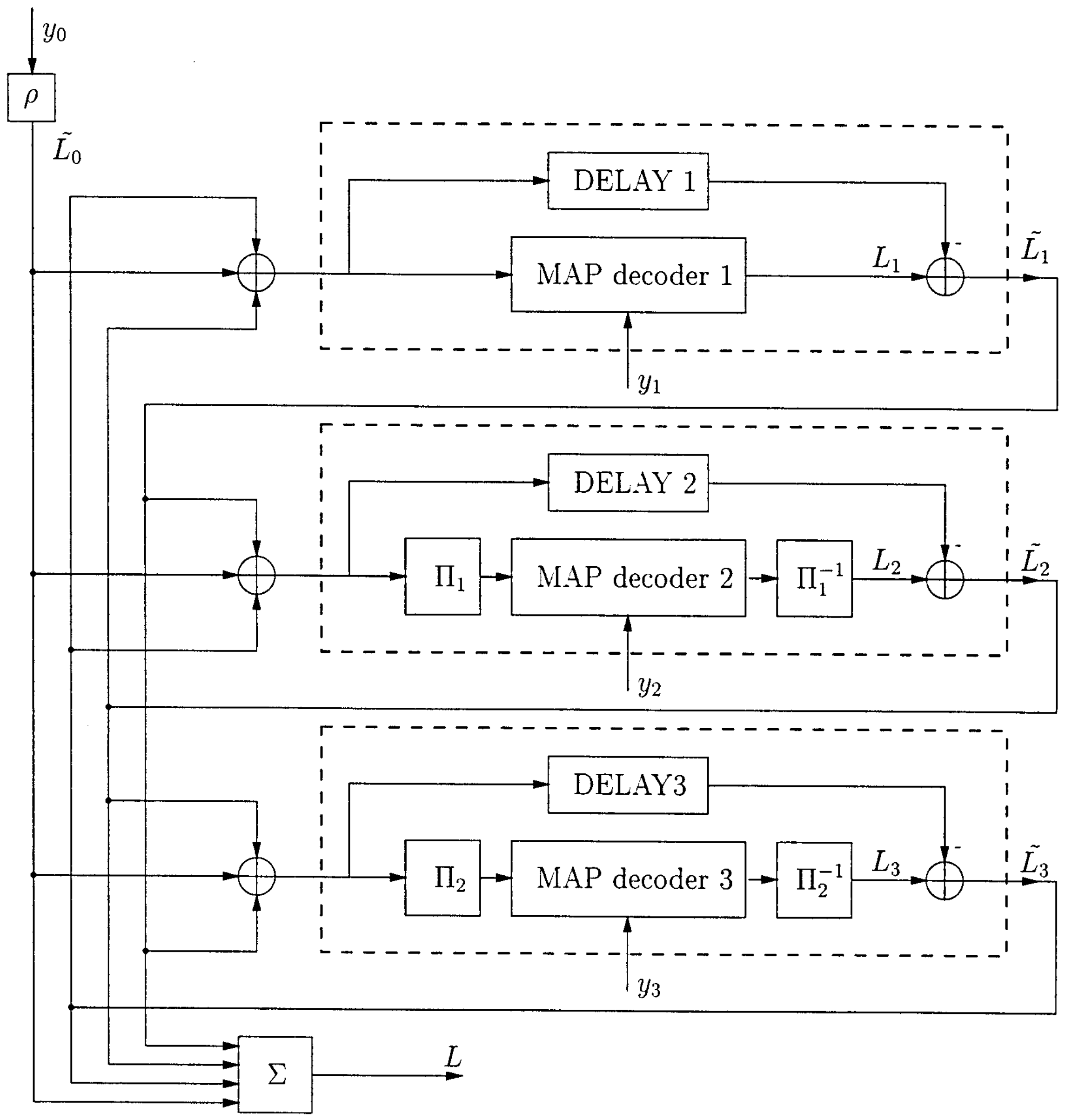


Fig. 2.8: Multiple Turbo Decoder

Chapter 3

Performance of Parallel Concatenated Codes

In this chapter, we evaluate the performance of turbo codes based on their weight distribution. Lower bounds for error probability can be derived for the case of block codes, which can also be applied to the terminated convolutional turbo codes. In order to find the lower bounds we need to know the weight distribution of the code. Weight distribution of a concatenated code can be calculated provided that the weight distribution of the constituent code is known. We find the weight distribution of convolutional codes, which enables us to calculate that of the concatenated codes. Later on, based on a lower bound, we analyze the performance of turbo codes and the parameters that affect their performance for different Signal-to-Noise Ratios(SNR).

3.1 Block Codes

An (n, k) block encoder, generates a codeword of length n from a sequence of k input bits. The output of the block encoder depends on the current k-bit input sequence and not on the previous input sequences. Since in an (n, k) block code there is one-to-one correspondence between the input sequences and the codewords, from the 2^n possible codewords only 2^k of them are used.

3.1.1 Weight Distribution of Block Codes

The weight distribution of a code enable us to compute the probability of undetected error and upper bounds of the word error probability, which are useful in computing probabilities of undetected or uncorrected error [16]. In this section we find the weight enumerating functions for block codes and later use this information to calculate word and bit error probability bounds for an Additive White Gaussian Noise (AWGN) channel.

Given an (n, k) linear systematic block code, its *weight enumerating function* (WEF) is

$$A^C(X) = \sum_{i=0}^{n} A_i X^i \tag{3.1}$$

where A_i is the number of codewords with Hamming weight (number of ones) i and X is a dummy variable. The WEF is used to calculate the word error probability of the code. The WEF of a code does not show the contributions of the input information bits to the total Hamming codeword weigh. For systematic block codes, a more detailed weight profile of a code is given by *input-redundancy weight enumerating function* (IRWEF)

$$A^C(W, Z) = \sum_{w,z} A_{w,z} W^w Z^z \tag{3.2}$$

where $A_{w,z}$ is the number of codewords generated by an input information sequence of Hamming weight w and the parity check sequence of Hamming weight z, so the total weight of codeword is $w + z$. For each term of WEF, the IRWEF shows the separate contribution of the information and the parity check bits to the total Hamming weight of the codewords, thus it can be used to calculate the bit error

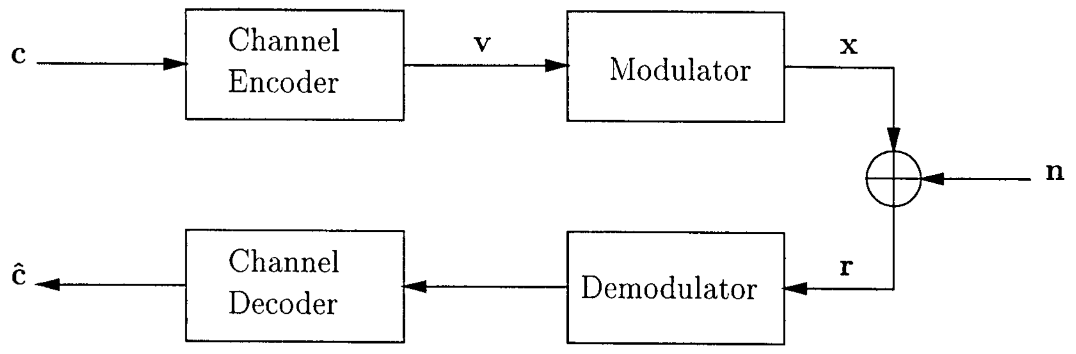


Fig. 3.1: Coded System Model

probability of the code [17].

The *conditional weight enumerating function* $A_w^C(Z)$ describes the parity check weights of the codewords generated by information sequence of weight w. It can be obtained from the IRWEF as

$$A_w^C(Z) = \sum_z A_{w,z} Z^z = \frac{1}{w!} \cdot \left. \frac{\partial^w A^C(W,Z)}{\partial W^w} \right|_{W=0} \tag{3.3}$$

and the inverse relation is

$$A^C(W,Z) = \sum_w W^w A_w^C(Z) \tag{3.4}$$

3.1.2 Performance Upper Bounds for Block Codes

Using the weight enumerating functions introduced in previous section, we can calculate the word error probability and bit error probability for a code. Assume a communication system with an (n,k) block code and binary phase shift keying (BPSK) modulation where the modulated signals are transmitted over an AWGN channel. The system model is shown in Fig. 3.1.

The signal in the channel is corrupted by additive white Gaussian noise. The received signal, at the time i, can be represented by

$$r_i = x_i + n_i \tag{3.5}$$

where n_i is a sample of a Gaussian noise with zero mean and variance σ^2. For this channel conditional probability $P(\mathbf{r}|\mathbf{v})$ is

$$\begin{aligned} P(\mathbf{r}|\mathbf{v}) &= \prod_{i=0}^{n-1} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(r_i - x_i)^2}{2\sigma^2}} \\ &= \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n e^{-\sum_{i=0}^{n-1} \frac{(r_i - x_i)^2}{2\sigma^2}} \\ &= \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n e^{-\frac{1}{2\sigma^2} d_E^2(\mathbf{r},\mathbf{x})} \end{aligned} \tag{3.6}$$

where

$$d_E^2(\mathbf{r}, \mathbf{x}) = \sum_{i=0}^{n-1} (r_i - x_i)^2 \tag{3.7}$$

represents the *Euclidean distance* between the received sequence and the modulated codeword. For Maximum Likelihood Decoding algorithm, the decoder computes the Euclidean distance of the received sequence with all modulated codewords and chooses the one that has the minimum Euclidean distance as the estimate of the transmitted codeword. An error event occur when the received sequence is closer in Euclidean distance to a code sequence $\hat{\mathbf{v}} \neq \mathbf{v}$. The probability that the decoder makes a wrong decision by selecting an erroneous sequence is called the *pairwise error probability* [18].

For code sequence $\hat{\mathbf{v}}$, $\hat{\mathbf{x}}$ represents the modulated signal. The pairwise error probability is given by

$$\begin{aligned} P_d &= P_r\left\{\sum_{i=o}^{n-1}|r_i - x_i|^2 \geq \sum_{i=0}^{n-1}|r_i - \hat{x}_i|^2\right\} \\ &= P_r\left\{\sum_{i=0}^{n-1} n_i(\hat{x}_i - x_i) \geq \frac{1}{2}|\hat{x}_i - x_i|^2\right\} \\ &= P_r\left\{\sum_{i=0}^{n-1} n_i(\hat{x}_i - x_i) \geq 2d\right\} \qquad (3.8) \end{aligned}$$

Where d is the Hamming distance of two codewords. Since n_i is a zero mean Gaussian noise sample with variance σ^2, the term $\sum_{i=0}^{n-1} n_i(\hat{x}_i - x_i)$ is also a Gaussian variable with zero mean with variance equal to $4d\sigma^2$. Thus the pairwise error probability will be [19]

$$P_d = Q\left(\sqrt{2dr\frac{E_b}{N_0}}\right) \qquad (3.9)$$

where $r = k/n$ is the code rate, E_b is the signal energy per bit, N_0 is the single sided power spectral density of the Gaussian noise .

3.1.2.1 Word Error Probability for Block Codes

For linear codes, we assume that the all-zero codeword is transmitted, so the probability of an error event of distance d is given by pairwise error probability. The number of such errors will be the number of codewords of weight d that can be obtained from the code WEF of Equation (3.1). The word error probability union bound is given by [18]

$$\begin{aligned} P_e &= \sum_{d=d_{min}} A_d P_d \\ &= \sum_{d=d_{min}} A_d Q\left(\sqrt{2dr\frac{E_b}{N_0}}\right) \qquad (3.10) \end{aligned}$$

where d_{min} is the *minimum Hamming distance* of the code and A_d is the number of error events with weight d.

3.1.2.2 Bit Error Probability for Block Codes

The bit error probability of an (n, k) block code can be upper bounded by a union bound as

$$\begin{aligned} P_b(e) &= \sum_{d=d_{min}} B_d P_d \\ &= \sum_{d=d_{min}} B_d Q\left(\sqrt{2dr\frac{E_b}{N_0}}\right) \end{aligned} \tag{3.11}$$

where B_d is the average number of bits in error caused by transition between all-zero codeword and codewords with weight d, which can be obtained from IRWEF of the code, Equation (3.2) [9]

$$B_d = \sum_{w+z=d} \frac{w}{k} A_{w,z} \tag{3.12}$$

considering the Q-function bound

$$Q(x) \leq 1/2e^{-\frac{x^2}{2}} \quad x \geq 0 \tag{3.13}$$

the bit error probability upper bound can be given by

$$\begin{aligned} P_b(e) &\leq \sum_{d=d_{min}} 1/2B_d e^{-dr\frac{E_b}{N_0}} \\ &= \sum_{w=1}^{k} \frac{w}{2k} W^w A_w(Z)\Big|_{W+Z+e^{-r\frac{E_b}{N_0}}} \end{aligned} \tag{3.14}$$

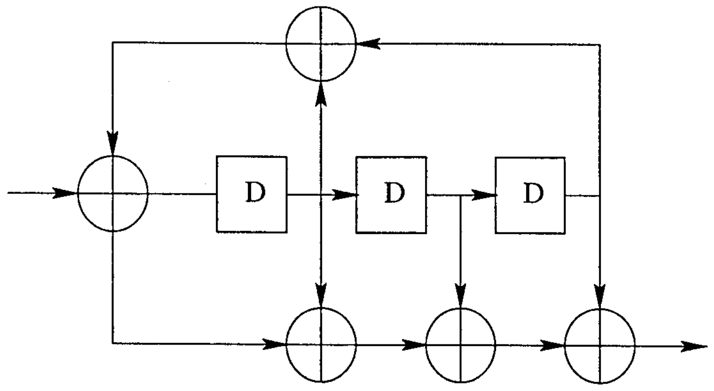


Fig. 3.2: Recursive systematic Encoder for Code Generator (13,17)

3.2 Convolutional Codes

Convolutional codes are represented by a triplet (n, k, m), where n is the length of the codeword, k is the length of the input sequence and m is the memory of the encoder. In Convolutional codes the codeword is not only dependent on the current input but also on previous m inputs which determines the state of the encoder. Since the output is determined by the input and the state of the encoder, each encoder can be described by a state machine. Another popular way of representing Convolutional codes is using *trellis diagram*, which again contains the information about the state transition and output of a code given the input and current state [19].

Convolutional codes can be either *systematic* or *non systematic*. In a systematic code, the output consists of a replica of the input sequence and a parity sequence. *Recursive Systematic Convolutional* (RSC) codes can be represented by their generator matrix with general form

$$G(D) = \begin{pmatrix} 1 & \frac{g_1(D)}{g_2(D)} \end{pmatrix} \tag{3.15}$$

where $g_1(D)$ and $g_2(D)$ are feed forward and feedback polynomials, respectively and

D is the delay element. For example the code in Fig. 3.2 has the generator matrix

$$G(D) = \begin{pmatrix} 1 & \frac{1+D+D^2+D^3}{1+D+D^3} \end{pmatrix}$$

The generator polynomials can also be represented in octal form, e.g., $(g_2, g_1) = (13, 17)_{oct}$. For an input sequence $U(D)$, the output codeword will be equal to $U(D) \cdot G(D)$ [20].

3.2.1 Weight Enumerators for Convolutional Codes

We use the equivalent block code of the convolutional code in order to find the weight enumerating functions of the convolutional codes. The equivalent block code, is the same as the convolutional code with additional constraint of being truncated at step N and also all the codewords being terminated to the zero state. The trellis representation of the equivalent block code is the truncation at step N of the trellis of the convolutional code and the trellis is terminated to zero state at step N [9].

The IRWEF of the equivalent block code can be derived from the *error events enumerating function* of the convolutional code. A single error path is a path that diverges from the all-zero path in the trellis and then emerges to the all-zero path within a finite number of branches. Using the above definition, any code word belonging to the equivalent block code is obtainable by combining the set of error events of the convolutional code with suitable sequences of "0" so that the total length equals N as shown in Fig. 3.3. For example, a single error event of length l, can contribute to code words that have $N - l$ zeros positioned before and after the error event. The total number of these code words is equal to the number of partitions of $N - l$ into two numbers:

$$K[l, 1] = \begin{pmatrix} N - l + 1 \\ 1 \end{pmatrix} = N - l + 1 \tag{3.16}$$

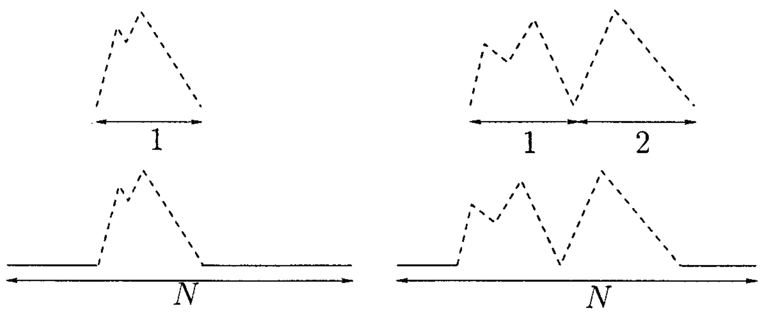


Fig. 3.3: How to obtain Codewords of the equivalent Block Code from Error events of the Convolutional Code

Codewords can be generated by concatenation of single error events, shown in Fig. 3.3. The general expression for the multiplicity of codewords produced by a single combination of n error events with total length l is given by

$$K[l,n] = \begin{pmatrix} N-l+n \\ n \end{pmatrix} \tag{3.17}$$

Let $T^C(W,Z,L,\Omega)$ enumerate all paths in the trellis diverting from the zero state in the first step and converge to the zero state at or before step N, with possible emerging into the zero state in between, with the constraint that it diverges from the zero state immediately after each reemerging

$$T^C(W,Z,L,\Omega) = \sum_{w,j,l,n} T_{w,j,l,n} W^i Z^j L^l \Omega^n \tag{3.18}$$

Where $T_{w,j,l,n}$ is the number of paths in the trellis produced by an input sequence of weight w, with weight of parity check equal to j, length l and n reemerging with the zero state (concatenation of n error events).

In the equivalent block code each path of length l and number of reemerging n belonging to $T_w^C(Z,L,\Omega)$ will contribute to $K[l,n]$ codewords with the same input and parity weights, so the conditional IRWEF $A_w^C(Z)$ of the equivalent block code can be obtained as

$$A_w^C(Z) = \sum_j A_{w,j} Z^j \tag{3.19}$$

where

$$\begin{aligned} A_{w,j} &= \sum_{l,n} K[l,n] T_{w,j,l,n} \\ &= \sum_{l,n} \begin{pmatrix} N-l+n \\ n \end{pmatrix} T_{w,j,l,n} \end{aligned} \tag{3.20}$$

The algorithm to compute $T_w^C(Z, L, \Omega)$ is described in [21], but in Section 3.3.2, we will use an approximation valid for large N which will enable us to analyze the performance of the concatenated codes.

3.3 Parallel Concatenated Codes

Consider now a parallel concatenation of two block codes or two convolutional codes whose equivalent block code WEF is known. The two encoders will encode the information sequence with Hamming weight w. The weight of parity sequence generated by first and second encoder are j_1 and j_2, respectively. The total Hamming weight of the codeword for the parallel concatenated code (C_P), will be $w+j_1+j_2$, as shown in Fig. 3.4. In the next section, we will find the WEF for concatenated codes.

3.3.1 Weight Enumerators for Concatenated Codes

We want to obtain the IRWEF $A^{C_P}(W, Z)$ of the parallel concatenated code (C_P) from the IRWEF of the constituent codes. The IRWEF of the second parity sequence, not only depends on the encoder, but also depends on the choice of interleaver and the particular pattern to which the inputs are mapped.

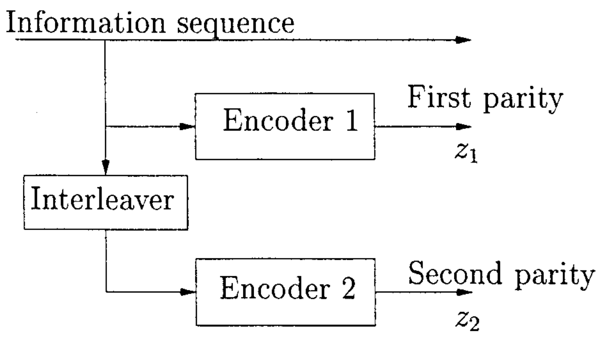


Fig. 3.4: Parallel Concatenated Code

To overcome this difficulty, concept of *uniform interleaver* is introduced. A *uniform interleaver* of length k is a probabilistic device which maps a given input word of weight w into all distinct $\begin{pmatrix} k \\ w \end{pmatrix}$ permutations of it with equal probability $1/\begin{pmatrix} k \\ w \end{pmatrix}$ [9].

By using the uniform interleaver, the conditional weight enumerating function $A_w^{C_2}(Z)$ of the second code becomes independent from that of the first code, because of the randomization produced by the interleaver.

The conditional weight enumerating function of the parallel concatenated code which uses the uniform interleaver is the product of the two conditional weight enumerating functions of the constituent codes, as follows

$$A_w^{C_p}(Z) = \frac{A_w^{C_1}(Z) \cdot A_w^{C_2}(Z)}{\begin{pmatrix} k \\ w \end{pmatrix}} \tag{3.21}$$

3.3.2 Performance Upper bound for Concatenated Codes

In Section 3.2.1, the exact formula for computing the conditional weight enumerating function $A_w^C(Z)$ for equivalent block code for convolutional code is given, but in this section we use an approximation which enables us to find the upper bound for concatenated codes [22].

For a convolutional code, the codewords of the equivalent block code are concatenations of error events of the convolutional code. Let

$$A(w, Z, n) = \sum_j A_{w,j,n} Z^j \tag{3.22}$$

be the parity-check enumerating function of the convolutional code, constructed by concatenation of n error events with input weight of w. $A_{w,j,n}$ is the number of the codewords with input weight w and parity weight of j and n error events being concatenated. The conditional weight enumerating function $A^C(w, Z)$ of the equivalent block code can be approximated by

$$A^C(w, Z) \sim \sum_{n=1}^{n_{max}} \begin{pmatrix} N \\ n \end{pmatrix} A(w, Z, n) \tag{3.23}$$

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

$A_{w,j,n}$ can be calculated using $T_{w,j,l,n}$ which is the number of codewords produced by an input sequence of weight w, with weight of parity check equal to j, length l and n error patterns given by Equation (3.18)

$$A(w, Z, n) = \sum_j \sum_l T_{w,j,l,n} Z^j \tag{3.24}$$

Substituting the WEF of the constituent codes Equation (3.23), into Equation (3.21) we get for the conditional WEF of the turbo code

$$A_w^{C_p}(Z) \sim \sum_{n_1=1}^{n_{max}} \sum_{n_2=1}^{n_{max}} \frac{\begin{pmatrix} N \\ n_1 \end{pmatrix} \begin{pmatrix} N \\ n_2 \end{pmatrix}}{\begin{pmatrix} N \\ w \end{pmatrix}} \cdot A(w, Z, n_1) A(w, Z, n_2) \tag{3.25}$$

if we use the following approximation

$$\begin{pmatrix} N \\ n \end{pmatrix} \sim \frac{N^n}{n!} \tag{3.26}$$

we get

$$A_w^{C_p}(Z) \sim \sum_{n_1=1}^{n_max} \sum_{n_2=1}^{n_max} \frac{w!}{n_1! \cdot n_2!} N^{n_1+n_2-w} \cdot A(w, Z, n_1) A(w, Z, n_2) \tag{3.27}$$

For large N, $A_w^{C_p}(Z)$ can be approximated by the terms in the summations having the highest power of N, which are $n_1 = n_2 = n_{max}$

$$A_w^{C_p}(Z) \sim \frac{w!}{n_{max}!^2} N^{2n_{max}-w} \cdot [A(w, Z, n_{max})]^2 \tag{3.28}$$

Considering the bit error probability upper bounds in Equation (3.14), we get the union upper bound for turbo code as

$$\begin{aligned} P_b(e) \;\; \leq \;\; & \sum_{w=1}^{N} w \cdot \frac{w!}{(n_{max}!)^2} N^{2n_{max}-w-1} \\ & W^w [A(w, Z, n_{max})]^2 \Big|_{W=Z=e^{-r\frac{E_b}{N_0}}} \end{aligned} \tag{3.29}$$

3.4 Turbo Codes Design

The performance analysis developed in the previous section, enables us to investigate many performance and design issues about turbo codes. We will discuss the benefit of using *recursive codes* instead of *non recursive convolutional codes* as constituent encoders. In addition we will determine the factors that determine the performance for each SNR region.

3.4.1 Recursive Constituent Codes

It is preferable to use *recursive* convolutional constituent encoders, instead of *non recursive* encoders. In this section we find the bit error rate for these two encoders and show the benefits of using *recursive* convolutional constituent encoder [22].

The bit error probability for *non recursive* convolutional constituent encoder can be found using Equation (3.29). Since the minimum weight of the input is equal to one ($w_{min} = 1$) we can have a finite-weight error sequence of length $2(v+1)$. An information sequence of weight w, can cause w error events corresponding to the concatenation of w error events of weight one, as a consequence $n_{max} = w$ this will conclude that the term

$$A(w, Z, n_{max}) = A(w, Z, w) = A(1, Z, 1)^w \tag{3.30}$$

the bound can be simplified as

$$\begin{aligned} P_b(e) \;\leq\; & \sum_{w=1}^{N} \frac{N^{w-1}}{(w-1)!} \\ & \times W^w [A(1, Z, 1)]^{2w} \Big|_{W=Z=e^{-r\frac{E_b}{N_0}}} \end{aligned} \tag{3.31}$$

Equation (3.31), shows that for the case $w = 1$, the bit error probability is independent of N, the size of interleaver. Indeed, no interleaving gain is possible.

On the other hand, in the case of *recursive* convolutional codes, w_{min} is always greater than one. Consider a code with generator matrix $G(D) = [1, \frac{n(D)}{d(D)}]$ and with memory v. The finite-weight error event is caused by input sequence that is divisible by the feedback polynomial $d(D)$. Feedback polynomial has the form $1 + \cdots + D^v$, which can not divide a polynomial of the form D^i for any value of i. As a consequence, the minimum weight of the input is equal to two ($w_{min} = 2$), that can cause $\lfloor w/2 \rfloor$ error events corresponding to the concatenation of error events of weight two ($n_{max} = \lfloor w/2 \rfloor = 2k$). It has been shown in [22], that the bit error probability upper bound will be

$$\begin{aligned} P_b(e) \;\leq\; & \sum_{k=1}^{\lfloor N/2 \rfloor} 2k \begin{pmatrix} 2k \\ k \end{pmatrix} N^{-1} W^{2k} \\ & \times [A(2, Z, 1)]^{2k} \Big|_{W=Z=e^{-R\frac{E_b}{N_0}}} \end{aligned} \tag{3.32}$$

Looking at Equation 3.32, it can be seen that the performance is affected by the interleaver size N. Increasing the interleaver size N, the bit error probability is reduced by a factor $1/N$, which is called the *interleaver gain.*

3.4.2 Free Distance and Performance at High SNR

Up to now, turbo code performance is discussed based on the asymptotic bounds of the bit error probability. In this section, we evaluate turbo code performance based on its distance spectrum. Using the IRWEF of the code Equation (3.11), the bit error probability upper bound can be expressed based on the code distance spectrum by

$$P_b(e) \leq \sum_{d=d_{free}} B_d Q\left(\sqrt{2dr\frac{E_b}{N_0}}\right) \tag{3.33}$$

where the set of all pairs of (d, B_d) is the code distance spectrum. For turbo codes, the error coefficients B_d can be represented by

$$\begin{aligned} B_d &= \sum_{d=w+z} \frac{w}{N} \cdot A_{w,z} \\ &= \sum_{d=w+z_1+z_2} \frac{w}{N} \cdot \frac{A^c_{w,z_1} \cdot A^c_{w,z_2}}{\begin{pmatrix} N \\ w \end{pmatrix}} \end{aligned} \tag{3.34}$$

where d_{free} is the *free distance* which is the minimum Hamming weight of all possible nonzero codewords.

For moderate and high SNR the bit error rate performance will be dominated by the *free-distance term* in Equation (3.33). Thus the asymptotic performance approaches

$$P_b(e) \sim B_{d_{free}} Q \left(\sqrt{2 d_{free} R \frac{E_b}{N_0}} \right) \tag{3.35}$$

Turbo codes have a relatively small free distance and consequently a relatively flat free-distance asymptote behaviour. This flattening of performance curve at high SNR is a phenomenon known as *error-floor.*

Error-floor can be manipulated in two ways: first, increasing the length of the interleaver while preserving the free distance, this will decrease the multiplicity and will improve the performance. If the size of interleaver is fixed, then the *error-floor* can be modified by increasing the free distance of the code while preserving the multiplicity. It is shown in [23] that for a fixed interleaver size, choosing the feedback polynomial to be a primitive polynomial results in an increased free distance and thus a steeper asymptote. We will show in Chapter 4 that free distance can also be increased by proper choice of interleaver.

3.4.3 Spectral Thinning and Performance at Low SNR

Unlike *error-floor* region, where the performance curve flattens, at low to moderate SNR, the performance curve has a steep decent known as *water fall* region. In order to explain this phenomenon, assume a turbo code with a uniform interleaver. Several low-weight input sequences associated with low-weight parity will be mapped to interleaved sequences which are associated with high-weight parity (since the average parity weight is high even for low-weight input). Compared with the weight distribution of its component RSC code, the number of low-weight input sequences associated with high-weight parity is increased while those associated with a low-weight parity remain few. This phenomenon is called spectral thinning because the low-weight portion of the distance spectrum retains a low multiplicity. Thus, although the free distance of a Turbo code may be small, but the multiplicity of low-weight codewords is lower (not just for the free distance codewords). This means that at low-medium SNR, when the effect of higher-weight codewords cannot be neglected, the performance of a Turbo code is better than a RSC code [24].

3.5 Summary

In this chapter we have presented the weight distribution for the convolutional codes, which are used as constituent codes in turbo codes. Based on this, we have found the weight distribution and the performance lower bounds for turbo codes.

We also observed that at low to moderate SNR, the performance of turbo codes is improved extremely, by decreasing the multiplicity of low weight sequences. On the other hand, at the moderate to high SNR where free distance asymptotic dominates the performance, the performance curve flattens. This is called the *error-floor* region.

Chapter 4

Interleaver Design

It is widely accepted by the research community that the interleaver plays a very important part in the construction of good Turbo codes. Interleaver design criteria aim at improving the performance of turbo codes. In this chapter, we review some known interleaver design criteria. Later in the chapter, we review some known interleaver design methods such as block, even-odd interleaver and etc we will emphasis more on some optimized interleaver design methods such as S-random and Code-matched interleavers and we will observe some simulation results for code-matched interleaver design for 3GPP standard.

4.1 Interleaving

Interleavers have been used in communication systems for a long time. Traditionally they were used to randomize the location of errors, enabling the use of random-error-correcting codes on channels with burst error patterns. Typically, bursty channels would include fading channels often found in wireless transmission.

Another use of interleaving is in concatenated coding, where information is passed through an interleaver before component codes. This action will construct a long block code from a small memory convolutional code. The input sequence of each component encoder is scrambled by this interleaver in order to decrease the correlation between bits of the input sequences. Interleaver also can be used to improve the minimum distance of the code, by breaking the low weight input sequences.

4.1.1 Effect of Interleaver Structure on Distance Spectrum

As we discussed in earlier chapters, the performance of turbo code at high SNR is dominated by the first several distance spectral lines, which are generated by low weight input sequences. These low weight codewords happen when a low weight input sequence generates low weight parity sequences at the output of the constituent encoders. However, interleaver can affect the mapping of the input patterns of the constituent encoders and it is capable of associating the patterns that cause low weights at the output to the patterns that generate higher weights at the output. By acting on low weight inputs we are able to improve the free distance of the code and consequently improve the performance of the code.

In order to design interleavers that improve the distance spectrum of turbo codes, we need to determine the input patterns that result in low weight codewords. After wards, the interleaver is designed to prevent such patterns from being mapped to each other. In this section, we introduce *divisibility principle* [25], which enables us to determine the patterns that generate low weight parity sequences, and in Section 4.5, we will introduce interleaver design method that benefits from this principle and eliminates first dominant spectral line of turbo codes.

4.1.1.1 Divisibility Principle

Consider a turbo code, constituting of a recursive systematic convolutional codes with generator polynomial $G(D) = \frac{F(D)}{B(D)}$, Fig. 4.1. We refer to $U(D)$ as information sequence polynomial and its interleaved version is referred as $V(D)$, the parity

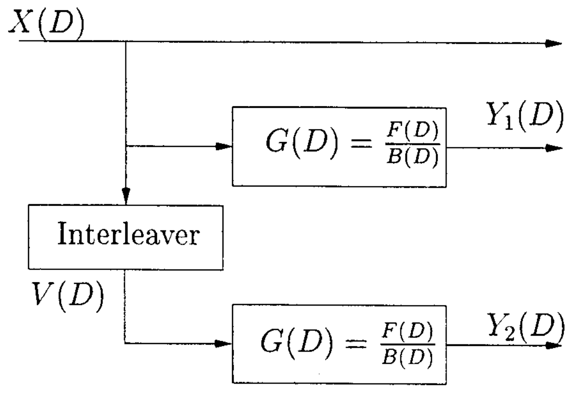


Fig. 4.1: Rate 1/3 Turbo code

sequence of each encoder will be given by

$$\begin{aligned} Y_1(D) &= U(D)\frac{F(D)}{B(D)} \bmod D^{n+m} \\ Y_2(D) &= V(D)\frac{F(D)}{B(D)} \bmod D^{n+m} \end{aligned} \tag{4.1}$$

If $U(D)$ or $V(D)$ are divisible by the feedback polynomial $B(D)$, then the parity sequence will have a finite Hamming weight. For example, for constituent encoder with feedback polynomial $B(D) = 1+D+D^2$, the input sequence $U(D) = 1+D^3$ is divisible by feedback polynomial and will generate parity sequence $Y_1(D) = (1 + D)(F(D))$.

Based on Divisibility principle, patterns divisible by the feedback polynomial generate low weight parity sequences. As a consequence, a *good* interleaver ought to permute divisible patterns to non divisible patterns for low weight input patterns. This procedure will ensure that the output of at least one of the encoders has high weight.

4.1.2 Effect of Interleaver Structure on Iterative Decoding

The interleaver, as mentioned in previous section, has two major tasks. One is to ensure a good distance spectrum for the code by breaking up the low weight divisible

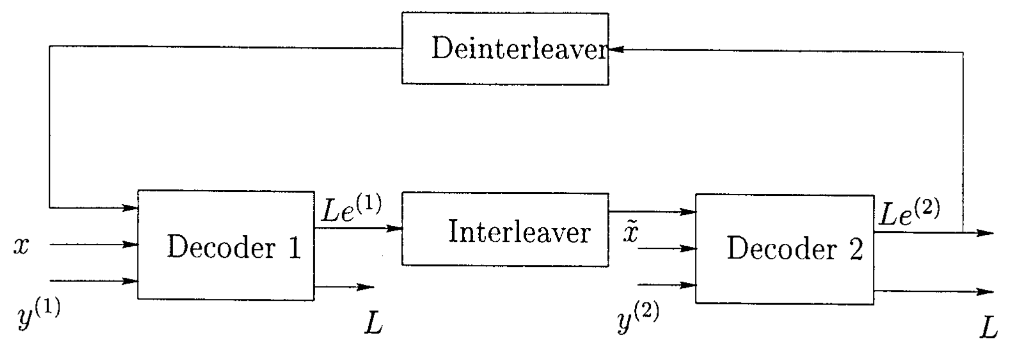


Fig. 4.2: Structure of Iterative Decoder

input sequences and the other is to decorrelate the input of component encoders to improve the performance of iterative decoding.

Turbo codes are decoded iteratively which is not optimal in the sense of making maximum likelihood decisions. The performance of the iterative decoding is dependent on the quality of extrinsic information being exchanged between the constituent encoders in the iterative decoding scheme. The choice of interleaver, as we will see in this section, affects the degree of correlation between extrinsic inputs [26].

The iterative decoding is depicted in Fig. 4.2. Each soft-input/soft-output constituent decoder has three inputs and two outputs. The three inputs are the systematic input x at the first decoder and the interleaved version of it $\tilde{x}$ at the second decoder, the parity inputs $y^{(1)}$ or $y^{(2)}$ and the extrinsic information from the previous decoder. The outputs are the soft decision and the new extrinsic output $Le^{(1)}$ or $Le^{(2)}$. The extrinsic information is used as *a priori* probability in the next decoding step. The systematic and parity inputs for each decoding step depend only on the received values for each specific bit. However, extrinsic inputs are dependent on a range of symbols in the received systematic and parity sequences, which we will investigate in the following.

After the first decoding step, since the *a priori* probability inputs to first decoder are all zero, the extrinsic outputs are only dependent on the systematic and parity input sequences. The decision variable at position i, is dependent on

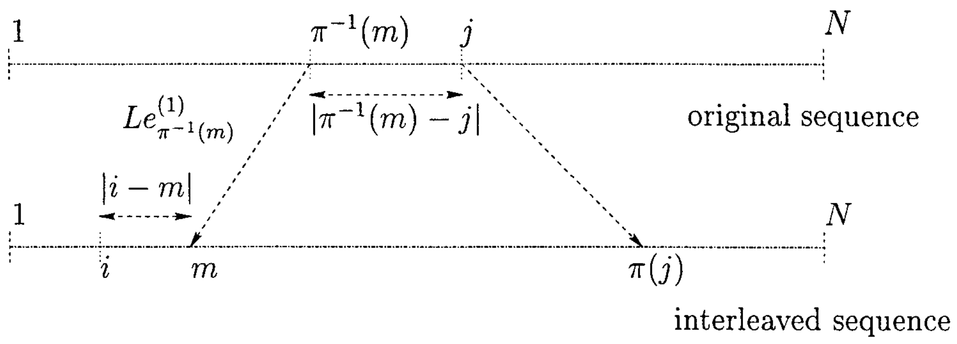


Fig. 4.3: Correlation for Extrinsic Information of Second Encoder

the decoder input at position i and as a result of the trellis structure of the code, it is also *decreasingly* dependant on the input at times $i \pm 1$, $i \pm 2$, $i \pm 3$ and so on. Letting $\rho^{(n)}_{Le_i,x_j}$ denote the approximated correlation coefficient between extrinsic output i and systematic input j after decoding step n. The simulation results in [27] show that these correlation coefficients decay exponentially and are dependent on the distance between i and j and not on their absolute value. As a consequence, after the first decoding step we have [27] :

$$\rho^{(1)}_{Le_i,x_j} = \begin{cases} 0 & \text{if } i = j \\ ae^{-c|i-j|} & \text{otherwise} \end{cases} \qquad i, j = 1, 2, \ldots, N \tag{4.2}$$

where N is the size of interleaver and constants a and c, which depend on the choice of the encoder, adjust the amplitude and the exponential decay rate respectively. The $\rho^{(1)}_{Le_i,x_i}$ being equal to zero, is explained by the way the decoder is implemented, where we subtract the weighted systematic value in order to get the extrinsic information.

Equation (4.3) also shows that the interleaver does not affect the correlation coefficients after the first decoding step, $\rho^{(1)}_{Le_i,x_j}$. However, investigating the same correlation coefficients after the second decoding step reveals the role of interleaver.

If interleaving is represented as $\pi(m)$, the correlation coefficients can be approximated as:

$$\rho^{(2)}_{Le_i,x_j} = 1/2 \quad \overbrace{ae^{-c|\pi(j)-i|}}^{\text{systematic input}} \quad +1/2 \overbrace{\sum_{m=1,m\neq i}^{N} a^2 e^{-c(|\pi(m)^{-1}-j|+|i-m|)}}^{\text{extrinsic input}} \tag{4.3}$$

The approximation can be intuitively justified using Fig. 4.3. The extrinsic output of the second decoder at position i is correlated to the *a priori* input at position m and this correlation is exponentially decreasing with the distance between i and m, i.e., $|i - m|$. Further more, the *a priori* input at position m stems from output $\pi^{-1}(m)$ from the first decoder and it is thus correlated to the systematic bit j according to Equation (4.2), i.e., exponentially decreasing with $|\pi^{-1}(m) - j|$. the total correlation between extrinsic output i from the second decoder and systematic input j, contributed from extrinsic output $\pi^{-1}(m)$ from the first decoder, is therefore approximated as $a^2e^{-c(|\pi(m)^{-1}-j|+|i-m|)}$. The systematic part in Equation (4.3) is the same as in the case of the first decoder, except that the systematic inputs are now interleaved.

The above discussion shows that the output i from the second decoder is not only dependant on the sequence of systematic and parity inputs in the vicinity of position i, but also on the *a priori* inputs in the same neighborhood. Each of these *a priori* inputs are in turn correlated to the channel inputs in the vicinity of its origin, before interleaving. Provided that the interleaver is suitably chosen, output i from the second decoder can be correlated to a wide range of channel inputs, not only to those in the vicinity of i. This will improve the quality of extrinsic information and therefore improve the performance of the system.

The best performance of the iterative decoder is achieved if all the *a priori* inputs are totally uncorrelated. Since this is not possible, we try to make *nearby a priori* inputs as uncorrelated to each other as possible. This is achieved by forcing the extrinsic outputs from the second decoder to be as uniformly correlated to the systematic sequence as possible, since correlated *a priori* inputs result in extrinsic outputs that are excessively correlated to some part of the systematic sequence. Thus, for each new interleaver element to be defined, say $\pi(j)$, the correlation coefficient $\rho^{(2)}_{Le_i,x_j}$ resulting from the already defined elements are approximated using Equation (4.3). The element $\pi(j)$ is then chosen at the position to which the correlation up to this point is the lowest.

4.2 Block interleaver

A block interleaver formats the input sequence in a matrix of m rows and n columns, such that $N = m \times n$. The input sequence is written into the matrix row-wise and read out column-wise. The deinterleaver stores the data into a matrix with the same format as the interleaver and data will be read out row-wise.

4.3 Odd-Even Block Interleavers

In the case of iterative decoding, improvement is obtained when the errors remaining from one decoder is correctable by the other decoder. Interleavers can improve the performance, provided that the sequence is spread appropriately. Odd-Even block interleavers, introduced in [28], improve the decoding ability of rate 1/2 punctured turbo codes, by means of providing an uniform error protection across the information sequence.

An odd-even interleaver, maps even position elements to even positions and odd position elements to odd positions. For example, Table 4.1 shows the input and parity sequence for the first encoder for a rate 1/2 turbo encoder. The even position parity bits are punctured, but each odd position information bit has its own coded bit present.

Table 4.1: Odd positioned coded bits of the first encoder

c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}
$v_{1,1}$	–	$v_{3,1}$	–	$v_{5,1}$	–	$v_{7,1}$	–	$v_{9,1}$	–	$v_{11,1}$	–	$v_{13,1}$	–	$v_{15,1}$

If the interleaver maps even position elements to even places and odd position elements to odd positions, after the second encoder we will have a sequence shown in Table 4.2 and as it can seen all the even positions information bits have their own coded bit present.

Table 4.2: Even positioned coded bits of the second encoder

c_1	c_6	c_{11}	c_2	c_7	c_{12}	c_3	c_8	c_{13}	c_4	c_9	c_{14}	c_5	c_{10}	c_{15}
–	$v_{6,2}$	–	$v_{2,2}$	–	$v_{12,2}$	–	$v_{8,2}$	–	$v_{4,2}$	–	$v_{14,2}$	–	$v_{10,2}$	–

The whole code word, after multiplexing is shown in Table 4.3. If the interleaving was done in a random way, some information bits would have had two coded bits associated with them and others would have no coded bits associated with them, so the coding power is not uniformly distributed among all the bits. However, in even-odd interleaver, all the information bits will have associated one and only one coded bit. This gives a uniform distribution of the correcting capability of the codes and better performance for turbo codes.

Table 4.3: Information bits and multiplexed bits for an 'odd-even' interleaver

c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}
$v_{1,1}$	$v_{6,2}$	$v_{3,1}$	$v_{2,2}$	$v_{5,1}$	$v_{12,2}$	$v_{7,1}$	$v_{8,2}$	$v_{9.1}$	$v_{4,2}$	$v_{11,1}$	$v_{14,2}$	$v_{13,1}$	$v_{10,2}$	$v_{15,1}$

4.4 S-random Interleaver

One of the interleavers that improves both the distance spectrum of the code and the iterative decoding suitability of the code, is S-random interleaver. S-random interleaver is based on random mapping of N elements conditioned by S-random constraint. The S-random constraint ensures that if two elements are placed closer than distance S, after interleaving they are placed in distance bigger than S [29]. In general the S-random interleaver can be described as

$$|\pi(i_1) - \pi(i_2)| \geq S \quad whenever \quad |i_1 - i_2| \leq S \tag{4.4}$$

where i_1, i_2 are the positions before interleaving and $\pi(i_1), \pi(i_2)$ are the positions after interleaving, as depicted in Fig. 4.4

S-random interleaver is able to break input patterns shorter than S, these patterns may include some of low weight divisible input patterns . As we showed in Section 4.1.1, these low weight divisible input patterns can deteriorate the performance of the turbo code. S-random interleaver will map divisible input patterns shorter than S into patterns larger than S. If the interleaved sequence, is a non divisible pattern by the feedback polynomial it will increase the parity weight considerably. However, if the interleaved sequence is again a divisible pattern, the weigh will increase slightly (still the Hamming weight of the parity sequence is finite). This shows that although S-random interleaver can improve the performance considerably, it is not capable of targeting in specific the low weight divisible input sequences. Another downside of S-random interleaver is that for large value of S the search time for interleaver will increase up to a point where converging becomes impossible. The attainable value for S are less than $\sqrt{N/2}$. In the next section we will see an interleaver design method, which targets divisible patterns specifically

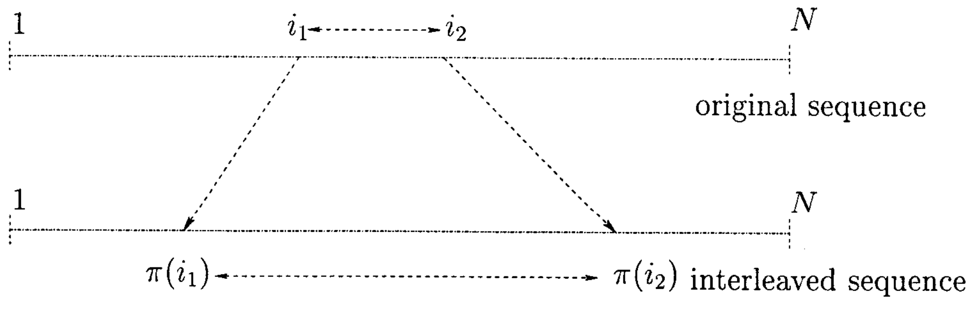


Fig. 4.4: Design Criterion for S-random Interleaver

and improves the *minimum free distance* of the code.

4.5 Code-Matched Interleavers

As discussed earlier, turbo codes, performance at moderate to high SNR is dominated by the first several distance spectral lines, which are generated by low weight input sequences. These low weight input patterns can be eliminated by proper interleaving. In code-matched interleaver design [30] for turbo codes, a particular interleaver is constructed to match the code weight distribution and the low-weight codewords with significant contribution to the error performance, are completely eliminated. These patterns can be eliminated by breaking up the input pattern or in other words, to ensure that they are not mapped to divisible patterns shorter than a certain length.

Designing a code-matched interleaver consists of two major steps. In the first step in order to determine the weights that need to be eliminated, the contribution of each spectral line to the error performance is found. In the second step, interleaver will be designed with the constraint of eliminating the patterns that generate the dominating weights [31].

As previously seen, a lower bound on the bit error probability of turbo code can be computed based on its distance spectrum using Equation (4.5)

$$P_b \leq \sum_{d=d_{free}} B_d Q\left(\sqrt{2dr\frac{E_b}{N_0}}\right) \tag{4.5}$$

Where B_d is

$$B_d = \sum_{d=w+z} \frac{w}{N} \cdot A_{w,z} \tag{4.6}$$

From the code distance spectrum, we can obtain the contribution of each spectral line to the BER performance. The contribution of a spectral line with distance d to the overall BER, denoted by $P_d(\gamma b)$, can be written as

$$P_d(\gamma b) = B_d Q\left(\sqrt{2dr\gamma b}\right) \tag{4.7}$$

where $\gamma b = E_b/N_0$. Relative contribution of each spectral line to the total BER can be represented as

$$\overline{P_d(\gamma b)} = \frac{P_d(\gamma b)}{\sum_d P_d(\gamma b)} \tag{4.8}$$

Then the contribution of the spectral line in a SNR range $[a, b]$ is obtained as

$$F_{ab}^d = \int_a^b \overline{P_d(\gamma b)} d\gamma b \tag{4.9}$$

where F_{ab}^d is called the *contribution integral* and a and b are the specified SNR values. F_{ab}^d can further be normalized as

$$\overline{F_{ab}^d} = \frac{F_{ab}^d}{\sum_d F_{ab}} \tag{4.10}$$

where $\overline{F_{ab}^d}$ is called the *relative contribution integral.* The *relative contribution integral* $\overline{F_{ab}^d}$ accurately represents the relative contribution of each spectral line in a given SNR range. It can be used as the criterion to determine the contribution of each spectral line in that SNR region. In Section 4.5.2, we find the *relative contribution integral* for a given system 3rd Generation Partnership Project (3GPP) and find the weights that contribute to the error performance at error-floor region and need to be eliminated.

4.5.1 Code-Matched Interleaver Design

The code performance analysis, will provide us with the weights that contribute to the error performance. The patterns in the input that generate these weights, can be found from the generator polynomial of the encoder. Usually these patterns are low weight input patterns (weight 2, 3 and 4) that are divisible by the feedback polynomial. For such input patterns we try to either eliminate the significant contributors to the error performance or to reduce the number of such codewords.

Now consider a weight-2 input sequence

$$P(D) = (1 + D^{\mu k_1})D^{\tau_1} \tag{4.11}$$

where $k_1 = 1, 2, 3, \ldots$ and μ is the minimum distance between two "1"s in the weight-2 input pattern that generates the finite weight codeword (it is divisible by feedback polynomial) and τ_1 is the time delay, $\tau_1 = 1, 2, 3, \ldots$. The input to the second encoder $Q(D)$, the interleaved version of the input sequence, will be presented as

$$Q(D) = (1 + D^{\mu k_2})D^{\tau_2} \tag{4.12}$$

where $k2 = 1, 2, 3, \ldots$ and τ_2 is the time delay, $\tau_2 = 1, 2, 3, \ldots$. Let us denote by z_{min} the minimum weight of the parity-check sequence generated by a weight-2 input pattern. Then the parity-check weight of any weight-2 input pattern with the form of Equation (4.11) is given by

$$k_j \times (z_{min} - 2) + 2 \tag{4.13}$$

where $k_j = 1, 2, 3, \ldots$ and $j = 1, 2$. The overall weight of the generated codeword can be calculated as

$$d = 6 + (k_1 + k_2) \times (z_{min} - 2) \tag{4.14}$$

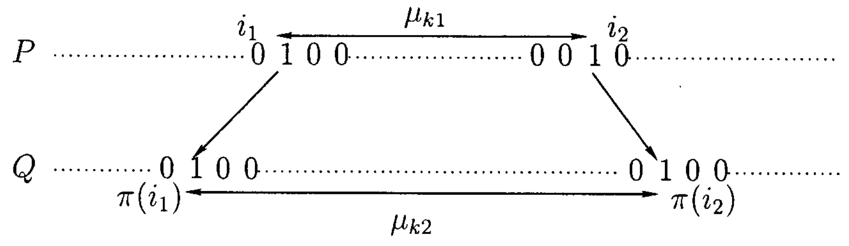


Fig. 4.5: A weight-2 input sequence is mapped to a sequence with the same pattern

Let i_1 and i_2 denote the positions of "1" in the weight-2 input sequence and $\pi(i_1)$ and $\pi(i_2)$ denote the positions of "1" in the interleaved input sequence. If an interleaver mapping function meets the following condition

$$|i_1 - i_2| \bmod \mu = 0 \;\; and \;\; |\pi(i_1) - \pi(i_2)| \bmod \mu = 0 \tag{4.15}$$

This interleaver will map a divisible input sequence, to another divisible input sequence as shown in Fig. 4.5. As a result, both encoders will generate finite weight parity-check sequences which is detrimental to the overall code error performance.

In order to avoid this type of mapping, the interleaver should have the following condition:

$$|\pi(i_1) - \pi(i_2)| \bmod \mu \neq 0 \;\; whenever \;\; |i_1 - i_2| \bmod \mu = 0 \tag{4.16}$$

However, only those input patterns that generate low weight codewords with large contributions to the performance need to be eliminated in the interleaver design. Let d^2_{max} denote the maximum weight of the codewords generated by the weight-2 input patterns that should be eliminated by the interleaving. The value of the d^2_{max} can be determined by the relative contribution integral. From Equation (4.14) we have:

$$6 + (k_1 + k_2) \times (z_{min} - 2) \leq d^2_{max} \tag{4.17}$$

$$k_1 + k_2 \leq \frac{d_{max}^2 - 6}{z_{min} - 2} \tag{4.18}$$

This shows that only patterns with $k_1 + k_2$ satisfying Equation (4.18) need to be eliminated.

In the case of weight-3 input patterns that generate low weight codewords, the mapping from one weight-3 input pattern to another weight-3 input pattern that generates low-weight codewords is not easy to make. Nevertheless, a similar method described above can be used to break those patterns. Another method that can be used to break these patterns is the S-random constraint. An interleaver with the S-random constraint can either break a short weight-3 input pattern with lengths up to $S + 1$ or expand it to a longer one with lengths more than $2(S + 1)$. These longer input patterns will produce higher weight codewords.

Now we will focus on the higher weight input patterns that generate compound error events. A compound error event is defined as a concatenation of two or more single error events. Consider weight-4 input sequences that generate compound error events consisting of two single error events, which are generated by two weight-2 input patterns. The weight-4 input sequences can be represented by a polynomial

$$P(D) = (1 + D^{\mu k_1'})D^{\tau_1} + (1 + D^{\mu k_2'})D^{\tau_2} \tag{4.19}$$

where $k_1', k_2' = 1, 2, 3, \ldots$ and τ_3 and τ_4 are time delays. The input to the second encoder after interleaving is :

$$Q(D) = (1 + D^{\mu k_3'})D^{\tau_3} + (1 + D^{\mu k_4'})D^{\tau_4} \tag{4.20}$$

where $k_1', k_2' = 1, 2, 3, \ldots$ and τ_3 and τ_4 are time delays. The overall weight of the

the generated codeword can be calculated as

$$\begin{aligned} d &= 4 + (k_1' + k_2') \times (z_{min} - 2) + 4 + (k_3' + k_4') \times (z_{min} - 2) + 4 \\ &= 12 + (k_1' + k_2' + k_3' + k_4') \times (z_{min} - 2) \end{aligned} \tag{4.21}$$

Let i_1, i_2, i_3 and i_4 denotes the position of "1"s in the weight-4 input sequence, where $i_1 < i_2 < i_3 < i_4$ and $\pi(i_1)$, $\pi(i_2)$, $\pi(i_3)$ and $\pi(i_4)$ the positions of "1"s in the interleaved sequence. If an interleaver mapping meets the following conditions:

$$\begin{cases} |i_1 - i_2| \bmod \mu = 0 \; and \; |i_3 - i_4| \bmod \mu = 0 \\ |\pi(i_1) - \pi(i_3)| \bmod \mu = 0 \; and \; |\pi(i_2) - \pi(i_4)| \bmod \mu = 0 \end{cases}$$

or

$$\begin{cases} |i_1 - i_2| \bmod \mu = 0 \; and \; |i_3 - i_4| \bmod \mu = 0 \\ |\pi(i_1) - \pi(i_4)| \bmod \mu = 0 \; and \; |\pi(i_2) - \pi(i_3)| \bmod \mu = 0 \end{cases}$$

This interleaver will map a divisible input sequence, to another divisible input sequence as shown in Fig. 4.6. Both encoders generate finite weight parity-check sequences which is detrimental to the overall code error performance.

In order to avoid this type of mapping, the interleaver should have the following condition:

$$\begin{aligned} &|\pi(i_1) - \pi(i_3)| \bmod \mu \neq 0 \; and \; |\pi(i_2) - \pi(i_4)| \bmod \mu \neq 0 \\ &whenever |i_1 - i_2| \bmod \mu = 0 \; and \; |i_3 - i_4| \bmod \mu = 0 \end{aligned} \tag{4.22}$$

$$\begin{aligned} &|\pi(i_1) - \pi(i_4)| \bmod \mu \neq 0 \; and \; |\pi(i_2) - \pi(i_3)| \bmod \mu \neq 0 \\ &whenever |i_1 - i_2| \bmod \mu = 0 \; and \; |i_3 - i_4| \bmod \mu = 0 \end{aligned} \tag{4.23}$$

As in the case of weight-2 input patterns, only those weight-4 input patterns that

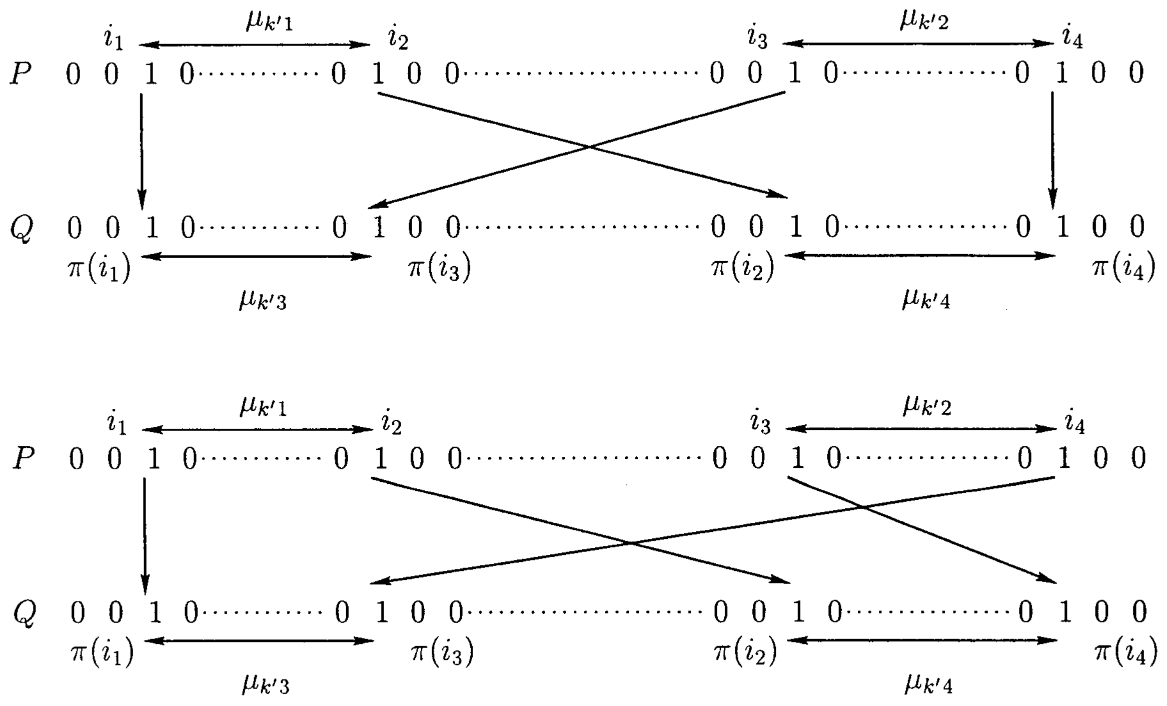


Fig. 4.6: A weight-4 input sequence is mapped to a sequence with the same pattern

generate low weight codewords, which have large contributions to the error performance, need to be eliminated. Therefore, if only the weight-4 input patterns that generate codewords with weight less than d^4_{max} are considered, so

$$12 + (k'_1 + k'_2 + k'_3 + k'_4) \times (z_{min} - 2) \leq d^4_{max} \tag{4.24}$$

$$k'_1 + k'_2 + k'_3 + k'_4 \leq \frac{d^4_{max} - 12}{z_{min} - 2} \tag{4.25}$$

This shows that only patterns with $k'_1 + k'_2 + k'_3 + k'_4$ satisfy Equation (4.25) need to be eliminated.

The effect of input patterns with weight bigger than four, on code error performance is small and they can be typically broken by the S-random constraint. Therefore, these input patterns are not important and do not need to be considered in the interleaver design, provided that the conditions of breaking weight-2, weight-3 and weight-4 input patterns are satisfied. In the next section we use these results, in order to design a code-matched interleaver for the 3GPP system.

4.5.2 Design of Code Matched Interleaver for 3GPP

In this section, we design a code-matched interleaver for a 3GPP system. Second generation (2G) mobile communication systems, which have been designed for mobile digital telephony, have a very low data capacity [32]. For example, GSM supports only data rate of $9.6kbit/s$. In order to transmit video and image on a mobile communication system, third generation (3G) standards have been proposed. International Telecommunication Union (ITU) has developed a standard called IMT-2000, which is capable of transmitting data rates up to $2Mbit/s$. Third Generation Partnership Project (3GPP) organization, has proposed 3GPP standard for 3G systems based on IMT-2000, which has been accepted by Europe and Japan. This standard has a Turbo code option which is shown in Fig. 4.7 and has generator polynomial $g_2(D)/g_1(D)$ where $g_2(D) = (15)_{oct}$ and $g_1(D) = (13)_{oct}$ [33].

In order to design a code-matched interleaver for this system, our first step is to find the weight enumerator functions for each constituent code and then find the weight enumerator function for the parallel concatenated code. For each constituent code, we use the method in [34] and [35] to find the weight enumerator functions. The state transitions will be presented by *state transition matrix* $\mathbf{A}(L, I, D)$ which is a $2^m \times 2^m$ matrix. Each element in this matrix has the monomial form of $L^l I^i D^d$ where l is the length and is always equal to 1, i and d are either 0 or 1, depending on whether the corresponding state transition, input and output bits are 0 or 1, respectively.

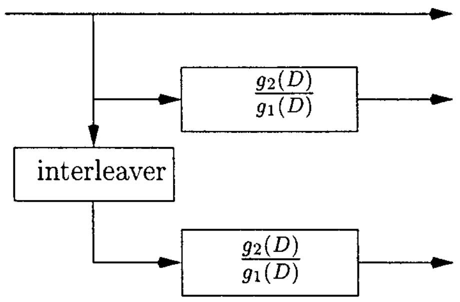


Fig. 4.7: 3GPP Encoder $N = 1440$

For the 3GPP system , the *state transition matrix* is equal to:

$$\mathbf{A}(L, I, D) = \begin{bmatrix} L & 0 & 0 & 0 & LID & 0 & 0 & 0 \\ LID & 0 & 0 & 0 & L & 0 & 0 & 0 \\ 0 & LI & 0 & 0 & 0 & LD & 0 & 0 \\ 0 & LD & 0 & 0 & 0 & LI & 0 & 0 \\ 0 & 0 & LD & 0 & 0 & 0 & LI & 0 \\ 0 & 0 & LI & 0 & 0 & 0 & LD & 0 \\ 0 & 0 & 0 & LID & 0 & 0 & 0 & L \\ 0 & 0 & 0 & L & 0 & 0 & 0 & LID \end{bmatrix} \tag{4.26}$$

Using the method in [34], the *transfer function* is defined by

$$T(L, I, D) = \sum_{l \geq 0} \sum_{i \geq 0} \sum_{d \geq 0} L^l I^i D^d t(l, i, d) \tag{4.27}$$

where $t(l, i, d)$, is the number of paths of length l, input weight i, and output weight d, starting and ending in zero state. If we ignore the effect of trellis termination[1], transfer function can be calculated as:

$$T(L, I, D) = [(\mathbf{I} - \mathbf{A}(L, I, D))^{-1}] \tag{4.28}$$

[1] In the original method, the transfer function considering trellis termination is equal to $T(L, I, D) = [(\mathbf{I} - \mathbf{A}(L, I, D))^{-1} \times A(1, 1, D)^m]$, where m is the code memory

The transfer function for the 3GPP system is equal to:

$$T(L,I,D) = \frac{N(L,I,D)}{D(L,I,D)}$$

where

$$\begin{aligned}
N(L,I,D) &= 1 - LID - L^2ID - L^3ID + L^4(I^3D^3 - ID) \\
&+ L^5(I^3D^3 - ID) + L^6(2I^3D^3 - I^5D - D^5I) \\
&+ L^7(2I^2D^2 - I^4 - 2I^4D^4 - D^4 + I^2D^6 + I^6D^2) \\
D(L,I,D) &= 1 - L(1+ID) + L^6(ID - ID^5 - I^5D + I^5D^5) \\
&+ L^7(-D^4 + ID^5 + 2I^2D^2 + I^2D^6 - 2I^3D^3 - I^3D^7 \\
&- I^4 - 2I^4D^4 + I^5D + 2I^5D^5 + I^6D^2 - I^7D^3) \\
&+ L^8(D^4 - 2I^2D^2 - 2I^2D^6 + I^4 + 4I^4D^4 \\
&+ I^4D^8 - I^6D^2 - 2I^6D^2 + I^8D^4)
\end{aligned} \tag{4.29}$$

From the transfer function we can get a recursive relation for $t(l,i,d)$.

$$\begin{aligned}
t(l,i,d) &= t(l-1,i,d) + t(l-1,i-1,d-1) - t(l-6,i-1,d-1) \\
&+t(l-6,i-1,d-5) + t(l-6,i-5,d-1) - t(l-6,i-5,d-5) \\
&+t(l-7,i,d-4) - t(l-7,i,d-5) + t(l-7,i-3,d-7) \\
&-2t(l-7,i-2,d-2) - t(l-7,i-2,d-6) + 2t(l-7,i-3,d-3) \\
&+t(l-7,i-4,d) + 2t(l-7,i-4,d-4) - t(l-7,i-5,d-1) \\
&+2t(l-7,i-5,d-5) + t(l-7,i-7,d-3) - t(l-7,i-6,d-2) \\
&+2t(l-8,i-2,d-6) - t(l-8,i-4,d) - 4t(l-8,i-4,d-4) \\
&-t(l-8,i,d-4) + 2t(l-8,i-2,d-2) + t(l-8,i-6,d-2)
\end{aligned} \tag{4.30}$$

$$-t(l-8,i-4,d-8)+2t(l-8,i-6,d-2)-t(l-8,i-8,d-4)$$
$$+\delta(l,i,d)-\delta(l-1,i-1,d-1)-\delta(l-2,i-1,d-1)$$
$$+\delta(l-3,i-1,d-1)-\delta(l-4,i-1,d-1)+\delta(l-4,i-3,d-3)$$
$$-\delta(l-5,i-1,d-1)+\delta(l-5,i-3,d-3)+2\delta(l-6,i-3,d-3)$$
$$-\delta(l-6,i-5,d-1)-\delta(l-6,i-1,d-5)-2\delta(l-7,i-4,d-4)$$
$$+2\delta(l-7,i-2,d-2)+\delta(l-7,i-6,d-2)-\delta(l-7,i-4,d)$$
$$+\delta(l-7,i-2,d-6)-\delta(l-7,i,d-4)$$

Based on the above recursive relation, the input-redundancy weight enumerating function (IRWEF) for one encoder can be found Equation (4.31).

$$A^{C_p}(w,z)=\sum_{z=z_1+z_2}\frac{A^{C_1}(w,z_1)\cdot A^{C_2}(w,z_2)}{\begin{pmatrix} k \\ w \end{pmatrix}} \tag{4.31}$$

The IRWEF for the turbo code is calculated from, with similar constituent encoders. The values for $A^{C_p}(w,z)$ are shown for low-weight input sequences in Table 4.4. Using the equations for design of code-matched interleaver given in Section 4.5, we have also found the *relative contribution* of the first spectral lines to the bit error rate which is depicted in Fig. 4.8 and also the *relative contribution integral* shown in Table 4.4.

From Fig. 4.8, we see that for given 3GPP system, the codewords with input weight 9, 13, 14, 17 and 18 are the dominant error events in the SNR range of $[2,8]dB$, this also agrees with the values for *relative contribution integral* that we found for these weights. These weights can be generated by different input weights, e.g., sequences with weight 13 can be generated by a weight-3 input and two parity sequences with weights 3 and 7 or by a weight-7 input and two parity sequences with weights 3 and 3. At first glance, one may think that we have to remove both weight-3 and weight-7 input patterns that cause such patterns, in order to remove the codewords with weight-13. But if we check the value of $A(w, j)$ for these sequences, we see that the number of patterns with weight-7 is only 5.2146e-11 compared to 1.3234e-01 of the patterns with weight-3. As a result, we can assume that if we remove the ones with weight-3 input, we have nearly removed all weight-13 codewords. This fact is also true for other weighs and explains why in the design of interleaver we just remove the patterns with low weight inputs (2, 3 and 4).

Now that we have found the weights that we need to remove, we have to determine the input patterns that need to be eliminated. As seen in Table 4.4, we need to remove weight-2 inputs with codeword weights less than 22. It can be seen that there is no dominant weight-4 input patterns that need to be eliminated and weigh-3 inputs will be removed using a S-random interleaver. Using Equation (4.18) we can find the upper limits for weight-2 inputs that need to be eliminated:

$$k_1 + k_2 \leq \frac{d^2_{max} - 6}{z_{min} - 2} = \frac{22 - 6}{6 - 2} = 4 \tag{4.32}$$

The values for z_{min} and μ depend on the encoder structure and are 6 and 7, respectively. The conditions we used in order to design our code-matched interleaver can be summarized as:

1. $|\pi(i_1) = \pi(i_2)| \bmod \mu \neq 0 \textit{ whenever } |i_1 - i_2| \bmod \mu = 0$
   where $k_1 + k_2 \leq 4$

2. $|\pi(i_1) = \pi(i_2)| \geq (S = 27) \textit{ whenever } |i_1 - i_2| \leq (S = 27)$

The simulation results for the designed interleaver are given in Fig 4.9 and Fig. 4.10. These results are compared with the result of a S-random interleaver with the same S parameter. As it can be seen the performance of the code has been improved in error-floor region, due to the increase of the minimum weight of the code.

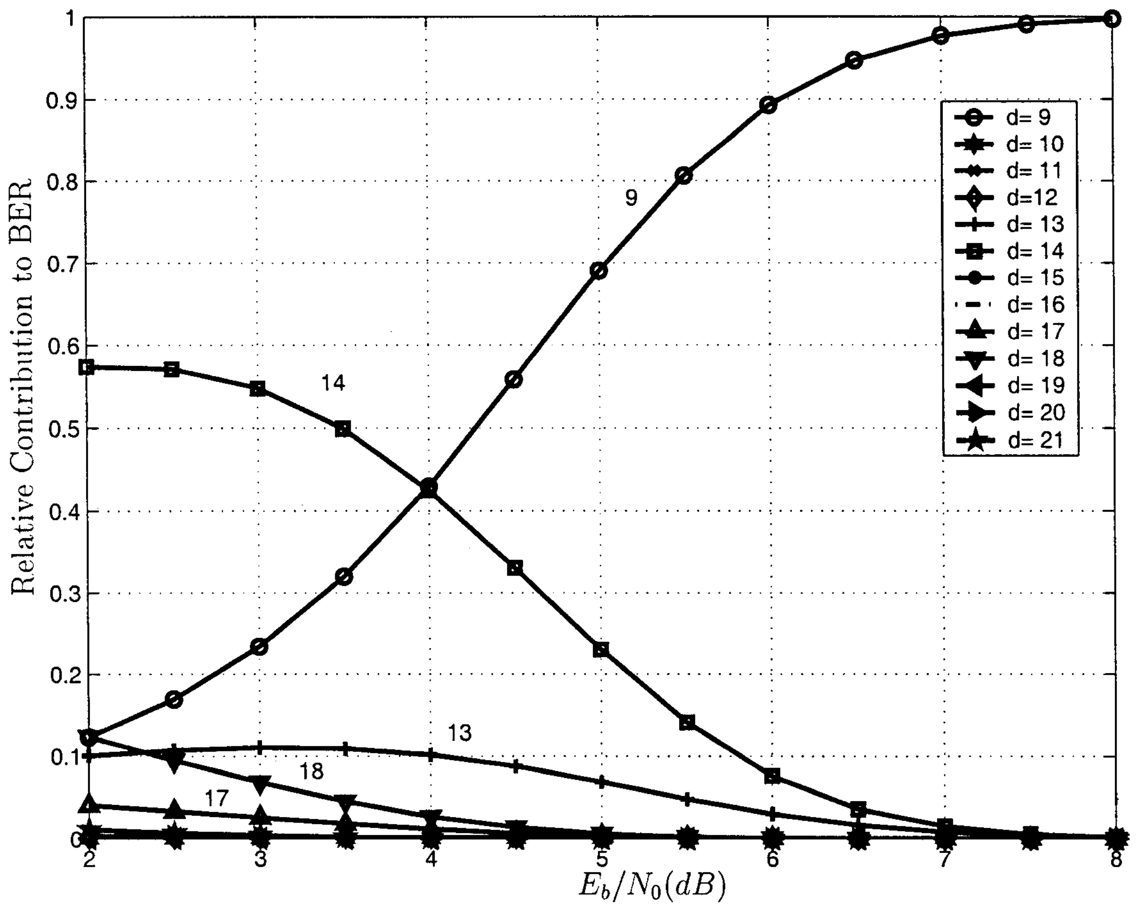


Fig. 4.8: Relative Contribution to Bit Error Rate $N = 1440$

Table **4.4**: Relative Contribution Integral

weight	$B(d)$	$\overline{F^{d}_{ab}}$	Input weight	Parity weight	$A(w,j)$
9	$3.4602e-05$	$6.3081e+01$	3	3	1.6609e-02
10	$6.99818e-13$	$3.4275e-07$	6	2	1.6756e-10
12	$2.0462e-06$	$1.1480e-01$	4	4	7.3662e-04
13	$2.757e-04$	$6.1596e+00$	3	$3,7$	1.3234e-01
			7	3	5.2148e-11
14	$2.7528e-03$	$2.6289e+01$	2	6	1.9820e+00
			6	$2,6$	1.0068e-06
			10	2	1.9860e-19
15	$1.5084e-07$	$6.4827e-04$	5	5	4.3443e-05
16	$2.3943e-05$	$4.8107e-02$	4	$4,8$	8.6196e-03
			8	4	1.004e-11
17	$1.0294e-03$	$9.9567e-01$	3	$3,7,11$	4.9410e-01
			7	$3,7$	3.0603e-07
			11	3	2.9835e-19
18	$5.4850e-03$	$2.6138e+00$	2	$6,10$	3.9446e+00
			6	$2,6,10$	1.5243e-03
			10	$2,6$	1.4109e-14
			14	2	1.1346e-27
19	$2.8409e-05$	$6.7960e-03$	5	$5,9$	8.1819e-03
			9	5	2.7588e-10
20	$4.9460e-04$	$6.0327e-02$	4	$4,8,12$	1.7806e-01
			8	$4,8$	6.1244e-08
			12	4	8.9238e-18
21	$2.5979e-03$	$1.6370e-01$	3	$3,7,11,15$	1.2459e+00
			7	$3,7,11$	4.5260e-04
			11	$3,7$	1.1857e-14
			15	3	4.7570e-27
22	$8.3262e-03$	$2.7411e-01$	2	$6,10,14$	5.8879e+00
			6	$2,6,10,14$	3.5645e-02
			10	$2,6,10$	2.5088e-10
			14	$2,6$	3.8316e-22
			18	2	2.0036e-35

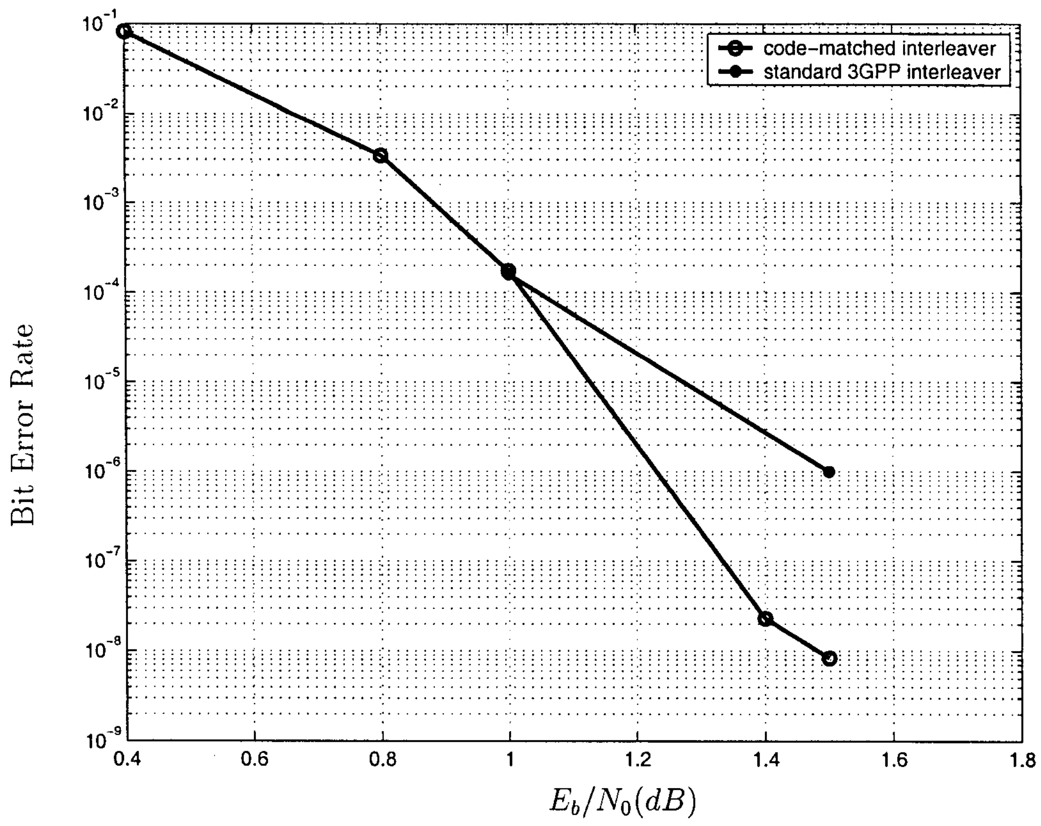


Fig. 4.9: Bit Error Rate for the 3GPP system, $N = 1440$, number of iterations =15

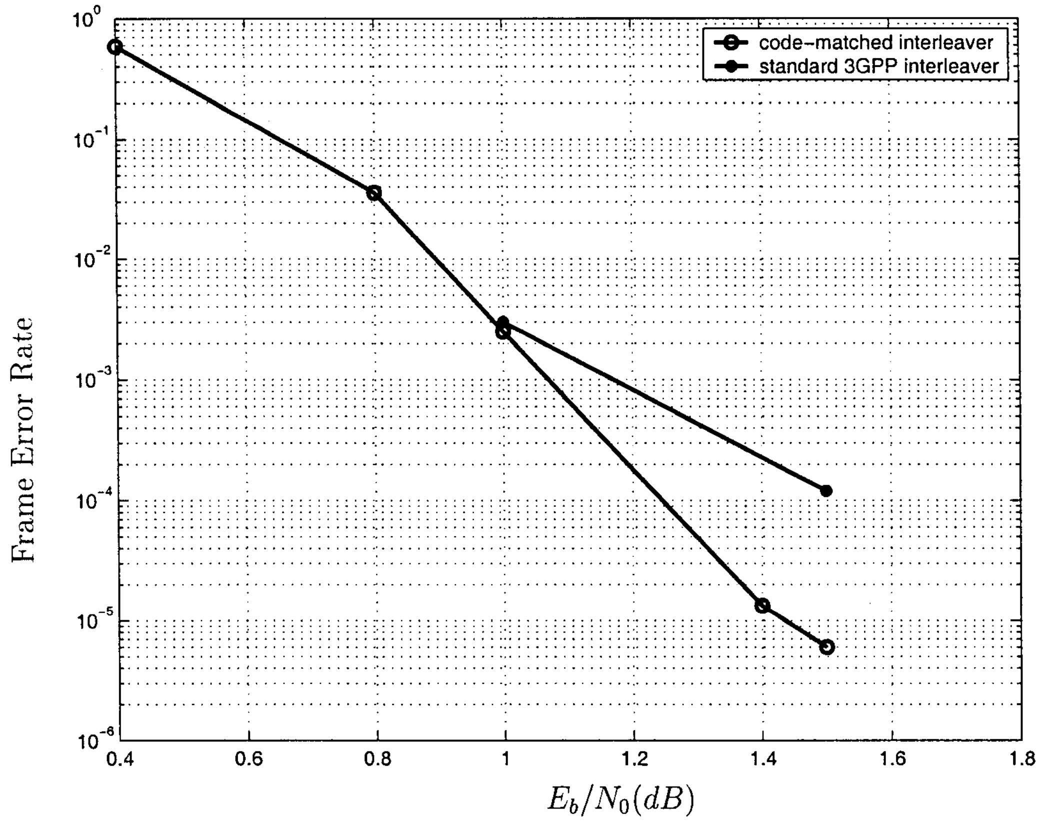


Fig. 4.10: Frame Error Rate for the 3GPP system, $N = 1440$, number of iterations $=15$

4.6 summary

In this chapter we studied the criteria which are used in design of interleavers, in order to improve both distance spectrum and the iterative decoding suitability of the code. S-random and code-matched interleavers were presented in detail. We also included the simulation results for a code-matched interleaver design for 3GPP system.

Chapter 5

Joint Interleaver Design for Multiple Turbo Codes

As discussed earlier, the performance of turbo codes is deteriorated in *error-floor* region, due to the low values of free distance of turbo codes. As we have seen in Chapter 4, improving the free distance of turbo codes, require to eliminate certain patterns that generate low-weight outputs from the input of the encoders.

As we have seen, criteria to improve the weight spectrum of the code and also criteria to improve the performance of the iterative decoding is known to us. While these criteria are known, the fact that hold us from designing a *good* interleaver is how to find an interleaver for given length that satisfy these conditions, in a reasonable search time (converge). In these cases we have to loosen the conditions in order that interleaver can converge, which will deteriorate the performance of the code.

In the case of *Multiple Turbo Codes*, where we have three or more constituent codes and we have at least two interleavers, each interleaver can be designed to eliminate some of the unwanted patterns at the input of the encoder and other patterns are eliminated by the other interleavers. This will decrease the burden on one interleaver to satisfy all the conditions at the same time and enables us to satisfy tougher conditions.

In this chapter, we see some simulation results for multiple turbo codes. Later we will introduce two methods for joint interleaver design for multiple turbo codes and investigate their effect through lower bound calculations and scatter plots.

5.1 Multiple Turbo Codes

Multiple turbo codes have been discussed in detail in Section 2.5. In this section, we present some practical issue that we were faced while simulating a multiple turbo coding system and some simulation results which are the basis for the future sections.

We have used the multiple turbo coding system presented by Divsalar in [15], whose encoder consists of three similar RSC encoders with memory size 2 with a total code rate of 1/4. Two versions of this system are shown in Fig. 5.1. In the first version (C_{57}), the constituent encoder has the generator polynomial $(5, 7)_{oct}$ and for the second one (C_{75}) the generator polynomial is $(7, 5)_{oct}$. In [15], simulation results for a 3-memory encoder is also given with generator polynomial $(11, 17)_{oct}$, which is the same as 2-memory code $(7, 5)_{oct}$. [1]

As seen in Section 2.5, different configurations can be chosen for the decoders. We have simulated two of the best performing configurations, which are extended serial (ES) and parallel (P) configurations. Each decoder consisted of three Log-MAP decoders and extrinsic information is added before each decoder, the only difference between the two configurations is that in the extended serial configuration the extrinsic information is updated after each decoder finishes its operation, whereas in parallel configuration the extrinsic information is updated after one complete iteration.

The Bit error rate (BER) and Frame error rate (FER) for the described systems is given in Fig. 5.2 and Fig. 5.3. The simulation results are for block length $N =$

[1] In [10] the generator polynomial for 3-memory code by mistake is mentioned as $(11, 13)_{oct}$

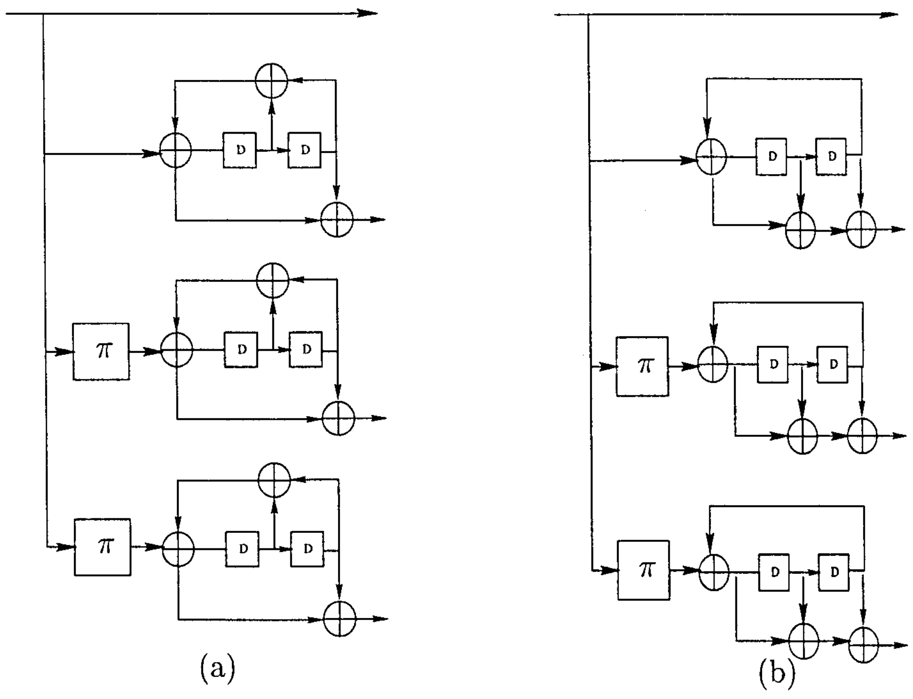


Fig. 5.1: Multiple Turbo Code with constituent encoder (a)$(5,7)_{oct}$ (b)$(7,5)_{oct}$

4096, with number of iterations equal to 20. The interleaver used is a S-random interleaver with S=31.

As expected, the performance of the extended serial configuration is better than the parallel configuration. It can also be seen that as we expected the 3-memory code $(11,17)_{oct}$, has approximately the same performance as the 2-memory code C_{75}. [2]

As it can be seen from Fig. 5.2 and Fig. 5.3 performance of the C_{75} is approximately 0.2 dB better than the code C_{57}. In order to explain this phenomena, we need to use *density evolution* method presented by Divsalar in [36].

[2] $(11,17)_{oct} = \frac{1+D^3}{1+D+D^2+D^3} = \frac{(1+D)(1+D+D^2)}{(1+D)(1+D^2)} = \frac{(1+D+D^2)}{(1+D^2)} = (7,5)_{oct}$

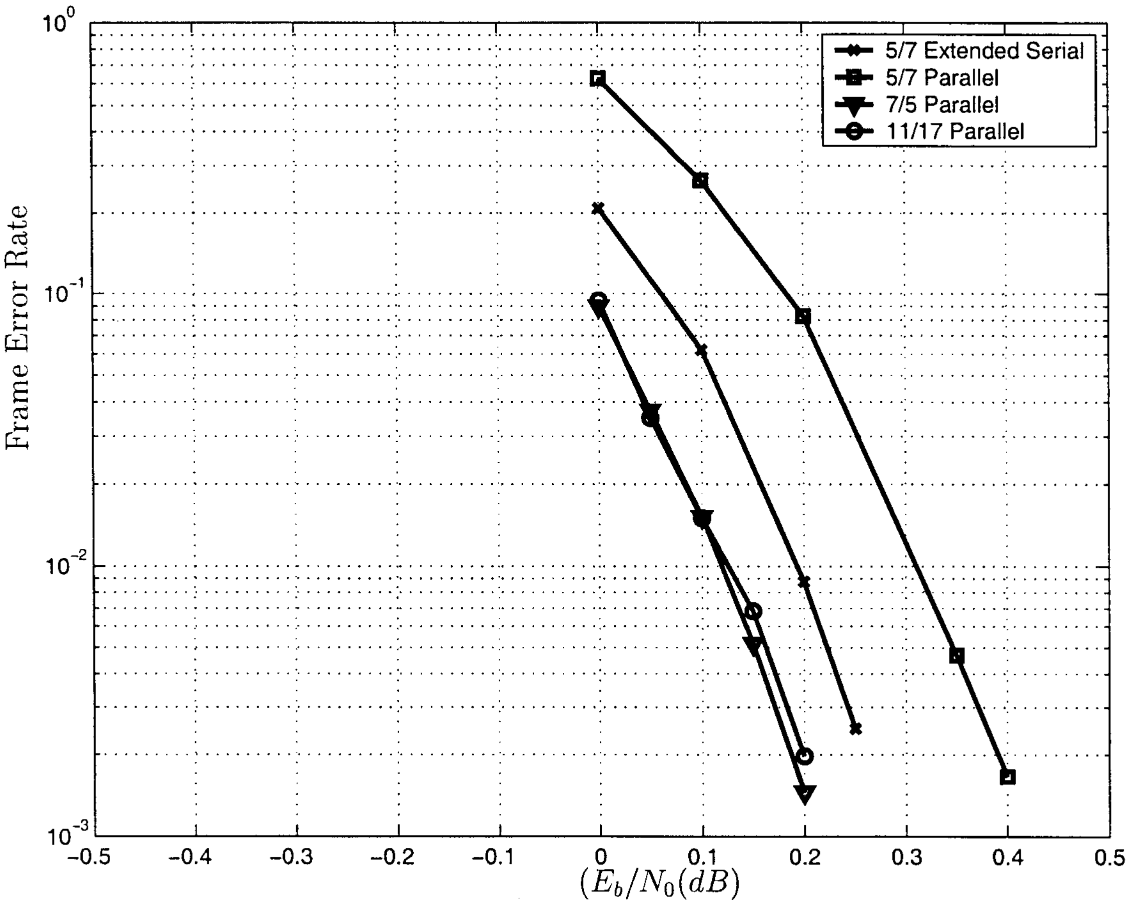


Fig. 5.2: Frame Error Rate of Multiple Turbo Codes Simulation (N = 4096 , r=1/4, S-random interleaver S=31)

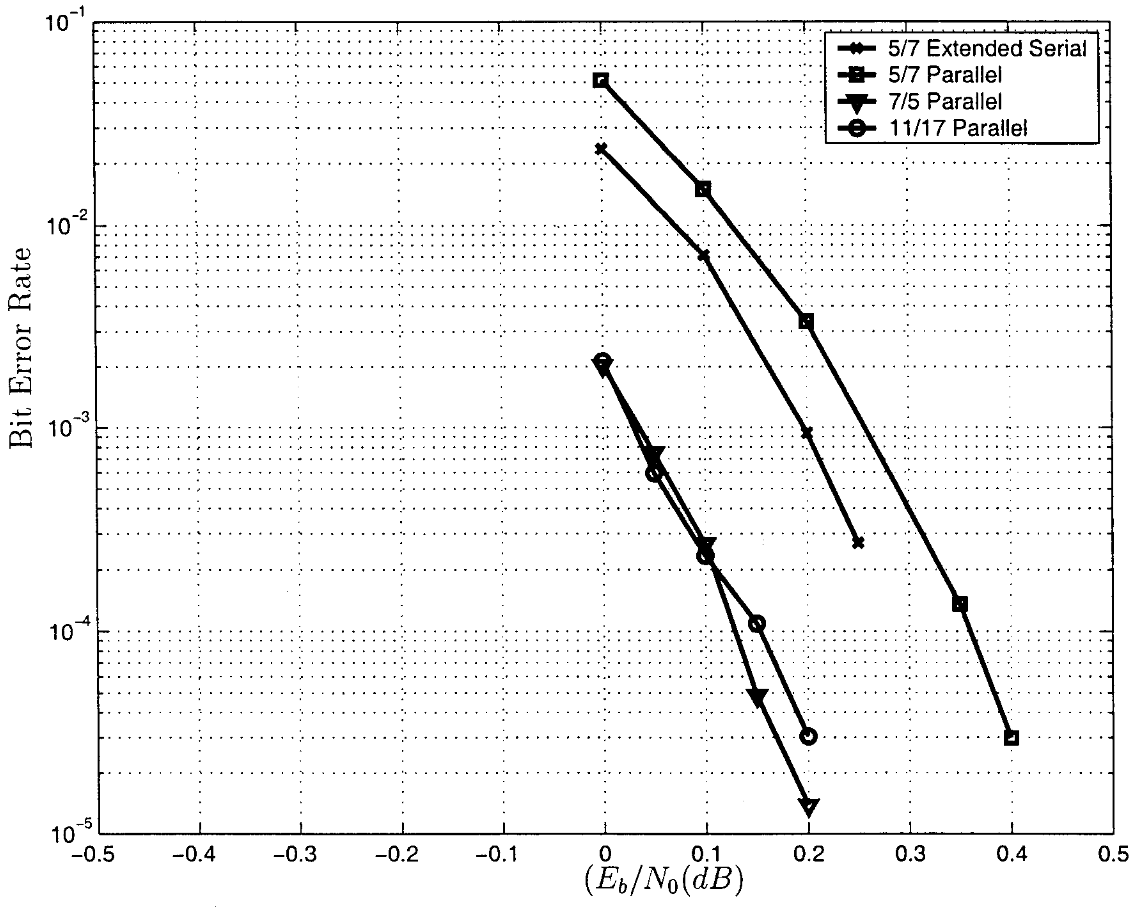


Fig. 5.3: Bit Error Rate of Multiple Turbo Codes Simulation (N = 4096 , r=1/4, S-random interleaver S=31)

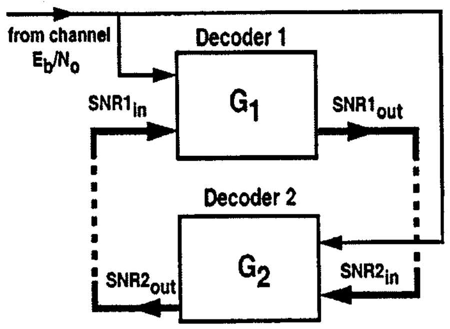


Fig. 5.4: Analysis of Turbo Decoding as a Nonlinear Dynamic System with Feedback Using Density Evolution

In this method, the density evolution is used to analyze the performance of iterative decoders for turbo codes. [3] Density of extrinsic information in iterative turbo decoder is tracked by actual density evolution which is approximated by symmetric Gaussian density functions. The evolution of these density functions is viewed through an iterative decoder which is viewed as a nonlinear dynamical system with feedback.

When the interleaver is very large and random the extrinsic information messages are independent and identically distributed, which will be approximated by a Gaussian density function. The probability density evolve with successive decoder iterations from narrow densities to broader Gaussian-shaped densities with increasing means as the iterations continue. A signal-to-noise ratio for this random variable can be defined as SNR= μ^2/σ^2, where μ and σ are the mean and variance of the Gaussian density function, respectively. For each decoder input and output, SNR can denoted by SNR$_{in}$ and SNR$_{out}$. In a turbo decoder, a nonzero E_b/N_0 from the channel enable the first decoder to produce a nonzero $SNR1_{out}$ for the output extrinsic information despite starting from SNR1$_{in}$ = 0. For a given value of E_b/N_0, the output SNR of each decoder is a nonlinear function of its input SNR, denoted by G_1, where we have SNR1$_{out}$ = G_1(SNR1$_{in}$, E_b/N_0). The same can be defined for

[3] the simulation results for density evolution is provided by Dr. Divsalar, answering the question about the better performance of C_{75} than C_{57}

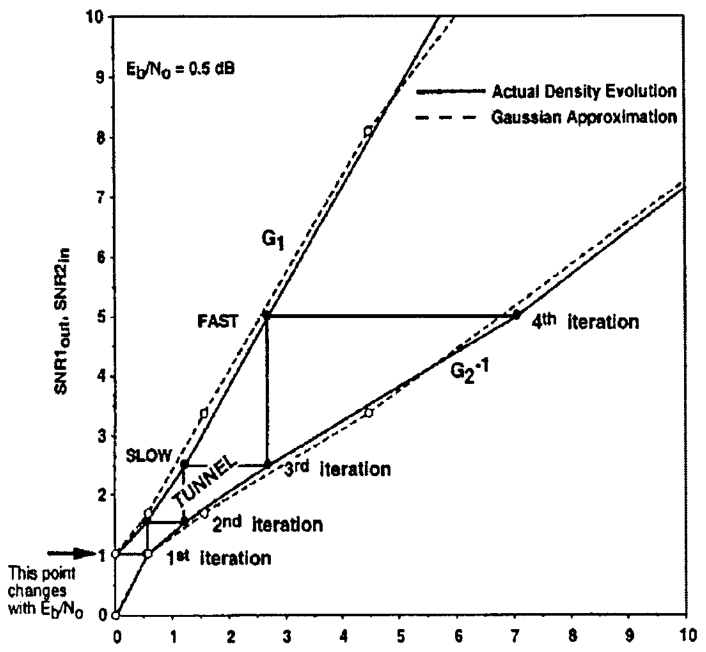


Fig. 5.5: Iterations and Convergence of a Turbo Decoder

other decoders, as shown in Fig. 5.4. The decoder's convergence can be assessed by measuring the change in the SNR of the extrinsic information from one iteration to the next. The convergence of the decoder can be tested by tracking the evolution of the extrinsic information's SNR. The analytical method is to plot the output SNR of decoder 1 versus its input SNR and the input SNR of decoder 2 versus its output SNR, as shown in Fig. 5.5. Fig. 5.5 graphically shows the progress of iterations. The improvement in the SNR of the extrinsic information and the corresponding improvement in the decoder's BER, follows a staircase path reflecting at right angles between the curves corresponding to G_1 and G_2^{-1}. The steps in this staircase are large when the bounding curves are far apart and small when they are close together. Where the curves are closest together, the improvement in BER slows down, as many iterations are required to bore through the narrow iterative decoding tunnel between the curves. If the iterative decoder successfully passes through the tunnel, convergence becomes very rapid as the two curves get farther apart at higher SNRs.

The initial displacement of the G_1 curve for $\text{SNR1}_{in} = 0$ is dependent on the E_b/N_0 due to the channel observations. If the value of 0.5 dB used in Fig. 5.5 is reduced, then at some point the two curves will just touch each other. That value represents the *iterative decoding threshold.* The iterative decoding tunnel will be closed at the SNR where the two curves touch and the staircase path will not go past this point. If E_b/N_0 is greater than this threshold, the decoder converges and the BER goes to zero as the iterations increase.

The density evolution has been employed for code C_{57} and C_{75} for length $N = 4096$, at $E_b/N_0 = -0.3\ dB$ and $E_b/N_0 = -0.5\ dB$ respectively. The results are presented in Fig. 5.6 and Fig. 5.7. These results show that the *iterative decoding threshold* for C_{75} is $0.2\ dB$ less than code C_{57}, that is the reason why this code enters the waterfall region sooner and performs better before error-floor region. At error-floor region as it can be seen from the figures the curves for code C_{57} are more separated. This means that we will have more gain with each iteration, this will result in better performance at error floor region for code C_{57}.

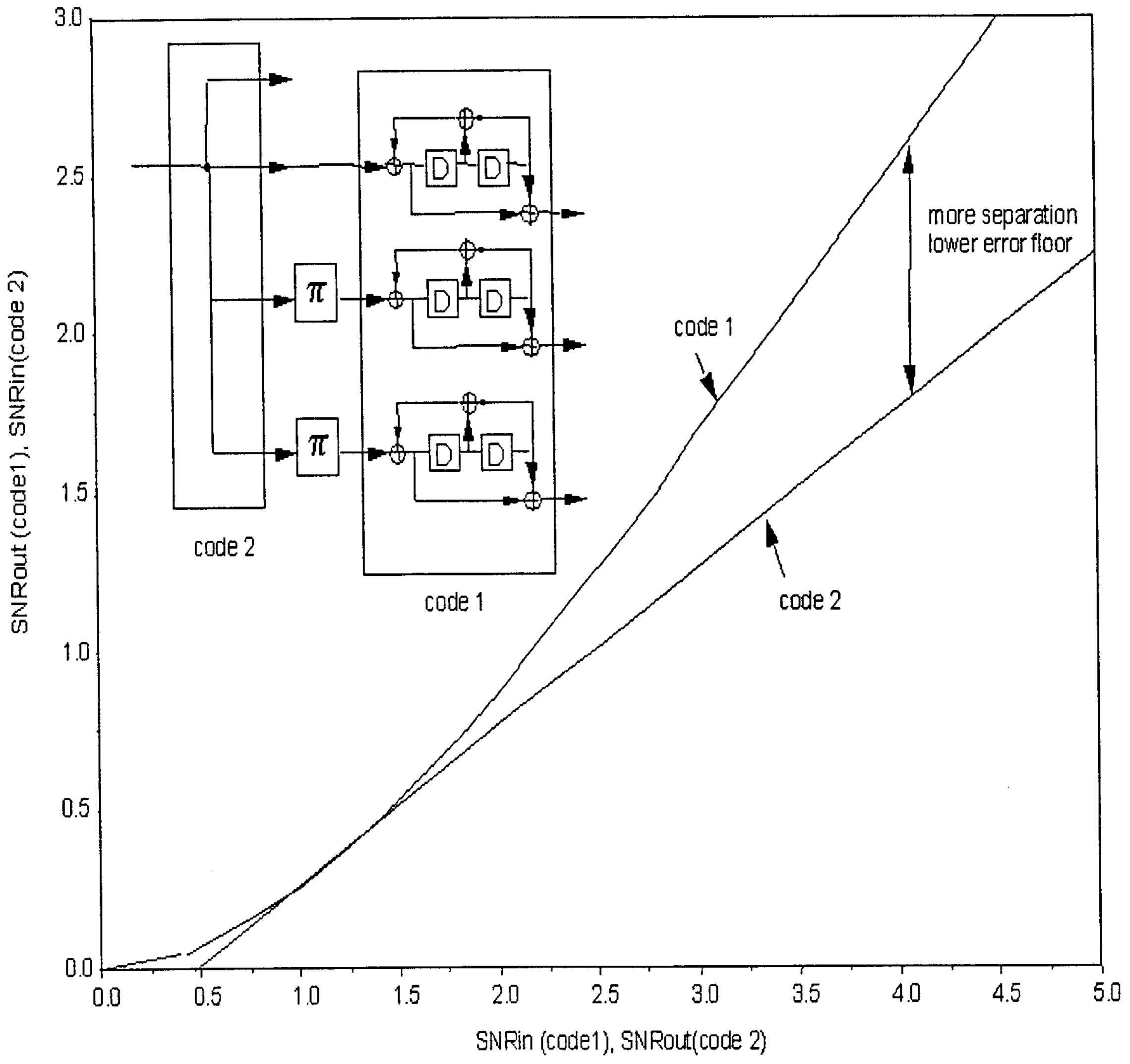


Fig. 5.6: Density evolution for Multiple Turbo Code C_{57}
(N=4096, r=1/4, $E_b/N_0 = -0.3dB$)

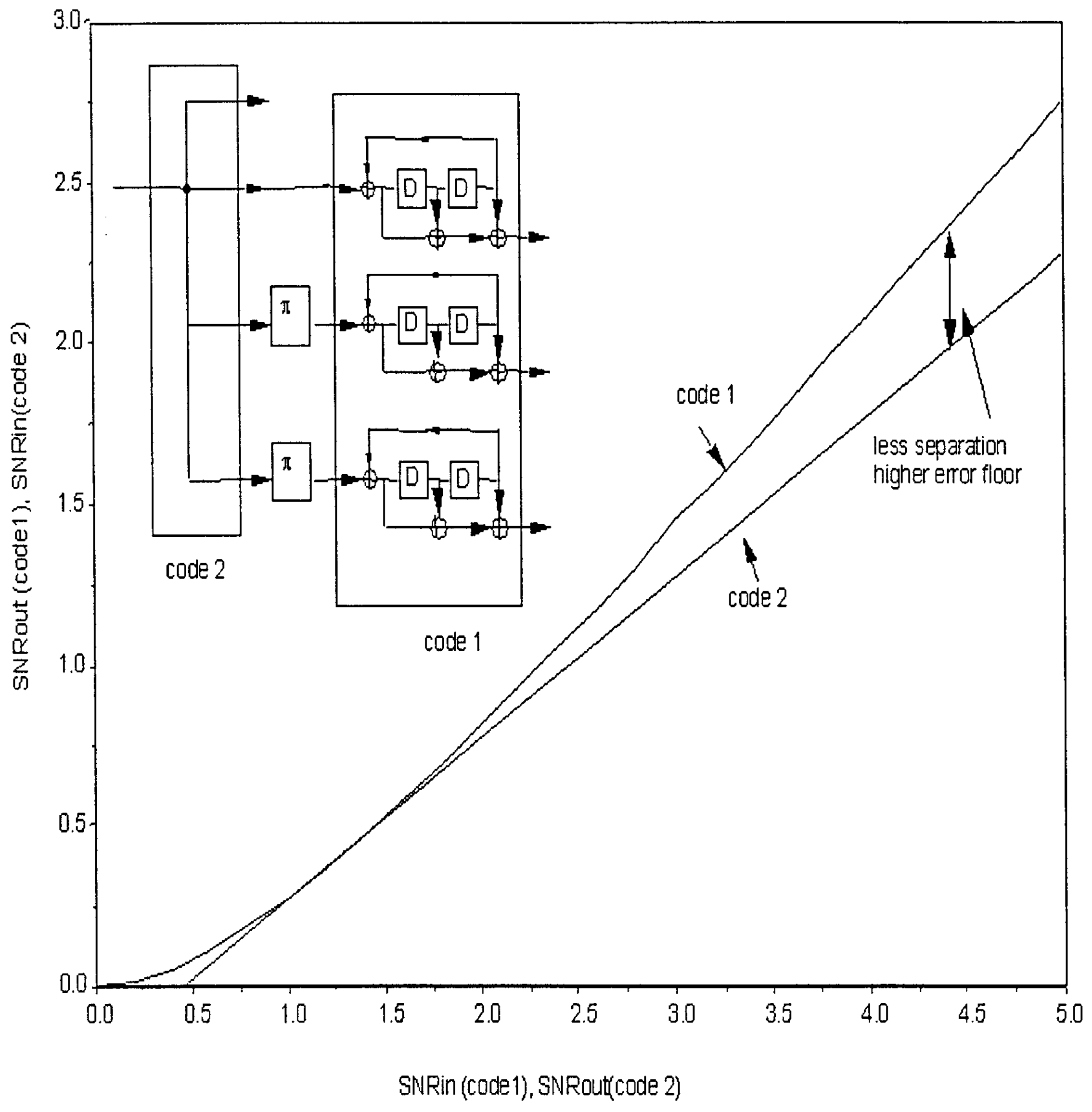


Fig. 5.7: Density evolution for Multiple Turbo Code C_{75}
(N=4096, r=1/4, $E_b/N_0 = -0.5dB$)

5.2 Performance Analysis of Multiple Turbo Codes

In the previous section, we have seen simulation results for two multiple Turbo code systems C_{57} and C_{75}. In this section, we examine their behaviour at error-floor region and in Section 5.2.2 we determine the parameters that affect their performance in error-floor region and outline the design criteria that we try to reach in future sections by means of *joint interleaver design* for multiple Turbo codes.

5.2.1 Performance Analysis in Error-Floor Region

In order to be able to examine the performance of two codes C_{57} and C_{75} at error-floor region, we have to simulate them for short frame lengths. Simulation results for these two systems are presented in Fig. 5.8 and Fig. 5.9 for length $N = 256$. We can see that in water fall region the performance of the C_{75} is better than that of the code C_{57}, but they cross over before error–floor region and at the error–floor region code C_{57} will show lower error–floor.

As we have seen in the previous chapters, the performance of turbo code at error floor region can be approximated using the lower bound of probability of error, which can be found provided that the weight distribution of the code is known. So,in order to examine the performance at the error-floor, first we have to find their weight distribution.

In order to be able to examine the effect of interleaver on weight distribution, in this section, unlike what we did in Chapter 4, we will not use uniform interleaver. For finding the weight distribution using the S-random interleaver, we will find the Hamming weight of the output codewords for all inputs with weight 2, 3 and 4. These are the input weights that usually cause low-weight codewords. After the output weights of all these input sequences are found, we can follow the formula in Section 4.5 to find the *Relative contribution* of each spectral line to the bit error

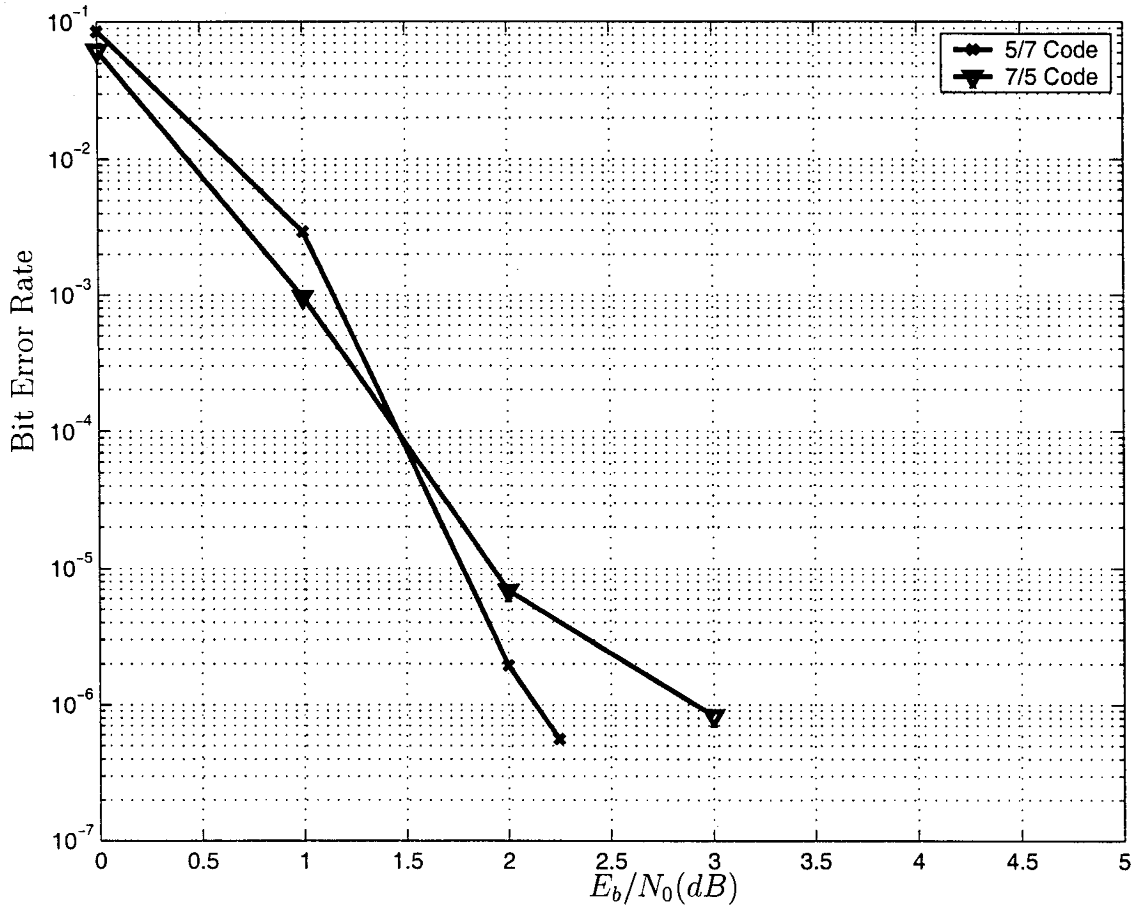


Fig. 5.8: Bit Error Rate for Multiple Turbo Code with N=256, Rate=1/4, S-random interleaver S=14

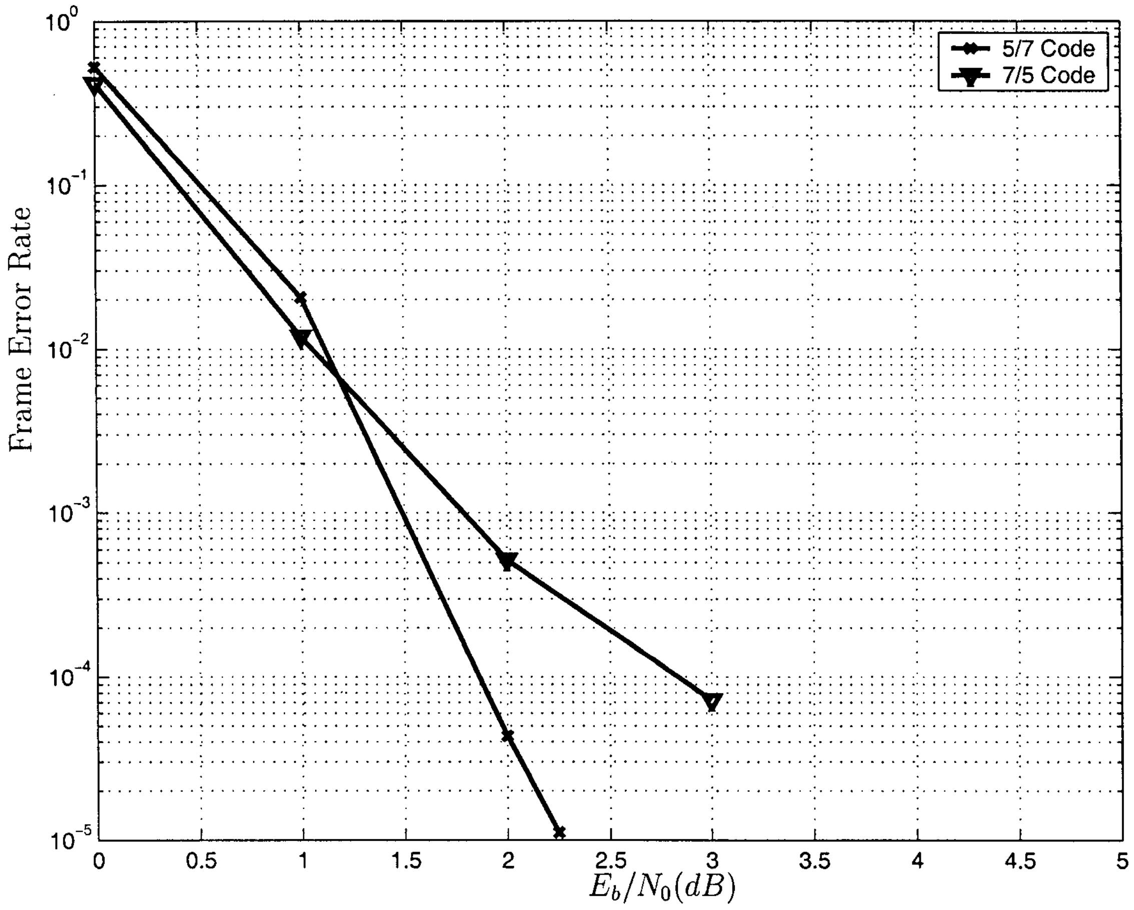


Fig. 5.9: Frame Error Rate for Multiple Turbo Code with N=256, Rate=1/4, S-random interleaver S=14

rate. The Equation (5.1) shows the lower bound for the probability of bit error.

$$P_b \leq \sum_{d=d_{free}} B_d Q\left(\sqrt{2dr\frac{E_b}{N_0}}\right) \tag{5.1}$$

The contribution of each spectral line to the probability of bit error is,

$$P_d(\gamma b) = B_d Q\left(\sqrt{2dr\gamma b}\right) \tag{5.2}$$

The relative contribution of each spectral line to bit error rate and contribution of each spectral line to probability of error for code (C_{57}), are shown in Fig. 5.10 and Fig. 5.11, respectively and for code (C_{75}), are shown in Fig. 5.12 and Fig. 5.13, respectively.

It can be seen from these figures that the dominant weights for the code C_{75} are lower than the weights for the code C_{57}, in other words the free distance of the first code is smaller than the second one, this will result in the poor performance at error-floor region. At SNR from 4 dB to 6 dB where the minimum weight terms dominate the performance, the lower bound of probability of error for code C_{57} is 0.01 times lower than the code C_{75}, Fig. 5.11 and Fig. 5.13.

It is clear that if we increase the minimum free distance of the code the performance of the code will improve considerably at error-floor region. In the next section based on the performance analysis, we will find the interleaver design criteria that we try to reach by means of joint interleaver design, in order to improve the performance at error–floor region.

5.2.2 Design Criteria for Multiple Turbo Codes

In this section, we outline the design criteria that we aim to fulfill by means of joint interleaver design, we will conclude the conditions that the interleaver has to satisfy in order to improve the performance at error-floor region. We will consider code C_{57}

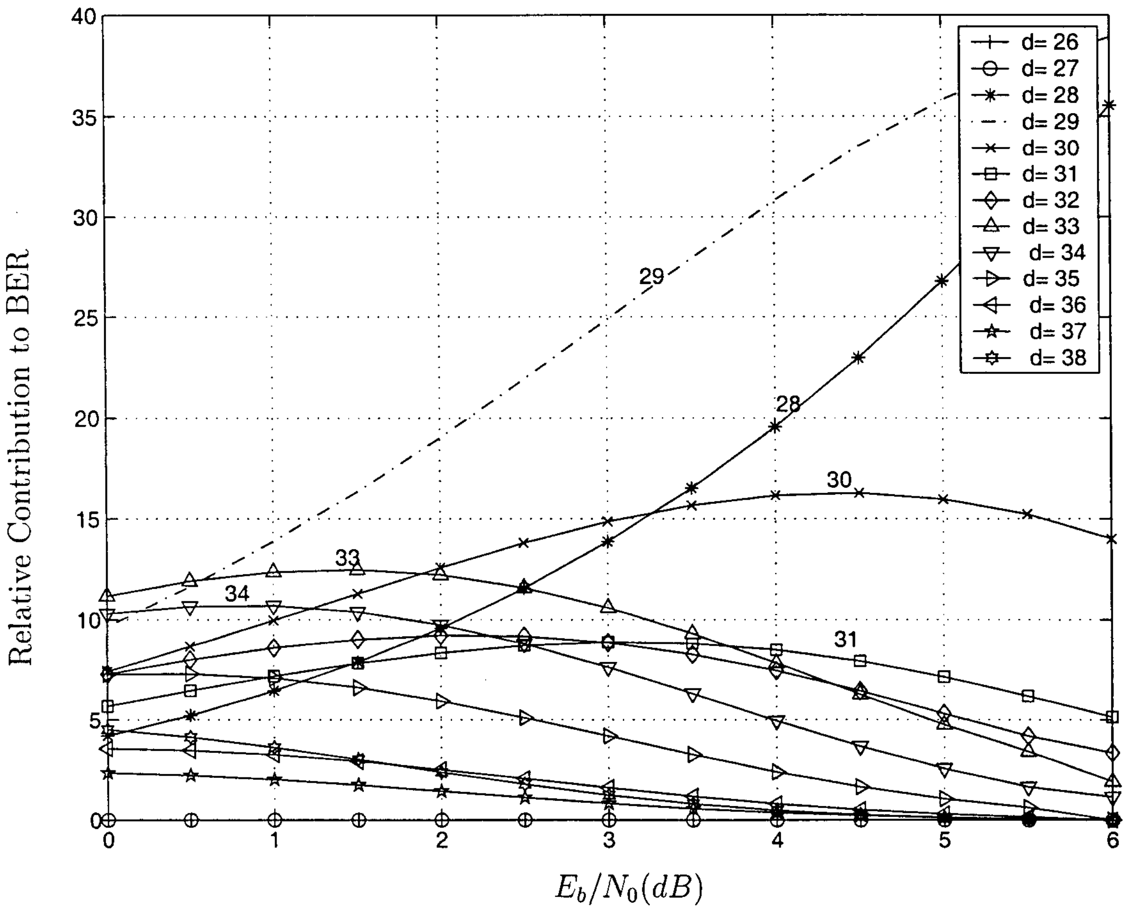


Fig. 5.10: Relative Contribution to Bit Error Rate code C_{57}, rate= 1/4, N=256

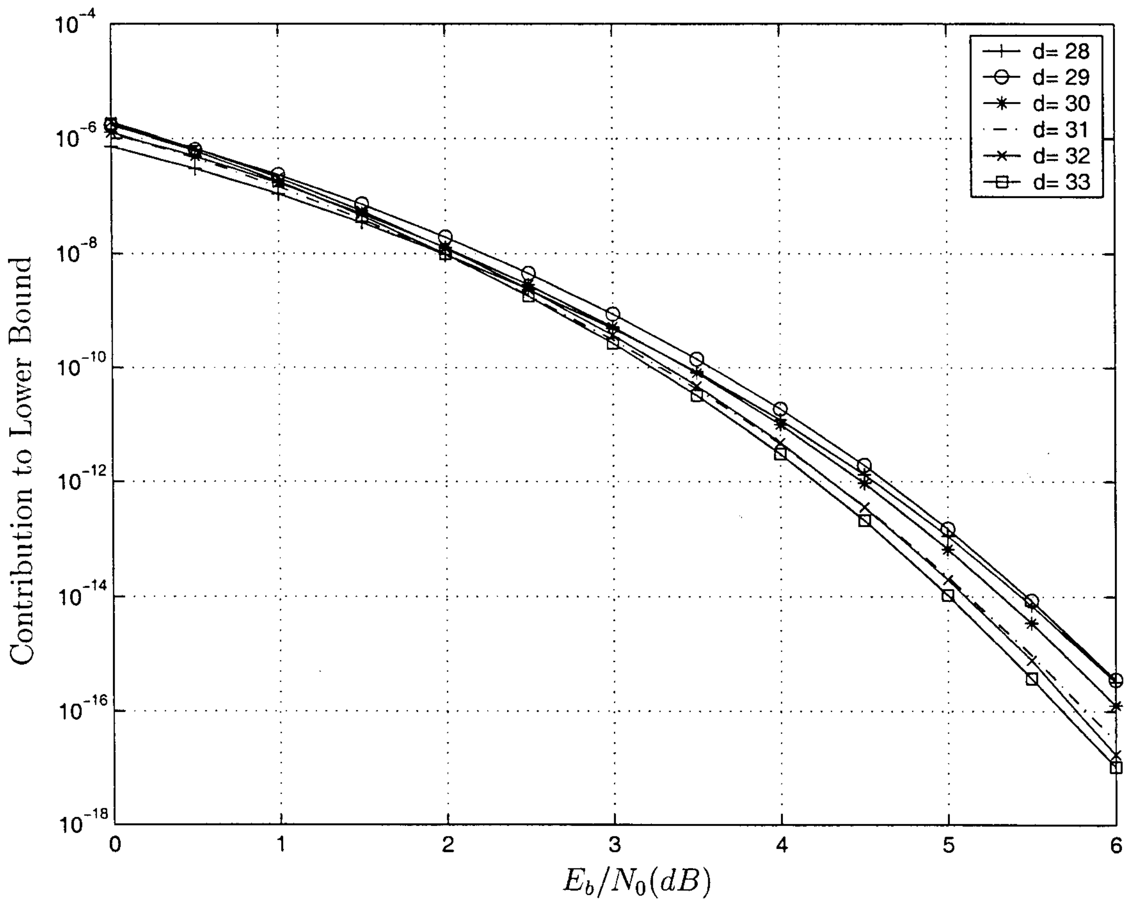


Fig. 5.11: Contribution to Lower Bound for Code C_{57}, rate= 1/4, N=256

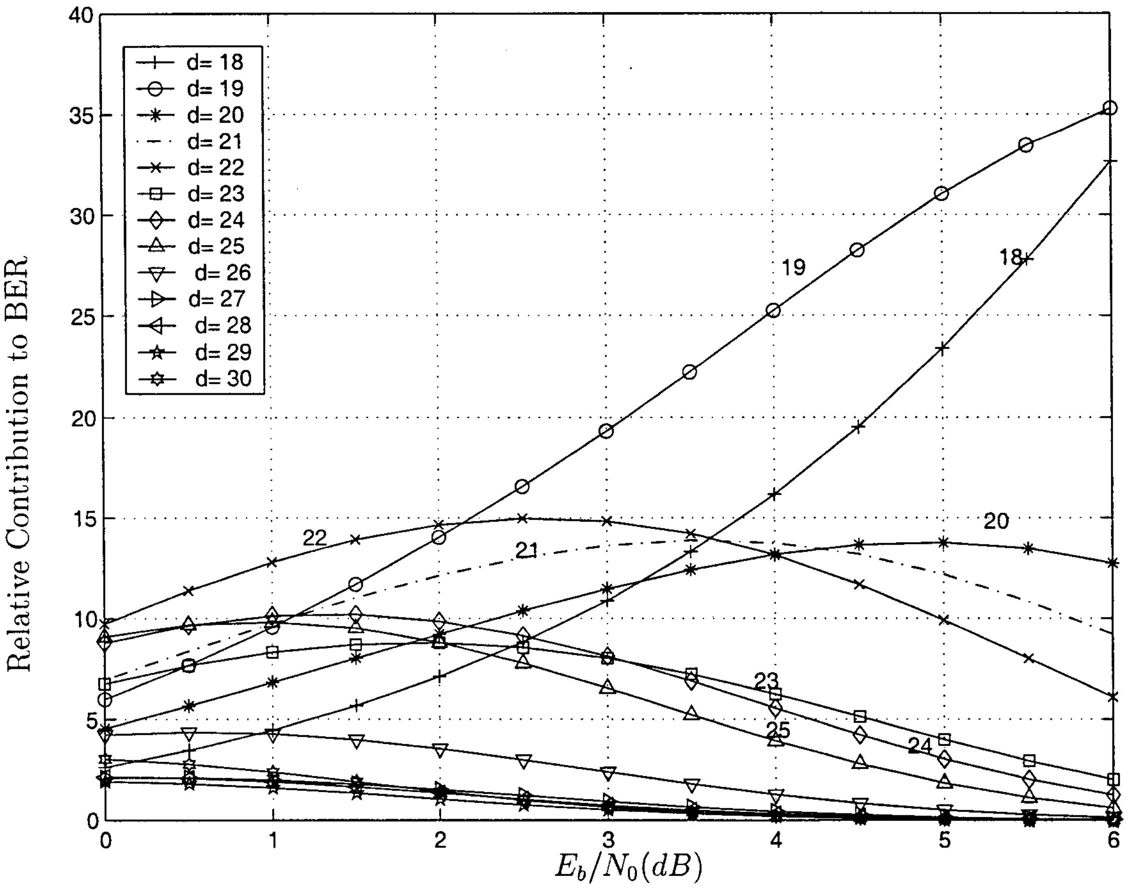


Fig. 5.12: Relative Contribution to Bit Error Rate C_{75}, rate= 1/4, N=256

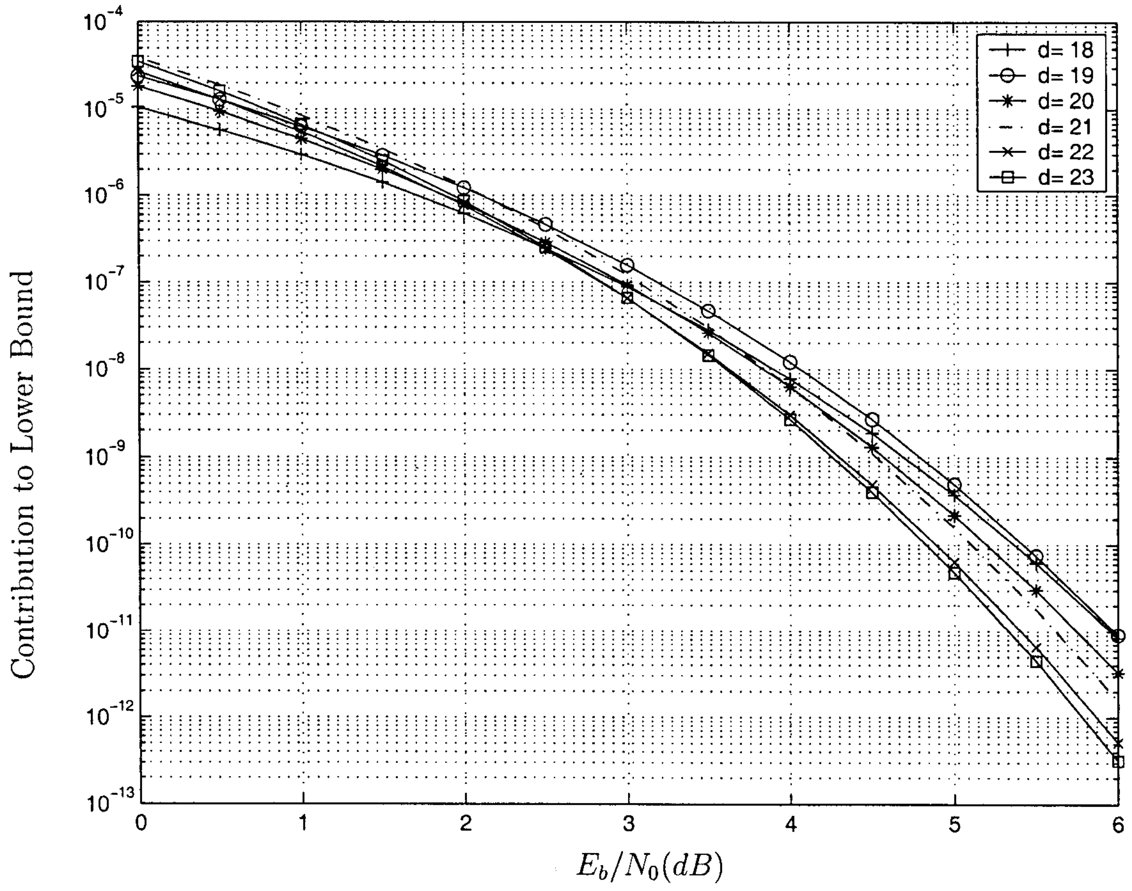


Fig. 5.13: Contribution to Lower Bound for Code C_{75}, rate= 1/4, N=256

as an example throughout the future sections, but these methods can be applied to any other code. As we have seen in the previous chapter, there are two important issue when we design the interleaver. First is the improvement of weight spectrum of the code and second is the improvement in the performance of iterative decoding.

In order to improve the weight spectrum of the code, we have to eliminate the patterns at the input of encoders that generate low-weight codewords. These input patterns are generally the low-weight input patterns that are divisible by the feedback polynomial of the encoder. In case of code C_{57}, the feedback polynomial is $1 + D + D^2$. For eliminating divisible weight-2 and weight-4 input sequences, we will use a code-matched like method, by removing the exact patterns as described in Section 4.5.1. The values for μ (the minimum distance between two "1"s in the weight-2 input pattern) is equal to 3 and z_{min} (the minimum weight of the parity-check sequence generated by a weight-2 input pattern) is equal to 4. Assuming that the allowable d^2_{max} (the minimum weight of the codewords generated by the weight-2 input patterns) is equal to 40 and d^4_{max} (the minimum weight of the codewords generated by the weight-4 input patterns) is equal to 40. We can conclude that the following patterns have to be eliminated by interleavers:

- weight-2 patterns

$$k_1 + k_2 \leq \frac{d^2_{max} - 6}{z_{min} - 2} = \frac{40 - 6}{4 - 2} = 17$$

$$|\pi(i_1) - \pi(i_2)| \bmod 3 \neq 0 \textit{ whenever } |i_1 - i_2| \bmod 3 = 0 \textit{ and } k_1 + k_2 \leq 17$$

- weight-4 patterns

$$k'_1 + k'_2 + k'_3 + k'_4 \leq \frac{d^4_{max} - 12}{z_{min} - 2} = \frac{40 - 12}{4 - 2} = 14$$

$$|\pi(i_1) - \pi(i_3)| \bmod 3 \neq 0 \; and \; |\pi(i_2) - \pi(i_4)| \bmod 3 \neq 0$$
$$whenever |i_1 - i_2| \bmod 3 = 0 \; and \; |i_3 - i_4| \bmod 3 = 0$$

$$|\pi(i_1) - \pi(i_4)| \bmod 3 \neq 0 \; and \; |\pi(i_2) - \pi(i_3)| \bmod 3 \neq 0$$
$$whenever |i_1 - i_2| \bmod 3 = 0 \; and \; |i_3 - i_4| \bmod 3 = 0$$

For other low-weight input sequences finding the exact divisible patterns is a task with overwhelming complexity. In this case, we rely on the ability of the S-random interleaver to break low-weight divisible input sequences. For interleaver size $N = 256$, we use S-random interleaver with S parameter equal to $S = \sqrt{N} = \sqrt{256} = 16$. So, the interleaver must have the condition of:

$$|\pi(i_1) - \pi(i_2)| \geq 16 \quad whenever \quad |i_1 - i_2| \leq 16$$

In order to improve the iterative decoding performance as we saw in Section 4.1.2, the aim is to make the extrinsic information as uncorrelated as possible. We use a suboptimal algorithm to minimize the correlation coefficients [27]. In this algorithm we ensure that

$$(|\pi(i) - \pi(j)| + |i - j|) \geq IDS \text{ for all defined i, j}$$

Where Iterative Decoding Suitability (IDS) is a design parameter, this will upper bound the correlation coefficients and ensure that this term is always less than a certain value depending on the value of IDS. For our simulations, we have used IDS=30, which means

$$(|\pi(i) - \pi(j)| + |i - j|) \geq 30 \text{ for all defined i,j}$$

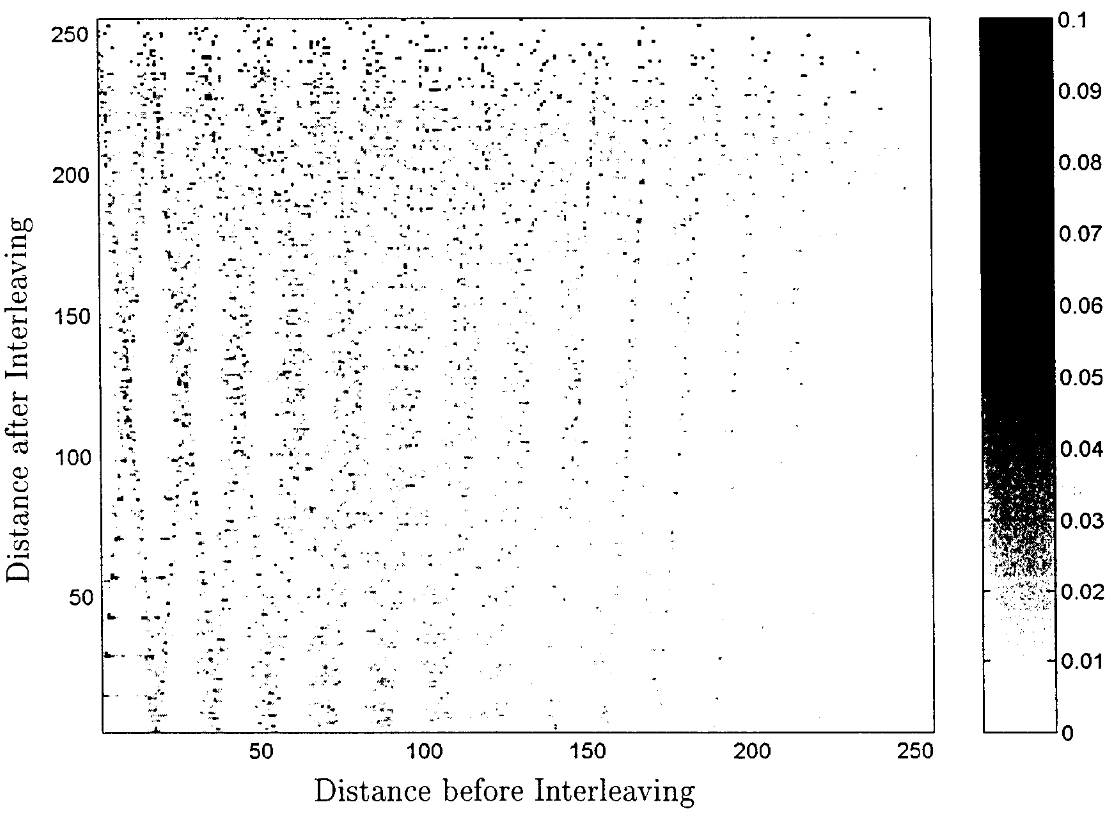


Fig. 5.14: Scatter Plot for S-random interleaver S=14, Max. distance=256

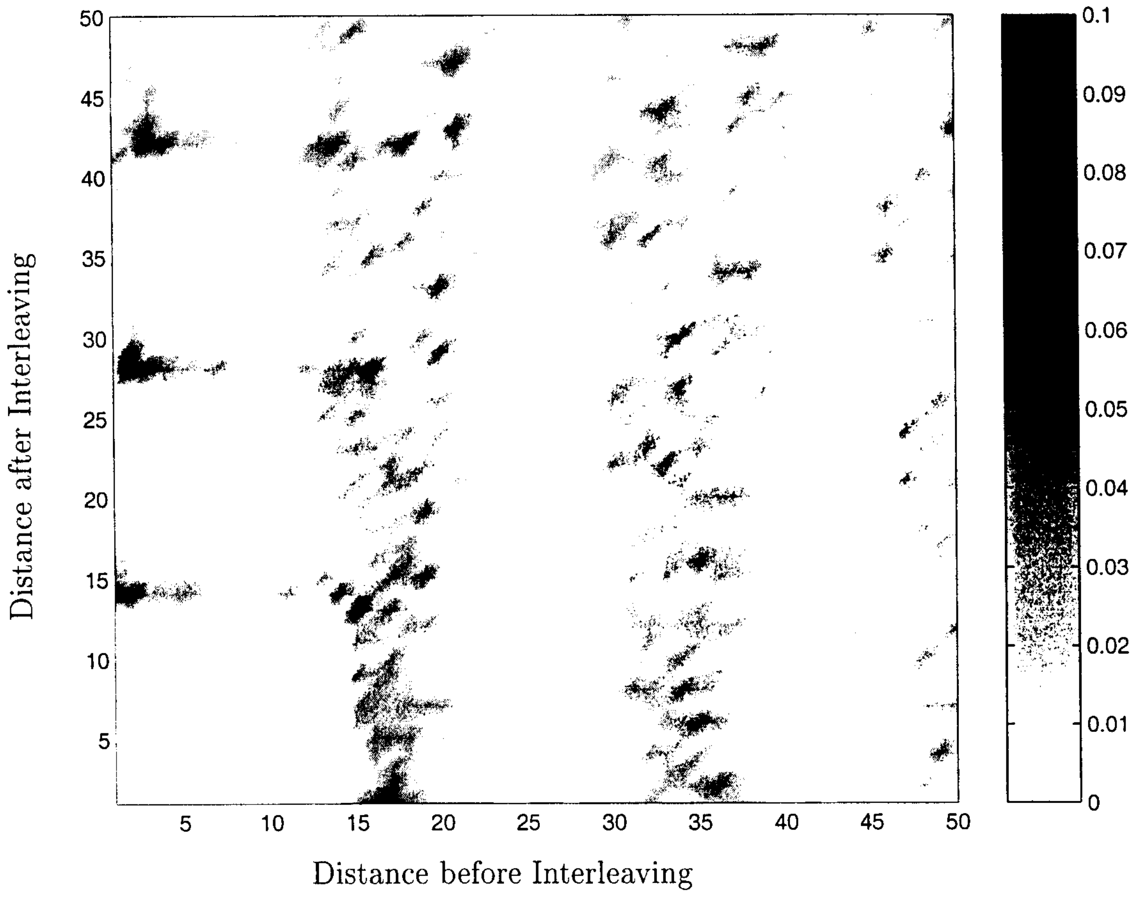


Fig. 5.15: Scatter Plot for S-random interleaver S=14, Max. distance=50

Scatter plots are used to show the spreading factor for interleavers [37]. The scatter plot for code C_{57} with $N = 256$ and S-random interleaver with $S = 14$, are shown in Fig. 5.14 and Fig. 5.15. The x-coordinate shows the distance between elements before interleaving ($|i - j|$), the y-coordinate shows the distance after interleaving ($|\pi(i) - \pi(j)|$) and the shade of each point shows the density at each coordinate. It can be seen that for S-random interleaver, there is no element in the area of ($x < 14$ and $y < 14$), which agrees with the condition of S-random interleaver Fig. 5.15. However, as it can be seen in Fig. 5.14 the density of distances is not uniformly distributed.

5.3 An Algorithm for Interleaver Design

In order to design an interleaver, we need a search program that finds an interleaver that satisfies the design criteria. Design criteria for each interleaver depends on the code and the desired performance. We also need to add some extra conditions for joint design of two interleavers as will be discussed in the next section. Considering an interleaver of size 256, for such interleaver there are 256! combinations. It is impossible to search all the possible solutions in order to find the interleaver that satisfy the design criteria. Usually to resolve this problem, we use random search and it proves to be a good solution.

In this section, we introduce the search algorithm used for designing interleavers in the following sections. The structure of the search will remain the same, but for different methods, the conditions applied to the interleaver will change. The interleaver matrix is presented by $z[i]$, which means that element $z[i]$ before interleaving, will be placed at position i, after interleaving. In this algorithm we assume that all different criteria are checked in one function, which will check whether $z[i]$ (element put in position i), satisfy all the conditions with the $i - 1$ elements placed before. The algorithm is summarized in page 96.

```
z [ i ] = a random number
check condition for z [ i ]

while ( ( condition not satisfied ) and ( not reached end of sequence ) )
        {  z [ i ] = z [ i ] + 1
        check condition for z [ i ]    }

if ( reached end of the sequence )
        {  z [ i ] = 1  }

while ( ( condition not satisfied ) and ( not all the numbers chosen once ) )
        {  z [ i ] = z [ i ] + 1
        check condition for z [ i ]  }

if ( all the numbers chosen and the condition is not satisfied )
        {  z [ i ] = 1
        while ( z [ i ] < N )
                {  j=1
                while ( j <= i-1 )
                        {  swap z [ i ] and z [ j ]
                          check condition for z [ i ]
                          check condition for z [ j ]
                          if ( condition not satisfied )
                                {  swap z [ i ] and z [ j ]
                                j++  }
                        }
                }
        }
```

5.4 Joint Interleaver Design for Multiple Turbo Codes

In the previous chapters, we discussed the interleaver design criteria that improve the distance spectrum of the code and also the iterative decoding suitability of the code. However, it is not always possible to fulfill all of these conditions in a reasonable amount of time. Especially for short length interleavers, the convergence of interleaver is harder and usually we have to loosen the required conditions, to allow the interleaver to converge in reasonable time.

In the case of Multiple turbo codes where we have at least two interleavers to design, we can benefit from this fact and design each interleaver to satisfy part of the conditions and the other interleaver to satisfy the rest. This will decrease the burden on one interleaver to satisfy all the conditions at the same time and enables us to satisfy tougher conditions.

There are two possible approaches. In the first approach, before designing the interleaver, we determine which conditions are supposed to be satisfied by each interleaver. This method enables us to design interleavers in parallel, because each interleaver is assigned a certain patterns that need to be eliminated, which does not overlap with the patterns assigned to the other interleaver. We call this method *Parallel joint interleaver design method.* This method will be discussed in detail in Section 5.5.

In the second approach, we first design an interleaver trying to satisfy as many conditions as it can, without loosening the conditions and ignore the conditions we cannot fulfill. Second interleaver has the duty of satisfying the conditions that the first interleaver was unsuccessful to achieve. We call this method *Sequential joint interleaver design method* and will discuss it in Section 5.6.

5.5 Parallel Joint Interleaver Design Method

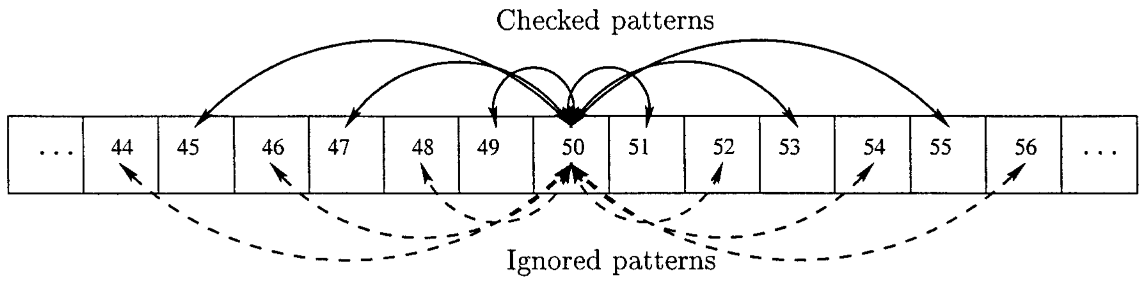
In parallel design method for joint interleavers, design of two interleavers is done separately. This separation requires that a *dividing criterion* be defined, i.e., to decide a priori on the patterns that each interleaver will check. Each pattern is checked by one of the interleavers. Dividing criterion will specify which patterns are to be checked by a given interleaver and which patterns will be ignored by that interleaver. In the following section, we discuss the characteristics that a dividing criterion must have and we will see the effect of different dividing criteria on the interleaver convergence.

5.5.1 Dividing Criterion

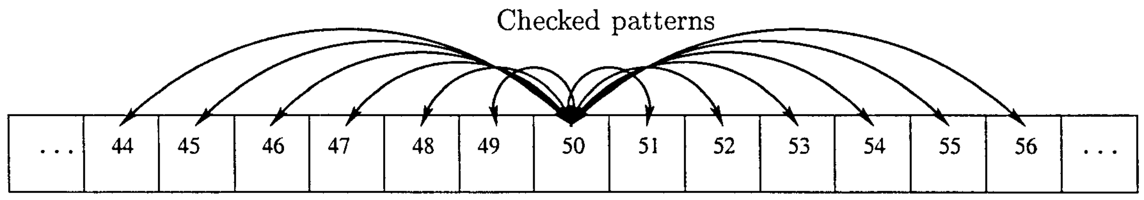
There are certain characteristics a dividing criterion must have in order to be able to improve the convergence of the interleaver. It is obvious that the patterns checked and eliminated by each interleaver must not overlap with the patterns checked by other interleaver and in total all patterns that need to be eliminated must be covered. In the other words, the set of target patterns should be partitioned mutually exclusive and exhaustive subsets and each subset be assigned to one interleaver.

Another preference considering the dividing criterion, rises from the practical limits we face while designing the interleaver. Due to the large number of possible combinations for even a short length interleavers, we usually use a random search. Due to the characteristic of a random search, the dividing criterion must be defined in a way that facilitate the convergence of the interleaver.

In our case, we have to design two interleavers. The elements can be divided based on whether their position in the original sequence is odd or even. One advantage of division based on being odd or even is that the elements are distributed uniformly through the sequence. Following, we will discuss three possible dividing criteria and examine their effect on the interleaver convergence.



(a) First interleaver



(b) Second Interleaver

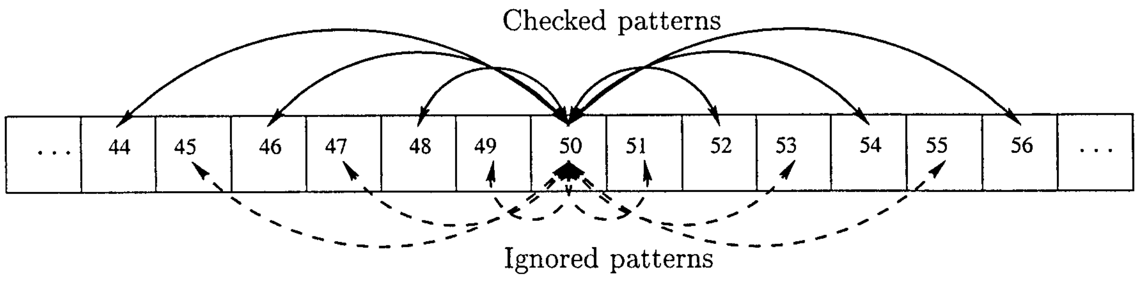
Fig. 5.16: Checked and Ignored Patterns for the First Dividing Criterion Based on the Position in the Original Sequence

5.5.1.1 First Dividing Criterion

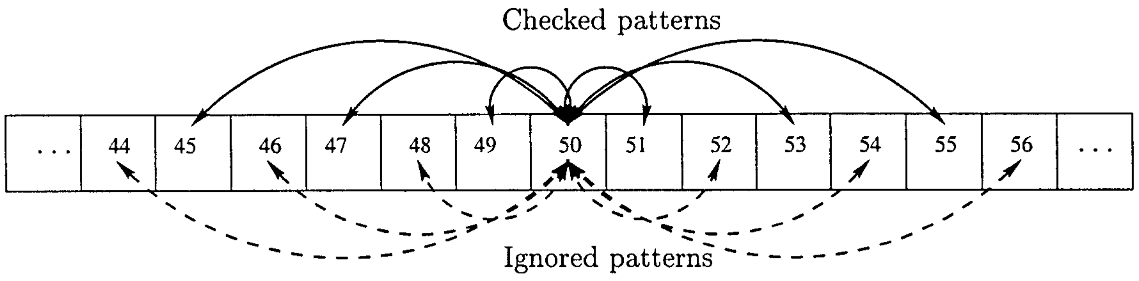
In the case that we design two interleavers without considering their joint design, in each interleaver all of the conditions will be checked and must be satisfied. One step forward, to ease this condition, is to let one interleaver ignore some specific patterns, but there still remain some overlap between the patterns that both interleavers check. In the first dividing criterion, one interleaver ignores the patterns with elements whose positions in the original sequence are all even numbers and the other interleaver ignores the patterns with all odd-positioned elements. Based on the original sequence, we can determine which patterns will be eliminated by each interleaver. For example consider an element positioned at position 50, for such element in the first interleaver as shown in Fig. 5.16(a), the patterns with element 49 and 53 (odd and even) will be checked, but patterns between the element 50 and 48 (even and even) will be ignored. In the second interleaver, as shown in Fig. 5.16(b), for element positioned at place 50 all the patterns will be checked.

It is obvious that this dividing criterion, causes an overlap between patterns assigned to two interleavers. This will reduce the capability of joint interleaver and we will not be able to gain much over separate design of interleavers. However, for this criterion, both interleavers converge with similar rate.

5.5.1.2 Second Dividing Criterion



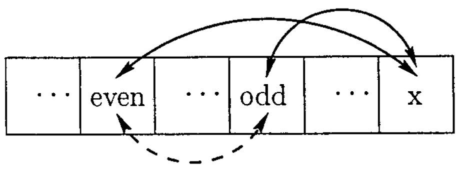
(a) First interleaver



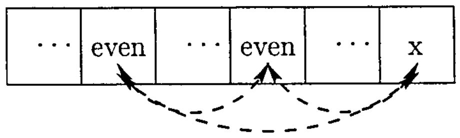
(b) Second Interleaver

Fig. 5.17: Checked and Ignored Patterns for Second Dividing Criterion Based on the Position in the Original Sequence

In order to solve the overlapping problem of the first dividing criterion, we introduce another criterion which does not contain any overlapping patterns. In the second dividing criterion, the first interleaver will check the patterns between the elements that the sum of their positions in the original sequence is even and the second interleaver will check the patterns between the elements that the sum of their positions in the original sequence is odd. For example, consider an element positioned at position 50, for such element in the first interleaver, as shown in



(a) First interleaver



(b) Second Interleaver

Fig. 5.18: Convergence Comparison between Two Interleavers

Fig. 5.17(a), the patterns with element 48 (sum equal even) will be checked, but patterns between the element 50 and 49 (sum equal odd) will be ignored. In the second interleaver, as shown in Fig. 5.17(b), the patterns with element 51 and 47 (sum equal odd) will be checked, but patterns between the element 50 and 48 (sum equal even) will be ignored.

This dividing criterion, does not cause overlap between the patterns checked by two interleavers and in total it covers all the patterns that need to be checked. In practice, when we execute random search for finding the interleavers, we observe that the interleaver with dividing criterion ignoring the elements with odd sum converges much slower than the interleaver ignores the elements with an even sum.

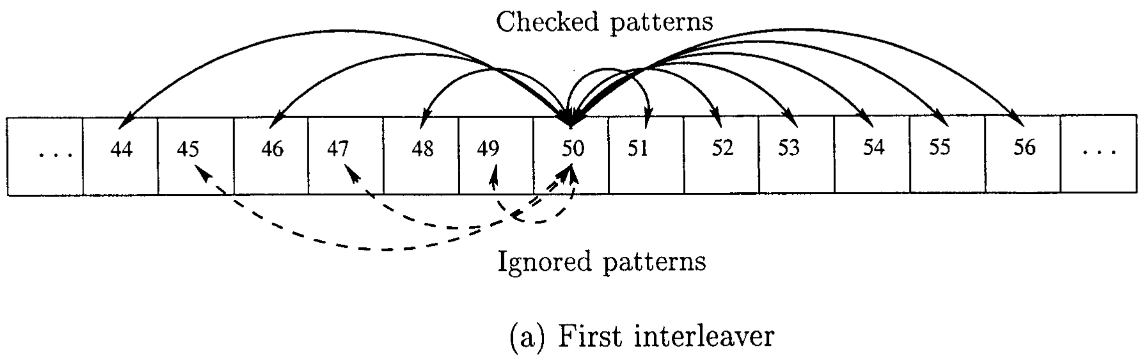
This behaviour can be explain by comparing the result of continuous ignoring in both interleavers. As shown in Fig. 5.18(a), consider a case that an even number is placed and another odd number is placed with ignoring the conditions between the even and odd number, which is acceptable because the sum of them will be odd. In this case, these two numbers can block any even and odd number in the neighbourhood of their original positions to be ignored. This happens because for odd numbers the conditions with odd number placed before must be satisfied and for even numbers the conditions with the even number placed must be satisfied. In other words, ignoring will obstruct the ignoring of other numbers. In the case of other interleaver, if we have placed two even numbers by ignoring the conditions between them (sum equal even), we can still ignore another even number in the neighbourhood, which means ignoring does not prevent us from ignoring more Fig. 5.18(b).

5.5.1.3 Third Dividing Criterion

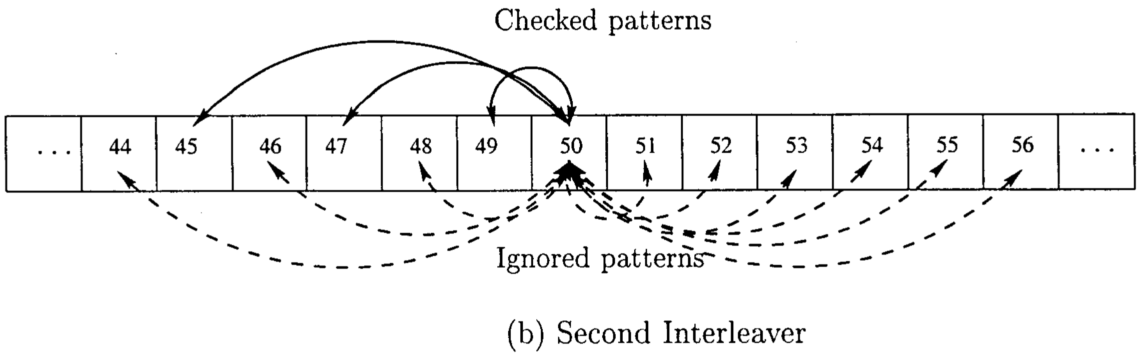
In the third dividing criterion, the first interleaver will check the patterns between the elements that the smaller element in the original sequence is even. The second interleaver will check the patterns between the elements that the smaller element in the original sequence is odd. For example, consider an element positioned at position 50, for such element in the first interleaver as shown in Fig. 5.19(a), the patterns with element 48, 51 and 52 will be checked, but patterns between the element 50 and 49 will be ignored. In the second interleaver, as shown in Fig. 5.19(b), the patterns with element 50 and 49 will be checked, but patterns between the element 50 and 48, 51 and 52 will be ignored. This dividing criterion, does not cause overlapping between patterns checked by two interleavers and in total it covers all the patterns that need to be checked and the two interleavers converge at the same rate.

5.5.2 Convergence Analysis of Parallel Joint Interleaver Design Method

In order to examine the convergence of different dividing criteria, we will consider four different sets of design criteria and then examine the convergence for each dividing criterion. The system under consideration is the code C_{57} as discussed in earlier sections with length $N = 256$. The sets are called V_i and presented in Table 5.1. As it can be seen, V_1 is the easiest and V_4 is the hardest to satisfy. Considering the



(a) First interleaver



(b) Second Interleaver

Fig. 5.19: Checked and Ignored Patterns for Third Dividing Criterion Based on the Position in the Original Sequence

above design criteria, we have designed interleavers satisfying these conditions with the described three dividing criteria. Depending on the the design criterion and the dividing criterion, the interleavers converge to some extent. For all design criteria and dividing criteria, the maximum percentage of elements which satisfy the design criteria, is shown in Fig. 5.20, for each possible combination. The first and second interleaver convergence is examined, due to the fact that some of the dividing criteria do not behave symmetrically. As it can be seen from Fig. 5.20, second dividing criterion has the best convergence performance, but as discussed earlier it does not

Table 5.1: Examined Design Criteria

	V_1	V_2	V_3	V_4
S	13	14	15	16
IDS	15	20	25	30
d^2_{max}	25	30	35	40
d^4_{max}	25	30	35	40

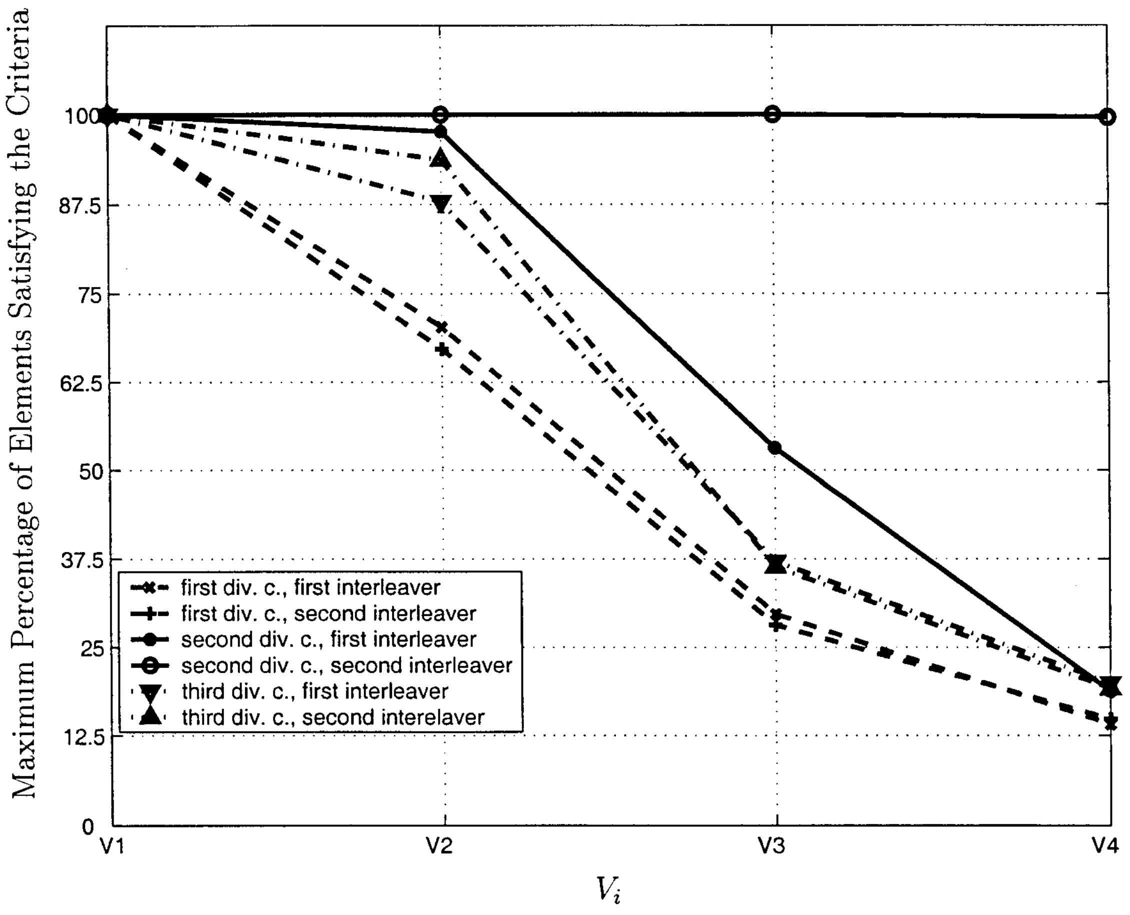


Fig. 5.20: Convergence for Different Dividing Critera

behave symmetrically. This means that one of the interleavers converges much faster than the other. The other two dividing criteria are symmetric and the third dividing criterion has better convergence than the first one. Considering the convergence analysis done in this section, we can conclude that although some dividing criterion can perform better than others but none of them can improve the performance as desired. This will lead us to *Sequential Joint Interleaver Design Method*, where the patterns ignored by each interleaver will be decided while the interleaver is being designed and as we will see later the convergence is much better than *Parallel Joint Interleaver Design Method.*

5.6 Sequential Joint Design Method

In the previous section, we saw that although different dividing criterion can be employed for joint interleaver design, there are two major obstacles in using dividing criterion, first the number of iterations needed for interleaver to converge is extremely sensitive to the choice of dividing criterion and second, some dividing criteria can be unsymmetrical, which results in nonuniform rate of convergence between interleavers. This discussion lead us to *Sequential joint interleaver design*, where the patterns ignored by each interleaver is decided during the design of each interleaver.

Due to the fact that the patterns ignored by each interleaver is not known initially, interleavers cannot be designed in parallel. The first interleaver will be designed first, trying to satisfy all the conditions and ignore the patterns when there is no other way. The second interleave has to check only for the patterns that the first interleaver has ignored.

For each element, we check different condition, e.g., weight-2 patterns, S-random, etc An element may satisfy some of the conditions and fail others (e.g., it may satisfy weight-2 condition but there maybe an element with which it does not satisfy the S-random condition). In such cases, we can choose two methodologies. In the first methodology, in the first interleaver we ignore 50% of the elements, so in the first interleaver when we come across such cases, we ignore the element and in the second interleaver the patterns that have been ignored will be checked. Employing this methodology, we observe that although we intended to ignore 50% of the elements in the first interleaver and check the rest in the second interleaver, the second interleaver converges much faster than the first interleaver and also the first interleaver is not as powerful as we desired. This phenomenon is related to the fact that for each element ignored in the first interleaver only some of the conditions have not been satisfied so the role of the second interleaver is to satisfy only those conditions. So the load on the second interleaver is much less than the first one. In the second methodology, instead of ignoring 50% of the elements we ignore 50% of each set of conditions. This means that 50% of weight-2 patterns, 50% of weight-4 patterns, and etc. are ignored. This will divide the conditions between two interleavers more uniformly and both interleavers will have the same rate of convergence and will perform much better.

In order to have a better rate of convergence, in practice, we need that the patterns ignored be divided uniformly through the interleaver. For weight-2, weight-4 and S-random conditions if we ignore the edge effect, the number of patterns checked for each element is constant so the number of allowable ignored conditions must be proportional to the length of the interleaver up to that point. For the iterative decoding suitability condition, further we proceed, further increase the number of conditions per element will increase because each element will be checked with all elements placed up to this point. So, we have to find the number of conditions up to this point which at place i is equal to $\sum_{k=1}^{i-1} = \frac{k(k-1)}{2}$ and the number of allowable ignored conditions is 50% of this number.

Considering the above discussion, in the next section we will present the algorithm for designing the first and second interleavers using joint interleaver design method.

5.6.1 An Algorithm for Sequential Joint Interleaver Design

In this section, based on the points discussed in the previous section, we introduce the algorithm for joint interleaver design. For design of the first interleaver, the main search algorithm is the one presented in Section 5.3. After the main search fails in finding an element to be placed in position i, we will check whether for this place we can find an element that we can ignore and do not exceed the 50% rule. The algorithm to check this is as follows:

```
z [ $i$ ] = a random number
check weight-2 condition for z [ $i$ ]
```

```
if (z [ $i$ ] not satisfy the weight-2 condition)
        {  if (number-weight2-ignored < max-weight2-patterns $\cdot 0.5 \cdot$ i/N)
                {  ignore this pattern
                   increase number-weight2-ignored    }  }

check weight-4 condition for z [ $i$ ]
if (z [ $i$ ] not satisfy the weight-4 condition)
        {  if (number-weight4-ignored < max-weight4-patterns$\cdot 0.5 \cdot$ i/N)
                {  ignore this pattern
                   increase number-weight4-ignored    }  }

check S-random condition for z[ $i$ ]
if (z [ $i$ ] not satisfy the S-random condition)
        {  if (number-Srandom-ignored < max-Srandom-patterns$\cdot 0.5 \cdot$ i/N)
                {  ignore this pattern
                   increase number-Srandom-ignored    }  }

check IDS condition for z [ $i$ ]
if (z [ $i$ ] not satisfy the IDS condition)
        {  if (number-IDS-ignored < $(\frac{i \cdot i-1}{2}) \cdot 0.5$ )
                {  ignore this pattern
                   increase number-IDS-ignored    }  }
```

This algorithm ensures that the ignored patterns are spread uniformly through the interleaver and also ensures that only 50% of each condition is ignored by the first interleaver.[4]

[4] the 50% ignored ratio, can be changed. One may prefer to ignore only 40% of patterns in the first interleaver to ensure that the second one will converge

The second interleaver has two tasks to do. For each element, it must check whether there are any pattern ignored by the first interleaver and if the answer is yes, it has the task to satisfy that condition for these elements. If the pattern has not been ignored in the first interleaver we will not check it in the second interleaver. Again we will use the main search algorithm, the only difference is in the part when we check conditions for each element. The algorithm for the part that checks the conditions is as following:

```
if ( pattern including z [ $i$ ] do not satisfy the weight-2 condition)
    {  if ( this pattern is ignored in the first interleaver)
        {  check the pattern  }
      else
        {  ignore the pattern  }  }
if ( pattern including z [ $i$ ] do not satisfy the weight-4 condition)
    {  if ( this pattern is ignored in the first interleaver)
        {  check the pattern  }
      else
        {  ignore the pattern  }  }
if ( pattern including z [ $i$ ] do not satisfy the S-random condition)
    {  if ( this pattern is ignored in the first interleaver)
        {  check the pattern  }
      else
        {  ignore the pattern  }  }
if ( pattern including z [ $i$ ] do not satisfy the IDS condition)
    {  if ( this pattern is ignored in the first interleaver)
        {  check the pattern  }
      else
        {  ignore the pattern  }  }
```

5.6.2 Performance Analysis of Sequential Joint Deign Method

In this section, we examine the convergence and performance of *Sequential joint interleaver design method.* In order to be able to compare the results, we will use the C_{57} code with interleaver of size $N = 256$ and rate=1/4. Interleaver design criteria used are $d^2_{max} = 40$, $d^4_{max} = 40$, $S = 16$and $IDS = 30$ for these design criteria, both interleavers converge with the same rate and converge in a few iterations.

For this system, we also find the relative contribution of each spectral line to bit error rate , shown in Fig. 5.21. Comparing these results with the one of conventional S-random interleaver design for multiple turbo code in Fig. 5.10, we can see that the minimum weight of the code is increased considerably. This will considerably improve the performance of turbo code at error-floor region. In Fig. 5.22, contribution of each spectral line to lower bound of probability of bit error is shown. Comparing the results with case of conventional interleaver design shown in Fig. 5.11, it can be observed that the error-floor is improved by a factor of 10^{-3}.

Scatter plot can also be drown for interleavers resulted form sequential joint interleaver design method. This is shown in Fig. 5.23 and Fig. 5.24. It can be seen that no element is present in the lower corner triangle, which is resulted from S-random interleaver condition with $S = 16$ and iterative decoding suitability condition with $IDS = 30$. It can also be seen that the distances is spread more uniform in this case that will result in weaker correlation between elements.

5.7 Summary

We have introduced two methods for joint interleaver design for multiple turbo codes. In the first approach, *Parallel joint interleaver design method*, two interleavers were designed in parallel. In order to do so, we need a *dividing criterion*, which determines which patterns will be ignored (or checked) by each interleaver. We have proposed three different dividing criteria. We have seen that the convergence of the interleaver

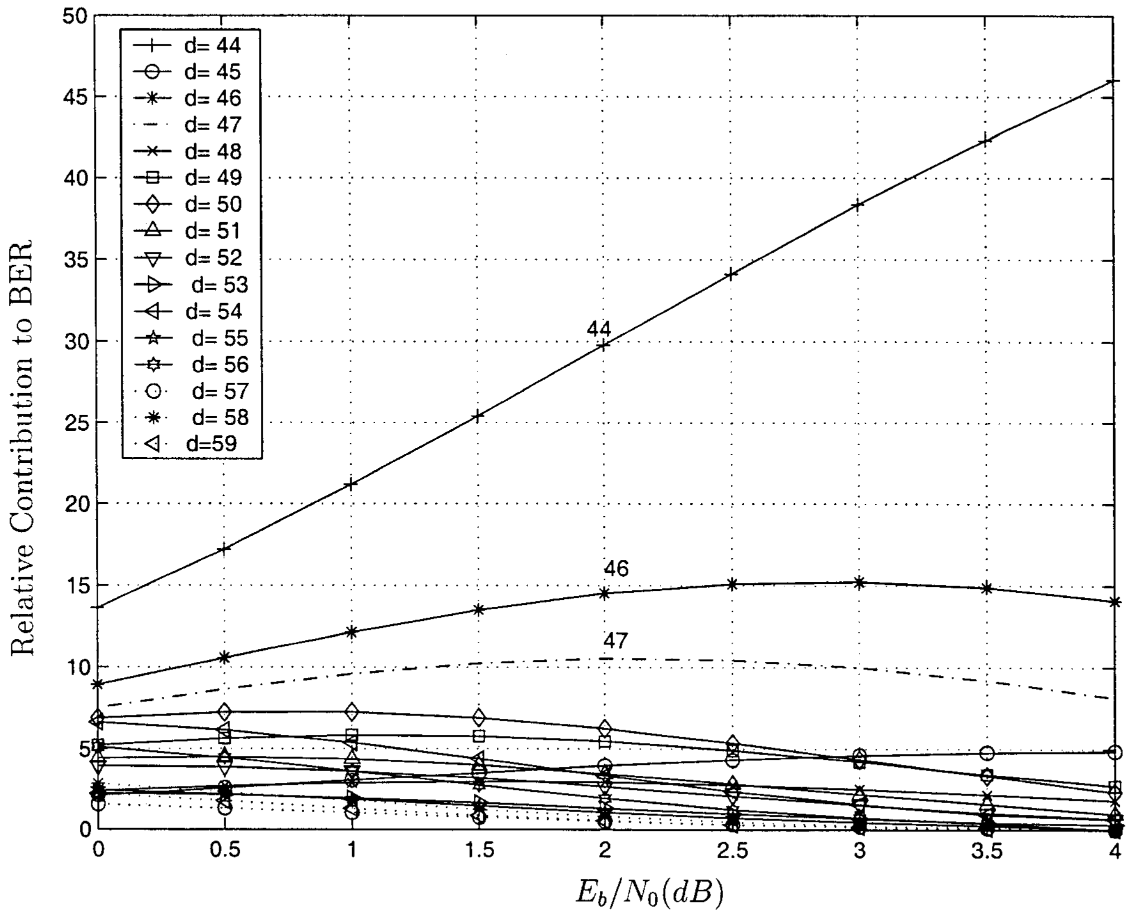


Fig. 5.21: Relative Contribution to Bit Error Rate code C_{57}, rate= 1/4, N=256

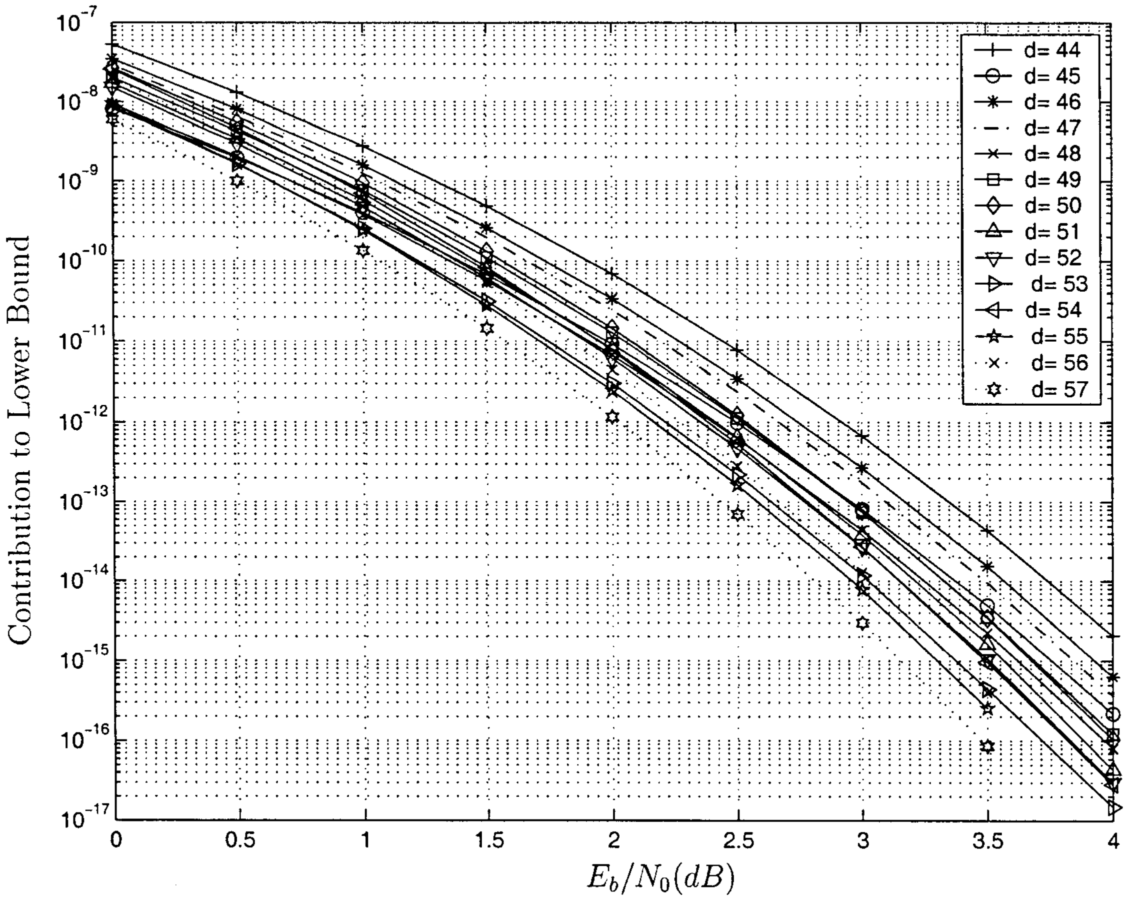


Fig. 5.22: Contribution to Lower Bound for Code C_{57}, rate= 1/4, N=256

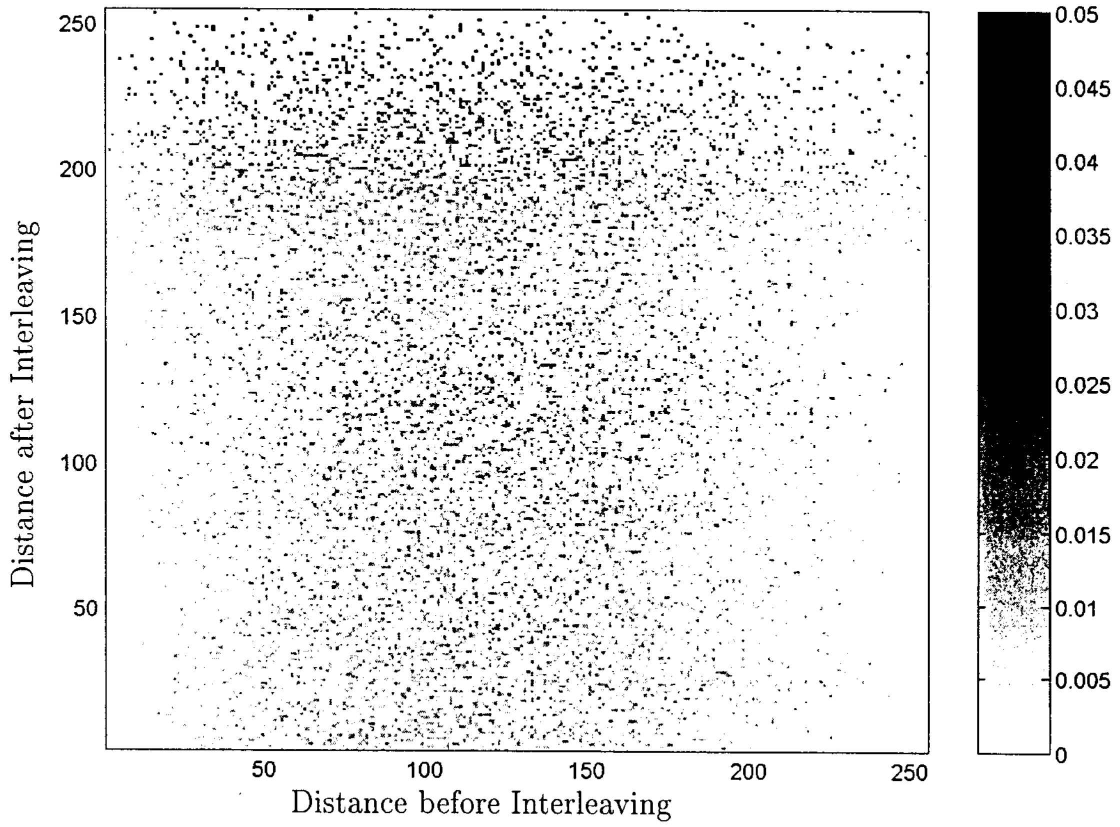


Fig. 5.23: Scatter Plot for Designed interleaver Max. distance =256

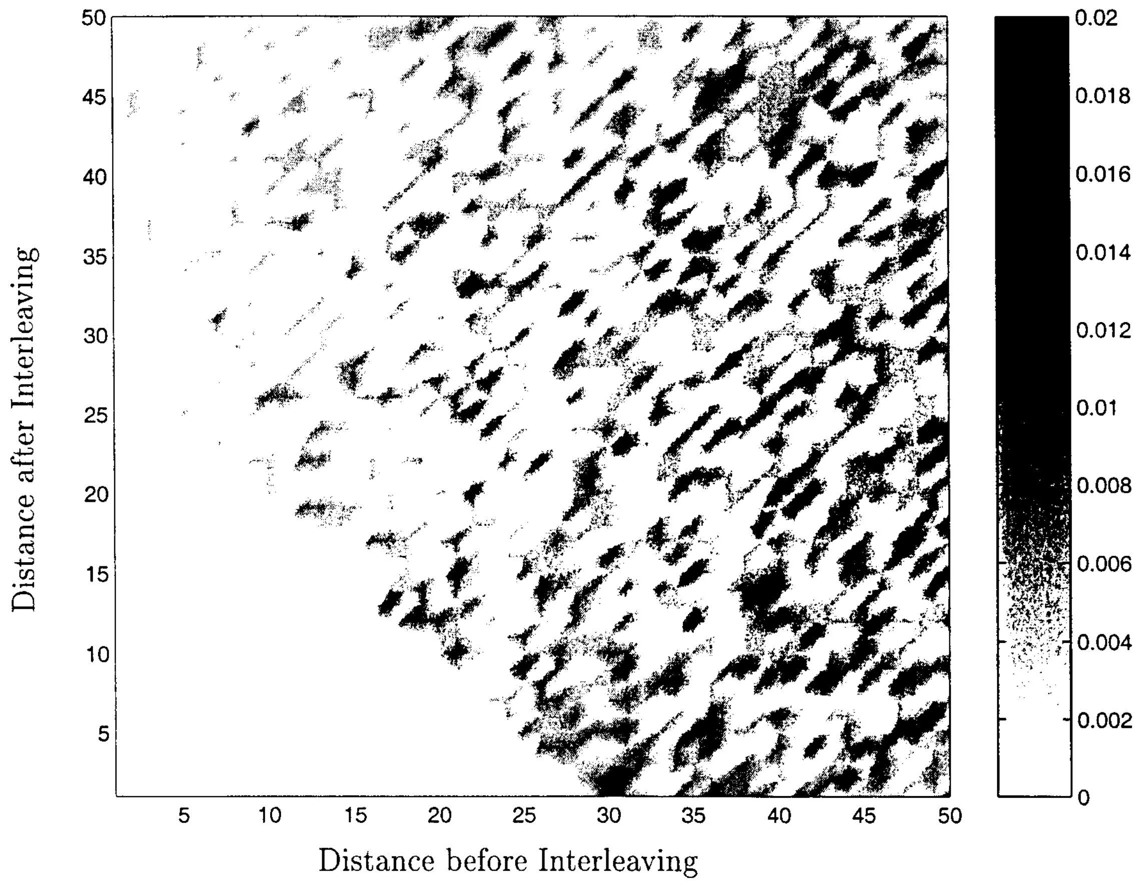


Fig. 5.24: Scatter Plot for Designed interleaver Max. distance =50

is extremely sensitive to the definition of the dividing criterion and for some dividing criteria, interleavers converge non symmetrically. This lead us to the conclusion that the concept of dividing patterns before designing interleaver, will slow down the convergence of the interleaver.

In the other method, *Sequential joint interleaver design method*, we design interleavers sequentially. The first interleaver will be designed trying to satisfy all the design criteria, we only ignore a pattern if there is no other way. the second interleaver has the duty of fulfilling all the conditions that the first interleaver has failed to reach. In order to have the better convergence, we must take care that the patterns ignored be distributed uniformly through the interleavers and also no more than 50% of conditions be ignored by first interleaver. Sequential joint interleaver design method converges very well and can satisfy very tough design criteria.

Chapter 6

Conclusion

The outstanding performance of turbo codes at low SNR is deteriorated in the *error-floor* region, due to the low values of free distance of turbo codes. In order to improve the minimum distance of turbo codes, we need to eliminate the low-weight input sequences that generate low-weight codewords, in other words, we have to break these low-weight input sequences.

One of the tasks of interleaver design for turbo codes is to improve the minimum distance of the code, by eliminating low-weight input sequences that result in low-weight code words. The other task of the interleaver is related to the fact that turbo codes are decoded using sub-optimal iterative decoding methods, where the performance is greatly dependant on the quality of extrinsic information exchanged between decoders. Interleaver has the task of decorrelating the extrinsic information as much as possible.

Although interleaver design criteria that improve the distance spectrum of the code and also the iterative decoding suitability of the code are known, we are not able to find an interleaver that fulfills all of these conditions in the reasonable time. Especially for short length interleavers, the convergence of interleaver is harder and usually we have to loosen the necessary conditions, so that the interleaver can converge in reasonable time. This, in turn degrades the performance.

In the case of *Multiple Turbo Codes*, where we have three or more constituent codes and we have at least two interleavers, each interleaver can be designed to satisfy some of the conditions and other conditions are satisfied by the other interleavers. This will decrease the load on one interleaver to satisfy all the conditions at the same time and enable us to satisfy tougher conditions.

We have introduced two methods for joint interleaver design for multiple turbo codes. In the first approach, *Parallel joint interleaver design method*, two interleavers were designed in parallel. In order to do so, we need a *dividing criterion*, which will determine which patterns will be ignored (or checked) by each interleaver. We have considered three different dividing criterion. We have seen that the convergence of the interleaver has strong dependence on the definition of dividing criterion and for some dividing criteria interleavers converge non symmetrically. This lead us to the conclusion that the concept of dividing patterns before designing interleaver, will slow the convergence of the interleaver.

In another method, *Sequential joint interleaver design method*, we design interleavers sequentially. The first interleaver will be designed trying to satisfy all the design criteria, we only ignore a pattern if we have no other choice. The second interleaver has the duty of fulfilling all the conditions that the first interleaver has failed to satisfy. In order to have better convergence, we must ensure that the patterns ignored are distributed uniformly through the interleaver and also not more than 50% of conditions be ignored by the first interleaver.

Sequential joint interleaver design method, proves to converge very fast and also can satisfy very tough design criteria. This method can be used to satisfy different design criteria for interleavers. These design criteria can be aimed toward increasing the minimum distance of the code which will improve the performance at error-floor region and also further more can aimed at improving the iterating decoding performance. Simulation results presented prove that this interleaver design method can improve the performance of the multiple turbo codes in the error-floor region considerably.

Although in this thesis we employed this method in context of multiple turbo codes with rate $1/4$, but puncturing methods can be used in order to get desired code rates. Future work, on this thesis can employ these method for designing interleaver for punctured multiple turbo codes, in order to reach higher code rates.

References

[1] C. E. Shannon, "A mathematical theory of communication," Tech. Rep. 23, Bell System Technical Journal, 1948.

[2] S. Benedetto, G. Montorsi, and D.Divsalar, "Concatenated convolutional codes with interleavers," *IEEE Commun. Mag.*, vol. 41, pp. 102–109, Aug. 2003.

[3] J. Hagenaur, E. Offer, and L. Papke, *Reed-Solomon Codes and Their Applications.* New York:IEEE Press, 1994.

[4] J. Lodge, R. Young, P. Hoeher, and J. Hagenauer, "Separable map filters for the decoding of product and concatenated codes," in *Proc. IEEE Int. Conf. on Communications*, pp. 1740–1745, May 1993.

[5] J. Hagenauer and P. Hoeher, "Concatenated viterbi-decoding," in *Proc. Joint Swedish-Soviet Int. Workshop on Information Theory*, pp. 29–33, Aug. 1989.

[6] J. Forney and G. Davi, *Concatenated Codes.* Cambridge, MA: M.I.T. Press, 1966.

[7] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding: Turbo codes," in *Proc. IEEE Int. Conf. Communications*, (Geneva, Italy), pp. 1064–1070, May 1993.

[8] L. R. Bahl, J. Cocke, F. Jelink, and J.Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. 20, pp. 284–287, Mar. 1974.

[9] S. Benedetto and G. Montorsi, "Unveiling turbo codes: Some results on parallel concatenated coding schemes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 409–428, Mar. 1996.

[10] D. Divsalar and F. Pollara, "Multiple turbo codes for deep-space communication," TDA Progress Report 42-121, Jet Propulsion Laboratory, Pasadena, CA, May 1995.

[11] J. Han and O. Y. Takeshita, "On the decoding structure of multiple turbo codes," in *Proc. IEEE Int. Symposium on Information Theory*, p. 98, June 2001.

[12] J. Hagenaur, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 429–445, Mar. 1996.

[13] M. R. Soleymani, Y. Gao, and U. Valaipornsawai, *Turbo Coding for Satellite and Wireless Communication.* Kluwer Academic Publishers, 2002.

[14] S. Benedetto, D. Divsalar, and G. M. F. Pollara, "Soft-output decoding algorithms in iterative decoding of turbo codes," TDA Progress Report 42-124, Jet Propulsion Laboratory, Pasadena, California, Feb. 1996.

[15] D. Divsalar and F.Pollara, "Multiple turbo codes," in *Proc. MILICOM'95*, (SanDiego, CA), Nov. 1995.

[16] S. Benedetto, E. Biglieri, and V. Castellani, *Digital Transmission Theory.* Englewood Cliffs, NJ: Prentice-Hall, 1987.

[17] S. Benedetto, D. Divsalar, and G. M. F. Pollara, "Serial concatenation of interleaved codes: Performance analysis, design, and iterative decoding," *IEEE Trans. Inform. Theory*, vol. 44, pp. 909–926, May 1998.

[18] B. Vucetic and J. Yuan, *Turbo Codes: Principles and Applications.* Kluwer Academic Publishers, 2000.

[19] J. G. Proakis, *Digital Communications.* New-York: McGrawHill, 2 ed., 1989.

[20] W. Ryan, "A turbo code tutorial," 1997. submitted to Globecom 1997.

[21] S. Benedetto, M. Mondin, and G. Montorsi, "Performance evaluation od trellis-coded modulation schemes," *Proc. IEEE*, vol. 82, pp. 833–855, June 1994.

[22] S. Benedetto and G. Montorsi, "Design of parallel concatenated convolutional codes," *IEEE Trans. Commun.*, vol. 44, pp. 591–600, May 1996.

[23] J. Seghers, L. C. Perez, and D. J. Costello, "On selecting code generators for turbo codes," in *Proc. of the 33rd Annual Allerton Conference on Communication, Control and Computing*, pp. 357–361, 1995.

[24] L. C. Perez, J. Seghers, and D. J. Costello, "A distance spectrum interpretation of turbo codes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 1698–1709, Nov. 1996.

[25] M. Z. Wang, A. Sheikh, and F. Qi, "Interleaver design for short turbo codes," in *Proc. IEEE Globecom Conf.*, 1999.

[26] J. Hokfelt, O. Edfors, and T. Maseng, "Turbo codes: Correlated extrinsic information and its impact on iterative decoding performance," in *Proc. IEEE VTC*, (Huston,Texas), May 1999.

[27] J. Hokfelt, O. Edfors, and T. Maseng, "A turbo code interleaver design criterion based on the performance of iterative decoding," *IEEE Commun. Lett.*, vol. 5, pp. 52–54, Feb. 2001.

[28] A. S. Barbulescu and P. S. S, "Interleaver design for turbo codes," *Electron. Lett.*, vol. 30, pp. 2107–2108, Dec. 1994.

[29] D. Divsalar and F.Pollara, "Turbo codes for PCS applications," in *Proc. IEEE Int. Conf. Communications*, June 1995.

[30] J. Yuan, B. Vucetic, and W.Feng, "Combined turbo codes and interleaver design," *IEEE Trans. Commun.*, vol. 47, pp. 484–487, Apr. 1999.

[31] J. Yuan, B. Vucetic, and W.Feng, "A code-matched interlevaer design for turbo codes," *IEEE Trans. Commun.*, vol. 50, pp. 926–937, June 2002.

[32] K. Richardson, "UMTS overview," *Electronics & Communication Engineering Journal*, vol. 12, pp. 93–100, 2000.

[33] European Telecommunications Standards Institute, *Universal Mobile Telecommunication System (UMTS): Multiplexing and Channel Coding (FDD)3GPP TS 125.212 version 4.0.3*, Dec. 2001.

[34] R. P. Stanley, *Enumerative Combinatorics.* Monterey, California: Wadsworth & Brooks/Cole, 1986.

[35] D. Divsalar, F. Pollara, and S. Dolinar, "Transfer function bounds on the performance of turbo codes," TDA Progress Report 42-122, Jet Propulsion Laboratory, Pasadena, California, Aug. 1995.

[36] D.Divsalar, S. Dolinar, and F.Pollara, "Iterative trubo decoder analysis based on density evolution," *IEEE J. Select. Areas Commun.*, vol. 19, pp. 891–907, May 2001.

[37] C. Heegard and S. B. Wicker, *Turbo Coding.* Kluwer Academic Publishers, 1999.