On the design of turbo codes with convolutional interleavers

Sina Vafi
University of Wollongong

On the Design of Turbo Codes with Convolutional Interleavers

A thesis submitted in fulfilment of the
requirements for the award of the degree

Doctor of Philosophy

from

THE UNIVERSITY OF WOLLONGONG

by

Sina Vafi
Master of Engineering

SCHOOL OF ELECTRICAL, COMPUTER
AND TELECOMMUNICATIONS ENGINEERING
2005

To my beloved Parents
and Sisters, Tina and Nikan

Abstract

Random interleavers are amongst the most effective interleavers for turbo codes. However, due to their random permutations, a compact representation of the code specification is a major obstacle. Thus, to date, much research has been conducted on the design of deterministic interleavers having performances close to random interleavers. These interleavers are mainly constructed as block interleavers, which allows the code to be analyzed as a block code.

In contrast to block interleavers, there are non-block interleavers. These utilize a reduced number of memories in their structures and have self-synchronization with their deinterleavers; this simplifies their design. Because of their non-block structures, turbo codes constructed by these interleavers must usually be analyzed in terms of the continuous performance. Previous research confirms that the codes' continuous performance is similar to their block performance, but at the expense of increased complexity of the code analysis and decoding. In order to analyze a turbo code constructed with non-block interleavers as a block code, it is necessary to consider the applied interleavers as block interleavers. This is accomplished by the insertion of stuff bits at the end of each input data block, returning the interleaver memories to zero state.

This thesis is related to the application of convolutional interleavers which are the most popular non-block interleavers for turbo codes. It introduces convolutional interleavers as good deterministic interleavers that can perform similar or even better than previous deterministic and random interleavers. The thesis presents two different structures of block-wise convolutional interleavers, created on the basis of distribution of stuff bits in the interleaved data. On the basis of convolutional interleaver

properties, a simple algorithm is introduced to analyze code performance at different signal to noise ratios. The code analysis is confirmed with simulation results, which allow selection of the most suitable interleaver.

Different models of the selected convolutional interleavers are verified. These models are constructed based on changing the period and space values, which are introduced as the constituent parameters of convolutional interleavers. The performance of interleavers with different periods and a space value 1 are investigated. For a similar number of stuff bits, these interleavers are compared with interleavers constructed with shorter periods and highest fixed space values than 1. Convolutional interleavers with variable space values operating as generalized convolutional interleavers are also presented and their performance is compared with interleavers using the fixed space value.

Turbo codes constituted with the mentioned interleavers are analyzed using different input bitstreams. Based on the analysis, suitable modifications are proposed for each model of interleaver so as to improve the turbo code performance through a reduced number of stuff bits. The performance of the modified convolutional interleavers is compared with good deterministic and random block interleavers. The results demonstrate that with an acceptable number of stuff bits contributed to each interleaved data, convolutional interleavers provide similar or improved performance when compared to block interleavers.

Finally, the application of designed convolutional interleavers in Unequal Error Protection (UEP) turbo codes is presented. Based on the code specifications and interleaver properties, three different techniques for UEP are suggested to improve protection of priority data, while reducing the overall number of stuff bits inserted into the interleaver memories.

Statement of Originality

This is to certify that the work described in this thesis is entirely my own, except where due reference is made in the text.

No work in this thesis has been submitted for a degree to any other university or institution.

Signed

Sina Vafi
December, 2005

Acknowledgments

I would like to express my gratitude and appreciation to my supervisor Associate Professor Tadeusz Wysocki for his patience, support and guidance in every step of my thesis. Thank you for sharing your time and experience giving me an opportunity to expand my knowledge in one of the fundamental telecommunication subjects. I sincerely appreciate my co-supervisor, Associate Professor Ian Burnett for his advice and encouragement and time devoted for this work.

Special thanks to my family, whose valuable help and support encouraged me to do my best. I hope my work, in some way, may repay them for their efforts.

I wish to express my deep appreciation to my friends, Dr. Habibollah Danyali, Dr. Abdollah Chalechale and Dr. Fardin Akhlaghian Tab for infinite discussion and help in all aspects of study and life in Wollongong.

My gratitude to Dr. Madeleine Strong Cincotta and Miss. Kate Hurley for the editing of my published papers and thesis.

I would also like to thank the School of Mathematics and Applied Statistics and ac3 technical support team for their prompt responses to my requests allowing me to get accurate and fast results from high-performance computing facilities.

Finally, my appreciation to Dr.Masoud Reisian, Mr. Ahmad Jalilehvand, Mr. Khousro Saghafi and Mr. Hamed Khousravi for their concern and assistance in the removing of a major obstacle in the initial steps of my study.

Contents

List of Figures

List of Tables

List of Abbreviations

3G	3rd Generation
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
Bi-SOVA	Bidirectional SOVA
BRC	Block-Random Chaotic
BSC	Binary Symmetric Channel
DAB	Digital Audio Broadcasting
DRP	Dithered Relative Prime
DVB	Digital Video Broadcasting
EEP	Equal Error Protection
FEC	Forward Error Correction
FSP	Finite State Permuter
LAN	Local Area Network
LLR	Log-Likelihood Ratio
MAP	Maximum A-Posteriori
ML	Maximum Likelihood
OFDM	Orthogonal Frequency Division Multiplexing
RSC	Recursive Systematic Convolutional
SNR	Signal to Noise Ratio
SOVA	Soft Output Viterbi Algorithm
UEP	Unequal Error Protection
XOR	Exclusive OR

Chapter 1

Introduction

1.1 Background

The main function of a communication system is to transmit information from the source to the destination with sufficient reliability. In the last two decades, there has been an explosion of interest in the transmission of digital information mainly due to its low cost, simplicity, higher reliability and possibility of transmission of many services in digital forms [1].

Figure 1.1 shows a simple block diagram of a digital transmission system. At the source, information suitable for transmission is produced. The input to this block is either analog or discrete. In the case of an analog input, appropriate processes, i.e. quantization, sampling and coding are performed so as to form a discrete signal. Discrete information obtained from the source block a certain sampling rate is then input to the source encoder block. In this block, symbol sequences are converted to binary sequences by assigning codewords to the input symbols according to a specified rule and based on reducing the redundancy of the encoded data. Since redundancy has been removed from the information source, encoded information is sensitive to noise in transmission media. Hence, a channel encoder inserts redundancies into the source encoded data so as to protect the required signal against channel errors. Then, the modulator converts the input binary stream to a waveform compatible with the channel characteristics and provides suitable conditions for transmission

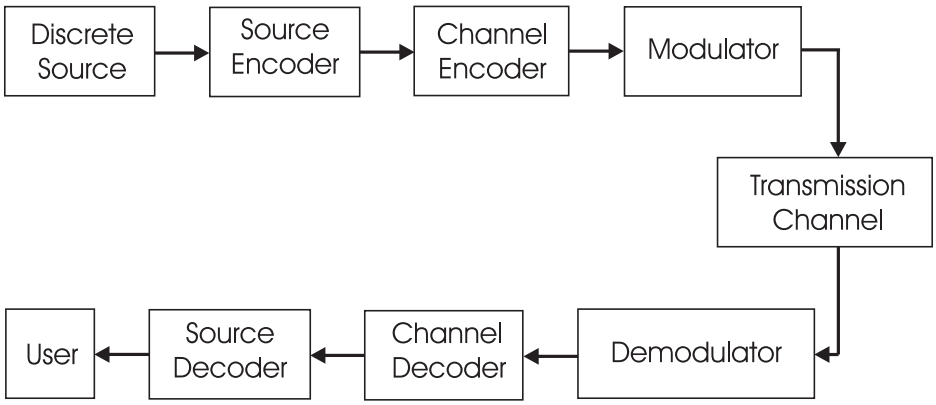

Figure 1.1 Block diagram of digital transmission systems.

of the signal. The remaining blocks of Figure 1.1 perform inverse operations of their corresponding blocks at the transmitter to finalize extraction of the required signal at the destination.

Theoretically, Shannon stated that the maximum rate of transmitted signal or capacity of a channel over Additive White Gaussian Noise (AWGN), with an arbitrarily low bit error rate depends on the Signal to Noise Ratio (SNR) and the bandwidth of the system (W), according to [2]:

$$C = Wlog_2(1 + \frac{S}{N}) \tag{1.1}$$

where C is the capacity of the channel, S and N are the signal and the average noise power, respectively. Based on this theory, it would be possible to transmit information with any rate (R) less than or equal to the channel capacity $(R \leq C)$, when suitable coding is applied. Instead of $\frac{S}{N}$, the channel capacity can be represented based on the signal to noise ratio per information bit ($\frac{E_b}{N_0}$). Considering the relationship between SNR and $\frac{E_b}{N_0}$, and the channel capacity (with value R) , Equation 1.1 can be rewritten as follows:

$$\frac{S}{N} = \frac{E_b}{N_0} \times \frac{R}{W} \tag{1.2}$$

$$\frac{C}{W} = log_2(1 + \frac{E_b}{N_0}.\frac{C}{W}) \tag{1.3}$$

In the case of an infinite channel bandwidth ($W \longrightarrow \infty, \frac{C}{W} \longrightarrow 0$), the Shannon

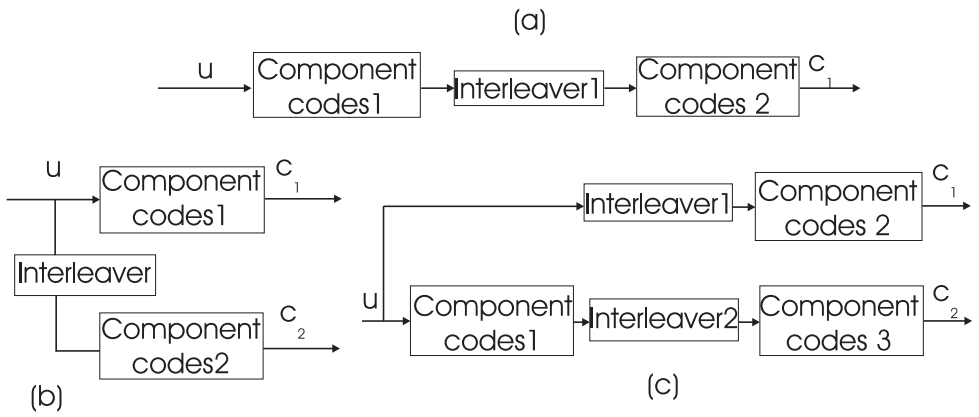

Figure 1.2 Structure of different concatenated codes. a) Serial concatenated codes, b) parallel concatenated codes and c) Hybrid concatenated codes.

bound is defined by:

$$\frac{E_b}{N_0} = \frac{1}{log_2 e} = 0.693 \tag{1.4}$$

In order to achieve this bound, i.e. $\frac{E_b}{N_0} = -1.59$ dB value, it would be necessary to use a code with a such long length that encoding and decoding would be practically impossible. However, the most significant step in obtaining this target, was by Forney, who found that a long code length could be achieved by concatenation of two simple component codes with short lengths linked by an interleaver [3]. Figure 1.2 shows basic structures of the serial, parallel and hybrid concatenated codes. Unlike serial and hybrid concatenated codes, turbo codes, which are basically implemented by parallel concatenation of two similar Recursive Systematic Convolutional (RSC) component codes, create a perfect balance between component codes with the close performance to the Shannon bound [4]. On the basis of these properties, the recent decade saw this type of coding as the subject of much research and its usage in several applications [5–9].

1.2 Choice of the Interleaver

Conventionally, a turbo code is analyzed as a block code by using a block interleaver and terminating RSC encoders to a known state at the end of each data block. Codes with this structure are generally decoded using an iterative decoding technique. De-

pending on the number of iterations, the decoding procedure can approach the Maximum Likelihood decoding. Similarly to linear block codes, the probability of error for turbo codes in an AWGN channel is upper bounded such that [10]:

$$P_b \leq \sum_{d=d_{free}}^{\infty} \sum_{\omega=1}^{L} a(\omega, d) \frac{\omega}{2L} erfc\left(\sqrt{d \frac{2RE_b}{N_0}}\right) \tag{1.5}$$

where R, ω, d, L, $a(\omega, d)$ and d_{free} denote the code rate, the information weight, the codeword weight, the information length, number of codewords with information weight ω and minimum codeword weight as the free distance value, respectively.

In this equation, $a(\omega, d)$ determines multiplicity of the calculated weights [10]. Hence it can be concluded that the codes with higher free distances and low multiplicities have the lower upper error bound, and consequently, better code performance. Analysis indicates that turbo codes with block interleavers often have a relatively low free distance value with high multiplicities which results in insufficient code performance in the medium to high signal to noise ratios. In fact, the error will not decrease proportionally with increments of signal to noise ratio. This phenomenon is called ”error floor”.

One of the most effective solutions to reduce the error floor is utilization of a suitable interleaver compatible with the structure of constituent RSC encoders. In this case, input bit streams, which produce low weights for the first RSC code are permuted by an interleaver in a way that prohibits generation of low weights for the second RSC code to increase free distance value of the turbo code. It has been accepted that the best performance of turbo code is achieved by random interleavers, which randomly permute input bitstreams to different memories of the interleaver [11]. Due to the existence of randomly interleaved data, determining of an adequate analysis to its implementation is a major obstacle. In addition, in order to make synchronization between random interleavers and deinterleavers, which performs the reverse function of interleavers, it is necessary to store interleaved data in the memory. This is not desirable in some applications, when the length of input bitstream is large [12]. Considering these issues, finding good deterministic interleavers to create similar performance to the random interleavers is followed.

In contrast to block interleavers, non-block interleavers are designed with lower numbers of memories and a self-synchronization property with their deinterleavers so as to reduce the design complexity. This can be considered as one of their advantages. In turbo code applications, the performance of the code is accomplished by using the continuous form at the expense of analysis at the decoder.

In order to utilize the advantages of non-block interleavers in turbo codes, these interleavers need to operate as block interleavers. This can be simply achieved when some stuff bits are inserted at the end of each data block forcing the interleaver memories to the known state, which is usually considered as the zero state. In addition to utilizing advantages of non-block interleavers, this operation simplifies the code analysis and the decoding procedure. The disadvantage of this procedure is a reduction of the available channel bandwidth due to the insertion of stuff bits that do not carry any information. In order to reduce the number of the stuff bits, some optimizations can be performed to the interleaver structure and RSC codes to decrease the number of stuff bits at the encoder output.

1.3 Aims of the Thesis

In this thesis, the block-wise operation of convolutional interleavers, being the most conventional deterministic non-block interleavers is considered for turbo codes. This type of interleaver involves a number of parallel lines, in which each line usually has a different number of memories compared to other lines. In this structure, the number of interleaver lines and different numbers of memories fixed between two adjacent lines represent the period and space value parameters, respectively. Depending on the stuff bits position in the interleaved data different convolutional interleavers can be defined. The main aim of thesis is to show that it is possible to construct such convolutional interleavers that can perform in turbo codes close or even better than most of conventional deterministic and random block interleavers. The thesis also aims to develop a method for using turbo codes with convolutional interleavers to provide unequal error protection codes.

The thesis introduces an optimization methods for this interleaver, which is per-

formed by deletion of stuff bits located in the end part of the interleaved data. The performance of the optimized convolutional interleavers is compared with the non-optimized convolutional interleavers. Applying the convolutional interleaver properties, a very simple and efficient algorithm is presented to calculate the weight distribution of the code. The obtained weight distributions confirm that the optimized interleaver provides a free distance value with low multiplicity, which can improve the code performance in the error floor region compared to block interleavers. In addition, conducted analysis and simulations conducted for different turbo codes, it is shown that optimized convolutional interleavers outperforms the non-optimized convolutional interleavers, when similar number of stuff bits are considered for both interleaved data. Also, different structures for optimized convolutional interleavers are presented based on different periods and space values. In each case, an appropriate modification is proposed improving the code performance with the reduced number of stuff bits. The performance of modified interleavers are compared with the most conventional block interleavrers. The results represent that with the reasonable number of stuff bits the designed interleavers have close or even better performance than the most conventional deterministic and random block interleavers. Finally, the application of the optimized convolutional interleaver in turbo codes with Unequal Error Protection (UEP) is presented.

1.4 Thesis Overview

- **Chapter 2** begins with an explanation of convolutional code structures. Since turbo codes are basically constructed by convolutional codes, the structure of convolutional codes and maximum likelihood decoding technique using the Viterbi algorithm is briefly presented. Then, the structure of turbo codes and their analysis based on maximum likelihood iterative decoding is considered. It is explained how low weight distributions of the code due to structure of RSC codes are generated. Several deterministic and random interleavers with block-wise performance are reviewed. For each group, relevant interleavers and their performance when applied to turbo codes are explained. Then, the structure of the turbo decoder using Soft Output Viterbi Algorithm (SOVA) is

presented. For this purpose, a concept of Log-Likelihood Ratios (LLR) usable for the iterative turbo decoding is explained. Based on the LLR definition and its characteristics, two modifications to the Viterbi algorithm are proposed to provide the Maximum Likelihood (ML) soft decoded information for the next steps of decoding. For SOVA, several methods are reviewed to improve the iterative decoding performance.

- **Chapter 3** deals with non-block interleaver structures, specifically the convolutional interleavers. First, one model of convolutional interleaver acting as a block interleaver is proposed by insertion of enough number of stuff bits to its memories. These stuff bits are located in the beginning and end part of the interleaved data. For each interleaver, a relevant iterative turbo decoding process is discussed. Based on the convolutional interleaver properties, a simple algorithm is implemented to calculate the weight distribution of turbo codes. On the basis of weights calculated by this algorithm, turbo codes constructed with convolutional interleavers are analyzed to determine contribution of calculated weights to the code performance. Simulations confirm that, in the case of similar numbers of stuff bits, application of the optimized interleaver leads to a better code performance than the non-optimized interleaver. The performance of turbo codes with an optimized convolutional interleaver with space value 1 and different periods is verified and compared with conventional block interleavers. For short bitstream lengths the comparison indicates that in the case of a suitable number of stuff bits, convolutional interleavers lead to a similar or even better performance than block interleavers. For the medium to high bitstream lengths, these interleavers do not perform well, especially in the error floor region. This is due to a low free distance value for the code.

- **Chapter 4** presents a modified algorithm for optimized convolutional interleavers. The modification increases the distance between adjacent bits of the original bitstream in the interleaved data to generate a higher free distance value for the code. The performance of these modified interleavers is analyzed by calculating weight distribution of the code and confirmed with simulation results.

- **Chapter 5** considers optimized convolutional interleavers designed with a higher space value. Different interleavers with this characteristic are designed and their performances compared with the proposed interleaver with higher periods and the space value 1. The comparison is conducted for interleavers having similar number of memories in their structures. This leads to interleavers designed with lower periods. In some cases, applying an interleaver with the lower period provides insufficient behavior for the code as it results in generation of a relatively low weight with high multiplicities. In order to remove this effect, appropriate modifications for the interleavers are suggested creating similar or better performance that interleavers with higher periods and space value 1.

- **Chapter 6** analyzes the code performance with generalized convolutional interleavers. In contrast to the interleavers proposed in the previous chapters, the space parameter of these interleavers is not assumed to be a constant value. Among numerous different cases, designed interleavers compatible with the RSC code structure are presented. In addition, for each generalized convolutional interleaver, a relevant modification is conducted. Finally, modified convolutional interleavers based on algorithms proposed in this and previous chapters are applied to construct good convolutional interleavers having better or close performance to the good block interleavers.

- **Chapter 7** presents the application of the optimized convolutional interleaver for Unequal Error Protection (UEP) turbo codes. Three different techniques are proposed based on the convolutional interleaver properties. The performance of the UEP turbo codes designed using these techniques are verified and the most suitable interleaver structure is selected for their application.

- **Chapter 8** concludes the thesis and gives some suggestions for the further research.

1.5 Contributions

The main contributions of this thesis are as follows:

1. Two models of convolutional interleavers acting as block interleaver, through insertion of a number of stuff bits to the interleaver memories to separate the blocks. For both models, relevant iterative turbo decoders are introduced [13, 14].

2. A simple algorithm for calculation of free distance value of turbo code using convolutional interleavers. The algorithm is then extrapolated to calculate distributions of other low weights, which influence the code performance [15–17].

3. Efficient modification techniques for convolutional interleavers, which improve the code performance with lower number of stuff bits. Modification techniques are applied for interleavers with different periods and the fixed space values [18, 19].

4. Design of generalized convolutional interleavers constructed with variable space value on the basis of weight-2 distribution of turbo codes. Relevant modifications for these interleavers are also proposed. The performance of these interleavers are compared with other convolutional and the most of the conventional block interleavers [20] .

5. Three different techniques for Unequal Error Protection (UEP) turbo codes, based on convolutional interleaver properties. The techniques are compared with each other to select the suitable one for the turbo code using short and long interleaver lengths [21].

1.6 List of Publications

The thesis contributions have been presented on the following published or submitted conferences and journals.

S.Vafi and T.Wysocki," Application of convolutional interleavers in turbo codes with unequal error protection", Accepted for the Journal of Telecommunications and Information Technology.

S.Vafi and T.Wysocki," Generalized convolutional interleaver and its performance in turbo codes", Submitted to IEEE Communications Letters.

S.Vafi and T.Wysocki, "Weight distribution of turbo codes with convolutional interleavers", Submitted to IEE Proceedings of Communications.

S.Vafi and T.Wysocki, "On the performance of turbo codes with convolutional interleavers", 11th Asia-Pacific Conference on Communications (APCC), pp.222-226, Oct.2005.

S.Vafi and T.Wysocki, "Performance of convolutional interleavers with different spacing parameters in turbo codes", 5th Australian Communication Theory Workshop (AusCTW05), pp.8-13, Feb. 2005,

S.Vafi and T.Wysocki, "Modified convolutional interleavers and their application in turbo codes", 2nd IEEE Symposium on Trends In Communications and joint ISI workshop on Mobile Future (SympoTIC), pp.54-57, Oct 2004.

S.Vafi and T.Wysocki, "Computation of free distance and low weight distribution of turbo codes with convolutional interleavers", 15th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC), pp. 1356-1359, Sep 2004.

S.Vafi and T.Wysocki, "Iterative turbo decoder design with convolutional interleavers", 4th international symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP), pp. 124-127, July 2004.

S.Vafi and T.Wysocki and I.Burnett, "Convolutional interleavers for unequal error protection of turbo codes", Joint 7th International Symposium on DSP and Communication Systems (DSPCS) and 2nd Workshop on the Internet, Telecommunications and Signal Processing (WITSP), pp.485-491, Dec.2003.

Chapter 2

The Structure of Turbo Codes

2.1 Introduction

Turbo codes are introduced as one of the most powerful error control codes. They are basically constructed by two parallel Recursive Systematic Convolutional (RSC) codes, which are linked by an interleaver. Generally, turbo encoded data are decoded by iterative decoding techniques. Due to the feedback connection from the output to the input of the RSC encoder, it is possible to find bitstreams that automatically return the RSC encoders to zero state. This generates codewords with low weight for the turbo code. As one of the effective solution to reduce this drawback, application of good interleavers is suggested. The interleavers are designed in such a way that to prohibit generation of bad bitstreams of the second RSC codes. This chapter reviews structure of turbo codes.

First, structure of the convolutional code and its maximum likelihood decoding methods utilizing with Viterbi algorithm is reviewed. Then the analysis of turbo codes based on weight distribution is presented. For the block-wise turbo code performance several interleavers are reviewed. Among several suggested algorithms, the iterative turbo decoding by Soft Output Viterbi Algorithm (SOVA) is discussed and some modifications improving its performance are presented.

2.2 Convolutional Encoder

A convolutional encoder with the rate $R = \frac{k}{n}$ is constructed on a basis of k input bits, n output bits and m memory units. The memory outputs and input data are joined each other in the required combination by an Exclusive OR (XOR) operator which generates the output bits [22] [23]. Figure 2.1(a) shows the convolutional encoder $(n = 2, k = 1, m = 2)$ structure. In the convolutional encoder, one bit entering the encoder will affect to the code performance for $m+1$ time slots, which represents the constraint length value of the code. Since an XOR is a linear operation, the convolutional encoder is a linear feedforward circuit. Based on this property, the encoder outputs can be obtained by convolution of input bits with n impulse responses. The impulse responses are obtained by considering input bitstream (100...0) and observing the output sequences [22]. Generally, these impulse responses are called generator sequences having lengths equal to the constraint length of the code. The generator sequences determine the existence connection between the encoder memories, its input and output. For the illustrated encoder in Figure 2.1(a) the generator matrix $G(D)$ is of the form $G(D) = [g_1, g_2]$ $(g_1 = (111)_2 = (7)_8, g_2 = (101)_2 = (5)_8)$.

A convolutional encoder can also be considered as a sequential circuit. Based on this approach it is possible to illustrate its behavior by the state diagram [22]. This diagram has 2^m distinct states corresponding with the possible memories state of the encoder. In the state diagram, 2^k branches leave each state and enter the new state to represent the state transitions for memories and the encoder outputs based on the combination of input data. Figure 2.1(b) shows the state diagram of the convolutional encoder (2,1,2) in the above example. In this figure, dotted and solid lines represent input bits of 0 and 1, respectively, while the values on the top of each line, indicate the first and the second encoder output bits, respectively.

2.3 Convolutional Decoding

Among the different decoding techniques proposed for convolutional codes, the Viterbi algorithm is the most popular due to its high error correction performance. The al-

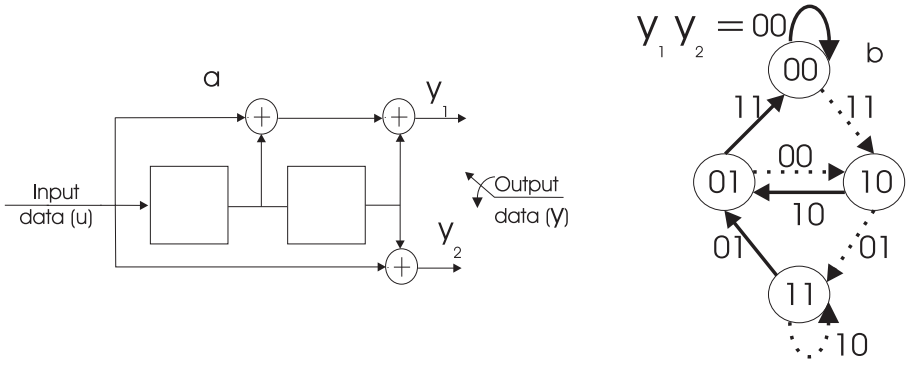

Figure 2.1 a) Convolutional encoder (2,1,2) structure, b) state diagram of the implemented code.

gorithm is basically implemented by a trellis diagram, which is the expansion of the state diagram in the time domain [24] [25]. Figure 2.2 shows the trellis diagram of the encoder (2,1,2) of Figure 2.1(a) with an input data length $L = 5$. Generally, a convolutional code can be regarded as a block code when input data blocks are considered as input and enough zero bits equal to the number of encoder memories are appended at the end of each data block returning the encoder memories to the zero state. This technique is called trellis termination [26, 27]. Practically, the length of the input data block relative to the number of encoder memories is considered to be high so as to reduce the effect of tail bits inserted at the end of each block as the overall code rate value. Tail bits are also considered in the trellis diagram to represent the code action and consequently should be involved in the decoding process of the code. Therefore, a trellis diagram has $L + m + 1$ time units for a code (n, k, m) with an input data length L and trellis termination. Instead of inserting zero bits, it is possible to generate the encoded data block using hardware resetting to return the encoder memories returning them to the zero state. This technique is called trellis truncation [26,27] and the trellis diagram of codes with this termination method have just $L + 1$ time units.

Similarly to the state diagram, the 2^k distinct paths enter or exit from each state of the trellis diagram. Thus, the diagram has a total 2^{kL} paths and the decoder should selects L paths from amongst them. The main function of the Viterbi algorithm is selection of the path having the maximum likelihood on the basis of the original

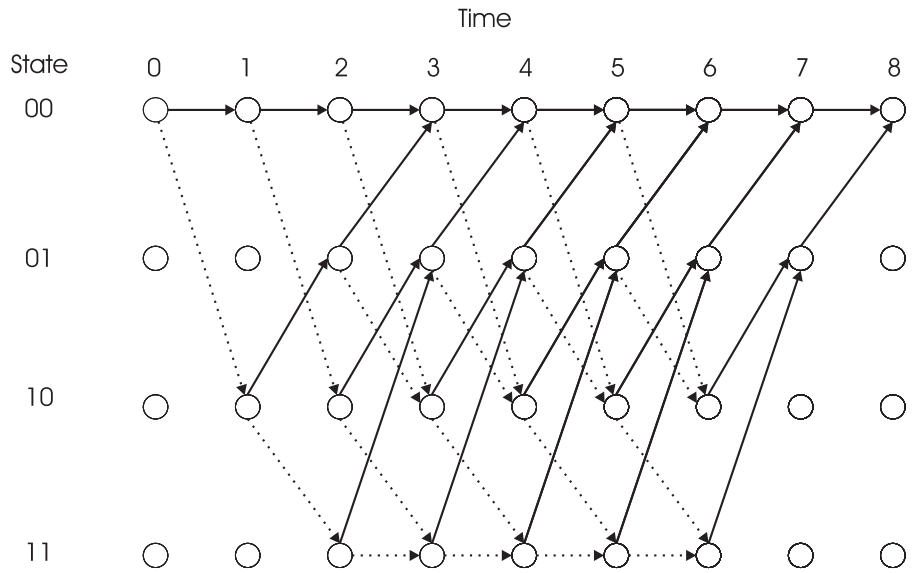

Figure 2.2 Trellis diagram of the convolutional code (2,1,2) with trellis termination for the data length $L = 6$.

data [28, 29].

The Viterbi algorithm can be implemented by two different approaches. In the hard decision approach, the received information from the noisy channel is quantized to 2 levels producing the binary data for the decoder input [30, 31]. In this approach, finding the most likelihood path is conducted based on selection of paths with the minimum Hamming distance. Hamming distance is defined as the number of different bits between the decoded data and the input data of the encoder. For example, due to the difference in the third and forth bits of data (101001) and (100101), Hamming distance is equal to 2. In the Viterbi algorithm this distance is introduced as branch metric, which for simplicity is called just metric.

At each time unit of the trellis diagram, branch metrics are computed based on how many bits are different between the received information and the transmitted information. Then, the path with the minimum branch metric is selected as the survivor path. The obtained branch metric at each time unit is added to the previously accumulated branch metric forming the path metric at the relevant time unit. This procedure is continued to decode all the received information. Following, an example is

presented to clarify this approach.

Assume the sequence (00,11,11,00,01,10,01,11) is the received information at the decoder input for the convolutional code (2,1,2) encoded by the encoder structure illustrated in Figure 2.1(a) from the input bitstream (0,1,0,1,1,1,0,0). This has been indicated in the trellis diagram of Figure 2.3. Initially, the decoding procedure is started from all zero state. From this stage, two paths are leaving to the state 00 and 10. The algorithm computes Hamming distance of the received symbol 00 with the symbols 00 and 11 and selects the path 0 as a survivor path. The computed distances, i.e. 0 and 2 related to the paths 0 and 1 in the time unit 1 are recorded for the decoding procedure for the next steps as the path metrics. In the next step, i.e. time slot $t = 1$, again the branch metrics are computed and added to the previously computed path metrics. Then, the path with the minimum accumulated distance value is selected as a survivor path at the time instant $t = 2$.

This procedure is repeated for other time slots to finalize the decoding process. The decoded information have been highlighted in Figure 2.3. It should be noted that in the case of two paths with the identical metric value merging to different states, a path is optionally selected. This condition has occurred at the time instant $t = 3$, where the algorithm selects '0' as the decoded bit. However, it is possible that this assumption does not provide the maximum likelihood decoded data. This is noticed that in the decoding process at the time instant $t = 5$. If at the time instant $t = 8$, the algorithm selects the path 0 which has an equal distance with the path 1, i.e.2, it will realize that the wrong path has been selected, because the minimum metric in this updated time, i.e. 2, is related to the path originated from the path 0 of the state (01) at the time instant $t = 7$. When the algorithm faces this problem, it will select another path relevant to the minimum path metric and will trace to find the other paths in the backward direction, which originated from the minimum path metrics in the previous time instants. In this example, at the time instant $t = 8$, the algorithm selects path 0 originated from the state (01) at the time $t = 7$ as the survivor path instead of the path 1 and decodes paths 0 and 1 as the survivor paths at time unit 6 and 5, respectively. In this case, the algorithm corrects the likely error that occurred due to the wrong assumption in the selection of the path 0 at the time $t = 5$. In this

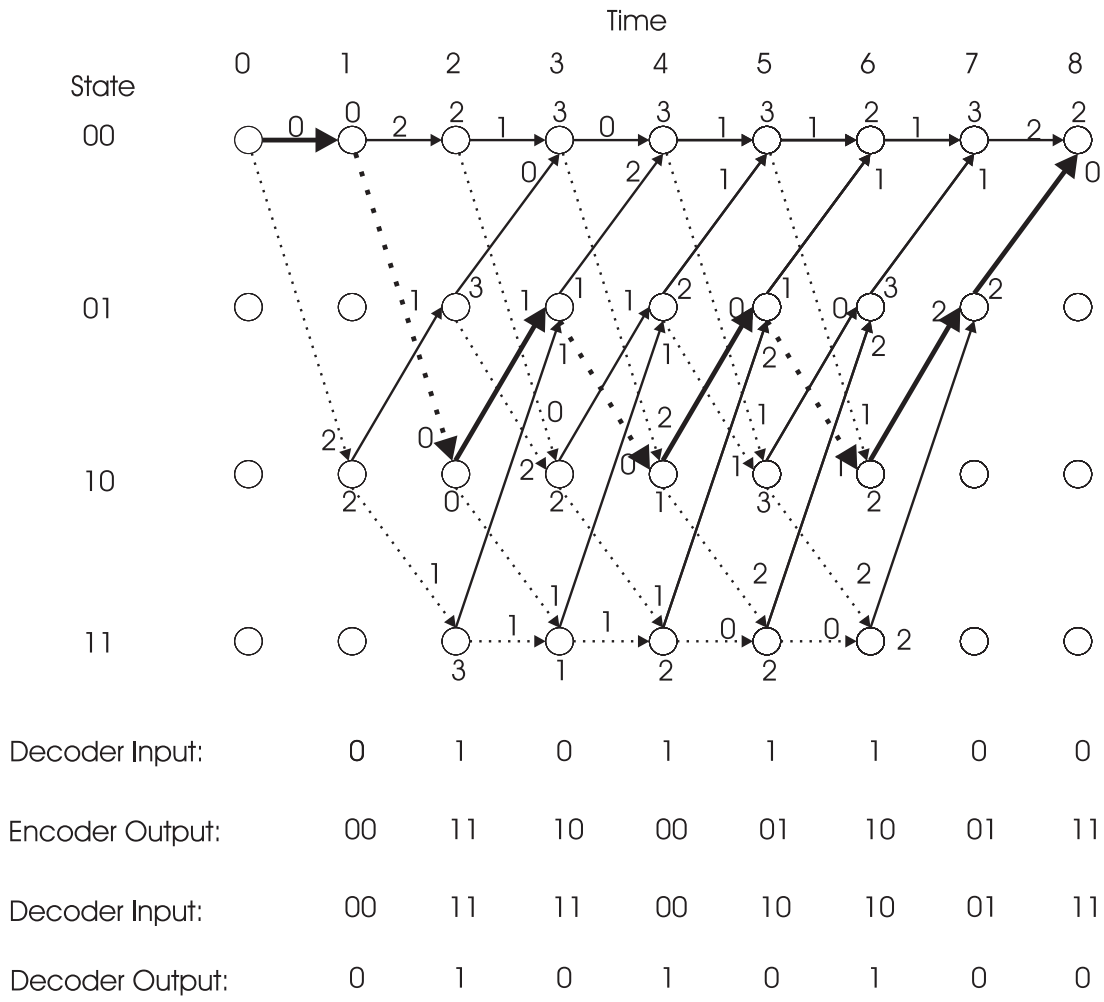

Figure 2.3 Trellis diagram for the hard decision decoding of the convolutional code (2,1,2).

example, the finally maximum likelihood decoded data provides 1 bit error which has occurred at the time unit $t = 5$.

On the other hand, the received information from the channel can be quantized by more than 2 levels and these information are utilized as input information of convolutional decoder. This approach is called soft decision decoding, which searches for the closest path based on the Euclidean distance [28]. In this case, for the code with the rate $R = \frac{1}{n}$ $(k = 1)$, the branch metric values of the trellis diagram for the time t can be computed as follows: [30, 32]

$$M_{i,j,t} = \sum_{l=0}^{n-1} (y_{l,t} - C_{i,j,t})^2 \tag{2.1}$$

where y_l and $C_{i,j}$ give the lth received information and the output transition information for the transition from state j to state i, respectively. Expanding this equation yields:

$$M_{i,j,t} = \sum_{l=0}^{n-1} y_{l,t}^2 - \sum_{l=0}^{n-1} 2y_{l,t}C_{i,j,t} + \sum_{l=0}^{n-1} C_{i,j,t}^2 \tag{2.2}$$

In order to minimize the $M_{i,j,t}$, the portions of the equation that are different for each path are concerned. The terms $\sum_{l=0}^{n-1} y_{l,t}^2$ and $\sum_{l=0}^{n-1} C_{i,j,t}^2$ are constant for all paths at the specific time. Therefore, they can be eliminated to form the following equation:

$$M_{i,j,t} = \sum_{l=0}^{n-1} -2y_{l,t}C_{i,j,t} \tag{2.3}$$

Since $M_{i,j,t}$ is a negative value, its minimum value occurs when the $\sum_{l=0}^{n-1} 2y_{l,t}C_{i,j,t}$ is maximum. Ignoring coefficient 2 from the above equation, the branch metric is given by:

$$M_{i,j,t} = \sum_{l=0}^{n-1} y_{l,t}C_{i,j,t} \tag{2.4}$$

At each time instant, the soft decision approach searches for the survivor path among paths that have left from the state including the survivor path at the previous time. Based on the selected state at the previous time, the path metrics of possible states at the next time instant are calculated and the state with the highest path metric is selected. If the path metric of this state not originated from the previously selected state, the algorithm considers another path with the highest path metric. Then it traces back till the end of the required time unit finding the path with the highest path metrics in the backward direction to obtain maximum likelihood decoded information. This procedure is followed for every time instant to finalize the decoding procedure.

As an example, for the previously presented code (2,1,2) with the length $L = 5$, the soft information obtained from the channel (-1.5,-1.3,-1.9,-0.7,1.7,-2.4,0.4,-1.2,-0.9,1.1,1.6,-0.6,-0.4,2.4,0.7,0.6) is considered as the input information of decoder. Based on the explained procedure, decoding has been accomplished and the decoded data has been highlighted in Figure 2.4 , which has correctly recovered the input bitstream of the encoder. In comparison with the hard decision approach, this ap-

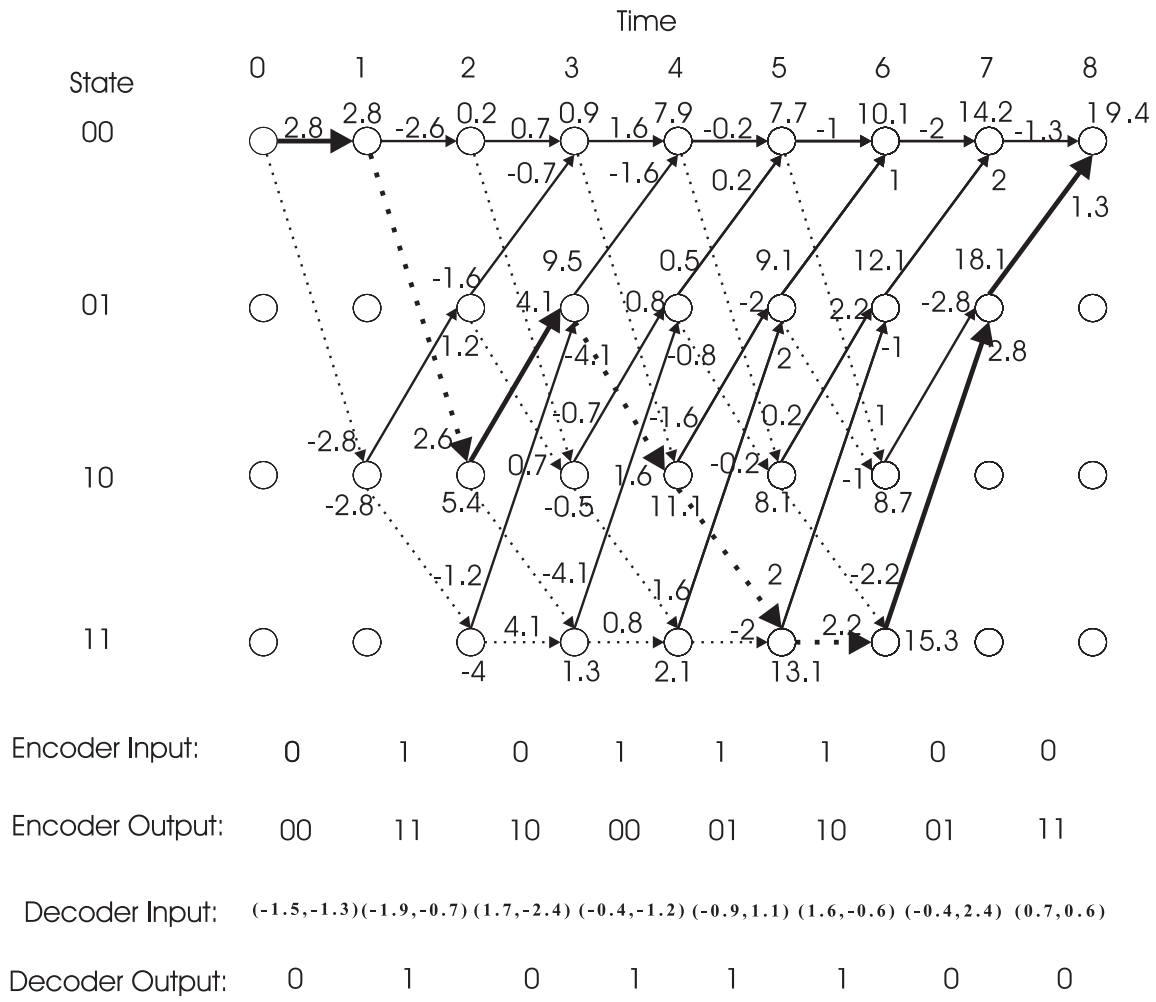

Figure 2.4 Trellis diagram for the soft decision decoding of the convolutional code (2,1,2).

proach has better performance in error correction of the code and hence in most of the applications the Viterbi decoding is implemented as soft decision algorithm.

2.4 Turbo Encoder

A turbo encoder is constructed by a parallel concatenation of two identical component codes, which are linked by an interleaver [33]. Generally, Recursive Systematic Convolutional (RSC) codes with the rate $\frac{1}{2}$ are applied as the component codes. However, it is possible to utilize block codes instead of convolutional codes as component codes [34]. Also, the structure of component codes can be different from each other

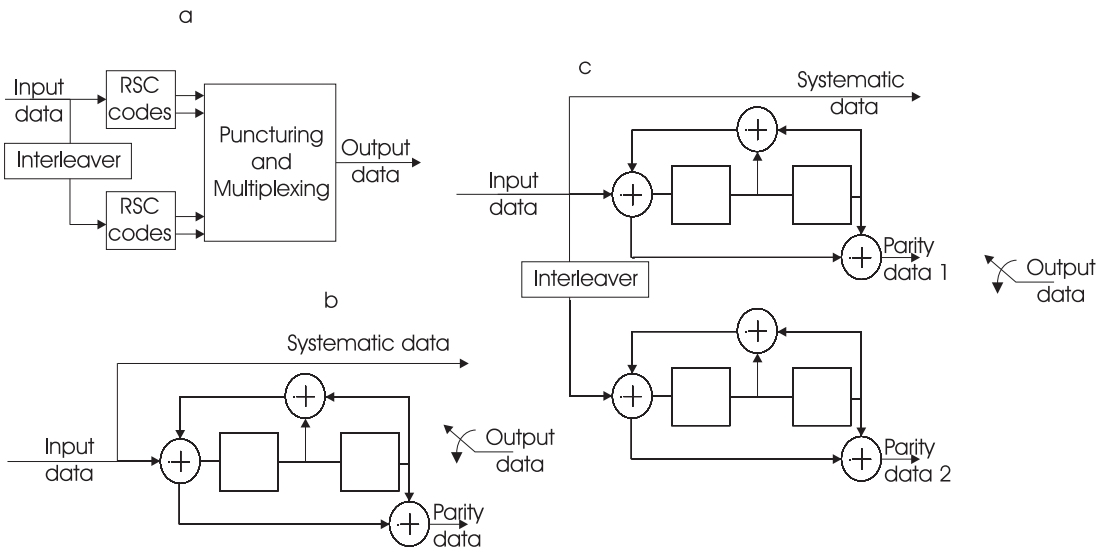

Figure 2.5 Turbo encoder structure. a) Block diagram of turbo encoders with rate $\frac{1}{2}$, b) RSC encoder $g_0 = (5)_8$, $g_1 = (7)_8$ and c) full rate 4-state turbo encoder$(1, \frac{5}{7})_8$.

resulting in asymmetric turbo codes [35]. For an RSC encoder with rate $\frac{1}{2}$, input bit-streams are directly transferred to the encoder output to form systematic data part of the encoder. Based on the feedback connection from another RSC encoder output to the encoder input, the systematic bits are encoded providing parity bits. Similarly to the convolutional code, it is possible to define the generator matrix for the RSC code. This matrix can be represented as $G(D) = (1, \frac{g_0}{g_1})$, where 1, g_0, g_1 introduce the systematic, feedforward and feedback connections of the RSC encoder, respectively. For the proposed RSC code, the polynomials $g_0 = 1 + D^2$ and $g_1 = 1 + D + D^2$ are obtained and represented by the values of 5 and 7 in the octal mode. Figures 2.5(a) and 2.5(b) show block diagrams of the turbo encoder and a simple structure of the RSC encoder $(1, \frac{5}{7})$ with rate $\frac{1}{2}$, respectively.

In most applications, channel codes are designed with rate $\frac{1}{2}$. In order to achieve this rate for a turbo code, it is necessary to puncture half of the bits of each RSC encoded data. Since puncturing of the systematic bits dramatically reduces the code performance [36], instead of sending two half punctured systematic bits from two RSC encoders, only systematic bits of the first RSC encoder are fully transmitted and puncturing is only performed on the parity bits forming the desired code rate.

Therefore, in the case of non-puncturing, a turbo code with rate $\frac{1}{3}$ is constructed. It has been confirmed that the equally distributed puncturing between two parity data creates the best performance for the turbo code [37]. Figure 2.5(c) shows the full rate 4-state turbo code $(1, \frac{5}{7})$ structure. Turbo encoders with lower rates can be achieved by parallelization of more RSC encoders connected by the interleavers to form multiple turbo codes [38, 39].

At the decoder, for each RSC code utilized at the encoder, a RSC decoder as a component decoder is considered. Based on the structure of turbo codes, the component decoders are connected to each other to construct a turbo decoder. Normally, decoding is accomplished by iterative methods [36]. In these methods, the decoder components are linked to each other by interleavers and deinterleavers. In every iteration, each decoder provides soft output information for the alternative decoder to be utilized as one of the soft decoder inputs for the next decoding iteration. Soft output information is represented by a Log-Likelihood Ratios (LLR) concept, which gives a probability of the decoded information in terms of the transmitted information. The iterative decoding is continued until the decoded information achieves the required maximum likelihood to the original bitstream. It is obvious that depending on the iteration number, decoding complexity will be increased. Practically, decoding is accomplished by 8 to 10 iterations [40].

2.5 Interleaving

Interleaving generally refers to a process which permutes symbols of an input sequence. It is especially utilized in forward error correction coding to reduce the effect of impulse noise and burst errors in fading and multipath channels. For the same reason it is also applied in magnetic recording systems [41]. Mathematically, the relation between the input sequence $x[k]_{k=-\infty}^{\infty}$ including infinite symbols and the interleaved sequence $y[k]_{k=-\infty}^{\infty}$ is defined by permuted law $\pi(k)$ as follows [41]:

$$y[k] = x[\pi(k)]$$

Table 2.1 An interleaved sequence with period 4, $d_{max} = 7$, $d_{min} = -2$ and latency 9.

i	...	-2	-1	0	1	2	3	4	5	6	7	8	9
$\pi(i)$	...	0	-5	-7	-2	4	-1	-3	2	8	3	1	6
i-$\pi(i)$	...	-2	4	7	3	-2	4	7	3	-2	4	7	3
$x(i)$	...	1	0	1	1	0	0	1	0	0	1	1	0
$y(i)$	...	1	$x(-5)$	$x(-7)$	1	1	0	$x(-3)$	0	1	0	1	0

An interleaver is considered as a periodic interleaver with the period T when it satisfies the following equation for all symbols of the sequence [41].

$$\pi(x + T) = \pi(x) + T \tag{2.5}$$

Depending on the time difference between an entrance of the symbols to the interleaver and the obtained interleaved sequences, maximum and minimum delay of the interleaver (d_{max}, d_{min}) are calculated by [41, 42]:

$$d_{max} = max(\pi(x) - x) \tag{2.6}$$

$$d_{min} = min(\pi(x) - x) \tag{2.7}$$

Based on the above definitions, the latency parameter is defined as the difference between maximum and minimum delay of the interleaver [41, 42]:

$$d = d_{max} - d_{min} \tag{2.8}$$

Also, an interleaver is referred to as a causal interleaver if a symbol can not exit before it enters. Therefore, a causal interleaver satisfies $d_{min} \geq 0$ or $d_{max} \geq d$ relations [41, 42]. Table 2.1 gives an example of the interleaved sequence $\mathbf{y}$ obtained from the input sequence $\mathbf{x}$ by the permutation law π. In this example, the interleaver has period 4, the latency 9, and maximum and minimum delay values 7 and -2, respectively. For each interleaver, a deinterleaver is defined which reforms the original data at the receiver. Of course, it must be designed to be compatible with the structure of the interleaver.

Practically, only interleavers with periodic and causal properties are utilized. Periodic interleavers are mainly subdivided into the block and non-block interleavers. In the block interleavers, permutation of a data block is independently conducted from

other blocks. For this purpose, a number of memory units equal to the length of an input sequence is required. The content of every sequence is fully written to the interleaver memories and, according to the permutation law, reading from the memories is accomplished. Therefore, its delay bound (D) can be assessed by [43]:

$$0 \leq D \leq 2(L-1) \tag{2.9}$$

where L gives the interleaver length. Those interleavers which delay bound is out of this range, i.e. $D > 2(L-1)$, are considered as non-block interleavers.

2.6 Turbo Codes Analysis

A turbo code can be analyzed as a code with a block-wise or continuous performance. For the block-wise performance, similarly to the convolutional code, an analysis is performed based on the trellis diagram of the code with the conventional termination methods applied to the RSC codes to provide isolated codewords of the specified length [44]. In this case, a block interleaver, whose length is equal to the length of input bitstream is usually utilized.

Since with the continuous performance, termination methods are not applied to RSC codes, the memories state of the encoder at the end of an input bitstream is considered as the initial state for the next bitstream. This leads to utilization of non-block interleavers to sufficiently permute the incoming bitstreams of the second RSC encoder. Based on this structure, continuous decoding is accomplished. In comparison with the usual iterative turbo decoding method applied to turbo codes with block-wise performance, continuous methods produce better performance at the expense of increased complexity, which is directly related to the interleaver length and its structure [45]. In addition, results show that continuous decoding is more reliable in turbo codes with the higher number of states, while in codes with the lower number of states, it does not improve iterative decoding methods utilized for the block-wise performance of code [46]. Hence, in most of the applications, a turbo code with block structure is preferred.

Since turbo codes are linear codes, their analysis can be conducted by determination

of the upper bound of probability of error or the Bit Error Rate (BER) value with the maximum likelihood decoding as follows [47]:

$$BER \leq \sum_{d=d_{free}}^{3(L+m)} \frac{N_d \tilde{w}_d}{L} Q(\sqrt{d \frac{2RE_b}{N_0}}) \tag{2.10}$$

where R, $\frac{E_b}{N_0}$, m, N_d and ω_d denote the code rate, the signal to noise ratio per information bit, the number of tail bits, the number of multiplicities for weight d and the average weight of information of weight d, respectively.

The upper bound equation specifies that the code with the higher free distance value has better performance in terms of error reduction. Because of the feedback connection between the RSC encoder output and its input, the effect of bit 1 from the impulse response of the code, will last until certain external bits are inserted to the RSC encoders returning their memories to the zero state. Therefore, a higher weight and consequently improved performance for the code compared to a non-recursive convolutional code is expected. For example, for the input bitstream (10000000...0) the obtained codeword from the convolutional code (2,1,2) illustrated in Figure 2.1(a) is (11101100000000...0) with a weight 5, while its equivalent RSC code (1,$\frac{5}{7}$) gives the codeword (11101101...) with infinite length, whose weight for every 3 inserted zero bits is periodically increased by 2 units. The finite weights for this code can be obtained when input bitstreams with higher weights than 1 are applied in such a way that they return the RSC code to the zero state, without inserting any external bits to the memories. These patterns are called self-terminating patterns. Depending on the weight of self-terminating patterns and their combinations, low weights can be obtained for the code. For example, the weight-2 self-terminating pattern (100100) for the RSC code (1,$\frac{5}{7}$), generates a codeword (11010111) with weight 6. The similar pattern generates the weight 10 (ten) relevant to the codeword (111011111011) obtained from the trellis terminated convolutional code (2,1,2).

Similar conditions can occur from self-terminating patterns with higher weights than 2. For example, the weight-6 self-terminating pattern (11100111) generates a codeword (1110110000111011) with the weight 10 (ten) for the RSC code (1,$\frac{5}{7}$), while this pattern produces a codeword (11101001111101100111) with the weight 14 for the trellis terminated convolutional code (2,1,2).

In addition, if trellis termination is considered for the RSC code, existence of weight-1 input bitstream whose weight is positioned in the end part of the bitstream, can generate a codeword with a lower weight than a codeword obtained from the convolutional code. For the RSC code $(1, \frac{5}{7})$, the weight-1 bitstream (000..001) generates a codeword (101110) with the weight 5, which is lower than the obtained weight 6 from a codeword (111011) for the trellis terminated convolutional code (2,1,2).

The above analysis can be extended to the turbo codes which incorporate similar RSC codes. The obtained results from the iterative decoding indicate that the turbo code has very good performance in error reduction at the low signal to noise ratios. This area is named waterfall. At medium to high signal to noise ratios, which is named error floor region, BER slope of the iterative decoder is reduced significantly. At very low signal to noise ratios, BER stays high and at an almost constant value. This area is named non-convergence [48]. Figure 2.6 specifies the mentioned regions of turbo codes performance with the maximum likelihood iterative decoding methods. Analysis of the turbo code indicates that at low signal to noise ratios, a large number of codewords with medium weight determines the code performance. The results confirm that by increasing the length of the interleaver the influence of those weights on the code performance can be reduced [49–51]. At medium to high signal to noise ratios, only the first items of the weight distributions contribute to the code performance. This indicates that the interleaver type has the essential influence on the code performance [49–51].

2.7 Interleavers for Turbo Codes

The effect of the error floor can be reduced by applying an interleaver tailored to the RSC code structure to prohibit generation of self-terminating patterns for the second RSC code. Since the free distance value approximately determines the code performance at the error floor region and it is usually obtained from the weight-2 input bitstream, interleavers are particularly designed to improve the weight-2 distribution of the code [48]. The obtained free distance from the weight-2 distribution is called the effective free distance of the code. An interleaver is ideal when it breaks all self-

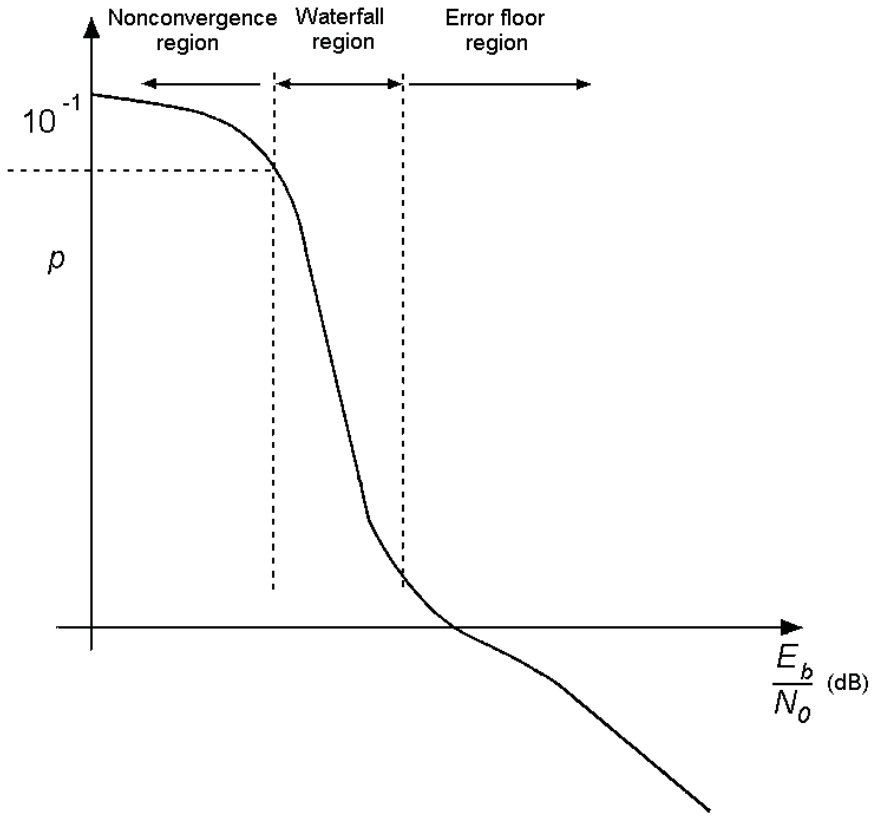

Figure 2.6 Turbo codes performance with the maximum likelihood iterative decoding.

terminating patterns generating higher weight for the code. In fact, the main function of interleaver is related to generation of a high weight for the second RSC code from the low weight input bitstream [52]. Since a detailed assessment of the interleaver is practically impossible, the interleavers are mainly assessed by the distance spectrum specification of the code, which is determined by $\sum_{\omega=1}^{L} a(\omega, d)\omega$ and is related to the number of input bitstreams weight w and codeword weight d as $a(\omega, d)$ parameter [10]. It is obvious that those interleavers that produce higher weights will have better performance in turbo code applications.

Considering the weight-2 distribution of the code, an interleaver can also improve the code performance when increases the distance between bits 1 in the interleaved data. The interleaver influence on the weight-2 distribution of the code is considered insufficient, when it permutes a self-terminating pattern to another self-terminating

pattern in such a way that it produces an equal distance between bits 1 of both patterns.

Consider the self-terminating (100100000000) pattern as a systematic data for the turbo code $(1, \frac{5}{7})$. Based on the performance of two independent interleavers, two self-terminating patterns (1001000000) and (10000000010) have been obtained. The second pattern due to the longer distance between two bits 1 generates a codeword with the weight 8, which is 4 units greater than the weight of codeword of the first pattern. Since the second interleaver increases the distance between two bits 1, the higher weight for the second parity data and consequently better performance for the code will be expected.

As another approach, interleavers can be designed based on the iterative decoding performance of the code to provide uniform correlation between soft output information created from each component decoder and the received information from the channel output. Existence of this correlation allows the component decoders to provide better maximum likelihood decoded information than the original bitstream [53, 54].

In the next section, interleavers constructed for turbo codes are reviewed. The proposed interleavers are mainly designed based on the distance spectrum specification of the code. However, examples of interleavers designed based on the iterative turbo decoding are also presented.

2.7.1 Interleavers Design Based on the Distance Spectrum

In this section, the typical conventional block interleavers used in turbo code applications are introduced. Based on deterministic or random permutation rules, block interleavers can be divided into two main groups. In the deterministic permutation, the positions of each interleaved bit for the whole interleaved data blocks is constant, while in the random permutation these positions are randomly changed. For each permutation rule, relevant interleavers are introduced.

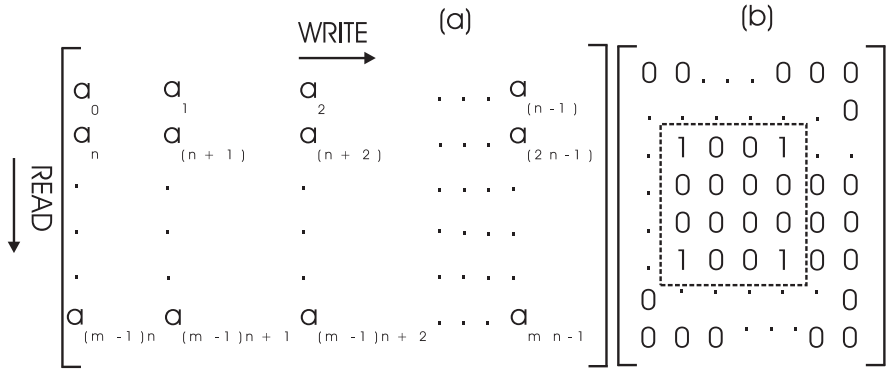

Figure 2.7 a) Permutation process of the row-column interleaver, b) a self-terminated pattern with weight-4 providing a low weight for the 4-state turbo code $(1, \frac{5}{7})$.

2.7.1.1 Row-column Interleaver

The row-column interleaver is a simple block interleaver, which has been utilized in turbo codes. In this interleaver, data are written to its memories row-by-row and then read from them column-by-column [55]. Hence, its permutation is carried out by the following equation:

$$\pi(i) = mn\lfloor\frac{i}{mn}\rfloor + n(imodn) + \lfloor\frac{kmodmn}{m}\rfloor \tag{2.11}$$

Figure 2.7(a) shows a general structure of the row-column interleaver. In the case of identical row and column dimensions, the distance between two adjacent bits of the input bitstream in the interleaved data with length L would be $\sqrt{L}$. Although the generated distance is relatively high, in some special cases of turbo code applications, it will not provide sufficiently interleaved data for the second RSC encoder. For example, in the case of the 4-state turbo code $(1, \frac{5}{7})$, an interleaved pattern with weight-4, as illustrated in Figure 2.7(b) returns both RSC encoders to the zero state, which produces the low weight for the code [35]. Conducted analysis of the turbo code confirms that the free distance value of code is achieved from weight-4 self-terminating patterns whose bits 1 are positioned in similar with bits 1 positions of Figure 2.7(b). The free distance obtained for the code with the row-column interleaver has a high multiplicity. In this case, the first term of the equation (2.10), which is related to free distance specifications, dominates the code performance [10, 47]. Even increasing the interleaver length will not significantly improve the code performance, because

Table 2.2 Pseudo-random function for Berrou-Glavieux interleaver.

ξ	$P(\xi)$
0	17
1	37
2	19
3	29
4	41
5	23
6	13
7	7

multiplicity of these self-terminating patterns will also increase in such a way that the ratio $\frac{N_{free}}{L}$ in equation (2.10) will remain approximately constant [47]. In addition, this interleaver cannot break down the bit 1 located at the end of the data block [56]. This behavior produces a low weight for the turbo code.

2.7.1.2 Berrou-Glavieux Interleaver

In order to improve performance of the row-column interleaver in the turbo codes application, Berrou and Glavieux designed an interleaver, which breaks bad weight-4 self-terminating patterns. The interleaver is constructed with $\omega \times \omega$ dimension (ω is a power of two) in which data are written row-by-row and read pseudo-randomly from the memories by the following rule: [36]

$$i_r = (\frac{\omega}{2} + 1)(i + j)(mod\omega) \quad (2.12)$$

$$\xi = (i + j)(mod8) \quad (2.13)$$

$$j_r = [P(\xi)(j + 1)] - 1(mod\omega) \quad (2.14)$$

where i and j are the row and column of writing a symbol and the i_r and j_r give row and column position reading. Also, $P(\xi)$ is a number relatively prime with ω, which is determined by the values presented in Table 2.2.

2.7.1.3 Helical Interleaver

The helical interleaver is another modified row-column interleaver that writes data row-by-row in the memories and then reads them diagonally from the bottom-left

entry of the interleaver using the following permutation [57]:

$$\pi(t) = i_r C + j_r \quad (2.15)$$

$$i_r = C.R - 1 - t(modR) \quad (2.16)$$

$$j_r = t(modC) \quad (2.17)$$

where R and C are relatively prime and represent row and column of the interleaver, respectively. Permutation is conducted in such a way that both RSC encoders are simultaneously terminated to the zero state by appending only tail bits for the first RSC encoders. In [58], a new interleaver with a similar property for the turbo code is defined, too.

2.7.1.4 Reverse Row-column Interleaver

The reverse row-column interleaver was proposed as an interleaver to remove the drawback of bit 1 existence located at the end part of the interleaver. In this interleaver, data are written to the memories row-by-row and then column-by-column reading started from the last column [59]. In a similar way to the row-column interleaver, it can not remove the problem of bad weight-4 input bitstream. However, it provides good performance for the code with very short block length and for the moderate and long block length at very low signal to noise ratios.

2.7.1.5 Rotated and Backward Interleaver

Rotated and backward interleavers have been also proposed to remove the effect of bit l located at the end part of the row-column interleaver [60]. Similarly to the row-column interleavers, data are written row-by-row. For the rotated interleaver, the written data are rotated 90° and then reading is performed row-by-row. Figure 2.8(a) shows the interleaved data of the rotated interleaver with the dimension 3×3. In this permutation, the last bit of the input pattern will still be close to the end part of the interleaver. In the backward interleaver, row-by-row reading data starts from the last column of the interleaver in backward direction. Figure 2.8(b) shows the procedure of backward interleaving. It can be seen that the backward interleaver has increased the distance of the last bit of the proposed pattern from the end part of the interleaver.

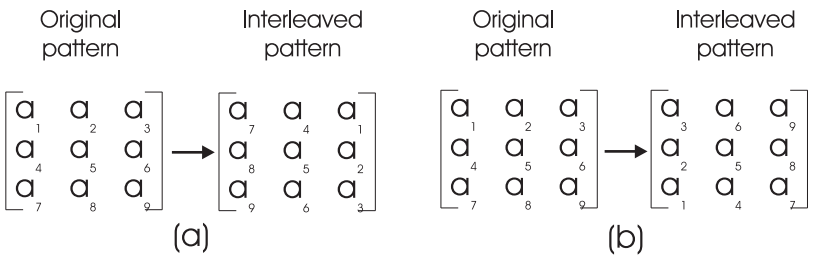

Figure 2.8 Interleaved data from modified block interleavers. a) Rotated interleaver and b) backward interleaver.

Application of these interleavers improves the turbo codes performance especially at medium signal to noise ratios for interleavers with short block lengths.

2.7.1.6 Circular-shift Interleaver

The circular-shift interleaver is an interleaver with the deterministic permutation rule, which permutes the input data by the following rule [61]:

$$\pi(j) = (aj + r)(modL) \tag{2.18}$$

Where r $(r < L)$ and a $(a < L)$ are relatively prime to L and represent the offset and step size of the interleaver, respectively. Normally, the offset value is considered to be equal to zero. For an interleaver with length $L = 11$, step size $a = 4$ and $r = 0$, the permutation process is illustrated in Table 2.3. In order to verify its performance based on weight-2 distribution, Divsalar $et.al$ calculated the maximum summation of the distance between two specified bits of the input bitstream and the interleaved data [11]. For the interleaved data in Table 2.3, the distance between two adjacent bits of the input sequence is 1, while the minimum and maximum distance between adjacent bits of the interleaved data are equal to 4 and 7, respectively. Then, the maximum summation of the distance between every two bits is calculated to be equal to 8. The results obtained for different lengths give $\sqrt{2L}$ as the maximum distance. The achieved distance represents good performance for the weight-2 distribution of the code. However, for higher weights, it is possible for the interleaver to permute self-terminating patterns to another self-terminating pattern, which provides the relatively low weight value with high multiplicities for the code.

Table 2.3
Permutation process of the circular shift interleaver with $L = 11$, $a = 4$ and $r = 0$..

i	0	1	2	3	4	5	6	7	8	9	10
$\pi(i)$	0	4	8	1	5	9	2	6	10	3	7

For example, an interleaver with $L = 32$ and $a = 7$ permutes input bitstream $\{10010$ $10010000000000000000000000000\}$ to $\{10010000000000000000010010000000\}$. For a turbo code $(1, \frac{5}{7})$ both patterns are considered as self-terminating patterns and will generate the similar weights with a value of 8.

2.7.1.7 Pseudo-random Interleaver

In contrast to the deterministic permutation role, it is possible to implement interleavers with random permutations. A simple random interleaver is constructed when it selects the memories randomly and reads their contents [11]. In the permutation, each memory is selected only once. The weight-2 distribution analysis of turbo codes indicates that the pseudo-random interleaver cannot provide suitably permuted data for the second RSC encoder. However, for input bitstreams with higher weights, or multiple turbo codes, the interleaver performs very well, breaking down bad input bitstream to improve the code performance.

2.7.1.8 Uniform Interleaver

Due to the unpredictable behavior of the pseudo-random interleaver, determining the weight distribution of the code, and consequently its analysis and design is a major obstacle [62]. In order to overcome this problem, Benedetto, $et.al$ proposed uniform interleavers, which consider equivalent probability of permutation for $L!$ interleavers (L is referred to the interleaver length) [49, 50, 63]. When this interleaver is utilized for the turbo codes, its performance is determined based on the average performance of all codes and it is expected that a code using the random interleaver has similar performance to a code utilizing this interleaver.

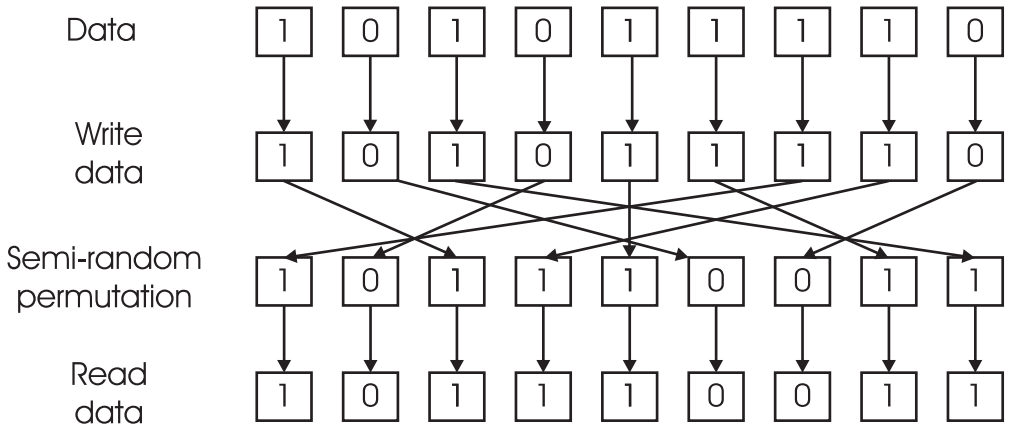

Figure 2.9 Permutation of data with a semi-random interleaver length $L = 9$ and $S = 3$.

2.7.1.9 Semi-random Interleaver

Semi-random interleavers are introduced as another type of random interleavers. They remove the drawback of pseudo-random interleavers for permutation of weight-2 input bitstreams. In order to guarantee that two bit 1s of these bitstream have sufficient distance from each other, a threshold value is considered in such a way that the distance between consecutive selected memories during the reading procedure is equal or greater than that value. In fact in this interleaver any two input bit positions with distance S can not be permuted to two bit positions, whose distance is less than S. Figure 2.9 shows the permutation process for the semi-random interleaver with length $L = 9$ and threshold value $S = 3$. As verified in [11], the best turbo code performance is achieved by the threshold value $\sqrt{\frac{L}{2}}$. Although the obtained distance in comparison with other interleavers such as the circular-shift and row-column is shorter, it efficiently breaks self-terminating patterns to provide the suitable pattern for the second RSC encoder.

2.7.1.10 Modified Pseudo-random Interleaver

In [64], a modification has been performed on the pseudo-random interleaver, which recognizes self terminating input bitstream when they are divisible by the generator matrix of the code $(G(D))$ and prohibits them to produce self-terminating interleaved data. Conducted simulations show that this interleaver gives better performance of

the turbo code compared to than the semi-random interleaver.

2.7.1.11 Swap Interleaver

A swap interleaver is introduced as an interleaver which slightly improves the semi-random interleaver performance [65]. First, a row-column interleaver is constructed and then two random positions are swapped. If they satisfy the threshold value considered for the semi-random interleaver, new positions are accepted. Otherwise, the above procedure is repeated for other position pairs.

2.7.1.12 Code-matched Interleaver

A modification on the semi-random interleaver based on the analysis of the code performance and its weight distribution characteristics is presented in [66]. A designed code-match interleaver breaks self-terminating patterns with weight no greater than 4 to eliminate the effect of first terms of weight distribution on the code performance, which determines the code performance at the error floor region.

Similarly to the method proposed in [64, 66], Abbasfar and Kao in [67] introduce an algorithm to fully eliminate all the codewords with weights less than a certain value for the code with random permutation. The algorithm tries to reduce the effect of low weights and their multiplicities having major contributions on the code performance. For different interleaver lengths, it was confirmed that the designed interleaver creates better performance than semi-random interleaver at the error floor region.

In [68], Yuan $et.al$ introduce an code-matched interleaver designed based on the generator matrix of the turbo code. This interleaver has better performance than interleaver proposed in [66] and maintains the error reduction property of the interleaver with the lower complexity.

Zhang $et.al$ in [69] present a Block-Random Chaotic (BRC) interleaver, which has lower complexity than the semi-random interleaver with the similar performance. Application of this interleaver has been verified for the 3rd Generation (3G) of mobile communications where the chaotic sequence are considered as input bitstream of the

turbo code.

2.7.1.13 Hokfelt Interleaver

In [70], an interleaver was designed in such a way that at least one RSC code generates a high weight whenever another RSC code produces a codeword with a low weight. The algorithm is applicable for the short bitstream lengths, while its complexity rapidly increases with increasing interleaver length. For the short bitstream length, it was confirmed that the Hokfelt interleaver has better performance than row-column, helical and pseudo-random interleavers.

2.7.1.14 Quadratic Interleaver

In [71] some deterministic interleavers such as quadratic interleavers are introduced. Quadratic interleavers are constructed based on a quadratic congruence. For an interleaver with length L $(L = 2^r)$, where r is an integer, an index mapping function is defined by the following equation:

$$C_i = \frac{ki(i+1)}{2}(modL)0 \leq i < k \tag{2.19}$$

where $C_0 = 0$ and k is odd. Then quadratic interleaver is defined as follows:

$$C_i \longrightarrow C_{(i+1)modL} \forall i \tag{2.20}$$

For example, for an interleaver with $L = 8$ and $k = 3$, the index mapping function is $\{0,3,1,2,6,5,7,4\}$. Based on equation (2.20), the quadratic interleaved data is $\{3,2,6,1,0,7,5,4\}$. In has been confirmed that these interleavers with simple permutation rules have similar performance to a random interleaver for long lengths.

2.7.1.15 Dithered Relative Prime Interleaver

Crozier proposed the Dithered Relative Prime (DRP) interleaver [72, 73] to increase the minimum distance of adjacent bits of the input bitstreams in the interleaved data. First, bitstreams with the length L are subdivided to smaller blocks as windows with the size M. Data of each window are locally permuted to the interleaver input. This process is called input dithering. Then the obtained data are cyclically shifted based on prime permutation $j = i + sp$, where s is a shift value, and p is prime relative

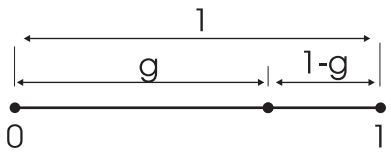

Figure 2.10 Illustration of the golden section principle.

to L. Finally, an output dithering similar to input dithering is performed. For the medium size blocks (i.e. $L = 256$ to $L = 4096$) a good M value is 8 [73]. The DRP interleavers perform well when both RSC encoders are terminated to the zero state. Comparisons show that for the full and punctured rate turbo codes this interleaver with a short block length, has similar performance to the semi-random interleaver but with lower complexity.

2.7.1.16 Golden Interleaver

Crozier also introduces new interleavers based on golden section definition in [74, 75]. For a given line segment of length 1, the golden section divide it into a long segment g and a shorter segment of $1 - g$ in such a way that the relation of the longer segment to the entire length is equal to the relation of the shorter segment to the longer segment. Therefore, $\frac{g}{1}=\frac{1-g}{g}$. solving this equation g value is given by $g \approx 0.618$. Figure 2.10 shows principle of golden section. This definition can be utilize to determine the golden section value for more points than 2 in a line segment. Conducted analysis in [75] confirm that the calculated golden section values generate uniform minimum distance between number of considered points. Therefore, it is expected that the interleaver constructed based on the golden section definition increases the distance between adjacent bits of input bitstreams in the interleaved data. Three proposed interleavers are called golden interleavers, golden relative interleavers and dithered golden interleavers. These interleavers apply golden section value and their cyclically shift permutation roles to increase the minimum distance of the code. Dithered golden interleavers typically provides the best performance [75]. This is especially evident for low code rates and large block sizes. However, the golden relative prime interleaver outperforms semi-random interleavers

for high punctured rates [75].

2.7.1.17 Matched Row-column Interleaver

Chan proposed a simple matched row-column interleaver for turbo codes in [76, 77]. For input bit streams with length L, a row-column interleaver with i columns is considered, where i varies from 1 to L. Based on L weight-2 distribution of the code obtained from these interleavers, an interleaver which provides a higher effective free distance value is selected. The results show that for short bitstream lengths, the designed interleaver outperforms the random interleaver and has similar performance to the golden interleavers. This technique is also applied for the reverse row-column interleavers, which outperforms the row-column matched block interleavers in the error floor region.

2.7.1.18 Simulated Annealing Algorithm Interleaver

Another deterministic interleaver, proposed in [78], is based on a Simulated Annealing (SA) algorithm. The algorithm defines an energy function for the interleaver parameters and the code characteristics. It tries to reduce the energy related to each parameter to obtain an optimized interleaver. The results suggest that applying suitable termination methods for the RSC code, the new interleaver has better performance than conventionally designed deterministic interleavers. In addition, for different bitstream lengths, it is concluded that in some cases, this interleaver outperforms the semi-random interleaver.

2.7.1.19 Quasi-cyclic Interleaver

An interleaver has been proposed based on quasi-cyclic permutation rule introduced in [79]. Similarly to the row-column interleaver, data are written row-by-row and then based on a randomly selected permutation data are read. Finally, some columns are cyclically shifted. In fact, the interleaver designed is considered as one of the random interleaver with the moderate algebraic structure. The results show that with less complexity, it provides better performance than the semi-random interleaver. Similarly, an interleaver has been proposed by Truhachev $et.al$. The method considers the effect of self-terminating patterns on the code performance and then applies

appropriate cycle permutation to break them down into the good patterns [80].

2.7.1.20 Permutation Monomials Interleaver

New interleavers have been constructed based on the monomial permutation approach, which permute the data with the length p^r over a finite field, where r is an integer number, $r \neq 1$ and p is any prime [81,82]. These interleavers have better performance than random interleavers and deterministic interleavers proposed in [71].

2.7.1.21 Integer Rings Interleaver

Recently, a new deterministic interleaver has been introduced based on permutation polynomials over integers ring [12]. It defines a permutation polynomial over integer ring when a polynomial $P(x) = a_0 + a_1x + a_2x^2 + ... + a_mx^m$ is considered as a permutation polynomial over integer ring Z_L $(L = 2^r)$, where r is an integer, if and only if 1) a_1 is odd 2) $a_2 + a_4 + a_6 + ...$ is even and 3) $a_3 + a_5 + a_7 + ...$ is even. In this polynomial a_0, a_1, ..., a_m and m are nonnegative integers. A permutation polynomial can be utilized to construct an interleaver. For example, for the bit stream $\{0,1,2,3,4,5,6,7\}$ $(L = 8)$, an interleaved data based on permutation polynomial $P(x) = 2x^2 + x + 3$ is $\{3,6,5,0,7,2,1,4\}$.

It was confirmed that applying this interleaver, the bitstreams with weight $2m$ dominates the code performance. Therefore, the interleavers are designed in such a way that to remove effect of these weights. For the short bitstream lengths, proposed interleavers outperforms the semi-random interleaver. For the long bitstream lengths they have close performance with the semi-random interleavers.

2.7.2 Interleaver Based on Iterative Decoder Performance

Figure 2.11 shows an iterative turbo decoder structure. Each component decoder accomplishes decoding based on received information from the decoder input and the information decoded from alternative decoder in the previous iteration. Hokfelt $et.al$ showed that the information obtained from the component decoder, which is called extrinsic information, correlates with the channel information output [53, 54]. This correlation deteriorates on every iteration and causes the code to have insufficient

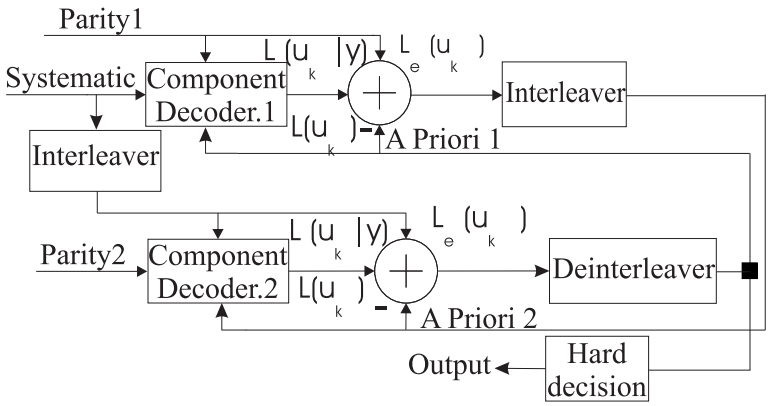

Figure 2.11 Iterative turbo decoder structure.

performance. In order to remove this drawback, an interleaver is constructed making a uniform correlation between extrinsic information and the channel information output on each iteration. Sloane $et.al$ in [83, 84] present a two stage interleaver. At the first stage, semi-random interleaver is designed such that it prohibits generation of the low weights for the code. Then an improvement is performed on it to guarantee existence of the uniform correlation between the extrinsic information and the received information from the channel. For some applications, where decoding speed is important, such as in optical and magnetic recording systems, [85] introduces an interleaver for parallel turbo decoding to prohibit data collisions between reading and writing process in component decoders.

2.8 Turbo Decoder

Turbo encoded data is conventionally decoded by iterative decoding techniques, as shown in Figure 2.11. In this technique, two component decoders are linked by an interleaver. Each component decoder provides soft output decoded information usable for the alternative component decoder at the next iteration. This recursive soft output information is called a-priori information [86]. In addition to the a-priori information, component decoders accept the systematic and parity information from the channel output. The obtained soft information from the component decoder output is subtracted from a-priori and systematic information to produce the extrinsic

information. The extrinsic information from each component decoder is interleaved or deinterleaved to create a-priori information for the alternative decoder at the next iteration step. At each iteration, the interleaver and deinterleaver rearrange the extrinsic information generating a new combination of soft information as a-priori information for other component decoders to provide decoded information having the maximum likelihood with the original bitstream. Generally, the component decoders are designed based on the Maximum A-Posteriori (MAP) algorithm [87], its logarithmic and optimum versions, i.e. Log-MAP and Max-Log-MAP, or Soft Output Viterbi Algorithm (SOVA). With more complexity, MAP and Log-MAP create better performance than SOVA. However, some modifications have been introduced to SOVA to improve its performance, while maintaining its low complexity design compared to two other methods. All of the above algorithms provide soft decoded information based on Log-Likelihood Ratios (LLR). The polarity of the LLR determines the sign of the decoded bit and its amplitude corresponds to the probability of a correct decision [40]. In this section, the concept of LLR and its application to the iterative turbo decoding by SOVA is explained. Finally, some methods to improve SOVA performance are presented.

2.8.1 Log-Likelihood Ratios

For a Binary Symmetric Channel (BSC), the LLR is defined as the logarithm of the ratio of probabilities of bit taking its two possible values [88]:

$$L(u_k) \triangleq ln\left(\frac{P(u_k=+1)}{P(u_k=-1)}\right) \tag{2.21}$$

where $P(u_k=+1)$ and $P(u_k=-1)$ are probabilities of the received bit 1 and -1, respectively. Taking exponent of both sides of this equation:

$$e^{L(u_k)} = \frac{P(u_k=+1)}{1-P(u_k=+1)}$$

Then

$$\begin{aligned} P(u_k=+1) &= \frac{e^{L(u_k)}}{1+e^{L(u_k)}} \\ &= \frac{e^{-L(u_k)/2}}{1+e^{-L(u_k)}}.e^{(L(u_k)/2)} \\ &= C^{(1)}_{L(u_k)}e^{(L(u_k/2))} \end{aligned} \tag{2.22}$$

where

$$C^{(1)}_{L(u_k)} = \frac{e^{-L(u_k)/2}}{1+e^{-L(u_k)}} \tag{2.23}$$

Due to the channel noise, the received information at the decoder input is different to the information transmitted from the encoder. Therefore, a conditional LLR is defined as follows [40]:

$$L(u_k|y) \triangleq ln\left(\frac{P(u_k=+1|y)}{P(u_k=-1|y)}\right) \tag{2.24}$$

where $P(u_k = \pm 1|y)$ is the probability of decoded u_k in terms of the received information y. Another definition of the conditional LLR can be presented by probability of the received signal y_k based on the transmitted information values as follows [40]:

$$L(u_k|y) \triangleq ln\left(\frac{P(y|x_k=+1)}{P(y|x_k=-1)}\right) \tag{2.25}$$

For the Gaussian channel noise with the binary modulation, probability of the received information y_k in terms of the transmitted information x_k is given by [40]:

$$P(y_k|x_k=+1) = \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{E_s}{2\sigma^2}(y_k-a)^2\right) \tag{2.26}$$

$$P(y_k|x_k=-1) = \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{E_s}{2\sigma^2}(y_k+a)^2\right) \tag{2.27}$$

where E_s is the transmitted energy per symbol, σ^2 is the noise variance and a is the fading amplitude, which is considered to have a value of 1 for the Gaussian channel. Substituting Equations 2.26 and 2.27 in Equation 2.25, the conditional LLR is given by [40]:

$$\begin{aligned} L(u_k|y) &\triangleq ln\left(\frac{P(y|x_k=+1)}{P(y|x_k=-1)}\right) \\ &= ln\left(\frac{exp(-\frac{E_s}{2\sigma^2}(y_k-a)^2)}{exp(-\frac{E_s}{2\sigma^2}(y_k+a)^2)}\right) \\ &= \left(-\frac{E_s}{2\sigma^2}(y_k-a)^2\right) - \left(-\frac{E_s}{2\sigma^2}(y_k+a)^2\right) \\ &= \frac{E_s}{2\sigma^2}4a.y_k \\ &= L_c y_k \end{aligned} \tag{2.28}$$

where $L_c = \frac{E_s}{2\sigma^2}4a$ is defined as the channel reliability value and only depends on the signal to noise ratio and the fading amplitude of the channel.

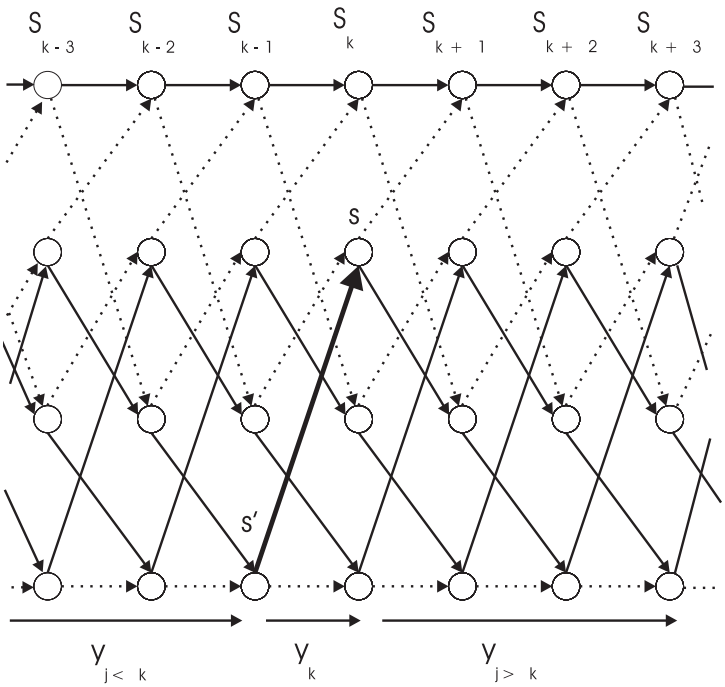

Figure 2.12 Trellis diagram of 4-state turbo code $(1, \frac{5}{7})$.

2.9 Soft Output Viterbi Algorithm

In order to apply the Viterbi algorithm to iterative turbo decoding, it is necessary to perform two modifications to the algorithm. Since every component decoder accepts the a-priori information from the alternative component decoder, the a-priori information on the alternative decoder output should be considered as the input of the component decoder in the next iteration [86, 89]. In addition, the output information of each component decoder should be produced in the soft form making it suitable as a-priori information for another component decoder.

2.9.1 Effect of the A-priori Information

Figure 2.12 shows the trellis diagram of the 4-state RSC code $(1, \frac{5}{7})$. In this figure the bold line is considered as a survivor path. The state of the survivor path at the time stage k (S_k) is determined by the appointed state of the survivor path at the time stage $k-1$ (S_{k-1}) and the transition path from the stage $k-1$ to k (S$'$ $\longrightarrow$S). Therefore,

the probability of the correctly received path at the state $S_k = s$ is obtained by:

$$P(S_k^s|y_{j\leq k}) = \frac{P(S_k^s \wedge y_{j\leq k})}{P(y_{j\leq k})} \tag{2.29}$$

where $P(y_{j\leq k})$ gives the probability of the received information for time instances before the time instance k. Since $P(y_{j\leq k})$ for all transition states up to the state k is equal, $P(S_k^s \wedge y_{j\leq k})$ maximizes the above equation. In fact, the value of this probability determines the survivor path of the trellis diagram for the state $S_k = s$ at the time instant k. For a memoryless channel, this probability can be extended as follows:

$$P(S_k^s \wedge y_{j\leq k}) = P(S_{k-1}^{s'} \wedge y_{j\leq k-1})P(S_k = s \wedge y_k|S_{k-1} = s') \tag{2.30}$$

Thus, the metric at the state $S_k = s$ is defined by:

$$\begin{aligned} M(S_k^s) &\triangleq ln\Big(P(S_k^s \wedge y_{j\leq k})\Big) && (2.31)\\ &= M(S_{k-1}^{s'}) + ln\Big(p(S_k = s \wedge y_k|S_{k-1} = s')\Big) \\ &= M(S_{k-1}^{s'}) + ln(\gamma(s', s)) && (2.32)\end{aligned}$$

where $\gamma(s', s)$ is the branch transition probability for the path from $S_{k-1} = s'$ to $S_k = s$. Based on Bayes rule, $\gamma(s', s)$ can be expressed as:

$$\begin{aligned} \gamma(s', s) &= P(y_k \wedge s|s') \\ &= P(y_k|s' \wedge s).P(s|s') \\ &= P(y_k|s' \wedge s).P(u_k) && (2.33)\end{aligned}$$

considering x_k as the transition information transmitted from the state $S_{k-1} = s'$ to the state $S_k = s$ as t_k, $P(y_k|s' \wedge s)$ is given for the memoryless channel by: [40]

$$P(y_k|s' \wedge s) \equiv P(y_k|x_k) = \prod_{l=1}^{n} P(y_{kl}|x_{kl}) \tag{2.34}$$

where y_{kl} and x_{kl} are l th bit of the transmitted and n is the number of these bits in each codeword y_k or x_k, respectively. For the Gaussian channel and binary modulation $P(y_{kl}|x_{kl})$ is given by:

$$P(y_{kl}|x_{kl}) = \frac{1}{\sqrt{2\pi}\sigma} exp\Big(\frac{-E_s}{2\sigma^2}(y_{kl} - ax_{kl})^2\Big) \tag{2.35}$$

Substituting Equation 2.35 in Equation 2.34 yields:

$$\begin{aligned} P(y_k|s^{'} \wedge s) &= \prod_{l=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} exp\Big(- \frac{E_s}{2\sigma^2}(y_{kl} - ax_{kl})^2 \Big) \\ &= \frac{1}{(\sqrt{2\pi}\sigma)^n} exp\Big(- \frac{E_s}{2\sigma^2} \sum_{l=1}^{n} (y_{kl} - ax_{kl})^2 \Big) \\ &= \frac{1}{(\sqrt{2\pi}\sigma)^n} exp\Big(- \frac{E_s}{2\sigma^2} \sum_{l=1}^{n} (y_{kl}^2 + a^2 x_{kl}^2 - 2ax_{kl}y_{kl}) \Big) \\ &= C_{y_k}^2 . C_{x_k}^{(3)} . exp\Big(\frac{E_s}{2\sigma^2} 2a \sum_{l=1}^{n} y_{kl}x_{kl} \Big) \end{aligned} \tag{2.36}$$

where

$$C_{y_k}^{(2)} = \frac{1}{(\sqrt{2\pi}\sigma)^n} . exp\Big(- \frac{E_s}{2\sigma^2} \sum_{l=1}^{n} y_{kl}^2 \Big) \tag{2.37}$$

and

$$\begin{aligned} C_{x_k}^{(3)} &= exp\Big(- \frac{E_s}{2\sigma^2} a^2 \sum_{l=1}^{n} x_{kl}^2) \\ &= exp(-\frac{E_s}{2\sigma^2} a^2 n \Big) \end{aligned} \tag{2.38}$$

$C_{y_k}^2$ and $C_{x_k}^{(3)}$ only depend on signal to noise ratio, fading amplitude of the channel and the length of the transmitted information. Based on the above equations, $\gamma(s', s)$ is represented as follows:

$$\begin{aligned} \gamma_k(s^{'}, s) &= P(y_k|s^{'} \wedge s) . P(u_k) \\ &= C . exp(u_k L_{u_k}/2) . exp\Big(\frac{E_s}{2\sigma^2} 2a \sum_{l=1}^{n} y_{kl}x_{kl} \Big) \\ &= C . exp(u_k L_{u_k}/2) . exp\Big(\frac{L_c}{2} \sum_{l=1}^{n} y_{kl}x_{kl} \Big) \end{aligned} \tag{2.39}$$

where C=$C_{L(u_k)}^{(1)} . C_{y_k}^2 . C_{x_k}^{(3)}$. Therefore

$$ln(\gamma(s^{'}, s)) \triangleq \Gamma_k(s^{'}, s) = \hat{C} + \frac{1}{2} u_k L(u_k) + \frac{L_c}{2} \sum_{l=1}^{n} y_{kl}x_{kl} \tag{2.40}$$

when the above equation is applied for the metric calculation of Equation 2.32, $\hat{C}$ is a constant value and can be omitted. Finally, the modified metric considering the effect of the a-priori information is expressed by:

$$M(S_k^s) = M(S_{k-1}^{s^{'}}) + \frac{1}{2} u_k L(u_k) + \frac{L_c}{2} \sum_{l=1}^{n} y_{kl}x_{kl} \tag{2.41}$$

2.9.2 Soft Decoded Information for the Viterbi Algorithm

As shown in Figure 2.12, every state of the trellis diagram consists of both survivor and discarded path. Therefore, the difference metrics of the survivor and discarded path for arbitrary state $S_k = s$, at the time stage k, is given by: [90]

$$\Delta_k^s = M(s_k^s) - M(\hat{s_k}^s) \geq 0 \tag{2.42}$$

where $M(s_k^s)$ and $M(\hat{s}_k^s)$ are the metric values of the survivor and discarded path, respectively. Probability of correctly selected path S_k^s at this point is calculated by:

$$P(correct\ decision\ at\ S_k = s) = \frac{P(s_k^s)}{P(s_k^s) + P(\hat{s_k}^s)} \tag{2.43}$$

Based on the metric definition in Equation 2.31

$$\begin{aligned} P(correct\ decision\ at\ S_k = s) &= \frac{e^{M(s_k^s)}}{e^{M(s_k^s)} + e^{M(\hat{s}_k^s)}} \\ &= \frac{e^{\Delta_k^s}}{1 + e^{\Delta_k^s}} \end{aligned} \tag{2.44}$$

Hence, LLR for the correct decision path is obtained by: [91]

$$\begin{aligned} L(correct\ decision\ at\ S_k = s) &= ln\Big(\frac{P(correct\ decision\ at\ S_k = s)}{1 - P(correct\ decision\ at\ S_k = s)}\Big) \\ &= \Delta_k^s \end{aligned} \tag{2.45}$$

In the Viterbi algorithm all surviving paths at the specific time instant (for example j), originated from the same path and the same point before j in the trellis. Hence, it is possible that the algorithm selects a discarded path, which will be merged with the survivor path after δ time stage, where δ is usually set to be five times the constraint length of the convolutional code [40, 86]. Figure 2.13 shows a discarded path in the trellis diagram of the 4-state turbo code $(1, \frac{5}{7})$, which is merged with the survivor path at the time instant $k + 5$. In order to calculate the LLR value of the algorithm, it is necessary to consider the effect of all discarded paths, which are a part of the survivor path and merged with the survivor path. For this purpose, the relevant bits of the discarded and survivor paths are compared with each other. If the bits were different, the log-likelihood ratio of a bit error at stage k is related to the metric difference of state k, i.e. Δ_k^l. Since there are $\delta + 1$ paths that possibly face with this problem, the maximum Log-Likelihood Ratio (LLR) value is obtained from the path

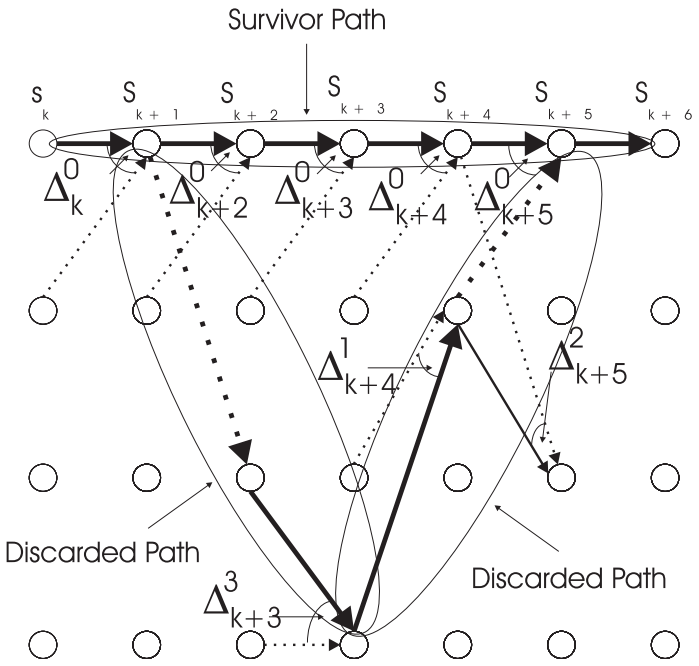

Figure 2.13 Description of SOVA for the simplified trellis diagram of the 4-state turbo code $(1, \frac{5}{7})$.

having the minimum metric difference. The LLR value of the overall error for the bit u_k is given by: [86]

$$L(u_k|y) \cong u_k \min_{\substack{i=k\ldots k+\delta \\ u_k \neq u_k^i}} \Delta_i^{s_i} \tag{2.46}$$

where u_k is the value of the bit given by ML path, and u_k^i is the value of this bit for the path which merged to the ML path and was discarded at the trellis stage i. The obtained soft LLR in equation (2.46) is called the extrinsic information. The extrinsic information with the above modification contributes with the a-priori information of another component decoder and the systematic information (y_{ks}) or its interleaved information, to form the information for the interleaver or deinterleaver input, i.e. $L_e(u_k)$, which is given by:

$$L_e(u_k) = L(u_k|y) - L(u_k) - L_c y_{ks} \tag{2.47}$$

The information obtained from interleaving/deinterleaving of $L_e(u_k)$ are called a-priori information and utilized as the input of alternative component decoder for the next iteration.

2.10 Improvement on the SOVA Performance

In despite of other proposed iterative decoding methods such as MAP and Log-MAP, SOVA does not produce soft output for all paths of the trellis. Therefore, a correlation between the extrinsic and intrinsic information exists [92]. This correlation is decreased by increasing the signal to noise ratio. A similar result is observed during iterative decoding of the received information in high iterations. In this case, after one or two iterations, the extrinsic information is much greater than the channel information, which voids effect of the channel information to the decoding process at the next iterations. In this section, some modifications on the extrinsic information obtained from each component decoder are proposed to maintain its correlation with the channel information. In addition, some modifications will be reviewed, which update the LLR values based on the decoding process in the trellis diagram.

2.10.1 Modification Based on Normalized Extrinsic Information

In order to maintain the correlation between extrinsic information and channel information in each iteration, a normalization is performed on the extrinsic information created at the each component decoder output. The normalization factor can be determined based on the channel characteristics. As mentioned before, the probability of the soft decoded data y_k for the transmitted information bits $u_k = +1$ is given by:

$$P(y_k|u_k = +1) = \frac{1}{\sqrt{2\pi}\sigma_y} exp\Big(\frac{-(y - m_y)^2}{2\sigma_y^2}\Big) \tag{2.48}$$

where $\sigma_y = \sqrt{E\{y^2\} - E\{y\}^2}$ $m_y = E\{y\}$ and $E\{v\}$ is the expectation of v. Therefore, the LLR value can be calculated by:

$$\begin{aligned} LLR = ln\Big(\frac{P(y_k|u_k = +1)}{P(y_k|u_k = -1)}\Big) &= ln\Big[e^{(\frac{-1}{2\sigma_y^2})[(y-m_y)^2-(y+m_y)^2]}\Big] \\ &= m_y \frac{2}{\sigma_y^2} y \end{aligned} \tag{2.49}$$

where $c = m_y \frac{2}{\sigma_y^2}$ represents the normalization factor that should be multiplied by the extrinsic information. Figure 2.14 shows the modified iterative turbo decoder, which utilizes the scaling block to normalize the extrinsic information obtained for each component decoder.

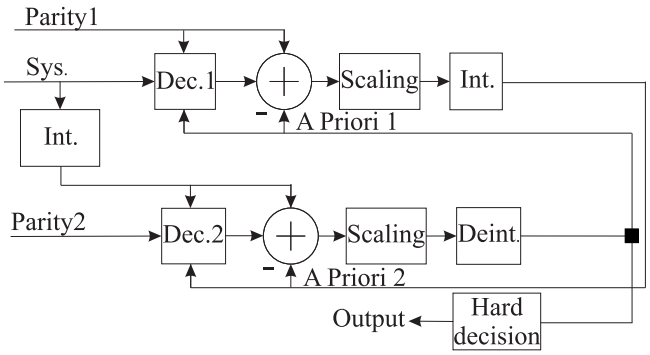

Figure 2.14 Improved iterative turbo decoder structure for SOVA.

Normalization of the extrinsic information can be conducted by different methods and [93] presents two such methods. In the first method a fixed scale factor is applied in the normalization procedure. The value of the scaling factor is increased when the BER value of the code decreases. Normally, the extrinsic information is normalized by a value between 0.5 and 1.0, which is determined by the trial and error. In another method, the extrinsic information is scaled by a constant value. This method increases the normalization factor in correspondence with increasing the rate of extrinsic information per iteration. The basic scale factor and the increment value in each iteration are determined by trial and error.

In comparison with [92], the second method proposed in [93] is implemented with a lower complexity. However, finding a suitable value for each iteration is required. Stirling and Gallacher in [94] present a new method which applies the scaling factor of Equation 2.49 for normalization of only one component decoder. Normalization of another component decoder is accomplished by a fixed value for all iterations. Conducted simulations confirm that with the low complexity in the decoder design, this approach has similar performance with the method presented in [93]. Application of the fixed scaling factor has also been verified for the two-step SOVA decoding [95]. Modifying the metric values with the fixed scaling factor as mentioned in [96] have been verified in [97]. Instead of updating the metric values, [98] applies a fixed scaling factor for the first iteration, while the scaling factor only increases at the last iteration. In comparison with the fixed scaling, this normalization improves the SOVA performance by 0.25 dB.

In [99], one method has been presented, which determines the scale factor based on a number of matched bits between signs of LLR value and the extrinsic information of every data block. For this purpose, hundreds of blocks are randomly selected and the number of matched bits within each block is counted. Depending on the obtained number of matched bits, the relevant scaling is calculated. For example, for the 4-state turbo code $(1, \frac{5}{7})$ with the row-column interleaver and the length $L = 4096$, the scaling factor has been computed using the following algorithm:

$$\begin{aligned} if Mb < 4000 \\ c \Longleftarrow 0.8 \\ else \\ c \Longleftarrow 0.8 + (Mb - 4000) * 0.0025 \end{aligned} \tag{2.50}$$

Where Mb and c represent the number of matched bits per block and the fixed scaling value, respectively. Similarly to this method, [100] presents a modification to the SOVA for the AWGN and fading channels.

Instead of applying a normalization factor, [101] considers a threshold value for the metric difference between the survivor and discarded path with the channel reliability $L_c = 1$, which is applied for the first few iterations of the decoding. The threshold values were determined by trial and error and should be powers of two to maximize the use of the available quantization level. However, [102] confirms that for the fixed L_c, different upper bounds should be considered for different signal to noise ratios. Moreover, when the real L_c value is employed, the greater threshold value than the threshold value considered for the decoding process with the fixed L_c value is obtained. Since in practice, L_c is considered to be equal to 1, [102] has proposed to apply two different upper bounds, one for the first few iterations and another for the later iterations. In comparison with the one upper bound, it slightly improves the code performance.

2.10.2 Modification on the LLR Value

As opposed to applying the scaling factor or threshold values, [103] updates the LLR value presented in Equation 2.46 improving the SOVA performance to be equivalent

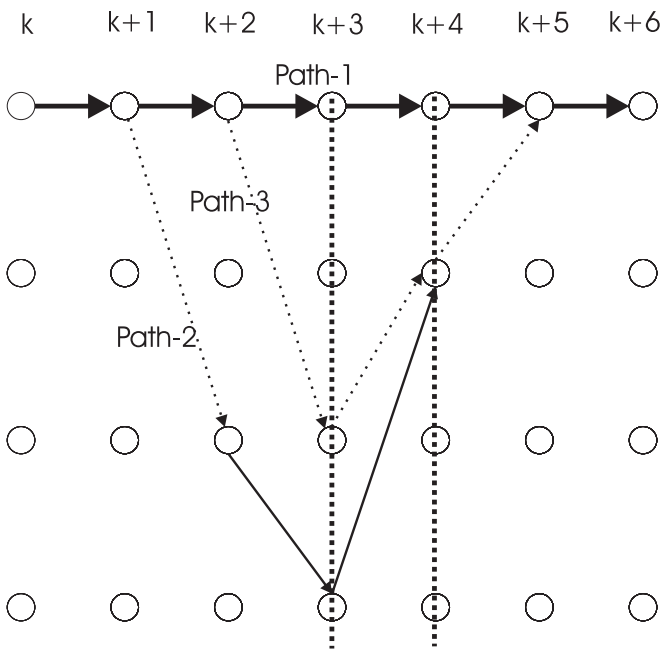

Figure 2.15 Example of possible case of path selection in decoding with SOVA

to the Max-Log-MAP algorithm.

Since SOVA only guarantees the Maximum Likelihood (ML) path, it is possible the best competitor path is not selected in the LLR calculation of Equation 2.46. This condition occurs when the best competitor path is discarded before it merges with the ML path. As verified in [104], the quality of the SOVA decoding in the backward direction is the same as with the forward direction decoding. Therefore, soft decoding can be performed in both directions for selection of the best competitor path. For example, in Figure 2.15 path-2 and path-3, respectively, are considered the best path for the ML path and the survivor path at the transition time from $k+3$ to $k+4$, which are merged with the Maximum Likelihood (ML) path. In the forward soft decoding, the decoder may not select path-3 as the best path, because it has been discarded before merging with the ML path, while it retains a chance to be selected the best path when decoding is conducted in the backward direction.

In the Bidirectional SOVA (Bi-SOVA) method, [104] the forward or backward decoding direction is determined by comparing the magnitude value of the metric differences in the forward and backward direction. If the relevant value of the backward

direction is less than the value calculated in the forward direction, it is substituted as a forward metric difference. Even if the best competitor path is not selected, the backward decoding direction can find another path with better quality than the path selected in the forward path [104]. Compared to the method proposed in [103], the Bi-SOVA has less complexity while it provides similar or even better performance than the Log-MAP decoding method.

2.11 Chapter Summary and Conclusions

In this chapter, structure of the turbo code has been reviewed. This type of code provides good performance at relatively low signal to noise ratios. However, in the medium to high signal ratios, which is called error floor region, the code can not reduce the error sufficiently. This is related to existence of a low free distance with high multiplicities for the code. Analysis of the code based on input bitstreams with different weights confirmed that self-terminating patterns have the major effect to the code performance. As one of the best solution to reduce the error floor effect, application of good interleavers was suggested. It was recognized that random interleavers provide better performance than the deterministic ones. However, some deterministic interleavers having similar performance to good random interleavers were presented.

The structure of iterative turbo decoder with SOVA was also reviewed. Some modifications to SOVA were presented, which improve its performance to be similar to other proposed iterative decoding methods.

Chapter 3

Iterative Turbo Decoder Design with Convolutional Interleavers

3.1 Introduction

Interleaving is known as an essential factor influencing performance of turbo codes. In most of designs, a turbo code is implemented as a block code when one block interleaver is used. Another interleaver family, are non-block interleavers, such as convolutional interleavers that have comparable delay with block interleavers, and a simplified implementation. In the case of convolutional interleavers, unlike block interleavers, one input data bit affects the interleaver more than once. Therefore, it is necessary to use continuous decoding methods for such a turbo code. In order to consider a turbo code with the convolutional interleaver as a block code, it is vital to return the interleaver memories to the known state by inserting stuff bits at the end of each block. Block codeword is then created, which makes it possible to implement a conventional iterative decoder that is known as a sub-optimum turbo decoder.

In this chapter, the structure of the best known non-block interleavers are briefly reviewed. Then, two different models of convolutional interleavers based on distribution of stuff bits in the interleaved data are proposed. Considering the interleaver properties, an algorithm is presented to compute the free distance value of the code. This algorithm is extended for calculating other low weights of the code. The analysis of the code based on calculated weight distribution is performed to achieve the

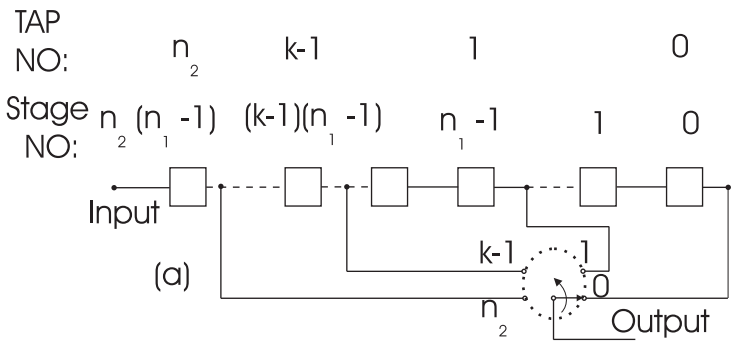

Figure 3.1 Ramsey type I interleavers.

suitable convolutional interleaver. Finally, for different bitstream lengths, the performance of convolutional interleavers are compared with the most conventional block interleavers.

3.2 Ramsey Interleavers

Early work on the non-block interleavers dates back to the 1970s [105]. Ramsey presented optimum interleavers that were designed based on the minimum possible interleaving delay and minimum overall number of memories applied in the interleaver and deinterleavers. In the following section, the structures of these interleavers are explained.

3.2.1 Ramsey Type I Interleaver

This interleaver is constructed by $[n_2(n_1-1)+1]$ shift-registers, which are separated by n_2 taps, where n_1 and n_2+1 are relatively prime and the taps positioning are started from the outermost register and allocated to every (n_1-1) register. The data are read from the tap in the reverse order of their distances from the input of the shift register. Figure 3.1 shows the general structure of the type I (n_2, n_1) interleaver. The minimum delay of the interleaver is equal to $n_2(n_1-1)$, where n_1 and n_2 are two positive integers satisfying $n_2 < n_1 < 2n_2$. The interleaved data of the input sequence $\{0,1,2,3,4,...\}$ from the interleaver $n_1=5$, $n_2=3$ is obtained as follows: $\{0,5,10,15,4,9,14,19,8,13,18,23,12,17,22,27,16,...\}$

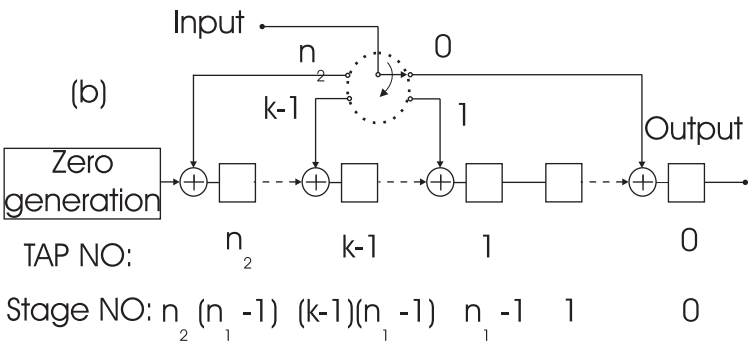

Figure 3.2 Ramsey type II interleavers.

3.2.2 Ramsey Type II Interleaver

for the type I Ramsey interleaver, a deinterleaver is defined to reorder the interleaved data. This device can also be considered as an interleaver, which is conventionally referred to as the type II Ramsey interleaver. Figure 3.2 shows the general structure of this interleaver. Similarly to the type I interleaver, n_2 and $n_1 + 1$ are relatively prime, where $n_2 > n_1 + 1$.

3.2.3 Ramsey Type III Interleaver

In this interleaver, the n_2 and n_1 parameters are relatively prime. It consists of $[(n_2 - 1)(n_1+1)+1]$ shift register with n_2 taps at every (n_1+1) registers. Figure 3.3 shows the general structure of this interleaver.

3.2.4 Ramsey Type IV Interleaver

This interleaver can be also considered as a deinterleaver for the interleaver type III. Again, n_1 and n_2 are relatively prime and the interleaver consists of $[(n_1 - 1)(n_2 + 1) + 1]$ registers. The taps are positioned at every $(n_2 + 1)$ register. Figure 3.4 shows the structure of type IV interleaver.

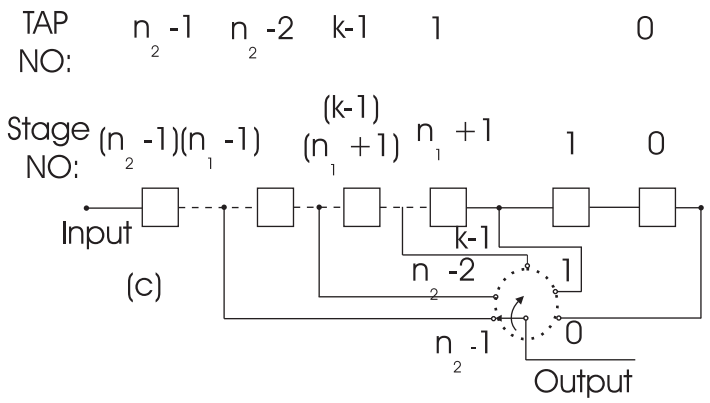

Figure 3.3 Ramsey type III interleavers.

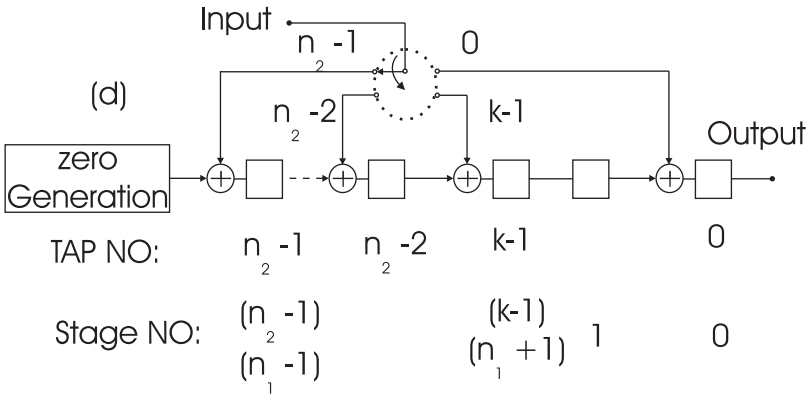

Figure 3.4 Ramsey type IV interleavers.

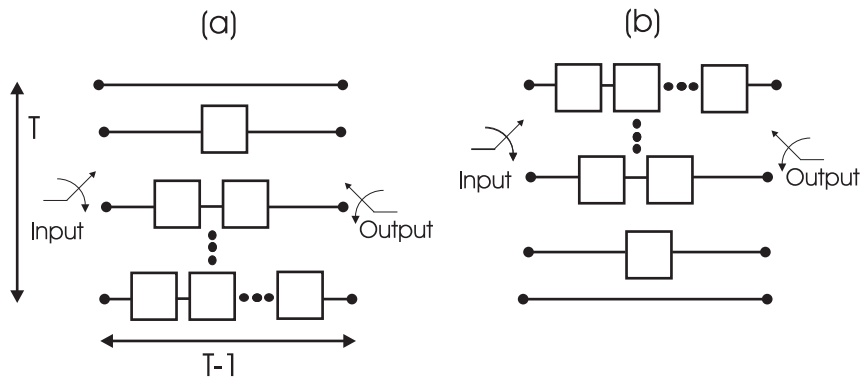

Figure 3.5 General structure of convolutional interlevears with period T and space value 1.

3.3 Convolutional Interleaver Structure

The structure of the convolutional interleaver is similar to the Ramsey Type I interleaver [43]. This interleaver is constructed by T parallel lines, which defines its period. Conventionally, each interleaver line has a different number of memories from the other lines. The difference in the numbers of memories between two adjacent interleaver lines is generally considered as a constant referred to as a space parameter of the interleaver [106, 107]. Figure 3.5 illustrates the general structure of convolutional interleavers and deinterleavers with period T and space value $M = 1$. The number of memories in each line of the deinterleaver is determined based on the number of memories applied at the corresponding interleaver line, such that distributed information in every corresponding interleaver and deinterleaver line will together pass through to the same number of memories. Also, the selectors applied for these structures have to synchronously operate with each other to properly recover the original data at the end of decoder. Based on the arithmetic sequence, the overall number of memories for the interleaver (T, M) is given by:

$$S = \sum_{i=1}^{T} s_i = M + 2M + ... + (T-1)M = \frac{T(T-1)M}{2} \tag{3.1}$$

A convolutional interleaver can be forced to operate as a block interleaver by inserting a number of zero stuff bits to its memories providing an interleaved data block which is isolated from the other blocks [13]. Initially, the interleaver memories are set

to zero values. Based on the number of applied memories in each line, bits distributed to the relevant line appear at different times. For example, the interleaved data block of an input bitstream with the length L=12 from the interleaver $(T = 3, M = 1)$ is given by: $\{x_0$,0,0,x_3,x_1,0,x_6,x_4,x_2,x_9,x_7,x_5, 0,x_{10},x_8,0,0,$x_{11}\}$.

Depending on the input sequence length and the interleaver parameters, distributed data are terminated at one of the interleaver lines, which is determined by the Rem(L,T) value, where $Rem(L, T)$ gives the reminder of $\frac{L}{T}$ operation. For an input bitstream $\{x_0$,x_1,x_2,...,$x_L\}$ with the length L, different interleaved data for the interleaver $(T = 3, M = 1)$ would be:

$Rem(L, T) = 0$:
$\{x_0, 0, 0, x_3, x_1, 0, x_6, ..., x_{L-7}, 0, x_{L-2}, x_{L-4}, 0, 0, x_{L-1}\}$
$Rem(L, T) = 1$:
$\{x_0, 0, 0, x_3, x_1, 0, x_6, x_4, ..., x_{L-3}, x_{L-5}, 0, 0, x_{L-2}\}$
$Rem(L, T) = 2$:
$\{x_0, 0, 0, x_3, x_1, 0, x_6, ..., x_{L-4}, x_{L-6}, 0, x_{L-1}, x_{L-3}\}$

When a convolutional interleaver with the above structure is applied in the turbo code structure, inserted stuff bits to the interleaver memories reduce channel bandwidth usage. Therefore, an optimization can be performed on the interleaver to control the number of those bits equal to the number of applied memories. For this purpose, one block is added after the interleaver controlling the data at the interleaver output in order to delete extra zero stuff bits that appear at the end part of the interleaver [13]. In this case, the memory contents at the end of each block have zero value. This remains until the beginning of the next block. Following the previous example, the optimized interleaver outputs for different $Rem(L, T)$ values are given by :
$Rem(L, T) = 0$:
$\{x_0, 0, 0, x_3, x_1, 0, x_6, ..., x_{L-2}, x_{L-4}, x_{L-1}\}$
$Rem(L, T) = 1$:
$\{x_0, 0, 0, x_3, x_1, 0, x_6, x_4, ..., x_{L-3}, x_{L-5}, x_{L-2}\}$
$Rem(L, T) = 2$:
$\{x_0, 0, 0, x_3, x_1, 0, x_6, ..., x_{L-4}, x_{L-6}, x_{L-1}, x_{L-3}\}$

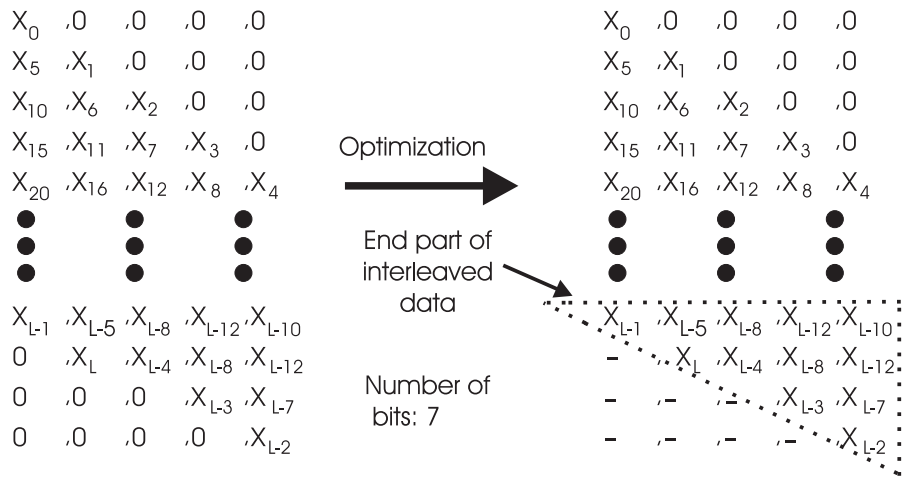

Figure 3.6 Interleaved data for an interleaver $(T = 5, M = 1)$, $Rem(L, T) = 2$ with non-optimized and optimized interleavers.

Figure 3.6 shows the non-optimized and optimized interleaved data of the interleaver $(T = 5, M = 1)$ with $Rem(L, T) = 2$ and length L.

Based on this property, for every interleaver, input bitstreams into T different groups are categorized, such that each group includes bitstreams with different lengths having the same $Rem(L, T)$ value. Therefore, in each group, one part of an interleaved data with a higher length is common with the interleaved data with the short length. In addition, the interleaved data with a higher length consists of an extra part, which is created due to the increasing of the length. Figure 3.7 shows these mentioned parts for two bitstreams with the length $L = 20$ and $L = 24$ considered as one group for the interleaver $(T = 4, M = 1)$ which gives the $Rem(L, T) = 0$ value.

3.4 Iterative Turbo Decoding with Convolutional Interleavers

A turbo code with the discussed convolutional interleaver structure can be analyzed as a block code if termination methods are applied to the constituent RSC encoders.

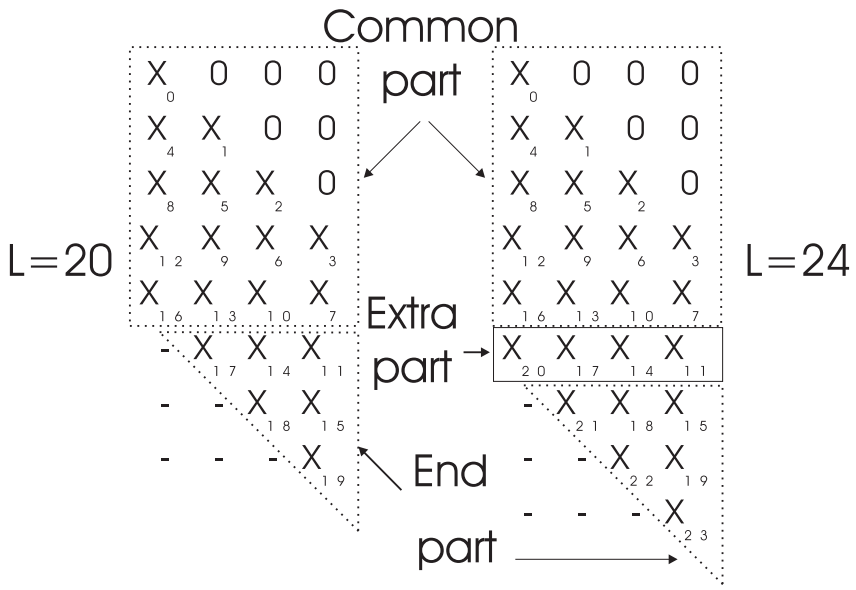

Figure 3.7 Comparison of different parts of interleaved data at the output of the interleaver with different lengths, similar period and $Rem(L,T)$ values,i.e. $T = 4$, $Rem(20,4) = 0$, $Rem(24,4) = 0$.

In our work, only the memory contents of the first RSC encoder are returned to the zero state, while the memories state of the second RSC encoder is maintained at the end of each data block. When stuff bits are inserted into the convolutional interleaver after trellis termination of the first RSC encoder they do not have any effect on the weight of systematic and the first parity data. Thus, they can be eliminated from the end part of the mentioned data, which reduces the overall number of stuff bits to $\frac{T(T-1)M}{2}$. This property makes it possible to employ one of the conventional iterative techniques for turbo decoding. Since the second parity data have a different length from the encoded systematic and the first parity data, relevant modification should be performed on the iterative turbo decoder in the proposed structure of turbo codes [14]. Figure 3.8(a) shows the structure of the iterative decoder for the convolutional interleaver with the non-optimized convolutional interleaver together with the information length in different positions of the decoder, which are shown in parentheses. In the first component decoder where the systematic and the first parity information are used, the extrinsic information length is L. When the systematic and the extrinsic information pass through the interleaver, their lengths, equal to the total

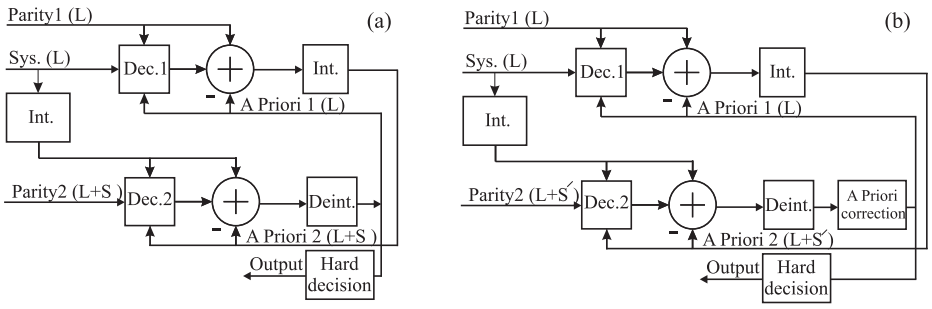

Figure 3.8 Iterative turbo decoder structure with a) the non-optimized convolutional interleaver b) the optimized convolutional interleaver.

number of the interleaver memories, i.e. S bits, will be increased. Therefore, the a-priori information length for the second decoder would be $L + S$, which is equal to the length of the second parity information. By passing the obtained extrinsic information through the deinterleaver, which inverts the action of the interleaver, the a-priori information length for the first component decoder will be changed to L.

The proposed decoder can be used for turbo codes with the optimized convolutional interleaver [14]. In this case, the interleaver adds S' stuff bits ($S' < S$) equal to $\frac{T(T-1)M}{2}$ value to the systematic and extrinsic information length of the first decoder, whose length will be equal to the length of the second parity data. Furthermore, the a-priori correction should be performed after deinterleaving in order to remove the added stuff bits from the extrinsic information obtained from the second decoder, and to generate correct information of length L, compatible with the conducted optimization at the interleaver. Figure 3.8(b) shows structure of the modified decoder for the optimized convolutional interleaver.

3.5 Weight Distribution of Turbo Codes using Convolutional Interleavers

Analysis of the block-wise turbo code performance with the convolutional interleaver and maximum likelihood iterative decoding is accomplished for a linear block code by the determination of its upper bound from Equation 2.10.

As mentioned in the previous chapter, the error floor phenomenon of the turbo code is directly related to the low weight values. Thus, determining a low weight distribution of the code would be helpful to verify the code behavior at the error floor region. The main issue of this calculation is influence of the interleaver behavior on the turbo code performance. Some algorithms have been proposed, based on mainly determination of the free distance value of the code.

3.5.1 Free Distance Computation of Turbo Codes

Conventionally, as mentioned in the previous chapter, the free distance (d_{free}) is defined as the minimum Hamming distance between any possible codewords. Since considering all input bit streams in the determination of d_{free} is impossible, especially for medium to large input block lengths, finding a suitable algorithm to compute the d_{free} for turbo codes has been followed in previous works.

In some work, effective free distance of turbo code has been calculated based on the weight-2 distribution of the code, which gives a good approximation for the performance of multiple turbo codes [108–111].

Another algorithm is devised based on the generator matrix properties of the turbo code to determine its effective free distance value. This method can be used for the performance of turbo code when both RSC codes are terminated [112]. A different algorithm is presented based on the trellis termination and truncation for the first and the second RSC encoders. It considers all possible self-terminating input bitstreams of the first RSC constituent encoder with the required length, and computes the weight of the code [10]. It is obviously observed that the complexity of the algorithm increases with increasing the interleaver length. Since the algorithm only computes the free distance value, which is related to the first term of the Equation 2.10, it is useful for turbo codes having a free distance with high multiplicities that the effect of other terms on the code performance is reduced.

A general algorithm which computes low weight terms of the code with a moderate length is presented in [113, 114]. The algorithm defines a constrained subcode based on the trellis structure of the code, and computes the minimum distance of a con-

Table 3.1
Patterns returning the RSC encoder $(1, \frac{5}{7})$ to the zero state for the convolutional interleaver $(T = 5, M = 1)$, $Rem(L, T) = 2$.

x_j	x_{j+1}	x_{j+2}	x_{j+3}	x_{j+4}	x_{j+5}	x_{j+6}
1	0	0	0	0	0	1
1	0	0	0	1	1	0
1	1	0	0	0	1	0
1	1	0	0	1	0	1
1	0	1	0	0	1	1
1	1	0	1	1	0	0
1	0	1	1	0	1	0
1	0	0	0	1	1	0
1	0	0	1	0	0	0
1	1	1	0	0	0	0

strained subcode relevant to the interleaved data to determine the free distance value of the code. Some improvements to this algorithm, reducing computation complexity are proposed in [115, 116], while [117] and [118] present alternative algorithms for some special interleavers.

3.5.1.1 Free Distance Computation of Turbo Codes with Convolutional Interleavers

For the turbo codes with optimized convolutional interleavers, the interleaver property is utilized to determine low weights of the code. The applied algorithm, similar to the previously proposed methods, considers low weight self-terminating input bitstreams that generate low weight codewords [15]. In the optimized convolutional interleaver, the minimum distance between two adjacent bits, before and after interleaving, is equal to the interleaver period, except for the end part of the interleaved data block, where it decreases due to zero bit deletion. Therefore, it is expected that the data interleaved in this part will create a lower weight than the data interleaved by other parts in the second RSC encoder. Using this interleaver structure and considering the effect of tail bits on the overall weight of the turbo code, an algorithm to estimate free distance value (d_{free}) is presented as follows [15]:

Firstly, among all input data block streams with the minimum weight (i.e. 1) those

Table 3.2
Free distance specifications for turbo codes $(1, \frac{5}{7})$ and $(1, \frac{35}{23})$ with interleavers $(T = 10, M = 1, L = 512)$ and $(T = 20, M = 1, L = 1024)$.

Turbo code	T	L	d_{free}	N_{free}	$\tilde{w}_{free}$
$(1, \frac{5}{7})$	10	512	10	3	3
$(1, \frac{5}{7})$	20	1024	10	3	3
$(1, \frac{35}{23})$	10	512	11	1	3
$(1, \frac{35}{23})$	20	1024	16	1	3

which return the first RSC encoder to the zero state are selected. Then, their bit 1 positions are compared with the bit positions that have been located at the end part of the interleaved data. If any pattern returns the first RSC encoder to the zero state and positions of its bits are in the end part of the interleaved data, then the overall weight of the corresponding codeword is computed and stored as a d_{free} value. In order to identify the other input bitstreams with the mentioned property, the same comparison is performed and the minimum obtained d_{free} value is considered as d_{free} at the end of the first step. A similar procedure is followed for higher input data weights until the computed d_{free} is lower than or equal to the weight of the input bitstream. The final d_{free} is assumed to be d_{free} of the turbo code.

Since bit 1 positions should be located at the end part of the interleaver, the pattern length consisting of all the 1s inside the bitstream , should not exceed number of bits at the end part of the interleaver. For this purpose, low weight patterns with a length equal to the number of bits at the end part of the interleaver are encoded returning the RSC encoder to the zero state.

For example, as shown in Figure 3.6 , the number of bits at the end of the interleaver with $(T = 5, M = 1)$, $Rem(L, T) = 5$ is equal to 7 and the patterns with the length 7 that return the RSC encoder $(1, \frac{5}{7})$ to the zero state have been listed in Table 3.1. The weight of these patterns is not greater than 3. The algorithm covers all the patterns shifted cyclically that satisfy the above condition. The computed d_{free} values of turbo codes $(1, \frac{5}{7})(1, \frac{35}{23})$ for convolutional interleavers with different lengths have been presented in Table 3.2, where N_{free} and $\tilde{w}_{free}$ represent the total number of multiplicities of the codewords with weight d_{free} and the average input data weight

Table 3.3
Weight distribution for turbo codes $(1, \frac{5}{7})$ and $(1, \frac{35}{23})$ at the end part part of the interleaver with $(T = 10, M = 1, L = 1024)$ and $Rem(L, T) = 2$.

Weight	Turbo code$(1, \frac{5}{7})$		Turbo code$(1, \frac{35}{23})$	
d	N_d	$\tilde{w}_d$	N_d	$\tilde{w}_d$
10	3	3.0	0	0
11	1	2.0	1	3.0
12	1	4.0	0	0
13	3	2.3	0	0
14	11	3.3	0	0
15	7	2.9	7	2.86
16	7	2.7	1	3.0
17	15	3.01	3	2.67
18	19	3.01	8	2.63
19	33	3.4	7	2.86
20	44	3.55	4	2.75
21	56	3.37	4	2.75
22	112	3.77	16	3.0
23	77	3.68	10	2.7
24	118	3.8	12	3.0
25	118	3.58	30	2.94
26	208	3.67	21	2.8

related to d_{free}, respectively. The results show that for the 4-state turbo code, increasing the period and length does not affect the free distance specifications, while for the 16-state code the free distance has been increased by 5 units and the multiplicity has been preserved. This behavior of the convolutional interleaver is fully discussed in the next section.

In comparison with most block interleavers, the convolutional interleaver generates d_{free} with fewer multiplicities, which can be considered as an advantage. Thus, the upper bound of the code is not dominated by its free distance value, and it is necessary to determine other codewords with low weights or codewords having relatively high multiplicities, which affect the code performance at the error floor region.

Depending on the interleaver period, the number of bits which are located at the end part of the interleaver change. In order for the algorithm to be usable for different interleaver periods, the area of the end part of the interleaver can be increased or

Table 3.4
Weight distribution for turbo codes $(1, \frac{5}{7})$ and $(1, \frac{35}{23})$ at the end part part of the interleaver with $(T = 20, M = 1, L = 1024)$ and $Rem(L, T) = 4$.

Weight	Turbo code$(1, \frac{5}{7})$		Turbo code$(1, \frac{35}{23})$	
d	N_d	$\tilde{w}_d$	N_d	$\tilde{w}_d$
10	3	3.0	0	0
11	0	0	0	0
12	0	0	0	0
13	1	3	0	0
14	4	2.75	0	0
15	0	0	0	0
16	4	2.5	1	3
17	1	3.0	0	0
18	7	2.58	0	0
19	3	2.67	2	3
20	78	3.82	3	3
21	11	2.9	1	3
22	33	3.18	1	3
23	20	3	1	3
24	91	3.49	1	3
25	38	3.2	3	2.67
26	63	3.3	6	3

decreased based on the interleaver period.

By increasing the area, more bits are involved in the calculation. This requires to consider the input bitstreams with the lower weights. Instead, shortening the area makes it possible to involve input bitstreams with the higher weights in the weight calculation of the code [15]. Tables 3.3 and 3.4 give some low weights of the 4-state $(1, \frac{5}{7})$ and 16-state turbo code $(1, \frac{35}{23})$, which have been calculated from the defined end part of the interleaver with length $L = 1024$ and input bitstream with weights no greater than 5 and 4, respectively.

3.5.2 Extrapolated Weight Distribution Computation Algorithm

Apart from the end part of the optimized interleaver, it is possible to find other input bitstreams having weight(s) outside the mentioned area producing weights that are affecting the code performance.

Table 3.5 Returning to zero patterns with weight-2 and 3 for 4-state turbo code $(1, \frac{5}{7})$.

Input Weight	Pattern	output Weight
2	$1\underbrace{0\ldots0}_{d=3k+2}1$ $k=0,1,2,\ldots,\lfloor(L-4)/3\rfloor$	$\frac{2d+8}{3}$
3	$11\underbrace{0\ldots0}_{d=3k}1$ $k=0,1,2,\ldots,\lfloor(L-3)/3\rfloor$	$2\left(\frac{d}{3}\right)+1$
3	$1\underbrace{0\ldots0}_{d=3k}11$ $k=0,1,2,\ldots,\lfloor(L-3)/3\rfloor$	$2\left(\frac{d}{3}\right)+1$
3	$1\underbrace{0\ldots0}_{d=3k+1}1\underbrace{0\ldots0}_{d'=3k'+1}1$ $k=0,1,\ldots,\lfloor(L-4-d')/3\rfloor$ $k'=0,1,\ldots,\lfloor(L-4-d)/3\rfloor$ $d+d'\leqslant L-3$	$\frac{2(d'+d)+8}{3}$
3	1 1 1	2

Similarly to the proposed algorithm in the previous section, low weights of the code from the input bitstreams that return the RSC constituent encoders to the zero state are determined. In order to reduce the complexity of the algorithm, especially for medium to high interleaver lengths, a new method to simplify computing weight distribution of the turbo code are presented, which is based on the convolutional interleaver properties [16].

Without considering the end part of the interleaved data, the distance between distributed bits in adjacent interleaver lines is always fixed by the product of T and M. Due to the deterministic behavior of the convolutional interleaver, it is possible to obtain self-terminating patterns for the second RSC encoder that have been interleaved from other or similar input self-terminating patterns. Therefore, it is possible to obtain patterns that return both RSC encoders to the zero state. Examining patterns with weight-2 presented in Table 3.5, for the 4-state turbo code with the convolutional interleaver $(T = 4, M = 1)$, it is found that input bitstream $(0001000000001000..0)_L$ with length L gives the interleaved data pattern $(000000000000100100...0)_L$, which returns the second RSC encoder to the zero state. The same happens for cyclical shifts of the original patterns presented in Table 3.5. Existence of these patterns

will create low weight codewords with high multiplicities and their effect should be considered in the upper bound computation of Equation 2.10 [16].

It is obvious that a number of input bitstreams satisfying the mentioned condition will increase with increasing the interleaver length. Therefore, for medium to high interleaver lengths, it would be difficult to consider all of these patterns in the weight calculation of the code. In order to overcome this issue, the following properties of the interleaver are utilized: [21]

When the full weights of a self-terminating pattern are positioned at the common part of two interleaved data sets categorized as one group, increasing the length of patterns will not affect the weight increment of the second RSC code and therefore, both interleavers produce a weight with similar multiplicity. Otherwise, i.e. when some, or all, weights of this self-terminating pattern are positioned at the extra part of an interleaver with a higher length, multiplicity of the weight is progressively increased proportional to the length difference of the two interleavers.

Additionally, in optimized interleavers, the distance of bits located in the end part of the interleaved data from the last bit of data block is constant. For example, in Figure 3.7 the distance between x_{21} and x_{23} for the interleaver of length $L = 24$ would be 4, which is the same as the equivalent bits for the interleaver of length $L = 20$, i.e. between x_{17} and x_{19} [16]. Therefore, the weights obtained from the end part of the interleaver would be independent of the interleaver length. In this case, weights obtained from the end part of the interleaved data with short length can be utilized as weights of a code with higher interleaver length [15].

Based on the above properties of these interleavers, it is possible to able to estimate the weight distribution of a turbo code with a desired length from an interleaver with shorter length. In this method, according to the interleaver period, the minimum applied interleaved data length is varied, and is equal to $T(T-1)M$, i.e. when all the interleaver memories have valid data. For interleavers $(T = 5, M = 1)$ and $(T = 4, M = 1)$ with the presented specifications in Figure 3.6 and 3.7, the minimum length is 12 and 6, respectively. Considering self-terminating patterns with the weight i, the algorithm computes $W_L^{(i)}$=$(w_{L_1}^{(i)}, w_{L_2}^{(i)}, ..., w_{L_k}^{(i)})$ with the multiplicity

Table 3.6
Weight distribution of 4-state turbo code for two input bitstreams $(100100...0)_L$,$(11100...0)_L$ and their cyclical shifts for the interleaver (T=10,M=1) with different lengths and identical $Rem(L,T) = 2$ value.

Pattern	$(100100...0)_L$			$(11100...0)_L$		
L	92	102	512	92	102	512
ω	N_ω	N_ω	N_ω	N_ω	N_ω	N_ω
10	0	0	0	3	3	3
11	1	1	1	0	0	0
12	0	0	0	0	0	0
13	2	2	2	1	1	1
14	1	1	1	2	2	2
15	2	2	2	0	0	0
16	3	3	3	1	1	1
17	2	2	2	1	1	1
18	5	5	5	2	2	2
19	2	2	2	0	0	0
20	5	5	5	2	2	2
21	6	6	6	37	45	373
22	7	7	7	2	2	2
23	4	4	4	0	0	0
24	1	1	1	0	0	0
25	9	9	9	1	1	1
26	46	53	340	2	2	2
27	2	2	2	1	1	1
28	5	5	5	2	2	2
29	9	9	9	0	0	0
30	46	53	340	2	2	2

$N_L^{(i)}$=$(n_{L_1}^{(i)}, n_{L_2}^{(i)}, ..., n_{L_k}^{(i)})$ as weight specifications of the turbo code for an interleaver (T, M) with length L. This is accomplished as follows : [16]

1) Design an interleaver (T, M) with the shortest length L_1 which satisfies $T(T-1)M \leq L_1 < L - T$ and $Rem(L_1, T) = Rem(L, T)$.

2) Compute the weight distribution of the code for the interleaver (T, M, L_1) from all self-terminating patterns with the weight i, as $W_{L_1}^{(i)}$=$(w_{L_{11}}^{(i)}, w_{L_{12}}^{(i)}, ..., w_{L_{1k}}^{(i)})$ and $N_{L_1}^{(i)}$=$(n_{L_{11}}^{(i)}, n_{L_{12}}^{(i)}, ..., n_{L_{1k}}^{(i)})$.

3) Increase the interleaver length T units $(L_2 = L_1 + T)$ and compute the weight

Table 3.7
Weight distribution of 4-state turbo code for two input bitstreams $(100100...0)_L$, $(11100...0)_L$ and their cyclical shifts for the interleaver $(T = 10, M = 1)$ with different lengths and identical $Rem(L, T)$ value.

Pattern	$(100100...0)_L$			$(11100...0)_L$		
L	96	106	256	96	106	256
ω	N_ω	N_ω	N_ω	N_ω	N_ω	N_ω
10	0	0	0	3	3	3
11	1	1	1	0	0	0
12	0	0	0	0	0	0
13	2	2	2	0	0	0
14	1	1	1	1	1	1
15	3	3	3	1	1	1
16	2	2	2	1	1	1
17	0	0	0	1	1	1
18	2	2	2	2	2	2
19	3	3	3	1	1	1
20	3	3	3	0	0	0
21	2	2	2	36	43	148
22	11	11	11	1	1	1
23	4	4	4	1	1	1
24	6	6	6	2	2	2
25	5	5	5	1	1	1
26	55	62	167	2	2	2
27	8	8	8	1	1	1
28	9	9	9	2	2	2
29	6	6	6	1	1	1
30	36	43	148	1	1	1

distribution specification of the code for the new interleaver length as $W_{L_2}^{(i)}$ and $N_{L_2}^{(i)}$: $W_{L_2}^{(i)}$=$(w_{L_{21}}^{(i)}, w_{L_{22}}^{(i)}, ..., w_{L_{2k}}^{(i)})$ $N_{L_2}^{(i)}$=$(n_{L_{21}}^{(i)}, n_{L_{22}}^{(i)}, ..., n_{L_{2k}}^{(i)})$.

4) If $n_{L_{1j}}^{(i)} = n_{L_{2j}}^{(i)}$ then $n_{L_j}^{(i)} = n_{L_{1j}}^{(i)}$ $(j = 1, 2, ..., k)$ else $n_{L_j}^{(i)} = n_{L_{1j}}^{(i)} + \frac{L-L_1}{T}(n_{L_{2j}}^{(i)} - n_{L_{1j}}^{(i)})$.

5) $W_L^{(i)}$=$W_{L_1}^{(i)}$=$W_{L_2}^{(i)}$=$(w_{L_1}^{(i)}, w_{L_2}^{(i)}, ..., w_{L_k}^{(i)})$ $N_L^{(i)}$=$(n_{L_1}^{(i)}, n_{L_2}^{(i)}, ..., n_{L_k}^{(i)})$.

Tables 3.6 and 3.7 give the calculated low weights of the 4-state turbo code $(1, \frac{5}{7})$ for the specified input bitstreams and an interleaver $(T = 10, M = 1)$ with minimum valid lengths $L = 92$, $L = 102$, $L = 96$ and $L = 106$. The weights obtained

Table 3.8
Combined self-terminating pattern 100011 with other self-terminating patterns of Table 3.5.

Weight	Pattern
5	10001100...01001 (d″ d″=0,...,L-10)
6	10001100...0100011 (d″ d″=0,...,L-12)
6	10001100...0110001 (d″ d″=0,...,L-12)
6	10001100...0101010 (d″ d″=0,...,L-12)
6	10001100...0111 (d″ d″=0,...,L-9)

from these lengths have been utilized to determine the weight of the code for an interleaver length $L = 512$ and $L = 256$. For both interleavers, the results show that the code for some weights $\omega = 26$, has high multiplicities related to the patterns that simultaneously return both RSC encoders to the zero state.

The above calculations indicate that multiplicities of some weights will remain constant for different interleaver lengths. These weights are mainly produced by input bitstreams with weights located in the end part of the interleaved data. In this algorithm, the minimum length of the interleaver is increased by increasing the required length of the interleaver. This may be impossible when the weight of the self-terminating pattern increases. In this case, the algorithm involves those self-terminating patterns having higher possibility to generate low weight codewords. They are specified when enough patterns constructed by the minimum number of zeros between bits 1 are considered.

3.5.2.1 Weight Distribution from Higher Input Bitstreams Weights

It is possible to combine low weight self-terminating patterns with each other making other self-terminating patterns with a higher weight. This is accomplished by separating low-weight self-terminating patterns from each other with a number of zero bits. Due to the increased weight, the number of new self-terminating patterns is increased in such a way that makes it impossible to consider all of them in the

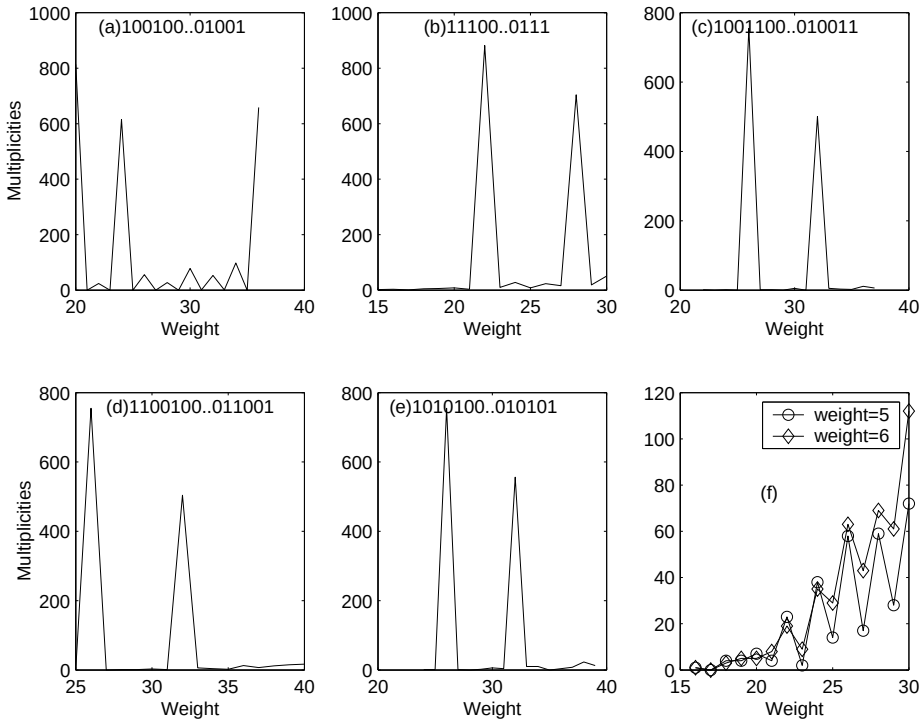

Figure 3.9 Weight distribution of 4-state turbo code $(1, \frac{5}{7})$ with the combined input bit-streams of Table 3.5.

weight distribution of the code, even for the short interleaver lengths.

In order to get a codeword with low weight, those self-terminating patterns that generate low weights are combined. This leads to apply self-terminating patterns with the minimum acceptable number of zeros between 1s. The weight distribution of the code from these combined patterns can be calculated by the algorithm presented in the previous section. Table 3.8 gives combinations of the (100011) pattern with other presented patterns of Table 3.5. The weight of the 4-state turbo code $(1, \frac{5}{7})$ applying the combined patterns with weights 4, 5 and 6 has been calculated and illustrated in Figure 3.9. Figures 3.9(a) to 3.9(e) show combination of identical patterns, while Figure 3.9(f) gives weight distribution of the code due to combining different patterns from Table 3.5. The results indicate that weights with high multiplicities is created by identically combined low weight self-terminating patterns, while combinations of different low-weight self-terminating patterns give weights with low multiplicities. Hence their effects on the performance of the proposed code

can be neglected. In conducted calculations, distance between two low weight self-terminating patterns in every combined pattern (d'') does not exceed 145.

3.5.2.2 Effect of Tail Bits on the Weight Distribution of Code

The algorithm should consider self-terminating patterns that are produced due to the effect of tail bits of the first RSC encoder. Tail bits are always located in the last part of a data block and therefore after interleaving, they automatically will be positioned in the end part of the interleaved data. The number of patterns which satisfy this condition will be increased by increasing the length of the interleaver and the constraint length of the RSC codes. Since the algorithm concentrates on patterns that possibly generate low weights for the code, bits 1 of the pattern should be close to each other and positioned at the end part of the input bitstream, because when this condition is not satisfied, the first RSC encoder provides higher weight, which consequently increases the weight of the code. Since, for different interleaver lengths and identical $Rem(L, T)$ value, the number of bits located in the end part of the interleaved data is constant, the explained technique for weight calculation of the code can also be applied for this purpose [16].

3.5.3 Simulation Results

The upper bound of BER for turbo codes with different interleavers has been computed based on weights obtained with the proposed algorithm and compared with the simulation results. The applied interleavers are designed in such a way that reduce the effect of stuff bits on the code performance. As mentioned before, since stuff bits are inserted into the interleaver memories after trellis termination of the first RSC encoder, they do not have any effect to the weight of the systematic and the first parity data and can be eliminated from these data parts. For simplicity in the simulations, in order to make equalized systematic and the parity data, the length effect of stuff bits is considered for the systematic and the first parity data as well as the second parity data. This means that increasing the length does not change the overall rate of the code. At the iterative decoder, decoding is accomplished based on presented model in Figure 3.8(b). In the iterative turbo decoder, the stuff bits inserted to the interleaver have value -1. In the weight calculation, the self-terminating patterns of

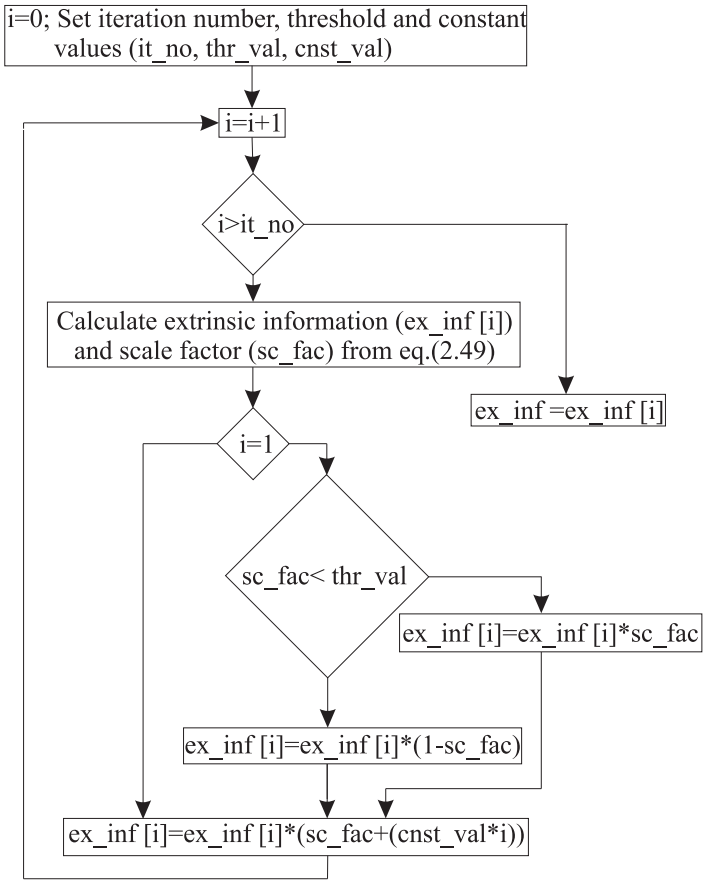

Figure 3.10 Scale factor computation algorithm applied for the SOVA.

the first RSC encoder with weights no greater than 4 have been considered. The information received from the AWGN channel is decoded by SOVA. The scale factor applied for the a-priori information of this iterative decoding method is conducted by the following algorithm.

First, the scale factor value from Equation 2.49 is calculated. Under ideal conditions, i.e. in a noiseless channel, the scale factor value is 1, while in noisy channels this value is less than 1 and is related to the received information [92]. Therefore, the threshold value is defined such that when the scale factor is less than the appointed threshold, it will be shifted to a value close to 1. This generates a suitable factor for the extrinsic information. Otherwise, i.e. when the scale factor is within the acceptable range, it is simply multiplied by the extrinsic information. Based on the simulation results, this modification has not have a visible effect on the performance

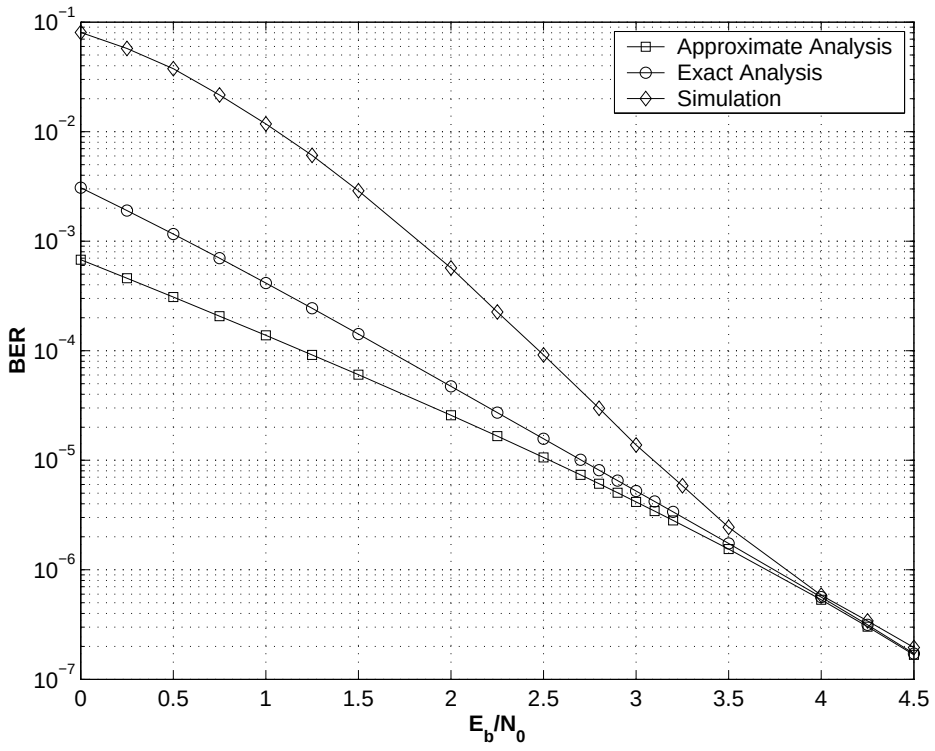

Figure 3.11 Analysis and simulation results of the 4- state turbo code $(1, \frac{5}{7})$ with the interleaver (T=10, M=1) and length L=512.

of decoder for higher iterations. Therefore, and in order to decrease decoding delay, the procedure is applied only at the first iteration. For higher iterations, a constant value increasing per iteration is added to the scale factor. With good approximation the above algorithm can be implemented with threshold and constant values fixed for different codes with different interleaver lengths. Generally, the best performance is achieved by the threshold and constant values 0.5 and 0.1, respectively. These values have been determined by the trial and error for the given codes and applied in the thesis unless explicitly stated. Figure 3.10 illustrates the above algorithm.

Ten iterations for the first and the third examples and 15 iterations for the second example have been performed. Again, the maximum distance between two low weight patterns in a combined pattern is set to 145.

3.5.3.1 Simulation Results for 4-state Turbo codes Using Interleaver $(T = 10, M = 1, L = 512)$

The results are presented in Figure 3.11, which shows performance of the 4-state turbo code $(1, \frac{5}{7})$ with the interleaver (T=10,M=1) and length L=512. For $\frac{E_b}{N_0} > 2dB$ 20,000 data blocks with the mentioned length were simulated. In each simulated block, an independent AWGN from other blocks is inserted. For different interleaver lengths with identical $Rem(L, T = 10) = 2$ values, the algorithm computes the weight of the code. In this example, the codeword weights for interleaver lengths L=92 and L=102 have been computed and then their results have been extrapolated for the desired length, i.e. L=512. The algorithm gives the minimum distance d_{free}=10 with $w_{free} = 3$ and N_{free}=3. The new upper bound gives a higher accuracy for the code performance for all signal to noise ratios. Among different combined low weight patterns, the pattern $(0...00100100...01110...0)_L$ produces the minimum weight with the following specifications: d=16, $N_d = 2$, $\omega_d = 6$. In this code, other combined patterns generate weights with high multiplicities far from the free distance and other low weights, and are not affecting the code performance.

3.5.3.2 Simulation Results for 4-state Turbo codes Using Interleaver $(T = 20, M = 1, L = 1024)$

Figure 3.12 illustrates analysis of the code for the interleaver (T=20, M=1) and length L=1024. For this code 10,000 blocks were simulated for different $\frac{E_b}{N_0}$ values. In this example, the code weight distribution is determined by the interleaver lengths $L = 382$ and $L = 402$. The result gives d_{free}=10 with $w_{free} = 3$ and N_{free}=3 as the free distance specifications of the code. In the considered example, almost all of the low weight codewords affecting the code performance are located at the end part of the interleaved data. However, as mentioned before, for a code with this interleaver it is possible to find many combined low weight patterns that are not located in the end part of the interleaved data influencing the code performance. This effect has been verified by the analysis of the code with and without the combined patterns. The relevant graphs illustrate about 0.2 dB difference at the error floor region between the results obtained when these two approaches are taken. In addition, analysis of the code considering the combined patterns at

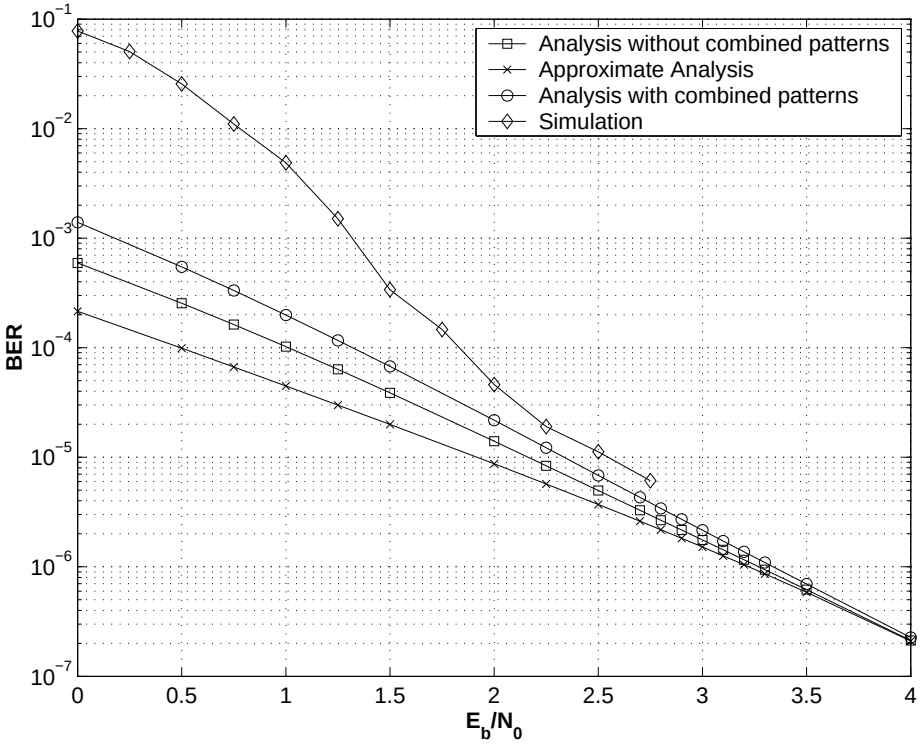

Figure 3.12 Analysis and simulation results of the 4- state turbo code $(1, \frac{5}{7})$ with the interleaver (T=20, M=1) and length L=1024.

the end of the interleaved data shows performance closer to the simulation results than the upper bound obtained without considering the effect of combined patterns. The tail bits weight effect on the code performance has been confirmed in the previous examples. The specifications of the minimum weight of the code is given by $(d, N_d, \omega_d) = (22, 1, 3)$ and $(d, N_d, \omega_d) = (26, 1, 2)$ values for the interleavers $(T = 10, M = 1)$ and $(T = 20, M = 1)$, respectively.

3.5.3.3 Simulation Results for 16-state Turbo codes Using Interleaver $(T = 35, M = 1, L = 4096)$

For 16 state turbo code $(1, \frac{35}{23})$, an interleaver $(T = 35, M = 1)$ and length $L = 4096$ is utilized. At least 2500 data blocks were simulated for $BER < 10^{-3}$. For this purpose, low weights of the relevant code are calculated from interleaver lengths $L = 1226$ and $L = 1261$. The algorithm gives some low weights from the combined low weight input bitstreams that return both RSC encoders to the zero state. For the

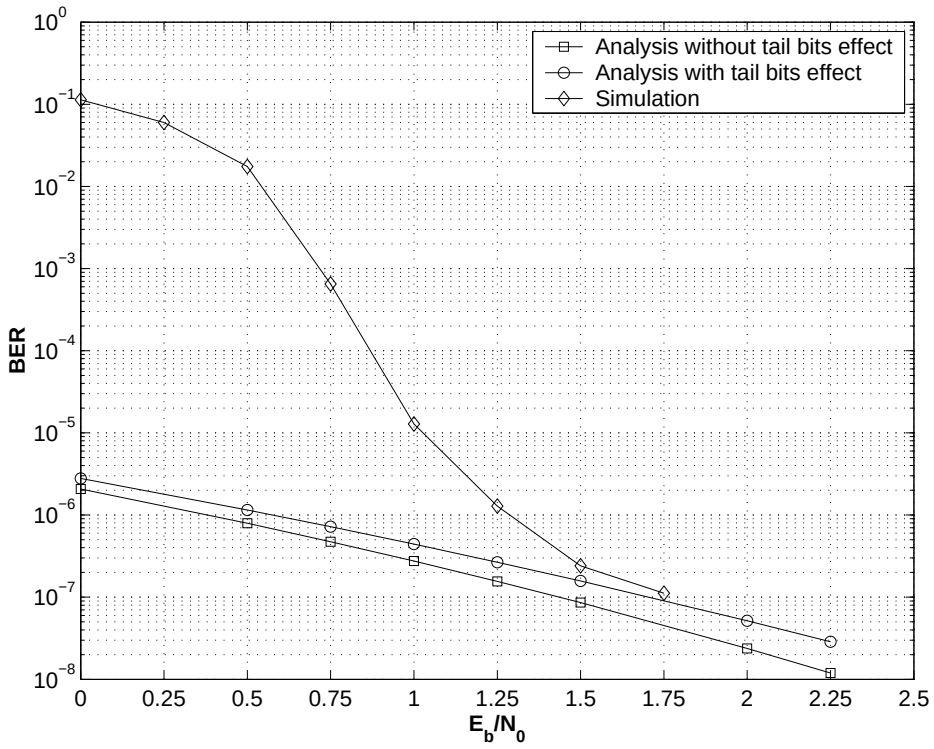

Figure 3.13 Analysis and simulation results of the 16- state turbo code $(1, \frac{35}{23})$ with the interleaver $(T = 35, M = 1)$ and length $L = 4096$.

combined two 10011 patterns, i.e. $(00...01001100...0001001100...0)_L$, the calculated minimum weight is 27 with 2 multiplicities. As expected, only a few low weights contribute to the code performance. In comparison with the two previous examples, the weight of the tail bits affects significantly the performance of these codes. Based on this assumption, the free distance specifications of the code are as follows: $d_{free} = 16$, $N_{free} = 1$ and $w_{free} = 3$. This effect on the upper bound has been illustrated as a separate graph in Figure 3.13, which gives a better approximation of the code performance at the error floor region.

3.6 Turbo Code Analysis With Convolutional Interleavers

In this section, the weight distribution of the turbo code computed by the proposed algorithm is utilized to verify effects of the optimized and non-optimized convo-

Table 3.9
Weight-2 distribution of turbo codes $(1, \frac{5}{7})$ with the optimized and non-optimization interleavers $(T = 10, M = 1)$ and length $L = 512$.

weight	Optimized interleaver	Non-optimized interleaver
d	N_d	N_d
11	1	0
12	0	0
13	2	0
14	1	0
15	2	0
16	3	0
17	2	1
18	5	0
19	2	0
20	5	0
21	6	0
22	7	0
23	4	0
24	1	1
25	9	1
26	340	336

lutional interleavers on the code performance. Since the optimized interleaver has basically the same construction as the non-optimized convolutional interleaver, the obtained weights for the code with the non-optimized interleaver should also appear at the optimized interleaver with similar multiplicities.

Depending on the non-optimized interleaver specifications and similarly to the optimized interleaver, the minimum interleaver length applied in the algorithm would be different. This minimum length is determined by the value of $((T(T-1)M) - Rem(L,T))$. Table 3.9 gives weight-2 distribution of the 4-state turbo code $(1, \frac{5}{7})$ with the non-optimized interleaver $(T = 10, M = 1, L = 512)$ and its comparison with the obtained weights from the optimized interleaver. As verified in the previous section, due to the deletion of zero stuff bits from the end part of the interleaved data, the distance between adjacent bits of the input bitstream in the interleaved data is reduced. Hence, the free distance value and some weights lower than the free distance

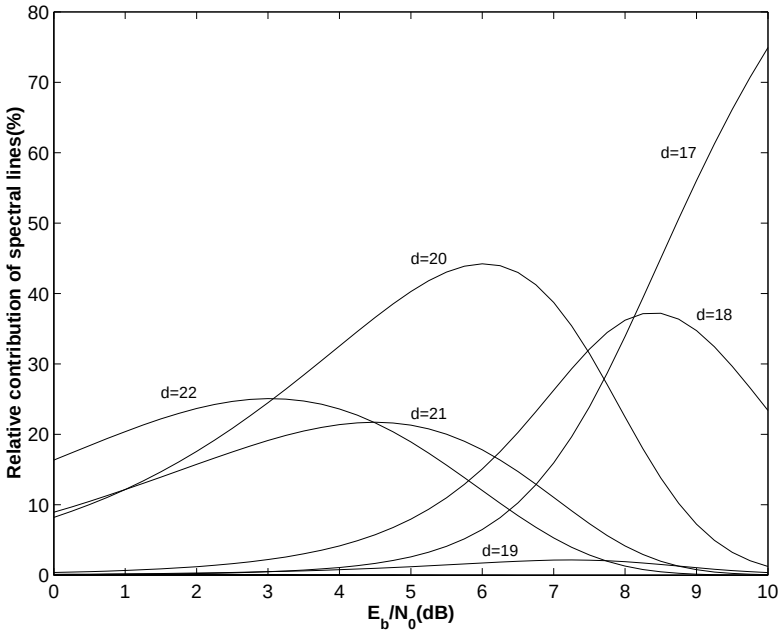

Figure 3.14 Weight contributions for the 4-state turbo code $(1, \frac{5}{7})$ with the non-optimized convolutional interleaver (T=10, M=1) and length L=512.

value of the code with the non-optimized interleaver have been obtained. In fact, unlike non-optimized interleavers, optimized interleavers generate low weights with low multiplicities, while similar weights to the free distance value of the code with non-optimized interleavers are maintained. Comparing the obtained weights of the code from two different interleavers, it is concluded that the optimized interleaver will rearrange some of the input bitstreams related to the free distance value of the code with the non-optimized interleaver to other low weights with low multiplicities. In order to verify how the generated new weights affect the code performance, the contribution of each weight to BER in the code is computed and compared with contribution of low weights in the code with the non-optimized interleaver. As indicated in [66], the contribution of each weight in the block-wise performance of the turbo code can be determined from Equation 2.10 by the following formula:

$$P_d(\gamma_b) = \frac{N_d \tilde{\omega_d}}{L} Q(\sqrt{2dR\gamma}) \tag{3.2}$$

Its relative contribution to the total BER is represented by:

$$\overline{P}_d(\gamma_b) = \frac{P_d(\gamma_b)}{\sum_d P_d(\gamma_b)} \tag{3.3}$$

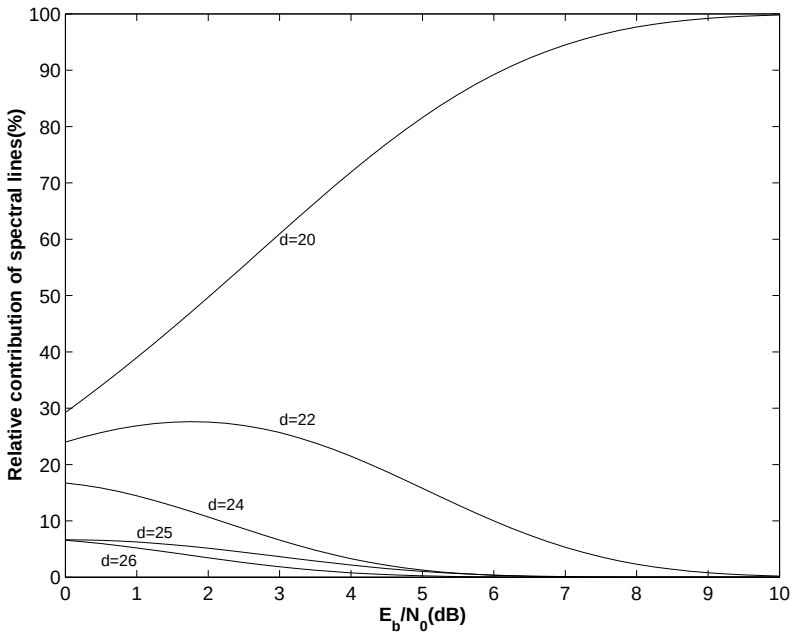

Figure 3.15 Weight contributions to BER for the 4-state turbo code $(1, \frac{5}{7})$ with the non-optimized convolutional interleaver (T=15, M=1) and length L=1024.

Again, the weight distribution of each code has been calculated based on self-terminating input bitstreams with weights no greater than 4. Figures 3.14 and 3.15 show contribution of the calculated weights for the turbo code $(1, \frac{5}{7})$ with the non-optimized convolutional interleaver lengths L=512 and L=1024, respectively. Similarly to previously obtained results for turbo code performance with block interleavers, for the short interleaver lengths, many weights contribute to the code performance, while with increasing the interleaver length only few weights are important [66].

The optimized interleavers are designed in a way that they produce a similar number of stuff bits with the applied non-optimized interleavers. Figures 3.16 and 3.17 show weight contribution for the 4-state turbo codes with the optimized interleaver (T=14, M=1) and (T=20, M=1) for the length L=512 and L=1024, respectively.

For the code with the interleaver length L=512, the optimized interleaver (T=14, M=1) has two major weights that contribute to the code performance, while the non-optimized interleavers have many weights. In this case, the obtained free distance value is 10 with 3 multiplicities, and its effect is almost dominant for all signal to

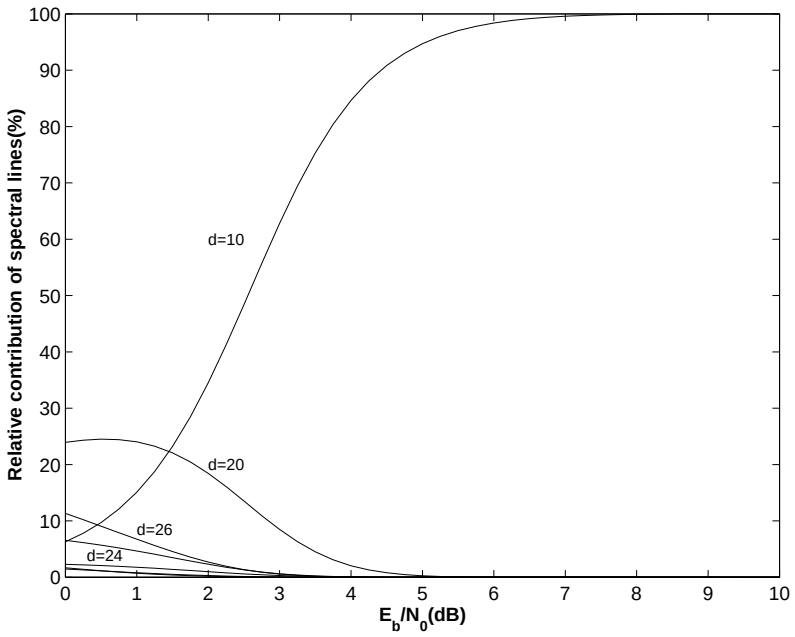

Figure 3.16 Weight contributions to BER for the 4-state turbo code $(1, \frac{5}{7})$ with the optimized convolutional interleaver (T=14, M=1) and length L=512.

noise ratios. In this code, weight 20 with 321 multiplicities has the second major contribution, while in the code with the non-optimized interleaver (T=10, M=1) weight 20 has the highest contribution with 196 multiplicities for the signal to noise ratios lower than 7 dB. For higher interleaver length $L = 1024$, the code with the non-optimized interleaver $(T = 15, M = 1)$ has the free distance value 20 with 585 multiplicities whose effect is dominant for the code performance. Applying the optimized interleaver $(T = 20, M = 1)$ with the higher period and similar number of stuff bits, i.e. 190 bits, reduces the free distance to the value of 10 with 3 multiplicities. In addition, the code also has the weight of 20 with 846 multiplicities and a lower contribution than corresponding weights in Figure 3.15, especially in signal to noise ratios greater than 2 dB.

The graphs demonstrate that although the code with the optimized interleaver generates some lower weights than the free distance value of the code with the non-optimized interleavers but due to their low multiplicities, they do not have a major influence on the code performance for low values of $\frac{E_b}{N_0}$. Similar conclusions can be reached from the weight 20 of Figures 3.16 and 3.17. Since the distance of

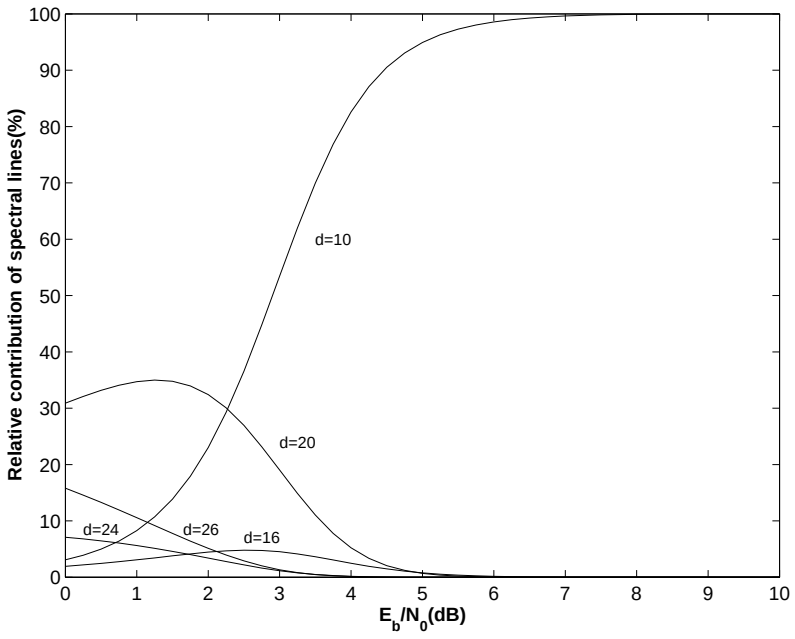

Figure 3.17 Weight contributions to BER for the 4-state turbo code $(1, \frac{5}{7})$ with the optimized convolutional interleaver $(T = 20, M = 1)$ and length $L = 1024$.

this weight to the free distance value is relatively high, its effect on the code performance is different from the identical weight in the code with the non-optimized interleaver. The obtained graphs for the optimized interleaver imply that patterns $(00..011100..0)_L$ and $(00..0100100...0100100..0)_L$ with length L have major contributions to the code performance, which respectively produce the free distance value 10 and the weight 20.

From the obtained results, one can conclude that the optimized interleaver generates a low free distance value and the effect of other weights on the code performance is approximately voided, particularly in the medium to high signal to noise ratios. In addition, apart from the end part of the optimized interleaved data, where the free distance value is obtained, increasing the interleaver period will increase the distance between adjacent bits of input bitstream in the interleaved data. This will create higher weights and reduce multiplicities of other low weights having major contributions to the code performance. Therefore, with a suitable selection of the interleaver characteristics, performance of the code can be improved.

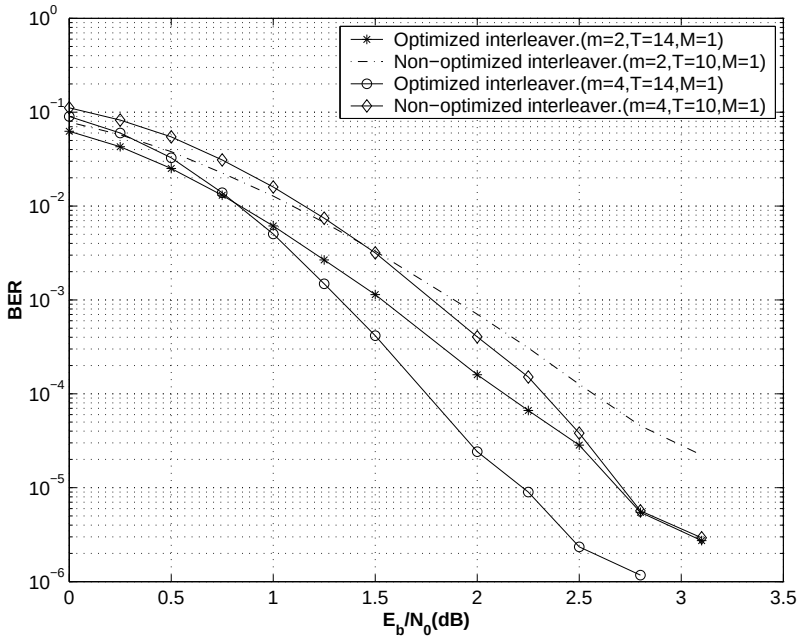

Figure 3.18 Performance of full rate turbo codes with the interleaver length $L = 512$.

3.6.1 Simulation Results

Simulations have been conducted for 4- and 16-state turbo codes $(m = 2, 1, 5/7)$ and $(m = 4, 1, 35/23)$, where m represents number of memories for the RSC encoders. Again, 20,000 and 10,000 and 2500 data blocks were simulated for the interleaver lengths $L = 512$, $L = 1024$ and $L = 4096$, respectively. The encoded data have been decoded using SOVA as selected iterative decoding method in 8 iterations and in a presence of AWGN.

3.6.1.1 Simulation Results for Interleaver Length $L = 512$

Figure 3.18 shows the performance of full rate 4- and 16-state turbo codes with the interleaver length $L = 512$. It can be observed that the optimized interleaver $(T = 14, M = 1)$ has improved performance by 0.5 dB for both codes. In addition, the 4-state code with the interleaver $(T = 14, M = 1)$ has better performance than the 16- state code with the interleaver $(T = 10, M = 1)$ in low signal to noise ratios with reduced complexity in the encoder structure and decoding process. Similar results have been achieved for the half rate turbo codes with similar interleavers and

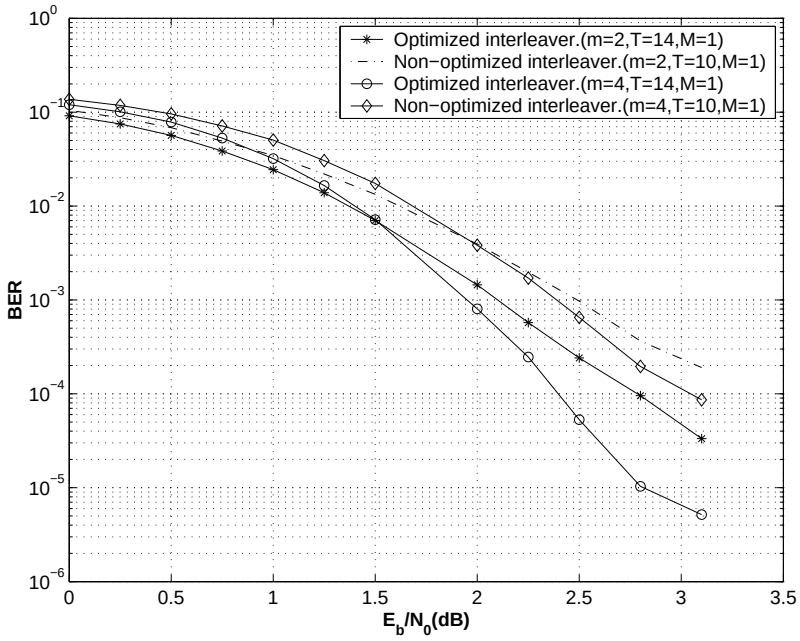

Figure 3.19 Performance of half rate turbo codes with the interleaver length $L = 512$.

are presented in Figure 3.19. These results show that the 4-state code with the optimized interleaver has 0.25 dB better performance than the 16- state code with the non-optimized interleavers for all signal to noise ratios.

3.6.1.2 Simulation Results for Interleaver Length $L = 1024$

Similar turbo codes to the above examples have been examined for a higher interleaver length, $L = 1024$. For the full rate 4-state turbo codes, as shown in Figure 3.20, the optimized interleaver $(T = 20, M = 1)$ slightly improves the code performance compared to the non-optimized interleaver $(T = 15, M = 1)$ for $\frac{E_b}{N_0} \leq 2$ dB, while for $\frac{E_b}{N_0} > 2$ dB, both interleavers have similar performance. This behavior can be easily explained by the weight contributions presented in Figures 3.15 and 3.17. In these Figures, the weight $w = 20$ has the highest contribution to BER. In the mentioned signal to noise ratio ranges, this weight in the code with the optimized interleaver has the slightly lower contribution than for the code with the non-optimized interleaver. Hence, the better performance for the code with the optimized interleaver is expected. After $\frac{E_b}{N_0} = 2$ dB, the contribution of this weight is replaced by weight 10 with low multiplicity, which creates a similar performance to the weight

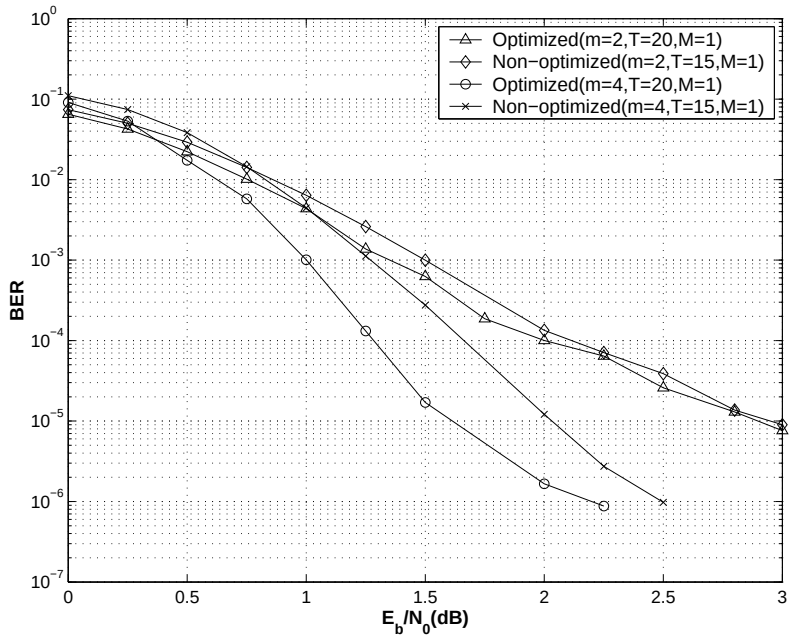

Figure 3.20 Performance of full rate turbo codes with the interleaver length $L = 1024$.

20 with high multiplicity. However, for the half rate code, the optimized interleaver has improved the code performance by 0.25 dB. For the 16-state code, it is simply confirmed that the non-optimized interleaver $(T = 15, M = 1)$ is unable to break weight-2 self-terminating patterns. This condition generates a free distance value with high multiplicity and consequently degrades the code performance. In contrast, the optimized interleaver $(T = 20, M = 1)$ outperforms this drawback and improves the full and half rate code performance by 0.5 dB in the waterfall and error floor regions.

3.6.1.3 Simulation Results for Interleaver Length $L = 4096$

More testing was conducted for the longer interleaver length. Figures 3.22 and 3.23 show full and half rate 4- and 16-state turbo code performance with optimized and non-optimized interleavers of length $L = 4096$. At both rates, optimized and non-optimized interleavers create a similar performance for the 4-state turbo code. The weight-2 distribution obtained from self-terminating patterns confirms that both interleavers create a low weights with the high multiplicities. For example, the weights $w = 20$ and $w = 15$ were calculated for the optimized interleaver $(T = 35, M = 1)$

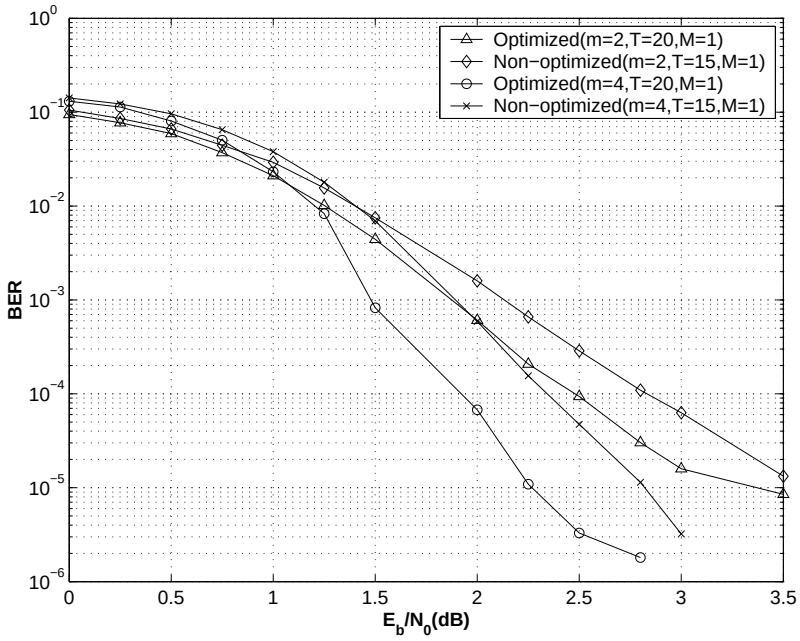

Figure 3.21 Performance of half rate turbo codes with the interleaver length $L = 1024$.

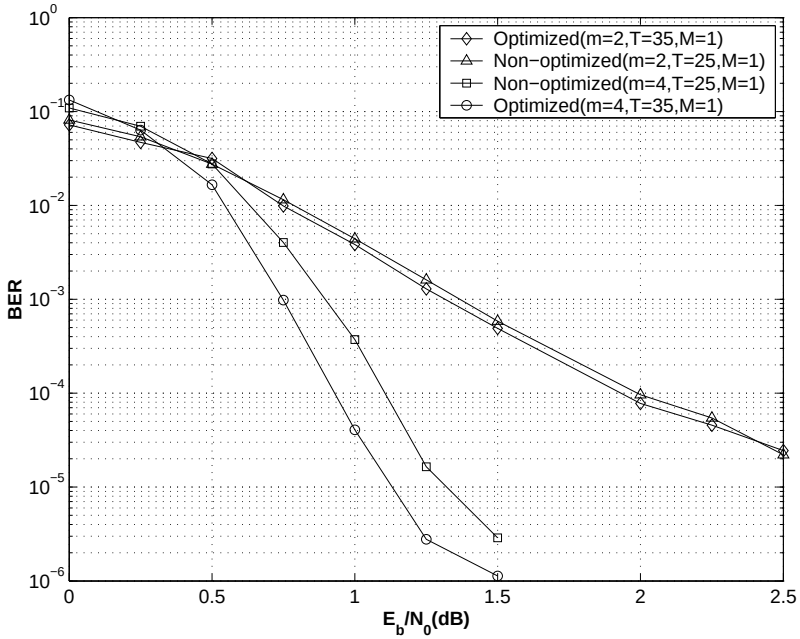

Figure 3.22 Performance of full rate turbo codes with the interleaver length $L = 4096$.

and the non-optimized interleavers $(T = 25, M = 1)$, respectively. It means that the applied interleavers are not sufficiently break low weight self-terminating patterns

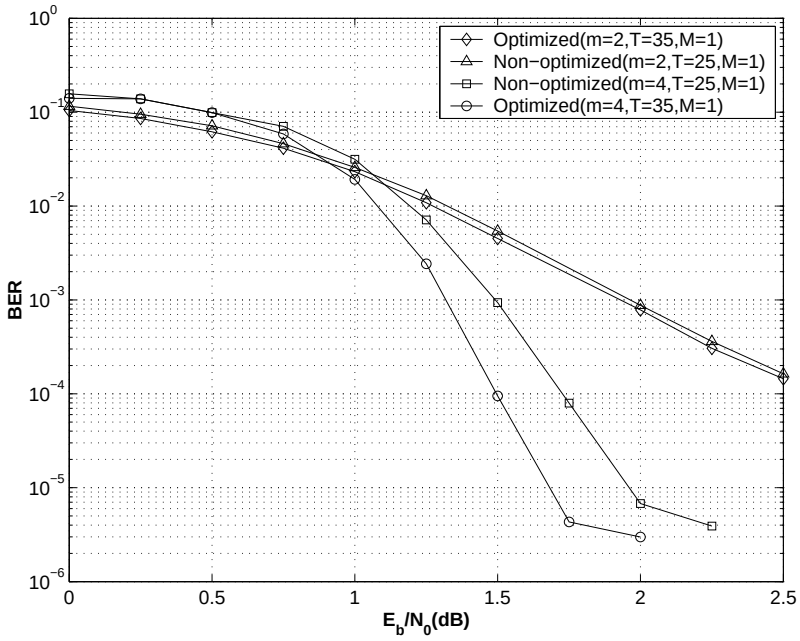

Figure 3.23 Performance of half rate turbo codes with the interleaver length $L = 4096$.

and hence both interleavers provide similar performance for the full and half rate codes. For the 16-state turbo code $(1, \frac{35}{23})$, the optimized interleaver $(T = 35, M = 1)$ improves the full and half rate code performance by 0.25 and 0.35 dB, respectively. In comparison with non-optimized interleaver $(T = 25, M = 1)$, the optimized interleaver $(T = 35, M = 1)$ creates higher distance between adjacent bits of input bitstreams in the interleaved data. This provides a higher weight with lower multiplicities, which can improve the code performance.

The simulations express that, although in the optimized interleavers the distance between bits in the end part of the interleaved data reduces, but due to applying a higher period this drawback is compensated. The results conclude that in case of the similar number of stuff bits, optimized interleavers outperforms the non-optimized interleavers.

3.7 Comparison with Block Interleavers

In this section, the performance of the optimized convolutional interleavers are compared with the semi-random and row-column interleavers, which are the conventional block interleavers for short and long data length, respectively [71]. The lengths $L = 169$ and $L = 1024$ are selected for the short and long interleaver lengths. 60,000 data blocks were utilized in simulations of each selected $\frac{E_b}{N_0}$ values of the turbo code with the interleaver length $L = 169$. $S = 8$ and $S = 22$ have been selected as threshold values of semi-random interleavers with lengths $L = 169$ and $L = 1024$, respectively. For the row-column interleavers, the numbers of row and columns are equal. Similarly to conducted simulations in section 3.6.1, SOVA with 8 iterations is applied in the presence of AWGN.

3.7.1 Simulation Results for Short Interleaver Lengths

Figures 3.24 and 3.25 show the full and half rate 4-state turbo code $(1, \frac{5}{7})$ performance. The results show that the convolutional interleavers have better performance than the row-column and semi-random interleavers in the waterfall region. The improvement is achieved when the convolutional interleaver period increases, for example the interleaver $(T = 13, M = 1)$ has 0.4 dB better performance than the interleaver $(T = 11, M = 1)$ at $\frac{E_b}{N_b} = 1.5$ dB. Of course, this improvement involves increasing the number of stuff bits. All convolutional interleavers have insufficient behavior at the error floor region, which is due to the presence of the small free distance of the turbo code when these interleavers are applied. The graphs show that increasing the period slightly improves the code performance in this region.
For the applied half rate code $(1, \frac{5}{7})$, as shown in Figure 3.25, the results demonstrate that convolutional interleavers have better performance than the row-column and the semi-random interleavers for all signal to noise ratios. In this figure, the convolutional interleaver $(T = 11, M = 1)$ with 55 stuff bits and a lower number of memories generates similar performance to the row-column interleaver. Better performance can be achieved when an interleaver with a higher period is applied. One example would be convolutional interleavers with periods $T = 12$ and $T = 13$, where improve the code performance is improved by 0.25 dB. In both codes, The convolu-

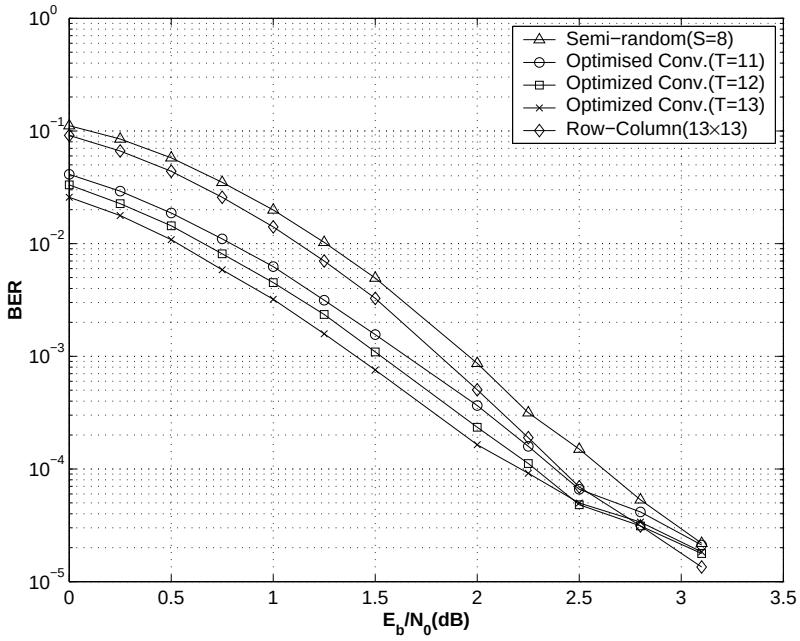

Figure 3.24 Performance of full rate 4-state turbo code with the interleaver length $L = 169$.

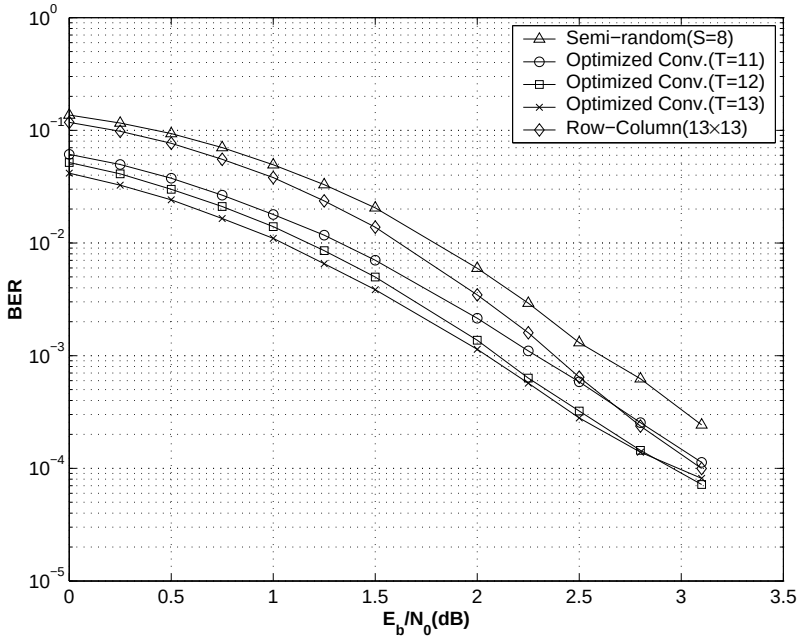

Figure 3.25 Performance of half rate 4-state turbo code with the interleaver length $L = 169$.

tional interleaver $(T = 11, M = 1)$ outperforms the semi-random interleavers $S = 8$ by 0.5 dB.

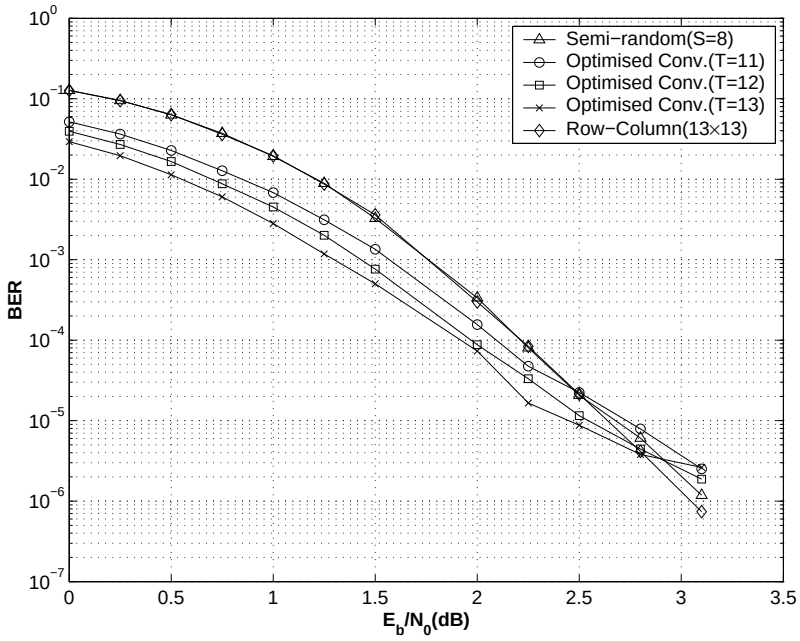

Figure 3.26 Performance of full rate 16-state turbo code with the interleaver length $L = 169$.

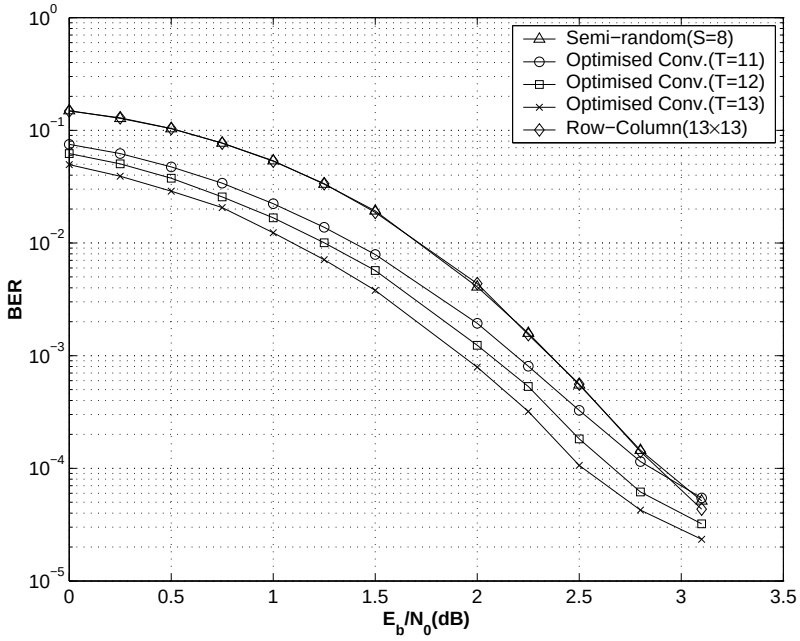

Figure 3.27 Performance of half rate 16-state turbo code with the interleaver length $L = 169$.

Figures 3.26 and 3.27 show the performance of the full and half rate 16-state turbo codes $(1, \frac{35}{23})$ with the considered interleaver length $L = 169$, respectively. The

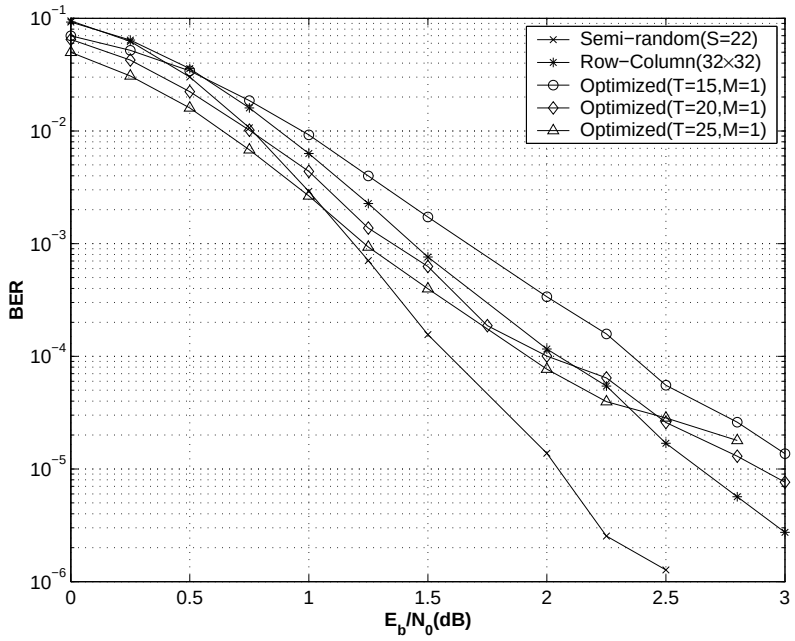

Figure 3.28 Performance of full rate 4-state turbo code with the interleaver length $L = 1024$.

convolutional interleavers provide better performance than the row-column and semi-random interleavers. In the full rate code, increasing the interleaver period from $T = 11$ to $T = 13$, improves the code performance by 0.25 dB. For the half rate code, The improvement is achieved by 0.7 dB in all signal to noise ratios.

3.7.2 Simulations Results for Long Interleaver Lengths

More verifications have been accomplished for the turbo code with the longer interleaver length $L = 1024$. Figure 3.28 shows the performance of the full rate 4-state $(1, \frac{5}{7})$ code. As discussed in the previous section, increasing the period of the convolutional interleaver only improves the code performance at the waterfall region, while due to existence of a small free distance value generated in the end part of the interleaved data, the code performance at the error floor region with different convolutional interleavers is similar. This is especially evident for the interleavers $(T = 20, M = 1)$ and $(T = 25, M = 1)$. In comparison with the row-column and semi-random interleavers, application of the convolutional interleaver $(T = 25, M = 1)$ produces 0.2 dB better and around 0.5 dB worse performance at

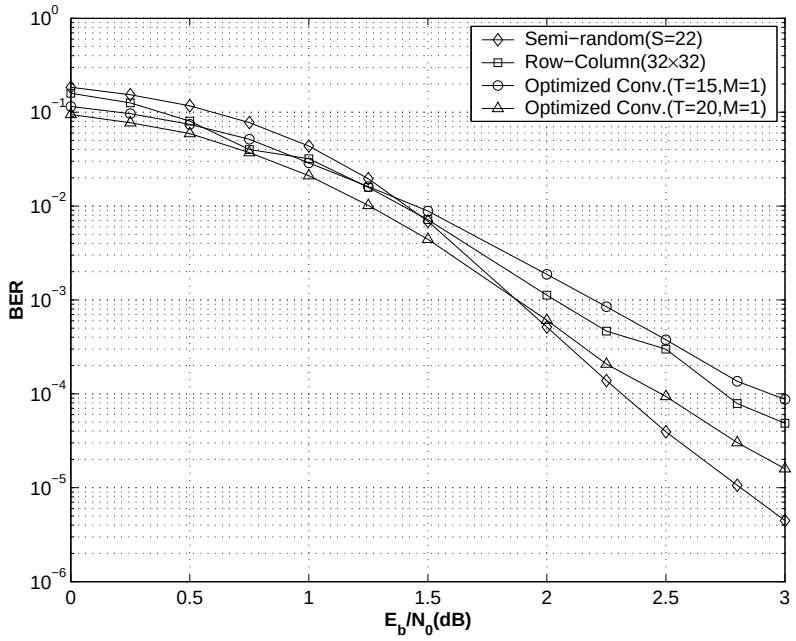

Figure 3.29 Performance of half rate 4-state turbo code with the interleaver length $L = 1024$.

the waterfall region, respectively. The half rate code with the convolutional interleaver $(T = 20, M = 1)$ with 190 stuff bits, has 0.2 dB better and 0.25 dB poor performance to the case when the row-column and semi-random interleavers are used, respectively, as shown in Figure 3.29. The same interleavers have been applied for the 16-state full rate turbo code $(1, \frac{35}{23})$ and the resulting performance is illustrated in Figure 3.30. The graphs show that a code with the interleaver $(T = 20, M = 1)$ has worse performance than with the semi-random and row-column interleavers. Increasing the period from $T = 20$ to $T = 30$, which increases the number of stuff bits by 68%, results in an improved code performance of 0.25 dB over the performance of code with the semi-random interleaver at the waterfall region. Similar results are achieved for the half-rate 16-state turbo code. Figure 3.31 shows that increasing the interleaver period from $T = 20$ to $T = 30$ improves the code performance by 1 dB. However, in comparison with block interleavers, degradation is again observed at the error floor region.

3.8 Chapter Summary and Conclusions

In this chapter, the new structure of the convolutional interleaver operating as a block interleavers has been presented. Based on the interleaver properties, a simple method was introduced to calculate weight distribution of the turbo code. Conducted analysis and simulations confirmed that in case of similar number of stuff bits, the optimized interleaver outperforms the non-optimized interleaver. In contrast to other block interleavers, optimized interleavers create a free distance with lower multiplicity than other block interleavers. This means to the optimized interleavers has better performance than the row-column and semi-random interleavers, which was evident for short interleaver lengths. The main problem with the optimized convolutional interleaver is the resulting small free distance value of the code, which degrades the code performance in the error floor region. In the next chapter, the structure of the optimized convolutional interleaver is modified in such a way that a higher free distance value of the code is obtained with a lower number of stuff bits, while a low multiplicity is maintained to sufficiently improve the code performance in the error floor region.

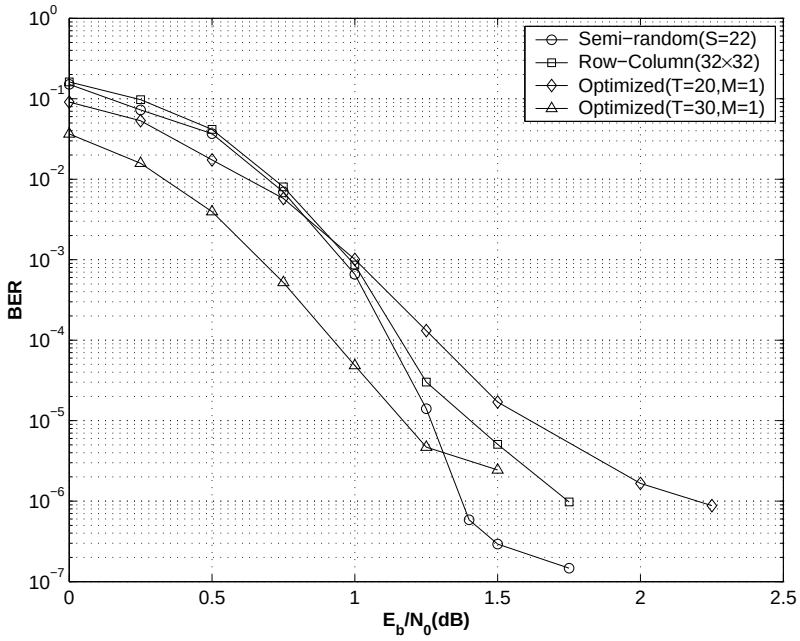

Figure 3.30 Performance of full rate 16-state turbo code with the interleaver length $L = 1024$.

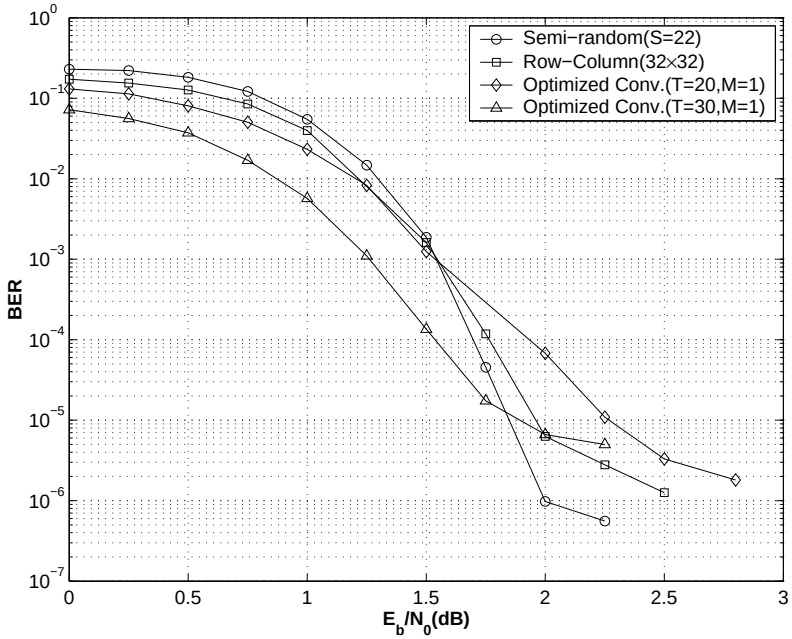

Figure 3.31 Performance of half rate 16-state turbo code with the interleaver length $L = 1024$.

Chapter 4

Modified Convolutional Interleavers

4.1 Introduction

Due to removing the stuff bits from the end part of the interleaved data of a convolutional interleaver, the free distance of the turbo code becomes low, which degrades the code performance in the error floor region. A remedy for it can be increasing of the interleaver period as suggested in Chapter 3, which, on the other hand increases the percentage of stuff bits. In this chapter, a modification to the optimized interleaver improving turbo codes performance without increasing the interleaver period is introduced. The improvement is achieved by increasing the distance of adjacent bits that are positioned in the original input bitstream during the interleaving procedure. The modified interleavers operation in different turbo code structures have been studied and results have been compared with those for the previously suggested interleavers. This is accomplished by an analysis of the code with different weight input bitstream, and its results are confirmed by the conducted simulations.

4.2 Modification Algorithm for Convolutional Interleavers

The analysis of turbo codes based on weight-2 distribution confirms that because an interleaver can increase the distance between two adjacent bits of the original bitstream, the codeword obtained from the second RSC encoder can have a higher

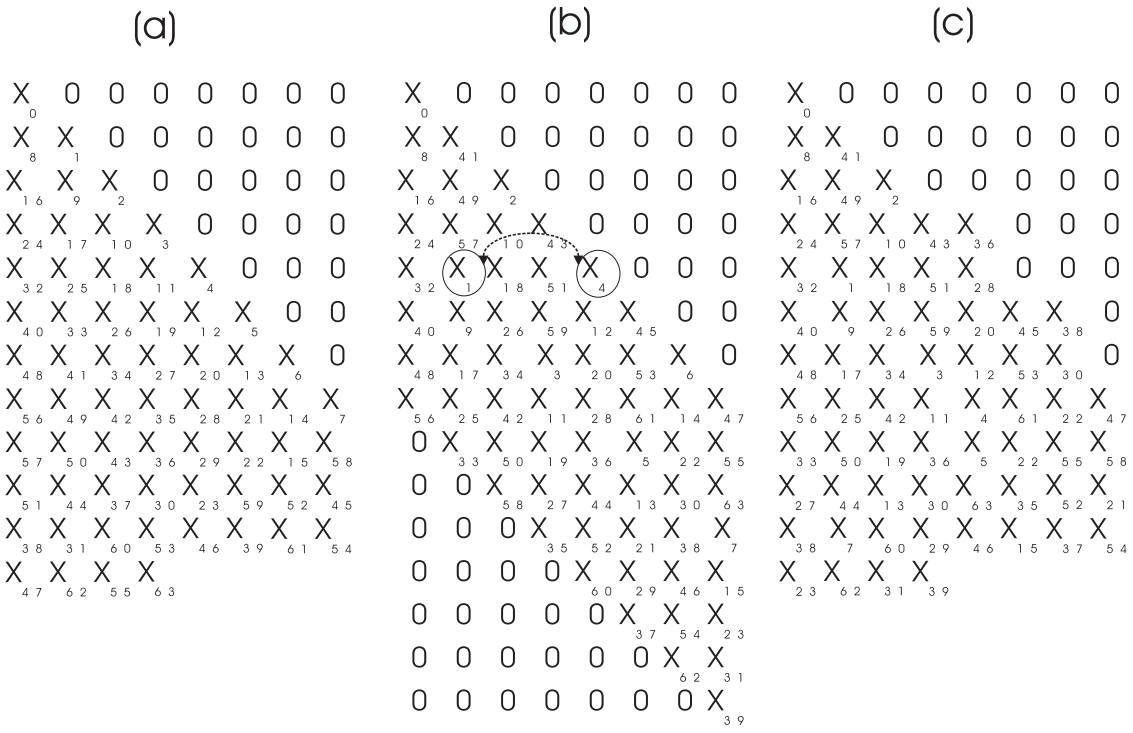

Figure 4.1 Interleaved data obtained from the interleaver $(T = 8, M = 1, L = 64)$. a) Without modification, b) just even column shifting and c) even and odd column shifting with zero bit deletion.

weight, which consequently improves the code performance [11].

As verified in Chapter 3, for the optimized convolutional interleaver, increasing the interleaver period generally improves the code performance at the waterfall region, while due to the deletion of stuff bits from the end part of the interleaved data, it is not an efficient solution to increase the distance between adjacent bits that have been located at the end part of the interleaved data. In order to improve code performance in both regions without increasing the interleaver period, replacing some bits located in the end part of the interleaved data with other bits positioned in higher interleaver parts are proposed. The new bits located in the end part of the interleaved data will be at a sufficient distance from the bits adjacent to them before interleaving. Also, this process should prohibit the generation of low weight self-terminating patterns having major effects on the second RSC code performance [18].

A modification is applied to the non-optimized convolutional interleaver, where by

input bitstreams are distributed regularly in the interleaver lines. In this case, the relevant interleaved data of each line can be represented in one column. A suitable shift of bits located in an interleaver column can increase the distance between adjacent bits, which are located in different columns. Of course, if a similar shift was performed for all of the columns, the distance would not be changed. Thus, different shift patterns should be used for different columns. For greater simplicity, distinct shifts only for odd and even columns are considered. Finally, zero stuff bits located at the end part of the interleaved data are deleted to optimize the conducted modification of the interleaver.

As mentioned in Chapter 3, the minimum distance generated between adjacent bits of the input bitstreams from an interleaver (T, M) is equal to the product of T and M. Therefore, in the modification some bits are shifted by a value greater than this value. For even columns, each bit is cyclically shifted by $(2M + 1) * T$ units. For each column, the number of shifted bits is assumed to be even. If the overall number is odd, the first stuff bit data before the first data in each column is also shifted to maintain the even bit number. The even number is selected in order to achieve an acceptable distance between adjacent bits [18].

However, due to even columns bits shifts, it is possible that the new interleaved data block is characterized by lower weights than that from the former interleaver and can generate more low weight patterns that return the second RSC encoder to the zero state. In the example shown in Figure 4.1(b), if bit 1 and bit 4 in the fifth row of the interleaver have value of 1, the second RSC encoder of turbo code $(1, \frac{5}{7})$ will be returned to the zero state with weight 4. In addition to the other bits of column 2, similar conditions can be observed between bits of other even columns and the corresponding odd column bits. Hence, it is necessary to shift these odd column bits in a way compatible with the applied RSC structure to eliminate low weight patterns from the interleaver and increase weight of turbo codes. It was found that reverse sorting of odd column bits, except in column 1 and 3, can provide a sufficient distance for RSC encoders with different states. Similarly as with even columns, the number of shifted bits is considered to be even [18]. Figure 4.1(c) shows the interleaved data block in the presented example after the modification and deletion

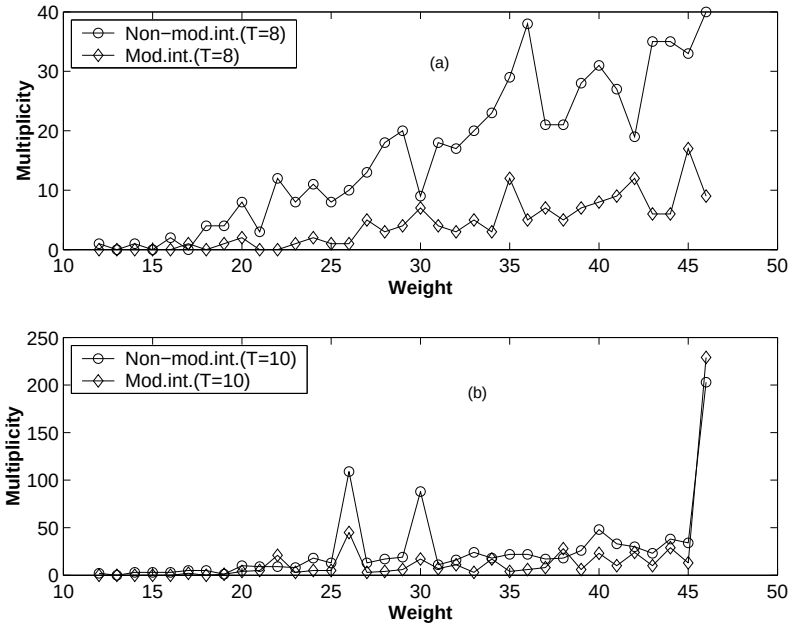

Figure 4.2 Weight-2 distribution of the turbo codes with the non-modified and modified interleavers with length $L = 169$. a) 16-state code $(1, \frac{35}{23})$ and b) 4-state code $(1, \frac{5}{7})$.

of zero stuff bits. In this example, if the location of bits 62 and 63 in Figures 4.1(a) and 4.1(c) is considered, the algorithm has increased their distance from 1 to 12. As shown in Figure 4.1(c), in one row of the interleaver, the distance between bits in column 1 and 6 before and after interleaving has not changed. This is due to the low input data length, $L = 64$, compared with the interleaver period, $T = 8$, in this example.

In practical designs, the interleaver period should be properly selected relative to the interleaver length in order to generate an acceptable number of stuff bits. In the case of higher interleaver lengths, when applying the presented modification, the distance between bits in column 1 and 6 before and after interleaving will differ because column 1 bits would remain constant while column 6 bits resorted from 5th or 6th row of the interleaver, depending on the number of bits in the column 6, such that the new column 6 bits positions have reasonable distance from the positioning of bits in column 1 in corresponding rows.

Figure 4.2 shows weight-2 distribution of the 4- and 16- state turbo codes with

length $L = 169$. For both codes, the graphs represent that the modified interleaver has improved the effective free distance value of the code with low multiplicity, while the effect of other low weights with high multiplicities has been also reduced.

4.3 Analysis of Turbo Codes Using the Modified Interleaver

Due to the variable distances between adjacent bits of the input data block in modified interleavers with different lengths, similar $Rem(L, T)$ and period values, the weight distribution algorithm proposed in Chapter 3 is not applicable to compute the weight distribution of the code with the medium to high interleaver lengths. However, it is possible to calculate the weight distribution of the code from some self-terminating patterns that have a major influence on the code performance.

4.3.1 Analysis of Weight-1 Input Bitstreams

The optimized interleaver permutes the tail bits generated from trellis termination of the first RSC encoder to the end part of the interleaved data. When weight of an input bitstream is located in the mentioned part, low weight value for the code is expected.

Since the proposed modification shifts the bits from the end part to the other parts of the interleaved data, a higher minimum weight for the code is expected. For example, applying modification on the optimized interleaver $(T = 20, M = 1)$ improves the minimum weight of the 4-state code $(1, \frac{5}{7})$ from 24 to 46 with multiplicity 1. Similar results is achieved from the 16 state turbo code $(1, \frac{35}{23})$. The modification on the interleaver has increased the minimum weight of the code from 27 to 70 with multiplicity 1.

4.3.2 Analysis of Weight-3 Input Bitstreams

Due to applying different shifting methods for bits distributed in the adjacent interleaver lines, it is expected that the utilized modification be able to break low weight-3 self-terminating patterns. In this analysis, input bitstreams $(000...11100...00)_L$ and

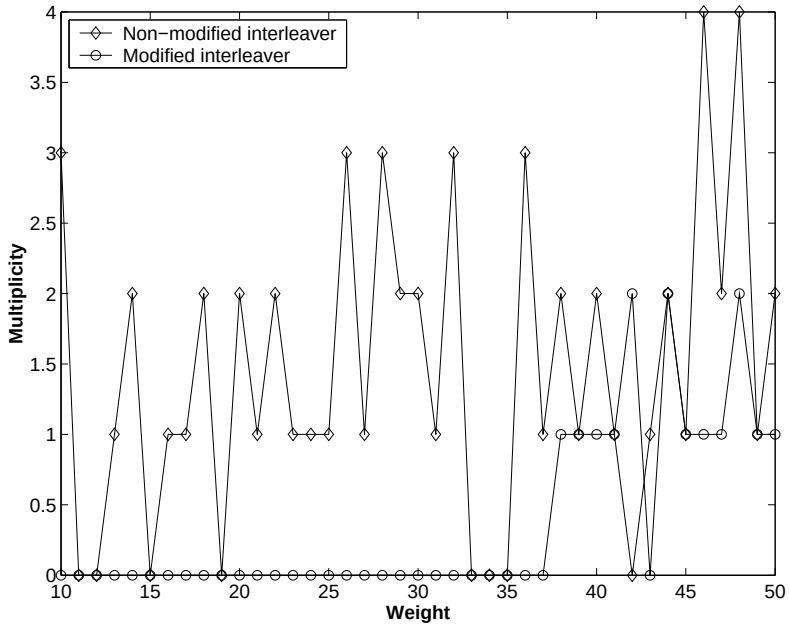

Figure 4.3 Weight-3 distribution of the 4-state turbo code with the non-modified and modified interleaver $(T = 20, M = 1)$ for self-terminating patterns $(00...011100..0)_{L=1024}$.

$(000...1001100...00)_L$ acting as the effective weight-3 self-terminating patterns for the 4- and 16- state turbo codes performance are considered, respectively. Figures 4.3 and 4.4 show weight-3 distributions of these codes. It is observed that applying the modification has sufficiently improved the minimum weight of the code. For the 4-state code, modified interleaver increases the low weight 10 (ten) and the multiplicity 3 to the weight 38 with multiplicity 1.

4.3.3 Analysis of Higher Weight Input Bitstreams

The analysis verifies the effect of input bitstreams $(00...0100100...010010..0)_L$ and $(00...011100...01110..0)_L$ with weight-$2m$ $(m = 2, 3)$ for the 4-state turbo codes with the interleaver $(T = 20, M = 1, L = 1024)$. In both patterns the distance between basic self-terminating patterns, i.e. (1001) and (111) is not exceeded than value 145. For weight-4 self-terminating patterns, the minimum weight has been improved from 20 with 845 multiplicity to weight 49 with 1 multiplicity. Similarly, for the weight-6 input bitstreams, the modification has improved the minimum weight of the code from 16 with 3 multiplicities to 32 with multiplicity 1. For weight-6

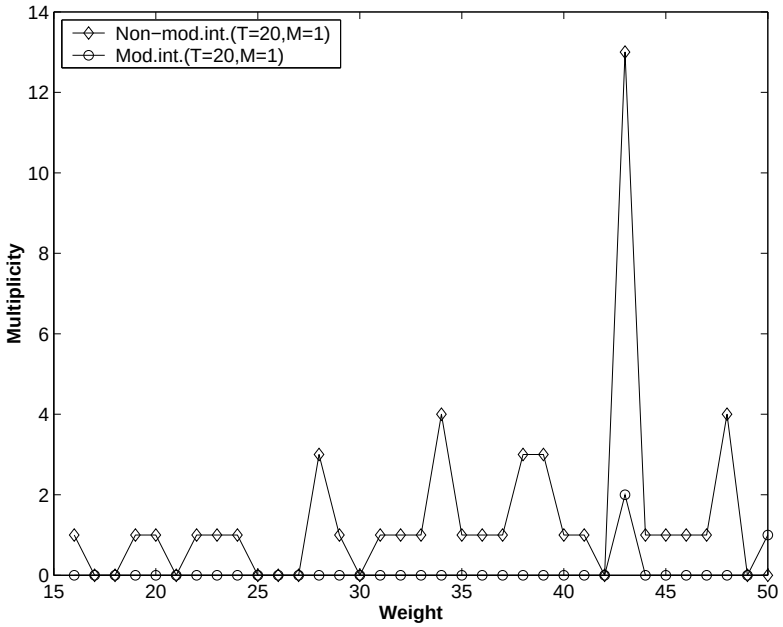

Figure 4.4 Weight-3 distribution of the 16-state turbo code from $(00...01001100...0)_{L=1024}$ with non-modified and modified interleavers $(T = 20, M = 1)$.

patterns, the results also confirm that the modified interleaver remove other weights with high multiplicities (such as weight 20 with 882), which can be effective to the code performance.

4.4 Simulation Results

In simulations, SOVA with 8 iterations is utilized as iterative decoding method for 4- and 16- states turbo codes $(1, \frac{5}{7})$ and $(1, \frac{35}{23})$. 0.5 and 0.1 are set for constant and threshold values of SOVA scaling, respectively. Simulations are conducted for short and long interleaver lengths. In each case, performance of the modified interleaver is compared with the results obtained for the optimized interleavers from Chapter 3.

4.4.1 Simulation Results for Interleaver Length $L = 169$

Figures 4.5 and 4.6 show results obtained from the full and half rate turbo codes with an input data length of $L = 169$ and different interleaver periods, respectively. Simulations have been conducted based on 60,000 data blocks for each $\frac{E_b}{N_0}$ value. In

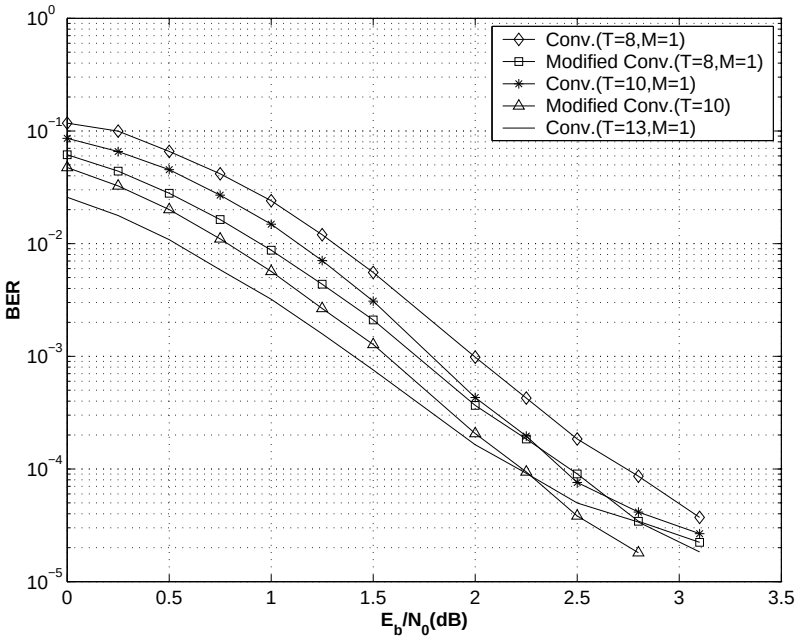

Figure 4.5 Performance of the 4- state full rate turbo code with different interleaver periods and length $L = 169$.

both interleavers the performance of the interleaver has improved with an increase of the period. The application of a new convolutional interleaver with period $T = 8$ results in a code with better performance than when a convolutional interleaver with period $T = 10$ is used. This means that the new interleaver has a reduced number of stuff bits, namely from 45 to 28, which is equal to 37 percent. The simulation results presented in Figures 4.7 and 4.8 for the 16-state turbo codes with length $L = 169$ confirm the above findings. Again, the new interleaver with a lower period, i.e. $T = 8$, has similar performance to the previous convolutional interleaver with a higher period $(T = 10)$. In addition, except for the half rate 16-state turbo code, the modified interleaver $(T = 10, M = 1)$ produces a similar performance to the convolutional interleaver $(T = 13, M = 1)$, while requiring the number of stuff bits to be reduced by 42 percent.

Considering two interleavers with similar period, a new interleaver with $T = 10$ improves turbo code $(1, \frac{5}{7})$ performance in the error floor region by 0.25 dB for full and half rates. For both half and full rate turbo codes $(1, \frac{35}{23})$, the improvement at the error floor region is equal to 0.2 dB.

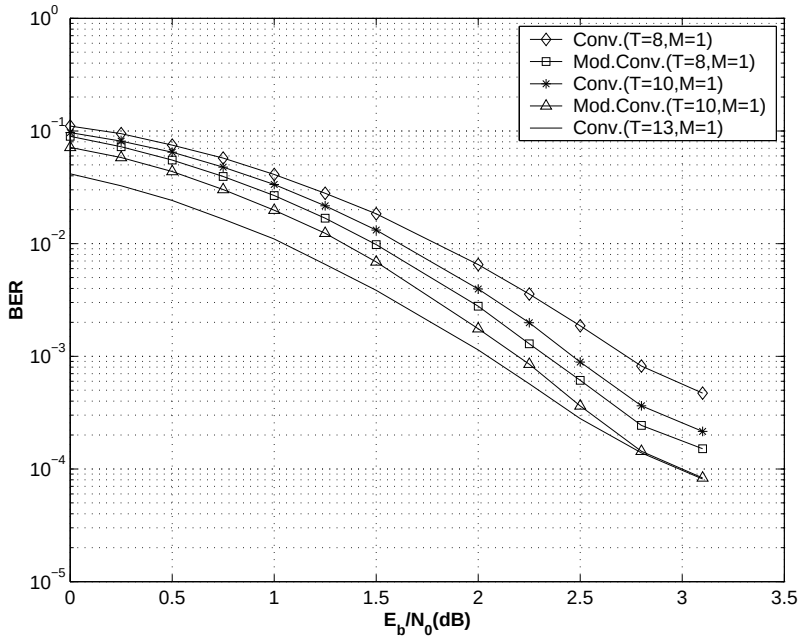

Figure 4.6 Performance of the 4- state half rate turbo code with different interleaver periods and length $L = 169$.

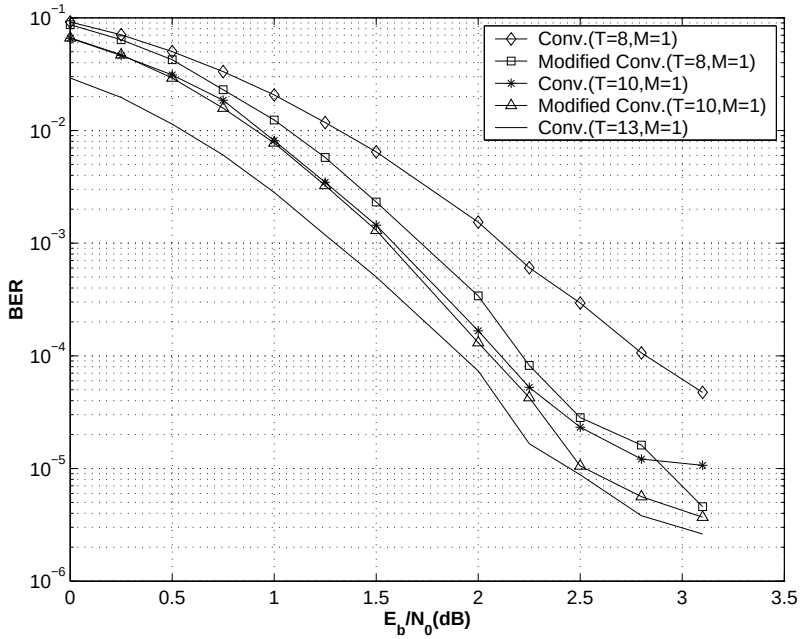

Figure 4.7 Performance of the 16- state full rate turbo code with different interleaver periods and length $L = 169$.

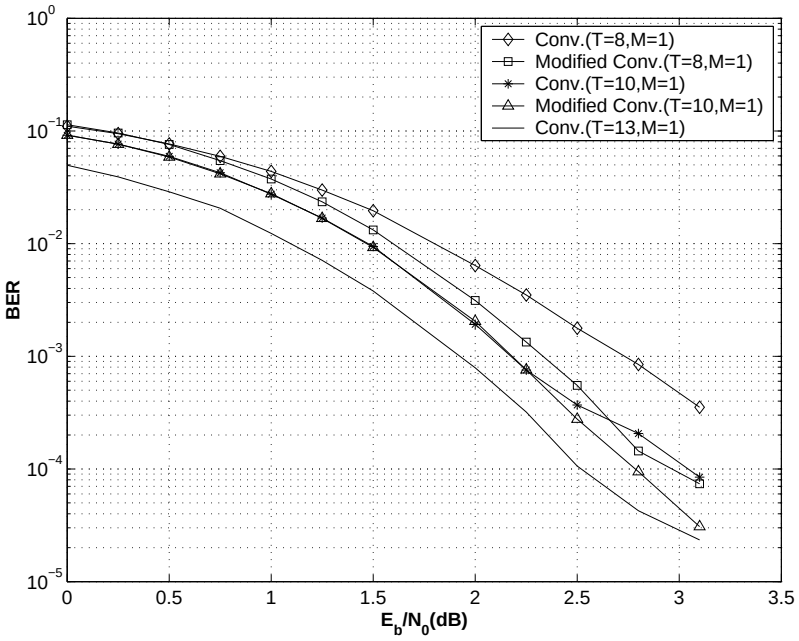

Figure 4.8 Performance of the 16- state half rate turbo code with different interleaver periods and length $L = 169$.

4.4.2 Simulation Results for Interleaver Length $L = 1024$

Figures 4.9 and 4.10 show simulation results for 4-state full and half rate turbo codes $(1, \frac{5}{7})$ with convolutional interleavers $(T = 15, M = 1, L = 1024)$ and $(T = 20, M = 1, L = 1024)$, respectively. Again, 10,000 data blocks were considered for different $\frac{E_b}{N_0}$ values in simulations. Applying a modification to the interleaver $(T = 20, M = 1)$ has improved the code performance by 0.25 dB. In addition, at the expense of a higher number of stuff bits, the modified interleaver $(T = 20, M = 1)$ has better performance than the modified interleaver $(T = 15, M = 1)$, this removes the drawback of the interleaver $(T = 20, M = 1)$ performance, when compared to the interleaver $(T = 15, M = 1)$ at the error floor region. For both half-rate codes, the modified interleaver $(T = 15, M = 1)$ has similar performance to an interleaver $(T = 20, M = 1)$ with a lower number of stuff bits. In the case of similar periods, modification of the interleavers $(T = 15, M = 1)$ and $(T = 20, M = 1)$ has improved the code performance by 0.2 dB and over 0.3 dB, respectively. Results for the full and half rate 16-state turbo codes $(1, \frac{35}{23})$ are shown in Figures 4.11 and 4.12.

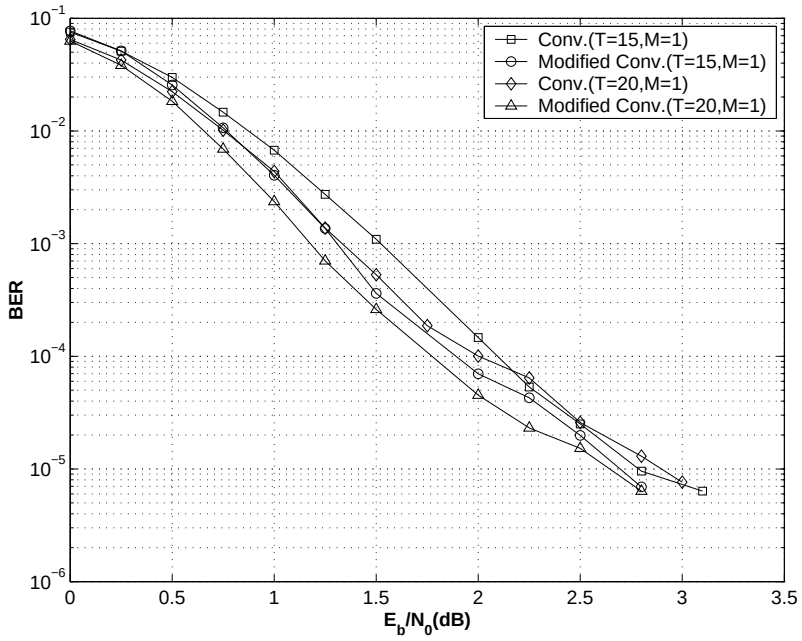

Figure 4.9 Performance of the 4- state full rate turbo code with different interleaver periods and length L=1024.

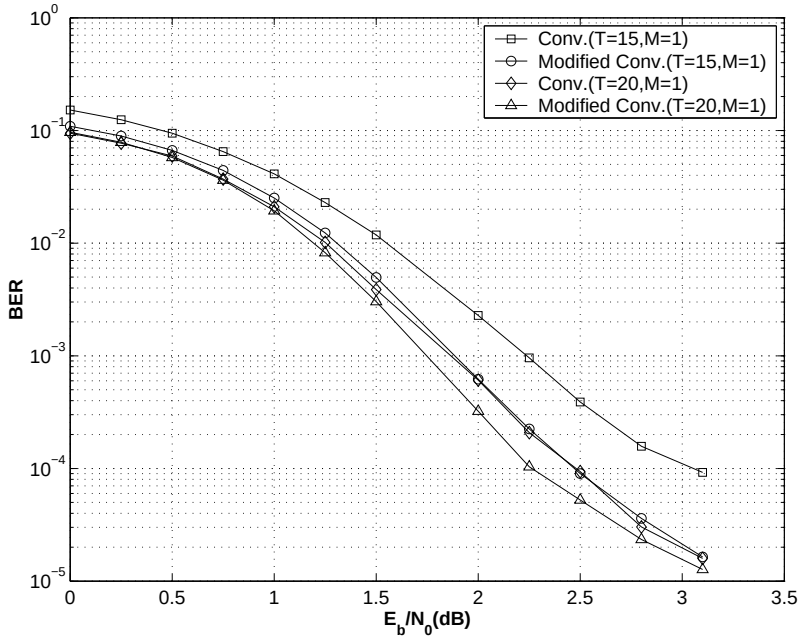

Figure 4.10 Performance of the 4- state half rate turbo code with different interleaver periods and length L=1024.

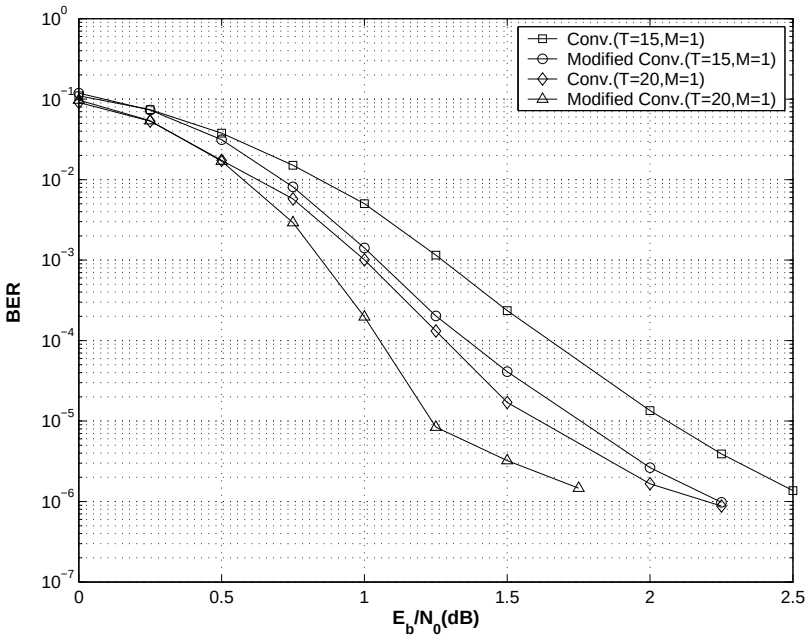

Figure 4.11 Performance of the 16- state full rate turbo code with different interleaver periods and length $L = 1024$.

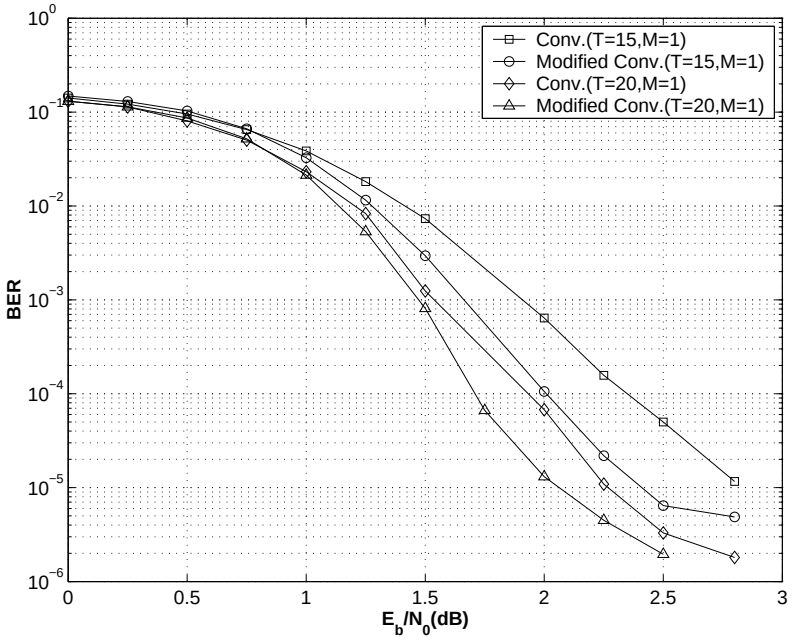

Figure 4.12 Performance of the 16- state half rate turbo code with different interleaver periods and length $L = 1024$.

In both cases the modified interleaver $(T = 15, M = 1)$ shows a similar performance to the interleaver $(T = 20, M = 1)$, while reducing the number of stuff bits by 45 percent. Applying the modified interleaver $(T = 20, M = 1)$ has improved the full and half rate turbo codes by 0.25 dB.

4.4.3 Simulation Results for Interleaver Length $L = 4096$

The performance of 4- and 16-state turbo codes with the interleaver $(T = 35, M = 1)$ have been shown in Figures 4.13 and 4.14. The results have been achieved based on 2500 simulated data blocks for each $\frac{E_b}{N_0}$ value. For the full rate 4-state turbo codes, the threshold value of the normalized SOVA is set by the value 0.15. At the error floor region, the modified interleaver has improved the full and half rate 4-state code performance by 0.5 dB and 0.4 dB, respectively. Similarly, this interleaver has improved the performance of 16-state code in the waterfall and the error floor region. For $\frac{E_b}{N_0}$=1.25 dB, the interleaver gives the BER of the full rate code less than 10^{-6}.

4.5 Chapter Summary and Conclusions

In this chapter, an efficient and simple modification to the optimized convolutional interleavers was presented. This can be accomplished by cyclical shifts of bits distributed in different interleaver lines. This increases the distance between adjacent bits of the input bitstream in the interleaved data without increasing the interleaver period. Performance of the suggested modification was examined through an analysis and simulations of different codes. In some cases, simulations confirmed that the modified interleaver outperforms the optimized interleaver, while the number of the stuff bits was reduced by 40 %. Instead of the period, increasing of the distance between adjacent bits in the interleaved data can be achieved by increasing the space value of the interleaver, i.e. M. In the next chapter, the performance of interleavers constructed with a space value greater than 1 will be verified and the relevant modifications for the applied interleavers suggested.

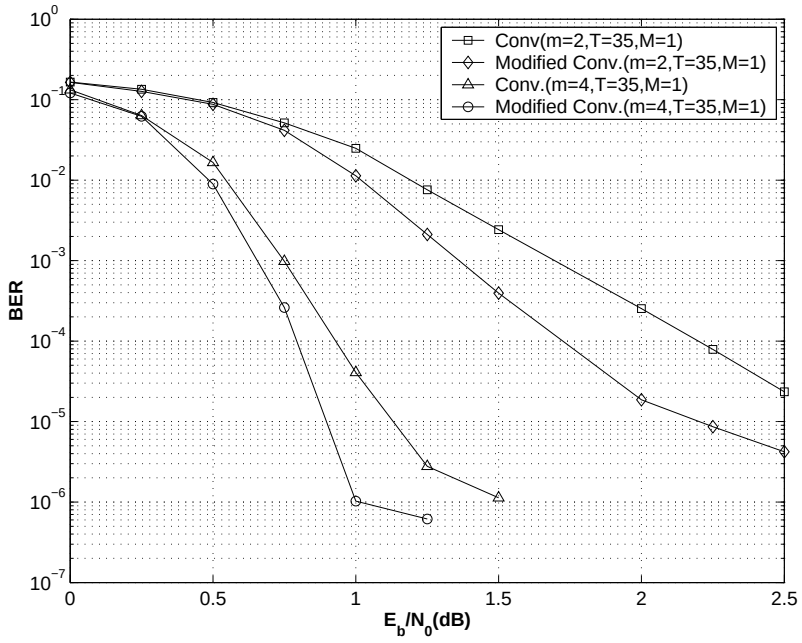

Figure 4.13 Performance of the 4- and 16- state full rate turbo codes with the interleaver$(T = 35, M = 1)$ and length $L = 4096$.

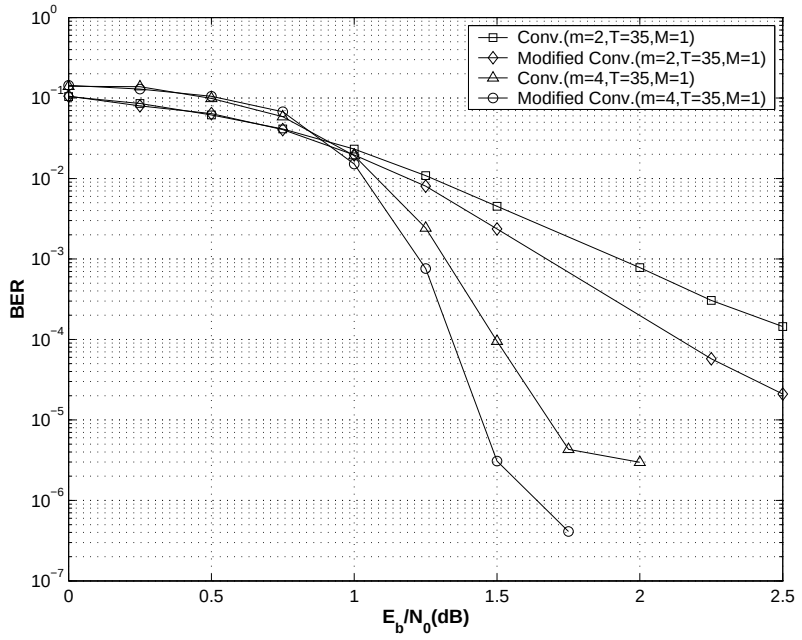

Figure 4.14 Performance of the 4- and 16- state half rate turbo codes with the interleaver $(T = 35, M = 1)$ and length $L = 4096$.

Chapter 5

Convolutional Interleavers with Different Value of the Space Parameter

5.1 Introduction

This chapter considers the influence of the space parameter of the convolutional interleaver on the turbo code performance. Considering interleaved data with similar number of stuff bits, the interleavers with a high space value can create a longer distance value between adjacent bits of input bitstreams in the interleaved data compared to interleavers with space value 1. Therefore, a higher free distance value and better performance for the code is expected.

In this Chapter, a structure of turbo code with interleavers having high space space value is explained. The codes constructed with new interleavers will be analyzed and compared with the results obtained from interleavers with the space value 1. This is accomplished based on the calculation of low weight distribution of the code. The analysis will be performed for the short and long interleaver lengths. For each proposed interleaver, suitable modifications compatible with the structure of the code will be suggested to improve the code performance with lower number of stuff bits. Finally, the conducted analysis will be confirmed by simulation results to determine the performance of the interleavers designed for different input bitstream lengths.

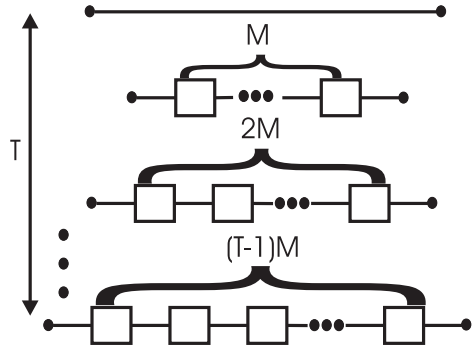

Figure 5.1 Structure of convolutional interleavers with period T and space value M.

5.2 Analysis of Turbo Codes using Interleavers with High Space Value

Period and space are known as two major convolutional interleaver parameters [106]. The input bitstreams are distributed into T parallel lines of the inteleavers. Depending on the space value M, each interleaver line has M more delay elements than the previous line. Hence, the interleaved data appear in different time slots at the interleaver output. Figure 5.1 shows the general form of the convolutional interleaver with period T and space M. Analysis on different turbo codes with optimized convolutional interleavers suggests that the resulting codes have relatively low free distance value with low multiplicities [18]. When an interleaver with a higher period is applied, the distance between the two adjacent bits of the original bitstream increases and, consequently, a higher free distance value for the code is expected.

As mentioned chapter 3, without considering the effect of zero bit deletion in the end part of the interleaver, this minimum distance between the two adjacent bits is generally governed by the product of T and M. Therefore, increasing the interleaver space value instead of its period can be considered as another way to provide a sufficient minimum distance between the adjacent bits. In order to evaluate the performance of such interleavers, they are constructed in a way resulting similar numbers of stuff bits as interleavers with the space value 1. Based on this assumption, the effect of space parameter to the turbo code performance for the short and long interleaver lengths is analyzed.

Table 5.1
Generated Minimum distance values between adjacent bits of input bitstream from different interleavers.

Space (M)	Period (T)	Minimum distance value	No. of stuff bits $T(T-1)M/2$
2	6	12	30
3	5	15	30
1	8	8	28
1	19	19	171
1	20	20	190
2	14	28	182
3	11	33	165
4	10	40	180
5	9	45	180
6	8	48	168
7	7	49	147

5.2.1 Analysis of Turbo Codes Using Short Interleaver Lengths

In turbo code applications, an interleaver parameters, i.e. period and space, are designed in such a way that produce reasonable number of stuff bits relative to the interleaver length. Therefore, the period value of an interleaver should be reduced, when it utilizes higher space in its structure. Table 5.1 shows the minimum distance for different interleavers with similar numbers of stuff bits. The table indicates that interleavers with higher space exhibit greater minimum distance values than interleavers with a space value 1.

Although the distance between two adjacent bits increases, due to the shorter periods, the distance between distributed bits in each interleaver line decreases. This distance can be specified by the value of $T-1$. When both bit 1 positions in the interleaved data with weight-2 are located in one line of the interleaver, and the generated interleaved data returns the second RSC encoder to the zero state, a low codeword weight for this encoder is achieved. Simultaneously, a similar condition occurs for the first RSC encoder because the distance of distributed bits in one of the interleaver lines is identical to the original distance in the input bitstream. Because of this and the low value of the interleaver period, the number of resulting low weight encoded data

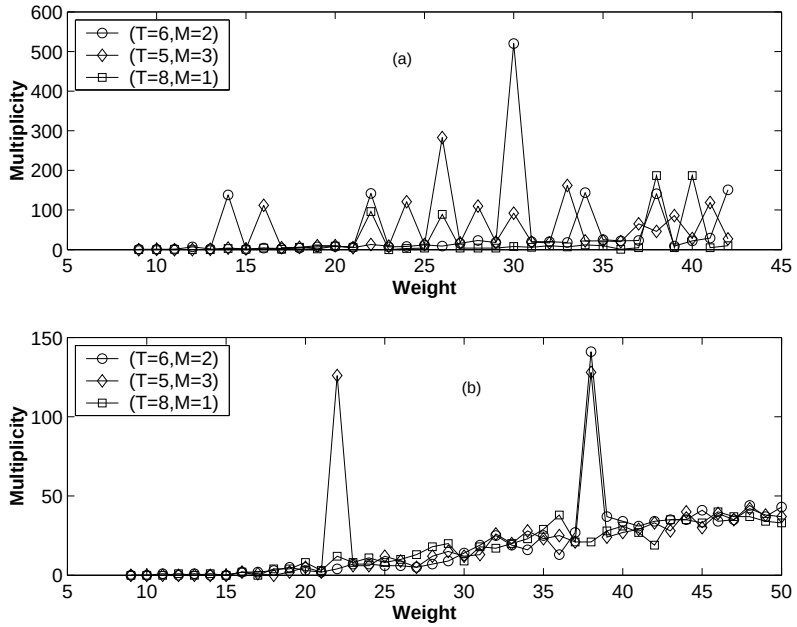

Figure 5.2 Weight-2 distribution of turbo code $(1, \frac{5}{7})$ with different interleavers. a) 4-state code $(1, \frac{5}{7})$ and b) 16-state code $(1, \frac{35}{23})$.

increases, leading to higher multiplicities of patterns with weight close to the free distance value. As a result, the performance of such codes dramatically degrades. For example, in the interleaver $(M = 2, T = 6)$, the distance between distributed bits in each interleaved data line is equal to 5. When this interleaver is used for turbo codes $(1, \frac{5}{7})$, the existence of pattern (00...001 00...0100...00)$_L$ with length L including $n = 3k + 2$ zeros $(k = 0, 1, ..., \lfloor \frac{L-4}{3} \rfloor)$ between two bit 1s will return both RSC encoders to the zero state and generate low weight codewords. Figure 5.2(a) shows weight-2 distribution of the 4-state turbo code $(1, \frac{5}{7})$ with different interleavers. For the interleaver $(T = 6, M = 2)$, some weights with high multiplicities, i.e. 22, 38 and 54 are obtained due to self-terminating patterns applied for the second RSC encoder. It is easily concluded that increasing the interleaver length contributes more self-terminating patterns to the code performance and consequently multiplicity of the mentioned weights will be increased.

Both interleavers with high space values have been generated an effective free distance value 9 with multiplicity 1, which is lower than the value 12 with multiplicity 1 calculated for the interleaver $(T = 8, M = 1)$. For the interleaver $(T = 8, M = 1)$,

the free distance of the code is calculated by weight-3 self-terminating patterns, which is value 10 and 2 multiplicities. This weight has major contribution in the code performance, especially in the error floor region [17]. The analysis of weight-3 distribution for considered interleavers with high space values gives minimum weights greater than their effective free distance values. The minimum weights 13 with 2 multiplicity and 10 with 1 multiplicity, were calculated for the interleavers $(T = 6, M = 2)$ and $(T = 5, M = 3)$, respectively. The above analysis express close performance between these interleavers and the interleaver $(T = 8, M = 1)$. It is expected that the interleavers with high space values can slightly improve the code performance for input bitstreams with higher weights compared to the interleaver $(T = 8, M = 1)$.

For the same interleavers utilized for the above example, the analysis of the 16 state turbo code $(1, \frac{35}{23})$ has been accomplished by calculation of the relevant weight-2 distributions. Since in the interleaver $(T = 5, M = 3)$, the distance between bits distributed in one column of the interleaved data is 14, an existence of patterns $(00...0100...01000)_L$ with length L including $n = 15k + 14$ $(k = 0, 1, ..., \lfloor \frac{L-16}{14} \rfloor)$ zeros between two 1s, simultaneously return both RSC encoders to the zero state and some weights with the high multiplicity is expected. Figure 5.2(b) shows the weight-2 distribution of the code for the considered interleavers. The effect of self-terminating patterns to the code performance is obviously observed for the interleaver $(T = 5, M = 3)$, which has produced a high multiplicity for the weight 22. These interleavers have been generated effective free distance values close to each other for the code. Analysis of the code with the weight-1 and weight-3 gives the similar results obtained from the weight-2 distribution. Hence, it is expected that again these interleavers create similar performance for the code.

5.2.1.1 Modifications on Interleavers with the High Space Value

Conducted analysis in the last section represents that a suitable modification is vital for an interleaver with the short period and high space value. The modification is accomplished to increase the distance between two adjacent bits that have been located in one line of the interleaver. The modification proposed for the interleaver

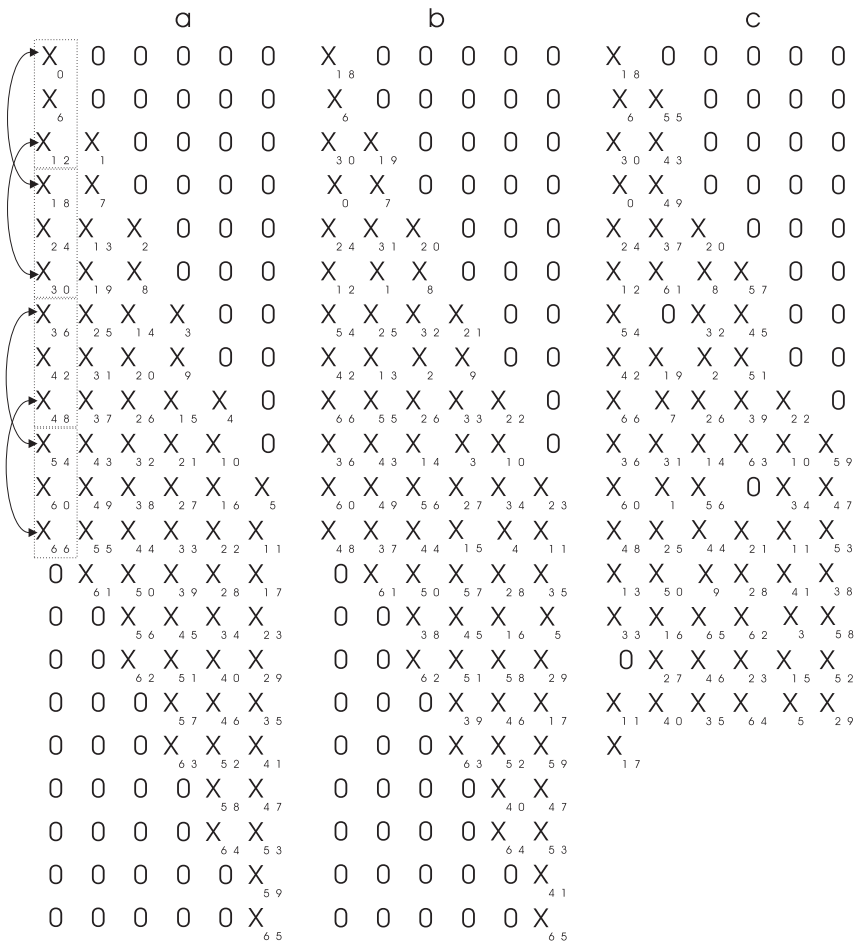

Figure 5.3 Conducted modification on the interleaver $(T = 6, M = 2)$. a) Original bit-stream, b) increasing column bits distance procedure and c) even column bits shifts equal to $5T$ and zero bit deletion from the end part of the interleaver.

$(T = 6, M = 2)$ utilized for the turbo code $(1, \frac{5}{7})$ is as follows [19]:

Three consecutive bits in one column are considered as one group. The first and third bit of each group are replaced with the adjacent bits of the next group to increase minimum distance of the interleaved bits in one column to be twice of the original distance. Then, even column bits shifting similar to the method presented in [18] is performed to provide a sufficient distance between adjacent bits that have been located in different columns. Finally, optimization is conducted by deleting stuff bits located at the end part of the interleaved data. Figures 5.3(b) and 5.3(c) show modification procedures for the proposed interleaver. Similarly, a special replacement of

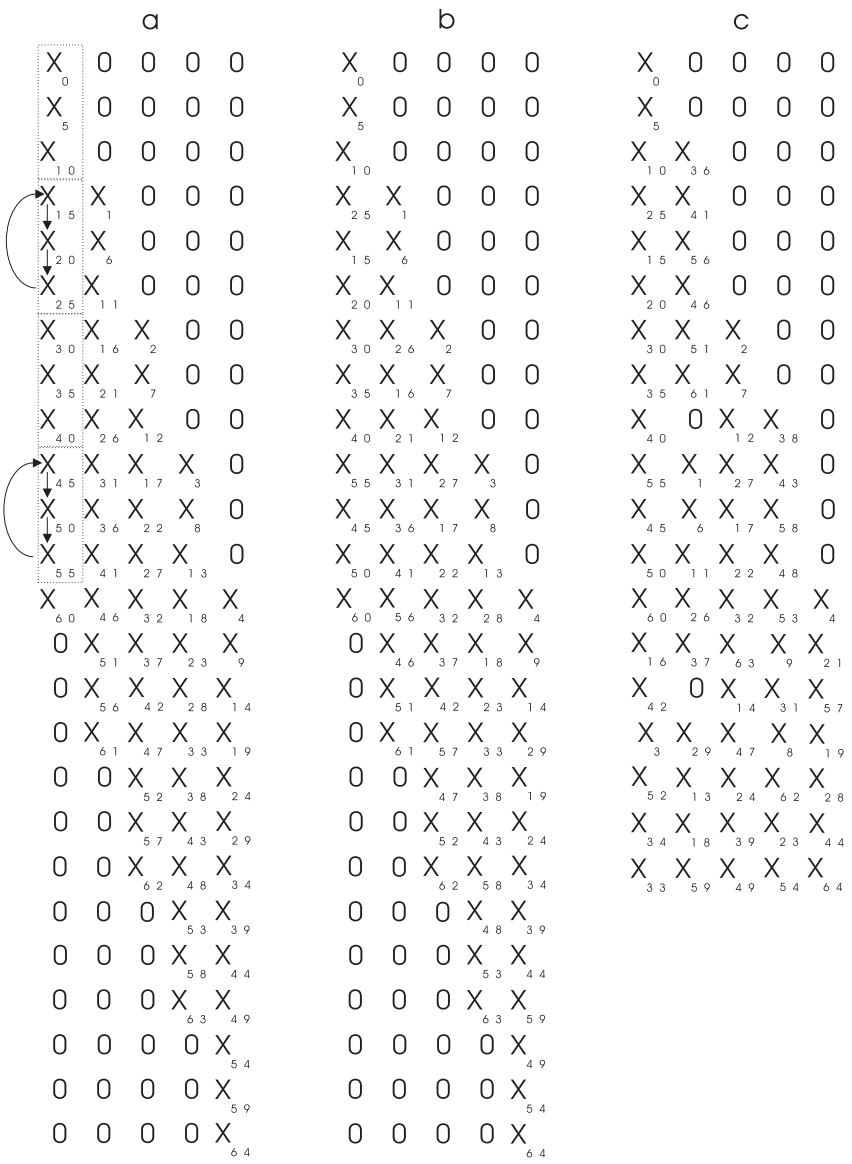

Figure 5.4 Conducted modification on the interleaver $(T = 5, M = 3)$. a) Original bit-stream, b) increasing column bits distance procedure and c) even column bits shifts equal to $6 * T$ and zero bit deletion from the end part of the interleaver.

bits located in one column of the interleaved data prior to the even column bit shifting is required for the interleaver $(T = 5, M = 3)$ utilized for the 16- state turbo code $(1, \frac{35}{23})$. In this modification, each three consecutively located bits in one column are considered as one group. Then, each bit is cyclically replaced with its adjacent bit. Finally, even column bits shifting is applied, in order to complete the modification procedure [19]. Figure 5.4 illustrates the modification process for the length

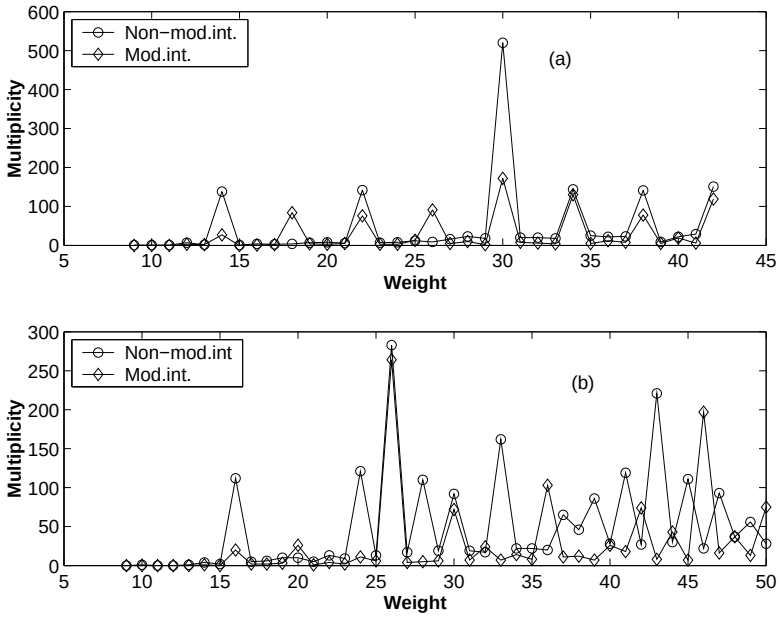

Figure 5.5 Weight-2 distribution of the 4-state turbo code with convolutional interleavers length L=169. a) Interleaver $(T = 6, M = 2)$ and b) interleaver $(T = 5, M = 3)$.

$L = 64$. For the 4-state turbo code with the interleaver $(T = 5, M = 3)$, weights with high multiplicities are generated from 1s distributed in different columns of the interleaved data. Therefore, cyclically shift of bits positioned in some columns will be enough to break relevant self-terminating patterns to those weights. Similar conclusion is obtained from the 16- state code with the interleaver $(T = 6, M = 2)$. Figures 5.5 and 5.6 show weight-2 distribution of the 4- and 16- state turbo code with the modified interleavers, respectively. In modifications, even columns of interleavers $(T = 5, M = 3)$ and $(T = 6, M = 2)$ are shifted by $15 * T$ and $12 * T$ units providing the the new interleaved data for the 4- and 16-state turbo code, respectively.

The graphs express that the modified interleavers efficiently reduce the effect of low weights with high multiplicities, while they generate higher effective free distance values with the minimum multiplicity for the codes.

5.2.1.2 Simulation Results

In this chapter, simulations are performed based on SOVA with 8 iterations, which is scaled by constant and threshold values of 0.5 and 0.1, respectively. Number

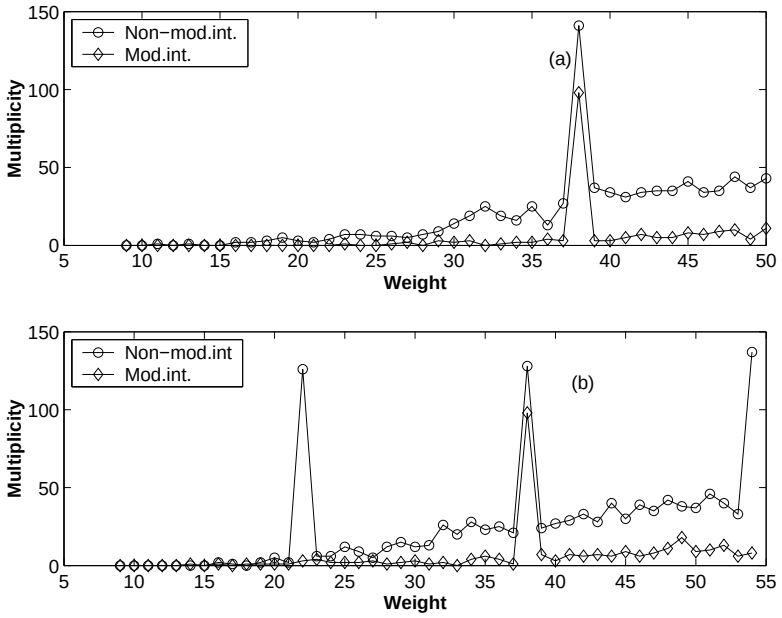

Figure 5.6 Weight-2 distribution of the 16-state turbo code with convolutional interleavers length L=169. a) Interleaver $(T = 6, M = 2)$ and b) interleaver $(T = 5, M = 3)$.

of simulated data blocks for different $\frac{E_b}{N_0}$ values in codes with different interleaver lengths are the same as the numbers applied in Chapters 3 and 4. Figures 5.7 and 5.8 show simulated results for the 4- state turbo code $(1, \frac{5}{7})$ with an interleaver length $L = 169$. The new designed interleavers have similar performance to the interleaver $(T = 8, M == 1)$, which confirm the results obtained from the analysis. The interleavers $(T = 6, M = 2)$ and $(T = 5, M = 3)$ create similar and 0.2 dB better performance than $(T = 8, M = 1)$ for the full and half rate codes, respectively. The effect of applied modification to the code performance is observed in both figures. The modified interleavers $(T = 6, M = 2)$ has been outperformed the convolutional interleaver $(T = 10, M = 1)$ by 0.25 dB, while number of stuff bits is reduced by 33 %. For these codes, the modified interleaver $(T = 5, M = 3)$ has also created a similar performance with the interleaver $(T = 10, M = 1)$. Similar results are achieved for the 16-state turbo code $(1, \frac{35}{23})$. Again, interleaver designed with higher space values have similar operation with the interleaver $(T = 8, M = 1)$ for the full and half rate codes. In both figures, modified interleavers have been outperformed the interleaver $(T = 10, M = 1)$ with lower number of stuff bits, which is specifically evident in the error floor regions.

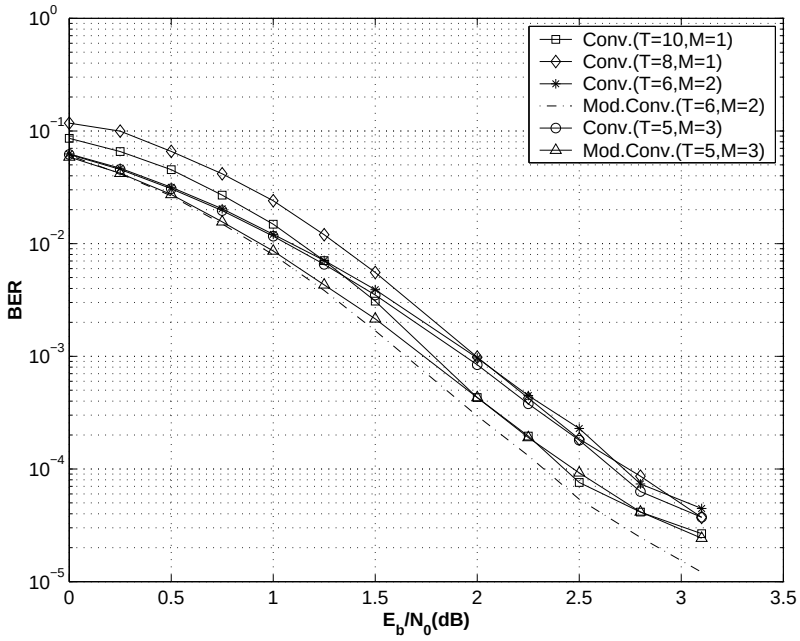

Figure 5.7 Simulation results for 4 state full rate turbo codes with interleavers length $L = 169$.

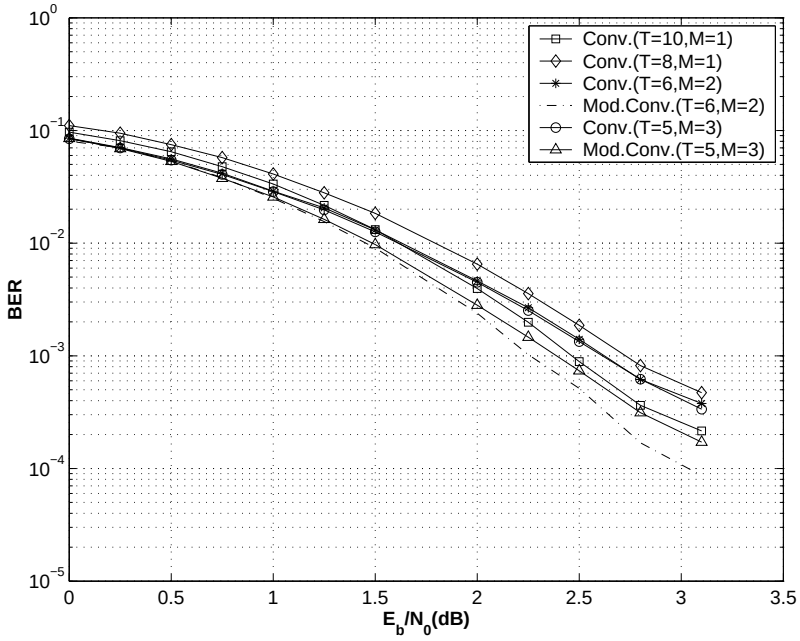

Figure 5.8 Simulation results for 4 state half rate turbo codes with interleavers length $L = 169$.

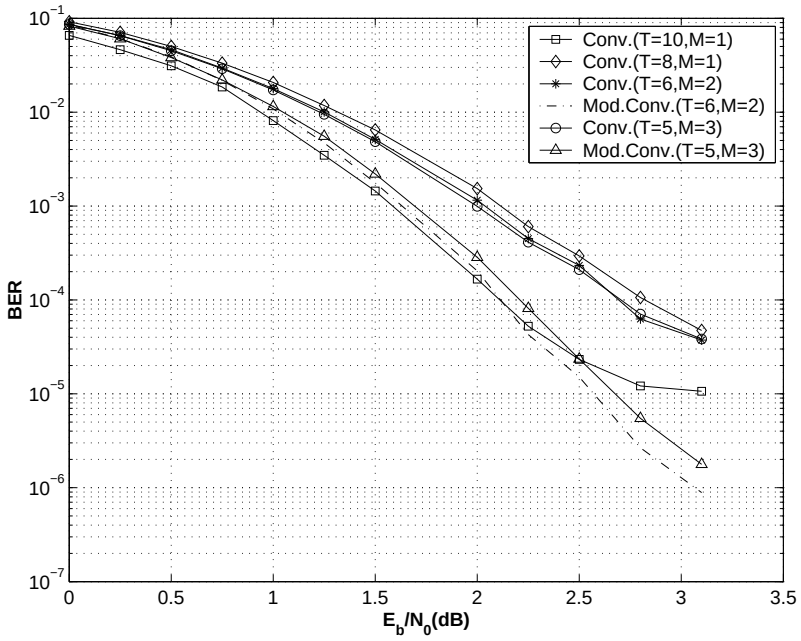

Figure 5.9 Simulation results for 16- state full rate turbo codes with interleavers length $L = 169$.

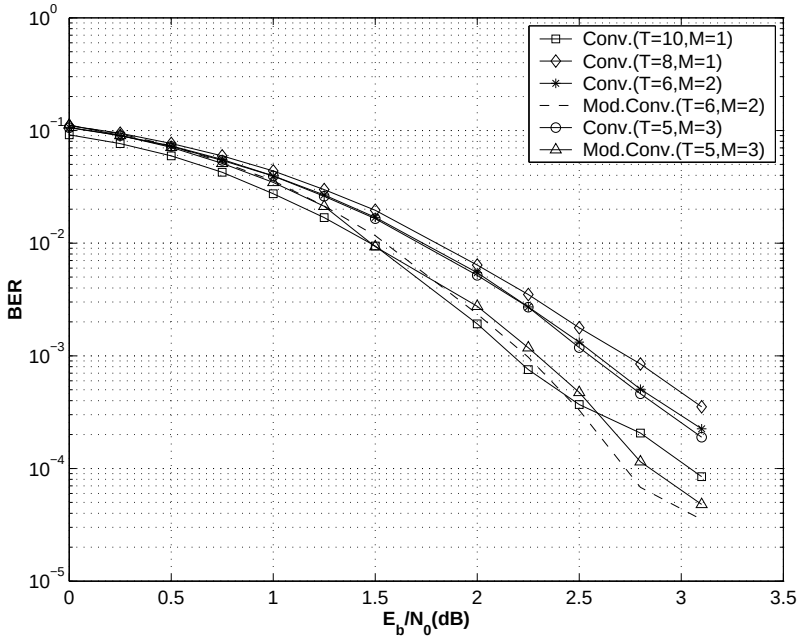

Figure 5.10 Simulation results for 16- state half rate turbo codes with interleavers length $L = 169$.

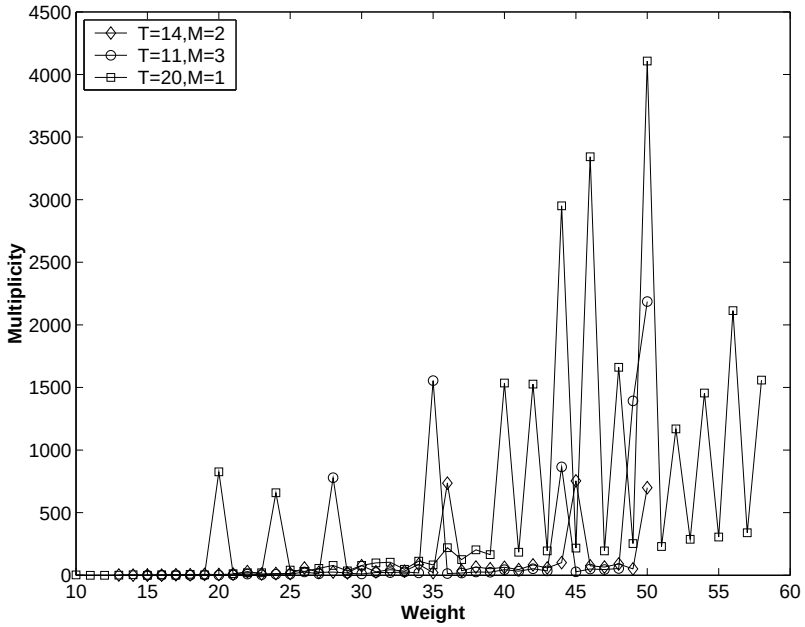

Figure 5.11 Estimated distance spectrum of the 4-state turbo code for interleavers $(T = 14, M = 2)$,$(T = 11, M = 3)$ and $(T = 20, M = 1)$ with length $L = 1024$.

5.2.2 Analysis of Turbo Codes Using Long Interleaver Lengths

The analysis is performed for the 4- and 16-state turbo codes with interleaver lengths $L = 1024$ and $L = 4096$. The codes have been analyzed based on determination of their estimated distance spectrum, which has been illustrated in Figure 5.11. Based on the algorithm presented in [17], the distance spectrum has been achieved by calculating the weight distribution of the code from self-terminating input bitstreams with weight no greater than 4 on the code performance. The obtained graphs indicate that the interleaver $(T = 11, M = 3)$ has a weaker performance than the interleaver $(T = 14, M = 2)$ breaking self-terminating patterns. In comparison with the interleaver $(T = 20, M = 1)$ both interleavers with high space values have been generated higher minimum weights with low multiplicities. Hence, it is expected that they provide better performance than the interleaver $(T = 20, M = 1)$ for the code.

Analysis of the 16-state tubo code with interleaver $(T = 35, M = 1)$ gives a free distance value 16 with multiplicity 1, when effect of tail bits is considered for the weight-1 input bitstream. For the interleaver $(T = 20, M = 3)$, the minimum weight

Table 5.2 Shifting unit values for modified turbo codes with different interleavers.

State Code	Interleaver specifications	Interleaver length	Shift Unit value
4	T=6,M=2	167	13*T
4	T=5,M=3	167	12*T
16	T=6,M=2	167	13*T
16	T=5,M=3	167	10*T
4	T=14,M=2	1024	15*T
4	T=11,M=3	1024	7*T
16	T=14,M=2	1024	10*T
16	T=11,M=3	1024	16*T
4	T=20,M=3	4096	21*T
16	T=20,M=3	4096	9*T

30 with 1 multiplicity from weight-3 self-terminating patterns has been calculated. This is an estimated free distance value from some of low weight-3 self-terminating patterns. In the analysis, effect of weight-1 and weight-2 self-terminating patterns have been also considered, which produce weights greater than 30 for the code. It is expected that the interleaver $(T = 20, M = 3)$ due to its higher free distance value can improve the code performance compared to the interleaver $(T = 35, M = 1)$.

For these interleavers, modifications are accomplished by cyclically shift of even column bits. Finding an optimum shift value to improve the code performance with the long interleaver lengths is achieved through several code analysis with different shift values. In each analysis, the code performance is verified how the modified interleaver can break self-terminating patterns and provide a higher minimum weight with low multiplicity compared to other shift values. For each interleaver, the value providing the best performance for the code is selected. Table 5.2 gives even column bit shift values for each utilized interleaver.

5.2.2.1 Simulation Results

Figures 5.12 and 5.13 show simulation results for the full and half rate 4 state turbo codes with the interleaver length $L = 1024$. In this code, reducing the number of stuff bits by 4%, the modified interleaver $(T = 14, M = 2)$ creates 0.5 and 0.25 dB better performance than the interleaver $(T = 20, M = 1)$, respectively. Again, the

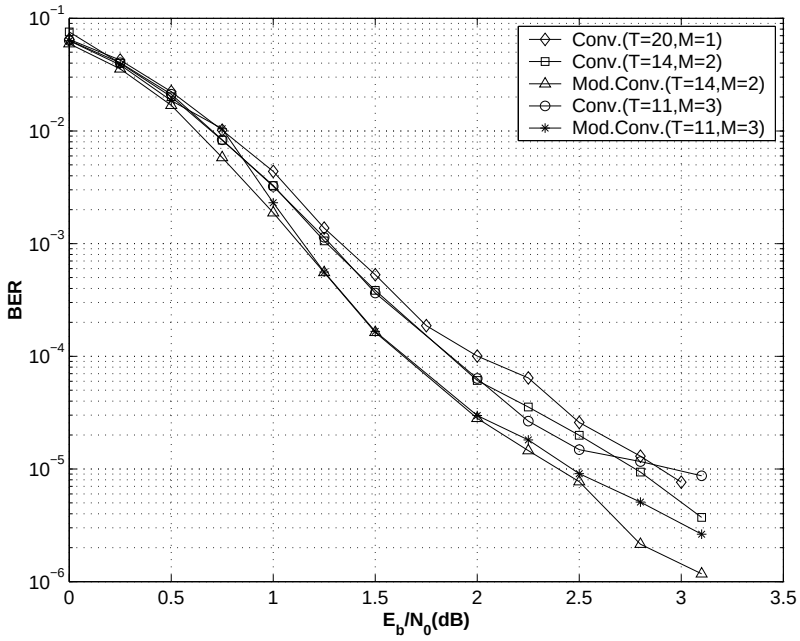

Figure 5.12 Simulation results for 4- state full rate turbo codes with interleavers length $L = 1024$.

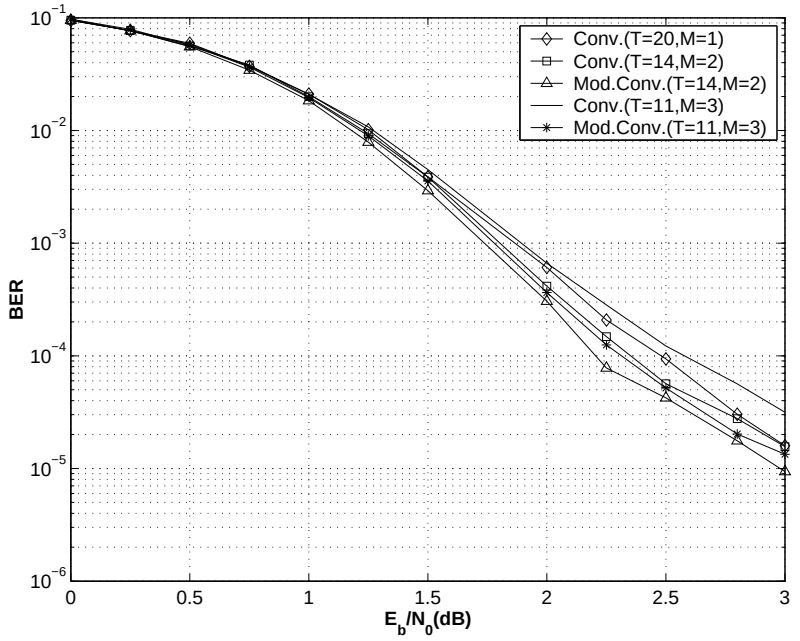

Figure 5.13 Simulation results for 4- state half rate turbo codes with interleavers length $L = 1024$.

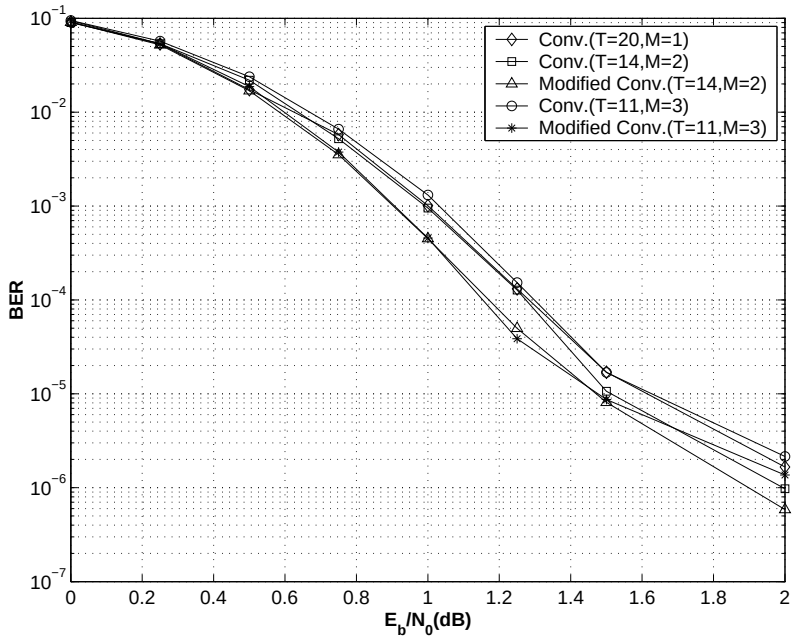

Figure 5.14 Simulation results for 16- state full rate turbo codes with interleavers length $L = 1024$.

results obtained from two interleavers with high space values confirm the obtained results from the analysis. The poor behavior of the interleaver $(T = 11, M = 3)$ is clearly observed in the error floor region of the code.

Simulation of the full and half rate 16- state turbo code is illustrated in Figures 5.14 and 5.15. Similarly to results obtained for the 4-state code, the interleaver $(T = 14, M = 2)$ has best performance for the full rate code. In comparison with the interleaver $(T = 20, M = 1)$, the modified interleavers $(T = 14, M = 2)$ and $(T = 11, M = 3)$ with a lower number of stuff bits improve the half rate code performance by 0.25 dB in the error floor region.

Figures 5.16 and 5.17 illustrate the 4- and 16- state code performance for the interleaver $(T = 20, M = 3)$ and $(T = 35, M = 1)$ with length $L = 4096$. In each figure, the interleavers $(T = 20, M = 3)$ provide similar or even better performance than interleavers $(T = 35, M = 1)$ with lower number of stuff bits. This is especially evident for the full rate 16-state turbo code, where the interleaver $(T = 20, M = 3)$ outperforms the interleaver $(T = 35, M = 1)$ by 0.25 dB.

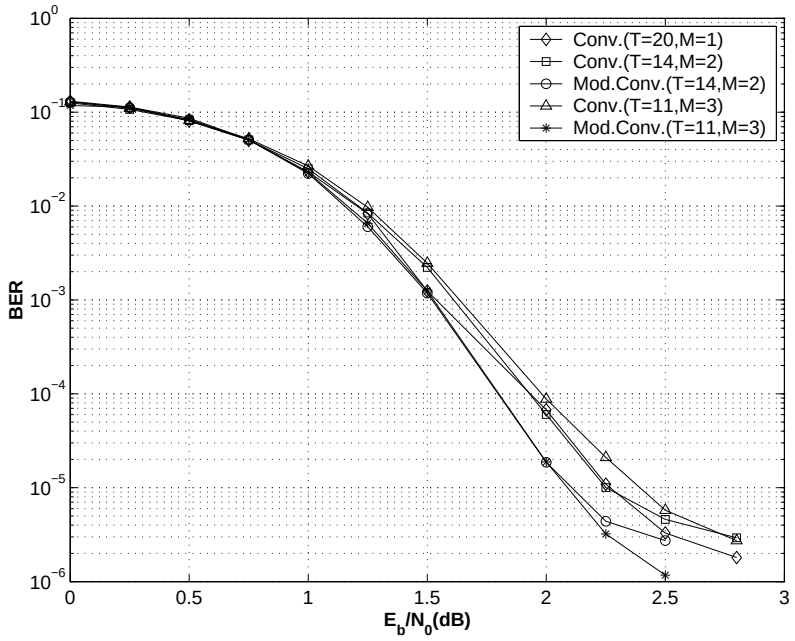

Figure 5.15 Simulation results for 16- state half rate turbo codes with interleavers length $L = 1024$.

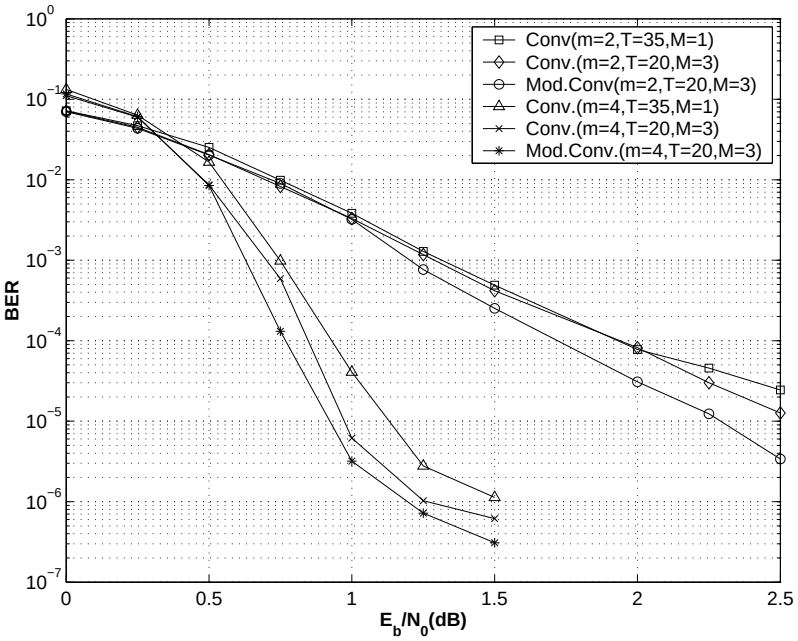

Figure 5.16 Simulation results for 4- and 16- state full rate turbo codes with interleavers length $L = 4096$.

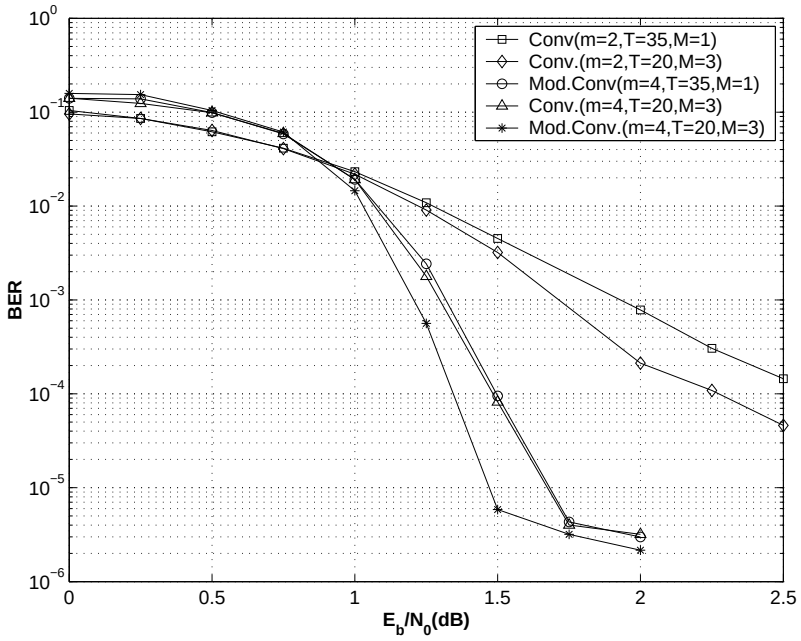

Figure 5.17 Simulation results for 4- and 16- state half rate turbo codes with interleavers length $L = 4096$.

5.3 Chapter Conclusion and Summary

In this chapter, the performance of turbo codes with the optimized convolutional interleaver having the space value greater than 1 was examined. The obtained analysis and simulation results for different codes showed that due to a short period, the proposed interleaver did not sufficiently improve the code performance compared to the interleaver with space value 1. The good performance of the interleavers with space value greater than 1 was achieved only for the long interleaver lengths. This was confirmed through simulation of codes with interleaver lengths $L = 1024$ and $L = 4096$, where designed interleavers $(T = 14, M = 2)$ and $(T = 20, M = 3)$ with space values 2 and 3 outperform the interleavers $(T = 20, M = 1)$ and $(T = 35, M = 1)$ in the error floor region, respectively. Some modifications related to these interleavers have been proposed to improve the code performance with reduced number of stuff bits. This was evident for the short interleaver lengths, where the modified interleavers $(T = 6, M = 2)$ and $(T = 5, M = 3)$ with 35 % lower number of stuff bits have better performance than the interleaver $(T = 10, M = 1)$.

Chapter 6

Generalized Convolutional Interleaver and Its Performance in Turbo Codes

6.1 Introduction

From Chapter 5, it can be concluded that in the case of a similar number of stuff bits for turbo codes, interleavers with higher space values and shorter periods usually have better or similar performance than interleavers with higher periods and the space value 1 [19]. However, when the space value increases, the period should be decreased to maintain the number of stuff bits in an acceptable range. As a result, low weights with high multiplicities are created in the encoded data and the code performance at the error floor region is degraded. Instead of constructing an interleaver with a fixed space value, convolutional interleavers having a variable space value in their structures can be utilized. This gives more flexibility to allocate a different number of memories to each interleaver line to design interleavers with a desired period and the sufficient overall number of stuff bits. Convolutional interleavers designed with these properties are referred to as ”generalized convolutional interleavers” [43, 119].

In this chapter, application of generalized convolutional interleavers tailored to the structure of turbo code is presented. These interleavers are constructed based on

Table 6.1 Weight-2 self-terminating patterns for the 4- and 16- state turbo codes.

Turbo code state	Patterns
4	$1\ \underbrace{0\ 0\ \bullet\bullet\bullet\ 0}\ 1$ $n=3k+2\ \ k=0,1,\ldots,\ \lfloor (L-4)/3 \rfloor$
16	$1\ \underbrace{0\ 0\ \bullet\bullet\bullet\ 0}\ 1$ $n=15k+14\ \ k=0,1,\ldots,\ \lfloor (L-16)/15 \rfloor$

weight-2 distribution of turbo codes. For each interleaver, suitable modification is accomplished to improve the code performance with lower number of stuff bits. Analysis of codes and their iterative decoding confirm that these interleavers have similar or even better performance than previously proposed convolutional interleavers. It can be noticed that with careful design, they provide close performance to the well-known block interleavers.

6.2 Generalized Convolutional Interleavers for Turbo Codes

A good generalized convolutional interleaver for turbo code applications is designed in a way that prohibits generation of self-terminating patterns for the second RSC code. Therefore, it is necessary to design the interleaver based on the structure of the RSC encoder, which produces a reasonable number of stuff bits. This assumption leads to the application of a similar number of memories for some interleaver lines.

An analysis of turbo codes with generalized convolutional interleavers may be accomplished by calculating the weight-2 distribution of the code. In this analysis, performance of the code is verified on the basis of self-terminating patterns. Then, the interleaver parameters are set such that prohibit generation of these self-terminating patterns. Table 6.1 gives relevant self-terminating patterns for the 4- state $(1, \frac{5}{7})$ and the 16- state $(1, \frac{35}{23})$ turbo codes. Considering these assumptions, reliable interleavers for these codes are designed.

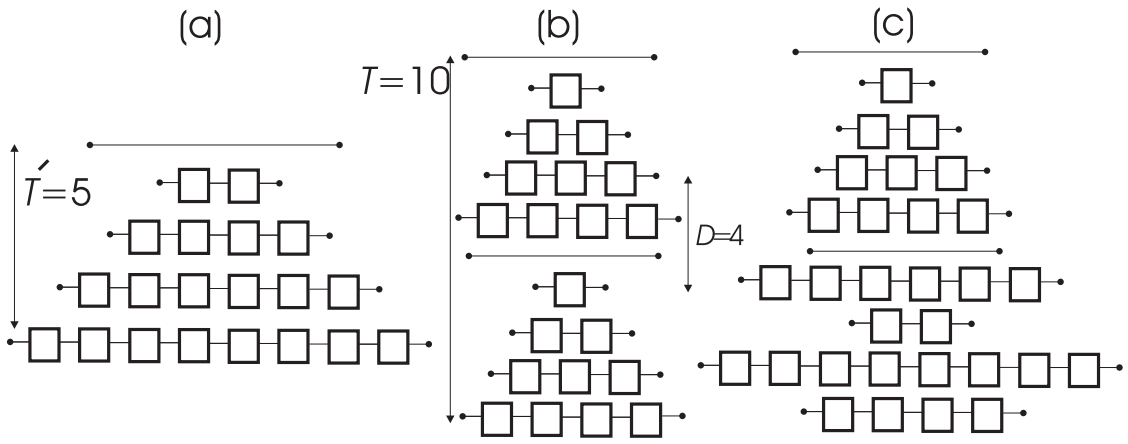

Figure 6.1 Making an interleaver with $T = 10$ and with 32 stuff bits from two-level jointed interleaver $(T' = 5, M = 1)$ for the 4-state turbo code $(1, \frac{5}{7})$.

6.2.0.2 Interleavers for 4-State Turbo Code $(1, \frac{5}{7})$

An interleaver with a specification of $(T', 2M)$ can be represented by a two-level joint interleaver, where each level has a specification (T', M). This property makes it possible to obtain an interleaver whose period is twice that of the original interleaver, i.e. $T = 2T'$. For example, as shown in Figure 6.1(a) and 6.1(b), the interleaver $(T' = 5, M = 2)$ produces similarly interleaved data as the two-level joint interleaver $(T' = 5, M = 1)$. For the 4- state $(1, \frac{5}{7})$ turbo code, in order to prohibit the generation of self-terminating patterns for the second RSC encoder, the separation between data distributed in one interleaver line, i.e. $T - 1$, is set to be different to the n values of Table 6.1. In this interleaver, the separation between two interleaver lines having identical number of memories, is set by $D = \frac{T}{2} - 1$ value. This value should also be set differently to the n values of Table 6.1.

Considering the two-level joint interleaver, a generalized interleaver is constructed by increasing the number of interleaver memories for some lines providing a sufficient number of stuff bits. A possible solution would be to increase the number of memories in the interleaver lines $\frac{T}{2} + i$ $(i = 2, 4, ..., T)$ by i units more than the number of memories allocated for the interleaver line $\frac{T}{2}$. Figure 6.1(c) illustrates a generalized convolutional interleaver with the period $T = 10$ and 32 stuff bits suitable for the 4-state turbo code $(1, \frac{5}{7})$.

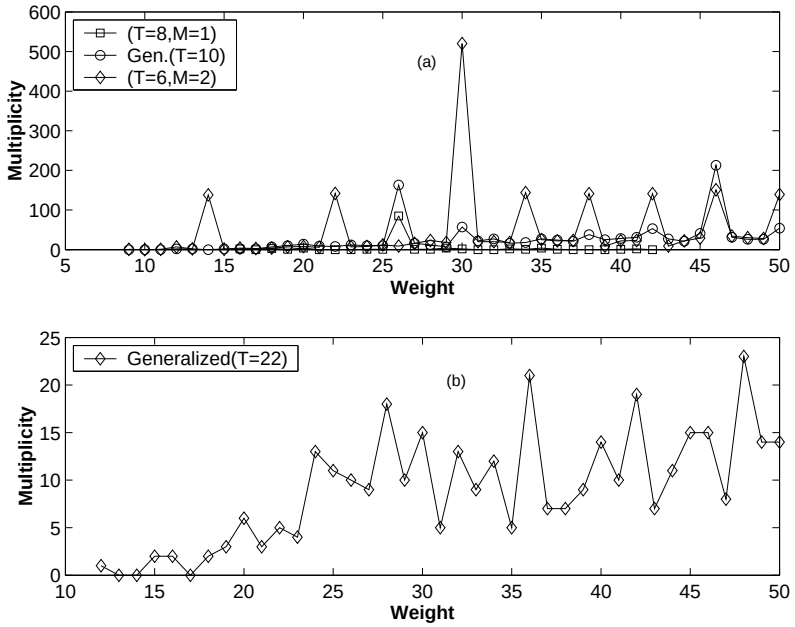

Figure 6.2 Performance of 4-state turbo code $(1, \frac{5}{7})$ with Generalized interleavers. a) Weight-2 distribution of the code for interleaver length $L = 169$ and b) estimated distance spectrum of the code for the generalized convolutional interleaver $(T = 22, L = 1024)$.

6.2.1 Analysis of 4-state Turbo Code $(1, \frac{5}{7})$ Using Generalized Convolutional Interleavers

The performance of generalized convolutional interleaver $T = 10$ is compared with the interleavers having the fixed space values. Figure 6.2(a) shows weight-2 distribution of turbo code for different interleavers with the length $L = 169$. It is observed that with a similar number of stuff bits, the generalized interleaver performs better than the interleavers $(T = 6, M = 2)$ and $(T = 8, M = 1)$ to reduce multiplicity of weights generated from self-terminating patterns. For the generalized interleaver $T = 10$, weight 26 has a relatively high multiplicity compared to other weights. The free distance of the code with the generalized convolutional interleaver is achieved from weight-3 self-terminating pattern $(00..01010100..0)_{L=169}$ by value 12 and 1 multiplicity. The major contribution of this low free distance value to the code performance will be observed in the error floor region of the code.

Figure 6.2(b) shows an estimated distance spectrum of turbo codes with interleaver

length $L = 1024$. This distance spectrum has been achieved from low weight self-terminating patterns no greater than 4. Although the designed interleaver performs well to break low weight self-terminating patterns but produces the low free distance value 12 with 1 multiplicity, which degrades the code performance.

6.2.2 Interleavers for 16-State Turbo Code $(1, \frac{35}{23})$

Similarly to the designed interleaver for the 4-state turbo code, two-level joint interleavers (T', M) are constructed from the interleaver $(T', 2M)$, whose period and D values are set differently to the n' values of Table 6.1. Due to a higher constraint length of this code than that of the 4-state code, the RSC encoders are returned to the zero state by a higher number of zeros between two 1s. This property makes it possible to design interleavers with a relatively high period close to n's. However, as mentioned before, this increment will increase the number of stuff bits. This limitation leads to a compromise through the selection of suitable values of period, D, and the number of stuff bits. In order to make an interleaver with a high period and reasonable number of stuff bits, the space parameter of each interleaver as part of the two-level joint interleaver is considered to be equal to 2 $(M = 2)$. Finally, the last line of the interleaver is deleted to reduce the overall number of stuff bits. Figure 6.3 shows structure of the generalized convolutional interleaver $(T = 9)$.

The above basic interleaver structures can be extended to an interleaver with higher periods. These are applicable for longer input bitstreams. Depending on the chosen interleaver period, memory combinations and the overall numbers of stuff bits are altered. Table 6.2 shows the proposed generalized convolutional interleavers for different periods and the corresponding numbers of stuff bits, which have been applied for the simulations in the next section. In this table, the numbers presented in the parentheses represent the number of memories in each interleaver line.

6.2.3 Analysis of 16-state Turbo Code $(1, \frac{35}{23})$ Using the Generalized Convolutional Interleaver

Figure 6.2 shows weight-2 distribution of the code for interleavers with the length $L = 169$. In comparison with other weight-2 distributions obtained from interleavers

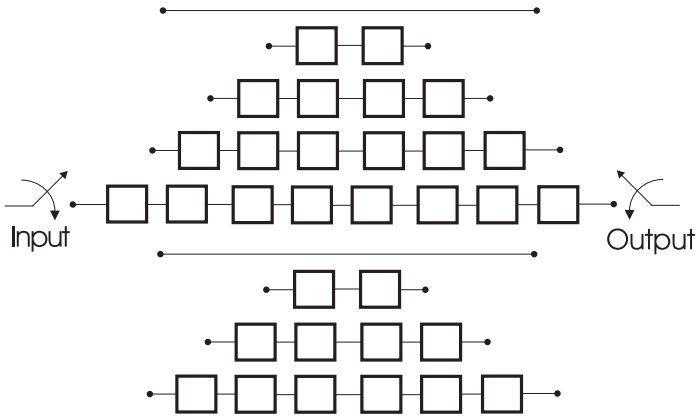

Figure 6.3 Generalized convolutional interleaver structure with $T = 9$ for the 16-state $(1, \frac{35}{23})$ turbo code.

Table 6.2
Generalized convolutional interleaver structures for the 4- and 16- state codes with different lengths.

Code state	Interleaver period	Generalized interleaver	No.of stuff bits
4	10	(0,1,2,3,4,0,6,2,8,4)	30
4	22	(0,1,2,3,4,5,6,7,8,9, 10,0,12,2,14,4,16,6,18 8,20,10)	165
16	9	(0,2,4,6,8,0,2,4,6)	32
16	11	(0,2,4,6,8,10,0,2, 4,6,8)	50
16	19	(0,2,4,6,8,10,12,14, 16,18,0,2,4,6,8,10, 12,14,16)	162

$(T = 5, M = 3)$ and $(T = 5, M = 4)$, it is clearly observed that the generalized interleaver has a better performance to break low weight self-terminating patterns[1]. It is concluded that, not only the last line deleted from the two-level joint interleaver

[1]The interleaver $(T = 5, M = 4)$ has been selected in this analysis, to compare the performance of the generalized interleaver $T = 9$ with the interleaver for which it has been developed.

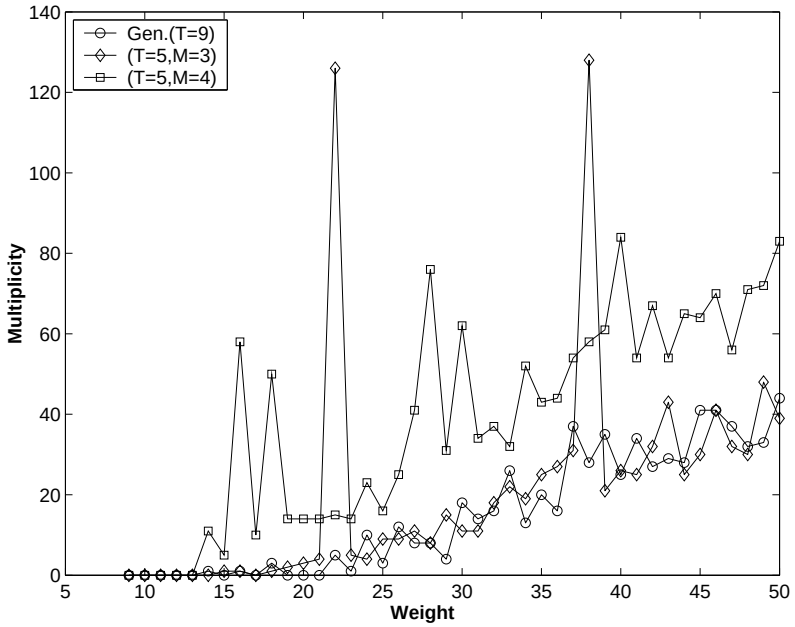

Figure 6.4 Weight-2 distribution of the 16-state turbo code $(1, \frac{35}{23})$ for interleavers with length $L = 169$.

$(T' = 5, M = 2)$, reduce the number of stuff bits, but also prohibits generation of the low weight self-terminating patterns for the second RSC encoder. Analysis of the code from some low weight-3 self-terminating patterns does not give any weight with high multiplicities or a weight lower than the effective free distance value of the code. weight with high multiplicity. In higher input bitstream length $L = 1024$, distance spectrum of the code with the generalized interleaver $T = 19$ has been calculated and compared with other interleavers. Figure 6.5 shows distance spectrum of 16-state code with different interleavers. In this analysis, These distance spectrums have been obtained from self-terminating patterns with weight no grater than 3. The results give the free distance value 18 for the generalized interleaver $T = 19$. This value is greater than the free distance values computed for the interleavers $(T = 11, M = 3)$ and $(T = 10, M = 4)$. Hence, an improvement for turbo code with this interleaver is expected.

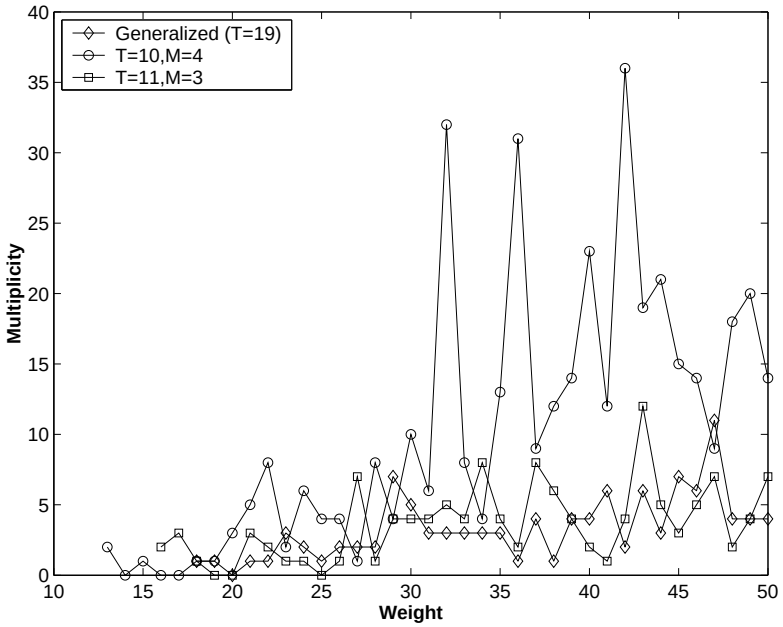

Figure 6.5 Distance spectrum of the 16-state turbo code $(1, \frac{35}{23})$ for interleavers with length $L = 1024$.

6.3 Simulation Results

The performance of the generalized convolutional interleavers with lengths $L = 169$ and $L = 1024$ was verified in 4- and 16- state turbo codes. This is accomplished by SOVA as selected iterative turbo decoding method with 8 iterations. Again, SOVA is scaled by constant and threshold values 0.5 and 0.1. 60,000 and 10,000 blocks were considered in the simulations of codes with the interleaver lengths $L = 169$ and $L = 1024$, respectively. The performance of the generalized convolutional interleavers was compared to the basic convolutional interleavers proposed in previous chapters and most conventional block interleavers. Row-column and semi-random interleavers are considered for the short and long data block, respectively. Generalized convolutional interleavers are designed in such a way that produce a similar number of the stuff bits to other presented convolutional interleavers.

For the generalized convolutional interleavers, modification is conducted to improve the code performance with lower number of stuff bits. Modification shifts bits located in some interleaver lines by the specific value, determined on the basis of the applied

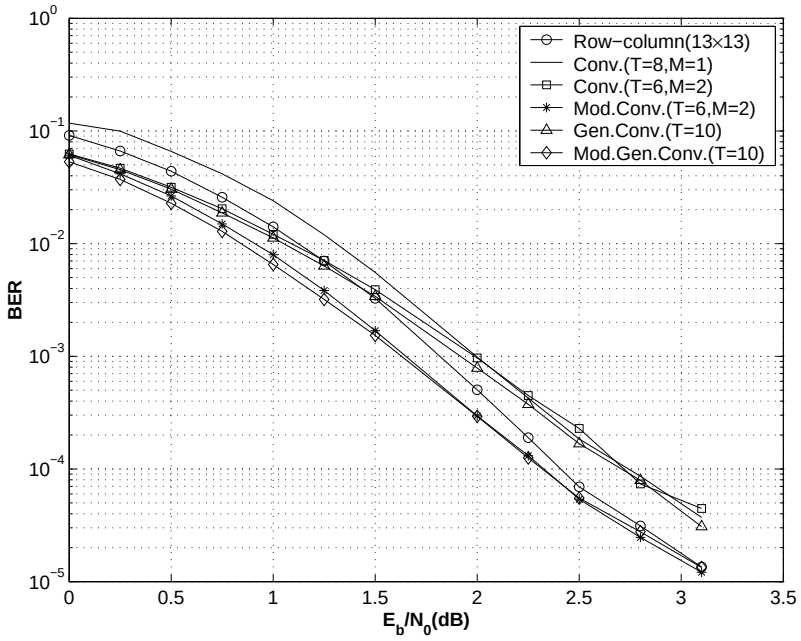

Figure 6.6 Performance of the 4- state full rate turbo code with interleavers length $L = 169$.

code and the interleaver structure [18] [19]. Table 6.3 gives the specification of the modifications for different turbo codes.

6.3.1 Simulation Results for Turbo Codes Using Interleaver Length $L = 169$

Figure 6.6 shows simulation results for the 4- state turbo codes with the length $L = 169$. The plots show that the generalized interleaver has similar performance to the interleaver $(T = 8, M = 1)$. This is due to existence of low free distance value for the code from both interleavers, which have been already analyzed. After modification, this interleaver has 0.1-0.25 dB better performance than the interleaver $(T = 10, M = 1)$, while the number of stuff bits is reduced by 33%. With the lower complexity in modification, the modified generalized interleaver $(T = 10)$ provides similar behavior to the interleaver $(T = 6, M = 2)$[2]. In comparison with the row-column interleaver (13×13), the generalized interleaver provides better performance in the waterfall region. For the half rate code, as shown in Figure 6.7,

[2]The modifications of interleavers $(T = 6, M = 2)$ for the 4-state code and $(T = 5, M = 3)$ for the 16-state code have been presented in Chapter 5

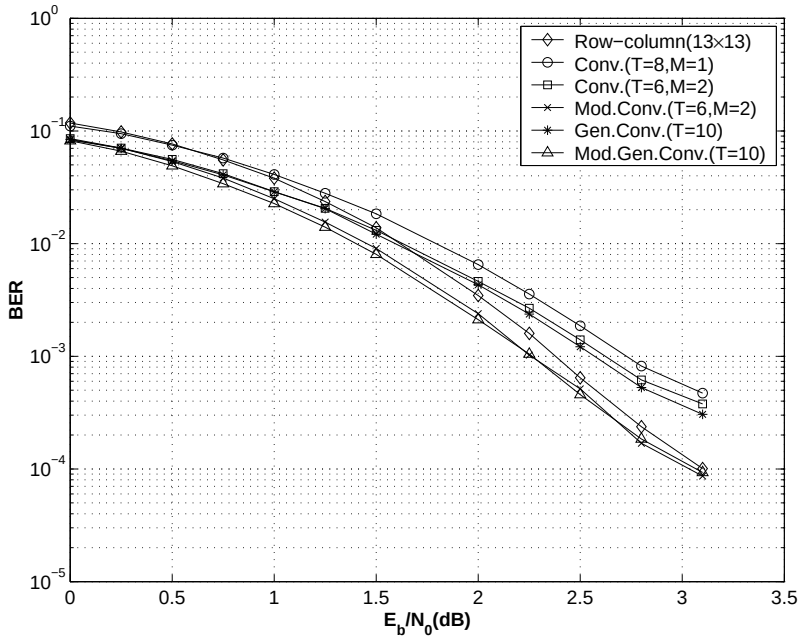

Figure 6.7 Performance of the 4- state half rate turbo code with interleavers length $L = 169$.

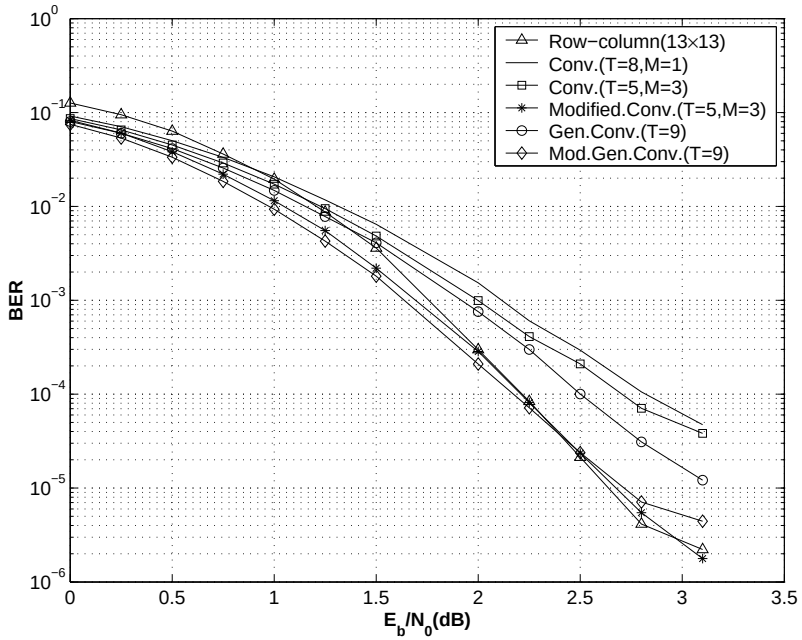

Figure 6.8 Performance of the 16- state full rate turbo code with interleavers length $L = 169$.

these generalized convolutional interleavers have better performance than that of the interleaver $(T = 8, M = 1)$ by 0.2 dB. The graphs express that contributing lower

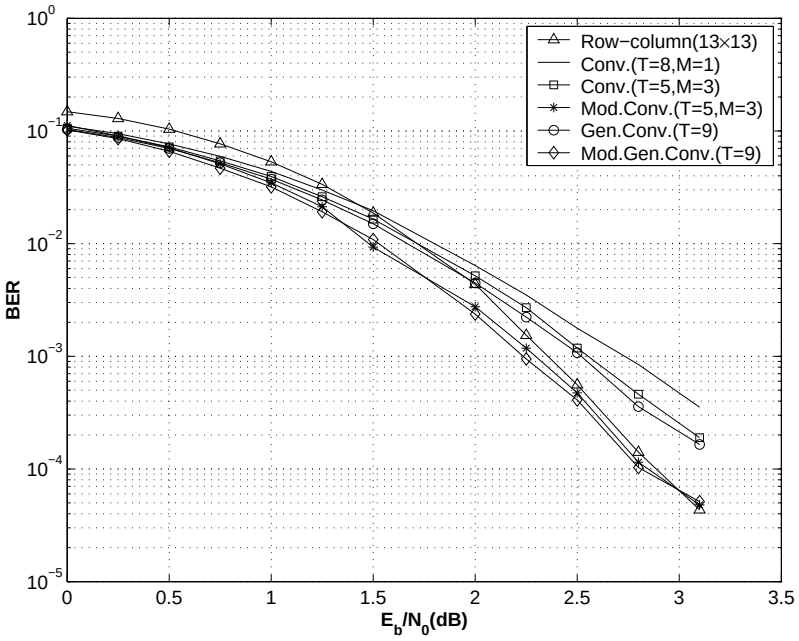

Figure 6.9 Performance of the 16- state half rate turbo code with interleavers length $L = 169$.

number of bits in the modification, results in performance of the modified generalized interleaver similar to the modified interleaver $(T = 6, M = 2)$ presented in [19]. Figure 6.8 illustrates the performance of the full rate 16-state turbo code. Again, The interleaver $(T = 9)$ outperforms the row-column interleaver (13×13) in the waterfall region.

In error floor region, both interleavers have the similar performance. This interleaver has been also improved 0.25 dB the code performance compared to the interleaver $(T = 8, M = 1)$. The modified interleaver $T = 9$ with 29% lower number of stuff bits generates a similar performance to the interleaver $(T = 10, M = 1)$.

The generalized interleaver $T = 9$ contributes a lower complexity in its modification to provide similar behavior to the $(T = 5, M = 3)$ and row-column interleavers. In Figure 6.9, results obtained for the half rate 16- state turbo codes have been shown, confirming the result achieved previously for the full rate 16-state code.

Table 6.3
Specifications of Modified generalized convolutional interleavers for 4- and 16- state turbo codes.

State Code	Interleaver Period	Shifted interleaver lines	Shifted Unit value
4	T=10	(2,3,4,5,7,9)	8*T
16	T=9	(2,4,7,9)	5*T
4	T=22	(3,9,13,15,17,21)	13*T
		(4,5,6,10,11)	8*T
16	T=19	even lines	13*T
		line 9	5*T

6.3.2 Simulation Results for Turbo Codes Using Interleaver length $L = 1024$

Figures 6.10 and 6.11 show performance of the full and half rate 4- state turbo codes $(1, \frac{5}{7})$ for interleavers with the length $L = 1024$. For both codes, with similar number of stuff bits, the generalized interleaver $T = 19$ creates better performance than that of the interleaver $(T = 11, M = 3)$ in the error floor region, which confirms the analysis presented in Figure 6.5.

Similar simulations have been conducted for the full and half rate 16-state code with the generalized convolutional interleaver $T = 19$, which have been illustrated in Figures 6.12 and 6.13, respectively. For the full rate, the generalized interleaver produces a similar performance to the interleaver $(T = 20, M = 1)$, while number of stuff bits has been reduced by 10%. In comparison with the modified interleaver $(T = 11, M = 3)$, the modified generalized interleaver improves the full and half rate code performance by 0.15 dB in the waterfall and error floor regions.

6.3.2.1 Comparison with Block Interleavers

Finally, performance of convolutional interleavers presented in this and the previous chapters is compared with semi-random interleavers. Comparisons are conducted between the modified convolutional interleavers and the semi-random interleavers with length $L = 1024$ and a threshold value $S = 22$. Table 6.4 gives specifications of the modified interleavers applied in simulations.

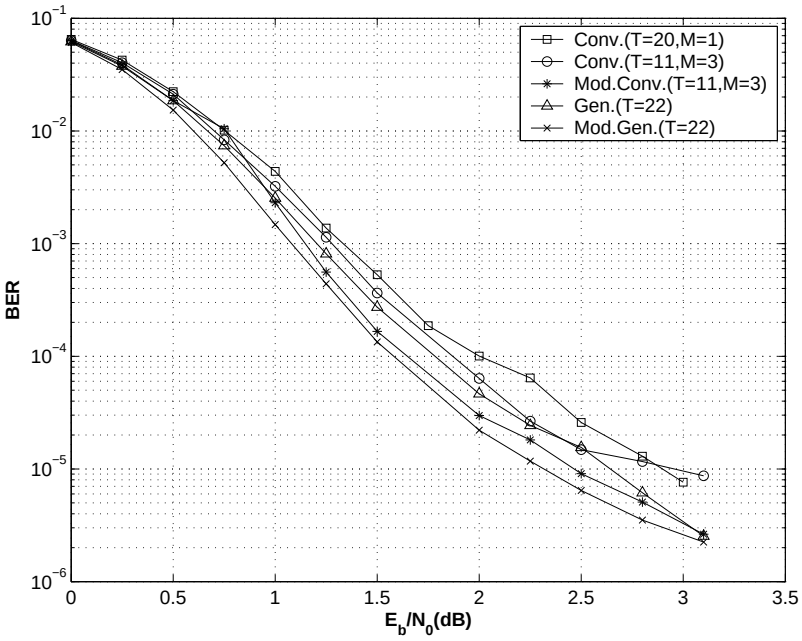

Figure 6.10 Performance of the 4- state full rate turbo code with interleavers length $L = 1024$.

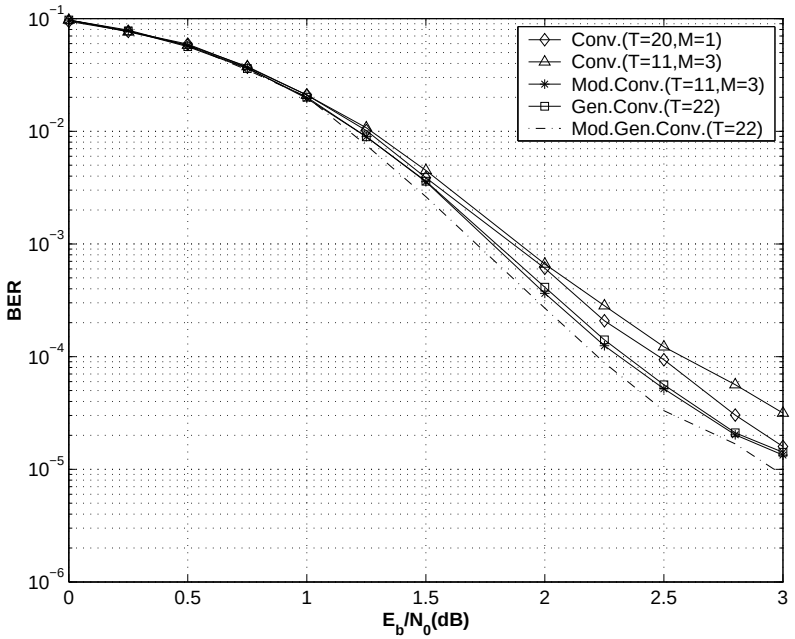

Figure 6.11 Performance of the 4- state half rate turbo code with interleavers length $L = 1024$.

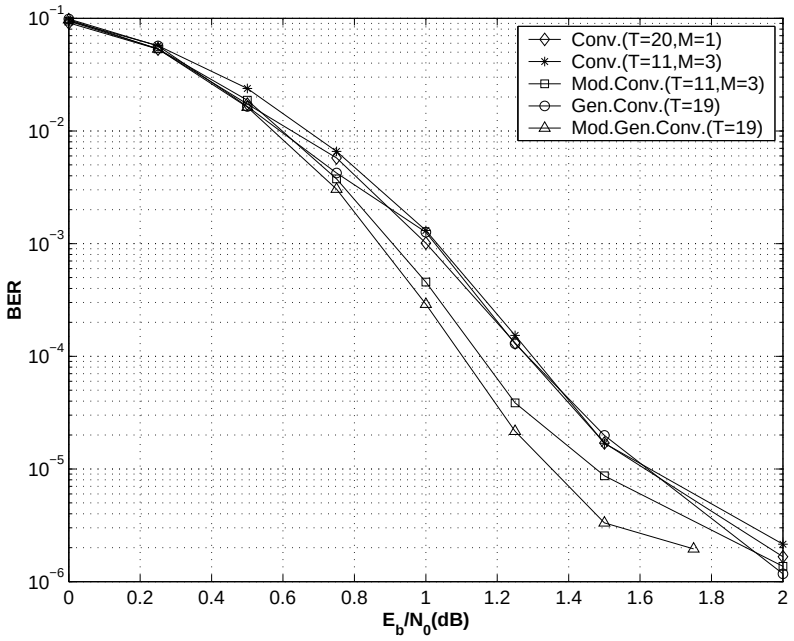

Figure 6.12 Performance of the 16- state full rate turbo code with interleavers length $L = 1024$.

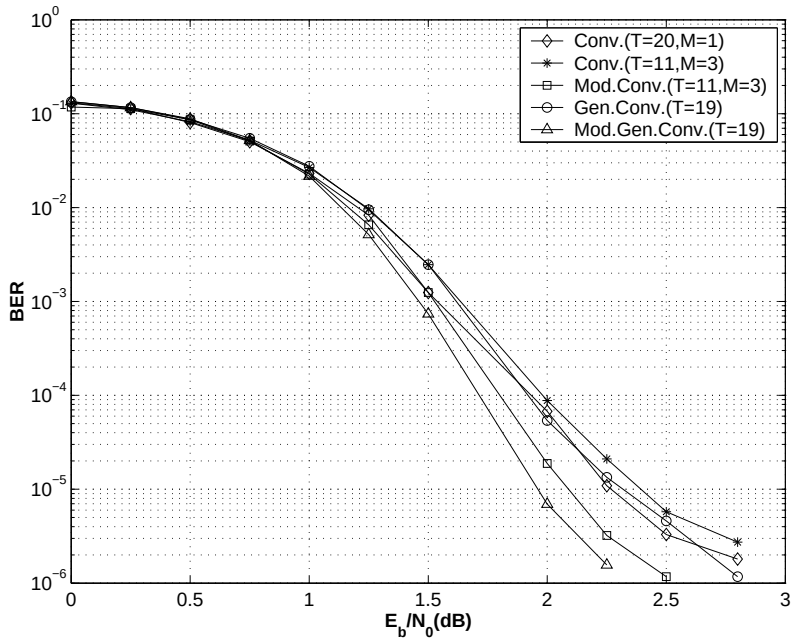

Figure 6.13 Performance of the 16- state half rate turbo code with interleavers length $L = 1024$.

Table 6.4
Specifications of Modified convolutional interleavers for 4- and 16- state turbo codes.

Code's State	Interleaver	Shifted interleaver lines	Shifted Unit value
4	Generalized $(T = 28)$	(2,5,8,20,24) $(11, 28)$ (10,12,14)	15*T 5*T 18*T
4	$(T = 24, M = 1)$	Even lines	16*T
4	$(T = 17, M = 2)$	Even lines	13*T
16	Generalized $(T = 23)$	Even lines	19*T
16	$(T = 24, M = 1)$	Even lines	18*T
16	$(T = 12, M = 4)$	Even lines	23*T

Figures 6.14 and 6.15 show the full and half rate $(1, \frac{5}{7})$ turbo code performance, respectively. It is concluded that the convolutional interleavers create better performance than the semi-random interleaver at the waterfall region. This is especially evident for the generalized convolutional interleaver $T = 28$, which improves the code performance by 0.25 dB for $\frac{E_b}{N_0} \leq 1.5dB$. In the error floor region, convolutional interleavers with similar behaviors have poor performance relative to the semi-random interleaver. This behavior can be resulted from generation of relative low free distance value of the convolutional interleavers. For the half rate 4-state code, this drawback has been slightly removed. The results give 0.2 dB poor performance of the generalized convolutional interleaver $(T = 28)$ for $E_b \geq 2.7dB$. In the Figure 6.15, all of convolutional interleavers have been outperformed the semi-random interleaver in the waterfall region.

Similar results are obtained from the full and half rate 16-state turbo code $(1, \frac{35}{23})$, which are illustrated in Figures 6.16 and 6.17. For the full rate code, the graphs show that the generalized interleaver $T = 23$ has better performance with interleavers $(T = 24, M = 1)$ and $(T = 12, M = 4)$, while the number of stuff bits has been been reduced by 10% and 7%, respectively . In comparison with the semi-random interleaver, the generalized interleaver has improved the code performance for $\frac{E_b}{N_0} \leq 1.5$ dB by 0.25 dB. The designed generalized convolutional interleaver has also very close performance to the semi-random interleaver in the error floor region. However, for the half rate 16-state turbo code, as shown in Figure 6.17, a relatively

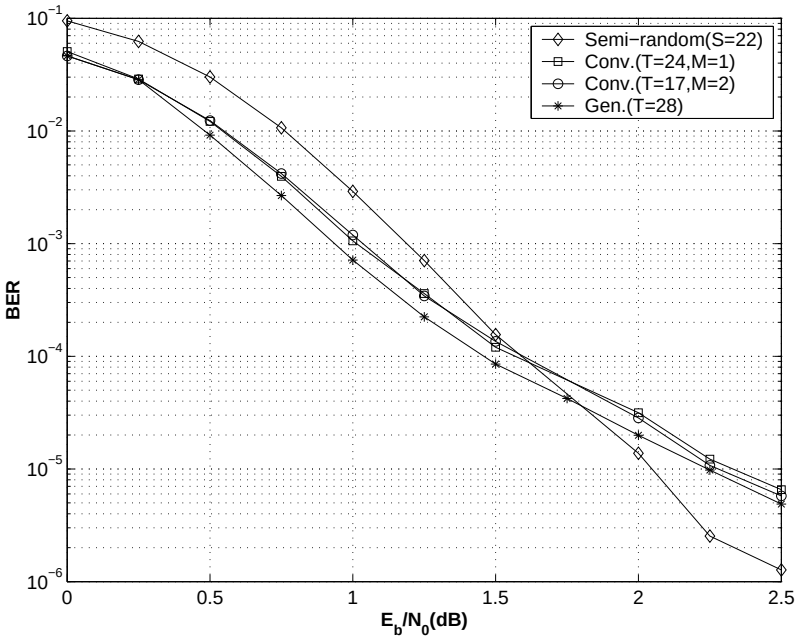

Figure 6.14 Performance of the 4- state full rate turbo code with interleavers length $L = 1024$.

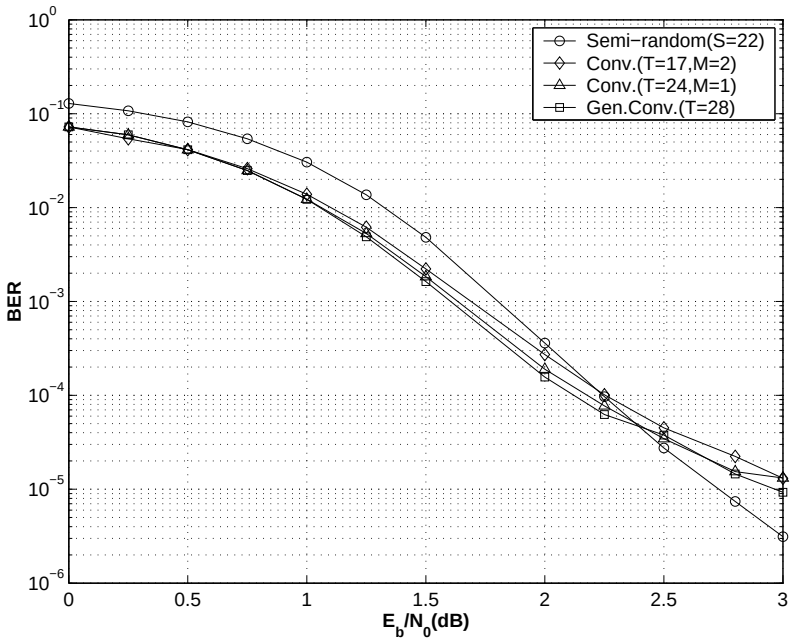

Figure 6.15 Performance of the 4- state half rate turbo code with interleavers length $L = 1024$.

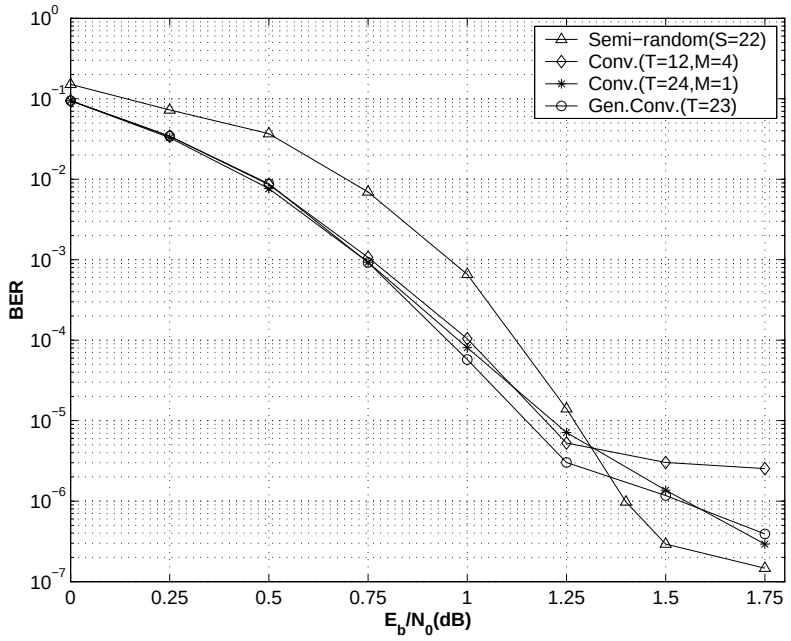

Figure 6.16 Performance of the 16- state full rate turbo code with interleavers length $L = 1024$.

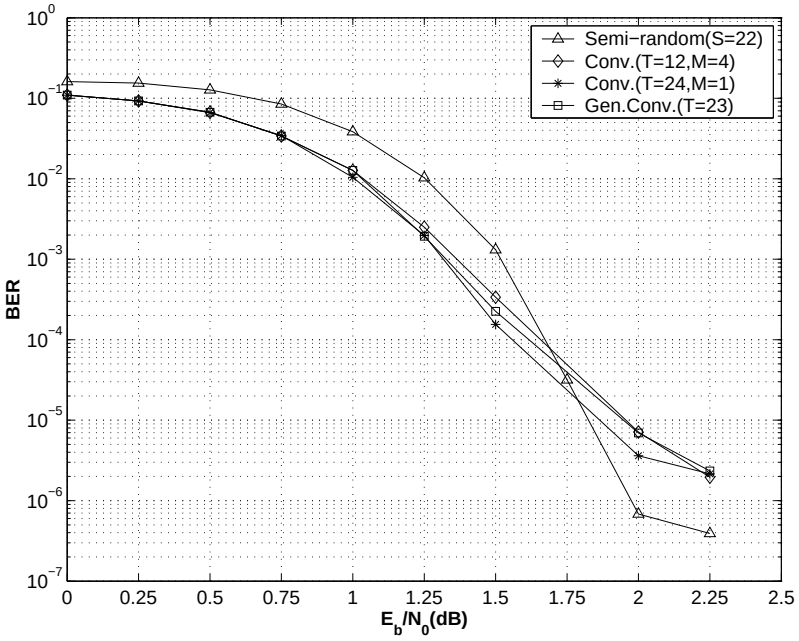

Figure 6.17 Performance of the 16- state half rate turbo code with interleavers length $L = 1024$.

poor performance of the convolutional interleavers compared to the semi-random interleaver is observed.

6.4 Chapter Summary and Conclusions

In this chapter, the structure of generalized convolutional interleavers with variable space values and their influence on the code performance was presented. These interleavers were designed with a higher period than previously proposed convolutional interleavers with the fixed space value, while the generated number of stuff bits is kept at similar level. In comparison with convolutional interleavers having the fixed space value, results confirmed that application of generalized convolutional interleavers improves performance of the turbo code. Finally, performance of the codes employing convolutional interleavers designed in this thesis have been compared with the codes using most conventional block interleavers.

The results for different codes with the short interleaver lengths indicate that convolutional interleavers, which utilize number of stuff bits equal to the 6% of the total number of encoded data, provide similar performance to block interleavers. For medium to high interleaver lengths, these interleavers with higher numbers of stuff bits should be designed in such a way that they have a similar performance to block interleavers. For a bitstream length $L = 1024$, the improvement is specifically obtained in the waterfall region of the turbo code performance curve, when convolutional interleavers contribute a number of stuff bits equal to 9% of the total number of encoded data. In the error floor region, these interleavers slightly degrade the code performance compared to semi-random interleavers, due to generation of a low free distance value for the applied turbo code.

Considering the overall number of applied memories in the proposed interleaver and deinterleaver, the generalized convolutional interleavers and deinterleavers utilize more memories in their structures than other interleavers and deinterleavers with a relatively high period. This increases synchronization complexity between the interleaver and deinterleaver. However, the results obtained from different convolutional interleavers reveal that without increasing the number of stuff bit, a good generalized

interleaver can be substituted for convolutional interleavers that are unable to break low weight self-terminating patterns. This issue is more applicable when they are compared with interleavers having a fixed space value grater than 2 $(M > 2)$.

Chapter 7

Convolutional Interleavers in Turbo Codes With Unequal Error Protection

7.1 Introduction

Unequal Error Propagation (UEP) was introduced as an efficient technique for Forward Error Correcting (FEC) codes to suitably protect encoded data against channel errors taking to account varying importance of different data fields. This is specifically utilized in the transmission of compressed information such as voice, video and multimedia data, were some parts of the data are much more sensitive to bit and burst errors than [120, 121]. UEP is generally accomplished in turbo codes by applying with different puncturing, which creates different code rates, for different protection levels. When the UEP property is implemented for a turbo code, a different interleaving appropriate for the data length determined for each protection level should be performed in addition to puncturing process.

This chapter deals with the application of convolutional interleavers to UEP turbo codes. Based on the properties of the convolutional interleaver, three different techniques for UEP turbo codes are presented. Simulation results confirm that utilizing this approach, the most important data parts can be better protected with lower number of stuff bits.

7.2 Interleavers for UEP Turbo Codes

To date, several methods have been suggested, mainly for conventional block interleavers, such as allocating an exclusive interleaver for each level or a single interleaver for all levels, where the interleaver length is adjusted for different levels. For a block interleaver with a fixed permutation, an interleaver for each level has been proposed in [57], while applications of one interleaver for all protection levels, which adjusts its length with the length of each level, are suggested in [122, 123]. In addition, a suitable interleaver for all protection levels has been designed, providing a UEP turbo code without the need for a puncturing process [124].

A structure of the semi-random interleavers usable for permutation of the data blocks with the variable length has been proposed in [125]. The obtained interleaver is named the prunable semi-random interleaver. In this interleaver, a semi-random interleaver is designed according to the shortest data length. Then for longer lengths, the new required position is randomly inserted. In this interleaver, if after several runs, the selected positions do not satisfy the appointed threshold value of the interleaver, the threshold value will be decreased and the above procedure followed based on the new threshold value. This reduction degrades the code performance and in order to overcome this problem, a new algorithm has been introduced to apply the semi-random interleaver for different data block lengths, without decreasing the threshold value [126]. Recently, Benedetto, $et.al$ presented a modification of the prunable interleaver, which improves the code performance with less complexity [127]. Similar to method suggested in [125], application of one semi-random interleaver for different protection levels has been presented in [128]. In this method, depending on the length of each level, the new threshold value is set.

On the other hand, Mohammadi $et.al$ have considered unequal error protection for the code based on contribution of systematic and parity data to the code performance, which is determined by distance spectrum of the code. It has been confirmed that for the short and medium to high bitstream lengths, the systematic and parity data have major contribution to the distance spectrum of the code, respectively. Therefore, it is expected that in correspondence with the interleaver length, suitable protections are

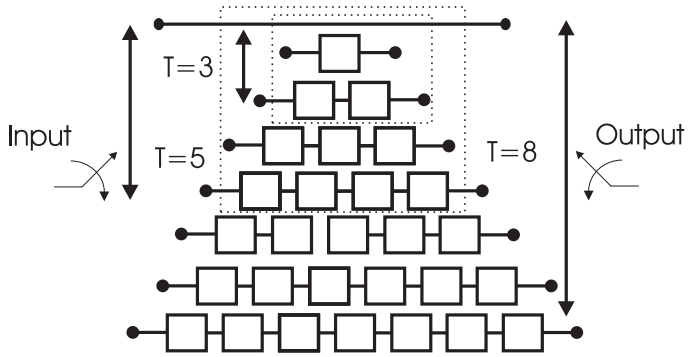

Figure 7.1 Consideration of convolutional interleaver $(T = 3, M = 1)$ and $(T = 5, M = 1)$ from the interleaver $(T = 8, M = 1)$.

conducted on different parts of the encoded data to improve the code performance [129, 130].

The main issue of the interleaver design for UEP turbo code applications is related to the flexibility of adjusting its specifications according to the varying length of data blocks. In contrast to block interleavers, convolutional interleavers are designed with less complexity to adjust their structures with the length variations of data blocks.

For the convolutional interleavers proposed in previous chapters that act as block interleavers by inserting stuff bits at the end of each data block, three different techniques are presented to design the UEP turbo codes [21]. These techniques are mainly implemented based on the interleaver period and code rate allocated to each level of protection. In the following sections, these techniques are explained.

7.2.1 Convolutional Interleavers with Different Periods and Code Rates

A convolutional interleaver with a specific period and space value has the flexibility to interleave data blocks with different lengths. In turbo code applications, when encoded data blocks with variable lengths obtained from an interleaver are punctured with different rates, UEP turbo codes can be achieved. Simulations of turbo codes with different interleaver lengths indicate that with an increment of the data block length, the period of the convolutional interleaver should be increased to provide

sufficient performance for the code with a reasonable number of stuff bits [15]. This is more sensitive for an interleaver with a short data block length and leads to a design of an interleaver compatible with the required performance of the code with the longest data block length for the given protection level.

However, since data with the highest protection level requires a lower code rate, the data block length is normally considered shorter at this level than at other levels. Hence, designing a convolutional interleaver based on the longest length for all protection levels, increases the number of stuff bits for levels with shorter lengths and can result in a greater number of stuff bits than valid data allocated to that level. This is observed when the length variations between different levels are relatively high. Therefore, the convolutional interleaver applied for this type of UEP turbo code is designed based on the shortest block length for all protection levels [21].

In order to apply an interleaver corresponding to the data specification of each level, it is necessary to employ an independently designed interleaver for each level. It is easily observed that by choosing some lines of an interleaver with the higher period another convolutional interleaver with a shorter period is obtained. For example, Figure 7.1 shows that convolutional interleavers $(T = 3, M = 1)$ and $(T = 5, M = 1)$ can be obtained from the convolutional interleaver $(T = 8, M = 1)$ when the relevant input bitstreams are distributed to the first three and five lines, respectively. These interleavers are created by controlling the distribution of input data blocks to some of the interleaver lines to generate different interleaved data. Interleaved data obtained from different periods are specifically punctured to provide UEP turbo codes. Based on the above observation, many interleavers with shorter periods can be constructed from an original interleaver with a longer period. For simplicity, interleavers with the space value 1 $(M = 1)$ are designed, where the distribution of data always starts from the line without the memory.

7.2.2 Convolutional Interleavers with Different Periods and Fixed Code Rates

Apart from applying different puncturing patterns, an interleaver for each level with different periods and a fixed code rate for all levels can be applied to provide different

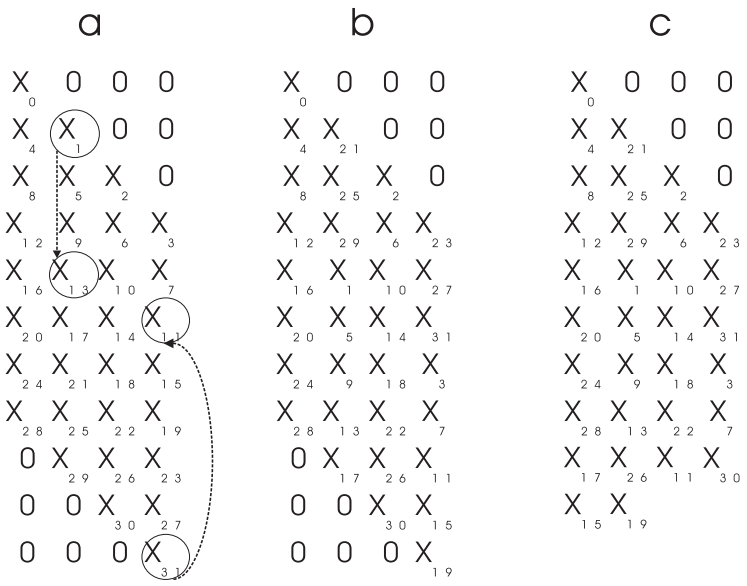

Figure 7.2 Modification procedure for the interleaver $(T = 4, M = 1)$. a) Interleaved data length $L = 32$, b) shifted even column bits equal to 3*T and c) deletion of zero bits at the end part of the interleaver.

protection levels. In this case, for the highest protection level, an interleaver with the longest period is designed such that it produces a reasonable number of stuff bits. Then, based on the order of other protection levels, interleavers with shorter periods are constructed by selecting some lines of the original interleaver [21]. For example, in Figure 7.1, contrary to the previous technique, the interleaver $(T = 8, M = 1)$ is applied for the highest protection level, while the interleavers $(T = 5, M = 1)$ and $(T = 3, M = 1)$ are used for to the second and third protection levels, respectively.

7.2.3 Convolutional Interleavers with Different Code Rates and Fixed Periods

In this technique, different puncturing is utilized for the different parts of an interleaved data obtained from one interleaver with the fixed period for all of protection levels [21].

For each technique, a modification can be performed to the interleavers, improving the code reliability with a lower number of stuff bits. This is generally accomplished by shifting the bits of the interleaved data located in even columns. Figure 7.2 shows

Table 7.1 Puncturing patterns for different protection levels.

Rate	l	**P**	**Q**	**O**
1/3	1	[1]	[1]	[1]
2/5	2	[1 1]	[1 0]	[1 1]
1/2	2	[1 0]	[0 1]	[1 1]
2/3	4	[1 0 0 0]	[0 0 1 0]	[1 1 1 1]
3/4	6	[1 0 0 0 0 0]	[0 0 0 1 0 0]	[1 1 1 1 1 1]

the modification procedure for the interleaver $(T = 4, M = 1)$. First, the input data blocks are regularly interleaved and then the bits located in the even columns are shifted by $3 * T$ units. Similarly to the modification proposed in [18], the number of shifted bits is considered even. In the case of an odd number of bits, the zero stuff bits located on the top of the first bit of even columns are involved in the modification process. Finally, zero stuff bits located at the end part of the interleaved data are deleted to optimize the number of stuff bits.

7.3 Simulation Results

In simulations, convolutional interleavers with short and long data block lengths have been applied for the three mentioned types of UEP with the 4-state tubo code $(1, \frac{5}{7})$. For the code, the trellis termination and truncation are utilized in the first and the second RSC encoders, respectively. To reduce the number of stuff bits to be equal to $\frac{T(T-1)M}{2}$, they will be removed from the end part of the systematic and the first parity data, since stuff bits are inserted after trellis termination and do not have any effect on the code performance. For simplicity, the effect of these stuff bits for the systematic and first parity data getting the exact code rate at each level are considered. At the decoder, iterative decoding is performed and the BER is only calculated based on the length of the original bitstream without stuff bits. Regarding this structure, the code rate of each level is calculated by:

$$R_i = \frac{l_i}{n_{\mathbf{P_i}} + n_{\mathbf{Q_i}} + n_{\mathbf{O_i}}} \tag{7.1}$$

Table 7.2
Specifications of protection levels with different interleaver periods and code rates.

Level	Length(L')	Int.Period(T)	Rate(R)	Number of simulated blocks
1	32	4	1/3	312500
2	48	5	2/5	208300
3	112	6	1/2	90000
overall	192	5	$\approx 1/2$	50000

Table 7.3
Specifications of protection levels with different interleaver periods and the fixed code rates.

Level	Length(L')	Int.Period(T)	Rate(R)	Number of simulated blocks
1	32	6	1/3	312500
2	48	5	1/3	208300
3	112	4	1/3	90000
overall	192	4	$\approx 1/3$	50000

Table 7.4
Specifications of protection levels with the fixed interleaver period and different code rates.

Level	Length(L')	Int.Period(T)	Rate(R)	Number of simulated blocks
1	32	4	1/3	312500
2	48	4	1/2	208300
3	112	4	2/3	90000
overall	192	4	$\approx 1/2$	50000

Table 7.5
Specifications of protection levels with different interleaver periods and code rates.

Level	Length(L')	Int.Period(T)	Rate(R)	Number of simulated blocks
1	128	7	1/3	75000
2	512	14	1/2	20000
3	1024	20	2/3	10000
4	2432	30	3/4	4000
overall	4096	25	$\approx 2/3$	2500

where l_i, $n_{\mathbf{O_i}}$, $n_{\mathbf{P_i}}$ and $n_{\mathbf{Q_i}}$ denote the length of the puncturing matrix, length of the matrix of 1s for the systematic data, and number of bit 1 in puncturing matrices of the ith level for the first and second RSC encoder with the length of l_i, respectively.

For the short and long data block lengths, three and four protection levels have been considered, respectively. Tables 7.1– 7.5 give specifications of puncturing patterns and protection levels of each UEP type.

In order to compare performance of the protection levels with the Equal Error Protection (EEP) codes, the overall specification of the code should be determined. With the employment of puncturing at each level, the average code rate with c protection levels is determined by: [131]

$$R_{av} = \frac{\sum_{i=1}^{c} L_i}{\sum_{i=1}^{c} \frac{L_i}{R_i}} \tag{7.2}$$

where $L_i = L'_i + N_i$ denotes the data block length of the i-th level after stuff bit insertion, obtained from summation of the original input data block length L'_i and the number of stuff bits N_i. The above protection parameters have been simulated by SOVA with 8 iterations in the presence of Additive White Gaussian Noise (AWGN). The number of simulated blocks considered for each $\frac{E_b}{N_0}$ has been presented in Tables 7.2– 7.5. The equivalent interleaver specifications can be determined based on the number of stuff bits or the interleaver periods and the data block lengths for each level. In this case, the equivalent interleaver period for the overall rate is given by:

$$T_{av} = \frac{\sum_{i=1}^{c} L_i T_i}{\sum_{i=1}^{c} L_i} \tag{7.3}$$

where T_i represents the interleaver period of the i-th level. In simulations, only the highest level of protection is modified. For interleavers $(T = 4, M = 1)$ and $(T = 7, M = 1)$, even column bits are cyclically shifted by $4 * T$ and $10 * T$ values, respectively. Figure 7.3 shows the code performance when different interleaver periods and code rates allocated for each protection level. In this figure levels 1 and level 2 are better than the overall performance of the code by 0.5 dB and 0.25 dB, respectively.

Figure 7.3 illustrates the code performance with different interleavers and code rates applied for protection levels based on the specifications in Table 7.3. In this figure, level 1 has 0.25 dB better performance than the overall level.

Figure 7.4 shows the code performance when different protection is achieved through different interleaver periods with the code rate fixed for all levels. Again, levels 1

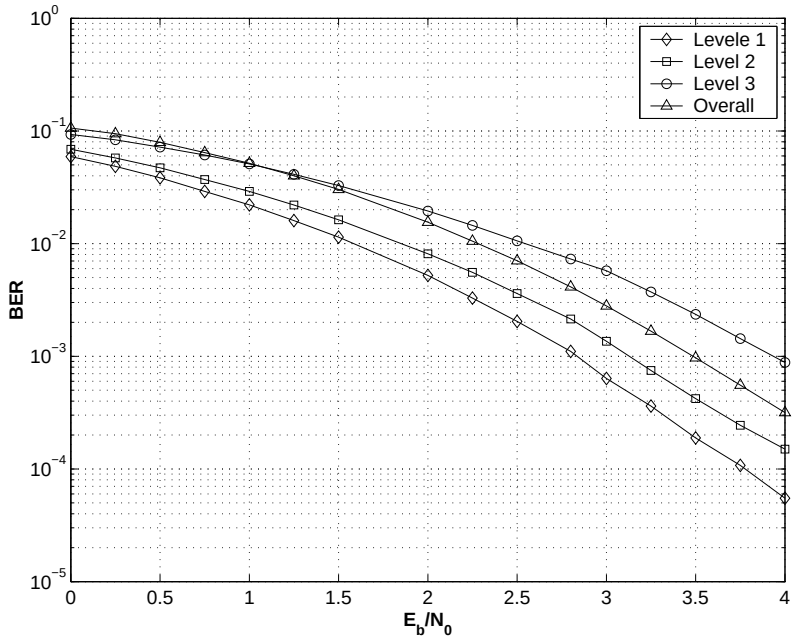

Figure 7.3 Unequal error protection for 4-state turbo codes with different interleaver periods and code rates.

and 2 in expense of higher periods than the overall level have improved the code performance by 1 and 0.5 dB, respectively. Since the level 3 and the overall level apply a similar code rate and an interleaver period value, their performance are very similar.

Figure 7.5 illustrates the code performance when an interleaver with the fixed period is applied. The graphs obviously represent more protections for the first and the second levels than the overall level. In this method, due to applying different rates, level 3 slightly shows the poor performance compared to the overall level.

Figure 7.6 shows the performance of the UEP turbo code with the four level protection and the interleaver length $L = 4096$. In this figure, levels 1 and 2 have 1 and 0.5 dB better performance than the average code performance, while number of stuff bits at these levels has been reduced by 93% and 69.6%, respectively. In addition, level 3 with a lower period and consequently less stuff bits has behavior close to the average for the code. However, due to application of the higher code rate and puncturing most of the encoded data, level 4 has the worst performance.

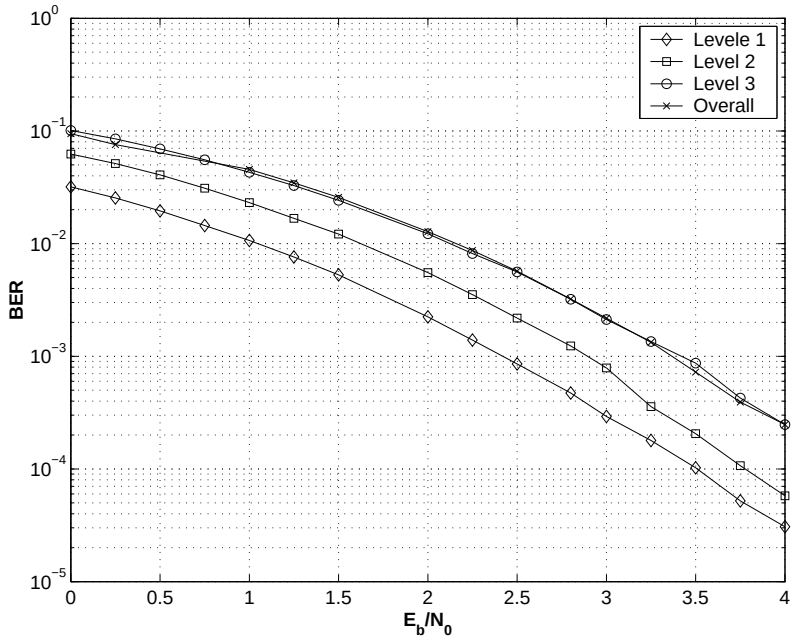

Figure 7.4 Unequal error protection for 4-state turbo codes with different interleaver periods and the fixed rate $R = \frac{1}{3}$.

7.4 Chapter Summary and Conclusions

In this chapter, three different techniques of UEP turbo code were suggested. These techniques are implemented based on applying different code rates or interleavers with different periods for each protection level. The results obtained from different types of the UEP turbo codes indicate that the convolutional interleaver has the flexibility to be utilized in UEP turbo code applications with short and long data block lengths. In particular, this is specifically observed for the second type of UEP, when the code is constructed with different periods and the fixed code rates for all levels. The improvement was specifically observed for the third protection level. However, type one and type three also improved the performance of levels 1 and 2 more than the overall level. Comparing the results obtained from the Figures 7.3 and 7.6 indicates that the first suggested technique is more applicable for the cases when the data lengths vary significantly for different protection levels. In such cases, the technique effectively protects the important parts of the data blocks with the shorter periods and lower numbers of stuff bits.

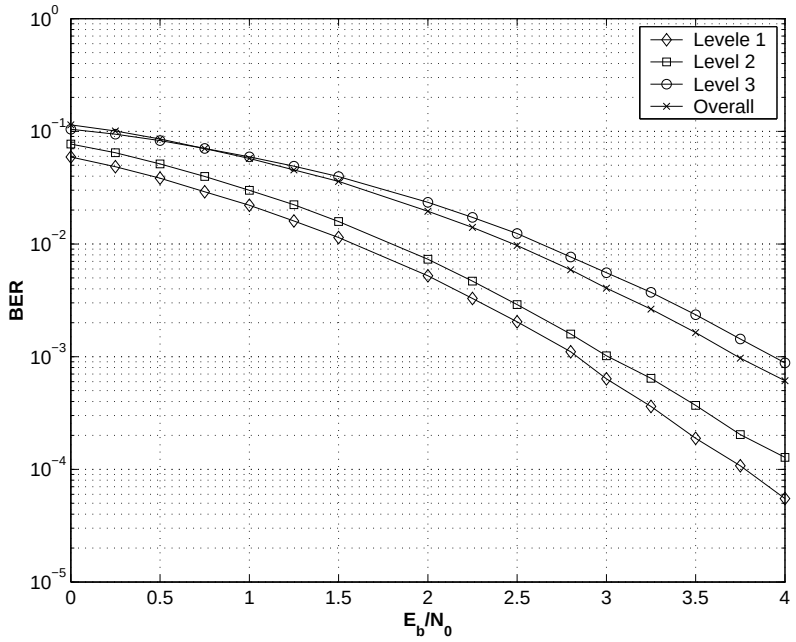

Figure 7.5 Unequal error protection for 4-state turbo codes with the fixed interleaver period $(T = 4)$ and different rates.

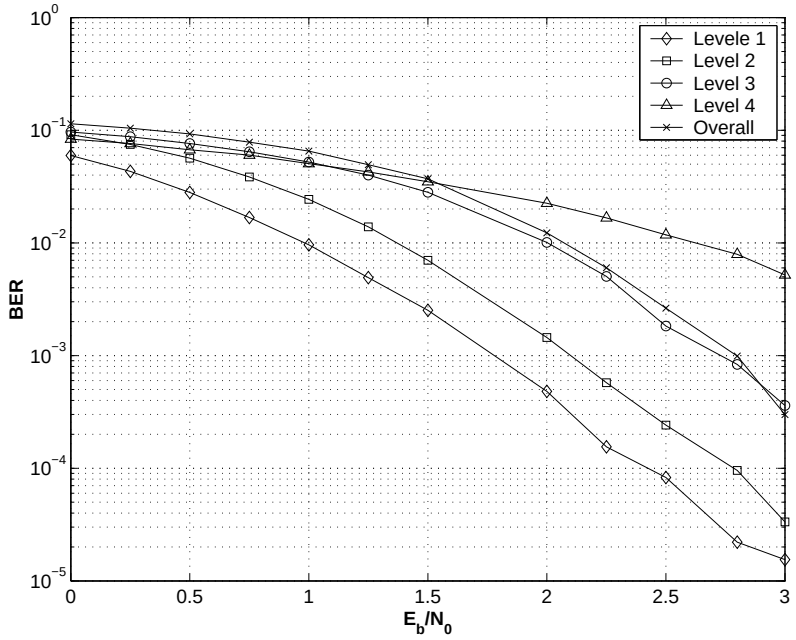

Figure 7.6 Unequal error protection for 4-state turbo codes with overall length $L = 4096$.

Chapter 8

Summary, Conclusions and Further Work

8.1 Introduction

The thesis was concerned with an application of block-wise convolutional interleavers in turbo codes. The block-wise operation of convolutional interleavers was obtained by insertion of enough number of stuff bits to the interleaver memories creating isolated data blocks. Different convolutional interleavers were constructed based on constituent parameters of the interleaver. To build a good convolutional interleaver with reasonable number of stuff bits, modification of the interleaver structure were proposed to achieve the interleaver compatible with the structure of RSC codes utilized in turbo codes. Performance of designed interleavers was analyzed based on calculation of weight distribution of turbo codes to demonstrate their performance in increasing the free distance value and breaking low weight self-terminating patterns. Simulation results confirmed the conducted analysis. The results indicated that the new interleavers can be utilized as good deterministic interleavers, which provide very close or even better performance than block interleavers with random permutations. Finally, application of these interleavers in Unequal Error Protection (UEP) was presented to protect the most important data parts with the lower overall number of stuff bits. The results obtained from previous chapters are gathered together and briefly explained here. Then, some suggestions for further research are presented.

8.2 Thesis Summary and Conclusions

Chapter 1 provided a brief introduction to the problem considered in the thesis and specified aims of the thesis. The major contributions of each chapter were explained as well as the publication resulting from the research listed.

Chapter 2 presented a literature survey on convolutional codes and different methods of its maximum likelihood decoding. Then, the structure of turbo codes constructed by parallel concatenation of codes and their analysis based on the performance of RSC constituent codes and the interleaver was explained. Some interleavers constructed as deterministic and random interleavers were introduced and their applications in turbo codes considered. It was concluded that interleaver with random permutation provides better randomization for the input bitstream than deterministic interleaver and improves the turbo code performance. Some issues of random interleavers were mentioned and deterministic interleavers producing close performance to the random interleavers were reviewed. Finally, a structure of iterative turbo decoding with SOVA was explained and some methods to improve its performance suggested.

Chapter 3 dealt with the structure of non-block interleavers. The structure of convolutional interleavers implemented to act as a block interleaver has been reviewed. This implementation was achieved by an insertion of some stuff bits at the end of each input data block returning the interleaver memories to the zero state. Based on this structure, an optimization was performed by deletion of the stuff bits at the end part of the interleaved data to reduce the number of stuff bits. For each convolutional interleaver, relevant iterative decoding performance was briefly explained. By considering the properties of the convolutional interleaver, an algorithm was implemented to calculate the free distance value of turbo codes, which effectively determines the code performance in the error floor region. The effect of other low weights in the code performance analysis was considered in another efficient and simple algorithm used to estimate weight distribution of the code. This algorithm considered low weight self-terminating patterns and computed the weight distribution of codes with short interleaver lengths. The calculated weights were extrapolated for weight

determination of a code with the required interleaver length. Conducted analysis and simulation results for different interleaver showed the reliability of the algorithm in determination of code performance in the error floor region. The results obtained from the code analysis confirmed that in the case of similar number of stuff bits, an optimized interleaver having a low free distance value with low multiplicity outperforms the non-optimized interleaver. This was more evident for interleavers with short block lengths. At the end of Chapter 3, the performance of optimized convolutional interleavers was compared with that of the most popular block interleavers, i.e. row-column and semi-random interleavers. For short interleaver lengths, the results indicated that with an acceptable number of stuff bits (approximately 5-8 % of the total number of encoded data) the designed convolutional interleavers have better performance than the block interleavers in the waterfall region, while due to their low free distance, the convolutional interleavers have a relatively weak performance in the error floor region. For the medium to high interleaver lengths, the poor performance of the convolutional interleaver compared to the semi-random interleaver was observed. Bad performance of convolutional interleavers was observed when its constituent parameters, i.e. period or space value, are set in such a way that self-terminating patterns are not broken to good patterns utilized for the second RSC encoder. The shortcoming of convolutional interleaver has been slightly reduced by increasing the period value of the interleaver.

Chapter 4 presented an algorithm to modify the optimized convolutional interleavers. The algorithm replaced bits positioned at the end part of the interleaved data with bits positioned in other interleaved data parts to increase the distance between interleaved bits adjacent int the original bitstream. This was accomplished by different cyclical shifts applied for the even and odd columns of the interleaver. In the development of this modification, an effect of bitstreams with weight-1, self-terminating patterns with weight-2 and weight-3 and their combinations making higher weight patterns was considered. The modification was performed in such a way that it breaks major low weight self-terminating patterns resulting in the code achieving a higher free distance with low multiplicity. The performance of modified convolutional interleavers was verified by determination of weight distribution of the code. In comparison with the unmodified interleaver, the calculated free distance values were significantly in-

creased. As a result, the code performance was improved in the error floor region without increasing the period, and consequently the number of stuff bits. Some examples demonstrated similar code performance between modified and unmodified interleavers, while number of stuff bits in the modified interleaver was reduced by 40%.

The interleavers constructed in previous chapters were designed based on the minimum space values. As one solution, to increase the distance between adjacent bits of the interleaved data, designing interleavers with higher space values was proposed in **Chapter 5**. In order to compare the performance of these interleavers with previously constructed interleavers, the number of stuff bits for both structures were kept in similar. This caused generation of high self-terminating patterns, which consequently degraded the code performance at the error floor region. It was particularly observed for interleavers with short periods and space values $M = 2$ and $M = 3$. Some modification techniques breaking low weight self-terminating patterns were proposed to improve the code performance. For the 4- and 16- state turbo code, the results showed that the modified versions of interleavers $(T = 6, M = 2)$ and $(T = 5, M = 3)$ create similar or even better performance than the convolutional interleaver $(T = 10, M = 1)$, while the number of stuff bits has been reduced by 33%. Similar results were achieved from interleavers with higher data lengths. For an interleaver length $L = 4096$, it was found that the interleaver $(T = 20, M = 3)$ provides a similar behavior to the interleaver $(T = 35, M = 1)$, while contributes a lower number of stuff bits.

In **Chapter 6**, a structure of generalized convolutional interleaver was presented. Unlike the interleavers proposed in previous chapters, these interleavers were designed with desired periods and variable space values. For the 4- and 16- state turbo code, different generalized convolutional interleavers have been proposed. They have been designed in a way that breaks weight-2 self-terminating patterns for the second RSC encoder. The interleavers were constructed based on the two-level joint interleavers. Some modifications on the two-level joint interleavers was conducted to provide generalized convolutional interleavers with a good period and a suitable distance between interleaver lines having the same number of memories. The simu-

lation results confirmed that these interleavers with a lower number of stuff bits provide similar or even better performance than the previously designed convolutional interleavers. This was specifically evident for the long generalized convolutional interleaver lengths, when they are compared with interleavers having the space parameter values greater than 2. Finally, the modified convolutional interleavers was carefully designed to provide close or even better performance than conventional block interleavers. This was assessed for different codes with different interleaver lengths.

In **Chapter 7**, an application of convolutional interleavers to UEP turbo codes was proposed. Three different techniques for the UEP turbo codes were designed based on the interleaver properties. The techniques were compared with each other in order to make it possible to select the technique most suitable to the given application.

8.3 Further Work

Considering the structure of convolutional interleaver applied in the thesis, some improvements and further studies are as follows:

- In modified interleavers, due to rearrangement of the interleaved data, the weight distribution algorithm proposed in Chapter 3 is not usable for determination of weight-distribution of turbo codes. Finding a simple and efficient algorithm based on the interleaver properties to compute the weight distribution of the code will give more accurate specifications of the code employing these interleavers. The results of such an analysis will be helpful to verify how a deterministic interleaver can be designed to provide similar performance to the semi-random interleaver.

- The thesis mainly analyzed turbo codes performance in the error floor region. However, the convolutional interleaver showed very good performance of the code in the waterfall region. The new analysis can be accomplished based on Extrinsic Information Transfer (EXIT) charts to verify the convergence behavior of the iterative turbo decoding algorithm. Iterative decoding was conducted

based on stuff bits value -1 for interleavers utilized at the decoder. In this case, EXIT charts analysis can determine an optimum value for stuff bits inserted to the interleaver memories to provide a suitable correlation between extrinsic and a-priori information in each decoding iteration.

- As another solution to reduce the number of stuff bits is replacement of bits located at the end part of interleaved data block with the stuff bits that appeared in the first part of the next interleaved data blocks. In this case, a deterministic interleaver without any stuff bits can be designed. Since distance between adjacent bits is not reduced, it makes possible to obtain a higher free distance with lower multiplicity than proposed block and non-optimized convolutional interleaver to improve the code performance in the error floor region.

- The generalized interleavers presented a very close performance to semi-random interleavers. These interleavers were simply designed based on weight-2 distribution of the code. Considering the effect of higher weights, new generalized interleavers with a suitable modification can be constructed improving performance of the turbo code in the error floor region, while lower number of columns is involved in the modification process.

8.4 Other Applications of This Work

Orthogonal Frequency-Division Multiplexing (OFDM) is introduced as an efficient technique for broadband wireless standards such as Digital Audio Broadcasting (DAB), Digital Video Broadcasting (DVB) and Wireless Local Area Network (LAN). For broadband wireless systems, which combine space-time coding with OFDM, interleaving also plays a major role. However, not much is done to assess the performance of the whole system depending on the characteristics of the applied interleavers. This thesis proves convolutional interleavers can be used as good deterministic interleavers. As it has been presented here, these interleavers can be well tailored to the specifications of the systems for which they are applied and improve the overall system' performance.

Bibliography

[1] S.Benedetto and E.Bieglieri, "Principles of digital transmission with wireless applications," *Kluwer academic/Plenum publishers*, 1999.

[2] C.E.Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, pp. 379–423,623–656, 1948.

[3] G.D.Forney, "Concatenated codes," *MIT Press*, 1966.

[4] C.Berrou, "The ten-year-old turbo codes are entering into service," *IEEE Communications Magazine*, pp. 110–116, August 2003.

[5] C.Berrou, P.Combelles, P.Penard, and B.Talibart, "An IC for turbo-codes encoding and decoding," *IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 90–91, February 1995.

[6] P.Coulton, B.Honary, M.Darnell, and S.B.Wicker, "Application of turbo codes to HF data transmission," *Seventh International Conference on HF Radio Systems and Techniques*, pp. 95–99, July 1997.

[7] Consultative Committee foe Space Data Systems, "Recommendations for space data systems, telemetry channel coding," *BLUE BOOK*, October 2002.

[8] 3rd Generation partnership project, "Multiplexing and channel coding (FDD)," *3G TS 25.212*, June 1999.

[9] ETSI, "Digital Video Broadcasting (DVB). interaction channel for Satellite Distribution Systems," *DVB-RCS001*, February 2000.

[10] J. Seghers, "On the free distance of turbo codes and related product codes," *Swiss Federal Institute of Technology, Zurich, Switzerland, Diploma project 6613*, vol. 42, August 1995.

[11] S.Dolinar and D.Divsalar, "Weight distributions for turbo codes using random and nonrandom permutations," *TDA Progress Report*, pp. 56–65, 15,August 1995.

[12] J.Sun and O.Y.Takeshita, "Interleavers for turbo codes using permutation polynomials over integer rings," *IEEE Transactions on Information Theory*, , no. 1, pp. 101 – 119, January 2005.

[13] S. Vafi, T. Wysocki, and I. Burnett, "Convolutional interleaver for unequal error protection of turbo codes," *Joint 7th International Symposium on DSP and Communication Systems (DSPCS) and 2nd Workshop on the Internet, Telecommunications and Signal Processing (WITSP)*, pp. 485–491, Dec. 2003.

[14] S. Vafi and T. Wysocki, "Iterative turbo decoder design with convolutional interleavers," *4th international symposium on Communication Systems, Networks and Digital Signal Processing, Newcastle (CSNDSP), UK.*, pp. 124–127, July 2004.

[15] S.Vafi and T.Wysocki, "Computation of the free distance and low weight distribution of turbo codes with convolutional interleavers," *15th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pp. 1356–1359, September 2004.

[16] S.Vafi and T.Wysocki, "Weight distribution of turbo codes with convolutional interleavers," *Submitted to IEE Proceedings Communications*, 2005.

[17] S.Vafi and T.Wysocki, "On the performance of turbo codes with convolutional interleavers," *Asia-Pasific Conference on Communications (APCC)*, pp. 222–226, October 2005.

[18] S.Vafi and T.Wysocki, "Modified convolutional interleavers and their performance in turbo codes," *IEEE Symposium on Trends In Communications and joint ISI workshop on Mobile Future (SympoTIC)*, pp. 54–57, October 2004.

[19] S.Vafi and T.Wysocki, "Performance of convolutional interleavers with different spacing parameters in turbo codes," *6th Australian communications theory workshop*, pp. 8–12, February 2005.

[20] S.Vafi and T.Wysocki, "Generalized convolutional interleaver and its performance in turbo codes," *Submitted to IEEE Communications Letters*, 2005.

[21] S.Vafi and T.Wysocki, "Application of convolutional interleavers in turbo codes with unequal error protection," *Accepted for Journal of Telecommunications and Information Technology (JTIT)*, 2005.

[22] Shu Lin and D.J.Costello, "Error control coding: Fundamentals and applications," *Prentice-Hall*, 1983.

[23] G.D.Forney, "Convolutional codes I: Algebraic structue," *IEEE Transactions on Information Theory*, vol. 16, no. 6, pp. 720–738, March 1972.

[24] A.J.Viterbi and J.K.Omura, "Principles of digital communication and coding," *McGraw-Hill Book Company, New York, NY*, April 1979.

[25] A.Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE. Transactions on Information Theory*, pp. 260–269, April 1969.

[26] V.S.Pless and W.C.Huffman, "Handbook of coding theory," *Elsevier Science*, vol. 2, 1998.

[27] R.McEliece, "The theory of information and coding," *Cambridge Unversity Press*, 2003.

[28] G.D.Forney, "Maximum-Likelihood sequence detection in the presence of intersymbol interference," *IEEE. Transactions on Information Theory*, pp. 363–378, May 1972.

[29] G.D.Forney, "The Viterbi Algorithm," *Proceedings of IEEE*, pp. 268–278, March 1973.

[30] L.H.Lou, "Implementing the Viterbi algorithm," *IEEE Signal Processing Magazine*, vol. 12, no. 5, pp. 42–52, September 1995.

[31] R.D.Wesel, "Convolutional codes appears in wiley encyclopedia of telecommunications(0-471-36972-1)," *John Wiley and Sons*, 2003.

[32] R.H.Morelos-Zaragoza, "The art of error correcting coding," *John Wiley and Sons*, 2002.

[33] C.Berrou, A.Glavieux, and P.Thitimajshima, "Near shannon Limit Error-Correcting coding and decoding: Turbo codes," *International Conference on Communications (ICC)*, pp. 1064–1070, May 1993.

[34] R.M.Pyndiah, "Near-optimum decoding of product codes: Block turbo codes," *IEEE Transactions on Communications Letters*, vol. 46, no. 8, pp. 1003–1010, August 1998.

[35] O.Takeshita, O.M.Collins, P.Massey, and D.J.Costello, "Asymmetric turbo codes," *IEEE International Symposium on Information Theory (ISIT)*, p. 179, August 1998.

[36] C.Berrou and A.Glavieux, "Near optimum error correcting coding and decoding: Turbo codes," *IEEE Transactions on Communications*, pp. 1261–1271, October 1996.

[37] M.A.Kousa and A.H.Mugaibel, "Puncturing effects on turbo codes," *IEE Proceedings on Communications*, vol. 149, no. 3, pp. 132–138, June 2002.

[38] D.Divsalar and F.Pollara, "Multiple turbo codes," *IEEE Military Communications Conference (MILCOM)*, pp. 279–285, November 1995.

[39] S.Huettinger and J.Huber, "Design of multiple-turbo-codes with transfer characteristics of component codes," *Conference on Information Sciences and Systems (CISS)*, March 2002.

[40] L.Hanzo, T.H.Liew, and B.L.Yeap, "Turbo coding, turbo equalisation and space-time coding for transmission over fading channels," *John Wiley and sons*, pp. 107–171, 2002.

[41] R.Garello, G.Montorsi, S.Benedetto, and G.Cancellieri, "Interleaver properties and their applications to the trellis complexity analysis of turbo codes," *IEEE Transactions on Communications*, vol. 1, pp. 793–807, May 2001.

[42] K.S.Andrews, "Turbo codes and interleaver design," *PhD Thesis of Cornell University*, August 1999.

[43] E.K.Hall and G.Wilson, "Stream-oriented turbo codes," *IEEE Transactions on Information Theory*, vol. 47, no. 5, pp. 1813–1831, July 2001.

[44] J.Hokfelt, O.Edfors, and T.Maseng, "On the theory and performance of trellis termination methods for turbo codes," *IEEE Journal on Selected Areas in Communications*, vol. 19, pp. 838–847, May 2001.

[45] M.Breilingrou and L.Hanzo, "The super-trellis structure of turbo codes," *IEEE Transactions on Information Theory*, vol. 46, pp. 2212–2228, September 2000.

[46] S.Benedetto and G.Montorsi, "Performance of continuous and blockwise decoded turbo codes," *IEEE Communications Letters*, vol. 1, pp. 77–79, May 1997.

[47] L.Perez, J.Seghers, and D.J.Costello, "A distance spectrum interpretation of turbo codes," *IEEE Transactions on Information Theory*, vol. 42, pp. 1698 – 1709, November 1996.

[48] S.Benedetto, G.Montorsi, and D.Divsalar, "Concatenated convolutional codes with interleavers," *IEEE Communications Magazine*, pp. 102–109, August 2003.

[49] S.Benedetto and G.Montorsi, "Unveiling turbo codes: Some results on parallel concatenated coding schemes," *IEEE Transactions on Information Theory*, vol. 42, pp. 409–428, March 1996.

[50] S.Benedetto and G.Montorsi, "Design of parallel concatenated convolutional codes," *IEEE Transactions on Communications*, vol. 44, no. 5, pp. 591–600, May 1996.

[51] J.Yuan, B.Vucetic, and W.Feng, "Combined turbo codes and interleaver design," *IEEE Transactions on Communications*, pp. 484–487, April 1999.

[52] C.Berrou and A.Glavieux, "Turbo codes:general principles and applications," *Proceedings of the 6th Trirrenia International Workshop on Digital Communications*, pp. 215–226, September 1993.

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

[54] J.Hokfelt, O.Edfors, and T.Maseng, "Turbo codes: Correlated extrinsic information and its impact on iterative decoding performance," *IEEE Vehicular Technology Conference*, vol. 3, pp. 1871–1875, May 1999.

[55] J.G.Proakis, "Digital communications," *McGraw-Hill*, 2001.

[56] T.M.Duman, "Interleavers for serial and parallel concatenated (Turbo) codes," *Wiley Encyclopedia of Telecommunications*, pp. 1141–1151, December 2002.

[57] A.S.Barbulescu and S.S.Pietrobon, "Terminating the trellis of turbo-codes in the same state," *IEEE Communications Letters*, vol. 31, pp. 22–23, January 1995.

[58] W.J.Blackert, E.K.Hall, and G.Wilson, "Turbo code termination and interleaver conditions," *IEE Electronics Letters*, vol. 31, no. 24, pp. 2082–2084, November 1995.

[59] H.Herzberg, "Multilevel turbo coding with short interleavers," *IEEE Journal Selected Areas in Communications*, vol. 16, pp. 303–309, February 1998.

[60] L.Lin and R.S.Cheng, "Improvements in SOVA-based decoding for turbo codes," *IEEE Global Telecommunications Conference (Globecom)*, vol. 2, pp. 644–648, November 1997.

[61] G.C.Clark and J.B.Cain, "Error-Correction Coding for Digital Communications," *Plunum Press*, 1981.

[62] B.Vucetic and J.Yuan, "Turbo codes: Principles and applications," *Kluwer Academic*, 2000.

[63] S.Benedetto and G.Montorsi, "Average performance of parallel concatenated block codes," *Electronics Letters*, vol. 31, no. 3, pp. 156–158, February 1995.

[64] A.H.Aghvami F.Said and W.G.Chambers, "Improving random interleaver for turbo codes," *Electronics Letters*, vol. 35, pp. 2194–2195, December 1999.

[65] B.G.Lee, S.J.Bae, S.G.Kang, and E.K.Joo, "Design of swap interleaver for turbo codes," *IEE Electronics Letters*, vol. 35, no. 22, pp. 1939–1940, October 1999.

[66] W.Feng, J.Yuan, and B.Vucetic, "A code-matched interleaver design for turbo codes," *IEEE Transactions on Communications*, pp. 926–937, June 2002.

[67] A.Abbasfar and K.Yao, "Interleaver design for turbo codes by distance spectrum shaping," *IEEE Wireless Communications and Networking Conference (WCNC)*, vol. 3, pp. 1616–1619, March 2004.

[68] X.Zhang, D.Yuan, and Ji.Luo, "Code-matched interleaver for turbo codes," *IEEE Wireless Communications and Networking Conference (WCNC)*, vol. 3, pp. 1607–1610, March 2004.

[69] H.Zhang, L.Wang, Q.Yuan, H.Wang, and J.Yu, "A Chaotic interleaver used in turbo codes," *International Conference on Communications, Circuits and Systems (ICCCAS)*, vol. 1, pp. 27–29, June 2004.

[70] J.Hokfelt and T.Maseng, "Methodical interleaver design for turbo codes," *International Symposium on Turbo Codes and Related Topics, Brest, France*, pp. 212–215, September 1997.

[71] O.Y.Takeshita and D.J.Costello, "New deterministic interleaver designs for turbo codes," *IEEE Transactions on Information Theory*, vol. 46, no. 6, pp. 1988 – 2006, September 2000.

[72] S.Crozier and P.Guinand, "High-performance low memory interleaver banks for turbo codes," *Vehicular Technology Conference, VTC 2001 Fall*, vol. 4, pp. 2394–2398, October 2001.

[73] S.Crozier and P.Guinand, "Distance upper bounds and true minimum distance results for turbo-codes designed with DRP interleavers," *3rd International Symposium on Turbo codes and Related Topics, Brest, France*, pp. 169–172, September 2003.

[74] S.Crozier, "Interleaving with golden increments," *European Patent Application, EP 0963049A2*, December 1998.

[75] S.Crozier, J.Jodge, P.Guinand, and A.Hunt, "Performance of Turbo-codes with relative prime and golden interleaving strategies," *Proceeding of Sixth International Mobile Satellite Conference (IMSC), Ottawa, Canada*, pp. 268–275, June 1999.

[76] F.Chan, "Matched interleavers for turbo codes with short frames," *Seventh Candian workshop on Information Theory*, June 2001.

[77] F.Chan, "Block interleavers for turbo codes with short frames," 3^{rd} *International Symposium on Turbo Codes and Related Topics, Brest, France*, pp. 563–566, September 2003.

[78] J.A.Briffa and V.Buttigieg, "Interleaving and termination in unpunctured symmetric turbo codes," *IEE Proceedings on Communications*, vol. 149, no. 1, pp. 6–12, February 2002.

[79] J.Boutros and G.Zemor, "Interleavers for turbo codes that yield a minimum distance growing with blocklength," *IEEE International Symposium on Information Theory (ISIT)*, p. 55, July 2004.

[80] D.Truhachev, M.Lentmaier, O.Wintzell, and K.SH.Zigangirov, "On the minimum distance of turbo codes," *IEEE International Symposium on Information Theory (ISIT)*, p. 84, 2002.

[81] C.J.C.Bravo and I.Rubio, "Algebric construction of interleavers using permutation monomials," *IEEE International Conference on Communications (ICC)*, pp. 911–915, June 2004.

[82] C.J.Corrada-Bravo and I.Rubio, "Deterministic interleavers for turbo codes with random-like performance and simple implementation," *3rd International Sympsium on Turbo codes and Related Topics,Brest,France*, pp. 555–558, September 2003.

[83] H.R.Sadjadpour, M.Salehi, N.J.A.Sloane, and G.Nebe, "Interleaver design for short block length turbo codes," *IEEE International Conference on Communications*, vol. 2, pp. 628–632, June 2000.

[84] H.R.Sadjadpour, M.Salehi, N.J.A.Sloane, and G.Nebe, "Interleaver design for turbo codes," *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 5, pp. 831–837, May 2001.

[85] A.Tarable and S.Benedetto, "Mapping interleaving laws to parallel turbo decoder architectures," *IEEE Communications Letters*, vol. 3, no. 8, pp. 162–164, March 2004.

[86] J.Hagenauer, E.Offer, and L.Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Transactions on Information Theory*, vol. 42, no. 2, pp. 429–445, March 1996.

[87] L.Bahl, J.Cocke, F.Jelinek, and J.Raviv, "Optimal decoding of linear codes for minimizing symbol error rate (Corresp.)," *IEEE Transactions on Information Theory*, vol. 20, no. 2, pp. 284–287, March 1974.

[88] P.Robertson, "Illuminating the structure of code and decoder of parallel concatenated recursive systematic(turbo) codes," *IEEE Global Telecommunications Conference (Globecom)*, vol. 3, pp. 1298–1303, December 1994.

[89] J.Hagenauer, P.Robertson, and L.Papke, "Iterative (turbo) decoding of systematic convolutional codes with MAP and SOVA algorithm," *ITG Conference on Source and Channel Coding*, pp. 429–445, October 1994.

[90] J. Hagenauer and P.Hoher, "A Viterbi algorithm with soft-decision outputs and its applications," *IEEE Global Telecommunications Conference (Globecom)*, pp. 47.1.1–47.1.7, Nov. 1989.

[91] J.Hagenauer, "Source-controlled channel decoding," *IEEE Transactions on Communications*, vol. 43, no. 9, pp. 2449–2457, September 1995.

[92] L.Papke, P.Robertson, and E.Villebrun, "Improved decoding with the SOVA in a parallel concatenated (Turbo-code) scheme," *IEEE International Conference on Communications (ICC)*, vol. 1, pp. 102–106, June 1996.

[93] Z.Blazek and V.K.Bhargava, "A DSP-based implementation of a turbo-decoder," *IEEE Global Telecommunications Conference (Globecom)*, pp. 2751–2755, November 1998.

[94] R.A.Stirling-Gallacher, "Performance of sub-optimal normalisation schemes for a turbo decoder using the Soft Output Viterbi algorithm," *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, , no. 2, pp. 888–892, September 2000.

[95] D.W.Kim, T.W.Kwon, J.R.Choi, and J.J.Kong, "A modified two-step SOVA-based turbo decoder with a fixed scaling factor," *IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 4, pp. 37–40, May 2000.

[96] J.Vogt and A.Finger, "Improving the Max-Log-MAP turbo decoder," *IEE Electronics Letters*, vol. 36, no. 23, pp. 1937–1939, November 2000.

[97] S.Papaharalabos, P.Sweeney, and B.G.Evans, "Modification of branch metric calculation to improve iterative SOVA decoding of decoding turbo codes," *Electronics Letters*, vol. 39, no. 19, pp. 1391–1392, September 2003.

[98] S.Papaharalabos, P.Sweeney, and B.G.Evans, "A new method of improving SOVA turbo decoding for AWGN, Rayleigh and Raician fading channels," *IEEE Spring Vehicular Technology Conference(VTC)*, vol. 5, pp. 2862–2866, May 2004.

[99] W.Zhongfeng and K.K.Parhi, "High performance, high throughput turbo SOVA decoder design," *IEEE Transactions on Communications*, vol. 51, no. 4, pp. 570–579, April 2003.

[100] C.X.Huang and A.Ghrayeb, "Improved SOVA and APP decoding algorithms for serial concatenated codes," *15th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, vol. 2, pp. 1121–1125, September 2004.

[101] L.Lin and R.S.Cheng, "Improvements in SOVA-based decoding for turbo codes," *IEEE International Conference on Communications (ICC)*, vol. 3, pp. 1473–1478, June 1997.

[102] W.Zhongfeng, H.Suzuki, and K.K.Parhi, "Efficient approaches to improving performance of VLSI SOVA-based turbo decoders," *IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 1, pp. 287–290, May 2000.

[103] M.Fossorier, F.Burkert, S.Lin, and J. Hagenauer, "On the equivalence between SOVA and Max-Log-MAP decodings," *IEEE Communications Letters*, vol. 2, no. 5, pp. 137–139, May 1998.

[104] J.Chen, M.Fossorier, S.Lin, and C.Xu, "Bi-directional SOVA decoding for turbo-codes," *IEEE Communications Letters*, vol. 4, no. 12, pp. 405–407, December 2000.

[105] J.Ramsey, "Realization of optimum interleavers," *IEEE Transactions on Information Theory*, vol. 16, no. 3, pp. 338–345, May 1970.

[106] G.D.Forney, "Burst-correcting codes for the classic bursty channel," *IEEE Transaction on Communications.*, vol. COM-19, pp. 772–781, October 1971.

[107] G.D.Forney, "Interleavers," *Us Patent 3652998*, March 1970.

[108] D.Divsalar and R.J.McEliece, "Effective free distance of turbo codes," *IEE Electronics Letters*, vol. 32, no. 5, pp. 445–446, February 1996.

[109] N.Kahale and R.Urbanke, "On the minimum distance of parallel and serially concatenated codes," *International Symposium on Information Theory (ISIT)*, p. 21, August 1998.

[110] I.Yoscovich and J.Snyders, "On the effective free distance of turbo codes," *IEEE Information Theory Workshop (ITW)*, pp. 120–121, June 1998.

[111] M.Ambroze, G.Wade, and Martin Tomlinson, "Dependence of d_{free} in MPCCC systems," *IEEE Transactions on Communications*, vol. 51, pp. 318–325, March 2003.

[112] W.J.Blackert, E.K.Hall, and G.Wilson, "An upper bound on turbo code free distance," *IEEE International Conference on Communications (ICC)*, vol. 2, pp. 957–961, June 1996.

[113] R.Garello, F.Chiaraluce, P.Pierleoni, M.Scaloni, and S.Benedetto, "On error floor and free distance of turbo codes," *IEEE International Conference on Communications (ICC)*, vol. 1, pp. 45–49, June 2001.

[114] R.Garello, P.Pierleoni, and S.Benedetto, "Computing the free distance of turbo codes and serially concatenated codes with interleavers: algorithms and applications," *IEEE Journal on Selected Areas in Communications*, vol. 19, pp. 800–812, May 2001.

[115] E.Rosnes and Ø.Ytrehus, "Improved algorithms for high rate turbo code weight distribution calculation," *10th International Conference on Telecommunications (ICT)*, vol. 1, pp. 104–110, March 2003.

[116] E.Rosnes and Ø.Ytrehus, "Improved algorithms for determination of turbo-code weight distributions," *IEEE Transactions on communications*, vol. 53, no. 1, pp. 120–121, January 2005.

[117] P.Yeh, A.Yilmaz, and W.Stark, "On the error analysis of turbo codes: Weight Spectrum Estimation (WSE) scheme," *IEEE Internation Symposium on Information Theory (ISIT)*, p. 439, June 2003.

[118] A. Huebner and D.J. Costello, "A simple method of approximating the error floor of turbo codes with S-type permutors," *International Symposium on Information Theory (ISIT)*, p. 473, 27 June-2 July 2004 2004.

[119] Cemaron, "Generalized convolutional interleaver/deinterleaver," *US patent,No.US 6,697,975 B2*, 24,Feb.2004.

[120] F.Marx and J.Farah, "Improved turbo-coded UMTS systems with unequal error protection of compressed video sequences transmitted over frequency-selective channels," *IEEE International Conference on Communications (ICC)*, pp. 3091–3095, June 2004.

[121] N.Thomos, N.V.Boulgouris, and M.G.Strintzis, "Wireless image transmission using turbo codes and optimal unequal error protection," *IEEE International Conference on Image Processing*, pp. 73–76, September 2003.

[122] M.Salah, R.A.Rains, and A.Temple, "A general interleaver for equal and unequal error protections of turbo codes with short frames," *International Conference on Information Technology: Coding and Computing*, pp. 412–415, 2000.

[123] G.Caire and G.Lechner, "Turbo codes with unequal error protection," *Electronics Letters*, pp. 629–631, March 1996.

[124] M.Grangetto, E.Magli, and G.Olmo, "Embedding Unequal Error Protection into turbo codes," *35th Asilomar Conference on Signals, Systems and Computers*, vol. 1, pp. 300–304, 2001.

[125] M.Ferrari, F.Scalise, and S.Bellini, "Prunable S-random interleavers," *IEEE International Conference on Communications (ICC)*, vol. 3, pp. 1711–1715, May 2002.

[126] P.Popovski, L.Kocarev, and A.Risteski, "Design of flexible-length S-random interleaver for turbo codes," *IEEE Communications Letters*, vol. 8, no. 7, pp. 461–463, July 2004.

[127] L.Dioni and S.Benedetto, "Design of prunable S-random interleavers," *International Symposium on Turbo Codes and Related Topics*, pp. 279–289, September 2003.

[128] M.Barazande-Pour, J.W.Mark, and A.K.Khandani, "Multi-level priority transmission of images over a turbo-coded channel," *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing(PACRIM)*, pp. 904–907, August 1997.

[129] A.Mohammadi and A.K.Khandani, "Unequal error protection on the turbo-encoder output bits," *IEEE International Conference on Communications (ICC)*, pp. 730–734, 1997.

[130] A.Mohammadi and A.K.Khandani, "Unequal error protection on turbo-encoder output bits," *Electronics Letters*, pp. 273–274, February 1997.

[131] G.Caire and E.Biglieri, "Parallel concatenated codes with unequal error protection," *IEEE Trans.on Communications*, vol. 46, no. 5, pp. 565–567, May. 1998.