University of Cambridge

Department of Engineering

# Performance Analysis and Design of Punctured Turbo Codes

Ioannis Ap. Chatzigeorgiou

Clare Hall

A dissertation submitted for the degree of

*Doctor of Philosophy*

Michaelmas 2006

To my father, *Apostolos*, who taught me that reinventing the wheel is sometimes necessary.

To my mother, *Thalia*, who taught me never to give up faith and hope.

To my partner, *Eirini*, who taught me to enjoy life as it comes, with its good and bad moments.

# Abstract

Turbo codes, formed by the parallel concatenation of two recursive systematic convolutional component codes separated by an interleaver, have been proposed for use in a variety of wireless applications, including mobile, satellite and fixed wireless systems. Analysis of their performance requires knowledge of their transfer function, which conveys their distance properties. Using the transfer function, a tight upper bound on their bit error probability, averaged over all interleavers of a given size, can be evaluated. The upper bound closely reflects the actual bit error rate performance of a turbo code on additive white Gaussian noise (AWGN) channels, if maximum-likelihood decoding is used. In practice, however, suboptimal iterative decoding is employed, nevertheless the bit error rate performance of a turbo code converges towards low bit error probabilities for an increasing number of iterations and coincides with the upper bound, when the turbo code operates in the error floor region.

This dissertation is concerned initially with the determination of the transfer function of turbo codes. We first introduce the augmented state diagram, a novel approach for the evaluation of the transfer function of rate-1/3 turbo codes. We then extend the concept of the augmented state diagram to account for turbo codes, whose output is periodically punctured according to a selected pattern, thus achieving rates higher than 1/3. Furthermore, we demonstrate that our technique can be used not only to determine the transfer function of turbo codes and, subsequently, compute their performance upper bound, but also to identify good puncturing patterns that lead to punctured turbo codes yielding a low error floor.

As the input block length, or equivalently, the size of the constituent interleaver increases, computation of the full transfer function of a turbo code becomes intensive. However, turbo codeword sequences generated by input information sequences having small Hamming weight, start playing a significant role in the error rate performance of the turbo code. Motivated by this observation, we present a simple method to quickly enumerate only the codeword sequences having low information weight, instead of all codeword sequences composing the full transfer function. Consequently, we obtain an accurate approximation of the performance upper bound of punctured and non-punctured turbo codes, when large interleavers are used.

We use our proposed approximation technique to demonstrate that there exist rate-1/2 punctured turbo codes that can achieve lower error floors than that of their rate-1/3 parent codes. In particular, we show that a particular puncturing scheme can be used to reduce the rate of a turbo code from 1/3 to 1/2 and at the same time achieve a coding gain at low bit error probabilities, when suboptimal iterative decoding is used.

# Acknowledgements

First and foremost I would like to thank my supervisor *Dr. Ian J. Wassell* for his support throughout this project and for allowing me the freedom to carry out a major part of my thesis in the way I thought most suitable. His door was always open and he was always keen to proof-read my work and discuss my findings. My thanks are also extended to *Professor Rolando Carrasco* at the University of Newcastle for the insightful discussions that have extended my knowledge in the field of communication theory.

I am especially grateful to *Dr. Miguel R. D. Rodrigues*, Senior Research Associate at Cambridge University, for helping me along the path to the completion of this thesis. His guidance, his encouragement and his belief in me were invaluable. At times when I felt like loosing sight of my way, he was always there to motivate me and point me in the right direction. But above all, I thank him for being a friend! All the best, Miguel, and *boa sorte*!

My most sincere appreciation is extended to the *Engineering and Physical Sciences Research Council* for sponsoring my work through a grant awarded to my supervisor, Dr. Wassell, for a joint research project between the universities of Cambridge and Newcastle. It is because of their support that I managed to pursue research at one of the finest academic institutions. I am also grateful to the *Engineering Department* and my college, *Clare Hall*, for the additional financial support.

I could never forget to mention my colleagues and friends in rooms SN21 and SN27 for making my time in the Digital Technology Group so enjoyable: *Ali*, *Elgan*, *Evan*, *Francisco*, *Jaime*, *Kam*, *Min* and *Stavros*, thank you all! I will always remember our long discussions, ranging from scientific and high-tech issues to Indian history and Tanzanian customs!

As always, the last paragraph is reserved for the loved ones, my parents, *Apostolos* and *Thalia*, and my partner, *Eirini*. I will always be obliged to my parents for their trust and confidence on my decisions and their continuous emotional support and encouragement during the past years. I am indebted to Eirini, who by now knows more about communication theory than I do about architecture and the arts. I thank her for her patience and support and for sharing with me moments of disappointment, inspiration and achievement.

# Declaration

As required by the University Statutes, I hereby declare that this dissertation is not substantially the same as any that I have submitted for a degree or diploma or other qualification at any other university. This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text.

I also declare that the length of this dissertation does not exceed 65,000 words, including footnotes and bibliography, and that the number of figures is less than 150.

Ioannis Chatzigeorgiou
Cambridge, 2006.

# Publications

The work discussed in this dissertation inspired the following publications:

1. Ioannis A. Chatzigeorgiou, Miguel R.D. Rodrigues, Ian J. Wassell and Rolando Carrasco, "A Union Bound Approximation for Rapid Performance Evaluation of Punctured Turbo Codes", submitted to the *2007 IEEE International Conference on Communications* (ICC'07), Glasgow, United Kingdom, June 2007.

2. Miguel R.D. Rodrigues, Ioannis A. Chatzigeorgiou, Ian J. Wassell and Rolando Carrasco, "Performance Analysis of Turbo Codes in Quasi-static Fading Channels", submitted to *IEEE Transactions on Wireless Communications*.

3. Ioannis A. Chatzigeorgiou, Miguel R.D. Rodrigues, Ian J. Wassell and Rolando Carrasco, "Comparison of Convolutional and Turbo Coding for Broadband FWA Systems", submitted to *IEEE Transactions on Broadcasting*.

4. Ioannis A. Chatzigeorgiou, Miguel R.D. Rodrigues, Ian J. Wassell and Rolando Carrasco, "Can Punctured Rate-1/2 Turbo Codes Achieve a Lower Error Floor than their Rate-1/3 Parent Codes?", *2006 IEEE Information Theory Workshop* (ITW'06), Chengdu, China, October 2006.

5. Ioannis A. Chatzigeorgiou, Miguel R.D. Rodrigues, Ian J. Wassell and Rolando Carrasco, "A Novel Technique To Evaluate the Transfer Function of Punctured Turbo Codes", *2006 IEEE International Conference on Communications* (ICC'06), Istanbul, Turkey, June 2006.

6. Ioannis A. Chatzigeorgiou, Miguel R.D. Rodrigues, Ian J. Wassell and Rolando Carrasco, "A Novel Technique for the Evaluation of the Transfer Function of Parallel Concatenated Convolutional Codes", *4th International Symposium on Turbo Codes and Related Topics* (ISTC'06), Munich, Germany, April 2006.

7. Ioannis A. Chatzigeorgiou, Miguel R.D. Rodrigues, Ian J. Wassell and Rolando Carrasco, "Punctured Binary Turbo-Codes with Optimized Performance", *62nd IEEE Semi-annual Vehicular Technology Conference* (VTC'05-Fall), Dallas, Texas, USA, September 2005.

8. Miguel R.D. Rodrigues, Ioannis A. Chatzigeorgiou, Ian J. Wassell and Rolando Carrasco, “On the Performance of Turbo Codes in Quasi-Static Fading Channels”, *2005 IEEE International Symposium on Information Theory* (ISIT’05), Adelaide, Australia, September 2005.

9. Ioannis A. Chatzigeorgiou, Miguel R.D. Rodrigues, Ian J. Wassell and Rolando Carrasco, “A Comparison of Convolutional and Turbo Coding Schemes for Broadband FWA Systems”, *12th International Conference on Telecommunications* (ICT’05), Cape Town, South Africa, May 2005.

10. Ioannis A. Chatzigeorgiou, Miguel R.D. Rodrigues, Ian J. Wassell and Rolando Carrasco, “Turbo Coded OFDM/SC-FDE Techniques for MIMO Broadband FWA Channels”, *1st International Symposium on Broadband Communications* (ISBC’04), Harrogate, Yorkshire, UK, December 2004.

# Contents

# List of Figures

# List of Tables

# Glossary

## Abbreviations

| | |
|---|---|
| AWGN | Additive White Gaussian Noise |
| BCJR | Bahl, Cocke, Jelinek and Raviv [decoding algorithm] |
| CPV | Column Puncturing Vector |
| EXIT | EXtrinsic Information Transfer [chart] |
| IOWEF | Input-Output Weight Enumerating Function |
| IRWEF | Input-Redundancy Weight Enumerating Function |
| LDPC | Low Density Parity Check [code] |
| LLR | Log-Likelihood Ratio |
| MAP | Maximum A-Posteriori |
| ML | Maximum Likelihood |
| NRNSC | Non-Recursive Non-Systematic Convolutional [code] |
| NS | Non Systematic |
| PCCC | Parallel Concatenated Convolutional Code |
| PN | Pseudo Noise [sequence] |
| PS | Partially Systematic |
| RSC | Recursive Systematic Convolutional [code] |
| SiSo | Soft-input Soft-output [decoder] |
| SOVA | Soft-Output Viterbi Algorithm |
| Sys | Systematic |
| WEF | Weight Enumerating Function |

## Symbols

| | |
|---|---|
| **bold** | Symbols in bold letters denote matrices, vectors and sequences |
| *italics* | Symbols in italic letters denote variables, constants and functions |
| $W$,$U$,$Z$ | Indeterminate variables associated with the information sequence, the systematic sequence and the parity check sequence, respectively |
| $D$ | Indeterminate variable associated with the codeword sequence |
| $L$ | Indeterminate variable associated with the trellis path length |
| $w$,$u$,$z$ | Hamming weight of an information sequence, a systematic sequence and a parity check sequence, respectively |
| $d$ | Hamming weight of a codeword sequence |
| $l$ | Length of a trellis path |
| $T(W, U, Z, L)$ | Transfer function of a convolutional code |
| $B(W, U, Z)$ | Transfer function of a convolutional block code |
| $\mathbf{G}$ | Generator vector |
| $\mathbf{P}$ | Puncturing pattern |
| $\mathbf{P_j}$ | $j$-th column puncturing vector |
| $d_f$ | Free distance |
| $d_{\text{free,eff}}$ | Free effective distance |
| $E_b$ | Energy per transmitted coded bit |
| $G(D)$ | Generator polynomial |
| $\mathsf{L}$ | Period of a feedback generator polynomial |
| $M$ | Period of a puncturing pattern |
| $N$ | Input block size, or equivalently, interleaver size |
| $N_0$ | Noise density |
| $R$ | Code rate |
| $P_b$ | Bit error probability |
| $P_b^u$ | Union bound on the bit error probability |
| $P(w)$ | Contribution to the union bound of codeword sequences having information weight $w$ |
| $\Lambda(x)$ | Log-likelihood ratio associated with a bit $x$ |

| | |
|---|---|
| $\nu$ | Memory size of an encoder |
| $\mathcal{C}$ | A convolutional code |
| $\mathcal{P}$ | A parallel concatenated convolutional code |
| dist($\mathbf{x}$,$\mathbf{y}$) | Hamming distance between codeword sequences $\mathbf{x}$ and $\mathbf{y}$ |
| rem($i$,$j$) | Remainder of the division between $i$ and $j$ |
| $\lfloor \xi \rfloor$ | Integer part of a real number $\xi$ |

# Chapter 1

# Introduction

This dissertation is concerned primarily with the development of techniques to accurately evaluate the full transfer function of a turbo code, and alternatively, to quickly compute the most significant terms of it. As a result, we can obtain tight upper bounds, or bound approximations respectively, on the bit error probability of turbo codes, operating in the error floor region. In particular, we concentrate on the performance analysis of punctured turbo codes, owning to the increased bandwidth efficiency they provide, which is attractive to wireless applications.

Although this brief description of our work may appear to be obscure at first, the motivation and application of our work will hopefully become evident as the thesis unfolds. However, before explaining the particulars of turbo codes and presenting our contributions, we first give a very short overview of the history of channel coding, from Shannon's theorem on channel capacity to today's dominant capacity-approaching codes. Next, we stress the importance of bounding techniques to the performance analysis and evaluation of turbo codes in particular, and we conclude the introductory chapter by giving an outline of the thesis structure.

## 1.1 A Brief History of Channel Coding

The history of channel coding, also referred to as *forward error correction*, dates back to Shannon's pioneering work [1] in 1948, in which he showed that it is possible to design codes with any desired small probability of error, whenever the transmission rate is smaller than the capacity of the channel. Unfortunately, Shannon provided no insights on how to actually design these codes.

Until the late 1940s, communication devices were equipped with error detection capabilities only. Hamming was the first to propose a single-error correcting code [2] in 1950, while Golay developed a more efficient scheme able to correct up to three erroneous bits [3]. Both Hamming and Golay codes group blocks of information bits together with parity check bits, the latter being computed using a mathematical combination of the information bits. Such codes are known as *block codes*. Popular variations of block codes are the *Reed-Muller* and the *cyclic redundancy codes*. Subclasses of cyclic redundancy codes, such as the *Bose-Chaudhuri-Hocquenghem* and *Reed-Solomon* codes, are still used in a wide variety of applications [4].

In 1955, Elias introduced the concept of *convolutional coding* [5]. The convolutional encoder makes use of shift registers to generate output bits based on the present input bit as well as past inputs. Contrary to block codes, partitioning of the information sequence into blocks is not required. The main advantage of convolutional codes over block codes is better error rate performance, owing to the optimal exploitation of soft channel observations by the decoding algorithm. In particular, Viterbi proposed a maximum likelihood sequence estimation algorithm [6] in 1967, while a more efficient but more complex algorithm, based on maximum a-posteriori decoding, was developed by Bahl *et al.* [7] in 1974. Convolutional coding was initially introduced in standards for satellite communication applications and deep space missions, but was later also adopted in mobile communication systems.

*Concatenation* of codes was the next significant step that enabled better performance of codes on communication channels that introduced burst errors. More specifically, Forney showed [8] in 1966 that a concatenated coding system with a powerful outer code can perform reasonably well, when its inner decoder is operated in the high error probability region. In principle, block encoders can be combined with convolutional encoders and interleavers, in parallel or serial schemes. A popular scheme, initially developed for NASA, combines a conventional convolutional inner code with a powerful Reed-Solomon outer code [9].

For years to come, channel coding was considered to have limited applicability to communications systems with bandwidth restrictions, hence research interest shifted to joint coding and modulation, with Ungerböck proposing an efficient scheme in 1982, known as *trellis-coded modulation* [10]. However, in 1993, interest shifted back to channel coding, when Berrou, Glavieux and Thitimajshima presented *turbo codes* [11]. Traditionally, a turbo encoder is the parallel concatenation of two convolutional codes separated by an interleaver. However, the name "turbo" comes from the similarity in logic between the turbo engine and the iterative decoding process between the two component soft-input soft-output (SiSo) convolutional decoders at the receiver. Owing to their spectacular error rate performance on additive white Gaussian (AWGN) channels as well as fast fading channels, turbo codes attracted intensive research in companies and universities all over the world, and they were eventually standardized in third generation mobile systems for image, video and mail transmissions, fixed wireless access systems, digital audio broadcasting and satellite communications [12].

Turbo codes made researchers realize the importance of the turbo principle, which can be extended to other concatenated schemes, such as modulation and coding or equalization and coding. Furthermore, the belief that other capacity-approaching codes existed was strengthened and, consequently, interest in channel coding was reignited. As a result, a family of codes known as *low-density parity check* (LDPC) codes, invented in the early 1960s by Gallager [13] and largely forgotten since then, was rediscovered in the late 1990s [14]. Like turbo codes, they also attain capacity by means of an iterative decoding process and they also get close to the theoretical Shannon limit. Today, turbo codes and LDPC codes have proven to be serious contenders for inclusion in next generation wireless network standards.

## 1.2 Performance Evaluation of Turbo Codes

Conventionally, the performance of a digital communication system consisting of a transmitter, a communication channel and a receiver, is measured by evaluating the probability of a bit error at the output of the receiver, over a range of signal-to-noise ratio values at the input of the receiver [15]. The main tools for the performance evaluation of a communication system is either *computer simulation* or the derivation of analytic expressions for the *bit error probability.* Computer simulation generates reliable error probability estimates as low as $10^{-6}$ for low signal-to-noise ratios. Performance analysis at lower error probabilities and, subsequently, higher signal-to-noise ratios, using computer simulation is time-consuming and impractical, hence an analytic approach is the only alternative.

When a simple uncoded communication system is considered, such as a two-level modulation scheme over a memoryless channel, a hard decision is made upon reception of each individual modulated bit. In such cases, it is relatively simple to derive an exact expression for the probability of error, provided that the probability distribution of noise on the channel is known. When high order modulation schemes are used, where bits are grouped into symbols, or when channel codes are employed, where a decoding decision can only be made upon reception of a sequence of bits, mathematical development of an exact expression for the probability of error is either too complex or impractical. In such cases, the usual approach is to derive an *upper bound* on the error probability.

A standard approach to upper bound the error probability is by taking the sum of the error probabilities for all possible erroneous events. In additive white Gaussian noise channels this upper bound, known as the *union bound*, progressively approaches and, at high values of signal-to-noise ratios, eventually merges with the curve of the actual performance of a turbo code [16]. Nevertheless, improved bounds that predict the performance at low signal-to-noise ratios [17] and on quasi-static fading channels [18] have also been derived.

## 1.3 Thesis Organization and Main Contributions

We begin our journey by providing an overview of convolutional and turbo codes in *Chapter 2*. We describe their structure, their representation and the optimal and suboptimal decoding algorithms employed. We review the analytic expressions for the evaluation of their bit error probability and we stress the significance of the relationship between the transfer function of a turbo code and the transfer functions of its constituent convolutional codes, when a uniform interleaver is used. In particular, we explain that knowledge of the transfer function of each constituent code allows the derivation of a union bound on the error rate performance of the corresponding turbo code, which, in turn, gives an accurate estimate of the error floor.

*Chapter 3* reports on the first contribution, an efficient method to compute the transfer function of a constituent code, based on the novel concept of the augmented state diagram. Existing techniques are presented and compared to our proposed approach, while the accuracy of our technique is validated by comparing simulation results to theoretical bounds.

Our proposed method can be used to obtain the transfer functions of constituent codes, when non-punctured rate-1/3 turbo codes are considered. In *Chapter 4* we extend the concept of the augmented state diagram to punctured convolutional codes, which can be used to construct turbo codes having rates higher than 1/3. We discuss the computational complexity of the modified method and we present ways to reduce it. As expected, tight bounds on the average performance of punctured turbo codes can be subsequently derived. Alternatively, we demonstrate that our approach can also be used to identify puncturing patterns that lead to high-rate turbo codes yielding low error floors.

Our technique can be used to derive the full transfer function of constituent codes separated by an interleaver of small or moderate size. When long interleavers are used though, computation becomes intensive and time-consuming. In *Chapter 5* we invoke a property of turbo codes according to which, codeword sequences generated by input information sequences having small Hamming weight play an increasingly significant role in determining the error rate performance, as the interleaver becomes larger. Motivated by this property, we present a simple method to quickly enumerate only the codeword sequences having low information weight, instead of all the codeword sequences comprising the full transfer function. Consequently, we can restrict ourselves to the methods based on the augmented state diagram when short interleavers are used, while we can use the technique introduced in this chapter to quickly obtain an accurate approximation of the performance upper bound of turbo codes, when long interleavers are employed.

*Chapter 6* opens by comparing the performance bound approximation of rate-1/2 punctured turbo codes using long interleavers to the bound approximation of their rate-1/3 parent turbo codes. Surprisingly, we observe that certain configurations of rate-1/2 punctured turbo codes yield a lower bound than that of their rate-1/3 parent codes. We perform an extensive analysis to demonstrate that a specific family of rate-1/2 turbo codes, which we call pseudo-randomly punctured turbo codes, always exhibit a lower error floor than that of their rate-1/3 parent codes, while their performance always converges towards low bit error probabilities, for an increasing number of iterations. This result reveals that certain puncturing patterns can be used to reduce the rate of a turbo code from 1/3 to 1/2 and at the same time improve the code performance at low bit error probabilities.

*Chapter 7* concludes the dissertation summarizing the main contributions and proposing future lines of work.

A brief description of the software tools we developed during the course of our work, is given in the *Appendix*.

Chapter 2

# An Overview of Convolutional and Turbo Codes

## 2.1 Introduction

Block codes and convolutional codes are the two major classes of codes for error correction. A turbo code, as it was originally conceived, is the parallel concatenation of convolutional codes separated by interleavers. Hence, the focus of the first part of this chapter is on convolutional codes. We give a brief account of their structure, their most common types and their representation using graphs. The fundamentals of the decoding operation are described in more detail, since they form the basis for the computation of a tight upper bound on the error probability, which in turn gives us insight into the performance of a convolutional code.

The second part of this chapter is concerned with the characteristics of turbo codes. The generic structure of the encoder is presented and the concept of iterative decoding is introduced. We briefly describe the optimal decoding algorithm as well as suboptimal but more practical alternatives. The second part concludes, demonstrating the dependency of the error probability of turbo codes on the properties of their constituent convolutional codes.

Example cases for both convolutional and turbo codes are given throughout the chapter.

## 2.2 The Convolutional Encoder

Convolutional codes, first introduced by Elias [5], are linear, time-invariant, finite-memory systems that output a codeword sequence for every input information se-

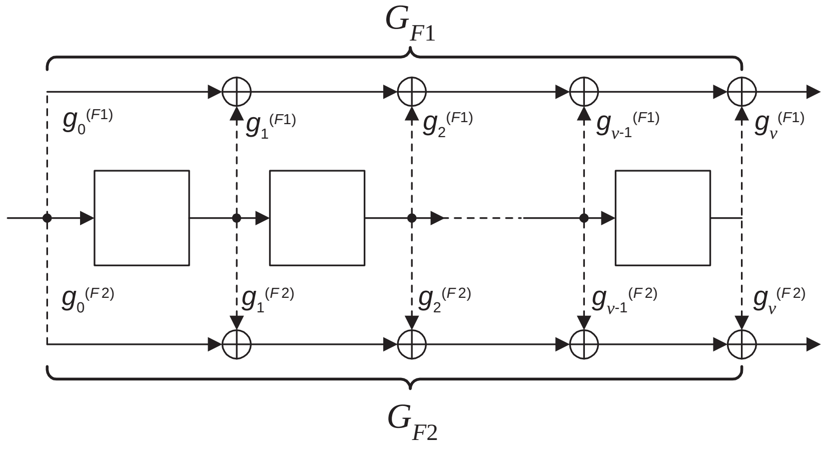


Figure 2.1: Schematic for a non-recursive non-systematic convolutional encoder.

quence. Conceptually, information and codeword sequences are of infinite length. The schematic of a binary convolutional encoder with one input, two outputs and $\nu$ memory elements in the form of shift registers, is shown in Fig.2.1. It is assumed that the encoder starts with all $\nu$ registers clear. In this case, we say that the encoder is in the *zero state*. At each time step, an input information bit is modulo-2 added to the stored values, depending upon the presence (or not) of connections between the registers and the modulo-2 adders. As a consequence, two output bits are generated and all bits stored in the registers are shifted to the right, while the input bit is moved to the leftmost register.

A measure of the redundancy introduced by a code, is given by the ratio $R$ of the number of input information bits to the output coded bits and is known as the *code rate*. In the case of convolutional codes, this is also equivalent to the ratio of the number of inputs to the number of outputs of the encoder[1]. Our example convolutional code has a code rate $R=1/2$.

The encoder in Fig.2.1, is also known as a non-systematic non-recursive convolutional (NRNSC) encoder. An encoder is said to be *systematic*, if the input information sequence appears unchanged at one of its outputs. Furthermore, the encoder is called *recursive*, when the input does not directly affect the memory state of the encoder, due to the presence of a feedback loop. The schematic of a rate 1/2 recursive systematic convolutional (RSC) is shown in Fig.2.2.

A convolutional encoder can be described by the connections between the input of the shift registers and the modulo-2 adders. To describe the NRNSC encoder of Fig.2.1, we use two row-vectors, $\mathbf{G}_{F1}=[g_0^{F1}g_1^{F1}\dots g_\nu^{F1}]$ and $\mathbf{G}_{F2}=[g_0^{F2}g_1^{F2}\dots g_\nu^{F2}]$,

[1]Throughout this thesis a code rate of 1/2 is always assumed, unless otherwise stated.

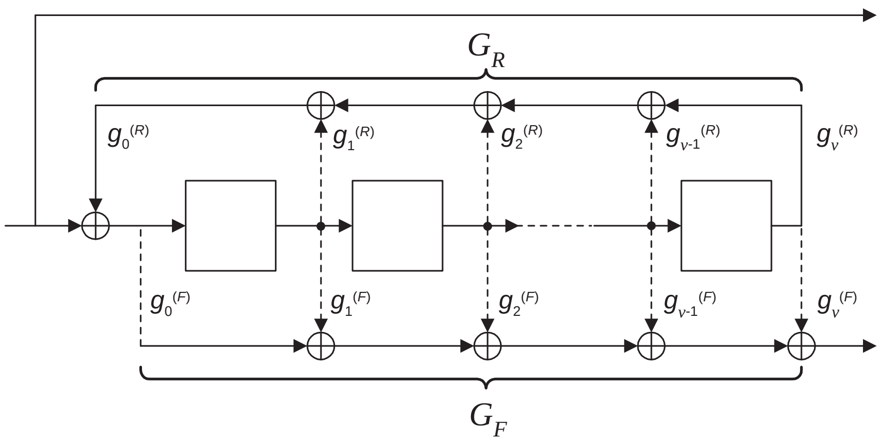
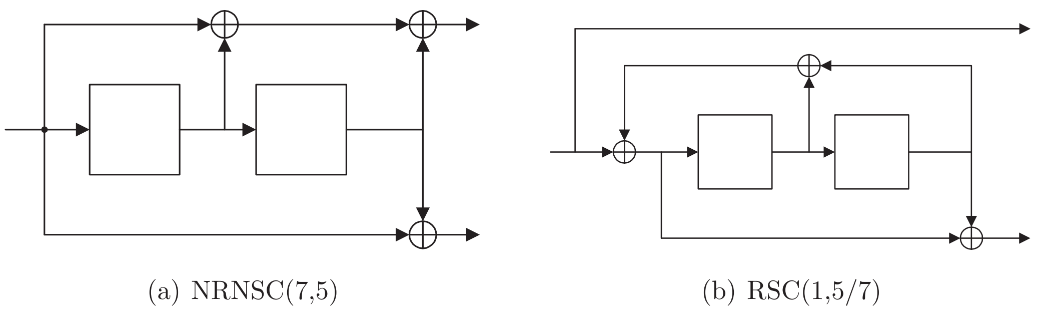


Figure 2.2: Schematic for a recursive systematic convolutional encoder.

which we call *generator vectors*. An element of a generator vector is set to “1” if there is a connection between the encoder input or the output of the corresponding shift register and the modulo-2 adder, otherwise it is set to “0”. In the case of the RSC encoder of Fig.2.2, we need a generator vector $\mathbf{G}_R$ to describe the connections of the feedback loop and a generator vector $\mathbf{G}_F$ to describe the connections of the feed-forward path. In order to distinguish between NRNSC and RSC encoders, we use the notation NRNSC($\mathbf{G}_{F1}$,$\mathbf{G}_{F2}$) and RSC(1,$\mathbf{G}_F/\mathbf{G}_R$), where “1” corresponds to the systematic output of the RSC encoder. Note that the generator vectors are more conveniently represented as octal numbers. For example, the NRNSC encoder of Fig.2.3(a) has generator vectors $\mathbf{G}_{F1}=[111]=7_8$ and $\mathbf{G}_{F2}=[101]=5_8$, whereas the RSC encoder of Fig.2.3(b) can be described by $\mathbf{G}_R=[111]=7_8$ and $\mathbf{G}_F=[101]=5_8$. A more compact representation of each encoder is NRNSC(7,5) and RSC(1,5/7), respectively.

Other significant parameters that characterise a convolutional code are the *memory size* or *memory order*, which is equal to the number of shift registers $\nu$ used by



(a) NRNSC(7,5) (b) RSC(1,5/7)

Figure 2.3: Block diagrams of two convolutional encoders.

the encoder, and the *constraint length*, which refers to the total number of bits involved in the encoding operation at each time step. For our example cases of rate 1/2 convolutional codes, the constraint length is equal to $\nu + 1$.

### 2.2.1 Encoder State Diagram and Trellis Representation

Generator vectors are an implementation-oriented way to describe convolutional codes. However, other ways of representing a convolutional encoder are possible. A convolutional encoder has a finite number of shift registers $\nu$ and therefore a finite number of memory states $2^\nu$. Hence, an encoder can be seen as a finite-state machine and can also be described by a graph, known as the *state diagram*.

The state diagram of a binary convolutional encoder consists of nodes, that represent the possible memory states of the encoder. A branch that interconnects two nodes, corresponds to the transition from one state to another state, caused by an input bit. Each branch is labeled by the input information bit and the output coded bits generated during the state transition. The RSC encoder in Fig.2.3(b) with $\nu = 2$ registers, has $2^\nu = 4$ possible memory states $s \in \{(00), (01), (10), (11)\}$. The state diagram for this encoder is shown in Fig.2.4. There are two branches leaving each state, since the encoder accepts only one input bit at a time, which can take two possible values causing an equal number of state transitions. We have labeled each branch with the input bit and the output bit generated by the feed-forward path, known as *parity check bit*. Due to the systematic nature of the encoder, the other output bit, which is called *systematic bit*, is identical to the input information bit. As an example, consider the state transition $(10) \rightarrow (11)$, which is labeled with 0/1. We understand that if a “0” is input to the encoder when its memory is in state (10), a transition to state (11) will occur, generating a parity check bit equal to “1”. If we take into account the systematic bit as well, the coded output will be “01”.

We can use the information bits of an input sequence to draw a path along the state diagram and thus determine the state transitions and the corresponding output codeword sequence. However, as the length of the input sequence increases, it becomes difficult to trace the path, since the same states are visited multiple times. It becomes evident that the state diagram needs to be modified, so as to take time into account. Such a modified diagram is known as a *trellis*.

The nodes of a trellis diagram, which refer to the memory states of the encoder, are drawn along the same vertical line and are replicated at each time step. The nodes of two consecutive time steps are interconnected, using the same logic as in the

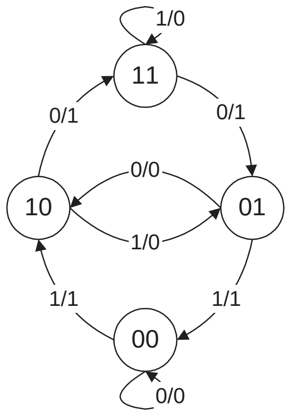


Figure 2.4: State diagram for the RSC(1,5/7) code.

case of the state diagram. The state diagram in Fig.2.4 can be modified so as to give the trellis diagram in Fig.2.5. For example, assume that the information sequence "1-0-1-1" is input to the encoder, which is initially in the zero state. Looking at the trellis diagram of the rate-1/2 RSC encoder in Fig.2.5, it is clear that the input sequence causes the state transitions (00) $\rightarrow$ (10) $\rightarrow$ (11) $\rightarrow$ (11) $\rightarrow$ (11), or, equivalently, $0 \rightarrow 2 \rightarrow 3 \rightarrow 3 \rightarrow 3$ in decimal form, which in turn generates the codeword sequence "11-01-10-10". Note that each pair in the codeword sequence consists of the systematic bit and the parity check bit.

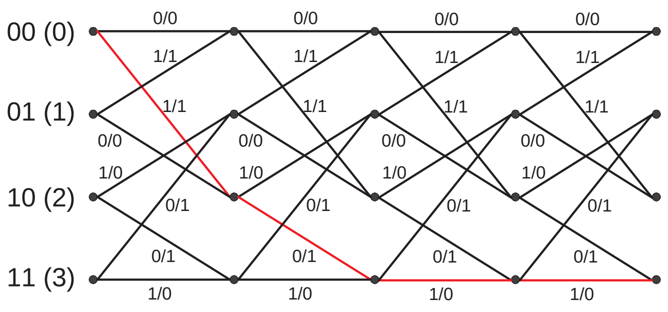


Figure 2.5: Trellis diagram for the RSC(1,5/7) code.

## 2.3 The Convolutional Decoder

In order to discuss the decoding of convolutional codes, we first briefly review the underlying transmission system and the fundamental probabilistic concepts applied by the decoding algorithm.

A binary information sequence $\mathbf{u}=(u_1, u_2, \ldots, u_t, \ldots)$ is input to a convolutional encoder. Polar signalling, which is the baseband equivalent of binary phase shift keying (BPSK), is used to map the output coded bits to antipodal symbols. We assume that each coded bit equal to "0" is mapped to "+1", otherwise it is mapped to "$-1$". The transmitted modulated codeword sequence $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t, \ldots)$ has the same length as the input sequence $\mathbf{v}$. A codeword $\mathbf{x}_t$ in $\mathbf{x}$, where $t > 0$, consists of the modulated coded bits generated by the convolutional encoder, when $u_t$ was the input bit. For example, if one of the rate-1/2 encoders of Fig.2.3 was used, an input bit $u_t$ would cause the generation of the codeword $\mathbf{x}_t = (x_t^{(1)}, x_t^{(2)})$, where $x_t^{(1)}$ and $x_t^{(2)}$ are the modulated coded bits. The received codeword sequence $\mathbf{y}=(\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_t, \ldots)$, which is impaired by the channel noise, needs to be decoded so as to obtain an estimate $\hat{\mathbf{x}}$ of the transmitted sequence $\mathbf{x}$.

Decoding of the received sequence $\mathbf{y}$ is an *inverse probability* problem, since it involves computing the conditional probability of the transmitted sequence $\mathbf{x}$, given the received sequence $\mathbf{y}$, i.e., $P(\mathbf{x} \mid \mathbf{y})$. We can obtain $P(\mathbf{x} \mid \mathbf{y})$, using Bayes' theorem

$$P(\mathbf{x} \mid \mathbf{y}) = \frac{P(\mathbf{y} \mid \mathbf{x})P(\mathbf{x})}{P(\mathbf{y})}. \tag{2.1}$$

We refer to $P(\mathbf{x} \mid \mathbf{y})$ as *a-posteriori* probability, whereas $P(\mathbf{x})$ is known as *a-priori* or *intrinsic* probability. The conditional probability density function $P(\mathbf{y} \mid \mathbf{x})$ depends on the communication channel and is called the *likelihood* of $\mathbf{x}$.

The goal of the convolutional decoder is to find the codeword sequence $\hat{\mathbf{x}}$ for which the a-posteriori probability $P(\mathbf{x} \mid \mathbf{y})$ is maximized. The normalizing factor $P(\mathbf{y})$ is a constant and can be neglected when finding the optimal decoding decision, therefore

$$\hat{\mathbf{x}} = \arg\max_{\mathbf{x}} \left(P(\mathbf{x} \mid \mathbf{y})\right) = \arg\max_{\mathbf{x}} \left(P(\mathbf{y} \mid \mathbf{x})P(\mathbf{x})\right). \tag{2.2}$$

This approach is known as *maximum a-posteriori* (MAP) decoding. It is common practice to assume that all codeword sequences are equally likely, thus $P(\mathbf{x})$ is also a constant and need not be computed, i.e.,

$$\hat{\mathbf{x}} = \arg\max_{\mathbf{x}} \left(P(\mathbf{y} \mid \mathbf{x})\right). \tag{2.3}$$

This method is called *maximum-likelihood* (ML) decoding.

Probabilities usually describe frequencies of outcomes in random experiments. However, probabilities can also be used to quantify a belief in a proposition. Decoding can be seen as a process that makes decisions based on the observation of the received sequence and provides degrees of belief, or else *reliability values*, associated with those decisions. In order to distinguish between probabilities and reliability values, we need to introduce the concept of the log-likelihood ratio.

### 2.3.1 Log-Likelihood Ratios

A coded bit $x_t^{(i)}$ in codeword $\mathbf{x}_t$ of the output sequence $\mathbf{x}$ is a random variable, which takes on a value from the set $\{-1, +1\}$ with probability $P(x_t^{(i)})$. The *log-likelihood ratio* (LLR) associated with the coded bit $x_t^{(i)}$ is defined as

$$\Lambda(x_t^{(i)}) = \ln \frac{P(x_t^{(i)} = +1)}{P(x_t^{(i)} = -1)}. \tag{2.4}$$

The sign of $\Lambda(x_t^{(i)})$ is the hard decision on the coded bit, whereas the magnitude $|\Lambda(x_t^{(i)})|$ is the reliability of this decision. For example, if $\Lambda(x_t^{(i)}) > 0$, then $x_t^{(i)}$ should be equal to "+1", with reliability $|\Lambda(x_t^{(i)})|$. We refer to $\Lambda(x_t^{(i)})$ as the LLR, or the *soft value*, of the coded bit $x_t^{(i)}$. Hagenauer *et al.* [19] presented an extensive analysis of the properties and the algebra of log-likelihood ratios.

If the LLR value $\Lambda(x_t^{(i)})$ is known, expressions for the probabilities $P(x_t^{(i)} = +1)$ and $P(x_t^{(i)} = -1)$ can be derived from (2.4), as follows

$$\begin{aligned} P\left(x_t^{(i)} = +1\right) &= \frac{exp\left(\Lambda(x_t^{(i)})\right)}{1 + exp\left(\Lambda(x_t^{(i)})\right)}, \\ P\left(x_t^{(i)} = -1\right) &= \frac{1}{1 + exp\left(\Lambda(x_t^{(i)})\right)}, \end{aligned} \tag{2.5}$$

and can be merged into a single compact expression

$$P\left(x_t^{(i)}\right) = \frac{exp\left(0.5\Lambda(x_t^{(i)})\right)}{1 + exp\left(\Lambda(x_t^{(i)})\right)} \cdot exp\left(0.5 x_t^{(i)} \Lambda(x_t^{(i)})\right). \tag{2.6}$$

Based on the definition of the log-likelihood ratio, the LLR of the received coded bit $y_t^{(i)}$ given the transmitted coded bit $x_t^{(i)}$, is equal to

$$\Lambda(y_t^{(i)} \mid x_t^{(i)}) = \ln \left( \frac{P(y_t^{(i)} \mid x_t^{(i)} = +1)}{P(y_t^{(i)} \mid x_t^{(i)} = -1)} \right). \tag{2.7}$$

The output of a discrete memoryless fading channel with additive white Gaussian noise, can be described statistically by the probability density function

$$P(y_t^{(i)} \mid x_t^{(i)}) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left( -\frac{E_c}{2\sigma^2} (y_t^{(i)} - x_t^{(i)})^2 \right), \tag{2.8}$$

where $\sigma^2 = N_0/2$ is the noise variance and $E_c$ is the transmitted energy per coded bit. If $E_b$ is the energy per input information bit and $R$ is the rate of the convolutional code, then $E_c = E_b R$. Substituting (2.8) in (2.7), we obtain

$$\Lambda(y_t^{(i)} \mid x_t^{(i)}) = \ln \left( \frac{\exp\left(-\frac{E_b R}{2\sigma^2}(y_t^{(i)} - 1)^2\right)}{\exp\left(-\frac{E_b R}{2\sigma^2}(y_t^{(i)} + 1)^2\right)} \right), \tag{2.9}$$

which can be further reduced to

$$\Lambda(y_t^{(i)} \mid x_t^{(i)}) = \Lambda_c y_t^{(i)}, \tag{2.10}$$

where

$$\Lambda_c = 4E_b R/N_0, \tag{2.11}$$

is known as the *channel reliability.* A relation between $P(y_t^{(i)} \mid x_t^{(i)})$ and $\Lambda(y_t^{(i)} \mid x_t^{(i)})$ can be found in a similar manner to (2.6), i.e.,

$$P(y_t^{(i)} \mid x_t^{(i)}) = \frac{exp\left(0.5\Lambda_c y_t^{(i)}\right)}{1 + exp\left(\Lambda_c y_t^{(i)}\right)} \cdot exp\left(0.5\Lambda_c x_t^{(i)} y_t^{(i)}\right). \tag{2.12}$$

## 2.3.2 Maximum-Likelihood Decoding

If we assume that the coded bits in the output sequence $\mathbf{x}$ are independent random variables, expression (2.2) can be rewritten for the case of an AWGN communication channel, as follows

$$\hat{\mathbf{x}} = \arg\max_{\mathbf{x}} \left( \prod_t \prod_i P(y_t^{(i)} \mid x_t^{(i)}) P(x_t^{(i)}) \right). \tag{2.13}$$

We introduce the LLR values associated with $P(x_t^{(i)})$ and $P(y_t^{(i)} \mid x_t^{(i)})$ using (2.6) and (2.12), respectively, but we neglect all terms that remain constant during the decoding operation. The estimated codeword sequence $\hat{\mathbf{x}}$ can be obtained as follows

$$\hat{\mathbf{x}} = \arg\max_{\mathbf{x}} \left( 0.5\Lambda_c \sum_t \sum_i x_t^{(i)} y_t^{(i)} + 0.5 \sum_t \sum_i x_t^{(i)} \Lambda(x_t^{(i)}) \right). \tag{2.14}$$

Since in ML decoding there is no knowledge of the a-priori LLR, $\Lambda(x_t^{(i)})$, it is assumed that the coded bits are uniformly distributed, hence $P(x_t^{(i)} = +1) = P(x_t^{(i)} = -1) = 0.5$ and, consequently, $\Lambda(x_t^{(i)}) = 0$. The scaling factor $0.5\Lambda_c$ can be ignored, so the ML decoding rule reduces to

$$\hat{\mathbf{x}} = \arg\max_{\mathbf{x}} \left( \sum_t \sum_i x_t^{(i)} y_t^{(i)} \right). \tag{2.15}$$

Although, conceptually, both the transmitted and the received codeword sequence are of infinite length, we assume for practical reasons that $t = 1, \ldots, N$, where $N$ is a large number. If the rate-1/2 convolutional decoder makes a hard decision on the received sequence $\mathbf{y}$ before performing the ML decoding operation, i.e., each received coded bit $y_t^{(i)}$ is set to "+1" if its value is greater than zero, otherwise it is set to "−1", the ML decoding rule reduces to

$$\hat{\mathbf{x}} = \arg\max_{\mathbf{x}} \left(N - \mathrm{dist}(\mathbf{x}, \mathbf{y})\right) \tag{2.16}$$

or, equivalently,

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} \left(\mathrm{dist}(\mathbf{x}, \mathbf{y})\right), \tag{2.17}$$

where $\mathrm{dist}(\mathbf{x}, \mathbf{y})$ denotes the number of coded bits that sequences $\mathbf{x}$ and $\mathbf{y}$ differ, known as *Hamming distance*. Consequently, the estimated codeword sequence $\hat{\mathbf{x}}$ is the sequence among all valid codeword sequences with the smallest Hamming distance from the received codeword sequence. The minimum Hamming distance between any two codeword sequences $\mathbf{x}'$ and $\mathbf{x}''$ of a convolutional code is known as *free distance* and is defined as

$$d_f = \min_{\mathbf{x}' \neq \mathbf{x}''} \mathrm{dist}(\mathbf{x}', \mathbf{x}''). \tag{2.18}$$

We conclude from (2.17) and (2.18) that the free distance plays an important role in the error rate performance of a convolutional code; the larger its free distance is, the more resilient to errors the codeword sequences are.

### 2.3.3 The Viterbi Algorithm

Maximization of the sum of inner products (2.15) between the received coded bits in sequence $\mathbf{y}$ and the coded bits in a codeword sequence $\mathbf{x}$, can be achieved by means of an exhaustive search among all valid codeword sequences generated by the convolutional encoder. This operation can be simplified considerably, if we observe that any valid codeword sequence $\mathbf{x}$ forms a path along the trellis diagram and a number of paths merge to a particular memory state at every time step.

For simplicity, we continue to pursue the example of rate-1/2 convolutional codes. The trellis diagram for a convolutional code of memory $\nu$, has a total of $2^\nu$ states. If after $T$ time steps, where $T$ is an arbitrary number, the paths of two codeword sequences $\mathbf{x}'$ and $\mathbf{x}''$ merge in state $s \in [0, \ldots, 2^\nu - 1]$, we can apply the ML decoding rule to $\mathbf{x}'$ and $\mathbf{x}''$ for $t = 1, \ldots, T$ only, no matter what the subsequent received coded bits are. If the sum of the inner products achieved by the first $T$ codewords of sequence $\mathbf{x}'$ is greater than that achieved by sequence $\mathbf{x}''$, we can eliminate from

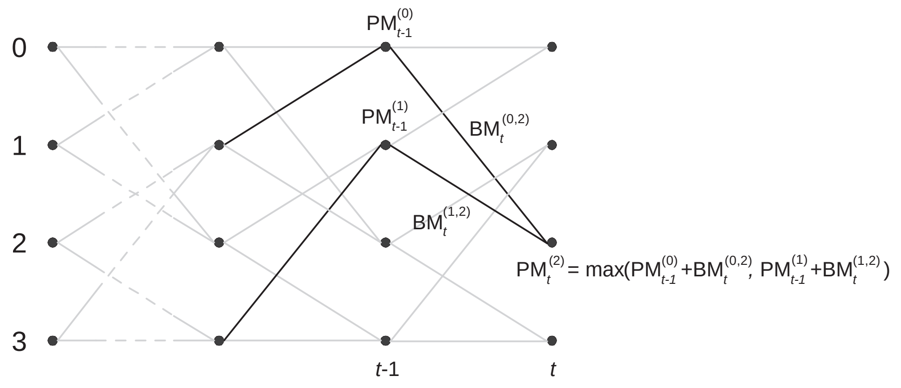

0
1
2
3
PM$_{t-1}^{(0)}$
PM$_{t-1}^{(1)}$
BM$_t^{(0,2)}$
BM$_t^{(1,2)}$
PM$_t^{(2)}$ = max(PM$_{t-1}^{(0)}$+BM$_t^{(0,2)}$, PM$_{t-1}^{(1)}$+BM$_t^{(1,2)}$)
$t$-1
$t$

Figure 2.6: Path metric calculation for Viterbi decoding.

the exhaustive search all sequences whose first $T$ codewords are identical to those of $\mathbf{x}''$. In order to materialize this observation, Viterbi proposed in [20] an efficient algorithm based on metrics comparison and path elimination.

For each branch of the trellis of a rate-1/2 convolutional code that interconnects two states $s', s \in [0, \dots, 2^\nu - 1]$ at a time step $t$, the *Viterbi algorithm* assigns a metric given by

$$\mathrm{BM}_t^{(s',s)} = \sum_{i=1}^{2} x_t^{(i)} y_t^{(i)} \tag{2.19}$$

where $\mathbf{x}_t = (x_t^{(1)}, x_t^{(2)})$ is the codeword associated with the branch and $\mathbf{y}_t = (y_t^{(1)}, y_t^{(2)})$ is the received codeword at time step $t$. This metric is called the *branch metric*.

A path is composed of a series of interconnected branches and the sum of their metrics composes the *path metric*, denoted as PM. As the path progresses in the trellis, subsequent branches become part of it and its metric changes accordingly. If two paths merge to a state $s$ at a time step $t$, the Viterbi algorithm selects the path with the higher metric, which is called the *survivor path*, and disregards the other. The path metric of the survivor path at a time step $t$ for a state $s$ is given by

$$\mathrm{PM}_t^{(s)} = \max\left(\mathrm{PM}_{t-1}^{(s')} + \mathrm{BM}_t^{(s',s)}, \mathrm{PM}_{t-1}^{(s'')} + \mathrm{BM}_t^{(s'',s)}\right) \tag{2.20}$$

where $s'$ and $s''$ are the states occupied by the competing paths at time step $t-1$, before they merge to state $s$ at time step $t$. The add-compare-and-select process is illustrated in Fig.2.6, for the case of a memory size 2 convolutional code.

In general, $2^\nu$ paths survive at each time step, i.e., one path per state, while all others are neglected. The corresponding $2^\nu$ path metrics are stored and updated at each time step. The most likely sequence of codewords, from time step $t=0$ to an arbitrary time step $t=T$, can be obtained by selecting the survivor path with the highest path metric, called the *ML path*. The algorithm stores the state sequence for each survivor path, so as to trace back the ML path and derive the estimated codeword sequence. The computational complexity of the Viterbi algorithm grows only linearly with the length of the input sequence, as opposed to conventional ML decoding, where computational complexity grows exponentially.

Alternative decoding methods include sequential decoding, developed by Wozencraft [21] and optimized by Fano [22], and threshold decoding, proposed by Massey [23]. Omura showed in [24] that the Viterbi algorithm, which emerged from the sequential decoding principles, yields ML decisions. Bahl *et al.* proposed a MAP decoding algorithm [7], but received little attention, because its complexity exceeded that of the Viterbi algorithm, while the advantage in bit error rate performance was small. Eventually, the Viterbi algorithm dominated and found numerous applications in high data rate communication systems.

## 2.4 Error Probability for Convolutional Codes

A thorough analysis of the performance of convolutional codes on the AWGN channel was performed by Viterbi in [20]. In this section, we shall briefly present the concepts described in [20] that lead to the derivation of upper bounds to the the bit error probability of convolutional codes.

### 2.4.1 Pairwise and First-Event Error Probabilities

For convenience, we assume that a convolutional encoder transmits a polar codeword sequence $\mathbf{x}$, with $x_t^{(i)} = +1$ for every value of $i$ and $t$, which corresponds to an all-zero binary input sequence. Performance analysis does not depend on the transmitted codeword sequence, due to the linearity of convolutional codes. The decoder receives a sequence $\mathbf{y}$, where each coded bit $y_t^{(i)}$ is an independent Gaussian variable with variance $N_0/2$ and zero mean.

If the path that corresponds to the correct codeword sequence $\mathbf{x}$ merges with a path that corresponds to an incorrect sequence $\mathbf{x}'$, the incorrect sequence will be selected only if

$$\sum_t \sum_i x_t'^{(i)} y_t^{(i)} \geq \sum_t \sum_i x_t^{(i)} y_t^{(i)}, \tag{2.21}$$

or equivalently,

$$\sum_t \sum_i \left( x_t'^{(i)} - x_t^{(i)} \right) y_t^{(i)} \geq 0, \tag{2.22}$$

based on the ML decoding rule. If $d$ is the Hamming distance between $\mathbf{x}$ and $\mathbf{x}'$, i.e., $d = \mathrm{dist}(\mathbf{x}, \mathbf{x}')$, expression (2.22) reduces to

$$2 \sum_{j=1}^{d} y_j \leq 0. \tag{2.23}$$

Indices $t$ and $i$ were replaced by $j$, which runs over the $d$ coded bits wherein paths $\mathbf{x}$ and $\mathbf{x}'$ differ.

The probability that the decoder will eventually select an incorrect sequence $\mathbf{x}'$ with $d$ coded bits in error, is known as *pairwise error probability* and is given by

$$P_d = \Pr \left\{ \sum_{j=1}^{d} y_j \leq 0 \right\}. \tag{2.24}$$

Recalling that the sum of independent Gaussian variables is also a Gaussian variable, the pairwise error probability assumes the form

$$P_d = Q \left( \sqrt{\frac{2E_b R}{N_0} d} \right), \tag{2.25}$$

where $E_b$ is the energy per transmitted coded bit, $R$ is the code rate and $Q(\xi)$ is defined as follows

$$Q(\xi) = \frac{1}{\sqrt{2\pi}} \int_{\xi}^{\infty} \exp\left( -\frac{r^2}{2} \right) dr. \tag{2.26}$$

The probability that a decision error will occur and an incorrect sequence will be selected at a certain time step, independently of the number of erroneous coded bits, is called *first-event error probability* and is denoted as $P_E$. The first-event error probability can be upper bounded by the sum of the pairwise error probabilities for all possible incorrect sequences, whose paths merge with the path of the correct sequence at that specific time step, i.e.,

$$\begin{aligned} P_E &\leq \sum_{d=d_f}^{\infty} T_d P_d \\ &= \sum_{d=d_f}^{\infty} T_d Q \left( \sqrt{\frac{2E_b R}{N_0} d} \right), \end{aligned} \tag{2.27}$$

where $T_d$ refers to the number of incorrect sequences that have the same Hamming distance $d$ to the correct sequence. Calculation of the coefficients $T_d$ requires knowledge of the transfer function of the underlying convolutional code, which is the theme of the following subsection.

### 2.4.2 Transfer Function of a Convolutional Code

The transfer function of a convolutional code enumerates all codeword sequences, other than the transmitted codeword sequence, that correspond to trellis paths which diverge from the path of the transmitted codeword sequence and re-merge with it at a later stage. For simplicity, it is assumed that the all-zero sequence is input to the convolutional encoder and, consequently, the all-zero codeword sequence is generated and transmitted. A codeword sequence is described by a monomial of the form $W^wU^uZ^zL^l$, where $W$, $U$, $Z$ and $L$ are indeterminate variables. The exponent of $W$ denotes the Hamming distance between the input sequence, that generated the codeword sequence in question, and the all-zero input sequence. In this case, the Hamming distance between the two binary sequences is the number of "1"s in the input sequence, also known as the the *Hamming weight* or, simply, the *weight* of the input sequence. Similarly, the exponents of $U$ and $Z$ denote the weight of the systematic and parity check output sequences, respectively. If the code is not systematic, $U$ can be omitted. Furthermore, if the output of a systematic code is not punctured, the exponents of $W$ and $U$ are identical. Finally, the exponent of $L$ corresponds to the length of the associated path in the trellis diagram. A generic form for the transfer function $T(W, U, Z, L)$ of a convolutional code is:

$$T(W,U,Z,L) = \sum_{w}\sum_{u}\sum_{z}\sum_{l} T_{w,u,z,l} W^w U^u Z^z L^l, \tag{2.28}$$

where $T_{w,u,z,l}$ denotes the number of codeword sequences having systematic weight $u$ and parity check weight $z$, which were generated by an input sequence of weight $w$, and correspond to paths of length $l$.

The transfer function can be obtained by modifying the state diagram of the code. We shall demonstrate the method by means of an example. The conventional state diagram of the RSC(1,5/7) code is illustrated in Fig.2.7. The label of each branch is updated so as to convey the associated input, systematic and parity check weights rather than the input and parity check bits. For example if the label of a branch was "1/0", the updated label would be "$WUL$", since both the input and the systematic bits have weight 1, whereas the parity check bit has weight 0, i.e., $W^1U^1Z^0L^1 = WUL$. Term $L$ is present in each branch, since it represents a single time step. The paths of all incorrect codeword sequences start from the zero state and re-merge to it, at some later stage, thus the self-loop at the zero state can be removed. Furthermore, we split the zero state into two separate states, namely the start state $X_S$ and the end state $X_E$, as depicted in Fig.2.7. We then express each

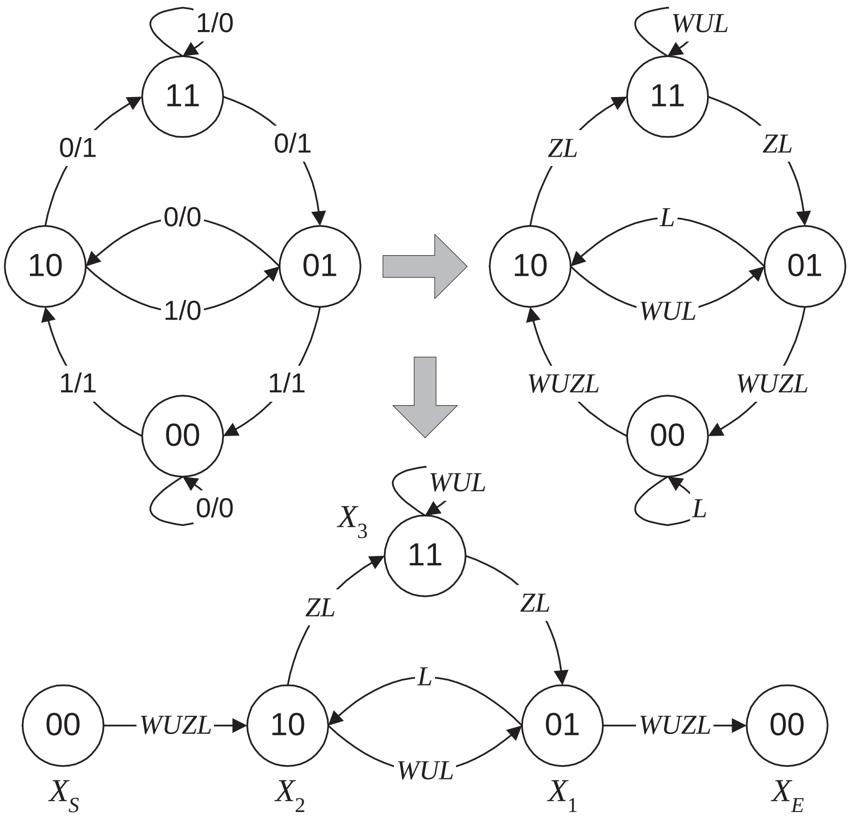


Figure 2.7: Derivation of the modified state diagram of the RSC(1,5/7) code. Top left: state diagram, top right: weight-labeled state diagram, bottom: modified state diagram.

state of the diagram as a function of the other states, so as to obtain the state equations

$$
\begin{aligned}
X_1 &= WUL \cdot X_2 + ZL \cdot X_3 \\
X_2 &= WUZL \cdot X_S + L \cdot X_1 \\
X_3 &= ZL \cdot X_2 + WUL \cdot X_3 \\
X_E &= WUZL \cdot X_1,
\end{aligned}
\tag{2.29}
$$

Upon solving these equations for the ratio $X_E/X_S$, we obtain the transfer function for RSC(1,5/7)

$$
\begin{aligned}
T(W,U,Z,L) = X_E/X_S = &W^3U^3Z^2L^3 + W^2U^2Z^4L^4 \\
&+ W^3U^3Z^4L^5 + W^4U^4Z^2L^5 + \dots,
\end{aligned}
\tag{2.30}
$$

which tells us that there is a codeword sequence of systematic weight 3 and parity check weight 2, which was generated by an input sequence of weight 3 and corresponds to a path of length 3, i.e., $W^3U^3Z^2L^3$, and so on.

Although the transfer function provides detailed information about all incorrect codeword sequences, there are more compact variations of the transfer function that provide specific information. For example, if the systematic code is not punctured and the path length of each codeword sequence is not of interest, the indeterminate variables $U$ and $L$ can be eliminated. The modified transfer function assumes the form

$$T(W,Z) = T(W,U{=}1,Z,L{=}1) = \sum_{w}\sum_{z} T_{w,z} W^w Z^z, \tag{2.31}$$

and is often called [16] the *input-redundancy weight enumerating function* (IRWEF) of the convolutional code. If we are interested in the overall output weight of each codeword sequence, we can use the *input-output weight enumerating function* (IOWEF), which is defined as

$$T(W,D) = T(W,U{=}D,Z{=}D,L{=}1) = \sum_{w}\sum_{d} T_{w,d} W^w D^d. \tag{2.32}$$

Variables $U$ and $Z$ have been replaced by $D$, allowing the addition of their exponents and, hence, conveying the overall codeword weight of a sequence. Finally, the *weight enumerating function* (WEF) yields the *distance spectrum* of the convolutional code, i.e., the number of codeword sequences $T_d$ having specific weight $d$. It is defined as

$$T(D) = T(W{=}1,U{=}D,Z{=}D,L{=}1) = \sum_{d} T_d D^d. \tag{2.33}$$

The weight enumerating function, $T(D)$, depends upon the code and influences the first-event error probability, as was demonstrated in the previous subsection, whereas the input-output weight enumerating function, $T(W,D)$, depends upon the encoder and influences the bit error probability [16, 20], as it will become evident in the next subsection.

### 2.4.3 Bit Error Probability

Whenever an erroneous decision is made, one or more information bits will be incorrectly decoded. Assume that we obtain only one incorrect sequence from the transfer function $T(W,U,Z,L)$ of the convolutional code, described by a monomial $W^w U^u Z^z L^l$, which corresponds to a path of length $l$, with input weight $w$ and overall output weight $d{=}u{+}z$. If only one decoding error was ever made, and a sequence $\mathbf{x}'$ was selected, there would be $w$ erroneous bits in the $l$ decoded information bits, due to the error event of length $l$. The probability of a bit error, given that $\mathbf{x}'$ was selected, is

$$P\left(\text{bit error} \mid \mathbf{x}'\right) = \frac{w}{l}. \tag{2.34}$$

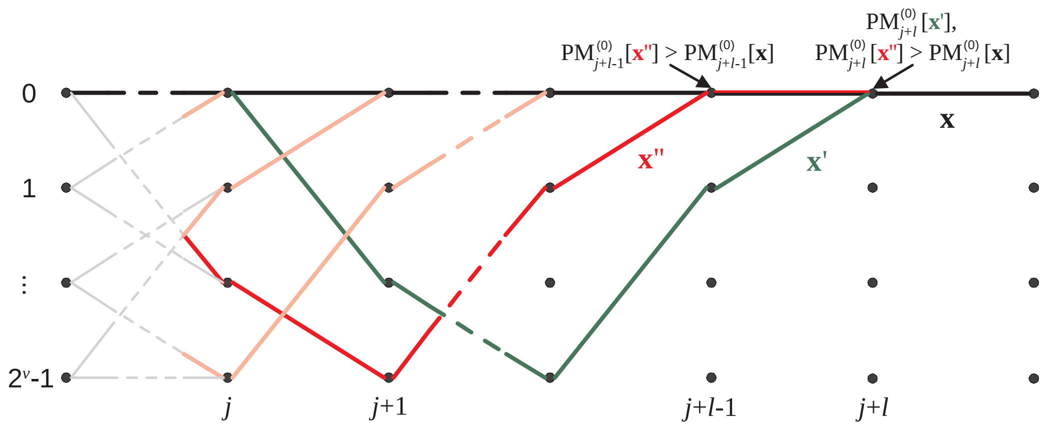

$\mathrm{PM}^{(0)}_{j+l}[\mathbf{x}']$,
$\mathrm{PM}^{(0)}_{j+l-1}[\mathbf{x}''] > \mathrm{PM}^{(0)}_{j+l-1}[\mathbf{x}]$
$\mathrm{PM}^{(0)}_{j+l}[\mathbf{x}''] > \mathrm{PM}^{(0)}_{j+l}[\mathbf{x}]$
0
1
⋮
$2^{\nu}$-1
$\mathbf{x}''$
$\mathbf{x}'$
$\mathbf{x}$
$j$
$j$+1
$j$+$l$-1
$j$+$l$

Figure 2.8: Example of a decoding decision (between $\mathbf{x}'$ and $\mathbf{x}''$), after an initial error event ($\mathbf{x}''$) has occurred.

In order to calculate the probability of a bit error $P_b$, we first need to derive the probability $P(\mathbf{x}')$ that the path related to sequence $\mathbf{x}'$ survived for $l-1$ time steps and was selected at the $l$-th time step, since $P_b$ is given by

$$P_b = P\,(\text{bit error} \mid \mathbf{x}')\,P(\mathbf{x}'). \tag{2.35}$$

The path of the selected codeword sequence $\mathbf{x}'$ spans from a time step $j$ to time step $j+l$, as shown in Fig.2.8. Assume that at time step $j+l-1$, a sequence $\mathbf{x}''$ is selected because its path metric is greater than that of the correct sequence $\mathbf{x}$, i.e., $\mathrm{PM}^{(0)}_{j+l-1}[\mathbf{x}''] > \mathrm{PM}^{(0)}_{j+l-1}[\mathbf{x}]$. At the next time step, it is also the case that

$$\mathrm{PM}^{(0)}_{j+l}[\mathbf{x}''] > \mathrm{PM}^{(0)}_{j+l}[\mathbf{x}], \tag{2.36}$$

since both paths share the same branch from time step $j+l-1$ to $j+l$. In order for $\mathbf{x}'$ to be selected at time step $j+l$, the metric of the corresponding path $\mathrm{PM}^{(0)}_{j+l}[\mathbf{x}']$ has to be greater than not only the path metric of the correct sequence $\mathrm{PM}^{(0)}_{j+l}[\mathbf{x}]$, but also the path metric of the previously selected sequence $\mathrm{PM}^{(0)}_{j+l}[\mathbf{x}'']$. Owing to (2.36), the probability of $\mathbf{x}'$ outlasting $\mathbf{x}''$ is bounded by the the probability of $\mathbf{x}'$ outlasting the correct sequence $\mathbf{x}$, i.e.,

$$\Pr\{\mathrm{PM}^{(0)}_{j+l}[\mathbf{x}'] > \mathrm{PM}^{(0)}_{j+l}[\mathbf{x}'']\} \leq \Pr\{\mathrm{PM}^{(0)}_{j+l}[\mathbf{x}'] > \mathrm{PM}^{(0)}_{j+l}[\mathbf{x}]\}. \tag{2.37}$$

The right hand side of the inequality is the pairwise probability $P_d$ between the codeword sequence $\mathbf{x}'$ and the correct sequence $\mathbf{x}$, whereas the left hand side is the probability $P(\mathbf{x}' : (j+l-1) \to (j+l))$ that sequence $\mathbf{x}'$ survived from time step $j+l-1$ to time step $j+l$. We can rewrite (2.37) as follows

$$P\,(\mathbf{x}' : (j+l-1) \to (j+l)) \leq P_d. \tag{2.38}$$

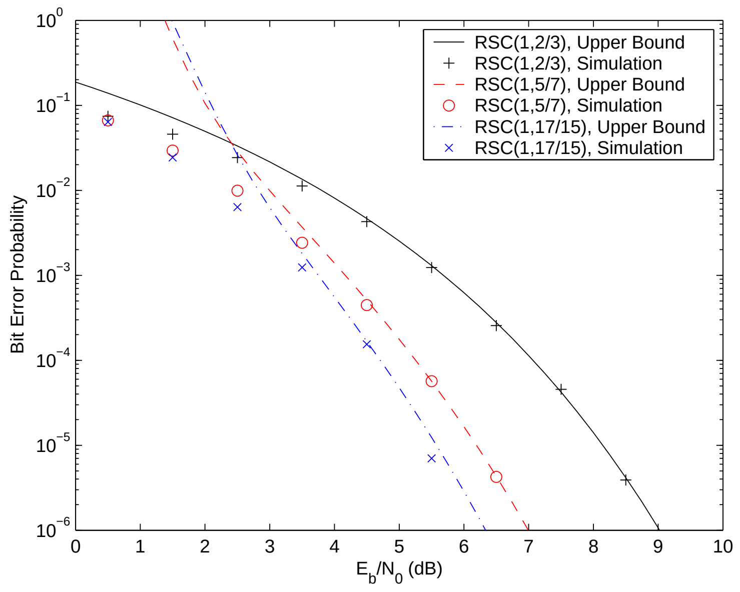


Figure 2.9: Comparison between simulation results and upper bounds to the bit error probability, for various RSC codes.

Hence, the probability of the incorrect sequence $\mathbf{x}'$ being selected after a previous decision error has occurred, is upper bounded by the pairwise error probability at that time step [20]. Consequently, the probability of the incorrect sequence $\mathbf{x}'$ being selected after $l$ decision errors have occurred, i.e., one decision error per time step, is upper bounded by the sum of the pairwise error probabilities for each time step, i.e.,

$$P(\mathbf{x}') = \sum_{i=0}^{l-1} P(\mathbf{x}' : (j+i) \to (j+l)) \leq \sum_{i=0}^{l-1} P_d = lP_d. \tag{2.39}$$

From (2.35), we find that the upper bound on the bit error probability for the special case of a code with a single codeword sequence $\mathbf{x}'$, other than the correct codeword sequence $\mathbf{x}$, only depends on the input weight $w$ and output weight $d$ of $\mathbf{x}'$

$$P_b \leq \frac{w}{l} lP_d = wP_d. \tag{2.40}$$

If the transfer function of a code consists of several codeword sequences having the same input and output weights, we can use the input-output weight enumerating function $T(W, D)$ to derive the exact number $T_{w,d}$ of all available codeword sequences. The bit error probability for this case is upper bounded by

$$P_b \leq wT_{w,d}P_d. \tag{2.41}$$

For the generic case where the transfer function of a convolutional code consists of codeword sequences of various input and output weights, the upper bound on the bit error probability can be found if we take the sum of $wT_{w,d}P_d$ over all possible values of $w$ and $d$, i.e.,

$$\begin{aligned} P_b &\leq \sum_{d=d_f}^{\infty} \sum_{w=1}^{\infty} wT_{w,d}P_d \\ &= \sum_{d=d_f}^{\infty} \sum_{w=1}^{\infty} wT_{w,d}Q\left(\sqrt{\frac{2E_bR}{N_0}d}\right). \end{aligned} \tag{2.42}$$

A sufficient approximation for the bit error probability is obtained even if the range of values for $w$ and $d$ is truncated and only the first terms of the input-output weight enumerating function $T(W, D)$ are taken into account [25].

As an example, a comparison between upper bounds and simulation results for RSC codes with memory size of 1, 2 and 3, is presented in Fig.2.9. In all three cases, only terms with an input and output weight of up to 40, i.e., $w \leq 40$ and $d \leq 40$, are considered. It can be observed that this truncation yields a sufficient and accurate approximation.

## 2.5 The Turbo Encoder

In 1993, Berrou, Glavieux and Thitimajshima first introduced turbo codes to the communication theory community as a new class of convolutional codes [11]. The schematic of a turbo encoder is shown in Fig.2.10. An information sequence of length $N$ is input to the first constituent systematic convolutional encoder of rate 1/2 and memory size $\nu_1$, as well as the interleaver $\Pi$ of size $N$. The interleaved information sequence is then input to the second convolutional encoder of rate 1 and memory size $\nu_2$. If the two constituent encoders are identical, the turbo encoder is called *symmetric*, otherwise it is known as *asymmetric*. The output of the rate 1/3 turbo encoder consists of the systematic sequence of the first encoder, the parity check sequence of the first encoder and the parity check sequence of the second encoder.

The *constituent convolutional encoders* could be either recursive or non-recursive. However, Benedetto and Montorsi showed in [16, 26] that the bit error rate performance of turbo codes using recursive encoders significantly improves as the size of the interleaver increases, an attribute known as *interleaving gain*. When non-recursive constituent encoders are used, no interleaving gain is achieved, thus it is necessary that turbo codes always use recursive encoders to attain exceptional performance.

The two constituent encoders are usually *terminated*, i.e., a number of tail bits are appended at the end of the information sequence that drive the encoders to the zero state. The length of the tail is equal to the memory size of the constituent

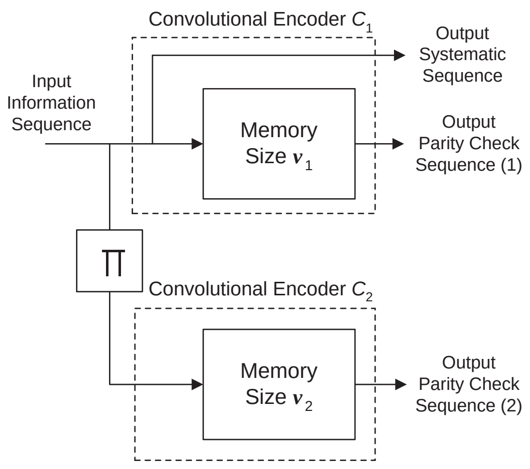

Convolutional Encoder $C_1$
Input Information Sequence
Memory Size $v_1$
Output Systematic Sequence
Output Parity Check Sequence (1)
Π
Convolutional Encoder $C_2$
Memory Size $v_2$
Output Parity Check Sequence (2)

Figure 2.10: Generic block diagram of a rate-1/3 turbo code.

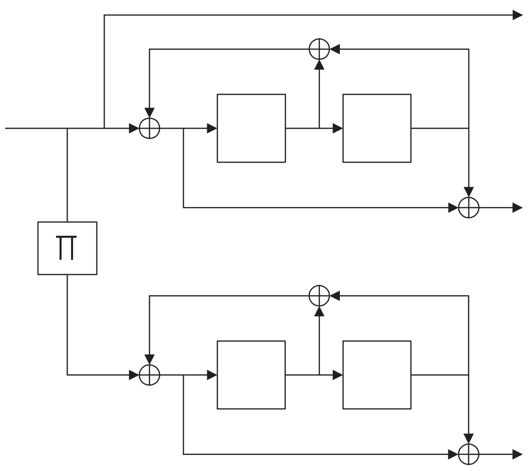


Figure 2.11: Schematic for the symmetric turbo code PCCC(1,5/7,5/7).

encoders. Termination facilitates the decoding procedure at the receiving side, since the final state is predetermined, and, effectively, transforms a linear convolutional code into a linear block code, since the input and output sequences have finite length. As a consequence, the turbo code is always seen as a linear block code.

The *interleaver* is usually assumed to be pseudo-random. The bits of the information sequence are stored sequentially and a pseudo-random generator determines the order they are read out. For short information sequences, a semi-random interleaver known as an *S-random* interleaver is found to perform better than most pseudo-random interleavers [27]. The purpose of the interleaver is to de-correlate the parity check output sequences of the turbo encoder but, more importantly, to increase the weight of the output codeword sequences. Its objective is to shuffle the stored information bits in such a way that most of the low-weight codeword sequences of the first constituent encoder are matched with high-weight codeword sequences of the second constituent encoder, and vice versa. When this is achieved, the turbo code exhibits codeword sequences with minimum weight much higher than the minimum weight codeword sequences generated by each individual constituent code.

A turbo encoder is generally described by the generator vectors of each constituent convolutional encoder. If $\mathbf{G}_{F1}$ and $\mathbf{G}_{R1}$ are the generator vectors for the feedforward and feedback connections of the first constituent encoder, and $\mathbf{G}_{F2}$ and $\mathbf{G}_{R2}$ are the generator vectors for the second constituent encoder, we use the notation PCCC(1,$\mathbf{G}_{F1}/\mathbf{G}_{R1}$,$\mathbf{G}_{F2}/\mathbf{G}_{R2}$) to describe the parallel concatenation of the two convolutional codes. As in the case of systematic convolutional codes, "1" corresponds to the systematic output of the turbo encoder. For example, both constituent encoders in Fig.2.11 have generator vectors $\mathbf{G}_{F1} = \mathbf{G}_{F2} = [101] = 5_8$ and $\mathbf{G}_{R1} = \mathbf{G}_{R2} = [111] = 7_8$, hence the symmetric turbo encoder is denoted as PCCC(1,5/7,5/7).

## 2.6 The Turbo Decoder

For linear constituent terminated convolutional codes with memories $\nu_1$ and $\nu_2$, the turbo encoder could also be seen as a linear terminated convolutional code [28], hence ML decoding and, more specifically, the Viterbi algorithm could be used to decode the received sequences. However, due to the presence of the interleaver, the number of memory states in the trellis could reach values on the order of $2^{\nu_1+\nu_2+N}$, where $N$ is the interleaver size. For a large interleaver size or, equivalently, a long input information sequence, the computational decoding complexity is intolerable.

In a similar manner to the turbo encoder, a turbo decoder uses two component decoders that work together to derive a good estimate of the transmitted information sequence, thus achieving a computational complexity on the order of $2^{\nu_1} + 2^{\nu_2}$, which is independent of the interleaver size. As Berrou noted [12], another way of thinking about the turbo decoding process is in terms of a crossword puzzle; you transmit not only the solution to the crossword, i.e., the systematic sequence, over a noisy channel, but also the clues for the horizontal and vertical words, i.e., the two parity check sequences. The first component decoder uses the constraints imposed by the horizontal words to solve the crossword and gives hints to the second decoder, which uses the constraints imposed by the vertical words. By exchanging hints in an iterative manner, the two decoders help each other find a solution to the crossword. The hints, as well as the received solution and the clues for the horizontal and vertical words, are all in the form of log-likelihood ratios, the native language of the component decoders. Before describing in more detail the operation of the soft-input soft-output (SiSo) decoders, we give a brief outline of the communication system.

A binary information sequence $\mathbf{u} = (u_1, u_2, \ldots, u_N)$ is input to a turbo encoder, employing an interleaver of size $N$. The output coded bits are mapped to polar symbols and the transmitted codeword sequence assumes the form $\mathbf{x} = (\mathbf{x}^{(u)}, \mathbf{x}^{(p1)}, \mathbf{x}^{(p2)})$, where sequences $\mathbf{x}^{(u)}$, $\mathbf{x}^{(p1)}$ and $\mathbf{x}^{(p2)}$ have length $N$ each and correspond to the systematic output sequence, the parity check output sequence from the first constituent encoder and the parity check output sequence from the second constituent encoder, respectively. For simplicity, we keep referring to the polar symbols as coded bits. A codeword $\mathbf{x}_t = (x_t^{(u)}, x_t^{(p1)}, x_t^{(p2)})$ generated by the input information bit $u_t$, is a snapshot of $\mathbf{x}$ at a time step $t \in [1 \dots N]$. The turbo decoder receives the codeword sequence $\mathbf{y} = (\mathbf{y}^{(u)}, \mathbf{y}^{(p1)}, \mathbf{y}^{(p2)})$, which is impaired by the channel noise, and distributes it to the component SiSo decoders.

The objective of the first component decoder is to produce an estimate of each information bit or, equivalently, an estimate of each systematic bit $x_t^{(u)}$ by processing sequences $\mathbf{y}^{(u)}$ and $\mathbf{y}^{(p1)}$, which correspond to the impaired output of the first encoder. The estimate, in the form of an LLR, is given by

$$\Lambda(x_t^{(u)} \mid \mathbf{y}^{(u)}, \mathbf{y}^{(p1)}) = \ln \frac{P(x_t^{(u)} = +1 \mid \mathbf{y}^{(u)}, \mathbf{y}^{(p1)})}{P(x_t^{(u)} = -1 \mid \mathbf{y}^{(u)}, \mathbf{y}^{(p1)})}. \tag{2.43}$$

If we assume independency between the received sequences $\mathbf{y}^{(u)}$ and $\mathbf{y}^{(p1)}$, and recall that the $t$-th decoded systematic bit $x_t^{(u)}$ only depends on the $t$-th received systematic bit $y_t^{(u)}$ in sequence $\mathbf{y}^{(u)}$, the above expression can be written as

$$\begin{aligned}\Lambda(x_t^{(u)} \mid \mathbf{y}^{(u)}, \mathbf{y}^{(p1)}) &= \ln \frac{P(x_t^{(u)} = +1 \mid y_t^{(u)}) P(x_t^{(u)} = +1 \mid \mathbf{y}^{(p1)})}{P(x_t^{(u)} = -1 \mid y_t^{(u)}) P(x_t^{(u)} = -1 \mid \mathbf{y}^{(p1)})} \\ &= \ln \frac{P(y_t^{(u)} \mid x_t^{(u)} = +1) P(x_t^{(u)} = +1 \mid \mathbf{y}^{(p1)}) P(x_t^{(u)} = +1)}{P(y_t^{(u)} \mid x_t^{(u)} = -1) P(x_t^{(u)} = -1 \mid \mathbf{y}^{(p1)}) P(x_t^{(u)} = -1)} \\ &= \Lambda_c y_t^{(u)} + \Lambda(x_t^{(u)} \mid \mathbf{y}^{(p1)}) + \Lambda(x_t^{(u)}).\end{aligned} \tag{2.44}$$

Therefore, the a-posteriori LLR of a decoded bit $x_t^{(u)}$ computed by the first SiSo decoder is the sum of the soft output of the channel $\Lambda_c y_t^{(u)}$ for the systematic bit $x_t^{(u)}$, indirect information about $x_t^{(u)}$ derived from the received parity check sequence $\mathbf{y}^{(p1)}$ [19], and a-priori information about $x_t^{(u)}$ obtained from the second encoder. For convenience, we use the compact notation $\Lambda_d^{(1)}(x_t^{(u)})$, $\Lambda_e^{(1)}(x_t^{(u)})$ and $\Lambda^{(1)}(x_t^{(u)})$ to refer to the LLRs $\Lambda(x_t^{(u)} \mid \mathbf{y}^{(u)}, \mathbf{y}^{(p1)})$, $\Lambda(x_t^{(u)} \mid \mathbf{y}^{(p1)})$ and $\Lambda(x_t^{(u)})$, respectively, associated with the first component decoder. Consequently, expression (2.44) assumes the form

$$\Lambda_d^{(1)}(x_t^{(u)}) = \Lambda_c y_t^{(u)} + \Lambda_e^{(1)}(x_t^{(u)}) + \Lambda^{(1)}(x_t^{(u)}). \tag{2.45}$$

The indirect information $\Lambda_e^{(1)}(x_t^{(u)})$ about $x_t^{(u)}$ is called *extrinsic information*, whereas the a-priori LLR $\Lambda^{(1)}(x_t^{(u)})$ is sometimes also referred to as *intrinsic information.* Initially, there is no a-priori knowledge, thus $\Lambda^{(1)}(x_t^{(u)}) = 0$, corresponding to a probability $P(x_t^{(u)} = +1) = P(x_t^{(u)} = -1) = 0.5$.

Whereas the a-posteriori LLR is the marriage of all available information about a decoded bit and provides an estimate of its value, the extrinsic LLR offers the "personal" opinion of the first component decoder about the bit in question, by

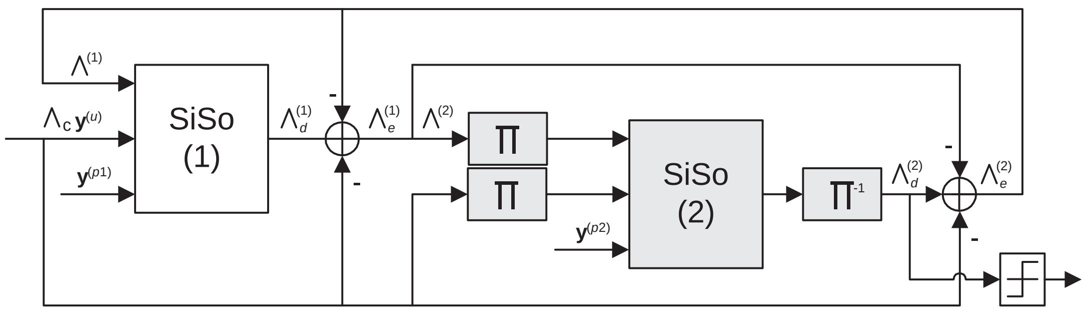


Figure 2.12: Iterative decoding of a turbo code.

examining only the received parity check sequence $\mathbf{y}^{(p1)}$, generated by the first constituent encoder. Subtracting both the received systematic sequence and the a-priori information from the output of the SiSo decoder, yields the extrinsic information, as shown in Fig.2.12.

The second component decoder takes into account the extrinsic information $\Lambda_e^{(1)}(x_t^{(u)})$ from the first decoder and uses it as a-priori information $\Lambda^{(2)}(x_t^{(u)})$, while processing the received interleaved systematic sequence together with the received parity check sequence $\mathbf{y}^{(p2)}$, generated by the second encoder, as depicted in Fig.2.12. Note that the two interleavers ($\Pi$), which convert the received systematic sequence and the a-priori information into the interleaved domain, are identical to the one used in the turbo encoder. Similarly, the output of the second SiSo decoder in the non-interleaved domain ($\Pi^{-1}$) is given by

$$\Lambda_d^{(2)}(x_t^{(u)}) = \Lambda_c y_t^{(u)} + \Lambda_e^{(2)}(x_t^{(u)}) + \Lambda^{(2)}(x_t^{(u)}), \tag{2.46}$$

where $\Lambda^{(2)}(x_t^{(u)}) = \Lambda_e^{(1)}(x_t^{(u)})$. In a similar manner, the extrinsic information from the second decoder is used as a-priori information by the first encoder, during the next iteration, i.e., $\Lambda^{(1)}(x_t^{(u)})=\Lambda_e^{(2)}(x_t^{(u)})$.

As the iterative process continues, the bit error rate of the decoded bits, measured at the output of the second SiSo decoder, is reduced. Depending on the application, the number of iterations varies between six to eight, since no significant improvement in performance is obtained with a higher number of iterations [29]. A hard decision on the decoded bits is made at the end of the last iteration.

## 2.6.1 The BCJR Algorithm

Although the Viterbi algorithm is optimal for ML decoding, it only provides a sequence of decoded bits but it does not assign reliability values, in the form of LLRs, to the decoded bits. This is essential for the constructive information exchange between the component SiSo decoders and the successful iterative decoding of turbo codes. Berrou *et al.* [11] have implemented a MAP decoding algorithm, based on the widely known *BCJR algorithm*, which was proposed by Bahl, Cocke, Jelinek and Raviv [7] in 1974. In this subsection we give a short description of the BCJR algorithm, as used by each individual component SiSo decoder of a turbo code.

We denote as $\tilde{\mathbf{y}}$ the received sequence of systematic and parity check soft values, processed by a SiSo decoder. In the case of the first SiSo decoder, $\tilde{\mathbf{y}}$ consists of the systematic sequence $\mathbf{y}^{(u)}$ and the parity check sequence $\mathbf{y}^{(p1)}$, whereas in the case of the second SiSo decoder, $\tilde{\mathbf{y}}$ is composed of the interleaved systematic sequence and the parity check sequence $\mathbf{y}^{(p2)}$. Similarly, we use the notation $\tilde{\mathbf{x}}$ to denote the input sequence to the communication channel, for which the soft output sequence $\tilde{\mathbf{y}}$ is produced.

The conditional probability $P(x_t^{(u)} = +1 \mid \tilde{\mathbf{y}})$ can be expanded, if we consider that an information bit $x_t^{(u)}$ causes a transition from state $s'$ to state $s$, at a time step $t$. Since the pair $(s',s)$ determines $x_t^{(u)}$, we obtain

$$\begin{aligned} P(x_t^{(u)}{=}{+1} \mid \tilde{\mathbf{y}}) &= \sum_{\substack{(s',s):\\ x_t^{(u)}=+1}} P\left((s',s) \mid \tilde{\mathbf{y}}\right) \\ &= \sum_{\substack{(s',s):\\ x_t^{(u)}=+1}} \frac{P(s',s,\tilde{\mathbf{y}})}{P(\tilde{\mathbf{y}})}. \end{aligned} \tag{2.47}$$

from the definition of conditional probability. The received sequence $\tilde{\mathbf{y}}$ can be split up into three sequences; the received codeword $\tilde{\mathbf{y}}_t$ associated with the state transition from $s'$ to $s$, the received sequence $\tilde{\mathbf{y}}_{j<t}$ from the beginning of the trellis up to time step $t-1$ and the received sequence $\tilde{\mathbf{y}}_{j>t}$ from time step $t+1$ up to the end of the trellis. Assuming a memoryless channel, the joint probability $P(s',s,\tilde{\mathbf{y}})$ can be also partitioned into the following independent probabilities

$$P(s',s,\tilde{\mathbf{y}}) = \underbrace{P(s',\tilde{\mathbf{y}}_{j<t})}_{\alpha_{t-1}(s')} \cdot \underbrace{P(\tilde{\mathbf{y}}_t \mid \tilde{\mathbf{x}}_t)P(x_t^{(u)})}_{\gamma_t(s',s)} \cdot \underbrace{P(\tilde{\mathbf{y}}_{j>t} \mid s)}_{\beta_t(s)}. \tag{2.48}$$

The values of $\alpha_{t-1}(s')$ and $\beta_t(s)$ can be derived by means of a forward and a backward recursion, respectively, based on

$$\begin{aligned} \alpha_t(s) &= \sum_{\forall s'} \alpha_{t-1}(s')\gamma_t(s',s) \\ \beta_{t-1}(s') &= \sum_{\forall s} \beta_t(s)\gamma_t(s',s), \end{aligned} \tag{2.49}$$

where $\alpha_0(0) = 1$ and $\beta_N(0) = 1$, since all incorrect codewords correspond to paths of length $N$ that start from and end in the zero state, due to the trellis termination. During the forward recursion, the decoder also calculates the branch metrics $\gamma_t(s', s)$, utilizing the a-priori probability $P(x_t^{(u)})$ received from the previous SiSo decoder and the probability density function $P(y_t^{(i)} \mid x_t^{(i)})$ of the memoryless channel, from which we can compute the conditional probability $P(\tilde{\mathbf{y}}_t \mid \tilde{\mathbf{x}}_t)$ as follows

$$P(\tilde{\mathbf{y}}_t \mid \tilde{\mathbf{x}}_t) = \prod_i P(y_t^{(i)} \mid x_t^{(i)}). \tag{2.50}$$

Note that index $i$ runs over the coded bits composing a codeword at time step $t$. For the case of an AWGN channel, $P(y_t^{(i)} \mid x_t^{(i)})$ is given by (2.8).

The conditional probability $P(x_t^{(u)} = -1 \mid \tilde{\mathbf{y}})$ can be also expanded, if we follow the same steps, hence the a-posteriori LLR at the output of a SiSo decoder assumes the form

$$\begin{aligned}\Lambda_d(x_t^{(u)}) = \Lambda(x_t^{(u)} \mid \tilde{\mathbf{y}}) &= \ln \frac{P(x_t^{(u)} = +1 \mid \tilde{\mathbf{y}})}{P(x_t^{(u)} = -1 \mid \tilde{\mathbf{y}})} \\ &= \ln \sum_{\substack{(s',s): \\ x_t^{(u)}=+1}} \alpha_{t-1}(s')\gamma_t(s', s)\beta_t(s) \\ &\quad - \ln \sum_{\substack{(s',s): \\ x_t^{(u)}=-1}} \alpha_{t-1}(s')\gamma_t(s', s)\beta_t(s).\end{aligned} \tag{2.51}$$

Summarizing, as the soft values from the output of the channel are received, a SiSo decoder performs a forward recursion to calculate the $\alpha_t(s)$ values, during which a-priori knowledge about the decoded bits is exploited to obtain the $\gamma_t(s', s)$ values. Once all the channel values are received, a backward recursion is used to calculate the $\beta_t(s)$ values and derive the reliability, $\Lambda_d(x_t^{(u)})$, of each decoded bit.

### 2.6.2 Decoding Algorithms used in Practice

The BCJR algorithm is considered extremely complex owing to the multiplications needed and the logarithmic operations required to compute the a-posteriori LLR for each decoded bit. Due to its high complexity, the BCJR algorithm was ignored for many years but interest was renewed when two modifications were proposed to reduce its complexity, while maintaining its performance.

The *Max-Log-MAP algorithm*, proposed by Koch and Baier [30] in 1990, simplifies the calculation of the a-posteriori LLRs by using the approximation

$$\ln\left(\exp(\chi_1) + \exp(\chi_2)\right) \approx \max(\chi_1, \chi_2). \tag{2.52}$$

Expression (2.51) can be written as

$$\begin{aligned}\Lambda_d(x_t^{(u)}) \approx & \max_{\substack{(s',s):\\ x_t^{(u)}=+1}} \left(A_{t-1}(s') + \mathit{\Gamma}_t(s',s) + B_t(s)\right) \\ & - \max_{\substack{(s',s):\\ x_t^{(u)}=-1}} \left(A_{t-1}(s') + \mathit{\Gamma}_t(s',s) + B_t(s)\right),\end{aligned} \tag{2.53}$$

with $A_t(s)$, $\mathit{\Gamma}_t(s', s)$ and $B_t(s)$ defined as follows

$$\begin{aligned} A_t(s) &\triangleq \ln(\alpha_t(s)), \\ \mathit{\Gamma}_t(s', s) &\triangleq \ln(\gamma_t(s', s)), \\ B_t(s) &\triangleq \ln(\beta_t(s)). \end{aligned} \tag{2.54}$$

Instead of recursively calculating $\alpha_t(s)$ and $\beta_t(s)$ and then applying the logarithmic operation to derive $A_t(s)$ and $B_t(s)$, respectively, we can use (2.49) and (2.52) to directly approximate their values, as follows

$$\begin{aligned} A_t(s) &\approx \max_{\forall s'} \left(A_{t-1}(s') + \mathit{\Gamma}_t(s', s)\right) \\ B_{t-1}(s') &\approx \max_{\forall s} \left(B_t(s) + \mathit{\Gamma}_t(s', s)\right). \end{aligned} \tag{2.55}$$

For the case of an AWGN channel, the expression for the branch metric, $\mathit{\Gamma}_t(s', s)$, assumes the form

$$\mathit{\Gamma}_t(s', s) = 0.5\Lambda_c \sum_i x_t^{(i)} y_t^{(i)} + 0.5 x_t^{(u)} \Lambda(x_t^{(u)}) \tag{2.56}$$

in a manner similar to (2.14).

All expressions have been considerably simplified, since multiplications have been transformed into additions in the log-domain, and the number of operations has been reduced. However, because of these approximations, only the ML path through an examined state is considered, rather than any path through the trellis to this state [31]. Therefore the performance is sub-optimal compared to that of the BCJR algorithm.

In 1995, Robertson *et al.* [31] corrected the approximation by using the Jacobian logarithm and presented the *Log-MAP algorithm*. Approximation (2.52) can be made exact as follows

$$\begin{aligned} \ln\left(\exp(\chi_1) + \exp(\chi_2)\right) &= \max(\chi_1, \chi_2) + \ln\left(1 + exp(-|\chi_1 - \chi_2|)\right) \\ &= \max{*}(\chi_1, \chi_2). \end{aligned} \tag{2.57}$$

The correction term $\ln\left(1 + exp(-|\chi_1 - \chi_2|)\right)$ takes a limited number of values and can be stored in a look-up table, reducing the complexity of the computation. Otherwise, if the correction term is computed exactly and no look-up table is used, the algorithm is known as *exact Log-MAP* and achieves the same performance as BCJR. Log-MAP is slightly more complex than Max-Log-MAP but achieves a performance almost identical to that of the optimal BCJR algorithm. The expressions for either Log-MAP or exact Log-MAP can be obtained from those of Max-Log-MAP, if the max(.) operation is replaced by the max $*$(.) operation.

A different approach to MAP decoding was proposed by Hagenauer and Hoher [32] in 1989. The so-called *soft-output Viterbi algorithm* (SOVA) proceeds by executing the conventional Viterbi algorithm, although the branch metric calculation also takes into account a-priori information $\Lambda(x_t^{(u)})$ about the decoded bit $x_t^{(u)}$, hence (2.19) becomes

$$\mathrm{BM}_t^{(s',s)} = 0.5\Lambda_c \sum_i x_t^{(i)} y_t^{(i)} + 0.5 x_t^{(u)} \Lambda(x_t^{(u)}), \tag{2.58}$$

which is identical to the branch metric expression used in the BCJR-based algorithms. At each time step $t$, there will be two paths that reach a state $s$; the survivor path, which goes through state $s'$ at time step $t-1$, and the competing path, which goes through another state $s''$ at time step $t-1$. Information concerning the competing path that was pruned at time step $t$ is maintained by storing the path metric difference between the two paths

$$\Delta(s,t) = \mathrm{PM}_t^{(s',s)} - \mathrm{PM}_t^{(s'',s)} \geq 0, \tag{2.59}$$

which corresponds to the LLR of the survivor path. When the end of the trellis is reached and the ML path is identified, SOVA traces it back and examines all competing paths from $t=N$ to $t=1$. If a competing path merges with the ML path at time step $t$ in state $s$, SOVA compares one by one the estimated decoded bits of the survivor path with the respective ones of the competing path, from time $t$ when the paths merge, back to $t-\delta$ when the paths diverge. When two bits are found different at time $\tau \in (t-\delta, t]$, the respective element $\Lambda_d(x_\tau^{(u)})$ of the sequence of LLR values is updated, according to the expression

$$\Lambda_d(x_\tau^{(u)}) \approx \min\left(\Lambda_d(x_\tau^{(u)}), \Delta(s,t)\right). \tag{2.60}$$

The LLR values $\Lambda_d(x_t^{(u)})$ with $t \in [1,\ldots,N]$, are initialized to a large number. As in the case of Max-Log-MAP, SOVA also compares two paths per time step. However, Max-Log-MAP looks at the best path with $x_t^{(u)} = +1$ and the best path with $x_t^{(u)} = -1$, at time $t$. One of them will always be the ML path [31]. SOVA correctly finds the ML path but not necessarily the other best path, since it may have been eliminated at an earlier time step. Concluding, Log-MAP and Max-Log-MAP are superior to SOVA in terms of bit error rate performance, however the number of operations for Max-Log-MAP is about twice the number of operations required by SOVA, assuming a memory size $\nu \leq 4$ [31].

## 2.7 Error Probability for Turbo Codes

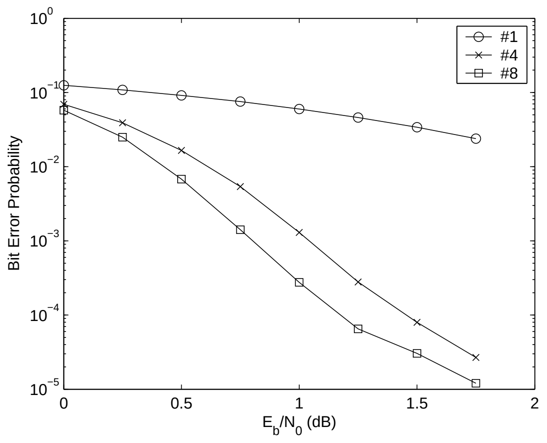
The bit error rate performance of a turbo code depends on various parameters, such as the number of iterations, the decoding algorithm, the interleaver size, the memory size and the generator vectors of the constituent codes. In Fig.2.13, we present simulation results for turbo codes using BPSK over AWGN channels.

Performance improves considerably as the number of iterations increases, however the additional coding gain achieved at the end of each consecutive iteration is constantly reduced [11, 33]. This trend is clearly illustrated in Fig.2.13(a). In Fig.2.13(b) we see that the exact log-MAP algorithm is optimal, at the expense of increased computational complexity. In agreement with [31], a trade-off between performance and complexity can be achieved if the iterative decoder uses a suboptimal algorithm, like log-MAP, max-log-MAP or SOVA. The interleaver size of a PCCC using recursive convolutional codes as constituent codes, also plays a significant role on the bit error rate performance, as depicted in Fig.2.13(c). The coding gain considerably increases with the size of the interleaver, at the cost of higher latency [16]. Finally, turbo codes employing constituent encoders with large memory size achieve better performance, but add to the computational burden of the iterative decoder. Nevertheless, we observe in Fig.2.13(d) that careful selection of the generator vectors of the constituent convolutional codes may result in turbo codes that achieve better performance than that of other turbo codes using constituent codes of the same memory size [26].

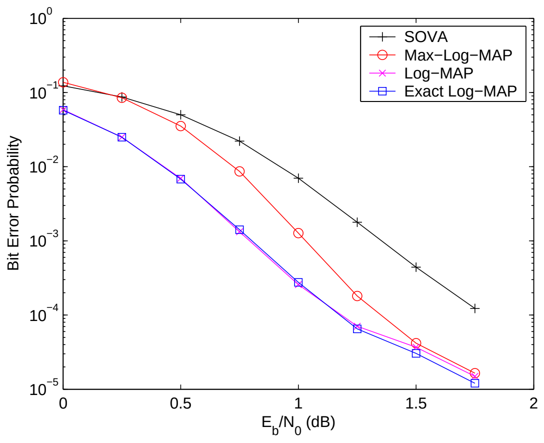
The design of a good code for a specific application is not always a quest for the identification of the best "ingredients" that contribute to an astonishing bit error rate performance. The candidate codes should comply with the requirements imposed by the application, such as hardware complexity and latency. Turbo codes are highly configurable due to the numerous parameters that determine their performance, hence they are attractive for use in a wide range of applications.

### 2.7.1 Performance Profile of Turbo Codes

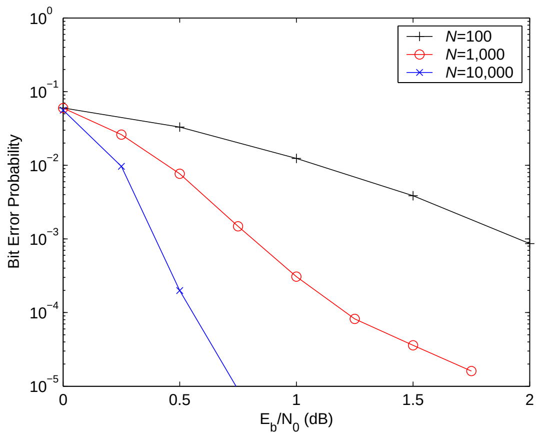
A generic representation of the bit error rate performance of a turbo code is shown in Fig.2.14. Iterative decoding is used and eight or more iterations are assumed.



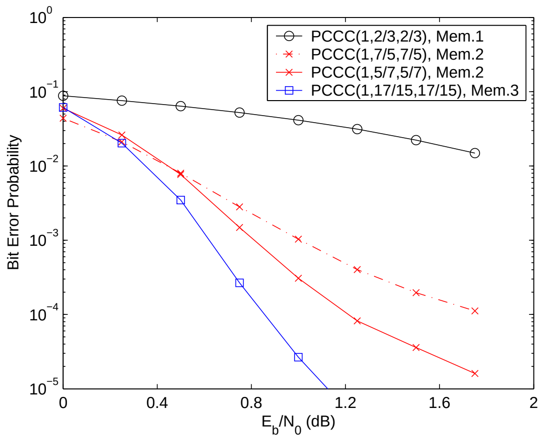
(a) Performance of PCCC(1,5/7,5/7) with $N = 1,000$, after 1,4 and 8 iterations, using the exact Log-MAP algorithm.



(b) Performance of PCCC(1,5/7,5/7) with $N = 1,000$, after 8 iterations, employing different decoding algorithms.



(c) Performance of PCCC(1,5/7,5/7) after 8 iterations, using the exact Log-MAP algorithm, for various interleaver sizes.



(d) Performance of various PCCC configurations with $N = 1,000$, after 8 iterations. The exact Log-MAP is used.

Figure 2.13: Effect of various parameters (number of iterations, decoding algorithm, interleaver size, memory size and constituent codes) on the bit error rate performance of a turbo code.

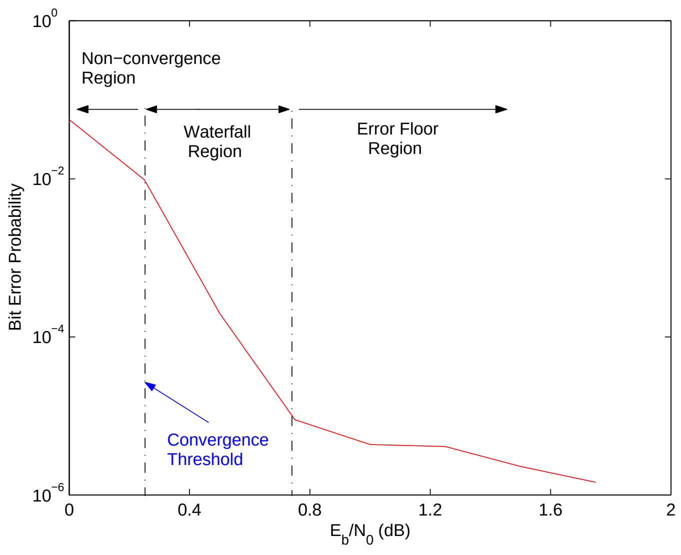


Figure 2.14: Regions that characterize the bit error rate performance of a turbo code.

For low $E_b/N_0$ values, the performance curve goes through the *non-convergence region*, where the bit error probability decreases slowly until a specific value of $E_b/N_0$, known as the *convergence threshold*, is reached. At the convergence threshold, which is determined by the constituent codes, the decoding algorithm and the interleaver size, the slope of the curve changes abruptly. The curve enters the *waterfall* or *turbo-cliff region*, where an increase in the transmit energy causes significant performance improvement. The performance of the turbo code in the waterfall region is dominated by the interleaver size. Eventually, the curve "hits" the error floor region, where the expenditure of additional energy does not achieve a marked improvement in the error rate performance. The error floor of a turbo code depends on the distance properties of the constituent codes, when large interleavers are used. For small interleaver sizes, the type of the interleaver also plays a dominant role [28].

Depending on the system requirements, two techniques are widely used for the performance evaluation of a turbo code or the design of turbo codes under specific constraints. The first approach, proposed by ten Brink [34], uses *extrinsic information transfer* (EXIT) chart analysis, which can accurately predict the convergence behavior of the iterative decoder for very large interleaver sizes (e.g., $N = 10^6$ bits) by performing separate simulations of the component decoders. It takes into account the structure of the constituent codes and depicts the effect of each iteration on the overall error rate performance.

The second approach, presented by Benedetto and Montorsi [16,26], relies on ML soft decoding. A tight upper bound on the bit error probability is derived, which provides an accurate estimate of the suboptimal iterative decoder performance at high $E_b/N_0$ values and successfully predicts the error floor of a turbo code. Since an optimal ML decoder is assumed, the performance bound is a function of only the interleaver size and the characteristics, i.e., memory size and distance properties, of each constituent encoder. Nevertheless, the performance of the actual suboptimal iterative decoder, employing the exact log-MAP algorithm, quickly converges to the upper bound, for an increasing number of iterations.

Concluding, the first technique is adopted when designing turbo codes that exhibit low convergence thresholds, whereas the second method is used for the design of turbo codes that exhibit low error floors. Initially we concentrate on the second approach, since rate-1/3 turbo codes exhibit quick convergence on AWGN channels, but we will combine both approaches when we consider punctured turbo codes of rates higher than 1/3.

### 2.7.2 Union Bound on the Bit Error Rate Performance

We have mentioned that trellis termination transforms the constituent convolutional codes into linear *convolutional block codes*. The trellis of a convolutional block code, whose input is a block of $N$ bits, is the truncation at step $N$ of the infinite length trellis of the original convolutional code. Furthermore, the transfer function $T^{\mathcal{C}}(W, U, Z, L)$ of a convolutional code $\mathcal{C}$ provides all single-error events, i.e., all paths that start from the zero state, diverge from the all-zero path at step one and at some point remerge with the zero state and remain at it. The transfer function $B^{\mathcal{C}}(W, U, Z)$ of the corresponding convolutional block code provides all single and multiple-error events, i.e., all paths of length $N$ that start from the zero state, can remerge with and diverge from the zero state more than once and terminate at the zero state.

Maximum-likelihood soft decoding of a convolutional block code, which has a finite trellis length, follows the same principles as ML soft decoding of the original convolutional code, which has an infinite trellis length. Thus, the pairwise error probability $P_d$ is also given by (2.25), whereas the first-event error probability, which is the sum of all pairwise probabilities, is renamed to *frame error probability* or *word error probability*, since a decision error that occurs while decoding a sequence of finite length is equivalent to a frame error. The frame error probability on an

AWGN channel is upper bounded by

$$\begin{aligned} P_w &\leq \sum_d B_d^{\mathcal{C}} P_d \\ &= \sum_d B_d^{\mathcal{C}} Q\left(\sqrt{\frac{2E_b R_{\mathcal{C}}}{N_0} d}\right), \end{aligned} \tag{2.61}$$

where $E_b$ is the energy per transmitted coded bit, $R_{\mathcal{C}}$ is the code rate and $B_d^{\mathcal{C}}$ are the coefficients of the weight enumerating function $B^{\mathcal{C}}(D) = B^{\mathcal{C}}(W=1, U=D, Z=D)$. The output weight $d$ takes values in the range from 1 to $2N$ since we have assumed a rate-1/2 convolutional block code, thus an input sequence of $N$ bits generates an output sequence of length $2N$.

The probability of a bit error is more straightforward to derive when codeword sequences of finite length are considered [35]. We assume that the all-zero sequence was transmitted and the decoder made an erroneous decision. A specific incorrect codeword sequence $\mathbf{x}'$ was selected with probability $P(\mathbf{x}') = P_d$, described by a monomial $W^w U^u Z^z$ having an input weight of $w$ and an overall output weight of $d = u + z$. The probability of a bit error, given that $\mathbf{x}'$ was selected, is

$$P\left(\text{bit error} \mid \mathbf{x}'\right) = \frac{w}{N}, \tag{2.62}$$

since $N$ is the length of the decoded sequence. Therefore, the probability of a bit error $P_b$ is given by

$$\begin{aligned} P_b &= P\left(\text{bit error} \mid \mathbf{x}'\right) P(\mathbf{x}') \\ &= \frac{w}{N} P_d \\ &= \frac{w}{N} Q\left(\sqrt{\frac{2E_b R_{\mathcal{C}}}{N_0} d}\right). \end{aligned} \tag{2.63}$$

However, we need to take into account the set of all possible incorrect codeword sequences that could be selected in the case of an erroneous decision. Using the union bound argument, in a similar manner to (2.42), we obtain

$$P_b \leq \frac{1}{N} \sum_d \sum_w w B_{w,d}^{\mathcal{C}} Q\left(\sqrt{\frac{2E_b R_{\mathcal{C}}}{N_0} d}\right), \tag{2.64}$$

where $B_{w,d}^{\mathcal{C}}$ are the coefficients of the input-output weight enumerating function $B^{\mathcal{C}}(W, D) = B^{\mathcal{C}}(W, U = D, Z = D)$. A rate-1/3 turbo code $\mathcal{P}$ using an interleaver of size $N$ is also a convolutional block code, hence the bit error probability is also bounded by [35, 36]

$$P_b \leq \frac{1}{N} \sum_d \sum_w w B_{w,d}^{\mathcal{P}} Q\left(\sqrt{\frac{2E_b R_{\mathcal{P}}}{N_0} d}\right). \tag{2.65}$$

Similarly, $B^{\mathcal{P}}_{w,d}$ are the coefficients of the weight enumerating function of the turbo code, $B^{\mathcal{P}}(W,D)=B^{\mathcal{P}}(W,U=D,Z=D)$. For $R_{\mathcal{P}}=1/3$, the output weight $d$ takes values in the range from 1 to $3N$.

If $\mathcal{C}_1$ and $\mathcal{C}_2$ are the constituent codes of the turbo code $\mathcal{P}$, we need to obtain the transfer function $B^{\mathcal{P}}(W,U,Z)$ of the turbo code in order to upper bound the bit error probability, that is assuming knowledge of the transfer functions $B^{\mathcal{C}_1}(W,U,Z)$ and $B^{\mathcal{C}_2}(W,U,Z)$ of the constituent codes. However, the transfer function of the turbo code also depends on the permutation scheme employed by the interleaver, since the weight of the parity check output sequence of the second constituent encoder depends on the sequence of input bits that have been previously permuted by the interleaver. The transfer function of the turbo code could, in principle, be derived by means of an exhaustive enumeration of all possible permutation cases, but the computational complexity would be overwhelming.

In 1996, Benedetto and Montorsi [16] introduced the concept of the *uniform interleaver*, an abstract probabilistic interleaver of size $N$ that maps an input sequence of weight $w$ to all possible permutations, with equal probability. The number of permutations is given by

$$\binom{N}{w} = \frac{N!}{(N-w)!w!}\,, \tag{2.66}$$

thus the probability of a specific permutation being selected is $1/\binom{N}{w}$. In order to exploit the properties of the uniform interleaver, we need to decompose the transfer function $B^{\mathcal{P}}(W,U,Z)$ of the turbo code into a sum of conditional weight enumerating functions $B^{\mathcal{P}}(w,U,Z)$ that correspond to codeword sequences of specific weight $w$, i.e.,

$$B^{\mathcal{P}}(W,U,Z) = \sum_{w} B^{\mathcal{P}}(w,U,Z)W^{w}, \tag{2.67}$$

where

$$B^{\mathcal{P}}(w,U,Z) = \sum_{u}\sum_{z} B^{\mathcal{P}}_{w,u,z}U^{u}Z^{z}. \tag{2.68}$$

Owing to the uniformly random permutations, the conditional weight enumerating function of the turbo code can be obtained from [16]

$$B^{\mathcal{P}}(w,U,Z) = \frac{B^{\mathcal{C}_1}(w,U,Z)\cdot B^{\mathcal{C}_2}(w,U=1,Z)}{\binom{N}{w}}. \tag{2.69}$$

The product $B^{\mathcal{C}_1}(w,U,Z)\cdot B^{\mathcal{C}_2}(w,U=1,Z)$ gives a polynomial, which is essentially the sum of all turbo codeword sequences, generated by an input sequence of information weight $w$, for all possible interleaver permutations. This sum is then normalized, using the exact number of possible permutations $\binom{N}{w}$, where $N$ is the size of the interleaver. Note that the systematic output sequence of the second constituent encoder is not transmitted, so it is eliminated by setting $U{=}1$ in $B^{\mathcal{C}_2}(w, U, Z)$.

The assumption of a uniform interleaver leads to the calculation of the transfer function of a turbo code, as the average of all transfer functions that correspond to turbo codes employing the same constituent codes and using all possible interleavers of a given size [16, 28]. Consequently, the upper bound to the bit error probability of a turbo code using a uniform interleaver coincides with the average of the upper bounds obtainable with the set of all turbo codes employing every possible interleaving scheme [16].

Nevertheless, derivation of the transfer function of a turbo code, requires knowledge of the conditional weight enumerating functions of the constituent convolutional block codes, which in turn, are derived from their transfer functions, according to the expression

$$B^{\mathcal{C}}(W, U, Z) = \sum_{w} B^{\mathcal{C}}(w, U, Z) W^{w}. \tag{2.70}$$

where $\mathcal{C}$ refers to either $\mathcal{C}_1$ or $\mathcal{C}_2$. The conventional approaches as well as a novel technique for the computation of the transfer function $B^{\mathcal{C}}(W, U, Z)$ of a convolutional block code will be presented in detail in Chapter 3.

## 2.8 Chapter Summary

The objective of this chapter was to present an overview of convolutional coding and introduce the concept of turbo coding, which is the main theme of this thesis.

We presented the structure of convolutional encoders as well as their representation, using trellis and state diagrams. In particular, we gave examples of recursive systematic convolutional codes, which are the building blocks of turbo codes. Furthermore, we introduced the concept of the log-likelihood ratio, we presented the theory behind ML decoding and we described the Viterbi algorithm, which yields ML decisions. Next, the derivation of the transfer function of a convolutional code was described, from which the weight enumerating functions can be obtained and, consequently, upper bounds to the bit error performance can be computed.

The second part of the chapter focused on turbo codes. The turbo encoder was presented and the significance of its components, namely the constituent codes and the interleaver, was discussed. A schematic of the turbo decoder was given and the operation of the component soft-input soft-output decoders was explained. The "ingredients" of the iterative process, namely the channel reliability as well as the a-priori, extrinsic and a-posteriori information, were introduced. Next, we emphasized the necessity of the decoding algorithm to produce a-posteriori soft values rather than hard estimates of the decoded bits, which dictates either the adoption of a MAP decoding algorithm, such as the BCJR algorithm, or the modification of the Viterbi algorithm, which gives rise to the SOVA algorithm. Although BCJR is more efficient than SOVA, it suffers from a high computational complexity. Algorithms such as the exact log-MAP, log-MAP and max-log-MAP, which perform MAP decoding in the log-domain and thus achieve a complexity lower than that of BCJR, were also described.

We demonstrated that the bit error rate performance of a turbo code depends on the interleaver size, the type and memory size of the constituent encoders. In addition, performance also depends on the configuration of the iterative decoder; the number of iterations and the decoding algorithm determine the slope of the curve, which represents the error rate performance. The significance of the relationship between the transfer function of a turbo code and the transfer function of its constituent convolutional block codes was stressed, since it allows the derivation of a union bound on the error rate performance, if ML soft decoding is assumed. The bound gives an accurate estimate of the error floor of a turbo code, hence it is a useful tool for designing codes required to satisfy particular specifications.

It is imperative to evaluate the transfer function of each constituent convolutional block code of a turbo code, in order to compute the union bound on its error rate performance. A novel technique for the evaluation of the transfer function of a convolutional block code, as well as conventional methods, will be presented and compared in the following chapter.

Chapter 3

# Performance Analysis of Non-Punctured Convolutional and Turbo Codes

## 3.1 Introduction

A tight upper bound on the bit error probability of a turbo code can be readily computed, if the distance properties of the code, conveyed by its transfer function, are known. Calculation of the transfer function of a turbo code is computationally intensive for deterministic interleavers but the assumption of a uniform interleaver drastically simplifies the calculations and reduces the computational burden. Nevertheless, calculation of the transfer function of a turbo code requires knowledge of the transfer functions of its constituent convolutional block codes. Consequently, it is imperative to devise an efficient method to compute the transfer function of a convolutional block code, accurately and rapidly.

In this chapter we introduce a new technique, based on the concept of the "augmented" state diagram, for the evaluation of the transfer function of a convolutional block code, and also compare it with existing approaches. We then use our technique to derive theoretical upper bounds on the average ML performance of turbo codes and we compare them with results obtained through the simulation of turbo codes using pseudo-random interleavers. We conclude by emphasizing that our novel method can be extended in a straightforward manner to punctured turbo codes, which is a subject studied and detailed in the following chapter.

## 3.2 The Augmented State Diagram

In this section we present a novel approach, according to which the modified state diagram of a convolutional code is “augmented”, enabling derivation of the transfer function of the respective convolutional block code.

We demonstrated in Chapter 2 that the transfer function $T(W, U, Z, L)$ of a convolutional code can be obtained by splitting the zero state of the state diagram into two separate states, $X_S$ and $X_E$, properly labeling the branches using monomials so as to reflect the input, systematic and parity check weight of the associated codeword, and solving the state equations for the ratio $X_E/X_S$. Assuming that the all-zero sequence is transmitted, the transfer function essentially provides those codeword sequences that correspond to single-error events, represented by paths in the trellis diagram that start from the zero state, diverge from it and later re-merge with it only once.

The transfer function $B(W, U, Z)$ of the respective convolutional block code, provides all codeword sequences, related to both single-error and multiple-error events. Multiple-error events can be seen as the concatenation of single-error events, hence they correspond to paths that re-merge with the zero state more than once, and can stay at it for a consecutive number of time steps. In order to consider paths that re-visit the zero state and remain at it for an indefinite number of time steps, we need to modify the state diagram of the convolutional code, accordingly.

We consider, without loss of generality, the case of the binary RSC(1,5/7) code to demonstrate the method for obtaining the augmented state diagram of a convolutional block code from the state diagram of the original convolutional code. In order to allow a path to revisit the zero state, we insert a node $X_0$, which is different to the states $X_S$ and $X_E$, as illustrated in Fig.3.1. States which are interconnected to $X_S$ or $X_E$, are also interconnected to this ‘intermediate” zero state $X_0$ in a similar manner. The self-loop is appended to $X_0$, since a path can remain at $X_0$ for an indefinite period of time. Furthermore, two branches, both with zero input and output weight, are added to interconnect $X_S$ with $X_0$ and $X_0$ with $X_E$, so as to permit paths to diverge from the all-zero sequence at a time step other than the very first, or re-merge with the all-zero sequence at a time step other than the very last. The resultant augmented state diagram of the RSC(1,5/7) block code, depicted in

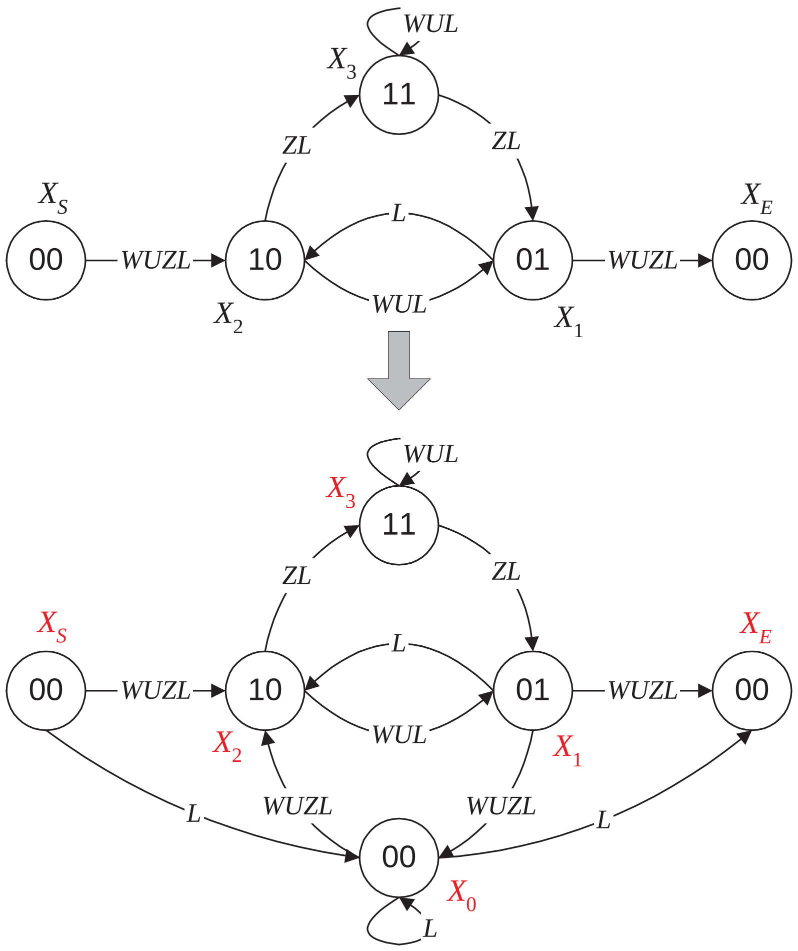


Figure 3.1: Derivation of the augmented state diagram of the RSC(1,5/7) block code from the modified state diagram of the RSC(1,5/7) code.

Fig.3.1, is used to derive the system of state equations

$$
\begin{aligned}
X_0 &= L \cdot X_S + L \cdot X_0 + WUZL \cdot X_1 \\
X_1 &= WUL \cdot X_2 + ZL \cdot X_3 \\
X_2 &= WUZL \cdot X_S + WUZL \cdot X_0 + L \cdot X_1 \\
X_3 &= ZL \cdot X_2 + WUL \cdot X_3 \\
X_E &= L \cdot X_0 + WUZL \cdot X_1.
\end{aligned} \tag{3.1}
$$

Upon solving the state equations for the ratio $X_E/X_S$, we obtain

$$
\frac{X_E}{X_S} = \frac{L^2}{1-L} + \frac{y'(W,U,Z,L)}{1-x'(W,U,Z,L)} \cdot \frac{L^2}{1-L} \tag{3.2}
$$

where

$$\begin{aligned} y'(W,U,Z,L) &= W^3U^3Z^2L + W^2U^2Z^4L^2 - W^4U^4Z^2L^2, \\ x'(W,U,Z,L) &= L + WUL + Z^2L^3 - WUL^3 - W^2U^2L^3 + W^3U^3Z^2L^3 \\ &\quad - Z^2L^4 + W^2U^2L^4 + W^2U^2Z^4L^4 - W^4U^4Z^2L^4, \end{aligned} \tag{3.3}$$

for the case of the RSC(1,5/7) block code.

The first term on the right hand side of (3.2) is a function of $L$ only and can be expanded, exploiting the property of the binomial series, i.e.,

$$\frac{L^2}{1-L} = L^2 \cdot \sum_{k=0}^{\infty} L^k = L^2 + L^3 + L^4 + \ldots, \tag{3.4}$$

which gives a sum of all-zero codeword sequences for various path lengths. These sequences correspond to paths that start from $X_S$, stay at $X_0$ for an indefinite number of steps by circulating around the self-loop, and finally terminate at $X_E$. The all-zero codeword sequences are not of interest in the computation of the transfer function, since only erroneous codeword sequences are considered, hence they are ignored.

The second term on the right hand side of (3.2) represents the set of all single-error and multiple-error events that correspond to paths of various lengths. This term, which we call the *extended transfer function* of a convolutional block code and denote as $f(W,U,Z,L)$, may be expressed as

$$\begin{aligned} f(W,U,Z,L) &= \frac{y'(W,U,Z,L)}{1-x'(W,U,Z,L)} \cdot \frac{L^2}{1-L} \\ &= \frac{y(W,U,Z,L)}{1-x(W,U,Z,L)} \end{aligned} \tag{3.5}$$

where

$$\begin{aligned} y(W,U,Z,L) &= L^2 \cdot y'(W,U,Z,L), \\ x(W,U,Z,L) &= x'(W,U,Z,L) - L \cdot x'(W,U,Z,L) + L. \end{aligned} \tag{3.6}$$

Using the property of binomial series, the extended transfer function can be written as

$$f(W,U,Z,L) = y(W,U,Z,L) \cdot \sum_{k=0}^{\infty} x^k(W,U,Z,L). \tag{3.7}$$

In our example, polynomials $y(W, U, Z, L)$ and $x(W, U, Z, L)$ assume the form

$$\begin{aligned} y(W, U, Z, L) &= W^3U^3Z^2L^3 + W^2U^2Z^4L^4 - W^4U^4Z^2L^4, \\ x(W, U, Z, L) &= 2L + WUL - L^2 - WUL^2 \\ &\quad + Z^2L^3 - WUL^3 - W^2U^2L^3 + W^3U^3Z^2L^3 \\ &\quad - 2Z^2L^4 + WUL^4 + 2W^2U^2L^4 + W^2U^2Z^4L^4 \\ &\quad - W^3U^3Z^2L^4 - W^4U^4Z^2L^4 \\ &\quad + Z^2L^5 - W^2U^2L^5 - W^2U^2Z^4L^5 + W^4U^4Z^2L^5. \end{aligned} \tag{3.8}$$

Substituting (3.8) into (3.7), we obtain the extended transfer function $f(W, U, Z, L)$ of the RSC(1,5/7) block code

$$\begin{aligned} f(W, U, Z, L) &= W^3U^3Z^2L^3 \\ &\quad + W^2U^2Z^4L^4 + 2W^3U^3Z^2L^4 \\ &\quad + 2W^2U^2Z^4L^5 + 3W^3U^3Z^2L^5 + W^3U^3Z^4L^5 + W^4U^4Z^2L^5 \\ &\quad + \dots \end{aligned} \tag{3.9}$$

If we define as the *conditional extended transfer function*, $f(W, U, Z, l)$, the sum of all codeword sequences of a specific path length $l$, i.e.,

$$f(W, U, Z, l) = \sum_w \sum_u \sum_z B_{w,u,z,l} W^w U^u Z^z. \tag{3.10}$$

we observe that, in general, the extended transfer function $f(W, U, Z, L)$ is given by

$$f(W, U, Z, L) = \sum_{l=1}^{\infty} f(W, U, Z, l) L^l. \tag{3.11}$$

We can identify in (3.9) the conditional extended transfer functions for path lengths, or equivalently input block lengths, of $l = 3$, 4 and 5. For example, the conditional extended transfer function $f(W, U, Z, l = 4)$ of RSC(1,5/7) is $W^2U^2Z^4 + 2W^3U^3Z^2$.

Consequently, for an input information sequence of length $N$, the transfer function $B(W, U, Z)$ of the convolutional block code is, by definition, equivalent to the conditional extended transfer function $f(W, U, Z, l = N)$, i.e,

$$B(W, U, Z) \triangleq f(W, U, Z, l = N). \tag{3.12}$$

Concluding, the proposed augmented state diagram can be used to derive the extended transfer function $f(W, U, Z, L)$ of a convolutional block code, which leads to the wanted transfer function $B(W, U, Z)$ for a specific input block length $N$, if we isolate the conditional extended transfer function $f(W, U, Z, l)$ for $l = N$.

### 3.2.1 Imposing Computational Constraints

After the evaluation of the polynomials $y(W,U,Z,L)$ and $x(W,U,Z,L)$, we can use (3.7) to derive the extended transfer function $f(W,U,Z,L)$. However, for an input sequence of length $N$, evaluation of the conditional transfer function $f(W,U,Z,l=N)$ is sufficient, which is equivalent to computing only those terms of $f(W,U,Z,L)$ that contain $L^N$ rather than computing all terms that contain $L^l$, up to an arbitrary high $l$, and then isolating the terms with exponent $l=N$. In order to avoid calculating unnecessary terms, and consequently achieve a reduction of the computational complexity for obtaining $f(W,U,Z,l=N)$, we can identify a set of exponents $k$ for $x(W,U,Z,L)$ in (3.7), which result in codeword sequences of path length $N$.

Similarly to the definition of $f(W,U,Z,L)$ as a function of $f(W,U,Z,l)$, polynomials $y(W,U,Z,L)$ and $x(W,U,Z,L)$ can also be expressed as

$$\begin{aligned} y(W,U,Z,L) &= \sum_l y(W,U,Z,l)L^l, \\ x(W,U,Z,L) &= \sum_l x(W,U,Z,l)L^l, \end{aligned} \tag{3.13}$$

where $y(W,U,Z,l)$ and $x(W,U,Z,l)$ are defined as

$$\begin{aligned} y(W,U,Z,l) &= \sum_w \sum_u \sum_z B^{(x)}_{w,u,z,l} W^w U^u Z^z, \\ x(W,U,Z,l) &= \sum_w \sum_u \sum_z B^{(y)}_{w,u,z,l} W^w U^u Z^z. \end{aligned} \tag{3.14}$$

Note that $B^{(x)}_{w,u,z,l}$ and $B^{(y)}_{w,u,z,l}$ can assume a zero, positive or negative integer value. For the sake of clarity, we drop the indeterminate variables $W$, $U$ and $Z$ from (3.13), since only the exponents of $L$ are of significance for our analysis. Therefore, (3.13) can be written as

$$\begin{aligned} y(L) &= \sum_{l=l_{y_1}}^{l_{y_Q}} y(l)L^l, \\ x(L) &= \sum_{l=l_{x_1}}^{l_{x_P}} x(l)L^l, \end{aligned} \tag{3.15}$$

where $l$ assumes values from the set $\{l_{y_1},\ldots,l_{y_Q}\}$ in the case of $y(L)$, and $\{l_{x_1},\ldots,l_{x_P}\}$ in the case of $x(L)$.

In order to obtain $x^k(L)$, as required by (3.7), we invoke the multinomial theorem as follows

$$\begin{aligned}\left(\sum_{l=l_{x_1}}^{l_{x_P}} x(l)L^l\right)^k &= \left[x(l_{x_1})L^{l_{x_1}} + x(l_{x_2})L^{l_{x_2}} + \ldots + x(l_{x_P})L^{l_{x_P}}\right]^k \\ &= \sum_{\substack{k_1,\ldots,k_P \\ k_1+\ldots+k_P=k}} \left[\frac{k!}{k_1!\ldots k_P!}\left(x(l_{x_1})L^{l_{x_1}}\right)^{k_1}\cdot\ldots\cdot\left(x(l_{x_P})L^{l_{x_P}}\right)^{k_P}\right]\end{aligned} \tag{3.16}$$

where the summation is taken over all combinations of the nonnegative indices $k_1$ through $k_P$ such that $k_1+k_2+\ldots+k_P=k$. Consequently, expression (3.7) assumes the form

$$f(L) = y(L)\cdot\sum_{k=0}^{\infty} x^k(L), \tag{3.17}$$

where

$$y(L) = y(l_{y_1})L^{l_{y_1}} + y(l_{y_2})L^{l_{y_2}} + \ldots + y(l_{y_Q})L^{l_{y_Q}}, \tag{3.18}$$

and

$$x^k(L) = \sum_{\substack{k_1,\ldots,k_P \\ k_1+\ldots+k_P=k}} \left[\frac{k!}{k_1!\ldots k_P!}\left(x^{k_1}(l_{x_1})\cdots x^{k_P}(l_{x_P})\right)L^{(l_{x_1}k_1+\ldots+l_{x_P}k_P)}\right]. \tag{3.19}$$

For an input block sequence of length $N$, we can compute only those terms of $f(L)$ that contain $L^N$, and thus directly obtain the conditional extended transfer function $f(l=N)$. The required terms can be determined iteratively based on (3.17)-(3.19), as follows:

1. We focus on those terms of $y(L)$ that contain $L^{l_{y_i}}$, where $l_{y_i}$ is selected from the set $\{l_{y_1}, l_{y_2}, \ldots, l_{y_Q}\}$, with $l_{y_1} < l_{y_2} < \ldots < l_{y_Q}$. If $l_{y_i} > N$, all required terms have been derived, hence the procedure terminates. Initially, $i=1$.

2. Since we are interested in those terms of $f(L)$ containing $L^N$ and we initially consider only those terms of $y(L)$ containing $L^{l_{y_i}}$, it is clear that we need to determine the terms in $x^k(L)$ that contain $L^{(N-l_{y_i})}$. These terms can be found by setting the exponent of $L$ in (3.19) equal to $N-l_{y_i}$ and solving the equation

$$l_{x_1}k_1 + l_{x_2}k_2 + \ldots + l_{x_P}k_P = (N - l_{y_i}) \tag{3.20}$$

for $k_1, k_2, \ldots, k_P$. Such indeterminate polynomial equations, which allow only integer solutions, are known as *Diophantine equations* [37]. To solve this type of equation, we simply try every combination of possible values for each one of the unknowns. In our case, due to the restriction of the multinomial theorem

(3.16), it is $k_j \geq 0$, with $j = 1, \ldots, P$, whereas $l_{x_j} \geq 0$ always, since paths having a negative length do not exist. Furthermore, since both $l_{x_j}$ and $k_j$ are nonnegative integers, each product $l_{x_j} k_j$ is upper bounded by $N - l_{y_i}$, i.e., $l_{x_j} k_j \leq (N - l_{y_i})$, according to (3.20). Consequently, the range of possible integer values for each unknown $k_j$ is

$$0 \leq k_j \leq \left\lfloor \frac{N - l_{y_i}}{l_{x_j}} \right\rfloor , \tag{3.21}$$

where $\lfloor (N - l_{y_i})/l_{x_j} \rfloor$ denotes the integer part of $(N - l_{y_i})/l_{x_j}$.

3. For a set $\mathcal{K} = \{k_1, k_2, \ldots, k_P\}$ of nonnegative solutions, we add the elements $k_1, k_2, \ldots, k_P$ to obtain the wanted exponent of $x(L)$, i.e.,

$$k = k_1 + k_2 + \ldots + k_P, \text{ for a set } \mathcal{K} = \{k_1, k_2, \ldots, k_P\} \tag{3.22}$$

4. We substitute the set of solutions $\mathcal{K}$ as well as the corresponding sum $k$ in (3.19) and we compute the product between $y(l_{y_i})L^{l_{y_i}}$ and the terms of $x^k(L)$, for the given value of $l_{y_i}$.

5. We repeat steps 3 and 4 for every possible set of solutions $\mathcal{K}$ obtained from equation (3.20).

6. We set $i \leftarrow i + 1$ and we repeat the procedure, provided that $i \leq Q$, where $Q$ is the total number of values for $l_{y_i}$.

When the end of the procedure is reached, all terms that correspond to codeword sequences having information, systematic and parity check weight less than or equal to $N$, as well as a path length equal to $N$, will have been identified. These terms compose the transfer function $B(W, U, Z)$ of the convolutional block code. Recalling expression (2.64), terms of low output weight mainly determine the upper bound on the bit error rate performance of the convolutional block code. If we are interested in these "most significant" terms, we can introduce additional constraints to the afore-mentioned procedure, so as to confine the search to terms having weights less than or equal to a pre-defined value, smaller than $N$.

## 3.3 Alternative Techniques to Derive the Transfer Function

Two alternative approaches for the evaluation of the transfer function of a convolutional block code are briefly described in the following subsections. The first approach, presented in 1995 by Divsalar *et al.* [27], builds upon the method described in [38]. The second technique, proposed by Benedetto and Montorsi [16] in 1996, is based on an algorithm developed by the same authors for the performance evaluation of trellis-coded modulation schemes [39].

### 3.3.1 Divsalar's Technique

The branches of the original state diagram of a convolutional code with memory size $\nu$ are properly labeled using the $W$, $U$, $Z$ and $L$ indeterminate variables. The updated state diagram for the case of the RSC(1,5/7) code can be seen at the top right of Fig.2.7 in Chapter 2. The information conveyed by the state diagram can be summarized by the $2^\nu \times 2^\nu$ *state transition matrix* $\mathbf{A}(W,U,Z,L) = \left[a_{ij}^{(1)}(W,U,Z,L)\right]$, where $a_{ij}^{(1)}(W,U,Z,L)$ is the monomial associated with the transition from state $i$ to state $j$ and corresponds to paths of length $l=1$. The state transition matrix for the RSC(1,5/7) code assumes the form

$$\mathbf{A}(W,U,Z,L) = \begin{bmatrix} L & 0 & WUZL & 0 \\ WUZL & 0 & L & 0 \\ 0 & WUL & 0 & ZL \\ 0 & ZL & 0 & WUL \end{bmatrix}. \tag{3.23}$$

Raising the state transition matrix $\mathbf{A}(W,U,Z,L)$ to an arbitrary power $N$ leads to a matrix $\mathbf{A}^N(W,U,Z,L) = \left[a_{ij}^{(N)}(W,U,Z,L)\right]$, where $a_{ij}^{(N)}(W,U,Z,L)$ is the polynomial associated with all paths of length $l=N$, originating from state $i$ and terminating at state $j$. However the transfer function $B(W,U,Z)$ of a convolutional block encoder that accepts input sequences of length $N$, only comprises of codeword sequences associated with paths of length $N$ that originate from the zero state and terminate at the zero state. Such paths are given by the (0,0) element of the matrix $\mathbf{A}^N(W,U,Z,L)$, denoted as $a_{00}^{(N)}(W,U,Z,L)$. All terms of $a_{00}^{(N)}(W,U,Z,L)$ contain $L^N$ owing to the fixed path length $N$. Hence, the indeterminate variable $L$ can be eliminated by setting $L=1$. The outcome contains all codeword sequences of path length $N$, including the all-zero sequence, which can be discarded by subtracting 1. Consequently, the transfer function $B(W,U,Z)$ of a convolutional block encoder for input sequences of length $N$ is given by

$$B(W,U,Z) = \left( a_{00}^{(N)}(W,U,Z,L)\Big|_{L=1} \right) - 1. \tag{3.24}$$

For large values of $N$, computation of $\mathbf{A}^N(W,U,Z,L)$ becomes increasingly complex. Instead of calculating $\mathbf{A}^N(W,U,Z,L)$, we compute the sum of all $\mathbf{A}^k(W,U,Z,L)$ associated with all possible block sizes, i.e., $k=0,1,\ldots,\infty$, invoking the equality

$$\mathbf{I} + \mathbf{A}(W,U,Z,L) + \mathbf{A}^2(W,U,Z,L) + \ldots = (\mathbf{I} - \mathbf{A}(W,U,Z,L))^{-1} \tag{3.25}$$

where $\mathbf{I}$ is the unity matrix and $(\mathbf{I}-\mathbf{A}(W,U,Z,L))^{-1}=[g_{ij}(W,U,Z,L)]$. Based on (3.25), we find that the (0,0) element, $g_{00}(W,U,Z,L)$, is

$$\begin{aligned} g_{00}(W,U,Z,L) &= 1+a_{00}^{(1)}(W,U,Z,L)+a_{00}^{(2)}(W,U,Z,L)+\dots \\ &= 1+\sum_{k=1}^{\infty} a_{00}^{(k)}(W,U,Z,L), \end{aligned} \tag{3.26}$$

which refers to all paths, starting from the zero state and terminating at the zero state, including the path of the all-zero codeword sequence, for all possible input block lengths.

Polynomial $g_{00}(W,U,Z,L)$ is equivalent to the ratio $X_E/X_S$ obtained after implementing our technique, based on use of the augmented state diagram. The transfer function $B(W,U,Z)$ of the convolutional block code for a specific input length $N$ can be derived from $g_{00}(W,U,Z,L)$ using the approach presented in Section 3.2.1, or by means of a recursion, as described in [27].

### 3.3.2 Benedetto and Montorsi's Technique

Benedetto and Montorsi proposed a different approach [16, 39] to derive a modified version of the transfer function of a convolutional code block code, defined as

$$T(W,U,Z,L,\Omega)=\sum_{w}\sum_{u}\sum_{z}\sum_{l}\sum_{n} T_{w,u,z,l,n}W^{w}U^{u}Z^{z}L^{l}\Omega^{n}. \tag{3.27}$$

Exponent $n$ of the new indeterminate variable $\Omega$ denotes the number of remergings with the zero state of a path corresponding to an incorrect codeword sequence. The transfer function $T(W,U,Z,L,\Omega)$ of a convolutional block code, that accepts an input block of length $N$, enumerates all paths having length equal to or less than $N$, that leave the zero state at step one, re-visit it one or more times but never remain at it, and eventually terminate at the zero state.

A codeword sequence, whose trellis path re-merges $n$ times with the path of the transmitted all-zero codeword sequence, can be seen as the succession of $n$ error events. However, $T(W,U,Z,L,\Omega)$ does not account for those codeword sequences with zeroes before, after or between the error events. It was shown in [16] that the total number $K[l,n]$ of codeword sequences having path length $N$, with zeroes before, after or between the $n$ error events, can be obtained from a single codeword sequence of path length $l\leq N$ associated with $n$ successive errors, as follows

$$K[l,n]=\binom{N-l+n}{n}=\frac{(N-l+n)!}{(N-l)!n!}. \tag{3.28}$$

Consequently, if there are $T_{w,u,z,l,n}$ codeword sequences that correspond to $n$ successive error events of length $l$, the total number of codeword sequences that correspond to $n$ error events of length $N$, with zeroes before, after or between the errors, is given by the product $K[l,n] \cdot T_{w,u,z,l,n}$. The coefficients $B_{w,u,z}$ and, ultimately, the transfer function $B(W,U,Z)$ of the convolutional block code, can be obtained from

$$B_{w,u,z} = \sum_{l} \sum_{n} K[l,n] T_{w,u,z,l,n}, \tag{3.29}$$

where the sum is taken over all valid values of $l$ and $n$.

### 3.3.3 Comparison of the Transfer Function Methods

In this section we demonstrate that our approach has similar complexity to Divsalar's method but it can be modified so as to achieve reduced complexity, similar to Benedetto and Montorsi's method. Nevertheless, the augmented state diagram which is the cornerstone of our technique, offers the benefit of simplicity, since it is a straightforward extension of the ordinary state diagram of a convolutional code. Furthermore, as it will become evident in the following chapter, the underlying advantage of our approach over the two alternative techniques is that it can be extended to punctured convolutional block codes, thus offering the means to evaluate the performance of PCCCs having code rates higher than 1/3.

We first compare our proposed technique to Divsalar's approach, which requires construction of the state diagram of the convolutional code, derivation of the state transition matrix $\mathbf{A}(W,U,Z,L)$ and computation of $g_{00}(W,U,Z,L)$, which is the (0,0) element of the $2^{\nu} \times 2^{\nu}$ parametric matrix $(\mathbf{I} - \mathbf{A}(W,U,Z,L))^{-1}$, where $\nu$ is the memory size of the code. A single element of an inverse matrix can be obtained using an inversion method, such as Gaussian elimination. The computational complexity of this approach is comparable to the complexity of our proposed technique, where we solve a system of $2^{\nu}+1$ equations only for the ratio $X_E/X_S$, which corresponds to $g_{00}(W,U,Z,L)$.

Benedetto and Montorsi's method consists of two stages. At the end of the first stage an intermediate transfer function $T(W,U,Z,L,\Omega)$ is algorithmically evaluated, whereas at the end of the second stage the transfer function $B(W,U,Z)$ of the convolutional block code is calculated using combinatorial logic. Although it is not obvious, direct calculation of $B(W,U,Z)$ using our approach is computationally intensive for medium interleaver sizes ($N < 1000$) and becomes intolerable for large interleaver sizes. Partitioning the procedure into two stages, as in Benedetto and

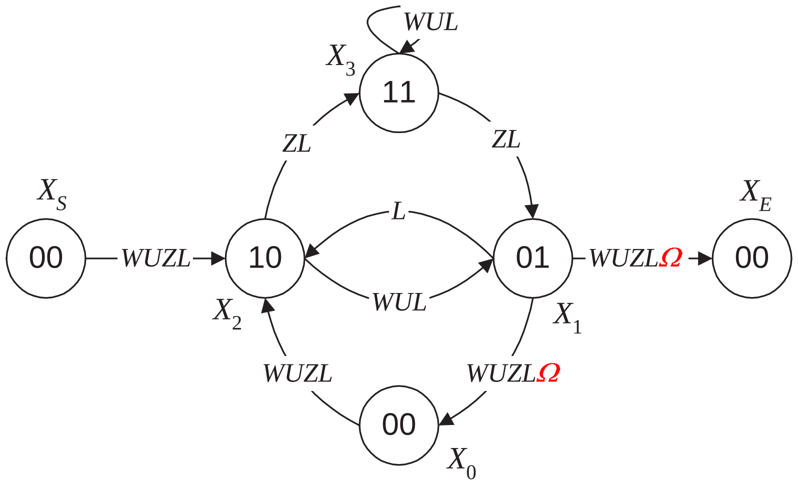


Figure 3.2: Augmented state diagram of the RSC(1,5/7) block code, where the number of remergings into the zero state are considered.

Montorsi's method, increases the memory requirements of the computational algorithm, owing to the introduction of the indeterminate variable $\Omega$, but dramatically reduces complexity.

Fortunately, the augmented state diagram can be easily modified to give the less computationally intensive intermediate transfer function $T(W, U, Z, L, \Omega)$, as is illustrated in Fig.3.2. The branch that connects $X_S$ with $X_0$ has been removed, since paths must leave the zero state at step one. The self-loop at $X_0$, as well as the branch that connects $X_0$ with $X_E$, have also been removed in order to force paths that re-visit the zero state to leave it at the following time step. Finally, the labels of those branches that lead to a remerging into the zero state, either $X_0$ or $X_E$, have been updated to include variable $\Omega$, as well. Upon solving the state equations for the ratio $X_E/X_S$ we obtain

$$\frac{X_E}{X_S} = T(W, U, Z, L, \Omega). \tag{3.30}$$

The steps for obtaining the required terms described in Section 3.2.1 still apply, with the exception of expression 3.20 which assumes the form

$$l_{x_1}k_1 + l_{x_2}k_2 + \ldots + l_{x_P}k_P \leq (N - l_{y_i}) \tag{3.31}$$

since we are interested in all paths of length $N$ or less. These paths are padded to a length of $N$ and compose the transfer function $B(W, U, Z)$ of the convolutional block code, using (3.29) as in the case of Benedetto and Montorsi's technique.

Summarizing, the technique we propose has similar complexity to Divsalar's method. Benedetto and Montorsi's approach is considerably less complex and can be efficiently used for turbo codes employing large interleaver sizes. Nevertheless, our proposed technique can be easily modified to offer similar functionality and computational complexity as Benedetto and Montorsi's approach. The main advantage of the augmented state diagram technique is that it can be easily extended to punctured convolutional block codes. Subsequently, we can accurately evaluate the performance of punctured turbo codes and identify puncturing patterns that result in good high rate turbo codes, in terms of error rate performance. A detailed description of our extended technique and its applications, is given in the following chapter.

## 3.4 Evaluation of the Performance Upper Bound

In order to calculate an upper bound on the ML average performance of a turbo code $\mathcal{P}$ using a uniform interleaver of size $N$ and two constituent RSC block codes, $\mathcal{C}_1$ and $\mathcal{C}_2$, we developed a configurable algorithm, which constructs the augmented state diagrams, obtains the state equations and eventually derives the transfer functions, namely $B^{\mathcal{C}_1}(W, U, Z)$ and $B^{\mathcal{C}_2}(W, U, Z)$, of the constituent convolutional block codes. The transfer function $B^{\mathcal{P}}(W, U, Z)$ of the turbo code can then be computed using the transfer functions of its constituent codes, as described in Chapter 2. The union bound requires knowledge of the input-output weight enumerating function $B^{\mathcal{P}}(W, D)$ of the turbo code, which can be obtained from $B^{\mathcal{P}}(W, U, Z)$ by setting $U = Z = D$. Based on (2.65), the union bound is an upper bound on the bit error probability $P_b$, and can be re-written as

$$\begin{aligned} P_b &\leq \frac{1}{N} \sum_{d=1}^{3N} \sum_{w=1}^{N} w B_{w,d}^{\mathcal{P}} Q\left(\sqrt{\frac{2E_b R_{\mathcal{P}}}{N_0} d}\right) \\ &= \sum_{d=1}^{3N} D_d Q\left(\sqrt{\frac{2E_b R_{\mathcal{P}}}{N_0} d}\right), \end{aligned} \tag{3.32}$$

with

$$D_d = \sum_{w=1}^{N} \frac{w}{N} B_{w,d}^{\mathcal{P}}, \tag{3.33}$$

where $E_b$ is the energy per transmitted coded bit, $R_{\mathcal{P}}$ is the code rate, which in our case is 1/3, and $B_{w,d}^{\mathcal{P}}$ are the coefficients of the weight enumerating function $B^{\mathcal{P}}(W, D)$. Note that $B_{w,d}^{\mathcal{P}}$ denotes the number, also known as *multiplicity*, of the output turbo codeword sequences having overall weight $d$, generated by input sequences of information weight $w$.

As suggested in both [27] and [16], computational complexity of the upper bound can be considerably reduced without compromising its accuracy, if only the most significant coefficients $B^{\mathcal{P}}_{w,d}$ are calculated, i.e., those having low input and output weights, $w$ and $d$, respectively. Consequently, the range of values for $w$ and $d$ can be truncated, such that $w \leq w_T$ and $d \leq d_T$, where $w_T$ and $d_T$ are called *truncation weights* and their typical values fall in the range between 25 and 38 [16]. Having obtained coefficients $D_d$ using (3.33) for $1 < d \leq d_T$, we estimate the remaining coefficients for $d_T < d \leq 3N$ by means of an extrapolation, based on the exponential law

$$D_d = \exp(\kappa_1 + \kappa_2 d). \tag{3.34}$$

Parameters $\kappa_1$ and $\kappa_2$ are derived through a mean-square optimization over the known values of $D_d$ [16].

A comparison between theoretical upper bounds, obtained for various PCCC configurations, and simulation results are presented in Fig. 3.3 and Fig. 3.4. We note that, in all cases, suboptimal iterative decoding employing the exact Log-MAP algorithm, converges to ML decoding for medium to large $E_b/N_0$ values, after only 8 iterations. For an increasing number of iterations, the iterative decoder converges to the ML performance at low values of $E_b/N_0$ [16]. Small deviations from the upper bound curve indicate that the simulated performance achieved by a PCCC configuration using a selected pseudo-random interleaver is slightly better or worse than the average performance of a PCCC using a uniform interleaver. The bounds derived using our novel approach were also compared to those computed using either Divsalar's technique [27] or Benedetto and Montorsi's method [16]. In all cases, the bounds were identical.

In Fig.3.3 the effect of the interleaver gain is illustrated. The slope of the upper bound, which identifies the error floor region of PCCC(1,5/7,5/7), can be lowered by increasing the size of the interleaver. In Table 3.1 the first 20 coefficients $D_d$, which were derived based on our proposed technique and used in the union bound calculation of the PCCC in question, are presented. Note that the free distance $d_{\text{free}}$ of the turbo code, which is the the smallest value of $d$ for which $D_d$ is non-zero, does not depend on the interleaver size. The main effect of the interleaver is to markedly reduce the value of each coefficient $D_d$. As David Forney stressed in his 1995 Shannon Lecture [40], "rather than attacking error exponents, they [turbo codes] attack multiplicities, turning conventional wisdom on its head".

In Fig.3.4 we compare the bit error rate performance of turbo codes using constituent convolutional codes of memory size 1, 2 and 3, separated by an interleaver of size 1,000 bits. Constituent codes described by generator vectors 2/3 and 17/15

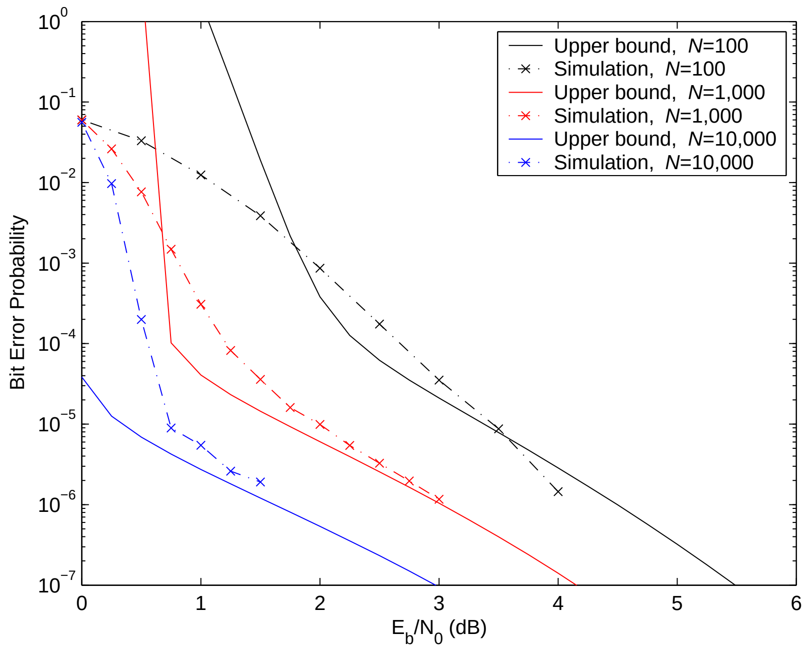


Figure 3.3: Comparison between bounds and simulation results after 8 iterations for PCCC(1,5/7,5/7) using interleaver sizes of 100, 1000 and 10000 bits. The exact log-MAP decoding algorithm is used.

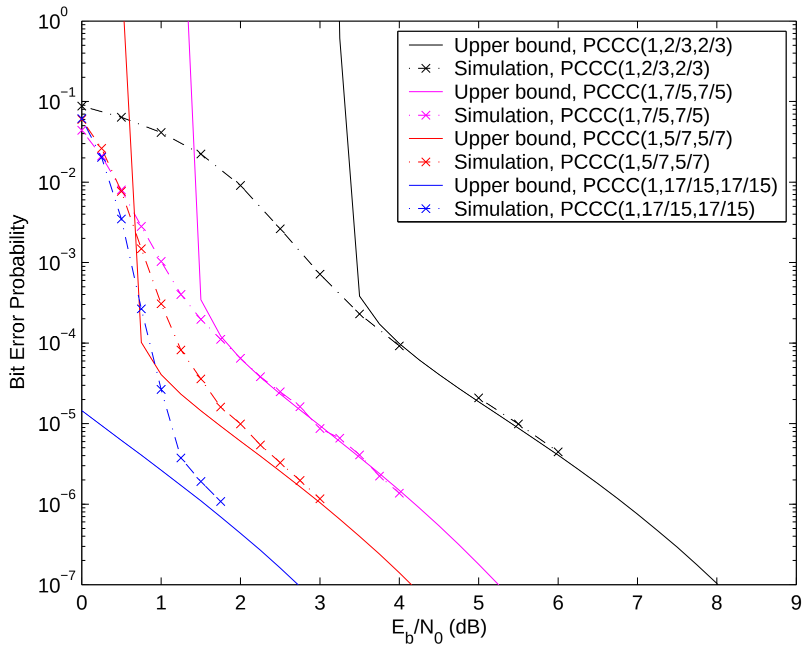


Figure 3.4: Upper bounds and simulation results after 8 iterations for PCCCs using various constituent codes. The interleaver size is 1000 bits. As in the previous figure, the exact log-MAP algorithm is applied.

Table 3.1: Coefficients $D_d$ for the calculation of the union bound for the rate-1/3 PCCC(1,5/7,5/7). Three interleaver sizes are considered.

| $d$ | $N=100$ | $N=1,000$ | $N=10,000$ |
|---|---|---|---|
| 1-6 | 0 | 0 | 0 |
| 7 | $1.782 \cdot 10^{-3}$ | $1.798 \cdot 10^{-5}$ | $1.800 \cdot 10^{-7}$ |
| 8 | $9.401 \cdot 10^{-5}$ | $9.581 \cdot 10^{-8}$ | $9.597 \cdot 10^{-11}$ |
| 9 | $1.041 \cdot 10^{-2}$ | $1.076 \cdot 10^{-4}$ | $1.080 \cdot 10^{-6}$ |
| 10 | $3.912 \cdot 10^{-2}$ | $3.981 \cdot 10^{-3}$ | $3.998 \cdot 10^{-4}$ |
| 11 | $3.209 \cdot 10^{-2}$ | $3.398 \cdot 10^{-4}$ | $3.418 \cdot 10^{-4}$ |
| 12 | $7.963 \cdot 10^{-2}$ | $7.942 \cdot 10^{-3}$ | $7.994 \cdot 10^{-4}$ |
| 13 | $7.387 \cdot 10^{-2}$ | $7.859 \cdot 10^{-4}$ | $7.914 \cdot 10^{-4}$ |
| 14 | 0.1382 | $1.199 \cdot 10^{-2}$ | $1.200 \cdot 10^{-3}$ |
| 15 | 0.1547 | $1.528 \cdot 10^{-3}$ | $1.530 \cdot 10^{-5}$ |
| 16 | 0.2845 | $1.661 \cdot 10^{-2}$ | $1.606 \cdot 10^{-3}$ |
| 17 | 0.3894 | $3.283 \cdot 10^{-3}$ | $3.233 \cdot 10^{-5}$ |
| 18 | 0.7064 | $2.278 \cdot 10^{-2}$ | $2.026 \cdot 10^{-3}$ |
| 19 | 1.1972 | $9.201 \cdot 10^{-3}$ | $8.982 \cdot 10^{-5}$ |
| 20 | 2.1205 | $5.562 \cdot 10^{-2}$ | $4.868 \cdot 10^{-3}$ |

have memory sizes of 1 and 3, respectively, while both generator vectors 5/7 and 7/5 correspond to a memory size of 2. As in the case of convolutional codes, constituent codes with an increasing memory order lead to turbo codes with better performance in terms of bit error probability. However, contrary to convolutional codes, maximization of the free distance of a turbo code by properly selecting the generator vectors, does not necessarily lead to minimization of the bit error probability. In Table 3.2, we note that $d_{\text{free}}=8$ for PCCC(1,7/5,7/5) whereas $d_{\text{free}}=7$ for PCCC(1,5/7,5/7), nevertheless the latter code performs better than the first, as is shown in Fig.3.4.

Benedetto and Montorsi showed in [26] that the most significant parameter through which a constituent code influences the performance of the PCCC, is the minimum weight of the parity check sequences, namely $z_{\min}$, generated by information sequences of weight 2. The minimum weights $z_{\min}^{\mathcal{C}_1}$ and $z_{\min}^{\mathcal{C}_2}$ of the constituent codes $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively, compose the *free effective distance* $d_{\text{free,eff}}$ of the PCCC given by

$$d_{\text{free,eff}} = 2 + z_{\min}^{\mathcal{C}_1} + z_{\min}^{\mathcal{C}_2}. \tag{3.35}$$

The effective free distance for PCCC(1,7/5,7/5) is found to be 8 as opposed to 10 for

Table 3.2: Coefficients $D_d$ for the calculation of the union bound of four rate-1/3 PCCCs. The size of the interleaver is $N = 1,000$.

| $d$ | (1,2/3,2/3) | (1,7/5,7/5) | (1,5/7,5/7) | (1,17/15,17/15) |
|---|---|---|---|---|
| 1-3 | 0 | 0 | 0 | 0 |
| 4 | $3.996 \cdot 10^{-3}$ | 0 | 0 | 0 |
| 5 | $7.984 \cdot 10^{-3}$ | 0 | 0 | 0 |
| 6 | $1.196 \cdot 10^{-2}$ | 0 | 0 | 0 |
| 7 | $1.594 \cdot 10^{-2}$ | 0 | $1.798 \cdot 10^{-5}$ | 0 |
| 8 | $4.380 \cdot 10^{-2}$ | $3.988 \cdot 10^{-3}$ | $9.581 \cdot 10^{-8}$ | $9.581 \cdot 10^{-8}$ |
| 9 | 0.1193 | $7.961 \cdot 10^{-3}$ | $1.076 \cdot 10^{-4}$ | 0 |
| 10 | 0.2659 | $1.192 \cdot 10^{-2}$ | $3.981 \cdot 10^{-3}$ | $5.735 \cdot 10^{-7}$ |
| 11 | 0.5070 | $1.586 \cdot 10^{-2}$ | $3.398 \cdot 10^{-4}$ | $1.610 \cdot 10^{-4}$ |
| 12 | 0.9846 | $1.989 \cdot 10^{-2}$ | $7.942 \cdot 10^{-3}$ | $3.526 \cdot 10^{-6}$ |
| 13 | 2.0762 | $2.419 \cdot 10^{-2}$ | $7.859 \cdot 10^{-4}$ | $2.139 \cdot 10^{-4}$ |
| 14 | 4.5098 | $2.905 \cdot 10^{-2}$ | $1.199 \cdot 10^{-2}$ | $3.960 \cdot 10^{-3}$ |
| 15 | 9.4755 | $3.485 \cdot 10^{-2}$ | $1.528 \cdot 10^{-3}$ | $7.107 \cdot 10^{-4}$ |
| 16 | 19.2880 | $6.577 \cdot 10^{-2}$ | $1.661 \cdot 10^{-2}$ | $3.888 \cdot 10^{-5}$ |
| 17 | 39.1300 | 0.1457 | $3.283 \cdot 10^{-3}$ | $9.573 \cdot 10^{-4}$ |
| 18 | 80.3950 | 0.2984 | $2.278 \cdot 10^{-2}$ | $8.026 \cdot 10^{-3}$ |
| 19 | 166.1300 | 0.5473 | $9.201 \cdot 10^{-3}$ | $1.977 \cdot 10^{-3}$ |
| 20 | 341.4900 | 0.9172 | $5.562 \cdot 10^{-2}$ | $4.914 \cdot 10^{-4}$ |

PCCC(1,5/7,5/7), which gives us an indication that the latter code will outperform the first. Therefore, the generator vectors of the constituent codes should be chosen to maximize the free effective distance of the turbo code.

Concluding, we demonstrated that our proposed technique can be used to accurately derive upper bounds on the average ML performance of turbo codes. We compared theoretical bounds to simulation results in order to illustrate that an upper bound correctly predicts the error floor of a suboptimal iterative decoder. For an increasing number of iterations, the performance of a turbo code converges to the ML bound for medium to high $E_b/N_0$ values, provided that the component decoders employ optimal MAP decoding. The parameters that affect the performance ML bound on the bit error probability are the interleaver size, the memory order and the generator vectors of the constituent codes.

## 3.5 Chapter Summary

In this chapter we introduced a novel approach for the evaluation of the transfer function of a convolutional block code. A tight upper bound on the average ML performance of a turbo code, which uses convolutional block codes as constituent codes, can be then computed and accurately predict the error floor of the suboptimal iterative decoder.

We compared our approach with two conventional techniques and we demonstrated that our proposed method exhibits identical complexity to that of Benedetto and Montorsi's technique, while it surpasses Divsalar's approach. Nevertheless, as we describe in detail in the following chapter, our method offers the advantage of being extendable to punctured turbo codes in a straightforward manner. Hence, it will enable us to both derive upper bounds on their error rate performance and identify puncturing patterns that can lower their error floor.

Chapter 4

# Punctured Convolutional and Turbo Codes on AWGN Channels

## 4.1 Introduction

The augmented state diagram can be used to obtain the transfer function of convolutional block codes, which is required for the computation of a tight upper bound on the average ML performance of a rate-1/3 turbo code that uses convolutional block codes as constituent codes. Higher rate turbo codes can be obtained by periodically deleting selected bits from the output of the turbo encoder. As a result, the trellis diagram of each constituent code also varies periodically with time.

In this chapter we extend the concept of the augmented state diagram, introduced in Chapter 3, to high-rate convolutional block codes with a periodically time-varying state diagram, known as *punctured convolutional block codes*. A punctured turbo code can be seen as the parallel concatenation of two punctured convolutional block codes separated by an interleaver. Hence, we can apply the same principles used previously in non-punctured turbo codes to derive the transfer function of the corresponding punctured turbo code from the transfer functions of its constituent codes. We demonstrate that our approach can be used to identify puncturing strategies leading to punctured turbo codes with a low error floor and compute tight bounds on the average ML performance of punctured turbo codes employing short interleavers. Furthermore, we show that, if certain conditions are met, our approach can be extended, in a straightforward manner, to punctured turbo codes using long interleavers.

## 4.2 Punctured Convolutional Codes

A convolutional encoder having $k_0$ inputs and $n_0$ outputs, achieves a code rate of $R = k_0/n_0$. When the number of outputs is much greater than the number of inputs, i.e., $n_0 \gg k_0$, the reliability of the decoded information sequence is increased at the expense of bandwidth efficiency, defined as the number of information bits per second per Hz ($\text{bits}\cdot\text{sec}^{-1}\cdot\text{Hz}^{-1}$). In many practical applications, bandwidth efficiency is of critical importance, hence the use of high rate codes, where $k_0/n_0 \to 1$, is imperative. However, complexity of the Viterbi decoder in high rate codes can be overwhelming, since the existence of $2^{k_0}$ branches leaving or entering a state in the trellis diagram requires $2^{k_0}$ branch metric calculations per state per time step. Consequently, computational complexity of the decoder algorithm grows exponentially with the number of inputs $k_0$ to the convolutional encoder.

A high-rate convolutional code can be obtained by periodic elimination, known as *puncturing*, of specific codeword bits from the output of a parent low-rate convolutional encoder, which has the benefit of not compromising the computational complexity of the decoder. If the parent convolutional encoder generates $n_0$ output sequences, we define which output bits are eliminated at each time-step by means of a *puncturing pattern* $\mathbf{P}$. The puncturing pattern, which is repeated periodically every $M$ time steps, is represented by a $n_0 \times M$ matrix

$$\mathbf{P} = \begin{bmatrix} p_{1,1} & \cdots & p_{1,M} \\ \vdots & \ddots & \vdots \\ p_{n_0,1} & \cdots & p_{n_0,M} \end{bmatrix}, \tag{4.1}$$

where $p_{i,j} \in \{0,1\}$, with $i = 1,\ldots,n_0$ and $j = 1,\ldots,M$. For $p_{i,j} = 0$ the corresponding output bit is punctured, while for $p_{i,j} = 1$ the output bit is transmitted.

For the case of a punctured RSC encoder having two outputs, i.e., $n_0 = 2$, the puncturing pattern $\mathbf{P}$ of the code consists of the puncturing vector $\mathbf{P}_U$ for the systematic output sequence and the puncturing vector $\mathbf{P}_Z$ for the parity check output sequence

$$\mathbf{P} = \begin{bmatrix} \mathbf{P}_U \\ \mathbf{P}_Z \end{bmatrix} = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,M} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,M} \end{bmatrix}. \tag{4.2}$$

As an example, the output of the rate-1/2 RSC(1,5/7) code in Fig.4.1 is punctured using the pattern

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}. \tag{4.3}$$

According to the puncturing pattern, the second bit in each group of 3 output systematic bits is punctured. Similarly, the first bit in each group of 3 output parity

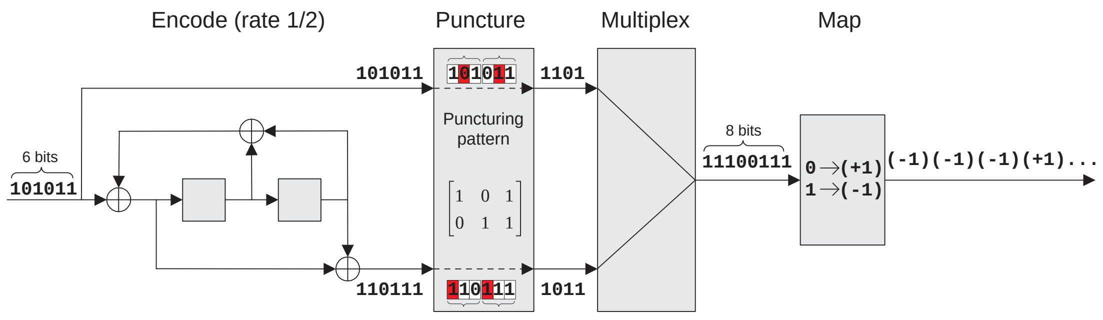


Figure 4.1: Puncturing of the rate-1/2 RSC(1,5/7) code to obtain a code of rate 3/4.

check bits is removed. Since 4 in every 6 codeword bits survive puncturing, an overall code rate of (1/2)(6/4)=3/4 is achieved.

The puncturing pattern $\mathbf{P}$ can also be seen as a matrix that consists of column vectors, namely column puncturing vectors (CPVs) $\mathbf{P}_j$, with $j = 1, \ldots, M$, that determine which bits will be punctured at each time-step. Accordingly, the puncturing pattern $\mathbf{P}$ can be expressed as:

$$\mathbf{P} = \begin{bmatrix} \mathbf{P}_1 & \mathbf{P}_2 & \ldots & \mathbf{P}_M \end{bmatrix} = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,M} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,M} \end{bmatrix}. \tag{4.4}$$

We use the latter representation throughout the following section, where we introduce the modified augmented state diagram for punctured convolutional block codes.

Soft decoding of punctured convolutional codes is performed in the same manner as the soft decoding of non-punctured convolutional codes, described in Chapter 2. However, when one or more bits in a branch of the trellis diagram are punctured, they do not contribute to the corresponding branch metric. In practice, a zero is inserted in each position of the received sequences, where a bit was punctured at the transmitter. Consequently, the decoding algorithm treats each inserted zero as a soft bit having an LLR of zero, meaning that a hard decision on that bit would give a "-1" or "+1" with equal probability. During the decoding process, zero LLRs do not contribute to the branch metrics, hence the decoded information sequence is computed based on the non-punctured received bits.

A decoder for the punctured RSC(1,5/7) code presented in Fig. 4.1, is illustrated in Fig.4.2. For clarity and simplicity, we assume that the channel output is noiseless. The puncturing pattern is used to identify the positions where bits have been punctured. Zeroes are inserted at the identified positions and the updated received

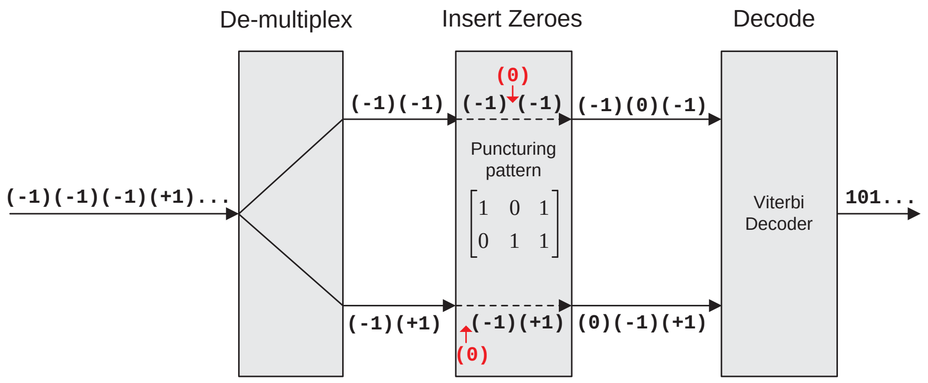


Figure 4.2: Insertion of zeroes before decoding a punctured convolutional code.

sequences are input to the convolutional decoder, which employs the conventional Viterbi algorithm described previously in Chapter 2.

## 4.3 Transfer Function of Punctured Convolutional Block Codes

In this section we extend the concept of the augmented state diagram to include the case of punctured convolutional block codes. For consistency, we continue to consider the rate-1/2 RSC(1,5/7) code, having a memory size of $\nu = 2$ and, consequently, $2^\nu = 4$ available states. In order to construct the augmented state diagram of the punctured RSC(1,5/7) block code, we need to introduce the CPV into the labeling procedure of each branch. Recall that a transition from a state $X_i$ to a state $X_k$, where $\{i, k\} \in \{0, 1, 2, 3\}$, is labeled $W^w U^u Z^z L$, where $w$ is the weight of the input sequence that caused a transition, which in turn generated a systematic output sequence with weight $u$ and a parity check output sequence with weight $z$. Let us concentrate on the same transition from $X_i$ to $X_k$ when puncturing occurs. If $\mathbf{P}_j = [p_{1,j} \;\; p_{2,j}]^{\mathrm{T}}$, where $j = 1, \ldots, M$, is the active CPV at a specific time-step, the label of the branch will change to $W^{w'} U^{u'} Z^{z'} L$, where $w'$, $u'$ and $z'$ are related to $w$, $u$, $z$ as well as the elements of $\mathbf{P}_j$, i.e., $p_{1,j}$ and $p_{2,j}$, as follows:

$$w' = w, \quad u' = u \cdot p_{1,j}, \quad z' = z \cdot p_{2,j}. \tag{4.5}$$

The values of $u'$ and $z'$ depend upon the active CPV, therefore the label of the branch that connects $X_i$ with $X_k$ cannot be a constant term, since it can assume up to $M$ different values due to the $M$ available CPVs.

To overcome this problem, we introduce $M$ sets of states. Each set $Y_j$ contains all possible states of the convolutional encoder. When $\mathbf{P}_j$ is the active CPV, a transition from state $X_i$ to state $X_k$ in the conventional state diagram corresponds to a transition from state $X_i$ in set $Y_{j-1}$, denoted as $X_i^{(j-1)}$ in the modified augmented state diagram, to state $X_k$ in set $Y_j$, denoted as $X_k^{(j)}$. At the next time-step, $\mathbf{P}_{j+1}$ becomes the active CPV and as time progresses, CPVs are repeated periodically, i.e., $\mathbf{P}_1$,$\mathbf{P}_2$,...,$\mathbf{P}_M$,$\mathbf{P}_1$,..., resulting in transitions to states that belong to sets $Y_1$,$Y_2$,...,$Y_M$,$Y_1$,..., respectively. Therefore, the problem of having $M$ different terms associated with a branch that connects state $X_i$ with state $X_k$, is overcome by having $M$ branches, each one of which pairs state $X_i$ of a set with state $X_k$ of a different set.

So as to better understand the concept of the augmented state diagram of a punctured convolutional block code, we give an example for a puncturing period of $M=2$. In order to increase the code rate of RSC(1,5/7) from 1/2 to 2/3, we use the following puncturing pattern:

$$\mathbf{P} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}, \text{ with } \mathbf{P}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \text{ and } \mathbf{P}_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \tag{4.6}$$

where $\mathbf{P}_1$ and $\mathbf{P}_2$ are CPVs. Since 1 in 4 codeword bits is punctured, 3 codeword bits are transmitted for every $M=2$ information bits, therefore the code rate of the punctured convolutional code is 2/3.The augmented state diagram of the rate-2/3 RSC(1,5/7) block code is presented in Fig.4.3. Blue colored lines, originating from states in set $Y_2$ and terminating at states in set $Y_1$, represent transitions during which $\mathbf{P}_1$ is the active CPV. Since both elements of $\mathbf{P}_1$ are equal to 1, the outputs of the encoder are not punctured, therefore the labels of the branches that connect the states of set $Y_2$ with the states of set $Y_1$ are identical to the labels of the corresponding branches of the augmented diagram in Fig.3.1 in Chapter 3. Red colored lines, originating from states in set $Y_1$ and terminating at states in set $Y_2$, represent transitions during which $\mathbf{P}_2$ is the active CPV. In this case, the parity check output of the encoder is punctured, therefore term $Z$ does not appear in any of the branch labels. To complete the augmented diagram, states $X_S$ and $X_E$ have been included. Since the encoder starts from state $X_S$, $\mathbf{P}_1$ is the active CPV during the transition from $X_S$ to a state in set $Y_1$ at the first time-step. At the last time-step, the encoder returns to the zero state, i.e., a transition to state $X_E$ occurs. In order to terminate the code, those states of each set which are connected to state $X_0$ of a different set, must also be connected to state $X_E$. We see in Fig.3.1 that states $X_0^{(1)}$ and $X_1^{(1)}$, as well as $X_0^{(2)}$ and $X_1^{(2)}$ are connected to states $X_0^{(2)}$ and $X_0^{(1)}$, respectively. Hence, these four states should also be connected to state $X_E$, as shown in Fig.3.1 using dashed lines.

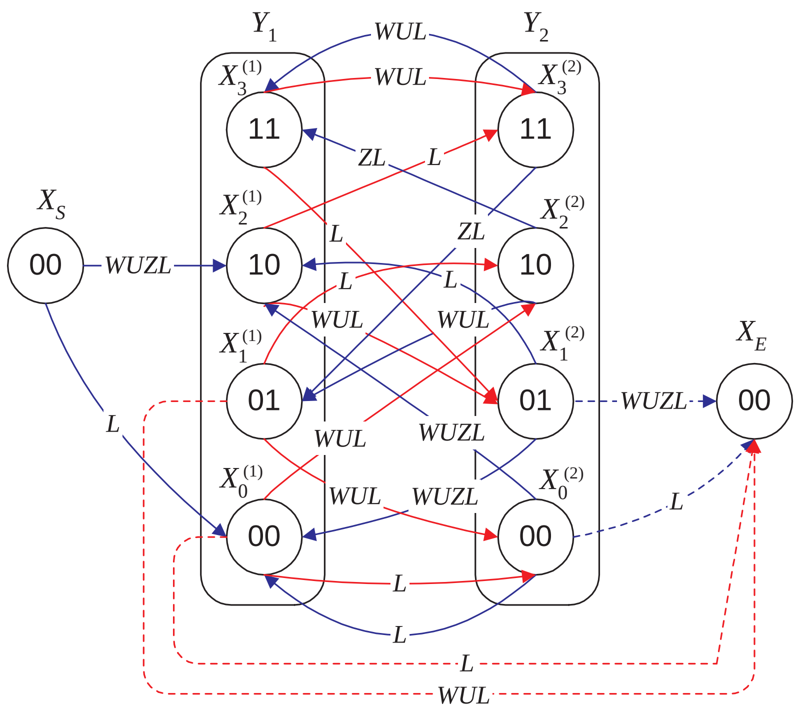


Figure 4.3: Augmented state diagram of the rate-2/3 RSC(1,5/7) block code.

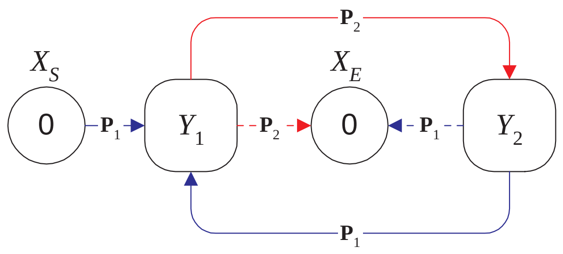


Figure 4.4: Generic diagram of a punctured convolutional block code ($M = 2$).

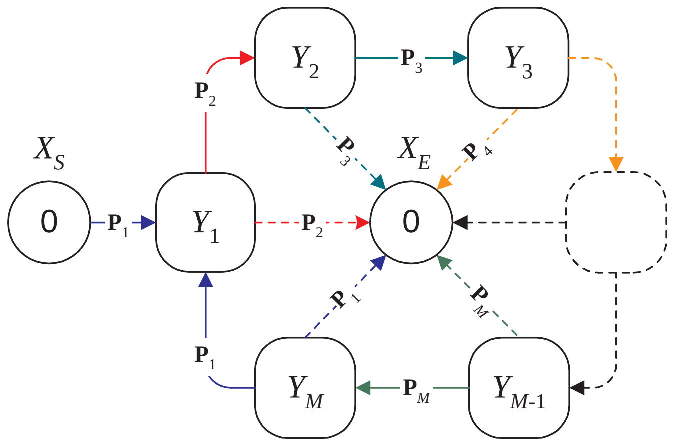


Figure 4.5: Generic diagram of a punctured convolutional block code.

A generic representation of the augmented state diagram of a convolutional block code, which is punctured using a pattern of period $M=2$, is shown in Fig.4.4. Both sets $Y_1$ and $Y_2$ contain all possible states of the convolutional encoder. A link between two sets represents the collection of branches of the augmented diagram that connect the states of the two different sets. Each link is labeled with the associated CPV. The same logic applies to the link that connects the start state $X_S$ with $Y_1$ as well as the links that connect all sets to the end state $X_E$.

The general case of a convolutional block code, which is punctured using a pattern of period $M$, is shown in Fig.4.5. Upon deriving the $(2^\nu \times M)+1$ state equations, we solve them for the ratio $X_E/X_S$ and, similarly to the case of non-punctured convolutional codes, we obtain a sum of two terms. The first term enumerates the all-zero codeword sequences and can be ignored, whereas the second term is the extended transfer function of the punctured convolutional block code. We have shown in Chapter 3 that the extended transfer function $f(W,U,Z,L)$ is the sum of the conditional extended transfer functions $f(W,U,Z,l)$, each one of which enumerates all codeword sequences of path length $l$, where $l = 1, 2, \ldots \infty$. Consequently, for an input information sequence of length $N$, the transfer function $B(W,U,Z)$ of the punctured convolutional block code is given by the conditional extended transfer function $f(W,U,Z,l=N)$, according to 3.12.

Nevertheless, it is worthwhile to associate the various CPVs and the puncturing period $M$ with the conditional extended transfer functions. If we link the path length $l$ to the puncturing period $M$ using the expression $l = \kappa M + j$, where $\kappa$ and $j$ are non-negative integers, the extended transfer function $f(W,U,Z,L)$ can be written as

$$\begin{aligned} f(W,U,Z,L) &= \sum_{l=1}^{\infty} f(W,U,Z,l)L^l \\ &= \sum_{\kappa M+j=1}^{\infty} f(W,U,Z,\kappa M+j)L^{\kappa M+j} \\ &= \sum_{\kappa=0}^{\infty}\sum_{j=1}^{M} f(W,U,Z,\kappa M+j)L^{\kappa M+j} \end{aligned} \tag{4.7}$$

or, equivalently,

$$f(W,U,Z,L) = \sum_{j=1}^{M}\left(\sum_{\kappa=0}^{\infty} f(W,U,Z,\kappa M+j)L^{\kappa M+j}\right). \tag{4.8}$$

We denote the sum

$$f^{\mathbf{P}_j}(W,U,Z,L) = \sum_{\kappa=0}^{\infty} f(W,U,Z,\kappa M+j)L^{\kappa M+j}, \tag{4.9}$$

as the *extended transfer function of the $j$-th CPV*, which enumerates all trellis paths of lengths $j, (M + j), (2M + j)$, etc. We see in Fig.4.5 that these paths terminate at state $X_E$ when $\mathbf{P}_j$ is the active CPV. Taking into account (4.9), the extended transfer function assumes the form

$$f(W, U, Z, L) = \sum_{j=1}^{M} f^{\mathbf{P}_j}(W, U, Z, L). \tag{4.10}$$

In order to compute the transfer function of the punctured convolutional block code for a specific input block length $N$, we only need to consider the conditional extended transfer function $f(W, U, Z, l = N)$, which is a component of a particular extended transfer function $f^{\mathbf{P}_j}(W, U, Z, L)$. The importance of this observation is made clear, if we revisit Fig.4.5. We notice that each set of states is interconnected with the end state $X_E$ through a link, shown with a dashed line in Fig.4.5, which is associated with the extended transfer function of a unique CPV. Hence, a total of $M$ links terminate at state $X_E$ and the extended transfer functions of all CPVs are considered. However, we can determine the $j$-th CPV, being active when the paths of length $N$ terminate at state $X_E$, retain the associated link and remove all others. Consequently, the terms in the state equation for $X_E$ are reduced from $2M$ to only 2, and the ratio $X_E/X_S$ no longer provides the extended transfer function $f(W, U, Z, L)$ but the extended transfer function of the $j$-th CPV, i.e., $f^{\mathbf{P}_j}(W, U, Z, L)$. The conditional extended transfer function $f(W, U, Z, l\!=\!N)$ can be obtained from $f^{\mathbf{P}_j}(W, U, Z, L)$ by imposing the computational constraints described in Chapter 3.

If we wish to derive the transfer function of the rate-2/3 RSC(1,5/7) block code, obtained after puncturing the rate-1/2 original code using the pattern in (4.6), we need to simplify the augmented state diagram in Fig.4.3 by removing the appropriate terminating branches. For an input information sequence of length $N\!=\!100$, we can compute the active CPV at the 100-th time-step using the expression

$$\kappa M + j = 100, \tag{4.11}$$

where $\kappa$ and $j$ are both non-negative integers with $\kappa \geq 0$ and $1 \leq j \leq M$. The puncturing period $M$ in our example is equal to 2. The unique solution to the above equation is $\kappa\!=\!49$ and $j\!=\!2$. Consequently, $P_2$ is the active CPV at the 100-th time-step, hence the branches originating from states $X_0^{(1)}$, $X_1^{(1)}$ and terminating at state $X_E$ need only be retained. The updated augmented diagram is shown in Fig.4.6, while the generic representation is shown in Fig.4.7. Note that, according to 4.9, both diagrams are valid for any input sequence having an even number of information

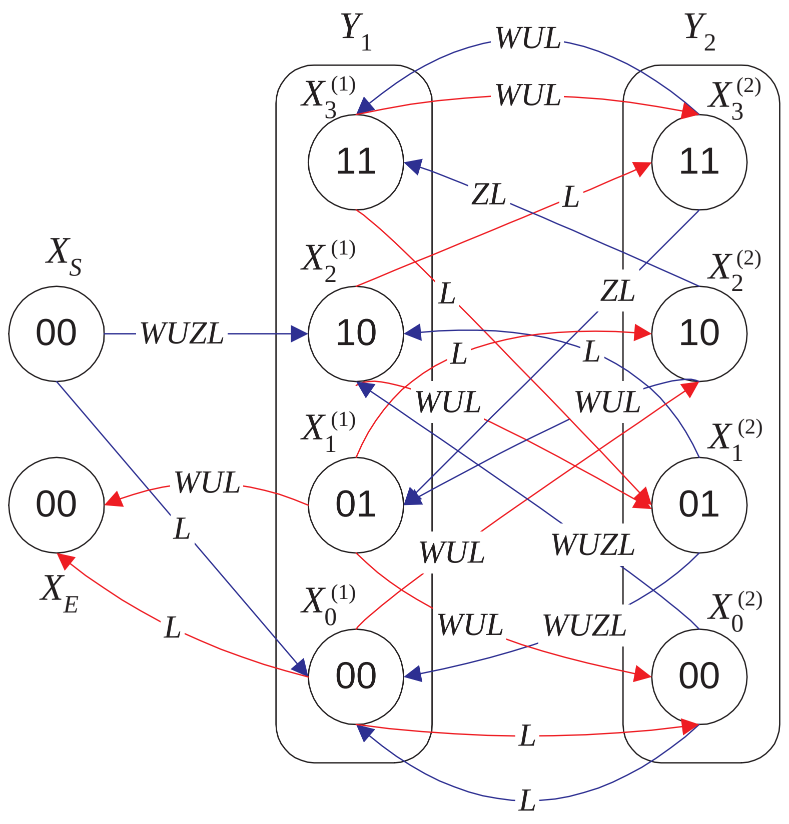


Figure 4.6: Augmented state diagram of the rate-2/3 RSC(1,5/7) block code, assuming that the input block length is an even number.

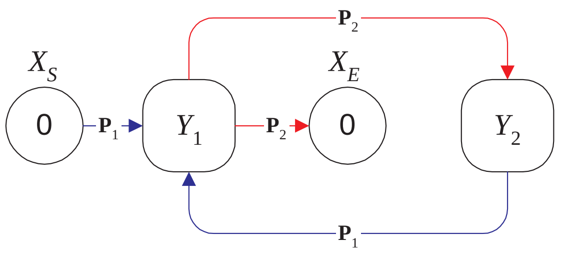


Figure 4.7: Generic diagram of a punctured convolutional block code having a puncturing period $M = 2$. The input sequence is assumed to have an even number of information bits.

bits, i.e., $N=2,4,6,\dots$ etc. Upon solving the state equations for the ratio $X_E/X_S$, we obtain the extended transfer function of the 2nd CPV, i.e., $f^{\mathbf{P}_2}(W,U,Z,L)$, from which we can derive the conditional extended transfer function $f(W,U,Z,l=100)$, and, ultimately, the transfer function of the punctured convolutional block code, $B(W,U,Z)$, for an input information sequence of length $N=100$.

## 4.4 Handling Long Puncturing Patterns and Long Input Sequences

Computation of the transfer function of a punctured convolutional block code based on the augmented state diagram requires the solution of $(2^{\nu}\times M)+1$ state equations for the ratio $X_E/X_S$, where $\nu$ is the memory size of the encoder and $M$ is the puncturing period. Consequently, the computational complexity of the transfer function of the code depends upon the period of the puncturing pattern. Furthermore, similarly to the case of non-punctured convolutional block codes, the inherent loop of zero input and output weight at the zero state introduces a computational burden in the calculation of the ratio $X_E/X_S$, which becomes intolerable when long information sequences are input to the punctured convolutional block encoder.

The augmented state diagram of a punctured convolutional block code is an alternative representation to the trellis diagram of length $N$, obtained after puncturing the outputs of a parent convolutional block code, using a pattern of period $M$. The puncturing pattern essentially converts the trellis diagram from time-invariant to time-variant with a period of $M$ time-steps. However, if the length of the trellis diagram or, equivalently, the length of the input sequence is a multiple of the puncturing period, i.e., $N=\kappa M$, time variability can be overcome. More specifically, it is valid to say that the input sequence of length $N$ consists of $\kappa$ groups of $M$ bits each, and thus causes $\kappa$ *composite transitions*. A composite transition between two states, e.g., $X_i$ and $X_k$, is caused by a group of $M$ information bits and essentially breaks down into a sequence of $M$ transitions starting from state $X_i$ and terminating at state $X_k$, $M$ time-steps later. Consequently, the time-variant trellis diagram of length $N$ and period $M$ can be replaced by a time-invariant trellis diagram of length $\kappa$, where $2^M$ branches emerge from each state instead of 2. In Fig.4.8 we see that the two CPVs of the puncturing pattern act differently upon the systematic and parity check outputs of the RSC(1,5/7) code, causing the trellis diagram to periodically vary with time. However, the time-variant trellis with period $M=2$ and

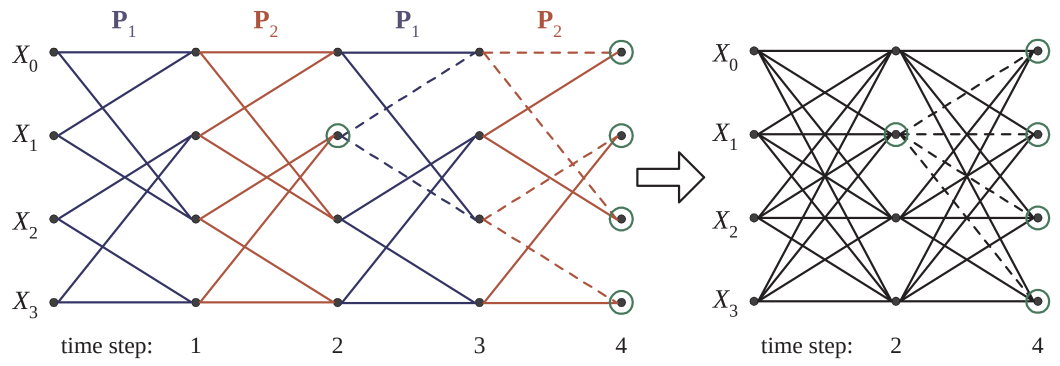


Figure 4.8: Conversion of the time-variant trellis diagram of RSC(1,5/7) with $M=2$ to a time-invariant trellis diagram. Each composite transition in the time-invariant trellis (right) breaks down to a sequence of two transitions in the time-variant trellis (left).

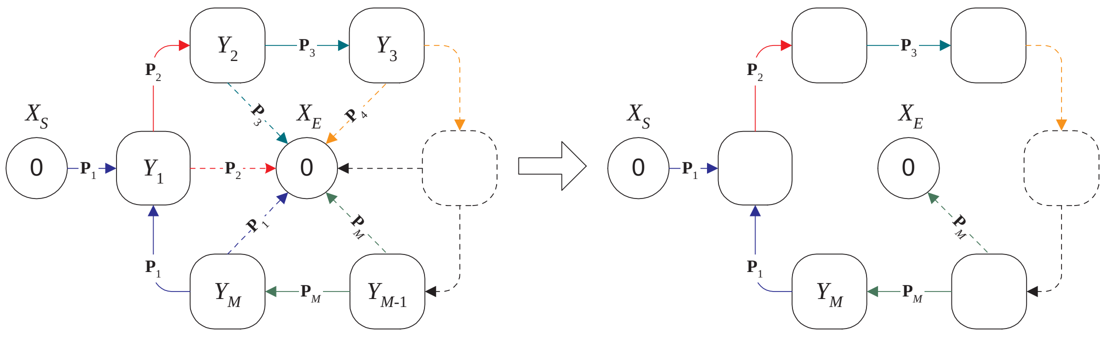


Figure 4.9: In the general case (left diagram), states from sets $Y_1$ to $Y_M$ are considered in the state equations. However, for input sequences of length $N=\kappa M$, only the states from set $Y_M$ need to be considered (right diagram).

length $N$ can collapse to an equivalent time-invariant trellis of length $N/2$. Note that 2 time-steps are required for the completion of a composite transition.

In general, $M$ time-steps are required for each composite transition, hence the states participating in the time-invariant trellis diagram are the states which belong in set $Y_M$ of the augmented state diagram. These states, as well as the start state $X_S$ and the end state $X_E$, are the only ones that need to be considered when deriving the state equations. The reduction of the augmented state diagram for the general case of a puncturing pattern having a period of $M$ is depicted in Fig.4.9. Note that, although each set $Y_j$, with $j \in \{1, \ldots, M\}$, of the modified diagram consists of $2^\nu$ states, variables $X_i$ with $i \in \{0, \ldots, 2^\nu - 1\}$ are only allocated to the states in set $Y_M$. Effectively, the augmented state diagram of $(2^\nu \times M) + 2$ states collapses to an augmented state diagram of only $2^\nu + 2$ states, where each branch represents a composite transition of $M$ time-steps. For example, the augmented state diagram of the rate-2/3 punctured RSC(1,5/7) block code shown in Fig.4.6, is reduced to the augmented state diagram illustrated in Fig.4.10. Consequently, the initial system of $(2^{\nu} \times M)+1$ state equations is also reduced to a system of only $2^{\nu}+1$ state equations. Upon solving the equations for the ratio $X_E/X_S$, we obtain the extended transfer function of the $M$-th CPV, i.e., $f^{\mathbf{P}_M}(W, U, Z, L)$, which enumerates all paths whose length is a multiple of the puncturing period.

For long input sequences and relatively short puncturing patterns, i.e., $N \gg M$, we can approximate $N$ to the nearest multiple of $M$, when $N \neq \kappa M$. Such an approximation does not yield an accurate expression for the transfer function of the punctured convolutional block code, however simulation results show that the bit error performance of turbo codes using punctured convolutional block codes as constituent codes, is not dramatically changed by small variations in the length of the input sequence or equivalently, the interleaver size. Nevertheless, as the length of the input sequence increases, enumeration of all paths having length $N$ becomes computationally intensive, due to the inherent loop of zero input and output weight at the zero state. We described in Chapter 3 that if no puncturing is applied, complexity is considerably reduced when we remove the inherent self-loop at state $X_0$, remove the branches that interconnect states $X_S$ and $X_E$ through state $X_0$ and introduce the indeterminate variable $\Omega$ to the label of those branches that lead to a re-merging into the zero state, either $X_0$ or $X_E$. The transfer function $B(W, U, Z)$ of the convolutional block code is then derived from the intermediate transfer function $T(W, U, Z, L, \Omega)$ obtained from the modified augmented state diagram. The same modifications can be performed on the collapsed augmented state diagram of a punctured convolutional block code to make computation of its transfer function feasible for long input sequences. For the case of the rate-2/3 punctured RSC(1,5/7) block code, the simplified augmented state diagram shown in Fig.4.11 has been obtained from the collapsed augmented state diagram depicted in Fig.4.10.

We explained in Chapter 3 that $T(W, U, Z, L, \Omega)$, the intermediate transfer function obtained from the simplified augmented state diagram, is a sum of monomials of the form $W^w U^u Z^z L^l \Omega^n$, each one of which corresponds to a codeword sequence represented by a path in the trellis diagram. However, the path length $l$ only assumes values that are multiples of the puncturing period $M$, whereas the $n$ re-mergings with the path of the all-zero codeword sequence occur at time intervals which are also multiples of the puncturing period $M$. From each monomial $W^w U^u Z^z L^l \Omega^n$, we can derive the total number $K'[l, n]$ of codeword sequences having an overall

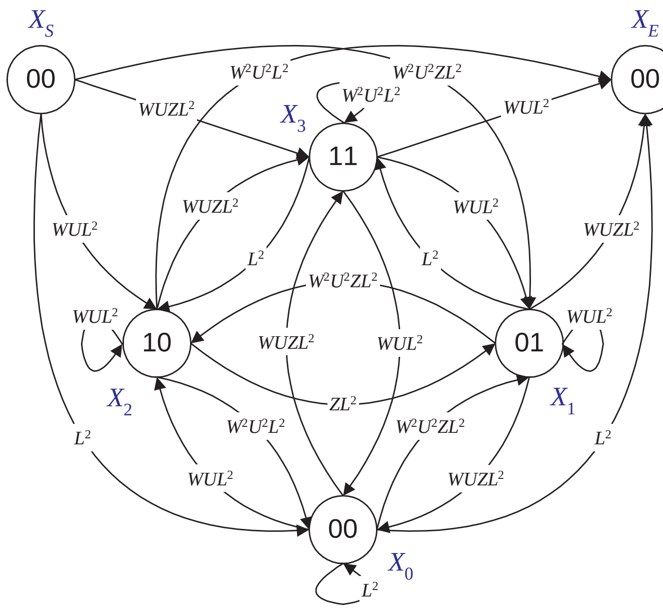


Figure 4.10: Collapsed augmented state diagram of the rate-2/3 RSC(1,5/7) block code for a puncturing period of $M = 2$.

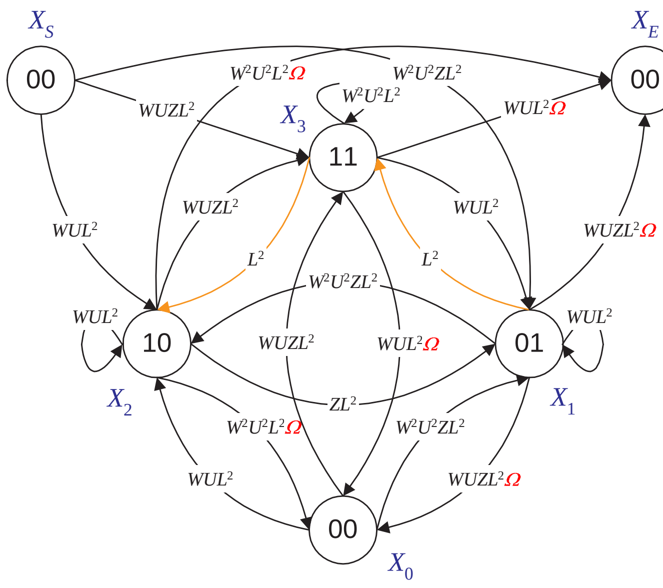


Figure 4.11: Simplified augmented state diagram of the rate-2/3 RSC(1,5/7) block code. No loops of zero input and output weight exist.

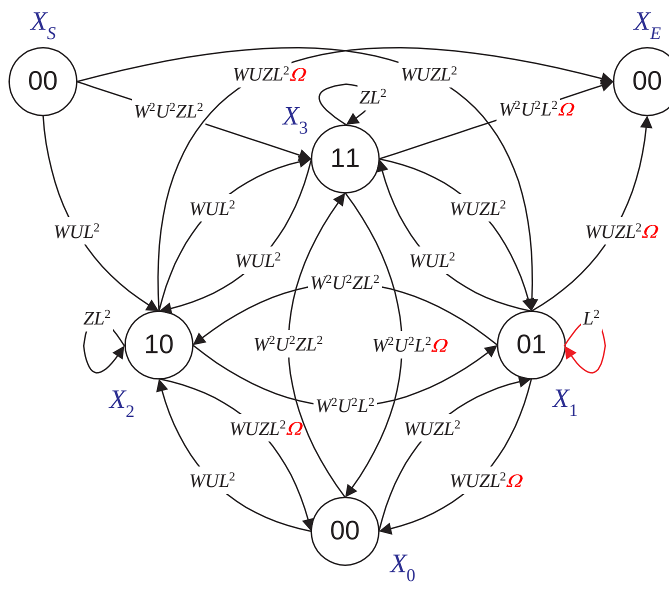


Figure 4.12: Simplified augmented state diagram of the rate-2/3 RSC(1,7/5) block code. There is a self-loop of zero input and output weight at state $X_1$.

path length $N$, with zeroes before, after or between the $n$ re-mergings, by modifying (3.28) as follows

$$K'[l,n] = \binom{\frac{N-l}{M}+n}{n} = \frac{\left(\frac{N-l}{M}+n\right)!}{\left(\frac{N-l}{M}\right)!n!}. \tag{4.12}$$

As described in Chapter 3, the coefficients $B_{w,u,z}$ and, ultimately, the transfer function $B(W,U,Z)$ of the punctured convolutional block code, can be obtained from

$$B_{w,u,z} = \sum_l \sum_n K'[l,n]T_{w,u,z,l,n}, \tag{4.13}$$

where $T_{w,u,z,l,n}$ are the coefficients of the intermediate function $T(W,U,Z,L,\Omega)$.

Computation of the transfer function $B(W,U,Z)$ is speeded up by simplifying the augmented state diagram of the punctured convolutional block code, only if there are no loops of zero input and output weight besides the self-loop at the zero state. We observe in Fig.4.11 that although branches having zero input and output weight exist, they do not form loops, hence computation of the transfer function can be speeded up. If the same pattern (4.6) is used to puncture the output of the RSC(1,7/5) encoder, we obtain the simplified augmented diagram depicted in Fig.4.12. We see that, even though the unwanted inherent loop has been removed, a loop of zero input and zero output weight still remains at state $X_1$. Expression 4.12 does not account for loops other than the inherent loop at the zero state, hence it will only compute a fraction of the total number of codeword sequences, corresponding to paths of length $N$. As a result, the transfer function of the punctured convolutional block code will not be successfully derived. Concluding, simplification of the augmented state diagram does result in computational complexity reduction and hence, permits the accurate and rapid calculation of the transfer function of a punctured convolutional block code for long input sequences, provided that there exist no loops of zero input and output weight in the simplified augmented state diagram.

## 4.5 Performance Analysis

We demonstrated in Chapter 2 that in the case of a non-punctured rate-1/3 turbo encoder, $\mathcal{P}$, composed of two RSC encoders, $\mathcal{C}_1$ and $\mathcal{C}_2$, a tight upper bound can be found when the transfer function $B^{\mathcal{P}}(W,U,Z)$ of the turbo code is computed. The assumption of a uniform interleaver of size $N$, allows us the straightforward calculation of the transfer function $B^{\mathcal{P}}(W,U,Z)$ of the turbo code, based on the transfer functions, $B^{\mathcal{C}_1}(W,U,Z)$ and $B^{\mathcal{C}_2}(W,U,Z)$, of the constituent codes. The union bound obtained from (2.65), provides an accurate upper bound on the average ML performance of the turbo code.

As depicted in Fig.4.13 and in a similar manner to that for punctured convolutional codes, a high-rate turbo code can be obtained by puncturing the three outputs of a rate-1/3 turbo encoder $\mathcal{P}$, that is the systematic output and the two parity check outputs, using a pattern of the form

$$\mathbf{P} = \begin{bmatrix} \mathbf{P}_U \\ \mathbf{P}_{Z_1} \\ \mathbf{P}_{Z_2} \end{bmatrix} = \begin{bmatrix} p_{1,1} & \cdots & p_{1,M} \\ p_{2,1} & \cdots & p_{2,M} \\ p_{3,1} & \cdots & p_{3,M} \end{bmatrix} \tag{4.14}$$

where $M$ is the puncturing period. The pattern can be decomposed into three puncturing vectors; one for the systematic output, $\mathbf{P}_U$, and two for the parity check outputs, $\mathbf{P}_{Z_1}$ and $\mathbf{P}_{Z_2}$. Vectors $\mathbf{P}_U$ and $\mathbf{P}_{Z_1}$ form a pattern, which is used to puncture the outputs of the first constituent RSC encoder, $\mathcal{C}_1$, and is denoted as

$$\mathbf{P}^{\mathcal{C}_1} = \begin{bmatrix} \mathbf{P}_U \\ \mathbf{P}_{Z_1} \end{bmatrix} = \begin{bmatrix} p_{1,1} & \cdots & p_{1,M} \\ p_{2,1} & \cdots & p_{2,M} \end{bmatrix}, \tag{4.15}$$

whereas the outputs of the second constituent RSC encoder, $\mathcal{C}_2$, are punctured according to the pattern

$$\mathbf{P}^{\mathcal{C}_2} = \begin{bmatrix} 0 \\ \mathbf{P}_{Z_2} \end{bmatrix} = \begin{bmatrix} 0 & \cdots & 0 \\ p_{3,1} & \cdots & p_{3,M} \end{bmatrix}. \tag{4.16}$$

Hence, the turbo encoder $\mathcal{P}'$, obtained after puncturing the outputs of the rate-1/3 turbo encoder $\mathcal{P}$ using pattern $\mathbf{P}$, can also be seen as the parallel concatenation

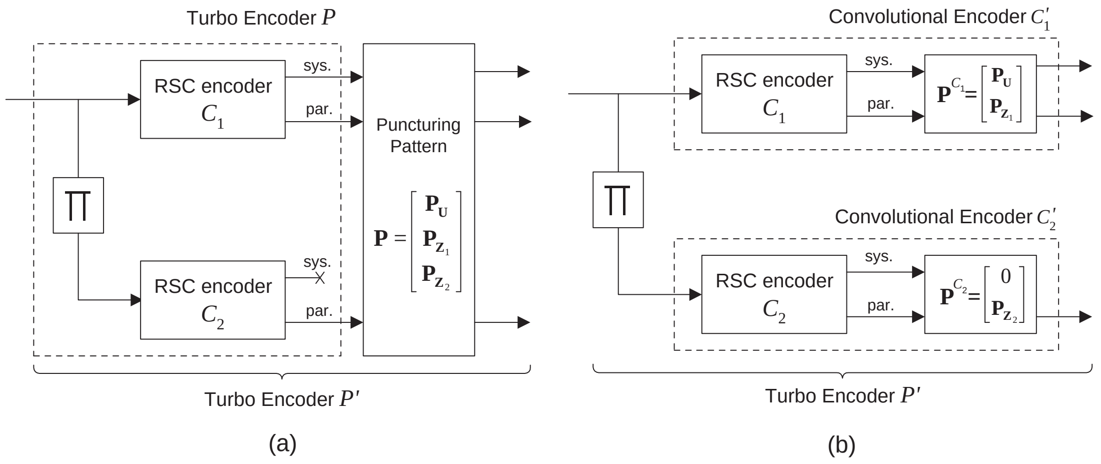

Turbo Encoder $P$
RSC encoder $C_1$
RSC encoder $C_2$
sys.
par.
Puncturing Pattern
$\mathbf{P} = \begin{bmatrix} \mathbf{P_U} \\ \mathbf{P_{Z_1}} \\ \mathbf{P_{Z_2}} \end{bmatrix}$
Turbo Encoder $P'$
(a)
Convolutional Encoder $C_1'$
$\mathbf{P}^{C_1} = \begin{bmatrix} \mathbf{P_U} \\ \mathbf{P_{Z_1}} \end{bmatrix}$
Convolutional Encoder $C_2'$
$\mathbf{P}^{C_2} = \begin{bmatrix} 0 \\ \mathbf{P_{Z_2}} \end{bmatrix}$
(b)

Figure 4.13: Equivalent approaches for puncturing a rate-1/3 turbo encoder to obtain a higher code rate.

of two punctured constituent convolutional codes, $\mathcal{C}_1'$ and $\mathcal{C}_2'$, each one of which is obtained by puncturing the outputs of a rate-1/2 RSC code, $\mathcal{C}_1$ or $\mathcal{C}_2$ accordingly, using the puncturing pattern $\mathbf{P}^{\mathcal{C}_1}$ or $\mathbf{P}^{\mathcal{C}_2}$, respectively. This equivalent representation of the punctured turbo encoder $\mathcal{P}'$ is presented in Fig.4.13.

Based on the latter representation, we can safely conclude that under the assumption of a uniform interleaver of size $N$, the transfer function of a punctured turbo code can be derived in a similar manner to that of the transfer function of a rate-1/3 turbo code. More specifically, we first obtain the transfer functions $B^{\mathcal{C}_1'}(W,U,Z)$ and $B^{\mathcal{C}_2'}(W,U,Z)$ of the constituent punctured convolutional block codes from their augmented state diagrams for input sequences of length $N$, and then we derive their conditional weight enumerating functions $B^{\mathcal{C}_1'}(w,U,Z)$ and $B^{\mathcal{C}_2'}(w,U,Z)$, respectively. The conditional weight enumerating function $B^{\mathcal{P}'}(w,U,Z)$ of the punctured turbo code can be obtained from

$$B^{\mathcal{P}'}(w,U,Z) = \frac{B^{\mathcal{C}_1'}(w,U,Z) \cdot B^{\mathcal{C}_2'}(w,U=1,Z)}{\binom{N}{w}}, \tag{4.17}$$

elaborating on (2.69), initially proposed in [16] for non-punctured rate-1/3 turbo codes. The sum of the conditional weight enumerating functions $B^{\mathcal{P}'}(w,U,Z)$ over all values of input weight $w$, where $w=1,2,\ldots,N$, provides the transfer function $B^{\mathcal{P}'}(W,U,Z)$ of the punctured turbo code. From the transfer function, we can obtain the input-output weight enumerating function $B^{\mathcal{P}'}(W,D)$ and use its coefficients

$B^{\mathcal{P}'}_{w,d}$ to compute the union bound

$$\begin{aligned} P_b &\leq \frac{1}{N}\sum_d \sum_w w B^{\mathcal{P}'}_{w,d} Q\left(\sqrt{\frac{2E_b R_{\mathcal{P}'}}{N_0}d}\right) \\ &= \sum_d D_d Q\left(\sqrt{\frac{2E_b R_{\mathcal{P}'}}{N_0}d}\right), \end{aligned} \tag{4.18}$$

which is an upper bound on the average bit error probability of the punctured turbo code. Note that $R_{\mathcal{P}'}$ is the rate of the turbo code after puncturing, while coefficients $D_d$ can be obtained from the coefficients $B^{\mathcal{P}'}_{w,d}$, as described previously in Chapter 3, i.e.,

$$D_d = \sum_w \frac{w}{N} B^{\mathcal{P}'}_{w,d}. \tag{4.19}$$

Based on the research carried out by Hagenauer [41] on rate-compatible punctured convolutional codes in 1988 and the work of Haccoun and Bégin [42] in 1989 on punctured convolutional codes, design criteria for punctured turbo codes were proposed by Barbulescu and Pietrobon [43] in 1995, Fan Mo *et al.* [44] in 1999, Açikel and Ryan [45] also in 1999, and Babich *et al.* [46] in 2002. Simulation-based analyses to identify a relationship between the structure of a puncturing pattern and the performance of the corresponding punctured turbo code were also carried out by Land and Hoeher [47] in 2000, Blazek *et al.* [48] in 2002 and Crozier *et al.* [49] in 2005. However, an analytic approach to evaluate the performance of punctured turbo codes was developed by Kousa and Mugaibel [45] in 1999. The authors proposed a technique based on Divsalar's approach, presented in Chapter 3, which they modified so as to take the puncturing pattern into account. Although elegant, the proposed approach can only be implemented when very short interleavers are considered and, consequently, the authors fail to compare theoretical bounds with simulation results. Furthermore, the authors draw conclusions on rate-1/2 turbo codes assuming that only the parity check outputs of the turbo encoders are punctured.

In the following subsections, we use our proposed method to investigate the impact on the bit error rate performance of puncturing both the systematic and the parity check outputs of a turbo encoder. In addition, we demonstrate that accurate bounds on the bit error probability of punctured turbo codes can be derived unconditionally, if short interleavers are used, or conditionally, if long interleavers are employed.

### 4.5.1 Good Puncturing Patterns

When a punctured turbo code uses a short interleaver, we can directly apply our technique to obtain the transfer function of each constituent code and compute the upper bound on the average bit error probability of the corresponding punctured turbo code, using (4.18.) For large interleaver sizes, and thus long input sequences, our technique is computationally feasible only if the puncturing pattern does not create loops of zero input and output weight in the augmented state diagrams of the constituent encoders. However, we can use our technique assuming a short interleaver, to identify patterns that generate good, in terms of bit error probability, punctured turbo codes, and extend our results to turbo codes using long interleavers.

More specifically, we set as our objective the design of a good punctured turbo code $\mathcal{P}'$ of rate $R_{\mathcal{P}'}$, obtained by puncturing the output of a rate-1/3 turbo code, using a pattern of period $M$. We need to consider every possible puncturing pattern of period $M$ and rate $R_{\mathcal{P}'}$, compute the transfer function of the corresponding punctured turbo code and use the union bound expression (4.18) to derive the required $E_b/N_0$ for a targeted bit error probability, or vice-versa. We then perform an exhaustive search to identify those punctured turbo codes that meet our performance requirements.

When a puncturing pattern of period $M$ is used at the output of a rate-1/3 turbo code to achieve a rate of $R_{\mathcal{P}'}$, an input sequence of $M$ information bits generates $M/R_{\mathcal{P}'}$ codeword bits. Hence, $M/R_{\mathcal{P}'}$ out of the $3M$ elements of the puncturing pattern are set to 1, while the rest are set to 0. Consequently, there are

$$\binom{3M}{M/R_{\mathcal{P}'}} = \frac{(3M)!}{\left(3M - \frac{M}{R_{\mathcal{P}'}}\right)!\left(\frac{M}{R_{\mathcal{P}'}}\right)!} \tag{4.20}$$

permutations of 0's and 1's, giving an equal number of patterns leading to a rate of $R_{\mathcal{P}'}$. For instance, there are 495 different patterns having a puncturing period of $M = 4$ that achieve a code rate of $R_{\mathcal{P}'} = 1/2$. For a code rate of $R_{\mathcal{P}'} = 2/3$, the number of patterns increases to 924.

As an example, we consider the rate-1/3 PCCC(1,5/7,5/7) to be the parent code, a puncturing pattern of period $M = 4$ is assumed and a code rate of $R_{\mathcal{P}'} = 1/2$ is targeted. In order to identify good puncturing patterns, we consider a short pseudo-random interleaver of size $N = 36$ and we compute the transfer function of the punctured turbo code for all 495 possible patterns. We use the transfer function to obtain the free effective distance $d_{\text{free,eff}}$ as well as the required $E_b/N_0$ value for a specific bit error probability, e.g., $10^{-6}$, exploiting (4.18). Finally, we order all 495 configurations according to their free effective distances and $E_b/N_0$ values.

Table 4.1: Rate-1/2 and 2/3 configurations for PCCC(1,5/7,5/7). The interleaver size $N$ is 36 bits.

| **Configuration** | $\mathbf{P}_U$ | $\mathbf{P}_{Z_1}$ | $\mathbf{P}_{Z_2}$ | $d_{\text{free,eff}}$ | $E_b/N_0$ for $P_b \leq 10^{-6}$ |
|---|---|---|---|---|---|
| Rate-1/2 Sys. | [1 1 1 1] | [1 0 1 0] | [0 1 0 1] | 6 | 6.438 dB |
| Rate-1/2 PS(3) | [1 1 1 0] | [1 0 1 1] | [0 1 1 0] | 6 | 6.243 dB |
| Rate-1/2 PS(2) | [0 1 0 1] | [1 1 1 0] | [1 1 0 1] | 7 | 6.291 dB |
| Rate-1/2 PS(1) | [0 0 1 0] | [1 1 0 1] | [1 1 1 1] | 7 | 5.959 dB |
| Rate-2/3 Sys. | [1 1 1 1] | [1 0 0 0] | [0 0 1 0] | 2 | 7.313 dB |
| Rate-2/3 PS(3) | [1 0 1 1] | [0 1 0 0] | [0 1 1 0] | 4 | 7.168 dB |
| Rate-2/3 PS(2) | [1 1 0 0] | [0 0 1 1] | [0 1 1 0] | 4 | 6.786 dB |

Traditionally, punctured codes are classified as systematic (Sys), partially systematic (PS) or non-systematic (NS), depending on whether all, some or none of their systematic bits are transmitted [47]. In Table 4.1, we have included four configurations that lead to good rate-1/2 punctured turbo codes. The first one is a systematic turbo code, since $\mathbf{P}_U$ allows all 4 systematic bits to be transmitted, while the remaining three configurations are partially systematic with a decreasing number of transmitted systematic bits. The upper bound for each rate-1/2 punctured turbo code is compared to simulation results, in Fig.4.14. We observe that all three partially systematic turbo codes achieve a lower ML bound and, consequently, a lower error floor than that of the systematic turbo code. However, we also observe that as the number of systematic bits is reduced, performance of the simulated suboptimal iterative decoder, after 10 iterations applying the BCJR decoding algorithm, does not quickly converge to the upper bound. These results emphasize the importance of the systematic sequence, which is used by both component decoders of the turbo decoder. In accordance with the findings of Blazek *et al.* [48] and Crozier *et al.* [49], puncturing of the systematic sequence expands the non-convergence region, hence performance converges to the ML bound only for high $E_b/N_0$ values. Therefore, our method can identify punctured turbo codes that exhibit low error floors, however convergence towards the error floor can be investigated using extrinsic information transfer (EXIT) chart analysis, proposed by ten Brink in [34]. A brief description of EXIT charts is provided in Chapter 6.

According to Table 4.1 and Fig.4.14, for a code rate of 1/2, configuration "PS(1)"

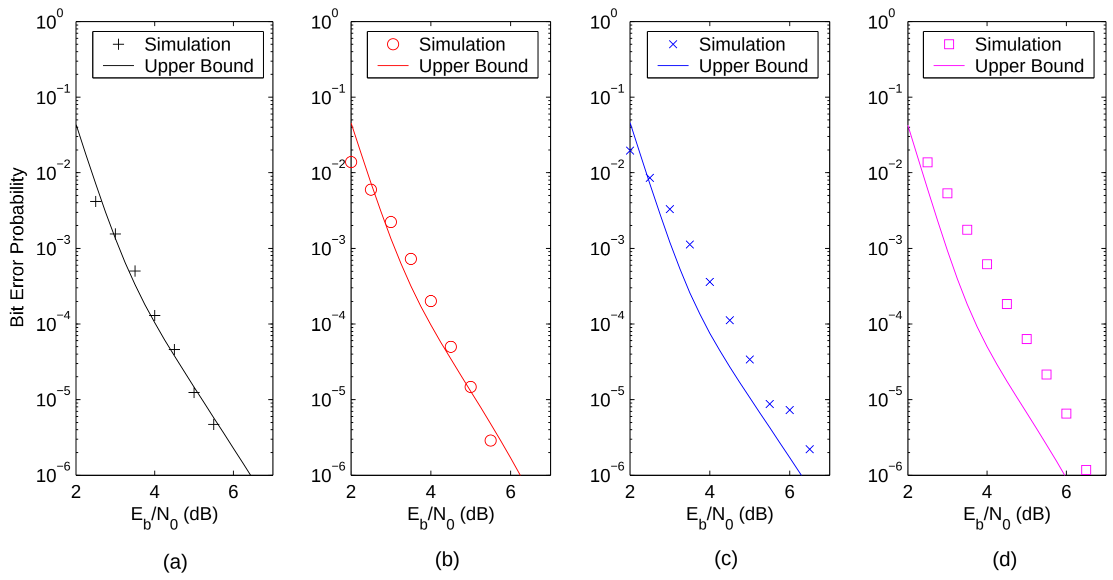


Figure 4.14: Comparison between bounds and simulation results after 10 iterations for various configurations of a PCCC(1,5/7,5/7) using an interleaver size of 36 bits: (a) Rate-1/2 Sys., (b) Rate-1/2 PS(3), (c) Rate-1/2 PS(2), (d) Rate-1/2 PS(1).

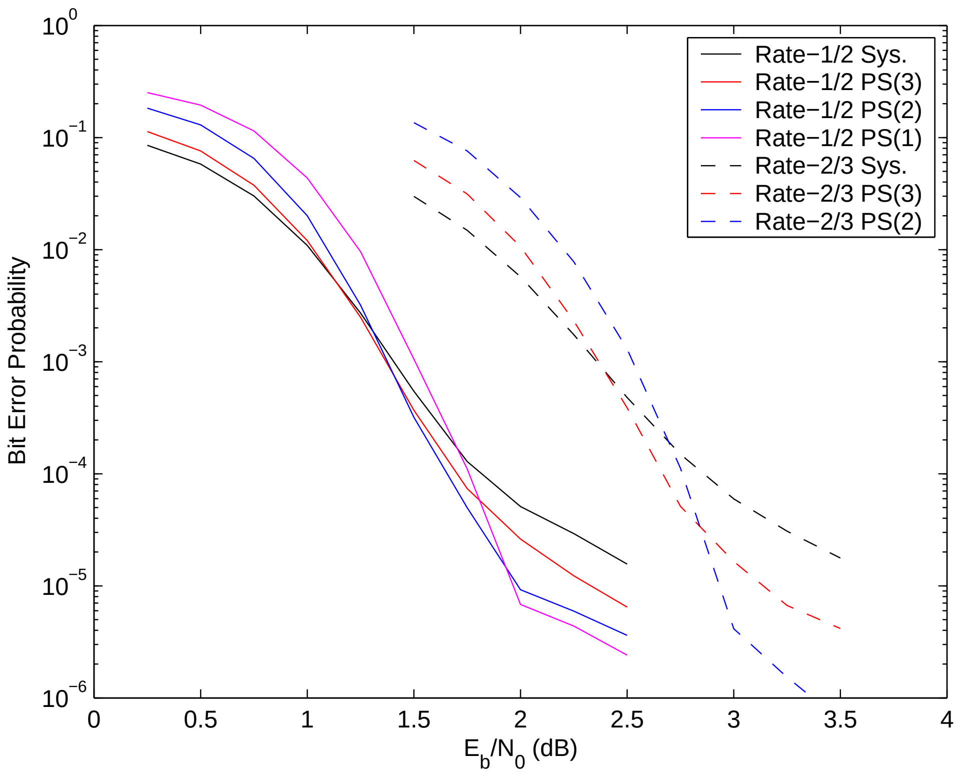


Figure 4.15: Simulation results after 10 iterations for various rate 1/2 and 2/3 configurations. An interleaver size of 1,000 bits is used.

achieves the lowest error floor, whereas “PS(2)” and “PS(3)” achieve comparable performances for an interleaver size of $N = 36$. However, the free effective distance of “PS(2)” is higher than that of “PS(3)”, thus “PS(2)” exhibits a lower error floor than “PS(3)”, for an increasing interleaver size [26]. The systematic punctured turbo code exhibits the worst error floor but the matching suboptimal iterative decoder is expected to converge quickly to it. Taking into account that the free effective distance of a turbo code does not depend on the interleaver size, a study of the performance behavior of punctured turbo codes for small interleaver sizes can lead to conclusions, which also apply to punctured turbo codes using large interleaver sizes. In Fig.4.15, we have plotted the performance of various rate-1/2 PCCC(1,5/7,5/7) codes using pseudo-random interleavers of size $N = 1,000$ bits. The rate-1/2 codes have been also obtained by puncturing the rate-1/3 PCCC(1,5/7,5/7) using the patterns provided in Table 4.1. We observe that the exact same trends, derived from Table 4.1 and presented in Fig.4.14, still apply.

An investigation for puncturing patterns having the same period, i.e., $M = 4$, but generate good rate-2/3 PCCC(1,5/7,5/7) codes was also performed. Three representative patterns, one systematic and two partially systematic, have been included in Table 4.1. As expected, conclusions drawn from theoretical results for a small interleaver size concur with simulation results for a large interleaver size, as we demonstrate in Fig.4.15. Furthermore, we observe in Fig.4.15 that, as in the previous case, puncturing the systematic sequence lowers the error floor of the turbo code at the expense of an expanded non-convergence region.

These results confirm the observation by Land and Hoeher [47] that “it is advantageous to put more puncturing to the systematic [sequence] than to the parity check [sequences]” and Blazek *et al.* [48] who claimed that “the performance benefits of the partially systematic turbo codes are mainly for higher signal-to-noise ratio values and number of iterations”.

## 4.5.2 Performance Upper Bounds

Computation of the transfer function of punctured convolutional block codes for long input sequences, and hence calculation of the transfer function of punctured turbo codes using long interleavers, can be accelerated only if the puncturing pattern does not cause the formation of loops having zero input and output weight in their augmented state diagram. Consequently, before proceeding with the derivation of the state equations, we need to examine the weights of each loop, either formed

Table 4.2: Puncturing pattern configurations that do not cause the formation of unwanted loops of zero input and output weight in the augmented state diagram of a punctured convolutional block code, for patterns of period $M = 2, 3, 4$.

| |
|---|
| **Parent Code: PCCC(1,5/7,5/7)** |
| No restrictions, for $M = 2$ and $M = 4$. |
| At least two 1's in $\mathbf{P}_{Z_1}$ and two 1's in $\mathbf{P}_{Z_2}$, for $M = 3$. |
| **Parent Code: PCCC(1,7/5,7/5)** |
| All elements in $\mathbf{P}_{Z_1}$ and $\mathbf{P}_{Z_2}$ should be equal to 1, for $M = 2$. |
| At least one 1 in $\mathbf{P}_{Z_1}$ and one 1 in $\mathbf{P}_{Z_2}$, for $M = 3$. |
| At least two consecutive 1's in $\mathbf{P}_{Z_1}$ and two consecutive 1's in $\mathbf{P}_{Z_2}$, for $M = 4$. |
| **Parent Code: PCCC(1,17/15,17/15)** |
| No restrictions, for $M = 2, 3, 4$. |

from individual branches interconnecting two or more states, or formed by a single branch which originates and terminates at the same state.

If unwanted loops are formed, the puncturing pattern used to configure the branch metrics of the augmented state diagram, is excluded from the list of patterns that allow simplification of the augmented state diagram and quick computation of the transfer function for long input sequences. Assessment of such a list of patterns leads to a number of design criteria, which are summarized in Table 4.2. For example, if we wished to obtain the transfer function of a rate-1/2 systematic PCCC(1,5/7,5/7) using a long interleaver, we could consider any pattern configuration of period $M = 4$, since there are no restrictions according to Table 4.2. However, if we used a puncturing pattern with $\mathbf{P}_U = [1111]$, $\mathbf{P}_{Z_1} = [1010]$ and $\mathbf{P}_{Z_2} = [0101]$, to obtain a rate-1/2 systematic PCCC(1,7/5,7/5), we would not be able to simplify the augmented state diagram, because the selected pattern does not comply with the design criteria. Nevertheless, we could change the puncturing vectors of the parity check sequences to $\mathbf{P}_{Z_1} = [1100]$ and $\mathbf{P}_{Z_2} = [0011]$, which are in accordance with the design rules.

A comparison between theoretical upper bounds and simulation results for the rate-1/2 systematic PCCC(1,5/7,5/7) is presented in Fig.4.16. The same comparison for a punctured turbo code having a greater memory size, namely the rate-1/2 systematic PCCC(1,17/15,17/15), is performed in Fig.4.17. In addition, the first 15

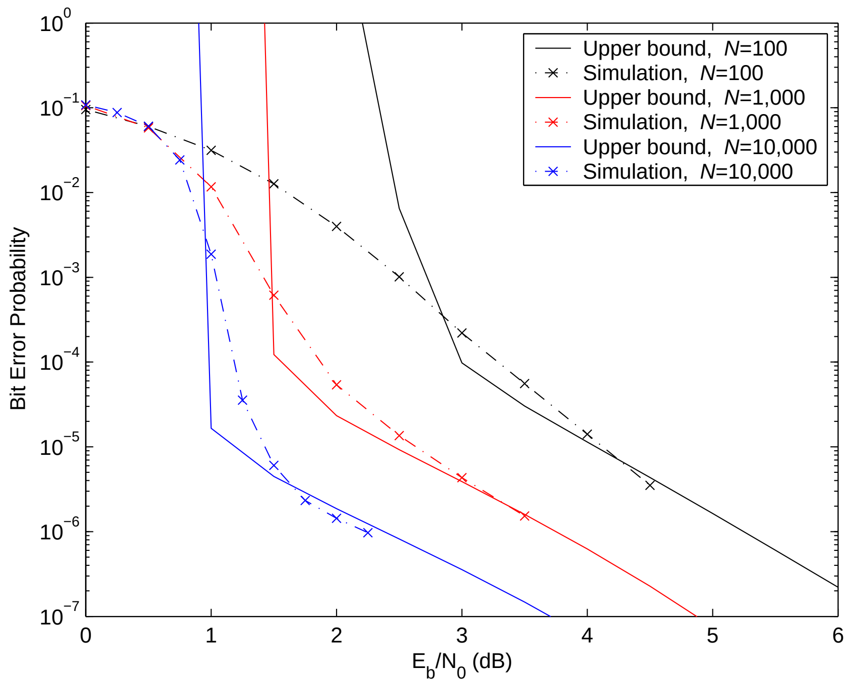


Figure 4.16: Comparison between bounds and simulation results after 8 iterations for various interleaver sizes of a rate-1/2 Sys. PCCC(1,5/7,5/7) employing the exact log-MAP decoding algorithm.

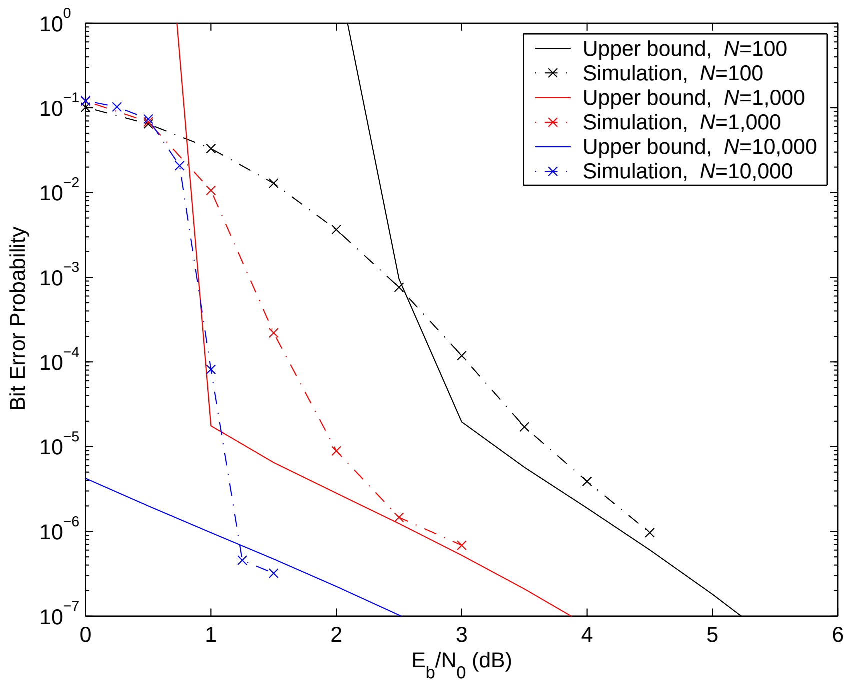


Figure 4.17: Comparison between bounds and simulation results after 8 iterations for various interleaver sizes of a rate-1/2 Sys. PCCC(1,17/15,17/15) employing the exact log-MAP decoding algorithm.

Table 4.3: Coefficients $D_d$ for the calculation of the union bound for two rate-1/2 systematic PCCCs. Three interleaver sizes are considered.

| | PCCC(1,5/7,5/7) | | | PCCC(1,17/15,17/15) | | |
|---|---|---|---|---|---|---|
| $d$ | $N=100$ | $N=1,000$ | $N=10,000$ | $N=100$ | $N=1,000$ | $N=10,000$ |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | $4.45 \cdot 10^{-4}$ | $4.49 \cdot 10^{-6}$ | $4.49 \cdot 10^{-8}$ | 0 | 0 | 0 |
| 4 | $2.35 \cdot 10^{-5}$ | $2.39 \cdot 10^{-8}$ | $2.39 \cdot 10^{-11}$ | $9.20 \cdot 10^{-5}$ | $9.56 \cdot 10^{-8}$ | $9.59 \cdot 10^{-11}$ |
| 5 | $1.11 \cdot 10^{-2}$ | $1.16 \cdot 10^{-4}$ | $1.16 \cdot 10^{-6}$ | 0 | 0 | 0 |
| 6 | $8.54 \cdot 10^{-2}$ | $8.93 \cdot 10^{-3}$ | $8.99 \cdot 10^{-4}$ | $1.13 \cdot 10^{-2}$ | $9.91 \cdot 10^{-4}$ | $9.98 \cdot 10^{-5}$ |
| 7 | $9.23 \cdot 10^{-2}$ | $1.00 \cdot 10^{-3}$ | $1.01 \cdot 10^{-5}$ | $4.84 \cdot 10^{-2}$ | $5.38 \cdot 10^{-4}$ | $5.43 \cdot 10^{-6}$ |
| 8 | 0.257 | $2.38 \cdot 10^{-2}$ | $2.39 \cdot 10^{-3}$ | $6.18 \cdot 10^{-2}$ | $3.98 \cdot 10^{-3}$ | $3.99 \cdot 10^{-4}$ |
| 9 | 0.394 | $3.99 \cdot 10^{-3}$ | $4.00 \cdot 10^{-5}$ | 0.143 | $1.55 \cdot 10^{-3}$ | $1.58 \cdot 10^{-5}$ |
| 10 | 0.995 | $4.36 \cdot 10^{-2}$ | $4.03 \cdot 10^{-3}$ | 0.259 | $8.43 \cdot 10^{-3}$ | $8.02 \cdot 10^{-4}$ |
| 11 | 2.303 | $1.92 \cdot 10^{-2}$ | $1.89 \cdot 10^{-4}$ | 0.606 | $4.58 \cdot 10^{-3}$ | $4.52 \cdot 10^{-5}$ |
| 12 | 6.178 | 0.199 | $1.795 \cdot 10^{-2}$ | 1.247 | $1.68 \cdot 10^{-2}$ | $1.37 \cdot 10^{-3}$ |
| 13 | 15.471 | 0.121 | $1.198 \cdot 10^{-3}$ | 3.134 | $1.44 \cdot 10^{-2}$ | $1.34 \cdot 10^{-4}$ |
| 14 | 37.189 | 0.820 | $7.280 \cdot 10^{-2}$ | 7.591 | $4.31 \cdot 10^{-2}$ | $2.91 \cdot 10^{-3}$ |
| 15 | 91.838 | 0.582 | $5.653 \cdot 10^{-3}$ | 18.195 | $5.37 \cdot 10^{-2}$ | $4.85 \cdot 10^{-4}$ |

coefficients $D_d$, which were derived based on our proposed method and are required for the union bound calculation of both PCCCs, as described in Chapter 3, are shown in Table 4.3. In both cases, the parity check outputs of the respective turbo encoder are punctured alternately, as indicated by the puncturing pattern of the "Rate-1/2 Sys." configuration in Table 4.1. Furthermore, no restrictions apply on patterns of period $M = 4$, according to Table 4.2. Performance of punctured systematic turbo codes, originally used by Berrou *et al.* [11] as well as Hagenauer *et al.* [19], quickly converges to the corresponding upper bound on the average ML performance, provided that puncturing is distributed equally between parity check bits and is well scattered, as suggested by Kousa and Mugaibel in [50]. As in the case of non-punctured turbo codes, we observe that suboptimal iterative decoding, employing the exact log-MAP algorithm, converges to ML decoding for medium to large $E_b/N_0$ values, after 8 iterations. Small deviations from the upper bound curve indicate that the simulated performance achieved by a punctured PCCC configuration using a selected pseudo-random interleaver is slightly better or worse than the average performance of a PCCC, which uses the same puncturing pattern and constituent codes but employs a uniform interleaver.

## 4.6 Chapter Summary

Whereas in Chapter 3 we introduced a novel approach for the evaluation of the transfer function of a convolutional block code, in this chapter we extended our approach to punctured convolutional block codes. In particular, we demonstrated how puncturing patterns can be incorporated into the structure of the augmented state diagram, in order to derive the transfer function of the corresponding punctured convolutional block code. Consequently, tight bounds on the average ML performance of high-rate turbo codes, which employ punctured convolutional block codes as constituent codes, can be derived. Nevertheless complexity becomes overwhelming as the interleaver size increases. Provided that specific conditions are met, a simplification process can take place, which considerably reduces complexity and makes computation of the transfer function feasible, even when a long input sequence or, equivalently, a long interleaver is considered. If those conditions are not met, our approach can still be used to identify puncturing patterns that lead to high-rate turbo codes, which exhibit low error floors.

Although it is not always feasible to derive an accurate expression for the average ML performance union bound of a turbo code, due to the limitations imposed by the structure of the augmented state diagram, we can derive a good approximation when long pseudo-random interleavers are used. In the following chapter, we demonstrate that the approximation technique does not rely on the transfer functions of the constituent codes, hence construction of their augmented state diagrams is not required. We then use our proposed approximation technique to show that rate-1/2 turbo codes, obtained by puncturing the output of a rate-1/3 parent turbo code, can achieve lower error floors than their parent code, provided that performance of their corresponding suboptimal iterative decoder converges towards the error floor region.

Chapter 5

# A Union Bound Approximation for Rapid Performance Evaluation

## 5.1 Introduction

The transfer function and, consequently, an upper bound on the bit error probability of a turbo code can be readily derived when short interleavers are considered. However, computation of the transfer function becomes intensive as the interleaver size increases and, although the complexity of the process can be reduced, simplification is not always possible in the case of punctured turbo codes.

In this chapter we demonstrate that as the interleaver becomes larger, codeword sequences yielding an information weight of two start to play an increasingly significant role in determining the bit error probability of a turbo code, provided that a uniform interleaver is used. We take into account this property of turbo codes and we propose an approximation of the upper bound on the bit error probability, which is accurate when long interleavers are employed. The bound approximation exploits only those terms of the transfer function that have a major impact on the overall performance. We revisit the structure of a constituent RSC encoder and we develop a rapid method to calculate the most significant terms of the transfer function of a turbo encoder. Contrary to the approach presented in the previous chapters, which is based on the augmented state diagram, this method does not impose restrictions on the code generator vectors, or the puncturing pattern. We apply our proposed method to a family of punctured codes, which we call pseudo-randomly punctured codes, and we conclude the chapter by evaluating the accuracy of the bound approximation for a variety of interleaver sizes for various non-punctured and punctured turbo codes.

## 5.2 A Simple Approximation of the Union Bound on the Bit Error Probability

In Chapter 2, we explained that the input-output weight enumerating function $B^{\mathcal{P}}(W, D)$ of a turbo code $\mathcal{P}$, obtained from the transfer function $B^{\mathcal{P}}(W, U, Z)$ by setting $U = Z = D$, enumerates every codeword sequence of information weight $w$ and codeword weight $d$, which has been incorrectly decoded due to an erroneous decoding decision. Based on the union bound argument, the probability $P_b$ of a bit error is upper bounded as follows

$$P_b \leq P_b^{\mathrm{u}} \tag{5.1}$$

where the union bound $P_b^{\mathrm{u}}$ considers all codeword sequences provided by the input-output weight enumerating function, and is defined as

$$P_b^{\mathrm{u}} = \sum_w P(w). \tag{5.2}$$

Here, the sum runs over all possible values of $w$, with $P(w)$ being the contribution to the union bound $P_b^{\mathrm{u}}$ of only those codeword sequences having a specific information weight $w$. An individual contribution $P(w)$ is given by

$$P(w) = \sum_d \frac{w}{N} B_{w,d}^{\mathcal{P}} Q\left(\sqrt{\frac{2R \cdot E_b}{N_0} \cdot d}\right), \tag{5.3}$$

where $B_{w,d}^{\mathcal{P}}$ are the coefficients of the input-output weight enumerating function, $N$ is the interleaver size and $R$ is the code rate of the turbo encoder.

In [26], Benedetto *et al.* investigated the performance of rate-1/3 turbo codes having the form of a parallel concatenation of convolutional codes separated by a uniform interleaver. The authors showed that interleaver gain can be achieved only if the constituent encoders are recursive. Furthermore, they observed that codeword sequences having the minimum possible information weight $w_{\min}$ are the main contributors to the bit error performance, as the interleaver size $N$ increases. Owing to the structure of the constituent encoders, the minimum information weight of an input sequence to a PCCC, which uses recursive constituent encoders, is always equal to two, i.e., $w_{\min} = 2$. Consequently, $P(2)$ is the dominant contribution to the union bound $P_b^{\mathrm{u}}$ on the bit error probability, when a large interleaver size is used.

The contribution of $P(2)$ to the union bound is depicted in Fig.5.1 as a percentage, where it is also compared to that of $P(3)$. Two symmetric turbo codes, PCCC(1, 5/7, 5/7) and PCCC(1, 17/15, 17/15) using uniform interleavers are considered. For both codes, $P(2)$ significantly contributes to the union bound for a narrow range of moderate bit error probabilities, when an interleaver size of $1,000$ bits is used. An increase of the interleaver size from $N=1,000$ to $N=10,000$ establishes $P(2)$ as the most significant contribution over a much broader range of bit error probabilities. More specifically, we observe in Fig.5.1(a) that the value of $P(2)$ is greater than $0.8P_b^{\mathrm{u}}$, when $P_b^{\mathrm{u}}$ assumes values between $10^{-5}$ and $10^{-8}$ and the interleaver size is $N=1,000$. An increase of the interleaver size to $N=10,000$, causes $P(2)$ to assume the same range of values for bit error probabilities ranging from $10^{-6}$ to $10^{-12}$. Furthermore, the value of $P(2)$ in Fig.5.1(b) reaches a maximum of $0.69P_b^{\mathrm{u}}$ when an interleaver size $N=1,000$ is considered, whereas the maximum value of $P(2)$ increases to $0.96P_b^{\mathrm{u}}$ if an interleaver of size $N=10,000$ is used.

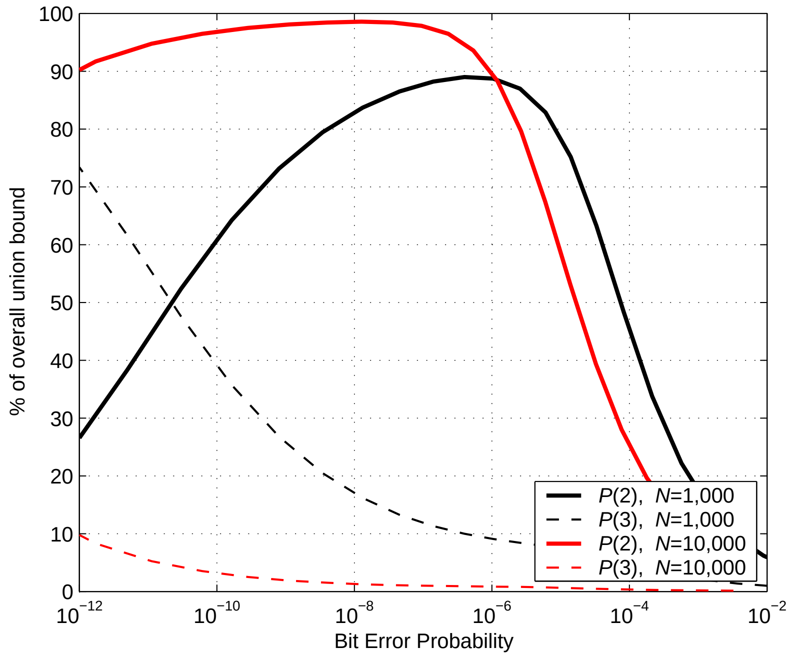
Therefore, the overall union bound on the bit error probability of a turbo code employing a long uniform interleaver, or equivalently, the overall union bound on the average bit error probability of a turbo code using a long random interleaver, can be approximated by the probability of all erroneous codeword sequences with information weight two

$$P_b^{\mathrm{u}} \approx P(2). \tag{5.4}$$

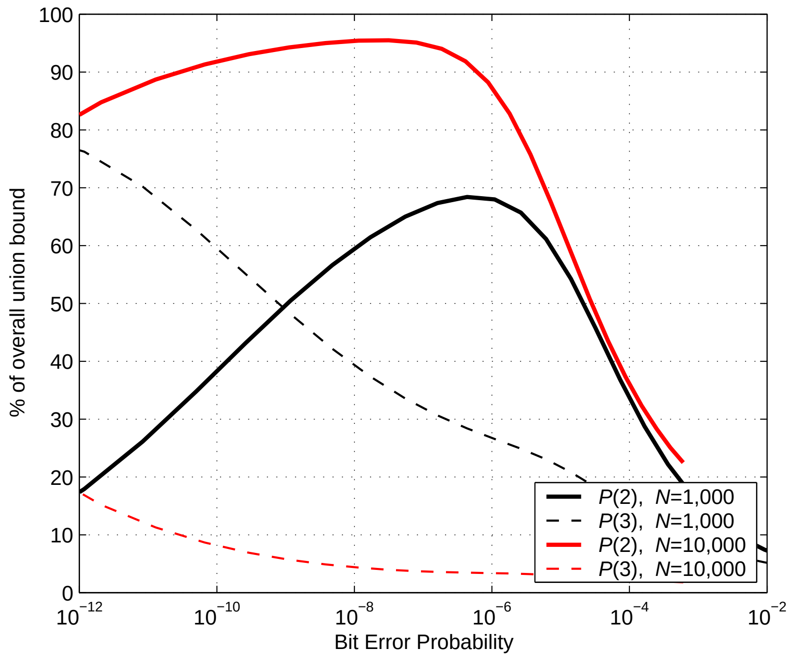
Note that the union bound is not necessarily accurate when short interleavers are used, owing to the fact that turbo codes using specific deterministic interleaver designs can achieve a superior performance than the average performance of turbo codes using random interleavers. However, when long interleavers are considered, turbo codes using random interleavers perform similarly or better than turbo codes using deterministic interleavers [51]. Hence, both the union bound $P_b^{\mathrm{u}}$ and $P(2)$, which is a close approximation of $P_b^{\mathrm{u}}$, provide a good indication of the error floor of a turbo code.

Although in certain applications, such as satellite communications, link reliability is of essence and low rate codes are used to support it, bandwidth occupancy is more important in wireless communications, therefore high rate codes are preferred. Depending on the required quality of service and the radio channel, various code rates are offered. Half rate, i.e., rate-1/2, codes are usually included in standards as a counter measure for poor channel conditions, whereas higher rates are used in better channel conditions. A half rate turbo code can be obtained by puncturing the outputs of a rate-1/3 turbo code, often referred to as parent code, as we described in Chapter 4.

Fig.5.2 shows the contribution of $P(2)$ to the union bound $P_b^{\mathrm{u}}$ on the average bit error probability of various punctured rate-1/2 turbo codes. The rate-1/3 parent codes considered are PCCC(1, 5/7, 5/7) and PCCC(1, 17/15, 17/15). Although

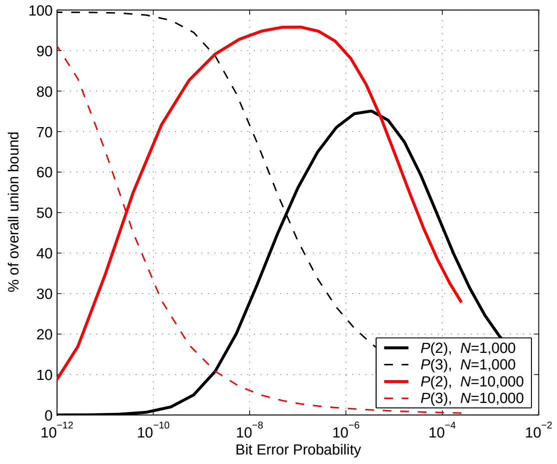


(a) Rate-1/3 PCCC(1,5/7,5/7)

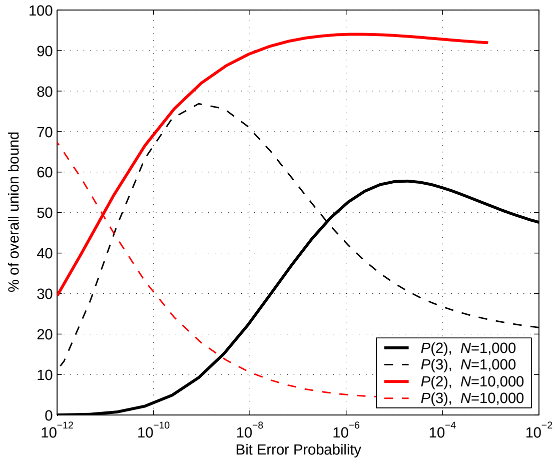


(b) Rate-1/3 PCCC(1,17/15,17/15)

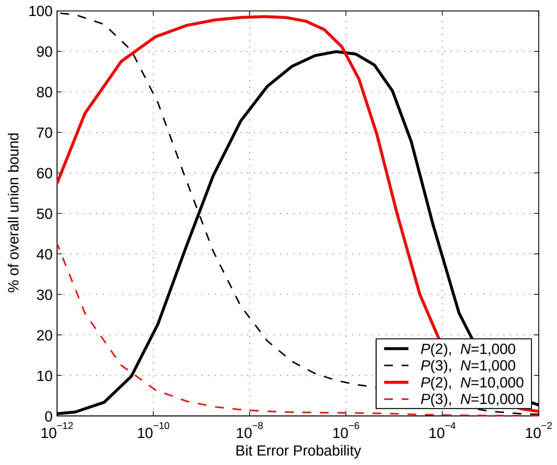
Figure 5.1: Contributions to the union bound on the bit error probability of various non-punctured turbo codes employing an interleaver of size either $N = 1,000$ or $N = 10,000$ bits.



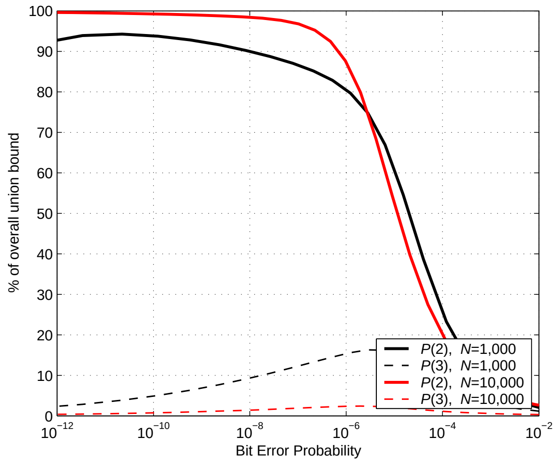
(a) Rate-1/2 NS-PCCC(1,5/7,5/7)



(b) Rate-1/2 NS-PCCC(1,17/15,17/15)



(c) Rate-1/2 Sys. PCCC(1,5/7,5/7)



(d) Rate-1/2 Sys. PCCC(1,17/15,17/15)

Figure 5.2: Contributions to the union bound on the bit error probability of various punctured turbo codes employing an interleaver of size either $N = 1,000$ or $N = 10,000$ bits.

there are a plethora of puncturing patterns leading to rate-1/2 systematic, partially-systematic and non-systematic turbo codes, we have selected two specific patterns to demonstrate that the same trend, observed previously in rate-1/3 turbo codes, is also observed in half rate punctured turbo codes. The first pattern, with vectors $\mathbf{P}_U = [00]$, $\mathbf{P}_{Z_1} = [11]$ and $\mathbf{P}_{Z_2} = [11]$ creates a rate-1/2 non-systematic turbo code, denoted as rate-1/2 NS-PCCC for brevity, whereas the second pattern, with $\mathbf{P}_U = [11]$, $\mathbf{P}_{Z_1} = [10]$ and $\mathbf{P}_{Z_2} = [01]$, is often used [11, 19] to generate rate-1/2 systematic turbo codes, which we refer to as rate-1/2 Sys. PCCCs. In all four cases, as the interleaver size increases, $P(2)$ becomes the most significant contribution over a broad range of bit error probabilities. Consequently, $P(2)$ can be used as a close approximation of the union bound $P_b^{\mathrm{u}}$ on the average bit error probability of a rate-1/2 punctured turbo code using a random interleaver, for medium to low bit error probabilities.

Concluding, $P(2)$ can be used to predict the error floor of either rate-1/2 punctured turbo codes or their rate-1/3 parent codes, hence only coefficients $B_{2,d}^{\mathcal{P}}$ need to be calculated using (5.3), which assumes the form

$$P(2) = \sum_d \frac{2}{N} B_{2,d}^{\mathcal{P}} Q\left(\sqrt{\frac{2R \cdot E_b}{N_0} \cdot d}\right), \tag{5.5}$$

for $w=2$. The coefficients $B_{2,d}^{\mathcal{P}}$ can be derived from the transfer function $B^{\mathcal{P}}(W, U, Z)$ of the turbo code, however the conditional weight enumerating function for $w=2$, denoted as $B^{\mathcal{P}}(w=2, U, Z)$ and referred to as *dominant conditional weight enumerating function*, is sufficient to extract the same necessary information. Based on the expression derived by Benedetto and Montorsi [16] that relates the conditional weight enumerating function of a turbo code $\mathcal{P}$ to the conditional weight enumerating functions of the constituent codes, $\mathcal{C}_1$ and $\mathcal{C}_2$, we obtain $B^{\mathcal{P}}(w=2, U, Z)$ as follows

$$B^{\mathcal{P}}(w=2, U, Z) = \frac{B^{\mathcal{C}_1}(w=2, U, Z) \cdot B^{\mathcal{C}_2}(w=2, U=1, Z)}{\binom{N}{2}}. \tag{5.6}$$

Therefore, in order to compute $B^{\mathcal{P}}(w=2, U, Z)$, and ultimately $P(2)$, we only need to calculate the dominant conditional weight enumerating function of each constituent convolutional block code, rather than their complete transfer function.

In Chapters 3 and 4, we demonstrated that computation of the transfer function is a computationally intensive process that involves construction of the augmented state diagram of the convolutional block code, solution of the state equations and derivation of the appropriate conditional extended transfer function. In this chapter, we exploit the structure of the convolutional encoder to devise a simple method to compute the less computationally demanding dominant conditional weight enumerating function.

## 5.3 Direct Computation of the Dominant Conditional Weight Enumerating Function of Non-Punctured Convolutional Block Codes

The dominant conditional weight enumerating function $B(w=2,U,Z)$ of a convolutional block code could be obtained by brute-force, i.e., input all valid sequences of weight two to the encoder and group the output codeword sequences according to their systematic and parity check weights. Although this approach is conceptually simple, it is extremely time-consuming, especially when a long input sequence is considered. Note that the length of the input sequence is equivalent to the size of the interleaver of a turbo code, when the convolutional block code in question is used as a constituent code.

In this section we use the properties of the trellis diagram of non-punctured recursive convolutional codes to express the dominant conditional weight enumerating function as a function of their memory size and generator vectors.

### 5.3.1 Revisiting the Structure of the RSC Encoder

The schematic of a constituent rate-1/2 non-punctured RSC(1,$\mathbf{G}_F/\mathbf{G}_R$) encoder is illustrated in Fig.5.3. As was explained previously in Chapter 2, the generator vectors $\mathbf{G}_R=[g_0^R g_1^R \ldots g_\nu^R]$ and $\mathbf{G}_F=[g_0^F g_1^F \ldots g_\nu^F]$ describe the connections between the encoder input, the output of the $\nu$ shift registers, $r_1, r_2 \ldots r_\nu$, and the modulo-2 adders. Coefficients $g_0^{(R)}$ and $g_\nu^{(R)}$ are always set to "1" for the encoder to be

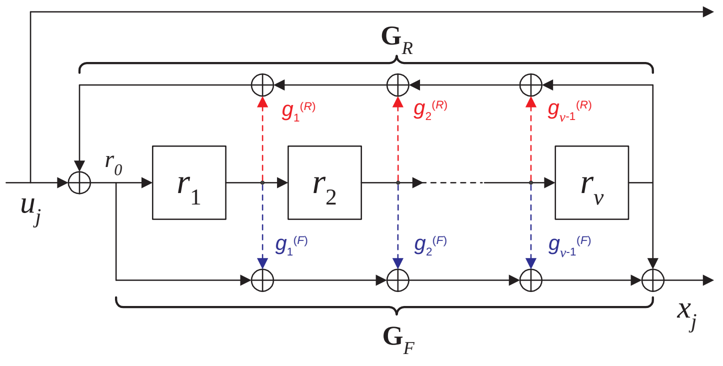


Figure 5.3: Block diagram of a rate-1/2 constituent RSC encoder.

recursive. A hypothesis commonly made [26, 33] so as to facilitate analysis of RSC codes, is that $g_0^{(F)}$ and $g_\nu^{(F)}$ are also set to "1". Furthermore, it is assumed that the encoder starts with all $\nu$ registers clear, i.e., $r_1 = r_2 = \ldots = r_\nu = 0$. For convenience, we sometimes use the decimal representation of the memory state, denoted as $s$, with $r_1$ being the most significant bit. Therefore at time step $j = 0$, the memory state of the encoder is $s_{j=0} = 0$.

At time step $j$, the input information bit to the RSC encoder is $u_j$, while the memory state is $s_j$. The input $r_0$ to the first register, depicted in Fig.5.3, assumes the value

$$r_0 = u_j \oplus (g_1^{(R)} \cdot r_1) \oplus \ldots \oplus (g_{\nu-1}^{(R)} \cdot r_{\nu-1}) \oplus r_\nu. \tag{5.7}$$

Symbols $\oplus$ and $\cdot$ denote the exclusive OR (i.e., modulo-2 addition) and the AND operation, respectively. Due to the systematic nature of the code, the output of the encoder consists of the input bit $u_j$ and a parity check bit $x_j$, given by

$$x_j = r_0 \oplus (g_1^{(F)} \cdot r_1) \oplus \ldots \oplus (g_{\nu-1}^{(F)} \cdot r_{\nu-1}) \oplus r_\nu, \tag{5.8}$$

based on the structure of the encoder in Fig.5.3. Substituting (5.7) into (5.8) and invoking boolean algebra, we obtain

$$x_j = u_j \oplus \left[ (g_1^{(F)} \oplus g_1^{(R)}) \cdot r_1 \right] \oplus \ldots \oplus \left[ (g_{\nu-1}^{(F)} \oplus g_{\nu-1}^{(R)}) \cdot r_{\nu-1} \right]. \tag{5.9}$$

At the end of time step $j$, the content of each register has been shifted to the successive register, i.e.,

$$r_1 \leftarrow r_0, \quad r_2 \leftarrow r_1, \quad \ldots, \quad r_\nu \leftarrow r_{\nu-1}, \tag{5.10}$$

and subsequently, the state of the encoder has changed from $s_j$ to $s_{j+1}$.

### 5.3.2 Properties of Information Weight-2 Codeword Sequences

Input sequences having the minimum information weight, which is two when recursive encoders are considered [26], correspond to paths in the trellis diagram that diverge from the zero state and re-merge with it only once, after a number of time-steps.

The zeroes preceding the two non-zero bits in an information sequence of weight two, do not change the state of the encoder, according to (5.7) and (5.10). For convenience, we assume that at time step $j = 0$, the first non-zero bit $u_0 = 1$ is input to the encoder. Based on (5.7), (5.9) and (5.10), we find that the input $r_0$ to the first register becomes "1", parity check bit $x_0 = 1$ is produced and the memory state

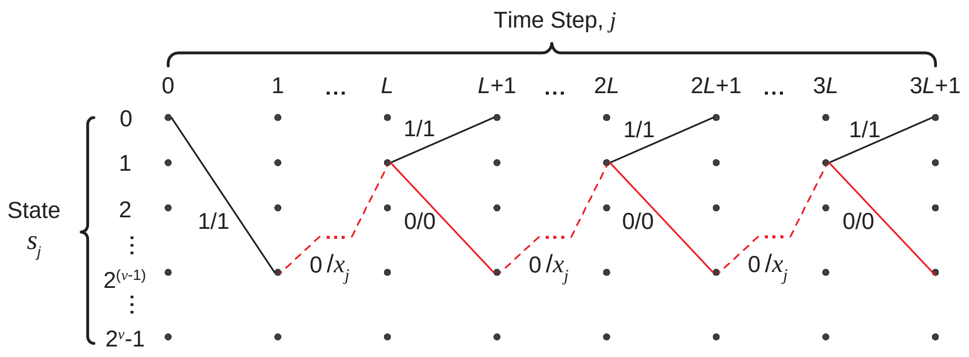


Figure 5.4: Trellis diagram for codeword sequences of information weight 2. The paths plotted in red are generated when the rate-1/2 RSC encoder operates as a pseudo-random generator. A pair $u_j/x_j$ next to a branch originating from a state at time step $j$, corresponds to the input bit $u_j$ at time $j$ and the output parity check bit $x_j$ generated at the end of the state transition.

changes from $s_0 = 0$ to $s_1 = 2^{\nu-1}$, as is illustrated in the trellis diagram of Fig.5.4. We understand from (5.7) that the zeroes following the first non-zero input bit do not affect the value of $r_0$, which depends solely on the feedback generator vector $\mathbf{G}_R$ and the bits stored in the shift registers. Therefore, for as long as a trail of zeroes follows the first non-zero input bit, the RSC encoder behaves like a *pseudo-random generator.*

A wealth of papers address the topic of pseudo-random sequences and pseudo-random generators. In 1976, MacWilliams and Sloane wrote a primer [52], describing their properties in detail. More specifically, it is stated that a pseudo-random generator can periodically revisit all distinct states except for the zero state. The period, denoted as $L$ depends on the feedback vector $\mathbf{G}_R$ and it is $L \leq 2^\nu - 1$. Thus, the path of the information weight-2 codeword sequence, depicted in red in the trellis diagram shown in Fig.5.4, will not re-merge with the zero state, for as long as zeroes follow the first non-zero bit.

Due to the memory state periodicity, the initial state of the pseudo-random generator, $s_1 = 2^{\nu-1}$, is also repeated every $L$ time-steps, i.e., $s_{kL+1} = s_1$, where $k$ is a positive integer. When the encoder is in state $s_{kL+1} = 2^{\nu-1}$, the first register is set, i.e., $r_1 = 1$, while all other registers are clear, i.e., $r_2 = \ldots = r_\nu = 0$. Bearing in mind that the input bit is zero, the bits stored in the shift registers at the immediately preceding time step should have been $r_1 = \ldots = r_{\nu-1} = 0$ and $r_\nu = 1$, according to (5.7) and (5.10), independently of the feedback generator vector $\mathbf{G}_R$. Therefore, the state previous to $s_{kL+1} = 2^{\nu-1}$ is always $s_{kL} = 1$, whereas the generated parity check bit is $x_{kL}=0$, as we observe in Fig.5.4.

The RSC encoder stops behaving like a pseudo-random generator, going through non-zero states, only when the second non-zero bit of the weight-2 information sequence is input to the encoder, forcing a return to the zero state. To achieve this, the second non-zero bit can be input to the encoder only when a particular state $s_j$, that precedes the zero state $s_{j+1}=0$, is reached. Using (5.7) and (5.10), we can show that when $u_j=1$, the state of the encoder goes to zero only if the preceding state is $s_j=1$. Only then is the logical "1", stored in $r_\nu$, canceled out with the input bit giving $r_0=0$ and, consequently, forcing the encoder to return to the zero state. We have already demonstrated that $s_j=1$, when $j=kL$. Therefore, when the second non-zero bit $u_{kL}=1$ is input to the encoder, parity check bit $x_{kL}=1$ is generated while the memory state changes to $s_{kL+1}=0$, as depicted in Fig.5.4.

Summarizing, weight-2 information sequences, which are input to an RSC encoder with generator vectors $\mathbf{G}_F$, $\mathbf{G}_R$ and memory size $\nu$, form paths in the trellis diagram that diverge from the all-zero path for $kL+1$ consecutive time-steps, where $k$ is a positive integer and $L \leq 2^\nu - 1$ is a constant, dependent on the feedback vector $\mathbf{G}_R$. For input sequences of length $N$, we can confine the value of $k$ between 1 and $\lfloor (N-1)/L \rfloor$, where $\lfloor \xi \rfloor$ denotes the integer part of $\xi$. Furthermore, the first non-zero information bit in the input sequence forces the path to leave the zero state, whereas the second non-zero information bit forces the path to return to the zero state, $kL+1$ time steps later. In both cases, the encoder outputs a non-zero parity check bit.

### 5.3.3 Enumeration of Information Weight-2 Codeword Sequences

When the outputs of an RSC encoder are not punctured, a systematic sequence of weight $u$ is identical to the input information sequence of weight $w$, hence $u=w$. We denote as $u_{\min}$ and $z_{\min}$ the minimum weight of the systematic and parity check output sequences, respectively, when a weight-2 information sequence is input to the RSC encoder. In this case $w=2$, therefore the weight of a systematic sequence is always $u=u_{\min}=2$.

The weight $z$ of a parity check sequence is at least 2, i.e., $z \geq 2$, since non-zero parity check bits are generated during the state transitions of the encoder from 0 to $2^{\nu-1}$ and from 1 back to 0, as we explained in the previous subsection. If the trellis path of the sequence diverges from the all-zero path for $kL+1$ consecutive time-steps, the state of the encoder transitions $k$ times from state $2^{\nu-1}$ to 1, as

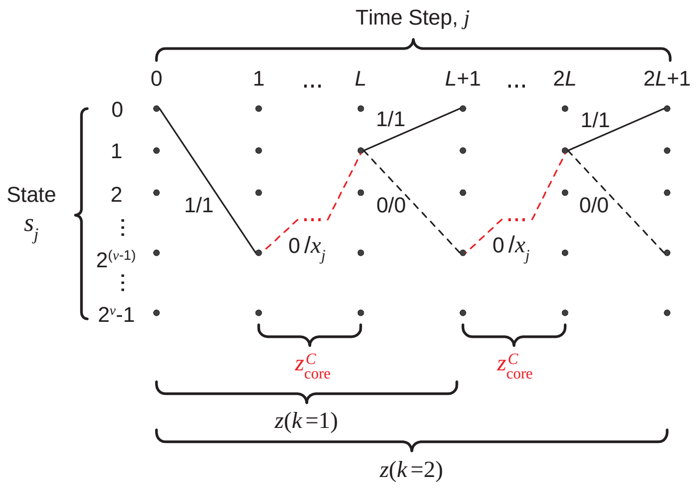


Figure 5.5: Weight calculation for a parity check sequence, generated by a weight-2 input information sequence.

is illustrated in Fig.5.5. Each time, the generated stream of parity check bits has the same weight, which we call the *core parity check weight* and denote as $z_{\text{core}}$. Consequently, the overall weight $z$ of a parity check sequence can be expressed as a function of $k$, i.e., $z(k)$, given by

$$z(k) = kz_{\text{core}} + 2, \quad \text{for } k = 1, 2, \ldots, \lfloor (N-1)/L \rfloor. \tag{5.11}$$

The minimum parity check weight $z_{\text{min}}$ can be derived from $z(k)$ by setting $k = 1$, hence

$$z_{\text{min}} = z(1) = z_{\text{core}} + 2. \tag{5.12}$$

The overall minimum weight of the output codeword sequence, denoted as $d_{\text{min}}$, is also a useful metric when the RSC encoder is used in a parallel concatenation scheme, since it affects its free effective distance. It is defined as

$$d_{\text{min}} = \min_{w=2}\{u + z\}, \tag{5.13}$$

which in the case of non-punctured RSC codes assumes the form

$$\begin{aligned} d_{\text{min}} &= u_{\text{min}} + z_{\text{min}} \\ &= z_{\text{core}} + 4. \end{aligned} \tag{5.14}$$

The objective of analyzing the structure of RSC encoders was to identify the properties of codeword sequences generated by weight-2 information sequences and facilitate the computation of the dominant conditional weight enumerating function, $B(w = 2, U, Z)$. As we have already seen, any codeword sequence can be written as a monomial, which assumes the form $W^2U^2Z^z$ when an information weight 2 is considered. We have demonstrated that the trellis path of such a codeword sequence diverges from the all-zero path for $k\mathsf{L}+1$ consecutive time steps and that its parity check weight is a function of $k$, hence its monomial can be written as $W^2U^2Z^{z(k)}$, for a given $k$. If the codeword sequence of parity check weight $z(k)$ is generated by an input information sequence of length $N$, the length of the corresponding trellis path will also be $N$. However, depending on the positions of the two non-zero bits in the input sequence, the trellis path could diverge from the all-zero path at *any* time step and re-merge with it $k\mathsf{L}+1$ time steps later. In particular, there are $N-(k\mathsf{L}+1)+1$ time steps when the path could diverge, thus there are an equal number of codeword sequences having the same parity check weight $z(k)$, for a given $k$.

The dominant conditional weight enumerating function $B(w=2, U, Z)$ is the sum of all codeword sequences having parity check weight $z(k)$, over all valid values of $k$, hence

$$B(w=2, U, Z) = \sum_{k=1}^{\lfloor (N-1)/\mathsf{L} \rfloor} (N - k\mathsf{L})\, U^2 Z^{z(k)}. \tag{5.15}$$

If we substitute $z(k)$ into the above expression, using (5.11), we obtain

$$B(w=2, U, Z) = \sum_{k=1}^{\lfloor (N-1)/\mathsf{L} \rfloor} (N - k\mathsf{L})\, U^2 Z^{k z_{\text{core}}+2}. \tag{5.16}$$

The values of both $\mathsf{L}$ and $z_{\text{core}}$ depend upon the selection of the feedback generator vector $\mathbf{G}_R$ and can be determined numerically, by initializing the state of the encoder to $2^{\nu-1}$ and inserting a zero information bit at each time step, until the same state is reached again. The number of time steps between the two states determines $\mathsf{L}$, whereas the weight of the generated parity check stream determines $z_{\text{core}}$.

### 5.3.4 The Benefit of Primitive Feedback Vectors

The feedback vector $\mathbf{G}_R$ can be defined in such a way that the RSC encoder visits all possible $2^\nu - 1$ non-zero states during a period $\mathsf{L}$. Inevitably, the period $\mathsf{L}$ will reach its maximum value of $2^\nu - 1$ time steps, i.e., $\mathsf{L} = 2^\nu - 1$. As Perez *et al.* point out in [53], maximization of $\mathsf{L}$ increases the path length of the shortest weight-2 information sequence, therefore increasing the chance of generating a codeword sequence that achieves a high minimum parity check weight $z_{\min}$. When such RSC codes are used in parallel concatenated turbo schemes, their free effective distance is maximized [26], hence their error floor is lowered.

A generator vector, in our case $\mathbf{G}_R = [g_0^{(R)} g_1^{(R)} \ldots g_\nu^{(R)}]$, of an RSC encoder having memory size $\nu$ can also be expressed as a polynomial $G_R(D)$, known as a *generator polynomial*, of the form

$$G_R(D) = 1 \oplus (g_1^{(R)} \cdot D) \oplus (g_2^{(R)} \cdot D^2) \oplus \ldots \oplus (g_{\nu-1}^{(R)} \cdot D^{\nu-1}) \oplus D^\nu, \tag{5.17}$$

since $g_0^{(R)} = g_\nu^{(R)} = 1$, owing to the recursive nature of the encoder. Addition and multiplication have been replaced by modulo-2 addition and the AND operation, respectively. The *degree* or *order* of the polynomial is the largest power of $D$, which in our case is always equal to the memory size $\nu$.

A polynomial, such as $G_R(D)$, is said to be *irreducible* if it cannot be factorized into first order polynomials, otherwise it is said to be *reducible*. In the following examples,

| | |
|---|---|
| $D^2 \oplus 1$ | is a reducible polynomial, since $D^2 \oplus 1 = (D \oplus 1) \cdot (D \oplus 1)$. |
| $D^2 \oplus D \oplus 1$ | is an irreducible polynomial, since it cannot be factorized. |

All irreducible polynomials of degree $\nu$ divide into $D^\varrho \oplus 1$ exactly, for a value of $\varrho$ lying in the range $1 \leq \varrho \leq 2^\nu - 1$. If $\varrho = 2^\nu - 1$, the irreducible polynomial is called *primitive*. For $\nu = 4$ we find that $\varrho = 2^4 - 1 = 15$, thus a primitive polynomial of degree $\nu = 4$ would only divide into $D^{15} \oplus 1$ without remainder. In the following examples,

| | |
|---|---|
| $D^4 \oplus D^3 \oplus D^2 \oplus D \oplus 1$ | divides into $D^5 \oplus 1$ as well. Hence, it is not a primitive polynomial. |
| $D^4 \oplus D \oplus 1$ | only divides into $D^{15} \oplus 1$, thus it is a primitive polynomial. |

If the feedback generator polynomial $G_R(D)$ of an RSC encoder is selected to be primitive, the encoder revisits all possible $2^\nu - 1$ non-zero states when it operates as a pseudo-random generator, during a period of

$$L = 2^\nu - 1 \tag{5.18}$$

time steps [52]. During this period, register $r_\nu$ has generated a sequence of $2^\nu$–1 bits, known as a *pseudo-random* sequence or *pseudo-noise* (PN) sequence [52]. Effectively, every register $r_i$, where $i = 1, \ldots, \nu$–1, outputs a copy of the PN sequence generated by $r_\nu$, cyclicly shifted to the left by $\nu$–$i$ positions. A cyclic shift of a PN sequence is also a PN sequence [52], hence all $\nu$ registers generate PN sequences of period $2^\nu$–1. According to the modulo-2 addition property of PN sequences, the sum of two or more PN sequences is another PN sequence. Therefore, each input bit $r_0$ to the first register, as well as each output parity check bit $x_j$, also form PN sequences during a period of $2^\nu$–1 time steps, based on (5.7) and (5.9). Consequently, a parity check stream generated during the transition from state $2^{\nu-1}$ at time step $j=(k-1)L+1$ to state $2^{\nu-1}$ at time step $j=kL+1$, as shown in Fig.5.5, is a PN sequence consisting of $2^{\nu}-1$ bits. Out of these $2^{\nu}-1$ bits, $2^{\nu-1}$ are non-zero, according to another property of PN sequences quoted in [52]. Hence, the weight of the parity check stream is $2^{\nu-1}$. However, the last bit of the parity stream, generated during the transition from state 1 at time step $j=kL$ to state $2^{\nu-1}$ at time step $j=kL+1$, is always zero, as we demonstrated previously. For this reason, the weight of the first $2^{\nu}-2$ bits of the parity check stream, which we called core parity check weight $z_{\text{core}}$, is also

$$z_{\text{core}} = 2^{\nu-1}. \tag{5.19}$$

Note that this outcome is valid only if the two generator polynomials (or vectors) are different, i.e., $G_F(D) \neq G_R(D)$. Otherwise, $x_j$ is always zero when $u_j=0$, according to (5.9), thus $z_{\text{core}}=0$.

Substituting (5.19) into (5.12), (5.14) and (5.16), we find that when a primitive feedback generator polynomial $G_R(D)$ is selected and a feedforward generator polynomial $G_F(D)$ different than $G_R(D)$ is used, the minimum parity check weight $z_{\min}$ of the corresponding RSC encoder is given by

$$z_{\min} = 2^{\nu-1} + 2, \tag{5.20}$$

the overall minimum weight $d_{\min}$ assumes the form

$$d_{\min} = 2^{\nu-1} + 4, \tag{5.21}$$

and the dominant conditional weight enumerating function $B(w=2,U,Z)$ is expressed as

$$B(w=2,U,Z) = \sum_{k=1}^{\lfloor (N-1)/(2^{\nu}-1) \rfloor} \left[N - k(2^{\nu}-1)\right] U^2 Z^{k2^{\nu-1}+2}. \tag{5.22}$$

We conclude that the minimum weights, $z_{\min}$ and $d_{\min}$, as well as the dominant conditional weight enumerating function $B(w=2,U,Z)$ depend upon the memory size of the RSC encoder and not the underlying code, when the feedforward polynomial of the RSC encoder is different from the primitive feedback polynomial. A list of primitive feedback polynomials for various memory sizes [4], represented as binary vectors or octal numbers, the period $L$ and the minimum weights $z_{\min}$ and $d_{\min}$ achieved by the RSC encoder, is provided in Table 5.1.

Table 5.1: Primitive feedback polynomials for various memory sizes [4].

| Memory size, $\nu$ | Binary $\mathbf{G}_R$ | Octal $\mathbf{G}_R$ | $L$ | $z_{\min}$ | $d_{\min}$ |
|---|---|---|---|---|---|
| 1 | [1 1] | 3 | 0 | 3 | 5 |
| 2 | [1 1 1] | 7 | 3 | 4 | 6 |
| 3 | [1 0 1 1]<br>[1 1 0 1] | 13<br>15 | 7 | 6 | 8 |
| 4 | [1 0 0 1 1]<br>[1 1 0 0 1] | 23<br>31 | 15 | 10 | 12 |
| 5 | [1 0 0 1 0 1]<br>[1 0 1 0 0 1]<br>[1 0 1 1 1 1]<br>[1 1 0 1 1 1]<br>[1 1 1 0 1 1]<br>[1 1 1 1 0 1] | 45<br>51<br>57<br>67<br>73<br>75 | 31 | 18 | 20 |

## 5.4 Direct Computation of the Dominant Conditional Weight Enumerating Function of Punctured Convolutional Block Codes

Rates higher than 1/2 can be achieved using a $2 \times M$ puncturing pattern $\mathbf{P}$ on a parent rate-1/2 RSC encoder. At a time step $j$, the systematic and parity check output bits of the punctured RSC encoder will be $u_j \cdot p_{1,m}$ and $x_j \cdot p_{2,m}$, respectively, where $u_j$, $x_j$ are the output bits of the parent rate-1/2 encoder and $p_{1,m}$, $p_{2,m}$ are the elements of the $m$-th column of the puncturing pattern $\mathbf{P}$, where $1 \leq m \leq M$. The elements of pattern $\mathbf{P}$ are circularly repeated every $M$ time steps, in such a way that $p_{1,m} = p_{1,(m+jM)}$ and $p_{2,m} = p_{2,(m+jM)}$. Hence, the active puncturing column $m$ at time step $j$ can be found from

$$m = \mathrm{rem}(j+1, M), \tag{5.23}$$

where $\mathrm{rem}(j+1, M)$ denotes the remainder of the division of $(j+1)$ by $M$.

In order to compute the dominant conditional weight enumerating function $B(w=2, U, Z)$ of the punctured RSC encoder, we need to express the weights, $u(k,m)$ and $z(k,m)$, of the two output sequences of the punctured RSC encoder, as a function of the puncturing elements, $p_{1,m}$ and $p_{2,m}$, and the output bits $u_j$ and $x_j$ of the parent RSC encoder. Although input sequences with information weight $w=2$ generate paths of length $kL+1$, we first consider paths of length $L+1$, i.e.,

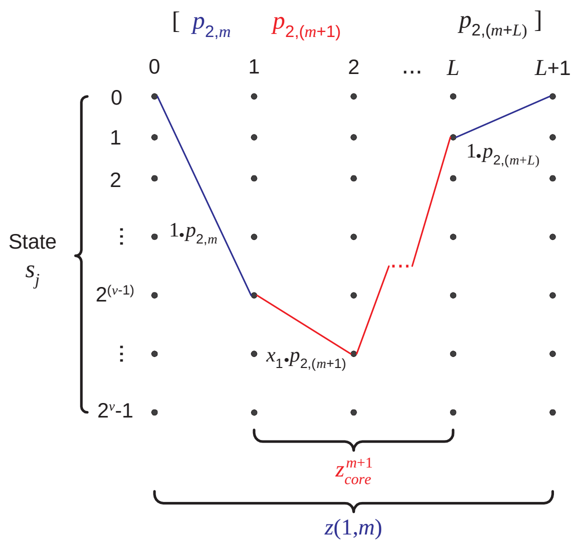


Figure 5.6: Trellis diagram for the parity check weight calculation of a punctured RSC code ($k=1$).

$k=1$, for simplicity. The systematic weight $u(k=1,m)$ of a codeword sequence, whose path diverges from the zero state when $p_{1,m}$ is active, is given by

$$u(k=1,m) = \sum_{j=0}^{L} \left(u_j \cdot p_{1,m+j}\right), \tag{5.24}$$

which is reduced to

$$u(k=1,m) = p_{1,m} + p_{1,(m+L)}, \tag{5.25}$$

since the two only non-zero bits occur at the very beginning and at the very end of the path, i.e., $u_j=1$ for $j=0$ and $j=L$, otherwise $u_j=0$.

Similarly, the weight $z(k=1,m)$ of the parity check sequence, whose path diverges from the zero state when $p_{2,m}$ is active, assumes the form

$$z(k=1,m) = \sum_{j=0}^{L} \left(x_j \cdot p_{2,m+j}\right). \tag{5.26}$$

Although the core parity check weight $z_{\text{core}}$ has a fixed value when a non-punctured code is considered, a puncturing pattern of period $M$ creates $M$ variants of $z_{\text{core}}$, denoted as $z_{\text{core}}^1, \ldots, z_{\text{core}}^m, \ldots, z_{\text{core}}^M$, where index $m$ indicates that the $m$-th column of the puncturing pattern is active at time step $j=1$, when the RSC encoder starts behaving as a pseudo-random generator. Consequently, the weight $z(k=1,m)$ of the parity check sequence can be written as

$$z(k=1,m) = p_{2,m} + z_{\text{core}}^{m+1} + p_{2,(m+L)}, \tag{5.27}$$

since the parity check bits, $x_0$ and $x_L$, at the beginning and at the end of the path respectively, are non-zero, as illustrated in Fig.5.6. In order to calculate $z(k=1,m)$ for every value of $m$, we need to first derive the $M$ variants of $z_{\text{core}}$ by applying the $M$ circularly shifted versions of the puncturing row vector $\mathbf{P}_Z=[p_{2,1}\,\ldots\,p_{2,M}]$ to the corresponding output parity check bits of the parent rate-1/2 RSC encoder, i.e,

$$z_{\text{core}}^{m}=\sum_{j=1}^{L-1}\left(x_j\cdot p_{2,(m+j-1)}\right). \tag{5.28}$$

If we extend our analysis to codeword sequences associated with paths of length $kL+1$, we need to include the variable $k$ into (5.25) and (5.27) to obtain the generic expressions for $u(k,m)$ and $z(k,m)$. The systematic weight $u(k,m)$ of a punctured codeword sequence still depends on the systematic puncturing elements, which are active at the very beginning and the very end of the input sequence, hence

$$u(k,m)=p_{1,m}+p_{1,(m+kL)}. \tag{5.29}$$

On the other hand, in order to compute the parity check weight $z(k,m)$ of a punctured codeword sequence corresponding to a path of length $kL+1$, we need to take into account the core weights $z_{\text{core}}^{m+1}$, $z_{\text{core}}^{m+L+1}$, ..., $z_{\text{core}}^{m+(k-1)L+1}$ of the $k$ consecutive parity check streams composing the parity check output sequence. Therefore, $z(k,m)$ can be expressed as

$$z(k,m)=p_{2,m}+\sum_{j=0}^{k-1}z_{\text{core}}^{m+jL+1}+p_{2,(m+kL)}, \tag{5.30}$$

where $z_{\text{core}}^{m+jM}=z_{\text{core}}^{m}$, due to the periodicity of the puncturing pattern.

The minimum weight $u_{\min}$ of the punctured systematic output sequence can be found numerically by storing the $M$ values of $u(k,m)$, for $k=1$ and $m=1,\ldots,M$, comparing them and selecting the minimum value, i.e.,

$$u_{\min}=\min_{m=1\ldots M}\{u(k=1,m)\}. \tag{5.31}$$

The minimum weight $z_{\min}$ of the punctured parity check output sequence is found in a similar manner

$$z_{\min}=\min_{m=1\ldots M}\{z(k=1,m)\}, \tag{5.32}$$

while the overall minimum weight $d_{\min}$ of the output codeword sequence is the minimum of the sum, $u(k=1,m)+z(k=1,m)$, for all possible values of $m$, thus

$$d_{\min}=\min_{m=1\ldots M}\{u(k=1,m)+z(k=1,m)\}. \tag{5.33}$$

Before proceeding with the calculation of the dominant conditional weight enumerating function, we first revisit the properties of information weight-2 codeword sequences, described in detail in the previous section. The trellis path of an information weight-2 codeword sequence, generated by a parent rate-1/2 RSC encoder and expressed as $W^2U^2Z^{z(k)}$, diverges from the all-zero path for $k\mathsf{L}+1$ consecutive time steps, where $\mathsf{L}$ is the period of the feedback generator polynomial. If an information sequence of length $N$ is input to the encoder, the length of the corresponding trellis path will also be $N$, thus $k\mathsf{L}+1 \leq N$, from which we find that the values of $k$ lie in the range from 1 to $\lfloor (N-1)/\mathsf{L} \rfloor$. Furthermore, we have shown that there are $N{-}k\mathsf{L}$ codeword sequences $W^2U^2Z^{z(k)}$ sharing the same parity check weight $z(k)$, for a given $k$.

When a puncturing pattern of period $M$ is used to increase the rate of the code, each codeword sequence of the form $W^2U^2Z^{z(k)}$ becomes $W^2U^{u(k,m)}Z^{z(k,m)}$ after puncturing, where $m$ is the column of the puncturing pattern, which is active when the trellis path of the codeword sequence diverges from the all-zero path. The value of $m$ lies in the range from 1 to $M$. For a given $k$, all the original $N-k\mathsf{L}$ non-punctured codeword sequences, having parity check weight $z(k)$, are mapped to an equal number of punctured codeword sequences. Owing to the puncturing period $M$, the $N{-}k\mathsf{L}$ punctured codeword sequences can be divided into $M$ groups. The group $m$ comprises of codeword sequences having the same weights $u(k,m)$ and $z(k,m)$. The number of sequences in group $m$, denoted as $B_{k,m}$, is either $\lfloor (N-k\mathsf{L})/M \rfloor$ or $\lfloor (N-k\mathsf{L})/M \rfloor + 1$, depending on whether the remainder of the division $(N-k\mathsf{L})/M$ is less than $m$ or not. As expected, the sum of $B_{k,m}$ over all values of $m$ is $N{-}k\mathsf{L}$, i.e.,

$$\sum_{m=1}^{M} B_{k,m} = N{-}k\mathsf{L}. \tag{5.34}$$

For example, consider a weight-2 information sequence of length $N=8$, input to a rate-1/2 RSC(1,5/7) encoder with memory size $\nu=2$. A primitive feedback polynomial is used, hence a period of $\mathsf{L}=2^2-1=3$ time-steps is achieved. The output of the encoder is punctured using a pattern of period $M{=}3$. For simplicity, we consider codeword sequences of length $k\mathsf{L}+1=4$, i.e., we assume that $k=1$. Therefore, there are $N-k\mathsf{L}=5$ punctured codeword sequences having systematic weight $u(1,m)$ and parity check weight $z(1,m)$, for $m{=}1,2,3$, as it is depicted in Fig.5.7. Owing to the three different values of $m$, the 5 codeword sequences can be divided into 3 groups, each one of which contains $B_{1,m}$ codeword sequences having the same weights $u(1,m)$ and $z(1,m)$, for a given $m$. We observe that the quotient and the remainder of the division $(N-k\mathsf{L})/M = 5/3$ are 1 and 2, respectively,

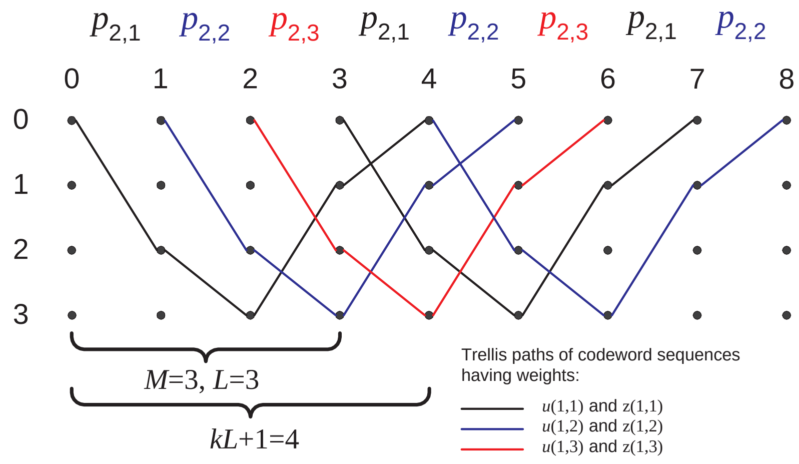

$p_{2,1}$ $p_{2,2}$ $p_{2,3}$ $p_{2,1}$ $p_{2,2}$ $p_{2,3}$ $p_{2,1}$ $p_{2,2}$
0 1 2 3 4 5 6 7 8
0
1
2
3
$M$=3, $L$=3
$kL$+1=4
Trellis paths of codeword sequences having weights:
$u$(1,1) and z(1,1)
$u$(1,2) and z(1,2)
$u$(1,3) and z(1,3)

Figure 5.7: Enumeration of information weight-2 codeword sequences having systematic weight $u(1,m)$ and parity check weight $z(1,m)$.

therefore $B_{1,m}\!=\!1$ if $m > 2$, otherwise it is $B_{1,m}\!=\!2$. We see in Fig.5.7 that, indeed, there are two codeword sequences having weights $u(1,1)$, $z(1,1)$, two codeword sequences having weights $u(1,2)$, $z(1,2)$ and only one codeword sequence having weights $u(1,3)$, $z(1,3)$.

From the above, we conclude that a group of information weight-2 puncturing codeword sequences sharing the same weights $u(k,m)$ and $z(k,m)$ consists of $B_{k,m}$ members, where $B_{k,m}$ is given by

$$B_{k,m} = \begin{cases} \left\lfloor \frac{N-kL}{M} \right\rfloor, & \text{if rem}\left((N-kL),M\right) < m \\ \left\lfloor \frac{N-kL}{M} \right\rfloor + 1, & \text{otherwise.} \end{cases} \tag{5.35}$$

The dominant conditional weight enumerating function $B(w\!=\!2,U,Z)$ needs to take into account the $B_{k,m}$ codeword sequences $W^2U^{u(k,m)}Z^{z(k,m)}$ in every group, for all possible values of $k$ and $m$. Consequently, $B(w\!=\!2,U,Z)$ can be written as

$$B(w\!=\!2,U,Z) = \sum_{k=1}^{\lfloor (N-1)/L \rfloor} \sum_{m=1}^{M} B_{k,m} U^{u(k,m)} Z^{z(k,m)}, \tag{5.36}$$

where $u(k,m)$, $z(k,m)$ and $B_{k,m}$ were defined in (5.29), (5.30) and (5.35), respectively.

## 5.5 Case Study: Pseudo-random Puncturing

In this section we study RSC encoders employing primitive feedback polynomials, therefore the period $L$ assumes the maximum value of $2^{\nu}\!-\!1$. Furthermore, we assume that the parity check output of an RSC encoder is pseudo-randomly punctured, i.e., the elements of the puncturing row vector $\mathbf{P}_Z = [p_{2,1} \ \ldots \ p_{2,M}]$ form a PN sequence, which has been generated by the same primitive polynomial employed by the RSC encoder. Consequently, the puncturing period of $\mathbf{P}_Z$ is $M = 2^\nu - 1$. Periods $M$ and $L$ can be used interchangeably, since $M = L$.

Expression (5.29) is reduced to

$$u(k,m) = u(m) = 2p_{1,m}, \tag{5.37}$$

since $p_{1,m+kM} = p_{1,m}$. Similarly, we can write (5.30) as follows

$$z(k,m) = kz_{\text{core}}^{m+1} + 2p_{2,m}, \tag{5.38}$$

since $z_{\text{core}}^{m+jL} = z_{\text{core}}^{m}$ and $p_{2,m+kM} = p_{2,m}$, due to the periodicity of the puncturing pattern. Computation of $z(k,m)$ and, consequently, $B(w=2,U,Z)$, requires numerical calculation of the $L$ values of $z_{\text{core}}^{m+1}$. However, the assumption of pseudo-random puncturing can further simplify the computation of $z_{\text{core}}^{m+1}$.

In order to express $z_{\text{core}}^{m+1}$ in a close-form, we first need to consider the autocorrelation function $\phi(i)$ of a polar sequence of length $L$, which is defined as [15]

$$\phi(i) = \sum_{j=1}^{L} (2x_j - 1)(2x_{j+i} - 1) \tag{5.39}$$

where $x_j = \{0,1\}$ is the parity check output of the parent rate-1/2 RSC encoder at time-step $j$ for an input sequence of information weight two, and $i$ is a non-negative integer with values lying in the range $0 \leq i < L$. In event of the parity check output sequence being a PN sequence, the autocorrelation function reduces to [52, 54]

$$\phi(i) = \begin{cases} 2^\nu - 1, & \text{if } i = 0 \\ -1, & \text{if } 1 \leq i < L. \end{cases} \tag{5.40}$$

If we expand (5.39), we obtain

$$\begin{aligned} \phi(i) &= \sum_{j=1}^{L} (4x_j x_{j+i} - 2x_j - 2x_{j+i} + 1) \\ &= 4\sum_{j=1}^{L} x_j x_{j+i} - 2\sum_{j=1}^{L} x_j - 2\sum_{j=1}^{L} x_{j+i} + L \\ &= 4\sum_{j=1}^{L} x_j x_{j+i} - 2(2^{\nu-1}) - 2(2^{\nu-1}) + (2^\nu - 1) \\ &= 4\sum_{j=1}^{L} x_j x_{j+i} - 2^\nu - 1, \end{aligned} \tag{5.41}$$

since there are $2^{\nu-1}$ non-zero bits in a PN sequence of length $L = 2^{\nu} - 1$, as we described in Section 5.3, thus

$$\sum_{j=1}^{L} x_{j+i} = 2^{\nu-1}, \tag{5.42}$$

for $0 \leq i < L$. Combining (5.40) and (5.41), we find that

$$\sum_{j=1}^{L} x_j x_{j+i} = \begin{cases} 2^{\nu-1}, & \text{if } i = 0 \\ 2^{\nu-2}, & \text{if } 1 \leq i < 2^{\nu} - 1. \end{cases} \tag{5.43}$$

The operation of multiplication on binary numbers, i.e., $x_j x_{j+i}$, is equivalent to the AND operation, i.e., $x_j \cdot x_{j+i}$, hence (5.43) can be re-written as

$$\sum_{j=1}^{L} (x_j \cdot x_{j+i}) = \begin{cases} 2^{\nu-1}, & \text{if } i = 0 \\ 2^{\nu-2}, & \text{if } 1 \leq i < 2^{\nu} - 1. \end{cases} \tag{5.44}$$

We observe that the AND operation between two different PN sequences, generates a third sequence having weight $2^{\nu-2}$. When the two PN sequences are identical, the weight of the generated sequence is $2^{\nu-1}$.

If one of the PN sequences in (5.44) is the parity check output stream of the parent rate-1/2 RSC encoder, generated during a sequence of state transitions from state $2^{\nu-1}$ back to state $2^{\nu-1}$, while the other PN sequence comprises of the elements of the puncturing row vector $\mathbf{P}_Z$, we can use the autocorrelation function to derive the weight of the punctured parity check sequence. More specifically, we first ensure that the elements of $\mathbf{P}_Z = [p_{2,1} \ldots p_{2,M}]$ form a PN sequence of period $M = L = 2^{\nu} - 1$ by setting the $(j+1)$-th element, $p_{2,j+1}$, equal to the $j$-th bit of the parity check stream, $x_j$, i.e., $p_{2,j+1} = x_j$, for $1 \leq j \leq L$. We then replace $x_{j+i}$ in (5.44) with its equivalent, $p_{2,j+i+1}$, to obtain

$$\sum_{j=1}^{L} (x_j \cdot p_{2,(j+i+1)}) = \begin{cases} 2^{\nu-1}, & \text{if } i = 0 \\ 2^{\nu-2}, & \text{if } 1 \leq i < 2^{\nu} - 1, \end{cases} \tag{5.45}$$

or

$$\sum_{j=1}^{L} (x_j \cdot p_{2,(m+j)}) = \begin{cases} 2^{\nu-1}, & \text{if } m = 1 \\ 2^{\nu-2}, & \text{if } 2 \leq m \leq 2^{\nu} - 1, \end{cases} \tag{5.46}$$

if $i+1$ is replaced by variable $m$, which assumes values in the range $1 \leq m \leq 2^{\nu} - 1$. We have shown that the last bit $x_L$ of the parity check stream $x_1, x_2, \ldots, x_L$, is always zero, subsequently $x_L \cdot p_{2,m+L} = 0$. From this observation, we deduce that all non-zero bits of the generated sequence $x_j \cdot p_{2,m+j}$ lie in the first $L-1$ positions, hence the weight of the generated sequence remains unchanged if we take the sum of only the first $L-1$ digits

$$\sum_{j=1}^{L-1}(x_j \cdot p_{2,(m+j)}) = \begin{cases} 2^{\nu-1}, & \text{if } m=1 \\ 2^{\nu-2}, & \text{if } 2 \leq m \leq 2^{\nu}-1. \end{cases} \tag{5.47}$$

The quantity on the left hand side of (5.47) corresponds to $z_{\text{core}}^{m+1}$ according to (5.28), therefore

$$z_{\text{core}}^{m+1} = \begin{cases} 2^{\nu-1}, & \text{if } m=1 \\ 2^{\nu-2}, & \text{if } 2 \leq m \leq 2^{\nu}-1, \end{cases} \tag{5.48}$$

which leads us to the conclusion that when the parity check output sequence of an RSC encoder, employing a primitive feedback polynomial, is punctured using a pseudo-randomly generated puncturing vector $\mathbf{P}_Z$, the core weight $z_{\text{core}}^{m+1}$ can be expressed in a close-form as a function of the memory $\nu$ of the RSC encoder only.

If we update (5.38) accordingly, we find that the weight $z(k,m)$ of the parity check output sequence of the RSC encoder is

$$z(k,m) = \begin{cases} k2^{\nu-1} + 2p_{2,m}\ , & \text{if } m=1 \\ k2^{\nu-2} + 2p_{2,m}\ , & \text{if } 2 \leq m \leq 2^{\nu}-1. \end{cases} \tag{5.49}$$

We explained earlier that the elements of the puncturing vector $\mathbf{P}_Z$ were assigned values according to the rule $p_{2,j+1} = x_j$, so as to ensure that a PN sequence would emerge. Since the parity check bit $x_L$ is zero, we deduce that $p_{2,L+1}$ is also zero and so is $p_{2,1}$, owing to the periodicity $M$ of the puncturing pattern which we take to be equal to $L$. Thus, the expression for the weight $z(k,m)$ assumes the final form

$$z(k,m) = \begin{cases} k2^{\nu-1}\ , & \text{if } m=1 \\ k2^{\nu-2} + 2p_{2,m}\ , & \text{if } 2 \leq m \leq 2^{\nu}-1. \end{cases} \tag{5.50}$$

We observe that $z(k,m)$ can assume three distinct values, namely $k2^{\nu-2}$, $k2^{\nu-2}+2$, and $k2^{\nu-1}$, in order of magnitude, but only when $\nu > 2$. Invoking the properties of PN sequences [52], we note that only $2^{\nu-1}-1$ out of the $2^{\nu}-1$ elements $p_{2,m}$ of the puncturing vector $\mathbf{P}_Z$ are equal to 0. If the memory size is $\nu = 2$, we find that $p_{2,m}$ assumes the zero value only once, and according to (5.50) this happens when $m = 1$. Consequently, the value of $z(k,m)$ for a memory size of 2 can either be $2k$ if $m = 1$, or $k+2$ if $m > 1$.

The minimum weight $z_{\min}$ of the parity check output sequences can be derived from $z(k,m)$, if we set $k = 1$ and select the minimum valid value. In particular,

$$z_{\min} = \begin{cases} 2, & \text{for } \nu = 2 \\ 2^{\nu-2}, & \text{for } \nu > 2, \end{cases} \tag{5.51}$$

since we cannot obtain a minimum weight of $2^{\nu-2}=1$ when $\nu=2$, based on the explanation given in the previous paragraph.

The rate of a punctured RSC encoder depends upon the number of codeword bits, both systematic and parity check, transmitted during the puncturing period $M$. We know that in the case of pseudo-randomly punctured RSC codes, vector $\mathbf{P}_Z=[p_{2,1}\ \ldots\ p_{2,M}]$ contains $2^{\nu-1}$ non-zero elements, hence $2^{\nu-1}$ parity check bits evade puncturing and, consequently, at least $2^{\nu-1}$ codeword bits are transmitted for every $M=2^{\nu}-1$ input information bits. The rate of the code can be reduced by increasing the number of non-zero elements in the vector $\mathbf{P}_U=[p_{1,1}\ \ldots\ p_{1,M}]$, which determines which systematic bits are eliminated during the puncturing period $M$.

The dominant conditional weight enumerating function $B(w=2,U,Z)$ can be obtained from (5.36), as follows

$$B(w=2,U,Z)=\sum_{k=1}^{\lfloor (N-1)/(2^{\nu}-1)\rfloor}\sum_{m=1}^{2^{\nu}-1} B_{k,m}U^{u(m)}Z^{z(k,m)}, \tag{5.52}$$

where $B_{k,m}$, defined in (5.35), assumes the form

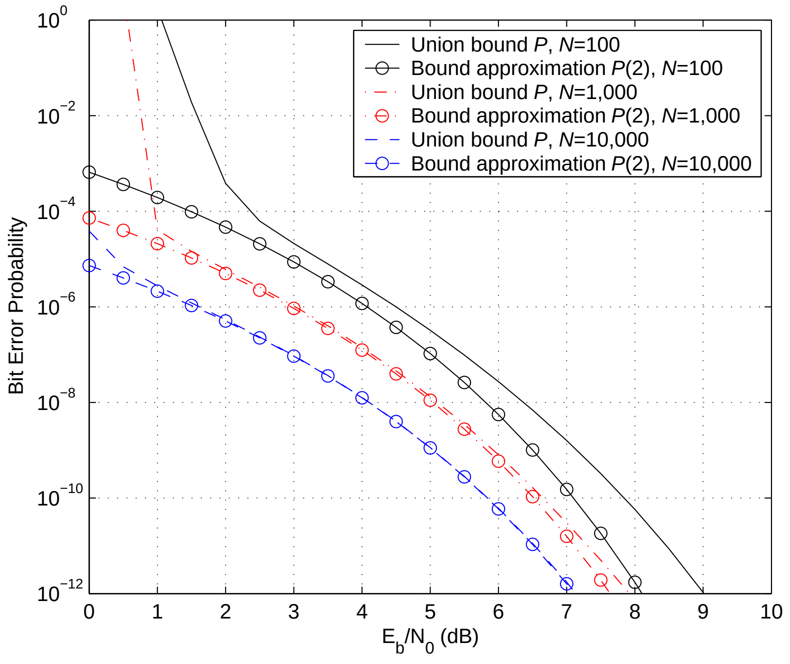
$$B_{k,m}=\begin{cases}\left\lfloor\frac{N}{2^{\nu}-1}\right\rfloor-k, & \text{if rem}\,(N,2^{\nu}-1)<m\\ \left\lfloor\frac{N}{2^{\nu}-1}\right\rfloor-k+1, & \text{otherwise,}\end{cases} \tag{5.53}$$

for $M=L=2^{\nu}-1$, while $u(m)$ and $z(k,m)$ have been defined for the case of pseudo-randomly punctured RSC codes in (5.37) and (5.50), respectively.

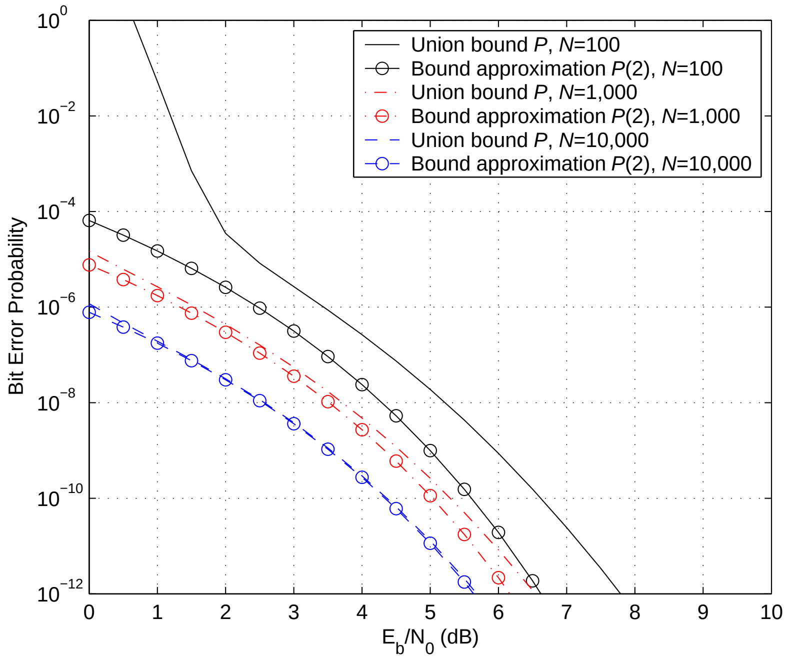
The dominant conditional weight enumerating function of a higher rate RSC encoder, obtained after puncturing the parity check output of the rate-1/2 RSC encoder using a pseudo-randomly generated vector, mainly depends upon the memory size of the RSC encoder and not the underlying code, as observed for a rate-1/2 RSC encoder employing a primitive feedback polynomial. A list of example pseudo-random row vectors $\mathbf{P}_Z$, appropriate for puncturing RSC encoders employing primitive feedback vectors $\mathbf{G}_R$ and feedforward vectors $\mathbf{G}_F$, is provided in Table 5.2.

## 5.6 Evaluation of the Bound Approximation for Non-Punctured and Punctured Turbo Codes

In Section 5.3, we derived expression (5.16) which we can straightforwardly use as the basis for numerically computing the dominant conditional weight enumerating function $B(w=2,U,Z)$ of a non-punctured rate-1/2 RSC encoder. Furthermore, we

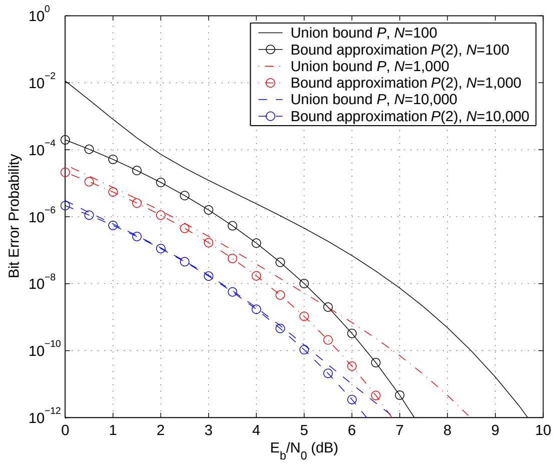


(a) Rate-1/3 PCCC(1,5/7,5/7)

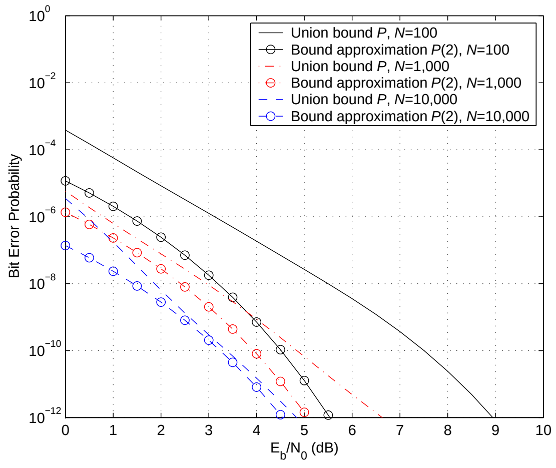


(b) Rate-1/3 PCCC(1,17/15,17/15)

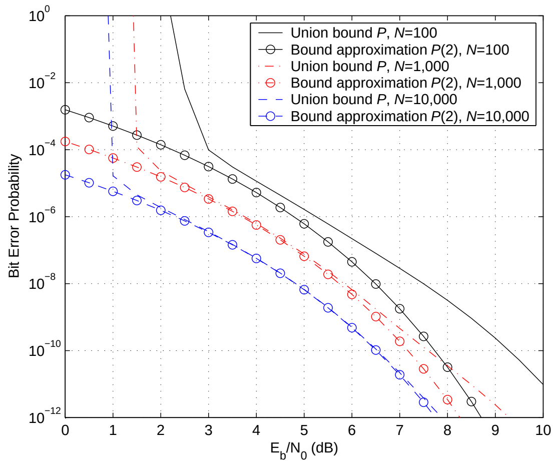
Figure 5.8: Exact union bounds and their approximation for various non-punctured turbo codes employing an interleaver of size 100, 1,000 or 10,000 bits.



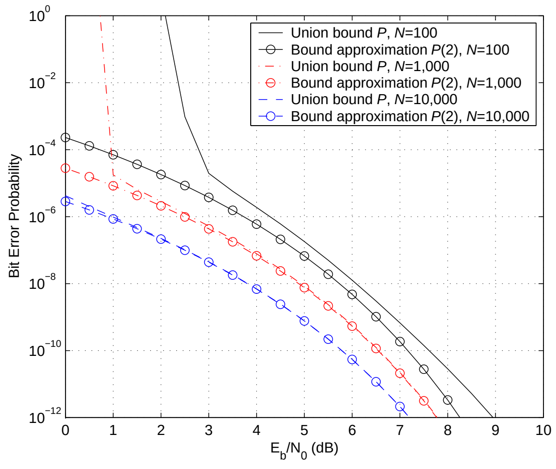
(a) Rate-1/2 NS-PCCC(1,5/7,5/7)



(b) Rate-1/2 NS-PCCC(1,17/15,17/15)



(c) Rate-1/2 Sys. PCCC(1,5/7,5/7)



(d) Rate-1/2 Sys. PCCC(1,17/15,17/15)

Figure 5.9: Exact union bounds and their approximation for various punctured turbo codes employing an interleaver of size 100, 1,000 or 10,000 bits.

Table 5.2: Pseudo-random row vectors $\mathbf{P}_Z$ for various RSC code configurations.

| Memory size, $\nu$ | Octal $\mathbf{G}_R$ | Octal $\mathbf{G}_F$ | $\mathbf{P}_Z$ | $z_{\min}$ |
|---|---|---|---|---|
| 2 | 7 | 5 | [0 1 1] | 2 |
| 3 | 15 | 17 | [0 0 1 1 1 0 1] | 2 |
| 4 | 31<br>31 | 33<br>27 | [0 0 0 1 1 1 1 0 1 0 1 1 0 0 1]<br>[0 1 0 1 1 0 0 1 0 0 0 1 1 1 1] | 4 |

demonstrated that when the feedback generator vector of the RSC encoder is primitive, $B(w=2,U,Z)$ reduces to (5.22) and becomes a function of the memory size of the encoder, only. The effect of puncturing the information weight-2 codeword sequences, generated by a rate-1/2 RSC encoder, was studied in Section 5.4. In a similar manner, we obtained expression (5.36), which accurately enumerates the information weight-2 codeword sequences comprising $B(w=2,U,Z)$, when punctured RSC codes are considered. In addition, we presented a case study, according to which the dominant conditional weight enumerating function of an RSC encoder employing a primitive feedback polynomial, can be expressed in a close-form, if the parity check output sequence of the encoder is punctured by a pseudo-randomly generated row vector. In particular, we can observe in (5.52) that $B(w=2,U,Z)$ becomes a function of the puncturing pattern and the memory size of the encoder.

Therefore, if $\mathcal{C}_1$ and $\mathcal{C}_2$ are constituent RSC codes of a turbo encoder $\mathcal{P}$, punctured or non-punctured, we can use (5.16), (5.22), (5.36) or (5.52), depending on the case, to calculate $B^{\mathcal{C}_1}(w=2,U,Z)$ and $B^{\mathcal{C}_2}(w=2,U,Z)$, respectively. Subsequently, the dominant conditional weight enumerating function $B^{\mathcal{P}}(w=2,U,Z)$ of the turbo encoder can be obtained from (5.6), under the assumption that a uniform interleaver of size $N$ is used. Knowledge of $B^{\mathcal{P}}(w=2,U,Z)$ allows us to compute the probability $P(2)$ of all error events with information weight 2, which is a good approximation of the union bound $P_b^{\mathrm{u}}$ when long interleavers are used, as we explained in Section 5.2.

In Fig.5.8 and Fig.5.9, we compare the union bound $P_b^{\mathrm{u}}$ with its approximation $P(2)$, for the same turbo code configurations considered in Fig.5.1 and Fig.5.2, respectively. As expected, $P(2)$ deviates from $P_b^{\mathrm{u}}$, when short interleavers are considered. However, for interleaver sizes of $N = 10,000$ bits or larger, the bound approximation $P(2)$ closely matches the union bound $P_b^{\mathrm{u}}$, for medium to low bit error probabilities. Hence, when ML soft decoding of rate-1/3 turbo codes or rate-1/2 punctured turbo codes is considered, $P(2)$ is indeed a good approximation of the union bound $P_b^{\mathrm{u}}$. If, on the other hand, suboptimal iterative decoding is investigated, $P(2)$ is a good indication of the error floor, provided that the performance of the decoder eventually converges towards the error floor at medium to high $E_b/N_0$ values. The convergence behavior of iterative decoding of rate-1/2 punctured turbo codes is studied in the following chapter.

## 5.7 Chapter Summary

In Chapter 3 and Chapter 4, we proposed techniques to evaluate the transfer function of constituent RSC codes and, ultimately, compute the transfer function of a turbo code, punctured or non-punctured. As a result, a tight upper bound on the bit error probability of the turbo code can be derived.

In this chapter, we proposed a rapid method, which exploits the properties of constituent RSC encoders, to obtain their dominant conditional weight enumerating functions, which only enumerate codeword sequences having information weight two. This is in contrast to their transfer functions, which enumerate all codeword sequences having non-zero information weight. We demonstrated that the dominant conditional weight enumerating function of a rate-1/3 turbo code, or a rate-1/2 punctured turbo code, significantly contributes to the upper bound on the bit error probability, as the interleaver size increases. Consequently, we proposed a bound approximation, which only takes into account the dominant conditional weight enumerating function rather than the full transfer function. The bound approximation is accurate when long interleavers are employed, thus it can give us insight concerning the ML decoding performance of rate-1/3 and rate-1/2 turbo codes.

In the next chapter we will use the bound approximation to evaluate the performance of rate-1/2 punctured turbo codes and investigate whether they can yield lower bounds on the bit error probability than those of their rate-1/3 parent turbo codes.

Chapter 6

# Punctured Turbo Codes Exhibiting Low Error Floors

## 6.1 Introduction

Having developed the tools to obtain a bound approximation, which is an accurate estimate of the union bound on the error probability of turbo codes, we put them into practice so as to evaluate the performance of punctured turbo codes using long interleavers.

In particular, we observe that certain configurations of rate-1/2 punctured turbo codes yield a lower approximate bound than that of their rate-1/3 parent codes. Specifically we show that this is the case, when the rate of a turbo code is increased from 1/3 to 1/2 using particular puncturing patterns. Although we can draw immediate conclusions from that result about the ML decoding performance of those rate-1/2 punctured turbo codes, we cannot be conclusive when suboptimal iterative decoding is used. Consequently, we study the convergence behavior of iterative decoding using the widely accepted EXIT chart analysis and we investigate whether the performance of the proposed rate-1/2 punctured turbo codes converges towards their union bound, or equivalently the bound approximation, at high signal-to-noise ratios.

We conclude the chapter by comparing analytic to simulation results and presenting a summary of our findings.

## 6.2 An Interesting Observation: Performance Improvement as a Consequence of a Code Rate Increase

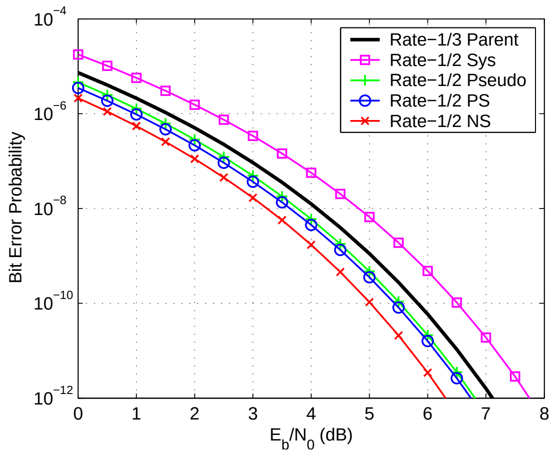
In the previous chapter, we established that the probability $P(2)$ of all error events with information weight 2 is a good approximation of the union bound on the average bit error probability of turbo codes using long random interleavers, when ML soft decoding is employed. In Fig.6.1, we compare the bound approximations of rate-1/2 punctured turbo codes, namely the systematic (Sys) and the non-systematic (NS) schemes presented in the previous chapter, with the bound approximation of their rate-1/3 parent turbo code. For consistency, the rate-1/3 parent codes considered are PCCC(1,5/7,5/7) and PCCC(1,17/15,17/15). In all cases, a long interleaver of size $N = 10,000$ is used.

In both Fig.6.1(a) and Fig.6.1(b), we have also included a partially systematic scheme, denoted as PS, and a pseudo-randomly punctured scheme, which we refer to as "Pseudo". Note that, the puncturing pattern for PS, which leads to rate-1/2 turbo codes exhibiting low ML performance bounds, has been obtained by means of an exhaustive search among all available patterns of period $M = 4$, as we described in Chapter 4. In the case of each "Pseudo" PCCC scheme, we use a pseudo-randomly generated row vector $\mathbf{P}_{Z1}$ to puncture the first parity check output of the parent PCCC. The second parity check output is not punctured, thus every element of the corresponding row vector $\mathbf{P}_{Z2}$ is set to 1. Finally, we choose the puncturing row vector $\mathbf{P}_U$ for the systematic output to be the complement of $\mathbf{P}_{Z1}$. Subsequently, the puncturing pattern composed of the three row vectors, $\mathbf{P}_U$, $\mathbf{P}_{Z1}$ and $\mathbf{P}_{Z2}$, increases the rate of the code from 1/3 to 1/2. The puncturing patterns for the afore-mentioned rate-1/2 turbo codes, are presented in Table 6.1.

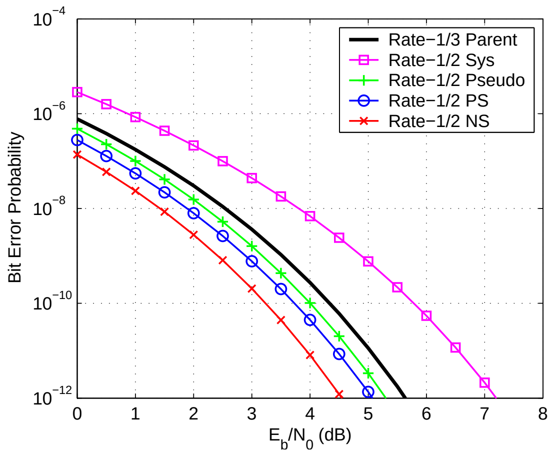
As in the case of convolutional codes, it would be reasonable to expect that puncturing a turbo code is a trade-off between bandwidth efficiency and channel

Table 6.1: Puncturing patterns for various rate-1/2 PCCC configurations.

| | Pseudo | | | |
|---|---|---|---|---|
| Sys | PCCC(1,5/7,5/7) | PCCC(1,17/15,17/15) | PS | NS |
| $\begin{bmatrix}1&1\\1&0\\0&1\end{bmatrix}$ | $\begin{bmatrix}1&0&0\\0&1&1\\1&1&1\end{bmatrix}$ | $\begin{bmatrix}1&1&0&0&0&1&0\\0&0&1&1&1&0&1\\1&1&1&1&1&1&1\end{bmatrix}$ | $\begin{bmatrix}0&0&1&0\\1&1&0&1\\1&1&1&1\end{bmatrix}$ | $\begin{bmatrix}0\\1\\1\end{bmatrix}$ |



(a) PCCC(1,5/7,5/7)



(b) PCCC(1,17/15,17/15)

Figure 6.1: Bound approximations of rate-1/3 parent turbo codes and rate-1/2 punctured turbo codes. The size of the interleaver is $N = 10,000$ bits.

reliability. Thus, an increase in the code rate from 1/3 to 1/2 would improve bandwidth efficiency at the expense of channel reliability. Indeed, we observe in Fig.6.1 that the bound approximation of rate-1/2 systematic turbo codes is higher than the bound approximation of their rate-1/3 parent codes. From that, we deduce that rate-1/3 turbo codes achieve a better performance, in terms of bit error probability, than rate-1/2 punctured systematic turbo codes, hence channel reliability degrades when the code rate is increased from 1/3 to 1/2, as expected. However, we see that the remaining three rate-1/2 configurations exhibit lower bound approximations than their parent code, with the non-systematic scheme achieving the lowest. Inevitably, the following question arises: "Under what conditions, do rate-1/2 punctured turbo codes achieve a superior performance to that of their rate-1/3 parent codes?". In the following sections, we try to answer this question using the bound approximation, when ML soft decoding is assumed, and EXIT charts, when suboptimal iterative decoding is considered.

## 6.3 Performance Evaluation using the Bound Approximation

Under the assumption of ML soft decoding, accurate identification of turbo codes that yield low bit error probability, for a given $E_b/N_0$ value, is performed by evaluating and comparing their analytical bounds on the bit error probability. Thus, if $\mathcal{P}$ and $\mathcal{P}'$ are two PCCCs, we say that $\mathcal{P}$ outperforms $\mathcal{P}'$, if their union bounds, $P_b^{\mathrm{u},\mathcal{P}}$

Table 6.2: Coefficients $B_{2,d}$ for the calculation of the bound approximation $P(2)$, for rate-1/2 punctured configurations of the rate-1/3 parent PCCC(1,5/7,5/7).

| $d$ | Parent | Sys | Pseudo | PS | NS |
|---|---|---|---|---|---|
| 1-5 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 4.4969 | 0 | 0 | 0 |
| 7 | 0 | 0 | 1.3325 | 0.9996 | 0 |
| 8 | 0 | 11.986 | 1.9986 | 1.4991 | 1.9990 |
| 9 | 0 | 0 | 2.6639 | 1.9985 | 0 |
| 10 | 1.9990 | 19.965 | 3.9956 | 2.9972 | 3.9968 |
| 11 | 0 | 0 | 3.9940 | 2.9967 | 0 |
| 12 | 3.9968 | 27.934 | 5.9910 | 4.9935 | 5.9934 |
| 13 | 0 | 0 | 5.3229 | 3.9942 | 0 |
| 14 | 5.9934 | 35.895 | 7.9848 | 6.4893 | 7.9888 |
| 15 | 0 | 0 | 6.6507 | 4.9910 | 0 |
| 16 | 7.9888 | 43.845 | 9.9770 | 7.9840 | 9.9830 |
| 17 | 0 | 0 | 7.9772 | 5.9871 | 0 |
| 18 | 9.9830 | 51.786 | 11.968 | 9.9764 | 11.976 |
| 19 | 0 | 0 | 9.3025 | 6.9825 | 0 |
| 20 | 11.976 | 59.717 | 13.957 | 11.469 | 13.968 |

and $P_b^{\mathrm{u},\mathcal{P}'}$ respectively, satisfy

$$P_b^{\mathrm{u},\mathcal{P}} < P_b^{\mathrm{u},\mathcal{P}'}, \tag{6.1}$$

which reduces to

$$P^{\mathcal{P}}(2) < P^{\mathcal{P}'}(2), \tag{6.2}$$

when long interleavers are considered. If the interleavers used by the two PCCCs have identical size, we can expand (6.2) as follows

$$\sum_d B_{2,d}^{\mathcal{P}} Q\left(\sqrt{\frac{2R^{\mathcal{P}}\,E_b}{N_0}\,d}\right) < \sum_d B_{2,d}^{\mathcal{P}'} Q\left(\sqrt{\frac{2R^{\mathcal{P}'}\,E_b}{N_0}\,d}\right). \tag{6.3}$$

As we mentioned in Chapter 2, Benedetto and Montorsi demonstrated in [26] that the most significant parameter of a turbo code is its free effective distance, $d_{\mathrm{free,eff}}$, which conveys the minimum weight of a codeword sequence for an input sequence of information weight 2. Furthermore, the authors suggested that the simplest form of performance optimization is the design of generator polynomials that maximize the free effective distance of a PCCC. Therefore, if $d_{\mathrm{free,eff}}^{\mathcal{P}}$ and $d_{\mathrm{free,eff}}^{\mathcal{P}'}$

denote the free effective distances of $\mathcal{P}$ and $\mathcal{P}'$ respectively, condition (6.2) collapses to

$$B^{\mathcal{P}}_{2,d_{\text{free,eff}}} Q\left(\sqrt{\frac{2R^{\mathcal{P}}\,E_b}{N_0}\,d^{\mathcal{P}}_{\text{free,eff}}}\right) < B^{\mathcal{P}'}_{2,d_{\text{free,eff}}} Q\left(\sqrt{\frac{2R^{\mathcal{P}'}\,E_b}{N_0}\,d^{\mathcal{P}'}_{\text{free,eff}}}\right), \qquad (6.4)$$

which only considers the first non-zero, that is the most significant term of each sum. When the code rates are equal, i.e., $R^{\mathcal{P}} = R^{\mathcal{P}'}$, the free effective distance of turbo codes plays a role similar to that of the free distance of convolutional codes [26]. Hence, the performance criterion can be simplified to

$$d^{\mathcal{P}}_{\text{free,eff}} > d^{\mathcal{P}'}_{\text{free,eff}}. \qquad (6.5)$$

In the case that the two codes yield the same free effective distance, we compare coefficients $B^{\mathcal{P}}_{2,d_{\text{free,eff}}}$ and $B^{\mathcal{P}'}_{2,d_{\text{free,eff}}}$. Turbo code $\mathcal{P}$ achieves a slightly better performance than $\mathcal{P}'$, only if $B^{\mathcal{P}}_{2,d_{\text{free,eff}}}$ is lower than $B^{\mathcal{P}'}_{2,d_{\text{free,eff}}}$.

In Table 6.2, the first 20 coefficients $B_{2,d}$ of the bound approximation $P(2)$ for the rate-1/3 parent PCCC(1,5/7,5/7), as well as the "Sys", "Pseudo", "PS" and "NS" configurations of rate 1/2, are presented. All coefficients were obtained using the method we described in Chapter 5. Recall that a uniform interleaver is assumed, hence a coefficient $B_{2,d}$ represents the average number of turbo codeword sequences having overall output weight $d$, which were generated by input sequences of information weight 2. The smallest value of $d$ for which $B_{2,d}$ is non-zero, corresponds to the free effective distance $d_{\text{free,eff}}$ of the turbo code. We focus on the rate-1/2 schemes and we observe that the systematic PCCC yields a lower free effective distance ($d^{\text{Sys}}_{\text{free,eff}} = 6$) than the non-systematic PCCC ($d^{\text{NS}}_{\text{free,eff}} = 8$). Subsequently, the bound of the non-systematic PCCC is expected to be lower than that of the systematic PCCC, while the bounds of the pseudo-randomly punctured and partially systematic turbo codes are expected to lie in between, since $d^{\text{Pseudo}}_{\text{free,eff}} = d^{\text{PS}}_{\text{free,eff}} = 7$. However, the partially systematic PCCC yields a lower coefficient, hence that code is anticipated to perform better than the pseudo-randomly punctured PCCC. Our conjectures, based on the simple criterion (6.5), are supported by the bound approximation curves presented in Fig.6.1(a).

Turbo codes using interleavers of the same size, but achieving different rates, can be evaluated using (6.4). Our objective in this section, is to perform an analytical performance comparison between rate-1/2 turbo codes and their parent rate-1/3 turbo codes. Based on the observations in Fig.6.1, we use criterion (6.4) to investigate when rate-1/2 non-systematic PCCCs and pseudo-randomly punctured PCCCs exhibit a lower bound approximation than that of their parent turbo codes. We do not consider the whole set of partially systematic turbo codes in our investigation, since their free effective distance and coefficients depend entirely upon their puncturing patterns. Consequently, particular partially systematic turbo codes yielding low bit error probabilities can be found only by means of an exhaustive search, for a given parent turbo code. Although pseudo-random punctured PCCCs form a subset of partially systematic PCCCs, we will demonstrate that their characteristics depend only upon the memory size of the constituent encoders.

Inevitably, we make references to Chapter 5 throughout our investigation, since we use the concepts and expressions, derived in that chapter, to identify the relationship between parameters that influence the performance of a turbo code, such as its free effective distance, and parameters associated with the structure of the constituent encoders, such as their memory size.

### 6.3.1 Analysis of Rate-1/3 Parent PCCCs

Criterion (6.4) requires knowledge of the free effective distance $d_{\text{free,eff}}^{\text{Parent}}$ and the coefficient $B_{2,d_{\text{free,eff}}}^{\text{Parent}}$. In general, the free effective distance $d_{\text{free,eff}}^{\text{Parent}}$ of a PCCC can be expressed as the sum of the minimum weight $d_{\min}$ of the codeword sequence generated by the first constituent encoder and the minimum weight $z_{\min}$ of the parity check sequence generated by the second constituent encoder. The result is the overall codeword weight of the output sequence for an input sequence of information weight 2. Observing that the turbo codes to be considered here are symmetric, i.e., the constituent encoders are identical, and the weight of the systematic output sequence is always 2, i.e., $u_{\min}=2$, we can write

$$\begin{aligned} d_{\text{free,eff}}^{\text{Parent}} &= d_{\min} + z_{\min} \\ &= (u_{\min} + z_{\min}) + z_{\min} \\ &= 2 + 2z_{\min}. \end{aligned} \tag{6.6}$$

From this relationship, we deduce that a turbo codeword sequence of weight $d_{\text{free,eff}}$ can only be obtained, when the weight of the systematic output sequence is 2 and the weight of each parity check output sequence is $z_{\min}$. Therefore, if $B_{2,d_{\text{free,eff}}}^{\text{Parent}}$ denotes the number of turbo codeword sequences having weight $d_{\text{free,eff}}$, it follows from (6.6) that $B_{2,d_{\text{free,eff}}}^{\text{Parent}}$ matches the number $B_{2,2,2z_{\min}}^{\text{Parent}}$ of turbo codeword sequences having systematic weight 2 and overall parity check weight $2z_{\min}$,

$$B_{2,d_{\text{free,eff}}}^{\text{Parent}} = B_{2,2,2z_{\min}}^{\text{Parent}}. \tag{6.7}$$

Note that, in both notations, the first index refers to the input information weight, which is 2. The number $B_{2,2,2z_{\min}}^{\text{Parent}}$ of codeword sequences, generated by a turbo encoder using a uniform interleaver of size $N$, can be associated with the number $B_{2,2,z_{\min}}$ of codeword sequences having systematic weight 2 and parity check weight $z_{\min}$, generated by the first constituent encoder, and the number $B_{2,0,z_{\min}}$ of parity check sequences having weight $z_{\min}$, generated by the second constituent encoder, if we elaborate on (2.69). In particular, we obtain

$$B^{\text{Parent}}_{2,2,2z_{\min}} = \frac{B_{2,2,z_{\min}} B_{2,0,z_{\min}}}{\binom{N}{2}}, \tag{6.8}$$

where $B_{2,2,z_{\min}}$ and $B_{2,0,z_{\min}}$ return the same value, since they both consider the same trellis paths, as we can deduce from our discussion in Chapter 5. If $\mathsf{L}$ is the period of the feedback generator polynomial used by the turbo encoder, we find from (5.15) that

$$B_{2,2,z_{\min}} = B_{2,0,z_{\min}} = N - \mathsf{L}. \tag{6.9}$$

Combining (6.7), (6.8) and (6.9), we can express $B^{\text{Parent}}_{2,d_{\text{free,eff}}}$ as a function of the intlerleaver size $N$ and the period $\mathsf{L}$

$$B^{\text{Parent}}_{2,d_{\text{free,eff}}} = \frac{2(N-\mathsf{L})^2}{N(N-1)}. \tag{6.10}$$

If a primitive feedback generator polynomial is used, the minimum weight $z_{\min}$ can be expressed in terms of the memory size $\nu$, i.e., $z_{\min} = 2^{\nu-1} + 2$, as we have shown in (5.20). Consequently, expression (6.6) assumes the form

$$d^{\text{Parent}}_{\text{free,eff}} = 6 + 2^{\nu}. \tag{6.11}$$

In the special case when the size $N$ of the interleaver is an integer multiple of the period $\mathsf{L}$ of the feedback generator polynomial used by a constituent encoder, i.e., $N = \mu\mathsf{L}$, we can rewrite expression (6.10) as

$$B^{\text{Parent}}_{2,d_{\text{free,eff}}} = \frac{2\mathsf{L}(\mu-1)^2}{\mu(\mu\mathsf{L}-1)}. \tag{6.12}$$

### 6.3.2 Analysis of Rate-1/2 Non-Systematic PCCCs

A rate-1/2 non-systematic turbo code is obtained from a rate-1/3 turbo code, when the systematic output sequence is not transmitted. Therefore, the main parameters of the non-systematic turbo code, namely the free effective distance $d^{\text{NS}}_{\text{free,eff}}$ and the coefficient $B^{\text{NS}}_{2,d_{\text{free,eff}}}$, can be derived from the equivalent expressions of the parent turbo code, if we eliminate the contributions of the systematic output. Subsequently, we take the weight of the systematic output to be always zero, thus $u'_{\text{min}}=0$. If we update (6.6) accordingly, we obtain the free effective distance

$$\begin{aligned} d^{\text{NS}}_{\text{free,eff}} &= d'_{\text{min}} + z_{\text{min}} \\ &= (u'_{\text{min}} + z_{\text{min}}) + z_{\text{min}} \\ &= 2z_{\text{min}}. \end{aligned} \tag{6.13}$$

Note that symbols $u_{\text{min}}$, $z_{\text{min}}$ and $d_{\text{min}}$, have been reserved for the minimum weight of the non-punctured systematic sequence, the minimum weight of the non-punctured parity check sequence and the overall minimum weight of the non-punctured codeword sequence, generated by a constituent encoder. When one or both output sequences are punctured, we use different symbols to denote their weights.

In a similar manner to rate-1/3 turbo codes, the number $B^{\text{NS}}_{2,d_{\text{free,eff}}}$ of codeword sequences having weight $d^{\text{NS}}_{\text{free,eff}}$, generated by a rate-1/2 non-systematic PCCC, is related to the number $B_{2,0,z_{\text{min}}}$ of parity check sequences having weight $z_{\text{min}}$, generated by each constituent encoder, as follows

$$\begin{aligned} B^{\text{NS}}_{2,d_{\text{free,eff}}} &= \frac{B_{2,0,z_{\text{min}}} B_{2,0,z_{\text{min}}}}{\begin{pmatrix} N \\ 2 \end{pmatrix}} \\ &= \frac{2(N-L)^2}{N(N-1)}. \end{aligned} \tag{6.14}$$

Note that $B_{2,0,z_{\text{min}}}=N-L$, according to (6.9).

Based on condition (6.4), a rate-1/2 non-systematic PCCC yields a lower bound on the bit error probability than that of its rate-1/3 parent PCCC, if

$$B^{\text{NS}}_{2,d_{\text{free,eff}}} Q\left(\sqrt{\frac{2(1/2)\,E_b}{N_0}\, d^{\text{NS}}_{\text{free,eff}}}\right) < B^{\text{Parent}}_{2,d_{\text{free,eff}}} Q\left(\sqrt{\frac{2(1/3)\,E_b}{N_0}\, d^{\text{Parent}}_{\text{free,eff}}}\right). \tag{6.15}$$

Taking into account (6.6) and (6.10), we observe that parameters $d^{\text{NS}}_{\text{free,eff}}$ and $B^{\text{NS}}_{2,d_{\text{free,eff}}}$ of the non-systematic PCCC can be represented in terms of the parameters $d^{\text{Parent}}_{\text{free,eff}}$ and $B^{\text{Parent}}_{2,d_{\text{free,eff}}}$ of the parent PCCC, i.e.,

$$\begin{aligned} d^{\text{NS}}_{\text{free,eff}} &= d^{\text{Parent}}_{\text{free,eff}} - 2, \\ B^{\text{NS}}_{2,d_{\text{free,eff}}} &= B^{\text{Parent}}_{2,d_{\text{free,eff}}}. \end{aligned} \tag{6.16}$$

Subsequently, condition (6.15) reduces to

$$Q\left(\sqrt{\frac{E_b}{N_0}\,\left(d^{\text{Parent}}_{\text{free,eff}} - 2\right)}\right) < Q\left(\sqrt{\frac{2E_b}{3N_0}\, d^{\text{Parent}}_{\text{free,eff}}}\right). \tag{6.17}$$

Function $Q(\xi)$, defined in Chapter 2, is a monotonically decreasing function of $\xi$, where $\xi$ is a real number. Therefore, if $\xi_1$ and $\xi_2$ are real numbers, with $\xi_1 > \xi_2$, we deduce that $Q(\xi_1) < Q(\xi_2)$, and vice versa, i.e,

$$Q(\xi_1) < Q(\xi_2) \Leftrightarrow \xi_1 > \xi_2. \tag{6.18}$$

So, inequality (6.17) is satisfied, when

$$\sqrt{\frac{E_b}{N_0}\left(d_{\text{free,eff}}^{\text{Parent}} - 2\right)} > \sqrt{\frac{2E_b}{3N_0}\, d_{\text{free,eff}}^{\text{Parent}}}, \tag{6.19}$$

or simply,

$$d_{\text{free,eff}}^{\text{Parent}} - 2 > \frac{2}{3}\, d_{\text{free,eff}}^{\text{Parent}}. \tag{6.20}$$

Rearranging (6.20), we reach the conclusion that a rate-1/2 non-systematic turbo code achieves a lower bound on the bit error probability than that of its rate-1/3 parent code, only if the free effective distance of the parent code meets the condition

$$d_{\text{free,eff}}^{\text{Parent}} > 6. \tag{6.21}$$

When primitive feedback generator polynomials are considered, condition (6.21) is always satisfied, since

$$6 + 2^{\nu} > 6 \tag{6.22}$$

holds true for any value of memory size $\nu$.

## 6.3.3 Analysis of Rate-1/2 Pseudo-randomly Punctured PCCCs

In Chapter 5 we established that an RSC encoder can be pseudo-randomly punctured only when its feedback generator polynomial is primitive. In a similar fashion, a rate-1/2 pseudo-randomly punctured turbo code can be obtained from a rate-1/3 symmetric turbo code, only if the feedback generator polynomial of each constituent encoder is primitive. In that case, the period $M$ of the puncturing pattern matches the period $L$ of the generator polynomial, i.e., $M = L = 2^{\nu} - 1$. In order to facilitate our analysis, and without loss of generality, we assume that the interleaver size $N$ is an integer multiple of the puncturing period $M$, i.e., $N = \mu M = \mu L$, where $\mu$ is a positive integer.

In order to compute the free effective distance of a pseudo-randomly punctured turbo code, we first need to obtain the minimum weights of the output sequences, produced by the constituent codes. We denote as $d''_{\min}$ the minimum weight of the punctured codeword sequence, generated by a first constituent encoder. The parity check sequence generated by the second encoder is not punctured, thus its weight remains $z_{\min}$. An expression for $d''_{\min}$ can be obtained if we combine (5.37) and (5.50) from Chapter 5. In particular, we find that $d''_{\min}$ is either

$$d''_{\min} = 2^{\nu-1} + 2p_{1,1}\ , \quad \text{when } m = 1, \tag{6.23}$$

or

$$d''_{\min} = 2^{\nu-2} + 2p_{1,m} + 2p_{2,m}\ , \quad \text{when } 2 \leq m \leq M, \tag{6.24}$$

depending on which expression returns the minimum value. Remember, that we set the puncturing row vector $\mathbf{P}_U = [p_{1,1}\ \ldots\ p_{1,M}]$ to be the complement of the puncturing row vector $\mathbf{P}_{Z1} = [p_{2,1}\ \ldots\ p_{2,M}]$, thus $p_{1,m} = 1 - p_{2,m}$. Furthermore, we know that the elements of $\mathbf{P}_{Z1}$ form a PN sequence and that the first element is zero, i.e., $p_{2,1} = 0$. As a result, $p_{1,1} = 1$ always. If we update expressions (6.23) and (6.24) accordingly, we observe that the output codeword weight is either $2^{\nu-1}+2$ or $2^{\nu-2}+2$, depending on the value of $m$. Subsequently, the minimum weight $d''_{\min}$ assumes the value

$$d''_{\min} = 2^{\nu-2} + 2\ , \quad \text{when } 2 \leq m \leq M. \tag{6.25}$$

Having obtained $d''_{\min}$, we compute the free effective distance $d^{\text{Pseudo}}_{\text{free,eff}}$ of a pseudo-randomly punctured turbo code, as follows

$$\begin{aligned} d^{\text{Pseudo}}_{\text{free,eff}} &= d''_{\min} + z_{\min} \\ &= (2^{\nu-2} + 2) + 2^{\nu-1} + 2 \\ &= 4 + 3(2^{\nu-2}), \end{aligned} \tag{6.26}$$

since $z_{\min} = 2^{\nu-1} + 2$, when primitive feedback generator polynomials are used.

When a particular puncturing column $m \neq 1$ is active, we deduce from (5.53) that the first constituent encoder can generate a total of $(\mu-1)$ codeword sequences having weight $d''_{\min}$, since $N = \mu(2^{\nu} - 1)$ and $\text{rem}(N, 2^{\nu}-1) = 0 < m$, for all valid values of $m$. We observe in (6.25) that $m$ can assume $(M-1)$ possible values in the range between 2 and $M$, where $d''_{\min}$ is defined, hence the total number of codeword sequences having weight $d''_{\min}$, independently of which puncturing column is active, is given by the product $(M-1)(\mu-1)$. Alternatively, we can adopt the equivalent expression $(\mathsf{L}-1)(\mu-1)$, since we have assumed that $M$ and $\mathsf{L}$ are equal quantities, thus they can be used interchangeably. Similarly to the second constituent encoder of the parent turbo code, the second constituent encoder of the pseudo-randomly punctured turbo code also generates a total of $(N-\mathsf{L})$ parity check sequences having weight $z_{\min}$, which is equivalent to $\mathsf{L}(\mu-1)$ since $N = \mu\mathsf{L}$.

Based on these deductions, we can compute the number $B^{\text{Pseudo}}_{2,d_{\text{free,eff}}}$ of codeword sequences having weight $d^{\text{Pseudo}}_{\text{free,eff}}$, generated by a rate-1/2 pseudo-randomly punctured PCCC, as follows

$$B^{\text{Pseudo}}_{2,d_{\text{free,eff}}} = \frac{[(L-1)(\mu-1)]\, L(\mu-1)}{\binom{N}{2}}, \tag{6.27}$$

which collapses to

$$B^{\text{Pseudo}}_{2,d_{\text{free,eff}}} = \frac{2(L-1)(\mu-1)^2}{\mu(\mu L-1)}. \tag{6.28}$$

The free effective distance $d^{\text{Pseudo}}_{\text{free,eff}}$ can be expressed in terms of the free effective $d^{\text{Parent}}_{\text{free,eff}}$, if we subtract (6.26) from (6.11)

$$d^{\text{Pseudo}}_{\text{free,eff}} = d^{\text{Parent}}_{\text{free,eff}} - (2 + 2^{\nu-2}). \tag{6.29}$$

Parameter $B^{\text{Pseudo}}_{2,d_{\text{free,eff}}}$ can also be represented in terms of $B^{\text{Parent}}_{2,d_{\text{free,eff}}}$, if we divide (6.28) with (6.12)

$$\begin{aligned} B^{\text{Pseudo}}_{2,d_{\text{free,eff}}} &= \left(\frac{L-1}{L}\right) B^{\text{Parent}}_{2,d_{\text{free,eff}}} \\ &= \left(\frac{2^\nu-2}{2^\nu-1}\right) B^{\text{Parent}}_{2,d_{\text{free,eff}}} \\ &< B^{\text{Parent}}_{2,d_{\text{free,eff}}}. \end{aligned} \tag{6.30}$$

We are now in the position to investigate when a rate-1/2 pseudo-randomly punctured PCCC exhibits a lower bound than its rate-1/3 parent PCCC, invoking condition (6.4)

$$B^{\text{Pseudo}}_{2,d_{\text{free,eff}}} Q\left(\sqrt{\frac{2(1/2)\, E_b}{N_0}\, d^{\text{Pseudo}}_{\text{free,eff}}}\right) < B^{\text{Parent}}_{2,d_{\text{free,eff}}} Q\left(\sqrt{\frac{2(1/3)\, E_b}{N_0}\, d^{\text{Parent}}_{\text{free,eff}}}\right). \tag{6.31}$$

We proved in (6.30) that $B^{\text{Pseudo}}_{2,d_{\text{free,eff}}} < B^{\text{Parent}}_{2,d_{\text{free,eff}}}$, thus both terms can be removed from inequality (6.31). If we substitute $d^{\text{Pseudo}}_{\text{free,eff}}$ with its equivalent, based on (6.29), the condition becomes

$$Q\left(\sqrt{\frac{E_b}{N_0}\, \left(d^{\text{Parent}}_{\text{free,eff}} - 2 - 2^{\nu-2}\right)}\right) < Q\left(\sqrt{\frac{2E_b}{3N_0}\, d^{\text{Parent}}_{\text{free,eff}}}\right). \tag{6.32}$$

Using property (6.18) of monotonic decreasing functions, we conclude that (6.32) holds true, only if inequality

$$d^{\text{Parent}}_{\text{free,eff}} - 2 - 2^{\nu-2} > \frac{2}{3}\, d^{\text{Parent}}_{\text{free,eff}}, \tag{6.33}$$

which is equivalent to

$$d_{\text{free,eff}}^{\text{Parent}} > 6 + 3(2^{\nu-2}), \tag{6.34}$$

is satisfied. However, we have shown in (6.11) that $d_{\text{free,eff}}^{\text{Parent}} = 6 + 2^{\nu}$, which can be rewritten as $d_{\text{free,eff}}^{\text{Parent}} = 6 + 4(2^{\nu-2})$, thus $d_{\text{free,eff}}^{\text{Parent}}$ is always greater than $6 + 3(2^{\nu-2})$.

The outcome of this investigation instructs us that a rate-1/2 pseudo-randomly punctured turbo code will always be expected to yield a lower bound on the bit error probability than that of its rate-1/3 parent turbo code, hence it will achieve a superior performance, provided that ML soft decoding is used.

## 6.4 Convergence Behavior Analysis using EXIT Charts

For an increasing number of iterations, the union bound on the bit error probability for ML soft decoding provides an accurate estimate of the suboptimal iterative decoder performance in the error floor region, as we demonstrated in Chapters 3 and 4. The error floor can also be estimated by the union bound approximation presented in Chapter 5, when the turbo code under consideration employs a long interleaver. However, when puncturing occurs, we need to explore whether the performance of the iterative decoder will eventually converge towards the union bound. For this reason, we use extrinsic information transfer charts, or EXIT charts for brevity, introduced by ten Brink [34] in 2001. EXIT chart analysis investigates the performance of a turbo code in the waterfall region and accurately predicts the convergence behavior of the iterative decoder for very large interleaver sizes (e.g., $N = 10^6$ bits).

In [34], ten Brink uses the concept of mutual information to describe the flow of extrinsic information through each component soft-input soft-output decoder of the iterative decoder. For this reason, we first review the concept of mutual information, before proceeding to EXIT chart analysis. In general, if $X$ and $Y$ are two random variables with possible outcomes $x_i$, $i = 1, 2, \ldots n_x$ and $y_j$, $j = 1, 2, \ldots n_y$, respectively, the information content provided by the occurrence of the event $Y = y_j$ about the event $X = x_i$ is called *mutual information* between $x_i$ and $y_j$ [15], and is defined as

$$\begin{aligned} I(x_i; y_j) &= \log_2 \frac{P(x_i|y_j)}{P(x_i)} \\ &= \log_2 \frac{P(x_i, y_j)}{P(x_i)P(y_j)}. \end{aligned} \tag{6.35}$$

The *average mutual information* between random variables $X$ and $Y$ can be obtained by weighting $I(x_i;y_j)$ by the probability of occurrence of the joint event and summing over all possible joint events [15], i.e.,

$$\begin{aligned} I(X;Y) &= \sum_{i=1}^{n_x}\sum_{j=1}^{n_y} P(x_i,y_j)\, I(x_i;y_j) \\ &= \sum_{i=1}^{n_x}\sum_{j=1}^{n_y} P(x_i,y_j)\log_2 \frac{P(x_i,y_j)}{P(x_i)P(y_j)}. \end{aligned} \tag{6.36}$$

When $X$ is a discrete random variable with two possible outcomes, $x_i = -1$ or $x_i = +1$, while $Y$ is a continuous random variable described by the probability density function $p(y)$, the mutual information provided about the event $X = x_i$ by the occurrence of event $Y = y$ is

$$\begin{aligned} I(x_i;y) &= \log_2 \frac{p(x_i|y)}{P(x_i)} \\ &= \log_2 \frac{p(y|x_i)}{p(y)}. \end{aligned} \tag{6.37}$$

Subsequently, the average mutual information between $X$ and $Y$ assumes the form

$$\begin{aligned} I(X;Y) &= \sum_{i=1}^{2}\int_{-\infty}^{+\infty} p(x_i,y)\log_2 \frac{p(x_i,y)}{P(x_i)p(y)}dy \\ &= \sum_{i=1}^{2}\int_{-\infty}^{+\infty} p(y|x_i)P(x_i)\log_2 \frac{p(y|x_i)}{p(y)}dy. \end{aligned} \tag{6.38}$$

If the outcomes of the discrete random variable $X$ are equiprobable, it follows that the probability of occurrence $P(x_i)$ of each outcome is 1/2. Furthermore, if the conditional probability density function $p(y|x_i)$ is Gaussian with mean $x_i$ and variance $\sigma^2$, we may express $p(y)$ as $0.5\,[p(y|X=-1) + p(y|X=+1)]$ [15]. Thus, the average mutual information obtained from (6.38) becomes

$$I(X;Y) = \frac{1}{2}\sum_{i=1}^{2}\int_{-\infty}^{+\infty} p(y|x_i)\log_2 \frac{2p(y|x_i)}{p(y|X=-1)+p(y|X=+1)}dy. \tag{6.39}$$

When the random variables $X$ and $Y$ are statistically independent, we obtain $I(X;Y)=0$. On the other hand, if the occurrence of an event $Y$ uniquely determines the occurrence of an event $X$, then $I(X;Y)=1$. In general, $0 \le I(X;Y) \le 1$.

As we explained in Chapter 2, each component SiSo decoder of the iterative decoder processes the channel observations, namely the received systematic and parity check bits of the corresponding constituent encoder, together with a-priori knowledge to produce extrinsic information on the systematic bits. The channel observations as well as the a-priori knowledge and the extrinsic information are all expressed in log-likelihood ratios. To this end, the information content of the a-priori knowledge, input to a certain SiSo decoder, is measured using the mutual information $I_A$ between the transmitted systematic bits and the a-priori input LLR values to the SiSo decoder. For very long interleavers, the a-priori LLRs remain fairly uncorrelated from the respective channel observations, over many iterations [34]. Thus, the a-priori LLR values can be modeled by a Gaussian random variable $A$, while the transmitted systematic bits can be modeled by a discrete random variable $X$. Subsequently, we can use (6.39) to compute $I_A = I(X; A)$, where $Y$ has been replaced by $A$. Mutual information is also used to quantify the extrinsic output information. The probability density functions of the extrinsic LLR values approach Gaussian-like distributions with an increasing number of iterations [34]. For this reason, we can also use a Gaussian random variable $E$ to model the extrinsic LLR values, while $X$ still denotes the transmitted bits. The mutual information $I_E = I(X; E)$ between the transmitted systematic bits and the extrinsic output LLRs of a particular SiSo decoder can be computed elaborating on (6.39).

Although separate expressions for the a-priori input $I_A$ and the extrinsic output $I_E$ were obtained for a particular SiSo decoder, it is of interest to express $I_E$ as a function of $I_A$, i.e.,

$$I_E = T(I_A), \tag{6.40}$$

for a fixed $E_b/N_0$ value. Unfortunately, $T(I_A)$ cannot be expressed in closed form but can be determined by Monte Carlo simulations, as described in [34]. Transfer characteristics $I_E = T(I_A)$ for various $E_b/N_0$ values are given in Fig.6.2. In this example, the exact log-MAP decoding algorithm is applied to a rate-2/3 punctured systematic RSC(1,5/7) code of memory 2, which is a constituent code of the rate-1/2 punctured systematic PCCC(1,5/7,5/7). We observe that, as the value of the mutual information $I_A$ at the input of the component decoder increases, the confidence of the decoder on the known transmitted bits becomes higher, the decoder generates more accurate estimates of the transmitted sequence, which in turn, lead to a growing mutual information $I_E$.

To account for both component decoders of a suboptimal iterative decoder, we need to consider the mutual information $I_{A_1}$ and $I_{E_1}$ at the input and the output, respectively, of the first SiSo decoder, as well as the mutual information $I_{A_2}$ and $I_{E_2}$ of the second SiSo decoder. Both decoder characteristics are plotted in a single diagram, referred to as EXIT chart. In Fig.6.3, the EXIT chart for the rate-1/2 punctured systematic PCCC(1,5/7,5/7) using an interleaver size of $10^6$ bits is depicted. Note, that the axes for the transfer characteristics of the second component decoder have been swapped. The exchange of extrinsic information between the two SiSo decoders can be visualized as a decoding trajectory, if we fix the value of $E_b/N_0$ and we let $n$ to be the iteration index. Initially, the first decoder does not have any a-priori knowledge, thus $I_{A_1,1}=0$, while the second decoder uses the extrinsic output $I_{E_1,1}=T(I_{A_1,1})$ of the first decoder as a-priori knowledge, i.e., $I_{A_2,1}=I_{E_1,1}$. The extrinsic output of the second decoder, $I_{E_2,1}=T(I_{A_2,1})$, is forwarded to the first decoder to become a-priori knowledge during the next iteration, i.e., $I_{A_1,2}=I_{E_2,1}$, and so on. Convergence to the $(I_A, I_E)=(1,1)$ point, which corresponds to low values of bit error probability, occurs if the transfer characteristics do not intersect. Thus, the iteration process proceeds as long as $I_{E_2,n+1}>I_{E_2,n}$ [34].

In Fig.6.3 we observe that for $E_b/N_0$ values between 0 dB and 0.6 dB, the transfer characteristics of the two component decoders intersect, thus an increasing number of iterations does not improve the bit error probability. However, for $E_b/N_0=0.8$ dB, the decoder characteristics do not cross and the decoding trajectory manages to go through a narrow tunnel. The minimum value of $E_b/N_0$ for which convergence to low values of bit error probability is possible over many iterations, is known as *convergence threshold*, $\gamma_{\text{th}}$ [34]. Consequently, the convergence threshold, which for the case of the rate-1/2 punctured systematic PCCC(1,5/7,5/7) is $\gamma_{\text{th}}=0.8$ dB, marks the beginning of the waterfall region. For $E_b/N_0$ values higher than $\gamma_{\text{th}}$, the bit error rate performance of a turbo code starts converging towards the error floor, which is indicated by the union bound on the bit error probability.

In the previous section we demonstrated that rate-1/2 non-systematic or pseudo-randomly punctured turbo codes exhibit a lower bound on the bit error probability than that of their rate-1/3 parent turbo codes. When ML decoding is used, this is equivalent to saying that certain configurations of rate-1/2 turbo codes can achieve a better bit error rate performance than that of their rate-1/3 parent codes. However, when suboptimal iterative decoding is used, convergence towards the union bound at low bit error probabilities needs to be investigated using EXIT charts. In the following subsections, we use EXIT chart analysis to explore whether rate-1/2 non-systematic turbo codes and rate-1/2 pseudo-randomly punctured turbo codes eventually converge towards their union bound, when iterative decoding is applied. For comparison purposes, we have also explored the convergence behavior of the corresponding rate-1/3 parent turbo codes.

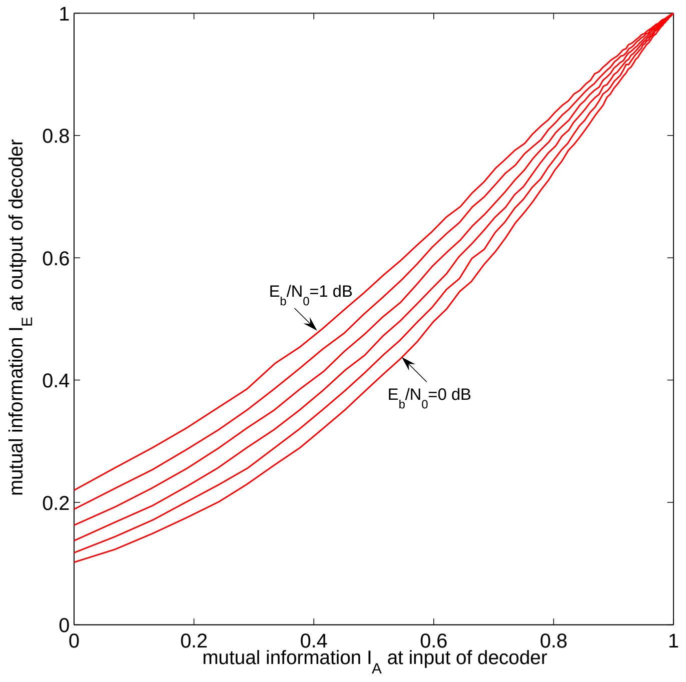


Figure 6.2: Transfer characteristics of a soft-input soft-output decoder for a rate-2/3 RSC(1,5/7,5/7) code. The value of $E_b/N_0$ ranges from 0 dB to 1 dB, with step 0.2 dB.

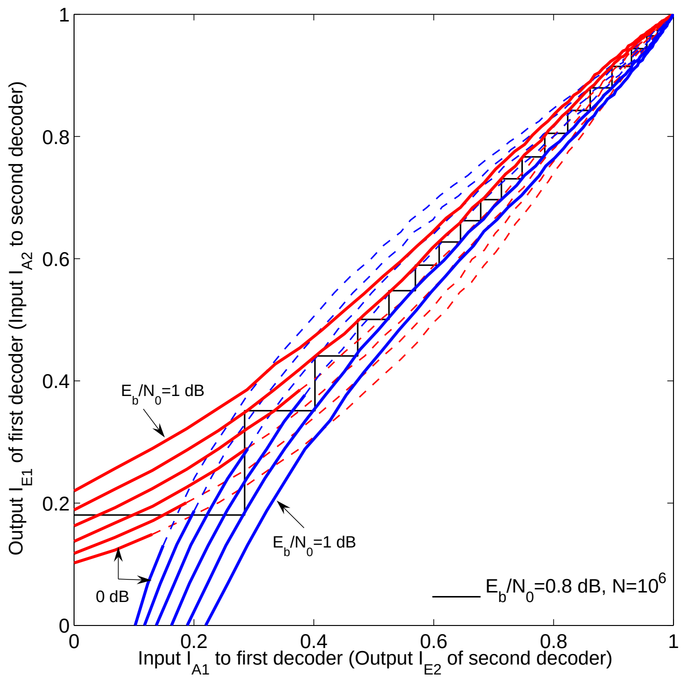
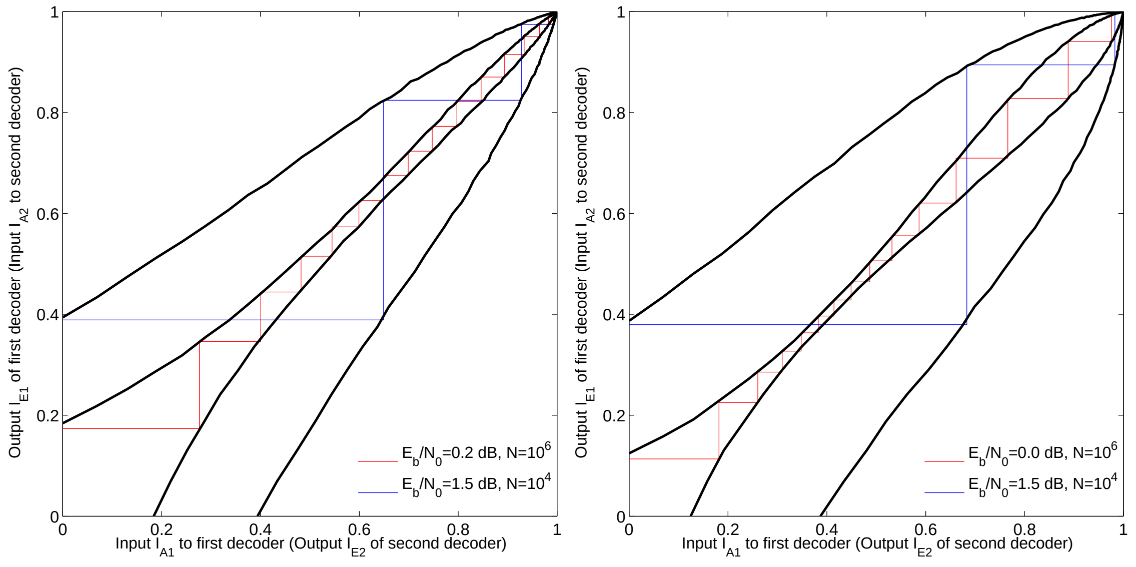


Figure 6.3: EXIT chart for the rate-1/2 Sys. PCCC(1,5/7,5/7). Transfer characteristics that intersect are plotted with dashed lines. A decoding trajectory at 0.8 dB is depicted. The exact log-MAP algorithm is applied and an interleaver size of $10^6$ bits is used.



(a) Rate-1/3 PCCC(1,5/7,5/7) (b) Rate-1/3 PCCC(1,17/15,17/15)

Figure 6.4: EXIT charts for rate-1/3 parent turbo codes.

## 6.4.1 Convergence of Rate-1/3 Parent PCCCs

EXIT charts for the two rate-1/3 parent turbo codes, employing constituent RSC codes of memory size 2 or 3, are presented in Figs 6.4(a) and 6.4(b), respectively.

In Fig.6.4(a), we observe that the transfer characteristics of the SiSo decoders, for the memory 2 constituent RSC codes, create an opening for the decoding trajectory at 0.2 dB. Hence, when an interleaver of size $N = 10^6$ is used, convergence of the rate-1/3 PCCC(1,5/7,5/7) towards low bit error probabilities is slow but possible, since the curves do not intersect. At 1.5 dB, the decoding characteristics are wide apart and convergence is fast, hence the iterative decoder has entered the error floor region. We have plotted the decoding trajectory at 1.5 dB for an interleaver size of $N = 10,000$ to illustrate that even for an interleaver shorter than $10^6$ bits, the trajectory matches fairly well with the transfer characteristics, therefore quick convergence is possible.

In Fig.6.4(b), the convergence threshold of the rate-1/3 PCCC(1,17/15,17/15) is 0 dB. A bottleneck region appears for medium values of mutual information, however the decoding trajectory manages to go through a tunnel. As in the previous case, fast convergence towards low bit error probabilities is achieved at 1.5 dB. Again, the use of a shorter interleaver, i.e., $N = 10,000$ bits, still drives the bit error rate performance of the suboptimal iterative decoder to the error floor region.

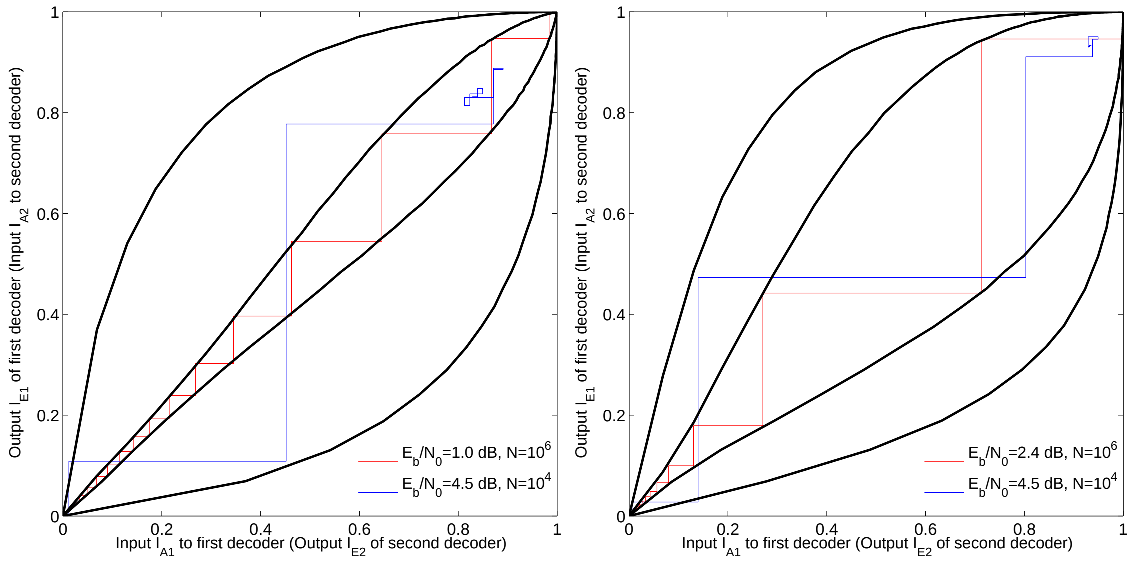
### 6.4.2 Convergence of Rate-1/2 Non-Systematic PCCCs

When the systematic output of the parent turbo code is not transmitted, so as to increase the code rate from 1/3 to 1/2, the convergence threshold rises markedly, as we see in Fig.6.5. In particular, at $E_b/N_0 = 1$ dB, the decoding trajectory of the rate-1/2 non-systematic PCCC(1,5/7,5/7), using an interleaver size of $N = 10^6$, manages to pass through a narrow opening, which appears close to the starting point $(0, 0)$ as it is illustrated in Fig.6.5(a). Low bit error probabilities are eventually achieved after 30 iterations. For comparison, iterative decoding of the rate-1/3 parent code yields a convergence threshold of 0.2 dB, while no more than 15 iterations are required to reach the error floor region. Similarly, we see in Fig.6.5(b) that iterative decoding of the rate-1/2 non-systematic PCCC(1,17/15,17/15) converges towards low bit error probabilities at a higher $E_b/N_0$ value and a larger number of iterations is required, compared with iterative decoding of the corresponding rate-1/3 parent turbo code.

We demonstrated that at $E_b/N_0$ values higher than the convergence threshold, the decoding trajectory of a rate-1/3 parent turbo code is in agreement with the transfer characteristics of the component decoders, even when an interleaver size smaller than $10^6$ bits is considered. We observe in Fig.6.5 that, contrary to their rate-1/3 parent codes, the decoding trajectories of both rate-1/2 non-systematic turbo codes die away after a few iterations, even for an $E_b/N_0$ value of 4.5 dB. We attribute this problem to the increasing correlation of extrinsic information passed between the component decoders, due to the absence of received systematic bits, which causes erroneous decisions. Thus, for small and more practical interleaver sizes, error propagation prohibits the performance of both rate-1/2 non-systematic turbo codes from converging towards the error floor, defined by the union bound.

### 6.4.3 Convergence of Rate-1/2 Pseudo-randomly Punctured PCCCs

The transmitted systematic bits of rate-1/2 pseudo-randomly punctured turbo codes assist the iterative decoder to converge earlier than rate-1/2 non-systematic turbo codes, as we observe in Fig.6.6. A comparison between the EXIT charts of pseudo-randomly punctured turbo codes and non-systematic turbo codes, reveals that the former codes require a lower $E_b/N_0$ value and less iterations in order to converge towards low bit error probabilities. Furthermore, for an increasing number of iterations, the iterative decoder quickly converges to the error floor region at $E_b/N_0 = 1.5$ dB, even when an interleaver of size $10,000$ bits is used. Thus, contrary to the



(a) Rate-1/2 NS-PCCC(1,5/7,5/7) (b) Rate-1/2 NS-PCCC(1,17/15,17/15)

Figure 6.5: EXIT charts for rate-1/2 non-systematic turbo codes.

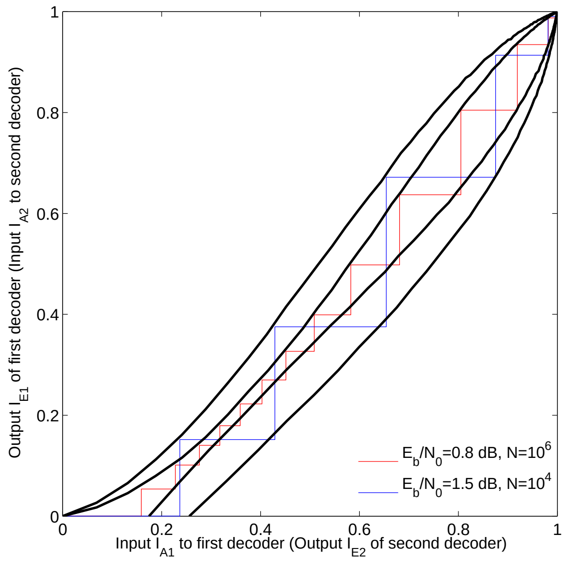
case of non-systematic turbo codes, iterative decoding of pseudo-randomly punctured turbo codes using medium to large size interleavers eventually approaches the union bound on the bit error probability.

## 6.5 Comparison of Analytic to Simulation Results

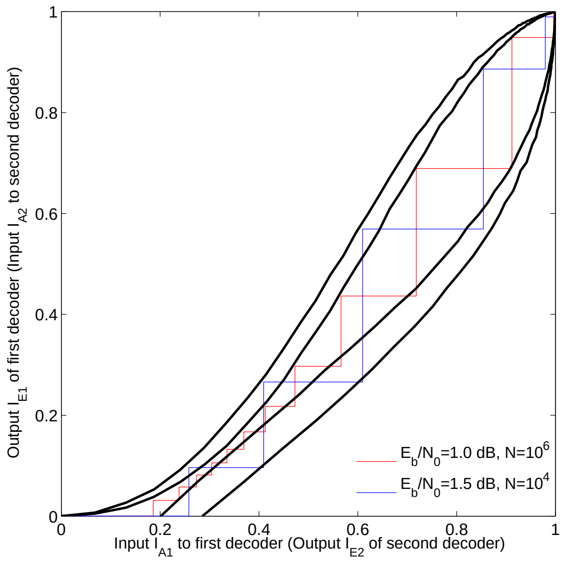
The objective of this section is to validate the results obtained from the bound approximation technique and the EXIT chart analysis by comparing them to simulation results.

Fig.6.7(a) compares bound approximations to simulation results of rate-1/3 and rate-1/2 PCCC(1,5/7,5/7) configurations in AWGN channels using iterative decoding. The component SiSo decoders employ the conventional exact log-MAP algorithm. A moderate interleaver size of $1,000$ bits was chosen, so as to allow the bit error rate performance of the turbo coding schemes to approach the corresponding bound approximations at bit error probabilities in the region of $10^{-6}$ to $10^{-7}$.

From the EXIT chart analysis, we know that the convergence thresholds for the parent PCCC, the pseudo-randomly punctured PCCC and the non-systematic PCCC are 0.2 dB, 0.8 dB and 1 dB, respectively. Subsequently, the waterfall region of the rate-1/3 parent turbo code is expected to begin first, while the waterfall region of the rate-1/2 non-systematic turbo code is expected to begin last. However, we observed that for interleavers shorter than $10^6$ bits, iterative decoding of the



(a) Rate-1/2 Pseudo PCCC(1,5/7,5/7)



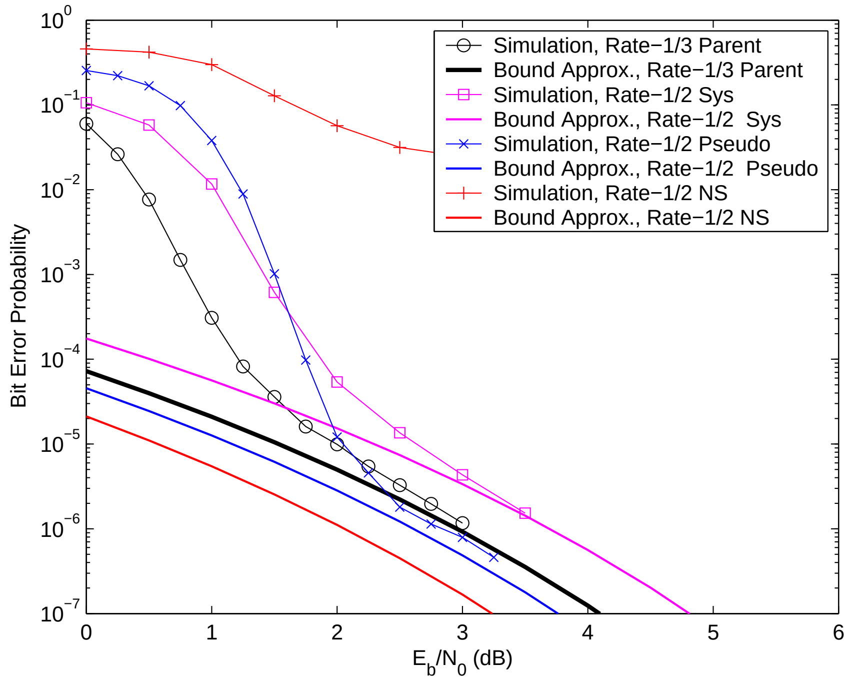
(b) Rate-1/2 Pseudo PCCC(1,17/15,17/15)

Figure 6.6: EXIT charts for rate-1/2 pseudo-randomly punctured turbo codes.

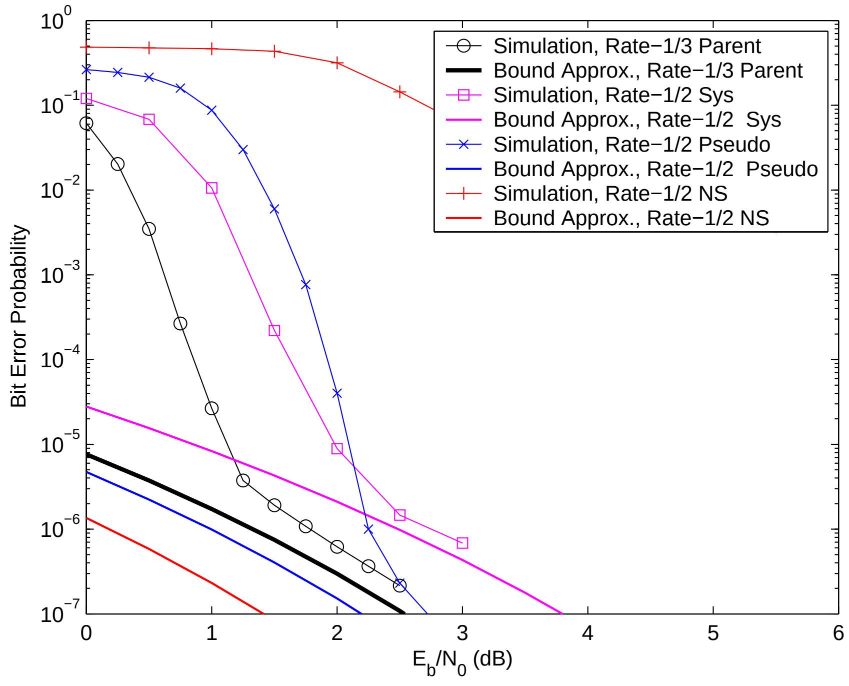
non-systematic PCCC(1,5/7,5/7) does not approach the error floor region, even for $E_b/N_0$ values much higher than the convergence threshold. We observe in Fig.6.7(a) that, indeed, performance of the rate-1/3 parent code converges to the bound approximation at low $E_b/N_0$ values, after only 8 iterations . For the same number of iterations but at higher values of $E_b/N_0$, the performance of the rate-1/2 pseudo-randomly punctured PCCC also converges towards the error floor. As expected, iterative decoding of the non-systematic PCCC, employing an interleaver of $1,000$ bits, cannot cope with the increasing correlation of extrinsic information caused by the absence of systematic bits, and eventually hits an error floor, which is much higher than that defined by the bound approximation.

From the performance analysis based on ML decoding, we showed that rate-1/2 pseudo-randomly punctured turbo codes are always expected to yield a lower union bound than that of their rate-1/3 parent codes. For an increasing number of iterations, the performance of iterative decoding approaches the union bound, if the EXIT chart of the turbo code indicates that the exchange of extrinsic information between the component decoders gradually reduces the bit error probability. Fig.6.7(a) confirms that, for $E_b/N_0$ values higher than 2.5 dB, the bit error probability of the rate-1/2 pseudo-randomly punctured turbo code is indeed lower than that of the rate-1/3 parent code, while the performance curves of both turbo codes closely follow the respective bound approximation curves.

Fig.6.7(b) explores the same trends, when PCCC(1,17/15,17/15) is used as a



(a) PCCC(1,5/7,5/7)



(b) PCCC(1,17/15,17/15)

Figure 6.7: Comparison of analytic bounds to simulation results. The exact log-MAP algorithm is applied over 8 iterations and an interleaver size of 1,000 bits is used.

parent code. As in the previous case, performance of the rate-1/2 pseudo-randomly punctured PCCC converges towards low bit error probabilities and eventually hits an error floor, which is lower than that of the rate-1/3 parent code. On the other hand, iterative decoding of the non-systematic PCCC does not approach the theoretical error floor and its error performance remains poor, despite any increase in $E_b/N_0$.

For comparison purposes, we have also included the performance of rate-1/2 systematic turbo codes in Fig.6.7. We observe that transmission of the systematic bits helps the iterative decoder to start converging to the error floor region at low values of $E_b/N_0$. However, rate-1/2 systematic turbo codes exhibit a higher error floor than that of their rate-1/3 parent codes or that of the rate-1/2 pseudo-randomly punctured turbo codes. As Blazek *et al.* [48] and Crozier *et al.* [49] reported, systematic turbo codes offer better error rate performance than partially systematic turbo codes, when they operate in the waterfall region. However, we have shown that rate-1/2 partially systematic turbo codes exist, obtained either by means of an exhaustive search or by using pseudo-random puncturing patterns, that outperform even their rate-1/3 parent codes, when they operate in the error floor region.

## 6.6 Chapter Summary

Using the bound approximation, we showed that rate-1/2 punctured turbo codes, that yield a lower union bound on the bit error probability than that of their rate-1/3 parent codes, exist. In particular, we demonstrated that rate-1/2 non-systematic turbo codes and rate-1/2 pseudo-randomly punctured turbo codes are always expected to outperform their rate-1/3 parent codes, if ML soft decoding is used.

In practice, however, suboptimal iterative decoding is employed. Using EXIT chart analysis, we showed that in a complete absence of systematic bits very long interleavers and an impractical number of decoding iterations are required to drive the bit error rate performance of a rate-1/2 non-systematic turbo code into the error floor region. On the other hand, the performance of iteratively decoded rate-1/2 partially systematic turbo codes eventually converges towards low bit error probabilities, over many iterations. Bearing this into mind, we established that rate-1/2 pseudo-randomly punctured turbo codes, which form a subset of rate-1/2 partially systematic turbo codes, not only approach the error floor region for an increasing number of iterations but yield a lower error floor than that of their rate-1/3 parent codes. Consequently, pseudo-random puncturing can be used to reduce the rate of a turbo code from 1/3 to 1/2 and at the same time achieve a coding gain at low bit error probabilities.

Chapter 7

# Conclusions

In this dissertation a number of contributions concerning turbo codes and their performance evaluation, have been made. Specifically,

- a novel approach to evaluate the transfer function of punctured and non-punctured convolutional block codes has been introduced, which is essential for the computation of an upper bound on the error rate performance of turbo codes,
- a rapid method to evaluate the most significant terms of the transfer function of punctured and non-punctured convolutional block codes has been proposed, which can be used to evaluate an accurate bound approximation on the error rate performance of turbo codes employing long interleavers,
- it has been demonstrated that rate-1/2 punctured turbo codes can achieve lower performance bounds on AWGN channels than those of their rate-1/3 parent codes. Furthermore, it has been shown that pseudo-random puncturing ensures that the error rate performance of the corresponding rate-1/2 turbo code converges towards the upper bound, at low bit error probabilities.

In particular, we introduced the concept of the augmented state diagram, based on which the full transfer function of a convolutional block code can be derived. We showed that when we puncture the output of a convolutional block encoder in order to improve bandwidth efficiency, the augmented state diagram can be modified so as to incorporate the puncturing pattern into its structure and, hence, allow computation of the transfer function of the corresponding punctured convolutional block code. Consequently, tight bounds on the average performance of ML soft decoding of turbo codes, which employ non-punctured or punctured convolutional block codes as constituent codes, can be derived. These bounds accurately predict the performance error floors of suboptimal iterative decoding, which is used in practice.

We observed that computation complexity of the transfer function may become overwhelming as the interleaver size increases, when punctured turbo codes are considered. To alleviate this problem, we proposed a rapid technique, which exploits the structural properties of constituent RSC encoders, to obtain only those terms of the transfer function that significantly contribute to the performance bound on the bit error probability, when long interleavers are employed. Hence, a bound approximation can be used, instead of the exact union bound, to give us insight on the ML decoding performance of turbo codes using long interleavers.

Using the bound approximation, we demonstrated that, when ML soft decoding is used, there exist rate-1/2 punctured turbo codes yielding a lower bound on the bit error probability than that of their rate-1/3 parent codes. However, when suboptimal iterative decoding is employed, puncturing the systematic output of a turbo encoder increases the convergence threshold. Moreover, a complete absence of systematic bits requires very long interleavers and an impractical number of decoding iterations to drive the bit error rate performance of a rate-1/2 non-systematic turbo code into the error floor region. Nevertheless, we showed that iterative decoding of a family of rate-1/2 partially systematic turbo codes, which we named pseudo-randomly punctured turbo codes, does approach the error floor region for an increasing but practical number of iterations and does yield a lower error floor than that of their rate-1/3 parent codes. Therefore, we established that it is possible to reduce the rate of a turbo code from 1/3 to 1/2 and at the same time achieve a coding gain at low bit error probabilities.

Looking forward, a number of themes emerge from this work as well as from the ever-growing body of literature.

- An investigation concerning the puncturing patterns employed at the output of a convolutional or turbo encoder and computation of the performance bound on the bit error probability, could help us identify coded bits that have a major impact on the decoding performance. Hence, development of efficient mapping techniques according to which coded bits, considered to be more important than others to the decoding process, would be mapped to constellation symbols that offer improved noise protection, would offer a performance gain to coded-modulation schemes.

- As in the case of iterative decoding of turbo codes, we expect that an upper bound on the error probability of soft ML decoding of a coded-modulation scheme, would give us insight on the error floor of suboptimal iterative decoding/demapping of the same scheme. Our proposed technique could be used to compute the transfer function of the outer code, which could be a convolutional or turbo code, punctured or non-punctured. Derivation of a tight bound on the bit error rate performance of iterative decoding-demapping of the complete coded-modulation scheme, composed of an outer code, a random interleaver and an inner mapper, would be a valuable topic of research.

- Although the main concern of this dissertation was the performance evaluation of punctured turbo codes on AWGN channels, investigation of their performance on quasi-static fading channels would be essential for fixed wireless access systems. It is expected that the performance of systems characterized by high spatial diversity would depend on the transfer function of the turbo encoder, since the turbo-coded multiple-input multiple-output system would effectively collapse into a single-input single-output system and the underlying channel would approach the classical AWGN channel. However, the contribution of the convergence characteristics of the iterative decoder to the overall error rate performance of systems characterized by low spatial diversity needs to be explored.

- Cooperative networks, a new diversity method to provide spectral efficiency and link reliability in multi-user wireless environments, is another area which would benefit from puncturing schemes. In a cooperative network, a user transmits data to a destination as well as to adjacent users, who relay their partner's data. In the case of regenerative relaying, each cooperating user decodes, re-encodes and retransmits the received data. Effectively, two or more cooperative users could be seen as constituent encoders of a parallel concatenated scheme. Puncturing could be used to reduce the power and bandwidth required to forward a partner's frame. Furthermore, design of puncturing patterns based on our proposed approach, could fine-tune the performance of a link, for a given range of signal-to-noise ratios.

- Another interesting area of research would be the investigation of performance vs. complexity trade-offs in coded systems on quasi-static fading channels. A complete framework for the analysis of the computational complexity of various decoding algorithms and performance comparison of low-diversity systems using either convolutional codes or turbo codes of similar decoding complexity, needs to be explored. Future work in this direction would unveil whether convolutional codes are potentially better candidates than turbo codes on quasi-static fading channels with limited antenna diversity.

Appendix A

# Software Tools

## A.1 Introduction

In this appendix, we provide a brief description of the software tools developed during the course of our research. The tools can be divided into three main packages, depending on their functionality:

- **System modeling and measurement of the bit error probability:** Collection of libraries used to simulate a turbo-coded communication system, including the AWGN channel. The bits in error at the output of the receiver are counted and the actual bit error probability (BEP) is calculated.

- **Computation of Performance Bounds on the BEP:** Group of programmes to compute the exact and approximate theoretical bounds on the BEP of PCCCs. These bounds coincide with the performance error-floor, when ML decoding is used.

- **EXIT Chart Analysis:** A package that investigates whether the performance of iterative decoding of a PCCC converges towards low bit error probabilities and thus, approaches the theoretical ML bound.

The first collection of libraries form a stand-alone package, entirely developed in ANSI C. Although C is a powerful programming language, it cannot match the user friendliness and the wealth of mathematical libraries that Matlab provides. For this reason, the second and third packages require the Matlab programming environment. However, Matlab programmes are not optimum from a speed of execution perspective, hence simulation time could be very long. In order to achieve a trade-off between performance and complexity, part of the code of the second and third packages has been developed in C and then converted to Matlab libraries, using the "mex" internal command. Note that all packages can be executed in machines running the UNIX, LINUX or WINDOWS operating system.

## A.2 System Modeling and BEP Measurement

The system model that we have used throughout this thesis consists of a turbo encoder, an AWGN channel and an iterative (turbo) decoder. The nominal rate of the turbo code is 1/3, however higher rates can be obtained if puncturing is applied. Each component of the system model has been developed entirely in C. Below, we briefly describe the inputs, outputs and functionality of each component:

1. **Turbo Encoder**

   Implements the parallel concatenation of two rate-1/2 convolutional encoders separated by a pseudo-random interleaver. The generator vectors are manually set. A puncturing pattern also needs to be provided, if the user aims for a rate higher than 1/3. The turbo encoder processes the input information bits and generates a sequence of coded polar symbols.

2. **Channel Realization**

   This function impairs the transmitted polar symbols by adding white Gaussian noise to them, based on the given code rate and desired $E_b/N_0$ value. The white Gaussian noise is generated using the Box-Muller transform. This method, described in [55], transforms uniformly distributed random numbers into normally distributed random numbers.

3. **Iterative Decoder**

   The received sequence of polar symbols is iteratively decoded and a sequence of bits, which are estimates of the transmitted information bits, is generated. The user should provide the generator polynomials and define the number of iterations as well as the decoding algorithm to be used by the SiSo component decoders (i.e., exact log-MAP, log-MAP, max-log-MAP or SOVA). If the rate of the code is higher than 1/3, the puncturing pattern should also be made available to this function, so it can insert zeros in the correct positions of the received sequence.

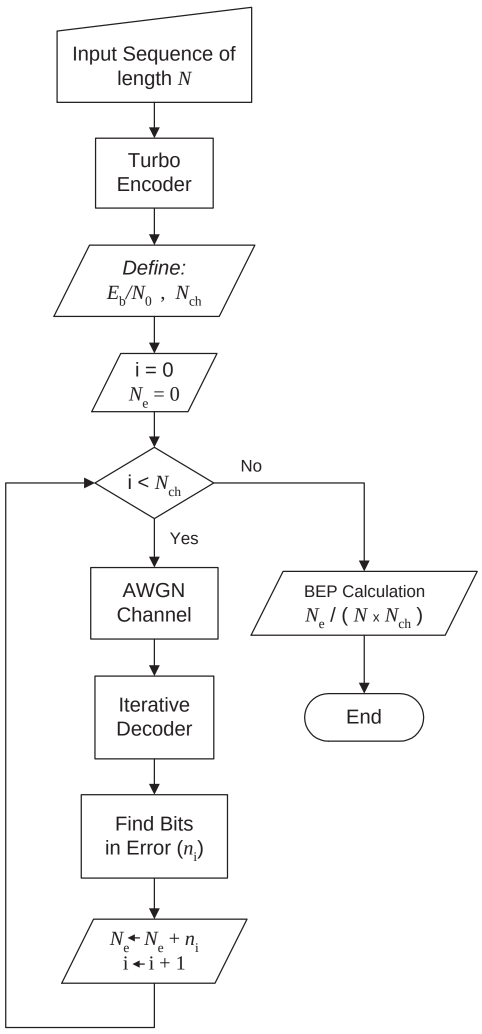

Input Sequence of length $N$
Turbo Encoder
Define: $E_b/N_0$ , $N_{ch}$
i = 0
$N_e = 0$
i < $N_{ch}$
No
Yes
AWGN Channel
BEP Calculation $N_e$ / ( $N$ x $N_{ch}$ )
Iterative Decoder
End
Find Bits in Error ($n_i$)
$N_e \leftarrow N_e + n_i$
$i \leftarrow i + 1$

Figure A.1: Configuration for the measurement of the bit error probability of a turbo-coded system. It is assumed that all variables, such as the generator polynomials of the encoder or the number of decoding iterations, have been defined.

In order to measure the bit error probability of a turbo-coded system, we use the configuration presented in Fig.A.1. In particular, we transmit a fixed frame of symbols over a large number of different channel realizations, denoted as $N_{ch}$. At the end of the $i$-th realization, where $i = 1, \ldots, N_{ch}$, we compare the decoded bits at the output of the iterative decoder to the information bits at the input of the turbo encoder, and we store the number of decoded bits in error, $n_i$. At the end of all realizations, we divide the total number of erroneous bits, $N_e = \sum_i n_i$, by the length $N$ of the input information sequence multiplied by the number of channel realizations, i.e.,

$$\text{BEP} = \frac{N_e}{N \times N_{ch}}.$$

If we want to obtain the BEP for a range of $E_b/N_0$ values, we repeat the same process for each value of $E_b/N_0$.

## A.3 Performance Bounds on the BEP

The collection of programmes for computing bounds on the BEP of a turbo code can be divided into two separate software packages; the first package determines the exact union bound on the BEP of a turbo code, whilst the second package computes an approximation to the union bound.

### A.3.1 Exact Union Bound Calculation

It is assumed that the reader is familiar with the content of Chapters 3 and 4, in which the derivation of the transfer function of a constituent convolutional block encoder, either non-punctured or punctured, respectively, is described. The process of obtaining the union bound on the BEP of a PCCC can be divided into six stages, each one of which corresponds to a programme. The functionality of each programme can be summarized as follows:

1. **State Equations Solver** (developed in Matlab)

   This programme constructs the augmented state diagram of a convolutional block code based on its memory size, its feedback generator vector and its feedforward generator vector. The self-loop at the zero state is eliminated and the number of remergings into the zero state are taken into account (see Sections 3.3.2 and 3.3.3). If the code is punctured, the puncturing pattern should also be provided to allow proper simplification of the augmented state diagram (see Section 4.4). The state equations are solved for the ratio $X_E/X_S$ and two polynomials, $y(W, U, Z, L, \Omega)$ and $x(W, U, Z, L, \Omega)$ that represent the numerator and denominator, respectively, of the intermediate transfer function $T(W, U, Z, L, \Omega)$, are produced.

2. **Polynomial Division** (developed in Matlab)

   Returns the most significant terms of a polynomial division. Terms having an output weight less than the truncation output weight $d_T$, which is manually defined, are considered significant. Here, the polynomials to be divided are $y(W, U, Z, L, \Omega)$ and $x(W, U, Z, L, \Omega)$. As a consequence, the intermediate transfer function $T(W, U, Z, L, \Omega)$ is obtained in polynomial form.

3. **Derivation of the Transfer Function of a Convolutional Block Code** (developed in C)

   Given the intermediate transfer function $T(W, U, Z, L, \Omega)$, this script computes the transfer function $B^{\mathcal{C}}(W, U, Z)$ of the convolutional block code for a specific input block length $N$ (see Section 3.3.2). If the code is punctured, the puncturing period should also be provided (see Section 4.4).

4. **Computation of the Transfer Function of a PCCC** (developed in C)

   The transfer functions of two convolutional block codes are combined to generate the average transfer function of a PCCC, $B^{\mathcal{P}}(W, U, Z)$, which uses the two convolutional block codes as constituent codes and employs a uniform interleaver of size $N$ (see Section 2.7.2).

5. **Derivation of the Union Bound Coefficients** (developed in Matlab)

   The union bound coefficients $D_d$ are derived from $B^{\mathcal{P}}(W, U, Z)$, the transfer function of the PCCC. Those coefficients that correspond to an output weight $d < d_T$ are directly computed, whilst the remaining coefficients are estimated by means of an extrapolation (see Section 3.4).

6. **Union Bound Calculation** (developed in Matlab)

   Having obtained coefficients $D_d$ for every allowable value of the output weight $d$, the union bound is calculated and plotted for a range of $E_b/N_0$ values.

In stages 1 to 4, we have used $k$-dimensional ($k$-D) arrays to represent polynomials using $k$ indeterminate variables. For example, if a polynomial is a function of two indeterminate variables, namely $W$ and $Z$, and $\alpha W^i Z^j$ is a term in the polynomial, the element in position $(i{+}1, j{+}1)$ of the equivalent 2-D array will assume the value of coefficient $\alpha$. Hence, if $3 + 2W + WZ$ is the polynomial in question, coefficients 3, 2 and 1 will be mapped to positions $(1, 1)$, $(2, 1)$ and $(2, 2)$, respectively, thus the equivalent 2-D array will assume the form

$$\begin{bmatrix} 3 & 0 \\ 2 & 1 \end{bmatrix}.$$

Polynomial arithmetic has also been defined accordingly.

Calculation of the transfer functions of both constituent convolutional block codes and PCCCs, requires computation of factorials of very big integers. For this reason, stages 3 and 4 have been developed in C to facilitate the representation and

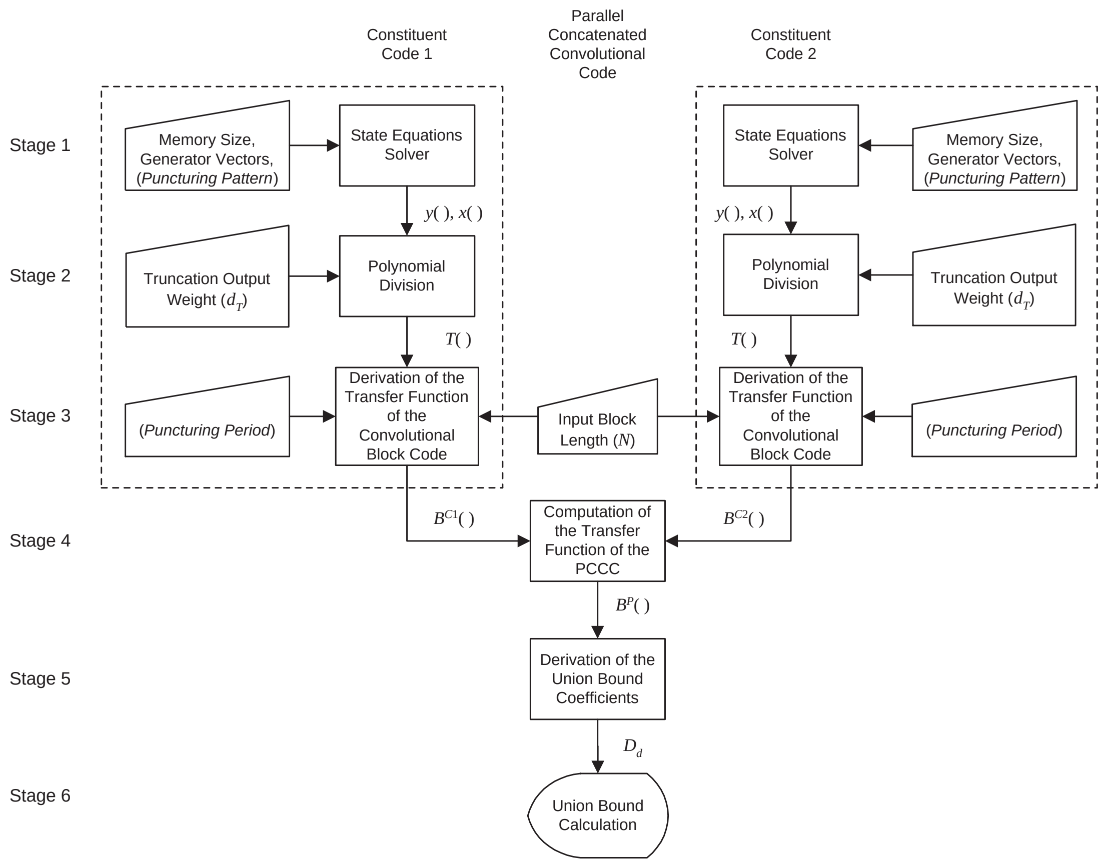

Constituent Code 1
Parallel Concatenated Convolutional Code
Constituent Code 2
Stage 1
Stage 2
Stage 3
Stage 4
Stage 5
Stage 6
Memory Size, Generator Vectors, (Puncturing Pattern)
State Equations Solver
$y(\,), x(\,)$
Truncation Output Weight ($d_T$)
Polynomial Division
$T(\,)$
(Puncturing Period)
Derivation of the Transfer Function of the Convolutional Block Code
Input Block Length ($N$)
$B^{C1}(\,)$
$B^{C2}(\,)$
Computation of the Transfer Function of the PCCC
$B^{P}(\,)$
Derivation of the Union Bound Coefficients
$D_d$
Union Bound Calculation

Figure A.2: Stages for obtaining the union bound on the BEP of a turbo code.

handling of numbers that cannot be stored in the conventional 32-bit buffers. Such non-negative integers are larger than $(2^{32}-1)=4,294,967,295$. In particular, we have used a non-decimal notation with a manually-set base. A total of 8 bits are allocated to the base, hence bases larger than 255 are not permitted. The number of digits in each non-decimal integer is manually defined but remains fixed during the execution of the C programmes. In the following example,

$$4,560,252,227_{10} = (0)(17)(67)(34)(25)(108)_{127},$$

a decimal integer longer than 32 bits has been expressed as a 6-digit base-127 integer. Consequently, a total of $6\times8=48$ bits have been allocated to the non-decimal integer. Functions that enable basic operations, such as addition, subtraction, multiplication and division, between non-decimal numbers have also been developed.

A flow chart depicting the six stages for obtaining the union bound on the BEP of a turbo code is presented in Fig.A.2. Note that stages 1 to 3 need to be executed twice, i.e., once for each constituent convolutional block code, unless the PCCC is symmetric.

### A.3.2 Calculation of a Union Bound Approximation

Computation of the exact union bound on the BEP could be a time consuming process, especially when the memory size of the encoder is large and the period of the puncturing pattern is long. In Chapter 5 we demonstrated that, for large interleaver sizes, the union bound on the BEP of a turbo code can be approximated by the probability of all erroneous codeword sequences with information weight two. The union bound approximation, denoted as $P(2)$, is not a function of the complete transfer function of each constituent encoder, hence construction of their augmented state diagrams and solution of their respective state equations is not required. Calculation of $P(2)$ only requires knowledge of the dominant conditional weight enumerating function (CWEF) of each constituent encoder, which is not a computationally intensive process since it neither involves division of polynomials nor generates very big integers. Consequently, the package for obtaining an approximate union bound on the BEP of a PCCC has been divided into only three stages, which have been entirely developed in Matlab. A short description of each stage is given below:

1. **Derivation of the Dominant CWEF of a Convolutional Block Code**

   Given the length of the input information sequence as well as the memory size and generator vectors of the convolutional block code, this function implements the methodology described in Chapter 5 to compute and return the dominant CWEF of the code (see Section 5.3 for non-punctured codes or Section 5.4 for punctured codes). If puncturing is applied, the puncturing pattern should also be provided.

2. **Derivation of the Dominant CWEF of a PCCC**

   At this stage, the dominant CWEFs of two constituent convolutional block encoders are combined to generate the average dominant CWEF of the respective PCCC (see Section 5.2).

3. **Calculation of the Union Bound Approximation**

   The last stage of this package uses the coefficients of the dominant CWEF of the PCCC under consideration, and computes $P(2)$ for a particular value of $E_b/N_0$ (see Section 5.2). If the PCCC is punctured, its code rate is derived from the puncturing pattern.

The three stages for obtaining an approximate union bound on the BEP of a turbo code are presented in Fig.A.3. Note that stage 1 is repeated twice, i.e., once for each constituent convolutional block code, unless the PCCC is symmetric.

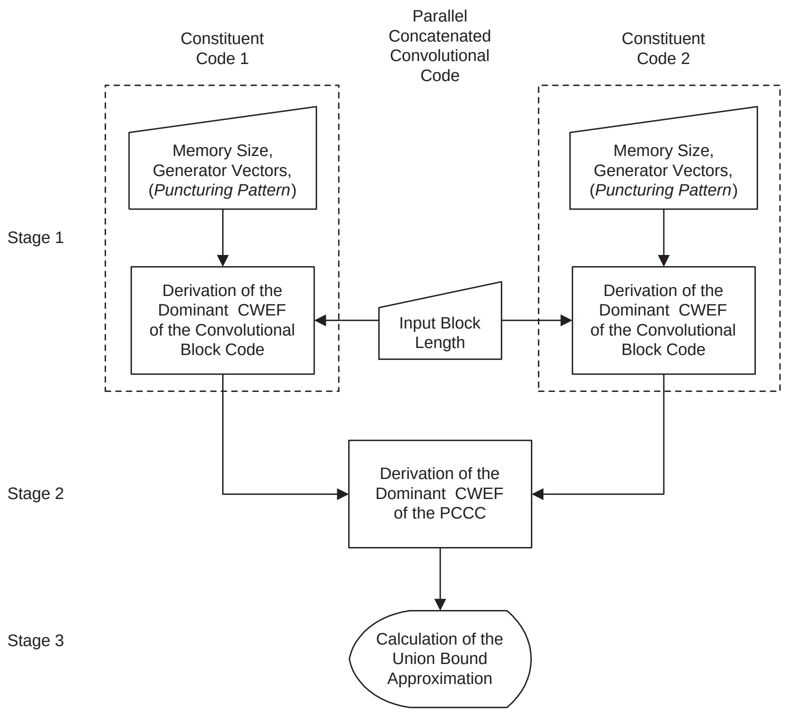

Constituent Code 1
Parallel Concatenated Convolutional Code
Constituent Code 2
Memory Size, Generator Vectors, (*Puncturing Pattern*)
Memory Size, Generator Vectors, (*Puncturing Pattern*)
Stage 1
Derivation of the Dominant CWEF of the Convolutional Block Code
Input Block Length
Derivation of the Dominant CWEF of the Convolutional Block Code
Stage 2
Derivation of the Dominant CWEF of the PCCC
Stage 3
Calculation of the Union Bound Approximation

Figure A.3: Stages for obtaining an approximate union bound on the BEP of a PCCC.

## A.4 EXIT Chart Analysis

EXIT charts analysis investigates the convergence behavior of an iterative decoder in the waterfall region, as we described in Chapter 6. In particular, the transfer characteristics of each SiSo decoder, conveyed by the relation between the mutual information $I_A$ at the input the decoder and the mutual information $I_E$ at the output of the decoder, are obtained separately and plotted in a single diagram. In this section we describe the configuration that we have implemented, in order to obtain the transfer characteristics of a constituent SiSo convolutional decoder. The components of the configuration, which is presented in Fig.A.4, are as follows:

1. **Convolutional Encoder** (developed in C)

   Implements a rate-1/2 encoder, which processes the input information bits and generates a coded sequence of polar symbols. The generator polynomials are manually defined. Rates higher than 1/2 can be obtained, if a puncturing pattern is provided.

2. **Gaussian Noise Generator** (developed in Matlab)

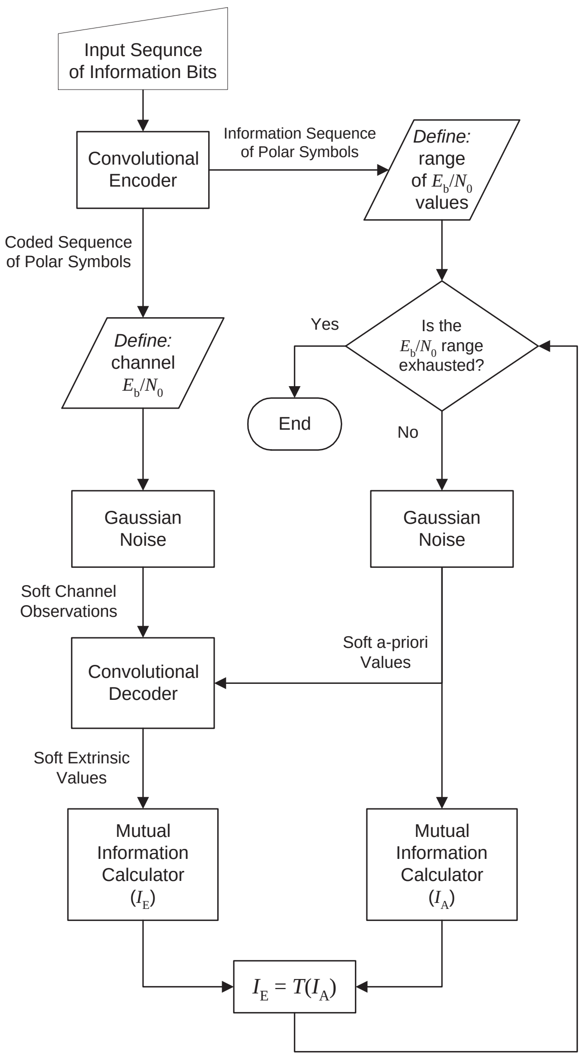

Input Sequnce of Information Bits
Convolutional Encoder
Information Sequence of Polar Symbols
Define: range of $E_b/N_0$ values
Coded Sequence of Polar Symbols
Define: channel $E_b/N_0$
Is the $E_b/N_0$ range exhausted?
Yes
No
End
Gaussian Noise
Gaussian Noise
Soft Channel Observations
Soft a-priori Values
Convolutional Decoder
Soft Extrinsic Values
Mutual Information Calculator ($I_E$)
Mutual Information Calculator ($I_A$)
$I_E = T(I_A)$

Figure A.4: Derivation of the transfer characteristics of a constituent convolutional decoder.

Impairs a sequence of polar symbols by adding normally distributed Gaussian noise, based on the given code rate and desired $E_b/N_0$ value.

3. **Convolutional Decoder** (developed in C)

   Considers the a-priori information and implements the exact log-MAP decoding algorithm on the received sequence of soft channel observations. This function generates a sequence of soft extrinsic values.

4. **Mutual Information Calculator** (developed in Matlab)

   Uses the soft values of a Gaussian random variable (i.e., either the a-priori values or the extrinsic values) and combines them with the input information bits to derive their mutual information (i.e., $I_A$ or $I_E$), based on the expressions presented in Chapter 6 (see Section 6.4).

Let us now concentrate on the left branch of the diagram in Fig.A.4. The input sequence of information bits is encoded and transmitted over an AWGN channel. The $E_b/N_0$ at the input of the convolutional decoder is fixed to a desired value. Initially, we assume that no a-priori information is available. The convolutional decoder processes the received sequence of channel observations and generates a sequence of soft extrinsic values, which is input to a mutual information calculator. A particular value of mutual information $I_E$ is obtained at the output of the calculator.

The right branch of the diagram is used to provide a-priori knowledge of the original information sequence to the convolutional decoder. The quality of the a-priori information at the input of the decoder is improved by gradually increasing its $E_b/N_0$ value, and, as a consequence, the value of the mutual information $I_A$ also increases. The impact of the soft a-priori values at the input of the decoder on the soft extrinsic values at the input of the decoder, or equivalently, the relation between the mutual information $I_A$ and the mutual information $I_E$, is recorded. Consequently, for a range of $E_b/N_0$ values, a range of $I_A$ values is obtained and, in turn, each $I_A$ value is mapped to a particular $I_E$ value. Hence, when all $E_b/N_0$ values in the desired range are tried, our objective to express $I_E$ as a function of $I_A$, i.e., $I_E = T(I_A)$, is achieved.

The same process needs to be repeated for the second component SiSo decoder, baring in mind that it operates in the interleaved domain. When the transfer characteristics of both constituent decoders are obtained and plotted in the same diagram to give the EXIT chart of the iterative decoder, convergence towards low bit error probabilities can be investigated.

## A.5 Summary

In this appendix, we classified and briefly described the software tools we developed in Matlab and C, so as to analyze and evaluate the bit error performance of punctured and non-punctured turbo codes.

# References

[1] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, pp. 379–427, 1948.

[2] R. Hamming, "Error detecting and error correcting codes," *Bell System Technical Journal*, pp. 147–160, 1950.

[3] M. Golay, "Notes on digital coding," *Proc. IEEE*, vol. 37, p. 657, 1949.

[4] P. Sweeney, *Error Control Coding: An Introduction.* New York: Prentice Hall, 1991.

[5] P. Ellias, "Coding for noisy channels," *IRE Nat. Conv. Record*, vol. 3, no. 4, pp. 37–46, 1955.

[6] A. J. Viterbi, "Error bound for convolutional codes and asymptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 260–269, Apr. 1967.

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

[8] G. D. Forney, *Concatenated Codes.* Cambridge, Mass.: MIT Press, 1966.

[9] S. Wicker, *Error Control Systems for Digital Communications and Storage.* Englewood Cliffs, NJ: Prentice Hall, 1995.

[10] G. Ungerböck, "Channel coding with multilevel/phase signals," *IEEE Trans. Inform. Theory*, vol. IT-28, no. 1, pp. 55–67, Jan. 1982.

[11] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in *Proc. IEEE International Conference on Communications (ICC'93)*, Geneva, Switzerland, May 1993, pp. 1064–1070.

[12] E. Guizzo, "Closing in on the perfect code," *IEEE Spectrum*, pp. 28–34, Mar. 2004.

[13] R. Gallager, "Low-density parity check codes," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, US, 1963.

[14] D. J. C. MacKay and R. M. Neal, "Good codes based on very sparse matrices," in *Proc. 5th IMA Conference on Cryptography and Coding*, Cirencester, UK, Dec. 1995, p. 100111.

[15] J. G. Proakis, *Digital Communications*, 4th ed.

[16] S. Benedetto and G. Montorsi, "Unveiling turbo codes: Some results on parallel concatenated coding schemes," *IEEE Trans. Inform. Theory*, vol. 42, no. 2, pp. 409–429, Mar. 1996.

[17] T. M. Duman and M. Salehi, "New performance bounds for turbo codes," *IEEE Trans. Commun.*, vol. 46, no. 6, pp. 717–723, June 1998.

[18] H. Bouzekri and S. L. Miller, "An upper bound on turbo codes performance over quasi-static fading channels," *IEEE Commun. Lett.*, vol. 7, no. 7, pp. 302–304, July 2003.

[19] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inform. Theory*, vol. 42, no. 2, pp. 429–445, Mar. 1996.

[20] A. J. Viterbi, "Convolutional codes and their performance in communication systems," *IEEE Trans. Commun. Technol.*, vol. 19, no. 5, pp. 751–772, Oct. 1971.

[21] J. M. Wozencraft, "Sequential decoding for reliable communication," *IRE Nat. Conv. Record*, vol. 5, no. 2, pp. 11–25, 1957.

[22] R. M. Fano, "A heuristic discussion of probabilistic decoding," *IEEE Trans. Inform. Theory*, vol. IT-9, pp. 64–74, Apr. 1963.

[23] J. L. Massey, *Threshold Decoding.* Cambridge, Mass.: MIT Press, 1963.

[24] J. K. Omura, "On the Viterbi decoding algorithm," *IEEE Trans. Inform. Theory*, vol. IT-15, pp. 177–179, Jan. 1969.

[25] M. Bossert, *Channel Coding for Telecommunications.* John Wiley & Sons, 1999.

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

[27] D. Divsalar, S. Dolinar, R. J. McEliece, and F. Pollara, "Transfer function bounds on the performance of turbo codes," JPL, Cal. Tech., TDA Progr. Rep. 42-121, Aug. 1995.

[28] S. Benedetto, G. Montorsi, and D. Divsalar, "Concatenated convolutional codes with interleavers," *IEEE Communications Magazine*, vol. 41, no. 8, pp. 102–109, Aug. 2003.

[29] J. P. Woodard and L. Hanzo, "Comparative study of turbo decoding techniques: An overview," *IEEE Trans. Veh. Technol.*, vol. 49, no. 6, pp. 2208–2233, Nov. 2000.

[30] W. Koch and A. Baier, "Optimum and sub-optimum detection of coded data distributed by time-varying inter-symbol interference," in *Proc. IEEE Conference on Global Communications (Globecom'90)*, San Diego, CA, Dec. 1990, pp. 1679–1684.

[31] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal map decoding algorithms operating in the log domain," in *Proc. IEEE International Conference on Communications (ICC'95)*, Seattle, WA, June 1995, pp. 1009–1013.

[32] J. Hagenauer and P. Hoeher, "A viterbi algorithm with soft-decision outputs and its applications," in *Proc. IEEE Conference on Global Communications (Globecom'89)*, Dallas, TX, Nov. 1989, pp. 1680–1686.

[33] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: Turbo codes," *IEEE Trans. Commun.*, vol. 44, no. 2, pp. 1261–1271, Oct. 1996.

[34] S. ten Brink, "Convergence behavior of iteratively decoded parallel concatenated codes," *IEEE Trans. Commun.*, vol. 49, no. 10, pp. 1727–1737, Oct. 2001.

[35] W. E. Ryan, "Concatenated convolutional codes and iterative decoding," in *Wiley Encyclopedia on Telecommunications*, J. G. Proakis, Ed. Hoboken, New Jersey: Wiley-Interscience, 2003, pp. 556–570.

[36] S. Benedetto and E. Biglieri, *Principles of Digital Transmission with Wireless Applications.* New York: Kluwer Academic/Plenum Publishers, 1999.

[37] I. G. Bashmakova, *Diophantus and Diophantine Equations.* Washington DC: The Mathematical Association of America, 1998.

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

[39] S. Benedetto, M. Mondin, and G. Montorsi, "Performance evaluation of trellis-coded modulation schemes," *Proc. IEEE*, vol. 82, no. 6, pp. 833–855, June 1994.

[40] G. D. Forney, "Shannon lecture," in *Proc. IEEE International Symposium on Information Theory (ISIT'95)*, Whistler, CA, 1995.

[41] J. Hagenauer, "Rate compatible punctured convolutional codes and their applications," *IEEE Trans. Commun.*, vol. 36, no. 4, pp. 389–400, Apr. 1988.

[42] D. Haccoun and G. Bégin, "High-rate punctured convolutional codes for Viterbi and sequential decoding," *IEEE Trans. Commun.*, vol. 37, no. 11, pp. 1113–1125, Nov. 1989.

[43] A. S. Barbulescu and S. S. Pietrobon, "Rate compatible turbo codes," *IEE Electronics Letters*, vol. 31, no. 7, pp. 535–536, Mar. 1995.

[44] M. Fan, S. C. Kwatra, and K. Junghwan, "Analysis of puncturing pattern for high rate turbo codes," in *Proc. Military Comm. Conf. (MILCOM'99)*, New Jersey, USA, Oct. 1999, pp. 547–550.

[45] Ö. Açikel and W. E. Ryan, "Punctured turbo-codes for BPSK/QPSK channels," *IEEE Trans. Commun.*, vol. 47, no. 9, pp. 1315–1323, Sept. 1999.

[46] F. Babich, G. Montorsi, and F. Vatta, "Design of rate-compatible punctured turbo (RCPT) codes," in *Proc. Int. Conf. Comm. (ICC'02)*, New York, USA, Apr. 2002, pp. 1701–1705.

[47] I. Land and P. Hoeher, "Partially systematic rate 1/2 turbo codes," in *Proc. Int. Symp. Turbo Codes*, Brest, France, Sept. 2000, pp. 287–290.

[48] Z. Blazek, V. K. Bhargava, and T. A. Gulliver, "Some results on partially systematic turbo codes," in *Proc. Vehicular Tech. Conf. (VTC-Fall'02)*, Vancouver, Canada, Sept. 2002, pp. 981–984.

[49] S. Crozier, P. Guinand, and A. Hunt, "On designing turbo-codes with data puncturing," in *Proc. Canadian Workshop on Inf. Theory*, Montreal, Canada, 2005.

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

[51] E. K. Hall and S. G. Wilson, "Design and analysis of turbo codes on rayleigh fading channels," *IEEE J. Select. Areas Commun.*, vol. 16, pp. 160–174, Feb. 1998.

[52] F. J. MacWilliams and N. J. A. Sloane, "Pseudo-random sequences and arrays," *Proc. IEEE*, vol. 64, no. 12, pp. 1715–1729, Dec. 1976.

[53] L. C. Perez, J. Seghers, and D. J. Costello, "A distance spectrum interpretation of turbo codes," *IEEE Trans. Inform. Theory*, vol. 42, no. 6, pp. 1698–1709, Nov. 1996.

[54] D. V. Sarwate and M. B. Pursley, "Crosscorrelation properties of pseudorandom and related sequences," *Proc. IEEE*, vol. 68, no. 5, pp. 593–618, May 1980.

[55] S. M. Ross, *A First Course in Probability*, 5th ed. Prentice Hall, 1997.