Analysis of the finite length performance of spatially coupled convolutional codes

Ardiana Osmani
Hector Eric Moreno Trujillo

Department of Electrical and Information Technology
Lund University

Advisor: Michael Lentmaier
Saeedeh Moloudi

November 19, 2015

Printed in Sweden
E-huset, Lund, 2015

Abstract

Error control coding is an essential part of any well-designed digital communication system. It has been used in structures to protect the information, by enabling reliable delivery of digital data over unreliable communication channels. Even though this is an old topic, enlisted by Shannon in the late 40's, a lot of research is still going on. There have been recently some interesting approaches about spatially coupled LDPC codes and it promises excellent performance over a broad range of channels.

Part of the current research by the Department of Electrical and Information Technology is the construction of spatially coupled turbo like codes, including braided convolutional codes (BCC) and their generalization. The behavior of this type of codes when the length approaches infinity has been analyzed and looks very promising.

In this thesis we have investigated such codes in the finite length regime. The first thing that we do is implementing two decoders of rate 1/2 and rate 2/3 based on the BCJR algorithm for convolutional codes. This is then used as a component decoder for all constructions. Then we implement two different kinds of codes, parallel concatenated codes (PCC) and braided convolutional codes (BCC). Furthermore we constructed three different ensembles for coupled codes, spatially coupled parallel concatenated codes and spatially coupled braided convolutional code for two different types which we call Type I and Type II. We also implement a sliding window decoder for the spatially coupled ensembles.

In order to visualize the results we implement a simulation environment, we estimated the bit error probability with different values of E_b/N_0 for all the constructions. Since the computation time for these simulations was high, we used the Alarik lunarc cluster facilities based in Lund university. We started by implementing everything in Matlab but after evaluating the processing time, we decided to implement the BCJR algorithm in C++. By doing this we managed to save a lot of simulation time.

We plot all the points for the different constructions in different figures. With the help of the figures we analyze the performance of the different constructions.

We can see that braided convolutional codes do not present an error floor, which is one of the drawback of parallel concatenated turbo like codes. This investigation enables us to observe that the performance of spatial coupling for finite length gives a significant BER performance for approaching the Shannon limit. We can also observe that spatial coupling for braided convolutional codes gives better performance than spatially coupled parallel codes.

Acknowledgments

First we would like to thank our supervisor Michael Lentmaier for providing all his help, assistance, guidance but specially for his time throughout the entire thesis. We really appreciate his professionalism and enthusiasm in the topic.

Secondly, we express our gratitude to our co-supervisor Saeedeh Moloudi, acting as a fist point of contact to assist with the everyday doubts and spending so many hours seated with us trying to implement the algorithms.

We are also grateful to the Lund University Global Scholarship and CONACYT who sponsored Hector's education in Sweden.

At last but not the least, thanks to our families and friends.

Preface

This thesis work was carried out by both Ardiana and Hector in collaboration with the Department of Electrical and Information Technology (EIT). Both authors were pursuing the same goal, construct different ensembles of spatially coupled convolutional codes and analyze their performance. The two authors have taken active part in most of the steps in the process and it is hard to separate exactly as individual work. The two BCJR decoders were implemented together. But the main responsibilities of Ardiana were to implement the spatially coupled parallel concatenated codes and the spatially coupled braided convolutional codes for Type II. Hector implemented the parallel concatenated code, the uncoupled braided convolutional codes and the spatially coupled braided convolutional codes for Type I. Ardiana wrote chapter 2 and Hector chapter 3, the rest of the report was written together.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Most of the communications nowadays are based in digital data transmission or storage systems. The process of the generating, transmitting and receiving binary digits has had an exponential increasing demand due to the high speed data networks used in commercial and non-commercial applications.

These bits have to be transmitted over a communication channel to a destination or stored in a medium, it is likely to find some of these bits corrupted by the added noise introduced by a non optimal communication channel. In *A Mathematical Theory of Communications* [1] Shannon splits this problem into two, represent the information source output as a sequence of binary digits in an effective way (source coding) and transmitting the sequences with redundant bits over the noisy channel (channel coding).

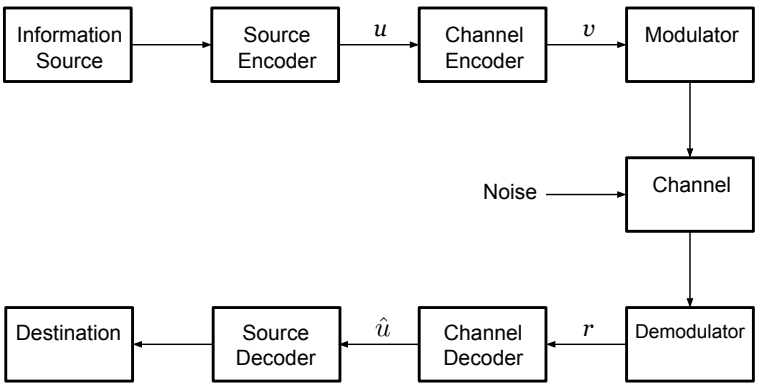

Figure 1.1: Transmission system

The digital transmission system can be depicted as in Figure 1.1. The *information source* can be any source of data, in a continuous waveform or discrete symbols. The *source encoder* creates the binary information sequence $\mathbf{u}$, for continuous

waveform it may use the help of analog-to-digital converters. This sequence is ideally minimized in such a way that less bits per second are needed to reconstruct the original signal. The *channel encoder* makes the transmitted message $\mathbf{v}$ less susceptible to noise and interference introduced in the channel. This is made by adding structured redundancy. A waveform is more suitable for being transmitted over a physical *channel*, so a *modulator* converts digital output symbols $\mathbf{v}$ in analog signals. Because the channel is subject of noise, distortion and interference, the channel output differs from the channel input. On the receiver side, the reverse process is applied in order to retrieve the original message. The *demodulator* processes the waveform generating the encoded sequence $\mathbf{r}$. The *channel decoder* uses the redundancy to detect and correct the errors in the received word and estimates a source code word $\hat{\mathbf{u}}$. The *source decoder* transform the estimated sequence $\hat{\mathbf{u}}$ into a source output and deliver it to the destination.

Shannon also states that every communication channel is defined by its capacity C_t and as long as the transmission data rate R_t is less than the capacity we can ensure reliability. With a proper encoding, the errors can be reduced to any desired level. It is also stated that the bits should be encoded in sequences in such a way that each bit has an influence not only in itself but in other bits transmitted. This principle gave birth to the coding theory.

Not only the data transmission has grown in the last years but also the semiconductor industry, giving the possibility to implement more complex algorithms for the error control which will protect more efficiently the digital data affected by the noisy channel.

1.1 Channel models

One task of the channel encoders is to adjust the information sequences for a given channel. The channel can be represented or modeled in many different ways and complexity, here we introduce a couple of the simplest but most commonly used channel models.

1.1.1 Binary Symmetric Channel

If we consider a communication system where binary modulation is used, the distribution of the noise is symmetric since the demodulator has only two levels, this scenario represent the simplest but important channel model called binary symmetric channel (BSC) where the crossover probability p describes entirely the channel as we can see in Figure 1.2. Where p is the probability of receive a wrong bit. The channel is also said to be memory less since the output depends only on the transmitted waveform during one time interval. [3]

The probability p is closely related to the signal to noise ratio E_s/N_0, in a binary modulation scenario. Assuming that a common noise disturbance additive white Gaussian noise (AWGN) is present in every system, p can be represented by

$$p = Q\left(\frac{2E_s}{N_0}\right)$$

where Q can be expressed in term of the complementary error function of Gaussian statistics by:

$$Q = \frac{1}{2}\text{erfc}\left(\frac{x}{\sqrt{2}}\right)$$

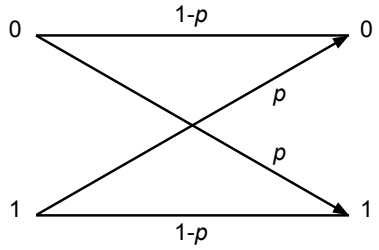

Figure 1.2: Probability diagram

1.1.2 Additive white Gaussian noise channel

This channel better known by its acronym AWGN is a model which gives a simple but with practical relevance for modeling a channel. It is a random process which, each sample is a zero-mean Gaussian random variable and its power spectral density is flat over the frequency range used with a level of $N_0/2$ Watts per Hertz. The white Gaussian noise channel can be simply described in terms of an input x and the output y by:

$$y = x + n$$

where n is the zero-mean Gaussian random variable with variance σ^2 which is independent of the input x. The conditional probability density function of the output y is given by:

$$f(y_j|x = x_i) = \frac{1}{\sqrt{2\pi\sigma}} exp\left[\frac{-(y_j - x_i)^2}{2\sigma^2}\right] \tag{1.1}$$

The AWGN channel is widely used because of its occurrence in many communication links such as satellite and deep-space links [3].

1.2 Contribution

Active research in the Department of Electrical and Information Technology in Lund University is about the effect of the spatial coupling over serial/parallel concatenated convolutional codes and braided convolutional codes. Different ensembles have been introduced and the density evolution for erasure probabilities

have been analyzed in order to calculate thresholds for different rates [25] [27] [28]. These thresholds promise good performance but since no implementation and simulations have been done, the performance is still a theoretical estimation. Beside the research made on the EIT not so many constructions of braided convolutional codes have been made in previous works. The aim of this work is to investigate the performance of the spatial coupling of parallel concatenated convolutional codes and braided convolutional codes in the finite length regime. For achieving that, the previously introduced constructions will be implemented together with the corresponding simulation tools in order to get some figures of merit that will be used for comparison and determine the feasibility of this class of codes. The final result of this project will contribute not only to the department research but to the general knowledge on the field.

1.3 Outline of the thesis

We start our thesis work with a short introduction of convolutional codes in Chapter 2, the basic properties of their structure and distances. Examples are given for the different types of encoders. It is also introduced the concept of trellis and the state diagram representation. Finally the BCJR algorithm is described in a detailed way.

In the following chapter we discuss mainly the parallel concatenation and the basic properties of their encoder and decoder structures, also the braided convolutional codes are presented. The spatial coupling is introduced in Chapter 4, therefore we can present the spatially coupled architectures of the parallel concatenated convolutional codes and braided convolutional codes. Also the window decoder is introduced in the same chapter.

In Chapter 5 we present the simulation results for all the different configurations and comparisons between coupled and non coupled codes in the finite length together with different scenarios to find out how does different parameters affect the final performance. Finally, in Chapter 6 some concluding remarks and a short discussion on future investigation.

Chapter 2

Theoretical Background

In general, there are two structurally different types of codes, *block codes* and *convolutional codes*. In block codes the information sequence is divided into blocks of k information bits each. Each block is called a message and is represented by

$$\mathbf{u} = (u_0, u_1, \ldots, u_{k-1})$$

It is possible to create 2^k different messages. Each message u is transformed into an n-tuple

$$\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$$

of bits, which is called a *codeword*. A set of 2^k codewords with length n is called a block code (n, k). The ratio $R = k/n$ is called the code rate, and this is the number of information bits entering the encoder per transmitted symbol. To have a useful code, redundant bits are added to each message which then forms a codeword [2] [3].

2.1 Convolutional Codes

Convolutional codes can be seen as nonblock linear codes over a finite field. It differs from block codes because the information sequence is not separated in blocks. They form an infinite sequence which then is shifted into a register. The convolutional encoder accepts k-bits blocks of the message sequence $\mathbf{u}$ and produces an encoded sequence $\mathbf{v}$ of n-symbol blocks also called a *codeword*. The encoded block does not only depend on the k-bit blocks, it also depends on the m (memory order) previous message block. The inputs that enter the encoder remains in the encoder for an additional m time units. The encoder will produce a set of all possible encoded outputs. These outputs will form the code. The number of information bits k entering the encoder per transmitted symbols n is called the code rate $R = k/n$. The redundant bits are added to the information sequence when $k < n$ or $R < 1$ for combating the channel noise. Large minimum distance and low error probabilities are achieved by increasing the memory order m, unlike with block codes where k and n are increased in order to have the same effect [2] [3].

2.1.1 Encoding

In general a rate $R = k/n$ convolutional encoder with the input (information) sequence

$$\mathbf{u} = (u_0, u_1, \ldots, u_t, \ldots)$$

where $\mathbf{u}_t = (u_t^1 u_t^2 \ldots u_t^k)$ and the output sequence

$$\mathbf{v} = (v_0 v_1, \ldots, v_t, \ldots)$$

where $\mathbf{v}_t = (v_t^1 v_t^2 \ldots v_t^n)$ must start at some finite time. The relationship between the information sequence and the output sequence is determined by

$$\mathbf{v}{=}\mathbf{u}\mathbf{G} \tag{2.1}$$

where

$$\mathbf{G} = \begin{pmatrix} G_0 & G_1 & \ldots & G_m & & & \\ & G_0 & G_1 & \ldots & G_m & & \\ & & G_0 & G_1 & \ldots & G_m & \\ & & & \ldots & \ldots & \ldots & \ldots \end{pmatrix}$$

is called the generator matrix of the encoder. The generator matrix $\mathbf{G}$ is a semi-infinite matrix composed by the sub-matrices:

$$\mathbf{G}_i = \begin{pmatrix} g_{1,i}^0 & g_{1,i}^1 & \cdots & g_{1,i}^n \\ g_{2,i}^0 & g_{2,i}^1 & \cdots & g_{2,i}^n \\ \vdots & \vdots & \ddots & \vdots \\ g_{k,i}^0 & g_{k,i}^1 & \cdots & g_{k,i}^n \end{pmatrix}$$

It is often convenient to express the sequences in terms of the delay operator D, also called the D-transform. Now the information sequence becomes

$$\mathbf{u}(D) = (\ldots + u_0 D^0 + u_1 D^1 + \ldots)$$

and the output sequence becomes

$$\mathbf{v}(D) = (\ldots + v_0 D^0 + v_1 D^1 + \ldots)$$

The relationship between the information sequence and the output sequence is

determined by

$$\mathbf{v}(D) = \mathbf{u}(D)\mathbf{G}(D) \tag{2.2}$$

where the generator matrix is

$$\mathbf{G}_i = \begin{pmatrix} g_1^0(D) & g_1^1(D) & \dots & g_1^n(D) \\ g_2^0(D) & g_2^1(D) & \dots & g_2^n(D) \\ \vdots & \vdots & \ddots & \vdots \\ g_k^0(D) & g_k^1(D) & \dots & g_k^n(D) \end{pmatrix}$$

and the generator polynomials is

$$g_j^i = g_{j,0}^i + g_{j,1}^i D + g_{j,2}^i D^2 + \ldots + g_{j,m}^i D^m$$

for $j = 1, 2, \ldots, k$ and $i = 1, 2, \ldots, n$ [2] [3].

But why do we call them *convolutional* codes?

If we write the outputs $v_t^{(j)}$ where $j = 1, \ldots, n$ as

$$v_t^{(j)} = \sum_{i=1}^{k} \sum_{l=0}^{m} u_{t-1}^{(i)} \, g_{i,l}^{(j)}$$

and writing the generator coefficients of input i and output j into a *generator vector*

$$\mathbf{g}_i^{(j)} = (g_{i,0}^{(j)}, g_{i,1}^{(j)}, \ldots, g_{i,m}^{(j)})$$

then we can write

$$\mathbf{v}^{(j)} = \mathbf{u}^{(1)} \cdot \mathbf{g}_1^{(j)} + \mathbf{u}^{(2)} \cdot \mathbf{g}_2^{(j)} + \ldots + \mathbf{u}^{(k)} \cdot \mathbf{g}_k^{(j)} = \sum_{i=1}^{k} \mathbf{u}^{(i)} \cdot \mathbf{g}_i^{(j)} \tag{2.3}$$

We can see that in equation 2.3 each output $v^{(j)}$ is related to each input $u^{(i)}$ by a convolution of $g_i^{(j)}$ and that is why we call them *convolutional* codes [4].

We can classify codes in two ways:

1. **Systematic and non-systematic:** The output of the non-systematic encoder does not contain the information bits due to the convolutional process. While in the systematic encoder the information sequence is unchanged among the code sequences, the k input sequences are a copy of the first k output sequences

2. **Recursive and non-recursive:** The recursive encoder uses both feedforward and feedback paths. While the non-recursive encoder only uses feed forward paths

The advantage of a systematic encoder is that the parity can easily be appended to the source sequence and if the receiver receives the correct sequence then it doesn't need to recover the original source symbols [3] [5]. The following examples are constructions that are used later on.

Example 2.1 *A rate R=1/2 non-recursive non-systematic convolutional encoder.*

Consider the polynomial generator matrix $G = (1 + D^2 + D^3 \;\; 1 + D + D^2 + D^3)$. The encoder for this generator matrix can be built as in Figure 2.1.

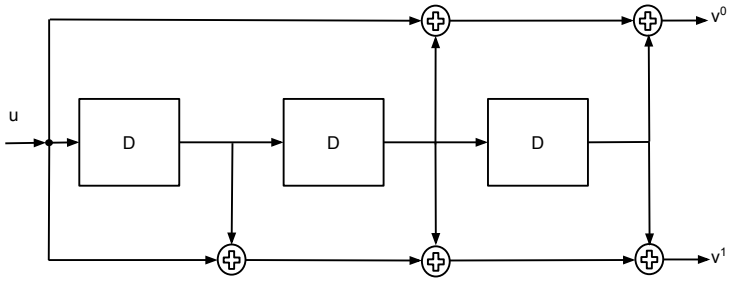

Figure 2.1: Rate R=1/2 non-recursive non-systematic convolutional encoder

Example 2.2 *A rate R=1/2 recursive systematic convolutional encoder.*

Consider the polynomial generator matrix $G = (1 \;\; \frac{1+D+D^3}{1+D^2+D^3})$. The encoder for this generator matrix can be built as in Figure 2.2.

Example 2.3 *A rate R=2/3 recursive systematic convolutional encoder.*

Consider the polynomial generator matrix $\mathbf{G} = \begin{pmatrix} 1 & 0 & \frac{1}{D^2+D+1} \\ 0 & 1 & \frac{D^2+1}{D^2+D+1} \end{pmatrix}$.The encoder for this generator matrix can be built as in Figure 2.3.

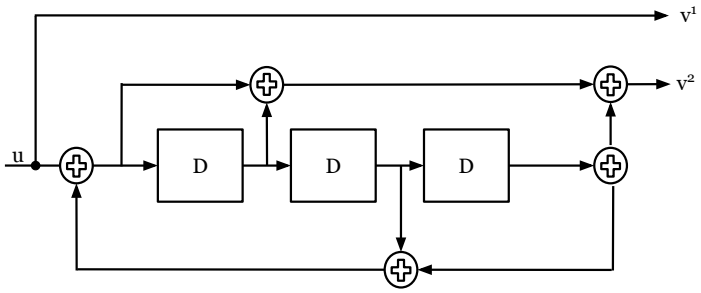

Figure 2.2: Rate R=1/2 recursive systematic convolutional encoder

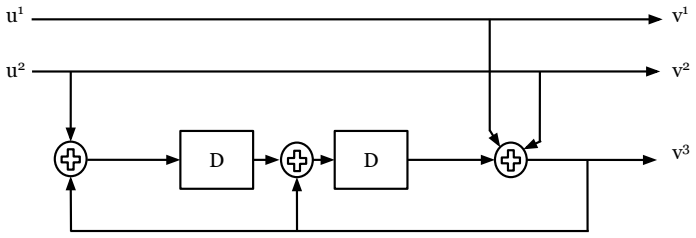

Figure 2.3: Rate R=2/3 recursive systematic convolutional encoder

2.1.2 State Diagram

There are two different ways to describe the encoding operation of an encoder, with the help of a state diagram or with the help of a trellis. The trellis is the most common representation since most decoding algorithms can be explained using it. The trellis will be explained in more detail in the next section.

A state of an encoder s_t is the stored output values of its delay elements at time t. There can be a total of 2^m different states where m is the memory [4].

Before we can construct a state diagram, we have to find the output values of the encoder (see Section 2.1.1) and the *nextstate*. The next state of an encoder depends on the current memory state s_t and on the information sequence u_t which is shifted every time instants. At time t the encoder is in state s_t.

Example 2.4 *The convolutional encoder in Figure 2.3 has a memory order m=2.* 2^m *-> * $2^2 = 4$ *states* $\{s_0(00), s_1(01), s_2(10), s_3(11)\}$. *The outputs of* $v^{(1)}$,$v^{(2)}$,$v^{(3)}$ *and the nextstate for the different inputs are presented in Table 2.1.*

State	In/Out	NS	In/Out	NS	In/Out	NS	In/Out	NS
s_0 00	00/000	s_0	01/011	s_1	10/101	s_3	11/110	s_2
s_1 01	00/001	s_3	01/010	s_2	10/100	s_0	11/111	s_1
s_2 10	00/000	s_1	01/011	s_0	10/101	s_2	11/110	s_3
s_3 11	00/001	s_2	01/010	s_3	10/100	s_1	11/111	s_0

Table 2.1: Input/Output and next state of the encoder in Figure 2.3

In Table 2.1 the column *state* represent the state that the encoder is in. *In* is the input to the encoder and *Out* is the output generated by the encoder. *NS* is the next state. For example if start in state s_0 and the input is 00 the output generated by the encoder will be 000, and the next state will be s_0. For input 01 the output will be 011 and the next state will be s_1 and so on. With the help of Table 2.1 a state diagram can be constructed. The state diagram shows all possible states, and all different transitions from one state to another state. Each transition is labeled with the input $u_t^{(1)}, u_t^{(2)}$ and the outputs $v_t^{(1)}$,$v_t^{(2)}$,$v_t^{(3)}$. The state diagram for the encoder in Figure 2.3 is constructed in Figure 2.4

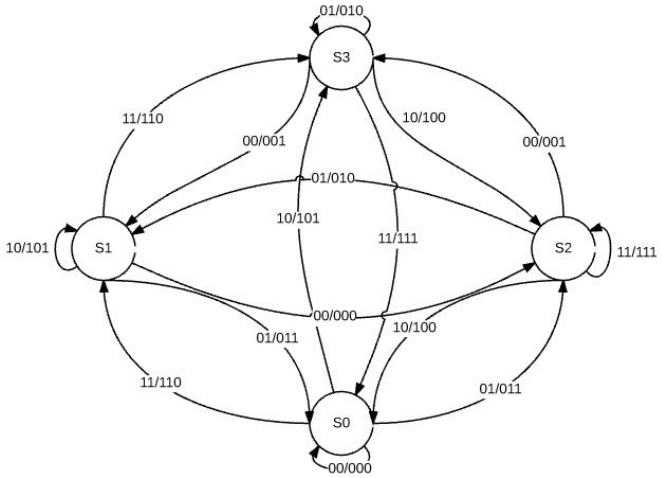

Figure 2.4: The state diagram of the convolutional encoder in Figure 2.3

2.1.3 Trellis representation

The codewords of a convolutional code is often represented as a path through a code tree also called a trellis code. A trellis consists of nodes and branches. The nodes represent the encoder's state and the branches represent the state transition. The nodes are ordered in rows and the columns correspond to a time slot. The trellis in Figure 2.5 is the trellis structure of the convolutional encoder in Figure 2.3. Since the encoder have two binary inputs there will be four branches stemming from each node. Each branch is labeled with the input bits and the output bits. These will represent the encoded sequence. To find the encoded sequence, the encoder will start in state zero and an infinite information sequence **u** will be fed

followed by some termination bits also called constraint length. In this case the constraint length will be one. The termination bits are added in order to drive the encoder back to state zero and terminate the convolutional code [3].

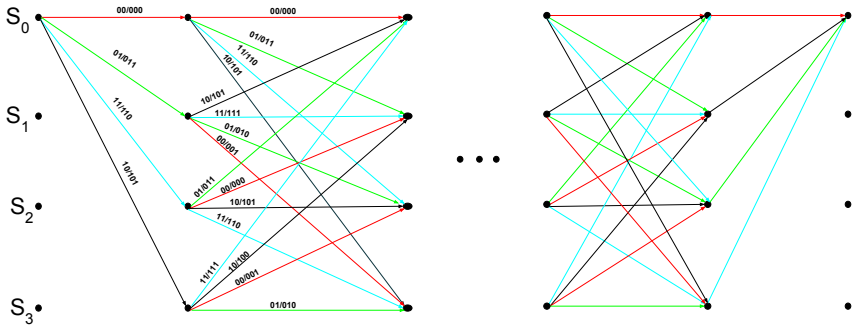

Figure 2.5: Trellis of the convolutional encoder in Figure 2.3

2.2 Distance Properties of Convolutional Codes

Ability of a code in error-correcting and error-detecting is determined by the distance properties of the code. The single most important distance property for determining the error correcting of convolutional codes is free distance. The free distance is the minimum Hamming distance between two different codewords [8]:

$$d_{free} = \min_{(v' \neq v'')} (d_h(v', v'')) \tag{2.4}$$

Since convolutional codes are linear the free distance is also the minimum Hamming weight among codewords. The minimum Hamming weight can be found by comparing all non-zero code sequences with all zero sequences [2]:

$$d_{free} = \min_{(v' \neq v'')} (w(v' + v'')) = \min_{(u \neq 0)} (w(v)) = \min_{(u \neq 0)} (w(uG)) \tag{2.5}$$

If the difference is 3 then the free distance is 3 [3].

2.3 Decoding

Convolutional decoders have a trellis structure and there are two different decoding algorithms that use this structure. A decoding algorithm for convolutional codes was introduced in 1967 by Viterbi and is known as the Viterbi algorithm since then [9]. Omura showed that the Viterbi algorithm was a programming solution of finding the shortest path. And later Forney showed that it was a maximum

likelihood (ML) decoding algorithm for convolutional codes [3]. The second algorithm was introduced in 1974 by Bahl, Cocke, Jelinek and Raviv as a maximum a posteriori probability (MAP) decoding method for convolutional codes and is known as the BCJR algorithm [11].

The difference between the MAP method and the ML method is that in the MAP method the probability of the information bit error is minimized and in the ML decoding the probability of codeword error is minimized. The performance of these two algorithms is essentially identical [3].

2.3.1 BCJR Algorithm

Since the BCJR algorithm is more complex than the Viterbi algorithm it was not used during 20 years in practical implementations. When turbo codes came into existence in 1993 Berrou, Glavieux and Thithimajshima created a modified version of the BCJR which then lead to a rebirth of the algorithm. The MAP decoder gives better performance when the a priori probabilities changes from iteration to iteration and that is why it is more preferred to use in iterative decoding such like turbo codes. Since the BCJR algorithm needs to perform many multiplications several versions such as max-log MAP and log-MAP have been proposed to reduce the computation complexity. The one that we are going to use in this thesis is the log-Domain BCJR algorithm (Log-APP) [2] [3] [6].

In this section we are going to describe the BCJR algorithm for the rate $R = k/n$ convolutional codes used on AWGN channel. The information or the message input bit $\mathbf{u}_i$ can take values 0 or 1 from the received sequence $\mathbf{r}$ which is the demodulated sequence after the channel effects with an a priori probability $P(u_i)$. The algorithm calculates the a posteriori probability (APP) L-values (log-likelihood ratios) of each information bit [7].

$$L(u_i) = log\left[\frac{p(u_i = 0|\mathbf{r})}{p(u_i = 1|\mathbf{r}}\right] \tag{2.6}$$

And the output of the decoder is given by

$$\hat{u_i} = \begin{cases} 0 \; if \; L(u_i) > 0 \\ 1 \; if \; L(u_i) < 0 \end{cases} \tag{2.7}$$

for $j = 0, 1, \ldots, j-1$ [3].

Since we are going to calculate the bit error (BER) in the simulations which is described in Chapter 5, the coded sequence is transmitted over a additive white Gaussian noise channel [3]. If we substitute 2.6 in 1.1, the new L_{ch} values can be expressed as

$$L_{ch}(u_i) = \log\left[\frac{\frac{e^{-(r-1)^2}}{2\sigma^2}}{\frac{e^{-(r+1)^2}}{2\sigma^2}}\right] = \log\left[\frac{e^{-(r-1)^2+(r+1)^2}}{2\sigma^2}\right] = \frac{2}{\sigma^2}r(u_i) \tag{2.8}$$

The BCJR algorithm has the following steps:

1. Compute metrics
2. Compute soft outputs

Since the BCJR algorithm uses a trellis structure we are going to describe how we can use the trellis structure to compute the steps. Figure 2.6 shows the trellis of a rate 1/2 convolutional code with four states $S = \{0, 1, 2, 3\}$. The solid line shows the branches generated by an input zero and likewise, the dashed line by an input one. Each transition has an output that is generated in the encoder. If we are at time t the corresponding state will be $S_t = s$, the previous state will be $S_{(t-1)} = s^{'}$ and the next state will be $S_{(t+1)} = s^{''}$. The output of the decoder will be $\mathbf{y}_t$. The complete sequence $\mathbf{y}$ will represent the past, the present and the future:

$$\mathbf{y} = \mathbf{y}_{t-1}\mathbf{y}_t\mathbf{y}_{t+1}$$

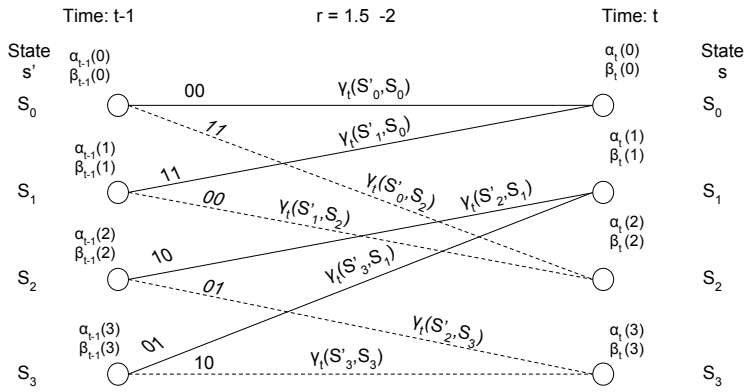

Figure 2.6: Trellis of rate 1/2 with 4 states

2.3.1.1 Calculation of Metrics

The first step in the BCJR algorithm is to compute the metrics gamma, alpha and beta. We start computing the gammas for each branch according to equation 2.9, where $L_a(u_t^i)$ is the a priori probabilities of the information bits and $L_{ch}(v_t^j)$ is

the channel value. Every time we compute we label the branch with the value of $\gamma_t(s', s)$ as in Figure 2.6.

$$\gamma_t(s', s) = \sum_{i=1}^{K} L_a(u_t^i)\left(\frac{1}{2} - u_t^i\right) + \sum_{j=1}^{N} L_{ch}(v_t^j)\left(\frac{1}{2} - v_t^j\right) \tag{2.9}$$

For example if we use Figure 2.6 and assume that the a priori probabilities of the information bits are equally likely [7]

$$L_a(u_t^i) = 0, t = 0, 1, 2$$

and that

$$L_{ch}(v_t^j) = \frac{2}{\sigma^2} r(u_i)$$

to compute the branch stemming from S_0' to S_2 ($\gamma_0(S_0', S_2)$) it will look as follow:

$$\gamma_0(S_0', S_2) = \sum_{j=1}^{N} L_{ch}(v_t^j)\left(\frac{1}{2} - v_t^j\right) = \left((1.5)\frac{1}{2} + (-2)\frac{1}{2}\right)$$

When we have the gammas which are related with the branches that arrive into the state, we compute the alphas according to equation 2.10 and label each state node with the value of $\alpha_t(s)$. Since this is a forward procedure we start with the initial condition 2.11 and repeat the procedure until we reach the end $\alpha_n(0)$ [7].

$$\alpha_t(s) = \max^*(\gamma_t(s', s) + \alpha_{t-1}(s)) \tag{2.10}$$

with the initial condition

$$\alpha_0(s) = \begin{cases} 0 \; if \; s = 0 \\ -\infty \; if \; s \neq 0 \end{cases} \tag{2.11}$$

and

$$\max^*(x, y) = \max(x, y) + \ln(1 + e^{-|x-y|}) \tag{2.12}$$

For example to compute $\alpha_t(0)$ we use the trellis in Figure 2.7. We can see that at time t two branches arrives at state S_0. One of the branches comes from S_0' and the second one comes from S_1'. The computation will be the following:

$$\alpha_t(0) = \max^*(\gamma_t(S_0', S_0) + \alpha_{t-1}(0), (\gamma_t(S_1', S_0) + \alpha_{t-1}(1))$$

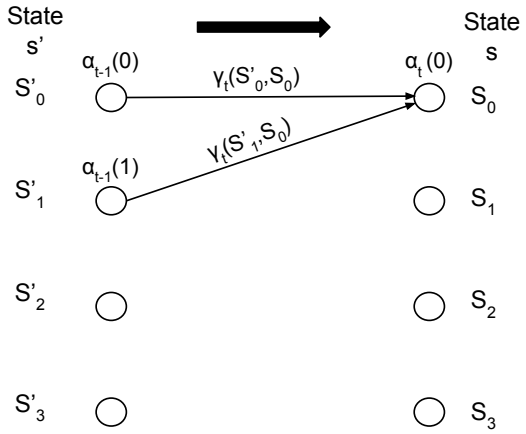

Figure 2.7: Trellis computation of alpha

The last metric to compute is beta. Beta can only be calculated after the complete sequence of gamma has been received. The procedure is computed in a similar way as alpha. The difference is that we compute backwards, instead of starting at the beginning of the trellis we start at the end with the initial condition 2.14 and repeat the procedure until we reach the beginning. The betas are obtained according to equation 2.13 by computing max* of the summation of $\beta_t(s')$ and $\gamma_t(s', s)$ that leave state $s_{t-1} = s'$ [7].

$$\beta_{t-1}(s') = \max^*(\gamma_t(s', s) + \beta_t(s)) \tag{2.13}$$

with the initial condition

$$\beta_{L+m}(s) = \begin{cases} 0 \; if \; s = 0 \\ -\infty \; if \; s \neq 0 \end{cases} \tag{2.14}$$

For example to compute $\beta_{t-1}(0)$ we use the trellis in Figure 2.8. We can see that at time t there is one branch stemming from state S_0 to state $S_0^{'}$ and another one from state S_2 to state $S_0^{'}$. The computation will be the following:

$$\beta_{t-1}(0) = \max^*(\gamma_t(S'_0, S_0) + \beta_t(0), (\gamma_t(S'_0, S_2) + \beta_t(2))$$

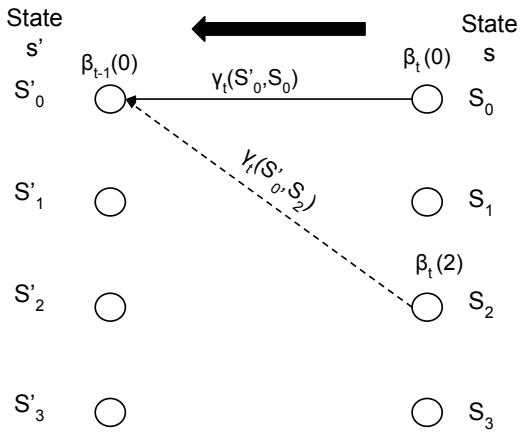

Figure 2.8: Trellis computation of beta

2.3.1.2 Calculation of Soft Output

The last step is to compute the soft output according to equation 2.15.

$$L(u_t^{(i)}) = \max^*_{(s',s):u_t^i=0}(\alpha_{t-1}(s') + \gamma_t(s',s) + \beta_t(s)) - \max^*_{(s',s):u_t^i=1}(\alpha_{t-1}(s') + \gamma_t(s',s) + \beta_t(s)) \tag{2.15}$$

Now that we have all the values of γs, αs and βs, we are able to compute the soft output. The soft outputs are obtained by taking the max* of the summation of $\alpha_{t-1}(s')$,$\gamma_t(s',s)$ and $\beta_t(s)$ for all branches corresponding to input 0 and subtract with the max* of the summation of $\alpha_{t-1}(s')$,$\gamma_t(s',s)$ and $\beta_t(s)$ for all branches corresponding to input 1. We start at the beginning of the trellis and repeat the procedure until it reaches the end. [7].

For example to compute Lu_0 we use the trellis in Figure 2.9. We can see that at time $t-1$ there are two branches stemming from each state, one with input zero and another one with input one. Since the max* defined in equation 2.12 can only be used to evaluate two values and in this case we have four values for each input. We have to take the max* of two separated states for each input at a time, for example $\max^*(S'_0, S'_1)$ and $\max^*(S'_2, S'_3)$. And then take the max* of the two resulting previous evaluations. The last step is to subtract the final values for input 0 with input 1. The computation will be the following:

$$x_{(S'_0,S'_1,input0)} = \max^*(\alpha_{t-1}(0) + \gamma_t(S'_0,S_0) + \beta_t(0), \alpha_{t-1}(1) + \gamma_t(S'_1,S_0) + \beta_t(0))$$

$$x_{(S'_2,S'_3,input0)} = \max^*(\alpha_{t-1}(2) + \gamma_t(S'_2,S_1) + \beta_t(1), \alpha_{t-1}(3) + \gamma_t(S'_3,S_1) + \beta_t(1))$$

$$y_{input0} = \max^*(x_{(S'_0,S'_1,input0)}, x_{(S'_2,S'_3,input0)})$$

$$x_{(S'_0,S'_1,input1)} = \max^*(\alpha_{t-1}(0) + \gamma_t(S'_0,S_2) + \beta_t(2), \alpha_{t-1}(1) + \gamma_t(S'_1,S_2) + \beta_t(2))$$

$$x_{(S'_2,S'_3,input1)} = \max^*(\alpha_{t-1}(2) + \gamma_t(S'_2, S_3) + \beta_t(3), \alpha_{t-1}(3) + \gamma_t(S'_3, S_3) + \beta_t(3))$$
$$y_{input1} = \max^*(x_{(S'_0,S'_1,input1)}, x_{(S'_2,S'_3,input1)})$$
$$L(0) = y_{input0} - y_{input1}$$

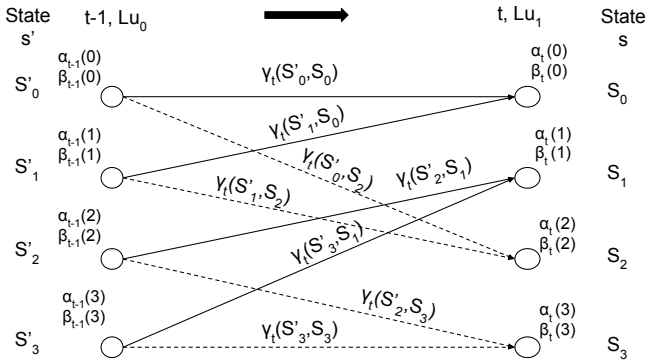

Figure 2.9: Trellis computation of soft output

2.3.1.3 Algorithm Summary

Here are all steps summarized together [7]

1. **Compute metrics**

$$\gamma_t(s', s) = \sum_{i=1}^{K} L_a(u_t^i)(\frac{1}{2} - u_t^i) + \sum_{j=1}^{N} L_{ch}(v_t^j)(\frac{1}{2} - v_t^j)$$

$$\alpha_t(s) = \max^*(\gamma_t(s', s) + \alpha_{t-1}(s)), \quad \alpha_0(s) = \begin{cases} 0 \; if \; s = 0 \\ -\infty \; if \; s \neq 0 \end{cases}$$

$$\beta_{t-1}(s') = \max^*(\gamma_t(s', s) + \beta_t(s)), \quad \beta_{L+m}(s) = \begin{cases} 0 \; if \; s = 0 \\ -\infty \; if \; s \neq 0 \end{cases}$$

2. **Compute soft outputs**

$$L(u_t^{(i)}) = \max^*_{(s',s):u_t^i=0}(\alpha_{t-1}(s') + \gamma_t(s', s) + \beta_t(s)) - \max^*_{(s',s):u_t^i=1}(\alpha_{t-1}(s') + \gamma_t(s', s) + \beta_t(s))$$

Chapter 3

Concatenated Convolutional Codes

A powerful technique of building long codes from short component codes known as concatenation of codes was invented by Forney in 1966 [12]. Forney's goal was to find codes where the probability of the errors decreased exponentially while the block length and decoding complexity will only be increased algebraically. The concatenated code consists of an inner code and an outer code. The inner code is relatively short and decoded with a soft decoding algorithm. The outer code is longer and it is decoded with an algebraic error-correction algorithm [3] [12].

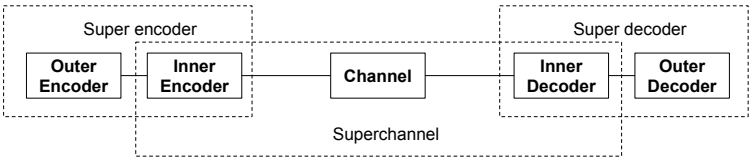

Figure 3.1: Concatenated system

Figure 3.1 illustrates this first approach which corresponds to a serial concatenated system. The outer encoder uses a Reed-Solomon code (a class of linear block code) and the inner encoder uses, for example, a convolutional code to clean up the channel. A maximum likelihood decoding algorithm is used in the inner decoder to correct most of the channel errors, but this will lead to a burst of errors. The output of the inner decoder becomes the input of the outer decoder. A Reed-Solomon code is used in the outer decoder since it is suited to cope with burst of errors. The minimum distance of the overall code is the product of each of the distance of each encoder. In order to decrease the burst of error, new components are introduced, an interleaver and a deinterleaver. In this case the distance will depend on the structure of the interleaver. The output of the outer encoder is interleaved before entering the inner encoder. Then the output error burst of the inner decoder is deinterleaved before entering the outer decoder. The concatenated system with interleaver and deinterleaver is shown in Figure 3.2 [12] [2].

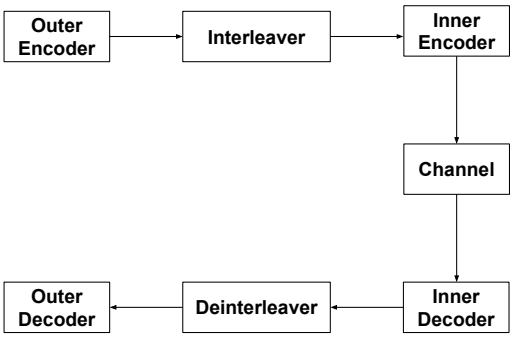

Figure 3.2: Concatenated system with interleaver and deinterleaver

Totally random codes of large length are the ones which can give a very good performance overall but that kind of codes are still not able to be decoded, on the other hand codes which are well-structured are easier to decode. For that reason the structure was one of the most important aspects and kept all the attention while designing codes. However after the introducing of turbo codes [16] by Berrou, Glavieux and Thitimajshima the research was focused again in codes with random-like properties. The main principle of turbo codes is the use of two weak or humble structured codes but still with enough construction for efficiently use iterative decoding techniques.

3.1 Parallel Concatenation

The concatenation in a parallel arrangement can be applied in both block codes and convolutional codes. In this report we will only refer to the convolutional case. Figure 3.3(a) shows the basic parallel concatenated encoding structure, consisting of a parallel arrangement of two recursive codes, called constituent or component codes. These two constituent codes can be the same code or different ones, systematic or non-systematic. The encoders receive the name of upper and lower encoders. Both receive the same input signal but for the lower one, the input is permuted by an interleaver Π. In other words the two encoders operate in *parallel* with different versions of the information. The task of the interleaver is to create independent input for each constituent code but it also has some other impacts that we will discuss later in this chapter.

The output of the turbo encoder, called codeword, is composed by both the systematic output $\mathbf{v}^{(1)} = \mathbf{u}_t$ and the parity sequences $\mathbf{v}^{(U)}$ and $\mathbf{v}^{(L)}$ plus a constraint sequence for ensuring to return the encoders to the zero state. This constraint sequence is also called termination bits. This is the final sequence which will be sent through the channel.

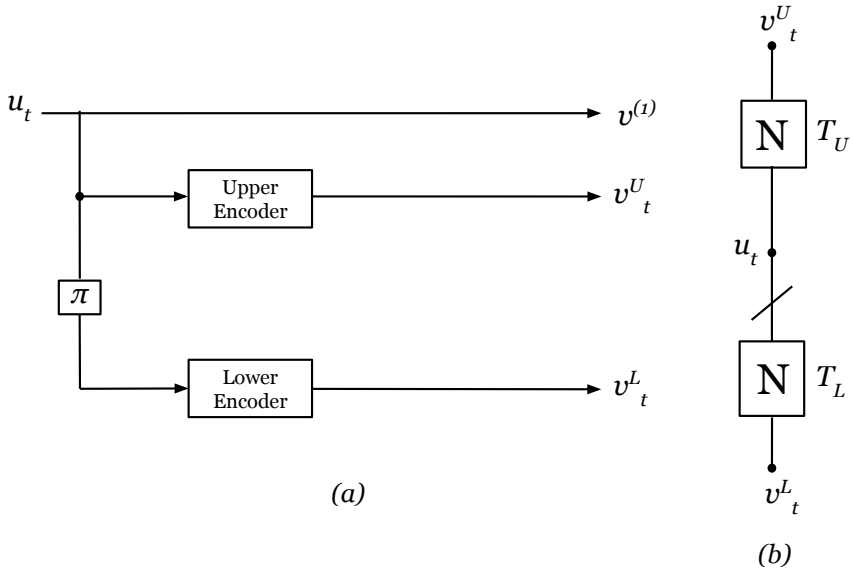

Figure 3.3: (a). Basic turbo encoding structure. (b) Compact graph

Figure 3.3(b) depicts a recent way of representation, the compact graph, introduced by Moloudi [27]. In compact graph representation each trellis is replaced by a square called factor node. The length of factor node is written in it. In the mentioned figure the upper and lower trellis are denoted by T_U and T_L, respectively. Information and parity sequences are replaced by black circles and called variable nodes $\mathbf{u}_t$, $\mathbf{v}_t^U$ and $\mathbf{v}_t^L$. The interleaver is just a cross line on the connecting edge between $\mathbf{u}_t$ and T_L, which is the same size as the sequences and the trellis N. Both graphs are equivalents.

We can point out that turbo codes are parallel concatenated convolutional codes with random-like properties, so simple SISO (soft-in soft-out) decoders with messages passing from one to the other and vice versa are used and will be described. The use of a pseudorandom interleaver is a very important part of the design and has a high weight in the final performance of the turbo codes.

3.1.1 Interleaving

The performance of a turbo code improves when the interleaver size is increased, that means it has positive influence on the properties of the code and the iterative decoding. Beside its main purpose mentioned before, the interleaver has a strong influence over two important issues associated with the performance.

1. It is closely related to the distance properties of the code, which as we will see are very important for the final performance.

2. It can be efficiently decoded. This means for the de-correlate nearby positions in the decoder input, which usually end up in similar close relation after interleaving. These are also called short cycle events. The latter short cycle events degrade the performance of turbo decoding [17].

In this report we refer normally to the interleaver as a Π, which will contain the permutation positions with an specific size N. However there has been some research around how to construct optimal pseudorandom interleaver who can give the larger distance of the code and avoid the short cycle events. Deterministic pseudorandom interleaver allow for discrete mapping of bits according to a designed scheme and semi-random (s-random) interleaver exhibit a good performance in this sector [18]

3.1.1.1 S-random interleaver

The purpose of the s-random interleaver is to create a sequence of length N with a minimum distance S between each position of the sequence. In [18] is stated that, for the interleaver Π, the new selected position $\Pi(i)$ is accepted only if its absolute difference from the previous S selected numbers $\Pi(j)$ is greater than S. This condition can be expressed as:

$$|i - j| \leq S \longrightarrow |\pi(i) - \pi(j)| > S$$

And also pointed out that interleaver can be generated in reasonable time for

$$S < \sqrt{\frac{N}{2}} \tag{3.1}$$

The algorithm used in this thesis project to generate the sequence is as follow:

1. Given the interleaver size N, generate an integer pool of N elements without replacement.
2. Set the maximum value of S according to expression 3.1 .
3. Select randomly an integer from the pool and check if it is outside of the range $\pm S$ of the S past vales. If it is outside, keep the value and delete it from the pool, otherwise reject it and place it back to the pool.
4. Repeat previous step until no integers are left in the pool or reach the maximum iteration previously set.
5. If maximum iteration is reached decrease the value of S and start from step 3.

3.1.2 Iterative decoding

Before starting with the functionality of the iterative decoding we should recall some algebraic properties for the log-likelihood ratios presented during the introduction of the BCJR algorithm and some definitions. Having the sequence U in the *GF(2)* Galois Field, with elements $\{0, 1\}$. The log-likelihood ratio of a binary

random variable u, $L(u_i)$ can be expressed as in the equation 2.6, where $p(u_i)$ denotes the probability of u of taking each of the values. The log-likelihood ratio will be called, more commonly in this report, the *L-value* of the random variable u [19]. We can define the operator for the log-likelihood ratio values $L(u)$, and use the operator $\boxplus$ as the notation for the addition:

$$L(u_1) \boxplus L(u_2) = L(u_1 + u_2)$$

following the aditional rules:

$$L(u) \boxplus \infty = L(u) \qquad L(u) \boxplus -\infty = -L(u) \qquad L(u) \boxplus 0 = 0$$

with some identities it can be demostrated that:

$$\begin{aligned} L(u_1) \boxplus L(u_2) &= \log \frac{1 + e^{L(u_1)} e^{L(u_2)}}{e^{L(u_1)} + e^{L(u_2)}} \\ &\approx \operatorname{sign}(L(u_1)) \cdot \operatorname{sign}(L(u_2)) \cdot \min(|L(u_1)|, |L(u_2)|) \end{aligned} \quad (3.2)$$

Where the reliability of the sum $\boxplus$ is determined by the smallest reliability of the terms.

According to what is previous stated we can see more clearly what the *L-value* or soft value of a channel stands for. After transmission over a *AWGN* the log-likelihood ratio of the x coded bits conditioned on the match filter output y can be calculated as follow:

$$\begin{aligned} L(x|y) &= \log \frac{\exp(-\frac{E_s}{N_o}(y-a)^2)}{\exp(-\frac{E_s}{N_o}(y+a)^2)} + \log \frac{P(x=0)}{P(x=1)} \\ &= L_{ch} \cdot y + L(x) \end{aligned}$$

Where $L_{ch} = 4a * \frac{Es}{No}$ for a fading channel. L_{ch} is the log-likelihood ratio of the crossover probability, is also called the channel reliability factor and $L(x)$ is the a priori *L-value*. Finally the output of a soft output decoder computes the a posteriori L values:

$$L(v_i) = L_a(v_i) + L_{ch}(v_i) + L_e(v_i)$$

The first two terms $L_a(v_i)$ and $L_{ch}(v_i)$ are known as the intrinsic part, correspond to the a priori and channel reliabilities of the symbol v_i itself. $L_e(v_i)$ is the extrinsic a posteriori value which correspond to an estimate of v_i based on other symbols.

The iterative decoding of two constituent codes uses iterative message passing (belief propagation) decoding. It can be made with reasonably low complexity using the decoder shown in Figure 3.4*(a)*. It works with two *SISO* decoders, using the *MAP* algorithm already presented in previous chapter (*BCJR*). The interleaver

Π must be the same used in the encoding process. At any time t, the decoder receives the information proceeding from the channel, the systematic information L_u and the parity sequences L_v^1 and L_v^2.

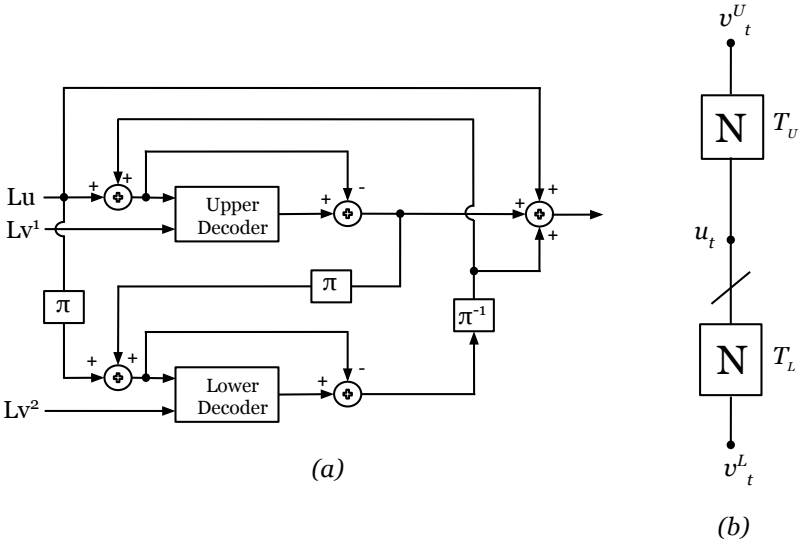

Figure 3.4: (a). Iterative turbo decoder. (b) Compact graph representation.

In the first iteration of the upper decoder, the input information are the *L-values* L_u and L_v^1, since we still do not have information coming from the lower decoder, the a priori information is set to zero as initial conditions. Information goes through the decoder and the extrinsic *L-value* of the output feed the lower decoder. That means that the inputs of the lower decoder are the channel values L_u, which will be permuted by Π, the parity L_v^2 and the permuted version of the extrinsic *L-values* coming from the upper one. After this information is processed by the lower decoder, the extrinsic information is deinterleaved and added to the upper decoder as a priori values. One decoding iteration is completed after both upper and lower decoder have been activated. Every iteration represents a improvement of the performance.

For the subsequent iterations, the a priori *L-values* are replaced by the extrinsic a posteriori L_{e2} after being deinterleaved. These extrinsic *L-values* which pass from one decoder to the other, represent the reliability information of the bits being decoded. This exchange of messages is the way of this suboptimum interactive process ensures that less information is lost compared to an optimal decoding which is much more complex. After an appropriate number of iterations, the turbo decoder output can be computed with the addition of the next terms, the extrinsic output of the upper decoder, the deinterleaved version of the a posteriori L values

of the lower decoder and the channel information L_u.

We can see in Figure 3.4*(b)* exactly the same compact graph we presented together with the turbo encoding structure. The advantages of using this graphs are obvious because they can represent the encoding block and the iterative turbo decoder. Factor graphs have been used for this representation purposes, however the factor graphs of codes with convolutional components, can get very large as the length of the component codes increases. So compact graphs will be used in this report.

The two inputs of the upper decoder, *L-values*, are placed on each side of the factor node T_U. It is known that only extrinsic values are exchanged between the component codes, so this operation is not shown in the compact graph. Another consideration we can point out is the use of the interleaver, when we *walk* towards the factor node, the sequence is permuted, but when we *walk* from the factor node, the *L-values* should be depermuted.

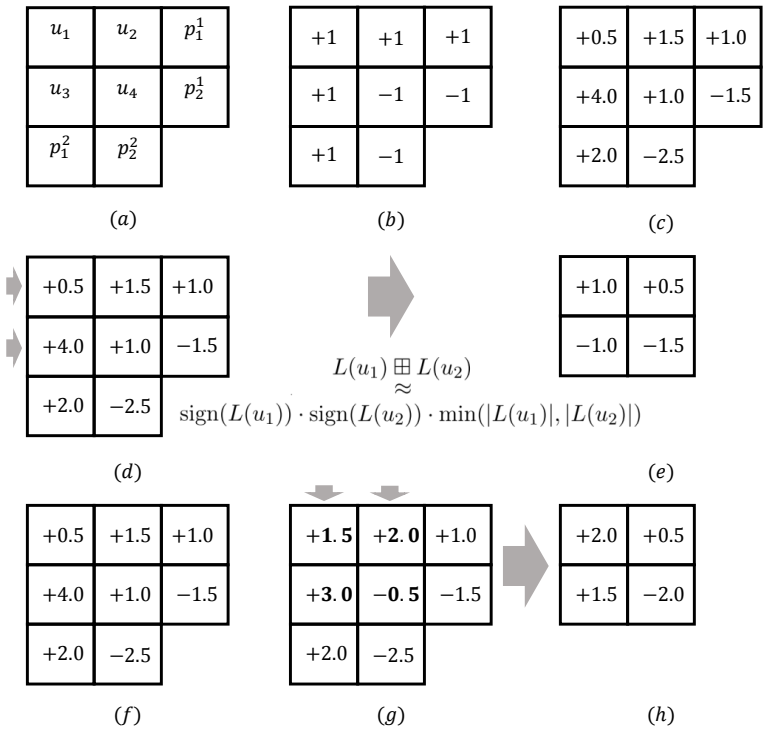

Figure 3.5: Iterative decoding example

We now present an example of a parallel concatenated code, to illustrate the principle and the advantages of the iterative decoding [20]. Where the input block are the vectors $\mathbf{u} = [u_0, u_1, u_2, u_3]$ and the parity vector of the first and second constituent codes are $p^1 = [p_1^1, p_2^1]$ and $p^2 = [p_1^2, p_2^2]$, respectively as is represented in the array showed in Figure 3.5*(a)*. The transmitted values after being modulated are in *(b)*. The output after the effect of the channel is in *(c)*. The horizontal decoding is calculated with the help of the expression 3.2 between the information and the parity vectors having as results the extrinsic *L-values* in *(e)*. The same for the vertical decoding but in this case considering the a priori *L-values* as in *(g)* to calculate the extrinsic *L-values* in *(h)*. After this calculation, the first iteration is finished and a posteriori information can be estimated adding the channel output with both extrinsic information from both decoders. In this example the message is properly decoded after the first iteration.

3.2 Parallel concatenated convolutional codes

After recalling the theory introduced in the previous sections. We can refer as a parallel concatenated convolutional code PCC, to the codes which use parallel concatenation, as depicted in Figure 3.3(a). Each of the constituent encoders are the binary systematic feedback convolutional encoders shown in the *Example 2.2* on chapter two. As it is in Figure 3.6. These encoders follow the generator matrix $G(D) = [1 \ \frac{1+D+D^3}{1+D^2+D^3}]$.

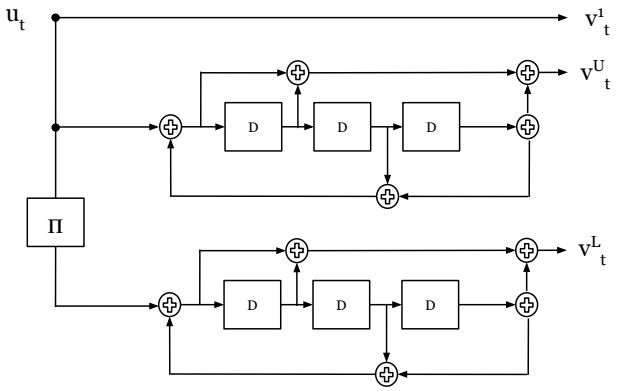

Figure 3.6: PCC

Since both constituent codes are rate R=1/2, the final PCC rate is R=1/3, having as an output the systematic $\mathbf{v}^1$ and the parity sequences $\mathbf{v}^U$ and $\mathbf{v}^L$. After a channel, the received L-values, L_u, L_v^1 and L_v^2 will become the input of the decoder introduced on Figure 3.4(a).

A s-random interleaver was added to the previously presented structures with an $S = 28$. The overall BER performance is shown as a function of SNR in Figure 3.7. Where a Turbo code 1/3 with a random interleaver of size 8192 and a turbo code 1/3 with s-random interleaver of the same size are shown. Both with the same systematic feedback component encoders. As we can see, the bit error rate (BER) graph of a parallel concatenated convolutional code can be divided in two sections:

1. Waterfall section. Where the bit error probability curve begins the characteristic sharp drop [3]. This happened just after the SNR threshold.
2. Error floor. Is where the BER curve starts to flatten out and is due the minimum distance of the code.

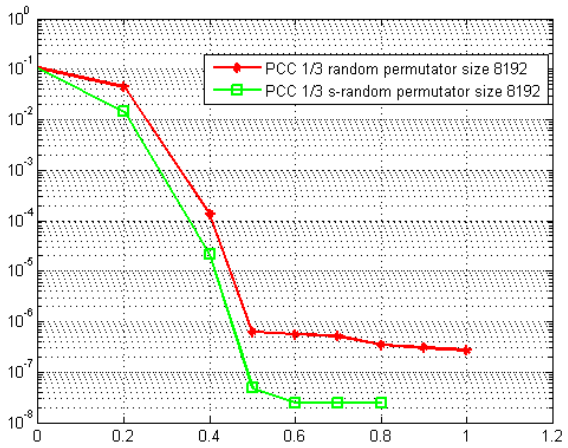

Figure 3.7: S-random interleaver performance enhancement on PCC

It is clear the improvement in the error floor section of the BER with the s-random interleaver, the error floor drops until the order of 10^{-7} for interleavers of the same size. Much more will be discussed about the BER graphs and different permutation length when we present all the results in Chapter 5.

3.3 Braided convolutional codes

Braided block codes were first introduced in [24] with two different families of codes based on the density of the storage array, tightly and sparsely braided block codes, being proved that sparsely braided block codes has an improved performance with iterative decoding over the tightly. Braided convolutional codes can be seen as two-dimensional sliding array in which the symbols are protected by horizontal and vertical component codes. In this project we will be referring as

BCC to the sparsely braided codes which use convolutional codes as component codes, the same way as PCC. The most important difference between PCC and BCC is that in BCC, the parity sequence of the first component encoder is used as input for the second encoder and vice versa [25].

On the array introduced in [26], the horizontal and vertical encoders are connected by the parity sequences feedback, and it is because of this characteristic where the systematic and parity symbols are *braided* together, that the BCC take their name.

A rate *R=1/3* block wise BCC is illustrated in Figure 3.8, composed by two systematic convolutional encoders of rate $R = 2/3$ as components, also called upper and lower encoders with the generator matrix

$$\mathbf{G} = \begin{pmatrix} 1 & 0 & \frac{1}{D^2+D+1} \\ 0 & 1 & \frac{D^2+1}{D^2+D+1} \end{pmatrix}.$$

This convolutional encoders were introduced in chapter two in the *example 2.3.* Another difference is that in this BCC encoder we will use three different interleavers of the same size but with different permutation order. The upper encoder has as input in any time instant t, a block of information symbols $\mathbf{u}_t$ and a parity sequence $\mathbf{v}_{t-1}^2$ coming from the output of the lower encoder after the interleaver Π^2. The inputs for the lower decoder is the permuted version of the information $\mathbf{u}_t$ by the interleaver Π and the permuted parity $\mathbf{v}_{t-1}^1$ from the upper encoder through the interleaver Π^1, so at a time t the output of the encoder is the tuple $\mathbf{v}_t = (\mathbf{v}_t^1, \mathbf{v}_t^U, \mathbf{v}_t^L)$.

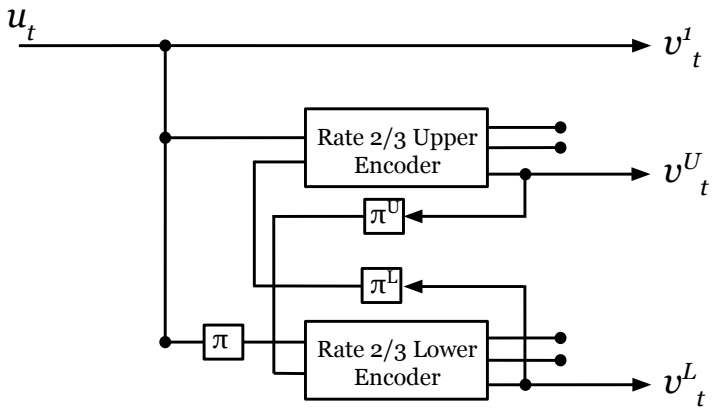

Figure 3.8: BCC encoder

The encoding of procedure of BCC is not a straightforward scenario since a equation system should be solve. For that reason, we choose in this thesis to

omit the BCC encoder and instead a zero sequence is transmitted over the channel.

For BCCs, we have considered iterative message passing decoding with BCJR component decoders, similarly to PCCs. The only considerations to be made are to initialize the extrinsic outputs of the lower decoder to zero for the first iteration.

A block diagram representing the BCC decoder is illustrated in the Figure 3.9. The inputs of the component decoders are the *L-values* ($L_{u_t}, L^1_{v_t}, L^2_{v_t}$) coming from the channel and the extrinsic outputs created by each BCJR decoder. The subtraction of the "intrinsic" information for each output is omitted in this diagram in order to make it more clear (it is shown in Figure 3.4. Iterative turbo decoder).

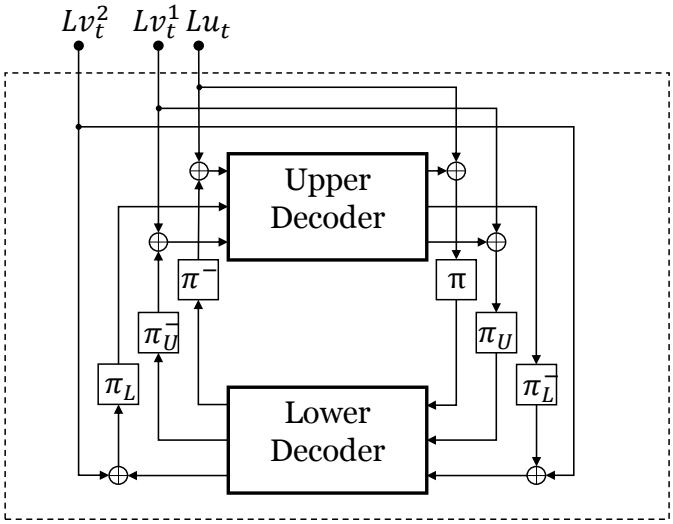

Figure 3.9: BCC decoder

Initial conditions should be considered for the first iteration of the upper decoder as zero and the output can be delivered from the summation of the a posteriori *L-values* properly deinterleaved at the output of the lower decoder, the extrinsic output of the upper decoder and the information u_t.

As we can realize, the compact graph representation's convenience is more obvious in the case of the BCC where the block diagram starts to be more complicated. The compact graph of the BCC is shown in Figure 3.10. It is important to point out the order of the inputs and outputs of the factor nodes, having as a convention the furthest to the left as first input/output and the one on the right as second

input/output. Also the same scenario as PCC, compact graph represents both encoder and decoder structures. This representation will become more helpful when we introduce the coupled versions in the next chapter.

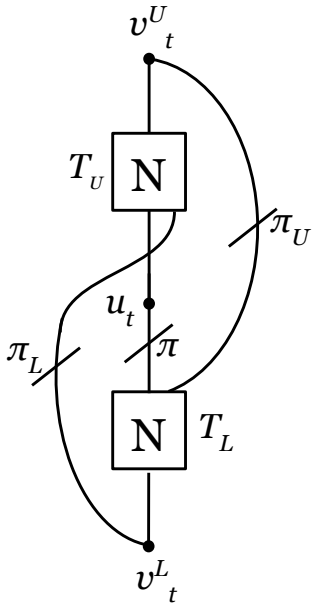

Figure 3.10: Compact graph of BCC

Chapter 4
Spatial Coupling

The concept of Spatial coupling (SC) has been exploited by LDPC codes for a long time. It has been proved that spatially coupled LDPC codes have both large minimum distance and capacity-achieving, mainly on the performance of the waterfall region. But the idea of coupling is not exclusive of the LDPC codes, recently it has been used for coupling turbo-like codes, showing promising performance in the *finite length* scenario. This can be achieved by replacing the traditional block interleaver in a turbo code by a convolutional interleaver [21].

Two new concepts are introduced in order to properly describe the coupling method. The coupling length L which is not more than the number of encoders considered in the arrangement. The other parameter is the coupling memory m. This parameter will give the information about with how many neighbor encoders at any time instant t the current encoder will exchange information, the way this information is exchanged depends on the type of concatenation. The scope of this project will only deal with constructions were m=*1*, diagrams will also be presented when introducing the constructions made for an easier visualization.

The good performance of spatially coupled codes is more evident when both, the block length of the codes and the coupling length are larger. However, as we stated before, the drawback is the complexity of the belief propagation *BP* decoding.

4.1 Spatially Coupled Parallel concatenated codes

In this section we are going to describe the implementation of spatially coupled parallel concatenated codes and then describe their decoding.

We consider a chain of L parallel concatenated codes with rate $R = 1/3$ in time slots $t = 0, \ldots, L$. The spatial coupling is obtained by connecting each block of the chain to the one on the right and to the one on the left, as illustrated in Figure 4.1. The information sequence $\mathbf{u}_t$ in Figure 4.1 is divided into two sequences $\mathbf{u}_{t,\text{previous}}$ and $\mathbf{u}_{t,\text{current}}$ by a demultiplexer. A copy of the information sequence $\mathbf{u}_t$ is reordered by a permutation Π_t and then divided into two sequences $\mathbf{u}_{t,\text{previous'}}$ and $\mathbf{u}_{t,\text{current'}}$ by another demultiplexer. The input of the upper encoder in the PCC at time t is $(\mathbf{u}_{t-1,\text{previous}}, \mathbf{u}_{t,\text{current}})$ reordered by a permutation Π_t^{Upper}. And the input of the lower encoder is $(\mathbf{u}_{t-1,\text{previous'}}, \mathbf{u}_{t,\text{current'}})$ reordered by a permutation Π_t^{Lower}. Figure 4.1 shows a block diagram of the encoder of spatially coupled parallel concatenated codes with $m = 1$. At time t the dashed line represent the information bits from the previous time slot $t-1$ and the solid line represent the information bits from the current time slot t that is used in the next time slot $t+1$. The information sequences at the end of the chain are choosen in a way that the output at time $t = L+1$ becomes $v_{L+1} = 0$. This is done to terminate the encoder of the SC-PCC. The transmitted sequence through the channel will be at any time instant t, $\mathbf{v}_t = (\mathbf{u}_t, \mathbf{v}_t^{\text{Upper}}, \mathbf{v}_t^{\text{Lower}})$.

As we have mention in Chapter 3 parallel concatenated code can be decoded using iterative message passing (belief propagation) decoding. The belief propagation decoding of spatially coupled parallel concatenated codes can be visualized with the help of the compact graph, as illustrated in Figure 4.2. In Figure 4.2 we can see that at time t the information message is divided into two sequences $\mathbf{u}_{t,\text{previous}}$ and $\mathbf{u}_{t,\text{current}}$ by a demultiplexer. A copy of the information sequence $\mathbf{u}_t$ is reordered by a permutation Π_t and then divided into two sequences $\mathbf{u}_{t,\text{previous'}}$ and $\mathbf{u}_{t,\text{current'}}$ by another demultiplexer. The input of the upper decoder in the PCC at time t is $(\mathbf{u}_{t-1,\text{previous}}, \mathbf{u}_{t,\text{current}})$ reordered by a permutation Π_t^{Upper} and the parity bits. It also receives a-priori information on the systematic bit from the lower decoder at time instant $t-1$, t and $t+1$. The input of the lower decoder is $(\mathbf{u}_{t-1,\text{previous'}}, \mathbf{u}_{t,\text{current'}})$ reordered by a permutation Π_t^{Lower} and the parity bits. It also receives a-priori information on the systematic bit from the upper decoder at time instant $t-1$, t and $t+1$ [23]

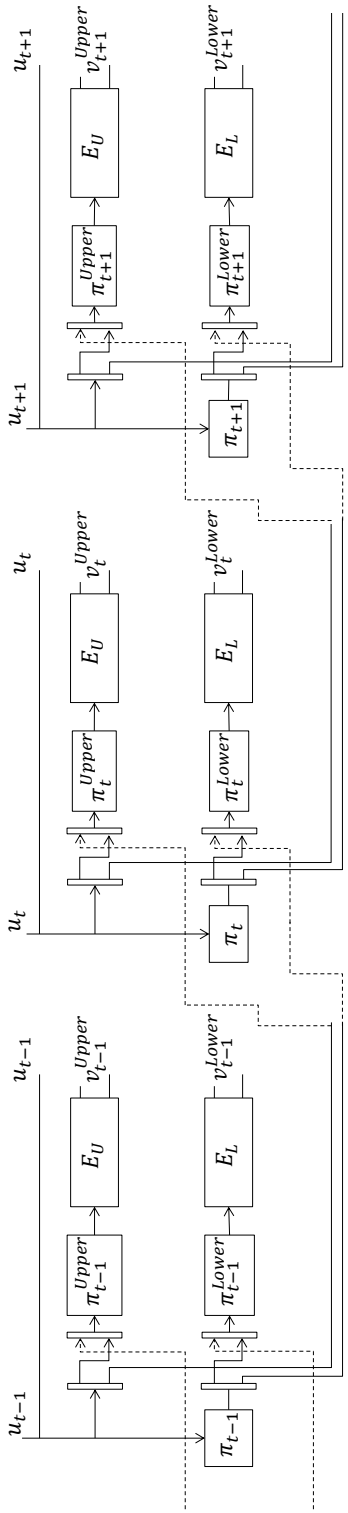

Figure 4.1: Block diagram of spatially coupled parallel concatenated codes

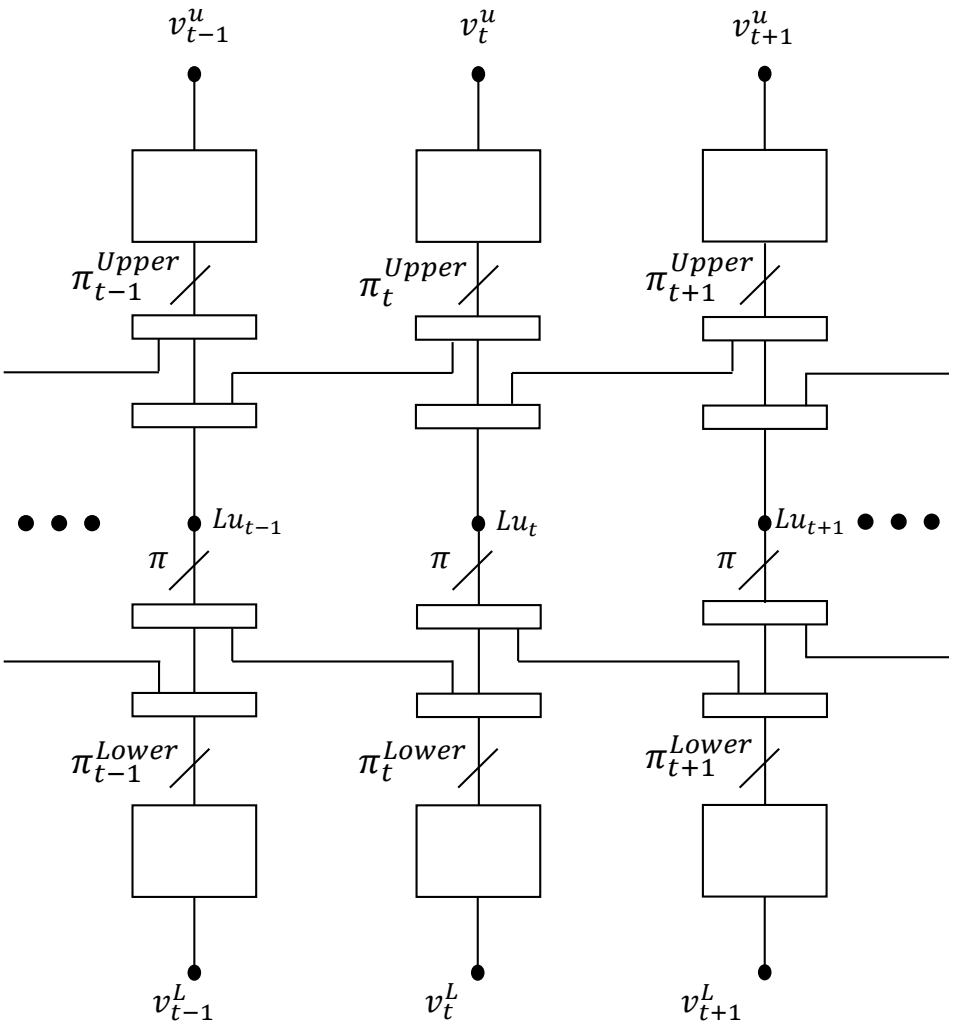

Figure 4.2: Compact graph of SC-PCC decoding process

4.2 Braided convolutional codes: a class of spatially coupled codes

As we point out early in this report, the main difference between BCC and PCC is that in BCC the parity sequence of the upper component encoder $\mathbf{v}^U$ is used as input for the lower encoder and vice versa with $\mathbf{v}^L$. Two different ensembles introduced in [28] are described in this section. BCC Type I is the generalization of original BCCs, which let us have higher coupling memories. And Type-II BCC in which the information symbols are coupled over the time instants together with the parity sequences.

4.2.1 Type I

It is considered the same rate-2/3 systematic convolutional encoder from the Figure 2.2 as each of the component encoders E_U and E_L of the BCC encoder. The Figure 4.3 represent the original BCC or BCC Type I with coupling memory equal to one. At the instant t, the first input of the upper encoder E_U is the information sequence $\mathbf{u}_t$, the second input is the permuted version with Π_t^U of the parity sequence of the lower encoder generated in previous time instant $\mathbf{v}_{t-1}^L$. In the same way, the fist input of the lower encoder E_L is the permuted version of $\mathbf{u}_t$ and the second input is the parity symbol from the previous time $\mathbf{v}_{t-1}^U$, after being reordered by the permutator Π_t^L.

In order to initialize the encoder, the values coming from $t < 1$ will be set as zeros since we do not have any parity sequence known. The transmitted sequence through the channel will be at any time instant t, $\mathbf{v}_t = (\mathbf{u}_t, \mathbf{v}_t^U, \mathbf{v}_t^L)$. Regardless $\mathbf{v}_t$ is the codeword transmitted, and as we saw in chapter two, terminations bits are created by the encoder in order to terminate in the zero state for both, systematic and parity sequences. We will also transmit these termination bits with the same effect of the channel in order to feed the next time encoder for coupling purposes.

The same restriction about the tail bits that drive the overall encoder to the zero state is not straightforward. So a suboptimal but simpler approach is to add a tail of zero bits to the information sequence. The length of the tail of zeros is a function of a new parameter which we will see at the end of this chapter (window size).

The main difference between the iterative decoding presented in the last chapter for the uncoupled BCC and SC-BCC is that in the coupled case, each decoding block receives *L-values* from the channel and the decoders at the same time instant t, and the neighbors blocks in $t-1$ and $t+1$. We use the help of the compact graph to show how the decoders exchange messages in different time instants as depicted in Figure 4.4. We know that the symbols coming from $t<1$ and $t>L$ are zero, so the *L-values* must be set to $+\infty$ [25].

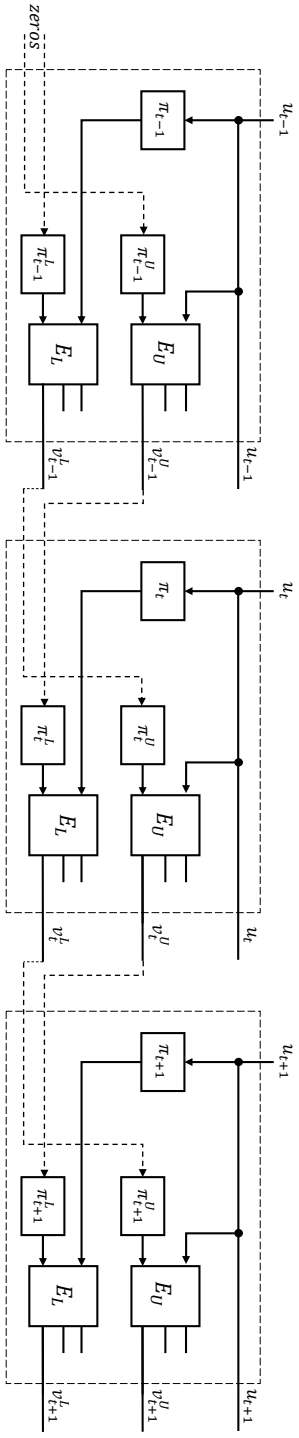

Figure 4.3: BCC encoder Type I

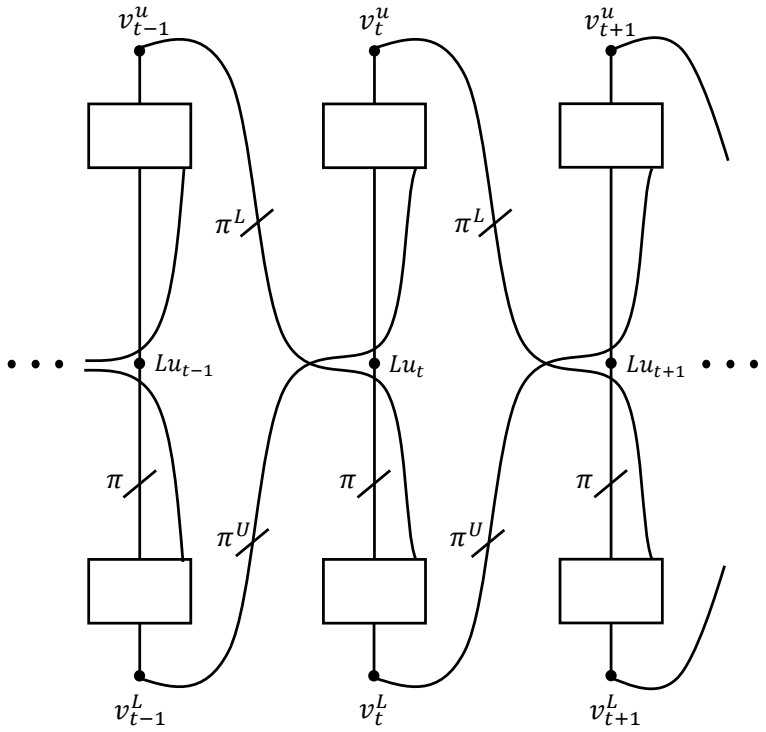

Figure 4.4: Coupled BCC decoder Type I

The first input of the Upper decoder at the time instant t is the *L-value* of the information symbol L_{u_t} plus the extrinsic value of the first output of the lower decoder after being de-permuted by Π^{-1}. The second input is the permuted version with Π^U of the summation of the third output of lower decoder and the parity symbol $L_{v^L_{t-1}}$ coming from $t-1$. And the third input is the parity sequence $L_{v^u_t}$ plus the second output of the lower decoder at $t = t+1$ de-permuted by Π_L^{-1}.

In the same way for the lower decoder at t, the first input is the permuted version of the information L_{u_t} after Π plus the fist output of the upper decoder at t. The second input comes from the previous time instant $t-1$, de-permuted version with Π_L^{-1} of the third output of the upper decoder plus the parity $L_{v^u_{t-1}}$ and the third input is the de-permuted with Π_U^{-1} version of the second output of the upper at $t+1$ plus the parity sequence $L_{v^L_t}$.

4.2.2 Type II

BCC Type II differs from BCC Type I in the first input of the upper(E_U) and lower(E_L) encoders. Figure 4.5 shows the block diagram of BCC Type II for coupling memory $m = 1$. According to this figure the information sequence $\mathbf{u}_t$ at time t is divided into two parts $\mathbf{u}_{t,\text{previous}}$ and $\mathbf{u}_{t,\text{current}}$. A copy of the information sequence $\mathbf{u}_t$ is reordered by a permutation Π_t and then divided into two parts $\mathbf{u}_{t,\text{previous'}}$ and $\mathbf{u}_{t,\text{current'}}$. The first input of the upper decoder is $(\mathbf{u}_{t-1,\text{previous}}, \mathbf{u}_{t,\text{current}})$ reordered by a permutation $\Pi_t^{U,1}$, the second input is the permuted version with $\Pi_t^{U,2}$ of the parity sequence of the lower encoder generated one time instant before $\mathbf{v}_{t-1}^{\text{Lower}}$. Likewise, the input of the lower encoder is $(\mathbf{u}_{t-1,\text{previous'}}, \mathbf{u}_{t,\text{current'}})$ reordered by a permutation $\Pi_t^{L,1}$ and the second input is the parity symbol from the previous time $\mathbf{v}_{t-1}^{\text{Upper}}$, after being reordered by the permutator $\Pi_t^{L,1}$. At time t the dashed line in Figure 4.5 represent the information bits from the previous time slot $t-1$ and the solid line represent the information bits from the current time slot t that is used in the next time slot $t+1$.

In order to initialize the encoder, the values coming from $t < 1$ will be set as zeros since we do not have both parity and information sequence known. The transmitted sequence through the channel will be at any time instant t, $\mathbf{v}_t = (\mathbf{u}_t, \mathbf{v}_t^{\text{Upper}}, \mathbf{v}_t^{\text{Lower}})$. Regardless $\mathbf{v}_t$ is the codeword transmitted, and as we saw in chapter two, terminations bits are created by the encoder in order to terminate in the zero state for both, systematic and parity sequences. We will also transmit these termination bits with the same effect of the channel in order to feed the next time encoder for coupling purposes. The same technique for termination which is used in Type I is also used for Type II.

The decoding of braided convolutional codes for Type II can be visualized with the help of the compact graph, as illustrated in Figure 4.6. In Figure 4.6 we can see that at time t the information message is divided into two sequences $\mathbf{u}_{t,\text{previous}}$ and $\mathbf{u}_{t,\text{current}}$ by a demultiplexer. A copy of the information sequence $\mathbf{u}_t$ is reordered by a permutation Π_t and then divided into two sequences $\mathbf{u}_{t,\text{previous'}}$ and $\mathbf{u}_{t,\text{current'}}$ by another demultiplexer. The first input of the upper decoder in the time instant t is $(\mathbf{u}_{t-1,\text{previous}}, \mathbf{u}_{t,\text{current}})$ reordered by a permutation $\Pi_t^{U,1}$ and the parity bits. It also receives a-priori information on the systematic bit from the lower decoder at time instant $t-1$, t and $t+1$. The second input is the permuted version with $\Pi^{U,2}$ of the summation of the third output of lower decoder and the parity symbol $\mathbf{v}_{t-1}^{L}$ coming from $t-1$. And the third input is the parity sequence $\mathbf{v}_t^u$ plus the second output of the lower decoder in $t = t+1$ depermuted by $\Pi_{L,2}$.

In the same way for the Lower decoder in time instants t, the first input is $(\mathbf{u}_{t-1,\text{previous'}}, \mathbf{u}_{t,\text{current'}})$ reordered by a permutation $\Pi_t^{L,1}$ and the parity bits. It also receives a-priori information on the systematic bit from the upper decoder at time instant $t-1$, t and $t+1$. The second input comes from the previous time instant $t-1$, depermuted version with $\Pi^{L,2}$ of the third output of the upper decoder plus the parity $\mathbf{v}_{t-1}^u$ and the third input is the depermuted with $\Pi^{U,2}$ version of the second output of the upper in $t+1$ plus the parity sequence $\mathbf{v}_t^L$.

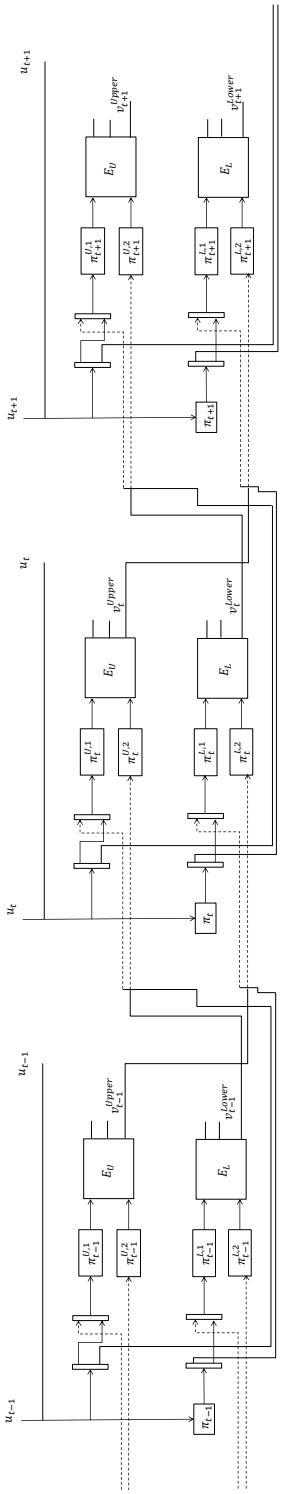

Figure 4.5: BCC encoder Type II

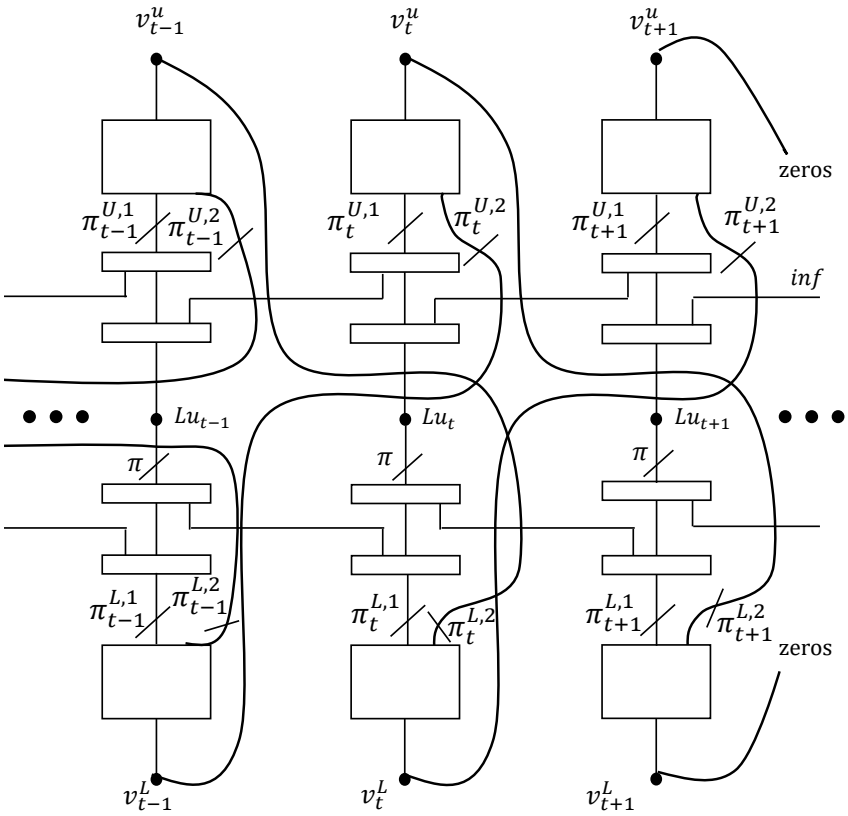

Figure 4.6: Coupled BCC decoder Type II

4.3 Sliding window decoder

In order to combat the complexity of the belief propagation, a *windowed decoder (WD)* is used to exploit the structure of the spatially coupled codes and decompose the *BP* scheme into suboptimal decoding steps but maintaining the advantages in terms of performance [22].

In order to illustrate how the window decoder works we can refer to the Figure 4.7, it shows a sliding window of size $w = 3$ which is shifted 3 times on a iteration message passing decoder with coupling length $L = 6$. As we previously presented in the parallel concatenated section, each decoding block is formed by an upper and a lower decoder. An active research is going on regarding the influence of the size of the window. Also the scheduling between the upper and lower decoders of each block inside the window is under investigation to find the optimal one.

Every position of the window decoder will perform a certain number of itera-

tions. For instance, for iteration number $I = 10$, the message passing decoder algorithm will go through every decoding block 10 times, then it will be shifted one position to the right and perform the same number of iterations starting from the second decoding block. With this behavior, the message is being updated, in total, the size of the window w times the number of iterations. Thus it can achieve a more accurate result when the iteration number increases.

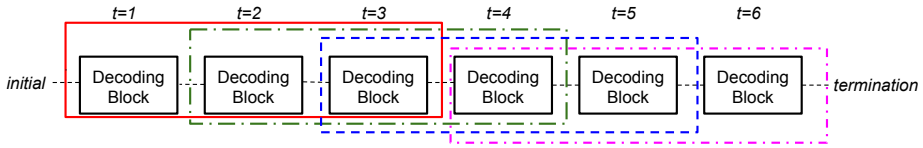

Figure 4.7: Window decoder

The simulation results that we will present use different window sizes in order to compare the impact on the performance, and the sequence is set to activate the upper decoder in the first decoding block, then the lower decoder and so on to the following blocks.

Chapter 5
Results

This chapter focuses on the error performance simulations as a function of the signal to noise ratio (SNR) E_b/N_0 for the special ensembles of PCCs and BCCs presented in Chapter 3 and 4. The simulation program and most of the functions were implemented in Matlab but the BCJR algorithm was implemented in C++. The advantage of the C++ implementation is the time spent on the loops. The algorithm was implemented in both programming languages having as result that C++ implementation was 10 times faster than Matlab. The performance is evaluated on an additive white Gaussian noise (AWGN) channel. Since many different ensembles were simulated, we used the Alaric Lunarc cluster facilities based in Lund. Alarik is a SouthPole solution with 208 nodes containing two 64-bit, 8-core AMD6220 (3.0 GHz), corresponding to a total of 3328 processors [29].

5.1 PCC uncoupled vs BCC uncoupled

The ensemble presented on Chapter 3, Sections 3.2 and 3.3 correspond to PCC and the uncoupled version of BCC respectively. We used two identical $R = 1/2$, 8 states component encoders which follow the generator matrix $G = (1 \ \frac{1+D+D^3}{1+D^2+D^3})$ for PCC. For uncoupled BCC, two identical $R = 2/3$, 4 states, component encoders with the generator matrix as in *Example 2.3* are used. Iterative decoding with the BCJR algorithm and $I = 100$ decoding iterations were used for all the ensembles. Permutation sizes of 1000 and 8000 are shown for both ensembles plus a 80000 permutation size for BCC uncoupled.

We can see in Figure 5.1 that PCC with a permutation length of 1000 exhibits an error floor at a BER of 10^{-5} and E_b/N_0 of 1.4 dB. By increasing the permutation size up to 8000 it achieves a steeper slope on the waterfall region and a error floor at a BER of 10^{-6} and E_b/N_0=0.5dB due to the enhancement of the minimum distance. At the same time the BCC do not show any error floor for any of the permutation lengths simulated. At low E_b/N_0 PCC has better performance than BCC, on the other hand for permutation size of 1000 BCC has better performance after E_b/N_0=2dB and at 1.2dB for 8000. A density evolution threshold has been calculated in [30] with a value of 0.98dB, by tracking the probability density functions of the decoder output L-values. For a permutation size of 1000, BCC achieves BER level under 10^{-6} at an E_b/N_0=2dB but as we keep increasing the permuta-

tion size up to 80 000, it can achieve a BER of 10^{-7} with a E_b/N_0=1.052dB, 0.07 dB away from the threshold and 0.3dB for permutation size of 8000.

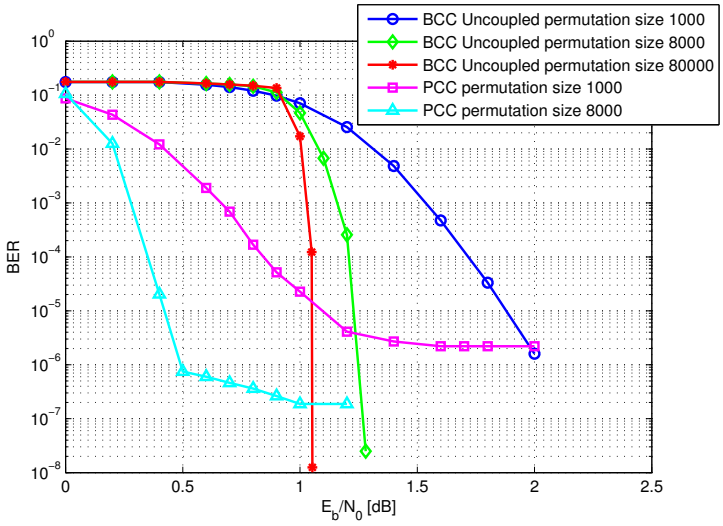

Figure 5.1: Error performance of R=1/3 BCC and PCC on a AWGN channel

5.2 PCC (coupled vs uncoupled)

In this section we present the simulation results for comparing the performance of the parallel concatenated codes and the spatially coupled parallel concatenated codes for permutation size of 1000 and 8000. For the spatially coupled case we consider two identical rate R=1/2 component encoders in the arrangement presented in Figure 4.1, with the same generator matrix as in PCC. A coupling length of L=100 is considered. Sliding window BCJR decoder with different decoding iterations is used. Different window sizes are used in order to investigate the impact in the performance.

Figure 5.2 shows the performance of PCC ensemble with permutation size 1000. We can also see the performance of the coupled case with similar permutation size for window size 5, 6 and 10 with 10 iteration each. The total number of iterations will be the product of the window size times the iteration number. That will be 50, 60 and 100 respectively. The waterfall region improves dramatically, reaching up to 0.8dB between PCC and SC-PCC with w=10. The three different windows exhibit an error floor at a BER in the order of 10^{-5} and E_b/N_0 of 0.4dB, while for the PCC version the error floor starts at E_b/N_0 of 1.2 dB. The uneven shape of the error floor suggest an affectation of error burst and perhaps a higher number of iterations will make it look smoother. We can also see that the performance improves even more when we increase the number of iterations, but at a w=6, the

performance will not improve much further.

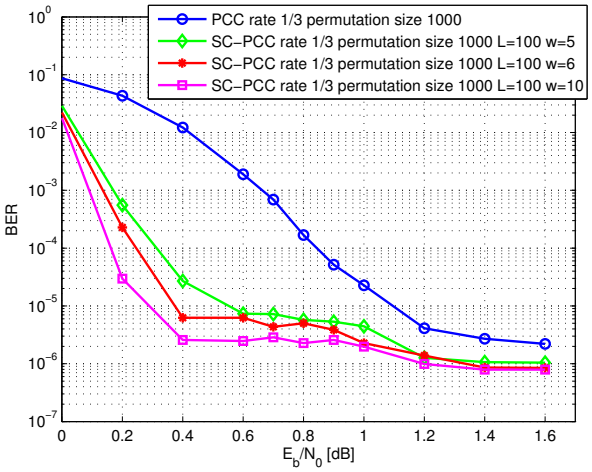

Figure 5.2: Error performance of R=1/3 SC-PCC permutation size 1000 on a AWGN channel

The effects of increasing the permutation size is shown in Figure 5.3. Where the waterfall region is closer to capacity and has a very steep slope. It achieves a gain up to 0.4 dB when comparing the w=10 with uncoupled case. For w=5 and 10, the error floor starts at E_b/N_0 of 0.15 dB and a BER of 10^{-6}. With the permutation size of 8000 is more evident that w=6 will already give a good performance and attempting a higher window size will not impact significantly.

The best results for each permutation length are presented in Figure 5.4 together with the uncoupled versions of them. As we can see, SC-PCC with permutation size 1000 has a better waterfall curve than the uncoupled PCC with 8000 as permutation size but because of the size of the interleaver it presents a worse error floor BER.

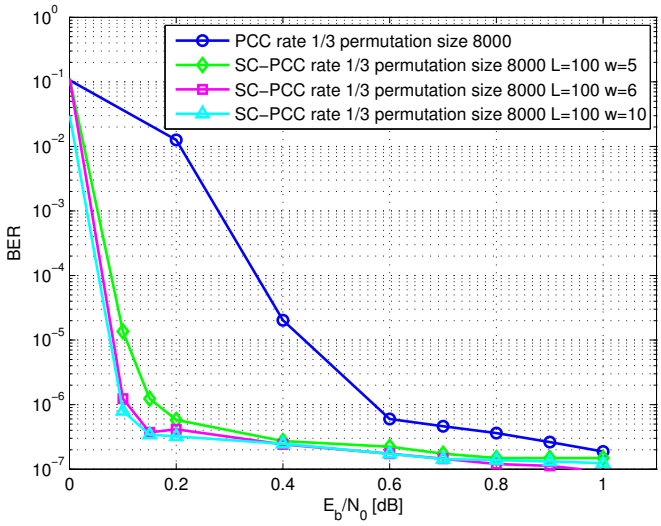

Figure 5.3: Error performance of R=1/3 SC-PCC permutation size 8000 on a AWGN channel

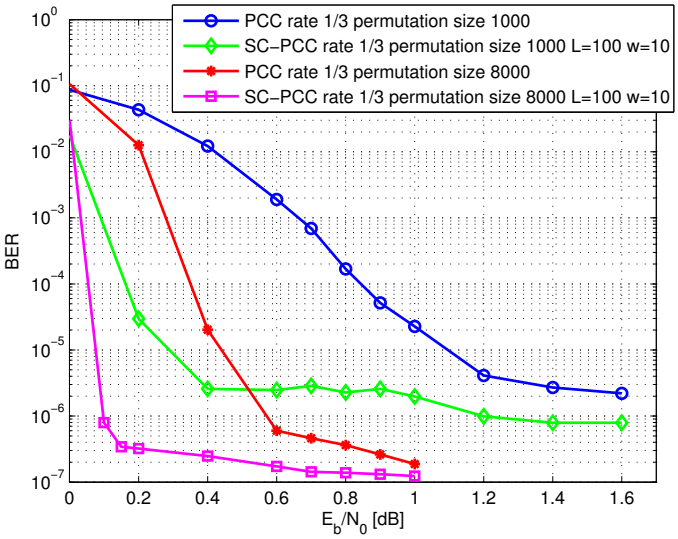

Figure 5.4: Error performance of R=1/3 SC-PCC. Best results on a AWGN channel

5.3 BCC Type I & II (coupled vs uncoupled)

We consider two identical rate R=2/3 component encoders in the arrangement presented in Figure 4.3 with the generator matrix in *Example 2.3*. The interleavers Π,Π^L and Π^U were constructed randomly for Type I. The same for interleavers of Type II which correspond to Figure 4.5. The coupling length for both types is *L=100*. A zero tail of *w-1* blocks is added since the termination is not straightforward, where w is window size. Thus we have a rate loss in both types. Sliding window BCJR decoder with different decoding iterations is used. Different window sizes are used in order to investigate the impact in the performance. Two permutation sizes are simulated, 1000 and 8000.

The result for a block length of 1000 is presented in Figure 5.5, where we can see a window w=5 and 80 iterations for both types compared to the uncoupled case. Type I reaches a BER of 10^{-4} at a E_b/N_0 of 1.3 dB while Type II reaches a BER performance of 10^{-7} at a E_b/N_0=0.45dB. The performance for Type I is very poor, we do not know the reason. Further investigation is required to make a conclusion. Meanwhile the performance of Type II shows a normal waterfall curve.

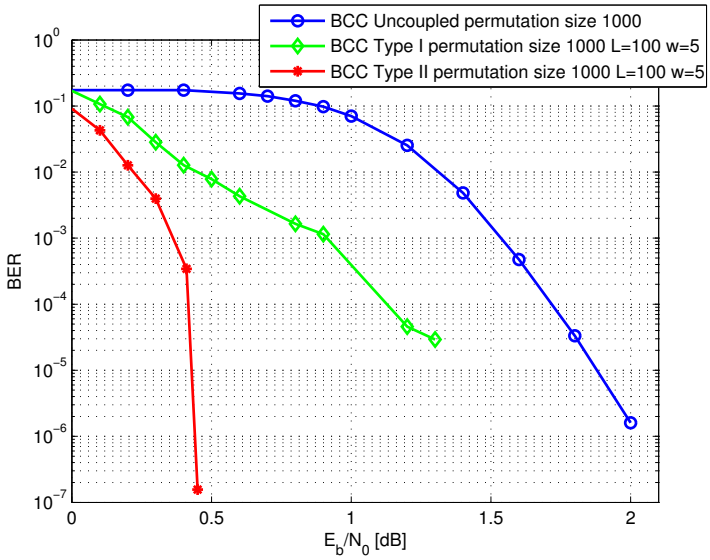

Figure 5.5: Error performance of rate R=1/3 SC-BCC permutation size 1000 on a AWGN channel

Two different setting for the BCC coupled case Type I and Type II are shown in Figure 5.6 for a permutation size of 8000, a window *w=5* with *I=80* iterations, and a *w=20* with *I=20* iterations. Like we described in the case of SC-PCC, the total number of iterations is the product of the window size times the iteration number. The effect of the coupling improves dramatically the performance in more

than 1dB for Type I and 1.47dB for Type II. We can see that with a window w=5, BCC achieves a BER of 10^{-5} at an E_b/N_0 of -0.05dB while with w=20 it reaches the same BER at 0.05dB for Type I. In the case of Type II with a window w=5, BCC achieves a BER of 10^{-6} at an E_b/N_0 of -0.192dB while with w=20 it reaches the same BER at -0.15dB. We can see that BCC do not present error floor which we expected after the uncoupled case. The effect of the iterations from 80 to 20 is stronger than the impact of the window size. Finally as is clearly shown, Type II have a better performance than Type I.

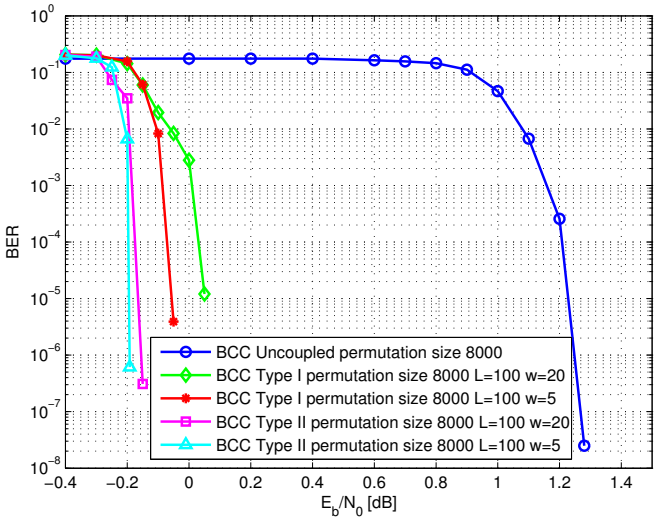

Figure 5.6: Error performance of rate R=1/3 SC-BCC permutation size 8000 on a AWGN channel

5.4 PCC coupled vs BCC coupled

In this section we present the best results that we have gotten from both coupled cases of PCC and BCC for Type I & II. We can see in Figure 5.7 that BCCs do not have a visible error floor. The BER performance is significantly higher in both SC-BCC compared with the SC-PCC. There is a difference of about 0.15dB if we compare the waterfall region of the SC-PCC with the SC-BCC Type I and 0.3dB with the SC-BCC Type II. We can conclude that BCC has a better performance with low E_b/N_0 and is close to Shannon capacity. Type II has a slightly better performance than Type I due to the effect of coupling both information sequences and parity symbols.

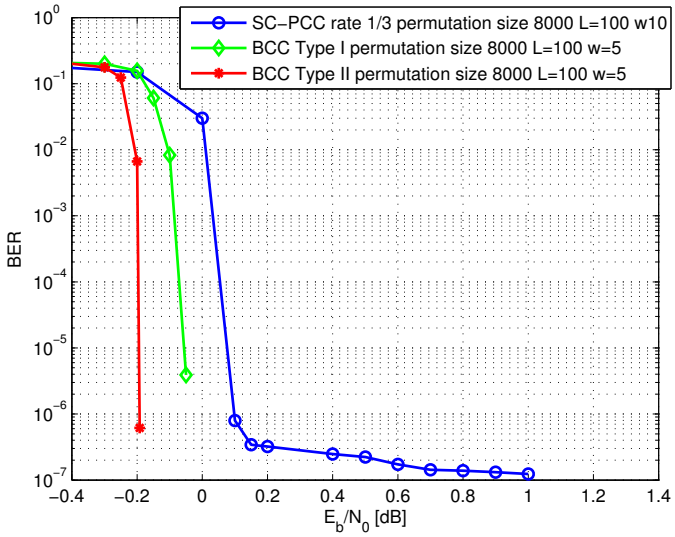

Figure 5.7: Error performance of rate R=1/3 SC-BCC Type I&II vs R=1/3 SC-PCC on a AWGN channel

Chapter 6
Conclusions

In this thesis the performance of spatially coupled convolutional codes has been described. Three different ensembles were constructed for coupled codes, one for PCC and two types for BCC. Uncoupled PCC present better BER performance with high SNR for small and long permutator lengths compared with uncoupled BCC, although BCC didn't show any error floor. On the other hand, when high E_b/N_0 the uncoupled BCC perform better. Spatial coupling gives a significant BER performance improvement overall. For SC-PCC gain up to 0.8dB and 0.4dB was reached compared to the uncoupled case for short and long length respectively. We can also conclude that with the coupling PCCs get slightly better distance properties. Results of SC-BCC with a small length suggest that the window decoder could get stuck in some error burst and the simulations are not reliable. Further investigation is suggested in this scenario. Moreover when the block length is higher the performance is dramatically enhanced for both constructions, getting closer to the Shannon limit. The ensemble Type II showed significant performance improvement over the "original" braided construction (Type I). Thus coupling both information symbols and parity symbols brings a performance enhancement. No error floor shown suggest that the braided convolutional codes have good distance properties. The results presented in this work give one step forward in the current research of the feasibility of spatially coupled codes.

6.1 Future work

An s-random interleaver was implemented and the performance improvements over the parallel concatenated convolutional codes were shown in section 3.2, Figure 3.7. We know that the distance properties improve over the PCC but an interesting approach would be the effect of the s-random interleaver over the BCC and SC-BCC.

As it was pointed out at the beginning of chapter 4, in all the coupled ensembles a coupling memory m=1 was used. It has already been shown in [28] that increasing m would lead to a performance improvement. Some modifications can be done to the simulations tools implemented in this work in order to compare the impact on the BER performance with higher coupling memory.

The number of iterations is not even in each decoding block inside the whole coupled chain. The blocks in the middle will have more iterations compared to the first blocks. Further research must be done regarding the scheduling inside the fist window positions to see the impact of the performance.

References

[1] C. E. Shannon, *A Mathematical Theory of Communications* Bell Syst. Tech J., 27:379-423, July 1948.

[2] Johannesson R, and K.S. Zigangirov, *Fundamentals of Convolutional coding* Wiley-IEEE Press, 1999.

[3] Daniel J. Costello Jr., *Error Control Coding Fundamentals and Application Second edition* Pearson Prentice Hall. 2004.

[4] Michael Lentmaier *Lecture notes, Principles of error control coding, Part III Convolutional codes* EDI042 Error Control Coding, p.6 2014/2015

[5] Rajpura, Punjab *Wireless commmunicatios T L Singal* Tata McGraw-Hill Education, 2010. Department of electronic and communictaion Engineering Chitkara Institute of Engineering and Technology

[6] Silvio A. Abrantes, *From BCJR to turbo coding: MAP algorithms made easier*, Information and Telecommunication Technology Center (ITTC) of the University of Kansas, Lawrence, USA April 2004

[7] Michael Lentmaier *Lecture notes, Optimal Decoding Methods, Part III* EDI042 Error Control Coding, p.1-14 2014/2015

[8] D. J. Costello, Jr: *A construction technique for random error correcting convolutional codes*, IEEE Trans. on Inform. Theory, IT-19:631-636. Sep. 196

[9] A.J. Viterbi, *Error Bounds for Convolutional Codes and an asymptotically Optimum Decoding Algorithm*, IEEE Trans. on Inform. Theory, IT-13:260-269. April. 1967

[10] J.K. Omura, *On the Viterbi Decoding Algorithm*, IEEE Trans. on Inform. Theory, IT-15:117-179. January. 1969

[11] L.R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, *Optimal Decoding of Linear Codes for Minimizing Symbol error Rate*, IEEE Trans. on Inform. Theory, IT-20:284-289. March. 1974

[12] G. David Forney, Jr., *Concatenated Codes*, Department of Electrical Engineeing, M.I.T, March. 1965

[13] Sergio Benedetto, Guido Montorosi, *Design of parallel Concatenated Convolutional Codes*, IEEE Trans. on Comm. VOL.. 44, NO.5, p:591-600. May. 1996

[14] Ali Ghrayeb, Taher Abualrub, , *On Parallel and Serial Concatenated Convolutional Codes over GF(4)*, IEEE ICCS p:327-331. 2002

[15] S. Bendetto, D. Divsalar, G. Montorsi and F. Pollara *A Soft-Output Maximum A Posteriori (MAP) Module to Decode Parallel and Seriacal Concatenated Codes*, TDA Progress Report p:1-20. November 1996

[16] C. Berrou, A. Glavieux and P. Thitimajshima, *Near Shannon Limit Error Correcting Coding and Decoding:Turbo Codes*, Proc. IEEE Intl. Conf. Commun. (ICC 93), pp.1064–70, Geneva, Switzerland, May 1993

[17] J. Hokfelt, O. Edfors and T. Manseg, *A Turbo Code Interleaver Design Criterion Based on the Performance of Iterative Decoding*, IEEE Communications letters, Vol 5, No 2, February 2001

[18] S. Dolinart and D. Divsalar, *Weight Distributions for Turbo Codes Using Random and Nonrandom Permutations*, TDA Progress Report 42–122, August 1995

[19] J. Hagenauer, E. Offer and L. Papke, *Iterative decoding of binary block and convolutional codes*, IEEE Trans. Inform. Theory, Vol 42. pp 429-445. Mar. 1996

[20] Michael Lentmaier *Lecture notes,Iterative Decoding of Concatenated Codes, Part II* EDI042 Error Control Coding, p.1-4 2014/2015

[21] M. Lentmaier, D. Truhachev and K. Zigangirov *To the theory of low density convolutional codes II* Problems of information transmission, vol. 37, pp 15-35, Oct.-Dec. 2001.

[22] A. Iyengar, P. Siegel, R. Urbanke and J. Wolf, *Windowed Decoding of Spatially Coupled Codes* IEEE Transactions on information theory, vol 59, No 4, April 2013.

[23] S. Moludi, M. Lentmaier, and A. G. Amat *Spatially coupled Turbo codes* International Symposium on Turbo Codes & Iterative Information Processing (ISTC), Bremen, Germany, 2014-08-18/2014-08-22

[24] D. Truhachev, M. Lentmaier, and K. Zigangirov *On braided block codes* IEEE International Symposium on Information Theory, Yokohama Japan, Jun 2003, p 32.

[25] S. Moloudi, and M. Lentmaier *Density Evolution Analysis of Braided Convolutional Codes on the Erasure Channel* IEEE International Symposium on Information Theory (ISIT), Honolulu, USA. Jun 2014.

[26] W. Zhang, M. Lentmaier, and K. Zigangirov *Braided Convolutional Codes: A new class of turbo-like codes* IEEE Transactions on Information theory, vol. 56:1, pp 316-331, 2010.

[27] S. Moloudi, M. Lentmaier, and A. Graell i Amat *Threshold Saturation for Spatially Coupled Turbo-like Codes over the Binary Erasure Channel* Still no publication data

[28] M. Lentmaier, S. Moloudi, and A. Graell i Amat *Braided Convolutional Codes - A Class of Spatially Coupled Turbo-Like Codes* International conference on signal processing and communications (SPCOM), 2014-07-22, Bangalore India.

[29] http://www.lunarc.lu.se/Systems/AlarikDetails

[30] M.B.S. Tavares, M. Lentmaier, G.P. Fettweis, and K.Sh. Zigangirov, *Asymptotic distance and convergence analysis of braided protograph convolutional codes,* in Proceedings of the 46th Annual Allerton Conference on Communication, Control and Computing, Monticello, IL, Sept. 2008.