Comparison of Various Concatenated Convolutional Code Ensembles under Spatial Coupling

GABRIEL IRO
RAJESHWARI KABBINALE

MASTER´S THESIS

DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY |
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

Comparison of various concatenated convolutional code ensembles under spatial coupling

Gabriel IRO
Rajeshwari KABBINALE

Department of Electrical and Information Technology
Lund University

Advisor:
Michael Lentmaier
Saeedeh Moloudi

January 23, 2017

Printed in Sweden
E-huset, Lund, 2017

Abstract

A big challenge faced by the digital communication world today is increasing the reliability of information which is being transmitted. During the transmission of information, there is a possibility that the information is corrupted or distorted. If this distortion is considerably high, it becomes difficult to decode and retrieve the original information. To help mitigate this effect, the information bits are encoded and the induced errors can be reduced or completely removed after decoding. This process of protecting the information bits is called error control coding (ECC). It is introduced to provide reliable transmission of information over the channel and this is done by adding redundancy to the message that is to be transmitted.

There are ongoing studies in different implementations of error control coding schemes based on both convolutional codes and block codes. One technique introduced to improve the performance of the codes is spatial coupling. This technique was first introduced for low-density parity check codes (LDPC) codes, however, spatial coupling is a general concept, and it can be applied on other classes of error control codes such as turbo-like codes (TCs). There is research at the Department of Electrical and Information Technology at Lund University on different construction of spatially coupled turbo-like codes such as: spatially coupled parallel concatenated codes (SC-PCCs), braided convolutional codes (BCCs), spatially coupled serially concatenated codes (SC-SCCs) and spatially coupled hybrid concatenated codes (SC-HCCs). These codes have shown to have asymptotically good performance, but their performances in the finite length regime is currently under investigation.

In this thesis we investigated the performance of SCCs and HCCs under spatial coupling in the finite length regime. At the beginning of our investigation, we defined the spatially coupled ensembles, then, implemented these ensembles in Matlab and C++. Thereafter, we simulated the ensembles considering different puncturing patterns to obtain higher code rates. We then compare our results with those for BCC and SC-PCC which has been carried out previously in the department.

The results show that all the investigated codes perform better when they are spatially coupled than when they are uncoupled. Among all the considered ensembles, BCC with spatial coupling is comparably the best performing code overall especially with an increased block length. Moreover, our results also shows that the pattern of puncturing applied on the code affects the performance of the code. The puncturing pattern which gives the better performance for uncoupled codes may not necessarily give the better performance for the coupled codes.

Acknowledgments

This Thesis would not exist without the support and guidance of

Supervisor: Michael Lentmaier, We thank him for all his help, guidance, discussion and feedback. We are also grateful for the time given during the entire thesis.

Co-Supervisor: Saeedeh Moloudi, We thank her for having patience to listen to our doubts and helping us all the time in her busy schedules.

Finally, we would like to thank our families and friends, for the support and encouragement.

Preface

In collaboratiotion with the Department of Electrical and Information Technology (EIT) this thesis work was carried out by both Rajeshwari Kabbinale and Gabriel Iro. The goal of authors was to construct different ensembles of spatially coupled convolutional codes and analyze their performance. Both authors have taken active part in most of the steps in this investigation. The two BCJR decoders were implemented collaboratively. The main responsibilities of Rajeshwari were to implement the spatially coupled serially concatenated codes. Gabriel implemented the spatially coupled hybrid concatenated code. Chapter 2 was written by Rajeshwari and Gabriel wrote Chapter 3, the rest of the report was written together.

Table of Contents

List of Figures

Chapter 1

Introduction

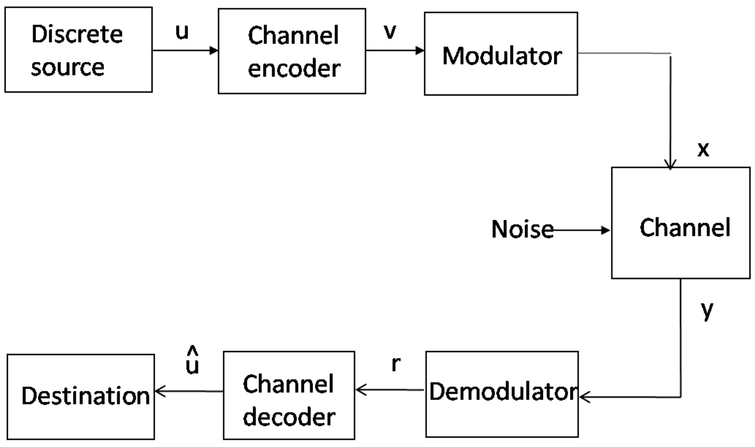


Figure 1.1: Block diagram of a communication system

A digital communication system is made up of several components, as seen in Figure 1.1. The information source can be any source of data, in a continuous waveform or discrete symbols. Here the information sequence and the source encoder are combined into a block called discrete source. Discrete source generates the information in the form of binary symbol with output **u**. The channel encoder takes these message bits and adds redundancy, thereby producing an encoded sequence **v** called codeword. Each output symbol from the channel encoder is transformed into a waveform by the modulator of duration T seconds that is suitable for transmission. This transmitted waveform gets corrupted by noise once it enters the channel.

In the receiver side, the demodulator processes the received wave forms from the channel and produces discrete/continuous output generating the received sequence **r**. The channel decoder transforms the received sequence **r** into a binary sequence **û** called the estimated information sequence. However, the output that we get from the channel decoder **û** is not equal to the sequence **u**. The noise added by the channel may cause some decoding errors. So the channel decoder exploits the redundancy to decide which message bits were actually transmitted.

In late 1940s, Shannon showed that it is possible to get low error probability using coding on any channel, provided that the bit-rate is below a threshold called “channel capacity” [1].

There are two main types of channel coding; block codes and convolutional codes. In this thesis, we consider convolutional codes. Two or more convolutional codes can be joined together to form a stronger code. This technique is called “concatenation”. When codes are concatenated in series, they are called “serially concatenated codes”[15][16], when they are concatenated in parallel, they are called “parallel concatenated codes” or “turbo codes”[13]. A combination of both serial and parallel concatenation is a hybrid concatenated code [14]. Braided convolutional code [17] is another type of concatenated code. They are like hybrid concatenated codes and are also formed by the combination of serially concatenated code and hybrid concatenated code.

It is shown that we can apply spatial coupling on these concatenated codes and get better asymptotic behavior [10], A research carried out in the department of Electrical and Information Technology at Lund University in this area is “spatially coupled turbo-like codes” in which the authors performed the density evolution (DE) analysis and computed the decoding thresholds and showed that the belief propagation (BP) decoder can achieve the threshold of maximum a posteriori (MAP) decoder of the underlying uncoupled ensembles occurs for large enough coupling memory [10].

Moreover, another master’s project carried out in the department is the analysis of finite length performance of spatially coupled convolutional codes [9]. The authors in this thesis investigated PCC and BCC for both when these ensembles were coupled and uncoupled in the finite length regime. However, the performances of SCC and HCC was not covered in their investigation.

1.1 Contribution

In this thesis, we investigated the performances of finite length hybrid and serially concatenated codes with and without spatial coupling. Firstly, we defined the spatially coupled ensembles to be used for this investigation. Then we implemented the ensembles in Matlab and C++. Afterwards, we applied puncturing to obtain higher code rates and then we simulated the ensembles. For the hybrid code, we investigated two different puncturing patterns. The results derived from our simulation were then compared with BCC and SC-PCC which has been carried out previously in the department [9]. In the results, we observed an improved BER performance for the SC-TCs in comparison to their respective uncoupled ensembles. BCC with spatial coupling is comparably the best performing code overall especially for higher block length. Moreover, the results also shows that the pattern of puncturing applied on the code affects its performance. The puncturing pattern which gives the better performance for the uncoupled code may not necessarily give the better performance for its coupled counterpart. This was observed in the results for the both puncturing patterns of the HCC ensembles.

1.2 Outline of Thesis

At the beginning of this thesis, in Chapter 2, we give a short introduction on convolutional codes. Different types of encoders are shown as examples. Trellis representation of one of the examples is shown and explained. Then the steps of BCJR algorithm is explained and finally we discuss basic encoding and decoding structure of parallel concatenated code[3].
In Chapter 3, we mainly discuss about SCCs [5] and HCCs and their encoding and decoding. We also introduce spatial coupling and present spatial coupled ensembles [10] of SCC and HCC. Finally, window decoder is also introduced.
In Chapter 4, we present simulation results of the configuration and compare coupled and uncoupled codes together with different scenarios and see the effect in the performance when different parameters are taken into consideration. We also compare the results of serially and hybrid concatenated convolutional codes with already simulated results of parallel and braided convolutional code [9].
Finally in Chapter 5, we conclude with a short discussion on future investigation.

Chapter 2

Theoretical Background

There are two different types of channel codes, block codes and convolutional codes. In block codes the information sequence is divided into message blocks of fixed length, to which a number of redundant bits are added. Each block of information is represented by

$$\mathbf{u} = (\boldsymbol{u}_0, \boldsymbol{u}_1 \cdots , \boldsymbol{u}_{k-1})$$

Each block of k data bits corresponds to n-tuple of bits called as *codeword.*

$$\mathbf{v} = (\boldsymbol{v}_0, \boldsymbol{v}_1 \cdots , \boldsymbol{v}_{n-1})$$

At the encoder output there are 2^k different possible codewords corresponding to the 2^k different possible messages. This set of 2^k codewords of length n is called an (n, k) block code. The ratio $R = \frac{k}{n}$ is called the code rate where k is the length of input block and n is the length of output block.

2.1 Convolutional Codes

In convolutional coding, each block of n coded bits not only depends on k data bits but also the $m \cdot k$ previous data bits, where m denotes the memory of the encoder. In convolutional encoder stream of information bits are taken and then converted into stream of transmitted bits. In a rate $R = k/n$ convolutional encoder with input memory m can be realized as k is input and n is output. Typically, k less than n, the information sequence and codeword is divided into blocks of length k and n, respectively. Low error probabilities and large minimum distance are achieved by increasing the memory order m and not only by increasing k and n.

Classification of the convolutional codes can be done in two ways, i.e. recursive, non-recursive and systematic, non-systematic. Recursive encoder uses both feed-forward and feedback paths and non-recursive encoder only uses feed-forward path in the encoding process. In the systematic encoder, the information sequence is not changed among the code sequences, the k input sequences are a copy of the first k sequences. In non-systematic encoder, due to the convolution process the output does not contain the information bits.

Example 2.1 *A rate R=1/2 recursive systematic convolutional encoder*
Consider polynomial generator matrix $G = (1 \quad \frac{1}{1+D})$

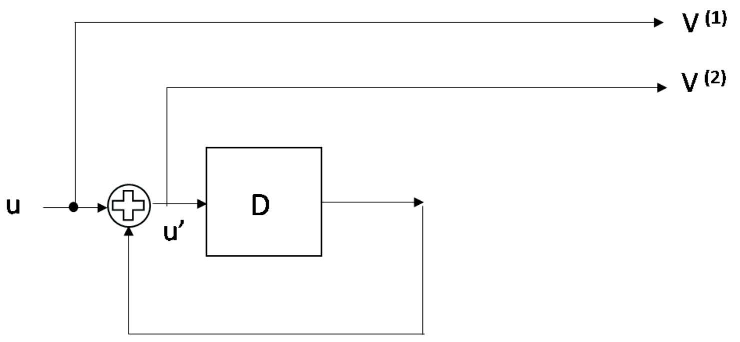


Figure 2.1: Rate=1/2 Recursive Convolutional Encoder

Here in the Figure 2.1, **u** is the information sequence that enters the encoder bit by bit and the encoder output sequence bits are denoted by $\mathbf{v}^{(1)}$ and $\mathbf{v}^{(2)}$. D is the delay operator and number of time units a bit is delayed with respect to initial bit in the sequence is denoted by the power of D. This is a systematic encoder. The output sequence $\mathbf{v}^{(1)}$ is the copy of information sequence. For the output sequence $\mathbf{v}^{(2)}$ the input bit is modulo 2 added to the stored values of previous input bit. This bit is fed back and modulo 2 added with input bit which gives $\mathbf{u}^{'}$ and output bit $\mathbf{v}^{(2)}$. $\mathbf{u}^{'}$ is then moved into the shift register.

Example 2.2 *A rate R=1/2 recursive systematic convolutional encoder*
Consider polynomial generator matrix $G = (1 \quad \frac{1+D^2}{1+D+D^2})$

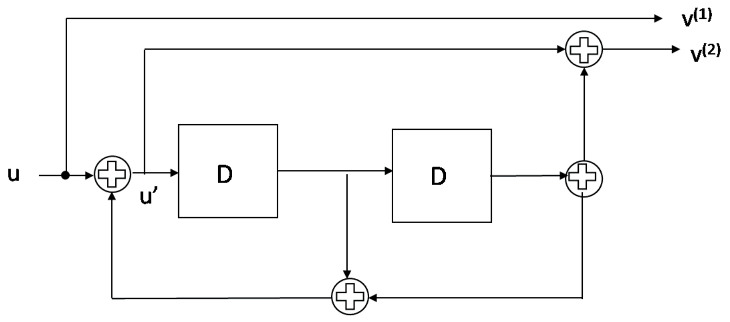


Figure 2.2: Rate=1/2 Recursive Systematic Convolutional Encoder

This is also a systematic encoder. The generator matrix is different here with two delay operator. Rest everything is similar to the Example 2.1.

Trellis representation
A graphical representation of a code shown in Figure 2.3, is called trellis. Any

encoded sequence or any codeword of convoulutional code is represented as a path on this graph. A trellis consists of nodes and branches. The nodes represent the encoder's state and every input sequence $\mathbf{u} = u_0, u_1, ...$ represent path or branches within the state transition diagram of the encoder. The labels on the branches are the codewords that the encoder outputs when that particular transition from a state at time t to a state at time $t+1$ is made in response to a input sequence u_t. Usually trellis start from state zero and end to state zero. In order to go back to the state zero, the termination bits are added. The termination bits are added according the number of memory of the encoder in order to take the encoder back to zero state and terminate the convolutional code.

Trellis diagram Example 2.2 is shown

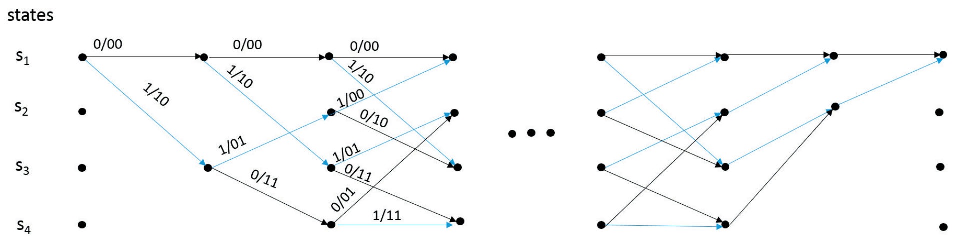


Figure 2.3: Trellis representation of Rate=1/2 Recursive Systematic Convolutional Encoder

2.2 Decoding

There are two famous decoding algorithms based on convolutional encoders trellis structure. One is Viterbi and the other one is BCJR. Viterbi introduced a decoding algorithm in the year 1967 for convolutional codes. Viterbi algorithm was a programming solution to the problem of finding the shortest path. Forney recognized that it was maximum likelihood ML decoding algorithm for convolutional codes. BCJR algorithm is a maximum aposteriori Probability (MAP) decoding method for convolutional code or block code with trellis structure and introduced in the year 1974. In ML decoding, the probability of codeword error is minimized. Viterbi decoding algorithm is simpler to implement, is usually preferred. In MAP decoding, the probability of information bit error is minimized and is used in iterative decoding application, such as turbo codes. In this project we focus on BCJR decoding algorithm[2][3].

2.2.1 BCJR algorithm

BCJR algorithm was named after Bahl, Cocke, Jelinek and Raviv who introduced a maximum aposteriori probability (MAP) decoding algorithm for convolutional code in 1974. This algorithm can be applied to any linear code, block or convolutional codes. Better performance is achieved with MAP decoding when the information bits are not equally likely. The computational complexity of this algorithm is higher than Viterbi algorithm. The BCJR algorithm for decoding uses trellis to a great extent. BCJR decoding passes the trellis twice i.e. from start to finish and back again. Because of this nature it is also called as forward-backward algorithm. BCJR not only provides the codeword, but also gives information about a posteriori log likelihood ratio. Hence, BCJR provides more information than Viterbi decoding[2][3]. When the turbo code in 1993 was introduced or came into existence, modified version of the BCJR algorithm was used by the inventors of BCJR. This is how the BCJR algorithm had reborn which was practically not used until then. By using iterative decoding apriori probabilities of the information bits change every iteration. So due to this, there is better performance in MAP decoder.

Let us consider convolutional encoder of rate $R = k/n$ to describe BCJR algorithm over AWGN channel. The information bit can take values 0 or 1 with a priori probability $P(u_i)$, from which we can define log-likelihood ratio(LLR). The decoder inputs are the received sequence $\mathbf{r}$ and the a priori L-values of the information bits. With the assumption of information bits not equally likely the algorithm calculates the aposteriori L-values.

$$L(u_i) = \log \left[\frac{p(u_i = 0|\mathbf{r})}{p(u_i = 1|\mathbf{r})} \right] \tag{2.1}$$

The numerator and denominator of the equation 2.1, contain aposteriori conditional probabilities, that is, probabilities computed after we know $\mathbf{r}$.

Decoder output obtained

$$\hat{u_i} = \begin{cases} 0, & \text{if } L(u_i) > 0 \\ 1, & \text{if } L(u_i) < 0 \end{cases} \tag{2.2}$$

where j=0,1 $\cdots$ $j-1$.

In case of coded sequence transmitted over AWGN channel we have

$$L_{ch}(u_i) = \frac{2}{\sigma^2}(r_i) \tag{2.3}$$

where $L_{ch}(u_i)$ is the channel L-value of coded bits.

To describe the BCJR algorithm let us consider the trellis (Figure 2.4) for the encoder shown in Figure 2.1 which has one memory unit with rate 1/2. It has 2-states $S = \{0, 1\}$. The dashed line is the branch produced by input one and the solid line show the branches of input zero. Each branch is labeled with an output or codeword that is generated by the encoder.

Now let us discuss the steps used in BCJR algorithm to calculate the soft output by an example. Consider the encoder in Figure 2.1 and the received vector $r = (-0.8, -0.1 - 1.0, +0.5 + 1.8, -1.11.6, +1.6)$. The steps of BCJR decoder are as follows.

1. Firstly, we obtain the trellis for the decoder indicating the number of states and number of time intervals involved. This trellis is shown in Figure (2.4)

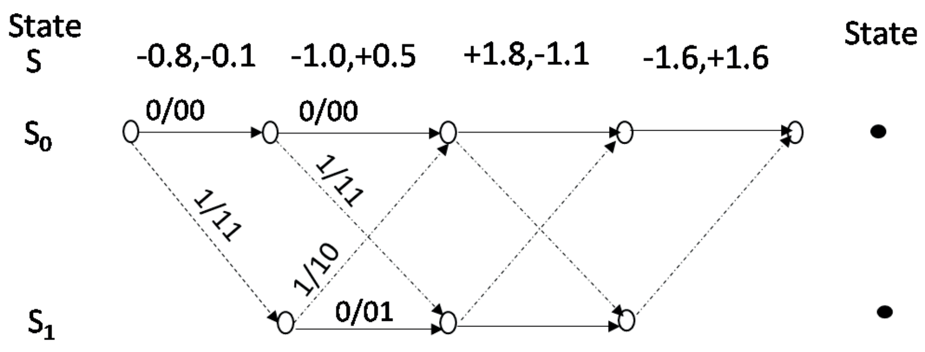


Figure 2.4: Rate 1/2 2-state trellis

2. We compute γ values or the branch metrics assuming that a priori probabilities of the information bits are equally likely, $L_a(u_t) = 0$, l=1,2 and we compute log-domain branch metrics. The branch metric from the previous state $s^{'}$ to the current state s is,

$$\gamma_t(s^{'}, s) = \sum_{i=1}^{K} L_a({u_t}^{(i)})(\frac{1}{2} - {u_t}^{(i)}) + \sum_{j=1}^{K} L_{ch}({v_t}^{(j)})(\frac{1}{2} - {v_t}^{(i)}) \tag{2.4}$$

In the above equation $L_a({u_t}^{(i)})$ is the a priori probabilities of the information bits. The required parameters to calculate the branch metric are the channel L-values $L_{ch}({v_t}^{(j)})$ and the codeword generated by encoder for every transition. Here the s denotes the current state $S_t = s$ for the time t, the previous state will be $S_{t-1} = s^{'}$ and next state is shown as $S_{t+1} = s^{''}$.
For the considered example, the branch matrices at $t = 0$ are as follows,

$$\gamma_0(s_0^{'}, s_0) = (-0.8)\frac{1}{2} + (-0.1)\frac{1}{2} = -0.45$$

$$\gamma_0(s_0^{'}, s_1) = (-0.8)\frac{-1}{2} + (-0.1)\frac{-1}{2} = +0.45$$

3. Carry out forward recursion

$$\alpha_t(s) = \max{}^*(\gamma_t(s^{'}, s) + \alpha_{t-1}(s)), \qquad \alpha_0(s) = \begin{cases} 0, & \text{if } s = 0 \\ -\infty, & \text{if } s \neq 0 \end{cases} \tag{2.5}$$

The next metric needed is α. To intialize, we consider that at time $t = 0$, the α for state zero is zero and the corresponding αs for the other states are $-\inf$. Then the sum of branch metric and α value of previous node is taken to update the current α node. When two branch meet at single node max* equation is used and the result is updated in the current α node. The max* equation is given by

$$\max{}^*(x, y) = \max(x, y) + \ln(1 + e^{(-|x-y|})} \tag{2.6}$$

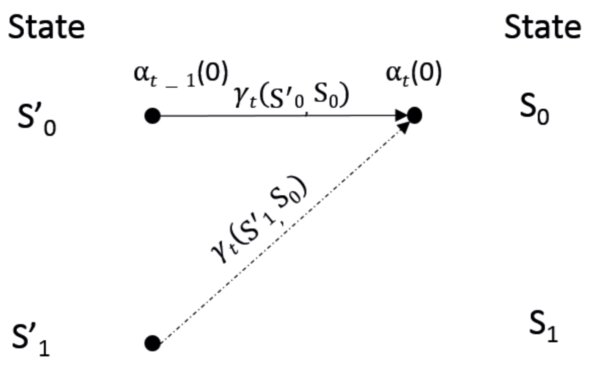


Figure 2.5: Forward recursion(alpha)

For example to calculate $\alpha_t(0)$ from the Figure 2.5 it can be observed that at time t two branch meet at current state S_0. One is from previous state zero $S_0^{'}$ and other from previous state one $S_1^{'}$. The computation can be done in the following way.

$$\alpha_t(0) = \max^*(\gamma_t(S_0^{'}, S_0) + \alpha_{t-1}(0), \gamma_t(S_1^{'}, S_0) + \alpha_{t-1}(1)) \tag{2.7}$$

For the considered example,

$$\alpha_0(s_0) = 0$$
$$\alpha_1(s_0) = \alpha_0(s_0) + \gamma_0(s_0, s_0) = -0.45$$
$$\alpha_1(s_1) = \alpha_0(s_0) + \gamma_0(s_0, s_1) = 0.45$$

$$\begin{aligned}\alpha_2(0) &= \max^*(\gamma_1(S_0^{'}, S_0) + \alpha_1(s_0), \gamma_1(S_1^{'}, S_0) + \alpha_1(s_1)) \\ &= \max(-0.7, +1.2) = 1.2 + ln(1 + e^{-1.9}) = 1.34\end{aligned}$$

4. Carry out the backward recursion

$$\beta_{t-1}(s') = \max^*(\gamma_t(s^{'}, s) + \beta_t(s)), \qquad \beta_{L+m}(s) = \begin{cases} 0, & \text{if } s = 0 \\ -\infty, & \text{if } s \neq 0 \end{cases} \tag{2.8}$$

The same procedure is followed as of α, but β values are calculated at each nodes considering the backward trellis. For example to calculate $\beta_{t-1}(0)$ from the Figure 2.6 we can see that one branch going from the state S_0 to $S_0^{'}$ and another from the state S_1 to $S_0^{'}$. The computation can be done in following way

$$\beta_{t-1}(0) = \max^*(\gamma_t(S_0^{'}, S_0) + \beta_t(0), \gamma_t(S_0^{'}, S_1) + \beta_t(1)) \tag{2.9}$$

For the considered example,

$$\beta_4(s_0) = 0$$
$$\beta_3(s_0) = \beta_4(s_0) + \gamma_3(s_0, s_0) = 0 + 0 = 0$$

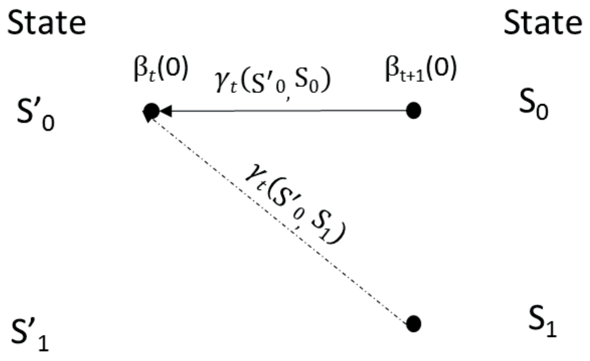


Figure 2.6: Backward recursion(beta)

$$
\begin{aligned}
\beta_3(s_1) &= \beta_4(s_0) + \gamma_3(s_1, s_0) = 1.6 + 0 = 1.6 \\
\beta_2(0) &= \max{}^*(\gamma_2(S_0^{'}, S_0) + \beta_3(s_0), \gamma_2(S_0^{'}, S_1) + \beta_3(s_1)) \\
&= \max(0.35, +1.25) = 1.59 \\
\beta_2(1) &= \max{}^*(\gamma_2(S_1^{'}, S_0) + \beta_3(s_0), \gamma_2(S_1^{'}, S_1) + \beta_3(s_1)) \\
&= \max(-1.45, 3.05) = 3.06
\end{aligned}
$$

Here, it can be seen that α and β are associated with the encoder states and γ is associated with the branches or transitions between states. The initial values of α and β mean the trellis is terminated, that is, begins and ends in the zero state. Therefore it will be necessary to add some tail bits(depends on memory unit) to the message in order that the trellis path is forced to return back to the initial state.

5.Computing APP L-values

$$
\begin{aligned}
L(u_t^{(i)}) = &\max{}^*_{(s',s):u_t^i=0}(\alpha_{t-1}(s^{'}) + \gamma_t(s^{'}, s) + \beta_t(s)) \\
&-\max{}^*_{(s',s):u_t^i=1}(\alpha_{t-1}(s^{'}) + \gamma_t(s^{'}, s) + \beta_t(s))
\end{aligned}
\tag{2.10}
$$

α and β values at each node and γ values at each branch of trellis enables the computation of soft output.
For example we can compute L_{u_0} from Figure 2.7 as

$$x(s_0^{'}s_1^{'}, input0) = \max{}^*(\alpha_{t-1}(0)+\gamma_t(S_0^{'}, S_0)+\beta_t(0), \alpha_{t-1}(1)+\gamma_t(S_1^{'}, S_1)+\beta_t(1)) = y_{input0}$$

$$x(s_0^{'}s_1^{'}, input1) = \max{}^*(\alpha_{t-1}(0)+\gamma_t(S_0^{'}, S_1)+\beta_t(1), \alpha_{t-1}(1)+\gamma_t(S_1^{'}, S_1)+\beta_t(0)) = y_{input1}$$

$$L_{u_0} = y_{input0} - y_{input1}$$

For the considered example,

$$L_{(u_0)} = (\alpha_0(0) + \gamma_0(s_0^{'}, s_0) + \beta_1(0)) - \alpha_0(0) + \gamma_0(s_0^{'}, s_1) + \beta_1(1))$$

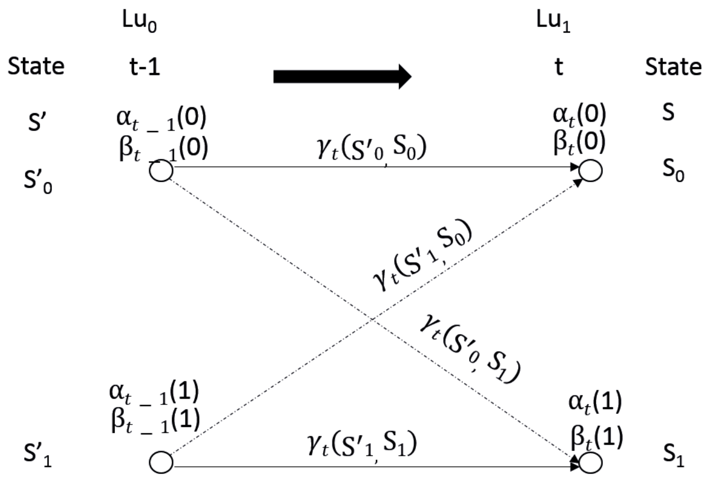


Figure 2.7: L-values

$$2.99 - 3.47 = -0.48$$

$$L_{(u_1)} = \max{}^*\{(\alpha_1(0) + \gamma_1(s_0^{'}, s_0) + \beta_2(0), (\alpha_1(1) + \gamma_1(s_1^{'}, s_0) + \beta_2(0))\}$$
$$-\max{}^*\{(\alpha_1(0) + \gamma_1(s_0^{'}, s_1) + \beta_2(1), (\alpha_1(1) + \gamma_1(s_1^{'}, s_1) + \beta_2(1))\}$$
$$= \max{}^*\{0.86, 2.76\} - \max{}^*\{2.79, 2.86\} = -0.62$$

In order to compute APP L-values the values of αs, βs and γs are required. Once these values are obtained, at every time instance 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 zero input branches subtracted from max* of the summation of $\alpha_{t-1}(s^{'})$, $\gamma_t(s^{'}, s)$ and $\beta_t(s)$ for all one input branches.

2.3 Turbo Codes

The original concept of turbo codes was first introduced in 1993 by Berrou, Glavieux and Thitimajshima [8]. From early days, the goal has always been to come close to the Shannon limit performance with a sustainable complexity. Turbo code with iterative decoding succeeds in achieving performance close to Shannon limit in terms of BER. Turbo codes are family of concatenated convolutional codes and are built using two or more simple constituent codes or component codes, arranged in variation of concatenation scheme along with message interleaving. Performance of a turbo code improves when the interleaver size is increased. Pseudo-random interleaver is an important part of the design and which has a positive influence on the code properties and the performance of turbo codes.
The basic turbo encoder is a parallel concatenation of two recursive systematic

convolutional (RSC) codes which are placed in parallel along with pseudo random interleaver. The turbo decoder uses soft-output values and iterative decoding that is the soft-values of one decoder are passed to the other and vice versa. In the iterative decoding,

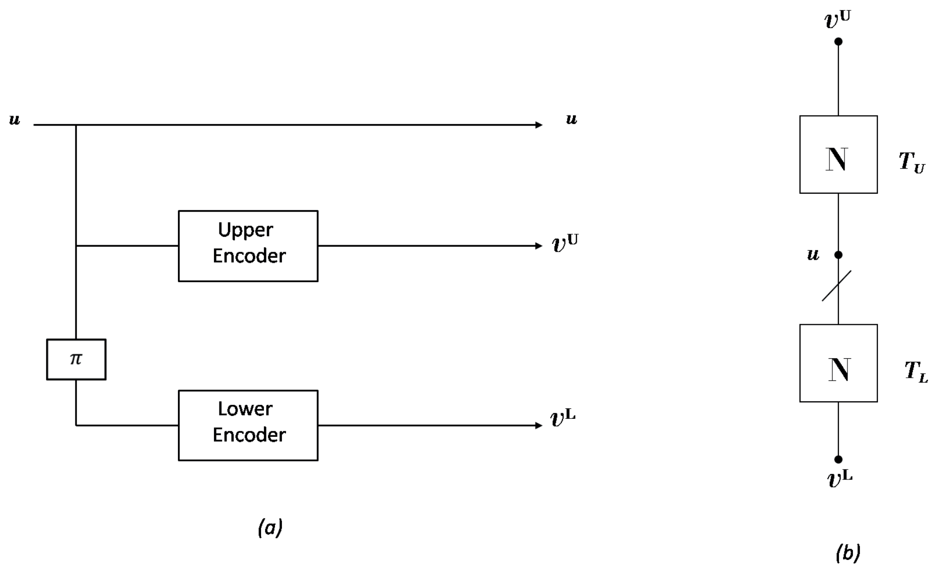


Figure 2.8: (a). Basic turbo encoding structure. (b) Compact graph

Figure 2.8(a) shows the basic turbo encoding structure also known as parallel concatenated code. It consist of information sequence or message bits $\mathbf{u}$, two systematic feedback recursive convolutional encoders placed in parallel and an interleaver denoted by π. The encoders take the name of upper and lower encoder respectively. The upper encoder gets the information sequence as the input. A reordered copy of information sequence is given to the lower encoder as an input.

The output of turbo encoders consist of systematic output $\mathbf{v}^{(1)} = \mathbf{u}$. The information $\mathbf{u}$ bits feed the upper encoder to create the upper parity sequence $\mathbf{v}^U$. Likewise, a reordered version of $\mathbf{u}$ is fed to the lower encoder to create the lower parity sequence $\mathbf{v}^L$. The termination bits of all ensures that the encoder will return back to zero state. This final sequence will be sent through the channel.

Figure 2.8(b) is the compact graph representation of turbo code. The information sequence $\mathbf{u}$, parity sequence of upper encoder $\mathbf{v}^U$ and parity sequence of lower encoder $\mathbf{v}^L$ are shown by a single black circle called variable node. The upper and lower trellis are denoted by T_U and T_L. The trellises are replaced by squares called factor node, which are labeled according to their length. The interleaver is shown in the graph by a line that crosses the edge which connects $\mathbf{u}$ to T_L. The sequence used in T_L is the reordered version of information sequence $\mathbf{u}$. Finally the transmitted sequence is $\mathbf{v} = \{\mathbf{u}, \mathbf{v}^U, \mathbf{v}^L\}$.

2.4 Iterative Decoding

Using log-likelihood algebra, it can be shown that any decoder can be used which accepts soft inputs including a priori values and delivers soft outputs that can be split into three terms : the soft channel and a priori inputs, and the extrinsic value. The extrinsic values are used in the iterations.
With $\mathbf{u}$ being in Galois Field(2) with the elements $\{0, 1\}$. The log-likelihood ratio of a binary random variable u_i, $L(u_i)$ given the received value $\mathbf{r}$ is defined as in equation (2.1)

The log-likelihood ratio $L(u_i)$ represents the soft value. $p_U(u)$ denotes the probability that the random variable u takes on the value u. The operator $\boxplus$ is used as the notation for addition

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

it can be demonstrated using some identities as

$$L(u_1) \boxplus L(u_2) = \log \frac{1 + e^{L(u_1)} e^{L(u_2)}}{e^{L(u_1)} + e^{L(u_1)}}$$

$$\text{sign}(L(u_1)).\text{sign}(L(u_2)).\min(|L(u_1)|, |L(u_2)|)$$

along with the additional rules
$L(u_1) \boxplus \infty = L(u)$ $L(u_1) \boxplus -\infty = -L(u)$ $L(u_1) \boxplus 0 = 0$
The reliability of the sum $\boxplus$ is therefore determined by the smallest reliability of the terms.
Now "soft values" of a channel are defined more clearly. After transmission over a fading channel or a Gaussian/binary symmetric channel (BSC) log-likelihood ratio of the $\mathbf{x}$ coded bit can be calculated conditioned on the matched filter output $\mathbf{y}$ as

$$L(x|y) = \log \frac{\exp(-\frac{E_s}{N_0}(y-a)^2)}{\exp(-\frac{E_s}{N_0}(y+a)^2)} + \log \frac{P(x=0)}{P(x=1)}$$

$$= L_{ch} \cdot y + L(x)$$

where L_{ch} channel reliability factor for the fading channel given by equation (2.3). $L_a(x)$ is the a priori L-values. At last soft output decoder computes the a posteriori L-values

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

$L_a(v_i)$ is the intrinsic part of the a priori L-value of v_i and $L_{ch}(v_i)$ is the received channel L-value corresponding to symbol v_i . $L_e(v_i)$ is the extrinsic part of the a posteriori L-values of v_i which does not depend on $L_a(v_i)$ or $L_{ch}(v_i)$. It is estimate of v_i based on other symbols.

In the first iteration as we can see in the Figure 2.9 for upper decoder the input are the channel L-values $\mathbf{L}_u$, $\mathbf{L}_v^U$ which are the L-values of systematic information

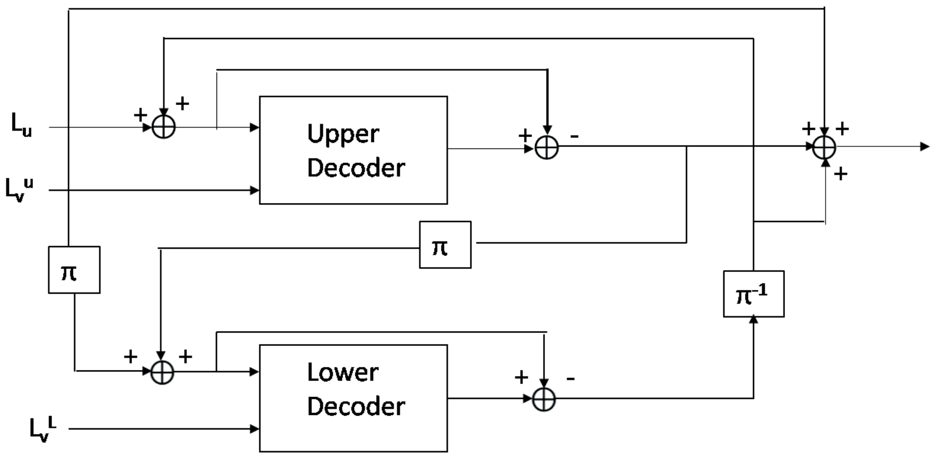


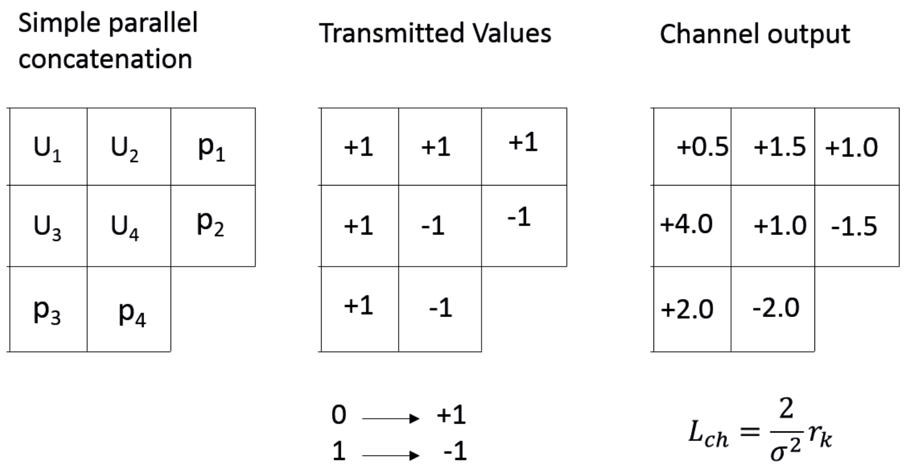
Figure 2.9: Basic turbo decoding structure

and parity sequence of the upper encoder. Since we still have no information coming from the lower decoder, the a priori information initially is set to zero. Message passes through the decoder and the extrinsic L-value of the output is permuted and fed to the lower decoder. In the first iteration of lower decoder, the input are the permuted version of the channel L-values $\mathbf{L}_u$, parity sequence $\mathbf{L}_v^L$. Then previously permuted extrinsic L-value coming from the upper decoder is used as a priori information for $\mathbf{u}$. At the output of the lower decoder the output is subtracted with the intrinsic information in order to get the extrinsic information. The extrinsic information is de-interleaved before it feeds the input to the upper encoder as apriori values. These extrinsic L-values, along with the L-values of the received information symbols, must be interleaved in the same pattern used at the encoder before entering lower decoder. First iteration is completed after both upper and lower decoder have been activated. Message passing is done to improve the performance all around. Convolutional code by itself is not enough to get with feasible complexity. If two are put together and then they are operating in this message passing turbo fashion, then they can help each other out and get to capacity.

For the following iterations, the a priori L-values are replaced by the extrinsic a posteriori L-values (L_{e2}) after being de-permuted. After each iteration, the extrinsic L-values, representing the reliability information about the bits to be decoded, are passed from one decoder to the other, ensuring that very little information is lost relative to optimal decoding.

Soft iterative decoding example

A simple example [4] of a parallel concatenated code using Log-Map algorithm to illustrate the principle of iterative decoding can be seen. In this example, vertical and horizontal single parity check codes are used. It consists of a block of



Simple parallel concatenation

U_1	U_2	p_1
U_3	U_4	p_2
p_3	p_4	

Transmitted Values

+1	+1	+1
+1	-1	-1
+1	-1	

Channel output

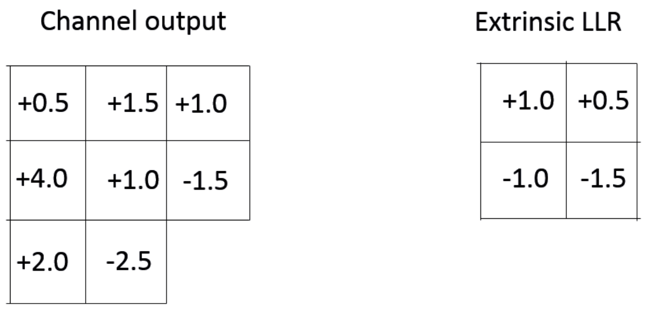
+0.5	+1.5	+1.0
+4.0	+1.0	-1.5
+2.0	-2.0	

$0 \longrightarrow +1$
$1 \longrightarrow -1$

$$L_{ch} = \frac{2}{\sigma^2} r_k$$

Figure 2.10: Iterative decoding example

input vectors $\mathbf{u} = [u_1, u_2, u_3, u_4]$ and the parity vector of the first constituent code $\mathbf{p}^1 = [p_1, p_2]$ and the parity vector of the second constituent code $\mathbf{p}^2 = [p_3, p_4]$. 8 transmitted bits are represented in a rectangular array as shown in Figure 2.9. The transmitted values after being modulated with output after the channel effect can be seen.
In the first iteration of decoder 1(horizontal decoding), the log MAP algorithm is applied to compute the a posteriori L-values for every input bits and the corresponding extrinsic L-values pass to decoder 2 (vertical decoding). The log-MAP algorithm uses these extrinsic a posteriori L-values received from decoder 1 as the a priori L-values to compute a posteriori L-values of input bits and the corresponding extrinsic L-values is fed back to decoder 1.



Channel output

+0.5	+1.5	+1.0
+4.0	+1.0	-1.5
+2.0	-2.5	

Extrinsic LLR

+1.0	+0.5
-1.0	-1.5

Figure 2.11: Horizontal Decoding

The horizontal decoding(decoder 1/upper decoder) is calculated with the help of the expression $L(u_i) \boxplus L(u_j)\text{sign}(L(u_1)) \cdot \text{sign}(L(u_j)) \cdot \min(|L(u_i)|, |L(u_j)|)$ between the information and the parity vectors which computes the aposteriori L-values for every input bit and the corresponding extrinsic aposteriori L-values.

The same expression as of the horizontal decoding is used in the calculation of

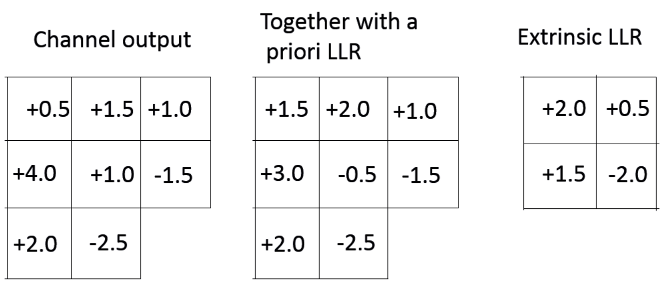


Figure 2.12: Vertical Decoding

the vertical decoding. Here the a priori L-values are considered for the calculation.

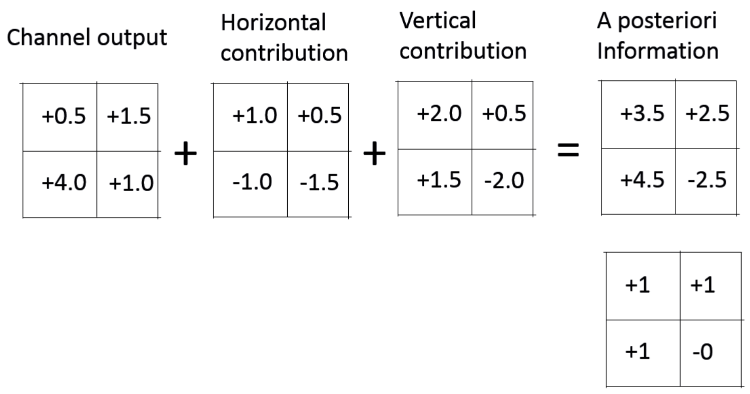


Figure 2.13: Decoding Result with a Single Iteration

Finally, by adding channel output with both decoders the aposteriori information can be calculated. In this example shown above the message is properly decoded after the first iteration.

Chapter 3

Spatially Coupled Turbo-like Codes

We apply spatial coupling technique in error control codes on pre-existing coding technique. Examples of such code could be block codes i.e. Low density parity check (LDPC) code, or on a convolutional code. In this chapter, we discuss spatial coupling of turbo-like codes. Turbo codes are part of the concatenated convolutional code family. In this thesis, our investigation is focused on serially and hybrid concatenated codes. Both codes are classified as a type of turbo-like codes. We apply the concatenation technique to form the code structure.

In error control coding, code concatenation is a technique used to form big codes from smaller codes. This was also discussed in the previous chapter. We can also see it as, multi-layer of encoding and decoding of the big codes which are from the smaller codes that are concatenated. With code concatenation we can reduce the bit error probability of such codes exponentially while only increasing the decoding complexity algebraically. Some of the various forms of concatenated codes are as follows;

- Parallel concatenated codes
- Serially concatenated codes
- Hybrid concatenated codes
- Braided concatenated codes

Parallel concatenated codes was already discussed in the previous chapter. There we described how it is made, it's properties and so on. In this chapter, we will discuss briefly braided concatenated codes in the spatial coupling section, this is because of it's code structure relating to how spatial coupling is being applied in other codes.

3.1 Serially Concatenated codes

Serially concatenated codes are concatenated code construction of component codes placed in cascade of each other. Or we could say, component codes are placed serially.

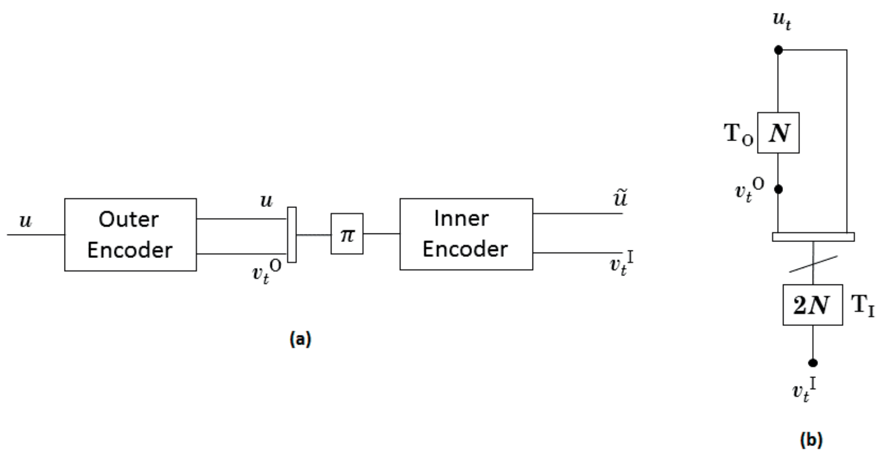


Figure 3.1: Serially Concatenated code

In Figure 3.1(a) is the encoder structure of a serially concatenated code. Here we can see two encoders placed in series of each other, the outer encoder receives the original sequence of information, encodes it, and then passes it on to the inner encoder. The block π symbol in the middle represents an interleaver. The interleaver permutes the bits received from the outer encoder before it is passed on to the inner encoder. By this, we can increase the Euclidean distance of the code sequence. Figure 3.1(b) is the compact graph of a serially concatenated code, in it, we can see how the code sequences are concatenated and permuted going starting from the un-coded sequence to the output sequence of the inner encoder and vice versa.

Serially concatenated codes can be applied in either block codes or convolutional codes, there can be more than two encoders in the code structure. It is important to point out that this technique has been in place since the 1960's when Forney investigated concatenated codes [11].

3.1.1 Serially Concatenated Codes with Iterative Decoding

In Chapter 2, we discussed iterative decoding in details. We will now apply this technique to serially concatenated codes. The decoding is based on the component codes, which exchange sequence apriori information, and by this technique, we can improve the performance of the code.

Figure 3.2 shows the iterative decoder for the serially concatenated code.

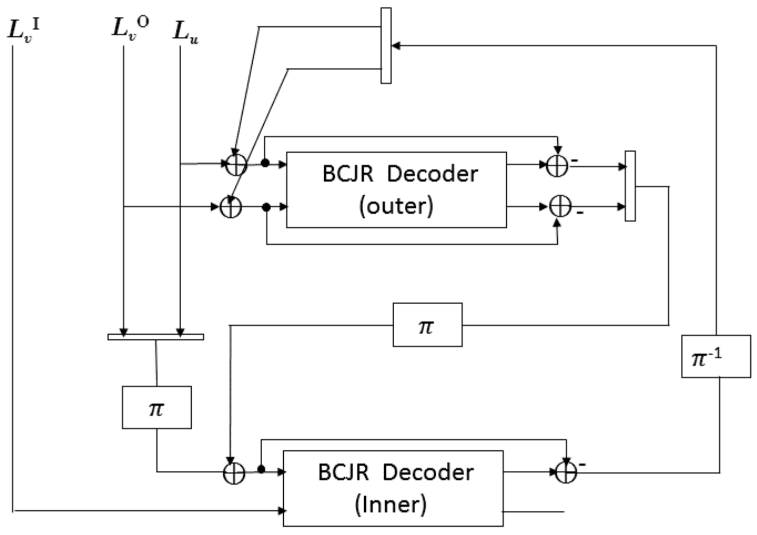


Figure 3.2: Serially concatenated code's iterative decoder structure

We can see that both the inner and outer decoders are connected and are exchanging information between them during iterations. For more clarity, see the Section 2.4 on "Iterative Decoding" in the previous chapter. From previous research, in terms of performance between a serially concatenated code and a parallel concatenated code, both codes have their advantages and disadvantages.

3.2 Hybrid concatenated codes

To say something is of a "hybrid" form literally means that it is a combination of two or more other known forms of that particular thing. The concept is the same in coding techniques. A hybrid concatenated code is a combination of two or more forms of concatenated coding schemes. We have discussed parallel concatenated code as well as serially concatenated code. Very often a hybrid concatenated code can be described as a combination of a serially concatenated code and a parallel concatenated code. A hybrid concatenated code with two or more interleavers is a code by which two or more codes are concatenated in parallel of each other to form the outer encoder and then, the outer encoder is concatenated in series with one or more codes to form the inner encoder. The input to the outer encoder is the un-permuted information sequence while the input to the inner encoder is the permuted concatenated output from the outer encoder.

We can analyze the hybrid concatenated code encoder structure in Figure 3.3 to that of the serially concatenated code found in Figure 3.1. Here, the outer

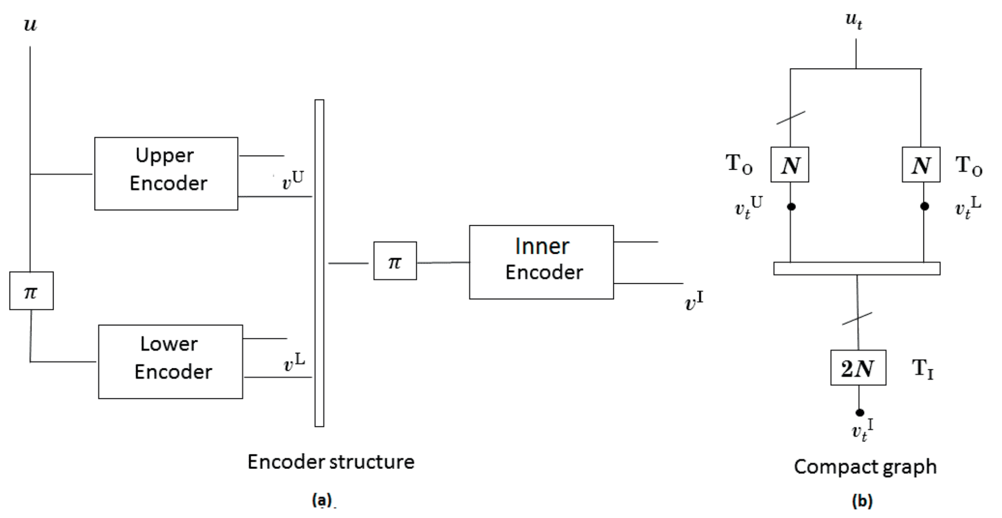


Figure 3.3: Hybrid concatenated code. Compact graph and encoder structure.

encoder has been replaced by two encoders placed in a parallel form, an upper encoder and a lower encoder. This is the exact form of the parallel concatenated code found in the previous chapter. Encoded sequences from the upper and lower encoders are then concatenated (or joined) and are then passed on to the inner encoder. An interleaver is placed just before the inner encoder to improve the linear distance properties of the code sequence.

3.2.1 Hybrid concatenated code with iterative decoding

An iterative decoder structure similar to that of serially concatenated code structure can be constructed. The decoder structure will have three component decoders because there are three component encoders during encoding. The decoder input sequences in the component decoders are connected according to their connections during encoding. This is visible from the compact graph seen in Figure 3.3(b). Also in Figure 3.3(b) we can see how the code sequences are concatenated and permuted going starting from the un-coded sequence to the output sequence of the inner encoder and vice versa.

Figure 3.4 shows the hybrid iterative decoder with all of its connections. Hybrid concatenated code construction brings us interesting prospects in terms of performance.

3.3 Spatial Coupling

Spatial coupling in coding is a technique used for achieving higher performance of a code. Let's call the total number of sequences that will be encoded the

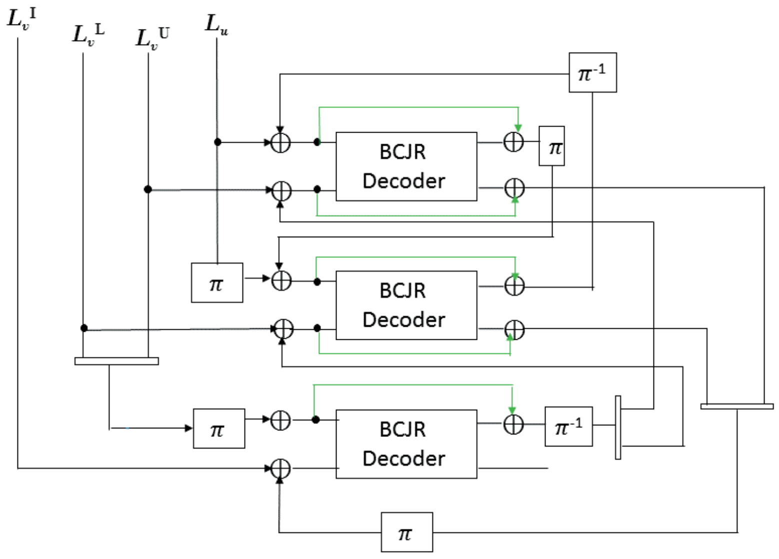


Figure 3.4: Iterative decoder structure of hybrid concatenated code

coupling length L. The dependence of each finite code sequence is determined by the coupling memory m. This is the number of neighbor encoders exchanging information at any particular time t instant. When larger code sequences is used with a larger coupling length we can achieve very good performance with spatially coupled codes, however, this can increase the complexity of the belief propagation (BP) decoding.

3.3.1 Braided convolutional codes

An uncoupled braided convolutional code is very similar to parallel convolutional code in terms of its structure. However, unlike in the latter, the parity sequence of the upper component encoder v^U is the input of the lower component encoder. The same is the case for the lower component encoder, its parity sequence v^L is used as the input for the upper component encoder.

Figure 3.5 shows the compact graph and the encoder structure of a braided convolutional code with two encoders. The main difference between uncoupled BCCs and spatially coupled BCCs (SC-BCC) is that in the latter, the parity sequences from both the upper and lower encoders at a time t, are used as input for their corresponding encoders and decoders in time $t + 1$. At time $t = 0$, the input from the previous time which are supposed to be the parity sequences from the encoders are set to zero. This we can see in Figure 3.6 below.

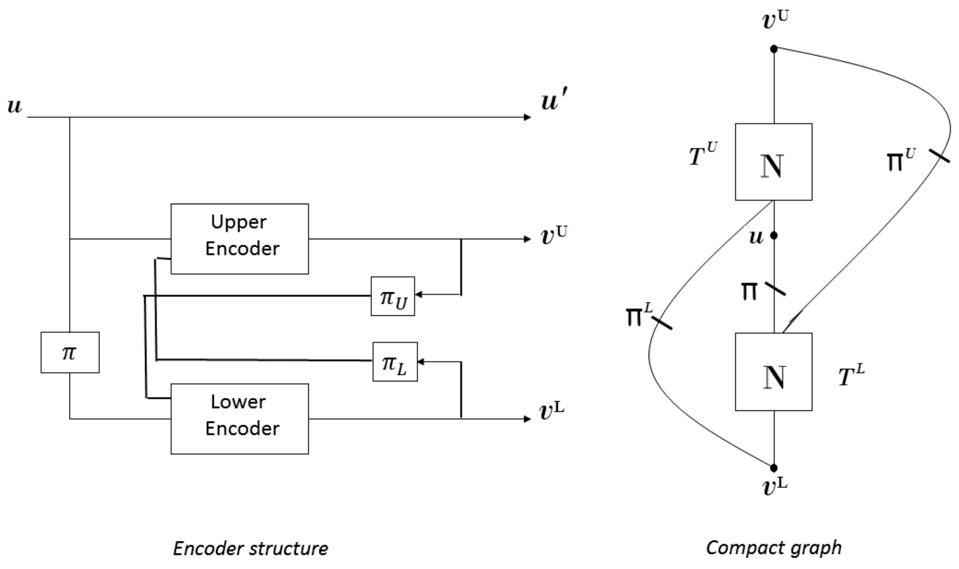


Figure 3.5: Braided Convolutional Code

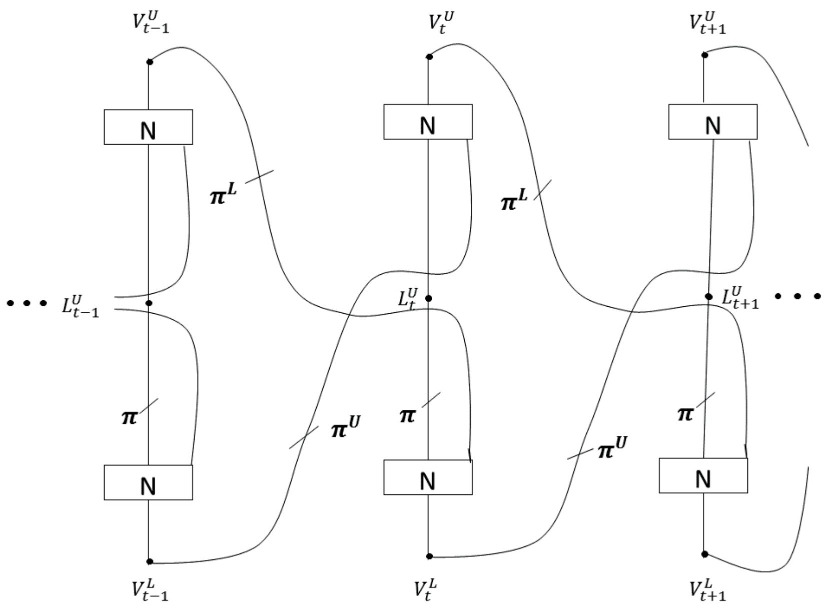


Figure 3.6: Braided Convolutional Code with Spatial Coupling

3.4 Serially Concatenated Codes with Spatial Coupling

In this section, we discuss spatially coupled serially concatenated codes.

Given block of length L serially concatenated codes with a 1/4 rate, which are to be transmitted in time for t=0,...,L, consider that the coupling memory $m = 1$, then each code sequence at time t, has part of its encoded sequences transmitted between time t, and $t + 1$. Likewise, at time $t - 1$ parts of it's sequences are transmitted at time t. If the coupling memory is greater than 1, we would have a sequence of code at a given time t transmitted over more than 1 time period, like in the former. This can be seen in Figure 3.7.

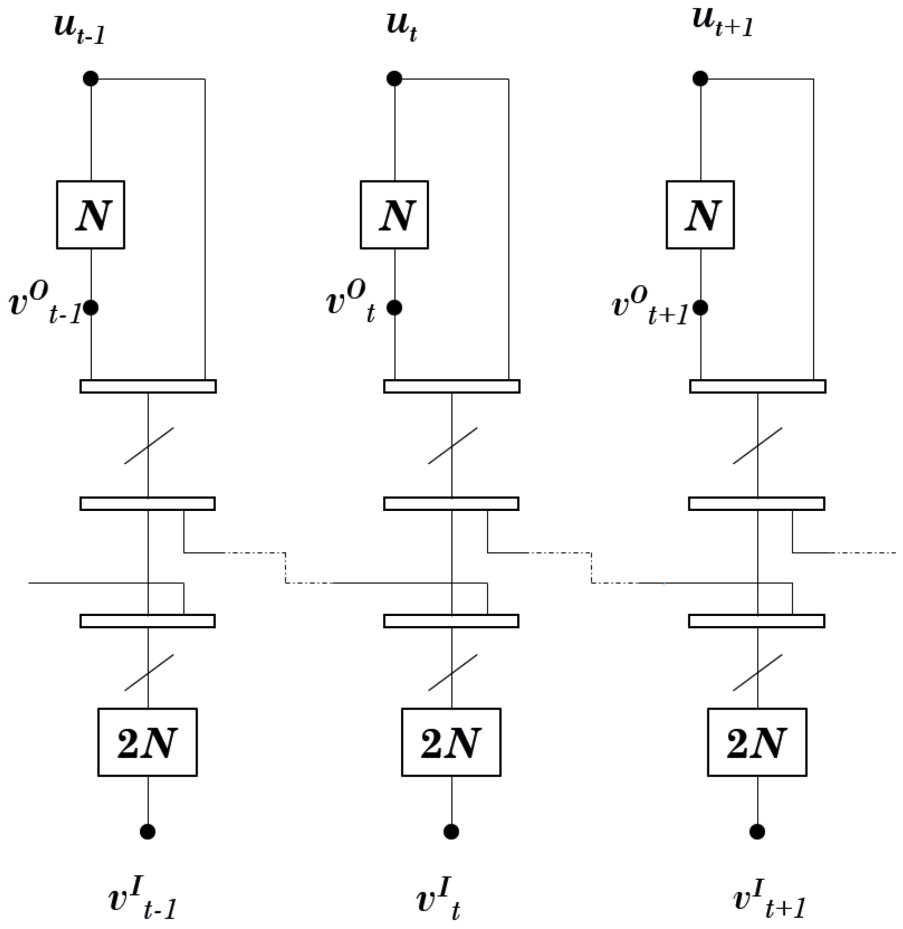


Figure 3.7: Compact graph of SP-SCC encoder

We can also see that in Figure 3.7, information sequence u_t is concatenated with the parity sequence of the upper encoder, and then permuted, then part of the sequences are sent to the next time slot, while it also receives part sequence from the time before. This forms a new sequence from both the current and previous time instance. The newly formed sequence is then permuted.

Figure 3.8 is the block diagram of spatially coupled serially concatenated code (SC-SCC). During the encoding, at time $t = 0$, sequences from the previous time are considered as null or zero. In the same way, we also want to ensure that our streams of sequences ends at $L_t = 0$. We then transmit an all zero sequence at time t=L. The termination stage is quite complex as it depends on the levels of coupling as well as the coupling memory. We must take this into consideration when terminating the chain. Improper termination of sequences can lead to certain error pattern at the end of the chain.

In a similar process we can also implement a SC-SCC decoder. Considering that channel values belong not to one time slot alone, and having information about the encoding process, decoded sequences at each time slot is also connected to other time slots accordingly. In Figure 3.9, we can see that extrinsic sequences in the outer decoder as well as the inner decoder are coupled with sequences at the previous and next time sequences respectively. Likewise the intrinsic channel values of the outer decoder. The decoder diagram might not be so clear at this point, because it is an iterative decoder, however this implementation is used with a window decoding technique which we will talk about at a later section in this chapter.

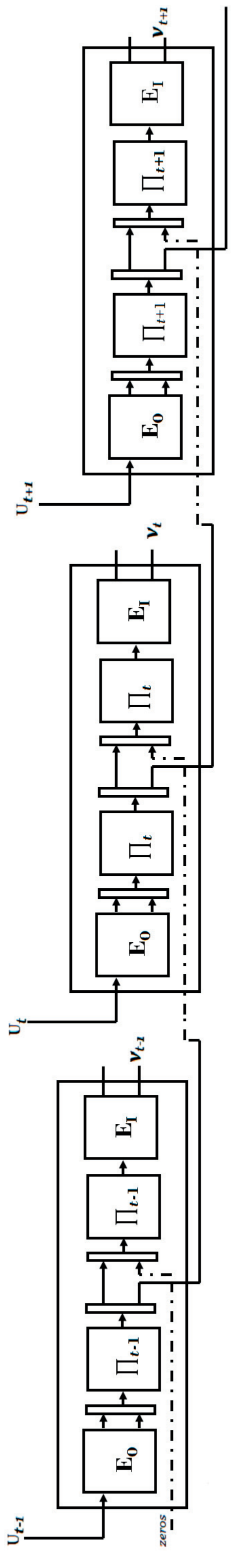


Figure 3.8: SC-SCC encoder block diagram

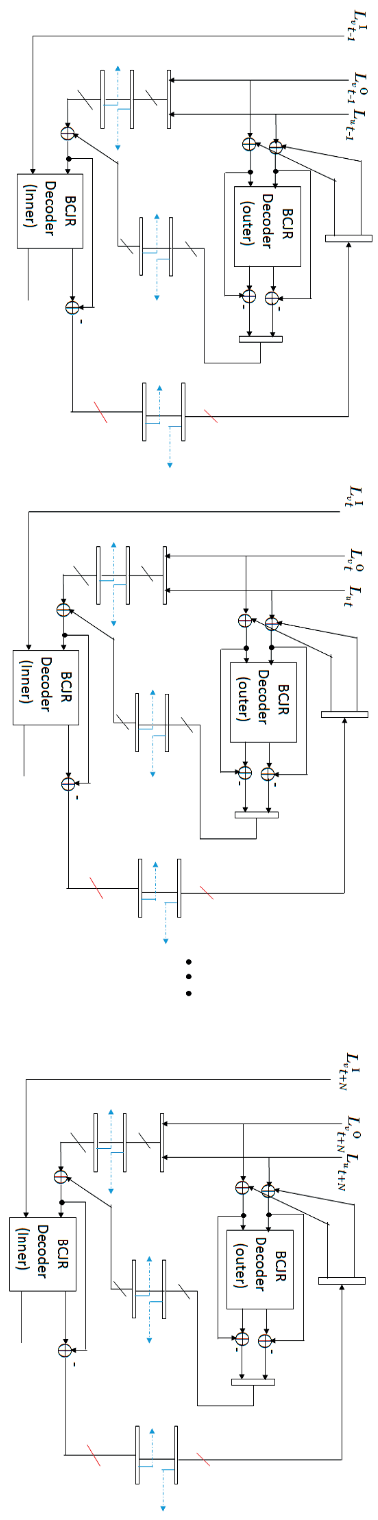


Figure 3.9: Spatially coupled SCC's Iterative Decoder

3.5 Hybrid Concatenated Code with Spatial Coupling

The implementation of spatially coupled hybrid conatenated codes (SC-HCC) are more complex than that of the SC-SCC. In our experiment, in addition to coupling the parity sequences, we also coupled the input sequence.

We encode a stream of length L SC-HCC with rate 1/5, which we will transmit over the time period for t=0,...,L. Consider that the coupling memory $m = 1$, then each code sequence at time t, has part of its encoded sequences as well as the original bit sequences transmitted between time t, and $t+1$. Likewise at time $t-1$ parts of it's sequences are transmitted at time t. If the coupling memory is greater than 1, we would have a sequence of code at a give time t transmitted over more than 1 time period, same like in the SC-SCC case.

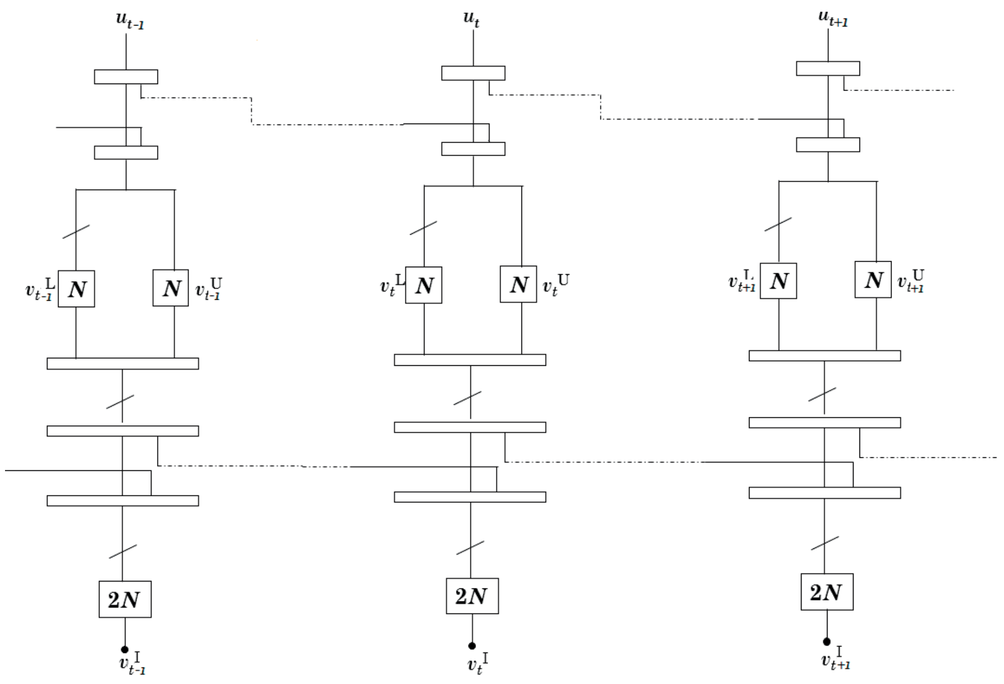


Figure 3.10: Compact graph of spatially coupled HCC encoder

Figure 3.10 shows the compact graph of the SC-HCC. Note how we both applied coupling both before and after encoding the outer sequences. Both the upper and lower encoders gets their input from the input sequence u, the input sequence of the lower encoder is permuted while that of the upper encoder isn't.

Figure 3.11 is the block diagram of the SC-HCC encoder. We set sequences from time $t < 0$ to null during encoding. We also should terminate the streams of sequences with nulls. Unlike in the SC-SCC case, we have to terminate with at least two streams of nulls. We then transmit an all zero sequence at the $t = L-1$, and $t = L$. This is because of the multiple levels of coupling.

In a similar way we can also implement a SC-HCC decoder. Considering that channel values belong not to one time slot alone, and having information about the encoding process, decoded sequences at each time slots are also connected to other time slots accordingly. In Figure 3.12, we can see that extrinsic sequences from the upper decoder as well as the lower decoder are concatenated and are then permuted, and then spatially coupled with the respective dependent time slots and blocks. In the same way, the extrinsic sequence from the inner decoder is de-permuted, and then spatial coupling is applied and the resulting sequence are fed into the connected input of the upper and lower decoders. Likewise the intrinsic channel values of the upper and lower decoder. This is also an iterative decoder implemented with SC-HCC. More details about the decoding process we will discuss in the Window Decoding section of this chapter.

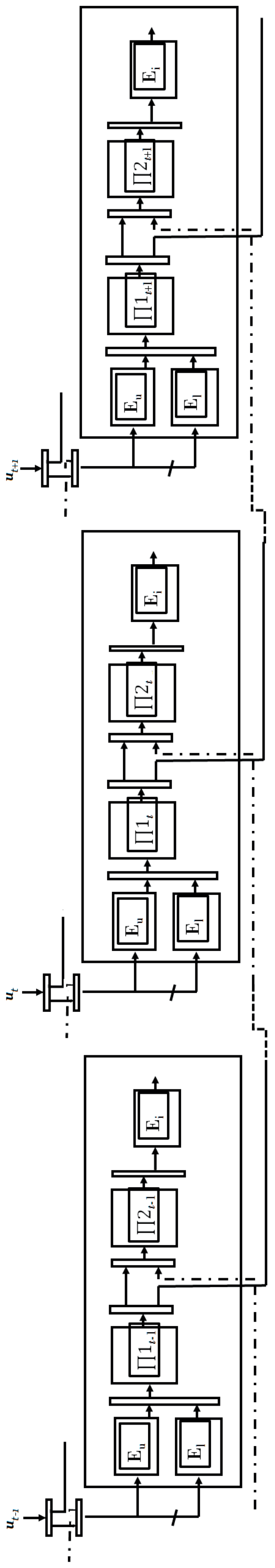


Figure 3.11: SC-HCC Encoder

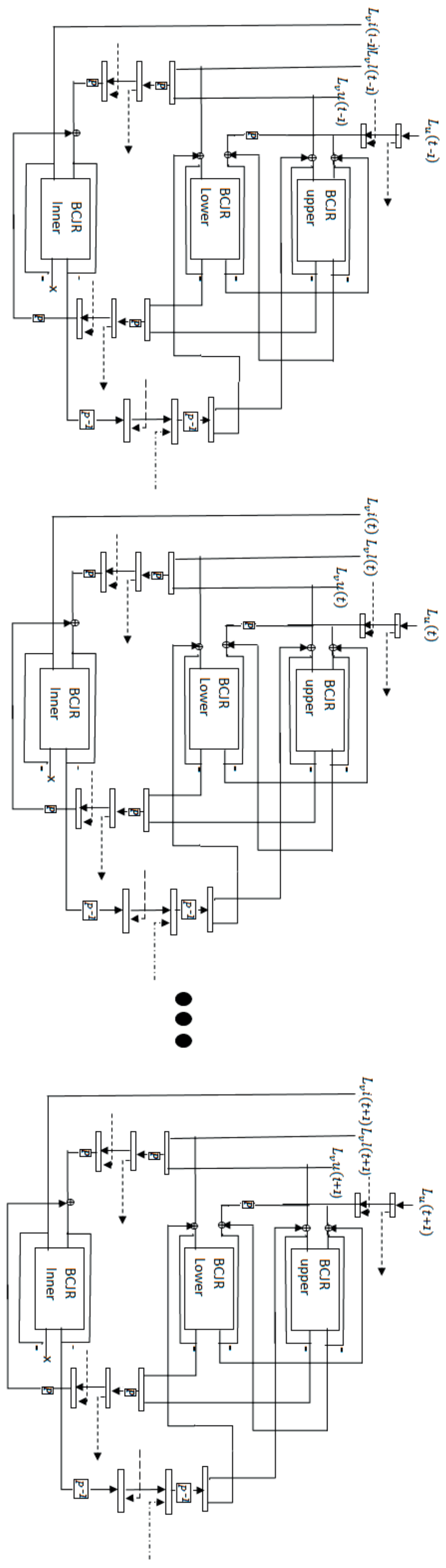


Figure 3.12: Spatially coupled HCC's Iterative Decoder

3.6 Window Decoding

In uncoupled version of both SCC and HCC, we used only iterative decoding technique. However, during spatial coupling, iterative decoding alone becomes complex to implement since sequences are coupled. If we decode our sequences one after the other, it is indifferent to the uncoupled version of our code and decoding all length L streams of sequences at the same time is too complex. Window decoding as a technique for decoding spatially coupled turbo-like codes, it is a technique used to combat the complexity of belief propagation between coupled sequences. Sequences are decoded in steps and withing a windowed time frame. This decomposes the belief propagation scheme into sub-optimal decoding steps as well as maintaining the code performance [12].

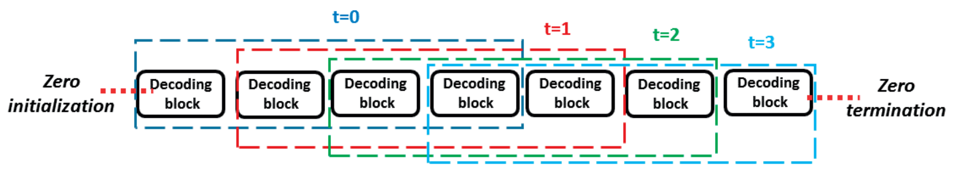


Figure 3.13: Window Decoder

A simple illustration of how a window decoder operates can be seen in Figure 3.13

It comprises of a sliding window with size $w = 4$, and moves a step at a time to the right till all streams of sequences have been decoded. To eradicate the problem of having errors at the beginning and the end of the coupling chain, each sequence must be decoded equal number of times. At every position of the window, the decoder iterates for certain number of times. I.e, iteration size $I = 5$, the window is then shifted “one” position to the right, and performs another I number of iterations until all sequences have been decoded. An increased number of iterations per window, the more accurate the decoding gets. The coupling chain must then be padded with $w-1$ sequences of zeros at the beginning of the chain as well as $(w-1)$ sequences of zeros at the end of the chain. By doing this, we ensure that the window starts with decoding the first sequence in the chain only, and slides through to the last sequence in the chain. The size of the window influences the code performance. The bigger the window, the better the performance, so also the decoding complexity. An active research is ongoing regarding the optimal threshold of the window size and how it influences the code performance.

Chapter 4

Results

Main focus of this chapter is to discuss our observation from our investigation in this project. The bit error rates for different ensembles are presented as a function of the signal to noise ratio. This chapter we present simulated results from following ensembles.

1. Uncoupled hybrid concatenated codes
2. Coupled hybrid concatenated codes
3. Uncoupled serially concatenated codes
4. Coupled serially concatenated codes

We also compare our results with results from thesis work carried out in reference [9] by Hector Moreno and Ardiana Osmani, *"Analysis of the finite length performance of spatially coupled convolutional codes"*. In their thesis, they investigated uncoupled and coupled braided convolutional code and parallel convolutional code. In our simulations we implemented BCJR algorithm in C++ and also implemented the main simulation functions in Matlab. We have also written a MEX function which acted as an interface between Matlab and C++. Our implementation of BCJR algorithm in C++ was due to the advantage C++ has over Matlab on time spent on loops. An additive white Gaussian noise (AWGN) channel is used to evaluate the performance. We used the Aurora Lunarc cluster facilities based in Lund University. Aurora consists out of 180 compute nodes for SNIC use and over 50 compute nodes funded by research groups at Lund University. Each node has two Intel Xeon E5-2650 v3 processors (Haswell), offering 20 compute cores per node. The nodes have 64 GB of DDR4 ram installed [13]

4.1 SCC (uncoupled vs coupled)

Rate 1/4

Under this topic simulated results of serially concatenated code and spatially coupled serially concatenated code of rate $R = 1/4$ is compared for input block length $N = 1000$ and $N = 8000$. For uncoupled case the ensemble of SCC can be seen in Chapter 3, Figure 3.1. Here two identical rate $R = 1/2$, 4-state component encoders are used with generator matrix $[G] = (1 \quad \frac{1+D^2}{1+D+D^2})$. BCJR algorithm with iterative decoding $I = 100$ iterations are considered. In the coupled case, we used the same component encoders as in the uncoupled case, and coupling length $L = 100$ and memory $m = 1$ is considered (see Figure 3.9). A sliding window algorithm with iterative decoding is used during decoding. To ensure that we have approximately 100 iterations per time slot and iterations per time slot is dependent of window size w, we set the number of iterations to equal $100/w$. Different window size is used for different input block length. By this we can see how much difference it brings out in the performance of the BER curves.

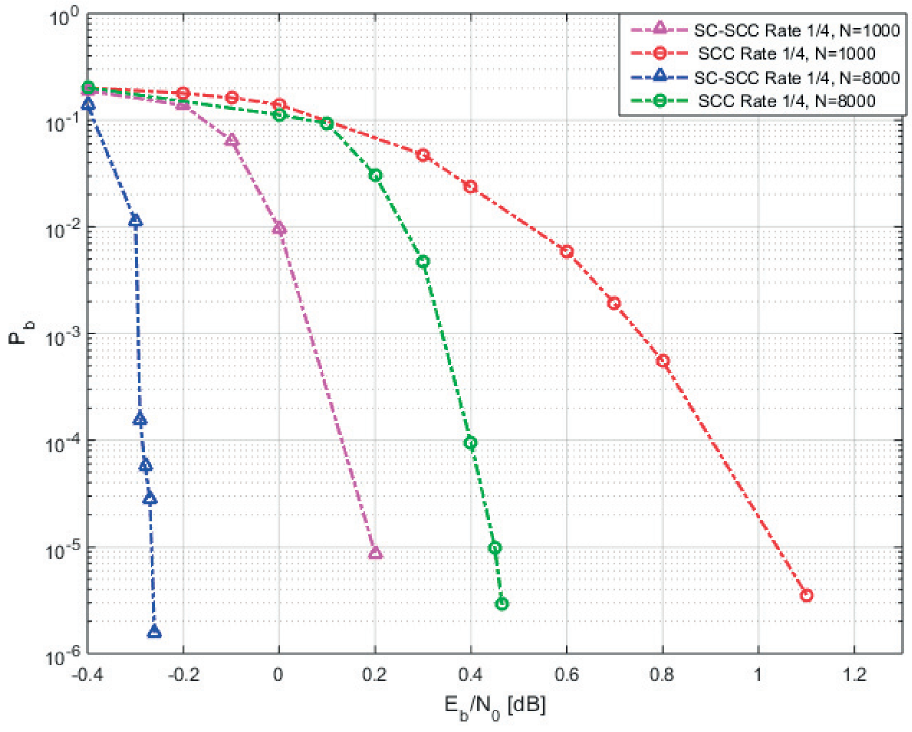


Figure 4.1: SCC coupled and uncoupled rate $R = 1/4$

In Figure 4.1 we can see the simulated results for both uncoupled and coupled case for Input block length $N = 1000$ and $N = 8000$. For the coupled case we have used window size of $w = 10$ and $w = 5$ for Input block length $N = 1000$ and $N = 8000$ respectively. Here we can see the effect of increasing the Input block length from $N = 1000$ to $N = 8000$, the waterfall region improves and with a very steep slope it gets closer to the capacity. For Input block length of $N = 1000$, SCC and SC-SCC $w = 10$ at BER=10^{-5}, we can see SC-SCC has a better performance than SCC with gain of around 0.8dB. For Input block length of $N = 8000$, SCC and SC-SCC $w = 5$ at BER=10^{-5}, there is gain of around 0.7dB. In either coupled case or uncoupled case we don't see any error floor because error floor is too low for serially concatenated code due to its good distance property.

Puncturing

We puncture a low rate code to a slightly higher rate code by removing some of the parity bits after the bits or sequences has been encoded. Original SCC code rate $R = 1/4$ and for HCC $R = 1/5$. To achieve rate $R = 1/3$ here puncturing is carried out on both SCC and HCC in order to compare the simulated results with results of PCC and BCC as both ensembles were simulated with rate $R = 1/3$. By puncturing several rates can be obtained from the same mother code making it possible to create a universal encoder/decoder. For example originally, for HCC, the code rate $R = 1/5$. That is to say, for every input sequence, we get five times that sequence as output. For example, in our code, we simulated for input block length $N = 1000$, after encoding we will get bit sequence of $N = 5000$ bits as output. To get to rate $R = 1/3$, we need to puncture away 2000 bits. We punctured HCC in two forms. For form-I, we punctured all bits of the outer encoder sequences, while for form-II, we punctured the even bits of the parity sequences. In HCC, we noticed a significantly different characteristics of the code performance when we used these different forms of puncturing, this will be discussed later in this chapter and in the conclusion chapter.

Serially Concatenated Code with rate R=1/3

Under this topic we compare the performance of SCC and SC-SCC of rate $R = 1/3$. The ensemble, generator matrix and block length used are same as rate $R = 1/4$. To achieve rate $R = 1/3$, the parity bits of the outer encoder are punctured.

In the Figure 4.2, we can see the performance of SCC ensemble with input block length $N = 1000$ and $N = 8000$. And also the performance of spatially coupled case of similar input block length can be seen. In the coupled case the window size used are $w = 10$ for input block length $N = 1000$ with iteration $I = 10$ and $w = 5$ for input block length $N = 8000$ with iteration $I = 20$. For the input block length $N = 1000$ the waterfall region gets better with gain of almost 1.25dB between SCC and SC-SCC. Similarly for input block length $N = 8000$ the waterfall region of the SC-SCC gets better with steeper slope closer to capacity compared to SCC case with gain upto 1.2dB.

4.2 HCC (Uncoupled vs Coupled) Puncturing form-I and Puncturing form-II

In this section, we discuss results from our simulations comparing uncoupled and coupled hybrid concatenated codes of input block length $N = 8000$ and $N = 1000$ for the form-I and form-II. In all the simulations, we used a rate $R = 1/2$ with a 2-state component encoder in the outer encoder while in the inner encoder, we

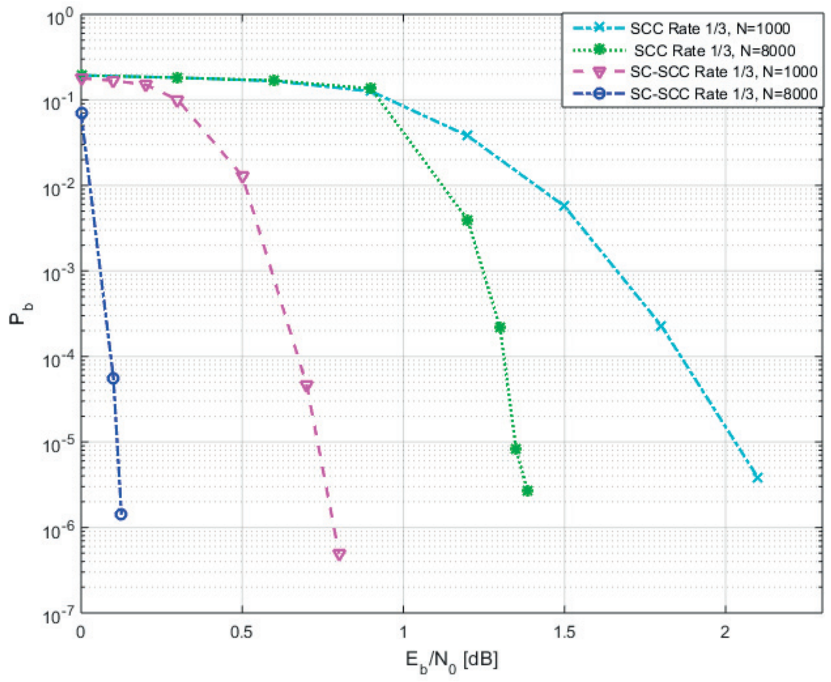


Figure 4.2: SCC uncoupled and coupled results

used a rate $R = 1/2$ with a 4-state encoder. We set the number of iterations during decoding to $I = 100$ for uncoupled simulations. To ensure 100 iterations per time slot in the coupled simulation as in the uncoupled, we set the number of iteration per time slot to $(100/w)$. For coupled simulation, we set the window size of $w = 10$ for input block length of $N = 1000$, and window size $w = 5$ for input block length of $N = 8000$. To ensure a rate of $R = 1/3$, we puncture every second bit of the outer and inner encoded sequences in all simulations for form-II while for form-I we punctured all the bits of the outer encoded sequences.

Figure 4.3 shows the performance of HCC and SC-HCC form-I ensembles of rate $R = 1/3$ with input block length of $N = 1000$ and $N = 8000$. As expected, the uncoupled results were worse off overall with input block length $N = 1000$ reaching what is like an error floor at 2.9 dB SNR. While that of block length $N = 8000$ uncoupled was better, we still can see it approached an error floor at an SNR of 2dB. Also, we see that input block length of $N = 8000$ is better than input block length of $N = 1000$ for the coupled results.

Figure 4.4 shows the performance of HCC ensembles of rate $R = 1/3$ with input block length of $N = 1000$ and $N = 8000$, coupled and uncoupled for form-II puncturing. With input block length of $N = 1000$ for SC-HCC, we can see that we have approximately 0.8dB gain when comparing to HCC. However, this

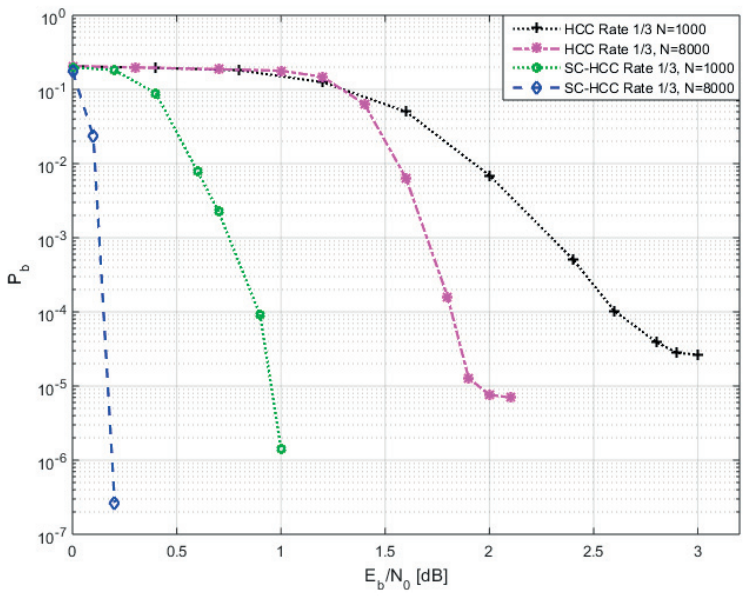


Figure 4.3: HCC form-I uncoupled and coupled results

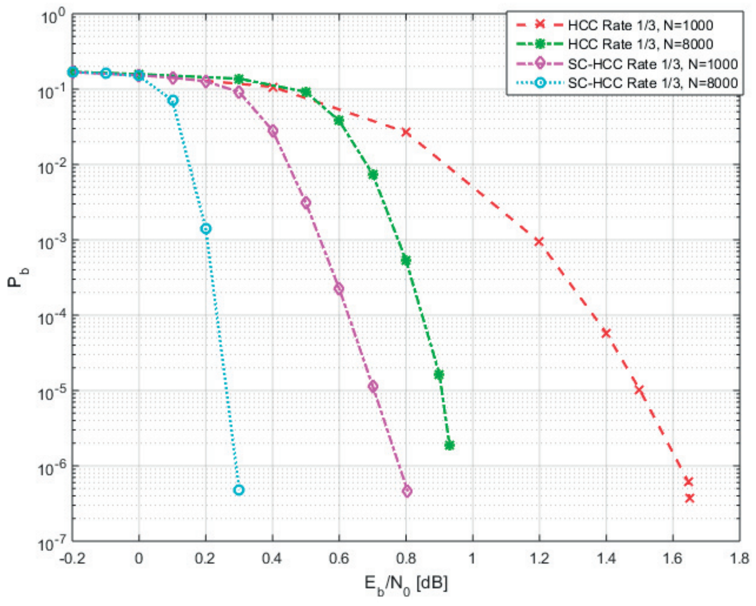


Figure 4.4: HCC form-II uncoupled and coupled results

gain is smaller when we compare that of input block length of $N = 8000$ which is approximately 0.6dB. In both uncoupled and coupled cases of the form-I and form-II, input block length of $N = 8000$ performs better than input block lengths of $N = 1000$ for their respective configurations.

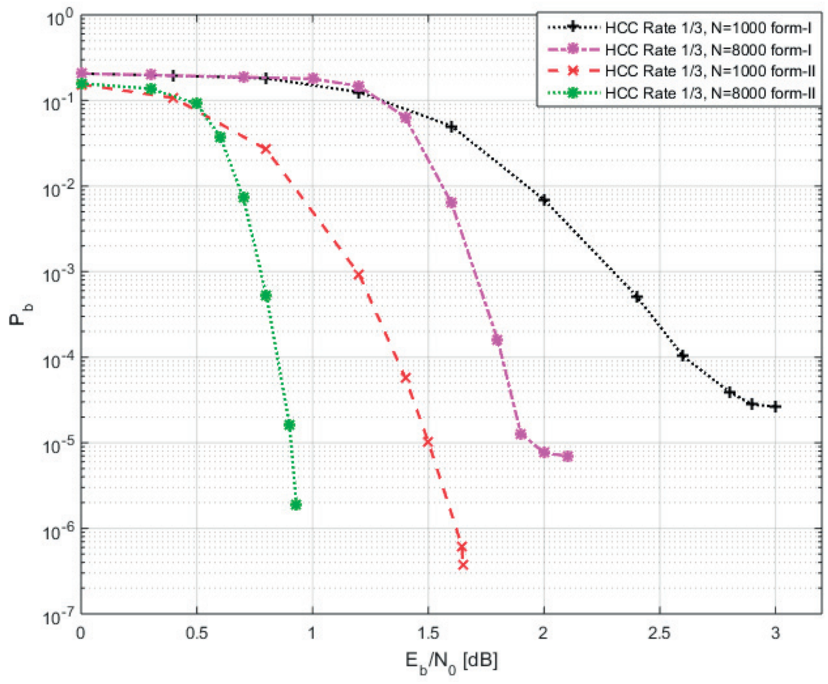


Figure 4.5: HCC form-I vs form-II uncoupled results

Figures 4.5 and 4.6 shows uncoupled simulations for form-I and form-II puncturing, and coupled simulations for form-I and form-II puncturing respectively. While form-II performed better for uncoupled with input block length of $N = 1000$ and $N = 8000$ and also for coupled $N = 1000$, it was form-I puncturing which is better for coupled $N = 8000$ with it's water fall region having approximately 0.1dB gain as over form-II puncturing.

4.3 HCC uncoupled vs SCC uncoupled

In this section, we compare uncoupled HCC and SCC results presented in Sections 4.1 and 4.2.

In the Figure 4.7, the performance of HCC compared to SCC with input block length $N = 1000$ is good and has a gain of almost 0.5dB. When we increase the

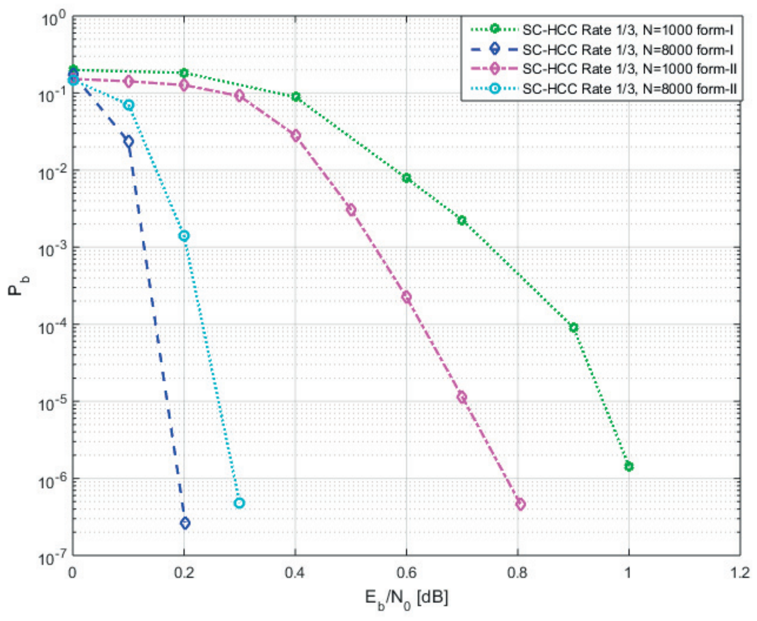


Figure 4.6: SC-HCC form-I vs form-II results

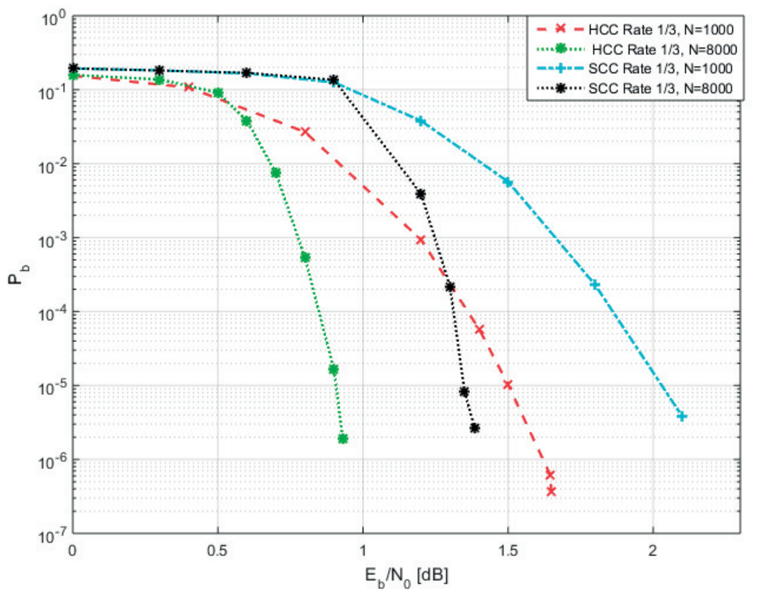


Figure 4.7: HCC form-II vs SCC results

input block length from $N = 1000$ to $N = 8000$ we can see a steeper slope with better performance for both HCC and SCC. At low E_b/N_0 HCC has better performance than SCC.This shows that increase in the permutation size also increases the performance.

4.4 Comparison between uncoupled parallel, serial, braided and hybrid concatenated code

In the research work carried out previously at the department by Hector Moreno and Ardiana Osmani, *"Analysis of the finite length performance of spatially coupled convolutional codes"*, they investigated braided and parallel concatenated convolutional codes. In this section, we compare, braided and parallel convolutional codes[7] with hybrid and serially convolutional codes which we investigated in this thesis.

The ensemble presented in Chapter 2, Sections 2.3 and 2.4 correspond to PCC as seen in Figure 2.8. For PCC two identical $R = 1/2$, 8-states component encoders with generator matrix $G = (1 \quad \frac{1+D^2+D^3}{1+D^2+D^3})$ are used. The ensemble presented in Chapter 3, Section 3.3.1 correspond to BCC. For BCC, two identical $R = 2/3$, 4-states, component encoders with generator matrix as shown in Example 2.3 in [7] are used. Both PCC and BCC are simulated for input block length $N = 1000$ and $N = 8000$. Iterative decoding with the BCJR algorithm with $I = 100$ iterations were used. Result plots for SC-PCC, PCC, SC-BCC and BCC are taking from reference [9], research by Hector Moreno and Ardiana Osmani, *"Analysis of the finite length performance of spatially coupled convolutional codes"*.

In Figure 4.8 both for PCC and HCC form-I we can observe some error floor, only we can see that PCC has a steeper slope in the waterfall region, and was initially better than the other ensembles. HCC form-II was the second best of all the ensembles as in this comparison, in terms of the waterfall region. We do not observe any error floor and it also crosses the error floor of PCC. We can see, from this observation, HCC form-II will perform better than PCC at some point.

In Figure 4.8 and Figure 4.9, we could see PCC had a good waterfall region but we could also observe the error floor. Notice HCC form-II is the second best performing ensemble in waterfall region when comparing with other ensembles for both cases of input block length $N = 1000$ and $N = 8000$. HCC form-I still performs worse when comparing the waterfall region with the other ensembles. However, we can observe a steeper waterfall region for input block length $N = 8000$.

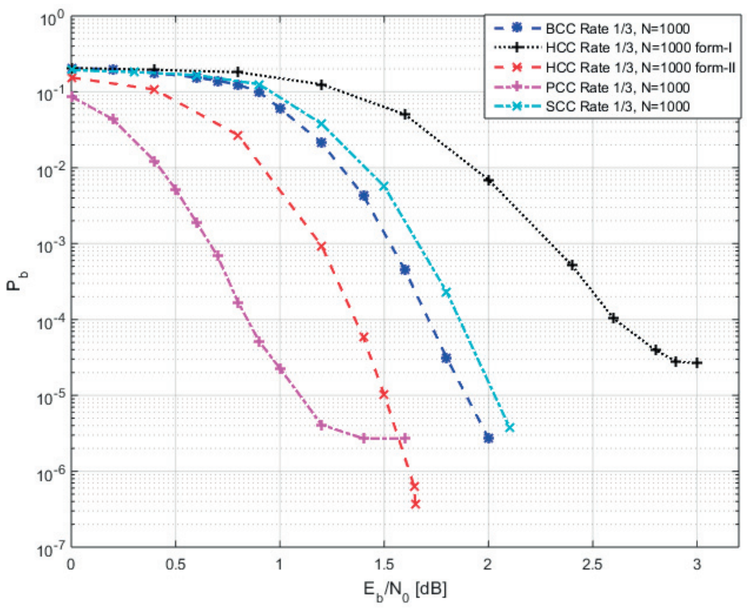


Figure 4.8: HCC(form-I and form-II), BCC, SCC and PCC results for input blcok length N=1000

4.5 Comparison between coupled parallel, serial, braided and hybrid concatenated code

In this section we compare performance of spatially coupled parallel, serially, hybrid and braided concatenated convolutional codes. A coupling length of $L = 100$ is considered. Sliding window with different decoding iterations are used. Sliding window of $w = 5, 6, 10$ with iteration $I = 10$ each were used but for our comparison we take the simulated results of window size $w = 10$ for input block length $N = 1000$ and window size $w = 5$ for input block length $N = 8000$.

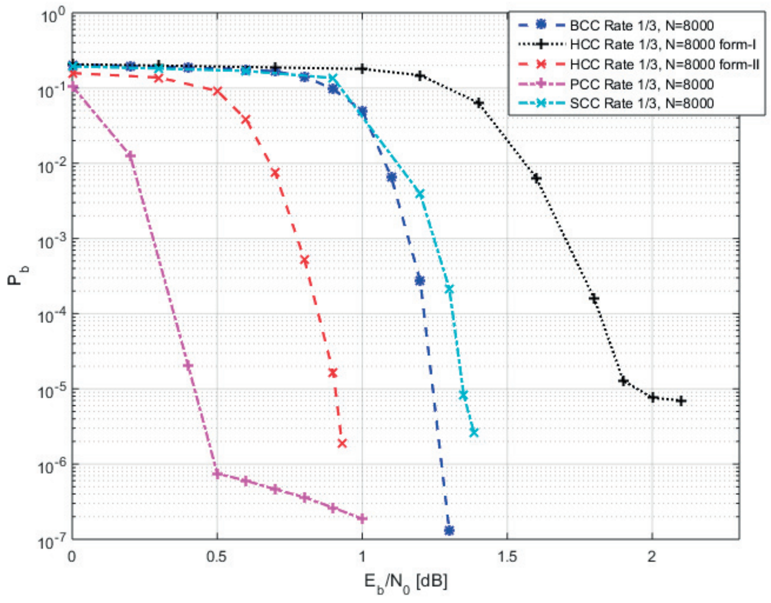


Figure 4.9: HCC(form-I and form-II), BCC, SCC and PCC results for input block length N=8000

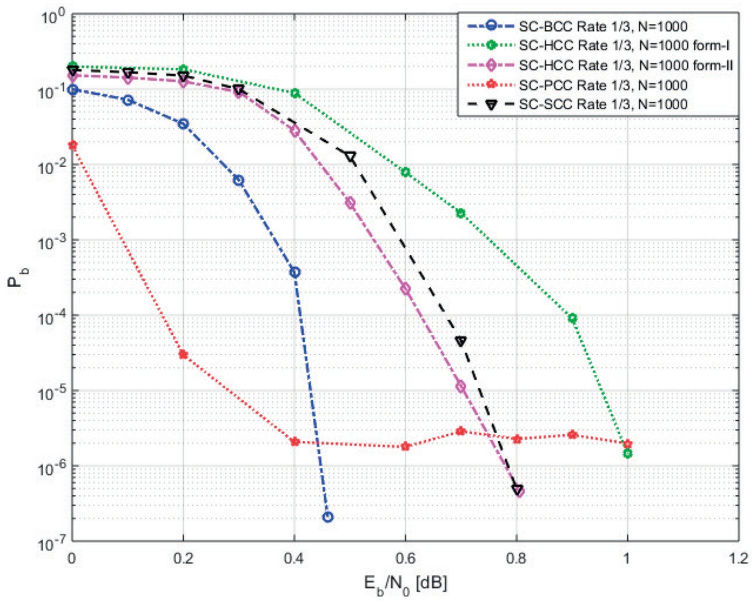


Figure 4.10: SC-HCC(form-I and form-II), SC-BCC, SC-SCC and SC-PCC results for input block length N=1000

In Figure 4.10, we observe that for SC-PCC, while we initially had a steep waterfall region, at the point where the SNR is equal to 0.2dB we started experiencing an error floor. Notice that this is not the case for the other ensembles. Apart from the good initial performance of the SC-PCC, SC-BCC outperforms the other ensembles as it had a steeper waterfall region compared to the others. We can also observe that performance of SC-HCC form-II and SC-SCC is almost similar.

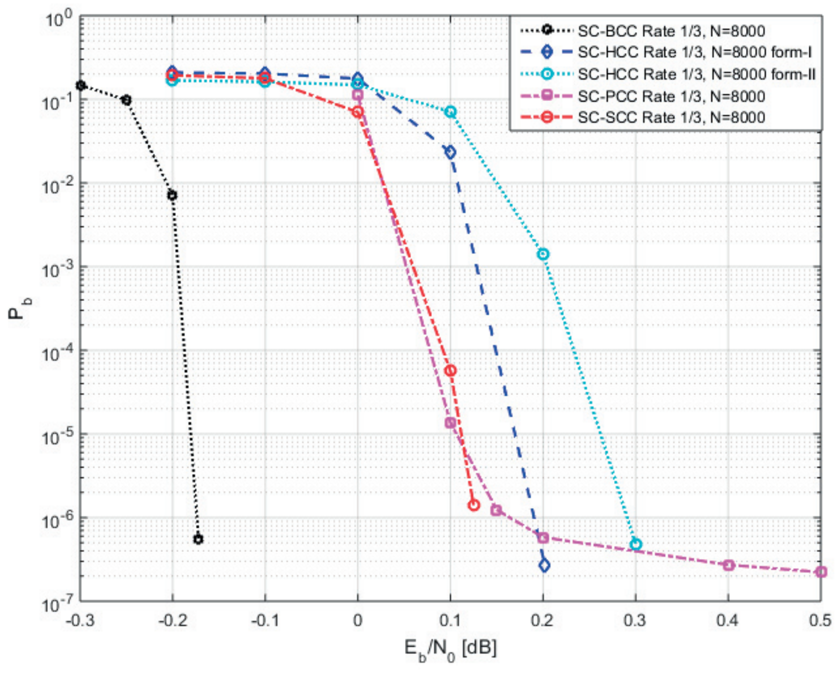


Figure 4.11: SC-HCC(form-I and form-II), SC-BCC, SC-SCC and SC-PCC results for input block length N=8000

In Figure 4.11, we observe that SC-BCC was the best of all comparison wise because it has a steeper waterfall region. After SC-BCC, we can also see that SC-SCC gets slightly steeper than SC-PCC. While SC-HCC (form-I and form-II) were slightly worse off.

Chapter 5

Conclusions

In this chapter, we present our observations in this thesis. We also discuss the differences or similarities in the different code ensembles which we investigated. The main codes ensembles which we investigated are namely: hybrid concatenated codes with and without spatial coupling and serially concatenated codes, also with and without spatial coupling. We had also compared some of our results with results from previous research work on braided convolutional code and parallel concatenated code ensemble, both of which having a spatially coupled form as well as their uncoupled forms. Of all the results presented in the results chapter, there are some notable observations which are the following;

For the uncoupled code ensembles;

- The longer the permutation size of the code, the better the code becomes until it reaches a saturation level
- When comparing puncturing forms in HCC which were investigated, form-I gives a low performance more than form-II in the uncoupled ensembles. By this we can conclude that, definitely, the puncturing pattern also has an effect on the performance of a code. The effects of different puncturing forms on a code can be investigated further in the future. Compare result of HCC uncoupled form-I and form-II in Figure 4.8
- The error floor in HCC uncoupled form-I indicates that the code has weak distance properties

For the coupled code ensembles;

- The longer the permutation size of the code, the better the code becomes until it reaches a saturation level.
- Coupling overall improves the code performance.
- In HCC, we have investigate two puncturing forms. While form-II was better than form-I in all the uncoupled cases, this is not the case for the coupled code ensembles. For the coupled ensembles, puncturing form-I became better than puncturing form-II when we simulated for input block length $N = 8000$, however, for the input block length of $N = 1000$, puncturing form-II was better off than puncturing form-I. More investigation on this behavior needs to be done. See the results in Figures 4.5 and 4.6.

5.1 Future work

There is a lot left to investigate in this area of this master thesis. We have investigated two different permutation sizes, sizes of one thousand and eight thousand. While we can observe that the longer the permutation size, the better the code becomes, there can be further investigations as to the threshold when it comes to the the permutation size.

For HCC, we have investigated two different puncturing forms. We can observe in the results that the puncturing pattern of a code can affect the code performance and it's distance properties. Further investigation can be done as to find a more efficient puncturing method which can be applied to a code to give better results.

Additionally, we can also investigate more on coupling memory and how large the coupling memory can be till we reach a threshold where it cannot improve the code performance anymore. In our simulation, we only used a coupling memory $m = 1$, and also used a window size of $w = 5$ for input block length $N = 8000$ and $w = 10$ for input block length $N = 1000$. More work needs to be done to find an optimal window size for a given code ensemble which yields the most optimal performance.

References

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

[2] Daniel J.Costello Jr. *Error Control Fundamental and Application Second edition* Pearson Prentice Hall. 2004.

[3] 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.

[4] Michael Lentmaier Lecture notes. *Iterative Decoding of Concatenated Codes,*EDI042 Error Control Coding,p.63-67 2015/2016.

[5] Sergio Benedetto, Dariush Divsalar, Guido Montorsi and Fabrizio Pollara. *Serially Concatenation of Interleaved Codes: Performance Analysis, Design, and Iterative Decoding* IEEE Transactions on Information Theory, May 1998.

[6] S. Moloudi, M. Lentmaier, and A. Graell i Amat.*Threshold Saturation for Spatially Coupled Turbo-like Codes over the Binary Erasure Channel*.

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

[8] 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.

[9] Ardiana Osmani and Hector Moreno. *Analysis of the finite length performance of spatially coupled convoulutional codes.* Master's Thesis, Department of Electrical and Information Technology Lund University November 19, 2015.

[10] S. Moloudi, M. Lentmaier, and A. Graell i Amat.*Spatially Coupled Turbo Codes.*

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

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

[13] G. David Forney, JR. *Concatenated codes.* Department of Electrical Engineering, M.I.T, December 1, 1965.

[14] Eirik Rosnes and A. Graell i Amat. *Performance Analysis of 3-D Turbo Codes.* 6th June 2011.

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

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

[17] W. Zhang, M. Lentmaier, K. Zigangirov, and D. J. Costello Jnr. *Braided Convolutional Codes: A New Class of Turbo-Like Codes*

[18] `http://www.lunarc.lu.se/resources/hardware/aurora/`

Printed by Tryckeriet i E-huset, Lund 2017