SOURCE-CONTROLLED BLOCK TURBO CODING

By

Shervin Pirestani

A thesis submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Master of Electrical Engineering

Summer 2005

SOURCE-CONTROLLED BLOCK TURBO CODING

By

Shervin Pirestani

ACKNOWLEDGMENTS

Javier Garcia-Frias, Ph.D. for his continuous advice, guidance, and academic support during the past several years.

My professional friend, Richard Demo Souza, who has supported and helped me throughout my graduate education.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

This thesis proposes the use of block turbo codes for transmitting non-uniform binary memoryless sources over AWGN channels. As opposed to standard convolutional turbo codes, block turbo codes are known to achieve good performance for high code rates, up to 98% of the channel capacity for uniform sources. The objective of this thesis is to investigate if such a performance is also achievable for non-uniform sources. The idea is to use a joint source-channel (or source controlled) coding scheme to exploit the statistics of the binary source at the decoder.

Block turbo codes are a class of product codes, which can be decoded by iteratively decoding the component codes. Each component code is decoded using the Chase algorithm, which is a low-complexity suboptimum algorithm [8] for near Maximum-Likelihood decoding of linear block codes. In this thesis, we propose decoding modifications to take into account the non-uniform nature of the source in the decoding of block turbo codes.

Chapter 1

INTRODUCTION TO TURBO CODES

In the first part of this Chapter we will introduce the concept of concatenated coding. We then present the Convolutional Turbo decoding scheme in section 1.2, and finally we conclude the Chapter by presenting Block Turbo codes in section 1.3.

1.1 Concatenated Coding

The theory of error correcting codes presents a large number of code constructions with their corresponding decoding algorithms. However, for applications where very strong error correcting capabilities are required, many of these constructions result in far too complex decoding solutions. One way to reduce complexity is to use concatenated coding, where two (or more) constituent codes are used after each other or in parallel - usually with some kind of interleaving. The constituent codes are decoded with their respective decoders, which exchange information in an iterative process to obtain the final result. This final result is usually sub-optimal. However, optimal schemes are not feasible, and concatenated coding offers a nice trade off between error correcting capabilities and decoding complexity.

An example of concatenated coding is illustrated in Figure 1.1 Here we represent the information bits as a square, and we generate horizontal parities by encoding each row with a block code, while vertical parities result from the encoding of each column. It is also possible to encode the horizontal and vertical parities to produce "parities of parities".

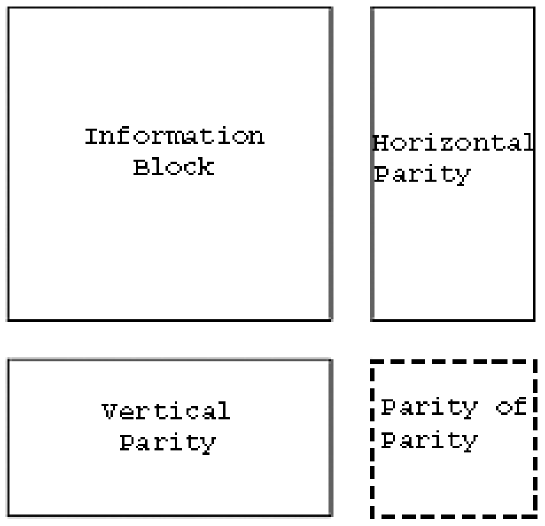

Figure 1.1: Example of Concatenated Coding.

1.2 Convolutional Turbo Codes

To provide a clear description of Convolutional Turbo Codes proposed in [5], we first start by explaining how the encoder functions. We then describe the turbo decoder and how iterative turbo decoding works. We conclude this section by discussing turbo codes performance.

1.2.1 Encoding

The general structure used in turbo encoders is shown in Figure 1.2. Two component codes are used to encode the same input bits, but an interleaver is placed between the encoders. For the case of Convolutional Turbo Codes, Recursive Systematic Convolutional (RSC) codes are used as the component codes. In this section we concentrate entirely on the standard turbo encoder structure using two RSC codes. Turbo codes using block codes as component codes are described in section 1.3. Figure 1.3 shows the RSC code we have used as component codes in most of our simulations.

The dramatic improvement in performance observed in turbo codes arises because of the interleaver used between the encoders, and because recursive codes are used as component codes. At moderate or high signal to noise ratios, turbo codes present a performance gain directly related to the interleaver length [2], [3].

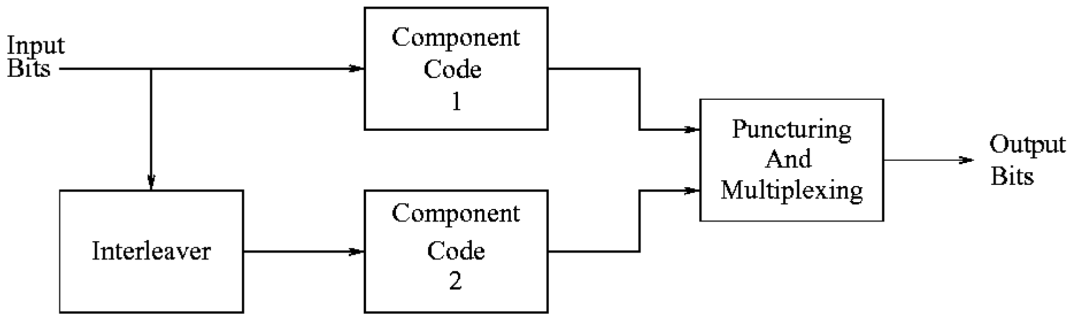

Figure 1.2: Turbo Encoder.

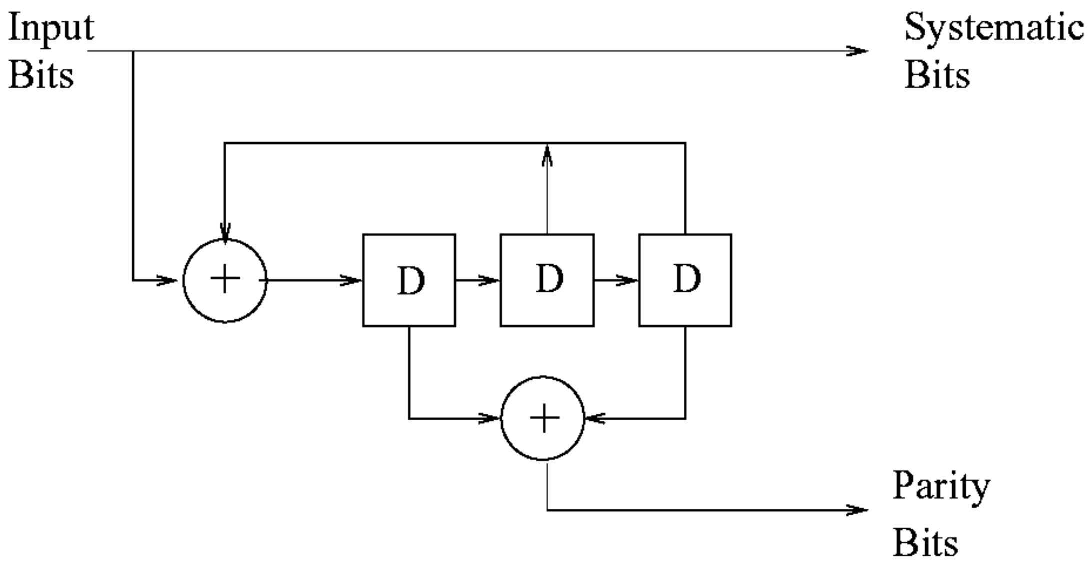

Figure 1.3: Recursive Systematic Convolutional (RSC) Encoder.

On the other hand the decoding complexity per bit does not depend on the interleaver length. Hence extremely good performance can be achieved with reasonable complexity by using very long interleavers.

1.2.2 Turbo Decoder

The general structure of an iterative turbo decoder is shown in Figure 1.4. There are two component decoders which are linked by interleavers in a structure similar to that of the encoder. As seen in the Figure, each decoder takes three inputs – the systematically encoded channel output bits, the parity bits transmitted from the associated component encoder, and the information from the other component decoder about the likelihood of the information bits. This information from the other decoder is referred to as extrinsic information. The component decoders have to utilize both, the input from the channel and this extrinsic information to perform decoding. They must also provide the so-called soft outputs for the decoded bits, which will be used as extrinsic information in the other decoder. This means that in addition to the decoded output bit sequence, the component decoders must also provide the associated probabilities for each bit. The soft outputs are typically represented in terms of the so-called Log Likelihood Ratios (LLRs), the magnitude of which gives the sign of the bit, and the amplitude the probability of a correct decision.

The decoder of Figure 1.4 functions iteratively. In the first iteration the first component decoder takes channel output values only, and produces a soft output as its estimate of the data bits. The soft output from the first encoder is then used as additional information for the second decoder, which uses this information along with

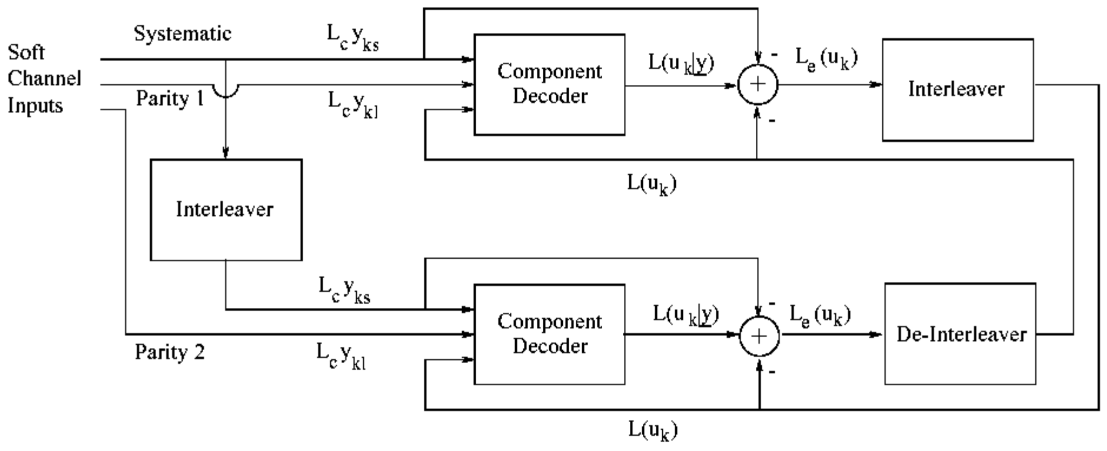

Figure 1.4: Turbo Decoder (Figure taken from [15]).

the channel outputs to calculate its estimate of the data bits. In the second iteration, the first decoder decodes the channel outputs again, but now with additional information about the value of the input bits provided by the output of the second decoder in the first iteration. This additional information allows the first decoder to obtain a more accurate set of soft outputs, which are then used by the second decoder as *a priori* information. This cycle is repeated, and with every iteration the Bit Error Rate (BER) of the decoded bits decreases. On the other hand, the enhancement in performance decreases as the number of iterations increases. Therefore, for complexity reasons, we usually perform only about eight iterations.

1.2.3 <u>BCJR Algorithm</u>

The BCJR algorithm proposed in [1] obtains, for a convolutional code, the maximum a posteriori (MAP) for each input bit, therefore minimizing the bit error probability. This is critical for the iterative decoding of turbo codes. The BCJR algorithm provides, for each decoded bit u_k, the probability that this bit was +1 or -1, given the received symbol sequence $\underline{y}$. We define the conditional Log Likelihood Ratio of $L\left(u_k \middle| \underline{y}\right)$ as:

$$L\left(u_k \middle| \underline{y}\right) = \ln\left(\frac{P\left(u_k = +1 \middle| \underline{y}\right)}{P\left(u_k = -1 \middle| \underline{y}\right)}\right). \tag{1.1}$$

The sign of the Log Likelihood Ratio (LLR), defines the bit decision, while its magnitude provides the decision reliability. Using Bayes'rule (1.1) can be rewritten as:

$$L\left(u_k \middle| \underline{y}\right) = \ln\left(\frac{P\left(u_k = +1, \underline{y}\right)}{P\left(u_k = -1, \underline{y}\right)}\right). \tag{1.2}$$

Let us now consider the RSC code shown in Figure 1.3. For this code $K = 3$ (where K is the number of stages for the shift register) there are four states, and, as it is a binary code, for each state two transitions are possible, one associated with input bit -1, and the other to +1. If the previous state S_{k-1} and the present state S_k are known, then the value of the input bit u_k, which caused the transition between these two states, will be known. Hence, the probability of $u_k = +1$ is equal to the probability that the transition from the previous state to the present state is associated with input bit +1. Therefore, we can rewrite Equation (1.2) as:

$$L\left(u_k \middle| \underline{y}\right) = \ln\left(\frac{\sum\limits_{(s',s)\Rightarrow u_k=+1} P\left(S_{k-1}=s', S_k=s, \underline{y}\right)}{\sum\limits_{(s',s)\Rightarrow u_k=-1} P\left(S_{k-1}=s', S_k=s, \underline{y}\right)}\right), \tag{1.3}$$

where $(s',s)\Rightarrow u_k=+1$ is the set of transitions from the previous state $S_{k-1}=s'$ to the present state $S_k=s$, associated to input bit +1.

We can split up the received sequence into three sections: the received codeword associated with the present transmission $\underline{y}_k$, the received sequence prior to the present transmission $\underline{y}_{j<k}$, and the received sequence after the present transmission $\underline{y}_{j>k}$. Obviously,

$$P\left(s', s, \underline{y}\right) = P\left(s', s, \underline{y}_{j<k}, \underline{y}_k, \underline{y}_{j>k}\right). \tag{1.4}$$

Using Bayes' rule and the Markov property, we can rewrite (1.4) as:

$$\begin{aligned} P\left(s', s, \underline{y}\right) &= P\left(s', s, \underline{y}_{j<k}, \underline{y}_k\right)\cdot P\left(\underline{y}_{j>k}\middle|s\right) \\ &= P\left(s', \underline{y}_{j<k}\right)\cdot P\left(\left\{\underline{y}_k, s\right\}\middle|s'\right)\cdot P\left(\underline{y}_{j>k}\middle|s\right) \\ &= \alpha_{k-1}(s')\cdot\gamma_k(s',s)\cdot\beta_k(s), \end{aligned} \tag{1.5}$$

where:

$$\alpha_{k-1}(s') = P(S_{k-1}=s', \underline{y}_{j<k}) \tag{1.6}$$

is the probability that the trellis is in state s' at time $k-1$ and the received channel sequence up to this point is $\underline{y}_{j<k}$.

$$\beta_k\left(s\right) = P\left(\underline{y}_{j>k} \middle| S_k = s\right) \tag{1.7}$$

is the probability that, given that the trellis is in state s at time k, the received channel sequence from this point on is $\underline{y}_{j>k}$.

$$\gamma_k\left(s, s'\right) = P\left(\left\{\underline{y}_k, S_k = s\right\} \middle| S_{k-1} = s'\right) \tag{1.8}$$

is the probability that if the trellis was in state s' at time $k-1$, it moves to state s and the received channel sequence at time k is $\underline{y}_k$.

The idea of defining parameters α and β is that they can be easily calculated through recursion, which leads to a simple decoding algorithm.

Forward recursive calculation of $\alpha_k(s)$ values:

$$\begin{aligned}\alpha_k(s) &= P(S_k = s, \underline{y}_{j<k+1}) \\ &= \sum_{\text{all } s'} P\left(s, s', \underline{y}_{j<k}, \underline{y}_k\right) \\ &= \sum_{\text{all } s'} P\left(\left\{s, \underline{y}_k\right\} \middle| s'\right) \cdot P\left(s', \underline{y}_{j<k}\right) \\ &= \sum_{\text{all } s'} \gamma\left(s', s\right) \cdot \alpha_{k-1}\left(s'\right).\end{aligned} \tag{1.9}$$

Thus, by knowing the $\gamma\left(s', s\right)$ values, $\alpha_k(s)$ can be calculated recursively. The initial condition for the recursion is (assuming initial $S_0 = 0$)

$$\alpha_0\left(S_0 = 0\right) = 1$$

$$\alpha_0\left(S_0 = s\right) = 0 \text{ for all } s \neq 0.$$

Backward recursive calculation of $\beta_k(s)$ values:

$$\beta_{k-1}(s') = P\left(\underline{y}_{-j>k-1} \middle| s'\right)$$

$$= \sum_{\text{all } s} P\left(\left\{ \underline{y}_k, \underline{y}_{j>k}, s\right\} \middle| s'\right)$$

$$= \sum_{\text{all } s} P\left(\underline{y}_{j>k} \middle| s\right) \cdot P\left(\left\{\underline{y}_k, s\right\} \middle| s'\right)$$

$$= \sum_{\text{all } s} \beta_k(s) \cdot \gamma(s', s). \tag{1.10}$$

Thus, by knowing the $\gamma(s', s)$ values, $\beta_{k-1}(s)$ can be calculated from the values of $\beta_k(s)$.

Calculation of $\gamma_k(s', s)$ values:

$$\gamma(s', s) = P\left(\left\{\underline{y}_k, s\right\} \middle| s'\right)$$

$$= P\left(\underline{y}_k \middle| \left\{s', s\right\}\right) \cdot P(u_k), \tag{1.11}$$

where u_k is the input bit that is needed for transition from state $S_{k-1} = s'$ to state $S_k = s$. From Equations (1.3) and (1.5) we can write the conditional LLR of u_k, given the received sequence $\underline{y}_k$

$$L\left(u_k \middle| \underline{y}\right) = \ln\left(\frac{\displaystyle\sum_{(s', s) \Rightarrow u_{k=+1}} \alpha_{k-1}(s') \cdot \gamma_k(s', s) \cdot \beta_k(s)}{\displaystyle\sum_{(s', s) \Rightarrow u_{k=-1}} \alpha_{k-1}(s') \cdot \gamma_k(s', s) \cdot \beta_k(s)} \right). \tag{1.12}$$

It is this conditional LLR $L\left(u_k \middle| \underline{y}\right)$ that the MAP decoder delivers.

The MAP decoding of a received sequence $\underline{y}$ given the a-posteriori LLR $L(u_k \mid y)$ can be calculated as we explain in this section. As the channel values y_k are received, they and the *a priori* LLRs $L(u_k)$ (which are provided in an iterative turbo decoder by the other component decoder) are used to calculate $\gamma_k(s', s)$. Thus, the forward recursion can be used to calculate $\alpha_k(s')$ and the backward recursion can be used to calculate $\beta_k(s)$. Finally all the calculated values of $\alpha_k(s')$, $\beta_k(s)$, and $\gamma_k(s', s)$ are used to calculate the values of $L(u_k \mid y)$.

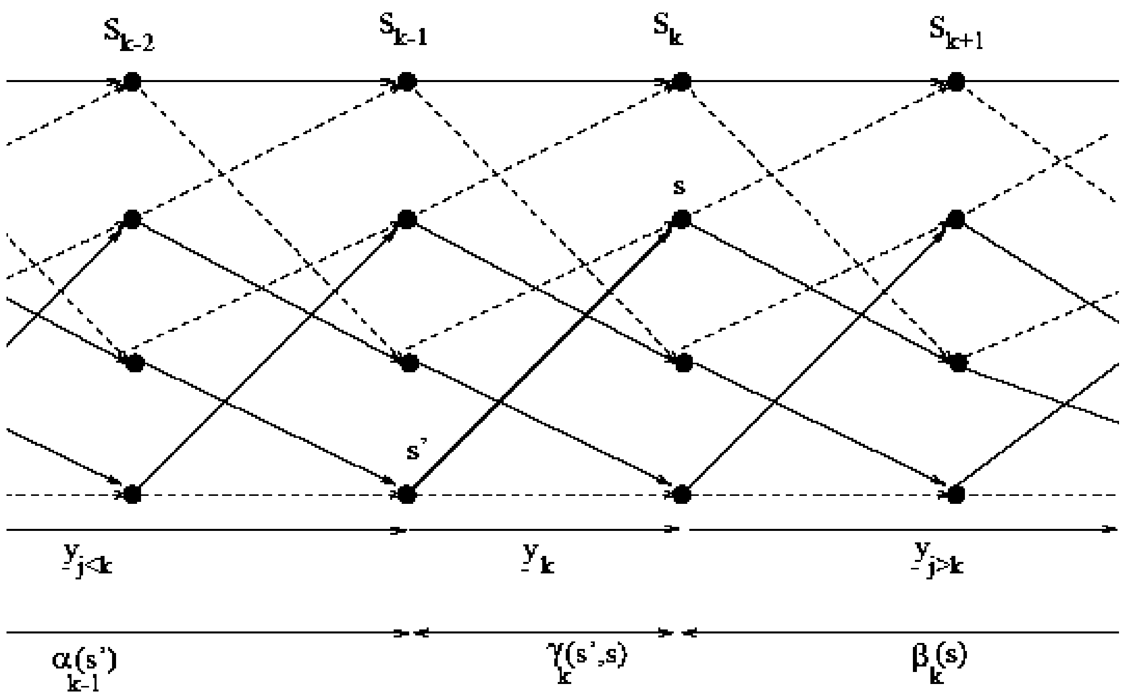

Figure 1.5: Trellis for the K=3 RSC code shown in Figure 1.3.

Figure 1.5 shows the trellis for the RSC represented in Figure 1.3. All the operations are summarized in the flowchart of Figure 1.6, and Figure 1.7 shows the recursive calculation of $\alpha_k(s)$ and $\beta_k(s)$. Care must be taken to avoid numerical underflow problems in the recursive calculation of $\alpha_k(s)$ and $\beta_k(s)$ but such problems can be avoided by using a logarithmic representation of these values.

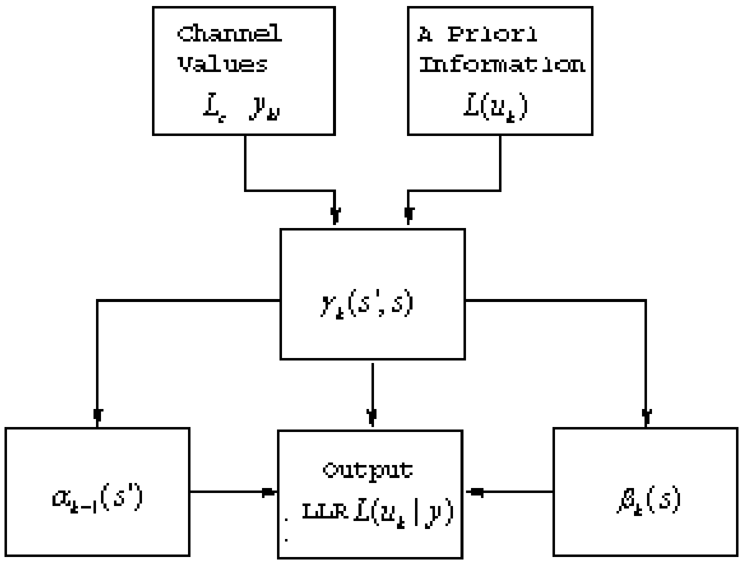

Figure 1.6: Summary of the MAP algorithm (Figure taken from [15]).

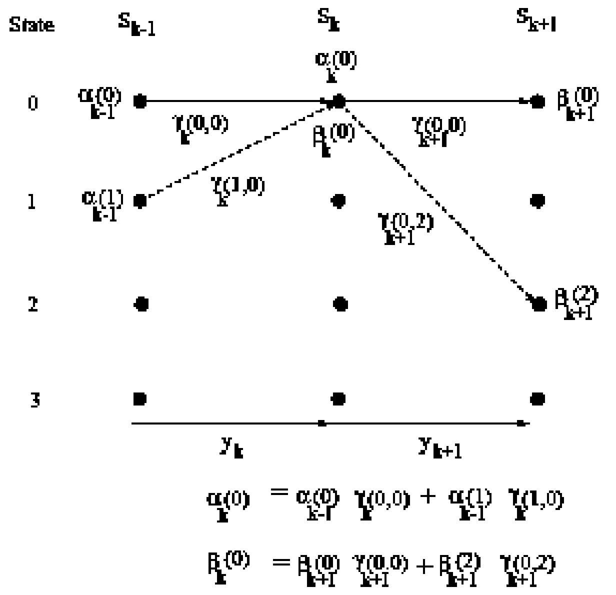

Figure 1.7: Recursive calculation of $\alpha_k(0)$ and $\beta_k(0)$ (Figure taken from [15]).

In the form described in this section, the MAP algorithm presents numerical problems due to the multiplications required in the equations for the recursive calculation of $\alpha_k(s)$ and $\beta_k(s)$. However, these numerical problems can be eliminated by using the Log-MAP algorithm, which gives the same performance as the MAP algorithm [26].

In order to define the Log-MAP algorithm, we first define $\max^*$ as:

$$\max_i{}^*(x_i) = \ln\left(\sum_i e^{x_i}\right),$$

where $\max^*$ is calculated as:

$$\max{}^*\{k_1, k_2\} = \max\{k_1, k_2\} + \ln\left(1 + e^{-|k_2 - k_1|}\right).$$

Then, by defining

$$\mathrm{A}_k(s) = \ln(\alpha_k(s))$$
$$\mathrm{B}_k(s) = \ln(\beta_k(s))$$

and

$$\Gamma_k(s',s) = \ln(\gamma_k(s',s)),$$

we can write Equation (1.9) as:

$$\begin{aligned} A_k(s) &= \ln(\alpha_k(s)) \\ &= \ln\left(\sum_{\text{all } s'} \alpha_{k-1}(s')\gamma_k(s',s)\right) \\ &= \ln\left(\sum_{\text{all } s'} \exp\left[\mathrm{A}_{k-1}(s') + \Gamma_k(s',s)\right]\right) \\ &= \max_{s'}{}^{*}\left(\mathrm{A}_{k-1}(s') + \Gamma_k(s',s)\right). \end{aligned} \quad (1.13)$$

The value of $A_k(s)$ should give the natural logarithm of the probability that the trellis is in state $S_k = s$ at stage k and that the received channel sequence up to this point has been $\underline{y}_{j<k}$.

Similar to Equation (1.12) we can rewrite Equation (1.10) as

$$\begin{aligned} B_{k-1}(s') &= \ln(\beta_{k-1}(s')) \\ &= \ln\left(\sum_{\text{all } s} \beta_k(s)\gamma_k(s',s)\right) \\ &= \ln\left(\sum_{\text{all } s} \exp\left[\mathrm{B}_{k-1}(s) + \Gamma_k(s',s)\right]\right) \\ &= \max_{s}{}^{*}\left(\mathrm{B}_k(s) + \Gamma_k(s',s)\right). \end{aligned} \quad (1.14)$$

Equation (1.11) can be rewritten as

$$\begin{aligned}\Gamma_k(s',s) &= \ln\left(\gamma_k(s',s)\right) \\ &= K + u_k L(u_k) + L_c \sum_{l=1}^{n} y_{kl} x_{kl}\end{aligned} \tag{1.15}$$

Finally, from Equation (1.12), we can write the a-posteriori LLRs $L\left(u_k \middle| \underline{y}\right)$ as:

$$L\left(u_k \middle| \underline{y}\right) = \ln\left(\frac{\sum\limits_{(s',s)\Rightarrow u_{k=+1}} \alpha_{k-1}(s')\cdot\gamma_k(s',s)\cdot\beta_k(s)}{\sum\limits_{(s',s)\Rightarrow u_{k=-1}} \alpha_{k-1}(s')\cdot\gamma_k(s',s)\cdot\beta_k(s)}\right)$$

$$= \ln\left(\frac{\sum\limits_{(s',s)\Rightarrow u_{k=+1}} \exp\left(\mathrm{A}_{k-1}(s') + \Gamma_k(s',s) + \mathrm{B}_k(s)\right)}{\sum\limits_{(s',s)\Rightarrow u_{k=-1}} \exp\left(\mathrm{A}_{k-1}(s') + \Gamma_k(s',s) + \mathrm{B}_k(s)\right)}\right)$$

$$\begin{aligned}&= \max_{(s',s)\Rightarrow u_k=+1}{}^{*} \left(\mathrm{A}_{k-1}(s') + \Gamma_k(s',s) + \mathrm{B}_k(s)\right) \\ &- \max_{(s',s)\Rightarrow u_k=-1}{}^{*} \left(\mathrm{A}_{k-1}(s') + \Gamma_k(s',s) + \mathrm{B}_k(s)\right).\end{aligned} \tag{1.16}$$

1.2.4 <u>Iterative Turbo Decoding</u>

We now describe in detail how the iterative decoding of turbo codes is carried out. In Figure 1.4, we showed the structure of an iterative turbo decoder. Consider initially the first component decoder in the first iteration. This decoder receives the channel sequence $L_c \underline{y}^{(1)}$ containing the received versions of the transmitted systematic bits, $L_c y_{ks}$ and the parity bits, $L_c y_{kl}$, from the first encoder. Usually, to obtain a half-rate code, half of these parity bits are punctured at the transmitter, and therefore the turbo decoder must insert zeros in the soft channel output $L_c y_{kl}$ for these punctured bits.

The first component decoder can then process the soft channel inputs and produce its estimate $L_{11}(u_k \mid \underline{y})$ of the conditional LLRs of the data bits u_k, k = 1, 2 … N. In this notation the subscript ij in $L_{ij}(u_k \mid \underline{y})$ indicates that this is the *a posteriori* LLR in the i iteration from the j component decoder. Note that (for uniform sources) in this first iteration the first component decoder will have no *a priori* information about the bits, and hence $L(u_k)$=0. Next, the second component decoder comes into operation. It receives the channel sequence $L_c \underline{y}^{(2)}$ containing the parity bits from the second encoder. However, in addition to the received channel sequence $L_c \underline{y}^{(2)}$, the decoder can use the conditional LLR $L_{11}(u_k \mid \underline{y})$ provided by the first component decoder to generate *a priori* LLRs $L(u_k)$ to be used by the second component decoder. Ideally these *a priori* LLRs $L(u_k)$ would be completely independent from all the other information used by the second component decoder. As shown in Figure 1.4, in iterative turbo decoders the extrinsic information $L_c(u_k)$ from the other component decoder is used as the *a priori* LLRs, after being interleaved to arrange the decoded data bits u in the same order as they were encoded by the second encoder. The reason for the subtraction paths shown in Figure 1.4 is that the *a posteriori* LLRs from one decoder have the systematic soft channel inputs $L_c y_{ks}$ and the *a priori* LLRs $L(u_k)$, subtracted to yield the extrinsic LLRs $L_c(u_k)$ which are then used as *a priori* LLRs for the other component decoder. The second component decoder thus uses the received channel sequence $L_c \underline{y}^{(2)}$ and the *a priori* LLRs $L(u_k)$ (derived by interleaving the extrinsic LLRs $L_c(u_k)$ of the first component decoder) to produce its a-posteriori LLRs $L_{12}(u_k \mid \underline{y})$. This completes the first iteration.

For the second iteration the first component encoder again processes its received channel sequence $L_c \underline{y}^{(1)}$, but now it also has *a priori* LLRs $L(u_k)$ provided by the extrinsic portion $L_c(u_k)$ of the *a posteriori* LLRs $L_{12}(u_k \mid \underline{y})$ calculated by the second component encoder, and hence it can produce an improved *a posteriori* LLR $L_{21}(u_k \mid \underline{y})$. The second iteration then continues with the second component decoder using the improved extrinsic LLRs $L_c(u_k)$ from the first encoder as *a priori* LLRs $L(u_k)$ which is used in conjunction with its received channel sequence $L_c \underline{y}^{(2)}$ to calculate $L_{22}(u_k \mid \underline{y})$ [15].

This iterative process continues, and with each iteration the BER of the decoded bits will decrease. However, as we will show in the next section, the improvement in performance for each additional iteration slows down as the number of iterations increases. Hence, for complexity reasons usually only about eight iterations are carried out, as no significant improvement in performance is obtained with a higher number of iterations. It is also possible to use a variable number of iterations up to a maximum, with some termination condition used to decide when it is believed that further iterations will produce marginal gain. This allows the average number of iterations, and the average complexity of the decoder, to be severely reduced [14] with only a small performance degradation.

1.2.5 <u>Simulation of Turbo codes</u>

The graph below shows the simulation of Turbo Codes for uniform sources over AWGN channels using BPSK modulation. The RCS parameters were n=2, k=1, K=3 and an interleaver of length 1000 was used.

This graph presents the results when different number of iterations are utilized. As the number of iteration increases, the bit error rate will decrease. This graph also presents the results for an uncoded system, which clearly show how effective Turbo Codes are in decreasing the bit error rate.

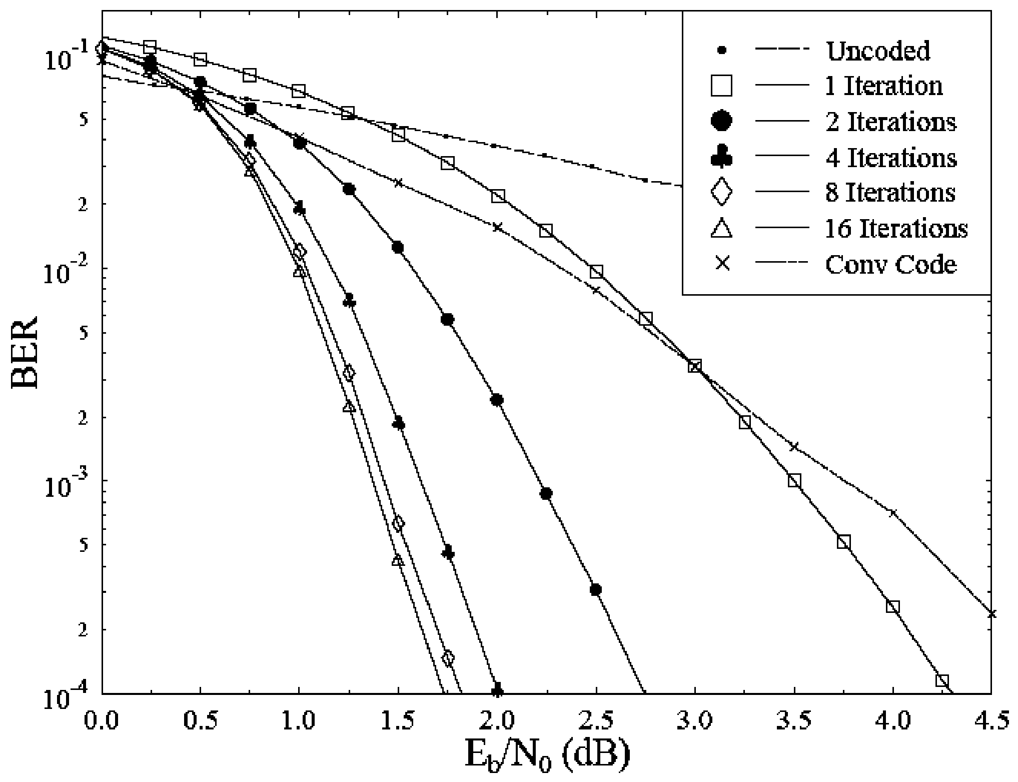

Figure 1.8: Simulation of Turbo codes. This simulation was performed over an AWGN (Additive White Gaussian Noise) channel using BPSK (Binary Phase Shift Keying) modulation. The RSC parameters were n=2,k=1,K=3 and an interleaver of length 1000 was used. (Graph taken from [15]).

1.3 Block Turbo Codes

To give a clear and complete description of Block Turbo Codes (BTC), we first describe the basic idea behind them. We start by briefly explaining product codes. Then, we describe the Chase algorithm which will be used to perform soft input decoding of the component codes, and BCH codes, which will be used as component codes. Afterwards, we discuss how to perform soft output decoding of block turbo codes. We conclude this section by presenting the results on the performance of block turbo codes for uniform sources.

1.3.1 Product Codes or Concatenation of Two Block Codes

Product codes were first introduced by Elias in 1954 [11]. Let us assume two systematic linear block codes C_1 with parameters (n_1, k_1, δ_1) and C_2 with parameters (n_2, k_2, δ_2) where n_i, k_i and δ_i (i=1,2) stand for codeword length, number of information bits and minimum Hamming distance, respectively. The concatenation of two block codes (or product code) $P = C_1 \times C_2$ is obtained (see Figure 1.9) by first placing (k_1, k_2) information bits in an array of k_1 rows and k_2 columns, and then coding each one of the k_1 rows using code C_2, and each one of the k_2 columns using code C_1.

The parameters of the product code P [18] are $n = n_1 \times n_2$, $k = k_1 \times k_2$, $\delta = \delta_1 \times \delta_2$ and the code rate R is given by $R_1 \times R_2$ where R_i is the code rate of code C_i. Thus, we can build very long block codes with large minimum Hamming distance by joining short codes with small minimum Hamming distance. Given the defined procedure, it is clear that the $(n_2 - k_2)$ last columns of the matrix are codewords of C_1. By using the generator matrix, we can show [18] that the first k_1 rows of matrix P are codewords of C_2. Hence all the columns of matrix P are codewords of C_1 and the first k_1 rows of matrix P are codewords of C_2 [22].

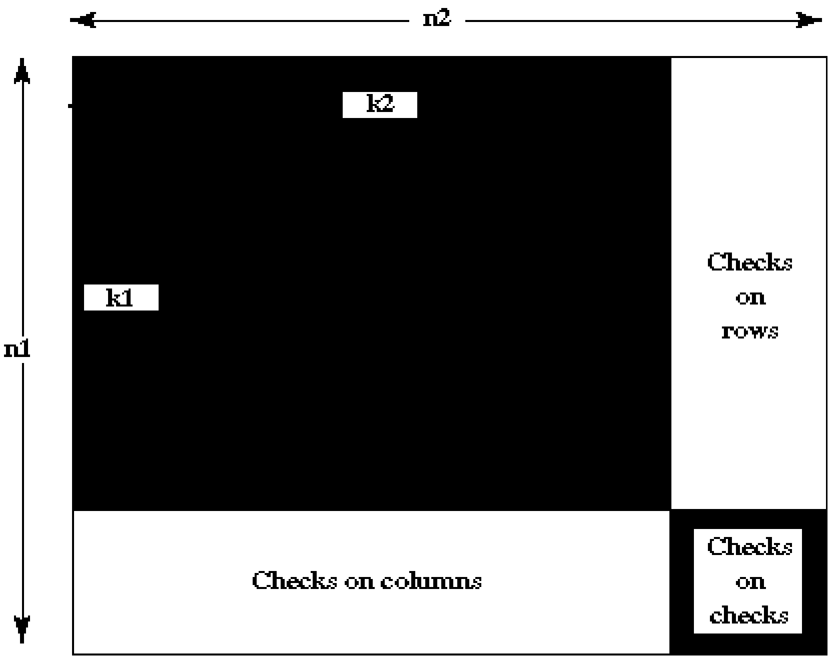

Figure 1.9 Example of a product code $P = C_1 \times C_2$.

1.3.2 Soft decoding of Block Codes

Let us assume transmission of binary elements $\{0,1\}$ coded by a linear block code C with parameters (n,k,δ) on a Gaussian channel using BPSK $\{-1,+1\}$. The observation $R=(r_1,r_2,...,r_n)$ at the output of the Gaussian channel for a transmitted codeword $E=(e_1,e_2,...,e_n)$ is given by

$$R=E+N\,,$$

where components n_i of $N=(n_1,n_2,...,n_n)$ are AWGN samples with zero mean and standard deviation σ. We know, by using Maximum Likelihood decoding, that the optimum decision will be

$$D=C_i \text{ if } \mathrm{P}\left(E=C_j \mid R\right)<\mathrm{P}\left(E=C_i \mid R\right). \tag{1.17}$$

Assuming equal probability for binary elements, decision D can be calculated [10] as:

$$D=C_i \text{ if } \left|R-C_i\right|-2\sigma^2 \ln\left(\mathrm{P}\left(E=C_i\right)\right) < \left|R-C_j\right|-2\sigma^2 \ln(\mathrm{P}(E=C_j)), \tag{1.18}$$

where

$$\left|R-C_i\right|^2=\sum_{l=1}^{n}\left(r_l-c_{il}\right)^2 \tag{1.19}$$

is the Euclidean distance between R and C_i. Obviously searching for the optimum codeword D using an exhaustive search increases the complexity exponentially as we increase k.

Chase Algorithm

Chase proposed a suboptimum algorithm of low complexity [8] for near ML decoding of linear block codes. We can say that for high SNR, ML decoding of codeword D is located, with a very high probability, in the sphere of radius of $(\delta - 1)$ centered on $Y = (y_1, y_2, ..., y_n)$, where $y_l = \frac{1}{2} + \frac{1}{2}\text{sgn}(r_l)$ (note that $y_l \in \{0,1\}$). Using this technique, we can limit the reviewed codewords in (1.17) to those in the sphere of radius $(\delta - 1)$ centered on Y so that the complexity decreases. The procedure for selecting these most probable codewords is specified in the following steps:

1. Find the position of the $p = \lfloor \delta/2 \rfloor$ least reliable binary elements of Y using R. The reliability of Y will be defined later.
2. Generate test patterns T^i defined as all the n-dimensional binary vectors generated by all possible combinations of "1"s and "0"s in the p positions and "0"s in the other positions.
3. Generate test sequence $Z^i = Y \oplus T^i$ for all test pattern T^i and then decode Z^i using a standard hard decoder (extended BCH decoding, see next section), adding the decoded codeword to subset Ω.
4. To find the final decision D, we apply decision rule (1.18) only to the codewords contained in subset Ω defined in step 3. Obviously, the components of the codewords are mapped from $\{0,1\}$ to $\{-1,+1\}$ before computing the Euclidean distance.

The reliability of y_j in step 1, is defined using the log-likelihood ratio of decision y_j,

$$\mathrm{R(y_j)} = \ln\left(\frac{\mathrm{P}\left(e_j = +1 \mid r_j\right)}{\mathrm{P}\left(e_j = -1 \mid r_j\right)}\right), \tag{1.20}$$

or

$$\mathrm{R(y_j)} = \left(\frac{2}{\sigma^2}\right) r_j .$$

Considering a stationary channel, we can normalize the log-likelihood ratio with respect to constant $2/\sigma^2$, so that the relative reliability of y_j is then given by $\left|r_j\right|$ [22].

The turbo decoding algorithm presented in this paper can be applied to any product code based on linear block codes. The results here concern to Bose-Chaudhuri-Hocqenghem (BCH) product codes.

<u>**BCH codes**</u>

BCH codes were proposed by Bose and Ray-Chaudhuri [16] and Hocquenghem [17]. A BCH code is a multilevel, cyclic, error-correcting code that is commonly used in communication systems. BCH codes are not limited to binary codes, but may be used with multilevel phase-shift keying whenever the number of levels is a prime number or a power of a prime number, such as 2, 3, 4, 5, 7, 8, 11, and 13. BCH codes are the most powerful linear block codes for short to moderate block lengths.

Before defining BCH codes we start by briefly discussing cyclic codes, since BCH codes are cyclic ones. Cyclic codes are linear block codes with the additional property that a cyclic shift of any codeword is also a codeword. This also means that the codewords constitute a group under the cyclic shift operation.

Assume $x=(x_0,x_1,...,x_{n-1})$ represents a codeword with elements in $\mathrm{GF}(q)$. We can associate it with a polynomial over $\mathrm{GF}(q)$ of degree at most $n-1$ defined as:

$$x(D)=x_0+x_1D^1+x_2D^2+...+x_{n-1}D^{n-1} \ .$$

If we consider a one-position right cyclic shift of x, producing $x^{(1)}=(x_{n-1},x_0,x_1,...,x_{n-2})$, the associated polynomial for this codeword will be:

$$x^{(1)}(D)=x_{n-1}+x_0D^1+x_1D^2+...+x_{n-2}D^{n-1},$$

which is another polynomial of degree at most $n-1$.

We can see that the two polynomial $x(D)$ and $x^{(1)}(D)$ are related by

$$x^{(1)}(D)=Dx(D)\bmod\left(D^n-1\right).$$

It can be shown that in general $D^jx(D)\bmod\left(D^n-1\right)$ is the code polynomial corresponding to the right-cyclic shift of codeword $x(D)$ by j positions.

Given a particular (n,k) cyclic code over $\mathrm{GF}(q)$, we define the generator polynomial $g(D)$ of the cyclic code as the monic polynomial (a polynomial with a leading coefficient of 1) of minimum degree among the set of non-zero codeword polynomials. We assume that the degree of this polynomial is $r \le n-1$, and represent it as:

$$g(D) = g_0 + g_1 D^1 + \ldots + g_r D^r ,$$

where the coefficients belong to $\mathrm{GF}(q)$. It is easy to show that there is a unique choice for the generator polynomial.

BCH codes are defined over a $\mathrm{GF}(q)$ field to obtain a simplified decoding algorithm. Given a field $\mathrm{GF}(q)$, a block length $n \ge 3$, which is a divisor of $q^m - 1$ for some m, and $3 \le \delta \le n$, an (n,k) BCH code over $\mathrm{GF}(q)$ is a cyclic code generated by [29]

$$g(D) = \mathrm{LCM}\left[m_{\beta^j}(D), m_{\beta^{j+1}}(D), m_{\beta^{j+2}}(D), \ldots, m_{\beta^{\delta+2-j}}(D) \right]. \tag{1.21}$$

The $m_{\beta^j}(D)$ are minimal polynomials of a primitive element β (with $\delta - 1$ succesive powers) whose order is n in extension field $\mathrm{GF}(q^m)$. A *minimal polynomial* is defined as follows: assuming α is defined in a given field $\mathrm{GF}(q)$. We say that α is algebraic if, for some *r*, the vector $(1, \alpha, \alpha^2, \ldots, \alpha^r)$ has an integer relation (i.e. $\sum_{i=1}^{r} \alpha_i \alpha^i = 0$ for some $\alpha_i \in \mathrm{Z}$). The integer coefficient polynomial of lowest degree, having α as a root, is determined uniquely up to a constant multiple and is called the *minimal polynomial* for α. LCM refers to the latest common multiple polynomial or smallest degree monic polynomial, for which all the indicated minimal polynomials are divisors. Since minimal polynomials are irreducible, finding the LCM polynomial will be equivalent to form the product of the distinct polynomials in (1.21).

It is important to understand that a BCH code is the largest set of codewords x whose corresponding polynomials $x(D)$ have as roots $\delta - 1$ successive powers of an element β of order n in an extension filed of $\mathrm{GF}(q)$ [29].

In Equation (1.21), $g(D)$ is a divisor of $D^n - 1$ since each minimal polynomial is a divisor of $D^n - 1$ which is sufficient enough to generate a cyclic code of length $n = q^m - 1$. The code's dimension, $k = n - \deg g(D)$, will depend on the degree of the polynomial in (1.21). The degree of $g(D)$ is less than or equal to $m(\delta - 1)$, since there are at most $\delta - 1$ distinct minimal polynomials (with at most degree of m) involved in the construction of $g(D)$. Thus, we define the following relations for BCH codes over any field:

$$n = q^m - 1 \tag{1.22}$$

$$n - k = \deg\ g(D) \leq m(\delta - 1) \tag{1.23}$$

δ is also called the design distance of the code and $t = \left[(\delta - 1)/2\right]$ is the designed error correction capability of the code [29]. Hard decoding algorithms with relatively low complexity can be easily designed for these codes [29].

1.3.3 Calculating the Soft Output

The Chase algorithm yields for each row (or column) the decision D corresponding to the component block code. To decode concatenated block codes by iterating between both component codes, we must compute the reliability of decision D.

To calculate the reliability of decision d_j at the output of the soft decoder, we require two codewords [20]. Hard decision D is one of these two codewords, and additionally, we need to calculate a "competing" codeword of D. The competing codeword of D, denoted as C, is obtained by applying (1.18) to the set Ω, excluding the hard decision D. Then, the soft output (reliability of decision d_j) can be obtained as [22] :

$$R_j = \left(\frac{|R-C|^2 - |R-D|^2}{4} \right) d_j \tag{1.24}$$

To find the competing codeword of D, codeword C, we must increase the size of the space scanned by the Chase algorithm. For this purpose, we increase the number of least reliable bits, p, used in the Chase decoder and also the number of test patterns. Unfortunately, the complexity of the decoder increases exponentially with p and we must find a trade-off between complexity and performance. To reduce complexity we utilize another method for computing the soft output [22], which is the following:

$$r_j = \beta(m) \times d_j \text{ with } 0 \leq \beta(m) \tag{1.25}$$

This formula gives a simplified and effective solution for calculating the soft output. The value of $\beta(m)$ was initially optimized by trial and error and depends on the iteration number [24].

1.3.4 <u>Iterative decoding of product codes</u>

Let us consider the decoding of the rows and columns of a product code P transmitted on a Gaussian channel using QPSK signaling. On receiving matrix [R] corresponding to a transmitted codeword [E], the first decoder performs the soft decoding of the rows or columns of P using [R] as input matrix. Soft Input / Soft Output decoding is performed using the algorithm described in the previous section. By subtracting matrix [R] from the soft output (computed using (1.24) or (1.25)) [25], we obtain the extrinsic information $\left[W(m)\right]$, where index m specifies that we are looking at the extrinsic information for the m^th^ iteration in the decoding of P. The soft input for the decoding of the columns or rows in the subsequent iteration of P is given by:

$$\left[R(m)\right] = [R] + \alpha(m)\left[W(m)\right] \tag{1.26}$$

Where $\alpha(m)$ is a scaling factor, which takes into account the fact that the standard deviation of samples in matrix $[R]$ and in matrix $[W]$ are different [5, 4]. In the first decoding steps, the standard deviation of the extrinsic information is very high, but it decreases with the iteration number. This scaling factor is also used to reduce the weight of the extrinsic information in the first decoding steps when the BER is relatively high. Therefore, it takes a small value in the first decoding steps and increases with the iteration number (i.e., as the BER tends to 0). A summary of the decoding procedure is presented in Figure 1.10.

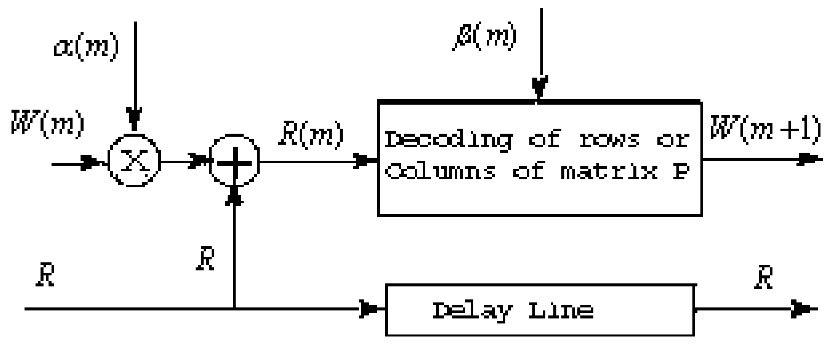

Figure 1.10 Block diagram of a iteration in the decoding of block turbo codes.

1.3.5 Simulation of Block Turbo Codes

The graph below shows the simulation of Block Turbo codes for uniform sources and AWGN channels using QPSK modulation. This graph shows 8 simulations for different code rates with 4 iterations of the decoding process. Notice that, as the rate increases, the bit error rate also increases. The simulation was done using BCH product codes, with component codes as indicated in the Figure.

It is interesting to notice that the slope of the BER curves increases as the rate of the code goes to 1 (see [22] and [21]). This graph also shows the uncoded simulation, which clearly shows how effective Turbo Block Codes are in decreasing the bit error rate.

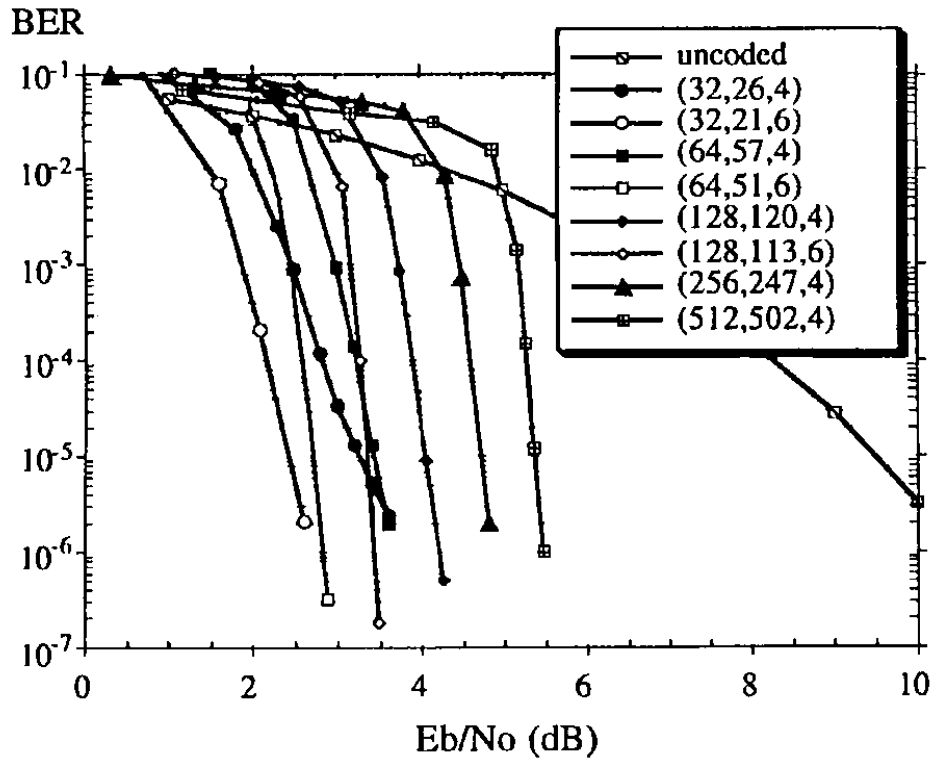

Fig 1.11 Performance of block Turbo codes for different component BCH codes. This simulation was done over an AWGN (Additive White Gaussian Noise) channel using QPSK modulation. The number of iterations is 4. (Figure taken from [22]).

Chapter 2

ASYMMETRIC SOURCES

In this Chapter we first introduce the concept of asymmetric sources. We then move on to defining and calculating the Shannon theoretical limits in this case. Finally, we conclude this Chapter by explaining the rationale for the proposed coding scheme in the case of non-uniform sources.

2.1 Definition

The output of an asymmetric source does not have the same distribution for zeros and ones. This means that the probability of the information bits, produced by the source, are not equal. Therefore, the entropy of a binary asymmetric source is less than one. Specifically, we will denote by p_0 the probability that the source generates bit 0. The source entropy is defined by:

$$H = -p_0 \log p_0 - p_1 \log p_1 .$$

2.2 Theoretical limits

In 1998, Shannon proved that it is possible to transmit information with arbitrarily low error probability through a noisy channel as long as the information rate is less than the channel capacity [27, 28]. The **Shannon limit** or **Shannon capacity** of a communications channel refers to the maximum rate of error-free data that can theoretically be transferred over the link.

Specifically for a channel subject to additive white Gaussian noise, the best possible achievable information rate R is given by:

$$R < C = \frac{1}{2} Log_2 \left(1 + \frac{2E_b}{N_0} R \right),$$

where

C = channel capacity (in bits per channel use),

R = Information rate (bits per channel use),

E_b = Energy per information bit,

$N_0/2$ = Power spectral density of the noise.

In the case of binary uniform sources $R = R_c$, where R_c is the code rate. However, for the case of non-uniform sources $R = R_c H$, where H is the entropy of the source [9]. Therefore, for an asymmetric source with entropy H transmitted through an AWGN channel using a code of rate R_c, the minimum signal to noise required for reliable communication is given by:

$$\frac{E_b}{N_0} > \frac{1}{2R_c H} \left(2^{R_c H} - 1 \right).$$

2.3 Joint Source -Channel Coding versus separated Source and Channel Coding

Although Shannon's information separation theorem points out that in a communications system we can optimize the source coder and the channel coder separately without sacrificing overall performance, this principle is only valid upon the assumption of infinitely long codewords (infinitely long delay). In other words, the Shannon separation principle only works in the limit, meaning that we need arbitrary large data set and no bound on coding delay. Joint source-channel coding can in fact improve coding efficiency in more realistic scenarios [7].

In our proposed scheme, instead of performing separated source and channel coding, we directly encode the source at the desired rate. The difference between this method and performing separated source and channel coding is that the source statistics have to be considered in the decoding. As an advantage with respect to the separate approach, no error propagation appears here. In addition, the complexity of the encoder is very low, and all the complexity moves to the decoding site. The decoder makes use of the *a priori* probabilities of the source to recover the transmitted input sequence. Furthermore, in many occasions the source and channel statistics do not need to be known at the decoder site, since they can be estimated jointly with the decoding process.

Chapter 3

CONVOLUTIONAL TURBO CODING FOR JOINT SOURCE-CHANNEL CODING OF NON-UNIFORM MEMORYLESS SORCES

In this Chapter, we review a joint source-channel coding approach for non-uniform memoryless sources using convolutional turbo codes. This scheme will be taken as a point of reference for the proposed block turbo coding system. The combination of source and channel coding is performed by a turbo code using an energy allocation scheme, properly designed to achieve good performance [7].

The rest of the Chapter is organized as follows: We first introduce the encoder modifications necessary to achieve performance close to the theoretical limit in the case of non-uniform sources, and show how to achieve the desired code rate for turbo codes. Finally we present simulation results.

3.1 Encoder for Non-Uniform Sources

The joint source-channel coding scheme considered here is based on the idea that symbols that are more likely to appear in the input sequence should be represented by less coded bits, and therefore, allocated less energy [6]. This asymmetric energy allocation scheme takes into account the *a priori* probability of the source to generate the sequence that will be transmitted over the noisy channel.

3.1.1 Energy allocation for systematic bits

If we represent 0 with $\sqrt{E_0}$ and 1 with $-\sqrt{E_1}$, the MAP decision criterion for an uncoded system is given by [7]

$$0 \text{ if } P_1 \cdot e^{\left(-\frac{1}{2\sigma^2}\left(x-\sqrt{E_1}\right)^2\right)} < P_0 \cdot e^{\left(-\frac{1}{2\sigma^2}\left(x+\sqrt{E_0}\right)^2\right)}$$

$$1 \text{ if } P_0 \cdot e^{\left(-\frac{1}{2\sigma^2}\left(x+\sqrt{E_0}\right)^2\right)} < P_1 \cdot e^{\left(-\frac{1}{2\sigma^2}\left(x-\sqrt{E_1}\right)^2\right)}.$$

The optimum energy allocation to minimize the probability of symbol-by-symbol errors, given the constraint $P_0E_0 + P_1E_1 = 1$ is $E_1 = P_0/P_1$ and $E_0 = P_1/P_0$ [7].

3.1.2 Energy allocation for coded bits

To allocate energy for the coded bits, we need to take into account the trellis structure of the convolutional encoders. We denote by E_0^c and E_1^c the energy to be assigned for the coded bits of the convolutional code with inputs 0 and 1 respectively. The average energy per symbol should stay constant (i.e. $P_0E_0^c + P_1E_1^c = 1$). We assign the energy for the coded symbols as:

$$E_0^c = \theta/P_0 \text{ and } E_1^c = (1-\theta)/P_1, \qquad (3.1)$$

where θ is a parameter which will be used to optimize the energy allocation.

3.2 Code Rate

In our simulation, we punctured the transmitted information bits to achieve the desired code rate. Puncturing the information bits is a standard process and could cause the code to become catastrophic if the proper measures are not taken.

Depending on the desired rate, different puncturing methods will be used to achieve the best results. In our case we are interested in code rates close to one. Since the rate of the mother code for turbo codes is usually 1/3, to achieve a code rate close to one, on average we need to puncture more than 2 out of 3 bits. The arrangement that is often favored, and the one we have used in our work, is to transmit most of the systematic bits from the first RSC encoder, and least of the parity bits from each encoder. Note that systematic bits are rarely punctured, since this degrades the performance of the code more dramatically. In our case, we achieved the best performance when we punctured one systematic bit after puncturing six parity bits. More specific details and resulting performance of joint-source-channel coding of memoryless sources using turbo codes can be found in [30], [31],[32],[13].[19] and [12].

3.3 Decoding Method

The idea of decoding modifications for non-uniform sources was first proposed in [13]. Specifically, we need to take into account the *a priori* probability of the input bits ($p_0, p_1 = 1 - p_0$) in the decoding process. Assuming $L(u_k)$ is the extrinsic information produced by decoder D, then the modified extrinsic information, $L_m(u_k)$, can be calculated by:

$$L_m(u_k) = L(u_k) + \log \frac{p_1}{p_0}.$$

In this decoding method we assume that p_1 is known at the decoder site. If this is not the case, it is possible to estimate this probability jointly with the decoding process.

3.4 Simulation Results

The Figure below shows the simulation of Turbo Codes for non-uniform memoryless sources over an AWGN channel. The simulation was done using the decoding scheme explained in this Chapter for rate R=0.93 and source probabilities p=0.1 and p=0.01. We achieved the best performance when we punctured one systematic bit after puncturing six parity bits. The gap between capacity and theoretical limit is 3.3 dB for probability p=0.1 and 3.1 dB for probability p=0.01.

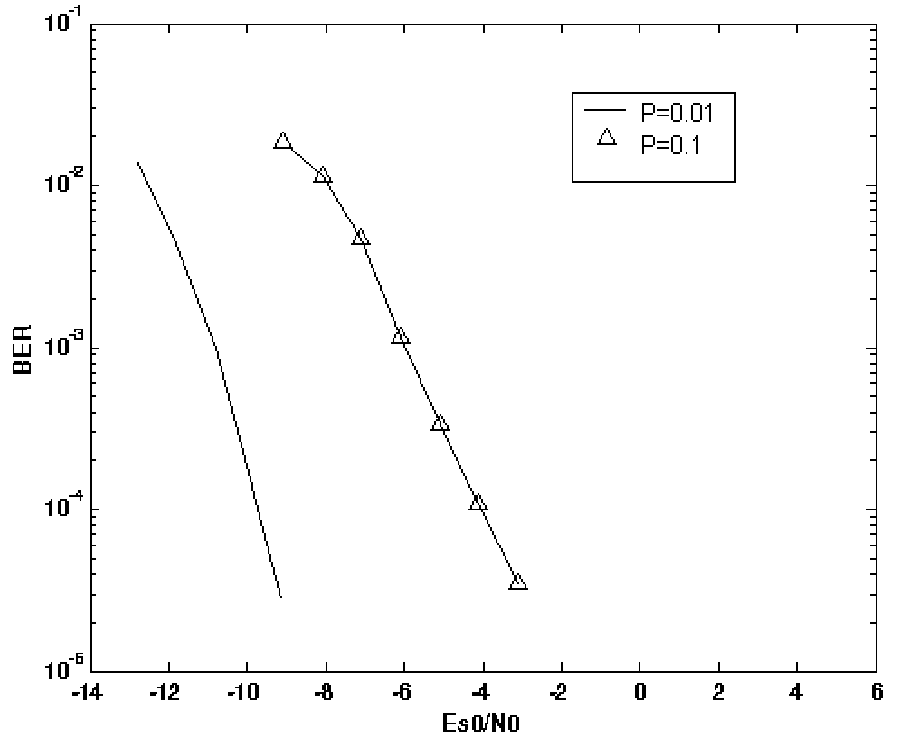

Figure 3.1: Turbo Codes Simulation for code rate R=0.93. This simulation was performed over an AWGN channel. The interleaver has length 16384 and spread 23. The code rate is R=0.93.

Chapter 4

BLOCK TURBO CODING FOR JOINT SOURCE-CHANNEL CODING OF NON-UNIFORM MEMORYLESS SOURCES

In this Chapter, we present different decoding methods to perform joint source-channel coding for block turbo codes with non-uniform memoryless sources. In order to do so, we will show how to modify the decoding algorithm when the *a priori* information is known. Since the decoder of standard block turbo codes makes use of different approximations, the decoding process can be modified in different ways by using the *a priori* information, which results in different decoding schemes for non-uniform sources.

As defined in Chapter 1, we will consider the transmission of a code using a linear block code on a Gaussian channel so that the received observation R is given by:

$$R = E + N ,$$

where E is the transmitted codeword and N the Gaussian noise.

4.1 Method 1

In this method we modify the calculation of the extrinsic information using *a priori* information. As explained in section 1.3, to consider the *a priori* probability in the decoding process of convolutional turbo codes, we modified the extrinsic information $L(U_k)$. The modified extrinsic information is obtained as:

$$L_m(U_k) = L(U_k) + \ln\left(\frac{p_1}{p_0}\right). \tag{4.1}$$

In block turbo codes, the extrinsic information $W(m)$ is calculated by subtracting the soft input from the soft output,

$$\left[R(m)\right] = [R] + \alpha(m)\left[W(m)\right] \tag{4.2}$$

Index m in R(m) and W(m) indicates that we are considering extrinsic information for the mth decoding iteration. The parameter $\alpha(m)$ is a scaling factor which takes into account the fact that the standard deviations of [R] and [W] are not the same.

The basic idea for the case of non-uniform sources is that *a priori* probability of the input bits $(p_0 \text{ and } p_1 = 1 - p_0)$ needs be considered in the decoding process. Specifically, $W'(m)$, the modified extrinsic information for the case of non-uniform sources can be calculated as:

$$W'(m) = \frac{\left(R(m) - R + \ln\left(\frac{p_1'}{p_0'}\right)\right)}{\alpha(m)}. \tag{4.3}$$

The probability of zeros, p_0', and ones, p_1', in the non-systematic bits and systematic bits will be different. For systematic bits $p_1' = p_1$, and $p_0' = p_0$. For non-systematic bits, p_1' and p_0' are equal to the ratio of ones and zeros in the coded bits.

Figure 4.1 shows the performance of the turbo block codes with constituent encoders BCH (256,247,4) using method 1. The code rate is 0.931 and BPSK is utilized over an AWGN channel. The number of iterations in this simulation is 4. The theoretical limits in this case are E_b/N_0 = 3.2,-3.4 and -11.9 dB for p_1=0.5, 0.1 and 0.01. Therefore, the gap between the performance obtained with this method and the theoretical limit is considerable, especially in the case of non-uniform sources.

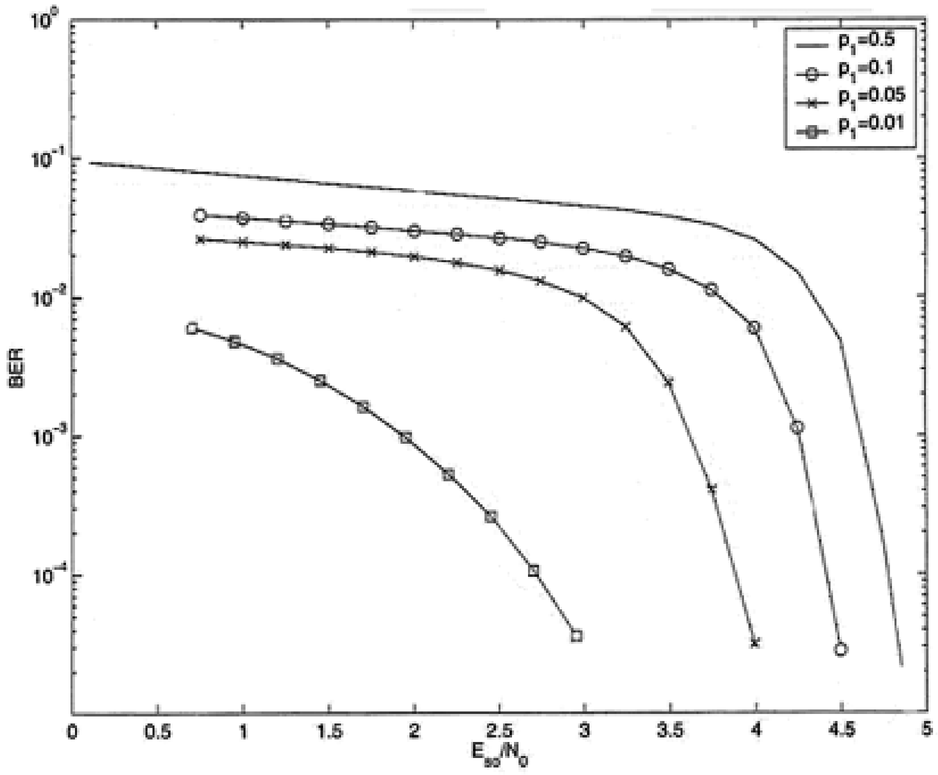

Figure 4.1: Performance of block turbo codes with constituent encoders BCH (256,247,4) when method 1 is applied. The code rate is 0.931 and BPSK is utilized over an AWGN channel. The number of iterations in this simulation is 4.

4.2 <u>Method 2</u>

In this method, we modify the way in which the soft output is calculated (section 1.3.3) by making use of the *a priori* source information. As discussed in Chapter 1, by using Maximum Likelihood decoding, the optimum decision, assuming equal probability for binary elements, can be calculated as [10]:

$$D = C_i \text{ if } \left|R - C_i\right| - 2\sigma^2 \ln\left(\mathrm{P}\left(E = C_i\right)\right) < \left|R - C_j\right| - 2\sigma^2 \ln(\mathrm{P}(E = C_j)). \quad (4.4)$$

To compute the reliability of decision d_j, where d_j is a component of decision $D = \{d_1, d_2, ...d_n\}$, required two codewords. It is obvious that soft decision D is one of these two codewords, so we must find the competing codeword of D (see [22]). Assuming C is the competing codeword of D, the soft output in the case of uniform sources is calculated as:

$$R_j = \left(\left|R - C\right|^2 - \left|R - D\right|^2\right) d_j / 4. \quad (4.5)$$

In the case of non-uniform sources, the soft output in the case of non-uniform sources can be calculated as:

$$R_j = \left(\left|R - C\right|^2 - \left|R - D\right|^2\right) d_j / 4 - \sigma^2 \ln\left(\mathrm{P_r}\left(C\right) / \mathrm{P_r}\left(D\right)\right), \quad (4.6)$$

where $\mathrm{P_r}\left(C\right)$ and $\mathrm{P_r}\left(D\right)$ are the calculated probabilities of codewords C and D using the *a priori* information.

Figure 4.2 shows the performance of turbo block codes with constituent encoders BCH (256,247,4) using method 2. The code rate is 0.931 and BPSK is utilized over an AWGN channel. The number of iterations in this simulation is 4. The theoretical limits in this case are $E_b / N_0 = 3.2$,-2.9 and -11.9 dB for p_1=0.5, 0.1 and 0.01. Although there is a minor improvement (about 0.5 dB) compared to method 1, the gap between the performance obtained with this method and the theoretical limit is still considerable, especially for non-uniform sources.

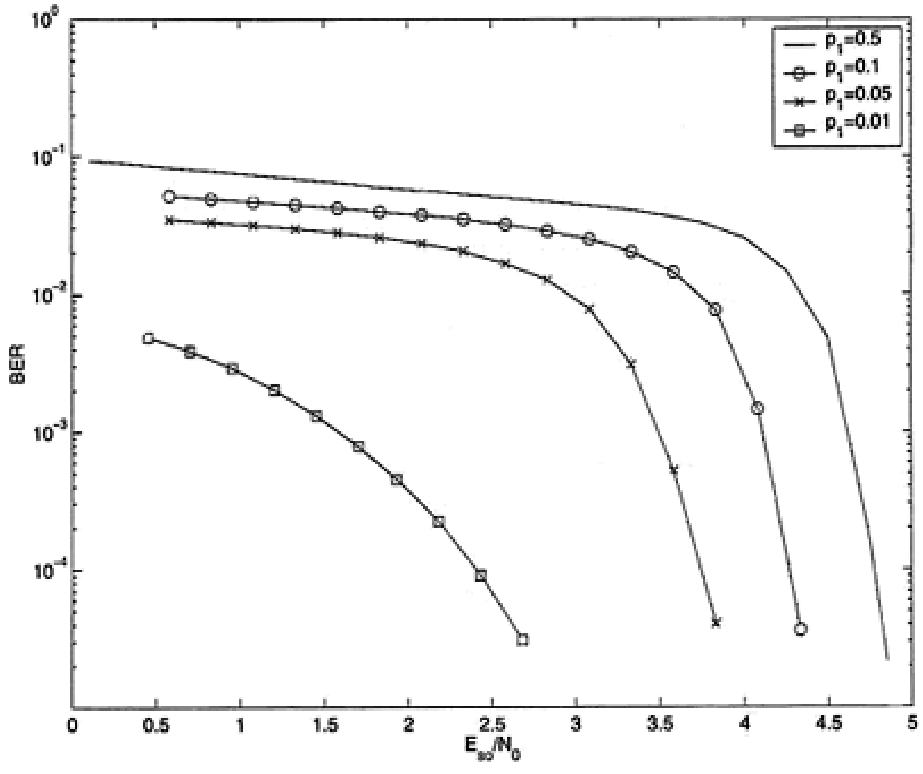

Figure 4.2: Performance of block turbo codes with constituent encoders BCH (256,247,4) when method 2 is applied. The code rate is 0.931 and BPSK is utilized over an AWGN channel. The number of iteration in this simulation is 4.

4.3 <u>Method 3</u>

In this method, we keep all the changes in method 2 and we also modify the Chase algorithm (section 1.3.2) by making use of the *a priori* source information. As we discussed in Chapter 1, we know that by using Maximum Likelihood decoding the optimum decision will be:

$$D = C_i \text{ if } \left|R - C_i\right| - 2\sigma^2 \ln\left(\mathrm{P}\left(E = C_i\right)\right) < \left|R - C_j\right| - 2\sigma^2 \ln(\mathrm{P}(E = C_j)).$$

Considering the Chase algorithm, at very high SNR, ML codeword D is located in the sphere of radius $(\delta - 1)$ centered on $Y = \left(y_1, y_2, \ldots, y_n\right)$ where $y_j = 0.5\left(1 + \mathrm{sgn}\left(r_j\right)\right)$ with a very high probability. The reliability of component y_j (which is used in the first step of the Chase algorithm see section 1.3.2) is defined using the log-likelihood ratio (LLR) of decision y_j [22]:

$$\Re\left(y_j\right) = \ln\left(\mathrm{P}(e_j = 1 \mid r_j) / \mathrm{P}(e_j = -1 \mid r_j)\right) \tag{4.7}$$

or

$$\Re\left(y_j\right) = 2\mathrm{r_j}/\sigma^2 . \tag{4.8}$$

Since the source is non-uniform, the *a priori* probability of the input bits (p_0 and $p_1 = 1 - p_0$) needs be considered in the decoding process. Then, we can calculate the relative reliability using:

$$\Re\left(y_j\right)=2\mathrm{r_j}/\sigma^2+\log\left(\frac{\mathrm{p_1}}{\mathrm{p_0}}\right) \tag{4.9}$$

In other words, for the case of non-uniform sources, the reliability of component y_j is calculated by considering *a priori* probability of the input bits. The basic idea is that the input bit that has higher probability will result in higher reliability.

Figure 4.3 shows the performance of turbo block codes with constituent encoders BCH (256,247,4) using method 3. The code rate is 0.931 and BPSK is utilized over an AWGN channel. The number of iterations in this simulation is 4. There are some improvements with respect to the previous methods, but there is still a huge gap between capacity and theoretical limit. For example for p_1=0.01 the gap between capacity and theoretical limit is greater than 14 dB. Compared to the two previous methods 1 and 2, method 3 results in 1 dB to 2 dB improvement.

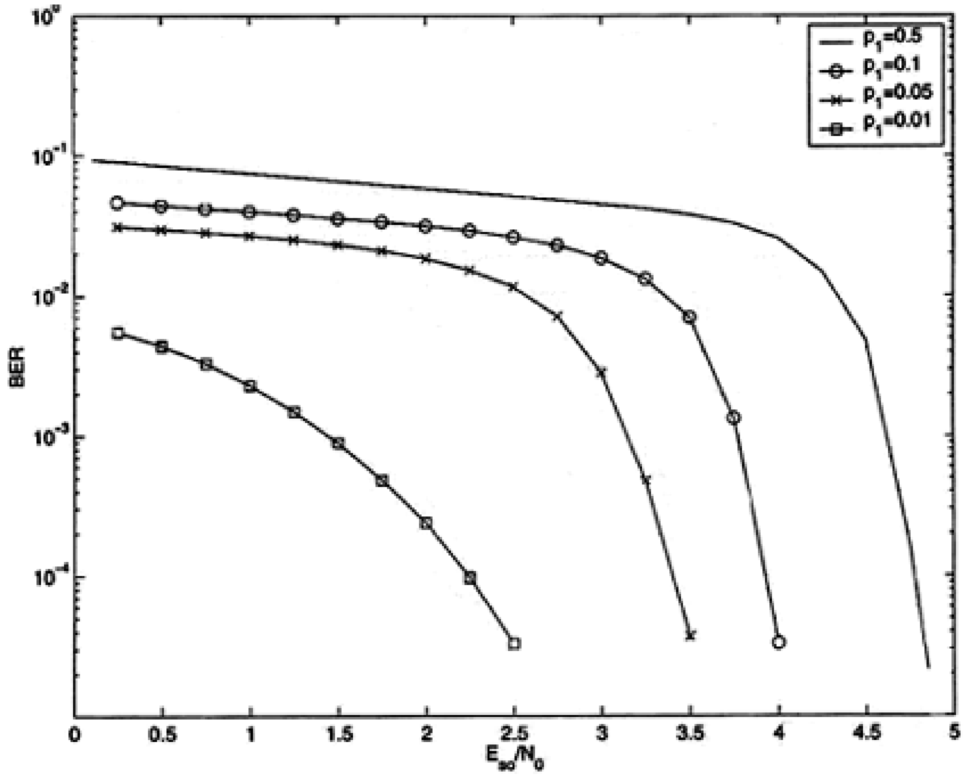

Figure 4.3: Performance of block turbo codes with constituent encoders BCH (256,247,4) when method 3 is applied. The code rate is 0.931 and BPSK is utilized over an AWGN channel. The number of iteration in this simulation is 4.

4.4 Method 4

In this approach, we propose a source controlled channel coding scheme based on the idea that symbols that are more likely to appear in the input sequence should be represented by less coded bits and therefore allocated less energy [6].

As we explained in Chapter 3, the decision criterion for systematic bits is given by:

$$0 \text{ if } P_1 \cdot e^{\left(-\frac{1}{2\sigma^2}\left(x-\sqrt{E_1}\right)^2\right)} < P_0 \cdot e^{\left(-\frac{1}{2\sigma^2}\left(x+\sqrt{E_0}\right)^2\right)}$$

$$1 \text{ if } P_0 \cdot e^{\left(-\frac{1}{2\sigma^2}\left(x+\sqrt{E_0}\right)^2\right)} < P_1 \cdot e^{\left(-\frac{1}{2\sigma^2}\left(x-\sqrt{E_1}\right)^2\right)}.$$

To minimize the probability of symbol-by-symbol errors, given the constraint $P_0E_0 + P_1E_1 = 1$ we use energy allocation $E_1 = \frac{P_0}{P_1}$ and $E_0 = \frac{P_1}{P_0}$.

Energy allocation for non-systematic bits can be done as follows. Before allocating energy to non-systematic bits the probability of ones and zeros for non-systematic bits needs to be calculated. The probability of zeros and ones will be almost 0.5 for codes with higher rates. In any case, after calculating the probability of zeros and ones for non-systematic bits, we can use the same MAP criterion detection as systematic bits to allocate energy for non-systematic bits (changing parameter θ in Equation (3.1) does not change performance in a significant manner).

Figure 4.4 shows the performance of turbo block codes with constituent encoders BCH (256,247,4) using method 4. The code rate is 0.931 and BPSK is utilized over an AWGN channel. The number of iterations in this simulation is 4. This Figure shows simulation for p_1=0.5, p_1=0.1 and p_1=0.01. There is still a big gap between capacity and theoretical limit (see table 4.1) but compared to previous methods, for the case p_1=0.01 the gap has decreased considerably.

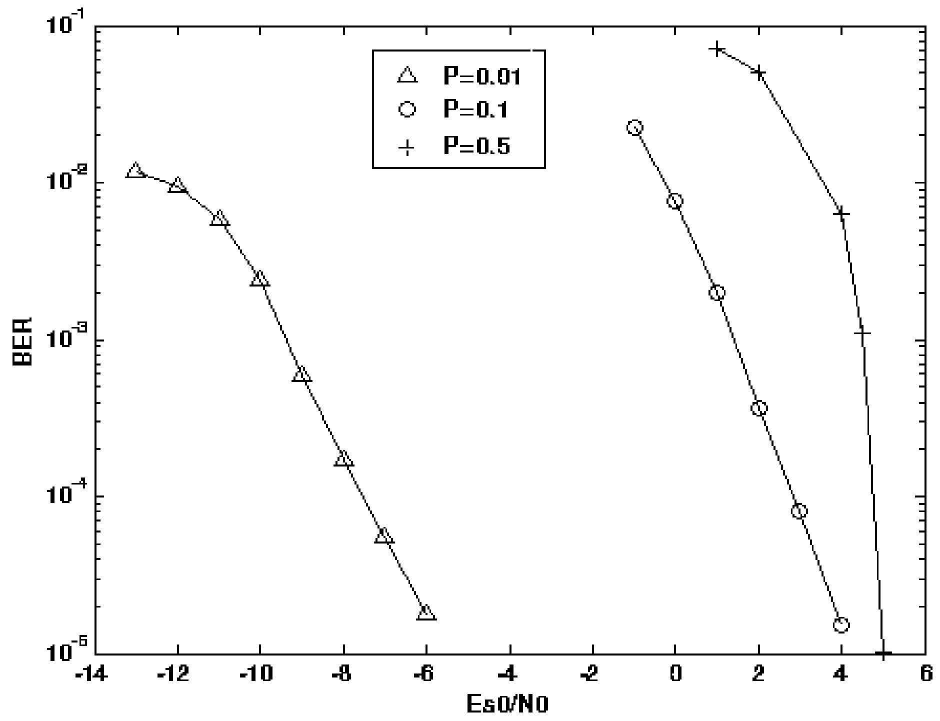

Figure 4.4: Performance of block turbo codes with constituent encoders BCH (256,247,4) when method 4 is applied. The code rate is 0.931 and BPSK is utilized over an AWGN channel. The number of iteration in this simulation is 4.

4.5 Method 5

This method combines the two previous methods, method 3 and method 4. In other words, we use an unequal energy strategy, modifying the way in which the soft output is calculated, and we also use the *a priori* information in the Chase algorithm.

Figure 4.5 shows the performance of turbo block codes with constituent encoders BCH (256,247,4) using method 5. The code rate is 0.931 and BPSK is utilized over an AWGN channel. The number of iterations in this simulation is 4. This Figure shows simulation for p_1=0.5, p_1=0.1, p_1=0.01, and p_1=0.005. Compared to method 4, the gap between capacity and theoretical limit has decreased in almost 2 dB.

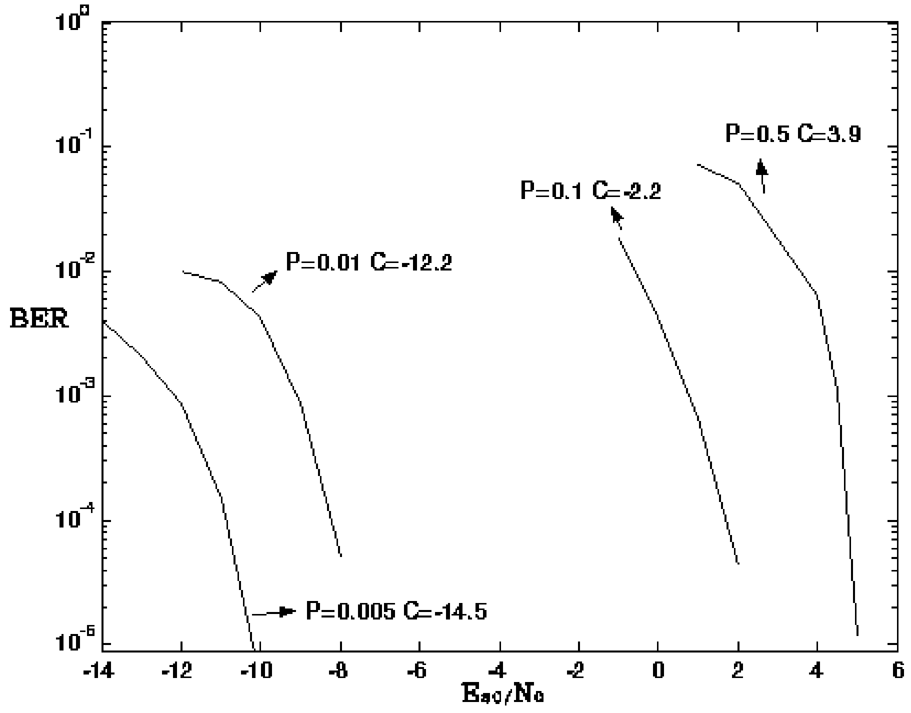

Figure 4.5: Performance of block turbo codes with constituent encoders BCH (256,247,4) when method 5 is applied. The code rate is 0.931 and BPSK is utilized over an AWGN channel. The number of iteration in this simulation is 4.

4.6 Performance Comparison

As we have seen throughout this Chapter, the performance of block turbo codes for non-uniform sources improves dramatically by going from decoding method 1 to decoding method 5 (over 10 dB improvement in method 5 compared to method 1, for the case of $p_1 = 0.01$). Table 4.1 shows the gap between capacity and theoretical limit for the five different decoding methods explained in this Chapter, as well as convolutional turbo codes defined in Chapter 3. As we can see in the table, for the case of non-uniform sources, convolutional turbo codes perform slightly better than proposed method 5, which is the best approach for block turbo codes (1.1 dB for the case of $p_1 = 0.01$ and 0.7 dB for $p_1 = 0.1$).

Table 4.1: Gap to Theoretical Limit (dB) for rate R=0.93 codes applied on non uniform sources and AWGN channels. Convolutional turbo codes are compared with the 5 decoding methods for block turbo codes.

p_1	**Method 1**	**Method 2**	**Method 3**	**Method 4**	**Method 5**	**CTC**
0.5	1.3	1.3	1.3	1.3	1.3	3.9
0.1	7.4	6.9	6.3	6.1	4.0	3.3
0.01	14.9	14.6	14.2	6.1	4.2	3.1

Chapter 5

FINAL REMARKS

Our goal in this research was to improve the performance of block turbo codes as we increased the non-uniformity of the source. As we discussed in the thesis, since block turbo codes perform very well for rates close to one in the case of uniform sources, we wanted to corroborate if this was also the case for non-uniform sources. The idea was to investigate if block turbo codes could outperform convolutional turbo codes in this context.

The results of this research show that block turbo codes are not able to outperform convolutional turbo codes for any rate in the case of non-uniform sources. However, it is important to remark that the decoding algorithm in block turbo codes is suboptimal in many aspects, with several details defined in an ad-hoc manner. Future research investigating how to choose the weight and the reliability features in the case of non-uniform sources could lead to improved performance gains.

REFERENCES

[1] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, “Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate,” *IEEE Trans. on Inform. Theory*, pp. 284-287, March 1974.

[2] S. Benedetto and G. Montorsi, “Design of Parallel Concatenated Convolutional Codes,” *IEEE Trans. Commun.*, pp. 591–600, May 1996.

[3] S. Benedetto and G. Montorsi, “Unveiling Turbo Codes: Some Results on Parallel Concatenated Coding Schemes,” *IEEE Trans. on Inform. Theory*, pp. 409–428, March 1996.

[4] C. Berrou and A. Glavieux, “Near Optimum Error Correcting Coding and Decoding: Turbo –Codes,” *IEEE Trans. Commun.,* pp. 1261-1271, October 1996.

[5] C. Berrou and A. Glavieux and P. Thitimajshima, “Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes,” *Proc. IEEE ICC’93*, pp. 1064–1070, May 1993.

[6] F. Cabarcas, R. D. Souza, and J. García-Frías, “Source-Controlled Turbo Coding of Non-Uniform Memoryless Sources Based on Unequal Energy Allocation,” *IEEE Proc. ISIT’04*, June 2004.

[7] F. Cabarcas, “Turbo Coding/Decoding Modifications for Improved Performance in Non-Standard Environments,” M.S. Thesis, ECE Dept., University of Delaware.

[8] D. Chase, “A Class of Algorithms for Decoding Block Codes with Channel Measurement Information,” *IEEE Trans. on Inform. Theory*, pp. 170- 182, January 1972.

[9] T. M. Cover, ”Elements of Information Theory,” Wiley Series in Telecomm, 1991.

[10] A. Elbaz, R. Pyndiah, B. Solaiman, and O. A. Sab, “Iterative Decoding of Product Codes with A Priori Information over a Gussian Channel for Still Image Transmission,” *Proc. IEEE* Globecom’99, pp. 2602-2606, 1999.

[11] P. Elias, "Error-Free Coding," *IRE Trans. Inform. Theory*, pp. 29-37, September 1954.

[12] J. García-Frías and J. D. Villasenor, "Combining Hidden Markov Source Models and Parallel Concatenated Codes," *IEEE Commun. Letters*, pp. 111-113, July 1997.

[13] J. Hagenauer, "Source-Controlled Channel Decoding," *IEEE Trans. Commun.*, pp. 2449–2457, September 1995.

[14] J. Hagenauer, E. Offer, and L. Papke, "Iterative Decoding of Binary Block and Convolutional Codes," *IEEE Trans. on Info. Theory,* pp. 429–445, March 1996.

[15] L. Hanzo, T.H. Liew, and B.L. Yeap, "Turbo Coding, Turbo Equalisation and Space-Time Coding," John Wiley, August 2002.

[16] R. Johannesson, "Some Rate 1/3 and 1/4 Binary Convolutional Codes with an Optimum Distance Profile," *IEEE Trans. on Inform. Theory*, pp. 281-283, 1977.

[17] R. Johannesson, "Some Long Rate One-Half Binary Convolutional Codes with an Optimum Distance Profile," *IEEE Trans. on Inform. Theory*, pp. 629-631, September 1976.

[18] F. J. Macwilliams and N. J. A. Sloane, "The Theory of Error Correcting Codes," North-Holland, 1978.

[19] P. Mitran and J.Bajcsy, "Turbo Source Coding: A Nosie-Robust Approach to Data Compression," *Proc. IEEE DCC'02*, p.465, April 2002.

[20] A. Morello, G. Montorosi, and M. Visintin, "Convolutional and Trellis Coded Modulations Concatenated with Block Codes for Digital HDTV," *Int. Workshop Digital Commun.*, pp. 237-250, September 1993.

[21] H. Nickl, J. Hagenauer, and F. Burkert, "Approching Shannon's Capacity Limit by 0.27 dB Using Simple Hamming Codes," *IEEE Trans. Commun.* pp. 130-132, September 1997.

[22] R. M. Pyndiah, "Near-Optimum Decoding of Product Codes: Block Turbo Codes," *IEEE Trans. Commun.,* pp.1003-1010, August 1998.

[23] R. Pyndiah, "Iterative Decoding of Product Codes: Block Turbo Codes," *Proc. IEEE Int. Symp. Turbo Codes & Related Topics*, pp. 71-79, September 1997.

[24] R. Pyndiah, A. Glavieux, A. Picart, and S. Jacq, "Near Optimum Decoding of Product Codes," *Proc. IEEE GLOBECOM'94*, pp. 339-343, November-December 1994.

[25] S. M. Reddy, "On Decoding Iterated Codes," *IEEE on Trans. Inform. Theory*, pp. 624-627, Sept 1970.

[26] P. Robertson and E. Villebrun and P. Hoher, "A Comparison of Optimal and Sub-Optimal MAP Decoding Algorithms Operating in the Log Domain," *Proc. IEEE ICC'95*, pp. 1009–1013, June 1995.

[27] C. E. Shannon, "A Mathematical Theory of Communication (part 1)," *Bell Syst. Tech.J.*, pp. 379–423, 1948.

[28] C. E. Shannon, "A Mathematical Theory of Communication (part 2). *Bell Syst. Tech.J.*, pp. 623–656, 1948.

[29] S.G. Wilson, "Digital Modulation and Coding," Prentice-Hall, Englewood Cliffs, 1996.

[30] G.-C. Zhu and F. Alajaji, "Design of Turbo Codes for Non-Equiprobable Memoryless Sources," *Proc. Allerton'01*, pp. 1253-1262, October 2001.

[31] G.-C. Zhu and F. Alajaji, "Turbo Codes for Non-Uniform Memoryless Sources over Noisy Channels," *IEEE Commun. Letters*, pp. 64–66, February 2002.

[32] G.-C. Zhu, F. Alajaji, J. Bajcsy, and P. Mitran, "Non-Systematic Turbo Codes for Non-Uniform i.i.d. Sources over AWGN Channels," *Proc. CISS*, pp. 1-5, March 2002.