Technical University of Crete
School of Electrical and Computer Engineering

Turbo Coding

Supervisor :
Assoc. Prof. George N. Karystinos
Committee members :
Prof. Athanasios Liavas
Assoc. Prof. Aggelos Bletsas

Abstract

Turbo Codes are a class of high-performance error-correcting codes and they were the first practical codes to achieve performance close to Shannon limit. Turbo codes are used in both 3G and 4G mobile communication networks, WiMAX, and (deep space) satellite communications as well as other applications where designers seek to achieve reliable information transfer over the bandwidth or latency constrained communication links in the presence of data-corrupting noise. In this thesis, we examine the basic principles behind channel coding and Turbo Coding and the essential parts of Turbo encoding and decoding algorithms which are used in several applications. Furthermore, we examine how the size of the interleaver, for both block and pseudorandom interleavers, affects the overall performance of the code.

Acknowledgments

I would like to thank my family for their support and encouragement. I would also like to thank my supervisor, Prof. Georgios N. Karystinos, for his guidance and cooperation throughout this work. Finally, I would like to thank my friends for their support all these years.

Contents

List of Figures

Chapter 1

Introduction

In telecommunication field, channel coding is a technique used for controlling errors in data transmission over unreliable or noisy channels. The central idea is that the sender encodes the message in a properly designed, redundant way by using a channel encoder which implements an error correction code. The redundant bits which are added to each message provide the code with the capability of combating the channel noise and, hence, allow the receiver to detect a limited number of errors that may occur during transmission.

1.1 General Informations

A common diagram of the general structure of a digital telecommunication system is :

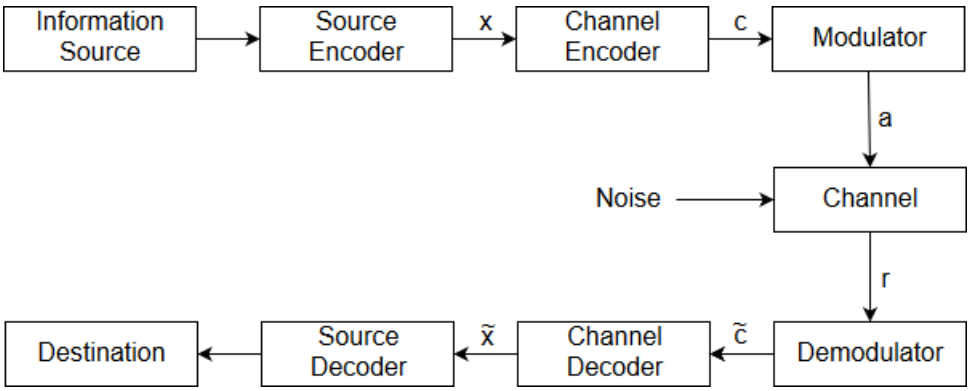

Figure 1.1: Digital Telecommunication System

The role of channel encoder is to protect the bits which are transmitted over a channel from noise, distortion and interference. Generally, the channel encoder transforms the input sequence **x**, which is called information sequence, into a discrete, alternative encoded sequence **c** possessing redundancy, whose role is to provide immunity from the various channel impairments. This encoded output sequence, **c**, is called codeword. In most instances, **c** is also a binary sequence, although in some applications nonbinary codes can be used. As it is mentioned before, the goal of the design and the implementation of channel encoders/decoders is to combat the noisy

environments in which codewords are transmitted. Hence, the redundancy provided by the channel encoder is introduced in a structured way, precisely to provide error control capability.

On the other hand, the role of the channel decoder is to recover the initial input to the channel encoder from the channel output in spite of the presence of noise, distortion and interference in the received codeword. That is, channel decoder transforms either the received sequence $\mathbf{r}$ or the estimated codeword $\tilde{\mathbf{c}}$ into a binary sequence $\tilde{\mathbf{x}}$, which is called estimated information sequence. Ideally, $\tilde{\mathbf{x}}$ will be a replica of the transmitted information sequence $\mathbf{x}$, although the noise may cause some decoding errors. The pair of channel encoder and decoder forms a (n,k) channel code, where k and n correspond to the length of the input and the output vectors of the channel encoder, respectively .

A well-defined channel coding schema has the following properties : it is easy to decode, it has large minimum distance d_{min}, and the number of codewords at distance d_{min} from any other codeword is small.

A class of both channel encoder and decoder are Turbo Codes. Turbo Codes, that were first published in 1993 from Berrou, Glavieux and Thitimajshima, are a class of high-performance error-correcting codes and they were the first practical codes to closely approach the channel capacity. Turbo codes are used in both 3G and 4G mobile communication networks, WiMAX, and in (deep space) satellite communications as well as other application where designers seek to achieve reliable information transfer over the bandwidth or latency constrained communication links in the presence of data-corrupting noise.

The performance of a coded communication system is measured by its probability of decoding errors, which is called error probability, and its coding gain over the uncoded system that transmits information sequences at the same rate. Typically, the best code designs contain a large amount of structure, either algebraic or topological, as is the case with most convolutional codes, which can be represented using trellis or tree diagrams. The structure is a key component of the code design, since it can be used to guarantee good minimum distance properties for the code and since particular decoding methods are based on this structure. In fact, one can say generally that the more structure a code contains, the easier it is to decode; however, structure does not always result in the best distance properties for a code, and most highly structured code designs usually fall far short of achieving the performance promised by Shannon. Turbo coding succeeds in achieving a randomlike code design with enough structure to allow for an efficient iterative decoding method. Because of this feature, these codes have exceptionally good performance, particularly at moderate BERs and for large block lengths. In fact, for essentially any code rate and information block lengths greater than about 10^4 bits, Turbo Codes alongside with iterative decoding schemes can achieve BERs as low as 10^{-5} at SNRs within 1 dB of the Shannon limit, that is, the value of $\frac{E_b}{N_0}$ for which the code rate equals channel capacity.

Another important feature of turbo codes is their basic structure. They are composed of two or more simple constituent codes along with a pseudorandom interleaver. Because the interleaver is part of the code design, a complete maximum likelihood decoder for the entire code would be prohibitively complex; however, because more than one code is used, it is possible to employ simple SISO decoders for each constituent code in an iterative fashion, in which the soft-output values of one decoder are passed to the other, and vice versa, until the final decoding estimate is obtained. This simple, suboptimum, iterative decoding approach almost always converges to the maximum likelihood solution.

In summary, turbo coding consists of two fundamental ideas: a code design that produces

a code with randomlike properties, and a decoder design that makes use of soft-output values and iterative decoding.

Although, turbo codes suffer from two disadvantages: a large decoding delay, owing to the large block lengths and many iterations of decoding procedure which are required for near-capacity performance, and significantly weakened performance at BERs below 10^{-5} owing to the fact that the codes have a relatively poor minimum distance, which manifests itself at very low BERs. This phenomenon is called error floor and it is due to unusual weight distribution of turbo codes. The large delay appears to make turbo codes unsuitable for real-time applications such as voice transmission and packet communication in high speed networks. It is possible, though, to trade delay for performance in such a way that turbo codes may be useful in some real-time applications involving block lengths on the order of a few thousands, or even a few hundred bits. Interleavers can be designed to improve the minimum distance of the code, thus, lowering the error floor. Also, an outer code, or a second layer of concatenation, can be used with turbo code to correct many of the errors caused by the small minimum distance, at a cost of small decrease in overall rate.

1.2 Basic Terms

1.2.1 Mathematical Notation

Throughout the analysis of channel coding, the following mathematical notation will be used. Let $\mathcal{F}_2$ be the set with only two elements in it, 0 and 1. In this field, the arithmetic operations are defined as :

$$\begin{array}{cccc} 0+0=0 & 0+1=1 & 1+0=1 & 1+1=0 \\ 0*0=0 & 0*1=0 & 1*0=0 & 1*1=1 \end{array}$$

Let $\mathcal{F}_2^n$ denotes the vector space of n-tuples and $\mathcal{F}_2^n$ denotes the vector space of k-tuples of elements of $\mathcal{F}_2$, respectively. For an (n,k) binary code and a telecommunication system as shown in figure (1.1), the input of channel encoder $x \in \mathcal{F}_2^k$ and the output of channel encoder $c \in \mathcal{F}_2^n$. A channel encoder adds redundancy to the information sequence by assigning to each message of k bits, a longer message of n bits which is called codeword. Hence, an (n,k) binary code is a set of 2^k distinct points in $\mathcal{F}_2^n$.

1.2.2 Codeword

A code with input vectors of length k and output vectors of length n constitutes a (n,k) code. For a (n,k) code, the set of all possible codewords is defined as the set of n-length vectors which are produced by the encoder for every input vector $x \in \mathcal{F}_2^k$. Each codeword of a binary code has length of n. Thus, generally, there are 2^n possible codewords. But, only a part of them forms the codeword set of the code. The input x of the encoder is a k-length vector, that is, there are 2^k possible input vectors. Additionally, a codewords should match to exactly one input vector x. Hence, only a part of 2^k discrete vectors from the total 2^n vectors which constitute the vector space $\mathcal{F}_2^n$ are the acceptable codewords of a (n,k) code. That is, the set of codewords of a (n,k) code forms a subspace of the vector space $\mathcal{F}_2^n$.

1.2.3 Linear Code

A code is called linear, if any linear combination of two (or more) codewords is, also, a codeword. That is, if x and y are codewords, then

$$z = ax + by \qquad \forall a, b \in \mathbb{R} \tag{1.1}$$

is also a codeword. Hence, for a linear code, the all-zero vector is always part of its codeword set, since $z = x + (-1) * x = a - a = 0$.

1.2.4 Rate of a Code

The rate of an (n,k) code is defined as :

$$R = \frac{k}{n} \tag{1.2}$$

where k is the number of code's input bits at each time unit and n is the number of code's output bits at the same time unit. The rate of a code expresses the average number of information bits carried by each encoded bit. For any code (n,k), n is greater to k ($k < n$), thus, the rate of a code is less or almost equal to unity. When $k < n$, $n - k$ redundant bits are added to each message to form a codeword. These redundant bits carry no new information and their main function is to provide the code with the capability of detecting and correcting transmission errors caused by the channel noise or interference.

1.2.5 Hamming Distance

The Hamming distance (denoted by d_H) of any two vectors $\mathbf{x}, \mathbf{y} \in \mathcal{F}_2^k$ is defined as the number of elements where the two vectors differ.

1.2.6 Euclidean Distance

The Euclidean distance (denoted by d_E) of any two vectors $\mathbf{x}, \mathbf{y} \in \mathcal{F}_2^k$ is defined as :

$$d_E(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{k} (x_i - y_i)^2} \tag{1.3}$$

1.2.7 Weight of a Codeword

The weight of a vector $\mathbf{x} \in \mathcal{F}_2^k$ is equal to the number of its nonzero elements. Alternatively, the weight of a vector x can be defined as the Hamming distance of $\mathbf{x}$ compared with the all-zero vector :

$$w(\mathbf{x}) = d_H(\mathbf{x}, \mathbf{0}) \tag{1.4}$$

1.2.8 Minimum Distance

The minimum distance of an (n,k) code is defined as :

$$d_{min} = \min_{i \neq j} d_H(\mathbf{x_i}, \mathbf{x_j}) \qquad \forall \mathbf{x_i}, \mathbf{x_j} \in \mathcal{X} \tag{1.5}$$

where $d_H(\mathbf{x_i}, \mathbf{x_j})$ is the Hamming distance of the vectors $\mathbf{x_i}$ and $\mathbf{x_j}$. For a linear code, the minimum distance of its codewords can be computed as :

$$\begin{aligned} d_{min} &= \min_{i \neq j} d_H(\mathbf{x_i}, \mathbf{x_j}) = \min_{i \neq j} d_H(\mathbf{x_i} - \mathbf{x_j}, \mathbf{x_j} - \mathbf{x_j}) \\ &= \min_{\mathbf{x}} d_H(\mathbf{x}, \mathbf{0}) = \min_{\mathbf{x}} w(\mathbf{x}), \ \forall \mathbf{x} \neq \mathbf{0} \end{aligned} \tag{1.6}$$

1.2.9 Weight Distribution

The weight distribution of a code is important for computing its error performance. We define A_i as the number of code sequences of weight i. Thus, as weight distribution of a code is defined the set $\{A_0, A_1, \dots, A_i, \dots\}$. Especially, for a linear code, its weight distribution can be defined as $\{A_{d_{min}}, A_{d_{min}+1}, \dots\}$, where d_{min} denotes the minimum distance of the linear code. Thus, we can conclude that for any linear code $A_1 = A_2 = \cdots = A_{d_{min}-1} = 0$.

It is convenient to represent the weight distribution as a polynomial :

$$A(z) = A_0 + A_1 z + A_2 z^2 + \cdots + A_n z^n \tag{1.7}$$

This polynomial is called the weight enumerator. The weight enumerator is the z-transform of the weight distribution sequence.

1.2.10 Memoryless Channel

As memoryless channel is defined the channel in which the output r_n at the n-th symbol time depends only on the current input s_n. In case of the combination of an M-ary input modulator, the physical channel and a Q-ary output demodulator can be modeled as a discrete memoryless channel (DMC). A DMC is completely described by a set of transition probabilities $P(j|i), 0 \leq i \leq M-1$ and $0 \leq i \leq M-1$, where i represents a modulator input symbol, j represents a demodulator output symbol and $P(j|i)$ is the probability of receiving j given that i was transmitted. Thus, given the input of the channel at time n, channel's output is statistically independent of the outputs at other times and the likelihood function $p(\mathbf{r}|\mathbf{s})$ can be factored as :

$$p(\mathbf{r}|\mathbf{s}) = p(r_1, r_2, \dots, r_m | s_1, s_2, \dots, s_m) = \prod_{i=1}^{m} p(r_i|s_i) \tag{1.8}$$

Memoryless channels are also called random-error channels and the codes which are designed for those channels are called random-error correcting codes.

1.2.11 Additive White Gaussian Channel

Additive White Gaussian Channel (AWGN) is a basic model used in information theory to mimic the effect of many random processes that occur in nature. The modifiers denote specific characteristics :

- Additive: because it is added to any noise that might be intrinsic to the information system
- White: refers to the idea that it has uniform power across the frequency band for the information system.
- Gaussian: because it has a normal distribution in the time domain with an average time domain value of zero.

AWGN model is used as a channel model in which the only impairment to communication is the linear addition of white noise with a constant spectral density and a Gaussian distribution of amplitude.

Suppose that the input to the modulator consists of symbols selected from a finite and discrete alphabet $X = \left[x_1, \ldots, x_{M-1}\right]$ and the output of the demodulator is left unquantized. In this case, the combination of the modulator, the physical channel and the demodulator results is a discrete-input, continuous-output channel. The channel output is a random variable. If the physical channel is subject only to AWGN with zero mean and one-side PSD N_0, then the channel output is a Gaussian random variable with zero mean and variance $\sigma^2 = \frac{N_0}{2}$. In this case, we obtain a discrete-input, continuous-output memoryless channel Gaussian channel. This channel is completely characterized by a set of M conditional probability density functions, $p(y|x), \forall x \in \{0, \ldots, M-1\}$.

A special case of the AWGN channel, which was described above, is the Binary-Input AWGN (BI-AWGN). For the BI-AWGN, the code bits are mapped to the channel inputs as $x_i = (-1)^{v_i} \in \{\pm 1\}$ so that $x_i = +1$ when $v_i = 0$. The BI-AWGN is completely characterized by the channel transition probability density function $p(y_i|x_i)$ given by :

$$p(y_i|x_i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[\frac{-(y_i - x_i)^2}{2\sigma^2}\right] \tag{1.9}$$

where σ^2 is the variance of the zero-mean Gaussian noise sample n_i that the channel adds to the transmitted value x_i, so that $y_i = x_i + n_i$

1.2.12 Binary Symmetric Channel (BSC)

At a sufficiently coarse level of detail, the modulator/demodulator system along with the additive AWGN channel can be viewed as a solid channel. This group of the modulator, the channel, and the detector collectively constitutes a channel which accepts bits at the input and emits bits at the output. This end-to-end system forms a BSC channel. The BSC crossover (transition) probability, p_e, is just the uncoded BPSK bit error probability for equally likely signals given by :

$$p_e = Q\left(\sqrt{\frac{2E_b}{N_0}}\right) \tag{1.10}$$

where

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty \exp^{\frac{-y^2}{2}} dy, x \geq 0 \tag{1.11}$$

is the complementary error function of Gaussian statistics. The above equation is used in evaluation of the error performance of a code. Both AWGN and BSC are memoryless channels.

1.2.13 Error Correction Capability

The error correction capability of any given code is equal to :

$$\left\lfloor \frac{d_{min} - 1}{2} \right\rfloor \tag{1.12}$$

This quantity measures the maximum number of errors which are introduced at the received codeword and the code can detect and correct successfully. Thus, it sets the lower achievable BER.

1.2.14 Constraint Length

As constraint length of a code we define the number of inputs (current input and previous) which are used by the code in order to produce its codewords. The constraint length may also be given as the number of memory elements, m, for convolutional codes or as the maximum possible states of the encoder (typically 2^m).

1.2.15 Coding Gain

The coding gain is defined as the reduction in the SNR required to achieve a specific error probability for a coded communication system compare to an uncoded system. Although, there is a threshold, especially for low values of SNR, below which the coding gain becomes negative. This SNR is called coding threshold.

1.2.16 Bandwidth Expansion

Two important and related parameters in any digital communication system are the speed of the information transmission and the bandwidth of the channel. Because one encoded symbol is transmitted every T seconds, the symbol transmission rate (baud rate) is $\frac{1}{T}$, In a coded system with rate $R = \frac{k}{n} < 1$, k information bits correspond to the transmission of n symbols. Thus, the information transmission rate (data rate) is equal to $\frac{R}{T}$ bits per second. In addition to signal modulation due to the effects of noise, most communication channels are subject to signal distortion owing to bandwidth limitations. To minimize the effect of this distortion on the detection process, the channel should have bandwidth W of at least $\frac{T}{2}$ Hz. In an uncoded system (R=1), the data rate $\frac{1}{T} = 2W$ is therefore limited by the channel bandwidth. In a binary coded system, with a code rate $R < 1$, the data rate $\frac{R}{T} = 2W$ is reduced by the factor R compared with an uncoded system. Comparing the data rate of an uncoded and a encoded system :

$$\frac{R}{T} = 2W \Leftrightarrow \frac{1}{T} = \frac{2W}{R} = 2W^{'} \Leftrightarrow W^{'} = \frac{W}{R} \tag{1.13}$$

where $W^{'}$ and W are the required bandwidth of the encoded and the uncoded system, respectively. Due to the fact that the rate $R < 1 \Leftrightarrow \frac{1}{R} > 1$, the required bandwidth of the encoded system is greater than the bandwidth of the uncoded one. Hence, to maintain the same data rate as an uncoded system, the encoded system requires bandwidth expansion by a factor of $\frac{1}{R}$. Thus, the coded system requires a larger channel bandwidth for signal transmission than the uncoded system. This is because redundant symbols must be added to the transmitted information sequence to combat channel noise. Therefore, coding gain is achieved at the expense

of bandwidth expansion. To overcome this problem, coding must be used in conjunction with bandwidth-efficient multilevel modulation (M-ary modulation with $M > 2$).

If coding and modulation are combined properly and designed as a single entity, coding gain can be achieved without bandwidth expansion. Such combination of coding and modulation is called coded modulation. The basic concept of coded modulation is to encode information symbols onto an expanded modualtion signal set (relative to that needed for the uncoded modulation). This signal set expansion provides the needed redundancy for the error control without increasing the signal rate and hence without increasing the bandwidth requirement. Coding is used to form structured signal sequences with large minimum Euclidean distance between any two transmitted signal sequences.

Basically, coding for error control is achieved by adding properly designed redundancy to the transmitted information sequence. The redundancy is obtained either by adding extra symbols to the transmitted information sequences or by the channel signal-set expansion in which a larger signal set is used for mapping transmitted information sequence into a signal sequence. Coding by adding redundant symbols results in bandwidth expansion and is suitable for error control in power-limited communication systems. Coding by channel signal set expansion allows coding gain without bandwidth expansion and is suitable for error control in bandwidth-limited communication systems.

1.2.17 Systematic Codes

For any code (n,k), the codeword $\mathbf{c}$ is generated by the transformation of input vector $\mathbf{x}$, as $\mathbf{c} = F(\mathbf{x})$ where $\mathbf{x} = [x_1, \ldots, x_k]$, and it is equals to $\begin{bmatrix} c_1 c_2 & \ldots c_n \end{bmatrix}$. When the code is systematic, the codeword $\mathbf{c}$ is equal to $\begin{bmatrix} x_1 & \ldots & x_k & p_1 & \ldots & p_{n-k} \end{bmatrix}$. Therefore, the codeword $\mathbf{c}$ can be rewritten in a more compact way as $\mathbf{c} = [\mathbf{x}\ \mathbf{p}]$, where $\mathbf{x}$ is an exact copy of code's input vector (that is, information sequence $\mathbf{x}$) and vector $\mathbf{p}$ is generated by the code. Those additional bits are called parity check bits, and the vector $\mathbf{p}$ is called parity check vector.

1.2.18 Hard-Decision and Soft-Decision Decoding

In figure (1.1), when binary demodulator output quantization is used (quantization levels $Q = 2$), the channel decoder processes binary inputs (hard decision inputs). This type of decoding is referred to as hard-decision decoding. Generally, the decoding algorithm which uses only the quantized received bit values and not their reliabilities is referred to as hard-decision decoding. Many coded digital communication systems use binary coding with hard-decision decoding owing to the resulting simplicity of the implementation. The metric which is used in hard-decision decoding in order to describe the channel decoding schema, is the Hamming distance. A hard decision of a received signal results in a loss of information, which degrades the performance of the decoder. This loss comes as result of the information loss which is being applied due to the demodulator's output quantization.

However, when the demodulator output is quantized to more than two levels ($Q > 2$) or if it is left unquantized , the demodulator is said to make soft decisions. In this case, the channel decoder must accept multilevel (or continuous valued inputs) and processes soft decision inputs. This type of decoding is referred to as soft-decision decoding. Generally, a decoding algorithm which takes into account reliability information or uses probabilistic or likelihood values rather than quantized data is called soft-decision decoding algorithm. Although this makes the decoder

more difficult to implement, soft-decision decoding offers significant performance improvement compared with hard-decision decoding. As a general rule,soft-decision decoding can provide as much as 3 dB of gain over hard-decision decoding.

1.2.19 SISO Decoder

A soft-input soft-output (SISO) decoder is a type of soft-decision decoder used with error correcting codes. Soft-input refers to the fact that the incoming data may takes values to a continuous space. Soft-output refers to the fact that each bit in the decoded output also takes a value on a continuous space. Generally, a soft-output decoder would provide decoded values accompanied by an associated reliability measure, or a probabilistic distribution of the decoded bits. Hence, the soft output is used as the soft input to an outer decoder in a system using concatenated codes, or to modify the input to a further decoding iteration such as in the decoding of turbo codes.

1.2.20 State Diagram

The state diagram is a graph that consists of nodes, representing the states of the diagram, and directed lines, representing the state transitions. Each directed line is labeled with an input/output pair. Given a current state, the input value shows which is the next state. Further, given a initial state, the information sequence at the input determines the path through the state diagram and the corresponding output sequence. For example,

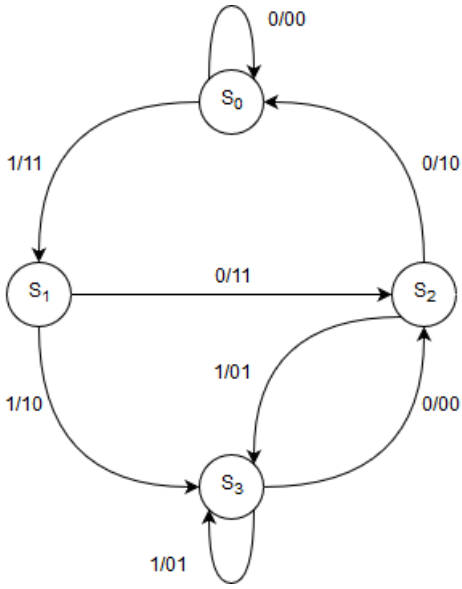

Figure 1.2: State Diagram

1.2.21 Trellis Representation

A Trellis representation is created by stacking bipartite graphs to show several time steps. These bipartite graphs are representing all possibles connections from the states at one time

instant to the states at the next time instant. Hence, a trellis diagram corresponds to a state diagram expanded over time. Further, a code trellis diagram can be treated as an edge-labeled directed graph in which every path represents a codeword for block codes, or a code sequence for convolutional codes. Generally, a trellis diagram is illustrated as :

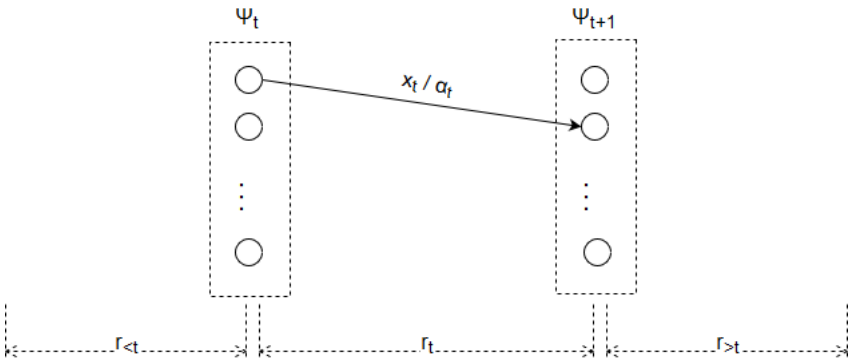

Figure 1.3: Trellis Diagram

where x_t is the code's input vector which produce the codeword c_t, a_t is the modulated version of the corresponding codeword c_t and r_t denotes the received vector at the same time unit t. Also, Ψ_t and Ψ_{t+1} denote the current state in time unit t and $t+1$, respectively. Note that the trellis stages between time units t and $t+1$ are exact replicas of the corresponding state diagram.

1.2.22 Decision Rule

Let $\mathbf{S}$ denotes the transmitted value with prior probability $P(\mathbf{S} = \mathbf{s})$. The receiver uses the received point $\mathbf{R} = \mathbf{r}$ to make a decision about what the transmitted signal $\mathbf{S}$ is. The decision rule which minimizes the probability of error is to choose $\hat{\mathbf{s}}$ to be that value of $\mathbf{s}$ which maximizes $P(\mathbf{S} = \mathbf{s}|\mathbf{r})$, where the possible values of $\mathbf{s}$ are those in the signal constellation $\mathcal{S}$. That is,

$$\hat{\mathbf{s}} = \arg\max_{\mathbf{s}\in\mathcal{S}} P(\mathbf{S} = \mathbf{s}|\mathbf{r}) \tag{1.14}$$

Using Bayes' rule, we can rewrite the above equation as :

$$\hat{\mathbf{s}} = \arg\max_{\mathbf{s}\in\mathcal{S}} \frac{P(\mathbf{R} = \mathbf{r}|\mathbf{S})P(\mathbf{S} = \mathbf{s})}{P(\mathbf{r})} = \arg\max_{\mathbf{s}\in\mathcal{S}} P(\mathbf{R} = \mathbf{r}|\mathbf{S})P(\mathbf{S} = \mathbf{s}) \tag{1.15}$$

since the denominator does not depend on $\mathbf{s}$. This is called the maximum a posteriori (MAP) decision rule. In the case that all the prior probabilities are equal, this rule can be simplified to :

$$\hat{\mathbf{s}} = \arg\max_{\mathbf{s}\in\mathcal{S}} P(\mathbf{R} = \mathbf{r}|\mathbf{S}) \tag{1.16}$$

which is called the maximum likelihood (ML) decision rule.

The above decision rules minimize the codeword error probability. However, a special case of MAP decision rule exists which minimizes the bit error probability and which is called bit-wise MAP decision rule. This bit-wise MAP criterion is :

$$\hat{\mathbf{s}} = \arg\max_{s_i\in\mathcal{S}} \frac{P(\mathbf{R} = \mathbf{r}|S = s_i)P(S = s_i)}{P(\mathbf{r})} = \arg\max_{s_i\in\mathcal{S}} P(\mathbf{R} = \mathbf{r}|S)P(S = s_i) \tag{1.17}$$

1.2.23 Shannon's Noisy Channel Coding Theorem

Shannon's noisy channel coding theorem implies that arbitrarily low decoding error probabilities can be achieved at any transmission rate R less than the channel capacity C by using sufficiently long block (or constraint) lengths. The channel capacity is a measure of how much information the channel can convey. In particular, Shannon proved that randomly chosen codes, along with maximum likelihood decoding (MLD), can provide capacity-achieving performance. He did this by providing that the average performance of a randomly chosen ensemble of codes results in an exponentially decreasing decoding error probability with increasing block (or constraint) length. However, the channel coding theorem is an existence theorem but no how to construct practical codes.

1.2.24 Channel Capacity

The capacity of a channel is defined as :

$$C = \max_{Pr(x)} I(X;Y) \tag{1.18}$$

where $I(X;Y)$ is the mutual information of the channel output Y and the channel input X. Furthermore, $I(X;Y)$ is equal to :

$$I(X;Y) = H(Y) - H(Y|X) = H(X) - H(X|Y) \tag{1.19}$$

where $H(Y) = E\{-\log_2 P(Y)\} = -\sum_y P(y)\log_2 P(y)$ and $H(Y|X) = E\{-\log_2 P(Y|X)\} = -\sum_y \sum_x P(x)P(y|x)\log_2 P(y|x)$ and which are called entropy and conditional entropy, respectively. Thus, the channel capacity is the maximum mutual information between the input X and the output Y, where the maximization is over the channel input probability distribution, or, the amount of information that can be transmitted reliably through the channel. All the codes with rate $R < C$ are said to be achievable in the sense that reliable communication is achievable at this rate. Conversely, if the rate of the code R is greater that the channel capacity C, that is $R > C$, no reliable communication can be achieved.

1.2.25 Error Floor

The error floor is a phenomenon encountered in modern iterated graph-based error correcting codes like Turbo Codes. As a result of this phenomenon, there is a region in high SNRs, which is called the error floor region, in which the performance of the code flattens. This performance floor is an outcome of the error correction capability of the code and, thus, it depends on the minimum distance of the code.

1.3 Typical Operation of Turbo Codes

The following remarks are related to the typical operation of Turbo Codes :

- To achieve performance close to the Shannon limit, the information block length and, as a result, the size of the interleaver K, is chosen to be very large, usually at least several thousand bits.

- The best performance at moderate BERs down to about 10^{-5} is achieved with short constraint length constituent encoders, typically v=4 or less.
- The constituent codes are normally generated by the same encoder, but this is not necessary for good performance. In fact, some asymmetric code designs have been shown to give very good performance
- Recursive constituent codes, generated by systematic recursive (feedback) encoders, give much better performance than nonrecursive constituent codes.
- Bits can be punctured from the parity sequences to produce higher code rates.
- Bits can also be punctured from the information sequence to produce partially systematic or nonsystematic turbo codes
- Because only the ordering of the bits is changed by the interleaver, the sequence that enters the second encoder has the same weight as the sequence that enters the first encoder.
- The interleaver is an integral part of the overall encoder, and thus the state complexity of turbo codes is extremely large, making trellis-based ML or MAP decoding impossible
- Suboptimum iterative decoding, which employs individual SISO decoders for each of the constituent codes in an iterative manner, typically, achieves performance within a few tenths of a decibel of overall ML or MAP decoding. The best performance is obtained when the BCJR, or any MAP, algorithm is used as the SISO decoder for each constituent code.
- Because the MAP decoder uses a forward-backward algorithm, the information is arranged in blocks. Thus, the first constituent encoder is terminated by appending v bits to return to the 0 state. Because the interleaver reorders the input sequence, the second encoder will not normally return to the 0 state, but this has little effect on the performance for large block lengths. If desired, though, modifications can be made to ensure termination of both encoders.
- Block codes can be also used as constituent codes in turbo encoders
- Decoding can be stopped, and final decoding estimate declared, after some fixed number of iterations (usually on the order of 10-20) or based on a stopping criterion that is designed to detect when the estimate is reliable with very high probability

Chapter 2

Basic Structure

A turbo encoder refers to the concatenation of two or more (n,k) recursive systematic convolutional encoders separated by interleavers. On the other hand, a turbo decoder consists of two (or more) soft-in/soft-out (SISO) convolutional decoders separated by interleavers which iteratively feed probabilistic information (soft information) back and forth to each other in a manner that is reminiscent of the turbo engine. This probabilistic information is called extrinsic information and it is updated at each iteration. Because the interleaver is part of the code design, a complete maximum likelihood decoder $P(\mathbf{x}|\mathbf{r})$ for the entire codeword is prohibitively complex. Thus, we use a different approach than maximum likelihood decoder, which is called bit-wise maximum likelihood decoder $P(X_t = x|\mathbf{r})$. This decoder approach is suboptimal but it almost always converges to the maximum likelihood solution. Turbo Codes with more than two encoders and decoders are called multiple turbo codes.

There are several classes of Turbo Codes, like Parallel Concatenated Convolutional Code (PCCC) and Serial Concatenated Convolutional Code (SCCC). The first class of Turbo Codes was based on the Parallel Concatenated Convolutional Code and was introduced by Claude Berrou. For reasons of bandwidth efficiency, parallel concatenation is preferred over serial concatenation. In fact, parallel concatenation that combines two codes with rates R_1 and R_2 gives an overall rate R equal to :

$$R = \frac{R_1 R_2}{1 - (1 - R_1)(1 - R_2)} \tag{2.1}$$

This is a rate higher than that of a serially concatenated code, which is :

$$R = R_1 R_2 \tag{2.2}$$

for the same values of R_1 and R_2, and the lower these rates, the larger the difference. Thus, with the same performance of component codes, parallel concatenated offers a better global rate, but this advantage is lost when the rates come close to unity. Some other popular classes of turbo coding are Turbo Product Codes and Repeat-accumulate Code.

2.1 Parallel Concatenated Convolutional Code - PCCC

Turbo codes are a subclass of the parallel-concatenated code schema where convolutional codes are being used as the encoding algorithm. The general structure of a PCCC encoder and decoder is described below.

2.1.1 Encoder

The conventional arrangement for the parallel-concatenated convolutional code (PCCC) encoder consists of two (or more) convolution encoders called constituent encoders and one (or more) interleaver, which permutes the input symbols prior to input to the next constituent encoder. This structure is called parallel concatenation because the two encoders operate on the same input sequence, in comparison with the serial concatenated codes where the second constituent encoder operates on the encoded output sequence from the first encoder. Hence, the basic structure of a PCCC encoder is :

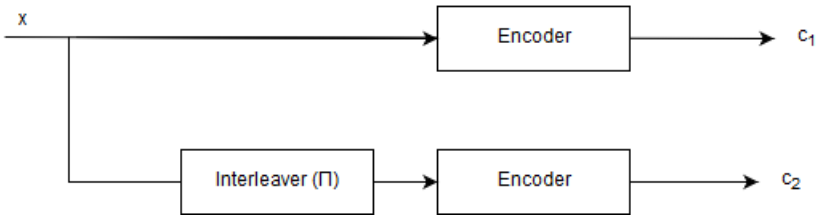

Figure 2.1: Basic Structure of a PCCC Encoder

2.1.2 Decoder

The decoder of a parallel-concatenated convolution code (PCCC) follows directly from the turbo principle. Per the turbo principle, only extrinsic log likelihood ratio information is sent from one decoder to the other, with appropriate interleaving-deinterleaving processing in accordance with the inteleaving mechanism of the PCCC encoder. After a number of iterations, the decoder can sum the log likelihood ratio from the channel, from its companion decoder and from its own computations to produce the total log likelihood ratio values for the data bits x_k. The decisions are made from the total log likelihood ration (LLR) values as : $\tilde{x}_k = sign\big[LLR(x_k)\big]$. The two (or more) constituent decoders are SISO decoders matched to the corresponding top and bottom encoders of the PCCC encoder. The basic structure of a PCCC decoder is :

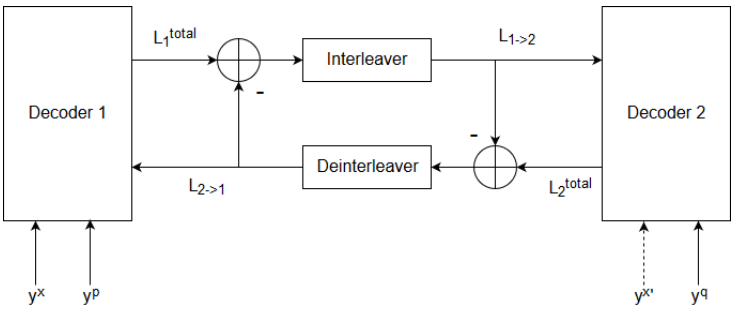

Figure 2.2: Basic Structure of a PCCC Decoder

where L_1^{total} and L_2^{total} is the total log likelihood information at the output of Decoder 1 and Decoder 2, respectively. The log likelihood quantity $L_{1\to2}$ is the extrinsic information sent from Decoder 1 to Decoder 2, and similarly for $L_{2\to1}$.

Chapter 3

Convolutional Codes

3.1 General Informations

Convolutional codes were first introduced by Elias in 1955 as an alternative to block codes and has been widely in use for wireless, space and broadcast communications since about 1970. In 1967, Viterbi proposed a maximum likelihood decoding algorithm that was relatively easy to implement for soft-decision decoding of convolutional codes with small constraint length, which is called Viterbi algorithm. In 1974, Bahl, Cocke, Jelinek and Ravik (BCJR) introduced a maximum a-posteriori probability decoding algorithm for convolutional codes with unequal a-priori probabilities for the information bits, which is called BCJR algorithm. The BCJR algorithm has been applied to soft-decision iterative decoding schemes in which the a-priori probabilities of the information bits change from iteration to iteration.

The convolutional codes are linear codes with additional structure in their generator matrix, which allows the encoding procedure to be viewed as a filtering (or convolutional) operation. Hence, the convolutional encoder can be viewed as a set of digital linear and time-invariant filters with the code sequence being the interleaved output of the filter output. Convolutional codes have better performance over block codes of comparable encode-decode complexity. Furthermore, they are among the earliest codes for which effective soft-decision decoding algorithms were developed.

The encoder of a convolutional code accepts k-bit blocks of the information sequence x and produces an encoded sequence c of n-symbol blocks. However, each encoded block depends not only on the corresponding k-bit message block at the same time unit but also on m previous message blocks. Hence, the encoder has a memory order of m.

A rate $R = \frac{k}{n}$ convolutional encoder with memory order m can be realized as a k-input, n-output linear sequence circuit with input memory m; that is, inputs remain in the decoder for additional m time units after entering. Typically, for convolutional codes, the arguments n and k are small integers with $k < n$. The encoder divides the information sequence into blocks of length k, and produce the output sequence that is divided into blocks of length n. In the important special case when $k = 1$, the input sequence is not divided into blocks and it is processed continuously as a continuous stream. In this case, convolution codes can be viewed as stream codes because they often operate on continuous streams of symbols and not on partitioned discrete message blocks. Unlike with block codes, large minimum distances and low error probabilities are achieved not by increasing k and n, but by increasing the memory

order m, as demonstrated below.

The capability of a noisy channel to transmit information reliably was determined by Shannon in his original work. This result, called noisy channel coding theorem, states that every channel has a capacity C, and that for any rate $R < C$, there exist codes of rate R that, with maximum likelihood decoding, have an arbitrarily small decoding error probability $P(E)$. In particular, for any (n, k) convolutional codes with rate $R < C$ and sufficiently large memory order m, the error probability is bounded by:

$$P(E) \leq 2^{-(m+1)nE_c(R)} \tag{3.1}$$

where m is equal to the memory of the encoder and $E_c(R)$ is a positive function of R and it is completely determined by the channel characteristics. Thus, we can conclude that for a given code with fixed rate R, arbitrarily small error probabilities and large minimum distance are achievable with convolutional coding by increasing memory order m, while holding k and n constant. A maximum likelihood decoding scheme for convolutional codes requires approximately 2^{km} computations to decode each block of k information bits. Hence, the computational complexity of the decoding scheme increases exponentially as the length k of the input block and the memory order m increase, and, as a result, we can not arbitrarily increase neither the code's input nor the code's memory order m.

Encoders for convolutional codes fall into two general categories: feedforward and feedback. A feedforward encoder (or FIR encoder) is called the encoder which has only polynomial entries in its generator matrix. A feedback encoder (or IIR encoder) is called the encoder which has either both rational and polynomial entries or only rational entries. Further, within each category, encoders can be either classified as systematic or non-systematic. For a given constraint length, the systematic feedback encoders have generally superior distance properties compared to systematic feedforward encoders.

For an (n, k) convolutional code, the encoder can be implemented using k shift registers to store the input sequences and n modulo-2 adders to form the output sequences. Specifically, the k input sequences enter the left end of each shift register, and the n output sequences are produced by modulo-2 adders external to the shift registers. Such realization is called Controller Canonical Form. Additionally, convolutional encoders can be implemented in Observer Canonical Form. In general, for observer canonical form encoder realization, there is one shift register corresponding to each of the n output sequences, the k input sequences enter modulo-2 adders internal to the shift register, and the outputs at the right end of each shift register form the n output sequences. Any feedforward and feedback encoder can be realized in both manners. Moreover, any encoder in controller canonical form can be transformed in observer canonical form, and vice versa. However, throughout our analysis, we use the controller canonical form because there is a direct correspondence between equations and the realization of an encoder in controller canonical form.

At this point, we will mention some useful terms that we are going to use during the following analysis of convolutional codes. Let v_i be the length of the ith shift register in a convolutional encoder with k input sequences, $i = 1, 2, \ldots, k$. The encoder memory order m, then, is defined as :

$$m = \max_{1 \leq i \leq k} v_i \tag{3.2}$$

and the overall constraint length v of the encoder is defined as :

$$v = \sum_{1 \leq i \leq k} v_i \tag{3.3}$$

Primarily, the performance of a convolutional code depends on the decoding algorithm employed and the distance properties of the code. The most important criterion in construction of good convolutional codes is the maximization of the minimum distance. Then, as secondary criterion, the first term of the weight distribution $A_{d_{min}}$, which is referred to the number of codewords with weight d_{min}, should be minimized. Generally speaking, d_{min} is of primary importance in determiniing performance at high SNRs, but as the SNR decreases, the influence of the number of nearest-neighbors $A_{d_{min}}$ increases, and for very low SNRs the entire weight spectrum plays a role. Because convolutional codes are linear codes, their minimum distance can be computed as :

$$\begin{aligned} d_{min} &= \min_{i \neq j} d_H(\mathbf{c_i}, \mathbf{c_j}) \\ &= \min_{i \neq j} w(\mathbf{c}_i - \mathbf{c}_j) \\ &= \min_{\mathbf{c}} w(\mathbf{c}) \\ &= \min_{\mathbf{x}} w(\mathbf{xG}),\ \forall \mathbf{x} \neq 0 \end{aligned} \tag{3.4}$$

where $\mathbf{c}$ is the codeword corresponding to the information sequence $\mathbf{x}$ and $\mathbf{G}$ is the generator matrix of the code. Based on the above equation, we can conclude that the minimum distance is defined by the generator matrix $\mathbf{G}$ of the code and, thus, is a feature of the code. For a certain code rate and constraint length, nonsystematic feedforward encoders imply more free distance in comparison with systematic feedforward encoders for the same code rate and constraint length. Thus, if a systematic encoder realization is desired, it is better to construct a nonsystematic feedforward encoder with large d_{min} and then convert it to an equivalent systematic feedback encoder.

Since a convolutional encoder can be realized as a linear sequential circuit, it can be described using a finite state machine (FSM). The state of the encoder is defined as its memory content (content of the shift registers). For a (n,k) encoder in controller canonical form, the ith shift register at time unit l, contains v_i bits, denoted as $\left(s_{l-1}^{(i)}, s_{l-2}^{(i)}, \ldots, s_{l-v_i}^{(i)}\right)$, where $s_{l-1}^{(i)}$ represents the contents of the leftmost delay element, and $s_{l-v_i}^{(i)}$ represents the content of the rightmost delay element, $1 \leq i \leq k$. As a result, the state of the encoder σ_l at time unit l is the binary tuple:

$$\sigma_l = \left(s_{l-1}^{(1)}, s_{l-2}^{(1)}, \ldots, s_{l-v_i}^{(1)}, s_{l-1}^{(2)}, s_{l-2}^{(2)}, \ldots, s_{l-v_i}^{(2)}, \ldots, s_{l-1}^{(k)}, s_{l-2}^{(k)}, \ldots, s_{l-v_i}^{(k)}\right) \tag{3.5}$$

Hence, for both encoding and decoding, it is useful to think of the state machine, which is a representation of the temporal relationships between the states portraying current state - next state relationship as a function of the inputs and the outputs. For an implementation with overall constraint length equal to v, there are 2^v states in the state diagram. Throughout this analysis, we assume that all shift registers have equal length. Thus, based on equations (3.3) and (3.2), the overall constraint length v of the (n, k) code is equal to km. As a consequence, there are 2^{km} states in the state diagram. Each new input block of k bits causes the encoder to shift; that is, there is a transition to a new state. Hence, there are 2^k branches leaving each state in the state diagram, one corresponding to each different input blocks. The codeword produced at each time unit depends on both current state and encoder's input sequence. Hence, the state machine of a convolutional code is a Mealy machine. Also, maximum likelihood decoding and maximum a posteriori probability decoding of convolutional codes require a decoding complexity that is proportional to the number of states in the encoder state diagram, thus, we always

seek an encoder realization with a minimal number of states, that is, with a minimal overall constraint length. These encoders are called minimal encoders. Another useful representation is a graph representing the connections from states at one time instants to states at the next time instant. This type of representation is called Trellis representation. We note here that the convolutional encoders whose generator matrix and the associated encoder diagram do not change with time are called time-invariant convolutional encoders. However, when the state diagram of a convolutional encoder changes with time, those encoders are called time-variant convolutional encoders. A special case of time-varying convolutional encoder is used in the class of Turbo Codes.

In the case of systematic convolutional encoders which are implemented based on controller canonical form, the realization of the encoder requires only $(n-k)$ shift registers, since k input sequences appear directly to the encoder's output sequences. Those $(n-k)$ shift registers are responsible for the computation of the $(n-k)$ parity check sequences. In many cases, where the number of the encoder's output sequences, n, and the number of the encoder's input sequences, k, are comparable, the realization of systematic convolutional encoder is more simple than the realization of the corresponding non-systematic convolutional encoder from both hardware and encoder's computational complexity perspective. From hardware perspective, fewer logical units (shift registers and modulo-2 adders) are used for the implementation of a systematic encoder. From the perspective of the encoder's computational complexity, as we mentioned earlier, the encoders complexity is equal to 2^v, where v refers to the overall constraint length of the encoder. The overall constraint length is given from the equation (3.3) and is equal to the sum of the size of the each shift register. Hence, the systematic convolutional encoder has smaller overall constraint length than the non-systematic encoder and, as a result, lower computational complexity.

3.2 Convolutional Codes

Generally, the encoder for an (n, k) convolutional code accepts blocks of k-bits from the information sequence and produces an encoded sequence of n-bits symbols blocks. However, each encoded block depends not only on the corresponding k-bit input message block at that current time unity but also on m previous input message blocks. Hence, the encoder has a memory of some order m. Typically, k and n are small integers, and more redundancy is added by increasing the memory order m of the code while holding k and n fixed.

The information sequence $\mathbf{x} = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 & \dots \end{pmatrix}$, that we want to encode, can be rewritten as $\begin{pmatrix} \mathbf{x}^{(1)} & \mathbf{x}^{(2)} \dots \mathbf{x}^{(k)} \end{pmatrix}$, where each $\mathbf{x}^{(i)}$ is equal to $\begin{pmatrix} x_0^{(i)} & x_1^{(i)} & x_2^{(i)} & \dots \end{pmatrix} = \begin{pmatrix} x_{i-1} & x_{i-1+k} & x_{i-1+2k} & \dots \end{pmatrix}$, $\forall i = 1, \dots, k$, and which, essentially, contains the information bits that correspond to the input of ith encoder at each time point. The output sequence $\mathbf{c}$ can be written as $\begin{pmatrix} c_0^{(1)} \dots c_0^{(n)} c_1^{(1)} \dots c_1^{(n)} & \dots \end{pmatrix}$ $= \begin{pmatrix} \mathbf{c}_0 & \mathbf{c}_1 \dots \mathbf{c}_t & \dots \end{pmatrix}$ where $\mathbf{c}_t$ denotes the codeword produced at the time unit t and is equal to $\begin{pmatrix} c_t^{(1)} \dots c_t^{(n)} \end{pmatrix}$. In a similar manner to the definition of $\mathbf{x}^{(i)}$, we define the sequence $\mathbf{c}^{(j)}$ which is equal to $\begin{pmatrix} c_0^{(j)} & c_1^{(j)} & \dots \end{pmatrix}$ and which represents the output sequence of the jth encoder at each time unit. The equation that produce each sequence $\mathbf{c}^{(j)}$ is :

$$\mathbf{c}^{(j)} = \sum_{i=1}^{k} \mathbf{x}^{(i)} \circledast \mathbf{g}_i^{(j)} \quad \forall j = 1, \dots, n \tag{3.6}$$

where $\circledast$ denotes the discrete convolution and all operation are modulo-2. The impulse response $\mathbf{g}_i^{(j)} = \left(g_{i,0}^{(j)}\ g_{i,1}^{(j)}\ \dots\ g_{i,m}^{(j)}\right)$, which corresponds the input i to the output j, is called generator sequences. This equation is the basic equation that produces any (n, k) convolutional code. The following diagram is schematically shown the procedure followed by a convolutional encoder to produce the output sequence $\mathbf{c}$, as described above.

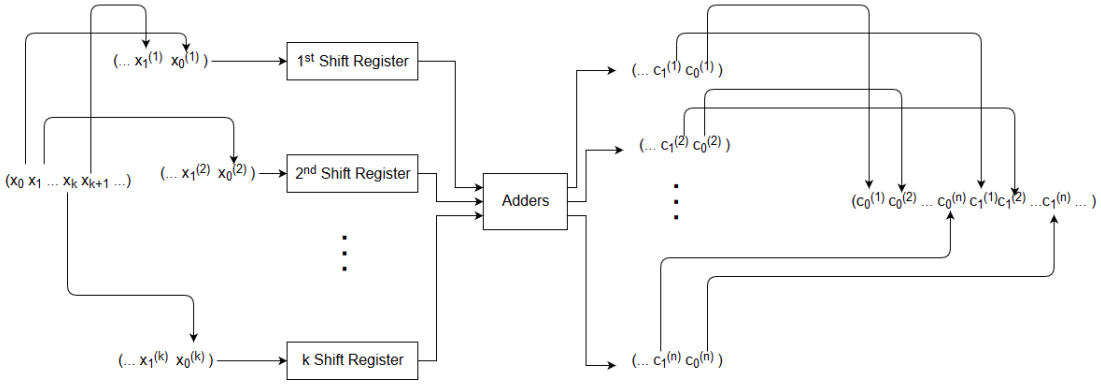

Figure 3.1: Convolutional Encoder

As we can see, the output sequences $\mathbf{c}^{(j)}$ are multiplexed for the purpose of producing the final output sequence $\mathbf{c}$. The module "Adders" denotes the set of modulo-2 adders that combine the components of the shift registers in order to produce the $c_t^{(j)}$ at each time unit.

In a matrix representation manner, the basic equation which generates an (n, k) convolutional codes can be expressed as :

$$\mathbf{c} = \mathbf{xG} \tag{3.7}$$

where all operations are, also, modulo-2. The matrix $\mathbf{G}$ is called the (time domain) generator matrix of the encoder. In the general case of an (n, k) feedforward encoder with memory order m, the generator matrix is given by:

$$\mathbf{G} = \begin{bmatrix} \mathbf{G_0} & \mathbf{G_1} & \mathbf{G_2} & \dots & \mathbf{G_m} & \mathbf{0} & \mathbf{0} & \dots \\ \mathbf{0} & \mathbf{G_0} & \mathbf{G_1} & \dots & \mathbf{G_{m-1}} & \mathbf{G_m} & \mathbf{0} & \dots \\ \mathbf{0} & \mathbf{0} & \mathbf{G_0} & \dots & \mathbf{G_{m-2}} & \mathbf{G_{m-1}} & \mathbf{G_m} & \dots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{3.8}$$

where each G_i is a (k x n) submatrix whose entries are :

$$\mathbf{G_i} = \begin{bmatrix} g_{1,i}^{(1)} & g_{1,i}^{(2)} & \dots & g_{1,i}^{(n)} \\ g_{2,i}^{(1)} & g_{2,i}^{(2)} & \dots & g_{2,i}^{(n)} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k,i}^{(1)} & g_{k,i}^{(2)} & \dots & g_{k,i}^{(n)} \end{bmatrix} \tag{3.9}$$

Each submatrix G_i is being constructed by the generator vectors, which represent the generator sequence corresponding to the input i and the output j. Note that each set of k rows of G is identical to the previous set of rows but shifted n places to the right and that G is semi-infinite

matrix, corresponding to the fact that the information sequence $\mathbf{x}$ is of arbitrarily length. If $\mathbf{x}$ has finite length h, then G has $\frac{h}{k}$ rows and $n(m+\frac{h}{k})$ columns. Consequently, $\mathbf{c}$ has length equal to $n(m+\frac{h}{k})$.

In case of systematic feedforward encoders, the generator matrix $\mathbf{G}$ is given by :

$$\mathbf{G} = \begin{bmatrix} \mathbf{IP_0} & \mathbf{0P_1} & \mathbf{0P_2} & \dots & \mathbf{0P_m} & \mathbf{0} & \mathbf{0} & \dots \\ \mathbf{0} & \mathbf{IP_0} & \mathbf{0P_1} & \dots & \mathbf{0P_{m-1}} & \mathbf{0P_m} & \mathbf{0} & \dots \\ \mathbf{0} & \mathbf{0} & \mathbf{IP_0} & \dots & \mathbf{0P_{m-2}} & \mathbf{0P_{m-1}} & \mathbf{0P_m} & \dots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

Based on the above notation, we can conclude that each $\mathbf{G_i}$ can be written either as :

$$[\mathbf{IP_0}] = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & p_{1,0}^{(1)} & p_{1,0}^{(2)} & \dots & p_{1,0}^{(n-k)} \\ 0 & 1 & 0 & \dots & 0 & p_{2,0}^{(1)} & p_{2,0}^{(2)} & \dots & p_{2,0}^{(n-k)} \\ 0 & 0 & 1 & \dots & 0 & p_{3,0}^{(1)} & p_{3,0}^{(2)} & \dots & p_{3,0}^{(n-k)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 & p_{k,0}^{(1)} & p_{k,0}^{(2)} & \dots & p_{k,0}^{(n-k)} \end{bmatrix}, i = 0$$

or as :

$$[\mathbf{0P_i}] = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 & p_{1,i}^{(1)} & p_{1,i}^{(2)} & \dots & p_{1,i}^{(n-k)} \\ 0 & 0 & 0 & \dots & 0 & p_{2,i}^{(1)} & p_{2,i}^{(2)} & \dots & p_{2,i}^{(n-k)} \\ 0 & 0 & 0 & \dots & 0 & p_{3,i}^{(1)} & p_{3,i}^{(2)} & \dots & p_{3,i}^{(n-k)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 0 & p_{k,i}^{(1)} & p_{k,i}^{(2)} & \dots & p_{k,i}^{(n-k)} \end{bmatrix}, i \neq 0$$

where $\mathbf{I}$ and $\mathbf{0}$ are the (k x k) identity matrix and all-zero matrix, respectively. Comparing the generator matrix of a general convolutional code to the generator matrix of a systematic feedforward code, we can conclude that, any $\mathbf{P_i}$ is a (k x (n-k)) matrix whose entries are equal to :

$$\mathbf{P_i} = \begin{bmatrix} g_{1,i}^{(k+1)} & g_{1,i}^{(k+2)} & \dots & g_{1,i}^{(n)} \\ g_{2,i}^{(k+1)} & g_{2,i}^{(k+2)} & \dots & g_{2,i}^{(n)} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k,i}^{(k+1)} & g_{k,i}^{(k+2)} & \dots & g_{k,i}^{(n)} \end{bmatrix} \tag{3.10}$$

3.2.1 Polynomial (Transform Domain) Representation

As we know, in any linear system, all time domain operations involving convolution can be replaced by more convenient transform domain operations involving polynomial multiplication. Because a convolutional encoder is a linear system, each sequence in the decoding equation can be replaced by a corresponding polynomial, and the convolution operation can be replaced by polynomial multiplication. In the polynomial representation of a binary sequence, the sequence itself is represented by the coefficients of the polynomial; that is, any sequence $x = \begin{pmatrix} x_0 & x_1 & x_2 & \dots \end{pmatrix}$ can be written in polynomial form like $x(D) = x_0 + x_1 D + x_2 D^2 + \dots$. Thus, the basic equation

of the convolutional codes $\mathbf{c}^{(j)} = \sum_{i=1}^{k} \mathbf{x}^{(i)} \circledast \mathbf{g}_i^{(j)}$ can be transformed to :

$$c^{(j)}(D) = \sum_{i=1}^{k} x^{(i)}(D) g_i^{(j)}(D) \tag{3.11}$$

where the output sequence $\mathbf{c}^{(j)}$ can be described as $c^{(j)}(D) = c_0^{(j)} + c_1^{(j)} D + c_2^{(j)} D^2 + \ldots$, the input sequence $\mathbf{x}^{(i)}$ as $x^{(i)}(D) = x_0^{(i)} + x_1^{(i)} D + x_2^{(i)} D^2 + \ldots$ and the generator sequence as $g_i^{(j)}(D) = g_{0,i}^{(j)} + g_{1,i}^{(j)} D + g_{2,i}^{(j)} D^2 + \ldots$. The polynomials $g_i^{(j)}(D)$ are called generator polynomials. The indeterminate D can be interpreted as a delay operator, where the power of D denotes the number of time units a bit is delayed with respect to the initial bit in the sequence.

Hence, the basic equation (3.7) can be written as :

$$c(D) = x(D)G(D) \tag{3.12}$$

where $x(D) = \begin{bmatrix} x^{(1)}(D) & x^{(2)}(D) & \ldots & x^{(k)}(D) \end{bmatrix}$ is the k-tuple of the input sequences, $c(D) = \begin{bmatrix} c^{(1)}(D) & c^{(2)}(D) & \ldots & c^{(n)}(D) \end{bmatrix}$ is the n-tuple of the output sequences and the generator matrix $G(D)$ is equal to :

$$G(D) = \begin{bmatrix} g_1^{(1)}(D) & g_1^{(2)}(D) & \ldots & g_1^{(n)}(D) \\ g_2^{(1)}(D) & g_2^{(2)}(D) & \ldots & g_2^{(n)}(D) \\ \vdots & \vdots & \ddots & \vdots \\ g_k^{(1)}(D) & g_k^{(2)}(D) & \ldots & g_k^{(n)}(D) \end{bmatrix}$$

The matrix $G(D)$ is called generator matrix. Moreover, from equation (3.12) we can conclude that a convolutional code in time domain can be represented as a linear block code, in transform domain. Thus, any theorem of a linear block code can be applied to a convolutional code after the transformation of the code from time to transform domain, and vice versa. As a consequence, we can conclude that block and convolutional codes are dual codes. After multiplexing, the overall encoder's output sequence can be expressed as :

$$c(D) = c^{(1)}(D^n) + Dc^{(2)}(D^n) + \cdots + D^{n-1} c^{(n)}(D^n) \tag{3.13}$$

We can rewrite the equations (3.12) and (3.13) to provide a means of representing the multiplexed codeword $c(D)$ directly in the terms of the input sequences. A little algebraic manipulation yields

$$c(D) = \sum_{i=1}^{k} x^{(i)}(D^n) g_i(D) \tag{3.14}$$

where $g_i(D) = g_i^{(1)}(D^n) + Dg_i^{(2)}(D^n) + \cdots + D^{n-1} g_i^{(n)}(D^n)$, $1 \leq i \leq k$ is a composite generator polynomial relating the ith input sequence to the multiplexed $c(D)$.

Using the polynomial notation, any systematic feedforward encoders can be written as :

$$G(D) = \begin{bmatrix} 1 & 0 & \ldots & 0 & g_1^{(k+1)}(D) & \ldots & g_1^{(n)}(D) \\ 0 & 1 & \ldots & 0 & g_2^{(k+1)}(D) & \ldots & g_2^{(n)}(D) \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 & g_k^{(k+1)}(D) & \ldots & g_k^{(n)}(D) \end{bmatrix} = \begin{bmatrix} I & P(D) \end{bmatrix}$$

where

$$P(D) = \begin{bmatrix} p_1^{(1)}(D) & p_1^{(2)}(D) & \dots & p_1^{(n-k)}(D) \\ p_2^{(1)}(D) & p_2^{(2)}(D) & \dots & p_2^{(n-k)}(D) \\ \vdots & \vdots & \ddots & \vdots \\ p_k^{(1)}(D) & p_k^{(2)}(D) & \dots & p_k^{(n-k)}(D) \end{bmatrix} = \begin{bmatrix} g_1^{(k+1)}(D) & g_1^{(k+2)}(D) & \dots & g_1^{(n)}(D) \\ g_2^{(k+1)}(D) & g_2^{(k+2)}(D) & \dots & g_2^{(n)}(D) \\ \vdots & \vdots & \ddots \vdots & \\ g_k^{(k+1)}(D) & g_k^{(k+2)}(D) & \dots & g_k^{(n)}(D) \end{bmatrix}$$

3.2.2 Systematic Recursive Convolutional Codes - SRCC

High-rate codes are most efficiently represented in a systematic feedback form. An application of systematic feedback convolutional encoders is turbo coding. Any nonsystematic feedforward convolutional code can be converted to a systematic feedback code using elementary row operations. These systematic feedback encoders generate the same codes as corresponding nonsystematic feedforward encoders but exhibit a different mapping between information sequences and codewords. The key characteristic of feedback encoders is that their impulse response has infinite duration. In particular, for nonrecursive codes, information sequence of weight 1 produce finite-length codewords, whereas for recursive codes, information sequences of weight 1 produce codewords of infinite length. Thus, the feedback shift register realization is an infinite impulse response (IIR) linear system. For this reason, the (time-domain) generator matrix $\mathbf{G}$ contains sequences of infinite length. Because the response to a single nonzero input in a feedback encoder has infinite duration, the codes produced by feedback encoders are called recursive convolutional codes (RSCC). Hence, a more convenient way to examine the feedback code is through transform domain representation. This property of recursive codes turns out to be a key factor in the excellent performance achieved by the class of turbo codes.

Feedback encoders are more easily described using the transform domain representation generator matrices $G(D)$ whose entries are contain rational functions, rather than the corresponding time-domain representation generator matrices $\mathbf{G}$ whose entries contain sequences of infinite length. Thus, we say that a systematic feedback encoder produces a systematic recursive convolutional code (RSCC). In general, any nonsystematic feedforward convolutional encoder with a k x n polynomial generator matrix $G(D)$ can be transformed to a RSCC with polynomial generator matrix $G'(D)$ using elementary polynomial row operations. It is important to note that elementary rows operations do not change the row space of a matrix. Thus, the RSCC produced by the systematic feedback encoder generates the same code as the initial nonrecursive convolutional code; however, the mapping between information sequences and corresponding codewords is different in two cases.

The generator matrix of a systematic feedback encoder has rational entries $\frac{a_i^{(j)}(D)}{b_i^{(j)}(D)}$, where $a_i^{(j)}(D) = a_0 + a_1 D + \cdots + a_m D^m$ and $b_i^{(j)}(D) = 1 + b_1 D + \cdots + b_m D^m$. Note that, b_0 should be always equal to 1 or, equivalent, the denominator polynomial $b_i^{(j)}(D)$ is delay free. This means that it can always be written in a form where the common factor D^i has been pulled out. The assumption of $b_0 = 1$ implies that $g_i^{(j)}$ is a causal transfer function and, hence, is realizable.

This can be proven as :

$$
\begin{aligned}
&c^{(j)}(D) = x^{(i)}(D) g_i^{(j)}(D) = x^{(i)}(D) \frac{a(D)}{b(D)} \Leftrightarrow \\
&b(D)c^{(j)}(D) = x^{(i)}(D)b(D) \overset{F^{-1}}{\Longleftrightarrow} b[t] \circledast c^{(j)}[t] = a[t] \circledast x^{(i)}[t] \Leftrightarrow \\
&\sum_{k=-\infty}^{+\infty} b[k]c^{(j)}[t-k] = \sum_{k=-\infty}^{+\infty} a[k]x^{(i)}[t-k] \Leftrightarrow \sum_{k=0}^{+\infty} b[k]c^{(j)}[t-k] = \sum_{k=0}^{+\infty} a[k]x^{(i)}[t-k] \Leftrightarrow \\
&b_0c^{(j)}[t] + b_1c^{(j)}[t-1] + \cdots + b_mc^{(j)}[t-m] = a_0x^{(i)}[t] + a_1x^{(i)}[t-1] + \cdots + a_mx^{(i)}[t-m] \Leftrightarrow \\
&b_0c^{(j)}[t] = a_0x^{(i)}[t] + a_1x^{(i)}[t-1] + \cdots + a_mx^{(i)}[t-m] - b_1c^{(j)}[t-1] - \cdots - b_mc^{(j)}[t-m]
\end{aligned}
\tag{3.15}
$$

If $b_0 = 0$, it makes the situation untenable, i.e, $c^{(j)}[t]$ can not be determined from the inputs $x^{(i)}[t], \ldots, x^{(i)}[t-m]$ and the outputs $c^{(j)}[t-1], \ldots, c^{(j)}[t-m]$. Consequently, a general (transform-domain) generator matrix $G(D)$ for a RSC code can be written as :

$$
G(D) = \begin{bmatrix}
1 & 0 & \ldots & 0 & \frac{a_1^{(k+1)}(D)}{b_1^{(k+1)}(D)} & \ldots & \frac{a_1^{(n)}(D)}{b_1^{(n)}(D)} \\
0 & 1 & \ldots & 0 & \frac{a_2^{(k+1)}(D)}{b_2^{(k+1)}(D)} & \ldots & \frac{a_2^{(n)}(D)}{b_2^{(n)}(D)} \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & 1 & \frac{a_k^{(k+1)}(D)}{b_k^{(k+1)}(D)} & \ldots & \frac{a_k^{(n)}(D)}{b_k^{(n)}(D)}
\end{bmatrix}
$$

Generally, it is not mandatory to use different denominators in each entry. Therefore, we can replace each $b_i^{(j)}(D)$ with a universal denominator $b(D)$.

The structure of a feedback (recursive) shift register is shown below :

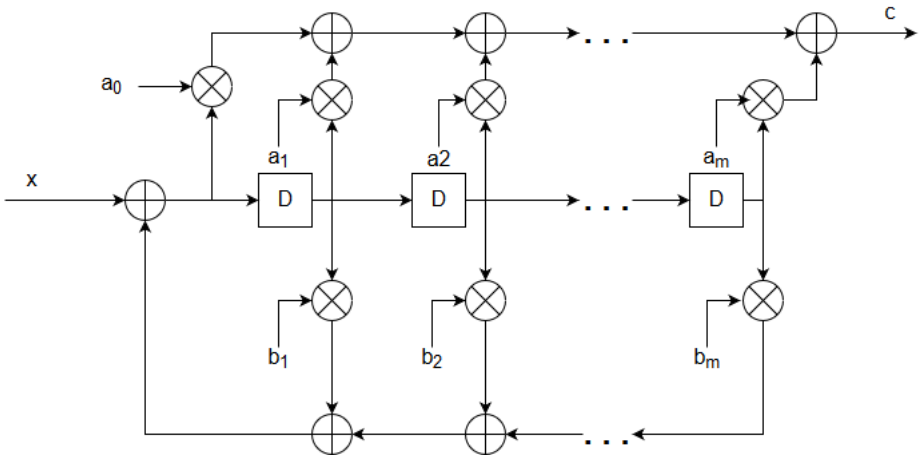

Figure 3.2: Feedback Shift Register

3.3 Catastrophic Encoders

A convolutional encoder is called catastrophic when a finite number of channel errors causes an infinite number of decoding errors. Catastrophic encoders should not be avoided. Generally, an encoder $G(D)$ for a convolutional code is catastrophic if there exists message sequence $x(D)$ such that $w(x(D)) = \infty$ and the weight of the coded sequence $w(c(D)) < \infty$. A mean which can be used in order to check if an encoder is catastrophic, is the existence of the inverse generator matrix, $G^{-1}(D)$. Additionally, an encoder is catastrophic if and only if the state diagram of the code contains a cycle with zero output weight other than the zero-weight cycle around the initial state S_0. Systematic convolutional encoders always posses a polynomial inverse matrix $G^{-1}(D)$, which makes them always noncatastrophic.

3.4 Rate of Convolutional Codes

An (n,k) convolutional encoder can generate infinitely long encoded sequences. Because n encoded bits are produced for each k information bits, $R = \frac{k}{n}$ is called the rate of the convolutional encoder. For a finite-length information sequence of h time units, or $K = kh$ bits, it is customary to begin encoding in the all-zero state; that is, the initial contents of the shift registers are all zeros. When we consider trellis-based decoding, unless the final state is known, the last several decoded bits will be somewhat unreliable. Thus, to facilitate decoding, it is necessary to return the encoder to an known state or the all-zero state after the last information block has entered the encoder. This requires that some additional input blocks equal to the memory order m of the encoder, should enter the encoder to force it to return to the all-zero state. This sequence of bits is called zero-append sequence. For feedforward encoders, it can be seen from the encoder diagram that these m termination blocks must be all zeros, whereas for feedback encoders, the termination blocks depend on the information sequence. Thus, we can not foresee which bit sequence can drive the encoder's state machine back to the all-zero state, as we can assume from figure (3.2). In both cases, these termination blocks are not part of the information sequence, since they are fixed and cannot be chosen arbitrarily. We note that for controller canonical form encoder realizations with $v < km$, only v bits are required for termination. Thus, $km - v$ bits in the termination block can be information bits. A similar situation holds in the case of observer canonical form realizations.

In case of additional blocks are added in order to drive the state machine back to the all-zero state, the actual rate of the code is smaller than the theoretical one. More precisely, for a finite-length information sequence with $K = kh$ bits and memory order m, the output sequence has length $N = n(h + m)$. Thus, the actual code rate is equal to $R = \frac{K}{N} = \frac{kh}{n(m+h)}$. If $h >> m$, then $\frac{h}{h+m} \simeq 1$ and the actual code rate is approximately equal with the theoretical one. Contrariwise, if h is small compared to the memory order m, the actual code rate is reduced below the convolutional encoder rate R by a fractional amount

$$\frac{\frac{k}{n} - \frac{kh}{n(h+m)}}{\frac{k}{n}} = \frac{m}{h+m} \tag{3.16}$$

which is called the fractional rate loss.

For short information sequences, the fractional rate loss can be eliminated by simply not transmitting the nm termination bits and considering the first $N = nh$ encoder output bits as

the codeword. However, if the termination bits are not transmitted, the performance of the code is degraded. One way to combat this problem is to allow the encoder to start in an arbitrary state (the initial contents of the shift registers are not necessarily all-zeros) and then force it to return to the same at the end of the information sequence. It turns out that if the starting state is chosen properly, no extra bits are required to force the encoder back to the same state, so no fractional rate loss is incurred. This technique results in so-called tail-bitting codes.

3.5 Decoders for convolutional codes

There exist several decoding algorithms for convolutional codes, including the majority-logic decoder, the sequential decoder, the Viterbi decoder and the BCJR decoder. The choice of the decoding algorithm depends on the application. In case of turbo codes, the chosen decoding algorithms are maximum-likelihood sequence decoder (MLSD), which is implemented via a variation of the Viterbi algorithm which is called soft-output Viterbi algorithm (SOVA) and the bit-wise maximum a-posteriori (MAP) decoder which is implemented via BCJR algorithm. The SOVA algorithm provides not only decoded symbols but also an indication of the reliability of the decoded values. The MLSD decoder minimizes the probability of code-sequence error, while MAP minimizes the probability of information-bit error. Suboptimal decoding algorithms are also occasionally of interest, particularly the constraint length is large.

Chapter 4

Viterbi

The Viterbi algorithm was proposed by Andrew Viterbi in 1967 and it is an optimal decoding algorithm for convolutional codes with low constraint length (typically ≤ 8). It is a maximum likelihood (ML) decoding algorithm for convolutional codes; that is, the decoder output selects always the codeword that maximize the conditional probability of the received sequence. Thus, Viterbi algorithm minimizes the codeword error probability. Essentially, it is a shortest path algorithm, roughly analogous to Djikstra's shortest path for computing the shortest path through the trellis associated with the code. Further, the Viterbi algorithm is equivalent to a dynamic programming solution to the problem of finding the shortest path through a weighted graph. The Viterbi algorithm has been applied to a variety of communication problems, including maximum likelihood sequence estimation in the presence of intersymbol interference and optimal reception of spread-spectrum multiple access communication. It also appears in many other problems where a "state" can be defined, such as in hidden Markov modeling.

Generally, a decoder based on the Viterbi algorithm takes the received sequence $\mathbf{r} = \{\mathbf{r_0}, \ldots, \mathbf{r_{L-1}}\}$ and determines an estimation of the transmitted data $\mathbf{a} = \{\mathbf{a_0}, \mathbf{a_1}, \ldots, \mathbf{a_{L-1}}\}$ and from that an estimation of the sequence of the input data $\mathbf{x} = \{\mathbf{x_0}, \mathbf{x_1}, \ldots, \mathbf{x_{L-1}}\}$, as is shown in the following figure :

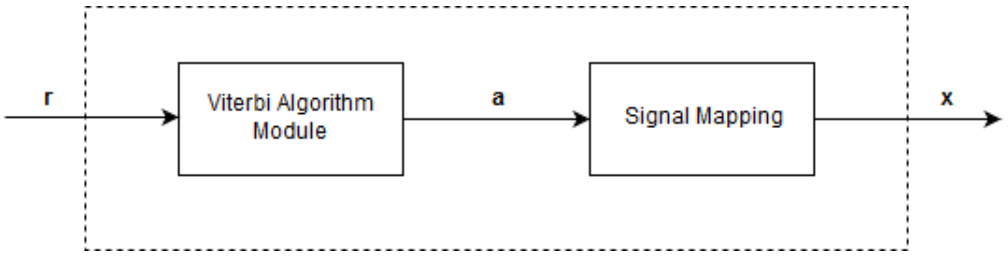

Figure 4.1: Viterbi Decoder

The basic idea behind the Viterbi algorithm is that a coded sequence $\{\mathbf{c}_0, \ldots, \mathbf{c}_{L-1}\}$, or its signal-mapped equivalent $\{\mathbf{a}_0, \ldots, \mathbf{a}_{L-1}\}$, corresponds to a path through the encoder's trellis. Due to noise in the channel, the received sequence $\mathbf{r}$ may or may not correspond exactly to a path through the trellis. The decoder finds a path through the trellis which is closest to the received sequence, where the measure of "closest" is determined by the likelihood function appropriate for the channel. Especially, for a BSC channel and an AWGN channel, the maximum

likelihood path corresponds to the path through the trellis which is closest to the Hamming and the Euclidean distance to $\mathbf{r}$, respectively.

4.1 Viterbi Algorithm

The Viterbi algorithm is a maximum likelihood sequence estimator in conjuction with the BCJR algorithm which is a bit-wise maximum a-posteriori bit estimator. Assume that the information sequence $\mathbf{x} = \{\mathbf{x_0}, \mathbf{x_1}, \ldots, \mathbf{x_{L-1}}\}$, where $\mathbf{x_i} = \{x_i^{(1)}, x_i^{(2)}, \ldots, x_i^{(k)}\}$ for each $i = 0, \ldots, L-1$, is encoded into the codeword $\mathbf{c} = \{\mathbf{c_0}, \mathbf{c_1}, \ldots, \mathbf{c_{L-1}}\}$, where $\mathbf{c_i} = \{c_i^{(1)}, c_i^{(2)}, \ldots, c_i^{(n)}\}$ for each $i = 0, \ldots, L-1$, and be transmitted over a memoryless channel, as shown in the following figure:

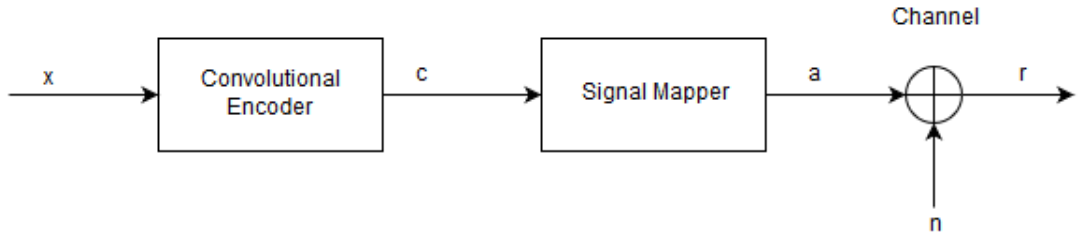

Figure 4.2: Transmission

where the received sequence is represented by $\mathbf{r}$. The decoder must produce an estimate sequence $\hat{\mathbf{x}}$ of the information sequence $\mathbf{x}$ based on the received sequence $\mathbf{r}$. A maximum likelihood (ML) decoder for a memoryless channel chooses the sequence $\hat{\mathbf{x}}$ that maximizes the log-lokelihood function $\log p(\mathbf{r}|\mathbf{a})$. The maximum likelihood function is equal to :

$$p(\mathbf{r}|\mathbf{a}) = p(\mathbf{r_0^{L-1}}|\mathbf{a_0^{L-1}}) = p(\mathbf{r_0}, \ldots, \mathbf{r_{L-1}}|\mathbf{a_0}, \ldots, \mathbf{a_{L-1}}) \tag{4.1}$$

If the channel is memoryless, the above equation can be computed as :

$$p(\mathbf{r}|\mathbf{a}) = \prod_{i=0}^{L-1} p(\mathbf{r}_i|\mathbf{a}_i) = \prod_{i=0}^{L-1} \left(\prod_{j=1}^{n} p(r_i^{(j)}|a_i^{(j)}) \right) \tag{4.2}$$

As regards the log-likelihood function $\log p(\mathbf{r}|\mathbf{a})$, it follows that :

$$\log p(\mathbf{r}|\mathbf{a}) = \sum_{i=0}^{L-1} \log p(\mathbf{r}_i|\mathbf{a}_i) \tag{4.3}$$

where $\log p(\mathbf{r}_i|\mathbf{a}_i)$ is a channel transition probability equals to :

$$\log p(\mathbf{r}_i|\mathbf{a}_i) = \log \prod_{j=1+}^{n} p(r_i^{(j)}|a_i^{(j)}) = \sum_{j=1}^{n} \log p(r_i^{(j)}|a_i^{(j)}) \tag{4.4}$$

Now, consider a sequence $\hat{\mathbf{x}}_0^{t-1} = \{\hat{\mathbf{x}}_0, \ldots, \hat{\mathbf{x}}_{t-1}\}$ which leaves the decoder in state $\Psi_t = p$ at time t. This sequence determines a path, or sequence of states, through the trellis for the code, which we abstractly denote as Π_t or $\Pi_t(\mathbf{x}_0^{t-1})$. Thus,

$$\Pi_t = \{\Psi_0, \Psi_1, \ldots, \Psi_t\} \tag{4.5}$$

The log-likelihood function for the sequence is :

$$\log\left[p\big(\mathbf{r}_0^{t-1}|\hat{\mathbf{x}}_0^{t-1}\big)\right] = \sum_{i=0}^{t-1} \log\left[p\big(\mathbf{r}_i|\hat{\mathbf{x}}_i\big)\right] \tag{4.6}$$

Let $M_{t-1}(p) = -\log\left[p\big(\mathbf{r}_0^{t-1}|\hat{\mathbf{x}}_0^{t-1}\big)\right]$ denotes the path metric for the path Π_t through the trellis defined by the sequence $\hat{\mathbf{x}}_0^{t-1}$ and terminating in state p (the minus sign means that we seek to minimize the path metric in order to maximize the likelihood function). Now, let the sequence $\hat{\mathbf{x}}_0^t = \{\hat{\mathbf{x}}_0, \ldots, \hat{\mathbf{x}}_{t-1}, \hat{\mathbf{x}}_t\}$ be obtained by appending the input $\hat{x}_t$ to $\hat{\mathbf{x}}_0^{t-1}$ and suppose the input $\hat{x}_t$ is such that the state at time t+1 is $\Psi_{t+1} = q$. The path metric for this larger sequence is :

$$\begin{aligned} M_t(q) &= -\sum_{i=0}^{t} \log p\big(\mathbf{r}_i|\hat{\mathbf{x}}_i\big) \\ &= -\sum_{i=0}^{t-1} \log p\big(\mathbf{r}_i|\hat{\mathbf{x}}_i\big) - \log p\big(\mathbf{r}_t|\hat{\mathbf{x}}_t\big) \\ &= M_{t-1}(p) - \log p\big(\mathbf{r}_t|\hat{\mathbf{x}}_t\big) \end{aligned} \tag{4.7}$$

Let $\mu_t(\mathbf{r}_t, \hat{\mathbf{x}}_t) = -\log p\big(\mathbf{r}_t|\hat{\mathbf{x}}_t\big)$ denotes the negative log-likelihood a prosteriori probability for the input r_t. The quantity $\mu_t(\mathbf{r}_t, \hat{\mathbf{x}}_t) = -\log p\big(\mathbf{r}_t|\hat{\mathbf{x}}_t\big)$ is called branch metric for the decoder. Since $\hat{\mathbf{x}}_t$ moves the trellis from state p to state q at time t+1, we can write $\mu_t(\mathbf{r}_t, \hat{\mathbf{x}}_t)$ as $\mu_t\big(\mathbf{r}_t, \hat{\mathbf{x}}_t^{(p,q)}\big)$. Then :

$$M_t(q) = M_{t-1}(p) + \mu_t(\mathbf{r}_t, \hat{\mathbf{x}}_t) = M_t(q) = M_{t-1}(p) + \mu_t\big(\mathbf{r}_t, \hat{\mathbf{x}}_t^{(p,q)}\big) \tag{4.8}$$

Thus, the path metric along a path to state q at time t is obtained by adding the path metric to the state p at time t-1 to the branch metric for an input which moves the encoder from state p to state q.

Suppose $M_{t-1}(p_1)$ is the path metric of a path ending at state p_1 at time t and $M_{t-1}(p_2)$ is the path metric of a path ending at state p_2 at time t. Suppose further that both of these states are connected to state q at time t+1. The resulting path metrics to state q are $M_{t-1}(p_1) + \mu_t(\mathbf{r}_t, \hat{\mathbf{x}}^{(p,q)})$ and $M_{t-1}(p_2) + \mu_t(\mathbf{r}_t, \hat{\mathbf{x}}^{(p,q)})$, as shown in the following figure :

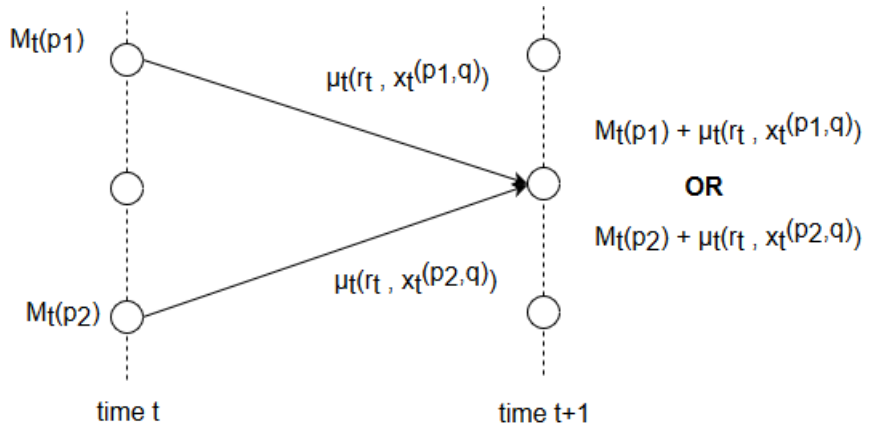

Figure 4.3: Merging paths

The governing principle of Viterbi algorithm is that in order to obtain the shortest path through the trellis, the path to state q must be the shortest possible. Thus, when two or more paths merge, the path with the shortest path metric is retained and the other paths are eliminated from further consideration. That is :

$$M_t(q) = \min_{p_i \in Q} \{ M_{t-1(p_i)} + \mu_t\big(\mathbf{r}_t, \hat{\mathbf{x}}^{(p_i,q)}\big) \} \tag{4.9}$$

where Q denotes the aggregation of the states of the trellis. The path with the minimal length becomes the path to the state q. This path is called survivor path. In the event that the path metrics of merging paths are equal, a random choice can be made with no negative impact on the decoding error probability.

Since it is not known at time $t \leq L$ which states the final path passes through, the paths to each state are found for each time. The Viterbi algorithm, thus, maintains the following data:

- A path metric to each state at time t,
- A path to each state at time t

Cumulatively, the Viterbi algorithm is summarized as:

- For each state q at time t+1, find the path metric for each path to state q by adding the path metric $M_t(q)$ of each survivor path to state p at time t to the branch metric $\mu_t\big(\mathbf{r}_t, \hat{\mathbf{x}}_t^{(p,q)}\big)$,
- The survivor path to q is selected as the path to state q which has the smallest path metric,
- Store the path and the path metric to each state q,
- Increment t and repeat until complete.

Consequently, we can assume that the Viterbi algorithm is a recursive algorithm. In the beginning of the decoding process, we assume that the initial state is always $\Psi_0 = 0$.

In case of BSC with crossover probability $p < \frac{1}{2}$, the received sequence $\mathbf{r}$ is binary and the log-likelihood function (4.4) becomes :

$$\log p(\mathbf{r}_i|\mathbf{a}_i) = d(\mathbf{r}_i, \mathbf{a}_i) \log \frac{p}{1-p} + n \log(1-p) \tag{4.10}$$

where $d(\mathbf{r}_i, \mathbf{a}_i)$ denotes the Hamming distance between $\mathbf{r}_i$ and $\mathbf{a}_i$. Because $\frac{p}{1-p} < 0$ and $n \log(1-p)$ is constant for all $\mathbf{a}$, an maximum likelihood decoding scheme for a BSC chooses $\mathbf{a}$ that minimizes the overall Hamming distance :

$$d(\mathbf{r}, \mathbf{a}) = \sum_{i=0}^{L-1} d(\mathbf{r}_i, \mathbf{a}_i) \tag{4.11}$$

Hence, when Viterbi algorithm is applied to the BSC, $d(\mathbf{r}_i, \mathbf{a}_i)$ becomes the branch metric and the algorithm must find the path through the trellis with the smallest metric, that is, the path closest to $\mathbf{r}$ in Hamming distance.

In case of AWGN channel, the a-posteriori probability $p(\mathbf{r}_i|\mathbf{a}_i)$ is equal to :

$$\begin{aligned} p(\mathbf{r}_i|\mathbf{a}_i) &= p\big(r_i^{(1)},\ldots,r_i^{(n)}|a_i^{(1)},\ldots,a_i^{(n)}\big) \\ &= \prod_{j=1}^{n}\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)\exp\left(-\frac{|r_i^{(j)}-a_i^{(j)}|^2}{2\sigma^2}\right) \\ &= \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n\prod_{j=1}^{n}\exp\left(-\frac{|r_i^{(j)}-a_i^{(j)}|^2}{2\sigma^2}\right) \\ &= \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n\exp\left(-\frac{\sum_{j=1}^{n}|r_i^{(j)}-a_i^{(j)}|^2}{2\sigma^2}\right) \\ &= \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n\exp\left(-\frac{\|\mathbf{r}_i-\mathbf{a}_i\|^2}{2\sigma^2}\right) \end{aligned} \tag{4.12}$$

where $\|\mathbf{r}_i-\mathbf{a}_i\|^2 = |\big(r_i^{(1)}-a_i^{(1)}\big)^2+\big(r_i^{(2)}-a_i^{(2)}\big)^2+\cdots+\big(r_i^{(n)}-a_i^{(n)}\big)^2| = |\sum_{j=1}^{n}\big(r_i^{(j)}-a_i^{(j)}\big)^2| = \sum_{j=1}^{n}|\big(r_i^{(j)}-a_i^{(j)}\big)^2|$ because each term $\big(r_i^{(j)}-a_i^{(j)}\big)^2$ is greater or equal than zero and according to inequality $|a_1+a_2+\cdots+a_n| \le |a_1|+|a_2|+\cdots+|a_n| \Rightarrow |a_1+a_2+\cdots+a_n| = |a_1|+|a_2|+\cdots+|a_n|$ only if $a_i \ge 0$ for each $i=1,2,\ldots,n$, we can conclude that $|\sum_{j=1}^{n}\big(r_i^{(j)}-a_i^{(j)}\big)^2| = \sum_{j=1}^{n}|\big(r_i^{(j)}-a_i^{(j)}\big)^2|$.

As regards the log-likelihood function $\log p(\mathbf{r}|\mathbf{a})$, the channel transition probability $\log p(\mathbf{r}_i|\mathbf{a}_i)$ is equals to :

$$\log p(\mathbf{r}_i|\mathbf{a}_i) = -n\log(\sqrt{2\pi\sigma^2}) - \frac{\|\mathbf{r}_i-\mathbf{a}_i\|^2}{2\sigma^2} \tag{4.13}$$

where $\|\mathbf{r}_i-\mathbf{a}_i\|^2$ is the squared Euclidean distance between $\mathbf{r}_i$ and $\mathbf{a}_i$. The quantity $\log p(\mathbf{r}_i|\mathbf{a}_i)$ becomes the branch metric.

We can further process the equation (4.4) as follows :

$$\begin{aligned} \log p(\mathbf{r}_i|\mathbf{a}_i) &= \log\prod_{j=1}^{n}p\big(r_i^{(j)}|a_i^{(j)}\big) = \log\prod_{j=1}^{n}\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)\exp\left(-\frac{|r_i^{(j)}-a_i^{(j)}|^2}{2\sigma^2}\right) \\ &= -n\log(\sqrt{2\pi\sigma^2}) - \sum_{j=1}^{n}\left(\frac{|r_i^{(j)}-a_i^{(j)}|^2}{2\sigma^2}\right) \\ &= -n\log(\sqrt{2\pi\sigma^2}) - \frac{1}{2\sigma^2}\sum_{j=1}^{n}\Big[(|r_i^{(j)}-a_i^{(j)}|^2)\Big] \\ &= -n\log(\sqrt{2\pi\sigma^2}) - \frac{1}{2\sigma^2}\sum_{j=1}^{n}\Big[(r_i^{(j)})^2-2r_i^{(j)}a_i^{(j)}+(a_i^{(j)})^2\Big] \\ &= -n\log(\sqrt{2\pi\sigma^2}) - \frac{1}{2\sigma^2}\sum_{j=1}^{n}\Big[(r_i^{(j)})^2+(a_i^{(j)})^2\Big] + \frac{1}{\sigma^2}\sum_{j=1}^{n}\Big[r_i^{(j)}a_i^{(j)}\Big] \\ &= -n\log(\sqrt{2\pi\sigma^2}) - \frac{1}{2\sigma^2}\sum_{j=1}^{n}\Big[(r_i^{(j)})^2+(a_i^{(j)})^2\Big] + \frac{1}{\sigma^2}\mathbf{r}_i\mathbf{a}_i \end{aligned} \tag{4.14}$$

In case of a BPSK modulation scheme, we can assume that the above equation depends only on the inner product (correlation) of the received vector $\mathbf{r}_i$ and codeword $\mathbf{a}_i$.

Thus, for a BSC channel , the Viterbi algorithm (MLSD decoding schema) chooses the code sequence which is closest to the channel output $\mathbf{r}$ in a Hamming distance sense, while for a AWGN channel, it chooses the code sequence which is closest to channel output in a Euclidean distance.

After the termination of the Viterbi algorithm, the final state of the Trellis diagram can be any possible state $\Psi_t \in Q$. However, a different approach can be used where the final state of the Trellis diagram should be a specific one, usually the all-zero state; that is, $\Psi_t = Q_0$. In order to achieve this, an all-zero sequence should be appended to the information sequence $\mathbf{x}$ before the encoding stage. The length of this sequence is equal to $k * m$, where m is the memory order of the encoder and which is given by the equation (3.2). This version of Viterbi algorithm is called zero append sequence Viterbi algorithm.

4.2 Code gain of Viterbi algorithm

In this section, we examine the coding gain of the Viterbi algorithm regarding with the performance of an uncoded telecommunication system.

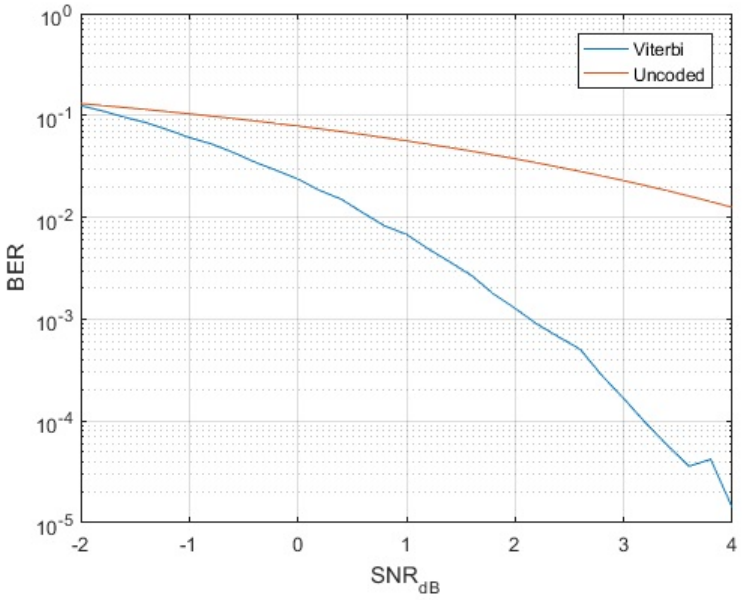

Figure 4.4: Coding gain of Viterbi Algorithm

As we can conclude, the Viterbi algorithm provides performance improvement compared to an uncoded system as it was expected from the theoretical analysis early at this chapter.

Chapter 5

BCJR Algorithm

In 1974, Bahl, Cocke, Jelinek and Raviv introduced a bit-wise maximum a posteriori probability (MAP) decoding algorithm that can be applied to any linear code, either convolutional or block codes, with unequal a priori probabilities for the information bits and trellis structure, which was named BCJR algorithm. The bit-wise MAP criterion minimizes the probability of the bit error rather than the probability of codeword error. This bit-wise MAP criterion is :

$$\tilde{x_t} = arg \max_{x_t} P(a_t = a|\mathbf{r}) = arg \max_{x_t} \frac{P(\mathbf{r}|a_t = a)P(a_t)}{p(\mathbf{r})} \tag{5.1}$$

where $a_t = 2x_t - 1$ and $\mathbf{r} = [r_0\ r_1\ \dots\ ]$. The optimality condition for the BCJR algorithm is slightly different than for the Viterbi algorithm: in MAP decoding, the probability of information bit error is minimized, whereas in ML decoding the probability of codeword error is minimized. In order to minimize the BER, the a posteriori probability $P(X_t = x|\mathbf{r})$ of the information bit x given the received sequence $\mathbf{r}$ must be maximized. The computational complexity of the BCJR is greater than that of the Viterbi algorithm and thus Viterbi decoding is preferred in case of equally likely information bits. But, even in case of equally likely information bits, Viterbi algorithm does not guarantee that BER is minimized Hence, Viterbi algorithm results in near-optimum BER performance compared to BCJR algorithm which results optimum BER performance. However, when the information bits are not equally likely or when iterative decoding is employed and, thus, the a priori probabilities of the information bits change from iteration to iteration, better performance is achieved with MAP decoding. The BCJR algorithm is a SISO decoding algorithm and it has been applied in recent years to soft-decision iterative decoding schemes in which the a priori probabilities of the information bits change from iteration to iteration. While the Viterbi algorithm produces an estimated bit sequence which corresponds to a continuous path through the trellis, the BCJR (and generally MAP algorithm) may not.

5.1 BCJR Algorithm

For an (n,k) code, the received vector $\mathbf{r_t}$ can be represented as $[r_t^{(1)} r_t^{(2)} \dots r_t^{(n)}]$. Thus, the input sequence can be divided as :

$$\mathbf{r} = \left[\mathbf{r}_0 \dots \mathbf{r}_{t-1}\ \mathbf{r}_t\ \mathbf{r}_{t+1} \dots\right] = \left[\mathbf{r}_{<t} \quad \mathbf{r}_t \quad \mathbf{r}_{>t}\right] \tag{5.2}$$

Hence, the a posteriori probability $P(X_t = x|\mathbf{r})$ can be analyzed as :

$$\begin{aligned}
P(X_t = x|\mathbf{r}) &= \frac{1}{p(\mathbf{r})} P(X_t, \mathbf{r}) = \frac{1}{p(\mathbf{r})} P(X_t = x, \mathbf{r}_{<\mathbf{t}}, r_t, \mathbf{r}_{>\mathbf{t}}) \\
&= \frac{1}{p(\mathbf{r})} \sum_{(p,q)\in S_x} p(\Psi_t = p, \Psi_{t+1} = q, \mathbf{r}_{<\mathbf{t}}, r_t, \mathbf{r}_{>\mathbf{t}}) \\
&= \frac{1}{p(\mathbf{r})} \sum_{(p,q)\in S_x} p(\mathbf{r}_{>\mathbf{t}}|\Psi_t = p, \Psi_{t+1} = q, \mathbf{r}_{<\mathbf{t}}, r_t) p(\Psi_t = p, \Psi_{t+1} = q, \mathbf{r}_{<\mathbf{t}}, r_t) \\
&= \frac{1}{p(\mathbf{r})} \sum_{(p,q)\in S_x} p(\mathbf{r}_{>\mathbf{t}}|\Psi_t = p, \Psi_{t+1} = q, \mathbf{r}_{<\mathbf{t}}, r_t) p(\Psi_{t+1} = q, r_t|\mathbf{r}_{<\mathbf{t}}, \Psi_t = p) p(\mathbf{r}_{<\mathbf{t}}, \Psi_t = p) \\
&= \frac{1}{p(\mathbf{r})} \sum_{(p,q)\in S_x} p(\mathbf{r}_{>\mathbf{t}}|\Psi_{t+1} = q) p(\Psi_{t+1} = q, r_t|\Psi_t = p) p(\mathbf{r}_{<\mathbf{t}}, \Psi_t = p)
\end{aligned} \tag{5.3}$$

where S_x denote the set of state transitions (p, q) which corresponds to the input x. Hence,

$$S_x = \{(p, q) : x^{(p,q)} = x\} \tag{5.4}$$

Furthermore, since knowledge of the state at any particular time renders irrelevant knowledge about prior states or received data, we can conclude that $p(\mathbf{r}_{>\mathbf{t}}|\Psi_t = p, \Psi_{t+1} = q, \mathbf{r}_{<\mathbf{t}}, r_t) = p(\mathbf{r}_{>\mathbf{t}}|\Psi_{t+1} = q)$ and $p(\Psi_{t+1} = q, r_t|\mathbf{r}_{<\mathbf{t}}, \Psi_t = p) = p(\Psi_{t+1} = q, r_t|\Psi_t = p)$. In equation (5.3), the factor $\frac{1}{p(\mathbf{r})}$ is common for each $P(X_t = x|\mathbf{r})$ $\forall x \in \mathcal{X}$ and it is nothing more than a normilization factor which does not need to be explicitly computed. Thus, we can proceed as :

$$p(X_t = x|\mathbf{r}) = \sum_{(p,q)\in S_x} p(\mathbf{r}_{>\mathbf{t}}|\Psi_{t+1} = q) p(\Psi_{t+1} = q, r_t|\Psi_t = p) p(\mathbf{r}_{<\mathbf{t}}, \Psi_t = p) \tag{5.5}$$

Based on the above equation (5.3), we define the factors :

$$\alpha_t(p) = p(\mathbf{r}_{<\mathbf{t}}, \Psi_t = p) \tag{5.6}$$

$$\beta_{t+1}(q) = p(\mathbf{r}_{>\mathbf{t}}|\Psi_{t+1} = q) \tag{5.7}$$

$$\gamma_t(p, q) = p(\Psi_{t+1} = q, r_t|\Psi_t = p) \tag{5.8}$$

As a result, equation (5.5) can be written as :

$$p(X_t = x|\mathbf{r}) = \sum_{(p,q)\in S_x} \alpha_t(p)\gamma_t(p, q)\beta_{t+1}(q) \tag{5.9}$$

We can further proceed the computation of $\gamma_t(p, q)$:

$$\begin{aligned}
\gamma_t(p, q) &= p(\Psi_{t+1} = q, r_t|\Psi_t = p) \\
&= p(r_t|\Psi_t = p, \Psi_{t+1} = q) P(\Psi_{t+1} = q|\Psi_t = p) \\
&= p(r_t|\Psi_t = p, \Psi_{t+1} = q) P(X_t = x^{(p,q)}) \\
&= P(X_t = x^{(p,q)}) p(r_t|a_t = a^{(p,q)})
\end{aligned} \tag{5.10}$$

Especially, in the case of AWGN channel, the computation of $\gamma_t(p,q)$ is equal to :

$$\gamma_t(p,q) = P(X_t = x^{(p,q)})\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n \exp\left(-\frac{\|r_t - a^{(p,q)}\|^2}{2\sigma^2}\right) \tag{5.11}$$

where $r_t^{(i)} = a_t^{(i)} + n_t^{(i)}$, $n_t^{(i)} \sim \mathcal{N}(0,\sigma^2)$ and $\sigma^2 = \frac{N_0}{2}$.

Given $a_t(p)$ for all states $p \in \{0,1,\ldots,Q-1\}$, where Q is equal to the number of total states of the decoder ($Q = 2^{km}$), the values $a_{t+1}(q)$ can be forward computed as :

$$\begin{aligned}
a_{t+1}(q) &= p(\mathbf{r_{<t+1}}, \Psi_{t+1} = q) = p(\mathbf{r_{<t}}, r_t, \Psi_{t+1} = q) \\
&= \sum_{p=0}^{Q-1} p(\mathbf{r_{<t}}, r_t, \Psi_t = p, \Psi_{t+1} = q) \\
&= \sum_{p=0}^{Q-1} p(r_t, \Psi_{t+1} = q | \mathbf{r_{<t}}, \Psi_t = p) p(\mathbf{r_{<t}}, \Psi_t = p) \\
&= \sum_{p=0}^{Q-1} p(r_t, \Psi_{t+1} = q | \mathbf{r_{<t}}, \Psi_t = p) p(\mathbf{r_{<t}}, \Psi_t = p) \\
&= \sum_{p=0}^{Q-1} p(r_t, \Psi_{t+1} = q | \Psi_t = p) p(\mathbf{r_{<t}}, \Psi_t = p) \Leftrightarrow \\
a_{t+1}(q) &= \sum_{p=0}^{Q-1} a_t(p)\gamma_t(p,q)
\end{aligned} \tag{5.12}$$

A backward recursion can similarly be developed for $\beta_t(p)$ as :

$$\begin{aligned}
\beta_t(p) &= p(\mathbf{r_{>t-1}} | \Psi_t = p) = p(r_t, \mathbf{r_{>t}} | \Psi_t = p) \\
&= \sum_{q=0}^{Q-1} p(r_t, \mathbf{r_{>t}}, \Psi_{t+1} = q | \Psi_t = p) \\
&= \sum_{q=0}^{Q-1} p(r_t, \Psi_{t+1} = q | \Psi_t = p) p(\mathbf{r_{>t}} | r_t, \Psi_t = p, \Psi_{t+1} = q) \\
&= \sum_{q=0}^{Q-1} p(r_t, \Psi_{t+1} = q | \Psi_t = p) p(\mathbf{r_{>t}} | \Psi_{t+1} = q) \Leftrightarrow \\
\beta_t(p) &= \sum_{q=0}^{Q-1} \gamma_t(p,q)\beta_{t+1}(q)
\end{aligned} \tag{5.13}$$

The $\alpha_t(p)$ probabilities are computed starting at the beginning of the trellis with the set $\alpha_0(p),\ p = \{0,1,\ldots,Q-1\}$, and forward through the trellis. This computation is called forward pass. The $\beta_{t-1}(q)$ probabilities are computed starting at the end of the trellis with the set $\beta_N(p),\ p = \{0,1,\ldots,Q-1\}$, and working backward through the trellis. This computation is called backward pass. Because the computation of $\alpha_t(p$ and $\beta_{t-1}(q)$ is such essential part of the

BCJR algorithm, it is sometimes also referred to as the forward-backward algorithm. Hence, the recursive equations (5.12) and (5.13) are initialized as :

$$\begin{bmatrix} a_0(0) & a_0(1) & \dots & a_0(Q-1) \end{bmatrix} = \begin{bmatrix} 1 & 0 & \dots & 0 \end{bmatrix} \tag{5.14}$$

because the encoder always starts in state 0. If the encoder terminates in the zero state (that is, the zero append sequence has been concatenated with the information sequence in the same manner as in case of the Viterbi algorithm), then :

$$\begin{bmatrix} \beta_N(0) & \beta_N(1) & \dots & \beta_N(Q-1) \end{bmatrix} = \begin{bmatrix} 1 & 0 & \dots & 0 \end{bmatrix} \tag{5.15}$$

On the other hand, if encoder terminates in any state with uniform probabilities, then :

$$\begin{bmatrix} \beta_N(0) & \beta_N(1) & \dots & \beta_N(Q-1) \end{bmatrix} = \begin{bmatrix} \frac{1}{Q} & \frac{1}{Q} & \dots & \frac{1}{Q} \end{bmatrix} \tag{5.16}$$

Since $P(X_t = x|\mathbf{r})$ is a probability mass function, we have $\sum_{x \in S_x} P(X_t = x|\mathbf{r}) = 1$. But ignoring the factor $\frac{1}{p(\mathbf{r})}$ from equation (5.3), this equality lapses. Thus, we can retrieve a posteriori probability $P(X_t = x|\mathbf{r})$ from equation (5.5) using a normalization formula, as :

$$P(X_t = x|\mathbf{r}) = \frac{p(X_t = x|\mathbf{r})}{\sum_{x \in \mathcal{X}} p(X_t = x|\mathbf{r})} \tag{5.17}$$

For notational purposes it is sometimes convenient to express the BCJR algorithm in a matrix formulation. Let

$$\alpha_t = \begin{bmatrix} \alpha_t(0) \\ \alpha_t(1) \\ \vdots \\ \alpha_t(Q-1) \end{bmatrix} \; and \; \beta_t = \begin{bmatrix} \beta_t(0) \\ \beta_t(1) \\ \vdots \\ \beta_t(Q-1) \end{bmatrix}$$

which are initialized as $[1 \; 0 \; \dots \; 0]^T$ and $[1 \; 0 \; \dots \; 0]^T$ or $[\frac{1}{Q} \frac{1}{Q} \dots \frac{1}{Q}]^T$, respectively. Further, we define the matrix G_t, which is the probability matrix with elements $g_{t,i,j}$ defined by $g_{t,i,j} = \gamma_t(i,j)$. Thus, we can express the equation (5.12) as :

$$\alpha_{t+1} = G_t^T \alpha_t \tag{5.18}$$

and the equation (5.13) as :

$$\beta_t = G_t \beta_{t+1} \tag{5.19}$$

Moreover, we need to define a matrix describing all the possible transitions in the trellis. Let $T(x)$ be defined with elements $t_{i,j}(x)$ by :

$$t_{i,j}(x) = \begin{cases} 1 & \text{if } (i,j) \text{ is a state with } x^{(i,j)} = x \\ 0 & \text{otherwise} \end{cases} \tag{5.20}$$

for each $x \in \mathcal{X}$. Thus, the equation (5.5) can be expressed as :

$$p(X_t = x|\mathbf{r}) = \begin{bmatrix} \alpha_t^T \big(T(x) \odot G_t\big) \beta_{t+1} \end{bmatrix} \tag{5.21}$$

where $\odot$ denotes the element-by-element multiplication of the matrices $T(x)$ and G_t.

In case of an (n,k) systematic convolutional code, the equation (5.5) can be further developed as :

$$\begin{aligned}
p(X_t = x|\mathbf{r}) &= \sum_{(p,q)\in S_x} p(\mathbf{r}_{>\mathbf{t}}|\Psi_{t+1}=q)p(\Psi_{t+1}=q, r_t|\Psi_t=p)p(\mathbf{r}_{<\mathbf{t}}, \Psi_t=p) \\
&= \sum_{(p,q)\in S_x} p(\mathbf{r}_{>\mathbf{t}}|\Psi_{t+1}=q)p(r_t|\Psi_t=p, \Psi_{t+1}=q)p(\Psi_{t+1}=q|\Psi_t=p)p(\mathbf{r}_{<\mathbf{t}}, \Psi_t=p) \\
&= \sum_{(p,q)\in S_x} p(\mathbf{r}_{>\mathbf{t}}|\Psi_{t+1}=q)p(r_t|\Psi_t=p, \Psi_{t+1}=q)p(X_t=x^{(p,q)})p(\mathbf{r}_{<\mathbf{t}}, \Psi_t=p) \\
&= P(X_t=x) \sum_{(p,q)\in S_x} p(\mathbf{r}_{>\mathbf{t}}|\Psi_{t+1}=q)p(r_t|\Psi_t=p, \Psi_{t+1}=q)p(\mathbf{r}_{<\mathbf{t}}, \Psi_t=p) \\
&= P(X_t=x) \sum_{(p,q)\in S_x} p(\mathbf{r}_{>\mathbf{t}}|\Psi_{t+1}=q)p(r_t^{(1:k)}, r_t^{(k+1:n)}|\Psi_t=p, \Psi_{t+1}=q)p(\mathbf{r}_{<\mathbf{t}}, \Psi_t=p) \\
&= P(X_t=x) \sum_{(p,q)\in S_x} p(\mathbf{r}_{>\mathbf{t}}|\Psi_{t+1}=q)p(r_t^{(1:k)}|\Psi_t=p, \Psi_{t+1}=q)p(r_t^{(k+1:n)}|\Psi_t=p, \Psi_{t+1}=q)p(\mathbf{r}_{<\mathbf{t}}, \Psi_t=p) \\
&= P(X_t=x) \sum_{(p,q)\in S_x} p(\mathbf{r}_{>\mathbf{t}}|\Psi_{t+1}=q)p(r_t^{(1:k)}|X_t=x^{(p,q)})p(r_t^{(k+1:n)}|a_t=a^{(k+1:n,p,q)})p(\mathbf{r}_{<\mathbf{t}}, \Psi_t=p) \\
&= P(X_t=x)p(r_t^{(1:k)}|X_t=x) \sum_{(p,q)\in S_x} p(\mathbf{r}_{>\mathbf{t}}|\Psi_{t+1}=q)p(r_t^{(k+1:n)}|a_t=a^{(k+1:n,p,q)})p(\mathbf{r}_{<\mathbf{t}}, \Psi_t=p)
\end{aligned} \tag{5.22}$$

where $r_t^{(1:k)} = [r_t^{(1)}\ r_t^{(2)}\ \dots\ r_t^{(k)}]$ and $r_t^{(k+1:n)} = [r_t^{(k+1)}\ r_t^{(k+2)}\ \dots\ r_t^{(n)}]$ correspond to the systematic bits and the parity bits of the received vector r_t, respectively. Moreover, $p(r_t^{(1:k)}, r_t^{(k+1:n)}|\Psi_t) = p, \Psi_{t+1}=q)$ is equal to $p(r_t^{(1:k)}|\Psi_t=p, \Psi_{t+1}=q)p(r_t^{(k+1:n)}|\Psi_t=p, \Psi_{t+1}=q)$ because both AWGN and BSC are memoryless channels. In the special case of an (2,1) systematic convolutional encoder and BPSK signal mapper, the above equation is equal to :

$$p(X_t=x|\mathbf{r}) = P(X_t=x)p(r_t^{(1)}|X_t=x) \sum_{(p,q)\in S_x} p(\mathbf{r}_{>\mathbf{t}}|\Psi_{t+1}=q)p(r_t^{(2)}|a_t=a^{(2,p,q)})p(\mathbf{r}_{<\mathbf{t}}, \Psi_t=p) \tag{5.23}$$

This special case is essential in turbo coding. Based on (5.23), we define the following quantities :

$$P_{p,t}(x) = P(X_t=x) \tag{5.24}$$

$$P_{s,t}(x) = p(r_t^{(1)}|X_t=x) \tag{5.25}$$

and

$$\begin{aligned}
P_{e,t}(x) &= \sum_{(p,q)\in S_x} p(\mathbf{r}_{>\mathbf{t}}|\Psi_{t+1}=q)p(r_t^{(2)}|\Psi_t=p, \Psi_{t+1}=q)p(\mathbf{r}_{<\mathbf{t}}, \Psi_t=p) \\
&= \sum_{(p,q)\in S_x} p(\mathbf{r}_{>\mathbf{t}}|\Psi_{t+1}=q)p(r_t^{(2)}|a_t=a^{(2,p,q)})p(\mathbf{r}_{<\mathbf{t}}, \Psi_t=p) \\
&= \sum_{(p,q)\in S_x} \alpha_t(p)p(r_t^{(2)}|a_t=a^{(2,p,q)})\beta_{t+1}(q)
\end{aligned} \tag{5.26}$$

where $P_{p,t}(x)$ and $P_{s,t}(x)$ are called Prior and Systematic probabilities, respectively. The quantity $P_{e,t}(x)$ is called Extrinsic Information. In assotiation with the equations (5.5) and (5.9), we can conclude that, in case of systematic coding, $\gamma_t(p,q)$ is equal to :

$$\gamma_t(p,q) = p(r_t^{(2)}|a_t = a^{(2,p,q)}) \tag{5.27}$$

Thus, (5.23) can be rewritten as :

$$p(X_t = x|\mathbf{r}) = P_{p,t}(x)P_{s,t}(x)P_{e,t}(x) \tag{5.28}$$

In case of a AWGN channel, $\gamma_t(p,q)$ can be computed as :

$$\gamma_t(p,q) = \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n \exp\left(-\frac{\|r_t - a^{(p,q)}\|^2}{2\sigma^2}\right) \tag{5.29}$$

The Prior probability $P_{p,t}(x)$ represents the a priori information available about the bits before any decoding occurs, arising from a source other than the received systematic or parity data. It is sometimes called Intrinsic Information, to distinguish it from the Extrinsic Information. In the iterative decoder, after the first iteration, the prior is obtained from the other decoder.

The Systematic probability $P_{s,t}(x)$ represents the information about x_t explicitly available from the measurement of the uncoded information $r_t^{(1)}$. This is equal to the a posterior probability.

The value $P_{e,t}$ is called Extrinsic Information of the systematic BCJR and it is the information produced by the decoder based on the received sequence and prior information, but excluding the information from the received systematic bit $r_t^{(1)}$ and the prior information related to the bit X_t. Thus, we can conclude that the extrinsic information depends only on the information which produced exclusively by the code. As we note from equations (5.12) and (5.13), both $\alpha_t(p)$ and $\beta_{t+1}(q)$ do not depend on X_t, but only on the received data at other times. Also, we note that $p(r_t^{(2)}|\alpha_t = \alpha^{(2,p,q)})$ does not depend on the received systematic information $r_t^{(1)}$. Thus, the extrinsic probability is independent from the information conveyed by the systematic data about X_t. The extrinsic probability $P_{e,t}(x)$ conveys all the information about $P(X_t = x)$ that is available from the structure of the code, separate from information which is obtained from an observation of X_t via $r_t^{(1)}$ or from prior information. In case of nonsystematic BCJR, as extrinsic information is used the whole quantity $P(X_t = x|\mathbf{r})$. The extrinsic information is an important part of the turbo decoding algorithm. It is the information passed between decoders to represent the prior probability.

Finally, the BCJR algorithm for an AWGN channel, is summarized as follows :

1. Initialize $\alpha_0(p)$, $\forall p = \{0, \ldots, Q-1\}$, according to the equation (5.14),

2. Compute and store $\gamma_t(p,q)$, $\forall t = 0, 1, \ldots, \tau$, based on either equation (5.11), if the code is nonsystematic, or on equation (5.29), if the code is systematic

3. For $t = 1, 2, \ldots, \tau$ compute the next values of $\alpha_t(q)$ using the recursive equation (5.12) for each $q = \{0, 1, \ldots, Q-1\}$

4. Initialize $\beta_N(q)$, $\forall q = \{0, 1, \ldots, Q-1\}$, according to the equation (5.15), if the encoder terminates in state all-zero, or according to the equation (5.16), if the encoder terminates in a random state

5. For $t = 1, 2, \ldots, \tau$ compute the next values of $\beta_t(p)$ using the recursive equation (5.13) for each $p = \{0, 1, \ldots, Q-1\}$

6. Compute the a posteriori probability $p(X_t = x|r)$ based on either the equation (5.9), if the code is nonsystematic, or based on the equation (5.28), if the code is systematic

7. Finally, for each $t = 0, 1, \ldots, \tau$, we choose the symbol x with the bigger a posteriori probability

5.1.1 Log BCJR Algorithm

For encoders with a single binary input, the log likelihood ration is usually used in the detection problem. In the special case we use BPSK as signal mapper with mapping function $2x - 1$, the log likelihood ration of an (2,1) systematic convolutional code is equal to :

$$\begin{aligned}
\lambda_t(\mathbf{r}) &= \log\left[\frac{P(X_t=1|\mathbf{r})}{P(X_t=0)|\mathbf{r})}\right] = \log\left[\frac{\frac{1}{p(\mathbf{r})}P(X_t=1,\mathbf{r})}{\frac{1}{p(\mathbf{r})}P(X_t=0,\mathbf{r})}\right] = \log\left[\frac{P(X_t=1,\mathbf{r})}{P(X_t=0,\mathbf{r})}\right] \\
&= \log\left[\frac{\sum_{(p,q)\in S_1} p(\Psi_t=p,\Psi_{t+1}=q,\mathbf{r})}{\sum_{(p,q)\in S_0} p(\Psi_t=p,\Psi_{t+1}=q,\mathbf{r})}\right] = \log\left[\frac{\sum_{(p,q)\in S_1} \alpha_t(p)\gamma_t(p,q)\beta_{t+1}(q)}{\sum_{(p,q)\in S_0} \alpha_t(p)\gamma_t(p,q)\beta_{t+1}(q)}\right] \\
&= \log\left[\frac{p(X_t=1)p(r_t^{(1)}|X_t=1)\sum_{(p,q)\in S_1} p(\mathbf{r}_{>\mathbf{t}}|\Psi_{t+1}=q)p(r_t^{(2)}|a_t=a^{(2,p,q)})p(\mathbf{r}_{<\mathbf{t}},\Psi_t=p)}{p(X_t=0)p(r_t^{(1)}|X_t=0)\sum_{(p,q)\in S_0} p(\mathbf{r}_{>\mathbf{t}}|\Psi_{t+1}=q)p(r_t^{(2)}|a_t=a^{(2,p,q)})p(\mathbf{r}_{<\mathbf{t}},\Psi_t=p)}\right] \\
&= \log\left[\frac{P_{p,t}(1)P_{s,t}(1)P_{e,t}(1)}{P_{p,t}(0)P_{s,t}(0)P_{e,t}(0)}\right] = \log\left[\frac{P_{p,t}(1)}{P_{p,t}(0)}\right] + \log\left[\frac{P_{s,t}(1)}{P_{s,t}(0)}\right] + \log\left[\frac{P_{e,t}(1)}{P_{e,t}(0)}\right]
\end{aligned} \tag{5.30}$$

Based on the above equation, we can define the following terms :

$$\lambda_{p,t} = \log\left[\frac{P_{p,t}(1)}{P_{p,t}(0)}\right] = \log\left[\frac{p(X_t=1)}{p(X_t=0)}\right] \tag{5.31}$$

$$\lambda_{s,t} = \log\left[\frac{P_{s,t}(1)}{P_{s,t}(0)}\right] = \log\left[\frac{p(r_t^{(1)}|X_t=1)}{p(r_t^{(1)}|X_t=0)}\right] \tag{5.32}$$

$$\lambda_{e,t} = \log\left[\frac{P_{e,t}(1)}{P_{e,t}(0)}\right] = \log\left[\frac{\sum_{(p,q)\in S_1} p(\mathbf{r}_{>\mathbf{t}}|\Psi_{t+1}=q)p(r_t^{(2)}|a_t=a^{(2,p,q)})p(\mathbf{r}_{<\mathbf{t}},\Psi_t=p)}{\sum_{(p,q)\in S_0} p(\mathbf{r}_{>\mathbf{t}}|\Psi_{t+1}=q)p(r_t^{(2)}|a_t=a^{(2,p,q)})p(\mathbf{r}_{<\mathbf{t}},\Psi_t=p)}\right] \tag{5.33}$$

which are called Log Prior Ratio, Log Systematic Ratio and Extrinsic Information, respectively. Consequently, the equation (5.30) can be written as :

$$\lambda_t(\mathbf{r}) = \lambda_{p,t} + \lambda_{s,t} + \lambda_{e,t} \tag{5.34}$$

Thus, the log likelihood ratio is expressed as the sum of the log of the posterior probabilities for the systematic bits, plus the log likelihood ration of the prior probabilities, plus the extrinsic information.

Especially, in case of AWGN channel with variance σ^2 and BPSK signal mapper, the Log Systematic Ratio, which is also called soft channel input, can be computed as :

$$\begin{aligned}
\lambda_{s,t} &= \log\left[\frac{p(r_t^{(1)}|X_t=1)}{p(r_t^{(1)}|X_t=0)}\right] = \log\left[\frac{\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(r_t^{(1)}-\sqrt{E_c})^2}{2\sigma^2}\right)}{\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(r_t^{(1)}-\sqrt{E_c})^2}{2\sigma^2}\right)}\right] \\
&= \frac{-1}{2\sigma^2}\left[\left(r_t^{(1)}-\sqrt{E_c}\right)^2+\left(r_t^{(1)}+\sqrt{E_c}\right)^2\right] \\
&= \frac{-1}{2\sigma^2}\left[\left(r_t^{(1)}\right)^2-2\sqrt{E_c}r_t^{(1)}+E_c-\left(r_t^{(1)}\right)^2-2\sqrt{E_c}r_t^{(1)}-E_c\right] \\
&= \frac{4r_t^{(1)}\sqrt{E_c}}{2\sigma^2} = \frac{2r_t^{(1)}\sqrt{E_c}}{\sigma^2} = L_c r_t^{(1)}
\end{aligned} \tag{5.35}$$

where L_c is called channel reliability and it is equal to :

$$L_c = \frac{2\sqrt{E_c}}{\sigma^2} = \frac{2\sqrt{RE_b}}{\sigma^2} \tag{5.36}$$

where E_c refers to the power of a codeword and E_b refers to the power of each bit.

Since $P(X_t=1)+P(X_t=0)=1$, we can compute the prior probabilities straight from Log Prior Ratio as :

$$\exp(\lambda_{p,t}) = \frac{P(X_t=1)}{1-P(X_t=1)} \Leftrightarrow \begin{cases} P(X_t=1) = \frac{1}{1+\exp(-\lambda_{p,t})} \\ P(X_t=0) = \frac{1}{1+\exp(\lambda_{p,t})} \end{cases} \tag{5.37}$$

The extrinsic information $\lambda_{e,t}$ is the information that is passed from one decoder to the next as the turbo decoding algorithm progresses. This extrinsic information can be computed directly from equation (5.34) as :

$$\lambda_{e,t} = \lambda_t(\mathbf{r}) - \lambda_{p,t} - \lambda_{s,t} \tag{5.38}$$

The decision on transmitted symbol x at each time unit is taken based on the sign of the total log likelihood ration. Thus, we have that :

$$\lambda_t(\mathbf{r}) = \log\left[\frac{P(X_t=1|\mathbf{r})}{P(X_t=0)|\mathbf{r})}\right] \underset{x_t=0}{\overset{x_t=1}{\lessgtr}} 0 \tag{5.39}$$

Finally, the log BCJR algorithm for an AWGN channel and BPSK as signal mapper, is summarized as follows :

1. Initialize and compute the terms $\alpha_t(p)$, $\gamma_t(p,q)$ and $\beta_{t+1}(q)$ as described in the previous section.

2. Compute the log likelihood ratio of equation (5.33) starting from computing the sum term over set S_1 and then over set S_0.

3. Compute the Log Prior and the Log Systematic ratios of equations (5.31) and (5.32), respectively.

4. Sum the Log Likelihood Ratios according to equation (5.34) and take a decision on the transmitted symbol x based on the $sign\big[\lambda_t(\mathbf{r})\big]$.

5.1.2 Max-Log BCJR Algorithm

The log likelihood ratio of an (2,1) convolutional code can be rewritten as :

$$\begin{aligned}\lambda_t(\mathbf{r}) &= \log\left[\frac{P(X_t=1|\mathbf{r})}{P(X_t=0|\mathbf{r})}\right] = \log\left[\frac{\sum_{(p,q)\in S_1}\alpha_t(p)\gamma_t(p,q)\beta_{t+1}(q)}{\sum_{(p,q)\in S_0}\alpha_t(p)\gamma_t(p,q)\beta_{t+1}(q)}\right] \\ &= \log\left[\frac{\sum_{(p,q)\in S_1} exp\big(\tilde{\alpha}_t(p)+\tilde{\gamma}_t(p,q)+\tilde{\beta}_{t+1}(q)\big)}{\sum_{(p,q)\in S_0} exp\big(\tilde{\alpha}_t(p)+\tilde{\gamma}_t(p,q)+\tilde{\beta}_{t+1}(q)\big)}\right]\end{aligned} \tag{5.40}$$

where $\tilde{\alpha}_{t+1}(q) = \log[\alpha_t(q)]$, $\tilde{\beta}_t(p) = \log[\beta_t(p)]$ and $\tilde{\gamma}_t(p,q) = \log[\gamma_t(p,q)]$.

It is known that the $max(x,y)$ function is equal to $log[\frac{e^x+e^y}{1+e^{-|x-y|}}]$, that is,

$$\begin{aligned}&\max(x,y) = \log\left[\frac{e^x+e^y}{1+e^{-|x-y|}}\right] = \log(e^x+e^y) - \log(1+e^{-|x-y|}) \Leftrightarrow \\ &\log(e^x+e^y) = \max(x,y) + \log(1+e^{-|x-y|})\end{aligned} \tag{5.41}$$

Then, we define the function :

$$\max_{x,y}^{*}(x,y) = \log(e^x+e^y) = \max(x,y) + \log(1+e^{-|x-y|}) \tag{5.42}$$

It can be proven that we can apply recursively the max* function to sums of more than two exponential terms. For example, $\max^*(x,y,z) = \max^*\big[\max^*(x,y),z\big]$.

As a consequence, equation (5.40) is equal to :

$$\begin{aligned}\lambda_t(\mathbf{r}) &= \log\left[\sum_{(p,q)\in S_1} exp\big(\tilde{\alpha}_t(p)+\tilde{\gamma}_t(p,q)+\tilde{\beta}_{t+1}(q)\big)\right] - \log\left[\sum_{(p,q)\in S_0} exp\big(\tilde{\alpha}_t(p)+\tilde{\gamma}_t(p,q)+\tilde{\beta}_{t+1}(q)\big)\right] \\ &= \max_{S_1}^{*}\Big(\tilde{\alpha}_t(p)+\tilde{\gamma}_t(p,q)+\tilde{\beta}_{t+1}(q)\Big) - \max_{S_0}^{*}\Big(\tilde{\alpha}_t(p)+\tilde{\gamma}_t(p,q)+\tilde{\beta}_{t+1}(q)\Big)\end{aligned} \tag{5.43}$$

Moreover, the recursive equations (5.12) and (5.13) can be, respectively, calculated as :

$$\begin{aligned}\tilde{\alpha}_{t+1}(q) &= \log[\alpha_t(q)] = \log\left[\sum_{p=0}^{Q-1}\alpha_t(p)\gamma_t(p,q)\right] \\ &= \log\left[\sum_{p=0}^{Q-1} exp(\tilde{\alpha}_t(p)+\tilde{\gamma}_t(p,q))\right] \\ &= \max_{p}^{*}\big(\tilde{\alpha}_t(p)+\tilde{\gamma}_t(p,q)\big),\ \forall p\in\{0,1,\ldots,Q-1\}\end{aligned} \tag{5.44}$$

and

$$\begin{aligned}\tilde{\beta}_t(p) &= \log[\beta_t(p)] = \log\left[\sum_{q=0}^{Q-1}\beta_{t+1}(q)\gamma_t(p,q)\right] \\ &= \log\left[\sum_{q=0}^{Q-1} exp(\tilde{\beta}_{t+1}(q)+\tilde{\gamma}_t(p,q))\right] \\ &= \max_q{}^*\big(\tilde{\beta}_{t+1}(q)+\tilde{\gamma}_t(p,q)\big),\ \forall q\in\{0,1,\ldots,Q-1\}\end{aligned} \tag{5.45}$$

Generally, in case of AWGN channel, the computation of $\tilde{\gamma}_t(p,q)$ according to (5.11) is equal to:

$$\tilde{\gamma}_t(p,q) = \log\big[\gamma_t(p,q)\big] = \log\left[P\Big(X_t = x^{(p,q)}\Big)\right] + n\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \frac{\|r_t - a^{(p,q)}\|^2}{2\sigma^2} \tag{5.46}$$

where, in our case, $n = 2$. Further, we can ignore the term $n\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)$ as a constant and independent of the algorithm.

The recursive equations (5.44) and (5.45) are initialized as :

$$\big[\tilde{\alpha}_0(0)\ \tilde{\alpha_0}(1)\ \ldots\ \tilde{\alpha_0}(Q-1)\big] = \big[0\ -\infty\ \ldots\ -\infty\big] \tag{5.47}$$

and as

$$\big[\tilde{\beta_N}(0)\ \tilde{\beta_N}(1)\ \ldots\ \tilde{\beta_N}(Q-1)\big] = \big[0\ -\infty\ \ldots\ -\infty\big] \tag{5.48}$$

if encoder terminates to zero state, or

$$\big[\tilde{\beta_N}(0)\ \tilde{\beta_N}(1)\ \ldots\ \tilde{\beta_N}(Q-1)\big] = \big[-\log Q\ -\log Q\ \ldots\ -\log Q\big] \tag{5.49}$$

if encoder terminates to a random state, respectively.

Now, if we intend to use this type of log likelihood ration BCJR algorithm in turbo coding, we can further continue our analysis based on the assumption that our code is an (2,1) systematic code and we use BPSK as signal mapper. Starting from the log likelihood MAP rule, we have that :

$$\begin{aligned}L(x_t) &= \log\left[\frac{P(x_t=1|r)}{P(x_t=0|r)}\right] = \log\left[\frac{\sum_{(p,q)\in S_1}\alpha_t(p)\gamma_t(p,q)\beta_{t+1}(q)}{\sum_{(p,q)\in S_0}\alpha_t(p)\gamma_t(p,q)\beta_{t+1}(q)}\right] \\ &= log\left[\frac{\sum_{(p,q)\in S_1}\alpha_t(p)P(X_t=1)p(r_t|a_t=a^{(p,q)})\beta_{t+1}(q)}{\sum_{(p,q)\in S_0}\alpha_t(p)P(X_t=0)p(r_t|a_t=a^{(p,q)})\beta_{t+1}(q)}\right] \\ &= \log\left[\frac{P(x_t=1)}{P(x_t=0)}\right] + \log\left[\frac{\sum_{(p,q)\in S_1}\alpha_t(p)p(r_t|a_t=a^{(p,q)})\beta_{t+1}(q)}{\sum_{(p,q)\in S_0}\alpha_t(p)p(r_t|a_t=a^{(p,q)})\beta_{t+1}(q)}\right] \\ &= \log\left[\frac{P(x_t=1)}{P(x_t=0)}\right] + \log\left[\frac{\sum_{(p,q)\in S_1}\exp\big(\tilde{\alpha}_t(p)+\log\big(p(r_t|a_t=a^{(p,q)})\big)+\tilde{\beta}_{t+1}(q)\big)}{\sum_{(p,q)\in S_0}\exp\big(\tilde{\alpha}_t(p)+\log\big(p(r_t|a_t=a^{(p,q)})\big)+\tilde{\beta}_{t+1}(q)\big)}\right] \\ &= \log\left[\frac{P(x_t=1)}{P(x_t=0)}\right] + \max_{S_1}{}^*\Big[\tilde{\alpha}_t(p)+\log\big(p(r_t|a_t=a^{(p,q)})\big)+\tilde{\beta}_{t+1}(q)\Big] - \\ &- \max_{S_0}{}^*\Big[\tilde{\alpha}_t(p)+\log\big(p(r_t|a_t=a^{(p,q)})\big)+\tilde{\beta}_{t+1}(q)\Big]\end{aligned} \tag{5.50}$$

Based on the above equation, we set the following term :

$$L^e(x_t) = \log\left[\frac{Pr(X_t = 1)}{Pr(X_t = 0)}\right] \tag{5.51}$$

Thus, the a priori probability of each symbol $x \in \mathcal{F}_2$ at a specific time stamp t can be computed as :

$$\begin{aligned} P(x_t) &= \left(\frac{\exp\left[\frac{-L^e(x_t)}{2}\right]}{1+\exp\left[\frac{-L^e(x_t)}{2}\right]}\right)\exp\left[a_t\frac{L^e(x_t)}{2}\right] \\ &= A_t\exp\left[a_t\frac{L^e(x_t)}{2}\right] \end{aligned} \tag{5.52}$$

where a_t stands for the modulated version of corresponding x_t. In equations (5.56), the term $\log\left(p(r_t|a_t = a^{(p,q)})\right)$ for an (2,1) systematic code is equal to :

$$\log\left(p(r_t|a_t = a^{(p,q)})\right) = \log\left(\frac{1}{2\pi\sigma^2}\right) - \frac{\|r_t - a^{(p,q)}\|^2}{2\sigma^2} \tag{5.53}$$

where the first term can be ignored since it is independent of x_t. Moreover, the quantity $\|r_t - a^{(p,q)}\|^2$ is equal to :

$$\begin{aligned} \|r_t - a^{(p,q)}\|^2 &= \left(r_t^{(1)} - a^{(1,p,q)}\right)^2 + \left(r_t^{(2)} - a^{(2,p,q)}\right)^2 \\ &= \left(r_t^{(1)}\right)^2 - 2r_t^{(1)}a^{(1,p,q)} + \left(a^{(1,p,q)}\right)^2 + \left(r_t^{(2)}\right)^2 - 2r_t^{(2)}a^{(2,p,q)} + \left(a^{(2,p,q)}\right)^2 \end{aligned} \tag{5.54}$$

Substitution of (5.53) and (5.54) into (5.56) yields :

$$\begin{aligned} \lambda_t(\mathbf{r}) = L^e(x_t) + \\ + \max_{S_1}{}^*\left[\tilde{\alpha}_t(p) - \frac{\left(r_t^{(1)}\right)^2}{2\sigma^2} + \frac{r_t^{(1)}a^{(1,p,q)}}{\sigma^2} - \frac{\left(a^{(1,p,q)}\right)^2}{2\sigma^2} - \frac{\left(r_t^{(2)}\right)^2}{2\sigma^2} + \frac{r_t^{(2)}a^{(2,p,q)}}{\sigma^2} - \frac{\left(a^{(2,p,q)}\right)^2}{2\sigma^2} + \tilde{\beta}_t(q)\right] - \\ - \max_{S_0}{}^*\left[\tilde{\alpha}_t(p) - \frac{\left(r_t^{(1)}\right)^2}{2\sigma^2} + \frac{r_t^{(1)}a^{(1,p,q)}}{\sigma^2} - \frac{\left(a^{(1,p,q)}\right)^2}{2\sigma^2} - \frac{\left(r_t^{(2)}\right)^2}{2\sigma^2} + \frac{r_t^{(2)}a^{(2,p,q)}}{\sigma^2} - \frac{\left(a^{(2,p,q)}\right)^2}{2\sigma^2} + \tilde{\beta}_t(q)\right] \end{aligned} \tag{5.55}$$

where only the terms $\frac{r_t^{(1)}a^{(1,p,q)}}{\sigma^2}$ and $\frac{r_t^{(2)}a^{(2,p,q)}}{\sigma^2}$ survive after the subtraction. Thus, our final

equation is determined as :

$$\begin{aligned}
\lambda_t(\mathbf{r}) &= L^e(x_t) + \max_{S_1}{}^*\left[\tilde{\alpha}_t(p) + \frac{r_t^{(1)}a^{(1,p,q)}}{\sigma^2} + \frac{r_t^{(2)}a^{(2,p,q)}}{\sigma^2} + \tilde{\beta}_t(q)\right] - \\
&- \max_{S_0}{}^*\left[\tilde{\alpha}_t(p) + \frac{r_t^{(1)}a^{(1,p,q)}}{\sigma^2} + \frac{r_t^{(2)}a^{(2,p,q)}}{\sigma^2} + \tilde{\beta}_t(q)\right] \\
&= L^e(x_t) + \max_{S_1}{}^*\left[\tilde{\alpha}_t(p) + \frac{r_t^{(1)}(+1)}{\sigma^2} + \frac{r_t^{(2)}a^{(2,p,q)}}{\sigma^2} + \tilde{\beta}_t(q)\right] - \\
&- \max_{S_0}{}^*\left[\tilde{\alpha}_t(p) + \frac{r_t^{(1)}(-1)}{\sigma^2} + \frac{r_t^{(2)}a^{(2,p,q)}}{\sigma^2} + \tilde{\beta}_t(q)\right] \\
&= L^e(x_t) + \max_{S_1}{}^*\left[\tilde{\alpha}_t(p) + \frac{r_t^{(1)}}{\sigma^2} + \frac{r_t^{(2)}a^{(2,p,q)}}{\sigma^2} + \tilde{\beta}_t(q)\right] - \max_{S_0}{}^*\left[\tilde{\alpha}_t(p) - \frac{r_t^{(1)}}{\sigma^2} + \frac{r_t^{(2)}a^{(2,p,q)}}{\sigma^2} + \tilde{\beta}_t(q)\right] \\
&= L^e(x_t) + 2\frac{r_t^{(1)}}{\sigma^2} + \max_{S_1}{}^*\left[\tilde{\alpha}_t(p) + \frac{r_t^{(2)}a^{(2,p,q)}}{\sigma^2} + \tilde{\beta}_t(q)\right] - \max_{S_0}{}^*\left[\tilde{\alpha}_t(p) + \frac{r_t^{(2)}a^{(2,p,q)}}{\sigma^2} + \tilde{\beta}_t(q)\right]
\end{aligned} \tag{5.56}$$

Compare to (5.34), we can conclude that $\lambda_{p,t} = L^e(x_t)$ and $\lambda_{s,t} = 2\frac{r_t^{(1)}}{\sigma^2}$, since it dependents only on the systematic part of the received vector r_t. Hence, as extrinsic information, we define the quantity :

$$\max_{S_1}{}^*\left[\tilde{\alpha}_t(p) + \frac{r_t^{(2)}a^{(2,p,q)}}{\sigma^2} + \tilde{\beta}_t(q)\right] - \max_{S_0}{}^*\left[\tilde{\alpha}_t(p) + \frac{r_t^{(2)}a^{(2,p,q)}}{\sigma^2} + \tilde{\beta}_t(q)\right] \tag{5.57}$$

As we mentioned earlier, in turbo decoding algorithm, the extrinsic information produced from one decoder is used as prior information to the other decoder. Thus, for decoder 1, the equation (5.56) can be expressed as :

$$\lambda_t(\mathbf{r}) = L_{21}^e(x_t) + 2\frac{r_t^{(1)}}{\sigma^2} + L_{12}^e(x_t) \tag{5.58}$$

where $L_{21}^e(x_t)$ denotes the extrinsic information which is transferred from decoder 2 to decoder 1. Thus, this log likelihood ratio is used as prior information to the decoder 1. Respectively, $L_{12}^e(x_t)$ denotes the extrinsic information which is transferred from decoder 1 to decoder 2. Hence, this log likelihood ratio is used as prior information from decoder 2. Similarly, for decoder 2, the equation (5.56) can be expressed as :

$$\lambda_t(\mathbf{r}) = L_{12}^e(x_t) + 2\frac{r_t^{(1)}}{\sigma^2} + L_{21}^e(x_t) \tag{5.59}$$

In equation (5.41), we can ignore the term $\log(1 + e^{-|x-y|})$, which is called correction term, and, thus, the function $\max_{x,y}{}^*(x, y)$ becomes equal to the original $\max_{x,y}(x, y)$ function. This type of Max-Log BCJR algorithm is suboptimal than the other versions of BCJR algorithm, but it has almost comparable complexity to Viterbi algorithm. Essentially, this type of BCJR is

like two iterations of Viterbi algorithm, one forward recursion, which is equivalent to a forward Viterbi, and one backward recursion, which is equivalent to a backward Viterbi algorithm. The performance loss of this version of BCJR algorithm is more pronounced when it is used for iterative decoding, since the approximation error accumulates as additional iterations are performed.

Chapter 6

Turbo Codes

Turbo Codes were introduced in 1993 as an approach to error correction coding which could provide very long codewords with relatively modest decoding complexity. Turbo codes have been also termed as parallel concatenated convolutional codes. They are based on the idea of parallel concatenated coding and the iterative decoding algorithms (Turbo Principle). Turbo coding consists of two fundamentals ideas; a code design that produces a code with randomlike properties, and a decoder design that makes use of soft-output values and iterative decoding. Because the decoding complexity is small for the dimension of the code, very long codes are possible through them, so that the bounds of Shannon's channel coding theorem become, for all practical purposes, achievable.

6.1 Encoder

The turbo encoder consists of two (or more) systematic codes which share message data via interleavers as shown in figure (2.1). In its most conventional realization, the codes are obtained from recursive (feedback) systematic convolutional (RSC) codes, because they have a better performance than feedforward codes.

The basic system of turbo coding consists of an information sequence x, two (2,1) systematic recursive convolutional encoders and an interleaver. The information sequence x contains K^* information bits plus m termination bits to return the first encoder to the all-zero state, where m is the memory order of the first encoder. Hence, the information sequence is considered to be a block of total length $K = K^* + m$. The interleaver reorders or permutes the K bits in the information sequence so that the second encoder receives a permuted information sequence $x^{'}$ different from the first. Hence, the second encoder may or may not return to the all-zero state. The performance degradation produced by an unknown final state of the second encoder is negligible when a pseudo-random interleaver is used with a large interleaving size N. Moreover, because the interleaver is part of the turbo code design, a complete maximum likelihood decoder for the entire codeword would be prohibitively complex. However, because more than one code is used, it is possible to employ simple soft-in soft-out (SISO) decoders for each constituent code in an iterative fashion, in which the soft output values of one decoder are passed to the other, and vice versa, until the final decoding estimation is obtained. This iterative decoding approximation is suboptimal. The overall code length has $N = 3K$ and rate $R = \frac{K^*}{N} = \frac{K-m}{3K} \approx \frac{1}{3}$. To achieve performance near to Shannon limit, the information block length K^* should be chosen to be very

large, usually at least several thousand bits. The best performance at moderate BERs down to about 10^{-5} is achieved with short constraint length (and, as a consequence, small memory order) constituent encoders, typically $v = 4$ or less.

In conclusion, the basic structure of turbo encoder consists of two binary rate-$\frac{1}{2}$ recursive systematic convolutional (RSC) encoders separated by an interleaver. Thus, the information sequence is encoded twice, with an interleaver between the two encoders serving to make the two encoded data sequences approximately statistically independent of each other. Both RSC encoders use the same generator matrix **G** because there is no evidence for any performance improvement, in case of different constituent encoders are used. Finally, the encoded sequences are multiplexed and transmitted.

Hence, the standard structure of a turbo encoder is :

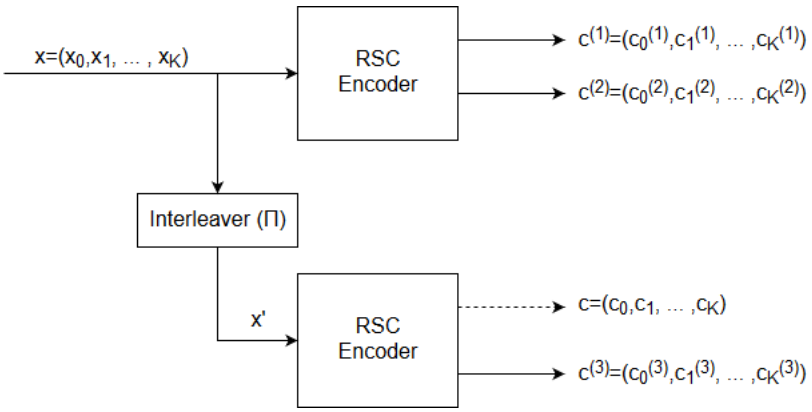

Figure 6.1: Turbo Encoder

where $x^{'}$ is the interleaved version of the input sequence x and the output sequence c is the systematic part of the overall output sequence produced by the second decoder. This sequence c is not transmitted. The polynomial generator matrix $G(D)$ of both constituent RSC encoders can be represented as :

$$G(D) = \left[1 \quad \frac{a(D)}{b(D)}\right]$$

Often, the two parity sequences $c^{(2)}$ and $c^{(3)}$ are punctured before being transmitted. This puncturing of the parity information allows a higher coding rate to be realised. Note that the systematic sequence are rarely punctured, since this degrades the performance of the code more dramatically than puncturing the parity sequences.

6.2 Decoder

In this section, we will describe the structure of each turbo decoder based on the different versions of BCJR algorithm that described previously. Generally, a turbo decoder use two soft-in soft-out decoders linked by inteleavers in a structure similar to that of that of the encoder. A key development in turbo codes is the iterative decoding algorithm. In the iterative decoding algorithm, decoders for each constituent encoder take turns operating on the received data. Each decoder produces an estimation of the probabilities of the transmitted symbols,that is, they are soft-output decoders. These probabilities of the transmitted symbols, which are called extrinsic information, are being reordered in a proper way and being passed to the other decoder, where they are used as a priori probabilities. Further, each constituent decoder takes two additional inputs; the systematic encoded channel output bits and the parity bits transmitted from the associated encoder. Hence, the decoder is also a soft-input decoder. These soft inputs along with the soft outputs provide not only an estimation on the initial information sequence, but also a likelihood ratio which gives the probability that a bit has been correctly decoded. In the first iteration the first decoder provides a soft output giving an estimation of the original information sequence based on the channel inputs alone. It also provides an extrinsic information for each bit of the information sequence x, which is used by the second decoder as a priori information. After that, the second decoder takes action, generating both his own estimation on the information sequence and his extrinsic information, which is passed back to the first decoder. Finally, the whole process starts again till a certain number of iterations elapses. Note that, with every iteration the BER of the decoded bits tends to fall. However, the improvement in performance obtained with increasing numbers of iterations decreases as the number of iterations increases. Hence, for complexity reasons, usually only about 10 iterations are used.

6.2.1 BCJR

This is the basic structure of a turbo decoder which is based on the equation (5.9) or (5.28).

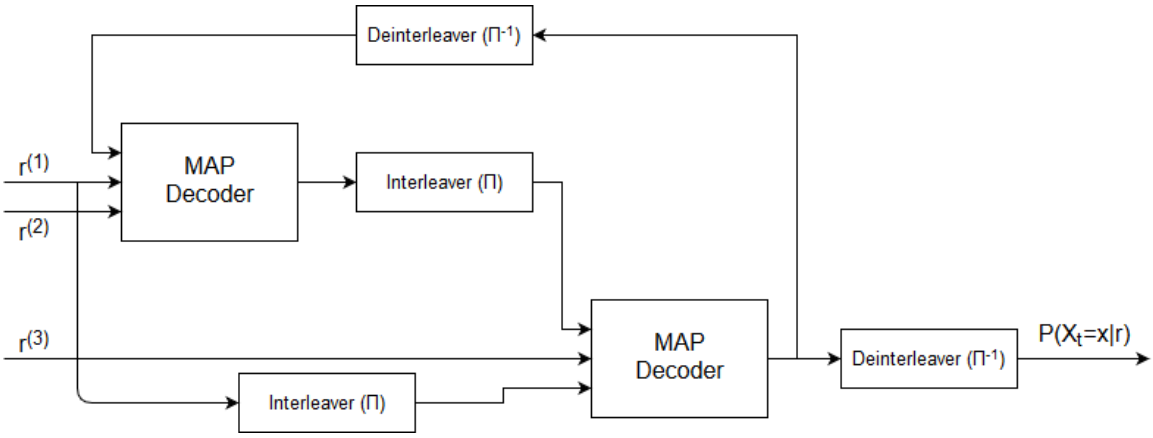

Figure 6.2: Turbo Decoder using BCJR algorithm

Initially, the received sequence $\mathbf{r}$ is divided into its structural sequences $r^{(1)}$, $r^{(2)}$ and $r^{(3)}$, which correspond to the systematic, encoded and interleaved-encoded sequences, respectively.

Afterwards, the concatenation of $r^{(1)}$ and $r^{(2)}$ sequences is driven to decoder 1 with uniform prior probabilities. At the same time, $r^{(1)}$ sequence is driven to the interleaver in order to be permuted and match with the order of $r^{(3)}$ sequence. When decoder 1 terminates its decoding process, it pipeline the extrinsic information to the second decoder (decoder 2), who use it as a priori probabilities for his own process. After the end of its decoding process, the decoder 2 passes its extrinsic information back to the decoder 1, in order to start the whole procedure again. These steps constitute an iteration. The turbo decoder repeats its decoding schema for 10 to 20 iterations, or until the decoder determines that the process has converged. Finally, when the iterative process stops, the decoder 2 produce an estimation about the information sequence $\mathbf{x}$.

6.2.2 Log BCJR

This version of a turbo decoder is based on the log likelihood BCJR as described from equation (5.34).

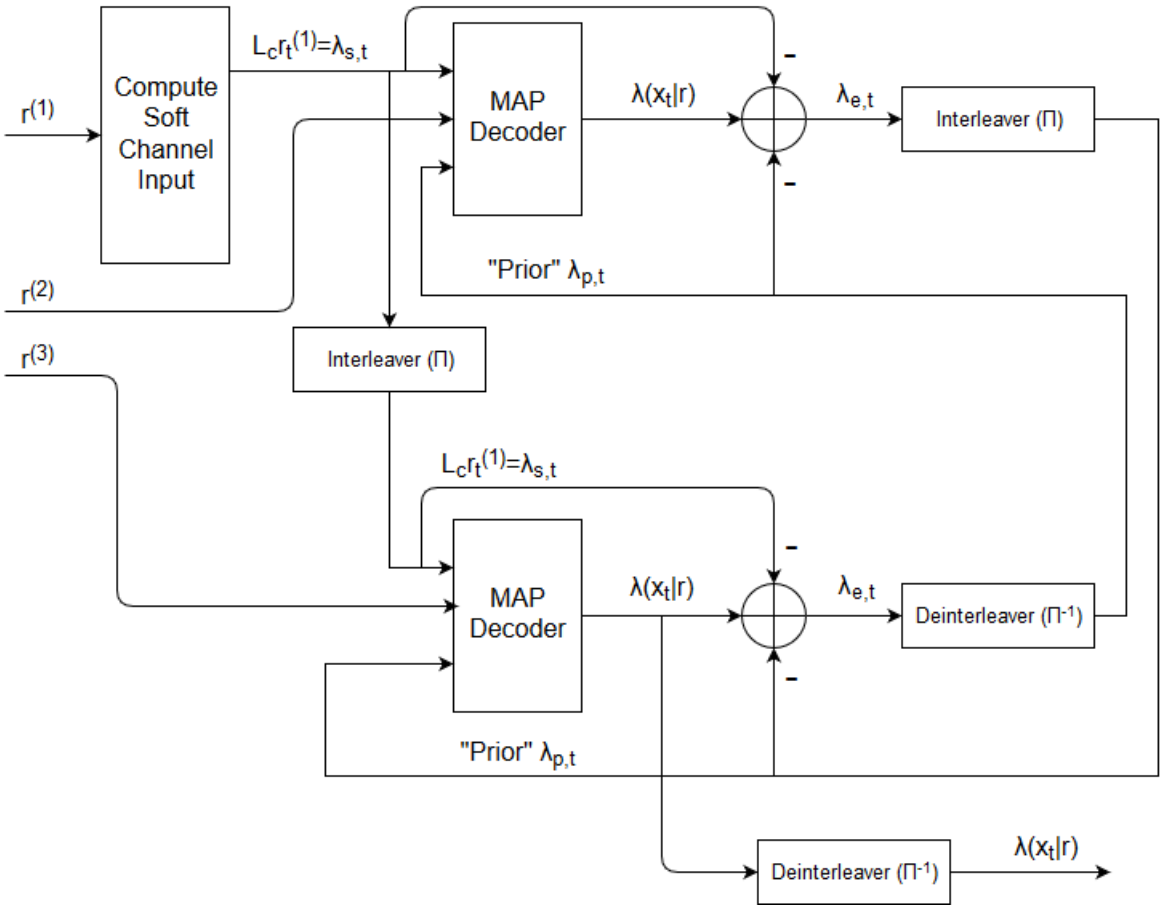

Figure 6.3: Turbo Decoder using Log BCJR

Initially, the demultiplexed sequence $r^{(1)}$ that corresponds to the systematic part of the received sequence, passes through the "Compute Soft Channel Input" module in order to com-

pute the log systematic ratio $\lambda_{s,t} = L_c r^{(1)}$. Then, turbo decoding process starts, as described previously. As we can see, the extrinsic information is computed based on the equation (5.38).

This version of a turbo decoder is based on the log likelihood BCJR as introduced by Prof. Shu Lin and it is described by the equation (5.56).

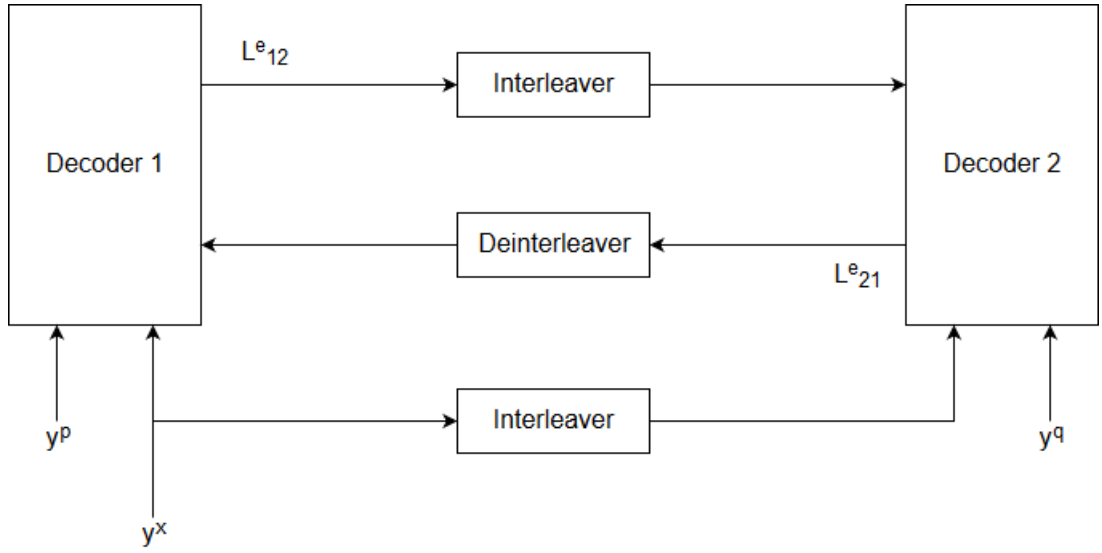

Figure 6.4: Turbo Decoder using Log BCJR

where the quantities L_{12}^e and L_{21}^e denote the extrinsic information which is passed from decoder 1 to decoder 2, and vice versa.

6.3 Stopping Criterion

As the decoding approaches the performance limit of a given turbo code, any further iteration results in very little improvement. Therefore, it is important to devise an efficient criterion to stop the iteration process and prevent unnecessary computations and decoding delay. Generally, decoding process can be stopped, and a final decoding estimate declared, after some fixed number of iterations (usually on the order of 10 - 20) or based on a stopping criterion that is designed to detect when the estimate is reliable with very high probability. One such stopping criterion has been devised based on the cross entropy between the distributions of the estimates at the outputs of the decoders at each iteration. This criterion is known as cross entropy criterion. It effectively stops the iteration process with very little performance degradation.

Another method of terminating the iterative decoding process is to use systematic cyclic redundancy checks (CRC). In the encoder, the input information is encoded by a eRe code, and the output of the CRC encoder is passed to the turbo encoder. In the receiver, the turbo decoder output at each iteration is fed to the eRe error detector. If the CRC code detects no errors in the decoder output, the iterative decoding is terminated.

Other stopping criterions are Sign Change Ratio (SCR) Criterion and Hard Decision Aided (HDA) Criterion.

Chapter 7

Interleaver

Interleaving is a process of rearranging the ordering of a data sequence in a one-to-one deterministic format. The inverse of this process is called deinterleaving which restores the received sequence to its original order. Thus, an interleaver takes a sequence of symbols and permutes them. At the receiver, the sequence is permuted back into the original order by a deinterleaver.

The definition of interleaver can be expressed as, an interleaving function of size N is any bijective function from $\Pi : \{1, \ldots, N\} \rightarrow \{1, \ldots, N\}$. Given an array $\mathbf{x}$ of length N, where $\mathbf{x} = [x_1, \ldots, x_N]^T \in \mathcal{X}^N$, an interleaver is a function $f_\Pi : \mathcal{X}^N \rightarrow \mathcal{X}^N$, where $\tilde{\mathbf{x}} = f_\Pi(\mathbf{x})$ with $\tilde{x}_k = x_\Pi(k)$ for each $k \in \{1, \ldots, N\}$. We abuse the notation and write $\tilde{x}_k = \Pi(x)$ instead of $\tilde{x}_k = x_\Pi(k)$.

The interleaver is an essential part of the Turbo Codes because it breaks the structure that has been introduced by the encoders and, as a result, the Turbo Codes become randomlike. Thus, Turbo Codes can achieve low decoding error probabilities, as Shannon's noisy channel coding theorem implies. The interleaver in turbo coding is a scrambler defined by a permutation of N elements with no repetition. The interleaving is employed before the information data is encoded by the second component encoder. The role of the interleaver is to decorrelate the inputs to the two decoders so that an iterative suboptimal decoding algorithm based on information exchange between the two component decoders can be applied. If the input sequences to the two component decoders are decorrelated, there is a high probability that after the correction of some of the errors in one decoder some of the remaining errors should become correctable in the second decoder. The interleaving pattern should be known to both encoder and decoder.

In the original Turbo Code schema, a pseudorandom block interleaver is used where the information is written row by row and read out following a non-uniform rule, based on randomly generated numbers. The interleaver of a turbo code affects the performance of the code at both low and high SNRs. Precisely, the length of the interleaver is critical for the code performance, particularly at low SNR while the structure of the interleaver is important for the performance at high SNR, as it affects the distance properties of the overall turbo code. It determines the code free distance which has a dominant effect on the asymptotic performace.

7.1 Spectral Thinning

The turbo code error performance is determined by the code distance spectrum. The interleaver in a turbo encoder can reduce the error coefficients of low weight codewords through a pro-

cess called spectral thinning. Specifically, in the parallel concatenated coding schema, as turbo coding, there has been a shift from the lower-weight codewords to higher-weight codewords relative to the convolutional code. This shifting of low-weight codewords toward higher weights in the parallel concatenation of feedback convolutional encoders has been termed spectral thinning and results when interleaving causes almost all the low-weight parity sequences in the first constituent code to be matched with high-weight parity sequences in the second constituent code.

The difficulties of low-weight codewords notwithstanding, turbo codes are outstanding performers. The interleaver ensures that if the input parity sequence has low weight, the output parity sequence has higher weight with high probability. The distance spectrum of a code is a listing of the (W_d, A_d) information as a function of the codeword weight d, W_d, and A_d which is the multiplicity of the codewords at weight d. Turbo codes are said to have sparse distance spectrum if the multiplicities of the low-weight codewords is relatively small. Higher multiplicities result in more contribution to the probability of error, so that a higher SNR must be achieved before the probability of error term becomes negligible.

The result of the thinned spectrum is that there are relatively few codewords of low weight, hence relatively few codewords near to other codewords. Thus codewords selected at random will, with high probability, be decoded correctly. Spectral thinning has little effect on the minimum distance, but it greatly reduces the multiplicities of the low weight codewords. Further, there is only a small spectral thinning effect if feedforward constituent encoders are used. This is an other reason why feedback encoders are used in turbo coding.

7.2 Interleaver Analysis

Turbo codes typically do not have large minimum distances causes the performance curve flatten out at BERs below 10^{-5}. This phenomenon is called error floor and it is due to the unusual weight distribution of the Turbo Codes. Interleavers can be designed to improve the minimum distance of the code and ,thus, lowering the error floor. Hence, the role of the interleaver at Turbo Coding is to add randomlike behavior and improves the error correction performance of the turbo coding schema.

The performance of a turbo code with RSC component codes is affected by the interleaver. The structure of the interleaver is important for the performance at high SNR's, as it affects the distance properties of the overall turbo code. It determines the code free distance which has a dominant effect on the asymptotic performance. There are two different approaches in the construction procedure of an interleaver, the deterministic (block and convolutional) and the pseudorandom interleavers. Generally, in order to achieve better error correction performance in turbo coding, it is important that low-weight parity sequences from the first encoder get matched with high-weight parity sequences from the second encoder almost all the time. This requires that the interleaver breaks the patterns in the input sequences that produce low-weight parity sequences after the encoding procedure. Interleavers with structure, such as block or convolutional, tend to preserve too many of these "bad" input patterns, resulting in poor matching properties and limited spectral thinning. Pseudorandom interleavers, on the other hand, break up almost all the bad patterns and thus achieve the full effect of spectral thinning. Hence, the best interleavers reorder the bits in a pesudorandom manner. It is important that the interleaver has pseudorandom properties.

The length N of the interleaver is critical for the code's performance, particularly at low SNR. In fact, the sparse distance spectrum is typically for long interleavers. Specifically, as the block length (interleaver size) N increases, the spectral thinning of the code becomes more dramatic. Increasing the size of the interleaver has the effect of randomizing the information sequence at the input of the second decoder. A general rule is that, as the size N of the corresponding interleaver increases, the weight distribution of parallel concatenated convolutional codes begins to approximate a randomlike distribution, that is, the distribution that would result if each bit in every codeword were selected randomly from an independent and identically distributed probability distribution. This rule can be also expressed as, for even larger values of N, the codeword and bit multiplicities of the low-weight codewords in the turbo code weight spectrum are reduced by roughly a factor of N, the interleaver size, compared with the terminated convolutional code. The interleaver size N is significantly larger than the code memory v and the interleaver vector elements are chosen randomly. The basic role of the interleaver is to construct a long block code from small memory convolutional codes, as long codes can approach the Shannon capacity limit. Secondly, it spreads out burst errors. More precise, by increasing the interleaver size N times, the bit error probability is reduced by a factor N. This is called the interleaving performance gain. Hence, long interleavers will be required in order to ensure that the soft information exchanged between the SISO modules remains highly uncorrelated.

Traditional block or convolutional interleavers do not work well in turbo coding, particularly when the block length is large. Hence, block or convolutional interleavers are used in turbo coding schema when the block lengths N of the interleaver is relatively short. However, due to their deterministic definition, they are more easy to be implemented. On the other hand, pseudorandom interleavers generate a weight spectrum that has the same characteristics as the binomial distribution, which is equivalent to the weight spectrum assumed by Shannon in his random-coding proof of the noisy-channel coding theorem. Pseudorandom interleavers offer a better performance than block interleavers in Turbo Codes but they are more complex in implementation and they affect overall channel coding delay.

By increasing the number of iterations in the decoding process the bit error probability approaches the channel capacity. The final role of the interleaver is to break low weight input sequences, and hence increase the code free Hamming distance or reduce the number of codewords with small distances in the code distance spectrum.

7.3 Block Interleavers

Traditional block or convolutional interleavers do not work well in turbo coding, particularly when the block length is large. A block interleaver of length N formats the input sequence in a matrix of m rows and n columns, such that N = m x n. The number of rows, m, in the interleaver matrix is also called the interleaver degree (or depth) and the number of columns, n, is called the interleaver span.

Block (Rectangular) interleavers and deinterleavers are easy to implement. However, they may fail to break certain low weight input patterns. Generally, a block interleaver is a memory matrix $m \times n$ filled with channel-coded symbols on a row-by-row basis, which are then read out on a column-by-column basis, as it is shown in the below diagram (7.1) :

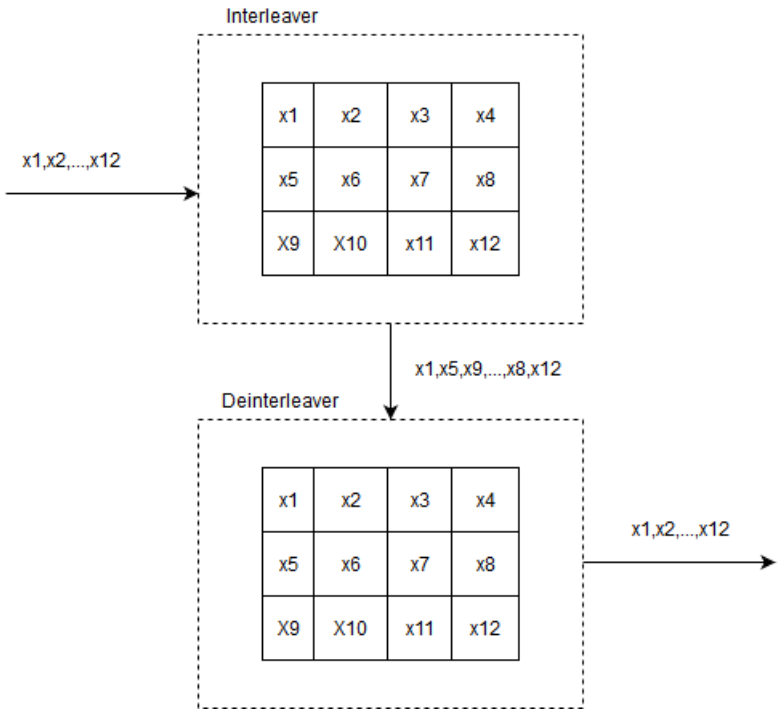

Figure 7.1: A 3x4 block interleaver and deinterleaver

A rectangular interleaver, which is the easiest from an implementation point of view, leads to degraded coder performance compared to a pseudorandom interleaver, because it can leads to a large value of N_{free}. Thus, it is important to use an interleaver which is closer to a true random interleaver.

7.4 Pseudorandom Interleavers

Pseudorandom interleavers generate a weight spectrum that has many of the same characteristics as the binomial distribution, which is equivalent to the weight spectrum assumed by Shannon in his noisy channel coding theorem. Turbo coding with pseudorandom interleaving results in a way of constructing codes with weight distribution similar to a binomial and a simple, near optimal iterative decoding scheme.

Pseudorandom interleaving patterns can be generated in many ways. One way is by using a primitive polynomial to generate a maximum-length shift-register sequence whose cycle structure determines the permutation. Another method uses a computationally simple algorithm based on the quadratic congruence :

$$c_m = \frac{Nm(m+1)}{2} \mod K, 0 \leq m < N \tag{7.1}$$

to generate an index mapping function $c_m \rightarrow c_{m+1}(modK)$, where N is the interleaver size, and N is an odd integer. For K a power of 2, it can be shown that these quadratic interleavers have statistical properties similar to those of randomly chosen interleavers, and thus they give good performance when used in turbo coding. Other good interleaving patterns can be generated by

varying N and r, and the special case $r = \frac{K}{2}$ results in an interleaver that simply interchanges pairs of indices. This special case is particularly interesting in terms of implementation, since the interleaving and deinterleaving functions are identical. Finally, when K is not a power of 2, the foregoing algorithm can be modified to generate similar permutations with good statistical properties.

7.5 Effect of interleaver

At this point, we will compare how the size and the type of the interleaver (block or random) affect the performance of Turbo coding. Further, we will examine how the number of iteration during decoding procedure will affects the performance of Turbo coding. The Turbo Coding schema that we use to extract the results is based on the basic (3,1) turbo code.

7.5.1 Different size of interleaver

In this section, we compare how the size of interleaver affects the accumulative performance of turbo decoding schema for both block and random interleavers. The decoding algorithm is executed for 10 repeats in order to ensure that the final BER will not be affected by the insufficient number of iterations. Below, there are three different figures which illustrate the performance of the turbo decoding for both block and pseudorandom inteleavers with size 100, 1000 and 10000 bits which will be determined as small, medium and large size interleavers, respectively.

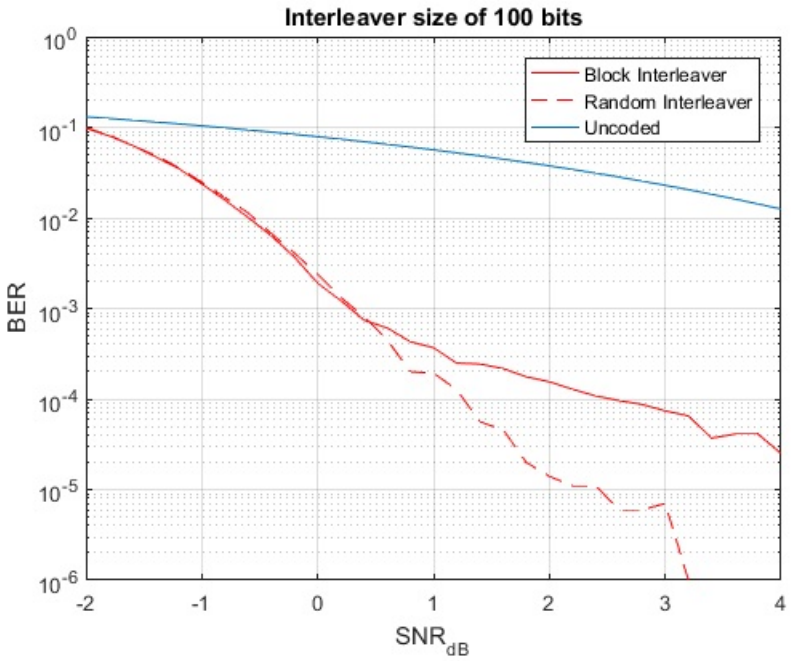

Figure 7.2: BER for interleaver of 100 bits size

As we can conclude from the above figure, for this small interleaver size, both types of interleavers (block and random) perform about the same, especially for small SNRs. This small size of interleaver is used during real time applications where little delay is desired.

At the following figure, we examine the performance of a medium size interleavers (interleaver size of 1000 bits).

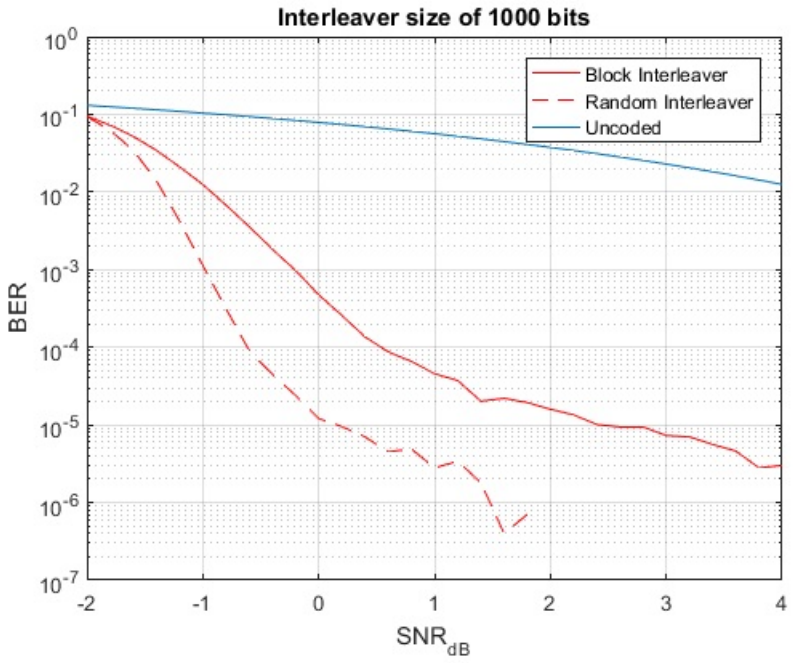

Figure 7.3: BER for interleaver of 1000 bits size

As we can conclude, for this medium size of interleaver, the overall performance of both types of interleaver is better than the performance of the small size interleaver. Hence, while we increase the interleaver size, we can conclude that the overall performance of the code is increased till the theoretical performance bound (error floor) is achieved. However, the BER for the pseudorandom interleaver is smaller than the respective performance of block interleaver. Thus, we can expect that the error floor performance should be reached faster for the (pseudo)random than for the block interleaver.

At the next figure, we examine the performance of a large size interleaver (interleaver size of 10000 bits).

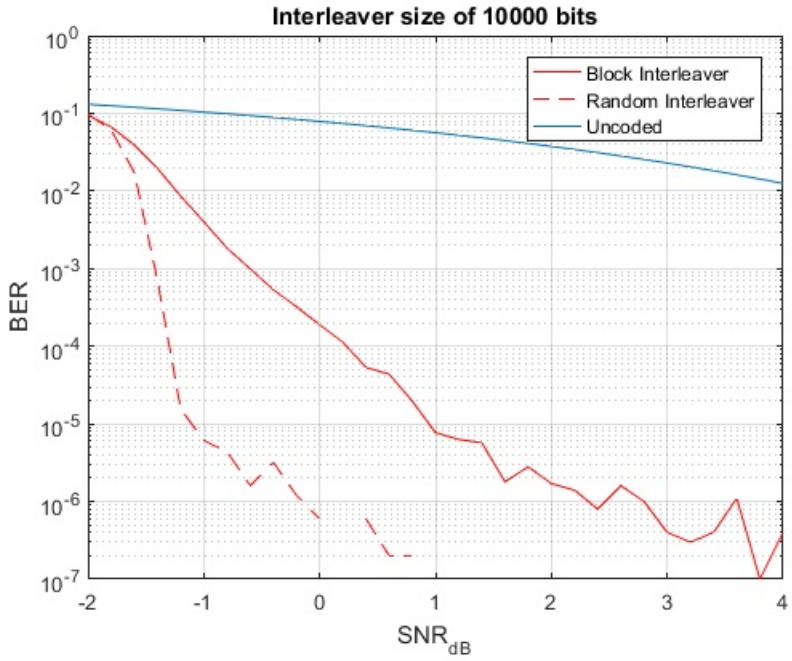

Figure 7.4: BER for interleaver of 10000 bits size

As we can conclude from the above figure of both large size block and random interleavers, the overall performance is greater than the performance of all the previous interleavers' sizes. Further, as we can see, the coding gain between the block and the random interleaver is greater than the coding gain between both interleavers' tyoe at the previous cases. Additionally, the random interleaver has reached the theoretical performance's limit of Turbo Coding, while the performance of the block one can be further improved.

As a summary of the previous results, we can conclude that while the size of the interleaver is increased, the overall performance of the code is improved. Further, for small interleaver size, the coding gain of both type of interleavers is approximately the same. However, while the interleaver size is increased, the coding gain of random interleaver is improved rapidly regarding the improvement's rate of the coding gain for the block interleaver.

Chapter 8

Conclusions

In this thesis, we examined the individual parts of the Turbo Coding schema and how the size of the interleaver effects the overall coding performance. We concluded that as the interleaver size is increased, the code performance is increased. However, when pseudorandom interleavers are used, the coding gain is improved more rapidly as the size of the interleaver is upsized in comparison with the case where block (non-random) interleavers are used. Further, while the size of the interleaver is increased (or while the number of decoding iterations is increased), the total delay of the Turbo Coding algorithm is increased. Thus, small size is preferred when the application is time dependent (as the voice transmission application). In this case, both type of interleavers perform similar for a certain SNR. Hence, a block interleaver realization is preferred over a pseudorandom one, because it is more easily to be implemented as part of hardware configuration of a telecommunication system.