A REDUCED COMPLEXITY DECODING ALGORITHM FOR TURBO PRODUCT CODES

Shirin Esfandiari

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements for the Degree of Master of Applied Science at Concordia University

Montreal, Quebec, Canada

August 2004

Your file Votre référence
ISBN: 0-612-94696-7
Our file Notre référence
ISBN: 0-612-94696-7

ABSTRACT

A REDUCED COMPLEXITY DECODING ALGORITHM FOR TURBO PRODUCT CODES

Shirin Esfandiari

For effective communication to take place between a source and a destination the emphasis lies on the reliable transport of information. Given the constraints of the transport medium, there is much emphasis on the research techniques that allow a good trade off between complexity and performance. Despite the superior performance of turbo product codes, one main concern is the implementation complexity of such systems. With this in mind, in this thesis we present, a reduced complexity decoding algorithm for turbo product codes. This scheme is based on the reduction of the complexity of a soft input soft output trellis based iterative decoder by means of simplifying the trellis structure. We present the details involved in pruning certain branches based on the values of the received channel information and the extrinsic information associated to each branch. We introduce the concept of branch pruning by means of using a threshold, and investigate the methods for compensating for the performance degradation (in terms of bit error rate) in a system where a structural complexity simplification such as trellis pruning is in effect.

Dedicated to my beloved parents and brothers...

ACKNOWLEGMENTS

I would like to express my sincerest gratitude to Dr. M.R. Soleymani for supervising my work. His in depth knowledge, guided patience, his devotion both to research and his graduate students, has been a source of motivation and an inspiration for me to pursue my graduate studies within the Electrical Engineering Department at Concordia University.

I am also grateful to all of my colleagues at the Wireless and Satellite Communication Laboratory for their acquaintance and support. But most importantly I would like to thank Mr. Pouryia Sadeghi and Mrs. Neda Ehtiati, for the gift of their valuable friendship, and all of their suggestions and much appreciated advice.

Most importantly, I would like to thank my loving parents, and my two brothers, for being such great role models and teaching me the values of life throughout their love and understanding, and always encouraging me to learn more; without their support none of this would have been possible.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ACRONYMS

FEC	Forward Error Correction
ARQ	Automated Repeat Request
BER	Bit Error Rate
SNR	Signal to Noise Ratio
AWGN	Additive White Gaussian Noise
BPSK	Binary Phase Shift Keying
SISO	Soft Input Soft Output
APP	A Posteriori Probability
LLR	Log Likelihood Ratio
VA	Viterbi Algorithm
MAP	Maximum A Posteriori
ML	Most Likely
FSM	Finite State Machine
HAD	Hard Decision Aided
SCR	Sign Change Ratio
BCH	Bose-Chaudhuri-Hocquenghem code
BCJR	Bahl-Cocke-Jelinek-Raviv
LAN	Local Area Network
DMC	Discrete Memoryless Channel

Chapter 1

INTRODUCTION

Throughout history reliable and quick communications over long distances has been a major goal, and a great deal of human effort has been exerted towards reaching this objective. The Roman Empire, for example, constructed a communications network ranging more than 4,500 kilometers based on smoke signals. In 1794, the first mechanical optical telegraph used a network of signal flags mounted on towers to transmit messages across Europe. Commercial electrical telegraph service began in 1844 when Morse sent a message from Washington D.C. to Baltimore. Today, telecommunications is a fascinating, rapid paced industry that affects every aspect of our lives including simple voice telephone calls, access to the Internet, high speed data communications, satellite communications, video conferencing and cable TV just to name a few. In Latin, the word *tele* signifies distance, hence telecommunication is distance communication, however thanks to the amazing developments in this field the word "distance" is hardly an obstacle and research is continuously innovating itself to make flexibility, accessibility and

freedom synonymous with personal communications. However with such a tremendous growth in mobile communications, great emphasis is placed on high-speed, reliable and most importantly cost effective techniques, allowing for an error free transmission of any kind of information in a typical communication system.

1.1 The Communication Model

For effective communication to take place between a source and a destination the emphasis lies on the reliable transport of information. A fundamental problem with the communication between sender and receiver is that errors or distortions can arise during the transport through the communication channel as a result of noise acting upon that medium. Depending on the requirements imposed by the receiver, the transport of information must be error-free to a certain degree. It must therefore be possible to correct errors, or the transmission must be good enough that certain errors, which are considered to be less serious, can be tolerated. Error control coding, or channel coding, is a method of adding redundancy to information so that it can be transmitted over a noisy channel to another party, and subsequently be checked and corrected for errors occurring during transmission. Channel coding is especially beneficial for wireless and multimedia applications, such as cellular phone communications and high definition television broadcasting. It is also favorable in deep space and satellite communications, and digital communication and storage. Figure 1.1 shows the basic layout of how information is transmitted and received in an error control coded channel.

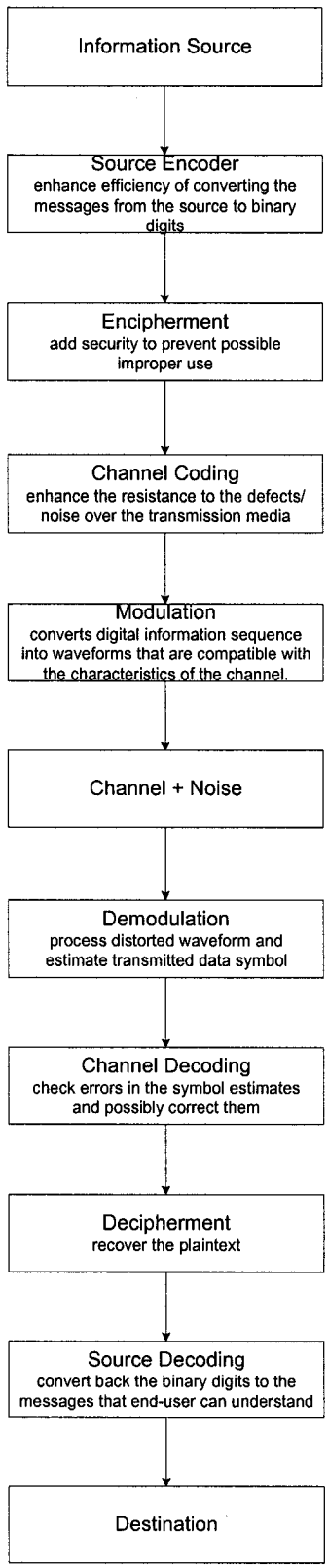

Fig. 1.1: Communication System Model

1.2 Basic Concept of Error Correcting Codes

Considering some of the constraints of a transmission system, namely limited amount of power and bandwidth availability, the goal of any error control coding scheme is to reduce the number of errors caused by the channel, while taking bandwidth limitations into consideration. Generally, there are two ways of achieving this goal, either by deploying Forward Error Correction (FEC) codes, or Automatic Repeat Request (ARQ).

An ARQ system detects the errors and asks for retransmission of erroneous packets. This will minimize the amount of wasted effort/bandwidth needed to control errors when both the forward-channel and reverse-channel communications are reliable. For a delay sensitive application, the performance of ARQ, in terms of delay between consecutive packets, deteriorates as the function of the distance and the transmission speed (bandwidth). Consequently, the use of ARQ is very limited in the communication systems deploying broadband real time applications.

On the other hand, FEC adds redundancy to the data stream at the transmitter end, so that the receiver can both detect and correct errors unilaterally. However, this requires expanded bandwidth to transfer the coded sequence including the original information and the error-correcting redundancy. Therefore, one of the engineering goals in this domain is to find a powerful forward error correcting coding scheme that balances the transmit power, bandwidth and data reliability.

While channel coding provides protection to the transmitted information, it is also required to reduce the transmit power which is normally represented in terms of coding gain. The channel coding performance is determined by the Bit Error Rate (BER) and

Signal to Noise Ratio (SNR) of the transmitted signal. The coded system requires less SNR than the uncoded system to achieve the same BER. This reduction expressed in decibels (dB), is referred to as the coding gain.

In 1948, Claude Shannon revolutionized the world of modern digital communication, with the publication of "A Mathematical Theory of Communication" [1], where he defined the capacity of the Additive White Gaussian Noise (AWGN) channel as:

$$C = W \log_2\left(1 + \frac{S}{N}\right) \tag{1.1}$$

where, W is the bandwidth channel in Hertz, S is the signal power, and $N = WN_o$ is the variance of the Gaussian Noise. Shannon's two classical theorems of source coding and channel coding have laid the foundation of much research in the field of error correcting codes. The source coding theorem deals with the least number of bits required to represent a given source without any loss. The channel coding theorem deals with the maximum theoretical data rate with a 'good' data code for reliable communication. In other words, if one would use a data rate less then the channel capacity, along with proper encoding/decoding techniques, one can achieve reliable transmission of information. In particular, Shannon's proof was none constructive, which left open the problem of finding specific good codes. Also, he assumed exhaustive maximum-likelihood decoding, whose complexity grows exponentially with the code length. It was clear that long codes would be required to approach capacity and that more practical decoding methods would be needed.

It was well understood by this time that the key obstacle to practically approaching the channel capacity was not the construction of specific good long codes, although the difficulties in finding asymptotically good codes were already apparent (as "expressed by the contemporary folk theorem: All codes are good, except those that we know of"[2]). Rather, it was the problem of decoding complexity. In the mid 1960s, G.D. Forney Jr. took a different approach to the performance vs. complexity problem. He found a class of codes and associated decoders such that the probability of error could be made to decrease exponentially at all rates less than capacity, while the decoding complexity increased only algebraically, so as to achieve an exponential tradeoff of performance vs. complexity, the solution was a multilevel coding structure called concatenated coding [3].

In [4] Gallager's work on low-density parity check codes, considered soft iterative probabilistic decoding of block codes composed of simple parity equations. Other ground breaking papers in the area of soft iterative decoding include those by Battail [5] and Lodge [6,7].

In 1993, a group of French researchers described a coding technology based on the iterative decoding of parallel-concatenated convolutional codes. They called this scheme "Turbo Coding" in analogy to a turbocharged engine. This provided a performance of only a few tenths of dB away from the Shannon limit. Unlike single pass decoding with hard decision output used in other schemes, turbo decoding uses two or more soft-input soft-output (SISO) decoders that retain confidence information, which in turn is feedback into the input of the other decoders. Each pass of data through the decoder improves the quality of error correction. The iterative decoding of turbo codes can be compared to a process used to solve a crossword puzzle. The first pass through a crossword puzzle is

likely to have a few errors. Some words seem to fit, but when the letters intersecting row and columns don't match, one tends to go back and correct the first-pass answers. Similarly, the turbo decoder iteratively decodes until it converges on the best answer. In other words, the decoder circulates estimates of the sent data like a turbo engine circulates air. When the decoder is ready, the estimated information is finally kicked out of the cycle and hard decisions are made in the threshold component. The result is the decoded information sequence. A more in depth view of the basic components of Turbo Codes will be discussed in Chapter 2.

1.3 Description of the Problem

The superior performance of turbo codes relies on the concepts such as parallel concatenated coding, recursive encoding, pseudo random interleaving, and iterative decoding. However, with such a superior performance comes the price of increased complexity. With this in mind, we embark on the quest to find a reduced complexity decoding algorithm for concatenated linear block codes that allow a tradeoff between complexity and performance. Our main objective is to develop strategies for complexity reduction in iterative decoding and to come-up with a reduced complexity algorithm, while not sacrificing the performance. At this point we expect the reader to be patient, until these issues are discussed in greater detail in the subsequent chapters.

1.4 Thesis Contribution

In this thesis, an in depth analysis of the complexity of an iterative SISO decoder for block turbo codes is presented. Moreover, we study the effects of combining the trellis-based Max-Log MAP decoding algorithm with the augmented list-decoding algorithm, namely the Chase type-II algorithm. The evaluation of this hybrid scheme, results in our proposed algorithm, where we aim at reducing the complexity of a SISO trellis-based iterative decoder by means of reducing the complexity of the trellis.

1.5 Overview

Having defined our objective as contributing towards complexity reduction in trellis-based iterative decoding, we will now go over the outline and structure of the concepts presented in this thesis. Since we are dealing with iterative detection and the concepts involved in encoding/decoding techniques, Chapter 2, consists of a formal description of linear block codes, followed by convolutional codes and concatenated coding schemes, which are the building blocks of turbo encoders. In addition, we define the concept of turbo product code by means of an example, and discuss some of the applications of these codes. Furthermore, we introduce and analyze the algorithms used in turbo iterative decoding. Mainly, we discuss trellis-based decoding algorithms such as the BCJR algorithm and its variants along with the Viterbi algorithm. Finally we end the chapter with a brief overview of the augmented list decoding for block turbo codes.

In Chapter 3, we present the complexity considerations involved in iterative decoding. We divide our analysis into two parts, namely the idea behind computational

complexity reduction, and the concepts involved in the structural simplifications of the iterative trellis-based decoder. We also discuss the efficiency of stopping criteria in terms of complexity reduction. Next, we present a new scheme that is a hybrid of the Chase type-II and the Max-Log MAP algorithm. This chapter is concluded with the analysis of results and the discussion of the problems associated with the presented hybrid scheme, which further deepens our understanding of complexity reduction and results in the algorithm proposed in the following chapter.

In Chapter 4, we present a new iterative decoding structural complexity reduction technique, which involves simplifying the trellis by means of branch pruning. We explain the details involved in pruning certain "reliable" trellis segments by comparing their received channel value along with their extrinsic information to a predefined threshold. Moreover, we also discuss how we compensate for BER degradation by using a correction factor. Lastly we present the simulation results and discuss the gain in complexity reduction.

Finally, in Chapter 5, we make a general conclusion of the main ideas involved in this thesis. We also discuss the potential extensions of the mentioned topics as suggestions for future work.

Chapter 2

PRINCIPLES OF TURBO CODES

Channel coding is a method of adding redundancy to information, such that once transmitted over a noisy channel, the receiver can check and correct errors that might have occurred as a result of noise. For the purpose of this thesis, we will consider a communication medium that can be modeled as an AWGN channel, which will use the FEC as an error control strategy. In this chapter, we will review the basic concepts of turbo codes. We shall begin our discussion with the introduction of linear block codes, convolutional and concatenated codes, which are all the building blocks of turbo codes and proceed with turbo decoding. Various trellis based decoding algorithms will be reviewed in depth with a brief overview of the augmented list decoding.

2.1 Linear Block Codes

A linear block code of length N over a field F, is a linear subspace of F^N. Let us define our source alphabet as the binary set $\chi = \{0,1\}$, with the modulo-2 addition and

multiplication, thus forming a finite field of order 2 called a Galois Field denoted as GF(2). Therefore, an (N, K) binary block code can be defined as a mapping of χ^K into χ^N where $N > K$. The amount of redundancy is determined by the rate that is defined by the ratio K/N.

Given a K-bit source sequence of message $\mathbf{m} = (m_0, m_1, \ldots, m_{k-1})$, the binary block code maps each K-bit into an N-tuple codeword $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$. In a binary linear code, the sum of any two codeword is another codeword.

A generator matrix for linear block code C of length N and dimension K is any $K \times N$ matrix G whose rows form a basis for C.

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ . \\ . \\ \mathbf{g}_{k-1} \end{bmatrix} = \begin{bmatrix} g_{00} & . & . & . & g_{0,n-1} \\ g_{10} & . & & & g_{1,n-1} \\ . & & . & & . \\ . & & & . & . \\ g_{k-1,0} & . & . & . & g_{k-1,n-1} \end{bmatrix} \tag{2.1}$$

where $\mathbf{g}_i = (g_{i0}, g_{i1}, \ldots, g_{i,n-1})$ for $0 \le i \le k$. If $\mathbf{m} = (m_0, m_1, \ldots, m_{k-1})$ is the message to be encoded, the corresponding codeword is given as follows:

$$\mathbf{c} = \mathbf{m}G = [m_0, m_1, \ldots, m_{k-1}] \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ . \\ . \\ \mathbf{g}_{k-1} \end{bmatrix} = m_0\mathbf{g}_0 + m_1\mathbf{g}_1 + \ldots + m_{k-1}\mathbf{g}_{k-1} \tag{2.2}$$

In a systematic code the codeword consists of K original information symbols, or the actual message, and $N - K$ parity symbols. The generator matrix of a systematic code

can be expressed as $G = [I_K|P]$, where I_K is a $K \times K$ identity matrix and P is a $K \times (N\text{-}K)$ parity check matrix. A codeword can also be defined using the parity-check matrix H. If C is an (N, K) linear block code with parity-check matrix H, then an N-tuple $\mathbf{c}$ is a codeword if and only if $\mathbf{c}H^{\mathrm{T}} = 0$, where H^{T} denotes the transpose of H which is an $(N\text{-}K) \times N$ matrix. For a systematic linear block code with generator matrix $G = [I_K|P]$ the corresponding parity check matrix will be $H = [P^T|I_{N\text{-}K}]$. This is further illustrated in the following example of a generator and a parity check matrix of a (7, 4) Hamming code [8]:

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \qquad H = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

As the name suggests, the parity-check matrix is used in error detection to test the validity of a codeword, an extensive proof can be found in [9]. At this point, it is worth mentioning that the minimum Hamming distance of a code, d_{min}, is defined as the smallest Hamming distance between two codewords in a code. As stated previously the linearity property of block codes requires that the modulo-2-sum of two codewords, result in another codeword. Consequently, the minimum distance of a linear block code is the smallest weight of the nonzero codeword in the code. The importance of the minimum distance parameter is due to the fact that it determines the error correcting and detecting capability of a code. For example, a binary block code with minimum distance

d_{min} can detect an erroneous N-bit vector with up to $d_{min} - 1$ erroneous bits, and can correct error patterns with up to $t = \left\lfloor \frac{d_{min} - 1}{2} \right\rfloor$ errors.

It is worth mentioning that one of the important features of the block codes is that a codeword depends only on the current input message and not on the past messages. Therefore, the encoder is a memoryless device. The reader can find a more detailed list of some common linear block codes in [8, 9, 10].

2.2 Convolutional Codes

Unlike block codes, convolutional codes provide a means for encoding stream of data of an arbitrary length, thereby eliminating the need for decoding with fixed data blocks. A convolutional encoder can be implemented as a multi-input/multi-output system, with K bit stream as the input and N bit stream as the output. Encoding begins with the division of the message bit into K input streams (frequently K =1), which are passed through an encoder to provide N output streams. These output streams are then multiplexed together to form the final codeword **c**. As in the case of block codes, the rate of convolutional codes is also $r = K/N$. In the convolutional encoder, one shift register is required for each input stream. The total memory required, M_c, is the sum of the lengths of all shift registers associated with each input. The constraint length, K_c, of an encoder is the maximum number of bits in an output stream that can be affected by an input bit [11]. While it is possible to change the code rate by varying K, *puncturing* is the common practice used to change the code rate. Puncturing consists of deleting some of the output bits of the encoder [9]. For example, a rate 1/2 code can be changed into rate 2/3 by

deleting every 4th output bit. Although this is a suboptimal design, punctured codes are very popular due to the fact that a single encoder/decoder can be used for implementing different code rates. Figure 2.1 illustrates a rate 1/2 convolutional encoder with a generator matrix:

$$g^{(1)} = 1 + D$$
$$g^{(2)} = 1 + D + D^3 \tag{2.5}$$

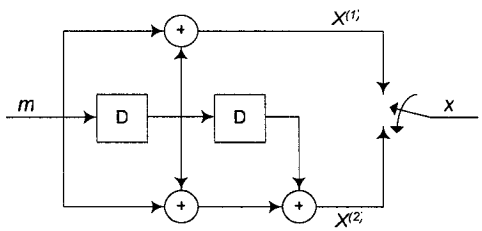

Fig. 2.1: Convolutional Encoder

A convolutional encoder with memory M_c has 2^{Mc} possible states, which are all determined by the contents of the shift register(s). The reception of each input bit results in transition from the present state to one of two possible next states, depending on whether a 0 or a 1 is received. Each state transition results in an N-bit output. This encoding operation is clearly laid out in the *state diagram* of a convolutional encoder. A state diagram consists of a set of nodes (representing the possible states of the encoder) connected by links labeled by m/x, where m represents the input bits and x the output bits corresponding to each state transition. Figure 2.2 illustrates the state diagram of the encoder in Figure 2.1.

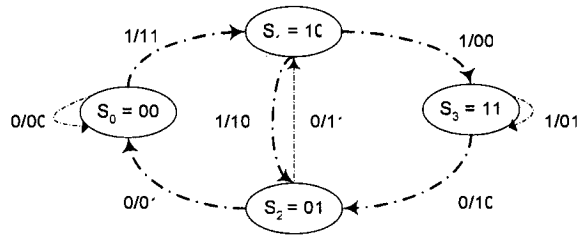

Fig. 2.2: State Diagram of a Convolutional Encoder

The *trellis* diagram is a more elaborate visual tool that incorporates the passage of time to the information already provided by the state diagram. Each node in a trellis diagram is labeled $S_{i,j}$, where i, represents a specific encoder state, and j represents a particular instance in time. Figure 2.3 illustrates a sample state diagram for the encoder given in Figure 2.1.

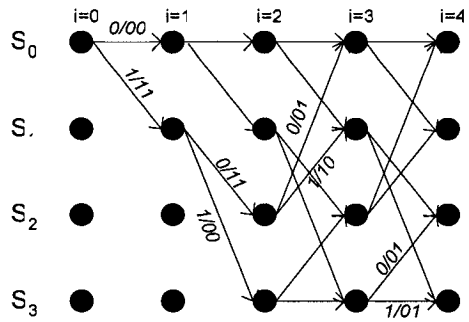

Fig. 2.3: Trellis Diagram of a Convolutional Encoder

For convolutional codes, a unique path through the trellis is associated with every codeword. This path is known as the *state sequence*. Although convolutional codes are characterized by having variable lengths, for certain applications, it is necessary for the input frames to have a concise length. In the literature this is referred to as *trellis*

termination, where the last M_c input bits are set to zero in order to force the trellis to terminate at the all zero state.

2.3 Concatenated Coding Scheme

A concatenated block code consists of two separate codes which are combined to form a larger code, for the purpose of increasing the coding gain, while maintaining a low decoding complexity. This can be achieved by different schemes, namely, serial, parallel and hybrid concatenated block coding. However, before further investigation into these schemes let us introduce the concept of interleaving of coded data for channels with burst errors. In almost all concatenated coding schemes an for the purpose of constructing a code with exceptionally long code words.

2.3.1 Interleaving and Burst Errors

By its very nature, an interleaver or a permutator receives an N-bit input sequence and rearranges it to construct a new N-bit output sequence. As was previously stated, we have considered an AWGN channel for our discussions, where the errors occurring during the transmission are statistically independent. However, some channels demonstrate bursty error characteristics. In essence, a burst of errors of length l is defined as an l-bit sequence beginning and ending with 1. As an example, let us consider a multi-path fading channel, where due to time variant multi-path propagation, the signal falls below the noise level and results in a large number of errors. Much work has been done on burst error correcting codes as stated in [5]. However, an effective approach for dealing with burst error channel is to interleave the coded data such that the bursty

channel is transformed into a channel with independent errors. As a result of the interleaving/de-interleaving, errors within a codeword appear to be independent since error bursts are spread out in time. For that reason, error codes designed for a channel with statistically independent errors can be used for bursty error channels with the proper interleaving techniques.

2.3.2 Parallel Concatenated

A parallel-concatenated block coding scheme consists of two encoders and an interleaver, where the k information bits are encoded twice by using the original information and its interleaved version. This concept is further illustrated in Figure 2.4, where the two systematic binary linear encoders are used to encode the information and the permuted information. Consequently, the original information bit is transmitted only once with 2 types of parity check bits.

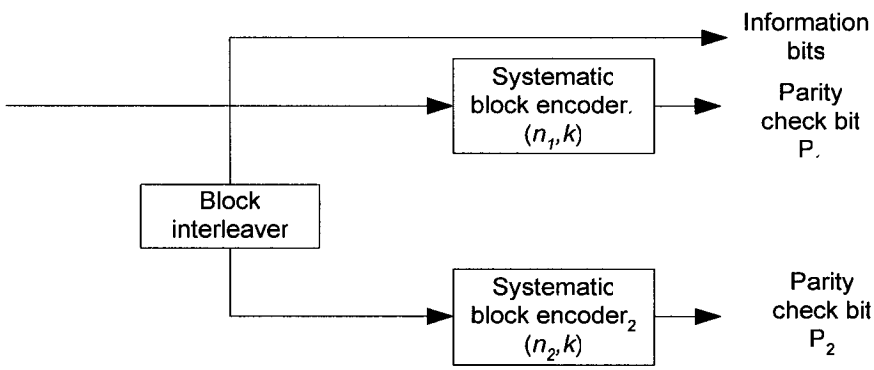

Fig. 2.4: Parallel Concatenated Block Code

A parallel convolutional encoder works in the same way, where there are two, or more convolutional encoders. The first encoder encodes the original information bits. Next, the original information bits are interleaved and used as the input of the second

convolutional encoder. Finally, the output of the encoders is in some cases punctured, and sent through the channel along with the original information bits.

2.3.3 Serial Concatenated Block Codes

In a serially concatenated block coding scheme, a K-bit block of information is initially fed into the first encoder, where the output is fed in turn to an interleaver, and the output of the interleaver is again coded by a second encoder. This concept is further clarified in Figure 2.5, where S corresponds to the number of outer codewords.

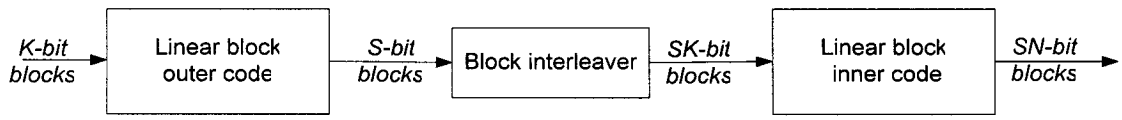

Fig. 2.5: Serial Concatenated Block Codes

2.3.3.1 Product Codes: An Example of a Serial Concatenated Block Code

A product code is a multidimensional block code, which combines short block codes in order to form a longer block code with moderate decoding complexity [9].

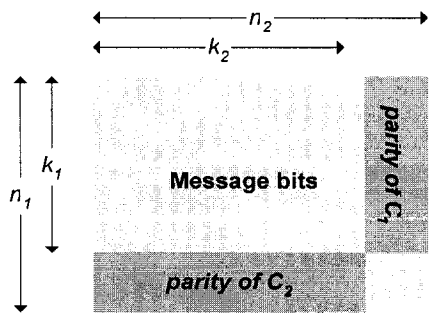

Fig. 2.6: Product Code

Let us consider the case of two codes C_1 (n_1, k_1, d_1) and C_2 (n_2, k_2, d_2), where n_i, k_i, d_i, are the length of coded data, the length of the original information data, and the minimum hamming distance of code Ci, i = 1, 2. As demonstrated in Figure 2.6, the product code can be viewed as a matrix PC, with n_1 columns, and n_2 rows. First, k_2 data blocks are encoded by code C_1, then the n_1 columns are encoded by code C_2. Since all the columns of the matrix PC are codewords of C_2, and all the rows are codewords of C_1; the matrix PC is decoded by alternating between decoding first the rows, and then the columns. This iterative decoding method will be described in more detail in the subsequent sections of this chapter.

In terms of the applications of turbo product codes, [9] presents a comprehensive study of various organizations and companies which use these applications in satellite communications, wireless Local Area Network (LAN), wireless internet access and mobile communications.

2.4 Turbo Codes

In 1993, Berrou *et al.* [12] introduced a coding scheme consisting of two parallel recursive systematic convolutional encoders, separated by an interleaver, and using an iterative *A Posteriori Probability* (APP) decoder. This scheme, achieved an exceptionally low BER at an SNR close to Shannon's theoretical limit. Since their introduction, turbo codes have been the focal point of much research in the field of error control coding. The two fundamental elements of turbo codes are parallel concatenated encoders and a decoding procedure with an iterative nature. Since we have already familiarized the

reader with the concepts behind parallel concatenated encoders in Section 2.3.2, henceforth we will focus on the turbo decoder.

2.4.1 Turbo Decoding

Let us start by describing the standard Soft-Input Soft-Output (SISO) iterative decoding used in turbo codes. Once the symbols are encoded in either parallel or a serial concatenated fashion as described in the previous section, the symbols are then fed into a modulator that sends out a stream of waveforms over the channel. For the purpose of our research we consider Binary Phase Shift Keying (BPSK) modulation. Once the symbols are detected in the receiver, sets of sufficient statistics are obtained by first matched filtering then sampling the received waveforms [10]. These sets of observables are the input to the iterative decoder. Since the outputs of each constituent encoder of a turbo code depend only on the present state and present input bit, the encoding process of a turbo code can be considered as two joint Markov processes. Given the fact that the two Markov processes run on the same input data; turbo decoding proceeds by first independently estimating each process, then refining the estimates by iteratively sharing the information between the two decoders. This signifies that the output of one decoder can be used as *a priori* information by the other decoder. In order to take advantage of this iterative decoding scheme, it is necessary for each decoder to produce soft-bit decisions in the form of Log Likelihood Ratios (LLRs). This serves to indicate that a transmitted bit x_i is either a 1 or a 0 given the received information y:

$$L(x) = \ln \frac{P[x_i = 1 \mid y]}{P[x_i = 0 \mid y]} \tag{2.5}$$

A decoder that accepts input in the form of *a priori* information, and produces output in the form of *a posteriori* information is known as a SISO decoder. The important information elements of the turbo decoding algorithm are the LLR of *a posteriori* probabilities, the LLR of *a priori* probabilities and the exchange of extrinsic values in each iteration. A turbo decoder is composed of two constituent SISO decoders that are serially connected through an interleaver, this is the same interleaver used in the encoding process. The above-mentioned LLRs and the extrinsic information are exchanged between each decoder and are updated iteratively.

Fig. 2.7: SISO Decoder

As expected the channel output is the soft input of the SISO decoder. Let us assume that a binary random variable x_k is conditioned on the vector y_k, then the conditional LLR, using Baye's theorem, will be:

$$L(x_k \mid y_k) = \ln \frac{P(x_k = +1 \mid y_k)}{P(x_k = -1 \mid y_k)} \tag{2.6}$$

$$= \ln \frac{P(y_k \mid x_k = +1) \cdot P(x_k = +1)}{P(y_k \mid x_k = -1) \cdot P(x_k = -1)} \tag{2.7}$$

$$= \ln \frac{P(y_k \mid x_k = +1)}{P(y_k \mid x_k = -1)} + \ln \frac{P(x_k = +1)}{P(x_k = -1)} \tag{2.8}$$

$$= L(y_k \mid x_k) + L(x_k) \tag{2.9}$$

However, after transmission over a channel with a fading factor a and additive Gaussian noise,

$$L(x_k \mid y_k) = \ln \frac{P(y_k \mid x_k = +1) \cdot P(x_k = +1)}{P(y_k \mid x_k = -1) \cdot P(x_k = -1)} \tag{2.10}$$

$$= \ln \frac{\exp(-\frac{E_s}{N_o}(y_k - a)^2)}{\exp(-\frac{E_s}{N_o}(y_k + a)^2)} + \ln \frac{P(x_k = +1)}{P(x_k = -1)} \tag{2.11}$$

$$= \ln\{\exp(-\frac{E_s}{N_o}(y_k{}^2 - 2 \cdot a \cdot y_k + a^2 - y_k^2 - 2 \cdot a \cdot y^k - a^2)]\} + L(x_k) \tag{2.12}$$

$$= \ln\{\exp[-\frac{E_s}{N_o}(-4 \cdot a \cdot y_k)]\} + L(x_k) \tag{2.13}$$

$$= \underbrace{\frac{E_s}{N_o}}_{L_c} 4 \cdot a \cdot y_k + L(x_k) \tag{2.14}$$

$$= L_c \cdot y_k + L(x_k) \tag{2.15}$$

where a =1, for a Gaussian channel.

As can be seen from Figure 2.7, the symbol-by-symbol SISO decoder has two outputs, namely the extrinsic, and the *a posteriori* LLR values, $L_e(\hat{x})$ and $L(\hat{x})$ respectively, where:

$$L(\hat{x}) = L(x \mid y) = \ln \frac{P(x = +1 \mid y)}{P(x = -1 \mid y)} \tag{2.16}$$

In other words, the *a posteriori* LLR is the logarithm of the ratio of the probability of a given bit having the value +1/-1, given observation y. The extrinsic information

$L_e(\hat{x})$ contains the soft output from all "other" coded bits in the code sequence and is not influenced by $L(x)$, and the $L_c \cdot y$ values of the "current" bit. For the systematic codes, the soft output of the transmitted bit is represented by:

$$L(\hat{x}) = L_c \cdot y + L(x) + L_e(\hat{x}) \tag{2.17}$$

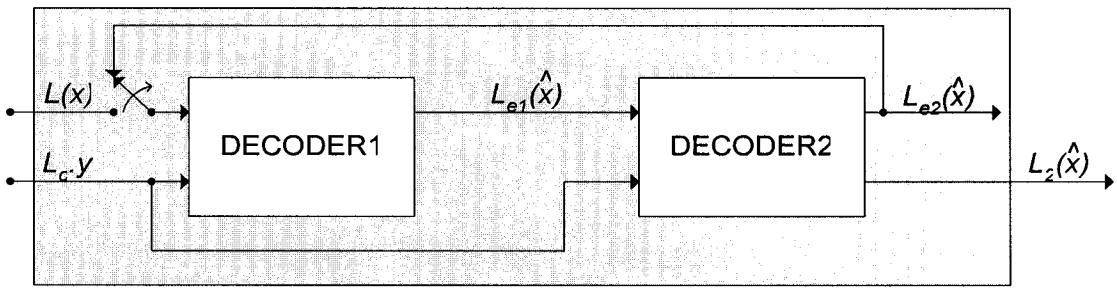

Fig. 2.8: Iterative Decoding Procedure

As indicated in the above figure, the extrinsic value for the first iteration is:

$$L_{e1}(\hat{x}) = L_1(\hat{x}) - [L_c \cdot y + L(x)] \tag{2.18}$$

Assuming equally likely information bits, we initialize $L(x) = 0$ for the first iteration. *Decoder2*, uses the extrinsic information of *Decoder1* as *a priori* information, and produces its own extrinsic information:

$$L_{e2}(\hat{x}) = L_2(\hat{x}) - [L_c \cdot y + L_{e1}(\hat{x})] \tag{2.19}$$

This is in turn considered as the *a priori* information of *Decoder1* in the second iteration and so on. This process is terminated after a fixed number of iterations; usually

when the soft output of *Decoder2* stabilizes and does not vary between iterations. Finally, *Decoder2* combines both extrinsic values such that:

$$L_2(\hat{x}) = L_c \cdot y + L_{e1}(\hat{x}) + L_{e2}(\hat{x}) \tag{2.20}$$

and a hard decision is made on bit $\hat{x}$ [9].

2.4.1.1 Trellis based Decoding Algorithms

Trellis based decoding algorithms are often but not always (as is the case in [13]) used for turbo decoding. This is mainly due to the fact that trellis based decoding methods are recursive, hence suitable for the estimation of the state sequence of a discrete-time-finite-state Markov process observed in memoryless noise [9]. Two well developed and widely used algorithms for determining the state sequence of a trellis encoder are the Viterbi algorithm (VA), and the Maximum *A Posteriori* (MAP) algorithm [14]. In 1967, Andrew J. Viterbi published a powerful and practical algorithm for the decoding of binary convolutional codes [15]. It was further recognized that the VA could be applied to a variety of applications such as channels with memory [16], and trellis coded modulation [17]. Although the BCJR algorithm (also referred to as MAP) predated the VA, and was explicitly applied to convolutional codes only a few years after the publication of the Viterbi algorithm, it was not until the discovery of turbo codes that the BCJR algorithm became popular.

Given the received observation sequence y, the VA determines the most probable state sequence:

$$\hat{s} = \arg\{\max_s P[s \mid y]\} \tag{2.21}$$

Thus, the states estimated by VA will always form a connected path through the trellis. Another variant of the VA is the Soft Output Viterbi Algorithm (SOVA), which also accepts and delivers soft sample values. SOVA gives not only the most likely path sequence on a finite-state Markov chain, but it also gives either the *a posteriori* probability for each bit, or a reliability value [18]. On the other hand, the MAP algorithm attempts to resolve each state transition without regard to the overall sequence of the trellis:

$$\hat{s}_i = \arg\{\max P[s_i \mid y]\} \tag{2.22}$$

Consequently, the resulting trellis is not a connected path. Performance wise, literature indicates that the VA results in lowering the FER (Frame error Rate), and MAP minimizes BER [9]. Although both of these algorithms can be used for SISO [14], for the purpose of our research we have mainly focused on variants of the MAP algorithm.

In terms of the soft symbol detection, we find it pertinent here to remind the reader that in its most general form, the communication model depicted in our research resembles a hidden Markov Model. This model is a process that is governed by three probabilistic phenomena described by three distributions:

1. An initial state distribution that describes the initial state for the underlying Markov chain.
2. A state transition distribution that describes the probability of the chain transitioning from any given state to another.

3. An output distribution that describes the observed random variable y, in terms of the transition of the "hidden" Markov chain[1].

As stated in [14], the problem encountered with this depicted model is that of estimating the above-mentioned distribution given the observation y. In the early 1960s, L. Baum and L. Welch [19] proposed an iterative solution to this problem. It was in 1974 however, that Bahl, Cocke, Jelinek and Raviv (BCJR) [20] modified the Baum-Welch algorithm by assuming a known output distribution and focusing on recursive means in the Baum-Welch algorithm for estimating the likelihood of the state transition. The BCJR algorithm is also known as the forward-backward or the *a posteriori* probability algorithm, or maximum *a posteriori* (MAP) algorithm. Ultimately this algorithm results in the sequence of APP, $\{P(x_k = i \mid observation), i = 0,1\}$, where $P(x_k = i \mid observation)$ is the APP of the data bit x_k given all the received sequence [9]. From an implementation point of view, excessive amounts of calculations, non-linear mathematical functions and the numerical representation of probabilities, are somewhat tedious and time consuming in terms of using both hardware and software resources. As an outcome, variants of this algorithm such as the Log-MAP and the Max-Log MAP algorithms are used in practice. Although both of these schemes are sub-optimal, their performance is close enough such that they can be substituted for the MAP algorithm.

2.4.1.1.1 Maximum A Posteriori Probability (MAP) Algorithm

For the purpose of this thesis, the emphasis of our research has been on turbo codes that are composed of linear binary block codes. Hence we will describe the MAP

[1] For the interested reader the Discrete Hidden Markov Source is described in detail in [14].

algorithm and its variants restricted to the case of binary block turbo codes. Let us consider the example described in Section 2.3.3.1, at this point we will remind the reader that the *PC* matrix depicted in Section 2.3.3.1 is decoded by alternatively decoding the rows, then the columns of the matrix *PC*. Iterative decoding is only efficient if a SISO algorithm is used. From the soft decision, an extrinsic information term is extracted and then used at the input of the next decoder. The following mathematical derivations are described both in [9][19]. To further describe this algorithm, it is helpful to consider the description of the binary trellis of a systematic block code.

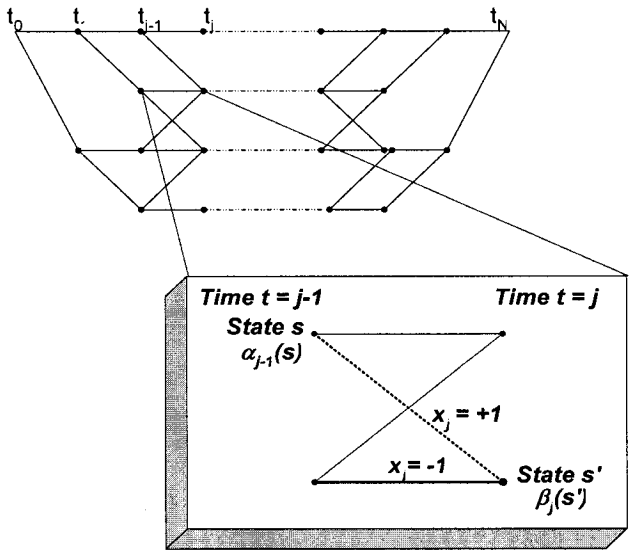

Fig. 2.9: A Sectional View of the Trellis of a Systematic Block Code, Representing a Transition From State s to s'.

We want to compute a soft decision measure for the j^{th} term of the information sequence, given the sequence $\vec{y} = \{y_1,...,y_j,...,y_N\}$, which is the output of the AWGN channel. From Figure 2.9 this LLR becomes:

$$\hat{L(x_j)} = L(x_j \mid \vec{y}) = \ln \frac{P(x_j = +1 \mid \vec{y})}{P(x_j = -1 \mid \vec{y})} = \ln \frac{\sum_{(s,s'),x_j=+1} P(s,s',\vec{y})}{\sum_{(s,s'),x_j=-1} P(s,s',\vec{y})} \tag{2.23}$$

The crux of the problem is the determination of the joint probability distribution $P(s,s',\vec{y})$. Given the fact that we are dealing with a Discrete Memoryless Channel (DMC) we can write $P(s,s',\vec{y})$ as:

$$P(s,s',\vec{y}) = P(s',\vec{y}_{t<j}) \cdot P(s',\vec{y}_t \mid s) \cdot P(\vec{y}_{t>j} \mid s') \tag{2.24}$$

$$= \underbrace{P(s',\vec{y}_{t<j})}_{\substack{\alpha_{j-1}(s) \\ past}} \cdot \underbrace{P(s \mid s') \cdot P(y_t \mid s,s')}_{\substack{\gamma_j(s,s') \\ present}} \cdot \underbrace{P(\vec{y}_{t>j} \mid s')}_{\substack{\beta_j(s') \\ futur}} \tag{2.25}$$

- $\alpha_{j-1}(s)$: This term is the MAP measure on n_j, based on the first *j-1* terms of $\vec{y}$. This is denoted by the $\vec{y}_{t<j}$ signifying the sequence of received symbols from the beginning of the trellis up to time *j-1*. The Greek letter α denotes this measure.
- $\gamma_j(s,s')$: This term is the measure on x_j based solely on y_j. In other words this is the branch transition probability, and is denoted by the Greek letter γ.
- $\beta_j(s')$: This last term is a measure on x_j, based on the last bits of $\vec{y}$ from y_{j+1} to the end of the trellis. Hence this is a measure based on the future and is denoted by the Greek letter β.

The above three definitions are the major key players in the MAP algorithm. $\alpha_j(s')$ and $\beta_j(s)$ are also referred to as the forward and backward recursion of the MAP algorithm where,

$$\alpha_j(s') = \sum_{s,s'} \gamma_j(s,s') \cdot \alpha_{j-1}(s) \tag{2.26}$$

$$\alpha_0(0) \equiv 1$$

$$\beta_{j-1}(s) = \sum_{(s,s')} \gamma_j(s,s') \cdot \beta_j(s') \tag{2.27}$$

$$\beta_N(0) \equiv 1$$

$$\gamma_j(s,s') = P(s'|s) \cdot P(y_j \mid s,s') \tag{2.28}$$

Based on the assumption that the information bits are statistically independent; given an *(N, K)* systematic block code, the branch probability is defined as:

$$\gamma_j(s,s') = \begin{cases} P(y_j \mid x_j) \cdot P(x_j) & 1 \le j \le K \\ P(y_j \mid x_j) & K+1 \le j \le N \end{cases} \tag{2.29}$$

where from [9, 19] we have:

$$P(y_j \mid x_j = \pm 1) = B_j \cdot e^{L_c y_j \cdot x_j / 2} \tag{2.30}$$

$$P(x_j = \pm 1) = A_j \cdot e^{L(x_j) \cdot x_j / 2} \tag{2.31}$$

Where both A_j, and B_j are equal for all transitions from time *j-1* to *j* and can be omitted by the ratio in Equation 2.23. Consequently the simplified branch transition equation can be rewritten as:

$$\gamma_j(s,s') = e^{(\frac{1}{2}x_j(L_c y_j + L(x_j)))} = e^{(L(x_j;y_j)\cdot x_j/2)} \tag{2.32}$$

$$\gamma_j(s,s') = \begin{cases} e^{((L_c\cdot y_j + L(x_j))\cdot x_j/2)} & 1 \le j \le K \\ e^{((L_c\cdot y_j)x_j/2)} & K+1 \le j \le N \end{cases} \tag{2.33}$$

The LLR associated with Equation 2.29, can be expressed by:

$$L(x_j;y_j) = \begin{cases} L_c\cdot y_j + L(x_j) & 1 \le j \le K \\ L_c\cdot y_j & K+1 \le j \le N \end{cases} \tag{2.34}$$

As indicated by the above equation in a systematic block code *a priori* probability $L(x_j)$ is equal to zero for all parity bits.

Finally, the soft output of the trellis-based MAP algorithm for systematic block turbo codes can be written as:

$$L(\hat{x}_j) = L_c\cdot y_j + L(x_j) + \ln\frac{\sum_{(s,s'),x_j=+1}\alpha_{j-1}(s)\beta_j(s')}{\sum_{(s,s'),x_j=-1}\alpha_{j-1}(s)\beta_j(s')} \tag{2.35}$$

2.4.1.1.1.1 MAX-LOG MAP ALGORITHM

Based on the fact that the MAP decoding algorithm involves non-linear functions and requires a significant amount of memory for exponential and multiplication operations, it is considered too complex for implementation. In practice, the Log-MAP algorithm is used in order to avoid exponential functions and to replace multiplication operations by additions. Furthermore, Max-Log MAP is a suboptimal algorithm, which

simplifies the representation of probabilities, and reduces the computational complexity with a negligible degradation in performance [21].

As the name suggests, Max-Log MAP uses logarithmic functions for the forward and the backward recursions and the branch probability in the MAP algorithm. Before proceeding to the details of this sub-optimal algorithm, we will remind the reader of the notations used:

- *j*: refers to the time instant *j*.
- *s,s'*: refer to the trellis state *s* at time *j-1* and *s'* at time *j*.
- *K,N*: refer to the length of the information sequence and the coded version, respectively.

Considering the logarithm of the transitional probability Equation 2.33 becomes:

$$\log\gamma_j(s,s') = \frac{L(x_j,y_j)\cdot x_j}{2} = \begin{cases} \dfrac{[L_c\cdot y_j + L(x_j)]\cdot x_j}{2} & 1\le j\le K \\ \dfrac{L_c\cdot y_j\cdot x_j}{2} & K+1\le j\le N \end{cases} \tag{2.36}$$

For simplifying the computation of $\alpha_j(s)$ and $\beta_j(s')$, the following approximation is used:

$$\log(e^{\delta_1}+e^{\delta_2}+\ldots+e^{\delta_n}) \approx Max_{i\in\{1,2,\ldots,n\}}\delta_i \tag{2.37}$$

where $Max_{i\in\{1,2,\ldots,n\}}\delta_i$ is obtained by successive computation of *(n-1)* maximum functions over 2 values. Hence the forward recursion becomes:

$$\log \alpha_j(s') = \log \sum_s \gamma_j(s,s') \alpha_{j-1}(s) \tag{2.38}$$

$$= Max_s[\log \gamma_j(s,s') + \log \alpha_{j-1}(s)] \tag{2.39}$$

$$= Max_s[\frac{L(x_j, y_j) \cdot x_j}{2} + \log \alpha_{j-1}(s)] \tag{2.40}$$

where at time $j=0$, $\log \alpha_0(0) \equiv 0$ and $\log \alpha_0(\underset{s \neq 0}{s}) \equiv -\infty$.

By the same argument, the backward recursion probability becomes:

$$\log \beta_{j-1}(s) = \log \sum_{s'} \gamma_j(s,s') \cdot \beta_j(s') \tag{2.41}$$

$$= Max_{s'}[\log \gamma_j(s,s') + \log \beta_j(s')] \tag{2.42}$$

$$= Max_{s'}[\frac{L(x_j, y_j) \cdot x_j}{2} + \log \beta_j(s')] \tag{2.43}$$

where at time $j=N$, $\beta_N(0) \equiv 0$.

Thus the estimated information is approximated by:

$$L_e(x_j) = \log \frac{\sum_{(s,s'),x_j=+1} \alpha_{j-1}(s) \cdot \beta_j(s')}{\sum_{(s,s'),x_j=-1} \alpha_{j-1}(s) \cdot \beta_j(s')} \tag{2.44}$$

$$= Max_{(s,s'),x_j=+1}[\log \alpha_{j-1}(s) + \log \beta_j(s')] - Max_{(s,s'),x_j=-1}[\log \alpha_{j-1}(s) + \log \beta_j(s')] \tag{2.45}$$

Finally the soft output of the Max-Log-MAP algorithm can be written as:

$$L(\hat{x}_j) = L_c \cdot y_j + L(x_j) + L_e(\hat{x}_j) \tag{2.46}$$

$$= L_c \cdot y_j + L(x_j) + \log \frac{\sum_{(s,s'),x_j=+1} \alpha_{j-1}(s)\beta_j(s')}{\sum_{(s,s'),x_j=-1} \alpha_{j-1}(s)\beta_j(s')} \tag{2.47}$$

$$= L_c \cdot y_j + L(x_j) + \log \sum_{(s,s'),x_j=+1} \alpha_{j-1}(s)\beta_j(s') - \log \sum_{(s,s'),x_j=-1} \alpha_{j-1}(s)\beta_j(s') \tag{2.48}$$

$$= L_c \cdot y_j + L(x_j) + Max_{(s,s'),x_j=+1}[\log \alpha_{j-1}(s) + \log \beta_j(s')] - Max_{(s,s'),x_j=-1}[\log \alpha_{j-1}(s) + \log \beta_j(s')] \tag{2.49}$$

Therefore, by changing the multiplication and exponential operations into comparison and addition operations, the Max-Log-MAP is an attractive algorithm in terms of feasibility of hardware implementation.

2.4.2 Augmented List Decoding

Apart from trellis based decoding, augmented list decoding is another SISO iterative method used for turbo block codes. The concept behind list decoding is to form a list of candidate codewords by considering the channel (soft) information. There exist numerous methods of producing this list of candidate codewords, such as the Chase-type II method [22], the pseudo maximum likelihood algorithm [23], and the Fang-Battail-Buda Algorithm [24]. In [13], Pyndiah *et al.* proposed a SISO decoding algorithm based on the Chase algorithm. The type of product code reviewed is similar to the example depicted in Section 2.3.3.1. The main idea behind this near optimum algorithm is to use the received channel information in order to limit the set of received candidate codewords to a set of highly probable codewords. As a consequence this algorithm is suboptimal since it does not perform a full search over all valid codewords. In order to find this set of

candidate codewords, a set of test patterns T, is generated by using a set of error patterns E and a hard decision vector $\vec{y}$ of the received sequence such that:

$$\vec{t} = \vec{e} + \vec{y} \quad \text{where} \quad \begin{matrix} \vec{t} \in T \\ \vec{e} \in E \end{matrix} \tag{2.50}$$

Note that E is generated based on the reliability of the received sequence. Once the set T is complete, each test pattern is decoded using an algebraic decoder. The decoder output is a set of candidate codewords. Two codewords are selected from this set, one is referred to as the *decision* codeword, and the other will be named the *competing* codeword. Both of these codewords have the highest correlation with the received sequence; however they differ from each other by one bit. Ultimately, given the received vector, the decision codeword and its competing codeword result in the soft output[2].

2.5 Summary

In this chapter, the reader is presented with an overview of the type of encoding/decoding schemes encountered in turbo codes. Linear block codes and convolutional codes were discussed, followed by concatenated codes, which were illustrated by an example of a product code. In terms of decoding schemes, the main focus was on SISO iterative decoding. A brief overview of trellis based decoding algorithms such as the VA followed by an in depth analysis of the MAP algorithm and its variants, namely the Log MAP and the Max-Log MAP were also discussed in detail.

[2] A detailed well defined description and a concise example of this algorithm is given in [9]. We refrain from repeating all that information here; although Pyndiah's work is inspiring, however the core of our work is founded on trellis based decoding.

Finally, we concluded the chapter by familiarizing the reader with the concept of augmented list decoding.

Chapter 3

DECODING COMPLEXITY OF BLOCK TURBO CODES

In its most general term, any system benefiting from an "unlimited" amount of resources can achieve superior performance. Realistically speaking, however, that is not the case. Since, for any type of implementable system, the designer has to consider a set of well defined criteria and limited resources for a given design concept. The general rule of thumb for any designer is to increase the system's performance while decreasing its complexity. Obviously, a better performance is usually observed as the complexity increases, unfortunately, iterative decoding is no exception to this general rule. Hence, before considering any type of reduced complexity techniques let us first understand the complexity considerations in iterative decoding.

As stated in Chapter 2, an iterative decoding system is composed of multiple components, which perform the counter actions of the modules in the encoding system. As expected the overall complexity of an iterative decoding system is thus dependent on

the complexity of its constituents. This brings us to the following question, which has generated much interest ever since the introduction of turbo codes: "*What kind of decoding complexities are we dealing with, and how can we reduce them*?" For the purpose of our research, we shall examine both the computational and structural complexity of iterative decoding of block turbo codes.

3.1 Computational Complexity

As previously mentioned, the additive version of a given SISO algorithm is less complex then the multiplicative version. An example of this complexity reduction is of course the Log-MAP algorithm, where the calculations are carried out in the logarithmic domain as opposed to the exponential domain. However, even in the log domain, we still need to calculate complex terms such as:

$$x = \ln \sum_i e^{x_i} \tag{3.1}$$

By using further approximation techniques, the above summation becomes a simple comparison as in the Max-Log MAP algorithm:

$$\ln \sum_i e^{x_i} \approx Max(x_i) \tag{3.2}$$

Further details regarding the sub-optimality of the Max-Log MAP can be found in [9]. In terms of the LLR computation for each received symbol, the MAP algorithm considers all paths in the trellis, but divides them into two sets, one that has a bit one at time t, and the other having a bit zero, it then proceeds by calculating the LLR for each of

these two sets. The Max-Log MAP however, considers only 2 paths per trellis segments, this being the best path with the bit one and the best path with the bit zero at time t. It then calculates the LLR for each path and returns their difference. Although these paths might change from one trellis segment to another, one will always remain the ML (Most Likely) path. A decoding complexity comparison table, between the MAP, Log-MAP, Max-Log MAP, and SOVA algorithms is presented in [25]. Although each algorithm is represented by the number of computation operations for a (N, K) convolutional code the comparison can also be applied to block codes. In addition, for the interested reader, [26] presents a good comparison of computational complexity for different SISO algorithms, where the authors have considered the delay, computation per symbol, and memory requirements in their analysis.

3.2 Structural Complexity

For trellis based decoding algorithms, structural complexity transcends to the complexity of the trellis diagram. Rewinding back in our statement, it is well known that whenever the encoders are represented by their FSMs (Finite State Machine), the decoding procedure can be carried out by using a trellis diagram [9, 10, 14]. By that account, the complexity of the BCJR algorithm grows exponentially with the number of states of the corresponding trellis [25]. Hence, reducing the number of states, reducing the number of branches, or a combination of both, in other words, *minimizing the size of the trellis*, is the main focus of decreasing the structural complexity of such decoders. The idea of state reduction has been the focal point of much research even before the

introduction of iterative decoding. Two reduced state decoders are proposed in [27,28]. However, both approaches consider transmission channels with ISI (Intersymbol Interference), which results in a lot of states in the decoder's trellis diagram, and it is very complex to decode otherwise. In [29], T. Larsson proposes a state space partitioning approach to trellis decoding in which the states in the trellis are divided into a number of classes; [30] proposes reduced state BCJR type algorithms in which the key idea is based on the construction of a "survivor map" corresponding to a reduced state trellis obtained in one recursion which is used in other recursions. V. Franz and J.B. Anderson in [31], present 2 techniques to reducing both branch and state complexity by removing certain nodes from the full complexity trellis diagram. J. Hagenauer and C. Kuhn propose another reduced complexity algorithm [32] based on the stack algorithm [8]. Also the method of early detection is used in [33], where confidence intervals are used to detect some information symbols, state variables and codeword symbols, in an earlier stage during the decoding procedure.

3.3 Stopping Criteria

A fundamental property of a turbo decoder is its iterative nature. In each iteration, a set of *a posteriori* probabilities that are produced by one decoder are considered as *a priori* input for the second decoder, and this set of soft information is "passed" between the decoders and is more refined in each iteration until a final hard decision is made. However, when the decoder approaches the performance limit of a given system, further iterations provide very little improvement. As expected, there is an associated cost to

each iteration and as the number of iterations will increase, the complexity will also augment. It is therefore of most importance to consider a good stopping criteria in order to reduce decoder complexity. In [34], Hagenauer *et al.* suggest a stopping criterion based on cross entropy between the distributions at the output of the 2 SISO decoders. In other words, the cross entropy is defined as a measure of the difference (closeness) of the two distributions. The suggested stopping criterion is to make a hard decision when the cross entropy falls within a certain range ~ $(10^{-2}, 10^{-4})$. In [35], Shao *et al.* present two simple and computationally effective rules referred to as the hard decision aided (HDA) criterion and the sign change ratio (SCR). Although, both methods are based on the same concept as [34], they require only integer operations and less memory space. For the SCR method, the decoder has to only count the number of sign changes between two consecutive operations. Whereas in HDA, the decoder has to store the hard decisions in one iteration and compare them to the decisions obtained in the next iteration. Several other stopping rules can be found in [36] and [37], most of them being based on the same basic concepts as the ones discussed.

Now that we have shed some light on the various complexity issues regarding block turbo codes, we will proceed by discussing the thought pattern that has led to our proposed algorithm.

3.4 A Hybrid Scheme Using Max-Log MAP and the Chase Algorithm

Inspired by the work done by Pyndiah in [34] and Chase [22], our goal is to generate a set of candidate bits, given a received sequence $\vec{Y}$, which are characterized as

the most reliable bits, therefore a decision will be made on these bits prior to trellis bit decoding of the received sequence. In other words, a hard decision will be made on these bits while Max-Log MAP will decode the rest of the received sequence. Our objective is to reduce the complexity of the trellis by pruning the branches corresponding to the most reliable bits. We will further clarify the above statement by discussing the background of Chase's proposed sub-optimum algorithm for near-ML decoding of linear block codes, as described in [34], and remove all ambiguity by means of an example.

3.4.1 Background of the Hybrid algorithm

Let us consider the transmission of binary symbols {0, 1} which are coded by a linear block code C (as in the example in Section 2.3.3.1), over an AWGN with the following mapping $\{0 \rightarrow -1, \quad 1 \rightarrow +1\}$. The observation $\vec{R} = (r_1,...,r_l,...,r_n)$ is given by:

$$\bar{r} = \bar{x} + \bar{n} \tag{3.3}$$

where $\vec{X} = (x_1,...,x_l,...x_n)$ is the transmitted codeword, and $\vec{N} = (n_1,...n_l,...n_n)$ is a vector consisting of the white Gaussian noise samples with standard deviation σ. According to [22], the optimum decision $D = (d_1,...,d_l,...d_n)$ corresponding to the transmitted codeword $\vec{X}$ is given by

$$D = C^i \quad if \left|R - \mathrm{c}^i\right|^2 \leq \left|R - c^l\right|^2 \quad l \in \{1,...,2^k\} \quad and \quad l \neq i \tag{3.4}$$

where $\mathbf{c}^i = (c_1^i,...c_l^i,...c_n^i)$ is the i*th* codeword of C and $\left|R - c^i\right|^2 = \sum_{j=1}^{n}(r_j - c_j^i)^2$ is the squared Euclidean distance between R and $\mathbf{c}^i$. In 1972, Chase proposed a suboptimum algorithm of low complexity for near-ML decoding of linear block codes. Chase

observed that at high SNR ML codeword D, is located in the sphere of radius $(\delta - 1)$ centered on $\mathbf{y} = (y_1,...,y_l,...,y_n)$ where $y_l = 0.5(1 + \text{sgn}(r_l))$ and $y_l \in \{0,1\}$. In order to reduce the set of reviewed codewords, only the set of the most probable codewords residing within the sphere are selected by using channel information R; this is done by the following procedure:

1. Determine the position of the $p = \lfloor \delta/2 \rfloor$ least reliable binary elements in $\mathbf{y}$ using $\vec{R}$, where δ is the minimum distance of the code.
2. Form a set of error patterns $\vec{E}$, defined as the combination of all n-bit binary sequences for the positions found in step 1. Therefore there will be 2^p possible error patterns including the all zero pattern.
3. Form the set of test patterns $\vec{T} = \vec{Y} \oplus \vec{E}$, where $\oplus$ denotes the modulo 2 addition operation.
4. Decode all test patterns using an algebraic decoder; with the valid codewords form a subset of restricted reviewed codewords.

We decided to take advantage of a similar concept in our hybrid scheme. We shall explain the details of this concept with the aid of an example.

3.4.3 Trellis Construction of a Linear BCH Code

Let us consider a product code based on linear block codes, more specifically, we have considered the BCH $(7,4,1)^2$ built as outlined in Section 2.3.3.1. Transmission is simulated over a Gaussian channel; using BPSK signaling, with white Gaussian noise, having a mean of zero and a variance of one. Since we will ultimately decode the

received information using a trellis, it is therefore, worth mentioning how we have constructed the trellis for this code.

In 1974, Bahl *et al.* [38] proposed a method of presenting codewords of a linear block code by means of a trellis. This method is referred to as the BCJR construction method, which we have used for our trellis. It is worth mentioning that in 1978, Massey [39] made a further study of the problem of representing a block code by a trellis, and proposed an alternative construction. In 1988 Forney [40] described what he called "the trellis diagram of a code", which resulted in an explosion of awareness in the subject. For the interested reader in [41], R.J McEliece provides an excellent, comprehensive tutorial on this subject. It is important to note that different trellis representations are obtained from different permutations of the symbol positions of any given block code. Therefore, there can be many different trellis configurations for a unique BCH code.

In order to remove all ambiguity we shall define what we mean by a trellis as stated in [25], and all its related terminology, used throughout this thesis. A trellis is a set of nodes interconnected by unidirectional branches. Every node is assigned an integer j which is referred to as the *depth* of the trellis, and an *(N-K)*-tuple known as a *state*, denoted by $S_i(j)$ for a certain integer i. Therefore, there exist at most 2^{N-K} states, which are ordered from *0* to $2^{N-K}-1$, with *0* referring to the all zero state *(N-K)*-tuple. There is only one node at depth *0* denoted by $S_0(0)$ and only one node at depth n, denoted $S_0(n)$. A path in the trellis of length L is a sequence of L branches. The set of node subscripts at depth j is denoted by I_j, representing a subset of the integers $\{0,1,2,...,2^{N-K}-1\}$. All that being said, the trellis for the linear block code is constructed as follows:

1. Initialize $S_0(0)$, i.e. at depth $j=0$ the trellis contains only one node.

2. For each $j = 0,1,...,(n-1)$ the set of states at depth *(j+1)* is obtained from the set of nodes at depth j by the following[3]:

$$\text{for all } i \in I_j, t \in I_{j+1}, l \in \{0,1\} \quad S_t(j+1) = S_i(j) + \alpha_i^l h_{j+1} \tag{3.5}$$

where α_i^l are binary inputs

$$\alpha_i^l = \begin{cases} 1 & l = 1 \\ 0 & l = 0 \end{cases} \tag{3.6}$$

3. For each i in I_j, form connecting branches between node $S_i(j)$ and two nodes at depth *(j+1)* for two different binary values $\alpha_i^l \in \{0,1\}$ according to Equation 3.6. Label each branch by the corresponding α_i^l value.

At depth $j=n$ the final node, $S_0(n)$ is given by:

$$S_0(n) = S_0(0) + \sum_{i=0}^{n-1} \alpha_i^l h_i = 0 \tag{3.7}$$

Since both $S_0(n)$ and $S_0(0)$ are the all-zero states, the above equation implies that:

$$\sum_{i=0}^{n-1} \alpha_i^l h_i = 0 \tag{3.8}$$

By deploying the above construction method, all possible binary N-tuples (2^N) are formed in the trellis.

---

[3] At this point we shall remind the reader that the parity check matrix H is what characterizes a code C such that for $c \in C$ where $c = (c_0, c_1, ...c_i..., c_{n-1})$, $c_i \in \{0,1\}$, $cH^T = c_0h_0 + c_1h_1 + ... + c_{n-1}h_{n-1} = 0$ .The parity matrix of the (7,4) BCH code is given by Equation 2.2.

We illustrate the trellis construction for a BCH (7,4) code with the parity check matrix:

$$H=\begin{bmatrix}1 & 0 & 1 & 1 & 1 & 0 & 0\\ 1 & 1 & 1 & 0 & 0 & 1 & 0\\ 0 & 1 & 1 & 1 & 0 & 0 & 1\end{bmatrix}=[h_0,h_1,...,h_6] \tag{3.9}$$

It is obvious that we have a total of 2^{7-4}=8 states as illustrated in Figure 3.1.

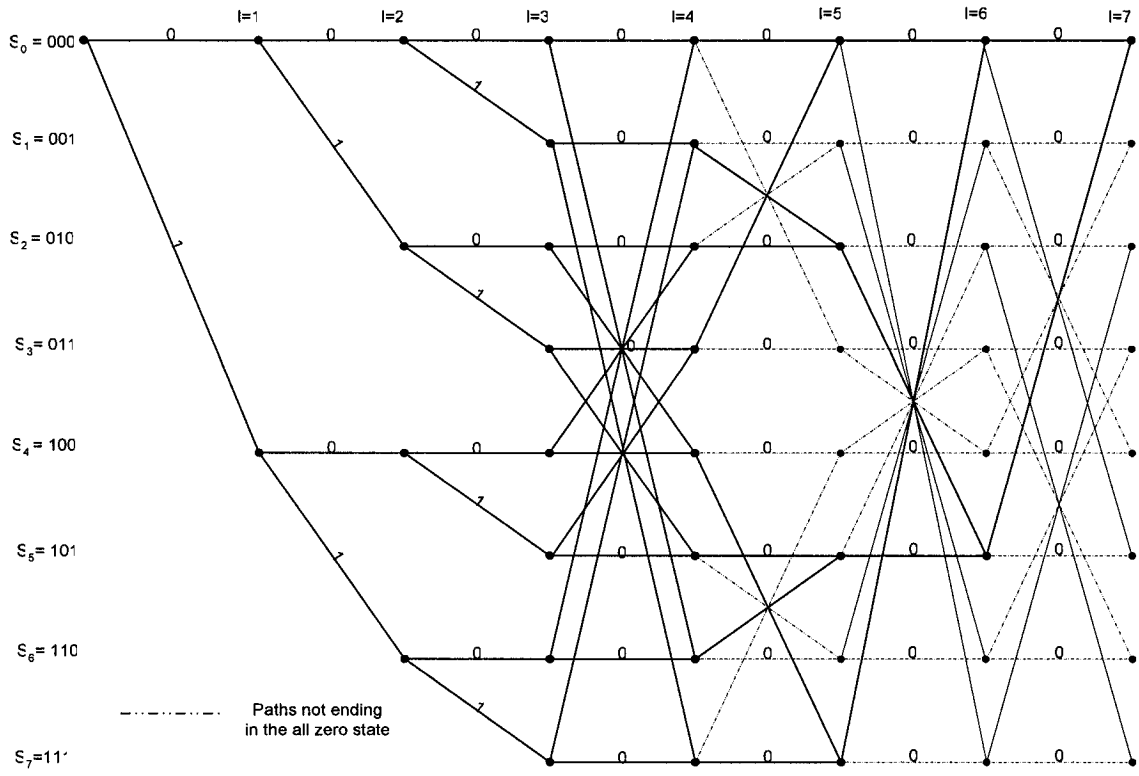

Fig. 3.1: Trellis of the Binary (7,4) BCH Code

Finally we proceed by removing all branches which are not part of the path ending in the final all-zero state at depth $n=7$. Hence the result is the expurgated trellis in Figure 3.2.

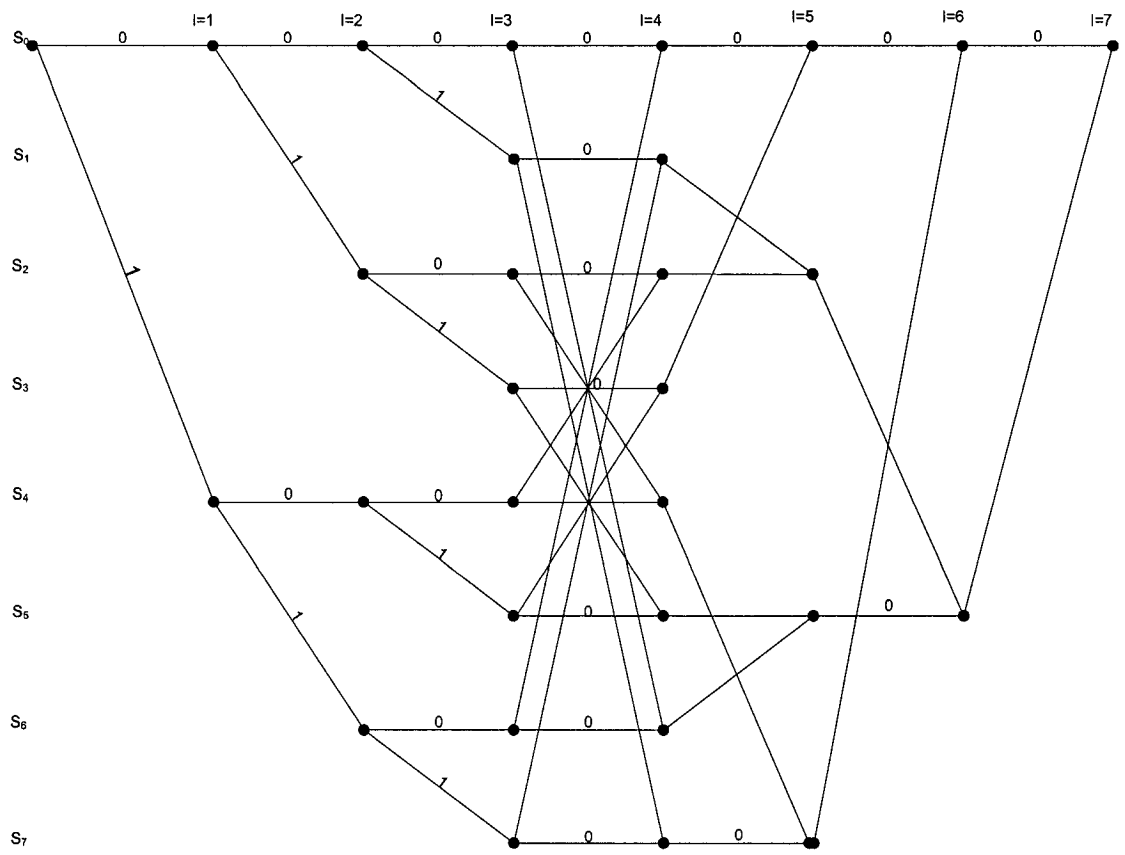

Fig. 3.2: Expurgated Trellis For the Binary (7,4) BCH Code

We would like to point out to the reader that the original simulation was carried out using both the (31,26) BCH and the (7,4) BCH codes. However, given the complexity of the trellis of the (31,26) BCH we have considered the (7,4) BCH code for our example throughout the rest of this chapter.

3.4.3 An Example of Trellis Reduction Using the Chase Algorithm

Given the received channel information:

$$\mathbf{r} = \{1.24, -0.76, 2.14, \underline{\mathbf{-0.14}}, 1.38, 2.06, 0.58\}.$$

We consider $p = 1$ least reliable[4] position. Therefore,

$$\mathbf{y} = \{1, 0, 1, \underline{\mathbf{0}}, 1, 1, 1\}$$

we have a total of $2^p = 2^1$ error patterns:

$$\boldsymbol{E}_1 = \{0, 0, 0, \underline{\mathbf{0}}, 0, 0, 0\}$$

$$\boldsymbol{E}_2 = \{0, 0, 0, \underline{\mathbf{1}}, 0, 0, 0\}$$

and a total of $2^p = 2^1$ test patterns:

$$\boldsymbol{T}_1 = \{1, 0, 1, \underline{\mathbf{0}}, 1, 1, 1\}$$

$$\boldsymbol{T}_2 = \{1, 0, 1, \underline{\mathbf{1}}, 1, 1, 1\}.$$

We can thus represent this test sequence by means of a trellis resulting in the reduced state trellis of Figure 3.3, which will be decoded using the Max-Log MAP algorithm.

[4] We define the term least reliable by the bit having the lowest absolute channel value given the sequence of channel values for a received codeword.

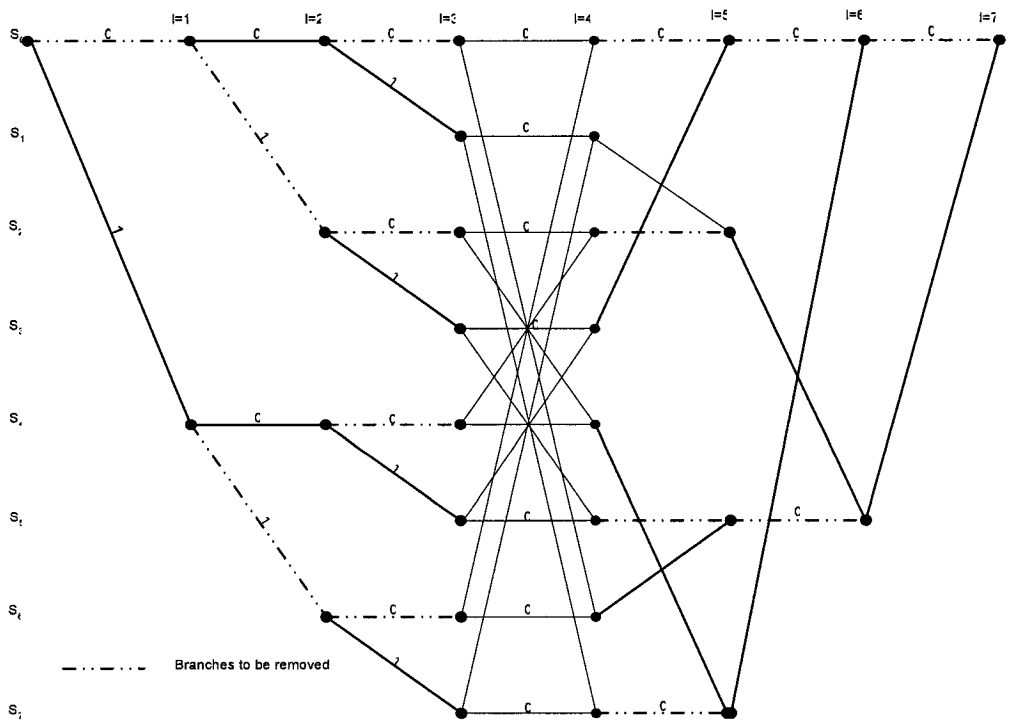

Fig. 3.3 Reduced State Trellis by Means Of the Least Reliable Bit

Compared to Figure 3.2, one can conclude that the trellis has been significantly reduced. On the other hand, using the same concept we can also reduce the trellis complexity by means of finding the *most* reliable bits. Again we consider the same received channel information where in this case the most reliable bit is the third bit:

$$\mathbf{r} = \{1.24, -0.76, \underline{\mathbf{2.14}}, -0.14, 1.38, 2.06, 0.58\}.$$

In this case the reduced state trellis will result in Figure 3.4.

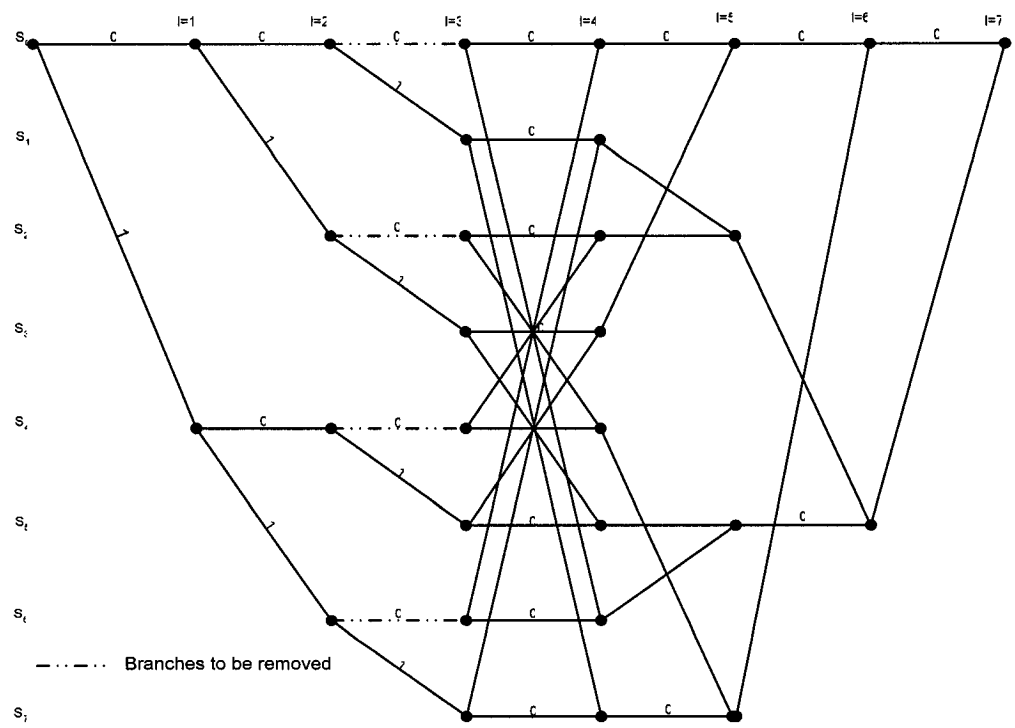

Fig. 3.4 Reduced State Trellis by Means of the Most Reliable Bit

3.4.4 Discussion of Associated Problems

In the first case, although we are capable of significantly reducing the trellis complexity, we pay a high price in terms of performance. Conversely in the second case we can achieve a performance similar to the Max-Log MAP algorithm; complexity wise, however, we don't have a considerable reduction. Rather than presenting results that have no significant importance, we shall attempt to analyze what went wrong. Referring to Figure 3.1, one can see that the trellis of the linear block code fully expands only at a depth of *n-k* trellis segments. Therefore, pruning prior to this expansion will remove the possibility of good paths and effectively result in error. On the other hand since we are dealing with an expurgated trellis as in Figure 3.2, where branches that are not part of the path ending in the final all zero state are removed; then it does not make sense to further

remove the remaining branches in the last *N-K* trellis segments. Hence we can only remove branches after a depth of *N-K* from the begging of the trellis and before a depth of *n-k* until the end of the trellis, since only in this region is the trellis fully expanded. In our particular example this region is limited to only one segment in the trellis, which is the reason why we have carried our simulations with the (31, 26) BCH code. Obviously as the code length increases the region where the trellis is fully expanded becomes more significant.

In [13] Pyndiah demonstrated noticeable results by using the Chase algorithm, however as stated earlier in Chapter 2, Pyndiah's LLR computation is based on augmented decoding and it does not use a trellis. The soft output depends on the decision D and its competing codeword **C**. To find **C** one must increase the size of the space spanned by the Chase algorithm, which will also increase the complexity. If the competing codeword is not available, the soft output is defined by $soft\_out = \beta \cdot D$ where β is set in an empirical manner, and it is a very rough estimate of the soft output. Ultimately, to exactly apply all of the various factors of Pyndiah's interpretation of the Chase algorithm to a trellis based algorithm such as the Max-Log MAP in order to reduce complexity, will in fact further increase the complexity of the decoding procedure. In this method the reliability of each bit is based only on the Euclidian distance. Clearly this cannot be the only constraint in a trellis-based decoding system which is based on the Max-Log MAP algorithm. Henceforth, in order to take advantage of the robustness of the Max-Log MAP algorithm we need to consider a new approach, one that takes into account the various components of the algorithm itself, while avoiding various reliability and weighting factors as is the case in [13].

3.5 Summary

In this chapter, two groups of complexity reduction techniques were discussed, namely, computational and structural complexity. We also talked about the efficiency of the stopping criterion in terms of complexity reduction. Furthermore, a method for constructing the trellis of a linear block BCH code was also demonstrated by using the BCJR construction technique. Moreover, we discussed a possible scenario of combining the Max-Log MAP with the Chase algorithm for the purpose of complexity reduction, and finally we wrapped up the chapter with the discussion of the associated problems with such a scheme.

CHAPTER 4

A NEW REDUCED COMPLEXITY ALGORITHM BASED ON TRELLIS PRUNING

As explained in the previous chapter, in terms of complexity reduction, we will only consider a structural reduction technique, rather then a computational method. In Chapter 3, it was concluded that for a SISO MAP decoder, the complexity of the decoder is proportional to the complexity of the corresponding trellis. Therefore, by reducing the trellis structure and combing that with a computationally reduced algorithm such as the Max-Log MAP, we expect to observe a large overall reduction in the decoder complexity. We know that the complexity of a SISO decoder is expressed by the sum of its various components. As we have already concluded, for the Max-Log MAP decoder, the three main components are the α module, the β module, and the γ module. From Equations 2.26 and 2.27 we have:

$$\alpha_j(s') = \sum_{(s,s')} \gamma_j(s,s') \cdot \alpha_{j-1}(s) \tag{4.1}$$

$$\beta_{j-1}(s) = \sum_{(s,s')} \gamma_j(s,s') \cdot \beta_j(s') \tag{4.2}$$

Clearly, the Equations 4.1 and 4.2 indicate that both the α, and β modules are dependant on γ. We can thus conclude that by reducing the complexity in the γ calculations we can affect the overall complexity of the decoder. We know that γ is the branch transitional probability, thus reducing the number of branches will also reduce the number of times γ needs to be calculated. From this statement we gather that using the average number of branches in a trellis section is considered a valid criterion in terms of complexity reduction. The trellis of a linear block code is shown in Figure 4.1. One can see that in each trellis segment, a total of two branches are leaving a particular state. However, as stated previously in Chapter 3, we only consider the segments where the trellis has fully expanded; we can make the following general statement pertaining to our trellis: *for every state in a fully expanded trellis a total of two branches depart from each node*[5].

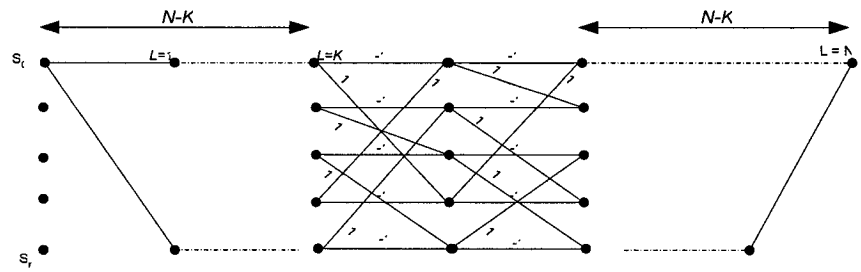

Fig. 4.1: An Example of a Trellis of a Linear Block Code

Given, a trellis of a linear block code $C(N, K)$ with number of states $S=2^{(N-K)}$, and I decoding iterations, we can define the complexity per SISO decoder as:

$$\varsigma = 2[N - 2(N - K)] \cdot S \cdot I \tag{4.3}$$

[5] Recall that a linear BCH code does not have a unique trellis representation. In this case we are referring to the one used in our simulations.

$$= 2(2K - N) \cdot S \cdot I \tag{4.4}$$

where N is also the length of the trellis, (N-K) is the regions where the trellis is not fully expanded, i.e. this region refers to the beginning of the trellis where the trellis is not fully expanded, and at the end where we are only considering the paths that lead to the all zero state. We multiply this by two, since there are always two branches entering and leaving each node. At this point, it is important to restate what was mentioned in Chapter 3. There is no unique trellis construction for a BCH code; hence various trellis constructions might have segments where all the states are not used. However, in our calculations we consider the worst case scenario, where there are branches leaving from every single state.

4.1 Pruning Trellis by Means of a Threshold

Obviously, our goal is to eliminate certain branches without paying a high price in terms of performance. In order to evaluate the value of different branches in the trellis in the decoding process, an in depth analysis of the branch transition probability (γ) is required. We recall from Chapter 2 that γ is a function of the received channel value ($L_c \cdot y$), and the *a priori* value $L(x)$ associated to each bit x. Given the fact that the channel information value does not change, obviously the only "variable" in each iteration is the *a priori* probability associated to each bit. As stated previously, the extrinsic output of one SISO decoder is the *a priori* input of the other SISO decoder. Intuitively, we know that in each iteration the extrinsic information that is exchanged between the decoders is continuously refined. Essentially this information is used in order to decrease the probability of error.

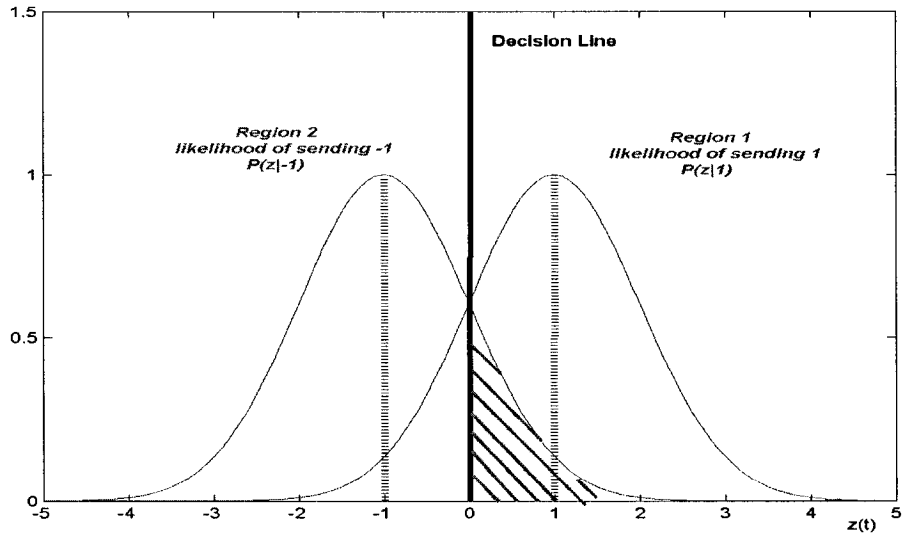

Fig. 4.2: Conditional Probability Density Functions: $P(z \mid 1)$, $P(z \mid -1)$

Figure 4.2 shows a random Gaussian variable *z(t)* with a mean of either +1 or –1, representing the channel value which is the transmitted signal plus noise. It is important to note that the only performance degradation we consider is due to white Gaussian noise. Figure 4.2, plainly indicated that the probability of error in detecting a +1 upon the transmission of –1 is in the shaded region of the diagram where the absolute value of *z(t)* is fairly small. Hence as $|z(t)|$ increases, the probability of error will decrease. This is exactly the goal of the iterative process, where in each iteration the extrinsic information is used to prevent erroneous detection of certain bits. We can also conclude that for the bits with a relatively higher channel value, the probability of making an error in symbol detection is less than the bits that have a lower$|z(t)|$, which are closer to the decision region of Figure 4.2. Referring back to the trellis of a linear block code, based on the formula in Equation 2.36, we can declare that for each trellis segment the absolute value

of gamma remains the same for all the branches within that segment, while its sign will change according to the value of the branch (if the branch corresponds to a $1 \rightarrow +\gamma$, if branch corresponds to a $-1 \rightarrow -\gamma$). That being said, we will proceed by pruning branches in a given trellis segment *j*, by comparing their corresponding gamma value to a predefined threshold $\tau > 0$, such that:

$$\underline{Case\ 1}\text{: } (if(L_c \cdot y_j + L(x_j) \geq \tau)$$

$$\{prune\ \ all\ \ -1\ \ branches\ \ within\ \ segement\ \ j\}$$

$$\underline{Case\ 2}\text{: } (if(L_c \cdot y_j + L(x_j) < -\tau)$$

$$\{prune\ \ all\ \ +1\ \ branches\ \ within\ \ segement\ \ j\}$$

In other words, by evaluating the channel and the *a priori* value we make an early decision on the transmitted bit. Let us further explain this concept by means of an example. Assuming that we have a scenario such as the one depicted in *case 1*, then for a corresponding trellis segment the number of gamma calculations, i.e. (the number of branches) will be reduced by one half as indicated in Figure 4.3.

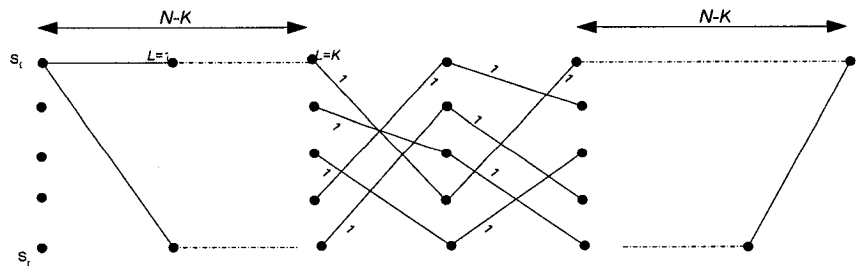

Fig. 4.3: A Pruned Trellis

4.1.1 Setting an Appropriate Threshold Value

One of the main questions that we are faced with is how to find an appropriate threshold value for a given system, which will provide a desirable trade off between the complexity and performance. Intuitively, we deduce that setting a high threshold value will have less depredating effect on the BER performance. Another important concern is that if we initially cut too many branches using a low threshold, it will result in a poor approximation of the extrinsic information for the subsequent iterations. Hence, this can negatively affect the convergence of the algorithm along with a negative effect on complexity reduction. Consequently, the threshold value is one of the most important factors governing the performance of the algorithm. The adopted method to find a suitable threshold is by means of computer simulations. Given the fact that different systems are governed by various properties, it is not possible to come up with a general value by means of a mathematical formula applicable to all systems. Furthermore, it is not feasible to simulate *every* possible system in order to come up with a unique value.

Therefore, we start our analysis by setting a threshold value based on the maximum received channel value within a codeword sequence, and setting that value as a threshold for all the bits within that particular codeword. In this method the system will automatically choose a threshold for each received codeword based on the channel information. For example, if we have a block of information where $N = 5$ as depicted in Figure 4.4.

-1.4	0.68	-0.05	***3.2***	0.91
0.07	2.6	1.34	0.43	***-2.89***
3.56	0.45	2.03	1.67	3.21
-0.09	1.23	-0.84	***3.89***	0.84
-2.3	-0.02	1.95	-0.74	-1.45

Fig. 4.4: Sample of the Range of Received Channel Values

We can see that for the first codeword the maximum channel information $|L_c \cdot y|$ is 3.2, therefore upon pruning in later iterations, the extrinsic information for all the bits within that codeword will be compared to 3.2. Whereas, for the second codeword the maximum $|L_c \cdot y|$ is |-2.89|, therefore all the bits within C_2 will be compared to this value and so on. We then proceed by changing the threshold value from 100% of the highest $|L_c \cdot y|$ value to 75% and gradually down to 50%. Figure 4.5 indicates the result of simulating with a turbo product code, where the horizontal codewords and the vertical codewords are encoded by identical BCH (31, 26) encoders which we shall refer to as BCH$(31,26)^2$. Pruning starts after the second iteration; where one iteration corresponds to a horizontal followed by a vertical decoding step. The complexity varies from 15% to 45%. The BER vs. SNR plot of Figure 4.5 indicates 4 curves, three of which correspond to pruned trellis and one is the original trellis with Max-Log MAP. As explained, we can see that as we decrease the threshold we pay a price in terms of BER performance. Interestingly when the threshold value is set to the highest channel value, we get the exact performance as the Max-Log MAP with the original trellis.

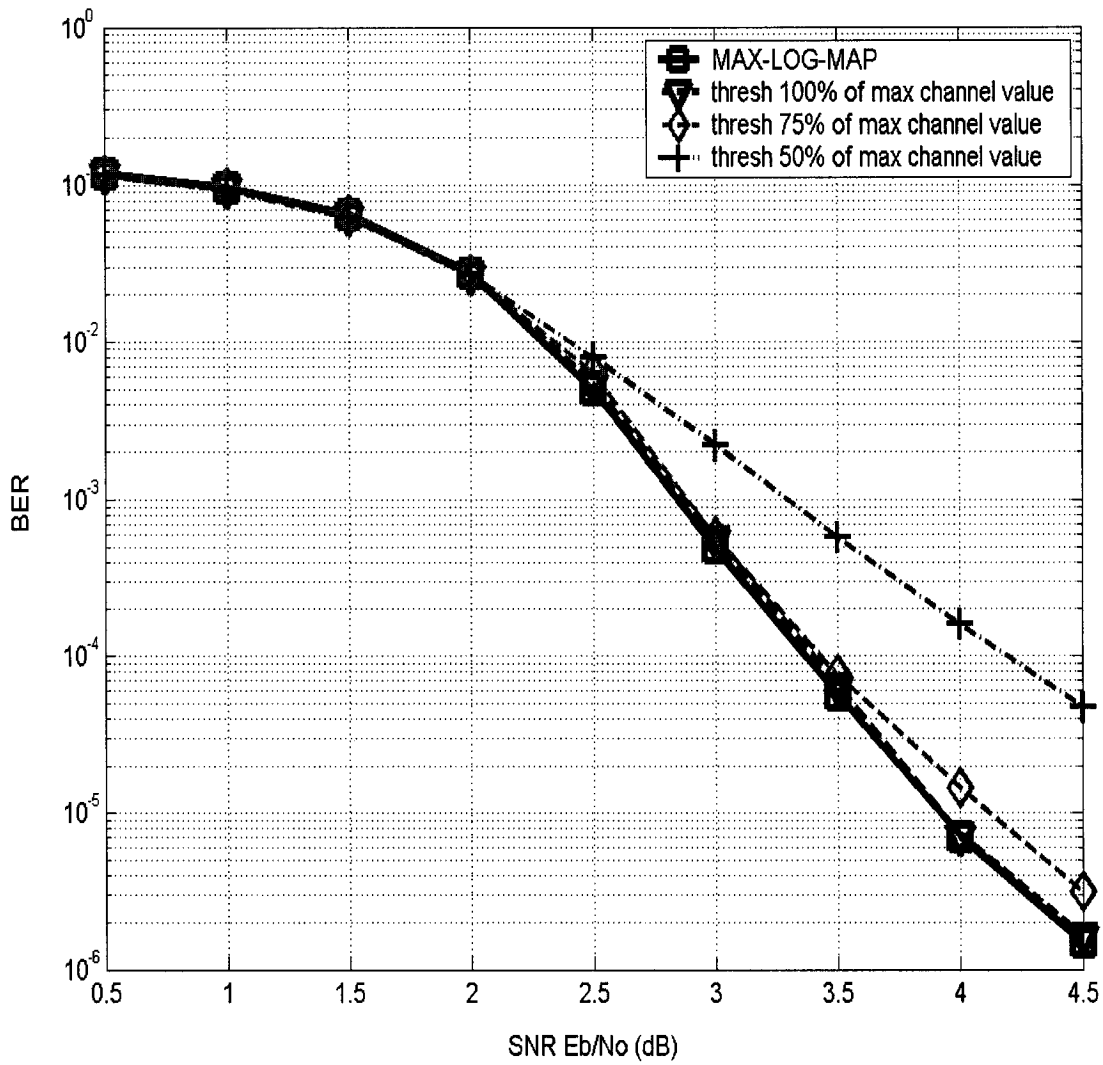

Fig. 4.5: BCH $(31,26)^2$, Pruning Starts at 2nd Iteration.

In this particular case we only have a complexity reduction of about 15%. This is an indication that not a lot of branches are being pruned. This implies that the threshold value is set too high since not many $|L_c \cdot y|$ values exceed the threshold value. Again at this point we are faced with the delicate balance of complexity versus performance. As previously mentioned, the threshold value plays a very important role in our algorithm,

thus rather then setting the threshold dynamically for each received codeword, we need to think of an alternative method.

This brings us to the following question in our analysis: "*How can we study the behavior of channel information given a predefined system?*" Of course this is a rather vague question, for the purpose of our research, however, we are only interested in how much this value varies for different intervals. For example we would like to know that if we sent coded and modulated information bits i.e. *{-1, 1}* through an AWGN channel with a SNR of 2.5 dB; what is the range of the received channel value (*r*), where $r = \{-1 + noise, 1 + noise\}$. In other words, by analyzing the range of the received channel values, one can get a better sense of finding an appropriate threshold. Furthermore, this analysis is also an indication of the approximate number of bits that could be affected by a given threshold. We proceed by plotting the density functions for a $(31, 26)^2$ BCH product code, for different SNR values in the range of 0.5dB to 4.5dB, where transmission is over an AWGN channel with BPSK modulation. The range of the received channel values in Figures 4.6, 4.7, and 4.8, are distributed over the transmission of 10 000 blocks for each SNR value, where 1 block corresponds to 31×31 information and parity bits.

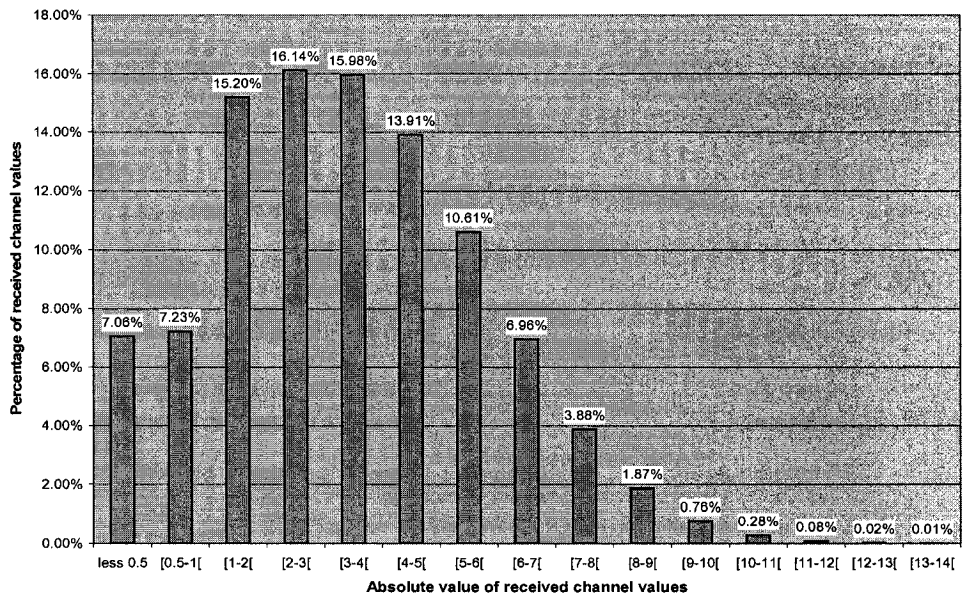

Fig.4.6: Percentage of the Number of Bits vs. the Absolute Value of the Range of their Corresponding Recieved Channel Values for 9.61×10^6 Bits transmitted Over an AWGN Channel at a SNR of 0.5 dB.

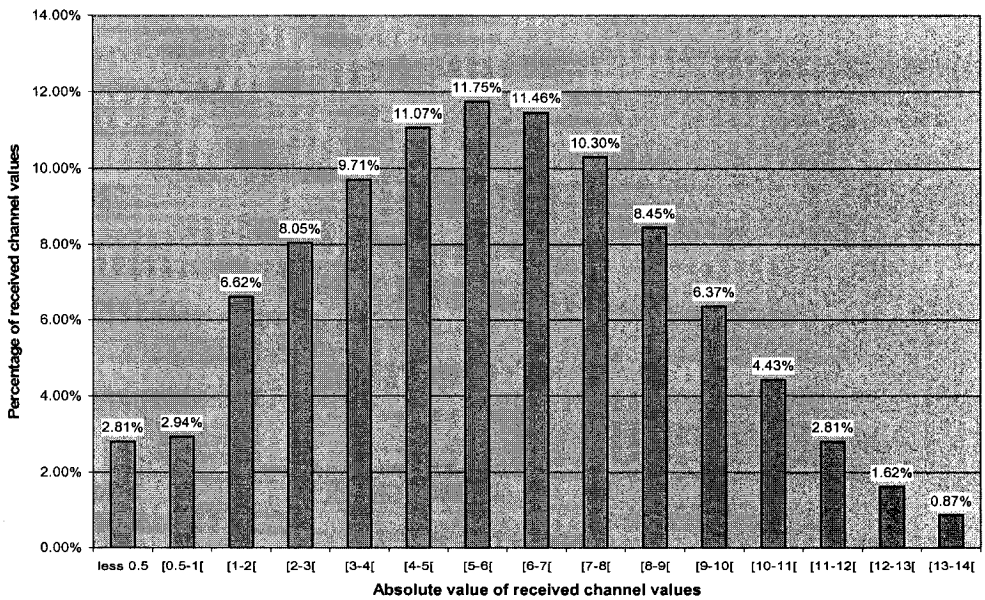

Fig 4.7: Percentage of the Number of Bits vs. the Absolute Value of the Range of their Corresponding Recived Channel Values for 9.61×10^6 Bits transmitted Over an AWGN Channel at a SNR of 3 dB.

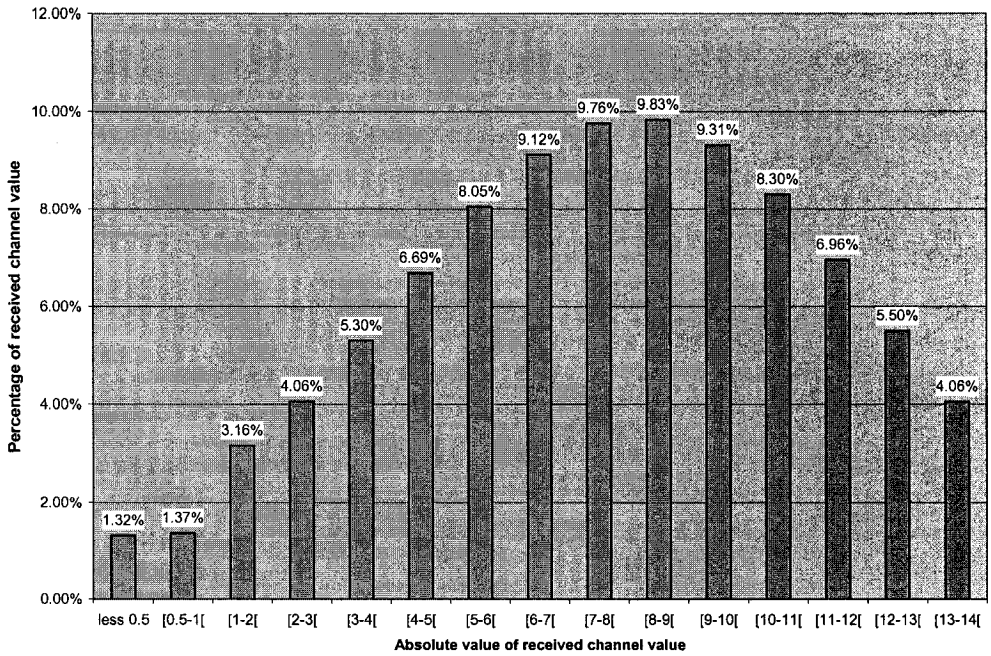

Fig. 4.8: Percentage of the Number of Bits vs. the Absolute Value of the Range of their Corresponding Recived Channel Values for 9.61×10^6 Bits transmitted Over an AWGN Channel at a SNR of 4.5 dB.

The analysis of Figures 4.6 through 4.8 is indicative of certain factors; such that as the SNR increases the percentage of channel values with a higher absolute value will also increase. Therefore, as the system's SNR value increases, the threshold value has to increase as well. Moreover, considering the example of Figure 4.6, it is obvious that a threshold value set to the highest absolute value of the received channel information, will only aim at less then 1% of the received bits. This further proves our hypothesis: it is not reasonable to set the threshold value as the highest received channel value, since for low SNRs, less then 1% of the $|L_c \cdot y|$ values fall within that range. Aiming at a complexity reduction of approximately 50%, we proceed by setting the threshold value within the range with the highest percentage value for each of the SNRs in Figures 4.6 through 4.8.

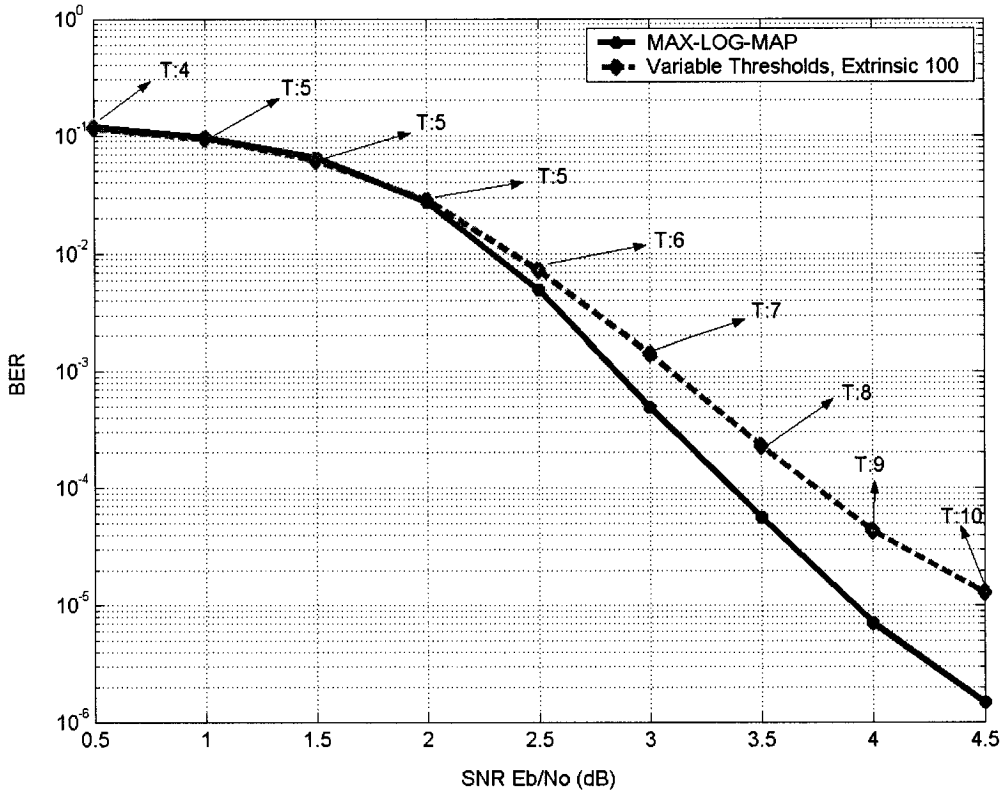

Fig. 4.9: BCH(31,26)[2] Pruning Starts From 2nd Iteration With Variable Thresholds

The BER vs. SNR curve of Figure 4.9 is the comparison of pruning with various thresholds, with that of the original trellis. However, the ~0.5dB loss in BER performance and a complexity of ~ 38% for a SNR of 3dB, is an incentive to consider the issue of finding an optimum threshold from yet another point of view.

At this point, it is important to remind the reader that the thought pattern behind the methodology leading to the steps involved in this algorithm were developed in a parallel fashion. Since the very nature of the propose algorithm deals with a balance between complexity and performance, it is not possible to only focus on one matter without dealing with the other. Going back to the issue of complexity, one main objective is a reduction of approximately 50%, obviously without a considerable degradation in the

performance. For example, an un-pruned trellis of a BCH (31,26) code, more specifically the one depicted in our simulations, requires (21*2)*32*31 or 41664 gamma calculations per block per decoder, we aim at half of this number or 20832 gamma calculations per decoding iteration. Therefore we will use this factor as a stopping criterion. Now that we have defined a limit for our system, we shall go back to the matter of finding a threshold.

Intuitively, we know that the higher the threshold value is, the fewer the number of pruned branches is; consequently there will be less BER degradation. We proceed by testing the limits of the system described above in terms of increasing the threshold value while studying the BER performance for a given SNR. Given our defined parameters, the curve in Figure 4.10 indicates that for a $(31,26)^2$ BCH product code transmitted over an AWGN channel using BPSK modulation with trellis pruning characterized by a 50% complexity reduction, at a SNR value of 2.5 dB, an optimum threshold value should reside within the range of *[13-15]*. We have followed the same methodology to find a threshold value corresponding to other SNRs given the same system parameters.

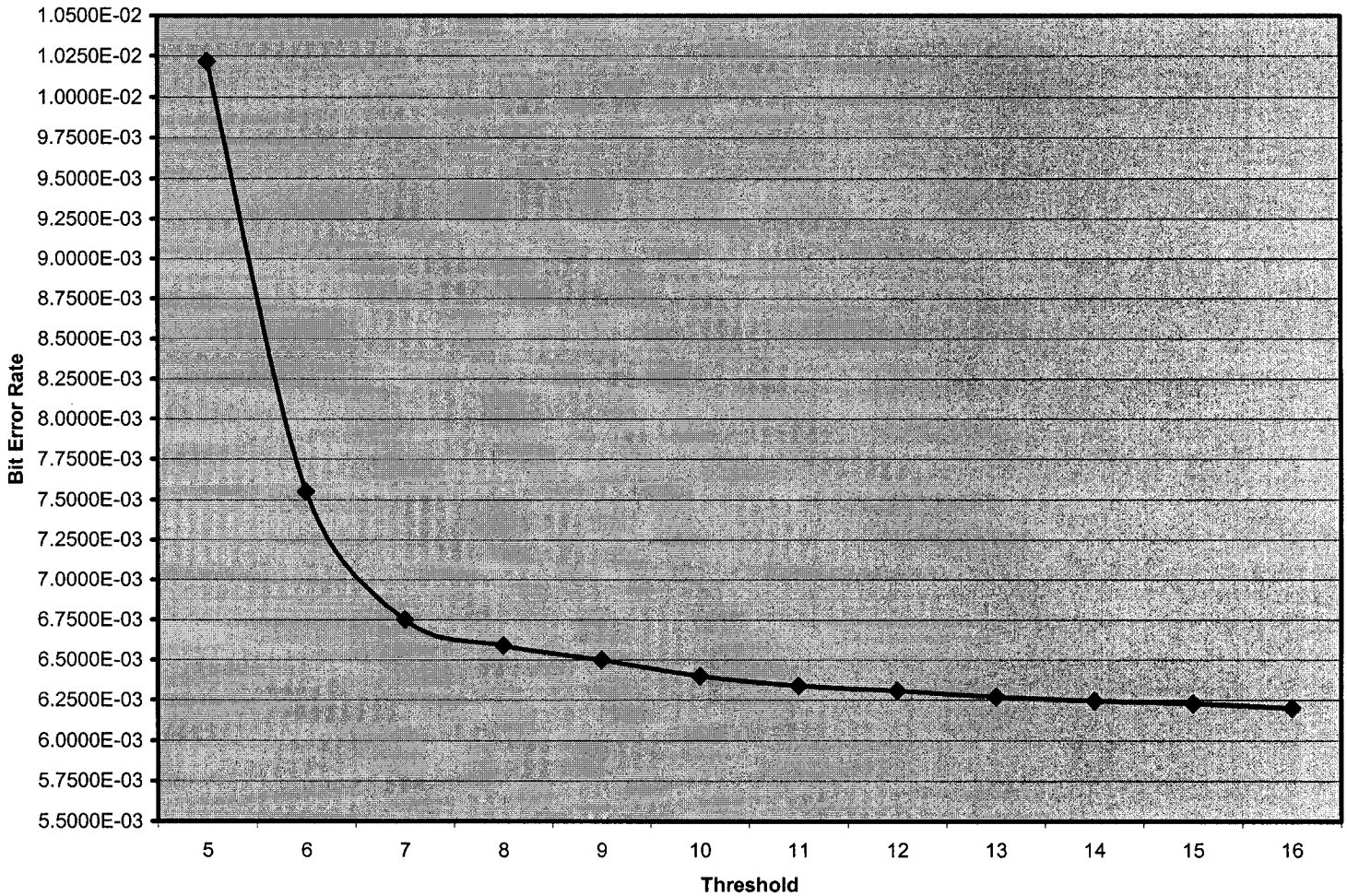

Fig. 4.10: Bit Error Rate vs Varying Threshold for 2.5dB BCH(31,26)^2 (NOTE: stopping criterion is 50% of the number of gamma calculations compared to a system with 10 H & V decoding iterations.)

4.3 The Addition of a System Level Correction Factor

Having dealt with the concepts behind finding a reasonable threshold value based on complexity/performance trade offs; we will now tackle another interesting challenge that arises as a result of pruning: "*How to avoid the BER degradation of a pruned system as opposed to a system using an un-pruned trellis?*" In order to overcome the BER performance issue, we need an in depth analysis of the Max-Log MAP operation. As outlined in Chapter 2, Equation 2.40 indicates the following:

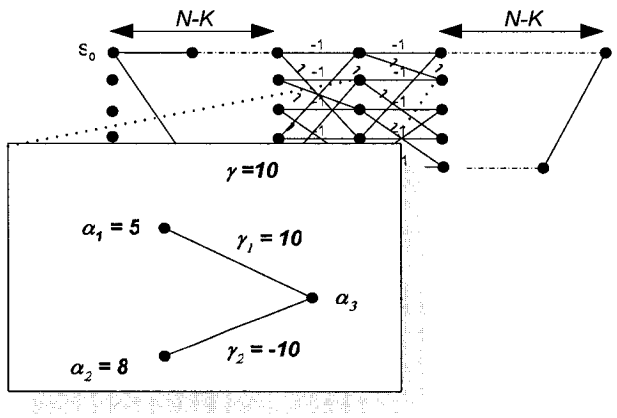

Fig. 4.11: An Example of Max-Log MAP

Assuming that we have a scenario as the one in Figure 4.11. The Max-Log MAP algorithm dictates that the value of α_3 is obtained by:

$$\begin{aligned} \alpha_3 &= Max\{(\alpha_1 + \gamma_1),(\alpha_2 + \gamma_2)\} \\ \alpha_3 &= Max\{(5+10),(8-10)\} \\ \alpha_3 &= 15 \end{aligned} \tag{4.5}$$

Let us assume that our threshold value is 8 and that we are to prune all branches pertaining to a value of (+1), and keep the (-1) branches. By that account, we have pruned

the branch with $\gamma_1 = +10$. Hence, Max-Log Map is now forced to choose $\alpha_3 = -2$. Obviously this will have a catastrophic effect on the rest of the trellis, since the value of α_3 will be used in the next trellis segment. Not to mention that once the trellis is simplified, the branches are permanently pruned for successive iterations; this is crucial for having a reduced complexity decoder. Therefore, there is no going back once a wrong decision has been made. We can conclude from the above analysis that even when we have a good threshold it is not enough to simply prune a branch in order to maintain a good BER. If we want to use the strength of the Max-Log MAP to our advantage we need to apply a system level correction factor to the entire trellis in addition to the segment-by-segment branch pruning. Referring back to the problem introduced at the beginning of this section; we observe that by increasing the γ value of a given branch, it is possible to avoid the problem of choosing the wrong α value as a result of pruning. This is further illustrated in Figure 4.12. In this case the pruning decision is to eliminate the (+1) branch, and keep the more reliable (-1) branch. Since we have already made a decision on the (-1) branch, we will further increase the reliability of this branch by increasing its corresponding branch transitional probability or gamma value by the multiplication of a correction factor. This same correction factor will be applied to all the branches within the trellis segment where pruning is taking place. The resulting gamma values will be the correction factor multiplied by the branch value. Referring to Figure 4.12, the gamma value pertaining to the (+1) branch, which was originally (10) will be multiplied by a correction factor of –10 and the new gamma value will now change from (+10 → -100). By that same account, the gamma pertaining to the (-1) branch, which was originally(-10)

will be multiplied by the same correction factor of -10 and the new gamma value will now change from (-10 → 100).

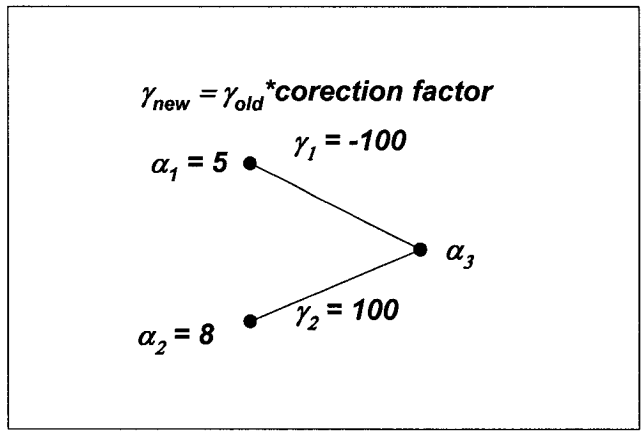

Fig. 4.12: Max-Log MAP with a Correction Factor

In this case the Max-Log MAP algorithm will choose the second branch, which is also the only branch remaining after the pruning operation.

$$\begin{aligned} \alpha_3 &= Max\{(\alpha_1 + \gamma_1),(\alpha_2 + \gamma_{21})\} \\ \alpha_3 &= Max\{(5-100),(8+100)\} \\ \alpha_3 &= 108 \end{aligned} \tag{4.6}$$

In other words by increasing the γ value we increase the *a priori* information of the bit that is to be pruned, which is exactly the same as increasing the reliability of the desired bit. Referring back to Figure 4.1, by increasing the reliability of the received bit, the value of *z(t)* will either increase towards $+\infty/-\infty$ depending on the pruning process. Nonetheless, it is important to note that if we increase the gamma values by the same constant value for all consecutive trellis segments where pruning is to take place, than we have not solved the problem at hand. Although the first segment will be corrected, since

all alpha values to be considered in the next segment will increase by the same correction factor then the problem will be repeated once again.

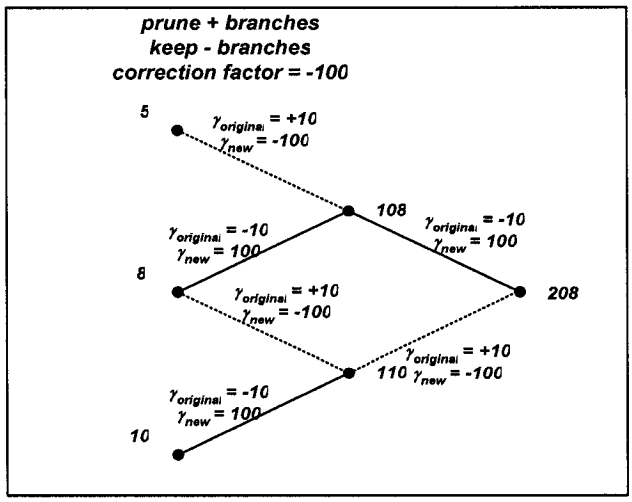

Fig. 4.13: Two Consecutive Segments in the Trellis Where Pruning Takes Place

From Figure 4.13 it is obvious that the set of alpha values in the next segment have also increased. For that reason, in order to prevent the scenario of Figure 4.11 from happening again, we will continuously increase the correction factor for each pruned segment as we traverse the trellis in the forward direction.

A detailed analysis of the simulation output at this point indicates yet another issue to be considered; this being the fact that the Max-Log MAP algorithm traverses the trellis once in the forward direction to obtain alpha values, and once in the reverse direction to obtain the corresponding beta values. Thus far, our proposed method to use a correction factor will only work while traversing the trellis in the forward direction.

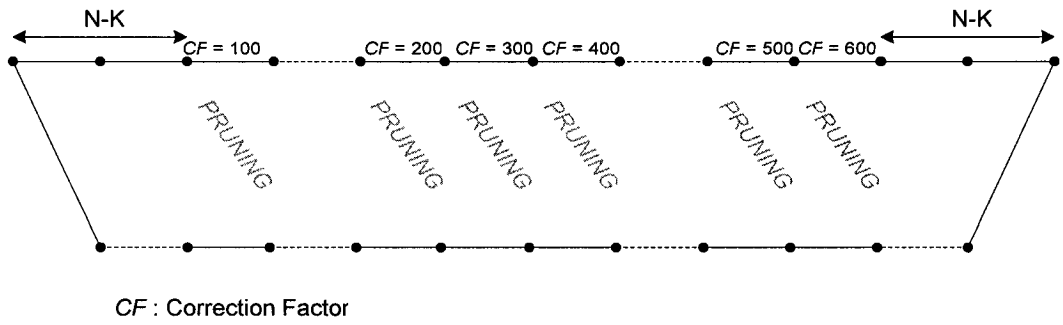

Fig 4.14: Modified Pruned Trellis with an Imposed Correction Factor in the Forward Direction.

If one would use Max-Log MAP on this modified trellis, there would be an obvious BER degradation due to errors occurring while traversing the trellis in the backward direction to obtain beta values. An alternate approach is to apply the correction factor to the trellis while traversing simultaneously in both directions. However as demonstrated in Figure 4.15, once we reach the middle section of the trellis, we will be faced again with the same problem where the corresponding alpha/beta values are within the same range thus nullifying the effect of the correction factor.

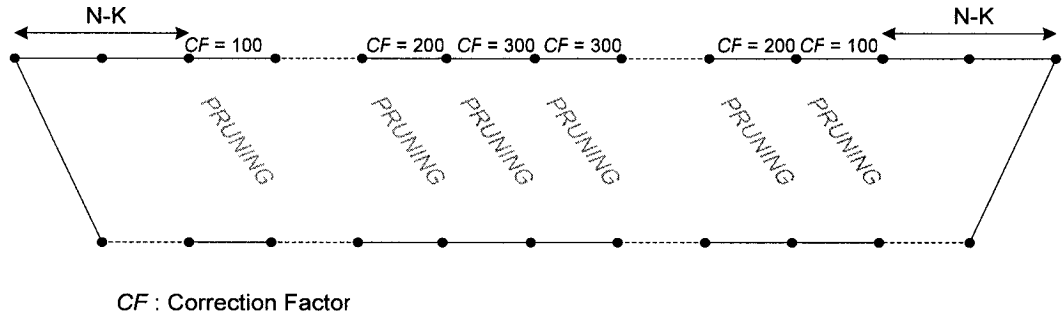

Fig 4.15: Modified Pruned Trellis with an Imposed Correction Factor in the Both Directions.

In order to overcome the previously mentioned shortcomings, we adopt the following method, where we traverse the trellis once in the forward direction, apply the correction factor to the segments which are to be pruned and obtain the gamma and alpha values. Similarly, for the set of gamma values being considered for beta calculations, we shall apply the same operations as described for obtaining alpha, but in the reverse order. Finally, we will illustrate this concept by the use of an example. Assuming we have the following received channel information:

$$R = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7\}$$

where the bits corresponding to positions r_2, r_4, r_5, and r_6 are to be pruned. Therefore, the set of gamma values, which are to be considered for the calculation of alpha, will have the following correction factors:

$$Gamma\_Alpha : \{\gamma_1, \gamma_2 \cdot 100, \gamma_3, \gamma_4 \cdot 200, \gamma_5 \cdot 300, \gamma_6 \cdot 400, \gamma_7\}$$

Conversely, the set of gamma values used in the calculation of beta will be altered such that:

$$Beta\_Alpha : \{\gamma_1, \gamma_2 \cdot 400, \gamma_3, \gamma_4 \cdot 300, \gamma_5 \cdot 200, \gamma_6 \cdot 100, \gamma_7\}$$

Although we are using two set of different gamma values for calculating alpha/beta, we are still consistent with the Max-Log MAP algorithm. Since, as previously mentioned in Equation 2.49 and repeated again in Equation 4.7, the final output of the Max-Log MAP algorithm is based on the choice of the branch with the *maximum* Log-alpha + Log-beta value, similar to our operation where we increase the branch reliability by using a correction factor.

$$L(\hat{x}_j) = L_c \cdot y_j + L(x_j) + Max_{(s,s'),x_j=+1}[\log \alpha_{j-1}(s) + \log \beta_j(s')] - Max_{(s,s'),x_j=-1}[\log \alpha_{j-1}(s) + \log \beta_j(s')] \quad (4.7)$$

In order to find a suitable correction factor, we adopt the same line of reasoning used in finding an optimum threshold value; we shall proceed by means of computer simulation. Using the same system as described previously, the simulation results indicate that a correction factor within the range of 100 is optimum in terms of performance. We have therefore adopted this value as one of our system parameters for the reduced complexity decoder.

4.4 Review of the Proposed Algorithm

Having discussed the issues regarding pruning and finding a suitable threshold, followed by applying a correction factor in order to gain a better BER performance; we shall now give a formal definition of the proposed algorithm using Figure 4.16 and the following steps:

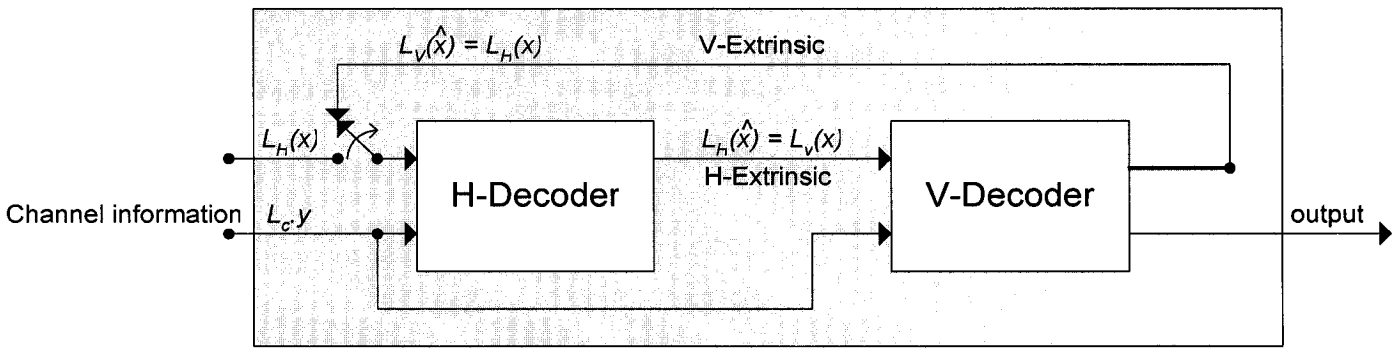

Fig. 4.16: Iterative Procedure of a Horizontal and Vertical SISO Decoder

1. Initially set the *a priori* value of the H-decoder $L_H(x_i) = 0 \qquad i = 1,...,N$.

2. Start the iterative process.

3. In each decoder, for each trellis segment within a depth of *N-K*, compare its corresponding γ value (i.e., $L_c \cdot y + L_H(x_i)$) to a predefined threshold τ such that for a given trellis section *j* we have:

{

if$((L_c \cdot y_j + L(x_j) > \tau)\{$

this indicates that we are pruning the branches pertaining to -1 and keeping the +1 branches within segment j
Prune_branch( ) *//eliminate all -1 branches within segment j*

Find_Gamma_Alpha$(\gamma_{\alpha j} = \gamma_{\alpha j} \cdot j \cdot 100)$

applying correction factor to trellis segment while traversing in the forward direction
Find_Gamma_Beta$(\gamma_{\beta j} = \gamma_{\beta j} \cdot (x - j) \cdot 100)$

applying correction factor to trellis segment while traversing in the backward direction

}

else *if*$((L_c \cdot y_j + L(x_j) < -\tau)\{$

this indicates that we are pruning the branches pertaining to 1 and keeping the -1 branches within segment j
Prune_branch( ) *//eliminate all +1 branches within segment j*

Find_Gamma_Alpha$(\gamma_{\alpha j} = \gamma_{\alpha j} \cdot j \cdot -100)$

Find_Gamma_Beta$(\gamma_{\beta j} = \gamma_{\beta j} \cdot (x - j) \cdot -100)$

}

Find_Extrinsic(pruned_Trellis, α *,* β*)*

}

At this point we ask the reader to pay attention to certain crucial aspects of this algorithm. Most importantly, the algorithm starts the pruning process at the second entire iteration, where an iteration is defined as one horizontal followed by a vertical decoding operation. In other words for the first iteration, the decoding algorithm is similar to Max-Log MAP. Naturally, we know that in the first iteration the extrinsic information is not yet refined enough so that it could be compared to a predefined threshold value. This is further realized in our observation, where pruning at the first vertical decoding produced noticeable performance degradation in terms of BER. Evidently, the effect of noise is still very significant at this stage, since we are at an early stage in the iterative process. Secondly, it is important to note that once the branches are removed from the trellis structure, the pruning is permanent, and thus it is the pruned trellis which is being used in the next subsequent decoding iteration. This enduring simplification of the trellis structure is crucial in terms of complexity reduction. Another important consideration is the number of iterations and our stopping criteria. Throughout our entire analysis our basis of comparison is the Max-Log MAP algorithm with 5 entire iterations (10 horizontal and vertical iterations). As stated previously the complexity of such a system in terms of gamma calculations can be expressed in terms of Equation 4.4, yielding a value of $\varsigma = 2(2*26 + 31)*32*10*31 = 416640$ per block. However since we are aiming at a complexity reduction of ~ 50% then our stopping criteria shall be when the number of gamma calculations has reached ~ 416640/2 per block.

Complexity wise, in our proposed algorithm, once the codeword is received from the channel, we compare each bit to a predefined threshold, and then add a correction factor to the trellis, and we perform the gamma, alpha, and beta calculation modules only

to the branches which remain in the trellis. Therefore for each codeword we only have comparison and addition operations.

On the other hand, in the augmented list decoding algorithm proposed by Pyndiah *et al.* in [13]; once the codeword is received from the channel several steps need to be performed prior to the algebraic decoding, and several steps have to be performed after the algebraic decoding in order to obtain the reliability or in other words the soft information of each bit. This implies forming test patterns and test sequences for each codeword as per the algorithm described in Chapter 3, followed by the algebraic decoding operation, then calculating the square Euclidean distance between each of the codewords in the set of possible codewords and the received channel information, in order to find the decision D, and yet again another comparison operation to find a competing codeword B. In the case where a competing codeword can not be found then each bit is subject to a reliability factor. In terms of complexity reduction the strength of this algorithm is based on the simplifications involved in the algebraic decoder. The complexity of this algorithm will increase with the number of test patterns which are initially considered by the algebraic decoder and the necessary calculations involved in obtaining a competing codeword as outlined in [13]. These test patterns are in fact dependant on the reliability of each of the bits in the received channel value. In a stationary AWGN channel, the reliability of a received channel value r_i, is given by its absolute value $|r_i|$. Whereas in our case the complexity depends only on the number of branches remaining in the trellis after a pruning operation, which is a simple comparison of $|r_i|$ to a predefined threshold. Figure 4.17 shows the performance of the proposed algorithm compared to the Pyndiah's algorithm.

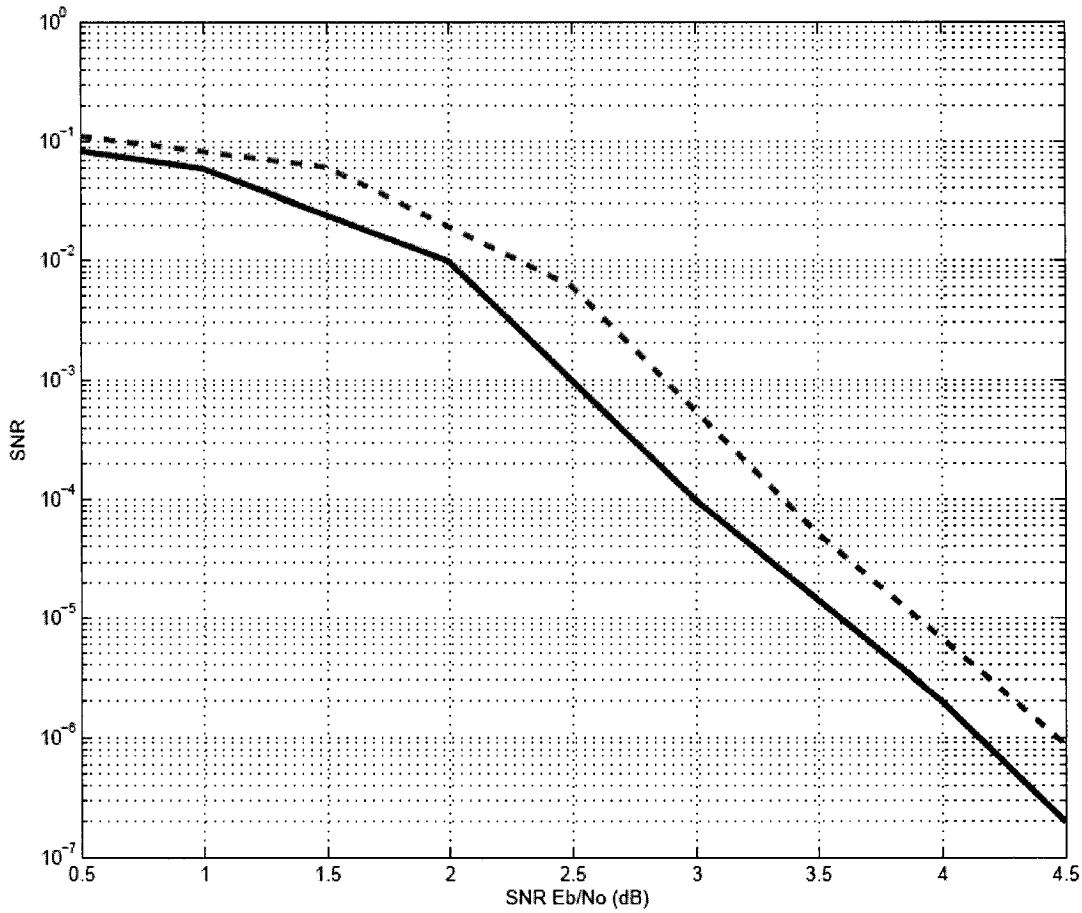

Fig. 4.17: BCH(31,26)^2 Product Code, Performance Comparison of the Pruning 50% of Branches with that of Pyndiah's Augemented List Deocding Algorithm.

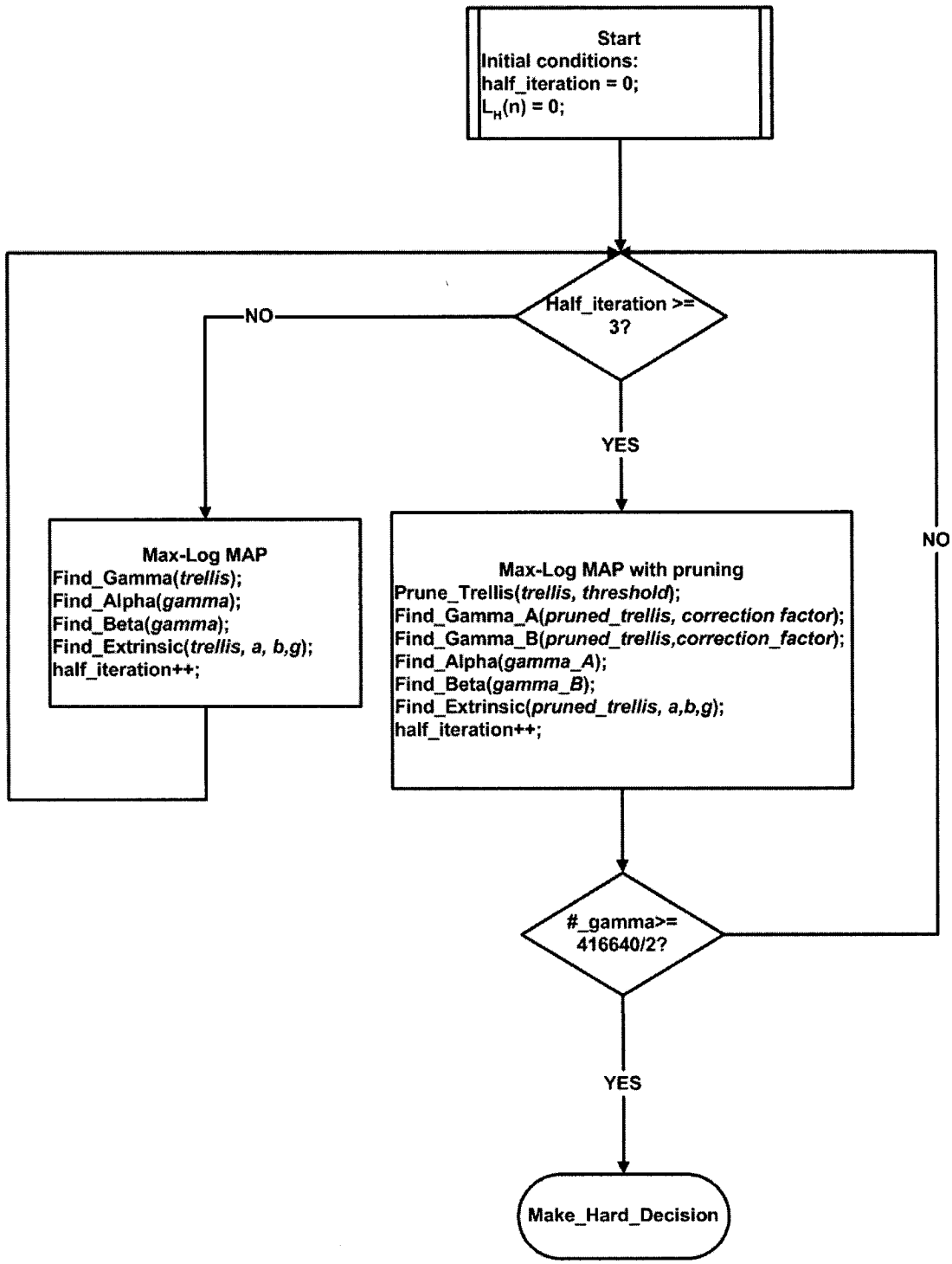

Fig. 4.18: Flow Chart of the Proposed Algorithm

We present the simulation results for a $(31, 26)^2$ BCH product code. We compare the performance of the pruned trellis with that of the un-pruned trellis under the same condition using Max-Log MAP. Figure 4.19 shows the performance of the proposed algorithm with BCH codes as component code. We use BPSK modulation over an AWGN channel. As a basis of comparison, we evaluate the performance of our complexity-reduced algorithm with the normal Max-Log MAP with a correction factor with 5 and 10 iterations. Figure 4.19, indicates that we get the same performance as that of the Max-Log MAP algorithm with correction factor with 10 iterations, with the complexity of a Max-Log Map algorithm with correction factor with only 5 iterations. Alternatively, by consider the results in Figure 4.19, we can say that given a trellis based iterative block turbo code decoder capable of obtaining a certain BER performance with a defined hardware capability of 10 iterations, by applying the proposed reduced complexity algorithm , the same BER performance can be achieved by using a decoder with only 5 iterations. Therefore, we save 50% in terms of complexity reduction, with an acceptable BER performance.

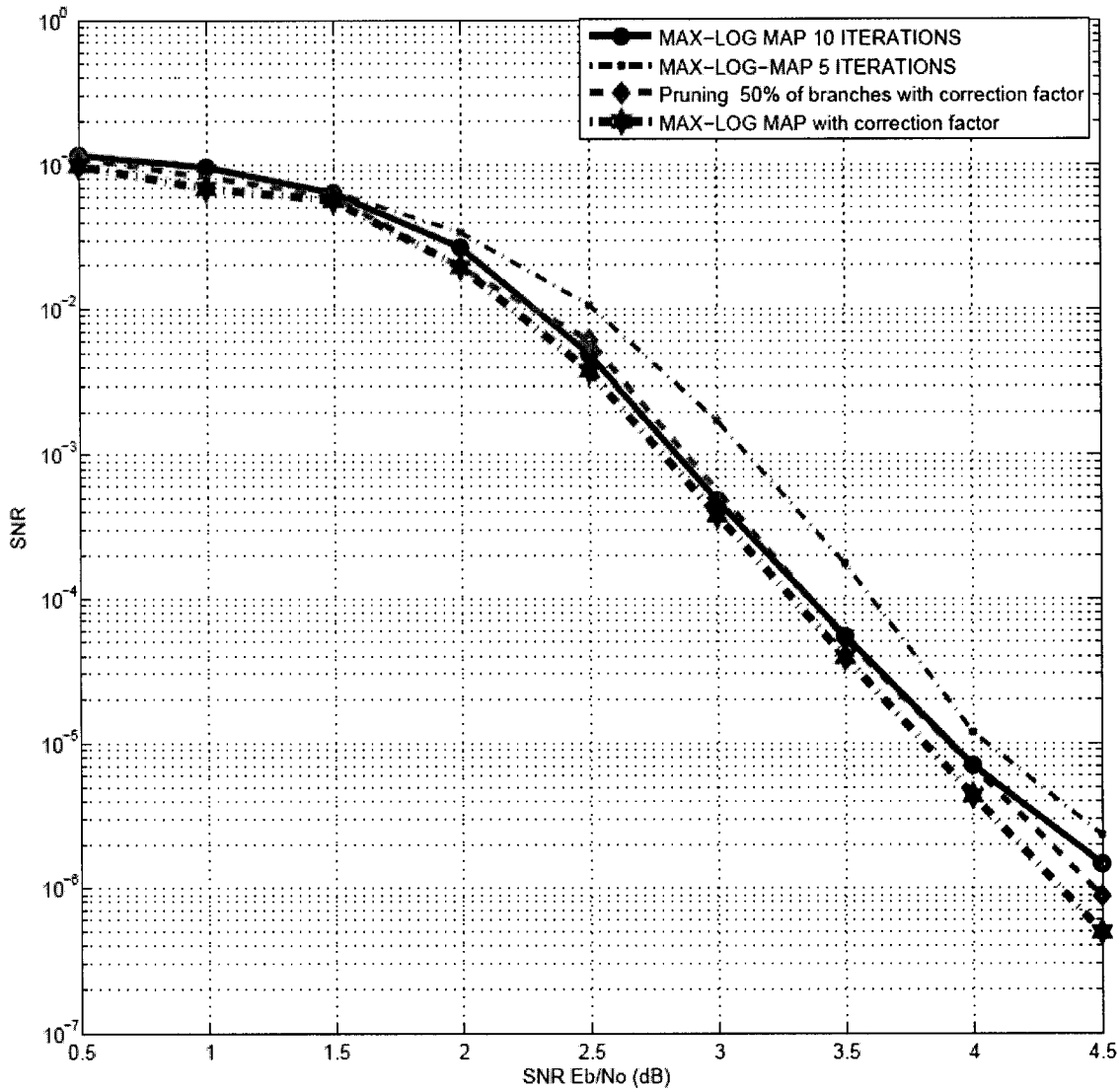

Fig. 4.19: BCH(31,26)^2 Product Code, Performance Comparison of the Pruning 50% of Branches with that of Normal Max-Log MAP with 5 and 10 Iterations

4.5 Summary

In this chapter, we discussed the details of our proposed algorithm. Most importantly we talked about the two crucial factors governing the performance of the algorithm, namely, finding an appropriate threshold value for branch pruning, and the application of an overall correction factor to compensate for the BER degradation. From the results of the simulation for a $(31,26)^2$ BCH product code, we can conclude that the proposed algorithm is capable of producing the BER performance of the Max-Log MAP algorithm with 10 iterations, however, with the complexity of only 5 Max-Log MAP iterations.

Chapter 5

CONCLUSION

In this study, we set up a general conceptual framework to address the complexity reduction in SISO iterative decoding. We start by introducing the general concepts and definitions in iterative encoding/decoding techniques, such as linear block codes, convolutional codes and concatenated block codes, namely the main components of turbo codes. We then proceed by emphasizing on trellis based decoding algorithms for iterative turbo decoding with a brief overview of augmented list decoding. Having discussed these basic concepts, we then analyze the problem of complexity reduction. We study the two basic approaches to reducing complexity: simplifying the computational complexity, and structural simplifications for the decoder. However, we focus on the latter, since a computationally reduced algorithm can be applied to a decoder with a reduced structure. Given the trellis representation of a linear block code, we conclude that the decoder complexity is proportional to the size of the trellis. From our observations, along with indications in literature, we deduce that trellis complexity can be measure in terms of

number of its branches. The idea of branch elimination is the main idea behind the algorithms proposed and studied in this thesis.

Next, we studied the effects of combining augmented list decoding such as the Chase type-II algorithm with a trellis based Max-Log MAP algorithm in a hybrid scheme. However the results were unsatisfactory mainly due to the fact that we did not employ an effective threshold unique to our code, instead we used the Chase algorithm for our pruning decisions, and realized that without the exact use of certain correction factors which were pointed out in [13], it is not possible to get an acceptable BER performance. However by studying this hybrid approach and the reasons of its failure, we were able to deeper our understanding of the proposed pruning algorithm. The underlying idea behind the pruning algorithm is simple. Starting from the second iteration where the soft information exchanged between the 2 SISO decoders is more refined, we start by pruning certain branches based on a pre-defined threshold obtained by means of computer simulations. Essentially, we prune the branches which have the most reliable information based on their received channel value and extrinsic information. Moreover, in order to compensate for the BER degradation, we subject the trellis to an overall correction factor. We measure complexity, in terms of the number of branches leaving each state for the segments where the trellis is fully expanded. Given the time variant nature of the trellis of a linear block code, we consider the worst case scenario where all states are used in terms of complexity calculations. Finally, we present the simulation results of a BCH $(31,26)^2$ turbo product code, where for the complexity of a Max-Log MAP SISO iterative decoder with only 5 iterations, we get the performance of the same system but with 10 iterations. Therefore we are able to reduce complexity by a factor of ~50%.

As for future work, it would be interesting to compare the performance of the proposed algorithm with a larger range of reduced complexity algorithms for block turbo codes. Another important suggestion is to evaluate the algorithm's performance for channel models other then AWGN, such as fading channels. Moreover, the idea of evaluating complexity in terms of hardware and FPGA implementation is another interesting avenue. Finally in this thesis, we propose a method to address the complexity reduction, the proposed algorithm can be applicable whenever the designer is faced with such a problem; the general ideas presented here can be customized to suite the desired applications.

REFERENCES

[1] C.E. Shannon, "A Mathematical Theory of Communication", *Bell System Technical Journal*, 27:379-423, 623-656, Oct. 1948.

[2] J.M. Wozencraft and B. Reiffen, *Sequential Decoding*. Cambridge, Ma: MIT Press, 1961.

[3] G.D. Forney, Jr., *Concatenated Codes*, Cambridge, MA: MIT Press, 1961.

[4] R. Gallager, "Low-Density Parity-Check Codes", *IRE Trans. on Inform. Theory*, pp. 21-28, January 1962.

[5] B. Battail, "Building Long Codes by Combination of Simple Ones, Thanks to Weighted-Output Decoding," *on Proc., URSI ISSSE*, Erlangen, Germany, pp. 634-637, Septemeber 1989.

[6] J. Lodge, P. Hoecher, and J. Hagenauer, "The Decoding of Multi-Dimensional Codes using Seperable MAP 'Filter'", *in Proc. 16th Bienneil Sympo. On Communications*, Queen's University, Kingston, Canada, pp. 343-346, May 1992.

[7] J. Lodge, P. Hoecher, and J. Hagenauer, "Seperable MAP 'Filters' for the Decoding of Product and Concatenated Codes," *in Proc. IEEE Int. Conf. On Communications*, Geneva, Switzerland, pp. 1740-1745, May 1993.

[8] S. Lin and D. J Costello, *Error Control Coding: Fundemental and Applications,* Prentice-Hall, Upper Saddle River, NJ, 1983.

[9] M. R. Soleymani, Y. Gao and U. Vilaipornsawai, *Turbo Coding for Satellite and Wireless Communications,* Kluwer Academic Publishers, 2002.

[10] J.G. Proakis, *Digital Communications*, 4th edition, McGraw-Hill Higher Education, 2001.

[11] Mathew C. Valenti, "*Iterative Detection and Decoding of Wireless Communication*," PHD Disseration, Virginia Polytechnic and State University, July 1999.

[12] C. Berrou, and A. Glavieux, and P. Thitimajshima, "Near Shannon Limit Error Correcting Coding and Decoding: Turbo Codes", *Proc. Of the 1993 Int/ Conf. on Commun. ICC1993,* pp.1064-1070, Geneva, Switzerland, May 1993.

[13] R. Pyndiah, A. Glavieux, A.Pcart, and S.Jacq, "Near Optimum Decoding of Product Codes", Proc. *IEEE GLOBCOM*, San Fransisco, USA, pp.339-343, Nov. 1994.

[14] C. Heegard, S.B. Wicker, *Turbo Coding,* Kluwer Academic Publishers, 1999.

[15] A.J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decdoing algorithm", *IEEE Journal on Selected Areas in Communications*, pp. 659-684, 1984.

[16] G. David Forney, Jr., "Maximum-likelehood sequence estimation of digital sequences in the presence of intersymbol interference", *IEEE Transactions on Infomration Theroy*, pp. 363-378, 1972.

[17] G.Ungerboeck, "Trellis-coded modulation with redundant signal sets- part I: introduction, - part II: State fo the art", *IEEE Communications Theory*, pp. 55-67, 1982.

[18] J. Hagenauer, and P. Hoeher, "A Viterbi Algorithm with Soft-Decision Outputs and its Applications", in *Proc. IEEE GLOBCOM'89*, pp.1680-1686, 1989.

[19] L.E. Baum and G.R. Sell, "Growth transformations for functions of finitie state Markov chains" *Ann. Matth. Stat.* pp. 1554-1563, 1966.

[20] L.R. Bahl, J. Coke, F. Jelinek and J.Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate", , *IEEE Trans. Inform. Theory,* pp.284-287, March 1974.

[21] R. Achiba, M. Mortazavi, W. Fizell, "Turbo Code Performance and Design Trade-Offs", 0-*7803-6521-6/2000 IEEE.*

[22] D. Chase, "A class of algorithms for decoding block codes with channel measurement information," *IEEE Trans. Inform. Theroy,* vol IT-18, pp.170-182, Jan. 1972.

[23] Advanced Hardware Architectures (AHA), PS4501: Astro 36 Mbits/s Turbo Product Code Encoder/Decoder

[24] J. Fang, F. Buda and E. Lemois, "Turbo Product Code: A Well Suitable Solution to Wireless Packet Transmission for Very Low Error Rates", in *Proc. Int. Symp. On Turbo Codes and Related Topics,* Brest, France, pp. 101-111, Sept. 2000.

[25] B. Vucetic and J. Yuan, *Turbo Codes Principle and Applications,* Kluwer Academic Publishers, 2000.

[26] S. Benedetto, D. Divsalar, G. Montorosi, and F. Pollara, "A soft input soft-output APP module for iteratuive decoding of concatenated coeds," *IEEE Communications Letters*, vil. 1, pp.22-24, Jan. 1997.

[27] P.R. Chevillat and E. Eleftheriou, "Decoding of trellis encoded signals in the presence of intersymbol interference and noise," in *Proc. IEEE International Conference on Communications,* June 1988, vol. 2, pp. 694-699.

[28] M.V. Eyuboglu and S. U. H. Qureshi, "Reduced-state sequence estimation with set partitioning and decision feedback", *IEEE Transactiond on Communications*, vol. 36, no.1, pp.13-20, Jan. 1988.

[29] T. Larsson, *A State-Space Partitioning Approach to Trellis Decoding,* zphD Thesis, Chalmers University of Technology, Goteborg, Sweden, Dec. 1991.

[30] G. Colavolpe, G. Ferrari, and R.Raheli, "Rediced-state BCJR-type algorithms, " in *Proc. IEEE International Symposium on Information Theory,* June 2000, p.322.

[31] V. Franz and J.B. Anderson, "Concatenated decoding with a reduced search BCJR algorthim," *IEEE Journal on Selected Areas in Coomunications*, vol. 16, no. 2, pp. 186-195, Feb. 1998.

[32] J. Hagenauer and C. Kuhn, "Turbo equalization for channels with highg memory using a list-sequential LISS equalizer," *in Proc. 3rd International Symposium on Turbo Codes,* Brest, France, Sep. 2003.

[33] B.J Frey anf F.R. Kschinschang,"Early detection and trellis splicing: Reduced-complexity iterstive decoding," *IEEE Journal on Selected Areas in Communications,* vol. 16, pp. 153-159, Feb. 1998.

[34] J. Hagenauer, E. Offer, and L. Papke, "Iterative Decoding of Binary Block and Convolutional Codes," *IEEE Trans. Inform. Theory*, Vol. 42, No.2, pp.429-445, March 1996.

[35] R.Y. Shao, S. Lin, and M.P.C. Fossorier, "Two simple stopping criteria for turbo decoding," *IEEE Transactions on Communications,* vol. 47, no. 8, pp. 1117-1120, Aug. 1999.

[36] A. Matache, S. Dolinar, and F. Pollara, "Stopping rules for turbo deocding," *TMO Progress Report,* Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, Aug. 2000.

[37] Y. Wu, D. Woerner, and J. Ebel, "A simple stopping criterion for turbo decoding," *IEEE Communication Letters,* vol. 4, no. 8, pp-. 258-260, Aug. 2000.

[38] B. Sklar, Digital Communcations Fundemental and Applications, Prentice Hall, New Jersey, 1988.

[39] J.L Massey, Foundation and Methods of Channel Encoding, *Proc. Int. Conf. Inform. Theory and Systems*, vol. 65, NTG-Fachberichte, Berlin, pp. 148-157, 1978

[40]G.D. Forney, Coset codes II:Bonary Lattices and Related Codes, *IEEE Trans. Inform. Theory,* vol.34 No.5, pp.1152-1187, Sept. 1988.

[41]R.J. McEliece, on the BCJR Trellis for Linear Block Codes, *IEEE Trans. Inform. Theory*, Vol. 42, No.4, pp. 1072-1092, July 1996.