AN ITERATIVE SOFT-DECISION DECODING ALGORITHM FOR REED-SOLOMON PRODUCT CODES

RONEN LEIBOVICI

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

April 2005

Your file Votre référence
ISBN: 0-494-04380-6
Our file Notre référence
ISBN: 0-494-04380-6

ABSTRACT

AN ITERATIVE SOFT-DECISION DECODING ALGORITHM FOR REED-SOLOMON PRODUCT CODES

Ronen Leibovici

Efficient and reliable transmission of data has become a necessity today for any communication system. This transmission depends on the reliable transport of information between a source and a destination. Researchers are left with the task of finding an appropriate trade-off between performance and complexity. While the decoding of turbo codes, specifically block turbo codes, is already deemed to be performant, it is possible to improve this performance while not rendering the communication system overly complex. This thesis presents an iterative soft-decision decoding algorithm for Reed-Solomon product codes. This algorithm differs from existing ones in the method it uses for selecting competing codewords, used for the soft-input soft-output iterative decoder. Selection of these codewords is based on the notion of suboptimum soft-decision decoding which will be discussed in detail.

Dedicated to my loving parents, my sister and my fiancée

ACKNOWLEDGMENTS

First and foremost, I would like to thank my supervisor, Dr. M.R. Soleymani. Not only did he give me the opportunity to undertake this study, but he was always available to help me with my research. His patience and never-ending supply of ideas allowed me to pursue and complete this work within the Electrical Engineering Department at Concordia University.

I must also extend my appreciation to all the professors which I have had throughout both my undergraduate and graduate studies at Concordia University.

I would also like to thank all my fellow researchers who were kind enough to help me whenever I needed their support. Specifically, I would like to thank Claudio Discepola and Vahid Mohammad Giahi for keeping me motivated and always being available to give me advice.

Lastly, I must thank my family and my fiancée, Rosalind Yee, for continuously supporting me through these times and always believing in my abilities. Without their love and support, I would have never been able to accomplish this task.

Contents

List of Figures

List of Tables

LIST OF ACRONYMS

ARQ	Automatic Repeat Request
AWGN	Additive White Gaussian Noise
BCH	Bose-Chaudhuri-Hocquenghem
BER	Bit Error Rate
BPSK	Binary Phase Shift Keying
BTC	Block Turbo Code
FEC	Forward Error Correction
FER	Frame Error Rate
GMD	Generalized Minimum Distance
GCD	Greatest Common Divisor
HDD	Hard-Decision Decoding
LDPC	Low Density Parity Check

LFSR	Linear Feedback Shift Register
LLR	Log-Likelihood Ratio
MAP	Maximum Aposteriori Probability
MDS	Maximum Distance Separable
ML	Maximum Likelihood
MLD	Maximum Likelihood Decoding
QPSK	Quadrature Phase Shift Keying
RS	Reed-Solomon
SDD	Soft-Decision Decoding
SISO	Soft Input Soft Output
SNR	Signal to Noise Ratio
SOVA	Soft Output Viterbi Algorithm
VA	Viterbi Algorithm
VBA	Vardy Be'ery Algorithm

Chapter 1

Introduction

The need to communicate with others has always existed. Different civilizations found different ways to carry out this task. With pre-historic man, it was the outstretched skin of an animal over a wood frame that was used as a drum. This sound could be heard over long distances and served as warnings of danger or other messages. In the Middle-East, sheep horns were being used as horns. In the Far East, the gong was being used. The American Indian used smoke signals and drums. In 1837, Samuel Morse developed the telegraph and what is now known as the Morse Code. Communication lines for this telegraph were being installed all over the United States and Western Civilization was introduced to its first long distance rapid communication system. Nowadays, communication technology is a fast growing market in the world. This technology affects all aspects of life including business, education and leisure. The word "telecommunication" is defined as the branch of electrical engineering concerned with the technology of electronic communication over distance [1].

It provides users services ranging from voice telephone calls, Internet access, cable television to video-conferencing and satellite communications. Due to its widespread use, researchers are now presented with the task of continuously innovating and finding better solutions than existing technologies. With these new solutions, emphasis remains on speed, reliability and, of course, cost-effectiveness which will present users with error-free transmission of all kinds of data in any type of communication system.

1.1 Composition of a Standard Digital Communication System

In a standard communication system, problems arise from the introduction of noise and interference occurring between an information source and its destination. Noise is the unwanted electrical or electromagnetic energy that degrades the quality of the information signal. Engineers constantly strive to reduce the effects of noise in order to minimize errors and render the transmission of information more reliable. The field of error control coding is aimed at devising efficient encoding and decoding procedures to achieve reliable transmission of information across channels which may be corrupted by noise. One such method uses redundancy to encode the information at the source allowing the destination to reliably decode the received message. Practical applications of error control coding include satellite and deep space communications (e.g., Voyager, Pioneer 9, U.S Army Satellite Communication Agency) [2], storage and wireless communications. Figure 1.1 illustrates the layout of a standard digital communication system.

The *information source* is the location from which the message to be sent originates. This message, depending on the type of source, can be either digital or analog.

The *source encoder* takes the message to be sent and transforms it into a sequence of bits. This is the information sequence. If the source was continuous, then an analog to digital converter would be necessary.

The *channel encoder* transforms the information sequence into an encoded sequence, called the *codeword.* It is at this level that redundancy is added in order to minimize transmission errors.

The *modulator* maps the codeword into a series of analog waveforms which is suitable for transmission across the *physical channel* (e.g., wireless channels, fibre-optic channels, satellite links). This channel is vulnerable to various types of noise (e.g. internal, external, thermal, distortion, interference) which can cause decoding errors. The *demodulator* takes the received waveform, which is now corrupted by noise, and produces an estimate of the transmitted codeword.

The *channel decoder* inspects the estimate of the transmitted codeword. The strategy which was used to first encode the codeword is now used in the decoding process. Actually, the physical separation between the demodulator and the decoder is conceptual. That is, in modern systems, these two steps are combined, i.e., soft-decision decoding. The purpose of this process is to obtain an information sequence that replicates the transmitted sequence as closely as possible, i.e., with the fewest number of errors.

The *source decoder* uses the estimate of the information sequence to obtain an estimate of the source output. This message is then delivered to the *destination.* If

the destination is continuous, then a digital to analog converter would be necessary. In an ideal system, the decoder output would be an exact replica of the source [3].

1.2 Error Correcting Codes and the Channel Coding Technique

In general, there are two methods used in error control coding in order to minimize the number of errors caused by a noisy channel, namely, Forward Error Correction (FEC) and Automatic Repeat Request (ARQ). Due to limited amounts of power and bandwidth in a transmission system, the choice of the above scheme becomes very important.

In ARQ, the receiver or destination has the ability to detect errors which have occurred during transmission, and to request that the sender or source retransmits the erroneous messages. Usually, the receiver will ask for a retransmission of the message until it is correctly received or until a predefined number of retransmissions has occurred. This method causes the system to be very efficient since the overhead of error detection is much less than the overhead required for error correction. With respect to real-time applications, ARQ is not favorable because the retransmission of information will delay the system and render it "non real-time". Until the source has had confirmation that the receiver has successfully received the information (an acknowledgment) which has just been sent, it will stall on sending the next information message. This results in delays which are too time-consuming for todays' applications. ARQ protocols include Stop-and-Wait, Selective Repeat, and Go-Back-N.

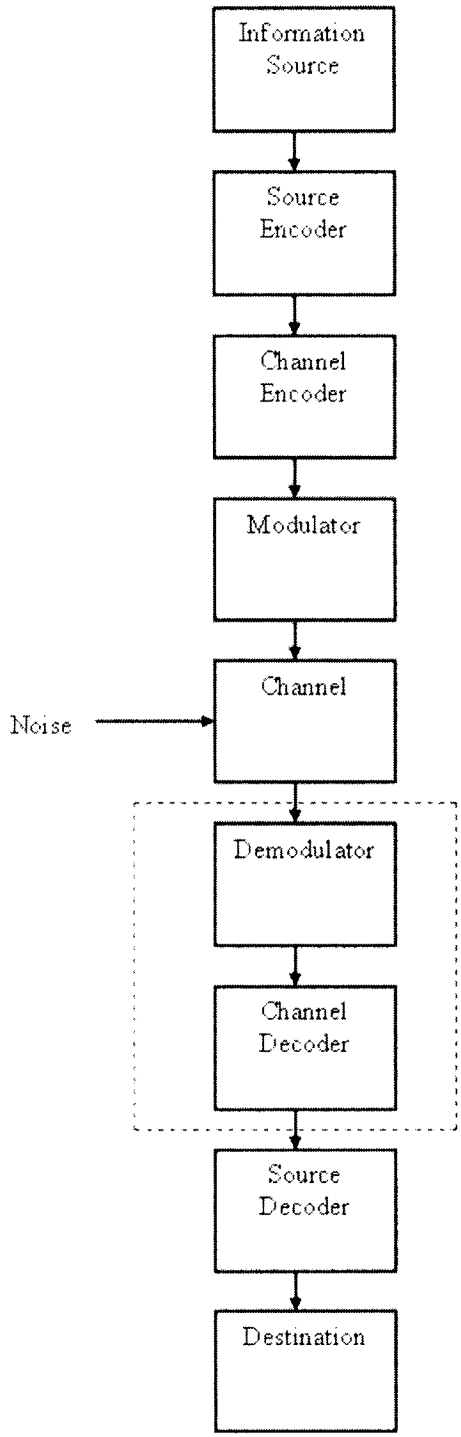

Figure 1.1: Composition of a Standard Digital Communication System

In FEC, errors which occur due to the noisy channel are automatically detected and corrected at the receiver. This is accomplished by the addition of redundant bits to the transmitted information using a variety of predetermined algorithms. Evidently, this scheme enables its users to accommodate any time-constrained application. However, this method also uses additional bandwidth since more bits are added to the transmitted information before going out over the channel. With this in mind, engineers constantly strive to find a medium through which reliability of the received data is maintained while limiting, as much as possible, the amount of required bandwidth. The two main categories of FECs are block codes and convolutional codes.

With respect to the performance of error correcting codes, the main parameters are usually the Bit Error Rate (BER) and the Signal to Noise Ratio (SNR). As its name indicates, the BER is the ratio between the number of incorrect bits transmitted to the total number of bits while the SNR is the ratio of the magnitude of the signal to the magnitude of the noise in the system. The SNR is often expressed in decibels (dB). The increase in efficiency that a coded signal provides over an uncoded signal (due to redundancy) is termed the coding gain, also expressed in dB.

In October 1948, a mathematician named Claude E. Shannon laid the foundation of modern information theory [4] when he stipulated the capacity of the Additive White Gaussian Noise (AWGN) channel as follows:

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

where W is the channel bandwidth in Hertz (Hz), P is the average transmitted power, and N is the variance of the white Gaussian noise (i.e., P/N is the signal-to-noise ratio). Although this theory was developed over 50 years ago, an eternity when it comes to modern day science, they still remain as relevant today as the day it was formulated. Today, the performance of all error correcting codes are compared with the Shannon Limit (calculated using Equation (1.1)). One thing that should be noted is that Shannon's theory was not specific in that it did not point to codes which did attain this limit. Since then, researchers have been striving to discover codes which could satisfy this limit. Shannon was also aware that for long codes, his theory would not be practical due to complexity issues. Along with his colleagues P. Elias, R.G. Gallager, R.W. Hamming, and J.M. Wozencraft, to name a few, Shannon set out to solve this problem. They observed that the solution did not lie in finding good long codes but rather, in designing codes with a low decoding complexity. Several techniques, influenced by these researchers, are briefly described as follows.

The first coding technique developed is block coding. In this scheme, the encoder accepts a message of k information symbols and produces a codeword of n coded symbols. Based on predetermined encoding rules, the n-k redundant symbols are added to the k information symbols to produce the codeword which is to be transmitted over the channel. This is referred to as an *(n,k)* block code. Hamming [5] devised the first set of error correcting block codes principally used for digital communications and data storage systems. Research in this area led to the discovery of Reed-Muller codes, in which a varying amount of errors in the same information sequence could

be corrected. This, in turn, paved the way for the Bose-Chaudhuri-Hocquenghem (BCH) code and the most important set of block codes, the Reed-Solomon (RS) code. A detailed study of these codes will be undertaken in Chapter 2. Gallager worked on Low Density Parity Check (LDPC) codes along with an iterative probability-based decoding algorithm in the early 1960's, although his studies went unnoticed until the discovery of the turbo code [6].

In 1954, Elias [7] introduced the convolutional code as an alternative to block coding. In the case of convolutional codes, the source encoder now contained memory. Hence, the n encoder output bits does not solely depend on the k input bits, but also on the previous m input bits. Wozencraft [8] thought of using Sequential decoding as a decoding algorithm for long convolutional codes. Soon after came the Viterbi Algorithm [9] (VA) which was found to be a solution for both maximum likelihood (ML) and maximum *a posteriori* probability (MAP) decoding for convolutional codes.

It was not until 1993 when a group of French Researchers, headed by Berrou [6], presented an FEC that finally came within a few tenths of a dB of the infamous Shannon limit. They called this FEC scheme "Turbo Coding". This scheme contained a combination of two (or more) recursive convolutional codes, an interleaver, and a MAP iterative decoding algorithm. This iterative decoding process enabled improved error correction by allowing the soft output from the decoders to be fed back and used as the input to the subsequent decoders. Therefore, as the number of iterations increases, the quality of the data being passed to the next decoders contain fewer errors in order to improve the overall quality of the process. One must be careful, though, not to allow too many iterations to occur since this is time-consuming and

irrelevant as the decoding converges at a certain number of iterations. It is important to note that instead of using convolutional codes, one could use block codes. Chapter 3 will further study these types of codes.

1.3 Thesis Contribution

Although it has already been stated that turbo codes provide superior performance compared to other error correcting codes, it is also vital to state that the complexity involved in decoding these types of error correcting codes is elevated. Our main objective is to improve the performance of the codes while either improving the complexity or not rendering the complexity an insurmountable barrier. I would kindly ask the reader to be patient while the details of these issues are pursued in the coming chapters. In this thesis, we study the effect of combining an ML algorithm for RS codes in conjunction with an iterative decoding algorithm for RS block turbo codes. This hybrid scheme, which will be the proposed algorithm, aims to improve the already near-Shannon limit performance while not rendering the system overly complex. In addition, this hybrid scheme will use a suboptimal soft-decision decoding algorithm, which does not cause too much performance degradation while highly improving the complexity vis-à-vis the already existing ML SDD algorithms.

1.4 Outline of the Thesis

Being that the main goal is to further improve the BER performance by modifying existing algorithms, the rest of this report is organized as follows. Chapter 2

will be dedicated to an in-depth study of FEC and RS codes. This will begin with a description of both linear and non-linear block codes. Then, the properties of both the BCH code and, more importantly, the RS code will be discussed. We will also review the existing decoding methods which are mainly hard-decision decoding (HDD) algorithms. This will include the Berlekamp Iterative algorithm and the Euclidean algorithm. The soft-decision decoding (SDD) algorithm on which the proposed algorithm is based will be introduced. But first, the Vardy-Be'ery Algorithm will be presented which inspired the ML algorithm used.

In Chapter 3, the second element of the hybrid scheme needs to be analyzed. This chapter will focus on product codes, both serial and parallel concatenated. This will begin with a brief example to explain convolutional codes. Then, the most important decoding processes used for these codes will be reviewed in detail, including turbo decoding and iterative decoding. This will be followed by a look at the iterative decoding algorithm, based on Pyndiah's ideas[36], which we used as part of the proposed algorithm.

In Chapter 4, a new iterative soft-decision decoding algorithm based on facts presented in Chapter 2 and 3 will be discussed. A method for reducing the complexity in decoding the codewords will be introduced. Also, a method for determining the competing codewords, differing from the currently used Chase Algorithm, is discussed. Our simulation results will then follow, thereby illustrating the improvement in performance while not increasing much of the complexity.

Lastly, Chapter 5 will revisit the main ideas discussed throughout the thesis and summarize the proposed algorithm and its potential significance in terms of

applications. The potential for future work and possible further research are also discussed.

Chapter 2

Reed-Solomon Codes

Error Control Coding is the method of adding redundancy to information in order to allow the receiver to both check and correct the received sequence, which has been corrupted by noise in the channel. In this thesis, it will be assumed that the noise injected by the channel is Additive White Gaussian Noise (AWGN). RS codes are the most widely used block codes. Some of their uses are storage devices (compact disc, DVD, etc), wireless/mobile communications (cellular, microwave, etc) as well as satellite communications. Interestingly enough, RS codes are so useful that they are also used in conjunction with convolutional codes in turbo coded systems. In this chapter, we study the basics of linear block codes. This will commence with a look at field arithmetic and the properties of RS codes. Then, a few HDD algorithms for RS codes will be overviewed. Finally, an in-depth look at a SDD algorithm for these codes will be reviewed.

Addition	Multiplication
$a+b=b+a$	$ab=ba$
$(a+b)+c=a+(b+c)$	$(ab)c=a(bc)$
$a(b+c)=ab+ac$	$(a+b)c=ac+bc$
$a+0=a=0+a$	$a\cdot 1=a=1\cdot a$
$a+(-a)=0=(-a)+a$	$aa^{-1}=1=a^{-1}a$ if $a\neq 0$

Table 2.1: Field Axioms

2.1 Introduction

2.1.1 Finite Fields: The Galois Field

A field is any set of numbers with two operations that satisfy the properties shown in Table 2.1.

If the number of elements in a set is finite, this field is called a Galois field (GF) [10]. The number of elements contained in a field is either a prime number or a power of a prime. The binary field, noted GF(2), is an important field in the study of channel coding and its use is widespread in the area of digital transmission. This can be explained by the fact that information is very often coded in a binary form. GF(2) is described as:

$$GF(2)=\{0,1\}$$

and arithmetic in the field follows modulo-2 addition and multiplication. Both encoding and decoding centers around these types of fields. In our discussions, fields and their associated arithmetic will be used extensively [10].

2.1.2 Linear Block Codes

Although not the only type of block code, the linear block codes are definitely the most popular. These block codes are expressed in terms of generator and parity-check matrices. Channel coding is the procedure of mapping a length-k message word into a length-n codeword, where k < n. The rate of the code, expressed by the ratio k/n, determines the amount of redundancy present in a code. If we have a message of length k, $m = (m_0, m_1, ..., m_{k-1})$, it is then transformed, through encoding, into an n-tuple codeword $c = (c_0, c_1, ..., c_{n-1})$. Thus, if we began with a linear binary block code, belonging to GF(2), with k bits, there is a total of 2^k distinct messages which would produce 2^k codewords. This set of 2^k codewords is called a block code and belongs to the *code space*. For a block code to be linear, the sum of any two codewords must also be a codeword.

An (n, k) linear block code, C, is defined by a generator matrix G with dimensions $(k \times n)$. The rows of this matrix generate the (n, k) code C.

$$G = \begin{bmatrix} g_0 \\ g_1 \\ \\ \\ \\ g_{k-1} \end{bmatrix} = \begin{bmatrix} g_{00} & g_{01} & g_{02} & \cdots & g_{0,n-1} \\ g_{10} & g_{11} & g_{12} & \cdots & g_{1,n-1} \\ & \cdot & \cdot & & \cdot \\ & \cdot & \cdot & & \cdot \\ & \cdot & \cdot & & \cdot \\ g_{k-1,0} & g_{k-1,1} & g_{k-1,2} & \cdots & g_{k-1,n-1} \end{bmatrix} \tag{2.1}$$

where $g_i = (g_{i0}, g_{i1}, g_{i2}, ..., g_{i,n-1})$, for $0 \leq i \leq k$. If $m = (m_0, m_1, ..., m_{k-1})$ is to be encoded, the corresponding codeword can be written as follows:

$$\begin{aligned} c &= mG = \begin{bmatrix} m_0, & m_1, & ..., & m_{k-1} \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ \\ \\ \\ g_{k-1} \end{bmatrix} \\ &= m_0 g_0 + m_1 g_1 + ... + m_{k-1} g_{k-1} \end{aligned} \tag{2.2}$$

Based on Equation (2.2), it is easy to see that the (n, k) linear block code is completely specified by the rows of the generator matrix (see Equation (2.1)). The other way to describe a code is by using the parity-check matrix, H. As its name indicates, this matrix is used to verify the validity of a codeword. Assuming we have an (n, k) linear block code C, the codeword c is valid if and only if $cH^T = 0$, where H^T is the transpose of the H matrix. The H matrix is a $(n - k) \times n$ matrix.

Often, it is desirable for a linear block code to be expressed in its *systematic form.* When in this form, a codeword is divided into two parts. One being the original message (length-k) sequence and the other being the redundant information (length $n - k$) added on by the encoder. This structure makes the encoding and decoding much simpler. In this case, the encoder generates $n - k$ parity symbols and

appends it to the message. In turn, the decoder must just strip the $n-k$ parity symbols to obtain the message. Now, both the generator and parity-check matrices take on a structured form (Equations (2.3) & (2.4)).

$$G = [P|I_k] \tag{2.3}$$

$$H = \left[I_{n-k}|P^T\right] \tag{2.4}$$

where I_k is a $k \times k$ identity matrix, P is a $k \times (n-k)$ matrix, and I_{n-k} is a $(n-k) \times (n-k)$ identity matrix [11].

It is essential to introduce the reader to the terminology: *minimum distance*, d_{min}. This distance is equal to the Hamming weight of the lowest-weight non-zero codeword, where the Hamming weight is defined as the number of non-zero elements in the codeword. This can also be viewed as the distance of the codeword to the all-zero codeword. This parameter is of significant importance because it determines the error correcting and detecting capability of a code. For example, a linear block code having a minimum distance d_{min} guarantees detection of all error patterns containing up to $d_{min}-1$ errors and has the capacity to correct up to $t = \left\lfloor \frac{d_{min}-1}{2} \right\rfloor$errors.

An important subclass of linear codes is that of cyclic codes. As its name indicates, if you take a codeword belonging to the code space and cyclically shift one place to the right, we have another codeword belonging to the code space. One of the mostly

used cyclic codes is the BCH code. BCH codes are either binary or non-binary. Of this latter, the RS code is the most important and was introduced by Reed and Solomon in 1960 [12].

2.2 Properties of RS Codes

We have discussed codes belonging to GF(2); these are called binary codes. There also exist codes with symbols from the Galois Field $GF(q)$. These are called q-ary codes, in which q is a prime or a power of a prime. In RS codes, a symbol is represented by m bits. An (n, k) RS code has $n = 2^m - 1$ code symbols of m-bits and k message symbols of m-bits. There are $n - k$ parity symbols of m bits each. Here are a few parameters:

$$\text{Block length: } n = q - 1$$

$$\text{Error-correction capability: } t = \frac{n-k}{2}$$

$$\text{Number of parity-check symbols: } n - k = 2t$$

$$\text{Minimum distance: } d_{min} = 2t + 1 = n - k + 1$$

For a linear (n, k, d) code over $GF(q)$, the Singleton bound states that $d \leq n - k + 1$. Because RS codes meet this bound with equality, they are called Maximum Distance Separable (MDS) codes. Instead of being generated by a generator matrix, RS codes (all cyclic codes) use a generator polynomial. The generator polynomial of an RS code is:

$$g(x) = \prod_{i=0}^{2t}(x + \alpha^i) = (x + \alpha)(x + \alpha^2)\cdots(x + \alpha^{2t}) \tag{2.5}$$

where α is a primitive element in $GF(2^m)$, meaning that it can generate all the non-zero elements belonging to this field. The codeword is constructed using:

$$c(x) \;=\; m(x)g(x) \tag{2.6}$$

where the message sequence $m(x) = m_0 + m_1 + \cdots + m_{k-1}$. All valid codewords are divisible by the generator polynomial. The parity polynomial, which has degree $2t - 1$, is the remainder of $\frac{x^{2t} \times m(x)}{g(x)}$. It is easy to see that the encoder of RS codes need only contain very simple feedback circuitry (see Figure 2.1).

After all k information symbols (stored in $m(x)$) have made their way through this circuit, the contents of the $2t$ shift registers will contain the parity symbols. The codeword is formed by appending these $2t$ parity symbols to the k information symbols. The properties of RS codes make them suitable for applications where burst errors occur. It does not matter how many bits of a symbol are incorrect because if one or all bits in a symbol are erroneous, it only counts as a single erroneous symbol. Therefore, if the channel is known to produce burst errors, using an RS code would be a good choice. Now that we have been introduced to the encoding of RS codes, it is important to become familiar with its decoding.

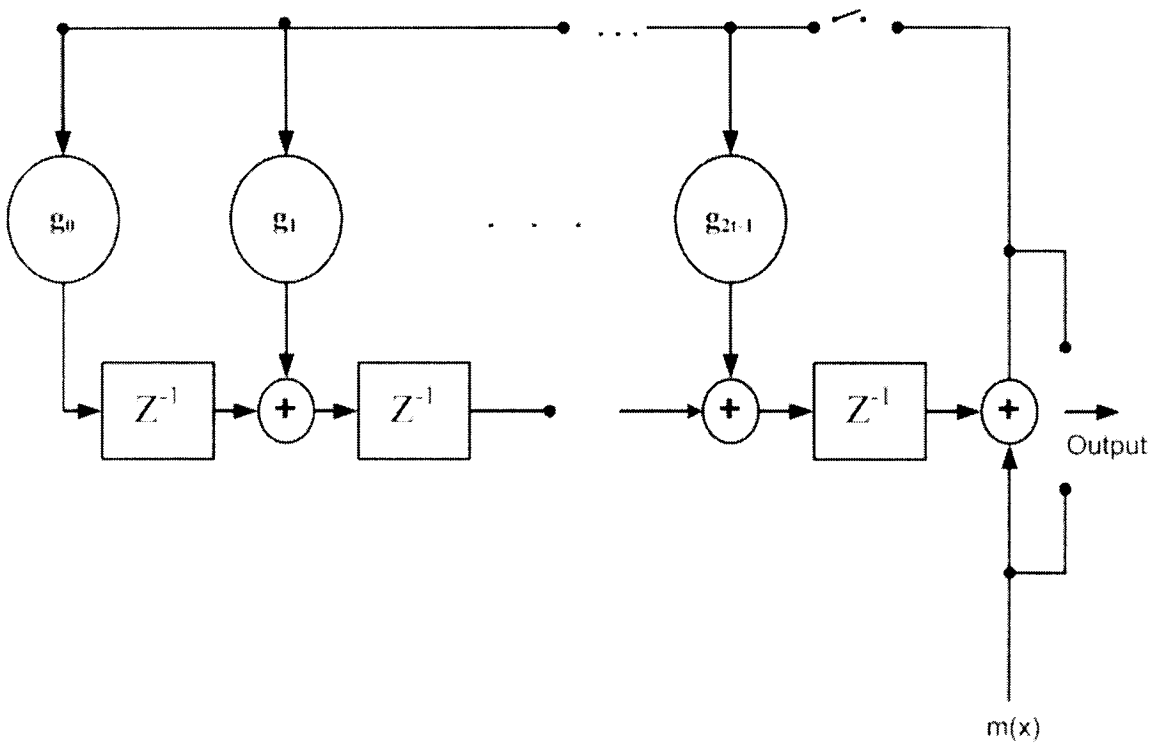

Figure 2.1: RS Encoder Circuit

2.3 Several Hard-Decision Decoding Algorithms

After passing through the channel, the sent codeword may have now been affected by noise. The received vector at the channel decoder now takes on the following form:

$$r(x) = c(x) + e(x) \tag{2.7}$$

where $c(x)$ is the transmitted codeword and $e(x)$ is the polynomial representing errors which have occurred. The RS decoder has the ability to identify the position and magnitude of up to t errors and correct them. Any RS code has $2t$ syndromes that depend solely on errors. If the roots of the generator polynomial are substituted into $r(x)$, the outcome is the syndrome. The syndrome calculation is performed on the received vector to verify whether it is a member of the codeword set or not. If the received vector is, in fact, a valid codeword, then the resultant syndrome would be the all-zero vector. Otherwise, errors are definitely present.

The location of symbol errors in an erroneous codeword is found by solving a set of simultaneous equations with t unknowns. There are two main steps in determining the location of the errors. Firstly, one must find the error location polynomial. The roots of this polynomial provide an indication of where the errors are located. Two well-known algorithms for finding this polynomial are the Berlekamp Iterative algorithm (which can be modified into the Berlekamp-Massey algorithm) and the Euclidean algorithm. The Berlekamp-Massey algorithm finds the shortest linear recurrence that will produce the errors. While this algorithm usually leads to more

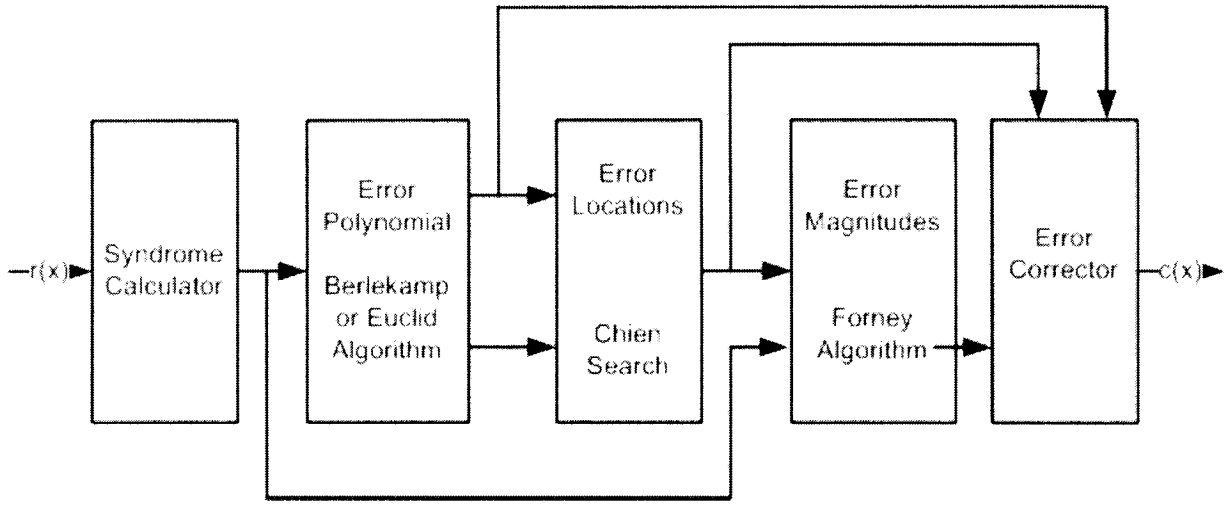

Figure 2.2: RS Decoder Architecture

efficient hardware and software implementations, it is the Euclidean algorithm which is more widely used in practice because of the fact that it is easier to implement. Independently of which algorithm is used, the roots of the error location polynomial are usually determined using the Chien Search[13]. Thus, once the error locations are determined, the correct symbol is given using the Forney algorithm by solving the equation with t unknowns. In summary, each error value and its corresponding location need to be established in order to successfully decode RS codes. The HDD for RS codes can be seen in Figure 2.2.

2.3.1 Berlekamp Iterative Algorithm

The steps of the Berlekamp Iterative algorithm will be enumerated before demonstrating it with an example.

1. Compute the syndromes $(0, \cdots, 2t)$ for the received codeword.

2. Set the variables: $\mu = -1$, $\sigma^{\mu}(x) = 1$, $d^{\mu} = 0$, $l^{\mu} = 0$

3. Set $\mu = \mu + 1$. Compute the discrepancy using:

$$d^{\mu} = S_{\mu+1} + \sigma^{(\mu)}(x)S_{\mu} + \sigma_2^{(\mu)}S_{\mu-1} + \cdots + \sigma_{l_{\mu}}^{(\mu)}S_{\mu+1-l_{\mu}} \tag{2.8}$$

4. If $d_{\mu} = 0$, then set $\sigma^{(\mu+1)}(x) = \sigma^{(\mu)}(x)$ and $l_{\mu+1} = l_{\mu}$

5. If $d_{\mu} \neq 0$, go back to the step, called ρ, where $d_{\rho} \neq 0$ and $\rho - l_{\rho}$ has the largest value and

$$\sigma^{(\mu+1)}(x) = \sigma^{(\mu)}(x) + d_{\mu}d_{\rho}^{-1}x^{(\mu-\rho)}\sigma^{\{\rho)}(x) \tag{2.9}$$

6. When $\mu + 1 = 2t$ and $\sigma^{(2t)}(x)$ is found, the error locator polynomial is found.

An example will now be presented to clarify the procedure [14]. Assume that we have a triple-error correcting RS code (i.e., $t = 3$) with symbols from the finite field $GF(2^4) = GF(16)$. As per our previous discussions (see Equation (2.5)) , the generator polynomial of such a code would be:

$$\begin{aligned} g(x) &= \prod_{i=0}^{2\times 3}(x + \alpha^i) \\ &= \alpha^6 + \alpha^9 x + \alpha^6 x^2 + \alpha^4 x^3 + \alpha^{14} x^4 + \alpha^{10} x^5 + x^6 \end{aligned} \tag{2.10}$$

μ	$\sigma^{(\mu)}(x)$	d_μ	l_μ	$\mu - l_\mu$
-1	1	1	0	-1
0	1	α^{12}	0	0
1	$1 + \alpha^{12}x$	α^7	1	0take $\rho = -1$
2	$1 + \alpha^3 x$	1	1	1
3	$1 + \alpha^3 x + \alpha^3 x^2$	α^7	2	1 take $\rho = 0$
4	$1 + \alpha^4 x + \alpha^{12} x^2$	α^{10}	2	2
5	$1 + \alpha^7 x + \alpha^4 x^2 + \alpha^6 x^3$	0	3	2 take $\rho = 2$
6	$1 + \alpha^7 x + \alpha^4 x^2 + \alpha^6 x^3$	$-$	$-$	$-$

Table 2.2: Example for determining Error Location Polynomial

Assume that we have transmitted the all-zero codeword, $c = (\mathbf{0000000000000000})$ and have received the vector $r = (\mathbf{000}\alpha^{\mathbf{7}}\mathbf{00}\alpha^{\mathbf{3}}\mathbf{00000}\alpha^{\mathbf{4}}\mathbf{00})$. The received codeword polynomial is thus $r(x) = \alpha^7 x^3 + \alpha^3 x^6 + \alpha^4 x^{12}$. The first step is to determine the syndromes for the received vector, as follows.

$$S_1 = r(\alpha) = \alpha^{10} + \alpha^9 + \alpha = \alpha^{12}$$

$$S_2 = r(\alpha^2) = \alpha^{13} + 1 + \alpha^{13} = 1$$

$$S_3 = r(\alpha^3) = \alpha + \alpha^6 + \alpha^{10} = \alpha^{14}$$

$$S_4 = r(\alpha^4) = \alpha^4 + \alpha^{12} + \alpha^7 = \alpha^{10}$$

$$S_5 = r(\alpha^5) = \alpha^7 + \alpha^3 + \alpha^4 = 0$$

$$S_6 = r(\alpha^6) = \alpha^{10} + \alpha^9 + \alpha = \alpha^{12}$$

Secondly, the error-location polynomial, $\sigma(x)$, must be found. To demonstrate this step, it is easier to use a table (see Table 2.2).

From the table, it can be seen that $\sigma^{(2t)}(x) = 1 + \alpha^7 x + \alpha^4 x^2 + \alpha^6 x^3$. This was our main goal. Berlekamp's iterative algorithm enabled us to determine the error-location polynomial. After certain calculations, the roots of $\sigma(x)$ can be found to

be $\alpha^3, \alpha^9, \alpha^{12}$. The error location numbers of the error pattern are the reciprocals of these roots which give $\alpha^3, \alpha^6, \alpha^{12}$. This means that the errors occur at positions x^3, x^6 and x^{12}. We can then determine the error pattern to be $e(x) = \alpha^7 x^3 + \alpha^3 x^6 + \alpha^4 x^{12}$. It can be seen that this error pattern is the exact difference between the transmitted vector c and the received vector r. To complete the decoding process, one has to just compute the following:

$$\widehat{c}(x) = r(x) - e(x) \tag{2.11}$$

where $\widehat{c}(x)$ is the estimate of the transmitted polynomial after decoding has been completed. In this case, the decoded polynomial is the all-zero polynomial which identically matches the transmitted polynomial. Although the Berlekamp Iterative algorithm was initially created in order to decode BCH codes, it is also used in the decoding of RS codes.

Another algorithm which is a simplified version of the above algorithm is the Berlekamp-Massey algorithm. Massey simplified the algorithm by demonstrating that finding the error-locator polynomial can be equivalent to a shift register synthesis problem. The syndromes are first determined through hardware with Massey's linear feedback shift register (LFSR) [15]. The syndrome equations can be likened to a filter with the taps given by the coefficients of $\sigma(x)$ These taps are not known initially but are determined iteratively such that the smallest LFSR generates the $2t$ syndromes. The smallest LFSR will then guarantee the error locator polynomial hav-

ing the smallest degree. A complete description of the Berlekamp-Massey algorithm can be found in [16, 17, 18].

2.3.2 Euclidean Algorithm

Before delving into how this algorithm is applied to RS codes, it is important to understand the basics of the algorithm. Euclid's algorithm is used to find the greatest common divisor (gcd) of two polynomials. This can be achieved through matrix recursion. Assume that we would like to find the gcd of $a(x)$ and $b(x)$, where $Deg(a(x)) \geq Deg(b(x))$ Let $a^{(0)}(x) = a(x)$ and $b^{(0)}(x) = b(x)$. The first step of the Euclid algorithm allows us to express $a^{(0)}(x)$ as

$$a^{(0)}(x) = Q^{(0)}(x)b^{(0)}(x) + b^{(1)}(x) \tag{2.12}$$

where $Q^{(0)}(x)$ is the quotient and $b^{(1)}(x)$ is the remainder. From (2.12), it is easy to see that $b^{(1)}(x) = a^{(0)}(x) - Q^{(0)}(x)b^{(0)}(x)$ The $gcd(a^{(0)}(x), b^{(0)}(x))$ divides $b^{(1)}(x)$. If we let $a^{(1)}(x) = b^{(0)}(x)$, we can write:

$$\begin{bmatrix} a^{(1)}(x) \\ b^{(1)}(x) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -Q^{(0)}(x) \end{bmatrix} \begin{bmatrix} a^{(0)}(x) \\ b^{(0)}(x) \end{bmatrix} \tag{2.13}$$

This process can be repeated as long as the remainder $b^{(1)}(x)$ is non zero. This recursion can be expressed as:

$$Q^{(r)}(x) = \left\lfloor \frac{a^{(r)}(x)}{b^{(r)}(x)} \right\rfloor \tag{2.14}$$

where r is the recursion step. This is done until the remainder is zero.

Now, for the purpose of RS codes, here is a list of the steps to perform this algorithm [19]:

1. Compute the Syndrome Polynomial, called $S(x)$.

2. Initialize $s^{(0)}(x) = x^{2t}$, $\sigma^{(0)}(x) = S(x)$, and $B^{(0)} = I_2$, where I_2 is a 2×2 identity matrix

3. Solve the following recursive formulas until $deg(\sigma^{(r')}(x)) < t - 1$

$$Q^{(r)}(x) = \left\lfloor \frac{s^{(r)}(x)}{\sigma^{(r)}(x)} \right\rfloor \tag{2.15}$$

$$B^{(r+1)} = \begin{bmatrix} 0 & 1 \\ 1 & -Q^{(r)}(x) \end{bmatrix} B^{(r)} \tag{2.16}$$

$$\begin{bmatrix} s^{(r+1)}(x) \\ \sigma^{(r+1)}(x) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -Q^{(r)}(x) \end{bmatrix} \begin{bmatrix} s^{(r)}(x) \\ \sigma^{(r)}(x) \end{bmatrix} \tag{2.17}$$

4. Then, let $\Delta = B_{22}^{(r')}|_{x=0}$. The solutions are then:

$$e(x) = \Delta^{-1}\sigma^{(r')}(x) \tag{2.18}$$

$$\sigma(x) = \Delta^{-1}B_{22}^{(r')} \tag{2.19}$$

The first thing that should be noticed is that the HDD algorithms for RS codes that have been mentioned are very algebraic in nature. Another important thing to realize is that while these algorithms have been extensively used, they are becoming less attractive because of the ever-growing need to have more reliable results. Because these decoders make hard-decisions, they are less favorable to newer innovations which have come in the form of soft-decision decoders. While SDD does produce better results as opposed to its counterpart (HDD), they have a downside. They are much more involved and hence complexity is increased. For proof of the superior performance of SDD over HDD, the reader is referred to [11], [20] and [21].

2.4 Soft-Decision Decoding of RS Codes

As already discussed, SDD has a very high complexity in comparison to HDD algorithms. One of the first attempts at using soft information in order to improve the decoding performance was made in 1966 by Forney [24]. In Forney's Generalized Minimum Distance (GMD) decoding method, a series of errors and erasures decoding is performed on the hard-decision until such time that the GMD criterion is satisfied.

This will be discussed in more depth in Section 3.3.1. Another attempt was made in 1972 by Chase [25], where a list of candidate codewords is generated; this is a list-decoding algorithm and will be discussed further in the next chapter.

One important quote states that "the major drawback with RS codes (for satellite use) is that the present generation of decoders do not make full use of bit-based soft-decision information [22]". The first algorithm discussed below will try to address this issue by making use of bit-level soft-decision information. The second algorithm can be considered as an extension to the first in that it attempts to improve the complexity of the first algorithm.

2.4.1 Description of the Vardy-Be'ery MLD Algorithm

The Vardy-Be'ery algorithm (VBA) [23] was developed in order to make use of bit soft-decision information of RS codes. This algorithm was proven to be several orders of magnitude more efficient than any decoding techniques available at the time of discovery. The complexity reduction of this technique can be explained in large part by the symmetric structure of the RS generator matrix over $GF(2)$.

Let us begin with an(n, k) RS code, called $\Re$, over $GF(2^m)$. Assuming that we are dealing with a binary channel, the encoder must convert the sequence of n symbols into mn bits. Thus, we would begin with a codeword $c = (c_0, c_1, \cdots, c_{n-1})$, where $c \in GF(2^m)$ and transmit it as $\phi(c) = (c_0^1, c_0^2, \cdots, c_0^m, c_1^0, c_1^1, \cdots, c_1^m, \cdots\cdots, c_{n-1}^0, c_{n-1}^1, \cdots, c_{n-1}^m)$, where m is the number of bits per symbol. Let us now define the binary BCH code B, having the same zeros as $\Re$. We can now define the codes $B_1, B_2 \cdots, B_m$ as

$$B_j = \{(\gamma_j b_0, \gamma_j b_1, \cdots, \gamma_j b_{n-1}) | b = (b_0, b_1, \cdots, b_{n-1}) \in B\}$$
$$for\, j = 1, 2, \cdots, m \tag{2.20}$$

where $b_i \in GF(2)$ and $\gamma_j b_i \in GF(2^m)$. The m BCH codes defined in Equation (2.20) are subcodes of $\Re$. The resultant binary BCH code has dimensions $k_{BCH} < k$ and $d_{min} \geq n - k + 1$.$\Re$ can be expressed as a union of cosets

$$\Re = \cup_{l=0}^{2^{\Delta}-1} C_l \tag{2.21}$$

such that $\Delta = m(k - k_{BCH})$ and $C_l = \{r^l + c | c \in C\}$, where $C = B_1 \oplus B_2 \oplus \cdots \oplus B_m$ is a direct sum of the m BCH codes found in (2.20), and r^l are cosets of C in $\Re$. In sum, this will allow for the breakdown of an RS code into a binary BCH code and its coset leaders ($c = b + r$) which we can then use to obtain an RS generator matrix (see Figure 2.3). In this figure, the empty spaces (illustrated by a circle) represent all-zero elements.

It is possible to generalize the above for the case where we would breakdown an RS code into several subcodes, where $GF(2) \subset GF(2^{p_1}) \subset GF(2^{p_2}) \subset \cdots \subset GF(2^m)$ considering that $m = p_1 p_2 \cdots p_q$. We would then have the following: $B \subset B^{(1)} \subset B^{(2)} \subset \cdots \subset B^{(q)} = \Re$. This means that we could potentially represent a given code by its constituents subcodes and produce a binary version of the RS generator matrix (as seen in Figure 2.3). This is demonstrated in the upcoming pages by an example.

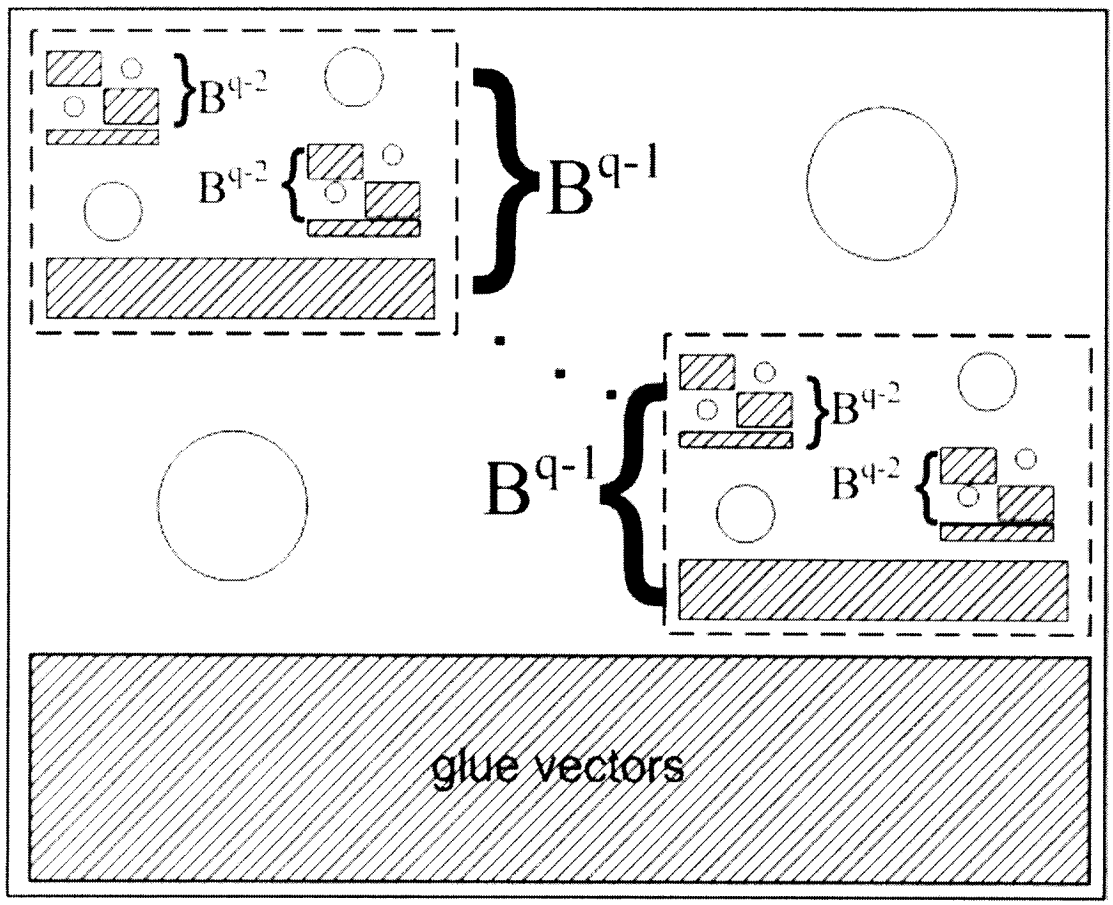

Figure 2.3: Recursive Structure of RS Generator Matrix

Consider the $(15, 11)$ RS code, denoted $\Re(15, 11)$, with elements belonging to $GF(2^4)$ and $d_{min} = 5$ and $t = 2$. According to Equation (2.5), we get $g_{RS}(x) = x^4 + \alpha^{13}x^3 + \alpha^6 x^2 + \alpha^3 x^3 + \alpha^{10}$. The factorization of $m = 4$ yields $4 = 2 \cdot 2$, i.e. $p_1 = 2$ and $q = 2$. So, we have $GF(2) \subset GF(2^2) \subset GF(2^4)$ and $B^{(q)} = \Re(15, 11)$. $B^{(1)}$ is a BCH code with $n = 15$, $d = 5$ over $GF(2^2) \subset GF(2^4)$. This is a $4 - ary$ BCH code. Equation (2.5) can also be rewritten as

$$g(x) = LCM\{\phi_1(x), \phi_2(x), \cdots \phi_{2t}(x)\} \tag{2.22}$$

where $\phi_i(x)$ is the minimal polynomial of α^i. After evaluating the values of the these minimal polynomials, we obtain:

$\phi_1(x) = x^2 + x + \alpha^5$

$\phi_2(x) = x^2 + x + \alpha^{10}$

$\phi_3(x) = x^2 + \alpha^{10}x + 1$

$\phi_4(x) = x^2 + x + \alpha^5$

So, from (2.22), $g(x) = \phi_1(x) \cdot \phi_2(x) \cdot \phi_3(x) = x^6 + \alpha^{10}x^5 + x^4 + x^3 + \alpha^5 x^2 + \alpha^5 x + 1$. This results in the $(15, 9, 5)$ BCH code, $B^{(1)}$. Now, $B^{(0)}$ is a BCH code with $n = 15$, $d = 5$ over $GF(2) \subset GF(2^4)$. Again, using Equation (2.22) and skipping a few steps, we obtain $g(x) = x^8 + x^7 + x^6 + x^4 + 1$. The resultant code is the $(15, 7, 5)$ binary BCH code, $B^{(0)}$. We have just shown how the $(15, 11)$ RS code can be broken down into parts and expressed by a $(15, 7)$ binary BCH code. For additional information about the generator matrix and its exact construction, the reader is

referred to [19] where every component of Figure 2.3 is explained.

With this background information, it is now possible to list the VBA. Note that the steps listed must be performed for each of the 2^{Δ} cosets and each coset leader r_l:

1. Determine the m codewords $\widehat{b}_1, \widehat{b}_2, \cdots, \widehat{b}_m$, where $\widehat{b}_j \in B$ by maximizing

$$M_j(b) = \sum_{i=0}^{n-1} log\, f(v_i^j | b_i + r_i^j) \tag{2.23}$$

   for all b where v is the observed output (received vector at the output of the channel) and r is the coset leader.

2. Then, calculate

$$M(c) = \sum_{j=0}^{m} M_j(\widehat{b}_j) = \sum_{j=1}^{m} \sum_{i=0}^{n-1} log\, f(v_i^j | c_i^j) \tag{2.24}$$

   where c_i^j is the sum of the BCH and coset leader parts.

3. Decoding terminates when we find $\widehat{c} \in \Re$ that maximizes (2.24).

Although this original algorithm is more of a conceptual nature because we assume a memoryless channel, it is easy to slightly modify the algorithm in order to render it more practical. If we now assume independent noise for each transmitted symbol, we can define the binary channel, including memory, by the 2^m probability densities:

$$f(v/\xi) = f(v^1, v^2, \cdots, v^m | \xi^1, \xi^2, \cdots, \xi^m) \tag{2.25}$$

where $\xi \in GF(2^m)$. Then, if we keep step 1 of the above algorithm, only step 2 has to be modified in order to take into account the channel memory.

$$\begin{aligned} M(c) &= \sum_{i=0}^{n-1} log\, f(v_i|c_i) \\ &= \sum_{i=0}^{n-1} log\, f(v_i^1, v_i^2, \cdots, v_i^m | \widehat{b}_i^1 + r_i^1, \widehat{b}_i^2 + r_i^2, \cdots, \widehat{b}_i^m + r_i^m) \end{aligned} \tag{2.26}$$

Decoding is carried out until $\widehat{c} \in \Re$ maximizes Equation (2.26).

Although these were the first attempts for using the bit properties of RS codes in their decoding process, it is obvious that this algorithm is not suitable for longer RS codes. This algorithm, albeit innovative, was just the beginning in maximum-likelihood RS decoding using bit soft-decision information.

2.4.2 Description of the Vucetic-Ponnampalam MLD Algorithm

The algorithm proposed by Vardy and Be'ery [23] was based on the structure of the generator matrix of RS codes. Another MLD algorithm has since been developed by Vucetic and Ponnampalam which takes full advantage of the structural properties of RS codes. It shows that an RS codeword is formed by interleaving a binary BCH

codeword with one of its cosets. Because of the nature of the RS codeword and its ability to be broken down into codes, it is feasible to derive a ML SDD algorithm to exploit this property.

The steps required to decode will be shown and this will be followed by a brief discussion about the differences which lie between this algorithm and the one discussed in Section 2.4.1.

To understand this algorithm, a few variables must first be defined:

$$\delta^2_{min}(y,l) = \min_{b \in C_{BCH}} d^2_E(y, s(b+l)) \tag{2.27}$$

where $d^2_E(a,b)$ is the squared Euclidean distance between a and b, i.e.,

$$d^2_E(a,b) = \sum_{i=1}^{m}(a_i - b_i)^2 \tag{2.28}$$

$$s(x) = \begin{cases} +1 & \text{if } x = 1 \\ -1 & \text{if } x = 0 \end{cases} \tag{2.29}$$

$$\Delta^2(y,l) = \sum_{j=1}^{m} \delta^2_{min}(y^{(j)}, l^{(j)}) \tag{2.30}$$

1. Calculate $\delta^2_{min}(y^{(j)}, l_s)$ for $1 \leq j \leq m$ and $1 \leq s \leq 2^{n-k_{bch}}$, where l_s is a coset

leader belonging to $C_{BCH} + l_s$

2. Find $\widehat{l}$, such that $\widehat{l} \in L$ and $\Delta^2(y, \widehat{l}) \leq \Delta^2(y, l)$

3. Calculate $\widehat{b}$ using

$$\widehat{b}^{(j)} = arg \min_{b \in C_{BCH}} d_E^2(y^{(j)}, s(b + \widehat{l}^{(j)})) \quad for\ 1 \leq j \leq m \tag{2.31}$$

4. The estimated codeword is found by computing: $v = \widehat{l} + \widehat{b}$

The Vucetic-Ponnampalam algorithm [26] differs from the VBA in the way which steps 1 and 2 above are carried out. In VBA, step 1 requires that the distance between each BCH code to its coset to be calculated separately for all $2^{n-k_{BCH}}$ cosets. There is a way to calculate the distance of the received segment, $y^{(j)}$, to each coset of the BCH code simultaneously. This involves the construction of a trellis based on the parity-check matrix (see Equation (2.4)) of the BCH code. Since the construction of the trellis is based on the parity-check matrix, it can be shown that the path corresponding to codewords from the same coset arrive at the same final node. Further discussions on this as well as an example for the $(15, 7, 5)$ binary BCH code (subcode of the $(15, 11, 5)$ RS code) will be presented in Chapter 4. In step 2, a total number of $2^{m(k-k_{BCH})}$ possible coset patterns need to be checked to determine the valid coset pattern. To determine the coset pattern $\widehat{l}$ needed in step 2, Vucetic and Ponnampalam use an algorithm based on partial ordering. They accomplish this by generating and then ordering the coset patterns (which are interleaved combinations

of BCH coset leaders). First, an ordering function Ω is defined in which all coset leaders are numbered: $\Omega : \varepsilon \rightarrow \{0, 1, \cdots, 2^{n-k_{BCH}} - 1\}$ where ε is the set of coset leader terms. Then, each coset pattern λ is assigned a level, denoted $L(\lambda)$, which is calculated as follows:

$$L(\lambda) = \sum_{j=1}^{m} \Omega(\lambda^{(j)}) \tag{2.32}$$

Knowing that $0 \leq \Omega(\lambda^{(j)}) \leq \kappa$, it is evident from Equation (2.32) that $0 \leq L(\lambda) \leq m\kappa$, where $\kappa = 2^{n-k_{BCH}} - 1$. A diagram which illustrates this algorithm is shown in Figure 2.4 [26].

This flow chart represents the optimal way of finding the estimate of the coset pattern, $\widehat{l}$, whereas the method to be described in Chapter 4 will be suboptimal in the sense that the decoding complexity is reduced with only a slight degradation in the performance.

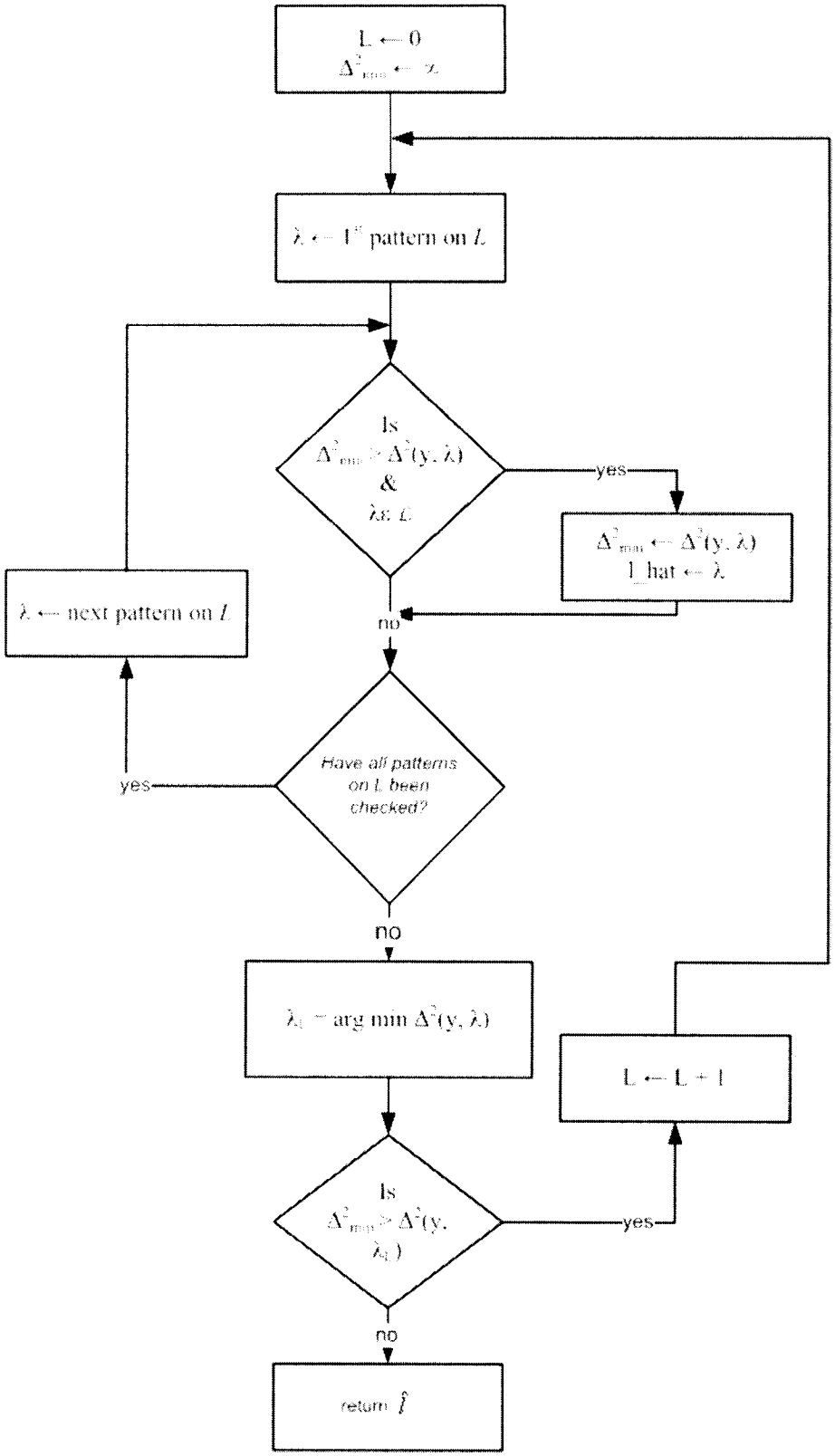

Figure 2.4: Flow Diagram for Vucetic-Ponnampalam Search Algorithm

2.5 Summary

In this chapter, we introduced the basic concepts of linear block codes as well as finite fields. The properties of RS codes were also presented in order to familiarize the reader with the discussions that will follow in the coming chapters. Two widely used hard-decision decoding algorithms were also presented, namely the Berlekamp Iterative Algorithm and the Euclidean Algorithm with the former being illustrated by means of a numerical example. Then, this sets the tone for the SDD algorithms which are the basis for this thesis. The Vardy-Be'ery and the improved version by Vucetic and Ponnampalam were also discussed in detail with the latter to be discussed even further in Chapter 4.

Chapter 3

Product Codes

One of the main concerns in data communications over a noisy channel is how to deal with the errors introduced by the channel. Shannon showed that this problem could be tackled by channel coding. With his ideas in mind, the search for the most powerful codes (i.e., with good error correction capability) began. As it has already been reviewed in the previous chapter, RS codes are the most widely used block codes. The ideas of making these codes even better, by concatenating them, as well as their properties will be reviewed. This will be preceded by a quick look at convolutional codes which can be used as an alternative to block codes. Then, the decoding algorithms used for such codes will be studied.

3.1 Introduction

3.1.1 Convolutional Codes

It has been shown that block codes are of a fixed length. Those codes were described by n, the length of the codeword at the output of the encoder, k, the length of the message at the input to the encoder. The ratio k/n is termed as the code rate which determines the amount of redundancy present. In convolutional codes, there are three integers considered: n, k, and K. The ratio k/n is still the code rate. K is the constraint length which represents the number of stages in the encoding shift registers. Generally speaking, convolutional encoders have kK shift registers and n adders. In Figure 3.1, there are 2 modulo-2 adders and $K = 3$. The rate of this encoder is 1/2. This is evident from the fact that if one bit is input to the system, the output is 2 bits. The generator polynomial for the upper connection is $g_1(x) = 1 + x + x^2$ while the lower one is $g_2(x) = 1 + x^2$. Now, assuming we had an input message of 3 bits, namely $m = 1\,0\,1$. This is equivalent to $m(x) = 1 + x^2$. It is imperative that after this sequence is sent into the shift registers, two more "0" bits must be sent in to "flush" the register. Without this, the 3 message bits would not make their way fully through the shift registers of the encoder. The number of flush bits needed is found to be $K - 1 = 2$ in this case. Now, the first output $u_1(x) = m(x)g_1(x) = 1 + x + x^3 + x^4$ and $u_2(x) = m(x)g_2(x) = 1 + x^4$. By combining these two results, we get the output of the encoder $U(x) = (1,1) + (1,0)x + (0,0)x^2 + (1,0)x^3 + (1,1)x^4$. It is clear now that although we had only 3 message bits, plus two flush bits, at the input of the encoder,

we now have 10 bits at the output.

There is another tool used to represent the encoder of convolutional codes. This is called a state diagram. The number of states depends on the amount of memory of the encoder. In Figure 3.1, there are 4 states. Generally, the number of states is 2^M, where M is the memory of the encoder or, in other words, the $K-1$ stages. When given an input, a transition occurs between the present state and one of two next states, depending on whether a 0 or a 1 was input. A state diagram corresponding to the encoder of Figure3.1 is shown in Figure 3.2. We can see that there are nodes representing the contents of the shift register at a present time (these are called states-usually present state). These nodes are connected by links labeled by m/u. The m label signifies the input bit while the u label represents the output bits (2, in this example) corresponding to each state transition [27].

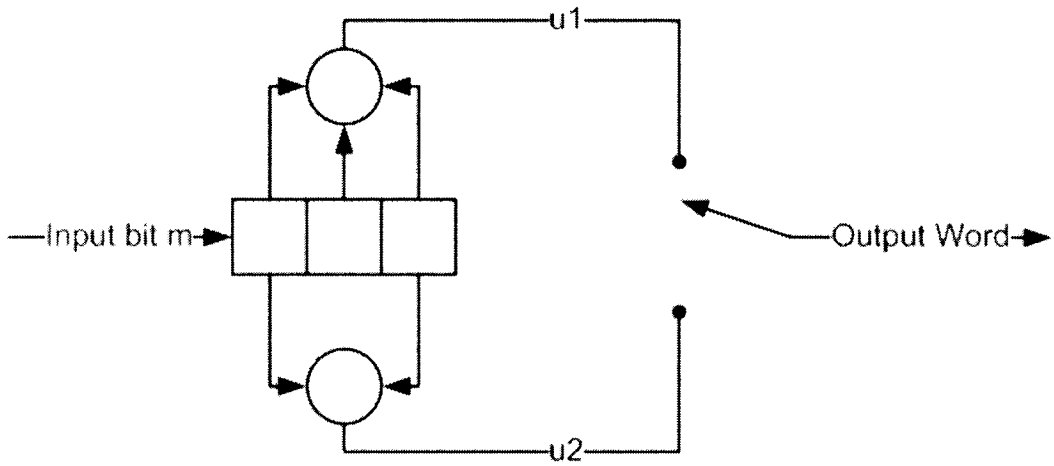

Figure 3.1: Non-Recursive Convolutional Encoder with $k = 1, n = 2$ *and* $K = 3$

After observing the state diagram of Figure 3.2, one can see that the structure repeats itself. For any given sequence of input bits, it is possible to follow through the state diagram and determine the output bits. Given this property, there is another useful way of representing the encoder structure. This is called the *trellis* diagram. A sample of a trellis for block codes will be shown in the upcoming chapter. The encoder shown in Figure 3.1 is known as a non-recursive convolutional encoder whereas the one shown in Figure 3.3 is recursive. Moreover, this encoder is also systematic meaning that the input bit is also part of the output sequence. The generator polynomials associated with the encoder of Figure 3.3 are $g_0(D) = 1 + D + D^2$ and $g_1(D) = 1 + D^2$, and has a rate of $1/2$ since for every one input bit, there are 2 associated output bits [35]. Also, the discussion about state diagrams and trellis applies in the same way to this type of encoder.

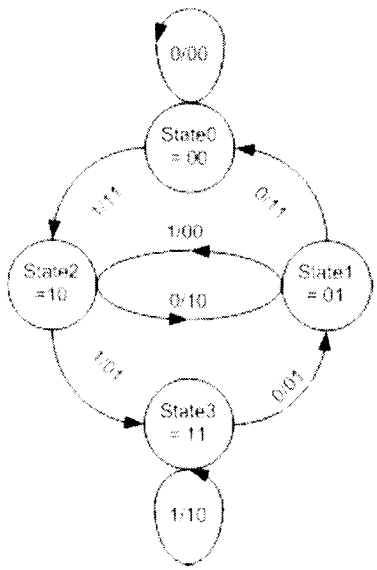

Figure 3.2: State Diagram of Convolutional Encoder

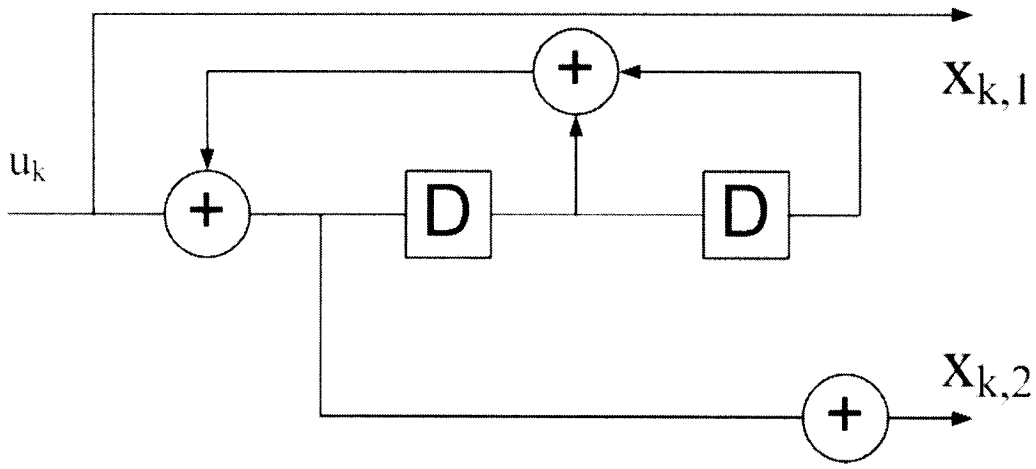

Figure 3.3: Recursive Convolutional Encoder

3.1.2 Concatenated Coding Scheme

This coding scheme uses two levels of coding, an inner and outer code to achieve the performance required. The purpose of these codes is to obtain long and powerful codes by using simple constituent codes. The objective is to attain good performance while having a complexity which is lower than that of a single long code. There are different types of concatenation with the most popular being serial and parallel. One important process that is used for most types of concatenation is that of interleaving. This device receives a given sequence of bits and rearranges it into a new sequence having identical length. The purpose here is to combat burst errors which occur in practical communication systems which are not memoryless. With the use of an interleaver, we are effectively turning a channel with memory into a memoryless one and error correction works better in a bursty channel. In addition, interleaving removes the low-weight codewords and reduces the correlation between the extrinsic

information from the two decoders. Now, we will follow with a brief description of two concatenated schemes.

3.1.2.1 Parallel Concatenated

In this scheme, the k information bits are encoded twice. In essence, the information is first passed through the first encoder. The same information sequence is also passed through the interleaver where it is permuted and encoded again. So, at the output, we have the original information accompanied with two parity bits which have come from the two encoders. This can be better seen in Figure 3.4. In this figure, the encoder could represent either systematic block encoders or recursive systematic convolutional encoders (i.e., this scheme can be used for both block and convolutional codes).

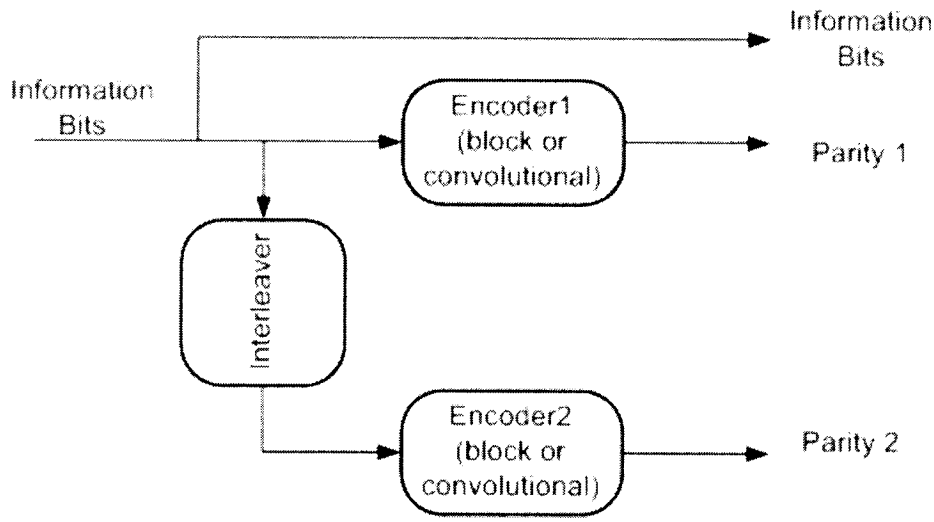

Figure 3.4: Parallel Concatenated Encoder

3.1.2.2 Serial Concatenated

In this scheme, the k information bits are passed through the first encoder after which the output is a codeword. Then, this sequence is passed through an interleaver which permutes the codeword. Finally, this new sequence gets fed into a second encoder. Product codes are derived from this concatenation scheme and they will be the focal point of this thesis.

3.2 Product Codes

3.2.1 Definition

These codes were first introduced by Elias in 1954 [28]. The concept allows for the construction of long codes from the concatenation of two or more smaller codes. Assume we have two block codes A and B with the following parameters: (n_A, k_A, d_A) and (n_B, k_B, d_B). The rates of these two codes are equal to:

$$r_A = k_A/n_A \quad , r_B = k_B/n_B \tag{3.1}$$

C, the product code of A and B can be obtained as follows:

1. Place $k_A \times k_B$ information symbols in an array of k_B rows and k_A columns
2. Code the k_B rows using A
3. Code the n_A columns using B

The construction of this code is shown in Figure 3.5. The resulting product codes parameters are $(n_A n_B, k_A k_B, d_A d_B)$ and the resulting rate will be $r_A r_B$. Another way to define product codes is by saying that codewords can be represented by all $n_A \times n_B$ matrices in such a way that each row and each column are codewords of A and B, respectively. We will use the notation $(n_A, k_A, d_A) \times (n_B, k_B, d_B)$ to denote the product code so as to avoid confusion. Also, it must be said that the minimum distance of the resulting product code is much larger than that of either of the constituent codes, A and B [29].

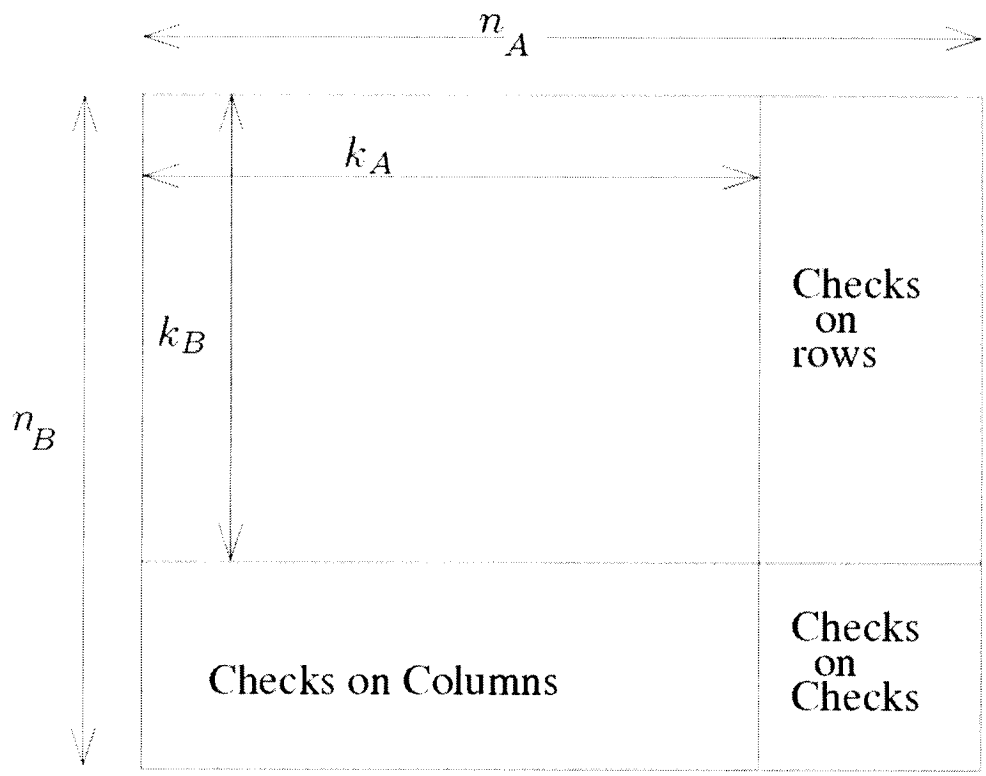

Figure 3.5: Product Code Construction

3.2.2 Properties and Advantages

Although product codes have minimum distances which are much smaller than that of RS codes of the constituent codes of comparable length, product codes remain invaluable because of their error correcting capability. This capability will now be illustrated. Product codes are especially useful for burst error correction. All error patterns that occur on a given number of rows (being less than half the minimum distance of the column code) are correctable. The same can be said for the columns. In addition, if we have random errors; if these errors are not more than half the minimum distance of the row code, these too are correctable. Interestingly, even if the errors are more than half this distance, it is still possible (not definite) to correct these errors. Another important property is the inherent presence of an interleaver in all product codes. As discussed earlier, the interleaver causes these types of codes to excel at burst-error correction. Finally, a major, yet simple, advantage is the basic structure of product codes which makes them even more attractive from a decoding perspective.

3.3 Decoding of Product Codes

Since their introduction in 1954 by Elias[28], there have been numerous decoding algorithms for such codes. Elias suggested that the rows of the received message be decoded using a decoder for A that can decode up to half the minimum distance of A. Then, the columns of this new array would be decoded using a decoder for B that could decode up to half the minimum distance of B. It was shown that this type

of decoder was only capable of correcting up to $(d_A d_B/4)$ errors. We will present some important decoding algorithms followed by the one used for the purpose of this thesis.

3.3.1 Generalized Minimum Distance

The idea of GMD was first suggested by Forney[24]. This method for decoding binary block codes would make use of the soft information from the channel while at the same time still using an algebraic decoder making hard decisions (i.e., either a 0 or 1 for each symbol). An easy way to define the generalized distance between two sequences is by taking the sum of the distances between the two regardless of what type of distance metric is being used. For example, if we were working with Hamming distances, then the generalized distance would be the hamming distance between the two sequences. If we were working with Euclidean distances, then the generalized distance would be the Euclidean distance between the two.

GMD refers to the minimum generalized distance that is correctable between a sequence and a codeword used in transmission using Forney's algorithm. If we had a code with minimum Hamming distance d_{min}, the GMD would be proportional to d_{min}. Instead of using generalized distance as the metric, one could use the squared Euclidean distance when decoding using GMD and the obtained results would be identical.

With GMD decoding, it is assumed that there are two separate decoders for both the row and column codes that can correct all errors up to half the minimum distance of the respective error correcting code. The first step that the GMD decoder takes

is to decode each row of the received array up to half the minimum distance of the row code. Then, each column of the new array is decoded up to half the minimum distance using the column code. The GMD decoder proceeds to delete two rows at a time, which are deemed less reliable, as long as the number of rows deleted is less than minimum distance of the column code. Each time, the columns are decoded over again. In the final step, the decoder chooses the codeword which was closest to the received array. In [24], it is shown that GMD decoding can correct all error patterns of Hamming weight less than half the minimum distance of the code.

It is important to note that the GMD decoder can take into account soft-information by decoding the rows of the product code using a GMD decoder instead of a decoder that corrects up to half the minimum distance of the row code.

3.3.2 Maximum Likelihood

A simple way to obtain the ML codeword is to compare all the distances between the codewords in a given code to the received codeword and select the codeword which is nearest to the received sequence. It is obvious that such a method is highly impractical because it is very time-consuming. This method can be useful in the case where the codes are extremely short and therefore the amount of comparisons that would have to be made are few. Viterbi [30] came out with a decoding algorithm for convolutional codes that would make ML decoding less impractical. Forney was then able to show [31] that the Viterbi algorithm (VA) would allow one to find the shortest path between the start and end node in a trellis of a code.

A trellis T representing a code C having a length n is made up of a set of vertices,

V, a set of edges, E, and a set of labels, L. These vertices are split into sets dependent on time. Edges connect the vertices V_{i-1} with V_i and then with V_{i+1}. For a trellis, each path following a sequence of edges, having a length n, is a codeword of the code C [32].

Although it had been known for a long time that the codewords of a linear block code could be represented by a trellis, a simple method of producing such a trellis was put forward by Bahl, Cocke, Jelinek and Raviv [33] and later popularized by Wolf [43]. It was also shown that this trellis was minimal in the sense that at any time in the trellis, the number of vertices was less than any other possible trellis representation for the same code.

The number of operations needed to perform Viterbi decoding on a Trellis T is equal to $2|E| - |V| + 1$. Also, one can show that the number of edges is related to the number of vertices. So, by looking at the number of vertices, this should give an appropriate estimate of the complexity that will be involved in the decoding of such a code using the Viterbi algorithm.

This discussion serves to show that the complexity of Viterbi decoding on the trellises of product codes is exponentially increasing with the size of the code in question. So, it should be noted that ML Viterbi decoding is not useful for the decoding of codes with large constraint length except when these codes are made from shorter constituent codes, or if the code has either a very high or a very low rate. When the BCJR algorithm was first introduced, it was seldomly used. It was not until Berrou that its full use was exploited in Turbo decoding.

3.3.3 Turbo Decoding

As has already been mentioned in Chapter 1, Turbo codes were introduced by Berrou and his fellow researchers in 1993. Their proposed algorithm was initially designed to iteratively decode two parallel concatenated convolutional codes using a Maximum Aposteriori Probability (MAP) soft decoder of the constituent convolutional codes. Berrou's MAP algorithm performed ML bit estimation which yielded information for each outputted bit. Berrou's algorithm can be viewed as a soft-input soft-output (SISO) decoder. This will be discussed in the next chapter. The decoder introduced in [33] is, in fact, a rework of the MAP algorithm which allows for decoding of a trellis representation of a code. MAP decoding on a trellis of a code is similar to Viterbi in that it starts at the beginning of the trellis until the end or from the end to the beginning. The complexity of MAP is similar to that of Viterbi decoding but the number of operations needed in the former is more than that needed in the latter. In terms of performance, using the VA results in a reduced frame error rate (FER) while the use of MAP results in a better bit error rate (BER) [34]. The Viterbi algorithm and the MAP algorithm are two widely used decoding algorithms often referred to as "trellis-based decoding algorithms"

In turbo decoding, the received sequence is first MAP decoded using the decoder for the first constituent code. Then, the real values obtained are used as the input to be MAP decoded using the decoder for the second constituent code. This procedure is repeated in the coming iterations using real soft values which have been output by the MAP decoders in previous iterations.

Assuming that the output of the AWGN channel is the following sequence: $y =$

$(y_1, y_2, \cdots, y_k, \cdots, y_n)$. The output of the MAP decoder is defined as the *a posteriori* log-likelihood ratio (LLR) for a transmitted "+1" and a transmitted "-1" in the information sequence [39]. If we want to compute the value for the k^{th} term of the information sequence, when we are going through a transition from state s' to s, we have:

$$L(\widehat{u_k}) = \ln \frac{P(u_k = +1|y)}{P(u_k = -1|y)} = \ln \frac{\sum_{u_k=+1}^{(s',s)} P(s', s, y)}{\sum_{u_k=-1}^{(s',s)} P(s', s, y)} \tag{3.2}$$

where

$$\begin{aligned} P(s', s, y) &= P(s', y_{j<k}) \cdot P(s, y_k|s') \cdot P(y_{j>k}|s) \\ &= \underbrace{P(s', y_{j<k})} \cdot \underbrace{P(s|s') \cdot P(y_k|s', s)} \cdot \underbrace{P(y_{j>k}|s)} \\ &= \alpha_{k-1}(s') \cdot \quad \gamma_k(s', s) \cdot \quad \beta_k(s) \end{aligned} \tag{3.3}$$

- $\alpha_{k-1}(s')$ is based on the first $k-1$ terms of the received sequence y. This is expressed by $y_{t<k}$ meaning the sequence of received symbols from the start of the trellis until time $k-1$
- $\gamma_k(s', s)$ is the branch transition probability which is a measure of the information symbol u_k based uniquely on the received symbol y_k
- $\beta_k(s)$ is a prediction measure (i.e. it is based on the future), from $k+1$ to the end of the trellis

These three parameters form the basis for the MAP algorithm.

Since the complexity associated with MAP decoding the constituent codes is relatively high, numerous attempts have been made to improve on this. Hagenauer [35] developed an algorithm known as the Soft Output Viterbi Algorithm (SOVA) that can be used to approximate the results obtained with MAP decoding. Later, Pyndiah *et al.* [36] developed another algorithm in which the MAP decoder is approximated by using a Chase decoder which returns a list of codewords that are then used to compare with the received sequence instead of using the entire code set. This is no longer a trellis-based decoding algorithm but a list-decoding algorithm. This algorithm is of particular interest to us in this thesis and will be described in the coming section.

3.3.4 Iterative Decoding using Pyndiah's Method

As we have already seen, RS codes are block codes having non-binary symbols belonging to $GF(q)$. These codes have the best rate for a given minimum Hamming distance. The original turbo codes used convolutional constituent codes. Pyndiah set out to try and achieve similar performance results with block turbo codes (BTC) using RS codes. The decoding algorithm they used deviates from the customary algorithms in that decisions are made from a reduced set of codewords as opposed to the entire codeword set. This algorithm is still based on soft-input soft-output decoders for the purpose of decoding each of the constituent codes of the product code. This iterative algorithm will now be described, which will include a deeper look into the list-decoding method that is used, namely the Chase algorithm.

Pyndiah considered the transmission of binary symbols $\{0, 1\}$ using Quadrature Phase Shift Keying (QPSK) over an AWGN channel. The observation

$$R = E + N \tag{3.4}$$

is at the input of the decoder, where $E = \begin{pmatrix} e_{11} & \cdots & e_{1j} & \cdots & e_{1n} \\ \vdots & \vdots & e_{ij} & \vdots & \vdots \\ e_{m1} & \cdots & e_{mj} & \cdots & e_{mn} \end{pmatrix}$ is the transmitter codeword and $N = \begin{pmatrix} n_{11} & \cdots & n_{1j} & \cdots & n_{1n} \\ \vdots & \vdots & n_{ij} & \vdots & \vdots \\ n_{m1} & \cdots & n_{mj} & \cdots & n_{mn} \end{pmatrix}$ is a matrix of AWGN samples. Note that n is the block length and m is the number of bits per symbol. To decode R using ML, the optimum decision D would be:

$$D = C^i \; if \; \left|R - C^i\right|^2 < \left|R - C^l\right|^2 \tag{3.5}$$

where $l \neq i$, C^i is the i^{th} codeword, and $|R - C^i|^2 = \sum_{j=1}^{n} \sum_{f=1}^{m} (r_{jf} - c_{jf}^i)^2$ is the squared Euclidean distance between R and C^i. Now, it is important to notice that for an (n, k, d) RS code, there are q^k codewords and that it would be very complex to attempt to optimally decode these codes.

Chase's algorithm is a suboptimal algorithm for near-ML decoding of block codes which has a much lower complexity while maintaining a high level of performance.

Chase observed that at high SNR values, the ML codeword D was on the sphere of radius $(\delta - 1)$ centered on $Y = \begin{pmatrix} y_{11} & \cdots & y_{1j} & \cdots & y_{1n} \\ \vdots & \vdots & y_{ij} & \vdots & \vdots \\ y_{m1} & \cdots & y_{mj} & \cdots & y_{mn} \end{pmatrix}$, where $y_{jf} = 0.5(1 + sgn(r_{jf}))$ and $y_{jf} \in GF(2)$. The procedure was meant to reduce the number of reviewed codewords by only looking at the most probable ones within the sphere. Here is the outline of this procedure:

1. Determine the position of the p least reliable elements by taking a hard decision on R which would give the first word Y^0

2. Form the $2^p - 1$ test patterns by modifying the signs of the elements that were deemed unreliable. These words are called Y^l, where $l = 1, 2, \cdots, 2^p - 1$

3. Algebraically decode the 2^p words to obtain the subset of codewords C^l used in the soft decoding process (see Equation (3.5))

Now that the decision D of a row or column has been found, it is important to compute the reliability of this decision in order to generate soft decisions at the output of the decoder. This reliability is given by the LLR of d_{jf} as:

$$LLR_{jf} = \ln \frac{Pr\{e_{jf} = +1|R\}}{Pr\{e_{jf} = -1|R\}} \tag{3.6}$$

where $e_{jf} = \{-1, +1\}$ in position (j, f) of E $(1 < j < n$ and $1 < f < m)$. Equation (3.6) can be approximated and normalized to obtain:

$$r'_{jf} = \frac{\sigma^2}{2} LLR_{jf} = r_{jf} + w_{jf} \tag{3.7}$$

where r'_{jf} is the estimated normalized LLR of the decision d_{jf}. To determine the normalized LLR_{jf}, a search for the codeword, among the codeword set produced by the Chase algorithm, at minimum Euclidean distance from R must be carried out. Let this code be called $C^{min(+i)}$. A search must then be performed for a codeword, $C^{min(-i)}$ at minimal Euclidean distance from R such that $c_{jf}^{min(+i)} \neq c_{jf}^{min(-i)}$. If there is a codeword that satisfies this condition, then the soft output for d_{jf} is given by:

$$r'_{jf} = \frac{M^{min(-i)} - M^{min(i)}}{4} \times c_{jf}^{min(i)} \tag{3.8}$$

where $M^{min(-i)}$is the Euclidean distance between $C^{min(-i)}$ and R, and $M^{min(i)}$ is the Euclidean Distance between $C^{min(+i)}$ and R. Now, it should be mentioned that the probability of finding a codeword for Equation (3.8) increases with the value of p. However, if p is increased, there will be more codewords in the codeword set attributed by the Chase algorithm and the complexity of the decoder will increase exponentially. So, there are cases when a competing codeword may not exist. The relation to be used is then:

$$r'_{jf} = \beta \cdot c_{jf}^{min(i)} \tag{3.9}$$

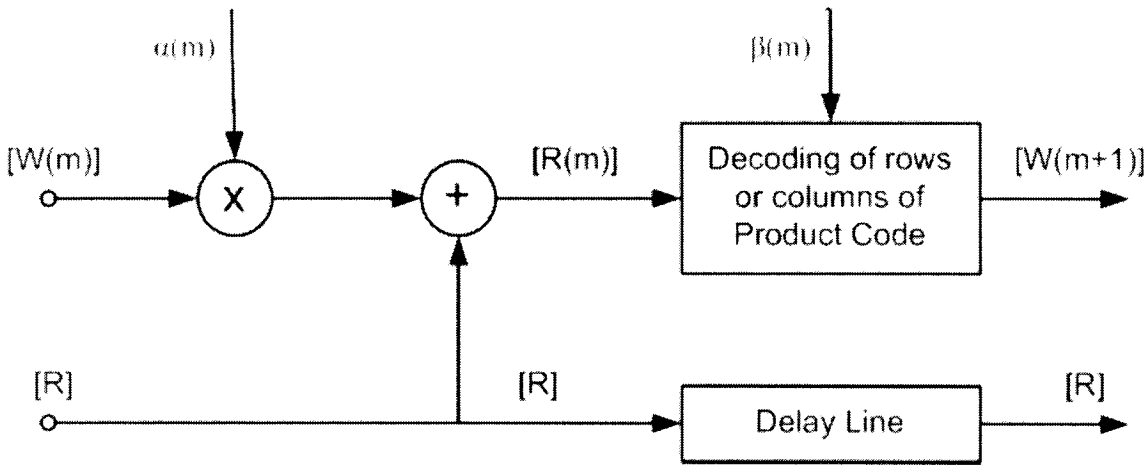

Figure 3.6: Block Diagram of RS Turbo decoder

Given that we can now determine the soft-outputs using Equations (3.8) and (3.9), we can illustrate the turbo decoding process used (see Figure 3.6).

By subtracting the soft input from the soft output, we obtain the extrinsic information, labeled $[W(m+1)]$ in Figure 3.6. So, if we were considering the second decoding of the product code, we would have $[R(2)] = [R] + \alpha(2)[W(2)]$. This turbo decoding algorithm considers a couple of parameters which must now be explained:

- weighting factor α: to reduce the dependency of α on the product code. This factor is used to reduce the effect of the extrinsic information in the soft decoder in the starting steps when the BER is high. α takes on a small value in the first iterations and increases to 1 as the BER approaches 0 [36]:

$$\alpha(m) = [0.0, 0.2, 0.3, 0.5, 0.7, 0.9, 1.0, 1.0]$$

- reliability factor β: normally determined as a function of the BER. In these simulations, the increase in β has been fixed as follows:

$$\beta(m) = [0.2, 0.4, 0.6, 0.8, 1.0, 1.0, 1.0, 1.0]$$

Moreover, the term *iteration* used for turbo decoding corresponds to a row decoding followed by a column decoding of the product code. If the readers are interested in delving further into this algorithm, they are kindly referred to [36, 37, 38]. This encompasses all the information needed to carry out the suboptimal decoding of RS BTCs.

3.4 Summary

In this chapter, the reader was introduced to the alternative of block coding, namely convolutional coding. In addition, code concatenation was discussed which brings us to the topic of serial concatenated codes or product codes. The definition as well as the properties and advantages of using product codes were also presented. Finally, various approaches for decoding product codes were presented, ranging from GMD and MLD to turbo decoding algorithms. A near-optimum turbo decoding algorithm was presented in detail because of its importance in the development of the proposed algorithm.

Chapter 4

SDD of RS Product Codes

As has already been discussed in Chapter 3, Soft-Input Soft-Output (SISO) decoding algorithms are used for iterative decoding. By using log-likelihood ratios, these decoders can accept soft inputs and produce soft outputs which can be broken down into three terms: the soft channel inputs, the *a priori* information and an extrinsic value. The latter is then used as *a priori* information at the following iteration. The decoding algorithm has three independent estimates for LLRs of the information bits: the channel value $L_c \cdot y$, the *a priori* values $L(u)$, and the values from an independent third estimator, $L_e(\widehat{u})$ which is in fact the extrinsic information. It is obvious that before decoding has begun, there is no a priori information and this value is initialized to zero. This SISO decoder which is used for product codes can be seen in Figure 4.1[39].

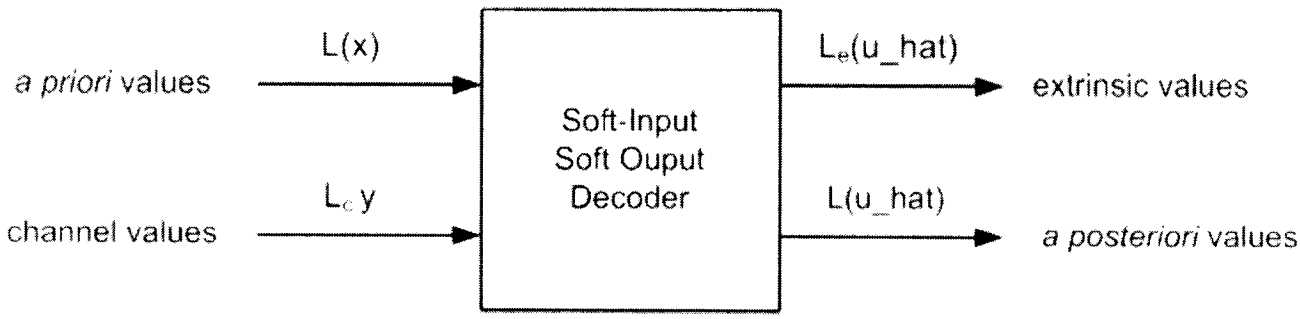

Figure 4.1: Soft-Input Soft-Output Decoder

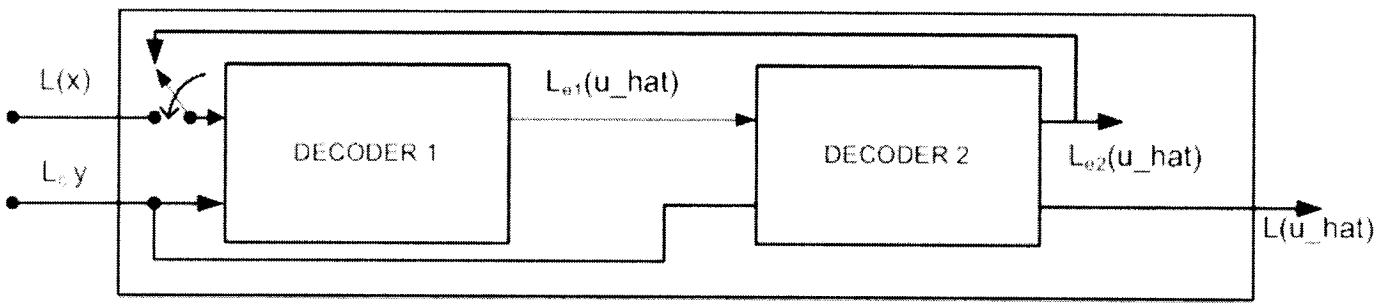

Figure 4.2: Iterative SISO Decoder

For the iterative decoding process, the SISO Decoder of Figure 4.1 can be expanded as to show the inputs of the row and column decoder which are used for product codes (see Figure 4.2). From the above Figure, we can see that the extrinsic value for the first iteration is given by:

$$L_{e1}(\widehat{u}) = L_1(\widehat{u}) - (L_c \cdot y + L(u)) \tag{4.1}$$

The second decoder will use the result of Equation (4.1) as a priori information and will produce extrinsic information in the form of:

$$L_{e2}(\widehat{u}) = L_2(\widehat{u}) - (L_c \cdot y + L_{e1}(u)) \tag{4.2}$$

The resultant of Equation (4.2) will then be used for the second iteration and this process will continue until there is not much variation between iterations. Once the iterations have been completed, the second decoder combines the extrinsic information from itself and Decoder 1 and a decision is made. In this thesis, a method is devised through which a soft-decision is made. The algorithm for carrying out the soft-decision decoding is based on the one described in Section 2.4.2 with a modification rendering that algorithm suboptimal for the purpose of keeping the overall complexity as low as possible. The performance, as we will see, is an improvement over the currently existing methods in which a hard decision is taken once the iterations have finished.

4.1 System Description and Model

The simulations performed considers the transmission of information using binary phase shift keying (BPSK) signaling over a Gaussian Channel. The RS product codes can be constructed based on bit concatenation or Q-ary symbol concatenation. In the first category, there are $(k_A \times k_B \times q^2)$ information bits as per the definition in Section 3.2.1 where q is the number of bits per symbol. In this case, each bit belongs to different Q-ary symbols in the row and column coding. In the second category, the product code is constructed from $(k_A \times k_B)$ Q-ary information symbols which means $(k_A \times k_B \times q)$ data bits. The q binary elements which correspond to the Q-ary symbol are located at the intersection of a row and a column (see Figure 4.3) [40]. It should be noted that q and Q are identical.

This second category is the one used throughout the simulations. Also, the RS product code is constructed using two identical constituent codes with $\Re_A(n_A, k_A, d_A) = \Re_B(n_B, k_B, d_B)$ therefore , we can say that $n_A = n_B = n$, $k_A = k_B = k$, $d_A = d_B = d$ and $r_A = r_B = r$. The system model on which this thesis is based is presented in Figure 4.4. It is proposed that by using a SDD algorithm to both select the set of reviewable codewords used to calculate the LLRs and make the decisions on the final codewords after iterations have been completed, we can improve the performance of these important codes and render them even more powerful than they presently are. While Pyndiah uses a Chase Decoder to obtain his list of reviewable codewords and later makes a hard decision to obtain his estimate, it would be more advantageous to make use of all the available channel information when making a decision because soft-decision decoding is an optimal detection method which achieves a lower prob-

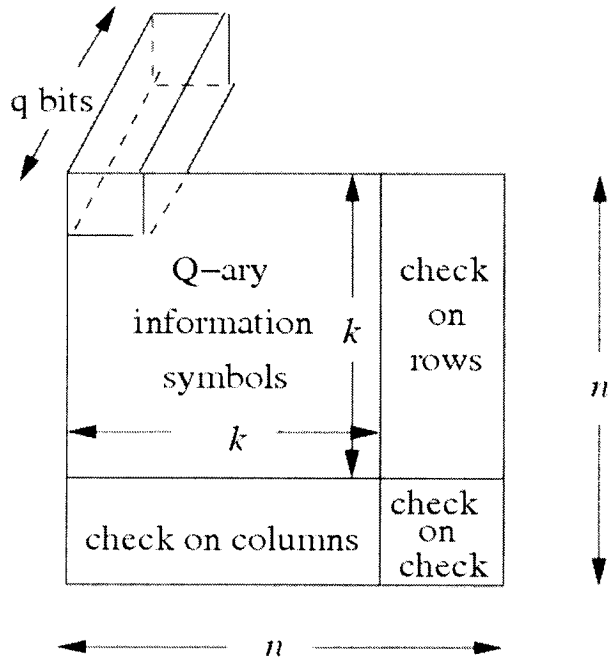

Figure 4.3: RS Product Code based on Q-ary symbol concatenation

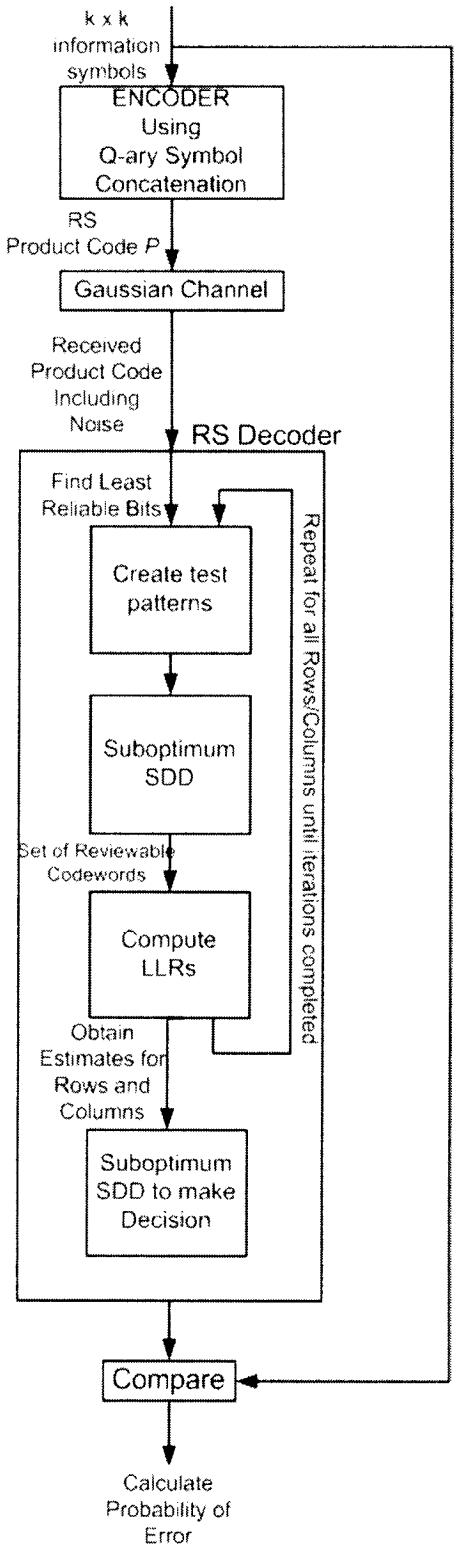

Figure 4.4: Proposed System Model

ability of error (i.e. better performance) [41]. The performance calculation can be easily derived by taking a closer look at Figure 4.3. Given the fact that the product code is constructed in such a manner, it is possible to state:

$$\begin{aligned} P_{BitError} &= \frac{Bit\,Errors}{Blocks \times \frac{bits}{block}} \\ &= \frac{Number\,of\,bit\,errors}{(Number\,of\,blocks \times k \times k \times q)} \end{aligned} \tag{4.3}$$

where the numerator is a reflection of how many bit errors have occurred during the transmission over the channel and *number of blocks* signifies the number of product codes that have been sent over the channel. Equation (4.3) is used in Section 4.3 to obtain the simulation results.

4.2 Suboptimum SDD of RS code

4.2.1 Complexity Reduction by means of a Trellis

One important way to simplify the decoding complexity in the SDD process is by reducing the amount of operations performed for determining the coset pattern (refer to step 1 of Section 2.4.2). If one were to use the Vardy-Be'ery algorithm to calculate $\delta^2_{min}(y^{(j)}, l_s)$, the distance between each interleaved segment of the received signal from each coset of the BCH code would have to be calculated separately. So, for a $(15, 11)$ RS code which is broken into a $(15, 7)$ binary BCH code, this would require a total of 2^{15} Euclidean distance calculations for every 15-bit pattern. This

could be explained by the fact that the binary BCH code has $2^{n-k_{BCH}}$ cosets and $2^{k_{BCH}}$ codewords. The standard array, containing all possible 2^n (and in this case 2^{15}) possible codewords, is an organizational tool used to contain information about a code. The first column of this array contains all possible coset leaders ($2^{n-k_{BCH}}$) and the first row contains all the BCH codewords ($2^{k_{BCH}}$) The rest of the array is filled out by modulo-2 adding the rows and columns (for example, $Element_{22} = Row_{02} \oplus Column_{20}$). In the end, the array will contain all possible 2^{15} 15-tuples for the BCH code. The standard array has the form:

$$
\begin{array}{ccccccc}
cl_0/cw_0 & cw_1 & \cdots & cw_i & \cdots & cw_{2^{k_{BCH}}} \\
cl_1 & cw_1 + cl_1 & \cdots & cw_i + cl_1 & \cdots & cw_{2^{k_{BCH}}} + cl_1 \\
cl_2 & cw_1 + cl_2 & \cdots & cw_i + cl_2 & \cdots & cw_{2^{k_{BCH}}} + cl_2 \\
\vdots & & \vdots & & \vdots & \vdots \\
cl_j & cw_1 + cl_j & \cdots & cw_i + cl_j & \cdots & cw_{2^{k_{BCH}}} + cl_j \\
\vdots & & \vdots & & \vdots & \vdots \\
cl_{2^{n-k_{BCH}}} & cw_1 + cl_{2^{n-k_{BCH}}} & \cdots & cl_{2^{n-k_{BCH}}} & \cdots & cw_{2^{k_{BCH}}} + cl_{2^{n-k_{BCH}}}
\end{array}
$$

where cl_j represents the $0 \leq j \leq 2^{n-k_{BCH}}$ coset leaders and cw_i represents the $0 \leq i \leq 2^{k_{BCH}}$ BCH codewords. Note that the array contains every n-tuple without any missing and without any replicas [42].

For longer codes, this amount of calculations is unacceptable. One way to reduce this amount is by calculating the distance of $y^{(j)}$ to all $2^{n-k_{BCH}}$ cosets simultaneously using a trellis. A Wolf Trellis [43] based on the parity-check matrix can be constructed

based on the Equation (2.4) of Section 2.1.2. Each node is represented by a binary $(n - k_{BCH})$-tuple, $S_p(a) = (a_1, a_2, \cdots, a_{n-k_{BCH}})$ where p represents the depth. The trellis can then be formed by following these steps:

1. At depth 0, there is only one node called $S_0(0) = (\mathbf{0}, \mathbf{0}, \cdots, \mathbf{0})$

2. Each node has 2 branches coming out of it based on whether the input was a 0 or 1. Therefore, $S_p(a)$ would branch out to $S_{p+1}(a'_0)$ and $S_{p+1}(a'_1)$ according to

$$S_{p+1}(a'_{\{0,1\}}) = S_p(a) + \{0, 1\} \times h_p^T \tag{4.4}$$

   where h_p is the p^{th} column of H

3. The branch that led to $S_{p+1}(a'_0)$ is marked with a 0 and the one that led to $S_{p+1}(a'_1)$ with a 1.

4. The trellis stops when $p = n$ and the nodes at this depth are called terminal nodes.

Once constructed, the trellis will contain 2^n different paths which correspond to all 2^n binary n-tuples. Finding the closest n-tuple from each coset to $y^{(j)}$can be likened to determining the shortest path (using squared Euclidean distance as the branch metric) from the initial node $S_0(0)$ to the $2^{n-k_{BCH}}$ terminal nodes. The VA is used to determine this shortest path. This concept will be better illustrated with an example for the $(15, 7)$ BCH code. After several calculations, it is possible to present the parity-check matrix which has the form shown in Equation (2.4) of Section 2.1.2:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Based on this matrix, it is possible to construct the associated trellis (seen in Figure 4.5). In this trellis, all the transitions after depth $p = 8$ are not drawn so as not to clutter the figure. These branches can be determined from Equation (4.4) and H. The dotted lines represent transitions based on an input of *0* and solid lines represent transitions based on a *1*. The two paths terminating at $S_{15}(0)$ demonstrates how 15-tuples of the same coset share a unique terminal node. In this case, the all-zero and all-one 15-tuple, which belong to the first coset of this code, both terminate at node 0 when $p = 15$ Now, since the VA uses 3 operations for every node, namely two add operations to update the metrics and one comparison to select the survivor, we can compute the total number of operations needed to search through the trellis [26]:

$$N(C_{BCH}) = 3[(k_{BCH} + 2)2^{n-k_{BCH}} - 2] \tag{4.5}$$

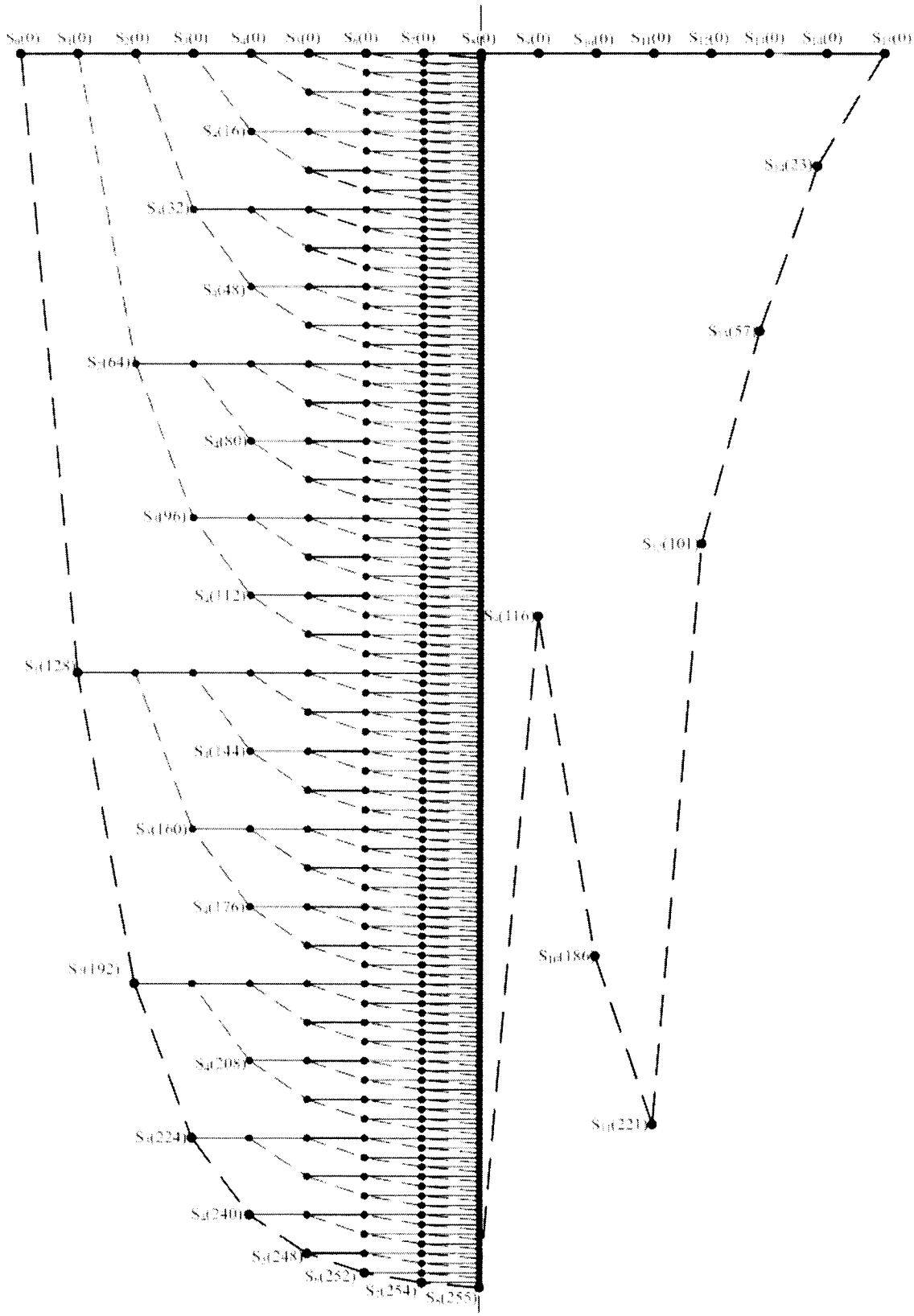

Figure 4.5: Trellis for $(15, 7)$ BCH Code

For our example, this comes out to $N = 3[(7+2)256 - 2] = 3[2304 - 2] = 6906$ operations which is much less than that required if we were to compute the distances for each 15-tuple of the standard array separately.

4.2.2 Complexity Reduction by putting a constraint on Levels Searched

A second way that would improve the complexity is to put a constraint on how many levels are to be examined (refer to Equation (2.32) and step 2 of Section 2.4.2). In step 2 of 2.4.2, we saw that through partial ordering, we separated the coset patterns and grouped them into levels. Back to our example of the $(15, 11)$ RS code, there were a total of $L = 1020$ levels. This means that the search for a coset pattern would continue until the conditions specified in Figure 2.4 is satisfied. Although this method guarantees the return of a coset pattern $\widehat{l}$ that satisfies the ML criterion, it is possible that the complexity at later levels causes the systems performance to degrade. This complexity could be reduced by relaxing the ML criterion and placing a constraint on the levels to be checked. A suboptimum search algorithm is also presented in [26] which differs from the one that is being proposed. In [26], if the suboptimum search has not yielded a valid coset pattern, a decoding failure is immediately declared and the next packet is sent for decoding. In the case of our suboptimum algorithm, a decoding failure is not immediately declared and the last valid pattern that was found is used as the coset pattern for the rest of the decoding process. This can be better understood after referring to Figure 4.6.

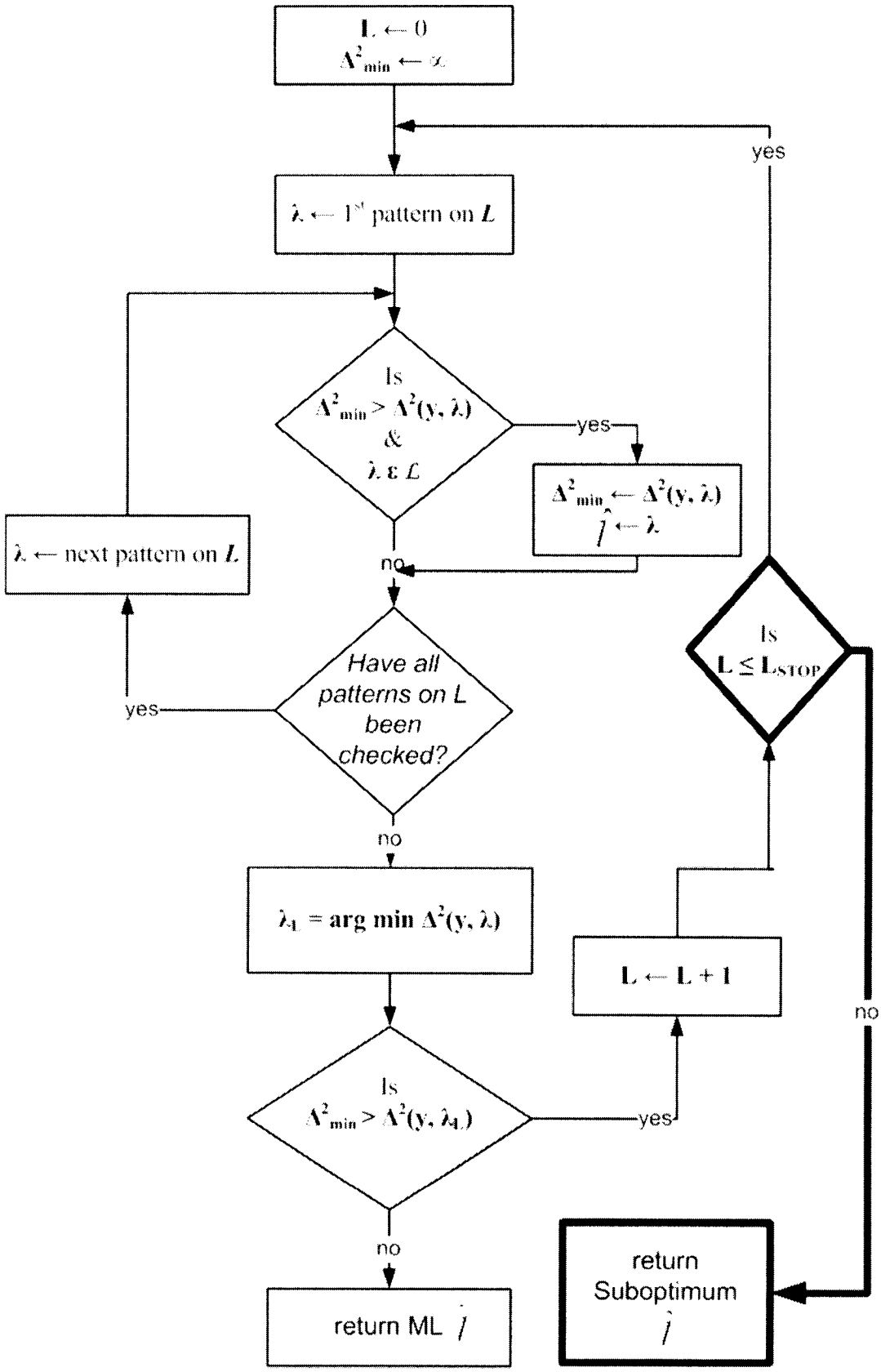

Figure 4.6: Flow Diagram Representing the Suboptimum Search Algorithm

L	S_L
10	1001
30	46376
50	316251
80	1.9295×10^6
100	4.59813×10^6
1021	4.29497×10^9

Table 4.1: Number of Coset Patterns from level 0 to L

The differences between the suboptimum search algorithm and the one in Figure 2.4 of Section 2.4.2 is clear. As can be seen, if the maximum level L_{STOP} has been reached without the final condition being satisfied (i.e. $\lambda^2_{max} < \lambda^2(y, \lambda_L)$), a suboptimum coset pattern is returned and used for the remainder of the decoding process. Let us analyze this for a moment by first presenting a formula used to calculate the total number of coset patterns in levels 0 to L:

$$S_L = \sum_{t=0}^{\lfloor L/\kappa+1 \rfloor} (-1)^t \begin{pmatrix} m \\ t \end{pmatrix} \begin{pmatrix} m+L-t(\kappa+1) \\ m \end{pmatrix} \tag{4.6}$$

Based on this we can show that, using the same $(15, 11)$ RS code, the following number of coset patterns need to be checked. It is important to first remember that the total number of levels is $L_{max} = 1021$ with $m = 4$ and $\kappa = 255$ for this code (see Table 4.1).

Obviously, although we would always desire to find the coset pattern satisfying the ML criterion, the number of patterns to be checked as L grows makes this task

very difficult. We would always hope that $\hat{l}$ is found in one of the lower levels, but if this is not the case, the decoding complexity is much too elevated not to use the suboptimum algorithm. The best tradeoff between performance and complexity was found when L_{STOP} was set to 50 as will be seen shortly.

4.2.3 Simulation Results

After applying the complexity saving methods described above, it was clear that a noteworthy and usable suboptimum algorithm has been found that could aptly be used to decode RS codes. The savings in complexity are shown with respect to a variable P_{ave} which represents the average number of coset patterns that need to be checked when performing the search for the required coset pattern $\hat{l}$. At lower values of the SNR, it is understandable that since the system is corrupted by a lot of noise, the suboptimum search algorithm will usually reach its stop level L_{STOP} before satisfying the ML criterion. The ML search algorithm will continue searching through the levels until it has found the coset pattern satisfying the ML criterion. This explains why at lower SNR, the suboptimum algorithm performs less operations then the MLD. We can also see that at higher values of SNR, the curves for MLD and the suboptimum algorithm converge. This can be explained by saying that when the signal is much stronger than the noise in the communication channel, the ML criterion is usually satisfied before reaching L_{STOP} in the suboptimum search and thus the number of cosets checked is almost identical. Based on Figure 4.8, we can say that this proposed algorithm is better at low SNR because as the SNR increases, the complexity of the proposed suboptimum algorithm (denoted "Proposed SUB")

approaches that of the MLD algorithm. In terms of performance, we can observe that the curves based on the MLD and that of the proposed suboptimum algorithm are similar in nature with the latter experiencing a slight decrease in performance.

For a block error rate of 10^{-5}, the difference between the MLD algorithm and the proposed suboptimum algorithm is about 0.2 dB which is acceptable when taking into account the savings that have been made in terms of decoding complexity. Therefore, the suboptimum algorithm achieves near-MLD performance whilst lowering the overall system complexity. It should also be noted that in this section, we are dealing with block error rate as opposed to bit error rate which will be used in the following section. In order to calculate the block error probability which is shown in Figure 4.7, the following formula is used:

$$P_{BlockError} = \frac{number\,of\,block\,errors}{total\,number\,of\,blocks} \tag{4.7}$$

where a block error occurs when any bit in the sequence has been found to be erroneous. Therefore, if even 1 out of the 44 ($11\,information\,symbols \times 4\,bits\,per\,symbol$) information bits is found to be wrong, a block error is declared. This is different from Equation (4.3) in which the bit error probability is calculated. Also, the MLD algorithm achieves a $P_{BitError} = 10^{-5}$ for an SNR value of 5.3 dB (see Table II of [26]) whereas the suboptimum algorithm achieves the same $P_{BitError}$ at an SNR of approximately 5.5 dB.

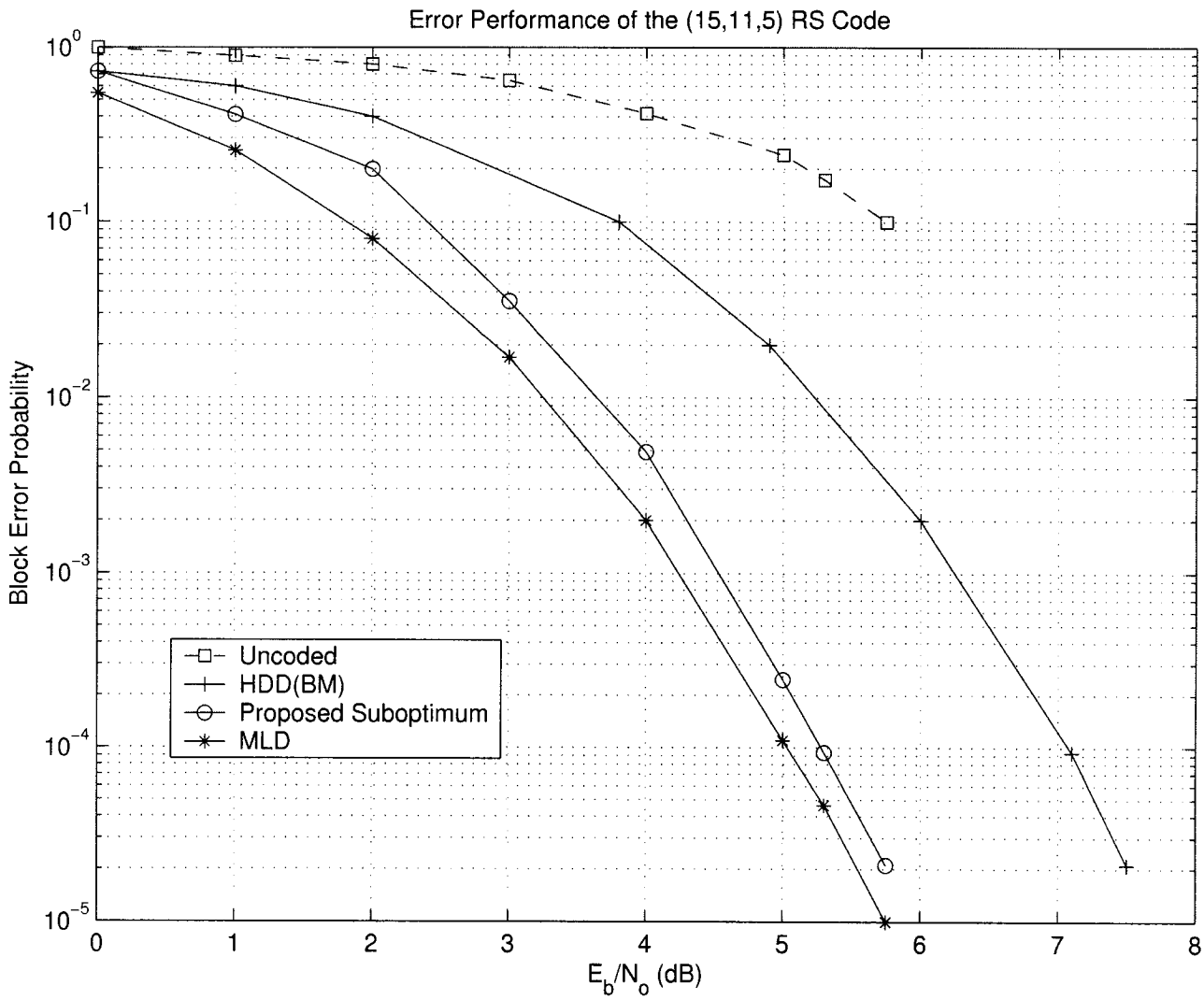

Figure 4.7: Error Performance of the $(15, 11, 5)$ RS Code

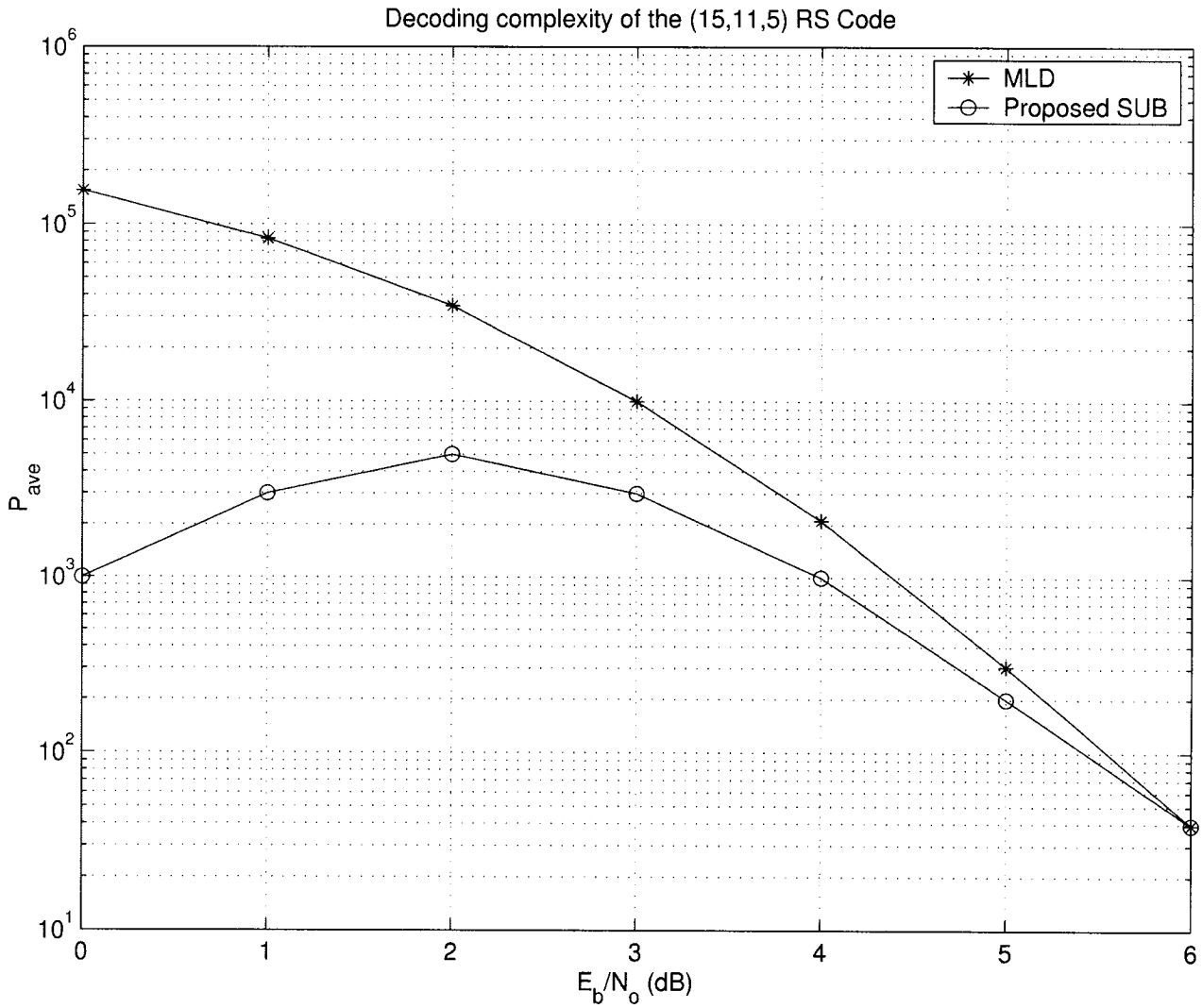

Figure 4.8: Decoding Complexity of the $(15, 11, 5)$ RS Code

Using this proposed suboptimum decoder of RS codes in tandem with RS product codes would seem to be a good way to achieve better results than have already been achieved by Pyndiah who used a List-decoder and performs a hard-decision. This will be investigated in the following section.

4.3 Iterative SDD of RS Product Codes

4.3.1 Incorporating Suboptimum SDD into Iterative Decoding Process

Now, the main issue lies in trying to find a way to incorporate the suboptimum algorithm devised in Section 4.2 into the RS turbo decoder illustrated in Figure 3.6. It would have been possible to just use the suboptimal RS decoder to make the decisions on the rows/columns of the product code but this idea was taken further and used prior to the last iteration of the decoding process. The procedure used by Pyndiah (and based on the Chase Decoder) to identify the set of most probable codewords has already been discussed. Now, the procedure which has been used in the proposed algorithm will be enumerated and examined.

1. Determine the position of the p least reliable elements of a given row or column, denoted R, in the RS product code P. Note that for the results contained within this thesis, p has been set to 5.

2. Form the 2^p potential test patterns by modifying the signs of the elements that were deemed unreliable in R

3. Do the following for R:

   - Break down each sequence y of $m \times n$ bits into m n-bit sequences $y^{(j)}$, where we have $y = (y_1^{(1)}, y_1^{(2)}, \cdots, y_1^{(m)}, y_2^{(1)}, \cdots, y_n^{(m)}$ and $y^{(j)} = (y_1^{(j)}, y_2^{(j)}, \cdots, y_n^{(j)})$
   - Calculate $\delta^2_{min}(y^{(j)}, l_s)$ for $1 \leq j \leq m$ and $1 \leq s \leq 2^{n-k_{BCH}}$, where l_s is a coset leader belonging to $C_{BCH} + l_s$
   - Find a valid coset pattern $\widehat{l}$ using the suboptimal search algorithm of Figure 4.6
   - Calculate $\widehat{b}$ using $\widehat{b}^{(j)} = arg\ \min_{b \in C_{BCH}}(y^{(j)}, s(b + \widehat{l}^{(j)}))$ for $1 \leq j \leq m$
   - The decision D is found by computing: $v = \widehat{l} + \widehat{b}$

4. Do the following for each of the 2^p words:

   - Break down each sequence y of $m \times n$ bits into m n-bit sequences $y^{(j)}$, where we have $y = (y_1^{(1)}, y_1^{(2)}, \cdots, y_1^{(m)}, y_2^{(1)}, \cdots, y_n^{(m)}$ and $y^{(j)} = (y_1^{(j)}, y_2^{(j)}, \cdots, y_n^{(j)})$
   - Calculate $\delta^2_{min}(y^{(j)}, l_s)$ for $1 \leq j \leq m$ and $1 \leq s \leq 2^{n-k_{BCH}}$, where l_s is a coset leader belonging to $C_{BCH} + l_s$
   - Find a valid coset pattern $\widehat{l}$ using the suboptimal search algorithm of Figure 4.6
   - Calculate $\widehat{b}$ using $\widehat{b}^{(j)} = arg\ \min_{b \in C_{BCH}}(y^{(j)}, s(b + \widehat{l}^{(j)}))$ for $1 \leq j \leq m$
   - The potential test pattern is found by computing: $v = \widehat{l} + \widehat{b}$

5. Compare the list of 2^p potential test patterns and compare with the decision D obtained from R. If any duplicate patterns exist in the set of reviewable

codewords, discard those patterns. There is now Z codewords that belong to the set of reviewable codewords where $0 \leq Z \leq 2^p$

Clearly, this differs from Pyndiah's method in several places. First of all, the received vector is sent as is into the suboptimum decoder and returns what will later be used as the decision for the LLR calculations rather than using Equation (3.5). This step is important because a hard decision is not made on the received vector. Instead, we use all the channel information and trust that the soft-decision decoder will produce a worthy decision D. Another major difference is the manner through which the list of reviewable codewords is obtained. Evidently, it is probably quicker and less complex to just make a hard-decision on R and then create the test patterns by simply modifying the signs of the unreliable bits. However, by sending these potential test patterns through the decoder (as listed in step 4 above), we are obtaining a better set of reviewable codewords to calculate the *a priori* and extrinsic information that is vital in the overall turbo decoding process.

Once we have the decision D as well as the set of reviewable codewords, we can now verify the reliability of each component of D and compute the soft-decision at the output of the soft-input decoder. Now, we conduct a search for a codeword, belonging to the subset of codewords (see step 5 above), that is at minimum Euclidean distance from R such $c_j \neq d_j$, where $0 \leq j < m \times n$. If such a codeword can be found amongst the subset, the soft-output of bit d_j is given by:

$$r'_j = \left(\frac{M^C - M^D}{4}\right) \times d_j \tag{4.8}$$

where M^D is the Euclidean distance between the decision D, and the received codeword R and M^C is the Euclidean distance between the found codeword C and R. If we cannot find a codeword belonging to the subset such that $c_j \neq d_j$ or if there are no codewords belonging to the set of reviewable codewords, the following relation is used:

$$r'_j = r_j + \beta \times d_j \tag{4.9}$$

It is easy to see that Equations (4.8) and (4.9) are very similar to Equations (3.8) and (3.9) of Section 3.3.4. One could notice that in (3.9), the estimated normalized LLR of the decision is calculated using only the reliability factor β and the decision bit. This is slightly different from (4.9) in which the estimated normalized LLR of the decision is calculated using β, the decision bit and the soft input bit. When determining the soft decision, it would seem only reasonable to take into account the most likely decision as well as other factors which include the previous soft input. While [38] does not include the soft-input in their formulas, [40] does and it was determined through experiment that the performance results obtained are better when using Equations (4.8) and (4.9).

4.3.2 Selecting p

If we are hoping to always find the ML codeword, then it would be best to set p in such a way that all possible codewords are verified. If this were the case, we

would always find the most likely codeword. The only problem here is that with longer codes, this is a waste of time as well as an unnecessary increase in system complexity. In the above description of the steps, it was stated that p, in this thesis, has been set to a value of 5. Consequently, there is $2^p = 2^5 = 32$ potential test patterns which will each be decoded using the suboptimum algorithm. Even when the iterations have made the information more reliable, this process will continue to decode 32 potential test patterns for each row and column of the product code until the final iteration and the next product code is sent for decoding. Therefore, if we are considering the $(15, 11, 5)$RS code, for each row, there would be 32 potential test patterns created as well as the decision D. This means a total of 33 words sent into the suboptimal decoder. After 4 full iterations (i.e. row and column decoding), this means that a total of 3960 codewords have been sent into the proposed decoder. Even though the decoder was seen to produce effective results in Section 4.2.3, it is still unreasonable efficiency-wise to decode all those words to obtain the decoding of one product code. If p was set to 4, the number of words to be decoded would be 2040. If p were 1, there would be 360 words to decode. Complexity-wise, this last value of p would be the most favorable. However, it would also rarely be able to produce reliable decisions from the decoding process.

It should be observed that as the iterations increase, the reliability of our decisions should be getting better. Therefore, it would be wise to take advantage of our previous decodings (in the lower iterations) to render our system more efficient (in the later iterations). If we continuously search out the least reliable bits even when they are not unreliable, we are wasting valuable time to the decoding of unnecessary

words. Let us look at an example assuming we have a codeword length of 6 and the following received sequence:

$$y_{initial} = [\ 0.89 \quad -0.01 \quad -1.24 \quad 0.04 \quad 0.95 \quad -0.98\ ]$$

and let us also assume that $p = 3$ Therefore, we would take the 3 least reliable bits out of these 6 and create 2^3 potential candidates. The 3 least reliable bits in terms of absolute value would be $|0.89|$, $|-0.01|$, $|0.04|$ corresponding to bits 1, 2 and 4, respectively. Although it is already clear that 0.89 is most probably a +1. Assume that three iterations have gone by and the vector is now:

$$y_{iteration=3} = [\ 1.75 \quad -0.72 \quad -1.80 \quad -0.94 \quad 1.56 \quad -2.24]$$

For this vector, the three least reliable bits are $|-0.72|$, $|-0.94|$ $|1.56|$ corresponding to bits 2, 4 and 5. Now, most of the bits seem to be converging to either a +1 or a -1. Despite this, there will still be 2^3 words that will be sent through the decoder at this stage. This should make it clear why it becomes redundant to the point of being wasteful to maintain a fixed value of the parameter p.

It is the contention of this thesis that by setting an upper limit on p and then looking for at most p least reliable bits that fall within a preset range of reliability that we can save time in decoding the product code. Again, let us refer to the example from above. Assuming we set this range to look for any bit that lies within $-0.2 \leq$

$|y_j| \leq 0.2$, then initially, the bits which would be modified would be $|-0.01|$, $|0.04|$. This would produce 4 vectors to be decoded. Then, after 3 iterations, when looking at $y_{iteration=3}$, there is not any bit that lies within our initial range and thus, there are no additional vectors to be decoded and the next row/column of the product code will now be checked. This saves a lot of time in addition to taking advantage of the reliabilities of the decisions made after each iteration. Thus, as the number of iterations increase, the bits will begin to converge and stray away from the preset range. This will reduce the decoding time as opposed to the methods described in [36, 37, 38].

4.3.3 Simulation Results

Using the methods described in Sections 4.3.1 and 4.3.2, the simulation of the $(15, 11, 5) \times (15, 11, 5)$ RS Product code was carried out. This product code has the following properties:

$$rate\ r = \frac{11}{15} \times \frac{11}{15} = 0.53778, \quad d = 5 \times 5 = 25$$

Figure 3.6 of Section 3.3.4 is still a good representation of the decoding process where we still need the weighting factor α as well as the reliability factor β. Simulations that have been performed used the following values for these variables:

$$\alpha(m) = [0.0, 0.2, 0.3, 0.5, 0.7, 0.9, 1.0, 1.0]$$

$$\beta(m) = [0.2, 0.4, 900.6, 0.8, 1.0, 1.0, 1.0, 1.0]$$

These variables can be modified slightly without experiencing a noticeable degradation in the overall system performance. Also, the value of the preset range through which the set of reviewable codewords is created is $-0.1 \leq |y_j| \leq 0.1$ Obviously, the larger this range is made, the higher the decoding complexity is made. The number of iterations used for obtaining the results was 4. For each additional iteration, the complexity still increases while for a BER of 10^{-5}, the improvement in terms of E_b/N_o becomes small enough to make these additional iterations unnecessary. In Figure 4.9, the BER of the $(15, 11, 5)^2$ product code using the proposed algorithm is shown. We can see that the coding gain improves between each of the 4 iterations. Looking specifically at a $BER = 10^{-5}$, there is a gain of 1.2 dB, 0.45 dB and 0.1 dB between the 1^{st} and 4^{th} iteration. After iteration 4, there is not much coding gain because the slope of the curve is already too steep. In Figure 4.10, the BER versus signal to noise ratio of the simulated RS turbo code and a similar RS turbo code simulated using Pyndiah's method are shown at the 4^{th} iteration. At $BER = 10^{-5}$, there is a coding gain of approximately 0.63 dB.

The two following figures illustrate the success in merging a soft-decision decoding algorithm for RS codes into the iterative decoding process of RS product codes.

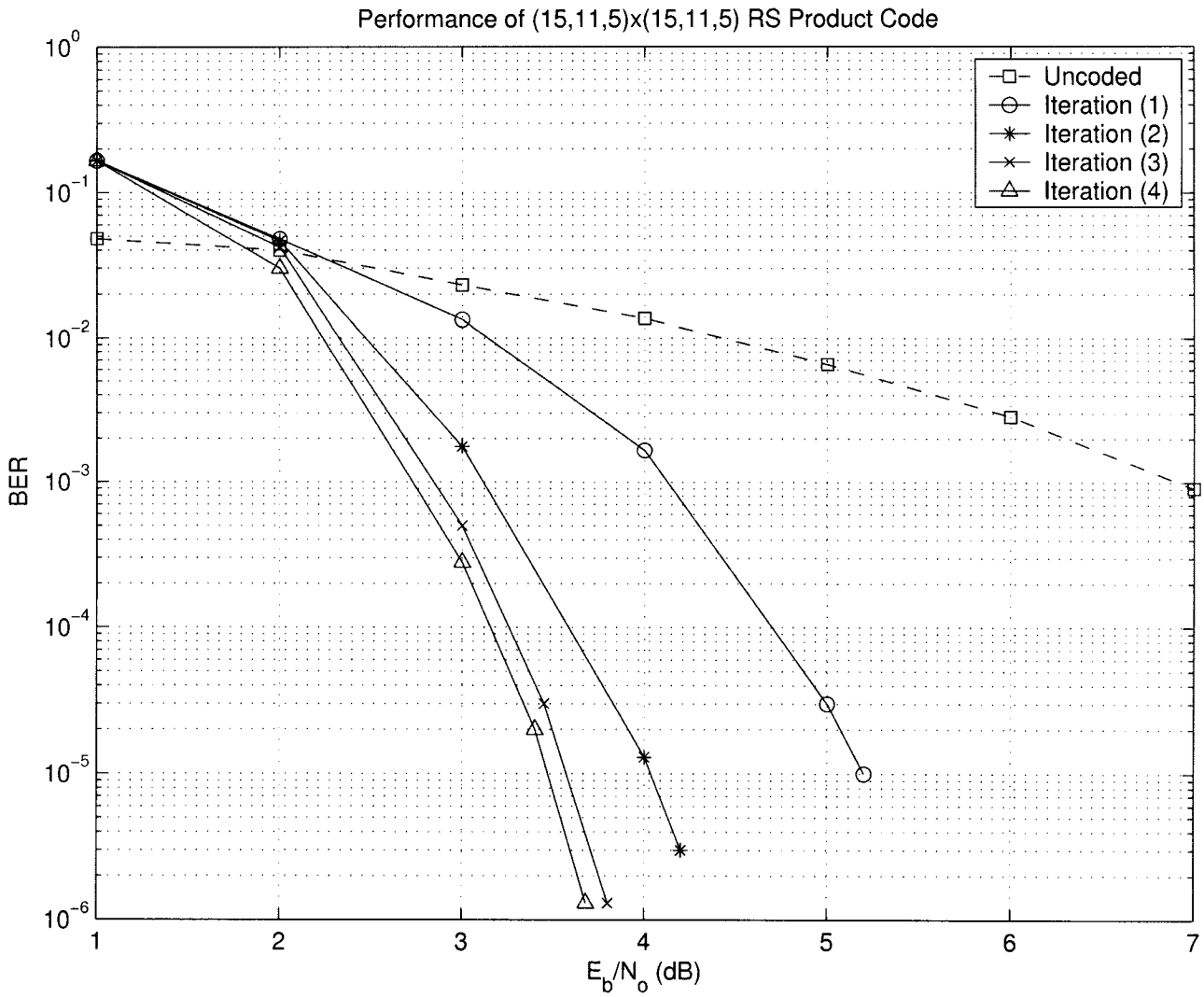

Figure 4.9: Performance of $(15, 11, 5)^2$ RS Product Code

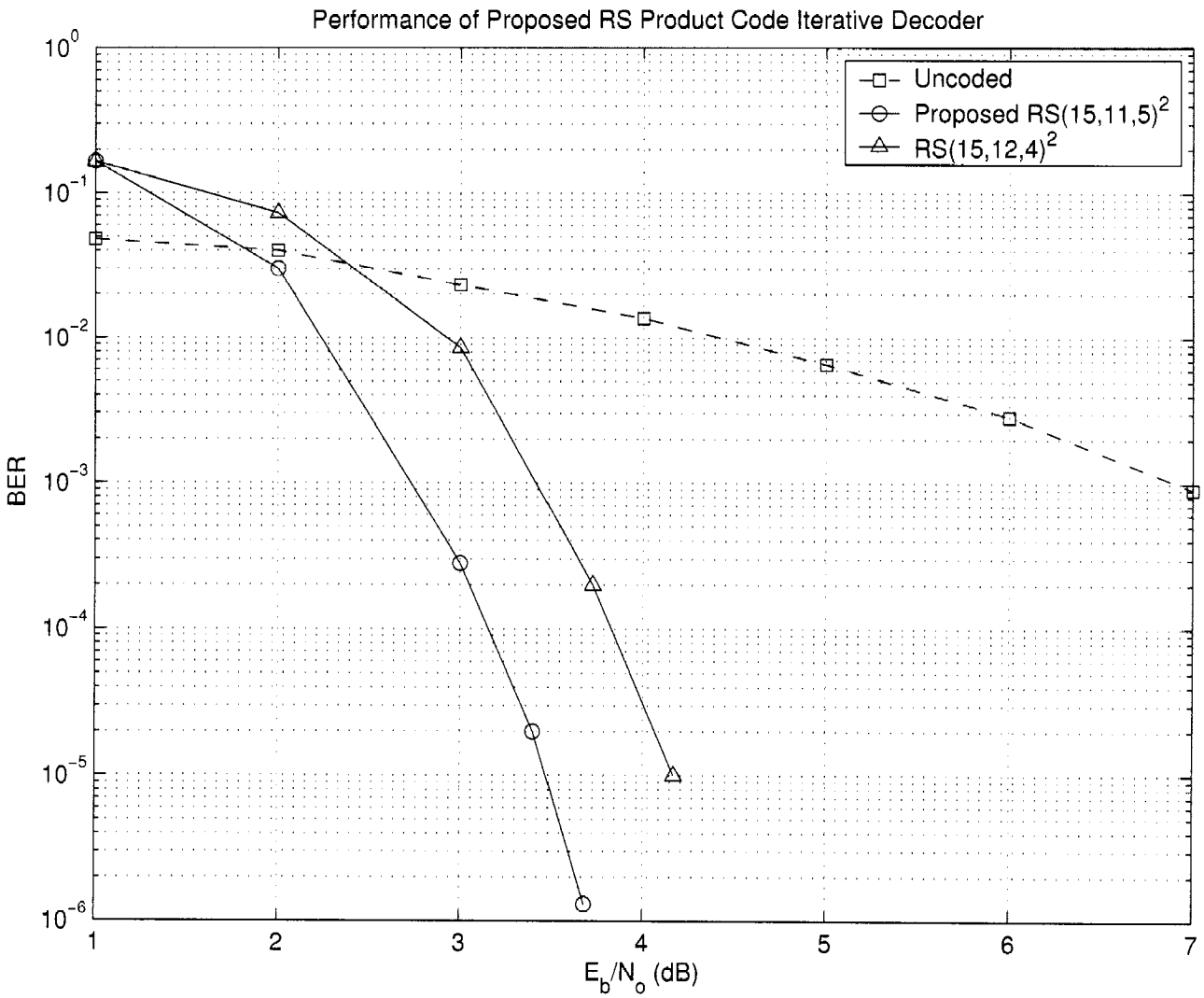

Figure 4.10: Performance of Proposed Iterative Decoder

4.4 Summary

This chapter presented the details of the proposed algorithm. This began with the description of a method to make a maximum likelihood soft-decision decoding algorithm for RS codes suboptimum while, at the same time, maintaining decent performance values. Then, a method for applying this type of suboptimal decoder in the iterative decoding process was also explained in addition to a discussion on the selection of the variable p that determines the number of test patterns to be created and used as competing codewords. We also discussed the importance of not setting a fixed value for p which reduced the overall decoding complexity significantly. Looking at the results obtained for the $(15, 11, 5) \times (15, 11, 5)$ RS product code, we can say that the algorithm is an improvement over already-existing algorithms for similar error-correcting codes.

Chapter 5

Conclusion

In communication systems design, the complexity of the decoder is the main obstacle in our quest to use powerful codes. This is explained by the fact that in most cases, the primary desire is for the communication system to be inexpensive. Using a powerful code in transmission has the advantage that the probability of resending data because of a noisy message is lowered. By decreasing the number of retransmissions, we are effectively minimizing the time needed for reserving the available bandwidth for each transmission. This is especially important when the bandwidth is limited. The major goal of this research was to develop a performant SISO iterative decoder using a secondary decoder. We started by introducing the error-correcting code used throughout this work, namely, the Reed-Solomon Code. An introduction into the existing decoding methods for RS codes was also reviewed, which included both hard and soft decision algorithms. Then, we looked at another error-correcting code formed using RS codes, the product code. The history as well as several meth-

ods currently used to decode these types of error-correcting codes were examined. After having discussed these topics, we were then presented with the problem of incorporating one algorithm into another in order to better the performance of the iterative decoder. This process was composed of two steps: reducing the complexity of the RS soft-decision decoder and incorporating this reduced complexity decoder into the RS Product code iterative decoder.

For the first step, instead of optimally searching for the maximum likelihood coset pattern for each RS codeword, we decided to put a constraint on the amount of levels being checked by the algorithm. Also, a decoding failure was not immediately stated if this level had been reached without satisfying the conditions of the search. Instead, the coset pattern which had been found to have the minimum Euclidean distance up to that point is used for the remainder of the decoding process. The process of placing a constraint on the levels checked showed a degradation in performance of about 0.2 dB when compared to the ML algorithm for the same code. This was deemed an acceptable trade-off between complexity and performance.

For the second step, the major concern was to find a suitable way to create the test sequences using the suboptimal decoder. Although the Chase decoder has been used successfully in the iterative decoding process, it is a hard-decision decoder and could, theoretically be improved upon. By using the suboptimal SD decoder to obtain the test sequences, we would hope to get better performance from the codes. Another important step was in determining the effect of least reliable bits p. We saw that by just setting an upper bound on this value, we are actually taking advantage of the previous decoding steps and thus, saving in terms of complexity. Because of this

method of searching for the least reliable bits in a range, not all the rows and columns need to be re-decoded at each iteration. The decrease in complexity associated with this feature as opposed to using a fixed number of test sequences throughout the entire decoding process is huge. The simulation results for the proposed iterative algorithm of the $(15, 11, 5) \times (15, 11, 5)$ RS product code were seen to exhibit a better performance than a similar code as well as a coding gain after each iteration.

Many parts of this thesis can be extended into future work. It would be useful to investigate how these simulations could be adjusted to perform testing on various other communication channels besides AWGN such as fading channels. In addition, performance of longer and more powerful block codes could be carried out. Another interesting area would be the hardware implementation of the proposed algorithm and the complexity analysis of carrying out such an implementation.

References

[1] *Merriam-Webster's Collegiate Dictionary*, Tenth Edition, Merriam-Webster, Inc., 1998

[2] S.Lin and D.J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, Englewood Cliffs, N.J.: Prentice Hall, Inc., pp. 533-545, 1983

[3] S.Lin and D.J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, Englewood Cliffs, N.J.: Prentice Hall, Inc., pp. 1-3, 1983

[4] C.E. Shannon, "*A Mathematical Theory of Communication*," Bell System Technical Journal, Vol. 27, pp. 379-423, 623-656, October 1948

[5] R.W. Hamming, "*Error detecting and correcting codes*," Bell System Technical Journal, Vol. 29, pp. 147-160, 1950

[6] C. Berrou, A. Glavieux, and P. Thitimajshima, "*Near Shannon limit error-correcting coding and decoding: turbo codes(1)*," in Proc. IEEE Int. Conf. Communications, Geneva, Switzerland, pp. 1064-1070, May 1993

[7] P. Elias, "*Coding for Noisy Channels*," IRE Convention Record, Part 4, pp. 37-46, 1955

[8] J.M. Wozencraft and B. Reiffen, "*Sequential Decoding*," Cambridge, MA: MIT Press, 1961

[9] A.J. Viterbi, "*Error bounds for convolutional codes and an asymptotically optimum decoding algorithm*," IRE Transactions on Information Theory, Vol. IT-13, pp. 260-269, April 1967.

[10] R. Lidl and H. Niederreiter, "*Introduction to Finite Fields and Their Applications*", Revised Edition, Cambridge, England: Cambridge University Press, 1994.

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

[12] I.S. Reed and G. Solomon, "*Polynomial Codes over Certain Finite Fields*," Journal of the Society for Industrial and Applied Mathematics, Vol. 8, pp. 300-304, 1960

[13] R.T. Chien, B.D. Cunningham, and I.B. Oldham, "*Hybrid Methods for finding roots of a polynomial with application to BCH decoding*," IEEE Transactions on Information Theory, Vol. 15, NO. 2, pp. 329-335, 1969

[14] S.Lin and D.J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, Englewood Cliffs, N.J.: Prentice Hall, Inc., pp. 175-176, 1983

[15] J.L. Massey, *Shift Register Synthesis and BCH Decoding*, IEEE Transactions on Information Theory, Vol. IT-15, NO. 1, pp. 122-127, January 1969

[16] S.B. Wicker, *Error Control Systems for Digital Communications and Storage*, Upper Saddle River, NJ, Prentice Hall Inc., 1995

[17] E.R. Berlekamp, *Algebraic Coding Theory*, New York, McGraw-Hill, 1968

[18] I.S. Reed and X. Chen, *Error Control Coding for Data Networks*, Kluwer Academic, 1999

[19] S.A. Tretter, "*Decoding BCH Codes using the Euclidean Algorithm for Finding Greatest Common Divisors*", University of Maryland, ENEE722, pp. 1-8, 1993

[20] G.C. Clark and J.B. Cain, *Error Correction Coding for Digital Communications*, New York: Plenum, 1981

[21] U.Cheng and G.K. Huth, "*Bounds on the bit error probability of a linear cyclic code over $GF(2^l)$ and its extended code*," IEEE Transactions on Information Theory, Vol.34, pp. 776-785, 1988

[22] E.R. Berlekamp, R.E. Peile, and S.P. Pope, "*The application of error coding to communications*," IEEE Community Magazine, Vol. 25, pp. 44-57, 1987

[23] A. Vardy and Y. Be'ery, "*Bit-Level Soft-Decision Decoding of Reed-Solomon Codes*," IEEE Transactions on Communications, Vol. 39, NO. 3, March 1991

[24] D.G. Forney, Jr., "*Generalized Minimum Distance Decoding*," IEEE Transactions on Information Theory, Vol. IT-12, pp. 125-131, April 1966

[25] D. Chase, "A Class of Algorithms for Decoding Block Codes With Channel Measurement Information," IEEE Transactions on Information Theory, Vol. IT-18. NO. 1, January 1972

[26] V. Ponnampalam and B. Vucetic, "*Soft Decision Decoding of Reed-Solomon Codes*," IEEE Transactions on Information Theory, Vol. 50, NO. 11, November 2002

[27] B. Sklar, "*Digital Communications: Fundamentals and Applications*," Second Edition, Upper Saddle River, N.J.: Prentice Hall, pp. 382-388, 2001

[28] P. Elias, Error-free coding, IEEE Transactions on Information Theory, Vol. 4, pp. 29-37, 1954

[29] F.J. MacWilliams and N.J.A. Sloane, *The Theory of Error Correcting Codes*, North-Holland, 1977

[30] A.J. Viterbi, *Convolutional codes and their performance in communication systems*, IEEE Transactions on Communication Technology, COM-19, pp. 751-772, 1971

[31] D.G. Forney, Jr., The viterbi algorithm, IEEE Transactions on Information Theory, Vol. 61, March 1973

[32] A. Vardy, *Trellis Structure of Codes*, In Handbook of Coding Theory, Elsevier Science Publishers, Amsterdam, 2000

[33] L.R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, *Optimal decoding of linear codes for minimizing symbol error rate*, IEEE Transactions on Information Theory, Vol. IT-20, March 1996

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

[35] J. Hagenauer and L. Papke, "Decoding 'Turbo' codes with the soft-output Viterbi algorithm(SOVA)," Proc. International Symposium on Information Theory, Trondheim, Norway, p. 164, June 1994

[36] R. Pyndiah, A. Glavieux, A. Picart and S. Jacq, "*Near-Optimum Decoding of Product Codes*," Proc. IEEE GLOBECOM, San Fransisco, CA, Vol. 1/3, pp. 339-343, November 1994

[37] R. M. Pyndiah, "*Near-Optimum Decoding of Product Codes: Block Turbo Codes*," IEEE Transactions on Communications, Vol. 46, NO. 8, August 1998

[38] O. Aitsab and R. Pyndiah, "*Performance of Reed-Solomon Block Turbo Code*," GLOBECOM, London, UK, Vol. 1, pp. 121-125, November 1996

[39] J. Hagenauer, E. Offer and L. Papke, "*Iterative Decoding of Binary Block and Convolutional Codes*," IEEE Transactions on Information Theory, Vol. 42, NO. 2, March 1996

[40] R. Zhou, A. Picart, R. Pyndiah and A. Goalic, "*Reliable Transmission with Low Complexity Reed-Solomon Block Turbo Codes*," 1st International Symposium on Wireless Communication Systems (ISWCS 2004), Ile Maurice, September 2004

[41] J.M. Wozencraft and I.M. Jacobs, *Principles of Communication Engineering*, New York, John Wiley & Sons, 1965

[42] B. Sklar, "*The Standard Array: A useful tool for Understanding and Analyzing Linear Block Codes*," excerpted from *Digital Communications: Fundamentals and Applications*, Second Edition, Upper Saddle River, N.J.: Prentice Hall, 2001

[43] J.K. Wolf, "*Efficient Maximum Likelihood Decoding of Linear Block Codes Using a Trellis*," IEEE Transactions on Information Theory, Vol. IT-24, NO. 1, January 1978