

UNIVERSITY OF THE WITWATERSRAND

MASTERS DISSERTATION

**Modifications to the Symbol Wise Soft Input
Parity check Transformation Decoding
Algorithm**

Author:

Yuval Odhiambo GENGA

*A Dissertation submitted in fulfillment of the requirements
for the degree of Master of Science*

in the

Centre for Telecommunication Access and Services
School of Electrical and Information Engineering

February 2016



The financial assistance of the Center for Telecommunications Access and Services (CeTAS) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the CeTAS.

Declaration

I, Yuval, Genga Odhiambo, declare that this Dissertation titled, “Modifications to the Symbol Wise Soft Input Parity check Transformation Decoding Algorithm” and the work presented in it are my own. I confirm that:

- ◊ This work was done wholly or mainly while in candidature for a research degree at this University.
- ◊ Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, has been clearly stated.
- ◊ Where I have consulted the published work of others, this is always clearly attributed.
- ◊ Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this dissertation is entirely my own work.
- ◊ I have acknowledged all main sources of help.

Signed:

Date:

UNIVERSITY OF THE WITWATERSRAND

Abstract

Engineering and the Built Environment
School of Electrical and Information Engineering

Master of Science

Modifications to the Symbol Wise Soft Input Parity check Transformation Decoding Algorithm

by Yuval Odhiambo GENGA

Supervisor: Dr. Jaco VERSFELD

Reed-Solomon codes are very popular codes used in the field of forward error correction due to their error correcting capabilities. Thus, a lot of research has been dedicated to the development of decoding algorithms for this class of code. The high performance decoding algorithms for Reed-Solomon codes suffer from high computational complexity. Recently, the development of a new decoding algorithm for Reed-Solomon codes, called the Parity check Transformation Algorithm (PTA), was introduced. The PTA outperformed other high performance symbol-level decoding algorithms for Reed-Solomon codes, and is said to be a less computational complex algorithm. The PTA, however, suffers from running a high number of iterations during the decoding process especially for low SNR regions.

The main focus of this dissertation is to reduce the number of iterations run by the PTA by developing an extension of the algorithm which is referred to as the Modified PTA (MPTA). This will be done by performing an analysis on the PTA algorithm and making modifications which will help meet the objective. This research will also work on improving the symbol error rate performance of the PTA with said modifications. By reducing the number of iteration run by the PTA, this will enable the algorithm to decode faster thus improving on its efficiency while decoding.

I dedicate this to God

*Who has not only given me the ability to come this far,
but has been with me through it all.*

I dedicate this to my Mother

*Mrs Florence Auma Genga, who always believed in me
even when I did not.*

I dedicate this to my Father

*Prof. Riewa. O. Genga, who has sacrificed it all to give
me and my siblings the gift education.*

Acknowledgements

I would like to take this opportunity, with deepest gratitude, to thank my supervisor Dr. D.J.J. Versfeld for the advice and counsel that he has provided through this period. It would have literally been impossible for me to take this work to completion without his incredible support and encouragement. I shall be forever grateful.

I would also like to thank the CeTAS program for the assistance they provided, in terms of finance and the lab equipment.

I would like to express my sincerest gratitude to Dr. Ranier Dreyer for the assistance provided with running of simulations on his machines at the Crunch Yard.

I am extremely grateful to the members of my research group, in particular Olayinka Ogundile, for the support and constructive criticism they have given throughout the course of my research.

I would like to express my sincere thanks to all my friends who have patiently extended all kind of help for accomplishing of this undertaking.

Most importantly, I would like to thank my family for being there for me through it all. I thank them for all the financial and moral support they gave me. To them I am forever grateful.

Contents

Declaration	i
Abstract	ii
Dedication	iii
Acknowledgements	iv
Contents	v
List of Figures	vii
List of Tables	ix
Abbreviations	x
Publications	xi
1 Introduction	1
1.1 Motivation and Problem Statement	2
1.2 Research Objectives	3
1.3 Relevance and Research Contribution	3
1.4 Methodology	4
1.4.1 Experimental setup	5
1.4.2 System description	6
1.5 Organization of Dissertation	6
2 Background Review:	
Forward Error Correction Coding for Reed-Solomon codes	8
2.1 Forward Error Correction Codes	8
2.1.1 Block Codes	9
2.2 Finite Fields for Block Codes	11
2.3 The Generator matrix, The Parity check matrix and The Syndrome of a linear block code	13
2.4 Reed-Solomon Codes	14
2.5 Reed-Solomon Encoding	16
2.6 Reed-Solomon Hard Decision Decoding	18
2.7 Reed-Solomon Soft Decision Decoding	18
2.7.1 Symbol-Level Decoding vs Bit-Level Decoding	19

2.7.2	Guruswami-Sudan Algorithm	19
2.7.3	Koetter-Vardy Algorithm	20
2.7.4	The Adaptive Belief Propagation Algorithm	22
2.7.4.1	Binary Image Expansion	22
2.7.4.2	The Bit-Level ABP Algorithm	24
2.7.4.3	The Symbol-Level ABP Algorithm	28
2.7.5	The Parity check Transformation Algorithm	30
2.8	Summary	34
3	The Modified Soft Input Parity check Tranformation Algorithm for Reed-Solomon Codes	35
3.1	Introduction	36
3.2	The Parity check Transformation Algorithm	37
3.3	Modifications to the PTA algorithm	39
3.3.1	Performance of the PTA based on the number of iterations .	39
3.3.2	Performance of the PTA based on the number of negative values in R	42
3.3.3	Performance of the PTA based on the Stagnant check. . . .	44
3.3.4	Performance of the PTA based on the value of the correction factor, δ	45
3.4	Performance Analysis	53
3.5	Conclusion	53
4	Performance Analysis for Different Code Rates	55
4.1	Introduction	55
4.2	Modification on the Syndrome check stage of the MPTA	56
4.3	Perfomance of the MPTA on low-Rate codes	57
4.4	Perfomance of the MPTA on High-Rate codes	59
4.5	The PTA vs The MPTA for different Code rates	61
4.5.1	Performance comparison for Low rate codes	61
4.5.2	Performance comparison for High-rate codes	64
4.6	Conclusion	66
5	Performance Comparison of the Symbol wise ABP and the MPTA	67
5.1	Introduction	67
5.2	Complexity Analysis	67
5.2.1	Complexity analysis of the Symbol-Level ABP	68
5.2.2	Complexity analysis of the PTA and the MPTA	70
5.2.3	Complexity Comparison of The ABP the PTA and the MPTA	71
5.3	SER performance analysis	73
5.4	Conclusion	76
6	Conclusion and Future Work	78
6.1	Conclusion	78
6.2	Future Work	81

List of Figures

1.1	Summary of the system setup.	6
2.1	Example of RS codeword	15
2.2	Block diagram of the KV algorithm	22
2.3	Symbols from corresponding powers of the primitive elements in bit form.	23
3.1	Number of errors compared to the number of iterations	40
3.2	Comparing the Hamming distance to the Number of Iterations. . .	41
3.3	Performance comparison of different values of I_{max}	42
3.4	Number of errors compared to the number of negative values	43
3.5	Perfomance of the MPTA based on the value of I_{stag}	46
3.6	Perfomance of the $aPTA^\gamma$ based on different values of δ	47
3.7	Performance of the PTA^γ with $\delta = 0.0025$ and $\delta = 0.005$ based on the number of iterations	49
3.8	Performance of the increasing δ MPTA	51
3.9	Performance of the decreasing δ MPTA	52
3.10	Performance of the MPTA compared to the PTA	54
4.1	Performance of different values of I_{stag} with the decreasing δ MPTA on a (15,4) RS code.	57
4.2	Performance of different values of I_{stag} with the increasing δ MPTA on a (15,4) RS code.	58
4.3	Performance of different values of I_{stag} with the single δ MPTA on a (15,4) RS code.	58
4.4	Performance of different values of I_{stag} with the decreasing δ MPTA on a (15,10) RS code.. . . .	60
4.5	Performance of different values of I_{stag} with the increasing δ MPTA on a (15,10) RS code.. . . .	60
4.6	Performance of different values of I_{stag} with the single δ MPTA on a (15,10) RS code.. . . .	61
4.7	Performance comparison for the PTA and the decreasing δ MPTA on a (15,4) RS code.	62
4.8	Performance comparison for the PTA and the increasing δ MPTA on a (15,4) RS code.	63
4.9	Performance comparison for the PTA and the single δ MPTA on a (15,4) RS code.	63
4.10	Performance comparison for the PTA and the decreasing δ MPTA on a (15,10) RS code.	64

4.11	Performance comparison for the PTA and the increasing δ MPTA on a (15,10) <i>RS</i> code.	65
4.12	Performance comparison for the PTA and the single δ MPTA on a (15,10) <i>RS</i> code.	65
5.1	Performance comparison for a (15,4) <i>RS</i> code.	74
5.2	Performance comparison for a (15,7) <i>RS</i> code.	74
5.3	Performance comparison for a (15,10) <i>RS</i> code.	75

List of Tables

2.1	Important field properties used in the encoding and decoding of RS codes	11
5.1	Actual values for the notation used in the complexity Analysis . . .	72
5.2	Maximum number of operations of the ABP, PTA and the MPTA per iteration for Low-rate RS code	72
5.3	Maximum number of operations of the ABP, PTA and the MPTA per iteration for Half-rate RS code	73
5.4	Maximum number of operations of the ABP, PTA and the MPTA per iteration for High-rate RS code	73

Abbreviations

BPSK	B inary P hase S hift K eying
PTA	P arity check matrix T ransformations A lgorithm
MPTA	M odified P arity check matrix T ransformations A lgorithm
AWGN	A dditive W hite G aussian N oise
KV	K oetter- V ardy
LDPC	L ow P arity D ensity C heck
SNR	S ignal-to- N oise R atio
SER	S ymbol E rror R ate
CER	C odeword E rror R ate
BER	B it E rror R ate
ABP	A daptive B elief P ropagation
FEC	F orward E rror C orrection
HDPC	H igh D ensity P arity C heck
LDPC	L ow D ensity P arity C heck
BP	B elief P ropagation
RS	R eed S olomon
GS	G uruswami- S udan
MDS	M aximum D istance S eparable
CD	C ompact D isk
DVD	D igital V ersatile D isk
WIMAX	W orldwide I nterpolation for M icrowave A ccess
GF	G alois F ield

Publications

Overlapping **Peer-reviewed journal papers directly related to this thesis**
chapter

- [3] O. Ogundile **Y. Genga** and D. Versfeld, “Symbol level iterative soft decision decoder for Reed-Solomon codes based on parity-check equations”, *Electronics Letters*, vol. 51, no 17, pp. 1332-1333, 2015.

CHAPTER 1

Introduction

Reed-Solomon (RS) codes are a class of error correction code that were developed in 1960 by Irving S. Reed and Gustave Solomon. For decades since their discovery, there have been countless applications for this class of codes. These applications range from telecommunications to storage devices and even space communications. In the twentieth century, RS codes played an important part in the expansion of the telecommunications revolution. Due to the popularity of these codes, a lot of research has been put into the development of appropriate decoding schemes.

Hard decision decoders, based on the minimum distance principle, have proved to work efficiently with these codes due to the fact that RS codes meet the Singleton bound. Hard decision decoders for RS codes include the Berlekamp-Massey (B-M) algorithm [15][20][2], the Euclidean algorithm [32], and the Berlekamp-Welch algorithm [30]. However, when soft information from the channel is used in decoding, a significant performance gain is exhibited when compared to hard decision decoding techniques. Soft decision decoding has been shown to experience a gain of about 2-3 dB when in an AWGN channel [31]. The Koetter and Vardy (KV) algorithm [11] is a popular symbol-level soft decision decoding algorithm for RS codes, as it significantly outperforms all previous hard decision decoding algorithms especially for low rate codes. However, in order to achieve these large coding gains, the computational complexity of the KV algorithm can be prohibitively large.

Most decoding algorithms of RS codes work on a symbol-level due the fact all RS codes are non-binary in nature. However, in an attempt to implement Belief Propagation on RS codes a bit-level decoder, referred to as Adaptive Belief Propagation (ABP) algorithm, was proposed by Jiang and Narayanan in [9]. The ABP

algorithm applies binary image expansion on the dense parity-check matrix so as to make it sparse. This helps with the implementation of Belief Propagation (BP) which has been shown to work well with sparse matrices in the form of Low Density Parity Check codes (LDPC) [14]. From the results obtained in [9] the ABP algorithm is shown to be a very powerful algorithm as it significantly outperforms the KV algorithm, but once again this is at the expense of computational complexity.

In [18] a symbol-level soft decision decoding algorithm, referred to as the Parity-check matrix Transformation Algorithm (PTA), was developed. The PTA uses the symbol reliability of the soft information in the reliability matrix to transform the parity-check matrix of a RS code. The algorithm then performs syndrome checks on each row of the transformed parity check matrix iteratively. The PTA algorithm is less computationally complex than the KV algorithm and has been shown in [18] to give a better symbol error rate (SER) performance. This makes the PTA a viable decoding algorithm for RS codes. The major drawback of the PTA is that it runs for many iterations, especially when a good performance is required.

1.1 Motivation and Problem Statement

The computational complexity of an algorithm can be defined as the number of operations the algorithm performs during one iteration. Good SER performance versus computational complexity is a common trade-off with most of the soft decision decoding algorithms used for RS codes. The PTA, however, outperforms other high performance symbol-level decoding algorithms at a lower computational complexity [18]. As stated in the previous section, this improved performance is attained at the cost of the PTA algorithm running numerous iterations especially for low SNR values. This is mainly attributed to the value of the correction factor (δ), used in the algorithm. A smaller correction factor improves the performance of the algorithm. This, however, comes at the cost of an increased number of iterations (this is explained in detail in Section 3.3.4. as well as [18]). In this regard,

research done in this dissertation is aimed at reducing the number of iterations of the PTA so as to improve the efficiency of the algorithm while maintaining good performance. Research is also done to improve the SER performance of the PTA algorithm.

Based on this, the research question for this dissertation is:

- *Can the number of iterations run by the PTA algorithm be reduced without the algorithm suffering from a significant SER performance loss?*

1.2 Research Objectives

The main objectives for the research presented in this dissertation are:

1. To reduce the number of iterations required to decode by the PTA algorithm.
2. To reduce the computational complexity of the PTA algorithm.
3. To develop an efficient early stopping condition that will terminate the algorithm if it is predicted, with high probability, that a decoding error will occur.
4. To implement modifications to the existing PTA algorithm so as to improve the SER performance of the algorithm without any additional computational complexity. These modifications shall be used to develop an extension of the PTA algorithm that runs for fewer iterations without a significant SER performance loss.

1.3 Relevance and Research Contribution

Reed-Solomon codes form a very important class of error correcting codes largely due to their error detecting capabilities. When correcting symbol errors for t check symbols of data, this class of code can correct up to $\lfloor t/2 \rfloor$ symbols. When dealing

with erasure codes, RS codes can correct up to t erasures [12]. Due to this attributes RS codes have several important applications which include digital media devices such as CDs and DVDs. They also have applications in data telecommunication technologies such as WIMAX and several broadcast systems. RS codes also have applications in satellite and space communications.

The research contribution is as follows

1. A new early stopping condition based on the syndrome checks is proposed for the PTA. The early stopping condition is referred to as the stagnant check.
2. A variable discreet approach is proposed for the implementation of the correction factor, δ , in the PTA.
3. An analysis of the performance of PTA algorithm based on the number of iterations, number of negative values in the reliable matrix and the rate of the code is done.

1.4 Methodology

The main objective of this research is to reduce the number of iterations required by the PTA to decode. This is done by implementing modifications to the algorithm. To be able to use the right techniques to modify the algorithm, an analysis of the PTA is done. The experimental set up required for this analysis should be able to give a performance comparison based on the variation of specified parameters used in the algorithm.

The performance of the PTA and its proposed modification is done by running computer simulations in MATLAB. The performance of the algorithm is determined based on two factors:

1. The Symbol Error Rate (SER).
2. The average number of iterations required by the algorithm to decode.

These two factors are tested across a range of signal to noise ratios (SNR). For the case of SER, a difference of less than 1 dB from the best performing system is considered adequate, and justifies use of the algorithm.

1.4.1 Experimental setup

The *RS* code is constructed in a field of $GF(16)$. The code is then broken down to its bit representation and transmitted over an AWGN channel using BPSK modulation with unit signal energy. The BPSK modulation transmits a bipolar sequence, where -1 represents a 0 and +1 represents a 1 [25]. A reliability matrix is constructed using the same technique as shown in [14], from the received vector at the output of the channel. The reliability matrix is what we feed into the PTA algorithm to be able to perform the analysis. The construction and transmission of the code is done using the same computer simulation software, MATLAB. The analysis is done based on the following parameters

- The number of iterations required to decode.
- The effect of negative values being updated in the reliability matrix.
- The performance of the algorithm with the new early stopping condition, based on different check points.
- The performance of the algorithm based on different values of the correction factor.

From the results obtained on the analysis of these parameters, modifications are made to the algorithm with the aim of meeting the primary objective of the research.

This set up is used to test the performance of the algorithm for different code rates. An analysis based on the SER performance is then given for different code rates with the Modified PTA. This is done to find the best performing Modification of

the PTA. From the selection of the Modified PTA, a comparison study is done between the Modified PTA, the PTA and the ABP.

1.4.2 System description

The system used to perform the analysis and comparison of the PTA and the Modified PTA consists of a source. This is where the random input data is generated. An *RS* encoding block is then used to encode the random data from the source. The encoded data is then modulated and AWGN noise added to the system. The remaining blocks required include the BPSK demodulation block which send data the PTA decoder and the Modified PTA decoder. The decoded data is then analyzed at the output. A block diagram of the summary of the system setup is depicted in Fig. 1.1.

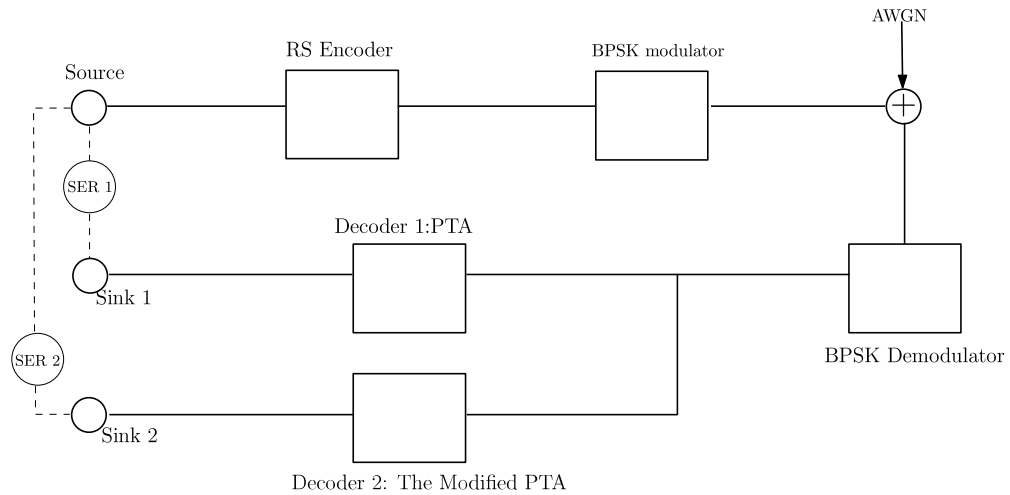


FIGURE 1.1: Summary of the system setup.

1.5 Organization of Dissertation

The rest of the dissertation is organized as follows.

In Chapter 2, a basic overview of the principles of forward error correction (FEC) is presented. A summary on Reed-Solomon codes is also presented in Chapter 2.

Chapter 2 also reviews soft decision decoding techniques for Reed-Solomon codes. From the soft decision decoders, the Adaptive Belief Propagation (ABP) algorithm [9] and The Parity check Transformation Algorithm (PTA) [18] are discussed in detail.

Chapter 3 focuses on the analysis of the PTA algorithm. Modifications to the PTA are proposed based on the analysis done and simulations run to test for the SER performance and the average number of iterations of these modifications. The results from this simulations are used in the development of the Modified PTA (MPTA). A comparative study is done to test for the SER performance and the average number of iterations of the MPTA and the PTA.

Chapter 4 proposes a modification to reduce the computational complexity of the MPTA. A comparative study based on the SER performance for different code rates is also done in this chapter for the PTA and the MPTA.

In Chapter 5, a complexity analysis between the Symbol-level ABP, the PTA and the MPTA decoding algorithm is presented. This is done for different code rates as well. The SER performance is also compared for Symbol-level ABP, the PTA and the MPTA.

Chapter 6 presents a summary of the research done in this dissertation. This summary is based on how the results from the simulations relate to the objectives of the research.

CHAPTER 2

Background Review:

Forward Error Correction Coding for Reed-Solomon codes

This chapter serves as the basis of the core literature used for this research. With this regard, a lot of emphasis is on existing work done and the reviewing of Reed-Solomon codes. A brief introduction into the forward error correction (FEC) schemes used in this dissertation is given in Section 2.1. Reed-Solomon (*RS*) codes are reviewed in Section 2.4. Hard decision decoding techniques used on *RS* codes are revised in Section 2.6. Section 2.7 mainly deals with *RS* soft-decision decoders, with a focus on the Adaptive Belief Propagation (*ABP*) as well as the Parity check Transformation Algorithm (PTA). Finally, Section 2.8 summarizes the chapter.

2.1 Forward Error Correction Codes

Forward error correction (FEC) is an error control strategy used in communication systems, deployed from the transmitter to the receiver. FEC, error correcting codes (examples include: Hamming codes, *RS* codes, Cyclic codes, LDPC codes, Turbo codes, etc) together with decoding algorithms are used to detect and correct errors that were introduced during transmission through a noisy communication channel [12]. Error detection and correction is achieved by transmitting additional symbols (parity symbols) together with the information symbols. These parity symbols allow the receiver to detect and possibly correct the error(s) that may

occur at the receiver of communication systems. There are two main types of FEC codes used in controlling errors in communication systems, namely: Block codes and Convolution codes. The major difference between these two categories of error correcting codes is the principle used in the encoding process. In block codes, the parity symbols usually come either after or before the information symbols to form a codeword of fixed length. This for a case of systematic block codes. For nonsystematic block codes, the codeword is comprised of a sequence of symbols of fixed length generated from an encoded message. On the other hand, Convolution codes spreads the information symbols along the sequence; that is, the information symbol sequence is sequentially convolved conforming to some rules [12, 17]. Research done in this chapter focuses on decoding of *RS* codes. With this regard, attention is focused on block codes; as *RS* codes are characterized by codewords with a fixed length.

2.1.1 Block Codes

Block codes are a class of forward error correcting codes that encode information data in blocks. In simple form, block codes have a sequence of k symbols at the input and a sequence of n symbols at the output. In order to add parity symbols, the length n is always greater than the length of k . These kind of codes are also denoted as (n, k) block codes. Examples of block codes include Hamming codes, *RS* codes, and LDPC codes, etc. These codes also belong to the family of linear codes; thus, they are referred to as linear block codes. In general, block codes are defined in terms of the following important parameters.

Definition 2.1: The Hamming distance is the total number of positions in which two codewords of the same length differ.

Definition 2.2: The minimum Hamming distance, d_{min} , of a block code is defined in terms of [12]:

$$d_{min} = \min\{d_h(c_1, c_2) : c_1, c_2 \in \mathcal{C}, c_1 \neq c_2\}, \quad (2.1)$$

where the notation $d_h(c_1, c_2)$ denotes the Hamming distance between any two codewords c_1 and c_2 of a code $\mathcal{C}$. The minimum distance d_{min} determines the number of errors that the code can detect or correct. In general, a code can reliably detect:

$$\varepsilon \leq d_{min} - 1 \quad (2.2)$$

errors.

Definition 2.3: For an (n, k) block code, k denotes the number of message symbols.

Definition 2.4: Likewise, n denotes the total length of a codeword of an (n, k) block code.

Definition 2.5: The parity symbols, h , of a block code are the number of redundancy symbols per transmitted codeword. It is represented as:

$$h = n - k. \quad (2.3)$$

The higher the value of h the better the error detection and correcting capability of the code. Although, this means the number of information symbols transmitted per codeword is reduced, given a fixed length n .

Definition 2.6: The code rate R of a block code is the ratio of the information symbols k to the codeword symbol n . It is represented as:

$$R = \frac{k}{n}. \quad (2.4)$$

From (2.4), it can be seen that more information symbols can be transmitted for higher code rates. Most times, the smaller the value of R the better the error detection and correcting capability of the code.

Definition 2.7: Maximum Distance Separable (MDS) codes are linear block whose minimum distance, d_{min} , is defined as

$$d_{min} = n - k + 1. \quad (2.5)$$

These codes are able to correct the largest number of errors for a given code rate R . MDS codes also have the property

$$n - k = 2t, \quad (2.6)$$

where t is defined as the number of correctable errors per transmitted codeword, and can be expressed as

$$t = \left\lfloor \frac{d_{min} - 1}{2} \right\rfloor. \quad (2.7)$$

Reed-Solomon codes form the most essential class of MDS codes [12].

2.2 Finite Fields for Block Codes

Definition 2.8: A field $\mathbb{F}$ is a set of elements where by the addition, subtraction, multiplication and division operations can be performed with the result remaining in the set. The addition “+” and multiplication “.” operations for any three element x , y , and z in a field $\mathbb{F}$ must satisfy the commutative, associative distributive, identity, and inverse properties as defined in Table 2.1.

TABLE 2.1: Important field properties used in the encoding and decoding of *RS* codes

Properties	Addition	Multiplication
Associative	$(x + y) + z = x + (y + z)$	$(x.y).z = x.(y.z)$
Commutative	$x + y = y + x$	$x.y = y.x$
Distributive	$x.(y + z) = (x.y) + (x.z)$	$(x.y) + (x.z) = x.(y + z)$
Identity	$x + 0 = x = 0 + x$	$x.1 = x = 1.x$
Inverse	$x + (-x) = 0 = (-x) + x$	$x.x^{-1} = 1 = x^{-1}.x$, if $x \neq 0$

Definition 2.9: A Galois Field GF is a field that contains a finite number of elements. The GF named after Evariste Galois is usually denoted as $GF(q^m)$ or $\mathbb{F}_{q^m}$, where m is a whole, non-negative number and q is a prime number. Multiplication and addition operations are performed using modulo- q operation when m is equal to one ($m = 1$). In this case, the GF is regarded as a prime field ($\mathbb{F}_q$) [12]. When

the value of m is greater than one ($m > 1$), elements of the field are delineated as polynomials of degree m with coefficients from the $\mathbb{F}_q$. In this scenario, the GF is regarded as a prime extension field ($\mathbb{F}_{q^m}$) [12]

Work done in this dissertation is aimed at decoding of RS codes, which are non-binary codes. Therefore, this research assumes that $m > 1$, specifically $m = 4$; hence, the GF in this dissertation are represented in prime extension field ($\mathbb{F}_{q^m}$). This means the construction of $\mathbb{F}_{q^m}$ for $m > 1$, requires specific polynomial of degree $m - 1$. Thus, the elements and polynomial related to the construction and decoding of the RS codes used in this research are defined.

Definition 2.10: The primitive element of the field $GF(q^m)$ is a nonzero element, α , in the field with the order $q^m - 1$. That is, the powers of the primitive element, α , can be used to generate all the nonzero elements in the field [12].

Definition 2.11: A monic polynomial $f(x)$ is a polynomial with coefficients from $\mathbb{F}_q$, with the additional condition that the coefficient of the highest power of x is equal to one. For example:

$$f(x) = x^4 + x + 1, \tag{2.8}$$

is a monic polynomial.

Definition 2.12: An irreducible polynomial of degree m over $\mathbb{F}_q$ is a polynomial such that it is not divisible by any polynomial of degree lower than m though greater than zero over $\mathbb{F}_q$.

Definition 2.13: A primitive polynomial $p(x)$ is an irreducible polynomial of degree m over $\mathbb{F}_q$ provided that the smallest positive whole number n for which $p(x)$ divides $x^n - 1$ is $n = q^m - 1$.

Definition 2.14: A bivariate polynomial is a polynomial with two variables. Bivariate polynomials are mathematically represented as:

$$F(x, y) = \sum_{u,v} J_{u,v} x^u y^v. \quad (2.9)$$

2.3 The Generator matrix, The Parity check matrix and The Syndrome of a linear block code

The generator matrix for an (n, k) linear block code $\mathcal{C}$ is defined as a matrix whose rows form the basis of the code [25]. Assuming there exists a message $\mathcal{M}$ of length k , then, a codeword c of the linear block code $\mathcal{C}$ can be generated as

$$c = \mathcal{M} \cdot G. \quad (2.10)$$

The matrix G is made up of k linearly independent codewords in the block code $\mathcal{C}$. Due to this row structure of the G matrix, every codeword c generated from $\mathcal{C}$ is a linear combination of this linearly independent codewords [25].

The general structure of a generator matrix is in its systematic form, and is expressed as

$$G = [I_k P], \quad (2.11)$$

where I_k represents an identity matrix of the dimensions $k \times k$, and P is referred to as the parity submatrix, with the dimensions $k \times (n - k)$.

The parity check matrix H of an (n, k) linear block code $\mathcal{C}$ is defined as the dual code of the generator matrix [25]. This basically means that a codeword c can only exist in $\mathcal{C}$ if and only if the constraint

$$c \cdot H^T = 0 \quad (2.12)$$

is satisfied [12]. Here H^T represents the transpose of the H matrix. The H matrix has the dimensions $(n - k, n)$ and is usually expressed in systematic form as

$$H = [P^T I_{(n-k)}], \quad (2.13)$$

where I_{n-k} represents an identity matrix with the dimensions $(n - k) \times (n - k)$, and P^T is the transpose of the parity submatrix used in the G matrix and is of the dimensions $(n - k) \times k$. The matrix H and G can only exist for a given linear block code $\mathcal{C}$ if and only if

$$G \cdot H^T = 0. \quad (2.14)$$

The syndrome of a linear block code is a constraint that is satisfied using (2.12). If the constraint is met, the syndrome forms a zero vector, S , of length $(n - k)$. The computation of the Syndrome can also be done by taking the dot product of the codeword c with each of the individual rows of the parity check matrix. For $1 \leq i \leq (n - k)$ (2.12) can be rewritten as

$$S_i = c \cdot H_i, \quad (2.15)$$

where H_i is the i^{th} row of H . Therefore, from (2.15), in order for a codeword c to exist in $\mathcal{C}$, (2.16) must be satisfied for each row of the parity check matrix.

$$S_i = \sum_{j=1}^n (c_j \cdot H_{i,j}) = 0. \quad (2.16)$$

2.4 Reed-Solomon Codes

RS codes are a set of cyclic codes introduced in [21] by Irving S. Reed and Gustave Solomon. They are Non-binary Bose, Chaudhuri and Hocquenghem (BCH) codes. *RS* codes are used in channel coding to correct errors over a noisy communication channel. They are widely used in correcting random symbol errors, burst errors,

and erasures in digital communication and data storage systems [12]. *RS* codes are q -ary linear block code based on finite fields or Galois Fields (GF). Similar to other error correcting codes, the transmitted message in *RS* codes is also divided into separate fixed block of data.

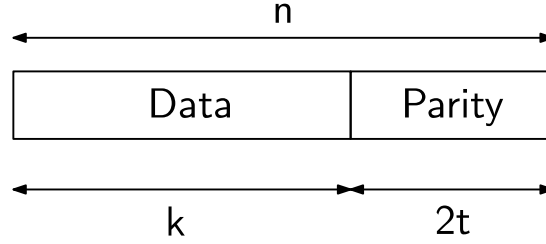


FIGURE 2.1: Example of *RS* codeword

As shown in Fig. 2.1, $h = 2t$ parity symbols, are added to each data block to form an *RS* codeword. This means that the encoder takes in k symbols and adds $(n - k = 2t)$ parity symbols to give a (n, k) *RS* codeword of fixed length n . With the same fixed block lengths at the encoder input and output, *RS* codes attain the largest feasible code minimum Hamming distance for any linear block code [3]. In general, an *RS* code with symbols from the field $GF(q^m)$ is defined by the following parameters [12]:

Codeword length, $n = q^m - 1$,

Length of information symbols, $k = n - 2t$,

Length of parity symbols, $h = n - k = 2t$,

Minimum distance, $d_{min} = 2t + 1$,

where q^m represents the number of elements in the field. From the aforementioned *RS* parameters, two cardinal features are deduced [12]:

1. The length of *RS* codes is less than the size of the code alphabet, q^m , by one.
2. The minimum distance d_{min} is greater than the number of parity symbols h by one.

These features indicate that the minimum distance, d_{min} , of a *RS* code is equal to the number of its parity symbols h plus one. Therefore, *RS* codes meet the Singleton bound with equality; that is, it achieves the largest minimum distance possible [3].

RS codes can be completely defined using their generator polynomials.

Definition 2.15: The generator polynomial $g(x)$ of a (n,k) cyclic code is a monic polynomial that has a unique minimal degree of $n - k$ [17]. The generator polynomial can be expressed as

$$g(x) = 1 + g_1x + g_2x^2 + \dots + g_{n-k-1}x^{n-k-1} + x^{n-k}. \quad (2.17)$$

From (2.17), a given codeword polynomial $c(x)$ can be defined as [12].

$$c(x) = \mathcal{M}(x)g(x) = (\mathcal{M}_0 + \mathcal{M}_1x + \dots + \mathcal{M}_{k-1}x^{k-1})g(x), \quad (2.18)$$

where $\mathcal{M}(x)$ represents the polynomial for the message symbols. From (2.18), encoding is done by multiplying the generator polynomial with the message polynomial. Thus, a given (n, k) cyclic code is completely defined by the nonzero code polynomial of minimum degree $g(x)$.

2.5 Reed-Solomon Encoding

The encoding of *RS* codes can be achieved in such a way that the encoded message forms a systematic code or non-systematic code. In its systematic form, the message polynomial is contained in the codeword as shown in Fig. 2.1. In the case of a non-systematic code, the message polynomial is not contained in the codeword. It is however desirable to encode the message polynomial using the systematic approach. This is because the decoder returns the transmitted message part as it was received when there is a decoding failure. The message symbols may contain errors; notwithstanding, the systematic encoding approach avoids losing the whole transmitted message. The systematic encoding of *RS* codes is described as follows.

Given that k information symbols is represented by a message polynomial $\mathcal{M}(x)$ of degree $k - 1$, such that:

$$\mathcal{M}(x) = \mathcal{M}_0 + \mathcal{M}_1x + \mathcal{M}_2x^2 + \dots + \mathcal{M}_{k-1}x^{k-1}, \quad (2.19)$$

where the coefficients $\mathcal{M}_0, \mathcal{M}_1, \mathcal{M}_2, \dots, \mathcal{M}_{k-1}$ are m -bit message symbols which implies an element of $GF(q^m)$.

The message polynomial is first scaled by a factor x^{n-k} as:

$$\bar{\mathcal{M}}(x) = \mathcal{M}(x).x^{n-k}, \quad (2.20)$$

the result is subsequently divided by the generator polynomial $g(x)$ which gives a quotient $\bar{q}(x)$ and a remainder $z(x)$ as defined by (2.21) [17]:

$$\bar{\mathcal{M}}(x) = \bar{q}(x)g(x) + z(x). \quad (2.21)$$

Therefore, the systematic codeword $c(x)$ is formed by combining the message polynomial $\mathcal{M}(x)$, the scaling factor x^{n-k} , and the remainder $z(x)$ as:

$$c(x) = \mathcal{M}(x).x^{n-k} + z(x). \quad (2.22)$$

It shows from (2.22) that the generated codeword embeds the message symbols and redundant symbols called the parity symbols.

For non systematic encoding of RS codes, the codeword c can be obtained using a generator matrix G of the form

$$G = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & \alpha^1 & \dots & \alpha^{(n-1)} \\ 1 & \alpha^2 & \dots & \alpha^{2(n-1)} \\ \vdots & \vdots & \dots & \vdots \\ 1 & \alpha^{(k-1)} & \dots & \alpha^{(k-1)(n-1)} \end{bmatrix}, \quad (2.23)$$

where α denotes the primitive element of the field.

This structure of the G is referred to as a Vandermonde matrix. The codeword c is obtained by encoding the message $\mathcal{M}$ as shown in (2.10). The corresponding parity check matrix H in the Vandermonde matrix form is

$$H = \begin{bmatrix} 1 & \alpha^1 & \dots & \alpha^{(n-1)} \\ 1 & \alpha^2 & \dots & \alpha^{2(n-1)} \\ \vdots & \vdots & \dots & \vdots \\ 1 & \alpha^h & \dots & \alpha^{h(n-1)} \end{bmatrix}. \quad (2.24)$$

2.6 Reed-Solomon Hard Decision Decoding

In general, the conventional hard decision algorithms for RS codes can correct errors up to:

$$t = \left\lfloor \frac{d_{min} - 1}{2} \right\rfloor = \left\lfloor \frac{n - k}{2} \right\rfloor. \quad (2.25)$$

This logically means that the decoder finds the Hamming sphere of radius t for the codeword centered at the received vector [16]. The decoder only reports a decoding success if the sphere seats a unique codeword, otherwise a decoding failure results. RS decoding algorithms that strictly adhere to this classical t error correcting bounds are referred to as Minimum distance decoding (MDD) algorithms. Examples of RS Hard decision decoders based on the minimum distance includes the Berlekamp-Massey ($B-M$) algorithm [2, 15], the Euclidean algorithm [32], and the Berlekamp-Welch algorithm [30].

2.7 Reed-Solomon Soft Decision Decoding

Decoding beyond the t error correction bound is possible using soft decision decoders. In this case, the decoder uses reliability information from the channel output to decode the received vector.

2.7.1 Symbol-Level Decoding vs Bit-Level Decoding

Different RS symbol-level and bit-level soft decision decoding algorithms have been proposed in literature, see for instance [1, 9, 11, 26]. Work done in [9, 26, 28] presents literature on some examples of bit-level decoding algorithms based on RS soft decision techniques. Examples of RS soft decision decoding algorithms done on a symbol-level can be found in [11, 18] .

For the field $GF(2^m)$ every symbol can be represented in the form of m -bits. The arithmetic of the bit representation can be done over the field $GF(2)$, as the field $GF(2^m)$ is an extension of this field. Bit-level decoders first convert the codeword of length n into its bit form of length nb , where $nb = n \times m$. The bit representation of the codeword is then used in the decoding process. Symbol-level decoders work on the $2^m - 1$ symbols of the elements in the field during the decoding process. Bit-level decoders have been shown in literature to outperform symbol-level decoders for RS codes [5, 9, 10, 27]. This is usually at the cost of complexity as the bit-level decoders work with longer codes and larger parity check matrices when compared to symbol-level decoders. More operations are also required with bit-level decoders as conversion from symbol to bits is done.

2.7.2 Guruswami-Sudan Algorithm

The Guruswami-Sudan (GS) algorithm is a symbol-level decoding algorithm based on the fact that it is possible to correct beyond the classical t error correction bound.

Briefly, the GS decoder works as follows [16]. Given that the code c is the codeword defined as:

$$c = (f(\hat{\alpha}_1), f(\hat{\alpha}_2), \dots, \dots, f(\hat{\alpha}_n)), \quad (2.26)$$

and the received vector r is defined as:

$$r = (\hat{\beta}_1, \hat{\beta}_2, \dots, \dots, \hat{\beta}_n), \quad (2.27)$$

where $f(\hat{\alpha})$ represents the n^{th} message polynomial in the codeword c and $\hat{\beta}$ represents the n^{th} symbol of the received vector r .

Let $f(x)$ be a polynomial of degree $k - 1$ which maps to a Reed-Solomon codeword with Hamming distance less or equal to the designed decoding radius ($t_{\mathfrak{M}}$) from the received vector r , where $\mathfrak{M}$ ($\mathfrak{M} \geq 1$) is an adjustable integer parameter in the *GS* algorithm called the interpolation multiplicity. Firstly, the *GS* decoder constructs a bivariate polynomial of the form [16]:

$$F(x, y) = \sum_{u,v} J_{u,v} x^u y^v, \quad (2.28)$$

where F has a zero of interpolation multiplicity $\mathfrak{M}$ at each of the points ($x = \alpha_u, y = \beta_u$). Also, the value of the $(1, k - 1)$ weighted degree of $F(x, y)$ is the smallest. This step of the *GS* algorithm is referred to as the *interpolation step* and is implemented using the Feng-Tzeng Interpolation Algorithm [6, 16].

Secondly, the *GS* decoder looks for all factors of $F(x, y)$ by finding a set of polynomials $b(x)$ such that $(y - b(x)) | F(x, y)$ [17]. This step of the *GS* algorithm is referred to as the *factorization step*. The factorization step is implemented using the Roth-Ruckenstein Factorization Algorithm [22, 23]. The output of the *GS* factorization step gives a list of polynomials (codewords). The list should contain the transmitted codeword. As such, the *GS* decoder is referred to as a list-decoding algorithm. The *GS* decoder provides a computational complexity of $(\mathcal{O}(n^2 \mathfrak{M}^4))$ [16].

2.7.3 Koetter-Vardy Algorithm

The Koetter-Vardy (*KV*) algorithm is a symbol-wise soft-decision algebraic decoder for *RS* codes that was developed in [11]. The algorithm extends the Guruswami-Sudan list-decoding algorithm by modifying the multiplicities used as the input to the *GS* algorithm. The modified multiplicities are calculated by assigning reliabilities proportional to that of the received symbols [7]. The *KV* algorithm gives a performance-complexity trade-off especially for cases when the length n of the

code is large.

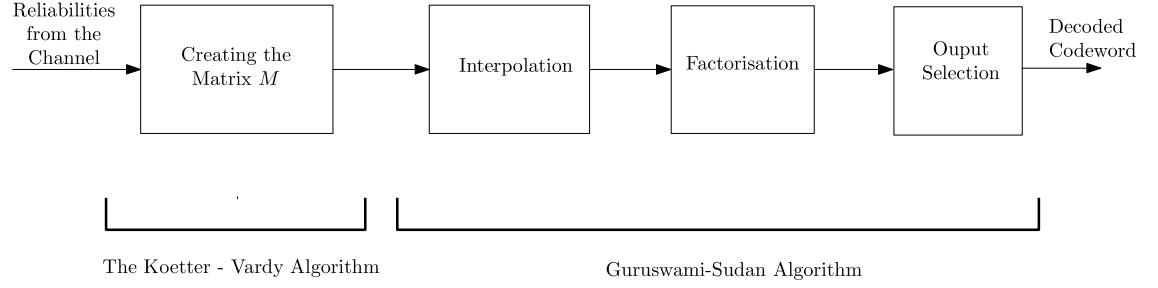
For the field $\text{GF}(q^m)$, a matrix P of dimensions $q^m \times n$ is fed into the KV decoder. The matrix P is made up of probabilities p_{ij} , and the columns of this matrix sum to unity. For $0 \leq i \leq q$ and $0 \leq j \leq n$, the notation p_{ij} represents the probability that the i^{th} symbol of the field is equal to the j^{th} element of the codeword. The values of p_{ij} are obtained from the soft information of the symbols transmitted through a memoryless channel. Assume the inputs and outputs to the channel are random variables S and T . The most likely value of the soft information of the symbol t_i , given that symbol s_i was transmitted, is the *a-posteriori* probability

$$p_{ij} = P(S = s_i | T = t_i). \quad (2.29)$$

The values of p_{ij} are not always probabilities and are sometimes referred to as reliabilities. With regard to this, the P matrix is commonly referred to as the reliability matrix. The P matrix is constructed from values of p_{ij} obtained using various techniques depending on the modulation scheme. With BPSK, the method shown in [14] can be used, for QAM the Euclidean distance method shown in [19] can be used, and for a general case the Bayesian probability techniques in [4] can be used.

The KV algorithm then translates the soft information from matrix P into a multiplicity matrix M , based on a set of weighted points. The matrix M is the same size as matrix P . The algorithm performs soft decision decoding on the P matrix, by allocating unequal multiplicities to various points corresponding to their relative reliabilities [7].

Once the KV algorithm has assigned all the multiplicities, the matrix M is then fed into the Guruswami-Sudan decoding algorithm. A summary of how the $KV - GS$ algorithm works is illustrated in Fig. 2.2 [7].

FIGURE 2.2: Block diagram of the KV algorithm

2.7.4 The Adaptive Belief Propagation Algorithm

Belief Propagation (BP) decoding techniques have been shown to give a near Shannon limit performance when used to decode Low Density Parity Check (LDPC) codes, defined over sparse matrices [14]. However, BP decoding techniques have also been shown to experience a considerable loss when applied to codes defined by High Density Parity Check (HDPC) matrices (like that of RS codes) [13]. This is because simulations with iterative decoders have shown unreliable bits saturate most of the checks due to the presence of many short cycles [9]. This causes the iterative decoder to get stuck at pseudo equilibrium points. As a result, the Adaptive Belief Propagation (ABP) algorithm for RS codes was developed in [9], with the aim of applying BP techniques to HDPC codes. The ABP algorithm has been shown to outperform the widely used KV algorithm, but this comes at the expense of computational complexity. The ABP algorithm was developed to work at both a bit-level and a symbol-level. To better understand how the ABP works, binary image expansion is explained.

2.7.4.1 Binary Image Expansion

For BP to work on a class of codes defined by a HDPC matrix, a binary image expansion is done on the parity check matrix to make it sparse. In this section, binary image expansion of RS codes is explained.

Consider an RS code over the field $GF(2^m)$. All $2^m - 1$ elements in this field can be defined using the powers of the primitive element α . The order in which

the elements span the field is of the form of $\alpha^\infty = 0, \alpha^0 = 1, \alpha^1, \alpha^2, \dots, \alpha^{2^m-2}$. Each element in the field can be expanded to form a m -dimensional binary vector over $GF(2)$. This binary representation is done on the symbols of the codeword c of the RS codes. Similarly, each entry in the parity check matrix H can be expressed in the form of a $m \times m$ matrix over $GF(2)$, using the basis which spans the elements in the field. To better explain this, an example over the field $GF(2^3)$ is now illustrated.

For the field $GF(2^3)$ every symbol has 3 bits and can be expressed in the order that span the field as shown in Fig. 2.3.

α^0	α^1	α^2	α^3	α^4	α^5	α^6
1	2	4	3	6	7	5
1	0	0	1	0	1	1
0	1	0	1	1	1	0
0	0	1	0	1	1	1

FIGURE 2.3: Symbols from corresponding powers of the primitive elements in bit form.

We now consider the parity check matrix, H , for a $(7,5)$ RS code in a Vandermonde matrix form.

$$H = \begin{bmatrix} \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \\ \alpha^0 & \alpha^2 & \alpha^4 & \alpha^6 & \alpha^1 & \alpha^3 & \alpha^5 \end{bmatrix}. \quad (2.30)$$

The $GF(2^3)$ symbol representation of the H matrix is

$$H = \begin{bmatrix} 1 & 2 & 4 & 3 & 6 & 7 & 5 \\ 1 & 4 & 6 & 5 & 2 & 3 & 7 \end{bmatrix}. \quad (2.31)$$

The binary image expansion works on the symbols in the H matrix based on the order in which the elements span the field as shown in Fig. 2.3. How this works is that a $m \times m$ matrix representation for each symbol is formed based on the bit representation of the required symbol and the bit representation of the next $m - 1$ symbols from the order shown in Fig. 2.3. For example, to perform a binary image expansion on the symbol α^0 , the columns of the $m \times m$ matrix is formed using the bit representation for the symbols of α^0 , α^1 and α^2 , and for the binary image

expansion of the symbol α^1 we use the bit representation for the symbols of α^1, α^2 and α^3 as shown

$$\alpha^0 = 1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \alpha^1 = 2 = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}. \quad (2.32)$$

This is done for all the symbols in the matrix H to form the matrix H_b

$$H_b = \begin{bmatrix} 1 & 0 & 0 & | & 0 & 0 & 1 & | & 0 & 1 & 0 & | & 1 & 0 & 1 & | & 0 & 1 & 1 & | & 1 & 1 & 0 & | & 1 & 0 & 0 \\ 0 & 1 & 0 & | & 1 & 0 & 0 & | & 0 & 1 & 1 & | & 1 & 1 & 1 & | & 1 & 1 & 0 & | & 1 & 0 & 0 & | & 0 & 0 & 1 \\ 0 & 0 & 1 & | & 0 & 1 & 0 & | & 1 & 0 & 1 & | & 0 & 1 & 1 & | & 1 & 1 & 1 & | & 1 & 1 & 1 & | & 1 & 1 & 0 \\ - & - & - & | & - & - & - & | & - & - & - & | & - & - & - & | & - & - & - & | & - & - & - & | & - & - & - \\ 1 & 0 & 0 & | & 0 & 1 & 0 & | & 0 & 1 & 1 & | & 1 & 0 & 0 & | & 0 & 0 & 1 & | & 1 & 0 & 1 & | & 1 & 1 & 0 \\ 0 & 1 & 0 & | & 0 & 1 & 1 & | & 1 & 1 & 0 & | & 0 & 0 & 1 & | & 1 & 0 & 0 & | & 1 & 1 & 1 & | & 1 & 0 & 0 \\ 0 & 0 & 1 & | & 1 & 0 & 1 & | & 1 & 1 & 1 & | & 1 & 1 & 0 & | & 0 & 1 & 0 & | & 0 & 1 & 1 & | & 1 & 1 & 1 \end{bmatrix}. \quad (2.33)$$

2.7.4.2 The Bit-Level ABP Algorithm

Consider a (n, k) RS code over the field $GF(2^m)$ defined by a Vandermonde $h \times n$ parity check matrix of the form shown in (2.24). The symbols of the RS codeword c can be expressed in their bit form, to give the row vector cb . The vector cb has a length nb , where $nb = n \times m$. A binary image expansion is then done on the H matrix, to form the matrix H_b . The matrix H_b has the dimensions $hb \times nb$, where $hb = h \times m$. The binary vector cb is then mapped onto signals, based on a selected modulation scheme, and then transmitted through an AWGN channel. For $1 \leq t \leq nb$, the received vector r obtained at the output of the channel is used to find the log likelihood ratios (LLR) from the initial reliabilities using (2.34)

$$L(c_t) = \log \frac{P(c_t = 1|r_t)}{P(c_t = 0|r_t)}, \quad (2.34)$$

where L represents the LLR vector of the codeword c . The LLR's are then fed into the ABP algorithm. The stages involved in the decoding process with the

ABP algorithm can be summarized as follows.

1. *The matrix updating stage:* The magnitude of the LLR's are first obtained to form the vector $|L|$. The vector $|L|$ is then sorted in ascending order and the indices noted. The indices of the vector $|L|$ are now in the form of (2.35)

$$I = [i_1, i_2, i_3, \dots, i_{hb}, \dots, i_{nb}]. \quad (2.35)$$

Since a higher magnitude implies a higher reliability for the bits, the first hb bits are considered to be the unreliable bits and the remaining $(nb - hb)$ bits are the most reliable bits. Vector row operations similar to Gaussian row reduction are done on the matrix H_b based on the indices vector, I . The Gaussian elimination is done in such a way that the indices of H_b matching the unreliable bits are reduced to match an identity sub matrix. The matrix H_b has a full rank because it is an expansion of the parity check matrix H for RS codes. This guarantees a total of hb independent columns [9]. However there is no guarantee that the independent columns match the indices vector I . If any of the indices are not able to reduce the column, we ignore that particular column and try to reduce the next one. Eventually, a total of hb columns are reduced to form an identity sub matrix in H_b . The purpose of the matrix updating stage is to prevent the propagation of unreliable information during the bit updating stage where BP is performed [9].

2. *The bit updating stage:* For the j^{th} iteration and for $1 \leq l \leq hb$, iterative BP is done on the matrix H_b to generate the extrinsic LLR vector L_e for each coded bit as shown in [9]

$$L_e^{(j)}(c_t) = \sum_{l=1, H_{b_{lt}}=1}^{hb} 2 \tanh^{-1} \left(\prod_{t'=1, t' \neq t, H_{b_{lt'}}=1}^{nb} \tanh \left(\frac{L^{(j-1)}(c_{t'})}{2} \right) \right). \quad (2.36)$$

The bit reliabilities are then updated using

$$L^{(j+1)} = L^j + \beta L_e^{(j)}, \quad (2.37)$$

where β is the damping coefficient and is of the value $0 < \beta \leq 1$

3. *The stopping condition:* The iterative process continues until the maximum number of iterations (I_{max}) is reached, or until the syndrome check is satisfied.

The pseudo code for the ABP algorithm is shown in Alogrithm 1.

Algorithm 1: The ABP decoding algorithm [9]

Input: The received vector, r , from the channel.

Output: The bit representation of the decoded message.

Initialize: Obtain the LLR vector: $L = \frac{4r}{N_0}$, where N_0 = Noise spectral density.

Set the value j_{max} . Set the value of β ,

repeat

The matrix updating stage

- Obtain the vector $|L|$, by finding the absolute values of the vector L
- Sort the $|L|$ in ascending order and note the swapped indices.
- Systematize the hb columns of the matrix H_b based of the reordered indices

The bit updating stage

- Generate $L_e^{(j)}$ by applying SPA in (2.36)
- Update the bit-level reliabilities using (2.37)

Hard desicion: Where $\hat{r}$ = decoded vector

if $L^{(j+1)}(\hat{r}_t) > 0$ **then**
 | $\hat{r}_t = 1$

else if $L^{(j+1)}(\hat{r}_t) < 0$ **then**
 | $\hat{r}_t = 0$

end

Stopping condition:

if $\hat{r} \cdot H_b = 0$ **then**
 | **break**

else
 | $j = j + 1$

end

until $j = j_{max}$

2.7.4.3 The Symbol-Level ABP Algorithm

The Symbol-level ABP algorithm works in a very similar way as the Bit-level ABP algorithm. The major difference between the two algorithms is that for the Symbol-level ABP, adaptation of the H matrix is done based on the reliability of the symbols instead of the bits. The Symbol-level ABP can be summarized as follows

1. *The matrix updating stage:* For the Symbol-level ABP, the codeword is not broken down to the bit representation at the matrix updating stage. Instead the construction of a reliability matrix P is done. The maximum reliabilities in each column of P are used to represent the symbol reliabilities. The reliabilities of the symbols are then sorted in ascending order, and the indices of the swap are noted. The first h reliabilities are used to represent the least reliable symbols. Row operations are then done on the H matrix such that the order of the diagonal of the systematic submatrix match the indices of the unreliable symbols, similar to what was done in the Bit-level ABP. A computational efficient way of doing this is shown in [29]. A binary image expansion is then done on the adapted H matrix.
2. *The bit updating stage/The Log Sum Product Algorithm (SPA):* The received codeword from the channel is broken down to its bit representation and the LLR's obtained. The LLR's are fed in the log SPA. The log SPA can be summarized using the pseudo code in Algorithm 2. The notation used is the same as the notation used for the Bit-level ABP.

Algorithm 2: The Log SPA decoding algorithm

Initialize: Obtain the initial LLR vector: $L^{(0)} = \frac{4r}{N_0}$. Set the value j_{max} . Set the value of $LQ_{l,t}^{(0)} = L^{(0)}$,

repeat

- *The Check node updating stage*

$$LR_{lt}^{(j)} = 2 \tanh^{-1} \left(\prod_{t'=1, t' \neq t, H_{b_{lt'}}=1}^{nb} \tanh \left(\frac{LQ_{l,t'}^{(j-1)}}{2} \right) \right) \quad (2.38)$$

- *The Variable node updating stage*

$$LQ_{l,t}^{(j)} = L^{(0)} + \sum_{t'=1, t' \neq l, H_{b_{lt'}}=1}^{hb} LR_{lt'}^{(j)} \quad (2.39)$$

- *LLR update stage*

Obtain the extrinsic information L_e

$$L_e = \sum_{l=1, H_{b_{lt}}=1}^{hb} LR_{lt}^{(j)} \quad (2.40)$$

Use the extrinsic information to update the LLR's

$$L^{(j+1)} = L^j + \beta L_e^{(j)} \quad (2.41)$$

- *Hard desicion:*

Hard decision similar to that in Algorithm 1 is performed on $L^{(j+1)}$ to form the vector $\hat{r}$

- *Stopping condition:*

-If the syndrome is satisfied for $\hat{r} \cdot H_b$ the algorithm breaks.

-If not then $j = j + 1$ and the $LQ_{l,t}$ is feed back into the check node update stage.

until $j = j_{max}$

3. *Stopping Criteria:* From Algorithm 2, if the syndrome constraint is satisfied the algorithm stops. If the syndrome is not satisfied $LQ_{l,t}$ is fed back into the check node update stage. This is also a major difference between the Symbol-level ABP and the Bit-level ABP. For the Bit-level ABP the matrix H_b is updated during each iteration. For the Symbol-level ABP, the matrix updating stage is only done during the initialization on the matrix H , and the same matrix H_b is used during each iteration. If the syndrome is not satisfied when maximum iterations is run the algorithm stops and records a decoding failure.

2.7.5 The Parity check Transformation Algorithm

The Parity Check Transformation Algorithm (PTA) was recently developed in [18]. The algorithm was shown to outperform the commonly used *KV-GS* and the *BM* decoding algorithms for *RS* codes. The simulations were done for a 16 QAM modulation scheme in an AWGN channel. The algorithm was also said to have less computational complexity when compared to the widely used *KV-GS* algorithm. The algorithm, however, suffers from running a large number of iterations for low SNR regions.

To understand how the PTA algorithm works, we consider an (n, k) *RS* code with a parity check matrix H in the field $GF(2^m)$. The symbols of a codeword c are mapped onto signals based on a given modulation scheme and then transmitted through an AWGN channel. The vector r is received at the output of the channel, and used to create a reliability matrix P . The matrix P is then used by the PTA algorithm in the decoding process. For one iteration of the PTA algorithm the following steps are performed.

1. *Finding the symbol reliability:* For each column of the matrix P , the indices of the highest values are noted. The indices of the k highest values are used to represent the symbols with highest reliability and are denoted by $\mathcal{R}$. The remaining $n - k$ values are used to represent the least reliable symbols and are denoted by $\mathcal{U}$.

2. *Transformation of the H matrix:* The matrix H is transformed into partial systematic form by performing row operations similar to Gaussian reduction. This transformation is based on the indices $\mathcal{R}$ and $\mathcal{U}$, such that the identity matrix matches the indices of $\mathcal{U}$ and the parity matrix matches the indices of $\mathcal{R}$. The transformed matrix H is denoted by H' . A computational efficient way of doing this is shown in [29].
3. *Hard Decision detection:* A vector $\hat{r}$ is obtained by performing hard decision detection on the matrix P .
4. *Satisfying the syndrome checks:* Syndrome check are performed on all the $(n - k)$ rows of the matrix H using the vector $\hat{r}$. For $1 < i < (n - k)$, the i^{th} row of the matrix H' can be denoted as H'_i . The syndrome checks are obtained by performing the dot product

$$S_i = \hat{r} \cdot H'_i. \quad (2.42)$$

Only one symbol with an index in $\mathcal{U}$ participates in the syndrome check for every row, because these indices match the partitioned identity submatrix in H' . All symbols with indices matching $\mathcal{R}$ participate in the syndrome check because their indices match the partitioned parity submatrix.

5. *The correction step:*
 - If the syndrome check is satisfied i.e, $S_i = 0$, the reliabilities with indices matching $\mathcal{U}$, participating in the syndrome check, are added by the value of a correction factor, δ , and the reliabilities with indices matching $\mathcal{R}$ are added by the value $\delta/2$.
 - If the syndrome check is not satisfied i.e, $S_i \neq 0$, the reliabilities with indices matching $\mathcal{U}$, participating in the syndrome check, are subtracted by the value of the correction factor, δ , and the reliabilities with indices matching $\mathcal{R}$ are subtracted by the value $\delta/2$.

By correcting the reliabilities with the indices of $\mathcal{U}$ by a larger correction factor as compared to that of $\mathcal{R}$, more unreliable symbols are corrected in each iteration. This enables the algorithm to output the correct codeword.

6. *Updating of the P matrix:* The corrected reliabilities matching the indices of $\mathcal{R}$ and $\mathcal{U}$ are then updated in the P matrix and used in the next iteration.
7. *The stopping criteria:* The algorithm has two stopping conditions
 - When all the $(n - k)$ syndrome checks are satisfied.
 - When a negative value is updated in the P matrix.

If none of the this conditions is met, the algorithm repeats steps 1 to 6.

The pseudo code run for the j^{th} iteration of the PTA algorithm can be seen in Algorithm 3 [18]

Algorithm 3: The PTA decoding algorithm**Input:** The received vector, r , from the channel.**Output:** The decoded vector $\hat{r}$.**Initialize:** Obtain the P matrix from r . **repeat**

- 1.
- Symbol reliability indexing:*

$$\hat{r}_r = \arg \max(P)$$

$$\mathcal{U} = [I_{\hat{r}_{r1}} \dots I_{\hat{r}_{r(n-k)}}]$$

$$\mathcal{R} = [I_{\hat{r}_{r(n-k)+1}} \dots I_{\hat{r}_{rn}}],$$

where $\hat{r}_r$ represents the received hard decision sequence of the reliabilities and I represent the indices in the P matrix.

- 2.
- H matrix transformation:*

Transform the H matrix into a rearranged systematic matrix H' based on $\mathcal{U}$ and $\mathcal{R}$.

- 3.
- Hard Decision detection:*

$$\hat{r}_s = \arg \max(P),$$

where $\hat{r}_s$ represents the received hard decision sequence of the symbols

- 4.
- Syndrome Check:*

$$s_i = \hat{r}_s \cdot H'_i.$$

- 5.
- The P matrix update:*

if $s_i = 0$

$$\mathcal{U}_r = \mathcal{U}_r + \delta$$

$$\mathcal{R}_r = \mathcal{R}_r + \delta/2$$

else if $s_i \neq 0$

$$\mathcal{U}_r = \mathcal{U}_r - \delta$$

$$\mathcal{R}_r = \mathcal{R}_r - \delta/2$$

end

where $\mathcal{U}_r$ and $\mathcal{R}_r$ are reliabilities at the indices $\mathcal{U}$ and $\mathcal{R}$ of the P matrix respectively.

- 6.
- Stopping condition:*

if

| Negative value is updated in P matrix.

else

$$j = j + 1$$

end**until** $\hat{r} \cdot H' = 0$

2.8 Summary

This chapter provides the core foundation of the work done in this dissertation. A review of much of the literature used to accomplish this research has been explained. A large portion of this chapter is focused decoding algorithms for *RS* codes. This is because work done in this dissertation is focused on soft decision decoding techniques, specifically the ABP and the PTA decoding algorithms. The reason for this is that both the ABP and the PTA have been shown to outperform other good SER performing decoding algorithms, including the widely used *KV-GS* algorithm, in [9] and [18] respectively. In this regard, a comparative study is done in Chapter 5 between the ABP, the PTA and the Modified PTA that is developed in Chapter 3. For the purpose of giving a fair comparison, in terms of SER and complexity, the Symbol-level ABP is used in the comparative study. This is because the PTA and the Modified PTA are both Symbol-level decoding algorithms. This is done to see how the performance of the PTA and the Modified PTA compares to the performance of the Symbol-level ABP. The study is done for different code rates.

CHAPTER 3

The Modified Soft Input Parity check Transformation Algorithm for Reed-Solomon Codes

In this chapter, an analysis of the PTA algorithm is done. The analysis is in terms of the number of iterations, the presence of negative values in the reliability matrix, the new stopping condition and the value of the correction factor. This analysis provides the basis of the modifications that are implemented on the PTA, with the aim of reducing the number of iterations required to decode and improving the SER performance. An extension to the PTA algorithm is developed from these modifications and is referred to as the Modified PTA (MPTA). The research done in this chapter is to be submitted in a peer-reviewed journal for publication. Therefore, work done in this chapter is structured in line with the journal paper. The paper submission details of the journal paper is as follows.

- ◊ **Y. Genga** and D. Versfeld “The Modified Soft Input Parity check Transformation Algorithm for Reed-Solomon Codes” -*To be Submitted to the IEICE Transactions on Communications letters*.

The modifications to the PTA decoding algorithm developed in this paper were conceptualized by Y. Genga. The manuscript preparation and the results obtained were also presented by Y. Genga. D. Versfeld supervised the manuscript preparation and the work done in the journal paper.

Summary

In this paper, we propose a modification to the recently developed Parity check Transformation Algorithm (PTA) used in the decoding of Reed Solomon codes. This extension of the PTA is referred to as the Modified Parity check Transformation Algorithm (MPTA). The MPTA is developed with the aim of reducing the number of iterations run by the algorithm during the decoding process, and to also improve on the SER performance of the algorithm.

key words: Reed-Solomon codes, Soft decision decoding, Iterative decoding

3.1 Introduction

Reed-Solomon (RS) codes [21] are a class of linear block codes that are widely used for applications that range from telecommunications to storage devices. These codes are largely popular because they meet the Singleton bound. What this means is that a hard decision decoder (HDD) based on minimum distance decoding can correct up to $[n - k]/2$ symbols for a given (n, k) RS code. Hard decision decoders for RS codes include the Berlekamp-Massey ($B-M$) algorithm [15],[2], the Euclidean algorithm [32], and the Berlekamp-Welch algorithm [30]. HDD algorithms for RS codes have been shown to be efficient, however, they are significantly outperformed by soft decision decoding techniques. Koetter and Vardy (KV) [11] presented a symbol level decoding algorithm that utilizes the soft reliability information from the channel to make a multiplicity matrix that is fed into the Guruswami Sudan (GS) [8] algorithm. The KV algorithm outperforms HDD algorithms significantly especially for low-rate codes. However, complexity can become prohibitively large when trying to achieve large coding gains with the KV algorithm [9].

The Parity check Transformation Algorithm (PTA) is a symbol wise soft decision decoding algorithm that was recently proposed in [18]. The algorithm transforms the parity check matrix, H , of an RS code after every iteration depending on the

reliability of the symbols. The reliability of the symbols is attained from soft information received at the channel output. In the same paper, the PTA algorithm was shown to outperform the widely used Koetter and Vardy (*KV*) algorithm and the Berlekamp-Massey (*BM*) algorithm for simulations run in the AWGN channel with the symbols being mapped onto a 16QAM modulation scheme. The performance was measured in terms of the symbol error rate (*SER*). The major drawback of the PTA is the high number of iterations needed to find the decoded codeword, especially for low SNR cases.

In this paper, modifications to the PTA are introduced with the aim of reducing the number of iterations required to decode. These modifications are used to develop an extension of the PTA, which is referred to as the Modified PTA (*MPTA*). The rest of the paper is structured as follows. A basic description of how the PTA decoding algorithm works is given in Section 3.2. The analysis and modifications to the PTA is explained Section 3.3. A performance comparison between the PTA and the *MPTA* algorithm is done in Section 3.4. Finally, a conclusion is given in Section 3.5.

3.2 The Parity check Transformation Algorithm

We now summarize the PTA algorithm [18] to establish notation. Consider an (n, k) *RS* code with a parity check matrix H in the field $\text{GF}(2^p)$. Let c be a codeword of length n . The symbols of the codeword c are mapped onto signals based on a selected modulation scheme and then transmitted through an AWGN channel. At the output of the channel a received vector r is obtained. Based on the vector r a reliability matrix R can be constructed. The matrix R is then fed into the PTA decoder.

The steps involved in one iteration of the PTA algorithm are as follows

1. *Getting the symbol reliability:* Find the highest values in each column of the R matrix. From these values, let the k highest values represent the most reliable symbols. The indices of these symbols are denoted by $\mathcal{K}$. The

remaining $n - k$ values represent the least reliable symbols and their indices are denoted by $\mathcal{U}$.

2. *The H matrix transformation step:* Transform the H matrix into matrix H^t by performing row operations similar to Gaussian such that H^t is in a rearranged systematic form based on the indices $\mathcal{U}$ and $\mathcal{K}$. The transformation is done in such a way that the indices of $\mathcal{U}$ match the partitioned identity matrix and the $\mathcal{K}$ indices match the parity partition as shown in [18]. A computationally efficient way of doing this is shown in [29].
3. *Performing the syndrome checks:* For $1 \leq i \leq (n - k)$, the i^{th} row of the matrix H^t can be denoted as H_i^t . A syndrome check is now performed using all the $n - k$ rows of the matrix H by performing hard decision detection on the R matrix to obtain a vector $\hat{r}$, and getting the dot product $\hat{r} \cdot H_i^t$. Due to the rearranged systematic structure of H_i^t , only one symbol with the index in $\mathcal{U}$ participates in the syndrome check. All the k symbols in $\mathcal{K}$ participate in the dot product.
4. *The correction step:* If the dot product is not zero, we decrease the symbol indexed in $\mathcal{U}$ by the value of a correction factor, δ , and decrease the symbols indexed in $\mathcal{K}$ by $\delta/2$. If the syndrome check is satisfied we add the δ to the symbol indexed in $\mathcal{U}$ and $\delta/2$ to the symbols indexed in $\mathcal{K}$.
5. *The update step:* These corrected reliabilities are then used to update the R matrix based on the $\mathcal{U}$ and $\mathcal{K}$ indices. The updated R matrix is then used in the next iteration.
6. *The stopping criterion:* steps 1 to 5 are repeated until all the $(n - k)$ syndrome checks are satisfied, or until a negative value appears in the matrix R .

From [18] it was shown that a smaller correction factor (δ) improves the performance of the algorithm. This, however, comes at the cost of increased number of iterations as the algorithm has to perform more corrections to the received codeword.

3.3 Modifications to the PTA algorithm

We now analyze the performance of the PTA and the effect of the modifications to the algorithm. Simulations are based on a (15,7) Reed-Solomon code using BPSK modulation in an AWGN channel. The value of δ used is 0.001.

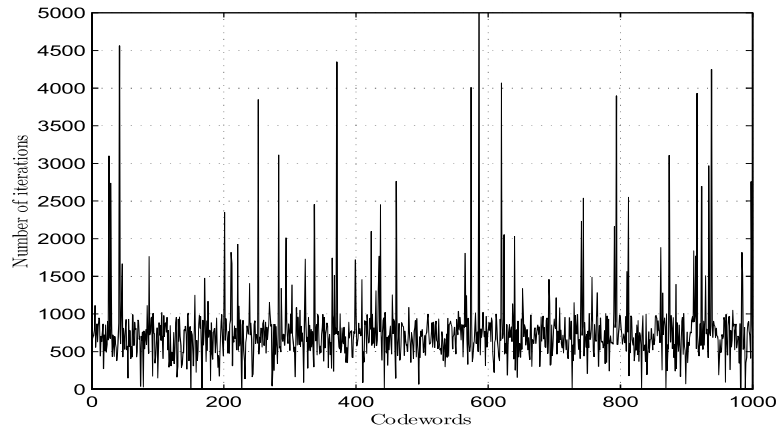
3.3.1 Performance of the PTA based on the number of iterations

To better understand how the PTA algorithm works, the early stopping condition used in [18] is not considered. The only stopping criterion for the algorithm is when all syndrome checks are satisfied. This version of the PTA defined by the syndrome check as the only stopping condition is denoted as PTA^γ .

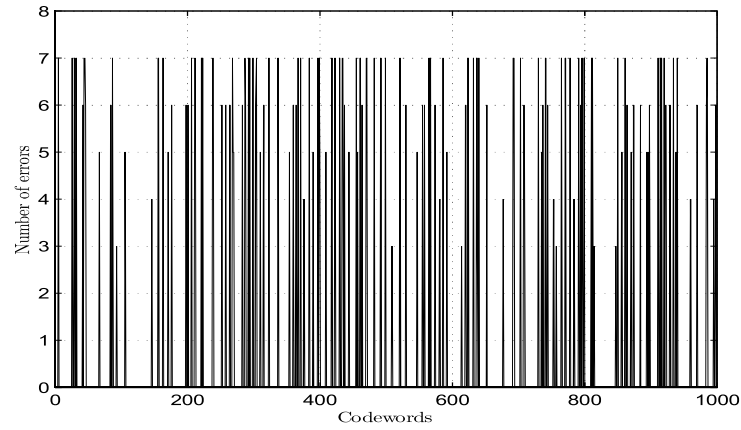
Fig. 3.1 depicts simulations for the PTA^γ with 1000 codewords at an SNR of 0 dB. The number of errors and the number of iterations required to satisfy the checks are recorded.

Fig. 3.2 shows simulations for the Hamming distance between the actual codeword and the decoded codeword at the output of each iteration of the PTA^γ . In Fig. 3.2, results for 4 codewords which are representative to most of the other codewords decoded using the PTA^γ are used.

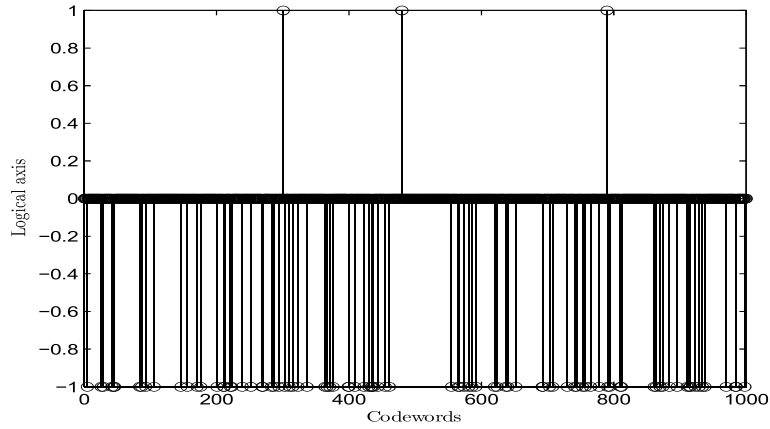
From Fig. 3.1 and Fig. 3.2 it can be seen that the PTA^γ algorithm outputs a codeword, regardless if it is correct or incorrect, when it satisfies the syndrome checks. What this means is that more iterations for the PTA^γ algorithm do not necessarily mean an improved performance with the algorithm. This can be seen especially when the algorithm runs for more than 1000 iterations. From Fig. 3.1 it can be seen that most of the syndrome checks for the correct codewords are also satisfied whenever the PTA^γ runs less than 1000 iterations. Most incorrect codewords are received from the PTA^γ whenever the algorithm runs for more than 1000 iterations. This is illustrated in Fig. 3.1(c). (Fig. 3.1(c) is a logical graph that plots a -1 whenever the PTA^γ run for 1000 or more iterations and satisfies



(a) Number of iterations required to satisfy the syndrome checks for each of the 1000 codewords



(b) Number of errors recorded for each of the 1000 codewords decoded with the PTA^γ



(c) Logical graph for 1000 iterations or more

FIGURE 3.1: Number of errors compared to the number of iterations

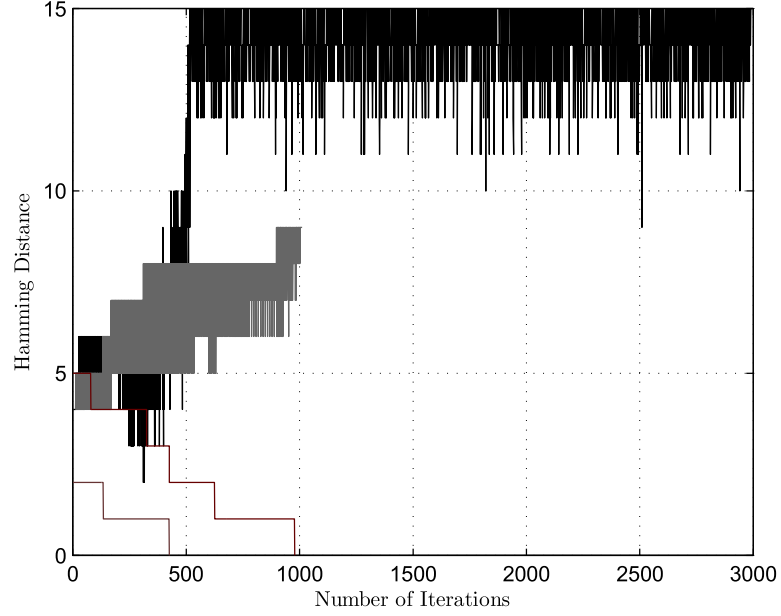
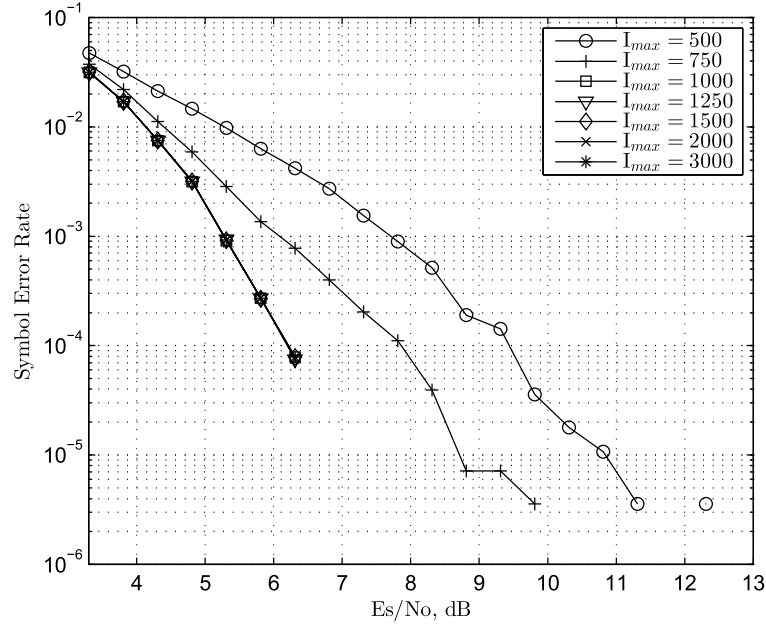


FIGURE 3.2: Comparing the Hamming distance to the Number of Iterations.

the syndrome checks with an incorrect codeword, and a 1 whenever the syndrome checks are satisfied with a correct codeword using 1000 or more iterations).

From Fig. 3.1(c) only 3 codewords that run for 1000 or more iterations with the PTA^γ are decoded correctly. This indicates the presence of a threshold in the PTA^γ which occurs when the algorithm runs for more than 1000 iterations. Fig. 3.3 investigates the effect of different thresholds on the PTA^γ by setting different maximum number of iterations (I_{max}) the algorithm can run.

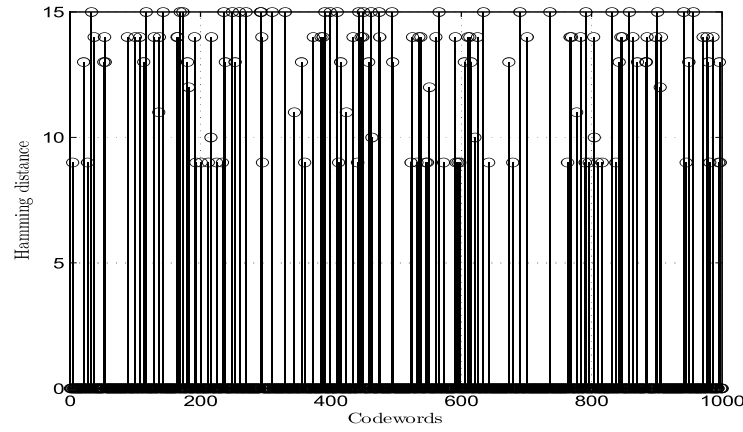
From Fig. 3.3 it can be seen that the threshold of the PTA^γ is at $I_{max} = 1000$ and that it serves as a point of saturation for the algorithm, as the performance appears to remain the same beyond this point. Thus, a set maximum value of 1000 iterations for the PTA^γ should be used to prevent redundancy.

FIGURE 3.3: Performance comparison of different values of I_{max}

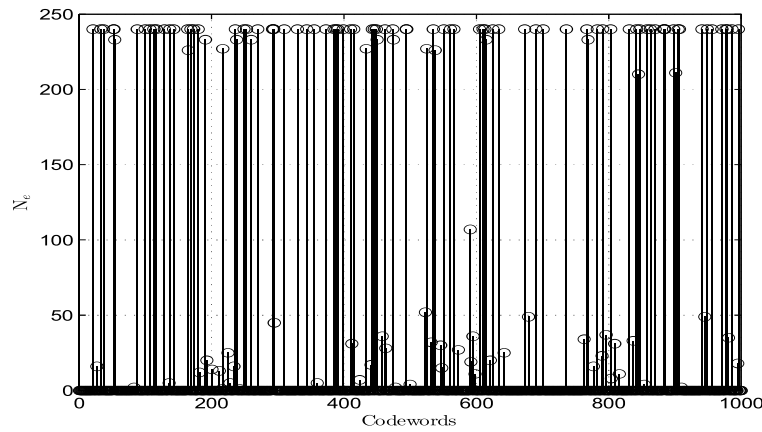
3.3.2 Performance of the PTA based on the number of negative values in R

A possible reason for the increased errors with more iterations could be the appearance of negative values formed in the reliability matrix, originating from the corrections done during each iteration. Simulations to find out how the number of negative values in the reliability matrix corresponded to the Hamming distance between the actual codeword and the received codeword at the output of the PTA ^{γ} can be seen in Fig. 3.4. The test was run for 1000 codewords. The parameter N_e represents the total number of negative values in the reliability matrix. A logical graph similar to that in Fig. 3.1(c) is also plotted in Fig. 3.4(c) to better illustrate the result. The logical graph plots a -1 when the Hamming distance is greater than 0 and $N_e > 0$, and a 1 when Hamming distance is equal to 0 and $N_e > 0$.

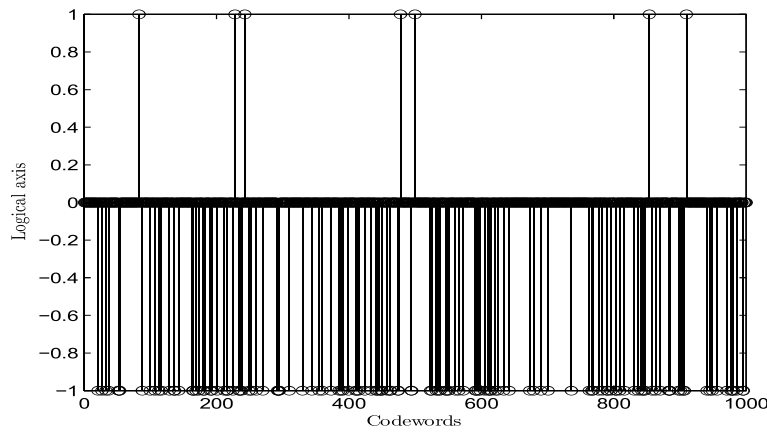
From Fig. 3.4, it can be seen that for most cases, the presence of a negative value in the reliability matrix directly affect the number of errors of the received codeword at the output of PTA ^{γ} . This result validates the early stopping condition of



(a) Hamming distance between the decoded codeword and the actual codeword



(b) Number of negative values found in the Reliability matrix PTA^γ



(c) Logical graph for N_e compared to the Hamming distance

FIGURE 3.4: Number of errors compared to the number of negative values

the PTA algorithm used in [18].

To prevent errors caused by the negative values, absolute values of the corrected reliabilites are used after each iteration of the PTA^γ . This modification of the PTA^γ with absolute values is referred to as the absolute PTA^γ and denoted as aPTA^γ .

3.3.3 Performance of the PTA based on the Stagnant check.

If the absolute values are to be used, the early stopping condition defined in [18] is not applicable as the reliability matrix does not run into a negative value. To prevent the aPTA^γ from running upto I_{max} iterations in the event it is unable to find the correct codeword, a new early stopping condition is required. A modification of the stagnant check [24], is added to the algorithm to perform this function. The stagnant check acts as a predictive algorithm, by trying to determine if the algorithm will have a decoding failure or a decoding success.

The stagnant check algorithm works is as follows

1. At a given iteration I_α , the weight of the unsatisfied syndrome checks ($w(s)$) is noted.
2. The value of $w(s)$ is noted for each iteration after I_α until a specified iteration I_{stag} is reached.
3. At iteration I_{stag} , a stagnant check is done.
 - If $w(s)$ is the same from I_α to I_{stag} then the algorithm is perceived to be stagnant (unable to find the correct codeword), and the decoding process is stopped.
 - If $w(s)$ has changed from I_α to I_{stag} then the aPTA^γ runs until all the syndrome checks are satisfied, or until I_{max} is reached.

The main aim of the stagnant check is to help reduce the number of iterations required by the aPTA^γ . To get an optimum performance of the aPTA^γ with

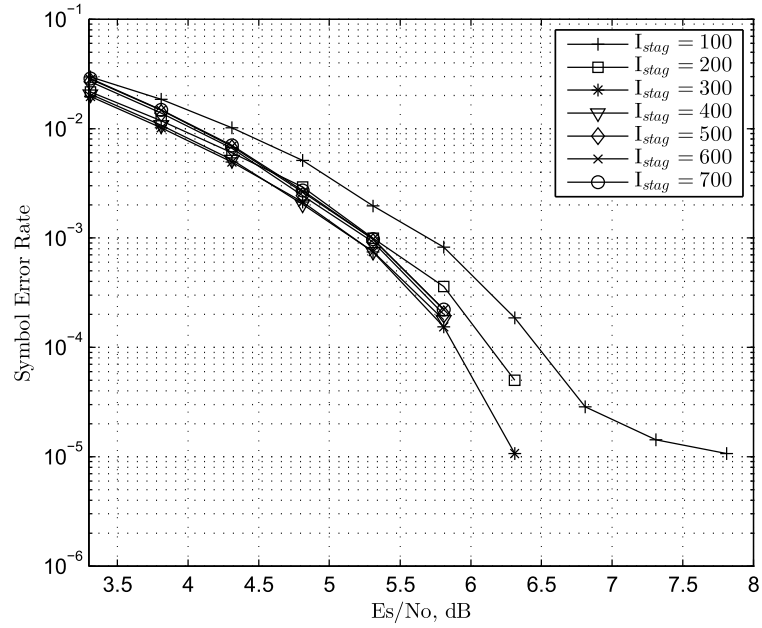
the stagnant check, simulations are done for different values of I_{stag} and run for $I_{max} = 1000$. Simulations are run for $I_{\alpha} = 1$ and the stagnant check was done at $I_{stag} = 100, I_{stag} = 200, I_{stag} = 300, I_{stag} = 400, I_{stag} = 500, I_{stag} = 600, I_{stag} = 700$. The average number of iterations run by each was noted. The results for these simulations can be seen in Fig. 3.5.

From Fig. 3.5 it can be seen that the aPTA $^{\gamma}$ with the value of $I_{stag} = 400$ gives the best performance when compared to the rest. The aPTA $^{\gamma}$ with the stagnant check is referred to as the Modified Parity check Transformation Algorithm (MPTA).

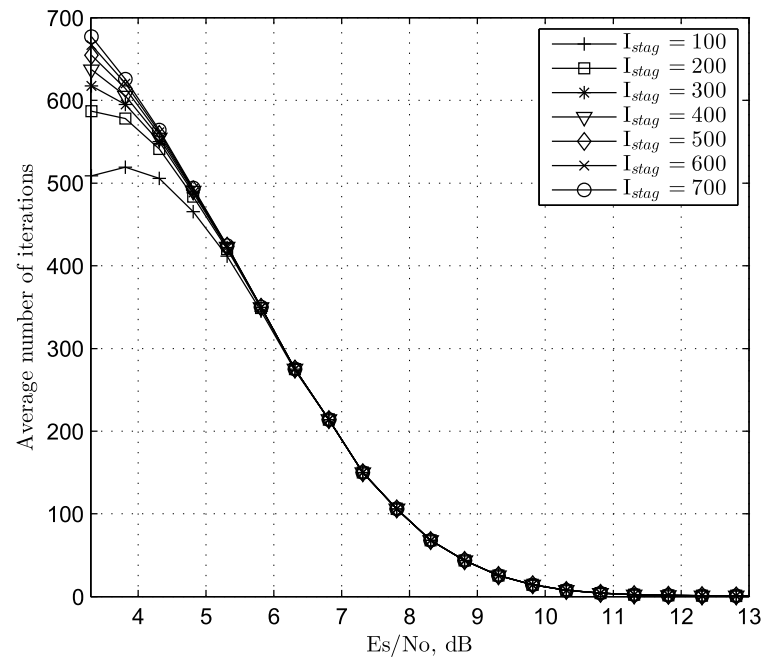
3.3.4 Performance of the PTA based on the value of the correction factor, δ

From Fig. 3.5(b), it can be seen that even with the stagnant check the algorithm still runs for many iterations in low SNR regions. The main reason for this is the use of a small value of δ . The smaller the value of δ the better the performance, but this is at the cost of an increased number of iterations as shown in [18]. This is because there is not a large difference in the corrections made by δ in each iteration for the wrong symbols as it attempts to satisfy the syndrome checks. Since the difference in the corrections made by a δ of 0.001 are quite small several iterations would be required to satisfy the checks. Simulations for different values of δ with the aPTA $^{\gamma}$ are done to test for the performance and the average number of iteration required by the algorithm. No stagnant check is used because an accurate representation of the average number of iterations run by the algorithm for each SNR is required. The results for these simulations can be seen in Fig. 3.6.

Fig. 3.6 validates the results shown in [18] with a BSPK modulation scheme for the PTA. It can also be seen that the number of iterations run by the aPTA $^{\gamma}$ between $\delta = 0.005$ and $\delta = 0.001$, increases as the value of δ reduces for lower SNR regions. However, the opposite seems to happen between $\delta = 0.01$ and $\delta = 0.05$. This is due to these values of δ making a large difference in the corrections made to the soft information. This causes the algorithm to change the reliability of the

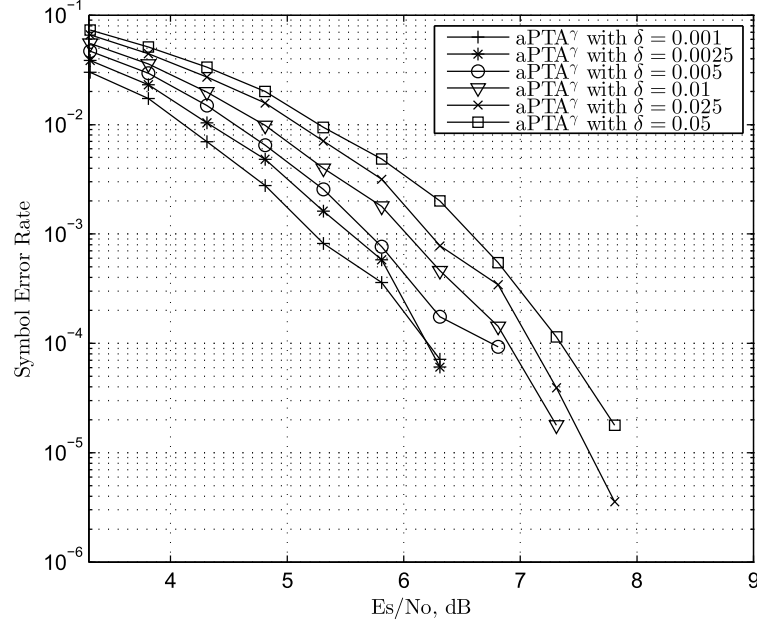


(a) SER Performance for different values of I_{stag} for the MPTA

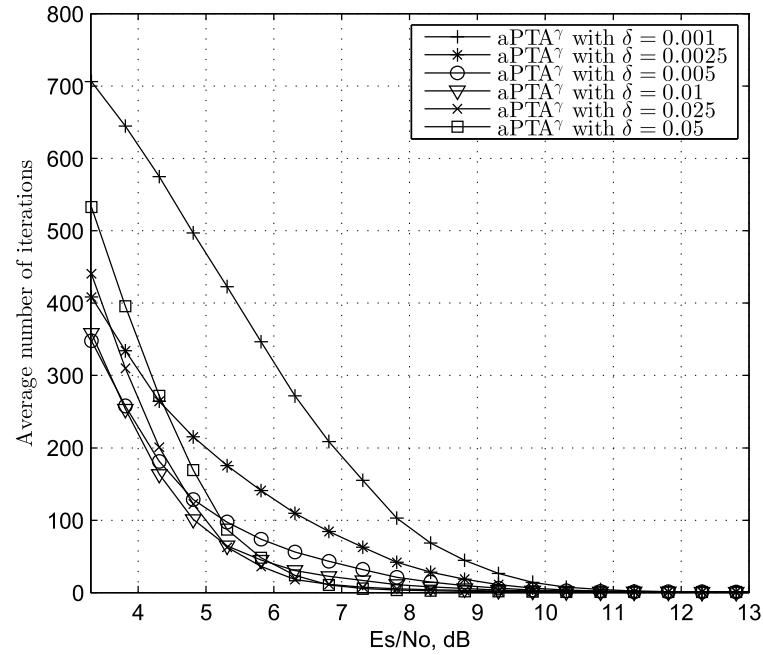


(b) Average number of iterations for the MPTA

FIGURE 3.5: Performance of the MPTA based on the value of I_{stag}



(a) SER Performance of different aPTA $^\gamma$ with different values of δ



(b) Average number of iterations for the aPTA $^\gamma$ with different values of δ

FIGURE 3.6: Performance of the aPTA $^\gamma$ based on different values of δ

symbols faster, thus passing the correct symbols as it satisfies the checks. It is evident from Fig. 3.6 that the value of the correction factor should be $\delta < 0.01$ for better SER results to be obtained with the algorithm.

To reduce the number of iterations run by the MPTA, a discreet variable δ approach is added to the algorithm. For this implementation, two scenarios are considered. A case of increasing values of δ , and a case of reducing values of δ . In both cases, the value of δ changes with respect to the number of iterations.

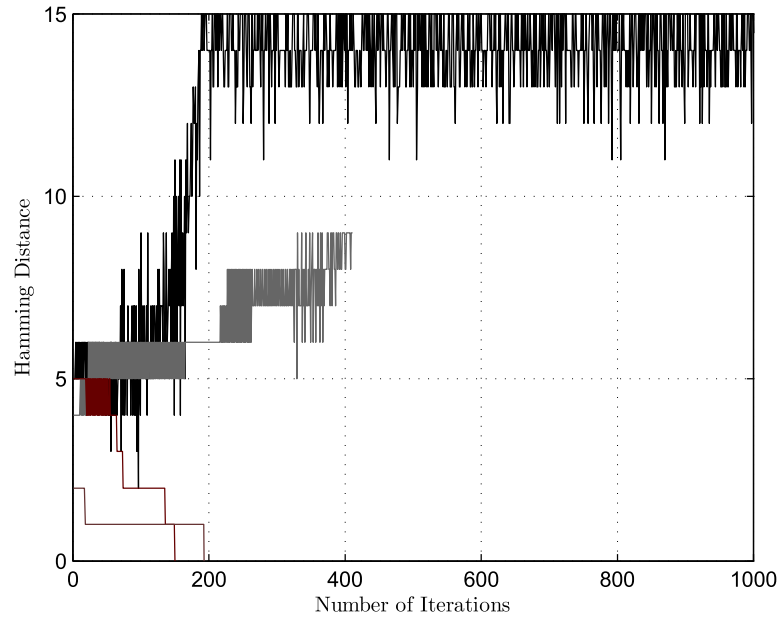
For this analysis, three values of δ are used. The values of δ chosen are $\delta = 0.005$, $\delta = 0.0025$ and $\delta = 0.001$. The reason these values of δ have been selected is because the performance difference between $\delta = 0.001$ and $\delta = 0.005$, from Fig. 3.6, is about 0.5 dB.

To help with the implementation of this form of discreet variable δ s with the algorithm, a similar simulation to the one done in Fig. 3.2 is done for the PTA ^{γ} with $\delta = 0.005$ and $\delta = 0.0025$. This is done to help with the assignment of the number of iterations that each δ receives. The results for these simulations can be seen in Fig. 3.7.

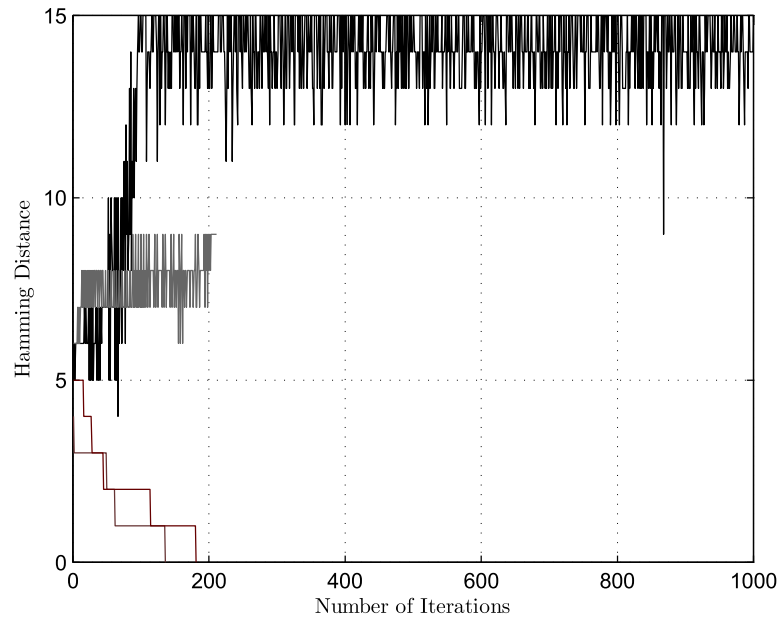
The simulations are let to run for as many iterations until the syndrome checks are satisfied. For Fig. 3.7, only the first 1000 iterations are shown. Once again, results for only 4 codewords that are representative of most of the other codewords decoded with the PTA ^{γ} are used.

From Fig. 3.7 it can be seen that for $\delta = 0.005$ most of the correct received codewords are found within 200 iterations. Beyond this, the algorithm begins to saturate and is unable to find the correct codeword to satisfy the syndrome checks. For $\delta = 0.0025$, the algorithm appears to saturate after about 400 iterations. Therefore, of the 1000 maximum iterations for the MPTA, the number of iterations each δ runs is assigned with respect to the SER performance shown in Fig. 3.6(a) and the performance based on the number of iterations required to satisfy the checks shown in Fig. 3.2 and Fig. 3.7. For the case of the increasing δ , the number of iterations is split as follows

- $\delta = 0.001$ runs for the first 625 iterations



(a) Comparing the Hamming distance to the Number of iterations for $\delta = 0.0025$



(b) Comparing the Hamming distance to the Number of iterations for $\delta = 0.005$

FIGURE 3.7: Performance of the PTA ^{γ} with $\delta = 0.0025$ and $\delta = 0.005$ based on the number of iterations

- $\delta = 0.0025$ runs the next 300 iterations
- $\delta = 0.005$ runs the last 75 iterations

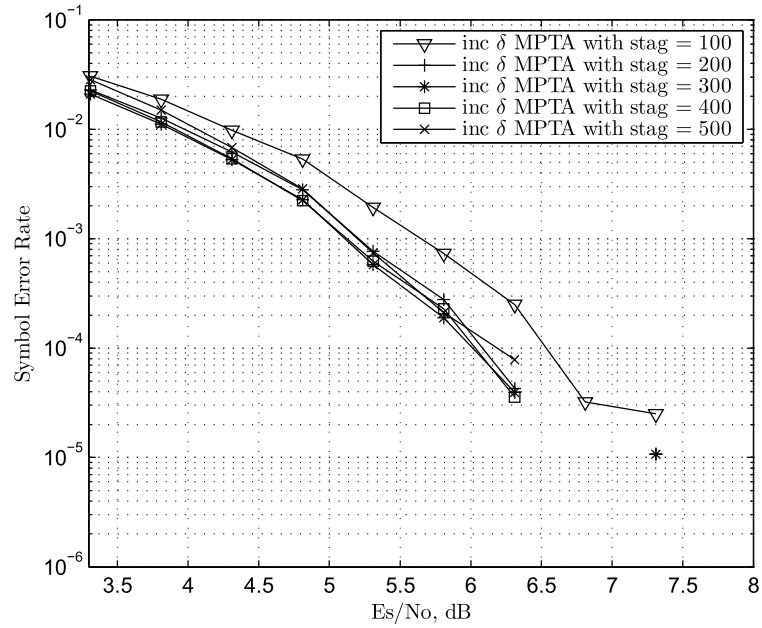
For the case of the decreasing δ , the number of iterations is split as follows

- $\delta = 0.005$ runs for the first 75 iterations
- $\delta = 0.0025$ runs the next 300 iterations
- $\delta = 0.001$ runs the last 625 iterations

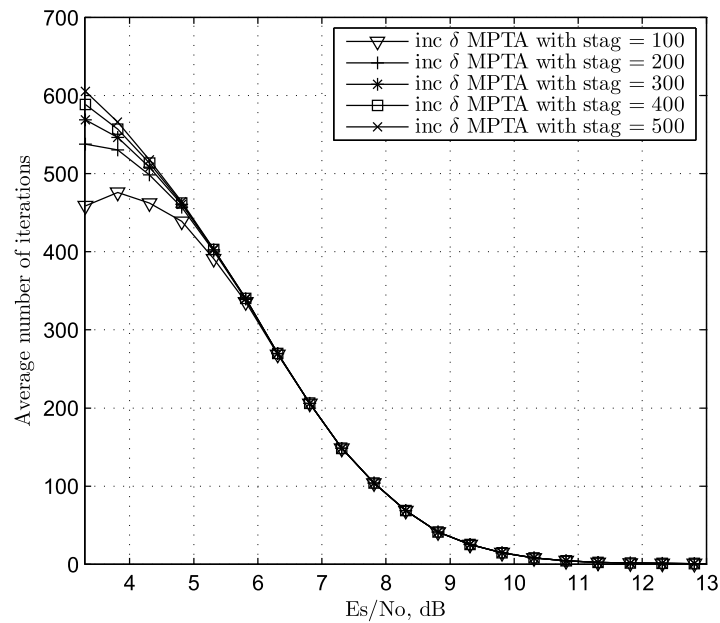
Simulations for the discrete variable δ version of the MPTA are done to test for the optimum performance with different values of I_{stag} . These simulations are for SER performance and the average number of iterations required by the algorithm. The values of I_{stag} used in the simulations for the decreasing δ are smaller than those used for increasing δ . This is because for the case of decreasing δ the first value of δ used is 0.005. An earlier stagnant check for $\delta = 0.005$ is required as the corrections it makes to the codeword during each iteration are larger than those of $\delta = 0.001$. As a result it runs into a stagnant state much earlier as shown in Fig. 3.7. The results of these simulations for the both cases of increasing and decreasing δ can be seen in Fig. 3.8 and Fig. 3.9 respectively.

From Fig. 3.8 it can be seen that the best performance, in terms of SER, is obtained when $I_{stag} = 400$. It is naturally outperformed by $I_{stag} = 100$, $I_{stag} = 200$ and $I_{stag} = 300$ in terms of the number of iterations required for the lower SNR values. Based on the SER performance, the value of $I_{stag} = 400$ was selected for the increasing δ version of the MPTA in the comparative study.

A similar observation can be done in Fig. 3.9 with $I_{stag} = 80$, as it gives best performance in terms of SER. For $I_{stag} = 40$ the same number of errors are recorded for the last two SNR values in which an error is found. The reason for this is because the stagnant check is too early and breaks dec δ MPTA before it can find the correct codeword. In terms of the number of iterations, it is outperformed by $I_{stag} = 20$, $I_{stag} = 40$ and $I_{stag} = 60$ for low SNR values. Based on the

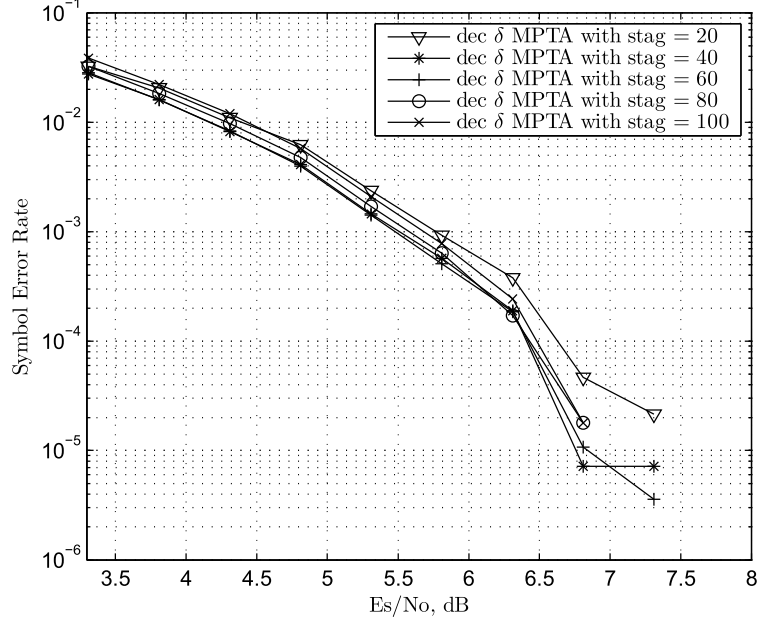


(a) SER Performance for increasing δ MPTA with different values of I_{stag}

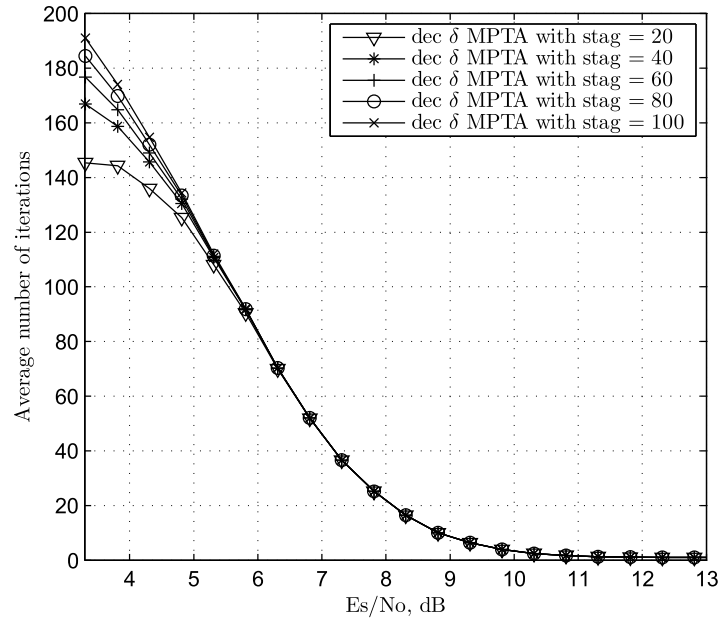


(b) Average number of iterations for the increasing δ MPTA for different values of I_{stag}

FIGURE 3.8: Performance of the increasing δ MPTA



(a) SER Performance for decreasing δ MPTA with different values of I_{stag}



(b) Average number of iterations for the decreasing δ MPTA for different values of I_{stag}

FIGURE 3.9: Performance of the decreasing δ MPTA

SER performance, the value of $I_{stag} = 80$ was selected for the decreasing δ version of the MPTA in the comparative study.

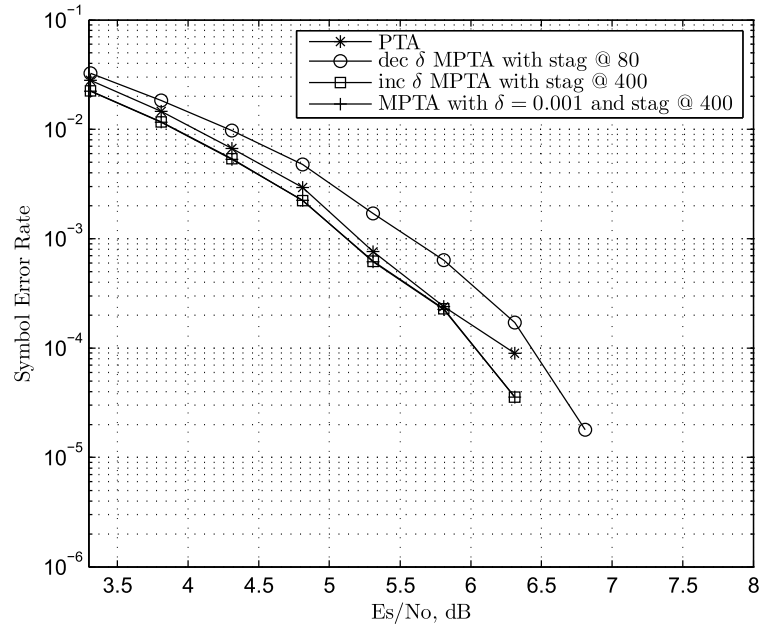
3.4 Performance Analysis

Fig. 3.10 shows simulations for the performance comparison of PTA to that of the MPTA with a single δ of 0.001 and $I_{stag} = 400$, an increasing δ with $I_{stag} = 400$ and a decreasing δ with $I_{stag} = 80$.

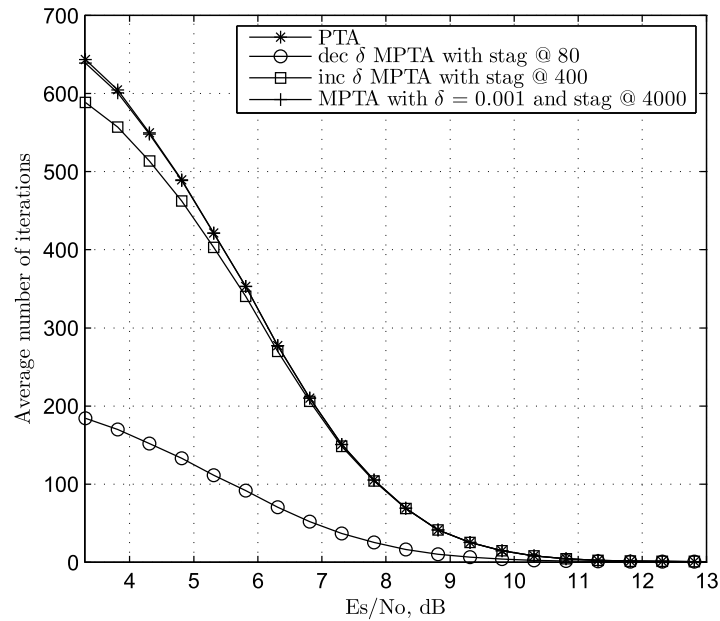
From Fig. 3.10 it can be seen that the single δ MPTA and the increasing δ MPTA have an almost identical performance and they both outperform the PTA in terms of SER. In terms of the average number of iterations for low SNR areas, the decreasing δ MPTA significantly outperforms all versions of the PTA. However, this is at a cost of about 0.5 dB. This SER performance difference justifies the use of the decreasing δ MPTA when a trade-off between SER performance and number of iterations is required. The increasing δ MPTA runs for less iterations than the PTA and gives a better performance. The single δ MPTA gives a better performance than the PTA but gives no difference in the average number of iterations.

3.5 Conclusion

In this paper, we are able to develop three different versions of the MPTA, each with its own trade-offs. The single δ MPTA increases the performance of the algorithm but at no reduction in the number of iterations. The increasing δ MPTA outperforms the PTA in terms of both the SER and the number of iterations required to decode. It also gives an almost identical performance to that of the single δ MPTA while running for less iterations in the low SNR regions. Finally, the decreasing δ MPTA significantly reduces the average number of iterations run by the algorithm by well over 400 iterations. This reduction in the number of iterations comes at the cost of a loss of less than 0.5 dB when compared to the PTA.



(a) SER Performance for the MPTA and the PTA



(b) Performance of the MPTA compared to the PTA

FIGURE 3.10: Performance of the MPTA compared to the PTA

CHAPTER 4

Performance Analysis for Different Code Rates

4.1 Introduction

From Chapter 3, the single δ MPTA and increasing δ MPTA, both with stagnant checks at 400, have been shown to outperform the PTA for a (15,7) *RS* code. A (15,7) *RS* code is approximately a half-rate code. The work in this chapter builds on what has been done in Chapter 3 by running simulations for the MPTA with different code rates. These results are used for the comparative study with the PTA algorithm at the end of this chapter.

To justify the selection for the stagnant check points used with the MPTA for the comparative study, simulations are done for different values of I_{stag} on a high-rate code and a low-rate code. The simulations done in Chapter 3 on the (15,7) *RS* code are used to represent the values of I_{stag} for a half-rate *RS* code. The best performing values of I_{stag} with the MPTA, for the different code rates, are used in the final comparative study.

A modification to the syndrome check stage of the MPTA is also added in this chapter. This modification is done with the aim of reducing the computational complexity of the algorithm.

The rest of this chapter is structured as follows: Section 4.2 looks into the reduction of computational complexity of the syndrome check stage of the MPTA. Section 4.3 presents the analysis of the performance for the MPTA on a low-rate *RS* code. Section 4.4 analyzes the performance of the MPTA on a high-rate *RS* code. Section 4.5 gives the analysis of the performance comparison between the PTA

and the MPTA for different code rates. Finally Section 4.6 summarizes the work done in this chapter.

4.2 Modification on the Syndrome check stage of the MPTA

The syndrome check stage in the PTA and MPTA involves finding the dot product between the hard decision received vector ($\hat{r}$) and the individual rows of the transformed parity check matrix (H') as shown in (2.42). Due to the partial systematic structure of the H' , only one of the unreliable symbols matching the column index of the identity submatrix participates in the syndrome check. This is because the rest of the unreliable symbols multiply with indices matching the zeros of the partitioned identity submatrix. All the k reliable symbols participate in the syndrome calculation because their indices correspond to the partitioned parity submatrix in matrix H' . Therefore, from (2.16), to perform the syndrome checks all we have to do is find the dot product of the indices $\mathcal{R}$ in $\hat{r}$ with the corresponding indices in each row of the matrix H' . This result is then added to the unreliable symbol that matches the 1 in the identity submatrix of H' as shown in (4.1).

$$S = \hat{r}_{\mathcal{U}} + (\hat{r}_{\mathcal{R}} \cdot H'_{\mathcal{R}i}), \quad (4.1)$$

where

- $H'_{\mathcal{R}i}$ = The i^{th} row of the partitioned parity submatrix in the H' matching the indices in $\mathcal{R}$,
- $\hat{r}_{\mathcal{U}}$ = The symbol in $\hat{r}$ that matches the index in both $\mathcal{U}$ and the corresponding 1 in the partitioned identity submatrix in H' , and
- $\hat{r}_{\mathcal{R}}$ = The k symbols in $\hat{r}$ that match the indices in $\mathcal{R}$.

This modification helps reduce the computational complexity of the algorithm for the syndrome check stage, by reducing the number of operations. This is better explained in Section 5.2.2, where the complexity analysis is done.

4.3 Performance of the MPTA on low-Rate codes

In this section the performance of the MPTA on a low-rate RS code is analyzed. A (15,4) RS code is used for the simulations using BPSK modulation in an AWGN channel. The value of the correction factor used for the single δ MPTA is $\delta = 0.001$. For the single δ MPTA and increasing δ MPTA simulations are run for $I_{stag} = 100$, $I_{stag} = 200$, $I_{stag} = 300$, $I_{stag} = 400$, $I_{stag} = 500$, $I_{stag} = 600$, $I_{stag} = 700$. For the decreasing δ MPTA simulations are run for $I_{stag} = 20$, $I_{stag} = 40$, $I_{stag} = 60$, $I_{stag} = 80$, $I_{stag} = 100$, $I_{stag} = 120$, $I_{stag} = 140$. Results for the decreasing δ MPTA, the increasing δ MPTA and the single δ MPTA can be seen in Fig. 4.1, Fig. 4.2 and Fig. 4.3 respectively.

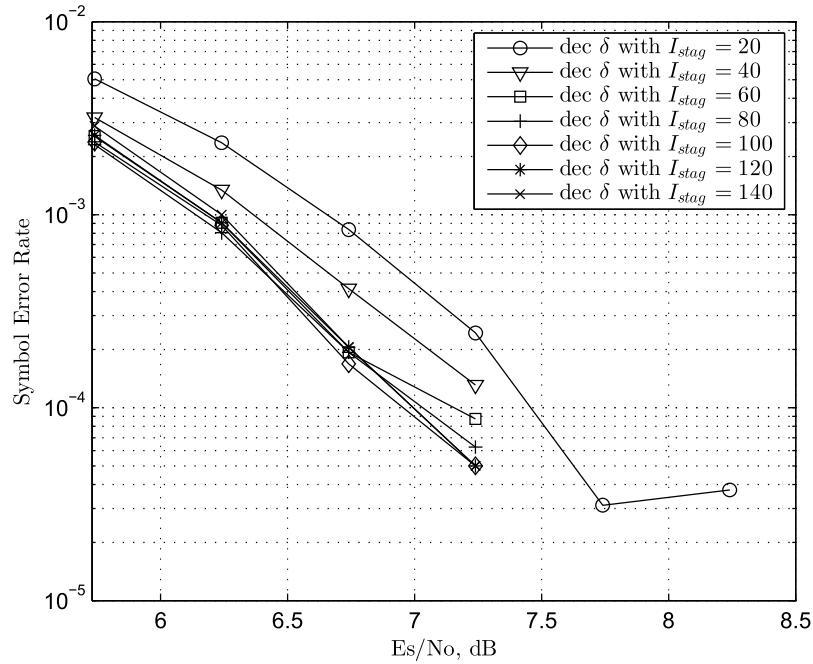


FIGURE 4.1: Performance of different values of I_{stag} with the decreasing δ MPTA on a (15,4) RS code.

From Fig. 4.1, it can be seen that the performance improves as the value of I_{stag} increases. This improvement peaks at $I_{stag} = 100$. The performance for $I_{stag} = 120$ and $I_{stag} = 140$ start to experience a loss from this point. The reason a loss is experienced for values of $I_{stag} > 100$ is because for stagnant checks performed at iterations greater than 100, the decoding process breaks after the reliabilities

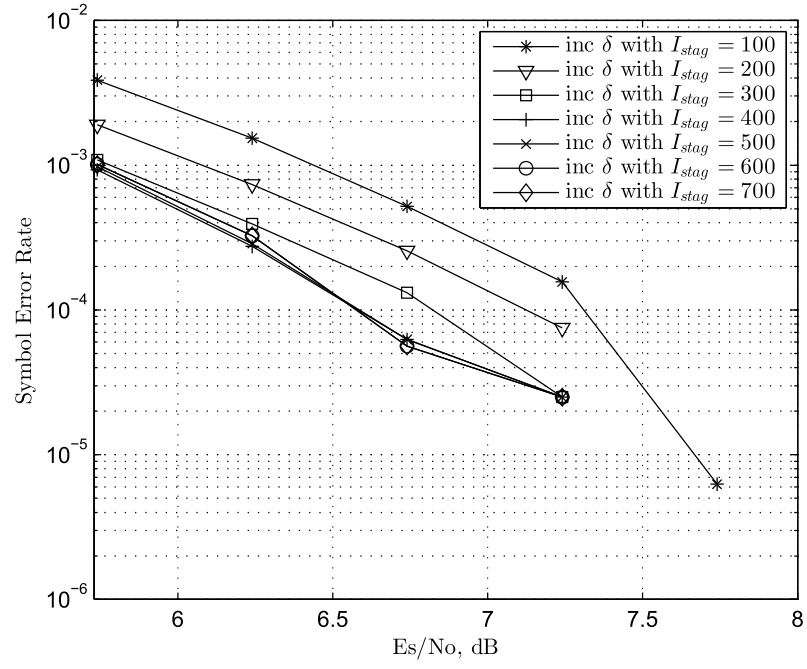


FIGURE 4.2: Performance of different values of I_{stag} with the increasing δ MPTA on a (15,4) RS code.

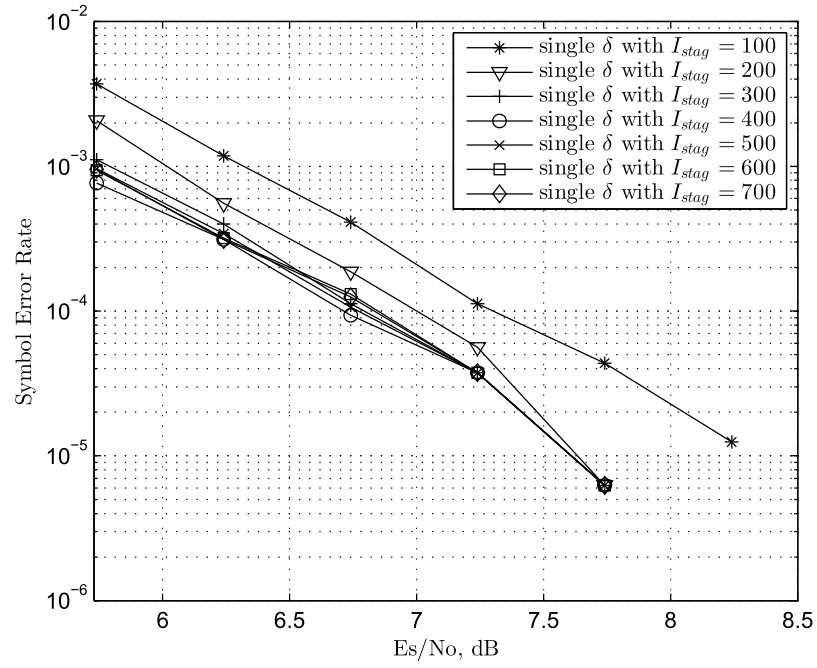


FIGURE 4.3: Performance of different values of I_{stag} with the single δ MPTA on a (15,4) RS code.

have been corrected beyond the right codeword. This means that for cases of a decoding failure, the algorithm starts to find another codeword that satisfies the syndrome checks for iterations greater than 100. Thus, to obtain minimum errors a stagnant check at 100 iterations is sufficient. With regard to this observation, the value of $I_{stag} = 100$ with the decreasing δ MPTA is used in our comparative study.

From Fig. 4.2, the increasing δ MPTA performs best when the value of $I_{stag} = 600$. The performance of the algorithm steadily improves as the value of I_{stag} increases. Based on this result, when decoding low-rate codes, the optimum stagnant check point for the increasing δ MPTA is at $I_{stag} = 600$.

In Fig. 4.3, the optimum performance of the single δ MPTA is seen when $I_{stag} = 400$. The performance of the algorithm improves from $I_{stag} = 100$ to $I_{stag} = 400$. The algorithm doesn't perform as good for higher values I_{stag} for the same reason given in the case of decreasing δ MPTA. Thus, for the single δ MPTA a value of $I_{stag} = 400$ is used in the comparative study.

4.4 Performance of the MPTA on High-Rate codes

Simulations on the MPTA used to decode a high-rate RS code is analyzed in this section. A (15,10) RS code is used for the simulations using BPSK modulation in an AWGN channel. The value of the correction factor used for the single δ MPTA is $\delta = 0.001$. For the single δ MPTA and increasing δ MPTA simulations are run for $I_{stag} = 100, I_{stag} = 200, I_{stag} = 300, I_{stag} = 400, I_{stag} = 500, I_{stag} = 600, I_{stag} = 700$. For the decreasing δ MPTA simulations are run for $I_{stag} = 20, I_{stag} = 40, I_{stag} = 60, I_{stag} = 80, I_{stag} = 100, I_{stag} = 120, I_{stag} = 140$. Results obtained for the decreasing δ MPTA, the increasing δ MPTA and the single δ MPTA can be seen in Fig. 4.4, Fig. 4.5 and Fig. 4.6 respectively.

In Fig. 4.4, The performance of the algorithm improves from $I_{stag} = 20$ to $I_{stag} = 60$. For values of $I_{stag} \geq 80$ the performance of the algorithms starts to experience a SER loss. Based on this observation, the best performing decreasing δ MPTA for the comparative analysis has a value of $I_{stag} = 60$.

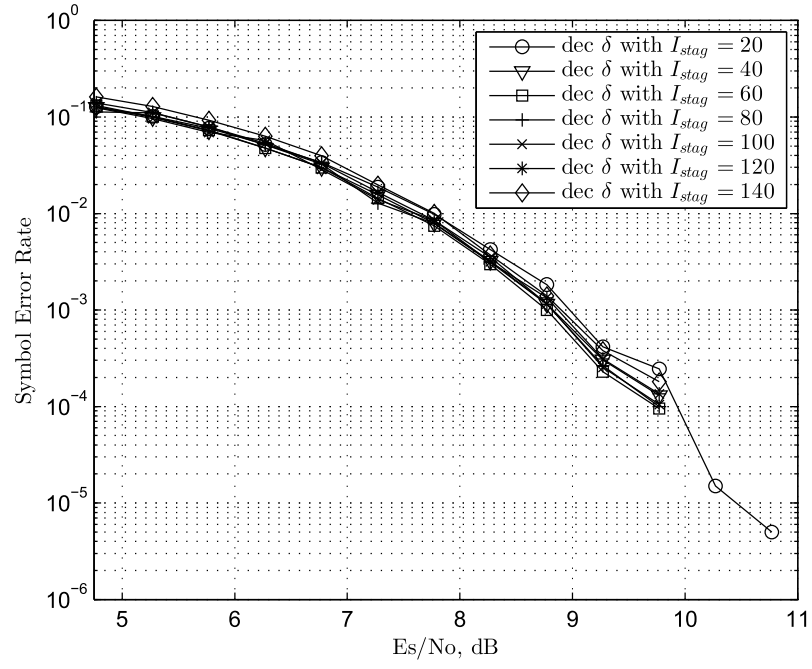


FIGURE 4.4: Performance of different values of I_{stag} with the decreasing δ MPTA on a (15,10) RS code..

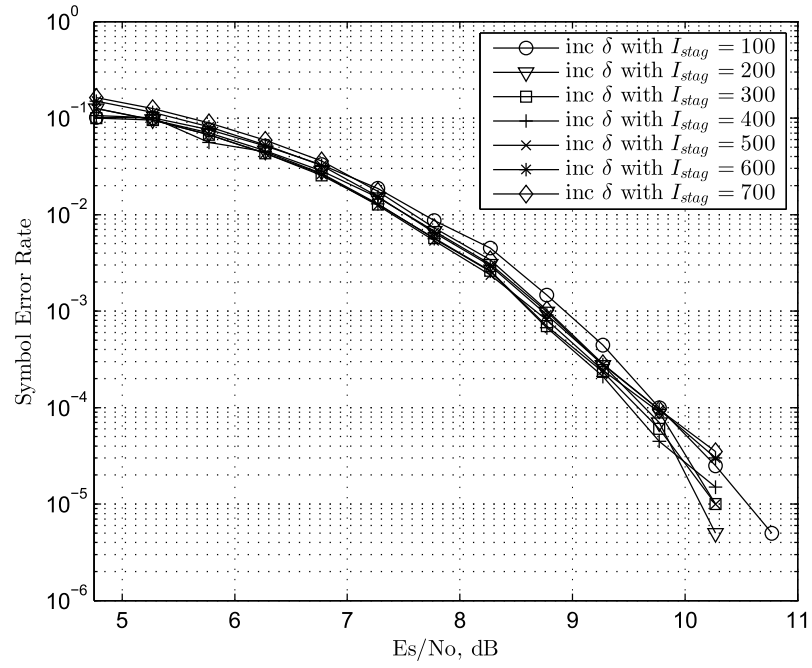


FIGURE 4.5: Performance of different values of I_{stag} with the increasing δ MPTA on a (15,10) RS code..

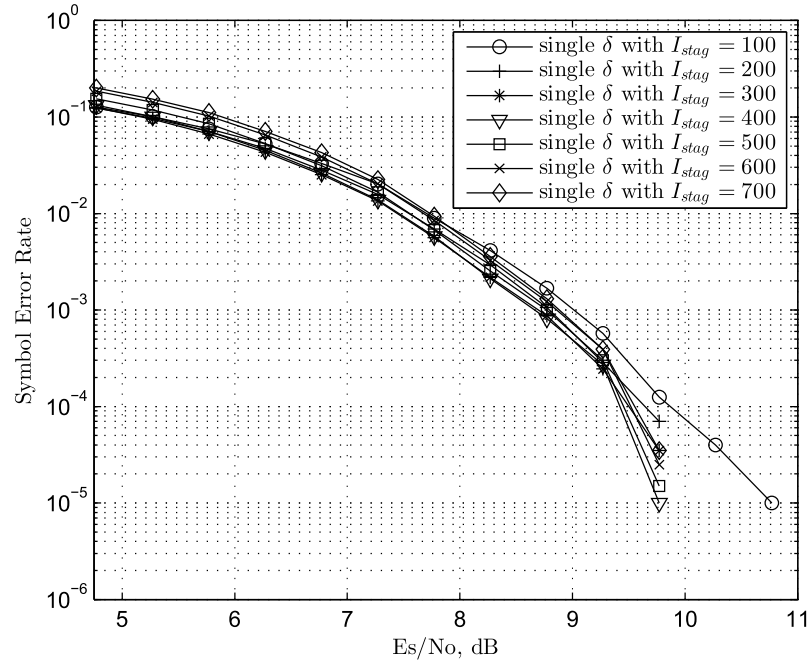


FIGURE 4.6: Performance of different values of I_{stag} with the single δ MPTA on a (15,10) RS code..

From Fig. 4.5, the optimal SER performance is exhibited when $I_{stag} = 200$, as all other values of I_{stag} experience a loss when compared this. From this result, for the increasing δ MPTA, $I_{stag} = 200$ is used in the comparative study.

For the case of the single δ MPTA on a high-rate RS code, the best performance is attained when $I_{stag} = 400$ as illustrated in Fig. 4.6. Based on this observation, the single δ MPTA with $I_{stag} = 400$ yields the best performance.

4.5 The PTA vs The MPTA for different Code rates

4.5.1 Performance comparison for Low rate codes

The performance of the MPTA, with the selected values of I_{stag} , is compared to that of the PTA in this section. Simulations for the SER performance for the PTA on a low-rate RS code are first compared to that of the decreasing δ MPTA, the increasing δ MPTA and the single δ MPTA. Simulations are done for the same

low rate (15, 4) *RS* code. These results are shown in Fig. 4.7, Fig. 4.8 and Fig. 4.9 respectively.

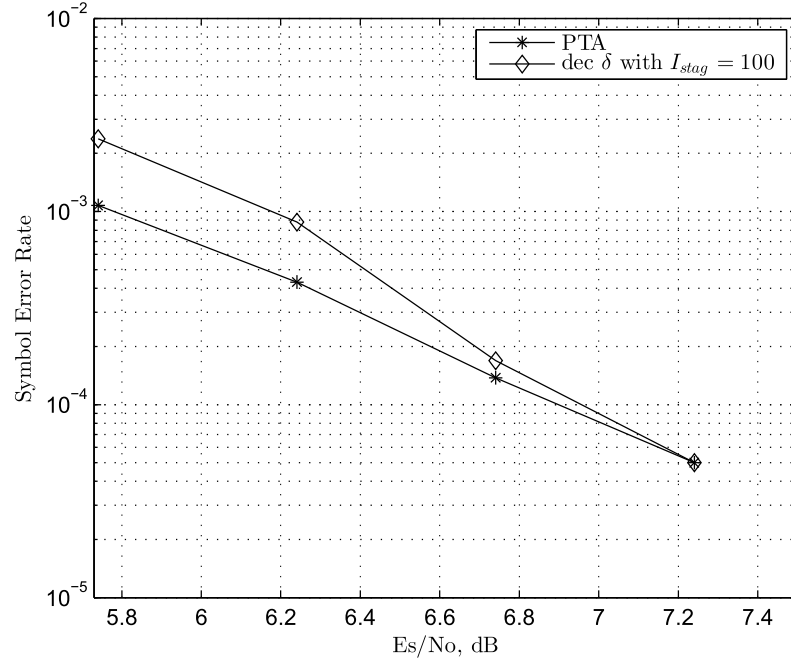


FIGURE 4.7: Performance comparison for the PTA and the decreasing δ MPTA on a (15,4) *RS* code.

It can be seen from Fig. 4.7 that the PTA outperforms the decreasing δ MPTA for low-rate codes at low SNR regions. The performance for the decreasing δ MPTA improves for higher SNR values when compared to that of the PTA. From Fig. 4.8, it can be seen that the increasing δ MPTA is also outperformed by the PTA for higher SNR values. For low SNR values the increasing δ MPTA yields a similar SER performance to that of the PTA. In Fig. 4.9, the single δ MPTA is shown to outperform the PTA for both low and high SNR values. From these results it can be seen that the PTA performs very well for low-rate codes when compared to the MPTA as it is only outperformed by the single δ version of the MPTA.

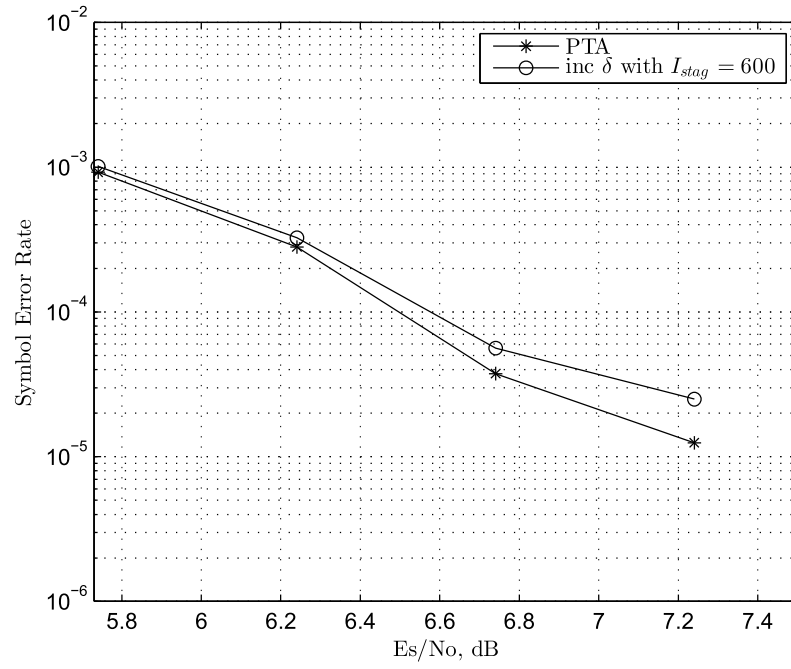


FIGURE 4.8: Performance comparison for the PTA and the increasing δ MPTA on a (15,4) *RS* code.

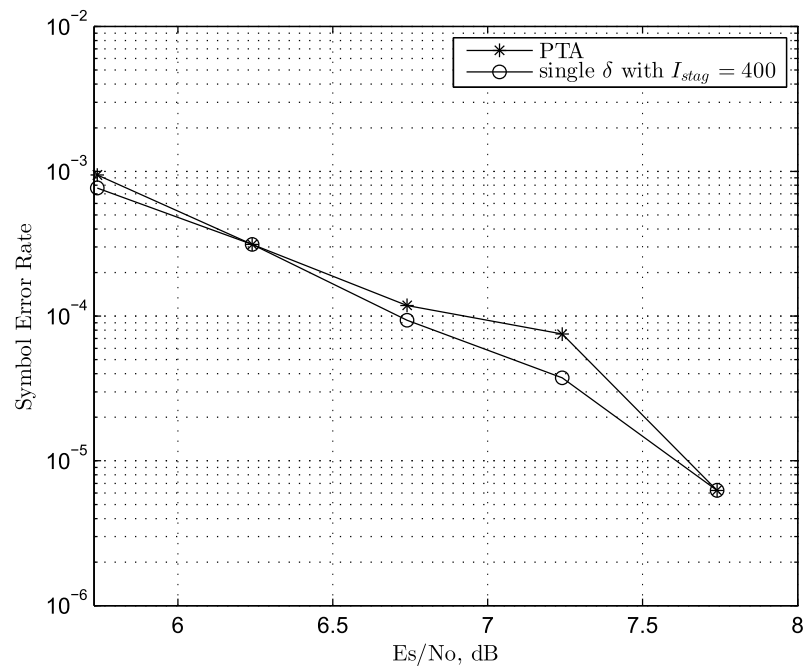


FIGURE 4.9: Performance comparison for the PTA and the single δ MPTA on a (15,4) *RS* code.

4.5.2 Performance comparison for High-rate codes

The performance comparison of the PTA and the selected values of I_{stag} with the MPTA is done for a high rate (15,10) *RS* code. The results for the different versions of the MPTA and the PTA can be seen in Fig. 4.10, Fig. 4.11 and Fig. 4.12

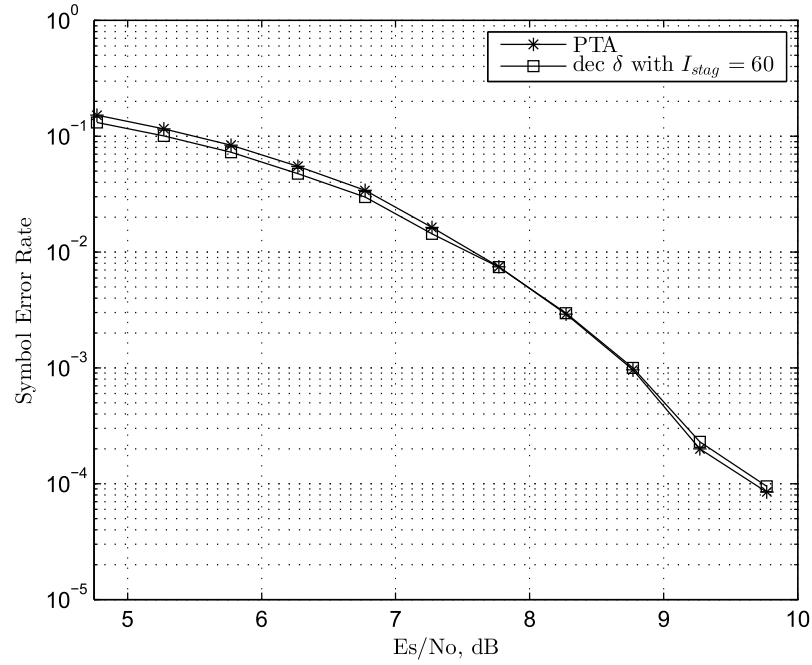


FIGURE 4.10: Performance comparison for the PTA and the decreasing δ MPTA on a (15,10) *RS* code.

From Fig. 4.10, the PTA is outperformed by the decreasing δ MPTA for low SNR values. The overall performance difference between the decreasing δ MPTA and the PTA is very small, as the waterfall graphs almost match. This result is significant because the decreasing δ runs for a considerably less number of iterations when compared to the PTA, as shown in Chapter 3. The PTA is also outperformed by the increasing δ MPTA and the single δ MPTA for the high-rate code, as shown in Fig. 4.11 and Fig. 4.12 respectively.

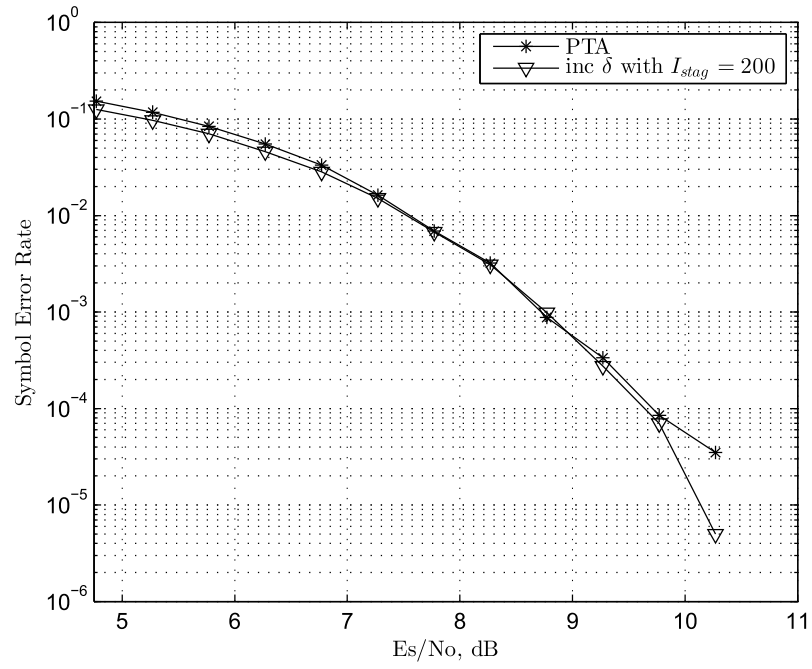


FIGURE 4.11: Performance comparison for the PTA and the increasing δ MPTA on a (15,10) RS code.

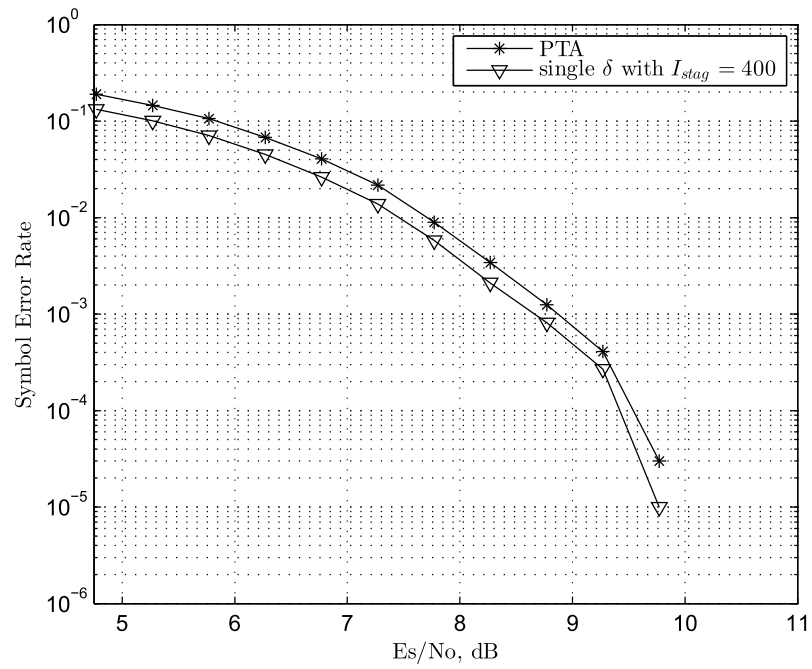


FIGURE 4.12: Performance comparison for the PTA and the single δ MPTA on a (15,10) RS code.

4.6 Conclusion

In this chapter, a final modification to the PTA is implemented. This modification is done to the syndrome check stage based on (2.16) in Chapter 2. This modification simplifies the dot product computation, at no compromise in the result obtained in the syndrome check step.

It is noted in this chapter that the optimum values of I_{stag} , for cases of a decoding failure, are the points where the algorithm is closest to finding the correct codeword before it begins finding another codeword that satisfies the syndrome checks. This is why less errors are found with this optimum values. This also explains why for most of the graphs the performance improves until this optimum value of I_{stag} , and then a loss is incurred for values of I_{stag} greater than this.

This chapter also presents an analysis of the SER performance of the PTA and the MPTA based on different code rates. The PTA is shown to perform well for lower rate codes. This is because it is only outperformed by the single δ version of the MPTA. However, for the high-rate code it is outperformed by all versions of the MPTA for low SNR values. It only slightly outperforms the decreasing δ MPTA in higher SNR regions.

It is also important to note the values of I_{stag} reduce as the code rate increases for both cases of increasing and decreasing δ MPTA. This means that the number of iterations required to decode by the increasing and decreasing δ versions of the MPTA reduce as the code rate increases. The value of I_{stag} for the single δ MPTA remains at 400 for all code rates. This means that the code rate does not affect the number of iterations required to decode for the single δ MPTA. From the results obtained for the selected code rates of the RS codes, it can be said that the single δ MPTA performs the best of all versions of the PTA algorithm.

CHAPTER 5

Performance Comparison of the Symbol wise ABP and the MPTA

5.1 Introduction

Work done in this chapter provides a comparative study on the performance of the MPTA to that of the PTA and the Symbol-level ABP. This study is done for SER performance and the computational complexity of the algorithms. As stated in Chapter 2, the Symbol-level ABP is selected for the comparative study for the purpose of giving a fair comparison. This is because the bit-level decoders work at a completely different level to that of symbol-level decoders. The fair comparison also extends to the complexity analysis done in this chapter. Bit-level decoding algorithms work at higher computational complexities when compared to symbol-level decoding algorithms as explained in Section 2.7.1, therefore, giving a biased result towards the symbol-level decoders. This is looked at for the case of the Symbol-level ABP and the Bit-level ABP in Section 5.2.1.

The rest of the chapter is organized as follows: Section 5.2 provides a complexity analysis between the PTA, MPTA and the Symbol-level ABP algorithms. SER performance is then be compared by analyzing simulations done for different code rates in Section 5.3. The chapter summary is then given in Section 5.4

5.2 Complexity Analysis

In this section a breakdown of the complexity of the Symbol-level ABP, the PTA and the MPTA is analyzed for one iteration. Except for the computation at

the syndrome check stage, the MPTA and the PTA perform exactly the same operations. Therefore, the complexity of the MPTA and the PTA is done in the same section with the computation of the syndrome check being explicitly specified. The complexity analysis is done for each stage of the decoding algorithm and this is used to obtain the overall complexity. The notation used for this section is

- H_b = The binary image expansion of the H matrix,
- b = The number bits in each symbol,
- n = The length of the codeword / The number of columns in the H matrix,
- n_b = The number of columns in the matrix H_b , $(n \times b)$,
- k = The length of the message,
- kb = The length of the bit representation of the message, $(k \times b)$,
- h = The length of the parity symbols, $(n - k)$ / The number of rows in the H matrix,
- h_b = The number of rows in the matrix H_b , $(h \times b)$,
- $\mathcal{W}_{hb}$ = The average row weight of the matrix H_b ,
- $\mathcal{W}_{nb}$ = The average column weight of the matrix H_b .

5.2.1 Complexity analysis of the Symbol-Level ABP

We first compute the complexity for the Symbol-level ABP algorithm for each stage of decoding.

1. *The Symbol reliability:* A sorting function is performed on the symbol reliabilities from the reliability matrix. Thus, a complexity of $\mathcal{O}(n)$ is incurred in this stage.

2. *Adapting of the H matrix:* This stage involves row reduction of the H matrix. The row reduction technique used in [29] is applied in this stage. The complexity attained in this stage is $\mathcal{O}(h^2)$. This method is computationally efficient when compared to Gaussian elimination. The computational complexity attained from Gaussian elimination is $\mathcal{O}(hn^3)$. This complexity becomes larger when compared to the Bit-level ABP. This is because the Bit-level ABP works with the matrix H_b , thus the complexity increases to $\mathcal{O}(h_b n_b^3)$. The row reduction technique used for the Symbol-level ABP cannot be used for the Bit-level ABP. This is due to the fact that h_b columns of the matrix H_b , matching the unreliable bits, are not guaranteed to be linearly independent [9].
3. *Binary Image expansion:* This is a direct substitution operation on the matrix H . The complexity is $\mathcal{O}(hn)$ because it is dependent on the number of elements in the H matrix.
4. *The check node updating stage:* In this stage the horizontal step of Belief Propagation is applied on the matrix H_b . From equation 2.38 the complexity of this stage is $\mathcal{O}(h_b \mathcal{W}_{hb}^2)$ [5].
5. *The variable updating stage:* The vertical step of belief propagation is applied on the matrix H_b yielding a complexity of $\mathcal{O}(\mathcal{W}_{nb} n_b)$
6. *The updating of the LLR reliabilities:* The extrinsic information is added to the initial LLR reliabilites in this stage. This computation is defined by (2.40) and (2.41). This stage gives a complexity of $\mathcal{O}(h_b n_b)$
7. *Hard Decision:* This is also a direct substitution operation on the updated LLR vector. The complexity depends on the length, n_b , of the LLR vector. Thus the complexity of this stage is $\mathcal{O}(n_b)$
8. *Satisfying the syndrome constraint:* This stage involves getting the dot product of the hard decision decoded vector of length n_b with the transpose of the matrix H_b . This yields a complexity of $\mathcal{O}(h_b n_b^2)$

Therefore, at most, the total operations done by the Symbol-level ABP for one iteration can be written as

$$\mathcal{O}_{SymABP} = \mathcal{O}(n + h^2 + hn + h_b \mathcal{W}_{hb}^2 + \mathcal{W}_{nb} n_b + h_b n_b + n_b + h_b n_b^2). \quad (5.1)$$

5.2.2 Complexity analysis of the PTA and the MPTA

The computational complexity for the PTA and the MPTA are computed similarly for every stage as follows

1. *The Symbol reliability:* The same sorting function is done to obtain the symbol reliability from the reliability matrix. Thus this stage has a complexity of $\mathcal{O}(n)$.
2. *The H matrix transformation:* The row reduction technique used in [29] is used here as well. Thus the complexity experienced in this stage is $\mathcal{O}(h^2)$.
3. *Hard Decision detection:* The maximum value for every column is obtained in this stage yielding a complexity of $\mathcal{O}(n)$.
4. *Satisfying the syndrome checks:*
 - **For the PTA:** The dot product of the hard decision vector, of length n , and the individual rows of the H matrix is done in this stage as shown in (2.42). A complexity of $\mathcal{O}(hn^2)$ is experienced here.
 - **For the MPTA:** The syndrome check for the MPTA is computed using (4.1). Therefore the complexity incurred in this stage is $\mathcal{O}(h(k^2 + 1))$. This gives a lower complexity when compared to that in the PTA. This is because for a given RS code, the value of $(k^2 + 1)$ is always less than n^2 .
5. *The correction step/Updating of the reliability matrix:* A search function is first run on the syndrome vector to find out the syndrome checks that are

satisfied and not satisfied. This yields a complexity of $\mathcal{O}(1)$. Operations done to update the reliability matrix also happen in this stage. The addition/subtraction are only done on one of the unreliable symbols and all the k reliable symbols for each row. Thus the complexity in this stage is $\mathcal{O}(h(k+1))$

6. *The stopping condition:* The same search function used in the correction step is used in this stage.

Thus, the maximum total number of operations performed by the PTA and the MPTA for one iteration can be computed as done in (5.2) and (5.3) respectively

$$\mathcal{O}_{PTA} = \mathcal{O}(n + h^2 + n + hn^2 + 1 + h(k+1)), \quad (5.2)$$

and

$$\mathcal{O}_{MPTA} = \mathcal{O}(n + h^2 + n + h(k^2 + 1) + 1 + h(k+1)). \quad (5.3)$$

5.2.3 Complexity Comparison of The ABP the PTA and the MPTA

The total computational complexity is compared based on the maximum number of operations each algorithm performs for one iteration. This analysis is done for all the 3 code rates. The value of $\mathcal{W}_{hb}$ was obtained by adapting the matrix H_b 1000 times based on the symbol reliabilities of random codewords received from an AWGN channel. The average row weight, $\mathcal{W}_{ra}$, was noted for each time H_b was adapted. The value of $\mathcal{W}_{ra}$ was then put in a row vector, rw . The vector rw has a length 1000 and stored $\mathcal{W}_{ra}$ values at the respective index. At the end of the 1000th adaption of H_b , the average value of rw was used as the value of $\mathcal{W}_{hb}$. This was done for each code rate. A similar approach was used to find the value for $\mathcal{W}_{nb}$. The value $b = 4$ is used because we are working in the field $GF(2^4)$. The rest of the actual values for the notations used in this section for the different code rates are displayed in Table 5.1

TABLE 5.1: Actual values for the notation used in the complexity Analysis

Notation	Low-rate <i>RS</i> code	Half-rate <i>RS</i> code	High-rate <i>RS</i> code
n	15	15	15
n_b	60	60	60
k	4	7	10
k_b	16	28	40
h	11	8	5
h_b	44	32	20
$\mathcal{W}_{hb}$	9.54	15.92	23.68
$\mathcal{W}_{nb}$	7.00	8.51	7.44

The compexity analysis for the low-rate, half-rate and high-rate *RS* code can be seen in Table 5.2, Table 5.3 and Table 5.4 respectively. The following notation is used in the analysis:

- O_s = The addition of the actual number of operations in each stage.
- O_i = The maximum total number of operations per iteration.

TABLE 5.2: Maximum number of operations of the ABP, PTA and the MPTA per iteration for Low-rate *RS* code

Algorithm	Operation Equations	O_s	O_i
Sym ABP	$\mathcal{O}(n + h^2 + hn + h_b \mathcal{W}_{hb}^2 + \mathcal{W}_{nb} n_b + h_b n_b + n_b + h_b n_b^2)$	$15 + 11^2 + (15 \times 11) + (44 \times 9.54^2) + (7.00 \times 60) + (40 \times 60) + 60 + (44 \times 60^2)$	165585.51
PTA	$\mathcal{O}(n + h^2 + n + hn^2 + 1 + (h(k + 1)))$	$15 + 11^2 + 15 + (11 \times 15^2) + 1 + (11 \times (4 + 1))$	2682
MPTA	$\mathcal{O}(n + h^2 + n + h(k^2 + 1) + 1 + h(k + 1))$	$15 + 11^2 + 15 + (11 \times (4^2 + 1)) + 1 + 11(4 + 1)$	394

From the results in Table 5.2, Table 5.3 and Table 5.4 it can be seen that the MPTA and the PTA are significantly less complex than the Symbol-level ABP. The Symbol-level ABP performs a large number of operations mainly because of the symbol to bit conversion. This makes the algorithm work with both a larger codeword and parity check matrix. The MPTA has also been shown to be less complex than the PTA, because of the modification at the syndrome check. This is best seen in Table 5.2, when working with low-rate codes, as it takes advantage

TABLE 5.3: Maximum number of operations of the ABP, PTA and the MPTA per iteration for Half-rate *RS* code

Algorithm	Operation Equations	O_s	O_i
Sym ABP	$\mathcal{O}(n + h^2 + hn + h_b \mathcal{W}_{hb}^2 + \mathcal{W}_{nb} n_b + h_b n_b + n_b + h_b n_b^2)$	$15 + 8^2 + (15 \times 8) + (32 \times 15.92^2) + (8.51 \times 60) + (32 \times 60) + 60 + (32 \times 60^2)$	125999.88
PTA	$\mathcal{O}(n + h^2 + n + hn^2 + 1 + h(k + 1))$	$15 + 8^2 + 15 + (8 \times 15^2) + 1 + (8 \times (7 + 1))$	1959
MPTA	$\mathcal{O}(n + h^2 + n + h(k^2 + 1) + 1 + h(k + 1))$	$15 + 8^2 + 15 + (8 \times (7^2 + 1)) + 1 + (8 \times (7 + 1))$	559

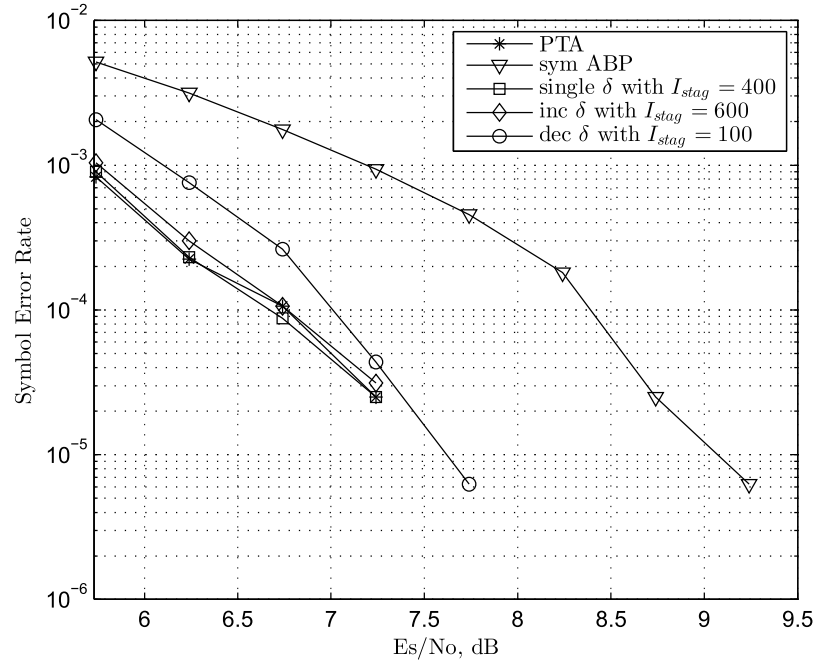
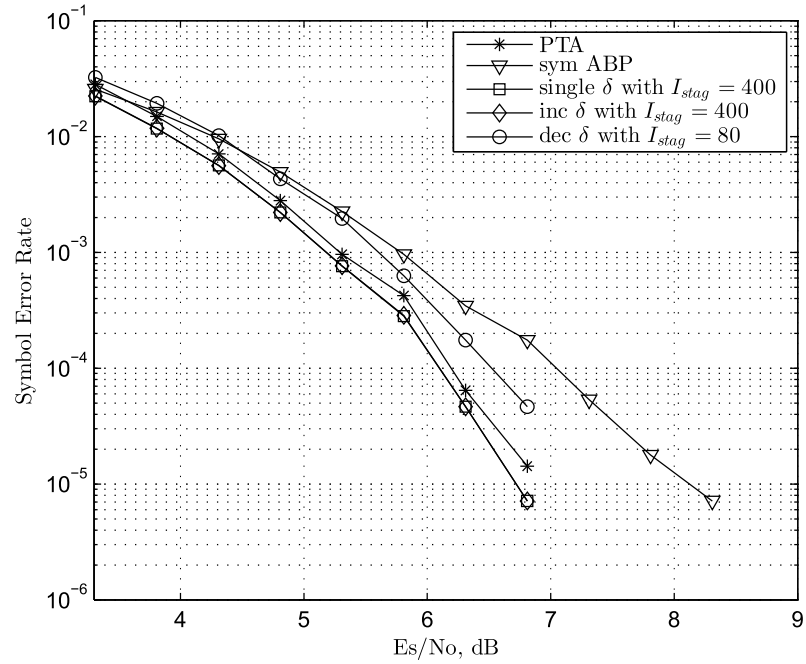
TABLE 5.4: Maximum number of operations of the ABP, PTA and the MPTA per iteration for High-rate *RS* code

Algorithm	Operation Equations	O_s	O_i
Sym ABP	$\mathcal{O}(n + h^2 + hn + h_b \mathcal{W}_{hb}^2 + \mathcal{W}_{nb} n_b + h_b n_b + n_b + h_b n_b^2)$	$15 + 5^2 + (15 \times 5) + (20 \times 23.68^2) + (7.44 \times 60) + (20 \times 60) + 60 + (20 \times 60^2)$	85036.25
PTA	$\mathcal{O}(n + h^2 + n + hn^2 + 1 + h(k + 1))$	$15 + 5^2 + 15 + (5 \times 15^2) + 1 + (5 \times (10 + 1))$	1236
MPTA	$\mathcal{O}(n + h^2 + n + h(k^2 + 1) + 1 + h(k + 1))$	$15 + 5^2 + 15 + (5 \times (10^2 + 1)) + 1 + (5 \times (10 + 1))$	616

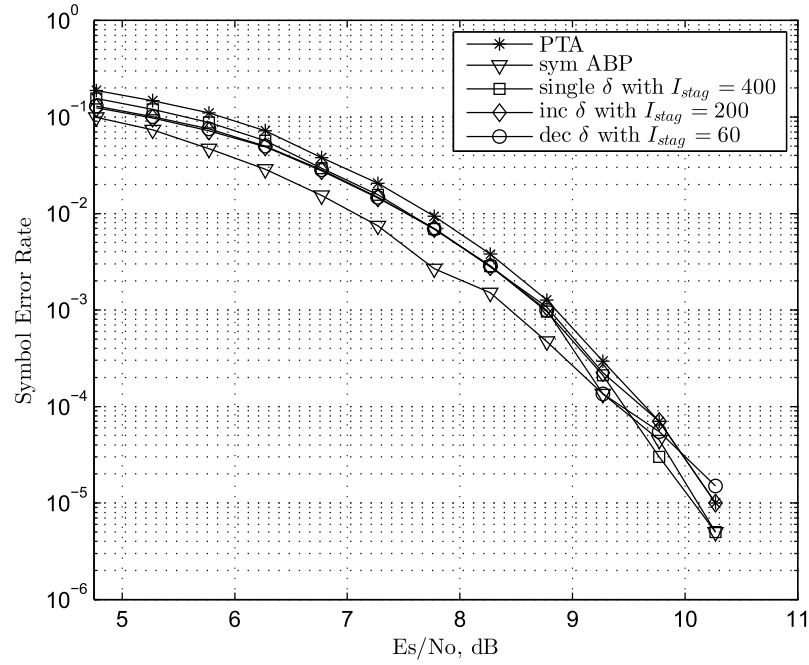
of the small value of k . However, because of this reason, the complexity of the MPTA increases as the code rate increases. The opposite happens for the PTA.

5.3 SER performance analysis

In this section a comparison of the SER performance of the Symbol-Level ABP, the PTA and the MPTA is done. Simulations are run for the three different code rates. Fig. 5.1 shows the results for a low-rate *RS* code. Results in Fig. 5.2 are used to represent the performance for a half-rate code and Fig. 5.3 shows the results for a high-rate code. The value selected for the damping coefficient used for the Symbol-level ABP in the bit reliability updating stage is $\beta = 0.12$ and maximum number of iterations, $I_{max} = 40$. This values of β and I_{max} are selected for the Symbol-level ABP as these are the values used in [9].

FIGURE 5.1: Performance comparison for a (15,4) *RS* code.FIGURE 5.2: Performance comparison for a (15,7) *RS* code.

Results in Fig. 5.1 shows the PTA outperforms all the versions of the MPTA for lower SNR values. However, the single δ MPTA is the only decoding algorithm

FIGURE 5.3: Performance comparison for a (15,10) *RS* code.

on the graph that outperforms the PTA for high SNR values. The performance gain, however, between the single δ MPTA and the PTA is quite small. The performance difference between the PTA and the MPTA algorithms is less than 0.5 dB. The Symbol-level ABP algorithm is significantly outperformed by the MPTA and the PTA algorithms. The performance gain from the MPTA when compared with the Symbol-level ABP is well over 2 dB.

A similar analysis made from the results in Fig. 3.10 can be made for Fig. 5.2. This is because the results for the PTA compared to the MPTA mirror the results shown in Chapter 3 for the half-rate code. The single δ MPTA and the increasing δ MPTA both outperform the PTA, and a performance loss of about 0.5 dB is incurred by the decreasing δ MPTA when compared to the PTA. The performance of the Symbol-level ABP improves for half-rate codes when compared to that of low-rate codes. This can be seen in that there is a performance difference of about 1.5 dB when compared to the MPTA for higher SNR values. The Symbol-level ABP also outperforms the decreasing δ MPTA for low SNR values.

From Fig. 5.3 the Symbol ABP outperforms the MPTA algorithms and the PTA

for low SNR regions. For high SNR values, it is only slightly outperformed by the single δ MPTA. This shows that the performance of the Symbol-level ABP improves as the rate of the code increases. For low SNR values the PTA is outperformed by all the algorithms but this changes as the SNR increases. The single δ MPTA once again yields the best performance when compared to the rest of the algorithms, as it is only outperformed by the Symbol-level ABP for low SNR values and it performs best for high SNR values.

5.4 Conclusion

This chapter presented a complexity analysis of the Symbol-level ABP, the PTA and the MPTA in terms of the maximum number of operations each algorithm does for one iteration. The MPTA was shown to perform significantly less operations for one iteration when compared to the PTA and the Symbol-level ABP. The Symbol-level ABP was shown to be the most computational complex algorithm in the comparative study done in this chapter. The comparative study also showed that the complexity of the MPTA increases as the code rate increases because the number of operations for the syndrome check stage is dependent on value of k . An opposite result is found for the PTA algorithm as the complexity reduces as the code rate increases. This is because the complexity of the PTA is dependent on the value of h . In spite of this, the overall complexity of MPTA at its highest is still always less than that of the PTA at its lowest. This is because of the modification done in the syndrome check stage, that gives a complexity of $\mathcal{O}(h(k^2 + 1))$ which is always less than the previous complexity of $\mathcal{O}(hn^2)$ for a given RS code.

This chapter also presents a SER performance comparison of the Symbol-level ABP to that of the MPTA and the PTA. The results in Fig. 5.2 and Fig. 5.3 has shown the Symbol-level ABP is significantly outperformed the PTA and MPTA for low-rate codes. However, the SER performance of the Symbol-level ABP improves as the code rate increases. This is especially true for the low SNR regions as it outperforms the PTA and the MPTA for the high-rate RS code. However, the performance difference for the high-rate code between the Symbol-level

ABP, the PTA and the MPTA algorithms is less than 0.5 db. This justifies a performance-complexity trade-off when using the MPTA for high-rate codes. The results obtained in this chapter for the PTA and MPTA can be used to confirm the results obtained from previous comparative studies done in Chapter 3 and Chapter 4.

CHAPTER 6

Conclusion and Future Work

6.1 Conclusion

A reduced complexity soft decision decoding algorithm is an important factor to look into in the applications for Reed-Solomon codes. Thus the primary objective of this research was to improve on the less computationally complex PTA decoding algorithm for Reed-Solomon codes. The ultimate goal was to develop an extension of the PTA that would improve the overall performance, by reducing the number of iterations the algorithm run and to give a better SER performance.

This is accomplished by the addition of various modifications to the algorithm done in Chapter 3. This included the use of absolute values in the reliability matrix. Absolute values were used to eliminate the negative values that directly affected the number of errors found in the decoded codeword. The previous early stopping condition for the PTA, which stops the decoding process when a negative value is found in the reliability matrix, is no longer applicable due to the use of absolute values. Therefore, the stagnant check is added to the PTA to prevent it from running maximum iterations in the event the algorithm is to find an incorrect codeword. The MPTA, with the stagnant check as a stopping condition, gave a better SER performance when compared to the PTA. However, this version of the MPTA gave a similar average number of iterations required for decoding as that of the PTA. The implementation of a new stopping condition for the PTA is one of the objectives for this research. The application of the stagnant check as a stopping condition for the MPTA satisfies this objective.

This work also exploited the fact that larger values of δ , that are less than 0.01,

run less iterations when used in the algorithm. The implementation of a discreet variable δ is added to the MPTA with the aim of reducing the number of iterations the algorithm runs. This modification works by changing the value of δ with respect to the number of iterations. This implementation is looked at for 2 cases, when the value of δ increases and when the value of delta δ decreases.

Work in Chapter 3 focuses on the performance of the MPTA and the PTA for a (15,7) *RS* code. This code is used to represent the performance of the algorithm with a half-rate code. For the case of the increasing δ MPTA, a similar performance to the single δ MPTA is achieved, with less iterations. For an SNR of 0 dB, the increasing δ MPTA run for less than an average of 600 iterations and the single δ MPTA run for an average of about 650 iterations. The increasing δ MPTA outperformed the PTA in terms of SER while running for a less average number of iterations. The gain attained from the increasing δ MPTA in terms of SER, when compared to the PTA, is similar to that of the single δ MPTA for the (15,7) *RS* code. For the case of decreasing δ MPTA, a significant drop in the average number of iterations is realized. The decreasing δ MPTA runs for an average number of iterations less than 200. This gives a difference of well over 400 average number of iterations when compared to that of the PTA. This, however, is at the cost of about 0.5 dB SER loss. This is not considered to be a significant loss and justifies the use of the decreasing δ MPTA when a trade-off between the SER performance and number of iterations run by the algorithm is required. The main objective for this research is to reduce the number of iterations of the algorithm without suffering from a significant performance loss. Therefore, the SER performance exhibited by both the increasing δ MPTA and the decreasing δ MPTA do not only meet this objective, but also answered the research question of this dissertation.

Work done in Chapter 4 shows the performance of the algorithm for a low rate (15,4) and a high rate (15,10) *RS* code. The results showed the that the PTA performs well for low-rate codes when compared to the MPTA. The performance gain, however, for using PTA over the MPTA for low-rate codes is less than 0.5 dB for all cases. For high-rate codes the PTA is outperformed by the single δ MPTA and the increasing δ MPTA. The decreasing δ MPTA matches the performance

of the PTA and this is at a significant drop in the number of iterations. The value of I_{stag} is also shown to reduce as the rate of the code increases for the cases of increasing and decreasing δ MPTA. This means that the number of iterations required by the discrete variable δ versions of the MPTA reduce as the rate of the code increases. The value of I_{stag} of the single δ MPTA doesn't change for different code rates. This means the number of iterations required to decode does not change with the code rate. However, this is the best performing version of the MPTA, when compared to the PTA, for all code rates. Improving the SER of the PTA, through the implementation of modifications, without any additional computational complexity is an objective of this work. This objective is satisfied by the single δ MPTA.

Chapter 5 presents a comparative study between the Symbol-level ABP and the PTA and MPTA. This study is done in terms of computational complexity and SER performance. The computational complexity is calculated based on the maximum number of operations performed by the algorithms for one iteration. The Symbol-level ABP is shown to be the most complex of the three algorithms. This is largely due to the fact that it works with a larger H matrix. The MPTA is shown have a significantly less computational complexity when compared to the PTA. This is because of the modification done on the syndrome check stage as shown in Section 5.2.2. Reducing of the computational complexity of the PTA without a significant SER performance loss is one of the objectives of this research. The modification to the syndrome check stage is able to reduce the complexity of the algorithm without affecting the result obtained from the syndrome check. This modification to the syndrome check stage meets the reduction in complexity objective.

The SER performance comparison between the Symbol-level ABP and the PTA and MPTA is also presented in Chapter 5. The Symbol-level ABP is significantly outperformed the PTA and the MPTA for the low-rate code. This result show that the Symbol-level ABP is unsuitable for low-rate applications of *RS* codes given its SER performance and computational complexity. These results also shows that the PTA and MPTA are suitable for short codes given the SER performance. This

is a promising result as short codes are increasingly becoming relevant due to the rise of their applications with the internet of things. The SER performance of the Symbol-level ABP improves as the rate of the code increases. This can be seen in Fig 5.3, where it is shown to give a better performance for low SNR values when compared to the PTA and MPTA for the high-rate code. However, the Symbol-level ABP is only outperformed by the single δ MPTA algorithm for higher SNR values for this same code rate.

The aim of this work is to improve on the PTA algorithm by reducing the number of iterations required by the algorithm without a significant performance loss and to improve on the SER performance of the PTA while reducing computational complexity. This is achieved through the development of three different versions of the MPTA algorithm. Both the increasing and decreasing δ versions of the MPTA reduced the number of iterations required by the algorithm. For the half-rate code and the high-rate code the increasing δ MPTA outperformed the PTA. For the high-rate codes the decreasing δ MPTA matched the performance of the PTA. The single δ MPTA gave the best performance as it outperformed the PTA for every code rate. This is done at a reduced computational complexity of the algorithm. However, there is no reduction in the number of iterations for single δ MPTA when compared to the PTA. The development of the three different versions of the MPTA satisfy the final objective of this research; that is, the development of an extension of the PTA algorithm that runs for less iterations without a significant SER performance loss.

6.2 Future Work

The work in this dissertation mainly focused on reducing the number of iterations of the algorithm while still giving a good SER performance. This is primarily achieved for the cases of the discrete variable δ implementation in the MPTA. The method used to switch the value of δ is sufficient, however, further work can be done in the development of an adaptive δ which would improve the performance

of the MPTA. This research would require the development of an intelligent algorithm that would be able to change the value of δ as per the channel conditions to give an optimum performance.

Further work can also be done on implementing the MPTA for other codes used in FEC. Implementation on LDPC codes would be a good area of research as these codes are widely used for various applications. All Reed-Solomon codes are characterized by matrices that are full rank. Thus, research done for this implementation would require work done on applying the MPTA on a rank deficient matrices as most LDPC codes fall under this category. This would also help give a comparison of the performance of the MPTA on a sparse matrix.

Further work can also be done on applying the MPTA to binary codes. It would be interesting to test the MPTA for a binary case as its performance has only been tested for Reed-Solomon codes, which are all nonbinary.

References

- [1] Marco Baldi and Franco Chiaraluce. A Simple Scheme for Belief Propagation Decoding of BCH and RS Codes in Multimedia Transmissions. *International Journal of Digital Multimedia Broadcasting*, vol.2008(Art.ID 957846):pages 1–12, June 2008.
- [2] E. R. Berlekamp. *Algebraic Coding Theory*. McGraw-Hill, Inc., New York, 1968.
- [3] M. Bin Ibrahim and N.B. Ahmed. SER performance of Reed-Solomon codes with QAM modulation scheme in AWGN channel. In *Signal Processing Its Applications, 2009. CSPA 2009. 5th International Colloquium on*, pages 391–394, March 2009.
- [4] L. Chen and R.A. Carrasco. Soft decoding of algebraic-geometric codes using Koetter-Vardy algorithm. *Electronics Letters*, 45(25):1334–1336, December 2009.
- [5] Li Chen. Iterative Soft Decoding of Reed-Solomon Convolution Concatenated Codes. *Communications, IEEE Transactions on*, 61(10):4076–4085, October 2013.
- [6] Gui-Liang Feng and K.K. Tzeng. A generalization of the Berlekamp-Massey algorithm for multisequence shift-register synthesis with applications to decoding cyclic codes. *Information Theory, IEEE Transactions on*, 37(5):1274–1287, Sep 1991.
- [7] Warren J. Gross, Frank R. Kschischang, Ralf Koetter, and P. Glenn Gulak. Simulation Results for Algebraic Soft-Decision Decoding of Reed-Solomon

- Codes. In *in Proceedings of the 21'st Biennial Symposium on Communications, (Queen's University)*, pages 356–360, June 2002.
- [8] V. Guruswami and M. Sudan. Improved decoding of Reed-Solomon and algebraic-geometry codes. *Information Theory, IEEE Transactions on*, 45(6):1757–1767, Sep 1999.
- [9] Jing Jiang and K.R. Narayanan. Iterative Soft-Input Soft-Output Decoding of Reed-Solomon Codes by Adapting the Parity-Check Matrix. *Information Theory, IEEE Transactions on*, 52(8):3746–3756, Aug 2006.
- [10] B. Kamali and A.H. Aghvami. Belief propagation decoding of Reed-Solomon codes; a bit-level soft decision decoding algorithm. *Broadcasting, IEEE Transactions on*, 51(1):106–113, March 2005.
- [11] R. Koetter and A. Vardy. Algebraic soft-decision decoding of Reed-Solomon codes. *Information Theory, IEEE Transactions on*, 49(11):2809–2825, Nov 2003.
- [12] Shu. Lin and Daniel J. Costello. *Error Control Coding*. Prentice Hall, 2 edition, June 2005.
- [13] R. Lucas, M. Bossert, and M. Breitbart. On iterative soft-decision decoding of linear binary block codes and product codes. *Selected Areas in Communications, IEEE Journal on*, 16(2):276–296, Feb 1998.
- [14] D.J. MacKey and R. M. Neil. Near Shannon limit performance of low density parity check codes. *Electro. Lett*, 1996.
- [15] J.L. Massey. Shift-register synthesis and BCH decoding. *Information Theory, IEEE Transactions on*, 15(1):122–127, Jan 1969.
- [16] R.J. McEliece. The Guruswami-Sudan Decoding Algorithm for Reed-Solomon Codes, 16th April 2003.
- [17] Todd K. Moon. *Error Correction Coding: Mathematical Methods and Algorithms*. John Wiley & Sons, July 2005.

-
- [18] O. Ogundile, Y. Genga, and D. Versfeld. Symbol level iterative soft decision decoder for Reed-Solomon codes based on parity-check equations. *Electronics Letters*, 51(17):1332–1333, August 2015.
- [19] O.O. Ogundile and D.J.J. Versfeld. Improved reliability information for rectangular 16-qam over flat rayleigh fading channels. In *Computational Science and Engineering (CSE), 2014 IEEE 17th International Conference on*, pages 345–349, Dec 2014.
- [20] John G. Proakis and Masoud Salehi. *Digital Communications*. McGraw-Hill Higher Education, 5 edition, Nov 2007.
- [21] I. S. Reed and G. Solomon. *Polynomial codes over Certain Finite fields*. J. Soc.Ind. Appl. Maths., 8:300-304, June 1960.
- [22] R.M. Roth and G. Ruckenstein. Efficient decoding of Reed-Solomon codes beyond half the minimum distance. In *Information Theory, 1998. Proceedings. 1998 IEEE International Symposium on*, pages 56–, Aug 1998.
- [23] R.M. Roth and G. Ruckenstein. Efficient decoding of Reed-Solomon codes beyond half the minimum distance. *Information Theory, IEEE Transactions on*, 46(1):246–257, Jan 2000.
- [24] Beomkyu Shin, Sang Hyo Kim, Jong Seon No, and Dong Joon Shin. New stopping criteria for decoding LDPC codes in H-ARQ systems. In *Information Theory and Its Applications, 2008. ISITA 2008. International Symposium on*, pages 1–5, Dec 2008.
- [25] S.Lin and W. E Ryan. *Channel Codes Classical and Modern*. Cambridge University Press, July 2009.
- [26] O. Ur-rehman and N. Zivic. Soft decision iterative error and erasure decoder for Reed-Solomon codes. *Communications, IET*, 8(16):2863–2870, 2014.

-
- [27] Vo Tam Van, S. Mita, Jing Li, Chau Yuen, and Yong Liang Guan. Bit-Level Soft-Decision Decoding of Triple-Parity Reed-Solomon Codes through Automorphism Groups. *Communications Letters, IEEE*, 17(3):553–556, March 2013.
 - [28] A. Vardy and Y. Be’ery. Bit-level soft-decision decoding of reed-solomon codes. *Communications, IEEE Transactions on*, 39(3):440–444, Mar 1991.
 - [29] D.J.J. Versfeld, J.N. Ridley, H.C. Ferreira, and A.S.J. Helberg. On Systematic Generator Matrices for Reed–Solomon Codes. *Information Theory, IEEE Transactions on*, 56(6):2549–2550, June 2010.
 - [30] R. Welch and R. Berlekamp. Error Correction for Algebraic Block Codes, Dec. 30 1986.
 - [31] S. Wicker and V. Bhargava. Reed-Solomon Codes and Their Applications. *New York: IEEE Press.*, 1994.
 - [32] S. Hirasawa Y. Sugiyama, M. Kasahara and T. Namekawa. A method for solving key equation for decoding goppa codes. *Information and Control*, 1975.