國 立 交 通 大 學

電信工程學系

碩 士 論 文

里德所羅門碼之改良式信念傳遞解碼演算法

An Improved Belief Propagation Based Decoding Algorithm for RS Codes

研究生：郭欣銓

指導教授：王忠炫 博士

中 華 民 國 九 十 八 年 七 月

里德所羅門碼之信念傳遞解碼演算法

An Improved Belief Propagation Based Decoding Algorithm for RS Codes

研究生：郭欣銓　　Student: Hsin-Chuan Kuo

指導教授：王忠炫　　Advisor: Chung-Hsuan Wang

國立交通大學

電信工程學系碩士班

碩士論文

A Thesis

Submitted to Department of Communication Engineering

College of Electrical and Computer Engineering

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Communication Engineering

July, 2009

Hsinchu, Taiwan, Republic of China

中華民國九十八年七月

里德所羅門碼之信念傳遞解碼演算法研究

研究生：郭欣銓　　　　　　　指導教授：王忠炫 博士

國立交通大學

電信工程學系碩士班

摘　　要

由於良好更正叢集錯誤的能力和有效率的代數硬式解碼演算法，里德所羅門碼 (Reed Solomon Codes) 在最近已成為許多通訊系統所選擇的錯誤更正碼。眾所皆知，里德所羅門碼的軟式決策解碼相較於傳統的硬式決策解碼能夠提供明顯的效能改善，但由於其軟式決策解碼的高複雜度，目前大部分的系統仍然使用傳統的硬式決策解碼器。Jiang 和 Narayanan 於 2006 年提出了一種利用適應性校驗矩陣，對里德所羅門碼進行軟式輸出輸入解碼的演算法 (JN 演算法)。JN 演算法能夠避免校驗節點飽和化以及低信任度位置之錯誤傳遞等問題，然而其潛在的問題在於過度信任高信任度位置的訊息。在本篇論文中，我們首先分析 JN 演算法失敗的理由。基於我們的討論，我們提出一個新的演算法來避免發生在高信任度的錯誤所對 JN 演算法帶來的影響並改進其效能。我們的演算法之構想來自於 1995 年由 Fossorier 和 Lin 所提出的 OSD 演算法。在做 JN 演算法之前，我們先對收到的向量做一些調整。如此一來，一些發生在高信任度位置的錯誤可以被減少。因此，發生在高信任度的錯誤所造成的影響也可以被減少。和原本的 JN 演算法比較，在可加性白色高斯雜訊通道並使用雙相位鍵移調變之下，我們的演算法有大約 0.5-1.2dB 的效能增進。

An Improved Belief Propagation Based Decoding Algorithm for RS Codes

Student: Hsin-Chuan Kuo Advisor: Chung-Hsuan Wang

Department of Communication Engineering

National Chiao Tung University

Abstract

Recently, Reed-Solomon (RS) codes have been the error-correction codes (ECCs) of choice in many communication systems because of their ability to correct burst errors and the availability of efficient algebraic hard decision decoding algorithms. It is known that soft-decision decoding (SDD) of RS codes provides significant performance gain over hard decision decoding (HDD), but most systems still are based on HDD because of the high complexity of SDD. In 2006, Jiang and Narayanan (JN) proposed an iterative soft-in soft-out decoding algorithm of RS codes by adapting the parity check matrix. The JN algorithm can avoid the problem of check node saturation and the error propagation from the least reliable positions, but the drawback of the JN algorithm is to over believe the messages from the most reliable positions. In this thesis, we first analyse the reason of decoding failure in the JN algorithm. Based on our discussions, we propose a new algorithm to avoid the influence of errors in most reliable positions in the JN algorithm and improve the decoding performance. The basic idea of our algorithm comes from the OSD algorithm, which was proposed in 1995 by Fossorier and Lin. We modify the received vector before the JN algorithm. Some errors in the most reliable positions may be reduced. Therefore, the influence of errors in high reliable positions can also be reduced. Compared with the original JN algorithm, the proposed algorithm has about a 0.5-1.2dB coding gain while decoding RS codes in the additive white Gaussian noise (AWGN) channel under binary phase shift keying (BPSK) modulation.

誌　謝

本篇論文得以完成，首先要感謝指導教授王忠炫博士的辛苦栽培，在兩年的過程中，教導我研究的態度和方法，並且適時地給予建議和方向。也感謝實驗室的同學、學長、學弟妹們的鼓勵和幫助，使得研究能夠順利進行，並度過了快樂的兩年研究生的生活。最後，也要謝謝一直在背後默默支持我的家人和朋友們。由衷地感謝大家。

民國九十八年七月

研究生郭欣銓謹識於交通大學

Contents

List of Figures

Chapter 1

Introduction

Reed-Solomon (RS) codes [1] are powerful error correction codes which are widely employed in many communication systems. For instance, they have been adopted as outer codes in the third-generation (3G) wireless standard, CDMA2000 high-rate broadcast packet data air interference, and are expected to be used as outer codes in concatenated coding schemes for future fourth-generation (4G) wireless systems. In most existing systems, RS codes are decoded via algebraic hard decision decoding (HDD) due to it's low complexity. However, HDD does not fully exploit the error correction capability of the code because of the loss of the soft information. For example, for the additive white Gaussian noise (AWGN) channel, the loss is believed to be 2-3dB under binary phase shift keying (BPSK) modulation. Efficiently utilizing the soft information available at the decoder input to improve the performance of RS codes is a long-time standing open problem.

The generalized minimum distance (GMD) algorithm [2] and the Chase algorithm [3] are popular soft decision decoding (SDD) algorithms to decode RS codes by modifying the least reliable positions (LRPs), then use an algebraic decoding algorithm to generate a candidate codeword. Finally, select the candidate codeword with the best metric as the decoded solution. These kinds of algorithms are called *LRP-reprocessing algorithms.* Another kinds of reliability-based soft decision decoding algorithms are referred to as *most reliable independent position (MRIP)-reprocessing algorithms.* The order statistics decoding (OSD) algorithm by Fossorier and Lin [4] is one of this kinds of algorithms. It sorts the received bits with respect to their reliabilities and reduces the columns in the generator matrix corresponding to the most reliable bits to an identity submatrix. This matrix is then used to generate codewords using the most reliable bits. Both LRP-reprocessing algorithms and MRIP-reprocessing algorithms are called reliability-based soft decision decoding algorithms.

It has been proved that maximum-likelihood decoding (MLD) of RS codes is NP-hard [5]. Therefore, it remains an open problem to find polynomial-time decoding algorithms with near ML performance. Guruswami and Sudan (GS) [7], invented a polynomial-time list decoding algorithm for RS codes capable of correcting beyond half the minimum distance of the code. Koetter and Vardy (KV) [8] developed an algebraic soft-decision decoding (ASD) algorithm for RS codes based on multiplicity assignment scheme for the GS algorithm. Alternative ASD algorithm, such as the Gaussian approximation algorithm in [9] and the algorithm proposed based on Chernoff bound [10], have better performance.

Jiang and Narayanan (JN) [11] developed an iterative algorithm based on belief propagation for soft decoding of RS codes. This algorithm compares favorably with other soft-decision decoding algorithm for RS codes and is a major step toward message passing decoding algorithm for RS codes. In the JN algorithm, belief propagation is run on an *adapted* parity check matrix, where the columns in the parity check matrix corresponding to the *least reliable independent positions* (LRIPs) are reduced to an identity submatrix. In [15], an algorithm which combines the OSD algorithm and adaptive belief propagation (ABP) such as the JN algorithm was proposed. Later in [16], the author modify the algorithm in [15] to get a better performance.

In this thesis, we propose a new decoding algorithm based on the JN and the OSD algorithm. We use the same concept as in the OSD algorithm to help the JN decoder. The outline of this thesis is as follows. Some preliminaries are given in Chapter 2. In Chpater 3, the OSD algorithm and the JN algorithm are introduced. We also reviewed the algorithm in [15] and [16]. The proposed algorithm is shown in Chpater 4 with it's simulation results. Finally, we conclude this thesis in Chapter 5.

Chapter 2

Preliminaries

In this chapter, we define the notations and system model that will be used in the following of this thesis. Then we give a brief review of RS codes. Finally, we introduce the binary image representation of RS codes.

2.1 Notations and System Model

We will use underline letters to denote vectors and bold face letters to denote matrices. For an (N, K) RS code, we use $\underline{c}_s = [c_1, c_2, ..., c_N]$ to represent it's codeword. Let $\boldsymbol{G}_s$ be it's generator matrix and $\boldsymbol{H}_s$ be the corresponding parity check matrix. In this thesis, we assume the channel is AWGN channel. The modulation scheme is binary phase shift keying (BPSK). In this scheme, 0 will be mapped to $+1$, and 1 will be mapped to -1. Let $\underline{r}_s = [r_1, r_2, \ldots, r_N]$ be the channel output and $\underline{n} = [n_1, n_2, \ldots, n_N]$ represent the noise vector with it's power spectrum density $N_0/2$. Then

$$\underline{r}_s = (-2\underline{c}_s + 1) + \underline{n}. \tag{2.1}$$

2.2 A Brief Review of RS codes

An (N, K) RS code over GF(2^m) is a kind of linear block code with it's parity check matrix

$$\boldsymbol{H}_s = \begin{bmatrix} 1 & \alpha & \ldots & \alpha^{N-1} \\ 1 & \alpha^2 & \ldots & \alpha^{2(N-1)} \\ & & \ldots & \\ 1 & \alpha^{d_{min}-1} & \ldots & \alpha^{(d_{min}-1)(N-1)} \end{bmatrix}, \tag{2.2}$$

where $d_{min} = N - K + 1$ and α is a primitive element in GF(2^m).

2.3 Binary Image Representation of RS codes

2.3.1 Binary Representation of a Symbol over GF(2^m)

Choosing $\{1, \alpha, \alpha^2, \ldots, \alpha^{m-1}\}$ as bases, any symbol $A \in$ GF(2^m) could be written as

$$A = \sum_{i=0}^{m-1} A_b^{(i)} \alpha^i, \quad \text{where} A_b^{(i)} \in \text{GF}(2). \tag{2.3}$$

Therefore, we can represent A by a binary vector $[A_b^{(0)}, A_b^{(1)}, \ldots, A_b^{(m-1)}]$.

Example: Consider in GF(2^3) with a primitive polynomial $1 + x + x^3$ and $\alpha = x$, any symbol in GF(2^3) could be represented as follows:

$$\begin{aligned} 0 &= 0 + 0 \cdot x + 0 \cdot x^2 = [0, 0, 0], \\ 1 &= 1 + 0 \cdot x + 0 \cdot x^2 = [1, 0, 0], \\ \alpha &= 0 + 1 \cdot x + 0 \cdot x^2 = [0, 1, 0], \\ \alpha^2 &= 0 + 0 \cdot x + 1 \cdot x^2 = [0, 0, 1], \\ \alpha^3 &= 1 + 1 \cdot x + 0 \cdot x^2 = [1, 1, 0], \\ \alpha^4 &= 0 + 1 \cdot x + 1 \cdot x^2 = [0, 1, 1], \\ \alpha^5 &= 1 + 1 \cdot x + 1 \cdot x^2 = [1, 1, 1], \\ \alpha^6 &= 1 + 0 \cdot x + 1 \cdot x^2 = [1, 0, 1]. \end{aligned}$$

2.3.2 Binary Representation for Additions over GF(2^m)

By the binary representation we defined above, additions over GF(2^m) could be finished by addition over GF(2).

Example: Consider in GF(2^3),

$$\begin{aligned}\alpha + \alpha^5 &= (0 + 1 \cdot x + 0 \cdot x^2) + (1 + 1 \cdot x + 1 \cdot x^2) \\ &= [0,1,0] + [1,1,1] = [1,0,1] = 1 + 0 \cdot x + 1 \cdot x^2 = \alpha^6.\end{aligned}$$

2.3.3 Binary Representation for Multiplications over GF(2^m)

In order to maintain the property of multiplication, the binary representation of a symbol to multiply another symbol should be replaced by multiplying a matrix. We give an example to explain it.

Example: Consider in GF(2^3), what's the binary representation of α^5 for multiplication?

Let

$$\alpha^5 = \begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix},$$

then

$$1 \cdot \alpha^5 = [1,0,0] \begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix} = [A,B,C] = \alpha^5 = [1,1,1].$$

$$\alpha \cdot \alpha^5 = [0,1,0] \begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix} = [D,E,F] = \alpha^6 = [1,0,1].$$

$$\alpha^2 \cdot \alpha^5 = [0,0,1] \begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix} = [G,H,I] = \alpha^7 = [1,0,0].$$

So we could use

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

to represent α^5 in multiplication. By the same manner, we can represent all the element in GF(2^3) as follows:

$$0 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{2.4}$$

$$1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

$$\alpha = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \tag{2.6}$$

$$\alpha^2 = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \tag{2.7}$$

$$\alpha^3 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{2.8}$$

$$\alpha^4 = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \tag{2.9}$$

$$\alpha^5 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \tag{2.10}$$

$$\alpha^6 = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{2.11}$$

2.3.4 Binary Image Representation of RS codes

Let $n = N \times m$ and $k = K \times m$ be the length of codeword and the information at the bit level, respectively. By the discussion above, we can transform a codeword $\underline{c}_s = [c_1, c_2, \ldots, c_N]$ of an (N, K) RS code over GF(2^m) to a binary form $\underline{c}_b = [c_1^{(0)}, c_1^{(1)}, \ldots, c_1^{(m-1)}, \ldots, \ldots, c_N^{(0)}, c_N^{(1)}, \ldots, c_N^{(m-1)}]$, where $c_i = \sum_{j=0}^{m-1} c_i^{(j)} \alpha^j$. Rewrite $\underline{c}_b$ as $\underline{c} = [c_1, c_2, \ldots, c_n]$. And we can also transform the parity check matrix

$$\boldsymbol{H}_s = \begin{bmatrix} 1 & \alpha & \ldots & \alpha^{N-1} \\ 1 & \alpha^2 & \ldots & \alpha^{2(N-1)} \\ & & \ldots & \\ 1 & \alpha^{d_{min}-1} & \ldots & \alpha^{(d_{min}-1)(N-1)} \end{bmatrix} \tag{2.12}$$

to an equivalent binary parity check matrix $\boldsymbol{H}_b$ by 2.3.3. Then $\boldsymbol{H}_b$ is an $(n-k) \times n$ binary parity check matrix.

Example: Consider (7,5) RS code over GF(2^3), it's parity check matrix

$$\boldsymbol{H}_s = \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & \alpha^1 & \alpha^3 & \alpha^5 \end{bmatrix}.$$

From (2.4) to (2.11) ,we can get the equivalent binary parity check matrix

$$\boldsymbol{H}_b = \begin{bmatrix} \mathbf{1\,0\,0} & 0\,1\,0 & \mathbf{0\,0\,1} & 1\,1\,1 & \mathbf{0\,1\,1} & 1\,1\,1 & \mathbf{1\,0\,1} \\ \mathbf{0\,1\,0} & 0\,0\,1 & \mathbf{1\,1\,0} & 0\,1\,0 & \mathbf{1\,1\,1} & 1\,0\,1 & \mathbf{1\,0\,0} \\ \mathbf{0\,0\,1} & 1\,1\,0 & \mathbf{0\,1\,1} & 1\,1\,1 & \mathbf{1\,0\,1} & 1\,0\,0 & \mathbf{0\,1\,0} \\ 1\,0\,0 & \mathbf{0\,0\,1} & 0\,1\,1 & \mathbf{1\,0\,1} & 0\,1\,0 & \mathbf{1\,1\,1} & 1\,1\,1 \\ 0\,1\,0 & \mathbf{1\,1\,0} & 1\,1\,1 & \mathbf{1\,0\,0} & 0\,0\,1 & \mathbf{0\,1\,0} & 1\,0\,1 \\ 0\,0\,1 & \mathbf{0\,1\,1} & 1\,0\,1 & \mathbf{0\,1\,0} & 1\,1\,0 & \mathbf{1\,1\,1} & 1\,0\,0 \end{bmatrix}.$$

Chapter 3

The Review of the Previous Works

In this chapter, the OSD algorithm in [4] and the JN algorithm in [11] are introduced. Then, the algorithm in [15] which combines both JN and OSD algorithm is reviewed. Right after that, the analysis of the reason of decoding failure in the JN algorithm is presented. Finally, the algorithm in [16] is also introduced.

3.1 Soft Decision Decoding Based on Order Statics

Let $\underline{r} = (r_1, r_2, \ldots, r_n)$ be the received sequence. The first step of OSD decoding algorithm is to find k *most reliable independent positions* (MRIPs) of the received sequence. We order the received sequence based on their reliability values in decreasing order. The resultant sequence is denoted by

$$\underline{r}' = (r'_1, r'_2, \ldots, r'_n), \tag{3.1}$$

with $|r'_1| > |r'_2| > \cdots > |r'_n|$. This reordering of the received symbols defines a permutation π_1 for which $\underline{r}' = \pi_1[\underline{r}]$. Let the corresponding generator matrix of the binary parity check matrix $\boldsymbol{H}_b$ is $\boldsymbol{G}_b$. We permute the columns of $\boldsymbol{G}_b$ based on π_1 and obtain the following matrix:

$$\boldsymbol{G}' = \pi_1[\boldsymbol{G}_b] = [\boldsymbol{g'_1}, \boldsymbol{g'_2}, \cdots, \boldsymbol{g'_n}], \tag{3.2}$$

where for $1 \leq i \leq n$, $\boldsymbol{g'_i}$ denotes the ith column of $\boldsymbol{G}'$. Note that code C' generated by $\boldsymbol{G}'$ is equivalent to C generated by $\boldsymbol{G}_b$. Although the first k positions of $\underline{r}'$ are the k most reliable positions, they are not necessarily independent, and therefore they do not always represent an information set. To determine the k MRIPs, we perform Gaussian eliminations to put $\boldsymbol{G}'$ in the reduced echelon form. There are k columns in $\boldsymbol{G}'$ in reduced echelon form that contain only one 1. These k columns are linearly independent. Consequently, the positions in $\underline{r}'$ that correspond to these k linearly independent columns are the k MRIPs. We use these k linearly independent columns as the first k columns of a new generator matrix $\boldsymbol{G}''$, maintaining the decreasing order of their associated reliability values. The remaining $n-k$ columns of $\boldsymbol{G}'$ in reduced echelon form give the next $n-k$ columns of $\boldsymbol{G}''$ arranged in order of decreasing associated reliability values. This process defines a second permutation π_2. It is clear that the code generated by $\boldsymbol{G}''$ is

$$C'' = \pi_2[C'] = \pi_2[\pi_1[C]]. \tag{3.3}$$

Rearranging the components of $\underline{r}'$ according to the permutation π_2, we obtain the sequence

$$\underline{y} = (y_1, \cdots, y_k, y_{k+1}, \cdots, y_n), \tag{3.4}$$

with $|y_1| > |y_2| > \cdots > |y_k|$, and $|y_{k+1}| > \cdots > |y_n|$. It is clear that $\underline{y} = \pi_2[\underline{r}'] = \pi_2[\pi_1[\underline{r}]]$.

We can permute the rows of $\boldsymbol{G}''$ to obtain a generator matrix $\boldsymbol{G}_1$ in systematic form,

$$\boldsymbol{G}_1 = [\boldsymbol{I}_k \boldsymbol{P}] = \begin{bmatrix} 1 & 0 & \cdots & 0 & p_{1,1} & \cdots & p_{1,n-k} \\ 0 & 1 & \cdots & 0 & p_{2,1} & \cdots & p_{2,n-k} \\ \vdots & & \ddots & \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 & p_{k,1} & \cdots & p_{k,n-k} \end{bmatrix}, \tag{3.5}$$

where $\boldsymbol{I}_k$ represents the $k \times k$ identity matrix, and $\boldsymbol{P}$ is the $k \times (n-k)$ parity-check matrix. Because the first k bits of $\underline{y}$ are the k most reliable independent bits, their hard decisions should contain very few errors. Based on this concept, the OSD algorithm generates a sequence of candidate codewords for testing by processing the k most reliable independent bits of $\underline{y}$. The candidate codeword $\underline{v}^*$ with the least correlation discrepancy with $\underline{y}$ is the decoded codeword. Then, $\pi_1^{-1}[\pi_2^{-1}[\underline{v}^*]]$ gives the decoded codeword in C. For $0 \le i \le k$, the OSD algorithm of order-i executes the following steps:

OSD Algorithm

Step 1. Perform hard-decision decoding of the k most reliable independent bits of $\underline{y}$ (the first k bits of $\underline{y}$). These k hard decisions give k binary digits, which form an information sequence $\underline{u}_0$.

Step 2. Construct the codeword $\underline{v}_0 = \underline{u}_0\boldsymbol{G}_1$ for the information sequence $\underline{u}_0$, and compute the distance of $\underline{v}_0$ with respect to $\underline{y}$.

Step 3. For $1 \leq l \leq i$, make all possible changes of l of the k most reliable bits in $\underline{u}_0$. For each change, form a new information sequence $\underline{u}$. Generate its corresponding codeword $\underline{v} = \underline{u}\boldsymbol{G}_1$. Compute the distance for each generated codeword. Record the codeword $\underline{v}_{best}$ that has the least distance. This step is referred to as the phase-l reprocessing of $\underline{u}_0$. It requires generating $\binom{k}{l}$ candidate codewords.

Step 4. Start the next reprocessing phase and continue to update $\underline{v}_{best}$ until the ith reprocessing phase is completed. The recorded codeword $\underline{v}_{best}$ is the decoded codeword.

Example: Consider a $(7, 4)$ code with

$$\boldsymbol{G} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}, \tag{3.6}$$

let

$$\underline{r} = (1.2, -0.01, 10, 4.3, -5.2, -6.6, -7), \tag{3.7}$$

then

$$\underline{r}' = (10, -7, -6.6, -5.2, 4.3, 1.2, -0.01). \tag{3.8}$$

The corresponding

$$\boldsymbol{G}' = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}. \tag{3.9}$$

Then

$$\boldsymbol{G}'' = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}, \tag{3.10}$$

$$\underline{y} = (10, -5.2, -6.6, -7, 4.3, 1.2, -0.01). \tag{3.11}$$

Perform hard-decision decoding, we can get

$$\underline{u}_0 = (0, 1, 1, 1). \tag{3.12}$$

$$\underline{v}_0 = \underline{u}_0 \boldsymbol{G}_1 = (0, 1, 1, 1, 1, 0, 0), \boldsymbol{d}_0 = 195.15. \tag{3.13}$$

In phase-1 reprocessing,

$$\underline{u}_1 = (0, 1, 1, 0), \underline{v}_1 = (0, 1, 1, 0, 0, 1, 1), \boldsymbol{d}_1 = 210.71. \tag{3.14}$$

$$\underline{u}_2 = (0, 1, 0, 1), \underline{v}_2 = (0, 1, 0, 1, 0, 0, 1), \boldsymbol{d}_2 = 204.31. \tag{3.15}$$

$$\underline{u}_3 = (0, 0, 1, 1), \underline{v}_3 = (0, 0, 1, 1, 0, 1, 0), \boldsymbol{d}_3 = 203.55. \tag{3.16}$$

$$\underline{u}_4 = (1, 1, 1, 1), \underline{v}_4 = (1, 1, 1, 1, 1, 1, 1), \boldsymbol{d}_4 = 239.91. \tag{3.17}$$

Therefore, the decoded codeword is $\underline{v}_0 = (0, 1, 1, 1, 1, 0, 0)$.

3.2 Discussions of the OSD Algorithm

The OSD algorithm of order-i consists of $(i + 1)$ reprocessing phases and requires processing of a total of

$$1 + \binom{k}{1} + \cdots + \binom{k}{i} \tag{3.18}$$

candidate codewords to make a decoding decision. The OSD algorithm of order-k is MLD, which requires processing of 2^k codewords. This not what we want. As pointed out earlier, the k first bits of $\underline{y}$ are the k most reliable independent bits, and their hard decisions most likely contain very few errors. That is to say that the information sequence $\underline{u}_0$ contains very few errors. Consequently, making all possible changes of a small number of positions of $\underline{u}_0$ most likely will produce the ML codeword. Therefore, an OSD algorithm with a small order-i should practically achieve the MLD error performance.

3.3 Iterative Decoding Algorithm by Adapting the Parity Check Matrix

Let $\underline{c} = [c_1, c_2, \ldots, c_n]$ be the binary representation of an RS codeword. The received vector is given by

$$\underline{r} = (-2\underline{c} + 1) + \underline{n}. \tag{3.19}$$

Thus, the initial reliability of each bit in the received vector can be expressed in terms of the log-likelihood ratios (LLR) as observed from the channel:

$$L^{(0)}(c_i) = \log \frac{P(c_i = 0|r_i)}{P(c_i = 1|r_i)}. \tag{3.20}$$

The JN algorithm is composed of two stages: the matrix updating stage and the bit-reliability updating stage. In the matrix updating stage, the magnitude of the received LLR's $|L(c_i)|$ are first sorted and let $i_1, i_2, \ldots, i_{N-K}, \ldots, i_n$ denote the position of the bits in terms of ascending order of $|L(c_i)|$, i.e., the bit c_{i_1} is the least reliable and c_{i_n} is the most reliable. Begin with the original parity check matrix $\boldsymbol{H}_b$ and first reduce the i_1^{th} column of $\boldsymbol{H}_b$ to a form $[1, 0 \ldots, 0]^T$. Then we reduce the i_2^{th} column of $\boldsymbol{H}_b$ to a form $[0, 1, 0 \ldots 0]^T$ and so on. It can be guaranteed to proceed until the $i_{(N-K)}^{th}$ column, since there are at least $(N - K)$ independent columns in $\boldsymbol{H}_b$. Then we try to reduce the i_{N-K+1}^{th} column to $[\underbrace{0, \ldots, 0}_{N-K}, 1, 0, \ldots, 0]^T$. However, there is no guarantee we can do this. If we are unable to do so, we will leave that particular column and try to reduce $i_{(N-K+2)}^{th}$ column to the above

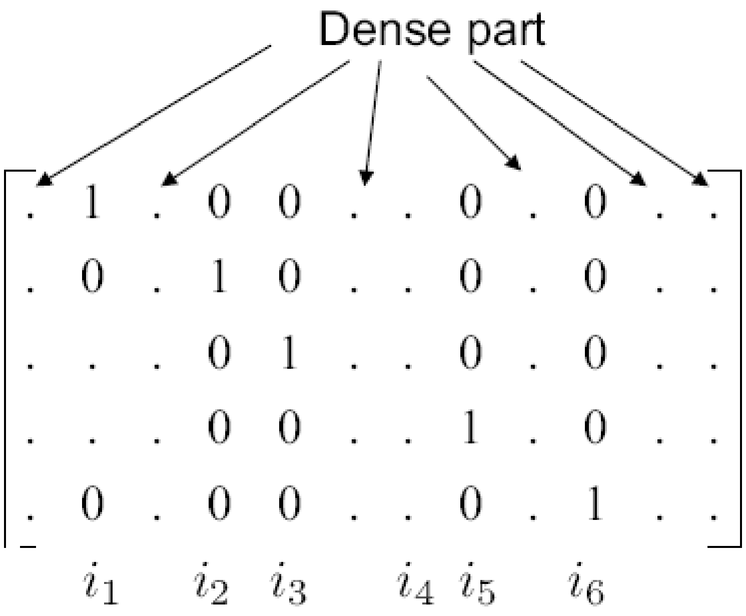

Figure 3.1: Form of the Parity Check Matrix Suitable for Iterative Decoding Obtained through Row Operations

form and so on. Finally, we can reduce $(n-k)$ columns among the n columns of $\boldsymbol{H}_b$ to be the identity matrix, since the matrix is $(n-k) \times n$ and is full rank. The matrix is thus reduced to a form as shown in Fig. 3.1. We denote the set of unreliable bits corresponding to the sparse submatrix as $\underline{B}_L$.

The JN algorithm is iterative and during the l^{th} iteration, we have a vector of LLR's as:

$$\underline{L}^{(l)} = [L^{(l)}(c_1), L^{(l)}(c_2), \ldots, L^{(l)}(c_n)], \tag{3.21}$$

where initially $\underline{L}^{(0)}$ is determined from the channel output. Then, the parity check matrix is reduced to a desired form based on $\underline{L}^{(l)}$:

$$\boldsymbol{H}_b^{(l)} = \phi(\boldsymbol{H}_b, |\underline{L}^{(l)}|). \tag{3.22}$$

In the bit-reliability updating stage, the extrinsic LLR vector $\underline{L}_{ext}^{(l)}$ is first generated according to $\underline{L}^{(l)}$ using the SPA based on the adapted parity check matrix $\boldsymbol{H}_b^{(l)}$:

$$\underline{L}_{ext}^{(l)} = \psi(\boldsymbol{H}_b^{(l)}, \underline{L}^{(l)}). \tag{3.23}$$

That is for each bit, the extrinsic LLR is updated according to:

$$L_{ext}^{(l)}(c_i) = \sum_{\substack{j=1 \\ H_{ji}^{(l)}=1}}^{n-k} 2\tanh^{-1}\Big(\prod_{\substack{p=1 \\ p\neq i, H_{jp}^{(l)}=1}}^{n} \tanh\big(\frac{L^{(l)}(c_p)}{2}\big)\Big) \tag{3.24}$$

The bit-reliability is then updated as:

$$\underline{L}^{(l+1)} = \underline{L}^{(l)} + \alpha \underline{L}_{ext}^{(l)} \tag{3.25}$$

where $0 < \alpha \leq 1$ is a damping coefficient. This is continued until a predetermined number of times $l_{max} = N_1$ or until all the checks are satisfied. A detailed description of the algorithm is given as follows:

JN Algorithm

Step 1. Initialization: set α, $l_{max} = N_1$, $l = 0$ and the LLR's for the coded bits from the channel observation: $\underline{L}^{(0)} = \frac{2}{\sigma^2}\underline{r}$.

Step 2. Reliability based parity check matrix adaption: $\boldsymbol{H}_b^{(l)} = \phi(\boldsymbol{H}_b, |\underline{L}^{(l)}|)$.

a) Order the coded bits according to the absolute value of the LLR's $|\underline{L}^{(l)}|$ and record the ordering indices.

b) Implement Gaussian elimination to systematize the $(n-k)$ unreliable positions which are independent in the parity check matrix.

Step 3. Extrinsic information generation: Apply SPA to generate the extrinsic LLR for each bit using the adapted parity check matrix $\boldsymbol{H}_b^{(l)}$: $\underline{L}_{ext}^{(l)} = \psi(\boldsymbol{H}_b^{(l)}, \underline{L}^{(l)})$ (according to 3.24)

Step 4. Bit-level reliabilities update: $\underline{L}^{(l+1)} = \underline{L}^{(l)} + \alpha \underline{L}_{ext}^{(l)}$, where $0 < \alpha \leq 1$.

Step 5. Hard decision: $\hat{c}_i = \begin{cases} 0, & L^{(l+1)}(c_i) > 0; \\ 1, & L^{(l+1)}(c_i) < 0. \end{cases}$

Step 6. Termination criterion: If all the checks are satisfied, output the estimated bits; else if $l = l_{max}$, declare a decoding failure; otherwise set $l \leftarrow l + 1$ and go to **Step2.** for another iteration.

3.4 Geometric Interpretation of the JN Algorithm

Define the operator $\nu : [-\infty, +\infty] \rightarrow [-1, 1]$ as a mapping from the LLR domain to tanh domain:

$$\nu(L) = \tanh(\frac{L}{2}) = \frac{e^L - 1}{e^L + 1} \tag{3.26}$$

where the mapping is one-to-one and onto. The inverse operator $\nu^{-1} : [-1, 1] \rightarrow [-\infty, +\infty]$ can be expressed as:

$$\nu^{-1}(t) = \ln(\frac{1+t}{1-t}) , \quad t \in [-1, +1]. \tag{3.27}$$

Apply the one-to-one tanh transform on the LLR's and get the reliability measure of the received signal in the tanh domain as:

$$\underline{T} = [T_1, T_2, \ldots, T_n] = [\nu(L(c_1)), \ldots, \nu(L(c_n))] \tag{3.28}$$

As in [12], we can measure the reliability of the j^{th} parity check node as:

$$r_j = \prod_{\substack{p=1 \\ H_{jp}=1}}^{n} \nu(L(c_p)) \tag{3.29}$$

Definition 1 *Define the potential function J as:*

$$J(\boldsymbol{H}_b, \underline{T}) = -\sum_{j=1}^{(n-k)} r_j = -\sum_{j=1}^{(n-k)} \prod_{\substack{p=1 \\ H_{jp}=1}}^{n} T_p \tag{3.30}$$

where J is a function of both the parity check matrix $\boldsymbol{H}_b$ and the received soft information $\underline{T}$.

The operator ν maps the original $n-$dimensional unbounded real space into an $n-$dimensional cube (since the output of tanh function is confined to $[-1, 1]$). The potential function J is minimized iff a valid codeword is reached, that is all the checks are satisfied and $|T_j| = 1$ for $j = 1, \ldots, n$, where $J_{min} = -(n-k)$. Besides, points with all $|T_j| = 1$ correspond to vertices of the $n-$dimensional cube. Therefore, valid codewords correspond to the vertices of the $n-$dimensional cube at which the potential function has the minimum value of $-(n-k)$. The decoding problem can be interpreted as searching for the most probable minimum potential vertex given the initial point observed from the channel.

Note that the potential function J is minimized iff a valid codeword is reached. It is quite natural to apply the gradient descent algorithm to search for the minimum potential vertex, with the initial value $\underline{T}$ observed from the channel. Consider the gradient of J with respect to the received vector $\underline{T}$. From (3.30), it can be seen that:

$$\nabla J(\boldsymbol{H}_b, \underline{T}) = (\frac{\partial J(\boldsymbol{H}_b, \underline{T})}{\partial T_1}, \frac{\partial J(\boldsymbol{H}_b, \underline{T})}{\partial T_2}, \ldots, \frac{\partial J(\boldsymbol{H}_b, \underline{T})}{\partial T_n}) \tag{3.31}$$

where the component wise partial derivative with respect to T_i is given by:

$$\frac{\partial J(\boldsymbol{H}_b, \underline{T})}{\partial T_i} = -\sum_{\substack{j=1 \\ H_{ji}=1}}^{(n-k)} \prod_{\substack{p=1 \\ p\neq i, H_{jp}=1}}^{n} T_p \tag{3.32}$$

Thus, the gradient descent updating rule can be written as:

$$\underline{T}^{(l+1)} \leftarrow \underline{T}^{(l)} - \alpha \nabla J(\boldsymbol{H}_b^{(l)}, \underline{T}^{(l)}) \tag{3.33}$$

where α is a damping coefficient as in standard gradient descent algorithms to control the step width.

Note that the reliabilities in tanh domain are confined to $T_i \in [-1, 1]$. However, the updating rule (3.33) does not guarantee this. Therefore, we use the following modified updating rule to guarantee that the updated T_i's $\in [-1, 1]$:

$$T_i^{(l+1)} \leftarrow \nu(\nu^{-1}(T_i^{(l)}) - \alpha[-\sum_{H_{ji}^{(l)}=1} \nu^{-1}(\prod_{p\neq i, H_{jp}^{(l)}=1} T_p^{(l)})]) \tag{3.34}$$

where $\nu^{-1}(x) = 2\tanh^{-1}(x)$. It can be seen that the above non-linear updating rule is exactly the same as Step3 - Step4 in the description of the JN algorithm.

When iterative decoding is applied to an HDPC code, with very high probability, the iterative algorithm will reach some local minimum points where $\nabla J(\boldsymbol{H}_b, \underline{T})$ is zero or is close to zero (since a few unreliable symbols will render the components of $\nabla J(\boldsymbol{H}_b, \underline{T})$ to be small or close to zero). We refer to these as pseudo-equilibrium points since gradient descent gets stuck at these points while these points do not correspond to valid codewords.

From (3.30), we observe that since J is also a function of $\boldsymbol{H}_b$, different choices of the parity check matrices $\boldsymbol{H}_b$, though span the same dual space, result in different potential functions J. More importantly, each $\boldsymbol{H}_b$ results in a different gradient $\nabla J(\boldsymbol{H}_b, \underline{T})$. The JN algorithm exploits this fact and when a pseudo equilibrium point is reached, by adapting the parity check matrix based on the bit reliabilities, switches to another $\boldsymbol{H}_b$ such that it allows the update in (3.34) to proceed rather than getting stuck at the pseudo-equilibrium point. However, note that the potential function that we want to minimize does not involve the Euclidean distance between the received codeword and current estimate at all. That is, the adaptive algorithm attempts merely to find a codeword that satisfies all the parity checks, without really enforcing that it be the one at minimum distance from the received word. Since small steps are taken in the gradient descent, very often we converge to the codeword at small distance from the received vector as well. However, there is no guarantee of convergence to the nearest codeword.

3.5 Variations to the JN Algorithms

In this section, several variations of the JN algorithm are discussed either to improve the performance or to reduce the decoding complexity.

3.5.1 Degree-2 Random Connection

One problem with the JN algorithm is that since each bit in the unreliable part $\underline{B}_L$ participates in only one check, it receives extrinsic information from one check only. If there is a bit error in the dense part participating in that check, the bit in $\underline{B}_L$ tends to be flipped and the decoder tends to converge to a wrong codeword. To overcome this drawback, we can reduce the matrix $\boldsymbol{H}_b$ to a form where the submatrix corresponding to the less reliable bits is sparse (say column weight 2 rather than 1). This can improve the performance since each less reliable bit now receives more extrinsic information while the submatrix corresponding to the unreliable bits still does not form any loops (i.i., there is no loop involving only unreliable bits). We can obtain this via a degree-2 random connection algorithm. The details are presented as follows:

Deg-2 Random Connection Algorithm

Step1. Apply Gaussian elimination to the parity check matrix and obtain an identity matrix in the unreliable part.

Step2. Generate a random permutation of numbers from 1 to $n-k$. Record all the indices, i.e., $p_1, p_2, \ldots, p_{n-k}$.

Step3. Adapt the parity check matrix according to the follow rule: add p_{i+1}^{th} row to p_i^{th} row, for $i = 1$ to $n-k-1$.

After the Deg-2 random connection , all the $(n-k-1)$ columns in the parity check matrix are of degree-2 except the p_1^{th} column. The last column p_1 can be left on degree-1, which will not significantly affect the performance. This appears to improve the performance of the JN algorithm in the high SNR's.

3.5.2 Various Grouping of Unreliable Bits

Another variation that can help to further imrpove the performance is to run the JN algorithm several times each time with the same initial LLR's from the channel but a different grouping of the less reliable bits. It is possible that some bits with $|L^{(l)}(c_i)|$ closes to those in the unreliable set $\underline{B}_L$ are also of the wrong sign and vice-versa. Hence, we can run the JN algorithm several times each time with a different grouping of the less reliable bits.

That is, we can swap some bits in the reliable part with those in the unreliable part near the boundary and run the matrix adaptation all over again, which gives a new $\boldsymbol{H}_b$. We then run the JN algorithm on that new matrix $\boldsymbol{H}_b$. Each time the JN algorithm is run, a different estimate codeword may be obtained due to the difference in the parity check matrix $\boldsymbol{H}_b$. All the returned codewords are kept in a list and finally the one that minimizes Euclidean distance from the received vector is chosen. This method can significantly improve the asymptotic performance, but also increases the worst case complexity.

3.5.3 Incorporated Hard Decision Decoding

A hard decision decoder can be used during each iteration in the proposed algorithm to improve the performance and accelerate decoding as well. Since the HDD may return a codeword which is different from the ML codeword, we do not stop the decoder once a codeword is returned by the HDD. Rather, we still iterate up to a maximum number of iterations to obtain all the codewords returned by HDD during each iteration and finally pick up the most likely codeword. This guarantees to perform no worse than the JN algorithm or HDD. Combining the adaptive scheme with other SIHO algorithms such as the KV algorithm has been investigated in [13].

3.5.4 Partial Reliable Bits Updating

The complexity in the bit level reliabilities update part can be further reduced via "partial reliable bits updating" scheme. The main floating-point operation complexity comes from the computation of the extrinsic information in the reliable part (where the submatrix is dense). However, in the adaptation of the parity check matrix, only some bits in the boundary will be switched from the reliable part to the unreliable part. Therefore, in the bit reliability updating stage, we only update the bits in the unreliable set $\underline{B}_L$ and some reliable bits with $|L^{(l)}(c_j)|$ close to those in the unreliable set $\underline{B}_L$. The number of bits in the reliable part M can be adjusted to control the complexity.

3.5.5 Symbol-level Adaption

Gaussian elimination requires serial update of the rows and is difficult to parallelize. Let $S_L = i_1, i_2, \ldots, i_{(N-K)}$ be as a set of $(N-K)$ least reliable symbols. In order to update the parity check matrix at the symbol level, we need to find a valid parity check matrix for which the sumbatrix corresponding to the symbols in S_L is an identity matrix. The detailed procedure is as follows: first, the submatrix corresponding to the symbols in S_L is filled with an $(N-K) \times (N-K)$ identity matrix and the rest of the matrix with unknown symbols in the parity check matrix is equivalent to finding $(N-K)$ valid codewords of the dual code which will be the rows of the parity check matrix for the original code. For the j^{th} row, the i_j^{th} entry is 1 and the $i_1^{th}, i_2^{th}, \ldots, i_{j-1}^{th}, i_{j+1}^{th}, \ldots, i_{N-K}^{th}$ entries are 0s and all other entries are erasures E (i.e., all the positions corresponding to the reliable symbols are erased). Since the dual code is an $(N, N-K)$ RS code with $d_{min} = K+1$ and there are exactly K erasures in each row, Forney's algorithm [14] can be used to compute the values in the erased positions. Each decoded codeword corresponds to one row in the original parity check matrix. By repeating this procedure for all $(N-K)$ rows, we can get a systematic parity check matrix over GF(2^m), where the submatrix corresponding to unreliable symbols is the identity matrix. Using the binary expansion, we can then get the binary parity check matrix and thereafter apply the SPA using it. Unlike Gaussian elimination, each element in the parity check matrix can be computed independently and, hence, the whole procedure can be parallelized. This provides a computationally efficient way to obtain a parity check matrix in the desired form for hardware implementation.

3.6 Combine the JN and the OSD Algorithm

In [15], a new algorithm which combines JN and OSD was proposed. Due to this combination, we'll use "JNOSD" to represent this algorithm in this thesis. At iteration l, the decoding procedure is as follows:

JNOSD Algorithm

Step1. Order-i reprocessing: Using the LLR vector $\underline{L}^{(l)}$, new MRIPs is generated. OSD(1) is performed to generate a candidate codeword. Sufficient conditions are applied to check if the optimum codeword has been found. If yes, the algorithm stop. If not, the algorithm proceeds to the next step.

Step2. JN Update: The adaptive parity check matrix $\boldsymbol{H}^{(l)}$ is constructed from $\underline{L}^{(l-1)}$. Using $\boldsymbol{H}^{(l)}$ and $\underline{L}_{ext}^{(l-1)}$, extrinsic information $\underline{L}_{ext}^{(l)}$ is generated and the bit reliabilities are updated to $\underline{L}^{(l)}$. This $\underline{L}^{(l)}$ is used in the next iteration of OSD to determine MRIPs.

This algorithm use the concept that after the JN algorithm, the updated LLRs seem to be more reliable than the original channel output. Passing this updated LLR sequence into OSD decoder, we will get a better decoding result. Fig. 3.2 shows that there is about a 0.5dB coding gain between JNOSD and JN or OSD algorithm.

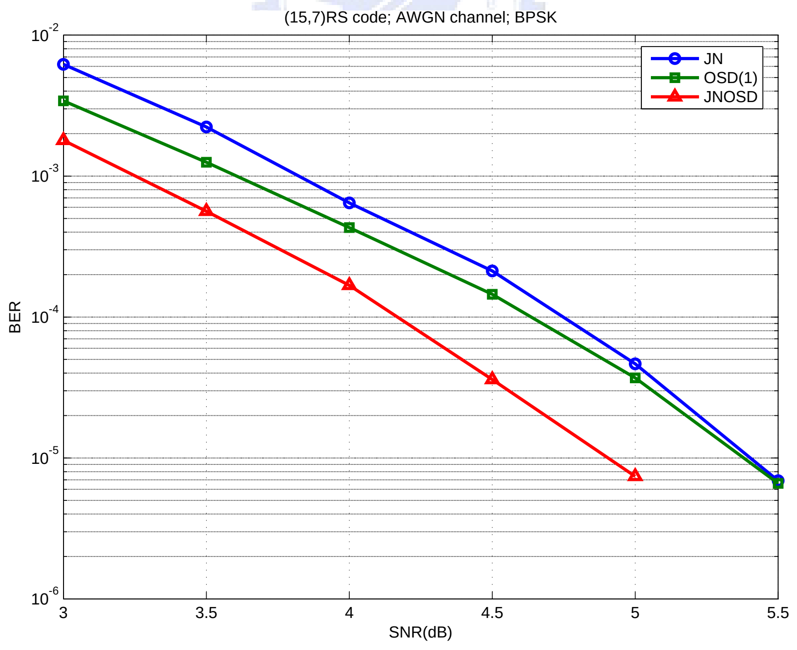

Figure 3.2: Performance comparison of three different algorithms of RS(15,7)

3.7 The Reason of Adapting the Parity Check Matrices

It is well known that belief propagation does not suit to high density parity check (HDPC) codes. If there is an error occurs in the high density part, it will affect many checks and the effects will propagate to other nodes.

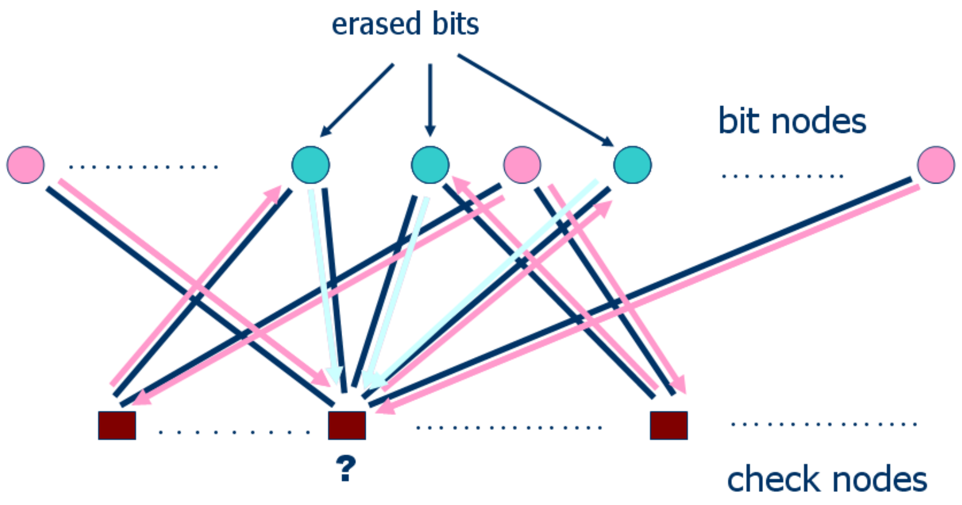

Figure 3.3: Erasure bits will saturate the check

In the AWGN channels, the larger the absolute value of LLR of one bit in the received sequence is, the more reliable this bit is. In the conventional belief propagation such as sum product algorithm, if two or more incoming messages of a check are erasures, the check is erased as shown in Fig. 3.3. In the JN algorithm, it adapt the parity check matrix such that the unreliable bits (bits that with small absolute value of LLR) will connect to only one check, and each check will connect to only one unreliable bit as shown in Fig. 3.4. It prevents the saturation of the check nodes and suppresses the error propagation caused by errors occur in LRPs. We give an example here to explain it.

If the Tanner graph is shown as in Fig. 3.5. Cycle-4 in the graph represents the high density part in the parity check matrix. In the discussion below, assume the correct LLR value is a positive value, and the wrong LLR value is a negative value. If no error occurs,

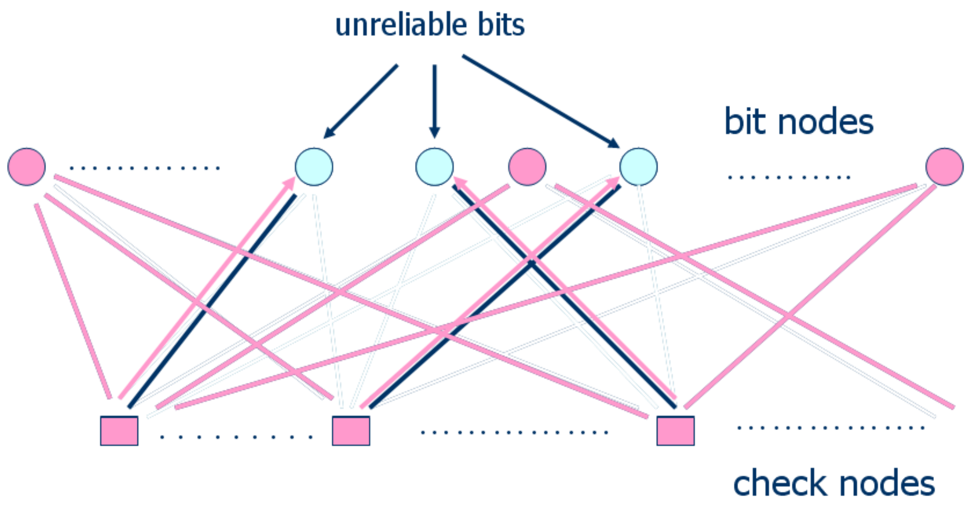

Figure 3.4: The modified tanner graph by JN algorithm

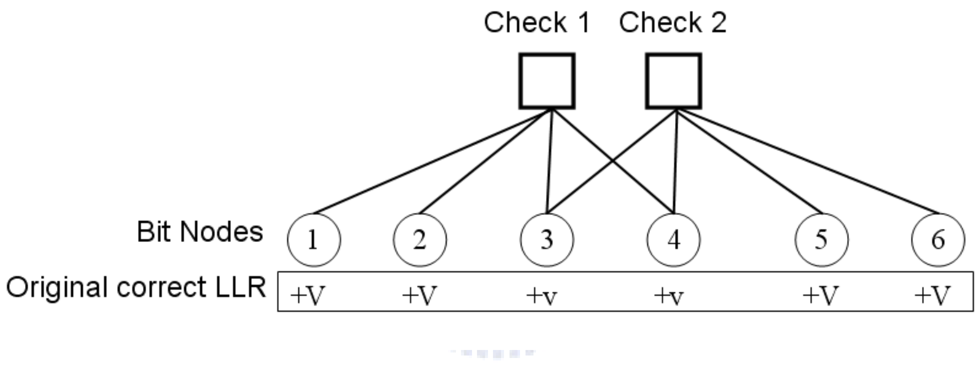

Figure 3.5: Original Tanner graph. Bit node 3 and 4 are LRPs

we assume the LLR value in MRPs are $+V$, the LLR values in LRPs are $+v$, and $V > v$. If error occurs in MRPs, we assume the LLR value is $-V$ and $-v$ if it occurs in LRPs. In Fig. 3.5, assume bit node 3 and bit node 4 are LRPs, other positions are MRPs. Now if the error occurs in bit node 3 and the LLR is $-v$, as shown in Fig. 3.6.

If the extrinsic information get from check node is correct, we use $+E$ to represent it and $E > 0$. On the other hand, if it is wrong, we use $-E$ to represent the extrinsic information. By this assumption, we can get the extrinsic information from the check nodes and the updated LLRs as shown in Fig. 3.7. All the bit nodes except 3 receive wrong information.

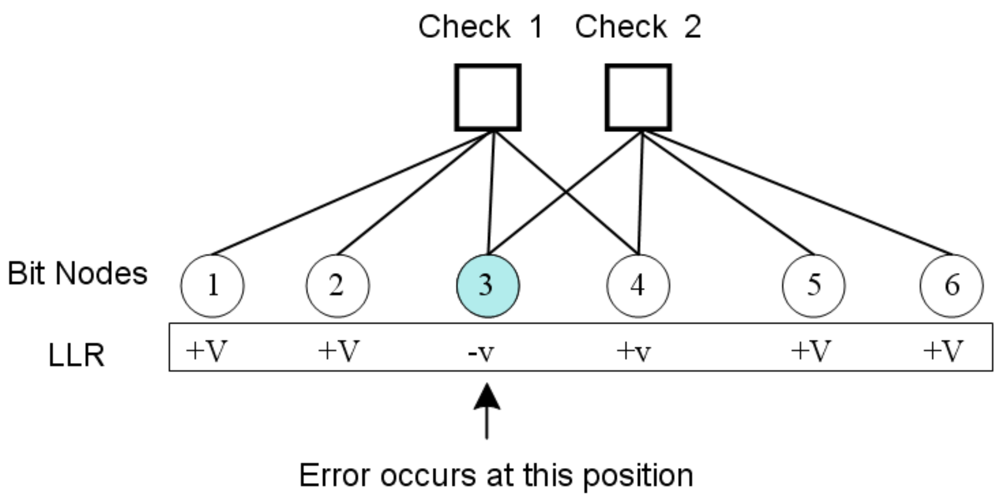

Figure 3.6: One error occurs in MRP(bit node 3)

The propagation of these kinds of errors may cause errors in other bits. Belief propagation will converge to the wrong codeword.

Consider the same case in Fig. 3.6 but now use JN algorithm to decode. As mentioned in last chapter, we can get another equivalent Tanner graph shown in Fig. 3.8 by Gaussian Elimination. In this figure, unreliable bits(bit node 3 and 4) connect to only one check and each check connect to only one unreliable bit. Therefore, the bits in high density part are more reliable. The updated LLR are shown in Fig. 3.9. Compare with Fig. 3.7, error propagation is reduced. Only one bit in MRPs got the wrong extrinsic information.

Now consider that two low reliability positions(bit node 3 and 4) are both in error as in Fig. 3.10. The result after message passing is shown in Fig. 3.11. Due to cycle-4(high density part) between two wrong bits, these two bits will always get wrong extrinsic information. Decoder will converge to a wrong codeword. If we use JN algorithm to decode, the modified Tanner graph is shown in Fig. 3.12. The result of corresponding message passing is shown in Fig. 3.13. Although error propagation still happen in this case, if V is large enough, bit nodes will still have the correct signs. And low reliability bits will have chances to be corrected during this iteration. It is better than original belief propagation in

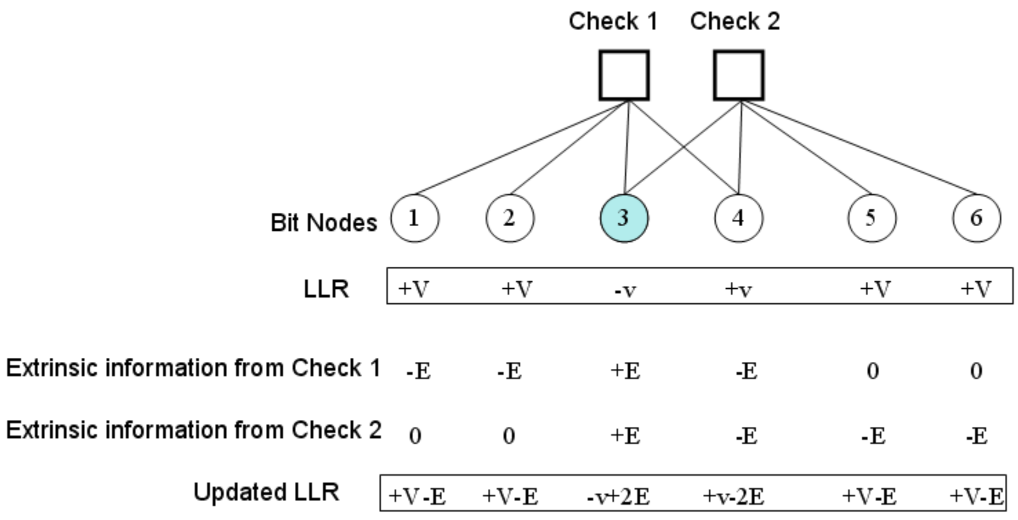

Figure 3.7: Message passing when error occurs in bit node 3

Fig. 3.11. By adapting the parity matrix according to the reliability during each iteration, JN algorithm can reduce the influence caused by errors in LRPs.

3.8 The Latent Problem in the JN algorithm

As our discussions above, JN algorithm decreases the error propagation caused by errors in LRPs by modify the parity check matrix such that LRPs are in the sparse part of the parity check matrix. But if an error occurs in MRP, due to the modified parity check matrix, this bits may connected to many other checks and the wrong message will widely propagate to the graph. The decoder will converge to a wrong codeword. Fig. 3.14 is the distribution of error patterns in MRPs when JN algorithm is fail. The number in x-axis represents the number of errors in MRPs after the first time Gaussian elimination once the decoded result is not correct. In this simulation, max iteration number $l_{max} = 20$, $\alpha = 0.1$. We can observe that if there is no error in MRPs after the first time Gaussian elimination, the decoded output will be correct. But if there are errors in MRPs, JN decoder may give the wrong answer.

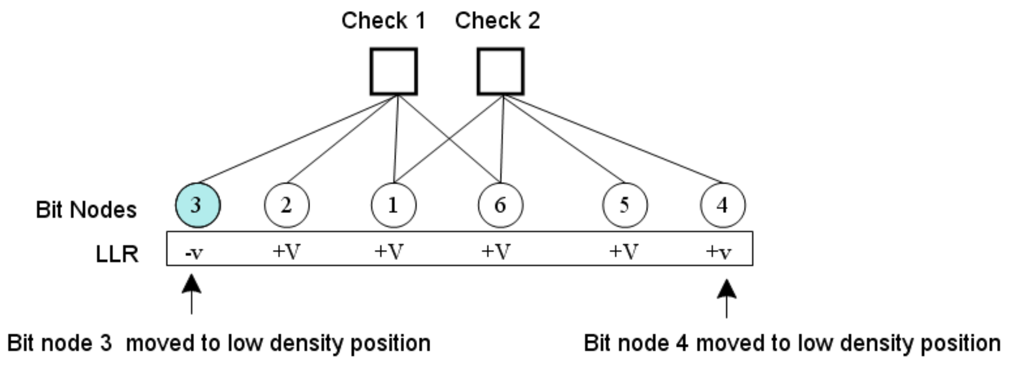
Check 1
Check 2
Bit Nodes
3
2
1
6
5
4
LLR
-v
+V
+V
+V
+V
+v
Bit node 3 moved to low density position
Bit node 4 moved to low density position

Figure 3.8: Modified Tanner graph by JN algorithm

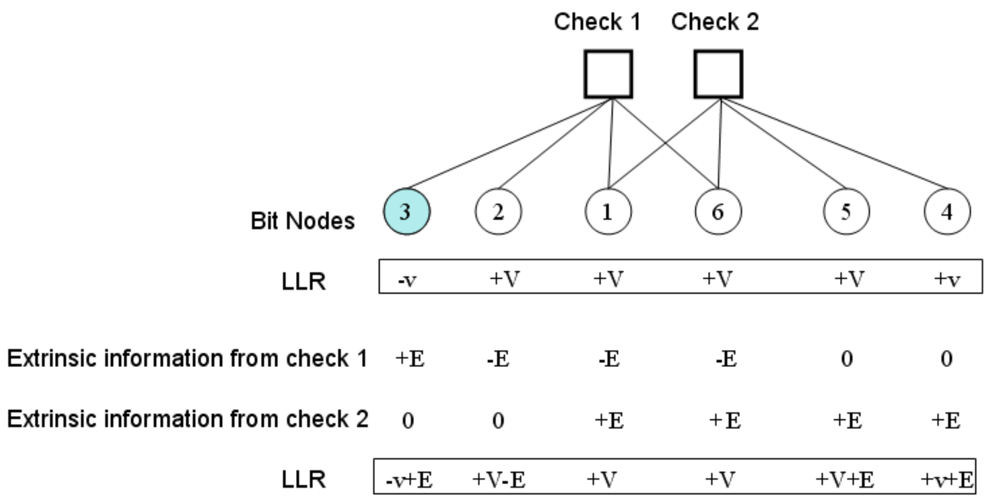

Figure 3.9: Message passing in JN algorithm

3.9 The Modified JNOSD Algorithm

In [16], the author proposed a new algorithm based on JNOSD to decrease the influence of errors in MRPs in the JN algorithm which is mentioned in the last section. To control the error propagation in the JN algorithm, [16] uses OSD algorithm to help the decoding of JN algorithm during each iteration. At iteration l, the decoding procedure is as follows:

Modified JNOSD Algorithm

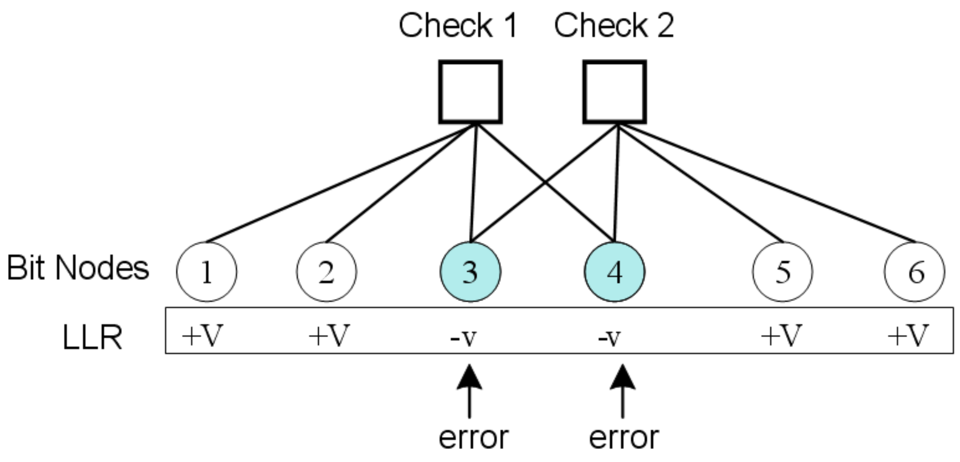

Figure 3.10: Error in bit node 3 and 4

Step1. Order-i reprocessing: Using the LLR vector $\underline{L}^{(l)}$, a new MRB is generated. OSD(1) is performed to generate a candidate codeword. Sufficient conditions are applied to check if the optimum codeword has been found. If yes, the algorithm stop. If not, the algorithm saves the best codeword comes from OSD(1) and proceeds to the next step.

Step2. Feedback: If the best codeword in the last step is 0 in some LRPs, add $+A(A > 0)$ to the corresponding positions of the LLR vector. If the best codeword in the last step is 1 in some LRPs, add $-A$ to the corresponding positions of the LLR vector. A is a parameter that could be decided by the user.

Step3. JN Update: The adaptive parity check matrix $\boldsymbol{H}^{(l)}$ is constructed from $\underline{L}^{(l-1)}$. Using $\boldsymbol{H}^{(l)}$ and $\underline{L}_{ext}^{(l-1)}$, extrinsic information $\underline{L}_{ext}^{(l)}$ is generated and the bit reliabilities are updated to $\underline{L}^{(l)}$. This $\underline{L}^{(l)}$ is used in the next iteration of OSD to determine MRB.

The difference between the JNOSD and the modified JNOSD algorithm is the Step2. Although OSD(1) can correct one error happened in MRPs, it can not help the JN algorithm if the number of errors in MPRs are more than one. By the feedback of the information

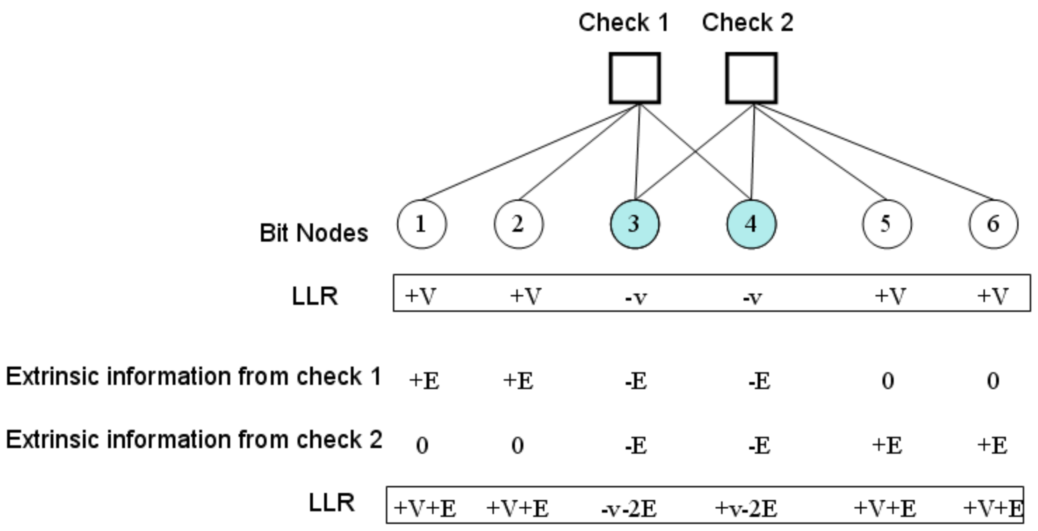

Figure 3.11: Message passing when errors are in bit node 3 and 4

from OSD, it could enhance the accuracy of bits in LRPs. Then by sorting according to the updated LLR by Step 2, some positions in MRPs and LRPs may exchange. On the boundary of MRPs and LRPs, there may be some bits in MRPs but still in error and vice versa. Exchanging the bits on the boundary may exchange the wrong bit in MRPs and the correct bit in LRPs. It can decrease the error number in MRPs and reduce the effect of error propagation.

From Fig. 3.15, the modified JNOSD algorithm really decrease the number of decoding failure when errors occurs in MRPs. The simulation result of (15,7) RS code is shown in Fig. 3.16. The coefficient setting this simulation is: $\alpha = 0.1$, $l_{max} = 20$ and $A = 3.0$. From the simulation, the modified JNOSD algorithm is 0.45dB better than JNOSD at ber=10^{-5}.

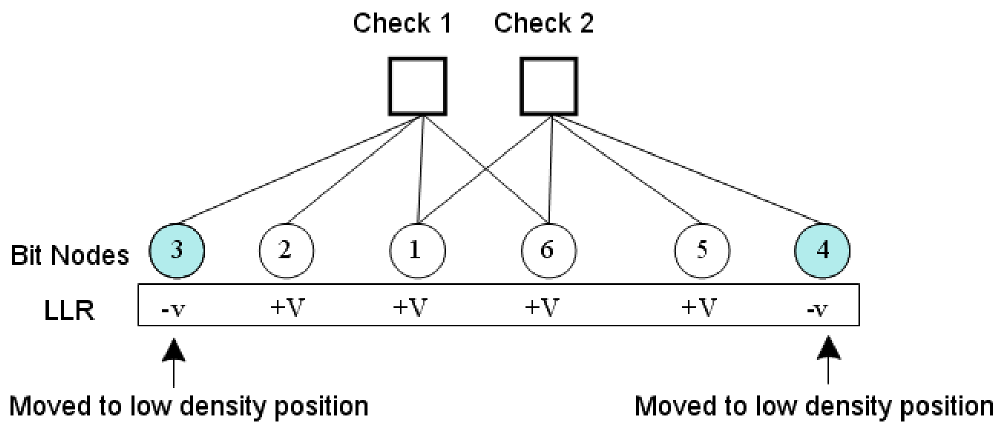

Figure 3.12: Modified Tanner graph by JN algorithm

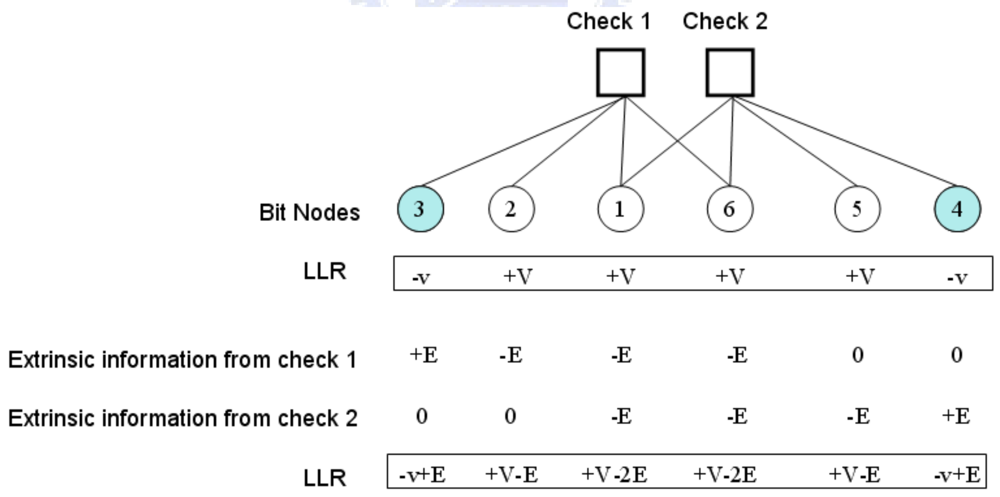

Figure 3.13: Message passing in JN algorithm

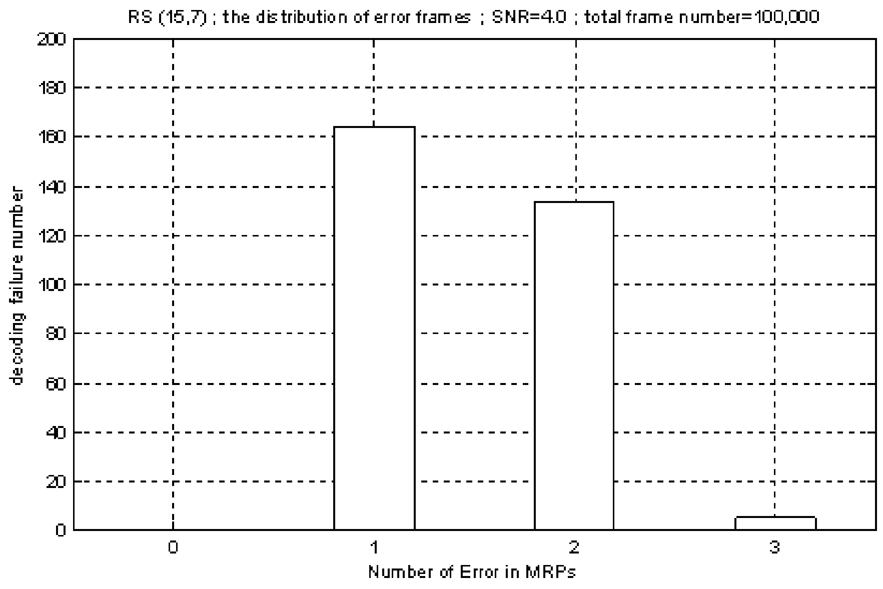

Figure 3.14: Distribution of error patterns in MRPs when JN algorithm is failed

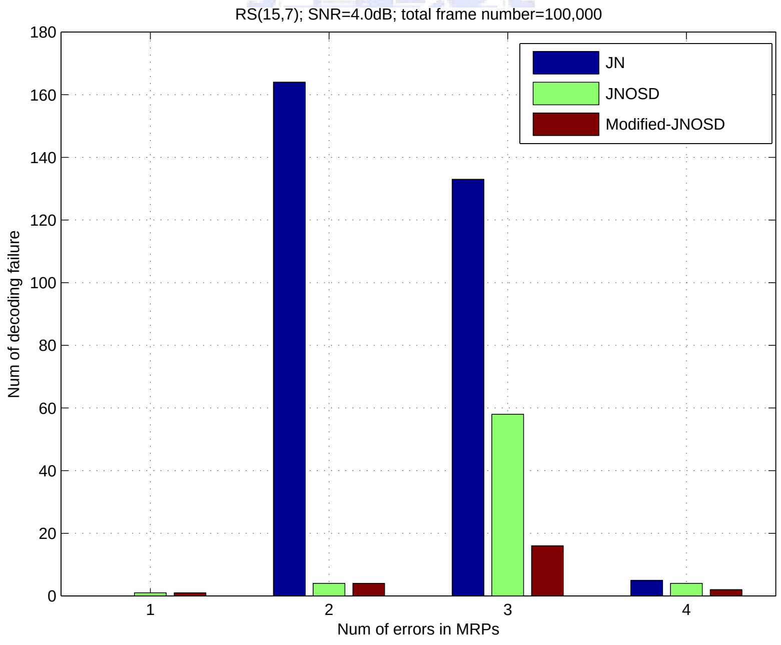

Figure 3.15: Distribution of error patterns in MRPs between three algorithms when they are failed

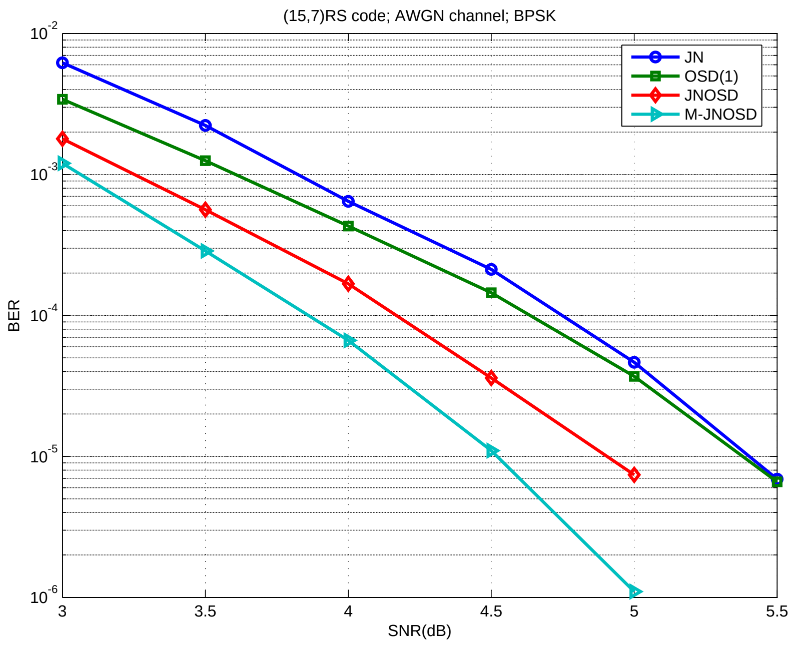

Figure 3.16: Performance comparison of four different algorithms of RS(15,7)

Chapter 4

A New Decoding Algorithm Based on the JN Algorithm

We've known the drawback of the JN algorithm from the last chapter. Now in this chapter, we propose a new algorithm to improve the performance of the JN algorithm.

4.1 The Proposed Algorithm

Once an error occurs in the MRPs, the error propagation may cause the failure of the JN decoder as mention in the last chapter. In [16], it uses the decoded result of OSD(1) to deal with this problem. By exchanging the bits on the boundary of MRPs and LRPs, this new decoder can give a better decoded result. This algorithm takes the information from OSD(1) to help the decoding in the JN algorithm. Considering the same problem(to ease the effect of error propagation in MRPs), we take another viewpoint.

Recall that in the OSD algorithm, the received sequence are partitioned into MRIPs and LRPs. The bits in MRIPs are thought to be more reliable than LRPs. Each time OSD(1) flip one position in MRIPs and re-encode to generate a new candidate codeword. Finally picks up the one with the maximum correlation. In the JN algorithm, it modify the parity check matrix according to the reliability due to the bits with higher reliability are thought to be more reliable. The corresponding Tanner graph of the modified parity check matrix has higher probability to correct errors in LRPs compared with the original belief propagation. But if there are errors in MRPs, the JN decoder may fail due to error propagation. From

Fig. 3.15, if no error happened in MRPs, the JN decoder will almost decode successfully. Our goal is to try to reduce the errors in MRPs before the JN algorithm. We uses the same concept as OSD(1). That is, flip one bit in MRPs at a time and use this new LLR sequence as the input of the JN decoder. There are k bits in MRPs, so we'll get k decoded outputs as the candidate codewords. With the original LLR sequence from channel, total number of candidates are $k+1$. Here are the decoding procedures:

The Proposed Algorithm

Step1. Initialization: set α, $l_{max} = N_1$, $l = 0$ and the LLR's for the coded bits from the channel observation: $\underline{L}^{(0)} = \frac{2}{\sigma^2}\underline{y}$.

Step2. Reliability based parity check matrix adaption: $\boldsymbol{H}_b^{(l)} = \phi(\boldsymbol{H}_b, |\underline{L}^{(l)}|)$.

- **a)** Order the coded bits according to the absolute value of the LLR's $|\underline{L}^{(l)}|$ and record the ordering indices.
- **b)** Implement Gaussian elimination to systematize the $(n-k)$ unreliable positions which are independent in the parity check matrix.

Step3. Use the sorted LLR vector as the input to the JN decoder which is the same as the original JN algorithm. For $1 \le i \le k$, flip the LLR value of i-th position in MRPs. Feed these k modified vectors as the inputs to the JN decoders. Finally, we can get $k+1$ candidates from the JN decoders.

Step4. Compute the correlation of each candidates. The one with the largest correlation will be the decoded output.

The idea of the proposed algorithm is shown in Fig. 4.1. Note that the number in each position of the MRPs is the LLR value.

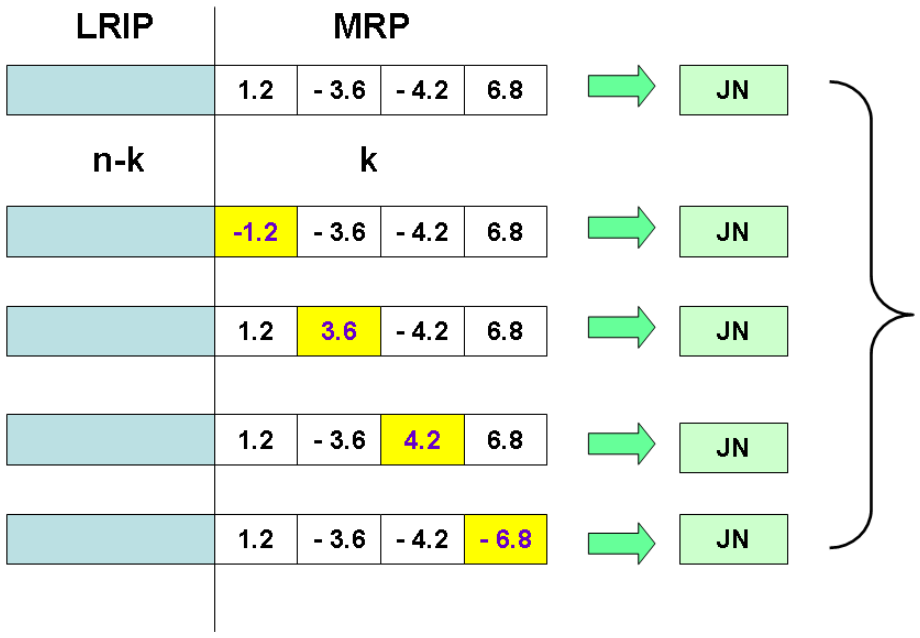

Figure 4.1: The proposed algorithm

4.2 Discussions of the Proposed Algorithm

To deal with the error propagation in the JN algorithm, we first modify the LLR vector before the JN decoder. The modification can reduce the number of errors in the MPRs. Therefore, the decoded output seems to be more correct. Although we need $k+1$ JN decoders, each decoder is independent of others. That is, $k+1$ JN decoders can run in parallel. Our modification is very like to OSD(1), but the proposed algorithm can have a better performance than OSD(1). In OSD(1), if the errors in MRIPs are more than one, the decoder will fail. However, in the proposed algorithm, due to the message passing in the Tanner graph, errors in MRPs may be corrected by the information from other bit nodes.

4.3 Simulation Results

In this section, simulation results of the proposed algorithm and the comparison between different algorithms are presented. We use “JN” to represent the JN algorithm in [11], “OSD(1)” to represent the OSD algorithm with order-1 in [4], “JNOSD” to represent the algorithm introduced in [15] and “M-JNOSD”(Modified-JNOSD) to represent the algorithm in [16]. We first show the simulation results of (15,7) RS code in Fig. 4.2. In this simulation,

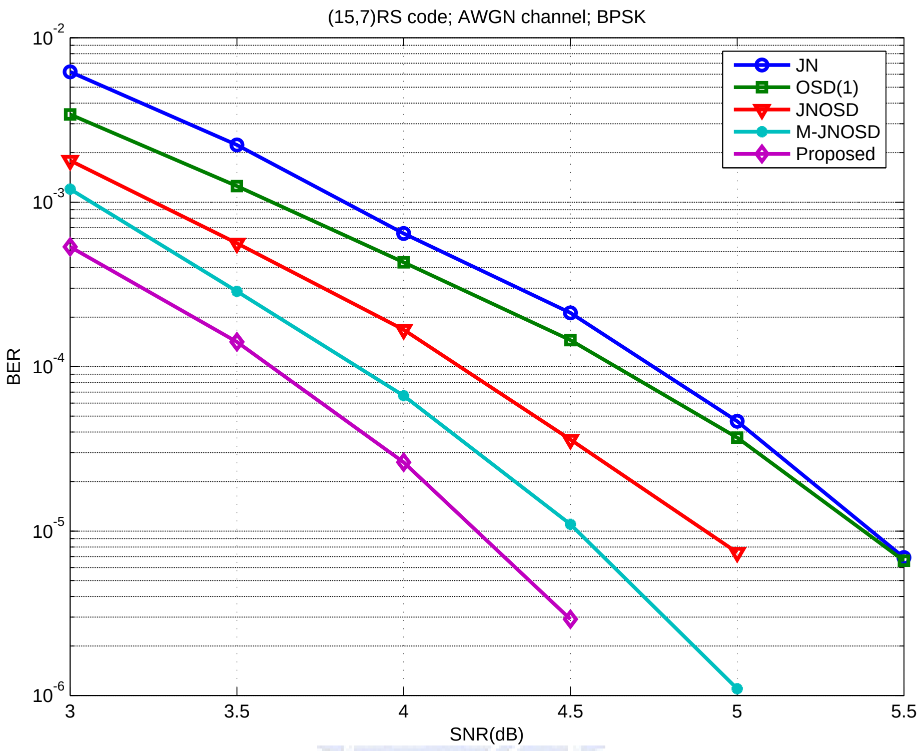

Figure 4.2: Performance comparison of RS (15,7)

$\alpha = 0.1$, $l_{max} = 20$ for the JN algorithm and $A = 3.0$ for the modified JNOSD algorithm. At bit error rate $= 10^{-5}$, JNOSD provides about a 0.5dB gain over JN and OSD algorithm. The modified JNOSD algorithm has a 0.45dB gain compare with JNOSD. The proposed algorithm, is 0.25dB better than the modified JNOSD algorithm and the performance is very close to the ML decoding.

Next we consider (15,9) RS code as shown in Fig. 4.3 The coefficients of simulation are set as follows: $\alpha = 0.1$, $l_{max} = 20$ and $A = 3.0$. At bit error rate $= 10^{-5}$, OSD(1) is 0.4dB better than the JN algorithm. JNOSD is 0.25dB better than OSD(1).The modified JNOSD is about 0.1dB better than the original JNOSD. The performance of the proposed algorithm is almost the same sa the modified JNOSD algorithm in this case. The reason is the performance of these two algorithms in this code are very close to the ML decoding.

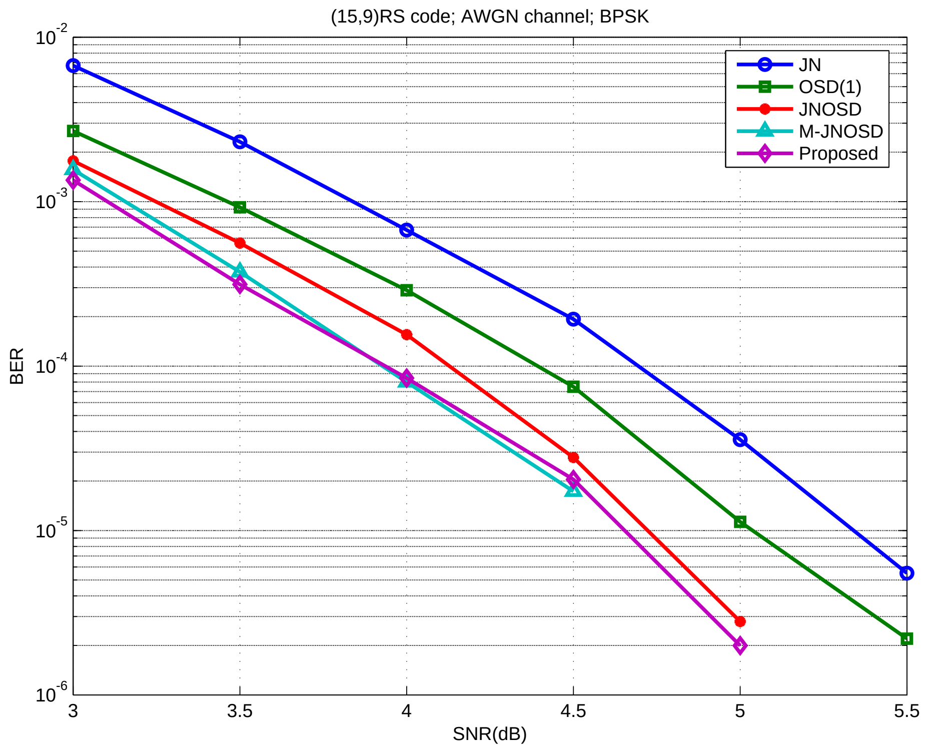

Figure 4.3: Performance comparison of RS (15,9)

The simulation results for the (31,23) RS code are shown in Fig. 4.4. At bit error rate $= 10^{-5}$, JNOSD provides a 0.25dB gain over JN and OSD algorithm. The modified JNOSD algorithm is 0.4dB better than JNOSD algorithm. The proposed algorithm performs 0.3dB better than the modified JNOSD algorithm.

Fig. 4.5 shows the simulation results for the (31,25) RS code. At bit error rate $= 10^{-5}$, JNOSD is about 0.2dB better than OSD(1) and 0.3dB better than the JN algorithm. The modified JNOSD is 0.4dB better than the original JNOSD algorithm. In this case, our algorithm has only a 0.06dB gain compared with the modified JNOSD algorithm.

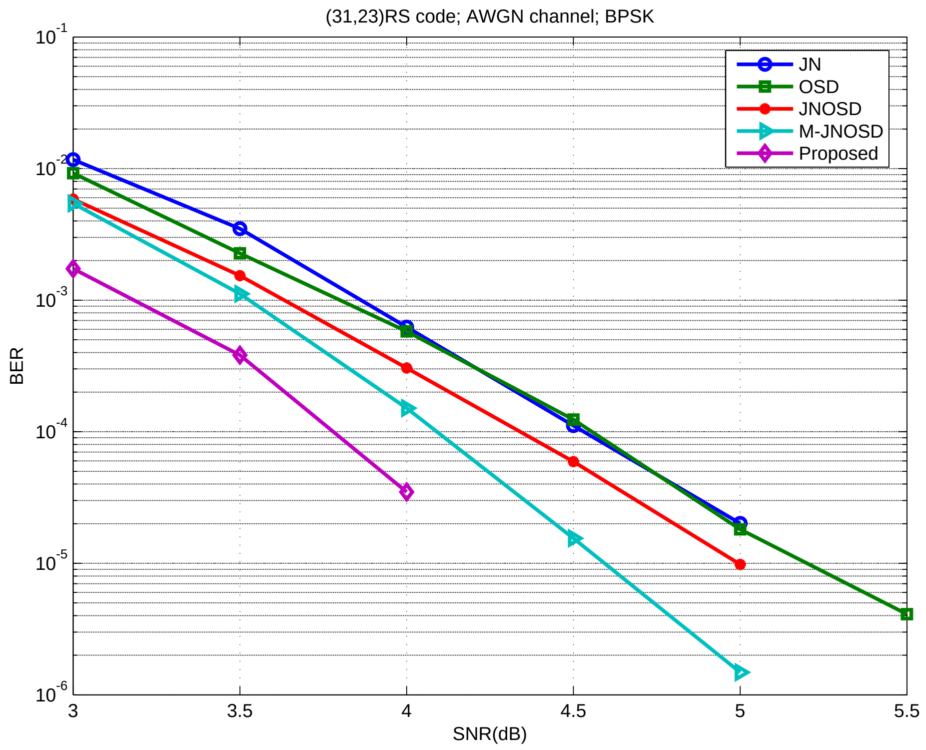

Figure 4.4: Performance comparison of RS (31,23)

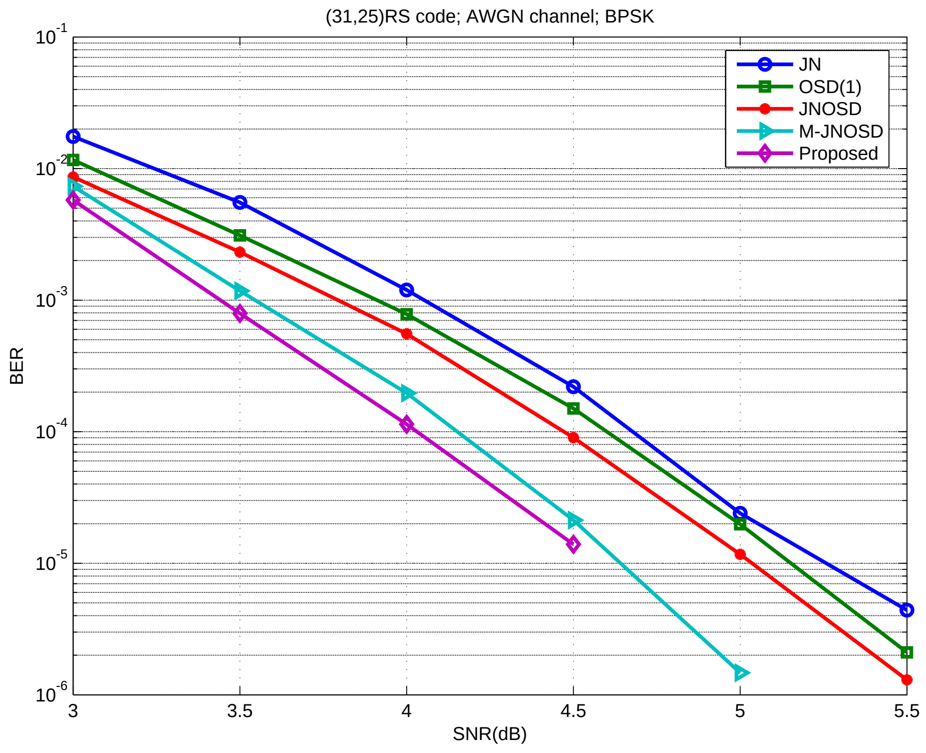

Figure 4.5: Performance comparison of RS (31,25)

Chapter 5

Conclusions

In this thesis, we first introduce how to map RS codes into it's binary representations. By doing so, we can transform the decoding problem of RS codes into the decoding of binary linear block codes. Many decoding algorithms designed for binary linear block codes can therefore be used. Then we briefly review the previous works of decoding RS codes. They are the OSD and the JN algorithm. After reviewing these two algorithms, we analyse the advantages and the drawbacks of the JN algorithm. We discover that the main reason of decoding failure in the JN algorithm is the errors in MRPs. In order to deal with this problem, we propose a new algorithm, which uses the concept in the OSD algorithm before doing the JN algorithm.

From the simulation results, the proposed algorithm has very good performance compared with the previous work such as JN and OSD(1). It also outperforms JNOSD and the modified JNOSD algorithm. Although many decoders are needed in the proposed algorithm, they can run in parallel. The same concept of our algorithm can also be applied to the modified JNOSD algorithm. It means that we modify the received vector before doing the modified JNOSD algorithm. The simulation results show that after doing so, the performance is better than the original modified JNOSD algorithm and is also better than the original proposed algorithm. To sum up, in this thesis, we propose a new soft input soft output decoding algorithm based on JN and OSD algorithm. The idea of the proposed algorithm can also applied to some other algorithms to get a better performance.

Bibliography

[1] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. Soc. Ind. Appl. Math*, vol. 8, pp. 300-304, 1960.

[2] G. D. Forney, "Generalized minimum distance decoding," *IEEE Trans. Inform. Theory*, vol. IT-12, pp. 125-131, Apr. 1966.

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

[4] M. P. C. Fossorier and S. Lin, "Soft-decision decoding of linear block codes based on ordered statics," *IEEE Trans. Inform. Theory*, vol. 41, pp. 1379-1396, Sep. 1995.

[5] V. Guruswami and A. Vardy, "Maximum likelihood decoding of Reed Solomon codes is NP-hard,"*IEEE Trans. Inform. Theory*, vol. 51, pp.2249-2256, Jul. 2005.

[6] A. Vardy and Y. Be'ery, "Bit level soft-decision decoding of Reed-Solomon codes", *IEEE Trans. Commun.,* vol. 39, no. 3, pp. 440-444, Mar.1991.

[7] V. Guruswami and M. Sudan, "Improved decoding of Reed-Solomon codes and algebraic geometry codes," *IEEE Trans. Inform. Theory*, vol. 45, pp. 1757-1767, Sep 1999.

[8] R. Köetter and A. Vardy, "Algebraic soft-decision decoding of Reed-Solomon codes," *IEEE Trans. Inform. Theory*, vol. 49, no. 11, pp. 2809-2825, Nov. 2003.

[9] F. Parvaresh and A. Vardy, "Multiplicity assignments for the algebraic soft-decoding of Reed-Solomon codes," in *Proc. Int. Symp. Inform. Theory*, Jul. 2003, p.205.

[10] M. El-Khamy and R. J. Mceliece, “Interpolation multiplicity assignment algorithms for algebraic soft-decision decoding of Reed-Solomon codes,” *AMS-DIMACS*, vol. 68, pp. 99-120, 2005. Vol. on Algebraic Coding Theory and Information Theory.

[11] J. Jiang and K. Narayanan, “Iterative soft decision decoding of Reed Solomon codes based on adaptive parity-check matrices,” in *Proc. Int. Symp. Inform. Theory*, Jul. 2004, p. 261.

[12] J. Hagenauer, E. Offer, and L. Papke, “Iterative decoding of binary block and convolutional codes,” *IEEE Trans. Inform. Theory*, vol. 42, pp. 429-445, Mar. 1996.

[13] M. El-Khamy and R. J. Mceliece, “Iterative algebraic soft decision decoding of Reed-Solomon codes,” in *Proc. ISITA*, Parma, Italy, Mar. 2004, pp. 1456-1461.

[14] S. B. Wicker, *Error Control Systems for Digital Communication and Storage*, 1st ed. Prentice Hall, Jan 1995.

[15] A. Kothiyal, O. Y. Takeshita, W. Jin and M. P. C. Fossorier, “Iterative reliability-based decoding of linear block codes with adaptive belief propagation,” *IEEE Lett. Commun.*, vol. 9, no. 12, pp. 1067-1069, Dec. 2005.

[16] Y. M. Hsieh, “A Study on Belief-Propagation Based Decoding Algorithms for Reed-Solomon Codes”, Master Thesis, National Chiao Tung University, Hsinchu, Taiwan, R.O.C., 2007.