CHALMERS

Evaluation of LDPC Decoder with Standardized Codes

Master's Thesis in Communication Engineering

ZHAO JUN

Department of Signals and Systems
Division of Communication Engineering
Chalmers University of Technology
Gothenburg, Sweden 2012
Master's Thesis 2012

Abstract

Low density parity check (LDPC) codes are one of the most popular channel codes. Based on traditional sum-product and max-product algorithms, various modified algorithms are tried to improve the performance of LDPC codes in terms of error rate, complexity and latency. Uniformly rweighted belief propagation (URW-BP) algorithm can offer better performance than traditional algorithm especially for regular LDPC codes. linear programming (LP) decoder is also competitive to traditional message passing algorithm. We describe two traditional message passing decoders, four reweighted message passing decoders based on URW-BP and LP decoder in this paper. Reweighted decoders can outperform traditional decoders especially when iteration number is limited. Considering complexity and latency, reweighted max product decoder of version 2 (R-MPD-II) is the most promising decoder.

Acknowledgements

First and foremost, I would like to show my deepest gratitude to my supervisor, Henk Wymeersch, who has provided me with valuable support and guidance throughout the whole thesis project. I shall extend my thanks to Federico Penna and Vladimir Savic for their previous work on this project.

The simulations were performed on resources provided by the Swedish National Infrastructure for Computing (SNIC) at C3SE. Tomas Svedberg at C3SE is acknowledged for assistance concerning technical aspects in making the code run on the C3SE resources.

Zhao Jun, Gothenburg 22/8/2012

Contents

1

Introduction

In communication systems, the goal is to transmit information bits through the medium to the receiver correctly. A channel with noise will introduce error bits to the information symbols. Channel code is a method encoding the information symbol by adding redundancy and decoding the codeword to reduce the error bits by error correction.

1.1 Background

Low density parity check (LDPC) code is one of the most popular channel codes. LDPC codes were first introduced in 1960 by R. Gallager [1]. However, due to the computational effort in implementing the decoder, the full power of LDPC codes were not realized until mid-1990's last century. Based on the developed understanding to graphical representation and iterative decoding, LDPC codes were rediscovered after the breakthrough invention of turbo codes. Because of different researches in different areas, the decoding algorithm has several names which were proved to be the same thing. The two main message passing algorithms conclude sum-product algorithm (or belief propagation algorithm or probability propagation algorithm) and max-product algorithm (or min-sum algorithm). In this thesis, sum-product algorithm and max-product algorithm are used. Sum-product algorithm was invented by Gallager [1] and max-product algorithm was introduced by Tanner [2].

Nowadays, turbo codes and LDPC codes are two major channel coding schemes adopted by mainstream wireless network systems. Turbo codes have good performance for intermediate block length while LDPC codes perform well for long block length. LDPC codes can reach a high performance close to the channel capacity. Fig. 1.1 shows the evolution of channel codes. Compared to the turbo codes, LDPC codes could have a lower bit error rate (BER) than turbo codes especially for long block length codes. On the other side, turbo codes have a fixed number of iterations which means they have a

fixed decoding latency while LDPC codes have a longer latency. However, LDPC codes have more potential in getting a better performance. More and more standards are using LDPC codes as their channel code in recent years like DVB-S2, WiMax.

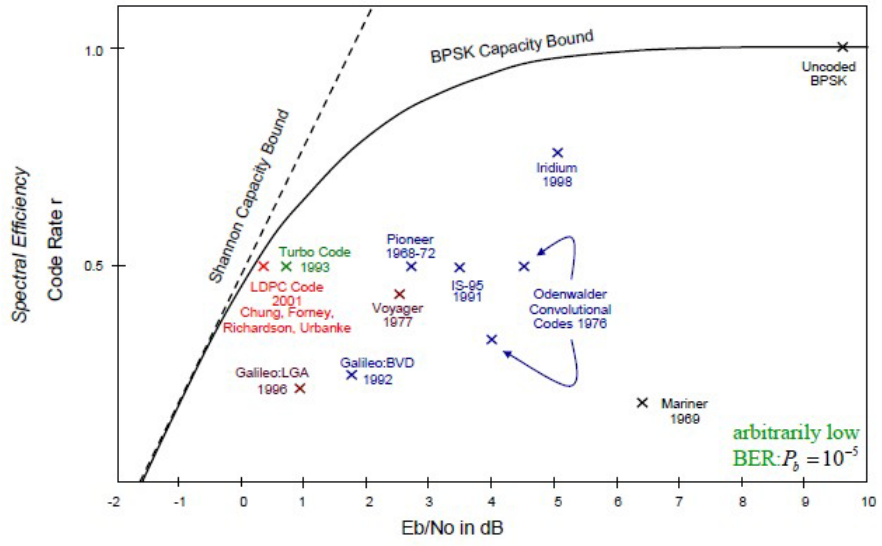

Figure 1.1: Evolution of Coding [3]

However, the disadvantage of LDPC codes is still the complexity both in the encoding and decoding module. The goal of our research is to reduce the complexity for certain BER. Based on the traditional message passing algorithm, tree-reweighted sum-product (TRW-SP) algorithm [4] is an improved algorithm to decrease the BER by using reweighted method. From tree-reweighted algorithm, uniformly reweighted belief propagation (URW-BP) algorithm [5] is first invented for the purpose of reducing the complexity of tree-reweighted algorithm.

1.2 Goal of this thesis

In this thesis, we will evaluate seven decoders:

1. Sum-product decoder (SPD),
2. Max-product decoder (MPD),
3. Reweighted sum-product decoder (R-SPD),
4. Reweighted max-product decoder (R-MPD),
5. Reweighted sum-product decoder version 2 (R-SPD-II),
6. Reweighted max-product decoder version 2 (R-MPD-II),
7. Linear programming decoder (LPD).

Details on six reweighted message passing decoders and the linear programming decoder

will be demonstrated. All the decoders will be implemented in Matlab. Numerical simulations are done with irregular LDPC codes from WiMax standard and regular codes from standard 802.3. The performances are discussed depending on error rate, computing complexity and latency.

1.3 Structure

Chapter 2 introduces the background of LDPC codes. It illustrates the basic knowledge of LDPC codes, representations of factor graph and the idea of iterative decoding. If the reader is familiar with LDPC codes, he/she can skip the background of LDPC codes. At the end of chapter 2, decoding strategy is presented. Chapter 3 will demonstrate the details of all message passing algorithms and also linear programming decoder. Chapter 4 describes the results in different forms. We will compare and discuss the performances of all decoders. Chapter 5 concludes the whole work and states the future work.

2

LDPC Decoding

To begin with the discussion, we will introduce the background of LDPC codes first. LDPC codes are a class of linear block codes. There are two major ways to represent LDPC codes. One is matrix representation like other linear block codes and the other is graphical representation. Both representations are equivalent. Transmission and decoding model will also be introduced in this chapter.

2.1 LDPC codes

2.1.1 Basics of linear block codes

We can generate a (N,K) linear block codes with a generator matrix $\mathbf{G}$ with N and K corresponding to the size of codeword and information word. The generator matrix $\mathbf{G}$ is a K by N binary matrix. Generating a codeword is a mapping function from the set of information words $\mathbf{b} \in \mathbb{B}^K$ to the set of codeword $\mathbf{c} \in \mathcal{C} \subset \mathbb{B}^N$

$$\mathbf{c} = f_{code}(\mathbf{b}) = \mathbf{bG}. \tag{2.1}$$

For all the codewords in this set, we can find the matrix $\mathbf{H}$ called parity check matrix which satisfy the formula,

$$\mathbf{Hc}^T = \mathbf{0}. \tag{2.2}$$

We should notice that the parity check matrix is not unique, however, only the codewords can satisfy the formula. By doing row permutations of the generator matrix $\mathbf{G}$, it can be changed into the form

$$\mathbf{G}_s = [\ \mathbf{I}_K \quad \mathbf{P}\ ], \tag{2.3}$$

where $\mathbf{I}_K$ is identity matrix with the size K by K and $\mathbf{P}$ is a K by $(N-K)$ matrix. The matrix with this kind of form is called systematic generator matrix. With the systematic generator matrix, it is easy to find a systematic parity check matrix

$$\mathbf{H}_s = [\ \mathbf{P}^T \quad \mathbf{I}_{N-K}\ ], \tag{2.4}$$

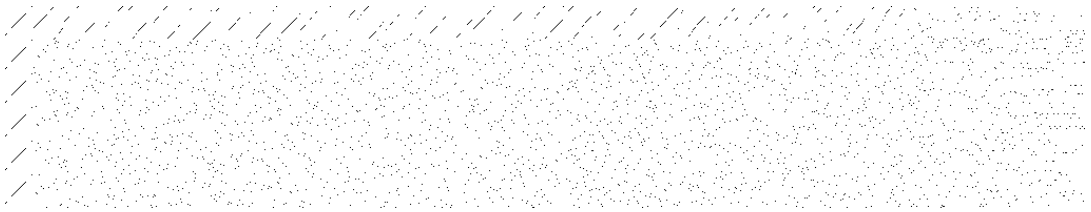

Figure 2.1: Parity Check Matrix **H**

corresponding to the systematic generator matrix.

2.1.2 Low density parity check codes

From the name of low density parity check codes, LDPC parity matrix has few 1's compare to a large number of 0's. We define d_c for the number of 1's in each row and d_v for the number of 1's in each column. d_c and d_v is much smaller than the size of matrix.

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

Eq.(2.5) shows an example of parity check matrix for a (8,4) LDPC code. Fig. 2.1 shows a parity check matrix of the LDPC code we used in the following simulation with the size of (384,2048). The black points in the figure represent 1's while the rest are 0's. From the figure, we can see the character of a sparse matrix. The placement of 1's shows the "randomness" of the matrix. Shannon's theory tells us that random codes with large block lengths by doing optimal decoding can reach the channel capacity. A completely random code can have good performance, however the complexity is exponential in K. LDPC codes using pseudo random and sparse parity check matrix helps to reduce the complexity. Furthermore it was observed that iterative decoding algorithms of sparse codes perform very close to the optimal maximum likelihood decoder [6]. As we can see in the fig 1.1, LDPC codes are the most potential codes near the Shannon limit.

2.1.3 Regular and irregular LDPC codes

Regular LDPC codes have the parity check matrix **H** that d_c and d_v are constant for each row and column. For instance, Eq.(2.5) shows an example of regular LDPC code. There are 4 1's in each row and 2 1's in each column. On the contrary, if the numbers are not a constant, then it is called irregular LDPC code. In general, irregular LDPC codes perform better than regular LDPC codes.

2.1.4 Graphical representation

Tanner explicitly introduced graphs to describe linear block codes. Large size of LDPC codes leads large complexity in both encoding and decoding LDPC codes. This is why LDPC codes was ignored for a long time. The graphical representation such as factor graphs promotes the trend of iterative processing in signal processing [7]. Iterative receiver decreases the complexity making the implementation of LDPC codes practical.

Tanner graph is bipartite graph, a graph with vertices separated into two sets and edges connecting nodes from different sets

$$\mathcal{S} = \mathcal{C} \cup \mathcal{V}. \tag{2.6}$$

There is no edge between nodes in the same set.

The Tanner graph of an LDPC code with parity check matrix $\mathbf{H}$ has two types of nodes. Nodes in $\mathcal{V}$ for each row of $\mathbf{H}$ are called variable nodes and the others in $\mathcal{C}$ for each column of $\mathbf{H}$ are called check nodes. There are edges between check node i and variable node j when $h_{ij} = 1$. Fig 2.2 shows the Tanner graph corresponding to Eq.(2.5).

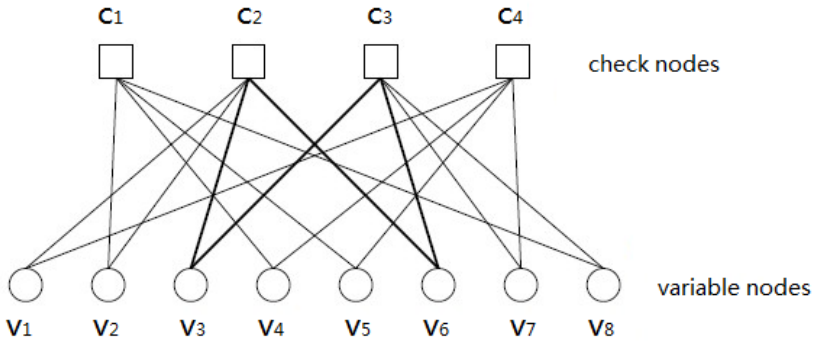

Figure 2.2: Tanner graph

In the Fig 2.2, the bold edges form a cycle of length 4. A cycle of length l is a path of l distinct edges which closes on itself. The shortest possible cycle in the graph has the length of 4. In the case of bipartite graph, cycle length is necessarily an even number. We call the minimum cycle length girth of the Tanner graph. Obviously the girth of Fig 2.2 is 4. Short cycles of Tanner graph have a negative influence on the performance of iterative decoding. So short cycles should be avoided when designing good LDPC codes. To avoid cycles of length 4, overlap number of 1's between any two columns should be at most 1. From the figure, it is also easy to find out the character of regular LDPC codes. Every check nodes have 4 edges connecting to it and every variable nodes have 2. We can also call them the check node degree d_c and variable node degree d_v.

From Eq.(2.1) and Eq.(2.2), the LDPC codewords can be represented as

$$\mathcal{C} = \left\{\mathbf{x} \in \mathbb{B}^N : \mathbf{H}\mathbf{x}^T = \mathbf{0}\right\} \tag{2.7}$$

and

$$\mathcal{C} = \left\{ \mathbf{uG} : \mathbf{u} \in \mathbb{B}^K \right\}, \tag{2.8}$$

where $\mathbf{H}$ is the parity check matrix and $\mathbf{G}$ is the generator matrix. Eq.(2.2) can be written as follows:

$$\begin{aligned} \mathbf{h}_1\mathbf{x}^T &= \mathbf{0} && (2.9)\\ \mathbf{h}_2\mathbf{x}^T &= \mathbf{0} && (2.10)\\ &... && (2.11)\\ \mathbf{h}_{\mathcal{N}-\mathcal{K}}\mathbf{x}^T &= \mathbf{0}. && (2.12) \end{aligned}$$

$\mathbf{h}_i$ is ith row of $\mathbf{H}$, where every row means one check requirement. Then the membership indicator function is

$$\mathcal{I}_{\mathcal{C}} = \begin{cases} 1, \, if \ \mathbf{x} \in \mathcal{C} \\ 0, \, else \end{cases} \tag{2.13}$$

Now we take the example of $\mathbf{H}$ from Eq.(2.5). Eq.(2.13) can be rewritten as

$$\begin{aligned} \mathcal{I}_{\mathcal{C}}(x_1,...,x_N) = \delta(x_2 \oplus x_4 \oplus x_5 \oplus x_8) \\ \cdot\delta(x_1 \oplus x_2 \oplus x_3 \oplus x_6) \\ \cdot\delta(x_3 \oplus x_6 \oplus x_7 \oplus x_8) \\ \cdot\delta(x_1 \oplus x_4 \oplus x_5 \oplus x_7) \end{aligned}, \tag{2.14}$$

where $\oplus$ denotes the binary addition. Each $\delta()$ functions correspond to each check.

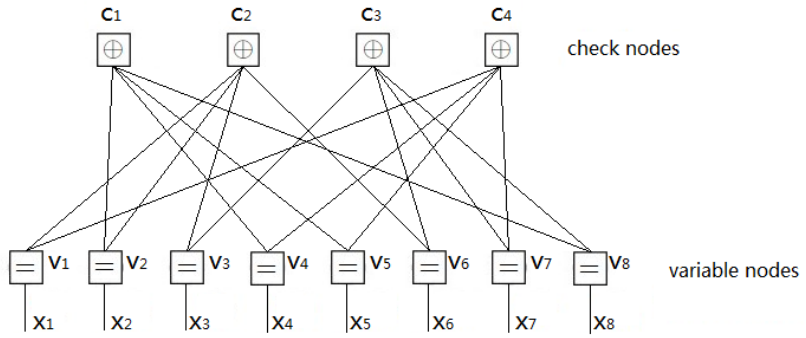

Figure 2.3: Factor graph

The factor graph Fig 2.3 is a graphical representation of the Eq.(2.14) and is often combined with message passing decoding algorithms [8]. As Fig 2.3 shows, check nodes work as binary addition. x_1,x_2,...,x_8 are the bits of the codeword with a codeword length of 8. At the beginning, the original received codeword information will be sent. Messages pass through the edges. x_1,x_2,...,x_8 will get new updated message. If all checks are satisfied, Eq.(2.14) equalling to 1, the codeword $\mathbf{x} = [x_1,x_2,...,x_8]$ will be a legal LDPC codeword. Otherwise, it will continue the iterations.

2.1.5 Hard decision decoding of LDPC codes

To introduce LDPC decoding algorithm, we first introduce hard decision decoding to have a basic understanding of the decoding process. We take Fig 2.3 as the example. Now a codeword $\mathbf{x}$ = [1,1,0,1,0,1,0,1] is received and the decoder will decodes the codeword iteratively until a legal codeword is got. LDPC decoder won't guarantee the result is the true codeword that was sent from the transmitter. But it will make sure it is a legal codeword. The decoding iteration can be divided by a few steps.

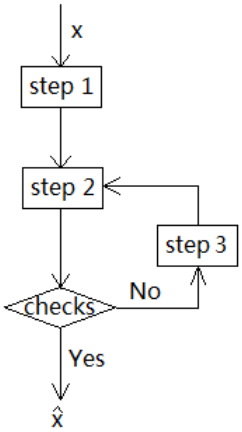

Figure 2.4: Iteration steps

1. Step 1 message updating from $\mathcal{V}$ to $\mathcal{C}$: Firstly, initialize the decoder. The original codeword $\mathbf{x}$ = [x_1,x_2,...,x_8] is assigned with [1,1,0,1,0,1,0,1]. All the variable nodes send the bit message to the check nodes. For example, the variable node v_1 will send the message $x_1 = 1$ along the edges connecting to the check nodes c_2 and c_4. The check node c_1 will receive the messages x_2, x_4, x_5 and x_8 from variable v_2, v_4, v_5 and v_8, respectively.

2. Step 2 message updating from $\mathcal{C}$ to $\mathcal{V}$: Secondly, the check nodes will do the checks of LDPC codes. We already know that the check function is binary addition. If the result is 0, the check node send back the message it received from the variable nodes. If the result is 1 which means the check fails, the check node will send the opposite message. For example, check node c_1 receives $x_2 = 1$, $x_4 = 1$, $x_5 = 0$ and $x_8 = 1$.

$$x_2 \oplus x_4 \oplus x_5 \oplus x_8 = 1 \tag{2.15}$$

The check node will send $x_2 = 0$, $x_4 = 0$, $x_5 = 1$ and $x_8 = 0$ back to variable nodes v_2, v_4, v_5 and v_8. In the theory of factor graph, the actual operation is sending the messages

$$x_2 = x_4 \oplus x_5 \oplus x_8 \tag{2.16}$$
$$x_4 = x_2 \oplus x_5 \oplus x_8 \tag{2.17}$$
$$x_5 = x_2 \oplus x_4 \oplus x_8 \tag{2.18}$$
$$x_8 = x_2 \oplus x_4 \oplus x_5. \tag{2.19}$$

In this hard decision decoding example, we simply say that it send the opposite message if the check fails. If all the checks are fulfilled, it means a legal LDPC codeword $\hat{\mathbf{x}}$ appears and the decoder terminates the iterations.

Step 3 decisions updating: The variable nodes will decide the codeword bits with the message responded by check nodes and the original message they sent to the check nodes. A simple way to do this is a majority vote [6]. For example, variable node v_2 receives messages $x_2 = 0$ from both c_1 and c_2. The original bit sending to check nodes is $x_2 = 1$, however the decoder will set the value of x_2 to be 0 because of the suggestions from the check nodes. After the modification of the bits, the variable nodes will send the messages back to the check nodes. The decoder will do step 2 again.

Step	Item	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
step 1	original messages	1	1	0	1	0	1	0	1
step 2	respond messages	0,1	0,0	1,0	0,1	1,0	0,1	0,0	1,1
step 3	modified messages	1	0	0	1	0	1	0	1
step 2	respond messages	1,1	0,0	0,0	1,1	0,0	1,1	0,0	1,1

Table 2.1: Decoding process

Fig 2.4 shows the flow chart of the decoding algorithm. Table 3.1 represents the stages of the hard decision decoding example. When the decoder comes to the second step 2, all four checks are fulfilled. So the decoder stops with the codeword $\hat{\mathbf{x}} = [1,0,0,1,0,1,0,1]$ which is a legal LDPC codeword in this case.

2.2 Decoding strategy

With a LDPC parity check matrix $\mathbf{H}$, we have a set of codewords $\mathbf{x} \in \mathcal{C} \subset \mathbb{B}^N$ which satisfy the checks $\mathbf{Hx} = \mathbf{0}$. Assuming the symbols are transmitted over the additive white Gaussian noise (AWGN) channel with modulation of binary phase shift keying (BPSK), we have received symbols

$$\mathbf{y} = 2\mathbf{x} - \mathbf{1} + \mathbf{n}. \tag{2.20}$$

where $\mathbf{n}$ is i.i.d Gaussian noise with variance σ^2.

The previous sections describe the method to detect the estimated codeword $\hat{\mathbf{x}}$ from a received word $\mathbf{y}$. Avoiding or minimizing bit error and word error is the main task for the decoder. For detection, maximum a posteriori (MAP) estimation and detection is optimal,

$$\hat{\mathbf{x}}(\mathbf{y}) = \arg\max_{\mathbf{x}} p(\mathbf{x}|\mathbf{y}). \tag{2.21}$$

From Bayes' rule,

$$p(\mathbf{x}|\mathbf{y}) = \frac{p(\mathbf{x},\mathbf{y})}{p(\mathbf{y})} \tag{2.22}$$

$$= \frac{p(\mathbf{y}|\mathbf{x})p(\mathbf{x})}{p(\mathbf{y})}. \tag{2.23}$$

Because random codewords are used in practical, all symbols' probabilities $p(\mathbf{x})$ are equally likely. $p(\mathbf{y})$ is not needed because it will be removed as the same value for the numerator and denominator in the form of log-likelihood described in the next chapter. It also can be calculated from numerical stability. So we can discard $p(\mathbf{x})p(\mathbf{y})$. Eq.(2.21) can be transformed into

$$\hat{\mathbf{x}}_{\mathrm{W}}(\mathbf{y}) = \arg\max_{\mathbf{x}} p(\mathbf{y}|\mathbf{x}) \tag{2.24}$$

$$= \arg\max_{\mathbf{x}\in\mathcal{C}} \prod_{n=1}^{N} p(y_n|x_n). \tag{2.25}$$

Considering minimization of bit error probability. Bitwise MAP decoder estimates the codeword like

$$\hat{\mathbf{x}}_{\mathrm{B}}(\mathbf{y}) = [\hat{x}_{\mathrm{B},1}(\mathbf{y}),\hat{x}_{\mathrm{B},2}(\mathbf{y}),\ldots,\hat{x}_{\mathrm{B},N}(\mathbf{y})]^T, \tag{2.26}$$

where

$$\hat{x}_{\mathrm{B},n}(\mathbf{y}) = \arg\max_{x_n} p(x_n|\mathbf{y}). \tag{2.27}$$

In the next chapter, we will describe a number of practical decoding methods that aim to approximate MAP detection.

3

Decoding algorithms

After introduction of the concept of the factor graph and the basic idea of passing message, seven distinct decoding algorithms are described in detail in this chapter. Six message passing algorithms and linear programming are introduced and compared in terms of the complexity and performance in theory.

3.1 Message passing decoder

In this section, sum-product algorithm, max-product algorithm and also reweighted versions will be introduced and compared.

3.1.1 Sum-product decoder (SPD)

We will first introduce the generic algorithm, the sum-product algorithm. Sum-product algorithm aims to compute the marginals

$$p(x_n|\mathbf{y}) = \sum_{\sim\{x_n\}} p(\mathbf{x}|\mathbf{y}),\ \forall n. \tag{3.1}$$

The message update rule of the sum-product algorithm: The message sent from a node s on an edge e is the product of the local function at s (or the unit function if s is a variable node) with all messages received at s on edges other than e, summarized for the variable associated with e [9].

The decoder will pass messages between variable nodes and check nodes iteratively. Every time the message $\mu_{V_n\to\psi_l}(x_n)$ from variable node V_n to check node ψ_l and the message $\mu_{\psi_l\to V_n}(x_n)$ transmitting in the opposite way are updated. After every iteration, marginals $b_{V_n}(x_n) = \mu_{V_n\to\psi_l}(x_n)\mu_{\psi_l\to V_n}(x_n)$ are computed to decide whether the termination operates [10].

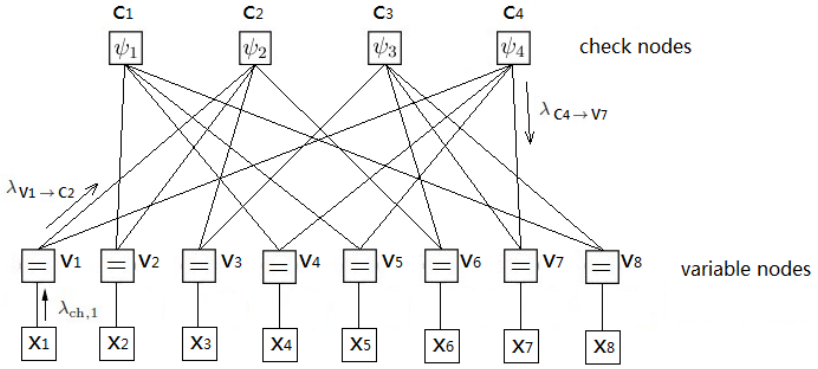

Figure 3.1: Factor graph for a example LDPC codes

$$\mu_{V_n\to\psi_l}(x_n) = p(y_n|x_n) \prod_{k\in\mathcal{N}(V_n)\backslash\{l\}} \mu_{\psi_k\to V_n}(x_n), \tag{3.2}$$

$$\mu_{\psi_l\to V_n}(x_n) = \sum_{\sim\{x_n\}} \psi_l(\mathbf{x}_l) \prod_{m\in\mathcal{N}(\psi_l)\backslash\{n\}} \mu_{V_m\to\psi_l}(x_m), \tag{3.3}$$

and

$$b_{V_n}(x_n) = p(y_n|x_n) \prod_{k\in\mathcal{N}(V_n)} \mu_{\psi_k\to V_n}(x_n), \tag{3.4}$$

where b_{V_n} the approximation of the marginals. Eq.(SPD2) shows where the name is from.

In the message passing algorithms, messages are often computed in the logarithmic domain. In this way, exponential terms disappear and multiplications become addition. It helps in practical computing system. Eq.(3.2) and Eq.(3.4) in the logarithmic domain becomes

$$\log \mu_{V_n\to\psi_l}(x_n) = \log p(y_n|x_n) + \sum_{k\in\mathcal{N}(V_n)\backslash\{l\}} \log \mu_{\psi_k\to V_n}(x_n), \tag{3.5}$$

$$\log b_{V_n}(x_n) = \log p(y_n|x_n) + \sum_{k\in\mathcal{N}(V_n)} \log \mu_{\psi_k\to V_n}(x_n). \tag{3.6}$$

In addition, sums can be approximated by maximization. It is found that sums in Eq.(3.3) can be replaced by max* function [11]:

$$\log \mu_{\psi_l\to V_n}(x_n) = \max^*_{\sim\{x_n\}} \left\{ \log \psi_l(\mathbf{x}_l) + \sum_{m\in\mathcal{N}(\psi_l)\backslash\{n\}} \log \mu_{V_m\to\psi_l}(x_m) \right\}, \tag{3.7}$$

where

$$\max^*[L_1,L_2] = \max[L_1,L_2] + \log(1 + e^{-|L_1-L_2|}). \tag{3.8}$$

Like ordinary *max* function, The max*-operation can be implemented recursively as

$$\max^*[L_1,L_2,\ldots,L_M] = \max^*[\max^*[L_1,L_2,\ldots,L_{M-1}],L_M]. \tag{3.9}$$

Note that the messages sent on an edge contains the probabilities of 1 and 0, these two probabilities can be conveniently expressed into log-likelihood ratio

$$\lambda_{\mathrm{ch},n} = \log \frac{p(y_n|x_n=1)}{p(y_n|x_n=0)}. \tag{3.10}$$

As we will use BPSK modulation over AWGN channel, the result is Gaussian distribution and two probabilities are

$$p(y_n|x_n=1) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y_n-1)^2}{2\sigma^2}}, \tag{3.11}$$

$$p(y_n|x_n=0) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y_n+1)^2}{2\sigma^2}}. \tag{3.12}$$

So the log-likelihood ratio can be simplified as

$$\begin{aligned} \lambda_{\mathrm{ch},n} &= \log\left(e^{-\frac{(y_n-1)^2-(y_n+1)^2}{2\sigma^2}}\right) && (3.13)\\ &= \frac{2y_n}{\sigma^2}. && (3.14) \end{aligned}$$

Furthermore, advantages are obvious to transform the Eq.(3.5)-(3.7) into the log-likelihood ratio forms from the knowledge of

$$\lambda_{A\to B} = \log \mu_{A\to B}(1) - \log \mu_{A\to B}(0). \tag{3.15}$$

So the updated equations of Eq.(3.2)-(3.4) become

$$\lambda_{V_n\to\psi_l} = \lambda_{\mathrm{ch},n} + \sum_{k\in\mathcal{N}(V_n)\backslash\{l\}} \lambda_{\psi_k\to V_n} \tag{3.16}$$

$$\lambda_{\psi_l\to V_n} = f_{\max^*}\left(\{\lambda_{V_m\to\psi_l}\}_{m\neq n}\right) \tag{3.17}$$

where since $\log \mu_{V_m\to\psi_l}(x_m)=(-1)^{(1-x_m)}\lambda_{V_m\to\psi_l}/2$,

$$\begin{aligned} &f_{\max^*}\left(\{\lambda_{V_m\to\psi_l}\}_{m\neq n}\right) = \\ &\max^*_{\mathbf{x}_l:x_n=1}\left\{\log\psi_l(\mathbf{x}_l) + \frac{1}{2}\sum_{m\in\mathcal{N}(\psi_l)\backslash\{n\}}(-1)^{1-x_m}\lambda_{V_m\to\psi_l}\right\} \\ &-\max^*_{\mathbf{x}_l:x_n=0}\left\{\log\psi_l(\mathbf{x}_l) + \frac{1}{2}\sum_{m\in\mathcal{N}(\psi_l)\backslash\{n\}}(-1)^{1-x_m}\lambda_{V_m\to\psi_l}\right\} \end{aligned} \tag{3.18}$$

Here, the definition of ψ_l is $\psi_l = \mathbb{I}\left\{\mathbf{h}_l^T\mathbf{x} = 0\right\}$ which refers to the l-th check. If $\mathbf{x}$ fulfils the check, $\log\psi_l = 0$. Otherwise $\log\psi_l \to -\infty$.

$$\lambda_{\mathrm{b},n} = \lambda_{\mathrm{ch},n} + \sum_{k\in\mathcal{N}(V_n)} \lambda_{\psi_k\to V_n} \tag{3.19}$$

The decision of the bit is made from Eq.(SPA3)

$$\hat{x}_n = \begin{cases} 1, \lambda_{\mathrm{b},n} \geq 0 \\ 0, \lambda_{\mathrm{b},n} < 0 \end{cases} \tag{3.20}$$

3.1.2 Max-product decoder (MPD)

Max-product algorithm aims to compute the max-marginals:

$$q(x_n|\mathbf{y}) = \max_{\sim\{x_n\}} p(\mathbf{x}|\mathbf{y}),\ \forall n. \tag{3.21}$$

Assuming that $p(\mathbf{x}|\mathbf{y})$ has a unique maximum, then $\hat{x}_n = \arg\max_{x_n} q(x_n|\mathbf{y})$ is equal to the n-th component of $\hat{\mathbf{x}}_{\mathrm{W}}(\mathbf{y})$, allowing us to approximately solve (2.21).

Updating rules for max-product algorithm is similar to sum-product algorithm. The only difference locates in Eq.(3.3), where the *sum* function is replaced by *max* function

$$\mu_{\psi_l\to V_n}(x_n) = \max_{\sim\{x_n\}} \psi_l(\mathbf{x}_l) \prod_{m\in\mathcal{N}(\psi_l)\setminus\{n\}} \mu_{V_m\to\psi_l}(x_m). \tag{3.22}$$

So in the log-likelihood form, max* function is replaced by *max* function. The updating rules in logarithmic domain are

$$\lambda_{V_n\to\psi_l} = \lambda_{\mathrm{ch},n} + \sum_{k\in\mathcal{N}(V_n)\setminus\{l\}} \lambda_{\psi_k\to V_n}, \tag{3.23}$$

$$\lambda_{\psi_l\to V_n} = f_{\max}\left(\{\lambda_{V_m\to\psi_l}\}_{m\neq n}\right), \tag{3.24}$$

and

$$\lambda_{\mathrm{b},n} = \lambda_{\mathrm{ch},n} + \sum_{k\in\mathcal{N}(V_n)} \lambda_{\psi_k\to V_n}. \tag{3.25}$$

The decision of the bit is the same with Eq.(3.20).

3.1.3 Reweighted sum-product decoder (R-SPD)

Message passing algorithm is a powerful way to compute the marginals in a graph. As we mentioned in previous chapter, good LDPC codes should avoid short cycles because short cycles will lead bad performance. When the factor graph is cycle-free, message passing algorithm will guarantee to converge and offer an optimal result within the scope of its ability. However, when the graph contains cycles, it may converge to a local optimum or even fail to converge [5].

Tree-reweighted sum-product (TRW-SP) algorithm [4] is an improved sum-product algorithm to compute marginals in graphs with cycles. Tree-reweighted sum-product algorithm has been found to out perform ordinary message passing algorithm in general. It introduced the factors called edge appearance probabilities which are the reweighted factor for edges between variable nodes and check nodes to optimize the computation and reduce the influence of cycles.

Tree-reweighted sum-product algorithm is complex to optimize the edge appearance probabilities for every edge. Typical LDPC codes with large size shows a very regular alike structure, hence a solution for the previous complexity problem is assign a constant reweighted factor ρ to all edges instead of calculating the whole probabilities. This algorithm is called uniformly reweighted belief propagation (URW-BP) algorithm [5]. Tree-reweighted sum-product algorithm was developed for graphs with pairwise interactions. R-SPD uses the method to convert factor graphs to a Markov random field with pairwise interactions.

The message updating rules are

$$\lambda_{V_n \to \psi_l} = \lambda_{\mathrm{ch},n} + \rho \sum_{k \in \mathcal{N}(V_n) \setminus \{l\}} \lambda_{\psi_k \to V_n} - (1-\rho)\lambda_{\psi_l \to V_n}, \tag{3.26}$$

$$\lambda_{\psi_l \to V_n} = f_{\max^*}\left(\{\rho \lambda_{V_m \to \psi_l}\}_{m \neq n}\right) - (1-\rho)\lambda_{V_n \to \psi_l}, \tag{3.27}$$

and

$$\lambda_{b,n} = \lambda_{\mathrm{ch},n} + \rho \sum_{k \in \mathcal{N}(V_n)} \lambda_{\psi_k \to V_n}. \tag{3.28}$$

3.1.4 Reweighted max-product decoder (R-MPD)

Tree-reweighted max-product (TRW-MP) message passing algorithm is introduced based on max-product algorithm which is reweighted max-product algorithm [12]. In the same way, $f_{\max^*}\left(\{\rho \lambda_{V_m \to \psi_l}\}_{m \neq n}\right)$ is replaced by $f_{\max}\left(\{\rho \lambda_{V_m \to \psi_l}\}_{m \neq n}\right)$.

The message updating rules are

$$\lambda_{V_n \to \psi_l} = \lambda_{\mathrm{ch},n} + \rho \sum_{k \in \mathcal{N}(V_n) \setminus \{l\}} \lambda_{\psi_k \to V_n} - (1-\rho)\lambda_{\psi_l \to V_n}, \tag{3.29}$$

$$\lambda_{\psi_l \to V_n} = f_{\max}\left(\{\rho \lambda_{V_m \to \psi_l}\}_{m \neq n}\right) - (1-\rho)\lambda_{V_n \to \psi_l}, \tag{3.30}$$

where

$$f_{\max}\left(\{\rho \lambda_{V_m \to \psi_l}\}_{m \neq n}\right) = \rho f_{\max}\left(\{\lambda_{V_m \to \psi_l}\}_{m \neq n}\right), \tag{3.31}$$

and

$$\lambda_{b,n} = \lambda_{\mathrm{ch},n} + \rho \sum_{k \in \mathcal{N}(V_n)} \lambda_{\psi_k \to V_n}. \tag{3.32}$$

3.1.5 Reweighted sum-product decoder II (R-SPD-II)

The new version of uniformly reweighted sum-product algorithm is a little different from the one described in previous section. Now it does not convert the factor graphs to a graph with only pairwise interactions. This kind of decoder is named Reweighted sum-product decoder-version 2 (R-SPD-II) [13].

The modified message passing rules are

$$\lambda_{V_n \to \psi_l} = \lambda_{\mathrm{ch},n} + \rho \sum_{k \in \mathcal{N}(V_n) \setminus \{l\}} \lambda_{\psi_k \to V_n} - (1-\rho)\lambda_{\psi_l \to V_n}, \tag{3.33}$$

$$\lambda_{\psi_l \to V_n} = f_{\max^*}\left(\{\lambda_{V_m \to \psi_l}\}_{m \neq n}\right), \tag{3.34}$$

and

$$\lambda_{b,n} = \lambda_{\mathrm{ch},n} + \rho \sum_{k \in \mathcal{N}(V_n)} \lambda_{\psi_k \to V_n}. \tag{3.35}$$

For R-SPD-II, the messages from check nodes to variable nodes do not relate to the reweighted factor ρ.

3.1.6 Reweighted max-product decoder II (R-MPD-II)

From the modification experience of MPD and R-MPD, we can also modify R-SPD-II in the same way by replacing $f_{\max^*}\left(\{\lambda_{V_m \to \psi_l}\}_{m \neq n}\right)$ with $f_{\max}\left(\{\lambda_{V_m \to \psi_l}\}_{m \neq n}\right)$. These max-product algorithms will always have a lower complexity than the corresponding sum-product algorithms because of replacing the complex function $\max^*$ with max function.

The message updating rules are

$$\lambda_{V_n \to \psi_l} = \lambda_{\mathrm{ch},n} + \rho \sum_{k \in \mathcal{N}(V_n) \setminus \{l\}} \lambda_{\psi_k \to V_n} - (1-\rho)\lambda_{\psi_l \to V_n}, \tag{3.36}$$

$$\lambda_{\psi_l \to V_n} = f_{\max}\left(\{\lambda_{V_m \to \psi_l}\}_{m \neq n}\right), \tag{3.37}$$

and

$$\lambda_{b,n} = \lambda_{\mathrm{ch},n} + \rho \sum_{k \in \mathcal{N}(V_n)} \lambda_{\psi_k \to V_n}. \tag{3.38}$$

3.2 Complexity

The complexity of message updating algorithm per iteration for SPD is $\mathcal{O}(Nd_{\mathrm{v}})$ additions, $\mathcal{O}((N-K)d_{\mathrm{c}})$ $\max^*$ functions and $\mathcal{O}(Nd_{\mathrm{v}})$ additions corresponding to steps Eq.(3.16), (3.17) and (3.19). For step 2 of updating messages from check nodes to variable nodes, the complexity can be simply calculated as $\mathcal{O}((N-K)2^{d_c})$ $\max^*$ functions because every edge between check nodes and variable nodes has two $\max^*$ functions. However, if serial implementation of check nodes update uses, which is also called forward-backward algorithm, the complexity can be easily reduced to $\mathcal{O}((N-K)d_{\mathrm{c}})$ $\max^*$ functions. Fig.

Decoders	Complexity
SPD	Step 1: $\mathcal{O}(Nd_{\mathrm{v}})$ additions
	Step 2: $\mathcal{O}((N-K)d_{\mathrm{c}})$ max* functions
	Step 3: $\mathcal{O}(Nd_{\mathrm{v}})$ additions
MPD	Step 1: $\mathcal{O}(Nd_{\mathrm{v}})$ additions
	Step 2: $\mathcal{O}((N-K)d_{\mathrm{c}})$ max functions
	Step 3: $\mathcal{O}(Nd_{\mathrm{v}})$ additions
R-SPD	Step 1: $\mathcal{O}(Nd_{\mathrm{v}})$ additions + $\mathcal{O}(N)$ multiplications
	Step 2: $\mathcal{O}((N-K)d_{\mathrm{c}})$ max* functions + $\mathcal{O}(N)$ multiplications
	Step 3: $\mathcal{O}(Nd_{\mathrm{v}})$ additions + $\mathcal{O}(N)$ multiplications
R-MPD	Step 1: $\mathcal{O}(Nd_{\mathrm{v}})$ additions + $\mathcal{O}(N)$ multiplications
	Step 2: $\mathcal{O}((N-K)d_{\mathrm{c}})$ max functions + $\mathcal{O}(N)$ multiplications
	Step 3: $\mathcal{O}(Nd_{\mathrm{v}})$ additions + $\mathcal{O}(N)$ multiplications
R-SPD-II	Step 1: $\mathcal{O}(Nd_{\mathrm{v}})$ additions + $\mathcal{O}(N)$ multiplications
	Step 2: $\mathcal{O}((N-K)d_{\mathrm{c}})$ max* functions
	Step 3: $\mathcal{O}(Nd_{\mathrm{v}})$ additions + $\mathcal{O}(N)$ multiplications
R-MPD-II	Step 1: $\mathcal{O}(Nd_{\mathrm{v}})$ additions + $\mathcal{O}(N)$ multiplications
	Step 2: $\mathcal{O}((N-K)d_{\mathrm{c}})$ max functions
	Step 3: $\mathcal{O}(Nd_{\mathrm{v}})$ additions + $\mathcal{O}(N)$ multiplications

Table 3.1: Decoding process

3.2 shows the example of serial implementation. Comparing the three equations, max* function is sure more complex than addition. It even contains a log function that takes a lot of time. A lot of methods are tried to reduce the complexity of the log function while maximize the accuracy like piecewise linear function approximation method, sign-min approximation method [14]. On the other hand, d_{c} is bigger than d_{v} generally, so the message update from check nodes to variable nodes dominate the decoding complexity.

For MPD, the complexity is smaller than SPD because the message update rules replace max* function with max function. It is equivalent to discard the log function of max* function. Like the approximation methods, it trades off accuracy for less complexity. The complexities of R-SPD and R-SPD-II are similar to SPD. For R-SPD, there are extra $\mathcal{O}(N)$ multiplications for all three steps. For R-SPD-II, step 1 and step 3 have extra $\mathcal{O}(N)$ multiplications. R-MPD and R-MPD-II do the same thing based on the complexity of MPD. When ρ comes to 1, R-SPD and R-SPD-II revert to SPD. Similarly, R-MPD and R-MPD-II revert to MPD.

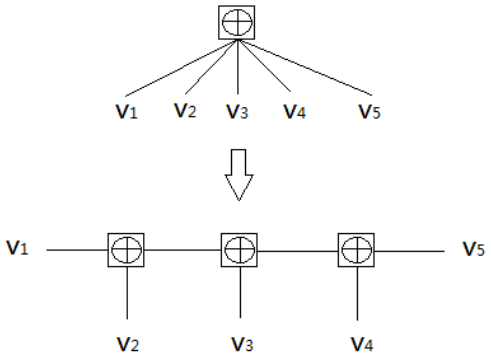

Figure 3.2: Serial implementation

3.3 Linear programming

In this section, we will first introduce linear programming decoding. Then decomposition method of linear programming decoding for large size LDPC codes will be demonstrated. LDPC codes can be decoded by several methods. One of the most popular methods is message passing algorithm and another method is linear programming decoding. It is found that for binary codes used over symmetric channels, a relaxed version of maximum likelihood decoding problem can be treated as a linear program (LP) [15].

3.3.1 LP decoder

From [16], the goal of linear programming decoder is to find the maximum likelihood codeword

$$\hat{\mathbf{x}}_{\mathrm{W}}(\mathbf{y}) = \arg\max_{\mathbf{x}} p(\mathbf{y}|\mathbf{x}) \tag{3.39}$$

$$= \arg\max_{\mathbf{x}\in\mathcal{C}} \prod_{n=1}^{N} p(y_n|x_n) \tag{3.40}$$

$$= \arg\max_{\mathbf{x}\in\mathcal{C}} \log \prod_{n=1}^{N} p(y_n|x_n) \tag{3.41}$$

$$= \arg\max_{\mathbf{x}\in\mathcal{C}} \sum_{n=1}^{N} \log p(y_n|x_n). \tag{3.42}$$

For Eq.(3.42), we have

$$\begin{aligned} \log p(y_n|x_n) &= \log p(y_n|x_n=0) + \log \frac{p(y_n|x_n)}{p(y_n|x_n=0)} && (3.43)\\ &= \log p(y_n|x_n=0) + x_n \log \frac{p(y_n|x_n=1)}{p(y_n|x_n=0)} && (3.44)\\ &= \log p(y_n|x_n=0) + \lambda_{\mathrm{ch},n} x_n. && (3.45) \end{aligned}$$

From Eq.(3.43) to Eq.(3.44), we use the fact that

$$\begin{aligned} &when\ x_n = 0, \qquad x_n \log \frac{p(y_n|x_n)}{p(y_n|x_n=0)} = 0\\ &when\ x_n = 1, \qquad x_n \log \frac{p(y_n|x_n)}{p(y_n|x_n=0)} = \log \frac{p(y_n|x_n)}{p(y_n|x_n=0)}. \end{aligned}$$

Because $p(y_n|x_n=0)$ is independent of x_n, so it could be considered as a constant value [17]. So Eq.(3.42) can be convert to

$$\begin{aligned} \hat{\mathbf{x}}_{\mathrm{W}}(\mathbf{y}) &= \arg\max_{\mathbf{x}\in\mathcal{C}} \sum_{n=1}^{N} \lambda_{\mathrm{ch},n} x_n && (3.46)\\ &= \arg\max_{\mathbf{x}\in\mathcal{C}} \mathbf{x}^T \boldsymbol{\lambda}_{\mathrm{ch}}. && (3.47) \end{aligned}$$

Then the problem we need to solve is

$$\begin{aligned} \text{minimize} \quad & -\boldsymbol{\lambda}_{\mathrm{ch}} \mathbf{x}^T\\ \text{subject to} \quad & \mathbf{h}_l^T \mathbf{x} = 0, \forall l\\ & \mathbf{x} \in \{0,1\}^N. \end{aligned}$$

Now what we have is a binary linear program. We would like to relax the constraints to a real field. So the constraint $\mathbf{x} \in \{0,1\}^N$ becomes $\mathbf{x} \in [0,1]^N$. In this way, the binary linear program could be solved with a standard LP solver. Because of relaxation, the result of LP decoder does not always give a binary codeword. If the result is a integer solution, it is a maximum likelihood result. Considering Eq.(2.5) as the example parity check matrix, we have

$$\begin{cases} check\ a : x_2 \oplus x_4 \oplus x_5 \oplus x_8 = 0\\ check\ b : x_1 \oplus x_2 \oplus x_3 \oplus x_6 = 0\\ check\ c : x_3 \oplus x_6 \oplus x_7 \oplus x_8 = 0\\ check\ d : x_1 \oplus x_4 \oplus x_5 \oplus x_7 = 0 \end{cases} \tag{3.48}$$

where we take *check a* as the example to demonstrate the transformation of the formulas. Set $\mathbf{x}_a$ to be the local codeword that contains the components of $\mathbf{x}$ that used in the *check a*:

$$\mathbf{x}_a = (x_2,\, x_4,\, x_5,\, x_8)^T \tag{3.49}$$

Define a matrix $\mathbf{S}_a$ that picks the components out

$$\mathbf{S}_a = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.50}$$

So $\mathbf{S}_a\mathbf{x} = \mathbf{x}_a$. Define a matrix $\mathbf{C}_a$ as the combinations of all local codewords that satisfy *check* a,

$$\mathbf{C}_a = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}. \tag{3.51}$$

Take one column from the matrix, you can see it is one possible codeword $\mathbf{x}_a$ that $x_2 \oplus x_4 \oplus x_5 \oplus x_8 = 0$. Now we have a set

$$\mathbf{1}^T\mathbf{w}_a = 1 \tag{3.52}$$

$$\mathbf{w}_a \in \{0,1\}^{2^{d_c-1}}, \tag{3.53}$$

in matrix forms,

$$\mathbf{w}_a \in \left\{ \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right\}. \tag{3.54}$$

It can be seen $\mathbf{x}_a = \mathbf{C}_a\mathbf{w}_a$, where these $\mathbf{x}_a$ satisfy *check* a. So the equation $\mathbf{h}_a^T\mathbf{x} = 0$ can be transformed into $\mathbf{S}_a\mathbf{x} = \mathbf{C}_a\mathbf{w}_a$.

After all, we have a new equivalent real linear program

$$\begin{aligned} \text{minimize} \quad & -\boldsymbol{\lambda}_{\text{ch}}\mathbf{x}^T \\ \text{subject to} \quad & \mathbf{S}_l\mathbf{x} = \mathbf{C}_l\mathbf{w}_l, \forall l \\ & \mathbf{1}^T\mathbf{w}_l = 1, \forall l \\ & \mathbf{w}_l \in \{0,1\}^{2^{d_c-1}}, \forall l \\ & \mathbf{x} \in \{0,1\}^N. \end{aligned} \tag{3.55}$$

where l represents l-th check.

3.3.2 Alternating direction method of multipliers

LP decoder introduced in previous section can be implemented with standard LP solver. However, the complexity grows exponentially when the degree of check nodes increases. It is too high to implement for large size LDPC codes. Different kinds of methods are tried to decompose the LP process. One useful decomposing method is an algorithm based on the Alternating Directions Method of Multipliers (ADMM) [18]. It introduced an iterative decoding way to work out the problem of linear program like message passing algorithm does.

ADMM is a classic convex optimization technique. It attracts great attention on solving problems of MAP in graphical models. We will apply LP problem with the template of ADMM which is similar to the process of message passing algorithm. Variable nodes update estimation of the codeword based on the information from check nodes and original measurements.

Add auxiliary variable $\mathbf{z}_l$ representing $\mathbf{C}_l\mathbf{w}_l$ in previous section. We have the LP problem:

$$\begin{aligned} \text{minimize} \quad & -\boldsymbol{\lambda}_{\text{ch}}\mathbf{x}^T \\ \text{subject to} \quad & \mathbf{S}_l\mathbf{x} = \mathbf{z}_l, \forall l \\ & \mathbf{z}_l = \mathbf{C}_l\mathbf{w}_l, \forall l \\ & \mathbf{1}^T\mathbf{w}_l = 1, \forall l \\ & \mathbf{w}_l \in \{0,1\}^{2^{d_c-1}}, \forall l \\ & \mathbf{x} \in \{0,1\}^N. \end{aligned} \tag{3.56}$$

To solve this problem, we could convert it into an augmented Lagrangian equation:

$$L_\mu(\mathbf{x},\mathbf{z},\boldsymbol{\sigma}) = -\boldsymbol{\lambda}_{\text{ch}}\mathbf{x}^T + \sum_l \boldsymbol{\sigma}_l^T(\mathbf{S}_l\mathbf{x} - \mathbf{z}_l) + \frac{\mu}{2}\sum_l \parallel \mathbf{S}_l\mathbf{x} - \mathbf{z}_l \parallel_2^2 . \tag{3.57}$$

Here $\boldsymbol{\sigma}_l \in \mathbb{R}^{d_c}$ are the Lagrange multipliers and μ is a fixed penalty parameter. Then we can get the iteration process of ADMM as

$$\mathbf{x}^{k+1} = \text{argmin}L_\mu(\mathbf{x},\mathbf{z}^k,\boldsymbol{\sigma}^k), \tag{3.58}$$

$$\mathbf{z}^{k+1} = \text{argmin}L_\mu(\mathbf{x}^{k+1},\mathbf{z},\boldsymbol{\sigma}^k), \tag{3.59}$$

$$\sigma_l^{k+1} = \sigma_l^k + \mu\left(\mathbf{S}_l\mathbf{x}^{k+1} - \mathbf{z}_l^{k+1}\right). \tag{3.60}$$

The $\mathbf{x}$ update can be derived to

$$\mathbf{x} = \prod_{[0,1]^N}\left(\mathbf{S}^{-1}\left(\sum_l \mathbf{S}_l^T\left(\mathbf{z}_l - \frac{1}{\mu}\boldsymbol{\sigma}_l\right) + \frac{1}{\mu}\boldsymbol{\lambda}_{\text{ch}}\right)\right). \tag{3.61}$$

where $\mathbf{S} = \sum_l \mathbf{S}_l^T\mathbf{S}_l$ and $\prod_{[0,1]^N}$ is the project function to the hypercube $[0,1]^N$. $\mathbf{S}_l^T\mathbf{S}_l$ is a diagonal matrix with (i,i) entry equal to $|N_v(i)|$. $N_v(i)$ are check nodes' indexes that

connecting to the i-th variable node. Let $\mathbf{z}_l^i$ be the ith component of $\mathbf{S}_l^T\mathbf{z}_l$ and $\boldsymbol{\sigma}_l^i$ be the ith component of $\mathbf{S}_l^T\boldsymbol{\sigma}_l$. Eq.(3.61) becomes

$$x_i = \prod_{[0,1]}\left(\frac{1}{|N_v(i)|}\left(\sum_{l\in N_v(i)}\left(\mathbf{z}_l^i - \frac{1}{\mu}\boldsymbol{\sigma}_l^i\right) + \frac{1}{\mu}\boldsymbol{\lambda}_{\text{ch}}^i\right)\right). \tag{3.62}$$

From [18], ADMM decoding algorithm is showed as below. Given parity check matrix $\mathbf{H}$, matrices $\mathbf{S}_l$, the log-likelihood ratio $\boldsymbol{\lambda}_{\text{ch}}$, ADMM algorithm does as following steps.

1. Initialize $\mathbf{z}_l$ and $\boldsymbol{\sigma}_l$ as all zeros vector
2. **do**
3. Update $x_i = \prod_{[0,1]}\left(\frac{1}{|N_v(i)|}\left(\sum_{l\in N_v(i)}\left(\mathbf{z}_l^i - \frac{1}{\mu}\boldsymbol{\sigma}_l^i\right) + \frac{1}{\mu}\boldsymbol{\lambda}_{\text{ch}}^i\right)\right)$
4. **for all** l **checks**
5. Update $\mathbf{v}_l = \mathbf{S}_l\mathbf{x} + \frac{\boldsymbol{\sigma}_l}{\mu}$
6. Update $\mathbf{z}_l = \prod_{\mathbf{PP}_d}(\mathbf{v}_l)$
7. Update $\boldsymbol{\sigma}_l = \boldsymbol{\sigma}_l + \mu(\mathbf{S}_l\mathbf{x} - \mathbf{z}_l)$
8. **end**
9. **while** $max_l \parallel \mathbf{S}_l\mathbf{x} - \mathbf{z}_l \parallel_\infty < \epsilon$

ϵ is the error tolerance. Signs $\prod_{[0,1]}$ and $\prod_{\mathbf{PP}_d}$ are project functions. Details about project functions are showed in [18].

The ADMM decoder works like message passing decoders. First, initialize the message from variable nodes. For all l checks, it will update $\mathbf{v}_l$, $\mathbf{z}_l$ and $\boldsymbol{\sigma}_l$ based on $\mathbf{x}$ which can be regard as respond message from check nodes to variable nodes. Then update the estimated codeword. The decoder will decode iteratively until it converges and fulfil the requirement of $max_l \parallel \mathbf{S}_l\mathbf{x} - \mathbf{z}_l \parallel_\infty < \epsilon$. If the error tolerance is small enough, we can say the constraints of $\mathbf{S}_l\mathbf{x} = \mathbf{z}_l$ in Eq.(3.56) are satisfied. In this way, the LP problem is solved.

4

Simulation results

4.1 Description of simulations

In this chapter, the simulations and results will be presented and interpreted. Seven different decoders are constructed and different kinds of LDPC codes are used to see the performances. The major aim we considered is bit error probability (BEP) and word error probability (WEP). Also, complexity and computing latency are cared about.

We will consider three distinct LDPC codes: (i) a long regular code from the 10Gb/s Ethernet standard, (ii) a short regular LDPC code; (iii) a long irregular LDPC code from Wimax standard.

4.2 Regular LDPC codes

In this section, we focus on the regular LDPC codes which are expected to have better performance than irregular LDPC codes. The reason as we said before, the four reweighted message passing algorithms are designed based on the fact that large size LDPC codes have regular alike character. So regular LDPC codes would be more appropriate to the reweighted algorithms than irregular codes.

In this section, regular sparse $\mathbf{H}$ with $K = 1664$ and $N = 2048$ is used. This LDPC code is from IEEE standard 802.3 [19]. The LDPC code has following degree profile:

- variable node degree: 6
- check node degree: 32

The matrix is shown in Fig. 2.1. We will focus on the performance with a SNR of 6dB, corresponding to the noise variable $\sigma^2 = 0.2512$. The results are similar for SNRs around 6dB. For six message passing decoders, 1000 iterations is set to be the maximum

iteration number. If the decoders converge to a legal LDPC codeword, the decoders will terminate the iteration. Otherwise the decoders will run up 1000 iterations until they stop. For ADMM LP decoder, the performance depends weakly on the setting of parameters when the parameters are large enough. From [18], we set penalty parameter $\mu = 5$ and error tolerance $\epsilon = 10^{-5}$. 10000 iterations is the maximum iteration number if the decoder converges to find a codeword. Otherwise the maximum iteration number extend to 11000 iterations. During the last 1000 iterations, the error tolerance is relaxed to $\epsilon = 10^{-4}$. We transmitted all-zero codewords, where random codewords and all-zero codewords are compared and proved to have almost the same performance. It means that the decoders are not sensitive to some specific codewords like all-zero codeword.

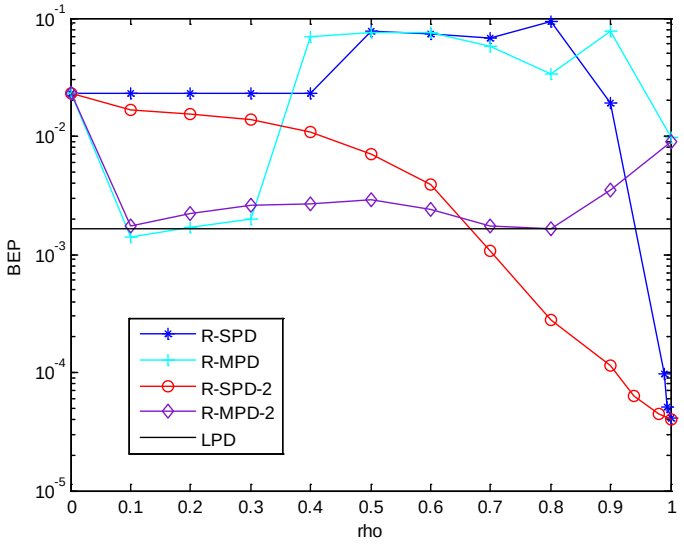

Figure 4.1: BEP after 1000 iterations

Fig. 4.1 shows the BEP after 1000 iterations for R-SPD, R-MPD, R-SPD-II and R-MPD-II. Since when $\rho = 1$ R-SPD and R-SPD-II convert to SPD, the two curves come to the same end. Respectively, R-MPD and R-MPD-II converge at $\rho = 1$. In terms of BEP, R-SPD-II offers the best performance when $\rho \geq 0.7$ among all six decoder except SPD. For R-MPD-II, the performance is quite close to LP decoder when $0.1 \leq \rho \leq 0.8$. R-MPD does the same thing when $0.1 \leq \rho \leq 0.3$. Unfortunately, R-SPD does not perform well until ρ gets close to 1. For R-SPD and R-MPD, their performances seem bad when $0.4 \leq \rho \leq 0.9$. Check the result, it is found that for this part of ρ the decoders come to a bad cycle. They fail to converge that almost all bits of the codeword flip to the opposite number. From the result, traditional SPD still has the best performance excluding complexity. On the contrary, R-MPD and R-MPD-II have better performance than MPD.

Fig. 4.2 shows the performance after 20 iterations which is more practical to implement in the hardware. It looks like the performance after 1000 iterations mostly.

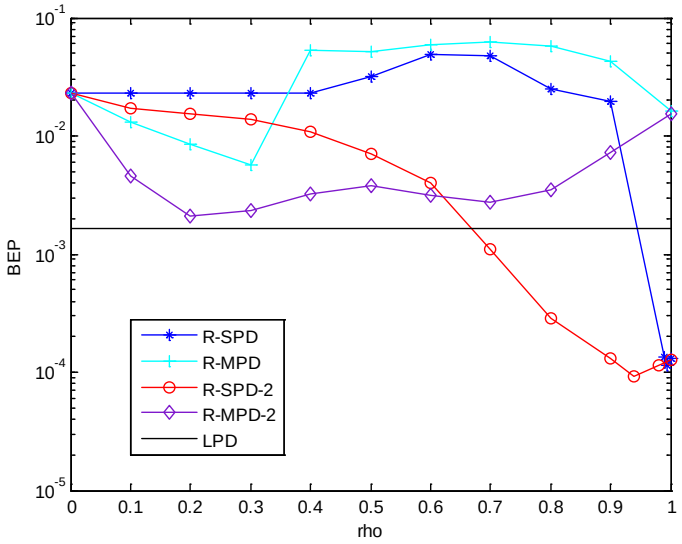

Figure 4.2: BEP after 20 iterations

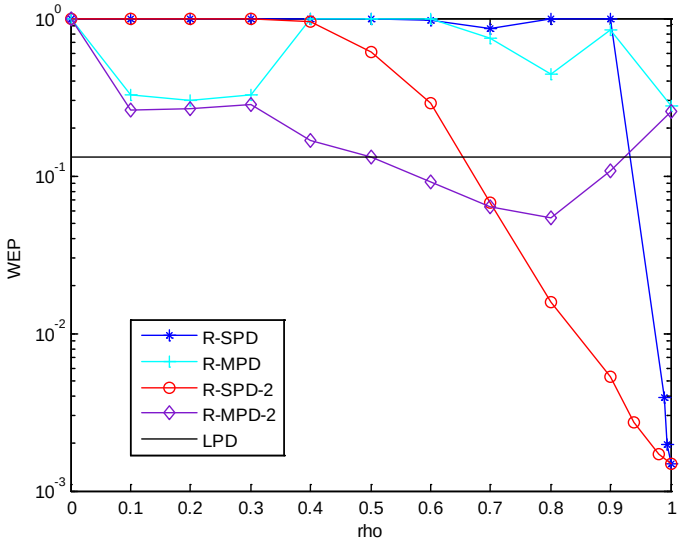

Figure 4.3: WEP after 1000 iterations

However, R-SPD and R-SPD-II have a better performance than SPD now comparing to the one after 1000 iterations. The curves have an obvious valley bottom before ρ comes to 1. We could find $\rho \approx 0.94$ for R-SPD-II and $\rho \approx 0.995$ for R-SPD that offer the best performance. The BEP of R-SPD-II is close to the BEP after 1000 iterations when $\rho \leq 0.9$. It means R-SPD-II has a quick converging speed since it reaches the

same magnitude of BEP. Because of slow convergence of SPD, reweighted sum-product algorithm could outperform SPD. R-MPD-II still has great result as it does after 1000 iterations and it is also close to the BEP after 1000 iterations. Unfortunately, R-MPD does not have good performance as R-MPD-II does while it does well for $0.1 \leq \rho \leq 0.3$. LP decoder performs well comparing to the reweighted message passing decoders when $\rho \leq 0.6$.

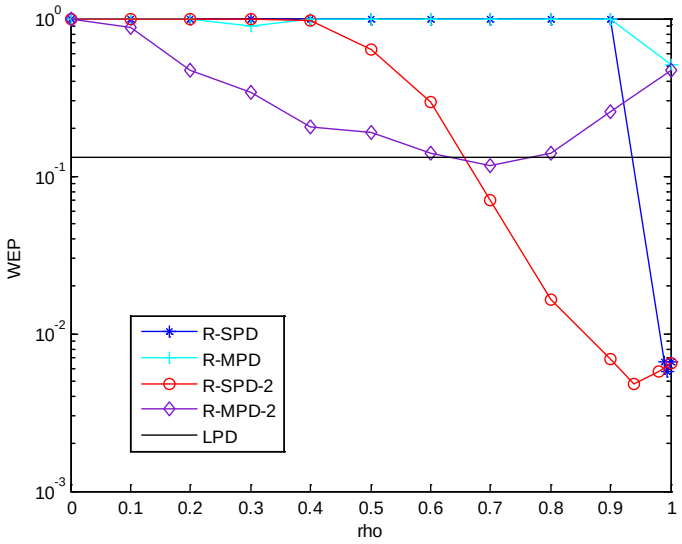

Figure 4.4: WEP after 20 iterations

In terms of WEP, Fig. 4.3 and Fig. 4.4 show the WEP after 1000 iterations and 20 iterations. The curves are similar to the ones of BEP. R-MPD-II has the different shape so that we can a better choice of ρ locates $\rho \approx 0.8$. From Fig. 4.4, we can see a bigger ρ got a lower WEP than small ρ. Comparing these decoders, it is easy to find out that reweighted message passing algorithms version 2 has better performance than version 1.

Results are recorded after every iteration. So we could trace the best ρ that minimize the BEP. Here we just consider BEP as the reference. The results are shown in Fig. 4.5. At 20 iterations, the best ρ is 0.995 for R-SPD and 0.94 for R-SPD-II. These two curves will return back to 1 when iteration number gets bigger. If more ρs are tried, the curves would be smoother. Because the BEP of MPD at 6dB is not in the same magnitude as SPD does. So we choose to find the best ρ for R-MPD and R-MPD-II at 6.5dB. The best ρ of R-MPD moves from 0.32 at 20 iterations to 0.2 at 1000 iterations. For R-MPD-2, the best ρ is always 0.7.

Fig. 4.6 shows the BEP as a function of iteration index at SNR=6dB. From this figure we can see the BEP after every iteration. It is clear that different message passing decoders have different convergence time. SPD, MPD, R-SPD, R-SPD-2 and R-MPD-2 can converge within 100 iterations while R-MPD needs about 200 iterations. Comparing

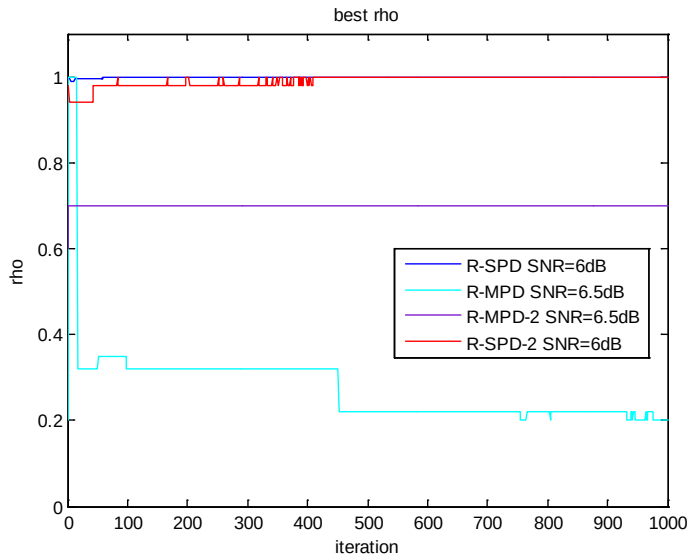

Figure 4.5: Best ρ for different iterations

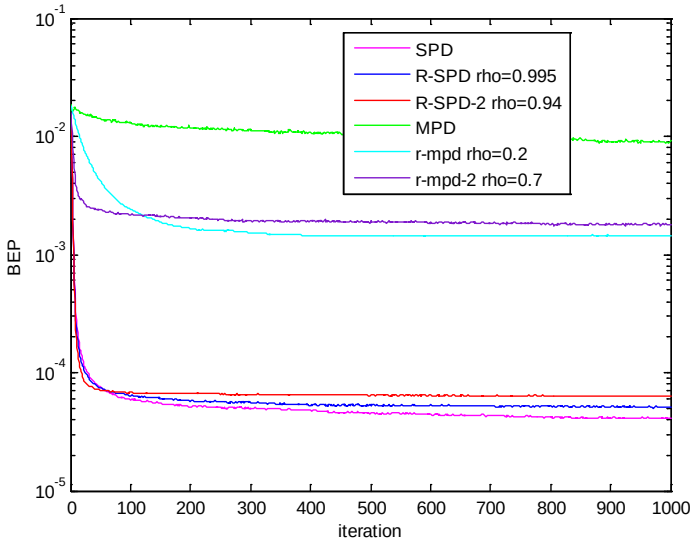

Figure 4.6: BEP vs iteration

R-MPD with $\rho = 0.2$ and R-MPD-2 with $\rho = 0.7$, R-MPD will win after fully converged. However, R-MPD-2 offers better performance within 100 iterations. The same situation happens for the serials of SPD. From Fig. 4.5 we know after 1000 iterations R-SPD and R-SPD-2 can not find a best ρ that can defeat SPD in terms of BEP.

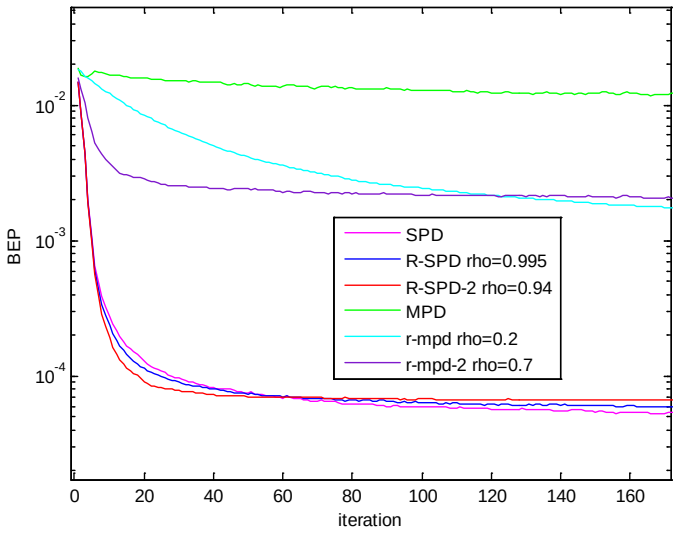

Figure 4.7: BEP vs iteration(enlarged)

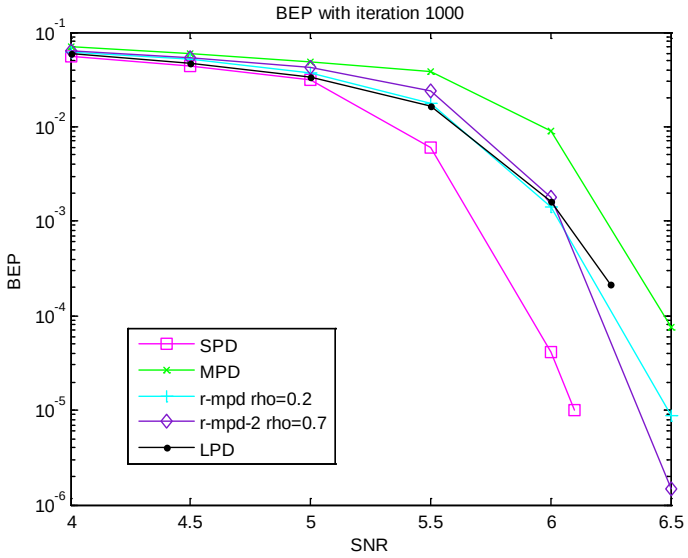

Figure 4.8: BEP with best ρ after 1000 iterations

Fig. 4.7 is the partially enlarged figure. After 100 iterations, it is obvious that SPD is the best. But because of slow convergence, SPD is the worst among the three decoders when iteration number is around 20. Reweighted decoders version 2 offer the quickest convergence. Both R-SPD-2 and R-MPD-2 just need about 20 to 30 iterations when they converge to a magnitude of BEP close to the one after 1000 iterations.

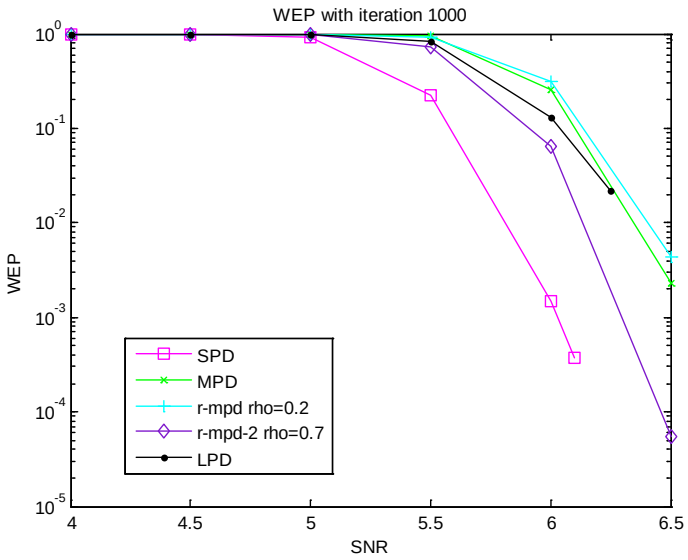

Figure 4.9: WEP with best ρ after 1000 iterations

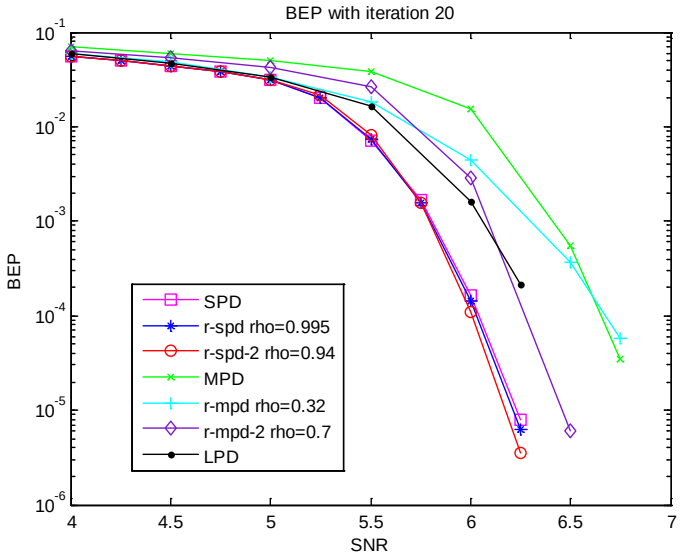

Figure 4.10: BEP with best ρ after 20 iterations

Fig. 4.8 shows the BEP curve of all decoders with the best ρ. Because R-SPD and R-SPD-II perform the best when $\rho = 1$, so they would be the same curve of SPD. Traditional MPD is about 0.5dB worse than SPD. The curves of R-MPD, R-MPD-II and LPD locate between the curves of SPD and MPD. From the trend of the curves, R-MPD-II performs better, followed by R-MPD and LPD. Fig. 4.9 shows the corresponding

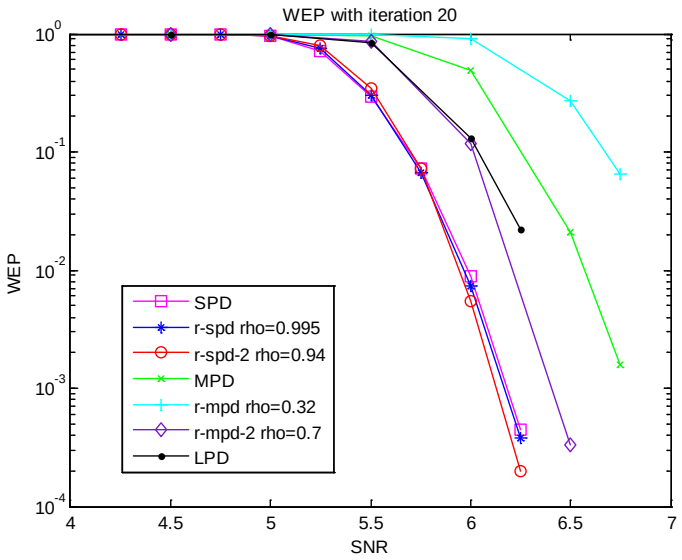

Figure 4.11: WEP with best ρ after 20 iterations

WEP. It is found that R-MPD-II is still the best one among the three decoders. R-MPD is even worse than traditional MPD.

Fig. 4.10 and Fig. 4.11 show the BEP and WEP after 20 iterations. Now R-SPD and R-SPD-II is slightly better than SPD with their best ρ. Among all the decoders, R-SPD-II is the best one if just BEP is considered. If complexity is considered which is very important in practical, R-MPD-II is also a good alternate. It is 0.25dB worse than the series of SPD. LPD is also good except that the compute latency is too large. If the maximum iteration number of ADMM LPD is reduced, the curve of BEP is sure to move right. The curves of WEP are similar except the one for R-MPD which is rather bad.

Fig. 4.12 and Fig. 4.13 show the mean iteration number the decoders need. The mean iteration number is also very important because the computing latency has a close relation with the iteration number. Besides, another important factor is complexity that we pay a lot of attention to. In general, the less mean iteration number used the BEP and WEP are smaller. Because a small average iteration number means quick convergence since the decoder will terminate the iteration when a codeword is detected. In the figures, MPD need more iterations than SPD does. As we mentioned before, max-product algorithm replaces the max* function with max function. It actually relax the accuracy of the message updated from check nodes to variable nodes. So it will slow down the convergence process which means it needs more iterations to converge. Comparing MPD and R-MPD-II in Fig. 4.13, at SNR=6dB the average iteration number of MPD is smaller than R-MPD-II but the BEP and WEP are bigger. It means MPD is more likely to converge to a wrong codeword. From the performance of MPD, R-MPD and R-MPD-II, it is clearly shown that R-MPD-II is the most accurate and powerful

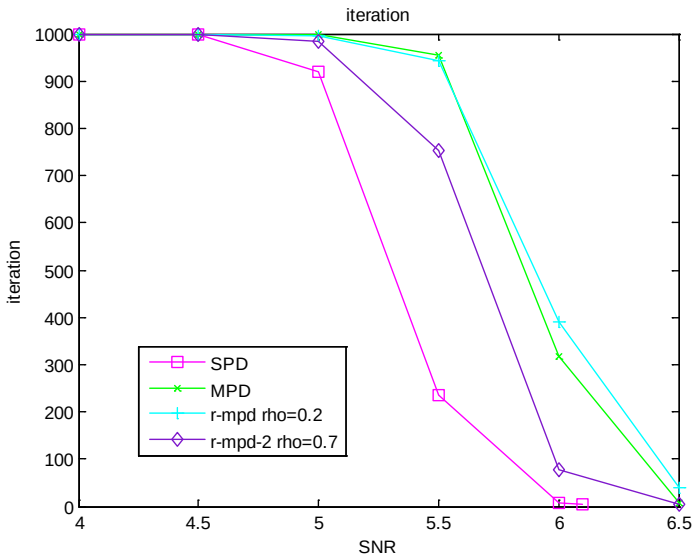

Figure 4.12: Mean iteration number within 1000 iterations

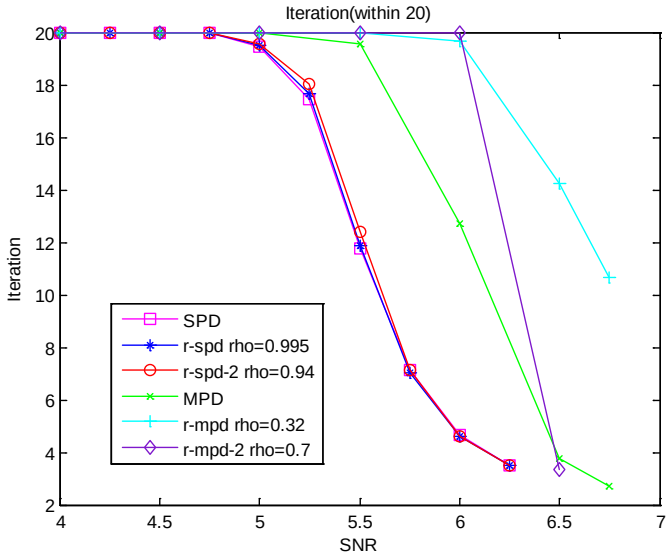

Figure 4.13: Mean iteration number within 20 iterations

one among the series of max-product algorithms. It is also comparable with the series of sum-product algorithms if complexity and latency are considered. The curves of SPD, R-SPD and R-SPD-II are very close to each other.

4.3 Short regular LDPC codes

Although long LDPC codes have better performance than short codes, short LDPC codes may win in complexity and latency. Short codes also have a certain application. Here we used a parity check matrix $\mathbf{H}$ with the size of $K = 128$ and $N = 256$.

- variable node degree: 3 (3 out of 256 nodes have the degree of 4)
- check node degree: 6 (3 out of 128 nodes have the degree of 7)

Although there are a few nodes have one more degree, it is still regular alike LDPC code. We will focus on the performance with a SNR of 4dB, corresponding to the noise variable $\sigma^2 = 0.3981$. The other settings are the same with the previous long regular code.

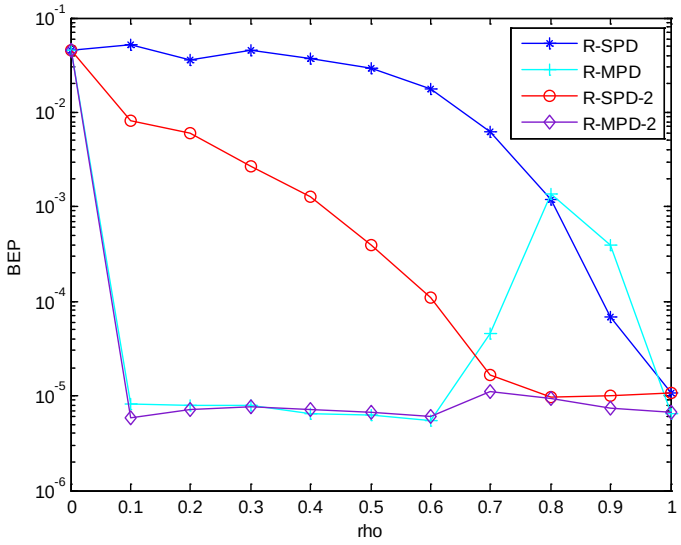

Figure 4.14: BEP after 1000 iterations

BEP after 1000 iterations is shown in Fig. 4.14. The performance is similar to Fig. 4.1 except that the effect of reweighted method is more obvious. R-SPD does not have good performance until ρ gets to 1. R-SPD-II will decrease the BEP gradually when ρ increases from 0 to 0.8. It has the performance as good as SPD when $\rho > 0.8$. R-MPD performs well for $0.1 \leq \rho \leq 0.6$ and then it turns into a bad performance like the case of long code. R-MPD-II again has the best performance as it works very good for almost any $\rho < 1$. R-SPD and R-MPD are bad for big ρ like 0.8, unlike the situation of long code that even worse than the BEP at $\rho = 0$, the decoders still succeed to converge. Different from the performance of traditional MPD in long code, now it is as good as SPD. It may because the codeword length is short and the code structure is simpler that makes the influence of relaxation is not big that it can still converge quickly compared to

SPD. For a longer code, it is sure that more iterations are needed. However, considering computing latency, it is not allowed to do so. Even 1000 iterations is very prohibitive in practical. Although the codes are different, we can still see the common points between these two codes.

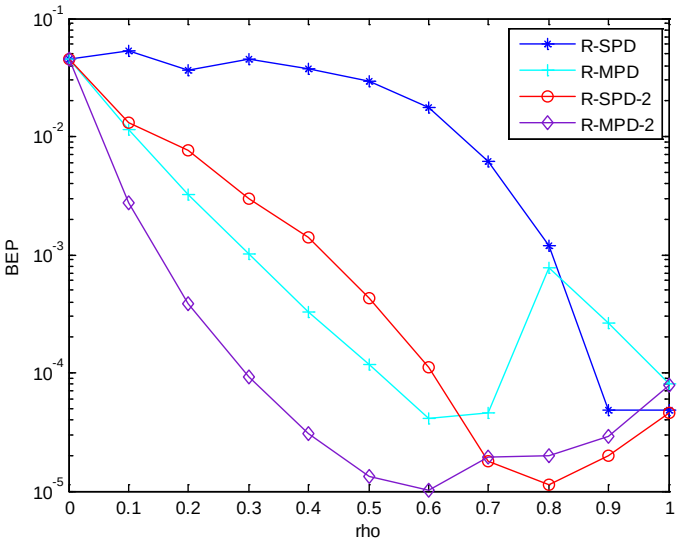

Figure 4.15: BEP after 20 iterations

Then we turn to the performance after 20 iterations shown in Fig. 4.15. Now SPD and MPD no longer offer better performance than R-SPD, R-SPD-II, R-MPD and R-MPD-II. The best ρ for R-SPD-II is $\rho \approx 0.8$ and the best one for R-MPD-II is $\rho \approx 0.6$. R-MPD with $0.6 < \rho < 0.7$ offers the best performance. Judging from the trend of the curve, R-SPD will also have valley bottom between $\rho = 0.9$ and $\rho = 1$.

Fig. 4.16 and Fig. 4.17 show the WEP after 1000 iterations and 20 iterations. The shape of the curves is almost the same except that the curves in Fig. 4.16 decline for $\rho > 0.8$ compared to the curves of BEP in Fig. 4.14. It shows that there are more error bits in the wrong codewords.

According to the performance of all decoders, R-MPD-II is the promising one followed by R-SPD-II. Generally speaking, uniformly reweighted message passing algorithm version 2 is better than version 1. Even the complexity is a little smaller.

4.4 Irregular LDPC codes

Regular LDPC codes can offer good performance, but more and more irregular LDPC codes appear to show better performance. Some new standards use irregular LDPC codes like WiMax. So it is significant to apply reweighted algorithms to irregular LDPC

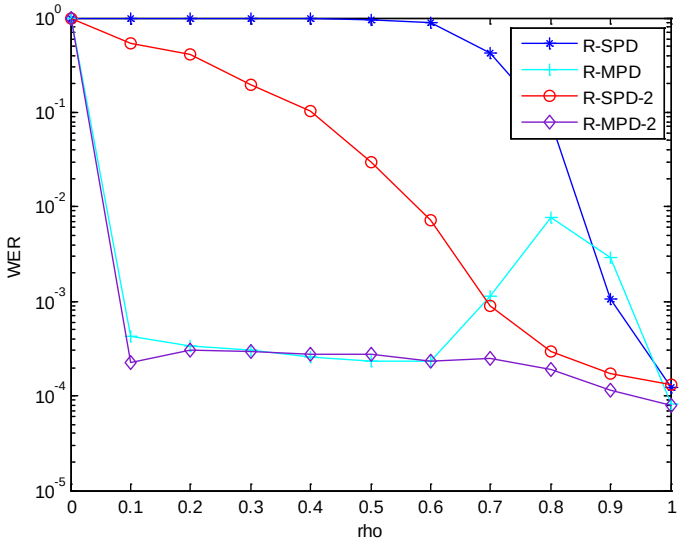

Figure 4.16: WEP after 1000 iterations

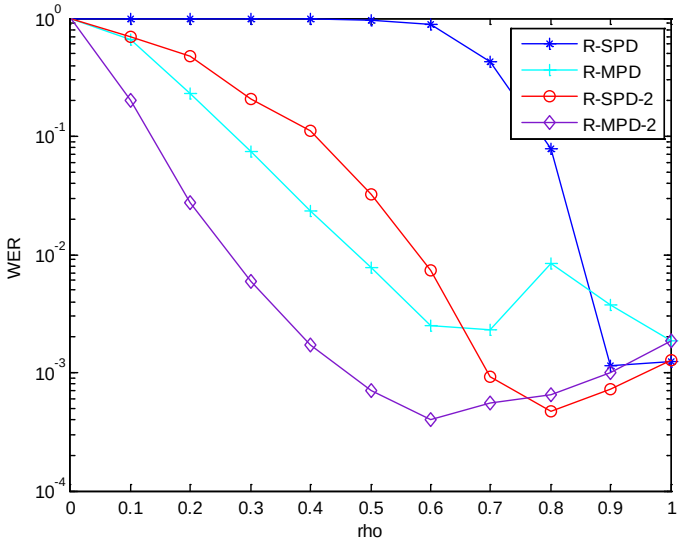

Figure 4.17: WEP after 20 iterations

codes. Here we used the parity check matrix **H** from WiMax standard with the size of $K = 1152$ and $N = 2304$.

- variable node degree: 2 (fraction 1056/2304), 3 (fraction 768/2304), 6 (fraction 480/2304)

- check node degree: 6 (fraction 768/2304), 7 (fraction 384/2304)

Fig. 4.18 shows the matrix. From the figure, we can see its character of irregularity. We will focus on the performance with a SNR of 1.75dB, corresponding to the noise variable $\sigma^2 = 0.6683$. The other settings are the same with the previous codes.

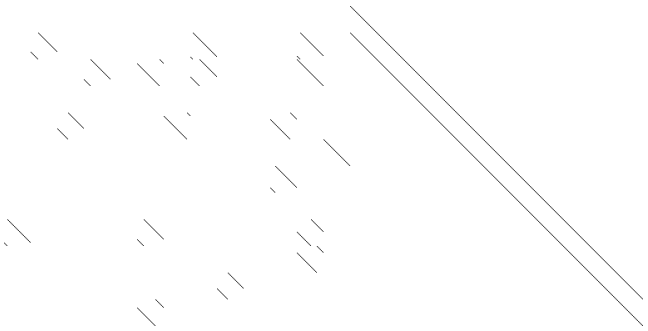

Figure 4.18: Matrix of **H**

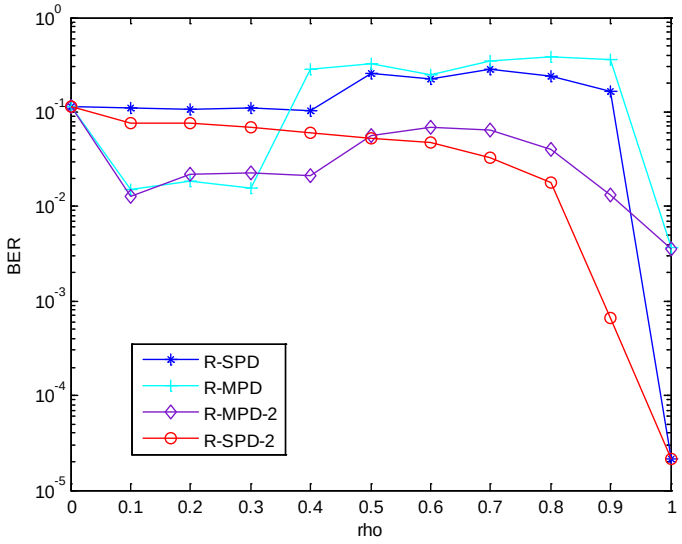

Figure 4.19: BEP after 1000 iterations

Fig. 4.19 and Fig. 4.20 show the BEP after 1000 iterations and 20 iterations. Comparing to Fig. 4.1 and Fig. 4.2, we can find that the shape of the curves are similar for $\rho \leq 0.8$. The performance is a little worse as the BEP is larger. But for $\rho > 0.8$, the curves decline quickly for all four reweighted method. Especially traditional SPD and MPD offer the best performance. There is no ρ for R-MPD and R-MPD-II that can

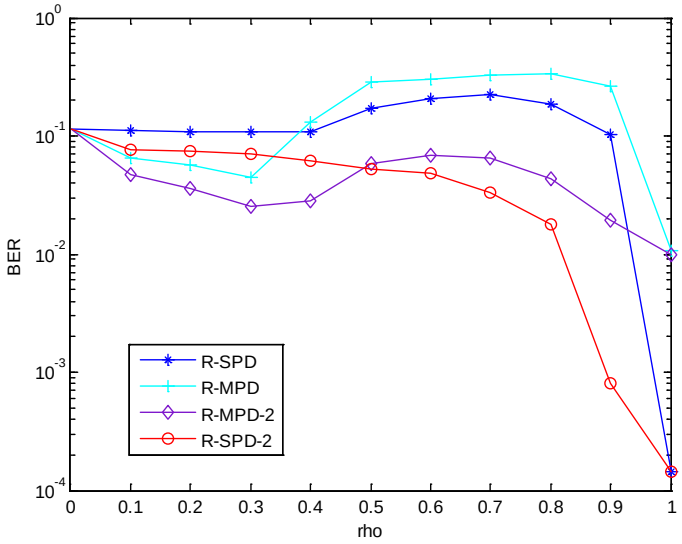

Figure 4.20: BEP after 20 iterations

defeat MPD. The situation is the same in the case of 20 iterations. The convergence time for SPD and MPD is quite short.

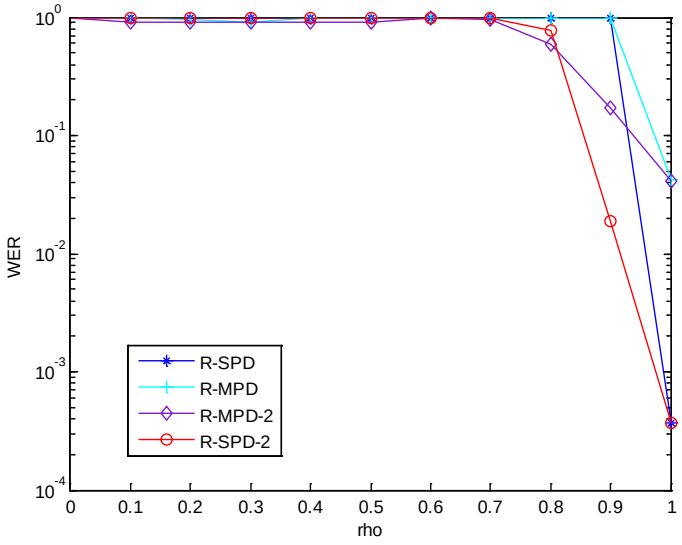

Figure 4.21: WEP after 1000 iterations

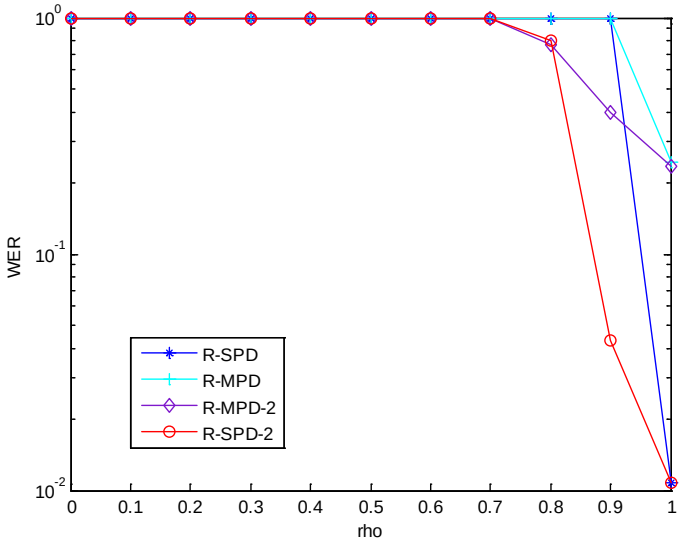

Figure 4.22: WEP after 20 iterations

Fig. 4.21 and Fig. 4.22 show the WEP after 1000 iterations and 20 iterations. The performance is even more terrible for any $\rho < 0.9$.

The four reweighted message passing algorithms do not perform well for the case of irregular LDPC codes. Because these four algorithms belong to uniformly reweighted belief propagation algorithm which is primarily designed based on the premise of a regular code structure. Actually, a regular LDPC code does not mean it has a perfect structure that fits uniformly reweighted algorithms. However comparing to irregular LDPC codes, regular codes are more closer to the structure that sets all reweighted factor to be the fixed number. So the four reweighted decoders are more feasible for regular LDPC codes. Despite the irregularity, looking at the right part of the parity check matrix $\mathbf{H}$, we can also feel that regular pattern of the code. The matrix $\mathbf{H}$ is constructed based on a base matrix in a certain way [20]. Short cycles can be counted in order to set the value of the reweighted factor ρ for every check node or even in more detail. There are methods for counting short cycles in [21] and [22]. A more sophisticated reweighted decoder may improve the performance of irregular LDPC codes [23].

5

Conclusion

The ultimate goal of this thesis is to implement six reweighted message passing algorithms and compare their performances based on different LDPC codes especially long standard codes. Seven decoders are compared to find the best decoder or a good scheme considering BEP, complexity and latency. Also three different kinds of LDPC codes are tried with reweighted decoders and compare with each other.

The four reweighted decoders are designed based on URW-BP which is suitable for regular LDPC codes. The results of the simulations support this. We can find reweighted decoders can outperform traditional SPD and MPD except that after large enough iterations SPD is still better than R-SPD and R-SPD-II in terms of BEP and WEP. Considering latency, the maximum iteration number will be limited to 20 iterations probably in practical. Because of slow convergence of SPD and MPD, the performances of R-SPD, R-SPD-II, R-MPD and R-MPD-II have more advantages. In general, the serials of SPD perform better than the serials of MPD. The BEP and WEP are smaller at the same SNR. Among MPD, R-MPD and R-MPD-II, R-MPD-II is the best choice. R-MPD-II is about 0.25dB worse than SPD in Fig. 4.10. If SNR in real channel is good enough, R-MPD-II is competitive with SPD. In addition, R-MPD-II has advantage in complexity and latency. So R-MPD-II is a good choice to apply in regular codes. In the results for R-SPD and R-MPD, it is found that for $0.4 < \rho < 0.9$ they fail to converge. It needs to be studied more in the future.

The performance of LP ADMM decoder is close to R-MPD-II. But according to the trend, R-MPD-II will have lower BEP and WEP from Fig. 4.10. Standard LP decoder is only feasible for codes with small size. When the size gets bigger, the complexity of standard LP decoder is prohibitive. We used ADMM algorithm to decompose the process so LP ADMM decoder can decodes the long code. However, the performance is not so good since we set a huge maximum iteration number. The computing latency is very large while the BEP and WEP are not as good as SPD.

For short regular codes, the advantages of reweighted decoders are obvious. Again, R-MPD-II is the best option. However, the reweighted decoders do not perform well in the case of irregular LDPC codes. Irregular LDPC codes may also have improved performance if the codes are close to a regular structure. In this thesis, we used the code from Wimax as the example of irregular code. It is so irregular seeing from Fig. 4.18 that uniformly reweighted algorithms can hardly make achievements. The solution of this problem is to set reweighted factor separately. But how to set the values stays to be studied. Different irregular has different structures, so every time we can only design a certain array of ρ for the specific matrix H. How to apply reweighted algorithm better to irregular LDPC codes remains to be solved while setting values of ρ for large size code is very complex. However, irregular codes have better performance than regular codes and there will be more good irregular codes. So it is a good trend to study more reweighted algorithm applied in irregular codes.

Bibliography

[1] R. Gallager, Low-density parity-check codes, IEEE Transactions on Information Theory 8 (1) (1962) 21–28.

[2] R. Tanner, A recursive approach to low complexity codes, Information Theory, IEEE Transactions on 27 (5) (1981) 533–547.

[3] M. Valenti, R. I. Seshadri, Turbo and ldpc codes: implementation, simulation, and standardization.
URL `http://wireless.vt.edu/symposium/tutorials/2006/Valenti.pdf`

[4] M. Wainwright, T. Jaakkola, A. Willsky, A new class of upper bounds on the log partition function, IEEE Transactions on Information Theory 51 (7) (2005) 2313–2335.

[5] H. Wymeersch, F. Penna, V. Savic, Uniformly reweighted belief propagation for estimation and detection in wireless networks, IEEE Transactions on Wireless Communications 11 (4).

[6] B. M. Leiner, Ldpc codes - a brief tutorial.
URL `http://bernh.net/media/download/papers/ldpc.pdf`

[7] H. Loeliger, An Introduction to factor graphs, IEEE Signal Processing Magazine 21 (1) (2004) 28–41.

[8] F. Kschischang, B. Frey, H.-A. Loeliger, Factor graphs and the sum-product algorithm, IEEE Trans. Inform. Theory 47 (2001) 498–519.

[9] Y. Mao, F. Kschischang, B. Li, S. Pasupathy, A factor graph approach to link loss monitoring in wireless sensor networks, Selected Areas in Communications, IEEE Journal on 23 (4) (2005) 820–829.

[10] H. Wymeersch, Iterative Receiver Design, Cambridge University Press, 2007.

[11] P. Robertson, E. Villebrun, P. Hoeher, A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain, Proc. IEEE International Conference on Communications 2 (1995) 1009–1013.

[12] Y. Jian, H. Pfister, Convergence of weighted min-sum decoding via dynamic programming on trees, Arxiv preprint arXiv:1107.3177.

[13] H. Wymeersch, F. Penna, V. Savic, Uniformly reweighted belief propagation: A factor graph approach, in: IEEE International Symposium on Information Theory, 2011.

[14] X.-Y. Hu, E. Eleftheriou, D. M. Arnold, A. Dholakia, Efficient implementations of the sum-product algorithm for decoding LDPC codes, Global Telecommunications Conference, 2001. GLOBECOM '01. IEEE 2 (2001) 1036–1036E vol.2.

[15] J. Feldman, Decoding error-correcting codes via linear programming, PhD thesis.

[16] J. Feldman, M. Wainwright, D. Karger, Using linear programming to decode binary linear codes, IEEE Transactions on Information Theory 51 (3) (2005) 954–972.

[17] N. Traore, S. Kant, T. L. Jensen, Message passing alogrithm and linear programming decoding for LDPC and linear block codes, Aalborg University.

[18] S. Barman, X. Liu, S. C. Draper, B. Recht, Decomposition methods for large scale lp decoding.

[19] Part 3: Carrier sense multiple access with collision detection (csma/cd) access method and physical layer specifications, IEEE 802.3 standard.

[20] Part 16: Air interface for fixed and mobile broadband wireless access systems, IEEE 802.16e standard.

[21] T. R. Halford, K. M. Chugg, An Algorithm for Counting Short Cycles in Bipartite Graphs, Information Theory, IEEE Transactions on 52 (1) (2006) 287–292.

[22] M. Karimi, A. H. Banihashemi, A message-passing algorithm for counting short cycles in a graph, Information Theory Workshop (Jan. 2010) 1–5.

[23] J. Liu, R. de Lamare, Low-latency reweighted belief propagation decoding for LDPC codes, Communications Letters, IEEE PP (99) (2012) 1–4.