

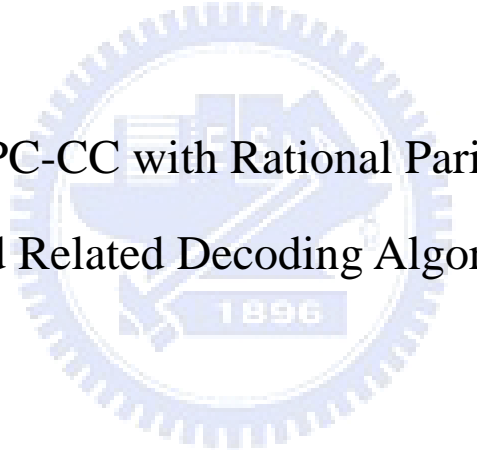
國立交通大學

電信工程學系碩士班

碩士論文

具分式校驗矩陣之低密度校驗迴旋碼及其相關解碼  
演算法研究

A Study on LDPC-CC with Rational Parity-Check Matrices  
and Related Decoding Algorithms

The logo of Tsinghua University is a circular seal with a gear-like outer edge. Inside the seal, there is a stylized building and the year '1896' at the bottom.

研究生：賴志傑

指導教授：王忠炫 博士

中華民國九十八年七月

具分式校驗矩陣之低密度校驗迴旋碼及其相關解碼演算法研究

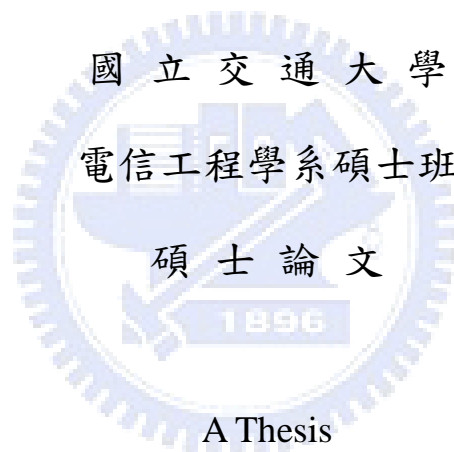
A Study on LDPC-CC with Rational Parity-Check Matrices and  
Related Decoding Algorithms

研 究 生：賴志傑

Student : Chih-Chieh Lai

指導教授：王忠炫 博士

Advisor : Dr. Chung-Hsuan Wang



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

# 具分式校驗矩陣之低密度校驗迴旋碼 及其相關解碼演算法研究

研究生: 賴志傑

指導教授: 王忠炫

國立交通大學電信工程學系碩士班

## 摘要

低密度校驗碼(LDPC)是近幾年來最爲人所熟知的靠近通道容量的編碼。低密度校驗螺旋碼(LDPC-CC)則爲結合低密度校驗碼與螺旋碼的特性，具有低密度的非零位置在校驗矩陣中，同時尤於螺旋碼的結構而能被編碼成任意的長度，卻只需要一個解碼器即可進行解碼的動作。除此之外，解碼器在經過一開始的延遲時間後就能連續地輸出解碼後的資料。由於以上的特色，使得在可能更改資料框架的大小以及即時作業的系統中，低密度校驗迴旋碼比起低密度校驗方塊碼來得更合適。因此本碩士論文即從兩方面討論低密度校驗迴旋碼的解碼特性。一爲提供一新觀點來針對具分式校驗矩陣之低密度校驗迴旋碼做解碼的動作，二爲提出針對低密度校驗迴旋碼之位翻轉解碼演算法的改善。

目前，具有良好更正能力的低密度校驗迴旋碼多爲利用代數結構所建造出來的，而常被廣泛使用的則是利用半循環式低密度校驗碼來建構出低密度校驗迴旋碼。但是其對校驗矩陣的限制是每個元素必須是零或單項式或是二項式。若有一元素爲三次多項式，則代表該碼的Tanner graph的最短周長小於等於六。這代表著在利用疊代信息傳遞解碼演算法進行解碼時容易造成信息相關性過大而使得編碼增益大幅降低。在本篇論文中，我們用新的觀點來針對具分式校驗矩陣之低密度校驗迴旋碼做解碼的動作，經由改良過的Tanner graph建造過程，我們避免了最短周長爲四的情況。在模擬結果中，更表述了我們的方法能使得以往認爲較差編碼增益的低密度校驗迴旋碼具有良好的編碼增益。於此同時，我們產生與具有良好編碼增益的低密度校驗迴旋碼之校驗矩陣的等價分式校驗矩陣，再經由我們所提出的方法進行解碼，所得的編碼增益媲美與原先的解碼結果。

另一方面，我們亦針對低密度校驗迴旋碼提出改善之位翻轉解碼演算法。在前人的研究中，針對低密度校驗迴旋碼之解碼器裡的每一個處理器提出了翻轉閾值的設定。此設定的模擬結果比Gallager提出的位翻轉解碼演算法有較佳的增益。但是事先設定閾值讓處理器不能對解碼

過程的真實情況做出對應的改變。因此我們提出了動態提高閾值的解碼演算法，每個處理器只有在必要時候才會提高閾值以做為因應。模擬結果更告訴了我們所提出的改善方法能更有效率的偵錯，並獲得更佳的編碼增益。



# A Study on LDPC-CC with Rational Parity-Check Matrices and Related Decoding Algorithms

Student: Chih-Chieh Lai

Advisor: Chung-Hsuan Wang

Department of Communication Engineering

National Chiao Tung University

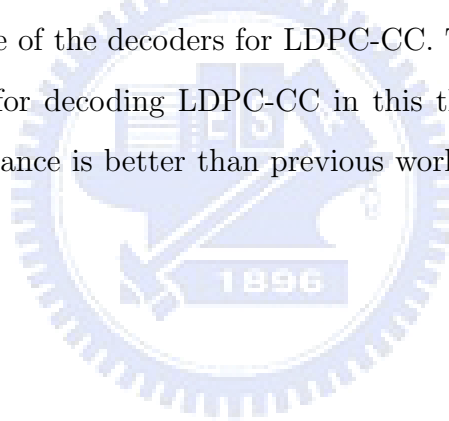
## Abstract

Low-density parity-check convolutional codes (LDPC-CC) are convolutional codes with low-density of ones in the scalar form of parity-check matrices. They have good features that do not exist in LDPC block codes such as they can be encoded with arbitrary length by simple shift registers, and can be decoded by only one decoder. On the other hand, a decoder for LDPC block code can only decode codewords with fixed length. Besides, the decoder pops out the decoded outputs continuously, and makes LDPC-CC more adequate to real time operating systems than LDPC block codes. In this thesis, we discuss the decoding algorithms for LDPC-CC in two perspectives. First, we propose a new perspective for decoding LDPC-CC with rational parity-check matrices. Second, we present an improved bit-flipping decoding algorithm for LDPC-CC.

Recently, good performance LDPC-CC are constructed from quasi-cyclic LDPC (QC-LDPC) codes. There are zeros, monomial or binomial in the parity-check matrices of those LDPC-CC. In this thesis, we discuss the possibility that LDPC-CC with rational parity-check matrices can still have good bit error rate (BER) performance. We propose a new perspective of constructing Tanner graph to represent the rational in parity-check matrix. There are no cycles of length 4, and the iterative message passing algorithm can be free from high dependence in the examples. The simulation results show that the BER performance is better than that of previous perspective for decoding LDPC-CC with rational parity-check matrices about 1 dB improvement and 1 dB away from maximum likelihood (ML) decoding results in the cases of LDPC-CC with small memory. In the cases of LDPC-CC with large

memory, our simulation results show about 0.7 dB improvement. We also generate rational parity-check matrices equivalent to the monomial parity-check matrices constructed from QC-LDPC. The simulation results show that there are no difference between two BER performance.

In addition to the decoding for LDPC-CC with rational parity-check matrices, we also discuss the hard decision decoding for LDPC-CC. In many applications such as the communication in flash memory, the power consumption and the volume of the system are the most important concern, and a simplified decoding algorithm for LDPC codes is needed. Bit-flipping algorithms are good choices because they provide decoded results very quickly by utilizing hard decisions of received sequence. There is much research on modified bit-flipping algorithms for different type of LDPC block codes but few discussion about LDPC-CC. Unluckily, the bit-flipping algorithm for LDPC block codes are not well suitable for decoding LDPC-CC due to the nature of the decoders for LDPC-CC. Therefore, we propose a modified bit-flipping algorithm for decoding LDPC-CC in this thesis. The simulation results show that the BER performance is better than previous work about 2 dB improvement in many cases.



## 誌 謝

本篇論文得以順利完成，首先要特別感謝我的指導教授 王忠炫博士。當我在課業和研究上有不能理解和困難時，老師總是適時的給予協助和引導，並耐心的教導和提出指正。在老師身上，讓我學習到很多通訊專業的知識和做研究有效的方法和嚴謹認真的態度，也謝謝老師對生活和未來規劃中給予的意見和鼓勵。

此外，也要感謝實驗室的學長們、同學們在課業和論文研究上的討論和幫助，尤其是博士班翁健家學長，在研究所的兩年內給予了非常多的幫助，使得在研究上以及通訊領域上的知道都比以前更加的紮實。

最後要感謝我的家人默默的支持著我所有的決定，因為有你們讓我能更安心的全力朝研究邁進。最後的最後，要感謝一直陪在身邊的女朋友楊植纓，因為有妳的鼓勵與照料，讓我每在氣力放盡的時候還能充滿電繼續努力。

感謝所有人的幫忙，未來我還會繼續努力的，謝謝。



# Contents

|   |             |
|---|-------------|
| <b>Chinese Abstract</b>   | <b>I</b>    |
| <b>English Abstract</b>   | <b>III</b>  |
| <b>Acknowledgement</b>  | <b>V</b>    |
| <b>Contents</b>   | <b>VI</b>   |
| <b>List of Figures</b>  | <b>VIII</b> |
| <b>1 Introduction</b>   | <b>1</b>    |
| 1.1 Motivation . . . . .  | 1           |
| 1.2 Organization of Thesis . . . . .  | 3           |
| <b>2 Introduction to Quasi-Cyclic Low-Density Parity-Check Codes and Low-Density Parity-Check Convolutional Codes</b> | <b>4</b>    |
| 2.1 Low-Density Parity-Check Codes . . . . .  | 4           |
| 2.2 Quasi-Cyclic Low-Density Parity-Check Codes . . . . .   | 7           |
| 2.2.1 Encoding and Decoding . . . . .   | 9           |
| 2.2.2 Properties of QC-LDPC Codes . . . . .   | 10          |
| 2.3 LDPC Convolutional Codes . . . . .  | 11          |
| 2.3.1 Definition . . . . .  | 11          |
| 2.3.2 Decoding Algorithm for LDPC Convolutional Codes . . . . .   | 12          |
| 2.3.3 Previous Algebraic Construction . . . . .   | 16          |



|          |  |           |
|----------|--|-----------|
| 2.3.4    | Properties of LDPC-CC . . . . .  | 18        |
| <b>3</b> | <b>LDPC Convolutional Codes with Rational Parity-Check Matrix</b>                                      | <b>21</b> |
| 3.1      | Tanner Graph with Induced Variable Nodes for Rational Entries in the Parity-<br>Check Matrix . . . . . | 23        |
| 3.2      | Decoding Behavior for the New Tanner Graph . . . . .   | 25        |
| 3.3      | Simulation Results . . . . .   | 28        |
| <b>4</b> | <b>Bit-Flipping for LDPC Convolutional Codes</b>   | <b>35</b> |
| 4.1      | Previous Works for Modified Bit Flipping Algorithms for Decoding LDPC<br>Convolutional Codes . . . . . | 36        |
| 4.2      | Proposed Bit Flipping Algorithms for LDPC Convolutional Codes . . . . .                                | 36        |
| 4.3      | Simulation Results . . . . .   | 38        |
| <b>5</b> | <b>Conclusion</b>  | <b>42</b> |
|          | <b>Bibliography</b>  | <b>43</b> |



# List of Figures

|      |  |    |
|------|--|----|
| 2.1  | Tanner graph of Example 1 . . . . .  | 5  |
| 2.2  | A shift register for encoding a (9,3) quasi-cyclic code. . . . .   | 9  |
| 2.3  | A shift register for encoding LDPC convolutional code.[6] . . . . .  | 13 |
| 2.4  | Tanner graph for LDPC convolutional codes of <b>Example 5</b> [6] . . . . .  | 13 |
| 2.5  | Decoding window for LDPC convolutional codes of <b>Example 5</b> [6] . . . . .   | 14 |
| 2.6  | The Tanner Graphs of the Quasi-cyclic Code and the Convolutional Code[6]   | 18 |
| 3.1  | Performance Comparison between different realizations in example 1. . . . .  | 23 |
| 3.2  | Bit nodes representing for $\mathbf{V}_0(D)$ , $\mathbf{V}_1(D)$ , $\mathbf{V}_2(D)$ , and $\mathbf{M}(D)$ for <b>Example 1</b>    | 25 |
| 3.3  | Subgraph of <b>Example 1</b> with first parity-check equation . . . . .  | 25 |
| 3.4  | Subgraph of <b>Example 1</b> with second parity-check equation . . . . .   | 26 |
| 3.5  | Subgraph of <b>Example 1</b> with relations between $\mathbf{M}(D)$ and $\mathbf{V}_0(D)$ . . . . .                                | 26 |
| 3.6  | Performance Results of example 1. . . . .  | 29 |
| 3.7  | Performance Results of example 2. . . . .  | 30 |
| 3.8  | Performance Results of example 3. . . . .  | 32 |
| 3.9  | Performance Results of example 3. . . . .  | 33 |
| 3.10 | Performance Results of example 4. . . . .  | 34 |
| 4.1  | The Performance Comparison between Gallager Bit Flipping and Modified Algorithms for a (128,3,6) LDPC Convolutional Code . . . . . | 37 |
| 4.2  | Bit-Flipping For LDPC Convolutional Codes in example 1. . . . .  | 39 |
| 4.3  | Bit-Flipping For LDPC Convolutional Codes in example 2. . . . .  | 40 |
| 4.4  | Bit-Flipping For LDPC Convolutional Codes in example 3. . . . .  | 41 |

# Chapter 1

## Introduction

### 1.1 Motivation

Low-density parity-check (LDPC) block codes were invented by Gallager in 1960s [1] but are overlooked for almost 35 years. They have been rediscovered and shown that the random construction of LDPC codes for long code length can approach the Shannon-limit. Generally, the random construction codes outperform algebraically constructed LDPC codes in long code length. However, for medium-length LDPC codes, say a few thousand bits, the situation is different. For these lengths, algebraic construction may have better BER performance than random ones. Besides, the lack of structure of random construction of LDPC codes leads to a serious problem for accessing the large size of parity-check matrix both in encoding and decoding. Therefore algebraic construction of LDPC codes is worthy to be discussed. Quasi-cyclic LDPC (QC-LDPC) codes are one type of algebraic construction of LDPC codes and they can be efficiently encoded by using simple shift registers [2, 7, 8]. After that, the convolutional counterpart of LDPC block codes called LDPC convolutional codes (LDPC-CC) can be constructed from QC-LDPC.

In 1999, LDPC-CC were proposed by Alberto Jiménez Felström and Kamil Sh. Zigangirov [5]. LDPC-CC are convolutional codes with low density of ones in the scalar form of parity-check matrices [6, 7, 10, 11]. They can be encoded efficiently and with arbitrary length by shift registers just like convolutional codes, and there is no modification for the decoder, i.e., a LDPC-CC decoder can decode the code with arbitrary length by modified

iterative message passing decoding algorithms. Furthermore, after an initial delay, the decoded outputs of the decoder are continuously popped out. These features make LDPC-CC more adequate than LDPC block codes in some communication systems such as IEEE 802.11 wireless standards that ethernet frames can vary in size from 64 Bytes to 1518 Bytes.

The recent algebraic construction for LDPC-CC comes from QC-LDPC codes [2, 6, 7]. However there are some restrictions on the parity-check matrices. For those codes, there is zero, monomial, or binomial in each entry in the parity-check matrices. It was shown that the girth of the Tanner graph corresponding to the QC-LDPC codes with trinomial in the parity-check matrices is smaller or equal to six. Because the LDPC-CC constructed from QC-LDPC codes has similar Tanner graph as the mother QC-LDPC codes, the girth property is also apply to them [6]. This result usually leads to poor bit error rate (BER) performance and is also the reason that mostly LDPC-CC consist of only zero, monomial and binomial in the parity-check matrices. But in this thesis, we propose a new perspective of decoding for LDPC-CC with rational parity-check matrices. We induce dummy variable nodes in the Tanner graph to represent the structure of rational and avoid the cycles of length 4. The simulation results show that the BER performance is better for about 1 dB and 0.7 dB improvement in the examples. Besides, we generate rational parity-check matrices equivalent to the monomial parity-check matrices and apply them to the new perspective of decoding. The BER performance is comparable with that of algebraic constructed LDPC-CC.

The decoding for LDPC-CC with any kinds of parity-check matrices in the above paragraphs are based on soft decision decoding. In many applications, the power consumption and the volume of the system are the most important concern. They require the decoders to work in low complexity or quickly. For these reasons, the iterative message passing algorithms for LDPC codes must be simplified. Bit-flipping (BF) algorithms are also originally proposed by Gallager and they are hard decision decoding algorithms and hence can provide decoded result very quickly. There are many modified bit-flipping algorithms for decoding different type of LDPC block codes but few discussions for LDPC-CC. In this thesis, we

proposed a modified BF algorithm for decoding LDPC-CC and compare the results with previous works. The BER performance of our modified BF algorithms is better than that of previous works about 2 dB improvement in many examples.

## 1.2 Organization of Thesis

The organization of this thesis is as follows. In Chapter 2 and 3, a review of LDPC block and convolutional codes is presented, respectively. The discussion of LDPC convolutional codes with rational parity-check matrix is given in Chapter 4. A proposed bit-flipping algorithm is presented in Chapter 5. Simulation results are presented in Chapter 6. Remarks are given in Chapter 7 to conclude this work.



# Chapter 2

## Introduction to Quasi-Cyclic Low-Density Parity-Check Codes and Low-Density Parity-Check Convolutional Codes

In this chapter, we introduce the concept of LDPC Codes and the decoding algorithm for LDPC codes. We also introduce QC-LDPC codes and their construction and properties in this chapter.

### 2.1 Low-Density Parity-Check Codes

**Definition 1** *LDPC codes.*

*An  $(j, k)$  LDPC code is a linear block code defined by the parity-check matrix which has low density of ones, where  $j$  ones in every column and  $k$  ones in every row. In addition, the number of ones in common between any two columns is no greater than 1.*

An LDPC code can be represented by a Tanner graph. A Tanner graph is a bipartite graph in which nodes can be partitioned into two classes, and no edge connects two nodes from the same class. A Tanner graph for an  $(j, k)$  LDPC code of length  $N$  is a bipartite graph such that there are  $N$  nodes, named "bit nodes" or "variable nodes", corresponds to the  $N$  coded bits and there are  $m$  nodes, named "check nodes" or "function nodes", corresponds to the  $m$  parity-check equations (Hence, the parity-check matrix  $H$  is of size

$m \times N$ ). An edge connects to a bit node and a check node if and only if the bit participates the parity-check equation. The neighborhood of a node is a set of nodes that connect to the node. There may be cycles in a Tanner graph and the length of the cycle must be even. The girth of a Tanner graph is the length of the shortest cycle in the graph.

**Example1:** An example of (10, 5) LDPC code.

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

The corresponding Tanner graph:

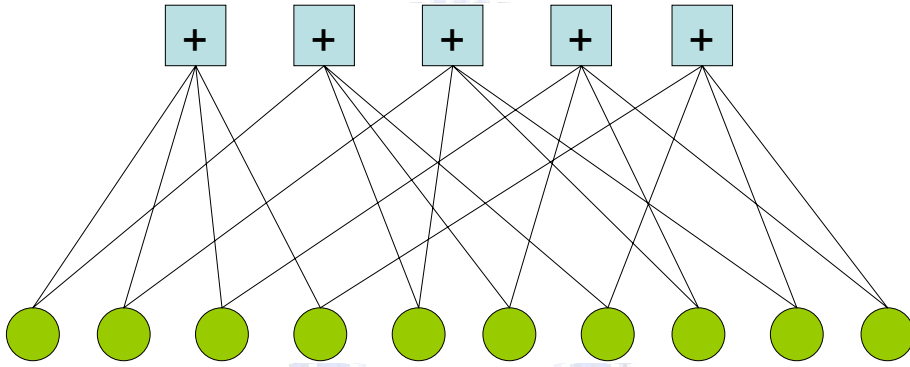


Figure 2.1: Tanner graph of Example 1

To encode LDPC codes, we eliminate the parity-check matrix to a systematic form, then using the shortcut

$$\text{If } \mathbf{H} = \left[ \mathbf{I}_{n-k} \mid \mathbf{P} \right] \text{ then } \mathbf{G} = \left[ \mathbf{B}^T \mid \mathbf{I}_k \right]$$

to get the generator matrix. There is also an efficient approach to get the generator matrix.

The decoding algorithm for LDPC codes would be Sum-Product algorithm (SPA). Both bit nodes and check nodes pass extrinsic information to the neighborhood in each iteration. SPA algorithm can operate both in the probability domain and log-likelihood domain.

Usually we operate SPA in the log-likelihood domain because of numerical stability. It takes five steps to complete the decoding tasks.

Step 1: Initialization. For all  $i$  and  $j$  such that bit  $X_i$  is included in parity check  $f_j$ , i.e., for all  $(i, j)$  such that  $h_{j,i} = 1$  the message from  $X_i$  to  $f_j$  would be:

$$L(q_{i,j}) = L(X_i) = 2y_i/\sigma^2$$

where  $L(q_{i,j})$  is the log-likelihood ratio of the message passed from bit  $X_i$  to the check  $f_j$ ,  $L(X_i)$  is the log-likelihood ratio of bit  $X_i$  and  $2y_i/\sigma^2$  is the log-likelihood ratio of bit  $X_i$  assuming  $X_i$  is priori equally likely to be  $+1$  or  $-1$  and is transmitted over the additive white Gaussian noise (AWGN) channel with variance  $\sigma^2$ .

Step 2: Pass information from check nodes to bit nodes. The message from check node  $f_j$  to bit node  $X_i$  would be:

$$L(r_{j,i}) = \left( \prod_{i' \in R_{j \setminus i}} \alpha_{i',j} \right) \phi \left( \sum_{i' \in R_{j \setminus i}} \phi(\beta_{i',j}) \right)$$

where  $\alpha_{i,j} = \text{sgn}(L(q_{i,j}))$  and  $\beta_{i,j} = |L(q_{i,j})|$  and  $\phi(x) = \log \frac{e^x + 1}{e^x - 1}$

Step 3: Pass information from bit nodes to check nodes. The message from bit node  $X_i$  to check node  $f_j$  would be:

$$L(q_{i,j}) = L(X_i) + \sum_{j' \in C_{i \setminus j}} L(r_{j',i})$$

where  $C_i$  denotes the location of the 1's in column  $i$  of  $H$  and  $C_{i \setminus j} = C_i \setminus \{j\}$ .

Step 4: Compute the log-APP (a posteriori probability) ratios for each bit position  $i$ :

$$L(q_{i,j}) = L(X_i) + \sum_{j \in C} L(r_{j,i})$$

Step 5: Compute the hard decisions and decide if it's time to stop.

$$\hat{X}_i = \begin{cases} +1, & \text{if } L(Q_i) > 0; \\ -1, & \text{otherwise} \end{cases}$$



If all the parity-checks are satisfied or the maximum number of iterations reached, then stop; otherwise, go to step 2.

The SPA algorithm processes the soft information while there is an algorithm for hard decision. Bit-Flipping (BF) algorithm was also proposed by Gallager and is an algorithm that flips the most probable error bits in each iteration. It takes 6 steps to complete the decoding task.

Step 1: Fix a "threshold" parameter  $\delta$ .

Step 2: For parity check  $j$  ( $0 \leq j \leq m - 1$ ), compute the associated syndrome  $S_j$ .

Step 3: If  $S_j = 0$  for all  $j$  or the maximum number of iteration is reached, stop.

Step 4: For each bit position  $i$  ( $0 \leq i \leq n - 1$ ), let  $g_i$  denote the number of non-zero syndromes that include bit  $i$ .

Step 5: Let  $A$  denote the bit positions that participate in more than or equal to  $\delta$  failed parity checks, i.e.  $A = \{i : g_i \geq \delta\}$ .

Step 6: Flip bit  $i$  for all  $i \in A$  and go to Step 2.

## 2.2 Quasi-Cyclic Low-Density Parity-Check Codes

In this section, we introduce an algebraic method to construct an LDPC code which results in a special class of LDPC codes named Quasi-Cyclic Low-Density Parity-Check (QC-LDPC) codes[2, 3].

**Definition 2** *Circulant matrix.*

*A circulant matrix is a special kind of Toeplitz matrix where each row vector is rotated one element to the right relative to the preceding row vector.*

An  $n \times n$  matrix  $\mathbf{C}$  of the form

$$\mathbf{C} = \begin{bmatrix} c_0 & c_{n-1} & \cdots & c_2 & c_1 \\ c_1 & c_0 & c_{n-1} & & c_2 \\ \vdots & c_1 & c_0 & \ddots & \vdots \\ c_{n-2} & & \ddots & \ddots & c_{n-1} \\ c_{n-1} & c_{n-2} & \cdots & c_1 & c_0 \end{bmatrix}$$

is called a circulant matrix.

**Definition 3** *The ring of binary circulant matrices.*

*The set of binary right circulant matrices of size  $r \times r$  forms a ring isomorphic to the ring of polynomials of degree less than  $r$ ,  $\mathbf{F}_2[X]/\langle X^r - 1 \rangle$ : to each circulant matrix  $\mathbf{M}$  we can associate uniquely a polynomial  $M(X)$  with coefficients the entries of the first row of  $\mathbf{M}$ .*

**Definition 4** *Quasi-Cyclic Codes.*

*A linear code  $C$  of length  $n \triangleq r \cdot P$  is called a quasi-cyclic (QC) code with period  $P$  if the right-shift by  $P$  positions of any codeword is again a codeword. Such a code can be represented by a parity-check matrix  $\mathbf{H}$  that consists of circulant matrices of size  $r \times r$ .*

**Definition 5** *Type-I and Type-II QC-LDPC codes.*

*We say that a quasi-cyclic code is of type-I if it is given by a polynomial parity-check matrix  $\mathbf{H}(X)$  with all entries either monomials or zero. We say that a quasi-cyclic code is of type-II if it is given by a polynomial parity-check matrix  $\mathbf{H}(X)$  with all entries either binomials (i.e. sum of two monomials), monomials, or zero.*

We introduce an algebraic method to construct a QC-LDPC codes. Let  $a, b$  be two nonzero element of the set  $\{0, 1, \dots, m-1\}$ , i.e.  $\text{GF}(m)$ , for  $m$  is a prime. Let the  $K$  and  $J$  be the corresponding multiplicative orders of  $a$  and  $b$ , respectively, i.e.,  $o(a) = K$  and  $o(b) = J$ . We form a  $J \times K$  matrix of the form

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_1 & \mathbf{I}_a & \mathbf{I}_{a^2} & \cdots & \mathbf{I}_{a^{K-1}} \\ \mathbf{I}_b & \mathbf{I}_{ab} & \mathbf{I}_{a^2b} & \cdots & \mathbf{I}_{a^{K-1}b} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \mathbf{I}_{b^{J-1}} & \mathbf{I}_{ab^{J-1}} & \mathbf{I}_{a^2b^{J-1}} & \cdots & \mathbf{I}_{a^{K-1}b^{J-1}} \end{bmatrix}$$

where  $\mathbf{I}_x$  is an identity matrix of size  $m \times m$  with each row shifting to the left by  $x-1 \pmod{m}$  positions.

**Example 2:** An example of QC-LDPC code.  $m = 31$ ,  $a = 2$ ,  $b = 5$ .

It is easy to know that  $o(a) = 5$  and  $o(b) = 3$ . Then the parity-check matrix is given by

$$H = \begin{bmatrix} \mathbf{I}_1 & \mathbf{I}_2 & \mathbf{I}_4 & \mathbf{I}_8 & \mathbf{I}_{16} \\ \mathbf{I}_5 & \mathbf{I}_{10} & \mathbf{I}_{20} & \mathbf{I}_9 & \mathbf{I}_{18} \\ \mathbf{I}_{25} & \mathbf{I}_{19} & \mathbf{I}_7 & \mathbf{I}_{14} & \mathbf{I}_{28} \end{bmatrix}_{93 \times 155}$$

From the example, we can know that a QC-LDPC code constructed by this way is a regular LDPC code with common column weight  $J$  and common row weight  $K$ .

### 2.2.1 Encoding and Decoding

To decode QC-LDPC codes, we can use SPA as we treat QC-LDPC codes a class of LDPC codes. But in encoding of QC-LDPC codes, there is an easier way to encode QC-LDPC codes. We can use the nature of quasi-cyclic code to encode with a shift register.

Consider an example of  $(9, 3)$  quasi-cyclic code generated by the following generator matrix:

$$G = \begin{bmatrix} 111 & 100 & 110 \\ 110 & 111 & 100 \\ 100 & 110 & 111 \end{bmatrix}$$

To encode the codeword, we can use a shift register as in Figure 2.2. Let  $(c_0, c_1, c_2)$  be the

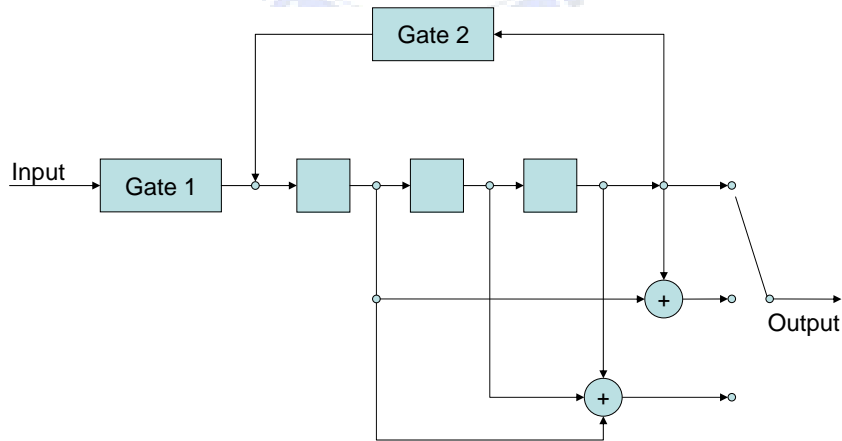


Figure 2.2: A shift register for encoding a  $(9,3)$  quasi-cyclic code.

message to be encoded. As soon as the three information symbols have been shifted into

the register, gate 1 is deactivated and gate 2 is activated. The information symbol  $c_2$  and two parity-check symbols  $p_2^{(1)}$  and  $p_2^{(2)}$  appear at the output terminals and then are shifted into the channel. The two parity-check symbols are given by

$$\begin{aligned} p_2^{(1)} &= c_0 + c_2, \\ p_2^{(2)} &= c_0 + c_1 + c_2. \end{aligned}$$

Next, the register is shifted once. The content of the register is now  $(c_2, c_0, c_1)$ . The information symbol  $c_1$  and two parity-check symbols  $p_1^{(1)}$  and  $p_1^{(2)}$  appear at the output terminals, and they are shifted into the channel. The parity-check symbols  $p_1^{(1)}$  and  $p_1^{(2)}$  are given by

$$\begin{aligned} p_1^{(1)} &= c_1 + c_2, \\ p_1^{(2)} &= c_0 + c_1 + c_2. \end{aligned}$$

At this point, the register is shifted once again. The content of the register is now  $(c_1, c_2, c_0)$ , and the information symbol  $c_0$  and two parity-check symbols  $p_0^{(1)}$  and  $p_0^{(2)}$  appear at the output terminals.

These three symbols are then shifted into the channel. The two parity-check symbols are given by

$$\begin{aligned} p_0^{(1)} &= c_0 + c_1, \\ p_0^{(2)} &= c_0 + c_1 + c_2. \end{aligned}$$

This completes the encoding. The codeword has the form

$$\mathbf{v} = (p_0^{(2)}, p_0^{(1)}, c_0, p_1^{(2)}, p_1^{(1)}, c_1, p_2^{(2)}, p_2^{(1)}, c_2)$$

which consists of three blocks, each of which consists of one unaltered information symbol and two parity-check symbols. This form may also be regarded as a systematic form.

### 2.2.2 Properties of QC-LDPC Codes

In this section, we introduce some important properties of QC-LDPC codes.

**Theorem 1** *Let  $\mathbf{C}$  be a  $(J, K)$ -regular type-I QC-LDPC code with a  $J \times K$  monomial parity-check matrix  $\mathbf{H}(X)$ . Then*

$$d_{min} \leq (J + 1)!$$

**Theorem 2** For any  $J \geq 2$  and  $K \geq 3$   $(J, K)$ -regular type-I QC-LDPC codes, the corresponding graph in this class can not have a girth greater than 12.

**Theorem 3** Let  $\mathbf{C}$  be a  $(J, K)$ -regular QC-LDPC code with trinomial or higher order polynomials parity-check matrix  $\mathbf{H}(X)$ . Then the girth of the corresponding Tanner graph can not greater than six.

The proof of Theorem 1 and Theorem 2 can be found in [3] and [2], respectively. Finally, it is easy to find a cycle of length six in the Tanner graph of the QC-LDPC code with trinomial or higher order polynomials in parity-check matrix  $\mathbf{H}(X)$ , and the proof of Theorem 3 is completed.

## 2.3 LDPC Convolutional Codes

In this chapter, we briefly introduce the concept of LDPC convolutional codes.

### 2.3.1 Definition

A rate  $R = b/c$  regular LDPC convolutional  $(m_s, J, K)$  codes is a code defined by a syndrome former  $\mathbf{H}^T$ , having exactly  $J$  ones in each row, where  $J \ll (c - b)m_s$ , and  $K$  ones in each column.  $m_s$  is the largest degree of the exponent of parity-check matrix.

The syndrome former  $\mathbf{H}^T$  satisfies  $\mathbf{v} \cdot \mathbf{H}^T = \mathbf{0}$ , where  $\mathbf{v}$  is a coded sequence  $\mathbf{v} = (\cdots, \mathbf{v}_0, \mathbf{v}_1, \cdots, \mathbf{v}_t, \cdots)$ ,  $\mathbf{v}_t \in \mathbb{F}_2^c$ . It is of the form:

$$\mathbf{H}^T = \begin{bmatrix} \mathbf{H}_0^T(0) & \cdots & \mathbf{H}_{m_s}^T(m_s) & & \\ & \ddots & & \ddots & \\ & & \mathbf{H}_0^T(t) & \cdots & \mathbf{H}_{m_s}^T(t + m_s) \\ & & & \ddots & \\ & & & & \ddots \end{bmatrix}$$

where  $\mathbf{H}_x^T(t)$  is the submatrix at time  $t$  and is of size  $c \times (c - b)$  and the largest value of  $x$  would be  $m_s$  which is code the memory of the syndrome former.

From above, we can know that the column weight constraint starts from the  $m_s \cdot (c - b)$ th column. Usually, we require  $\mathbf{H}_0^T(t)$  to be full rank for help fast encoding.

**Example: 3** An example of fast encoding for rate  $R = 1/3$  (3, 2, 3) LDPC convolutional codes.

The code is specified by the transpose of parity-check matrix[6]

$$\mathbf{H}^T = \begin{bmatrix} 1 & D^3 \\ D & D^2 \\ D^3 & 1 \end{bmatrix}$$

The transpose of parity-check matrix is a superposition of different time delay matrix.

$$\mathbf{H}^T = \begin{bmatrix} 1 & D^3 \\ D & D^2 \\ D^3 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} + D \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \end{bmatrix} + D^2 \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} + D^3 \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 0 \end{bmatrix}$$

Let  $\mathbf{v}_t = (v_t^{(0)}, v_t^{(1)}, v_t^{(2)})$  represent the three codedbits of time  $t$  in the coded sequence. Then it follows that

$$\mathbf{v}_t \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} + \mathbf{v}_{t-1} \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \end{bmatrix} + \mathbf{v}_{t-2} \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} + \mathbf{v}_{t-3} \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

And we have two equations from the above.

$$\begin{cases} \mathbf{v}_t^{(0)} + \mathbf{v}_{t-1}^{(1)} + \mathbf{v}_{t-3}^{(2)} = 0 \\ \mathbf{v}_t^{(2)} + \mathbf{v}_{t-2}^{(1)} + \mathbf{v}_{t-3}^{(0)} = 0 \end{cases}$$

If we let  $\mathbf{v}_t^{(2)}$  be the information bit  $u_t$ , then the corresponding parity-bits  $\mathbf{v}_t^{(0)}, \mathbf{v}_t^{(1)}$  can be determined by solving the equations. Therefore, we can use a shift-register in Figure 3.1 to solve the equation and encode the pairty-bits. From Example 3, we know that as the time delay zero submatrix is full rank, we can encode the corresponding paritybits by solving the equations.

### 2.3.2 Decoding Algorithm for LDPC Convolutional Codes

We use BP algorithm to decode LDPC convolutional codes. Because the nature of the convolutional code, we can use a decoding window to slide through the Tanner graph corresponding to the LDPC convolutional code and have continuous outputs after an initial delay.

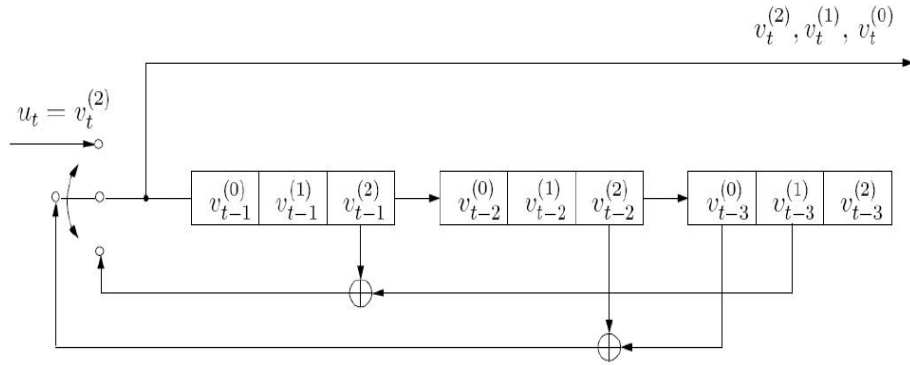


Figure 2.3: A shift register for encoding LDPC convolutional code.[6]

**Example: 4** Decoding window for a LDPC convolutional code.

A rate  $R = b/c = 1/3$  (3, 2, 3) LDPC convolutional code is specified by the parity-check matrix

$$\mathbf{H}^T(D) = \begin{bmatrix} 1 & D^3 \\ D & D^2 \\ D^3 & 1 \end{bmatrix}$$

From the parity-check matrix, we have a corresponding Tanner graph. The black or gray

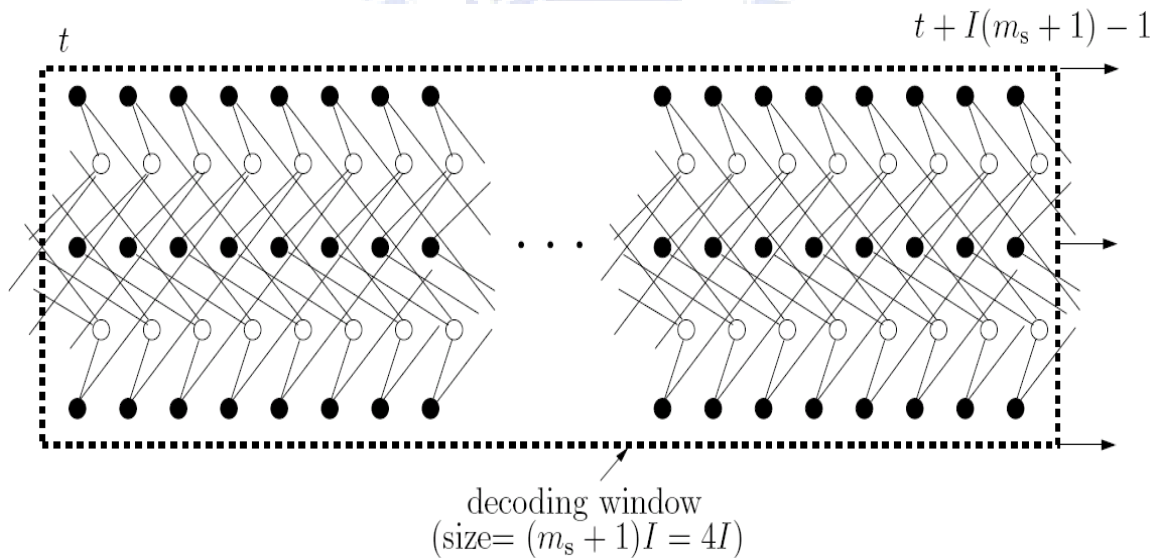


Figure 2.4: Tanner graph for LDPC convolutional codes of **Example 5**[6]

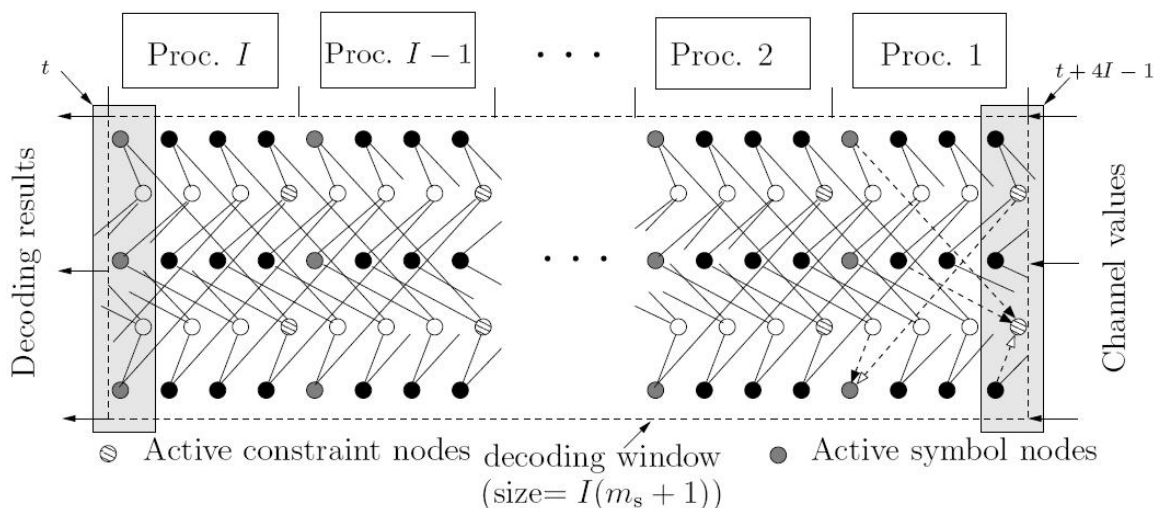


Figure 2.5: Decoding window for LDPC convolutional codes of **Example 5**[6]

circles represent bit nodes while the white circles stand for check nodes. Each time instant the code generates three coded bits, i.e. three bit nodes, and has two parity-checks, i.e. two check nodes. Therefore, at the right most of the Tanner graph there is a rectangle surrounding three bit nodes and two check nodes and that means those five nodes are in one time instant. We can find that the Tanner graph of LDPC convolutional codes is periodic with period one time instant.

The memory of this code is  $m_s = 3$  and that means every bit node whose associated check nodes must all appear in three time instants. This nature gives a decoding window to decode the Tanner graph in pipeline. The decoding window has  $I$  processors corresponding  $I$  maximum iterations and the size of window is  $I(m_s + 1)$  time instants. The Tanner graph flow through the decoding window from the right side, and is popped out from the left side. The nodes popped out are updated  $I$  times and can be decoded after hard decision for the LLR. That means once the nodes are processed by a processor, the LLR of the nodes are updated once. The  $i$ th processor performs  $i$ th iteration of message passing,  $i = 1, 2, \dots, I$ . The influence of one processor is those bit nodes and check nodes in  $(m_s + 1)$  time instants.

Suppose that each edge of the Tanner graph has associated with it a memory element to store the message passed along that edge during iterations. Channel messages are stored



in memory elements associated with the symbol nodes. A processor can access the memory elements along edges connected to constraint nodes under its influence.

Each processor first activates the  $c - b = 2$  check nodes and then  $c = 3$  bit nodes referred to as "active" nodes in Figure 3.3. The check nodes are updated corresponding to those that just enter into the influence of the  $i$ th processor and the bit nodes are those will exit the influence of the  $i$ th processor once updated. When a check node is activated, it reads the memory locations corresponding to the edges which it connects to and updates these locations with new messages. Activating a symbol node leads to exactly the same set of operations. In this case access to the memory locations containing the channel values is also needed to update messages. The message updates are calculated according to the particular message passing algorithm being used, for example in the case of the BP algorithm the update equations are as in Chapter 2.

Consider the part of the Tanner graph under the influence of the first processor in Figure 3.3, i.e., the interval  $[t + 4(I - 1) - 1, t + 4I - 1]$ . The bottom of the two active check nodes at time  $t + 4I - 1$  is connected to three bit nodes, the first at time  $t + 4I - 1$ , the second at time  $t + 4I - 3$ , and the third at time  $t + 4(I - 1)$ . The corresponding edges are shown using dashed lines in Figure 3.3. Consider the situation just before the check nodes at time  $t + 4I - 1$  are activated by processor 1. The edges corresponding to bit nodes in the past (shaded arrows) already contain the channel values (in general the values from the previous iteration). The check node only needs the channel value along the edge (in general the value from previous iteration) corresponding to the symbol node at  $t + 4I - 1$  to be able to update messages along its connecting edges.

The last of the three active bit nodes at  $t + 4(I - 1)$  connects to two check nodes, one at time  $t + 4(I - 1)$ , and the other at time  $t + 4I - 1$ . The corresponding edges are shown using dashed lines. Once more let us look at the situation just before the check nodes at  $t + 4I - 1$  are activated by processor 1. Since the check node at time  $t + 4(I - 1)$  was activated earlier (in fact 3 time units back) this edge (shaded arrow) contains an updated message. The symbol node only lacks the message along the other edge (empty arrow), i.e.,

from the check node three time units away.

Once processor 1 activates the two check nodes at  $t + 4I - 1$  each of them can use the values along the edges to which it connects and calculate the new message for each edge. Thus, all edges connected to the two check nodes now contain updated messages. When the bit nodes are then activated all its edges contain updated messages from the associated check nodes. The symbol nodes can now calculate the new messages to be output along each of the connecting edges and the first iteration is complete for bit nodes at time  $t + 4(I - 1)$ . Observe that each processor can operate independent of the other processors. The  $I$ th processor calculates APP values and outputs the final decoded values for the three active bit under its influence. This process is now repeated as the decoder slides along the Tanner graph.

### 2.3.3 Previous Algebraic Construction

This section describes a method to construct a LDPC convolutional code from a QC-LDPC code. We use the structure of multiplicative groups of integers modulo  $m$  to put circulant matrices into the parity-check matrix so as to form regular quasi-cyclic LDPC codes. We briefly review the procedure to construct a regular quasi-cyclic LDPC code. For a prime  $m$ , the integers  $\{0, 1, 2, \dots, m - 1\}$  forms a field under addition and multiplication modulo  $m$ , i.e. the Galois field  $\text{GF}(m)$ . The non-zero elements of  $\text{GF}(m)$  form a cyclic multiplicative group. Let  $a$  and  $b$  be two non-zero elements with multiplicative order  $o(a) = K$  and  $o(b) = J$  respectively. Then we form the  $J \times K$  matrix  $\mathbf{P}$  of elements from  $\text{GF}(m)$  that has its  $(s, t)$ th entry  $\mathbf{P}_{s,t} = b^s a^t$  as follows:

$$\mathbf{P} = \begin{bmatrix} 1 & a & a^2 & \dots & a^{K-1} \\ b & ab & a^2b & \dots & a^{K-1}b \\ \dots & \dots & \dots & \dots & \dots \\ b^{J-1} & ab^{J-1} & a^2b^{J-1} & \dots & a^{K-1}b^{J-1} \end{bmatrix}$$

where  $0 \leq s \leq J - 1$  and  $0 \leq t \leq K - 1$ .

The quasi-cyclic LDPC code is specified by its parity-check matrix  $\mathbf{H}$ .  $\mathbf{H}$  contains a

$J \times K$  array of circulant submatrix as shown below:

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_1 & \mathbf{I}_a & \mathbf{I}_{a^2} & \cdots & \mathbf{I}_{a^{K-1}} \\ \mathbf{I}_b & \mathbf{I}_{ab} & \mathbf{I}_{a^2b} & \cdots & \mathbf{I}_{a^{K-1}b} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \mathbf{I}_{b^{J-1}} & \mathbf{I}_{ab^{J-1}} & \mathbf{I}_{a^2b^{J-1}} & \cdots & \mathbf{I}_{a^{K-1}b^{J-1}} \end{bmatrix}$$

where  $\mathbf{I}_x$  is an identity matrix of size  $m \times m$  with each row shifting to the left by  $x - 1 \pmod m$  positions.

The LDPC convolutional codes come from the quasi-cyclic LDPC codes. Each circulant matrix can be specified by a unique polynomial. The polynomial represents the entries in the first column of the circulant matrix. For example, a circulant matrix with its first column  $[1 \ 0 \ 0 \ 1 \ 0 \ 1 \ 1]^T$  can be represented by a polynomial  $1 + D^3 + D^5 + D^6$ . Thus the parity-check matrix of the quasi-cyclic LDPC codes constructed by the method above can be expressed in polynomial form (with indeterminate  $D$ ) to obtain the following  $J \times K$  matrix:

$$\mathbf{H}(D) = \begin{bmatrix} D^0 & D^{a-1} & D^{a^2-1} & \cdots & D^{a^{K-1}-1} \\ D^b & D^{ab-1} & D^{a^2b-1} & \cdots & D^{a^{K-1}b-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ D^{b^{J-1}-1} & D^{ab^{J-1}-1} & D^{a^2b^{J-1}-1} & \cdots & D^{a^{K-1}b^{J-1}-1} \end{bmatrix}_{J \times K}$$

**Example 5:** An example of LDPC code constructed by the QC-LDPC code with  $m = 31$ ,  $a = 2$ ,  $b = 5$ .

The parity-check matrix of a QC-LDPC code is given by

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_1 & \mathbf{I}_2 & \mathbf{I}_4 & \mathbf{I}_8 & \mathbf{I}_{16} \\ \mathbf{I}_5 & \mathbf{I}_{10} & \mathbf{I}_{20} & \mathbf{I}_9 & \mathbf{I}_{18} \\ \mathbf{I}_{25} & \mathbf{I}_{19} & \mathbf{I}_7 & \mathbf{I}_{14} & \mathbf{I}_{28} \end{bmatrix}_{93 \times 155}$$

The parity-check matrix of corresponding LDPC convolutional code is as follows:

$$\mathbf{H}(D) = \begin{bmatrix} D^0 & D^1 & D^3 & D^7 & D^{15} \\ D^4 & D^9 & D^{19} & D^8 & D^{17} \\ D^{24} & D^{18} & D^6 & D^{13} & D^{27} \end{bmatrix}_{3 \times 5}$$

### 2.3.4 Properties of LDPC-CC

In this section, we introduce some important properties of LDPC convolutional codes constructed above. To be more specific, the relation between the block and convolutional Tanner graphs, quasi-cyclic block codes viewed as tail-biting convolutional codes, girth and the minimum distance.

#### Relation Between the Block and Convolutional Tanner Graphs

Consider the Tanner graphs of the following two codes[6]:

$$\mathbf{H}^T = \begin{bmatrix} \mathbf{I}_1 & \mathbf{I}_4 \\ \mathbf{I}_2 & \mathbf{I}_3 \\ \mathbf{I}_4 & \mathbf{I}_1 \end{bmatrix}$$

$$\mathbf{H}^T(D) = \begin{bmatrix} 1 & D^3 \\ D & D^2 \\ D^3 & 1 \end{bmatrix}$$

We can observe that the Tanner graph of the convolutional code is very similar to that

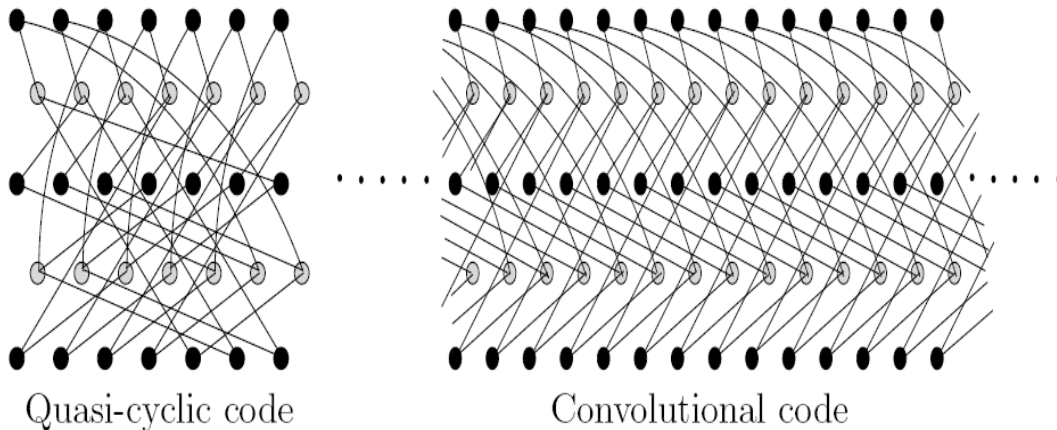


Figure 2.6: The Tanner Graphs of the Quasi-cyclic Code and the Convolutional Code[6]

of the quasi-cyclic code. The Tanner graph of the convolutional code can be viewed as being obtained by unwrapping that of the quasi-cyclic code. If we numerate the time index of the Tanner graph of the quasi-cyclic code as done for the convolutional code, we can

observe that some bit nodes of the Tanner graph of the quasi-cyclic code connecting to some check nodes in past while all that of the convolutional code connecting to the future. In other words, the Tanner graph of the quasi-cyclic code can be viewed as being obtained by truncating the convolutional code into block code with finite length and wrapping the edges associated to the end of the bit nodes back to the check nodes at the beginning.

### Quasi-cyclic Block Codes Viewed as Tail-biting Convolutional Codes

Tail-biting is a technique to convert a convolutional code into a block code without loss of rate. An encoder of the tail-biting convolutional code can be obtained by performing each entry of the generator matrix of the convolutional code modulo  $D^m + 1$  for some  $m$  and replacing a circulant matrix of size  $m \times m$  whose first column vector is specified by the polynomial entry it is going to replace. Since the generator matrix consists of array of circulant matrices only, the obtained block code is a quasi-cyclic code.

**Theorem 4** *Let  $\mathcal{C}$  be a length  $Km$  quasi-cyclic code with the  $Jm \times Km$  parity check matrix  $\mathbf{H}$ , where  $\mathbf{H}$  is composed of  $m \times m$  circulants (i.e., its period is  $K$ ). Let  $\mathbb{C}$  be a convolutional code obtained by unwrapping  $\mathbf{H}$ . Then the quasi-cyclic block code (tail-biting convolutional code)  $\tilde{\mathcal{C}}$  of length  $Km$  constructed from  $\mathbb{C}$  is a sub-code of  $\mathcal{C}[6]$ .*

### Girth

In chapter two, the properties of quasi-cyclic code constructed by the method we described cannot have girth larger than twelve because we always can find a cycle of length twelve in the Tanner graph. The prove is described as follows. We first index the columns and the rows of the parity-check matrix of the quasi-cyclic code, and a path in the parity-check matrix is expressed by a sequence of coordinate of the entries. Then we can find a path of length twelve in which the sum of the column coordinates and that of the row coordinates are both equal to zero and that means the path is a cycle of length twelve. So the girth of the quasi-cyclic code can not greater than twelve.

The girth of the convolutional code obtained by the quasi-cyclic code has girth greater

than or equal to that of the mother quasi-cyclic code. For a cycle in the convolutional code Tanner graph we can find an equivalent cycle in the QC code Tanner graph but not vice versa. For the detail of the proof, see [6].



## Chapter 3

# LDPC Convolutional Codes with Rational Parity-Check Matrix

In chapter 2, a property of regular QC-LDPC codes is that if there were trinomial or higher order polynomials in the parity-check matrix  $\mathbf{H}(X)$  then the girth of the corresponding Tanner graph can not be greater than six. The LDPC convolutional codes constructed by the QC-LDPC code inherit the property, i.e., if there were trinomial or higher order polynomials in the parity-check matrix  $\mathbf{H}(D)$  of the LDPC convolutional code then the girth of the corresponding Tanner graph can not greater than six. This property is usually not welcome for SPA decoders because low girth usually leads to high dependence in message passing and results poor error correcting behavior.

For conventional convolutional codes, we use Viterbi algorithm (VA) as the decoding algorithms and the result is also the maximum likelihood (ML) result. For these convolutional code, the memory is usually under twenty, while the memories of LDPC convolutional codes are usually higher than one hundred. Therefore, trellis decoding is not suitable for LDPC convolutional code; instead, we utilize the feature of the low-density of the scalar form parity-check matrix of the LDPC convolutional code  $\mathbf{H}(D)$ .

In VA decoder, the performances are the same for equivalent parity-check matrix  $\mathbf{H}(D)$  no matter the entries in  $\mathbf{H}(D)$  are monomial, binomial, trinomial, higher order polynomial or rational.

If there are rational entries in  $\mathbf{H}(D)$  of LDPC convolutional codes, there would be a

problem when expanding the  $\mathbf{H}(D)$  into scalar form. One straight way is to expand the rational entries into infinite series entries, which results high density of ones in the scalar form of parity-check matrix. Using SPA decoder, the performance with the resultant Tanner graph is poor due to the high dependence in message passing. Another way is to multiply  $\mathbf{H}(D)$  with the polynomial equal to the least common multiple (l.c.m) of the denominators of rational entries. This may results the parity-check matrix with higher order polynomial in many entries.

**Example 1:** A LDPC convolutional codes with rational parity-check matrix  $\mathbf{H}(D)$  given by

$$\mathbf{H}(D) = \begin{bmatrix} 1 & D & \frac{D^3}{1+D^3} \\ D^3 & D^2 & 1 \end{bmatrix}$$

One way is to expand the rational into infinite series

$$\mathbf{H}(D) = \begin{bmatrix} 1 & D & D^3 + D^6 + D^9 + \dots \\ D^3 & D^2 & 1 \end{bmatrix}$$

We refer this realization as "direct I realization." Another method is to multiply  $\mathbf{H}(D)$  by  $1 + D^3$  and has another equivalent parity-check matrix

$$\mathbf{H}(D) = \begin{bmatrix} 1 + D^3 & D + D^4 & D^3 \\ D^3 & D^2 & 1 \end{bmatrix}$$

We refer this realization as "direct II realization." Using SPA decoder, we can have a performance comparison between different realizations in Figure 3.1. The green line is the performance of VA. The blue is the performance of direct realization II with termination in length 3000. The cyan line is the performance of BPSK transmission. The red line is the performance of direct I realization with termination in length 1000. From the figure, we can observe that the performance of direct I realization is even poorer than that of uncoded BPSK. The performance of direct realization is better than that of uncoded but 1 dB decrement than that of VA at BER  $1e^{-5}$ .



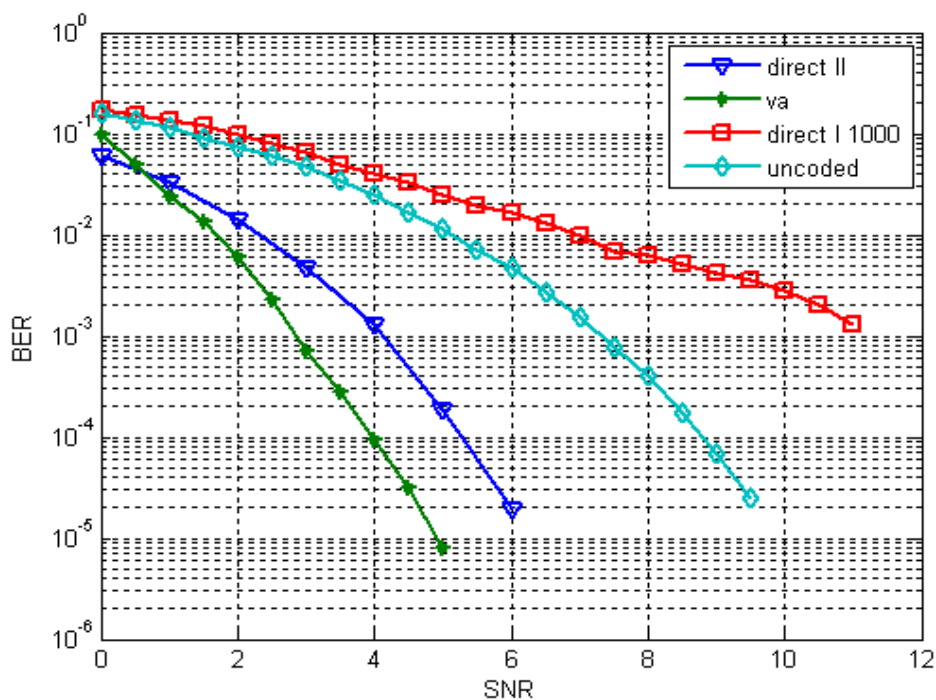


Figure 3.1: Performance Comparison between different realizations in example 1.

### 3.1 Tanner Graph with Induced Variable Nodes for Rational Entries in the Parity-Check Matrix

To solve the problem of high density of ones in scalar form of parity-check matrix in both direct I and II realizations, we propose another realization trying to represent the rational and make the Tanner graph suitable for SPA decoder.

We start from the idea of Tanner graph. There are two classes of nodes named bit nodes and check nodes. There are no edges between the same class of nodes. A set of bit nodes linked by a check node means the values of the set of bit nodes satisfy a parity-check equations, for example, they sum to zero modulo 2 if the values are in binary field. For this concept, Tanner graph is a graph representation describing the relations between bit nodes and bit nodes.

From the example 1, the parity-check matrix describes two parity-check equations for

codeword polynomial. Assume the codeword polynomial is  $\mathbf{V}(D) = (\mathbf{V}_0(D), \mathbf{V}_1(D), \mathbf{V}_2(D))$ . They must satisfy the two parity-check equations.

$$\begin{cases} \mathbf{V}_0(D) \cdot 1 + \mathbf{V}_1(D) \cdot D + \mathbf{V}_2(D) \cdot \frac{D^3}{1+D^3} = 0 \\ \mathbf{V}_0(D) \cdot D^3 + \mathbf{V}_1(D) \cdot D^2 + \mathbf{V}_2(D) \cdot 1 = 0 \end{cases}$$

For these two equations, if we can represent each time instant of each term in the equations by a bit node then we have a graph with each check node linked by three bit nodes. The code rate is  $R = 1/3$ , that is each time instant there are three coded bits, i.e.  $\mathbf{v}_t = (v_t^{(0)}, v_t^{(1)}, v_t^{(2)})$  for time  $t$ . So we use three bit nodes to represent the three coded bit and two check nodes to describe the relations for bit nodes in each time instant. Therefore, in the second parity-check equation, there is a check node linking to a bit node for  $v_{t-3}^{(0)}$ , a bit node for  $v_{t-2}^{(1)}$ , and a bit node for  $v_t^{(2)}$  for time  $t$ . Here would be the problem that how to represent the one time instant of the term  $\mathbf{V}_2(D) \cdot \frac{D^3}{1+D^3}$  by a bit node. We induce a dummy variable  $\mathbf{M}(D)$  with  $\mathbf{M}(D) = \mathbf{V}_2(D) \cdot \frac{D^3}{1+D^3}$ . We also induce a bit node to represent each time instant of  $\mathbf{M}(D)$ . The rest of the work would be find the relations between the induced bit nodes and the bit nodes for  $\mathbf{V}(D)$ . From the equation

$$\mathbf{M}(D) = \mathbf{V}_2(D) \cdot \frac{D^3}{1 + D^3}$$

We multiply the denominator  $1 + D^3$  and put  $\mathbf{M}(D)$  at the left side of the equation and we have

$$\mathbf{M}(D) = \mathbf{V}_2(D) \cdot D^3 + \mathbf{M}(D) \cdot D^3$$

We have an equation for  $\mathbf{M}(D)$  and  $\mathbf{V}_2(D)$  and we can decide the value of  $\mathbf{M}(D)$  once we know  $\mathbf{V}_2(D)$ ; we can decide the value of each time instant of  $\mathbf{M}(D)$  once we know the value of bit nodes for  $\mathbf{V}_2(D)$ . Figure 3.2 illustrates the representation of bit nodes for variable  $\mathbf{V}_0(D)$ ,  $\mathbf{V}_1(D)$ ,  $\mathbf{V}_2(D)$ , and  $\mathbf{M}(D)$ .

It is easy to know that both direct I and II realizations produce cycle of length 4 in the corresponding Tanner graphs, while the proposed method has a girth equal to 8 in the corresponding Tanner graph. Figure 3.3 illustrates the bit nodes and check nodes connection

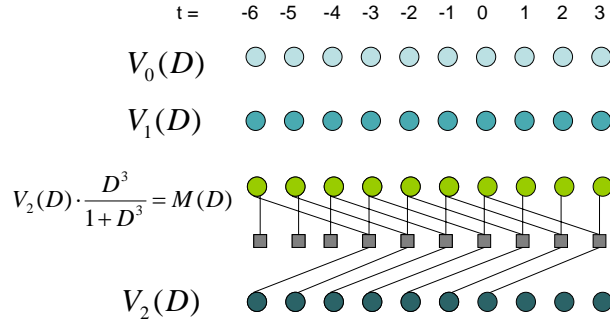


Figure 3.2: Bit nodes representing for  $V_0(D)$ ,  $V_1(D)$ ,  $V_2(D)$ , and  $M(D)$  for **Example 1**

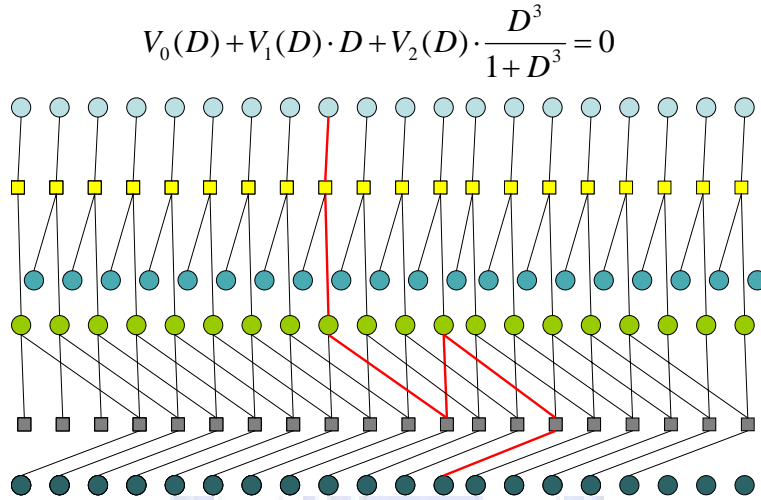


Figure 3.3: Subgraph of **Example 1** with first parity-check equation

constrained by the first parity-check equation, while Figure 3.4 illustrates that by the second parity-check equation. The bold red line indicates one possible cycle of length 8.

### 3.2 Decoding Behavior for the New Tanner Graph

In the process of SPA, bit nodes assign the messages to the check nodes with channel value in first iteration. But we can find out that the bit nodes for each time instant of  $M(D)$  do not have channel values and therefore the outgoing message values are zero. This makes the failure of update in step 2 of decoding for the associated check nodes because the outputs of  $\phi\left(\sum_{i' \in R_{j \setminus i}} \phi(\beta_{i',j})\right)$  equals to zero when there exist at least one  $\beta_{i',j}$  such that  $\beta_{i',j} = 0$ .

$$V_0(D) \cdot D^3 + V_1(D) \cdot D^2 + V_2(D) = 0$$

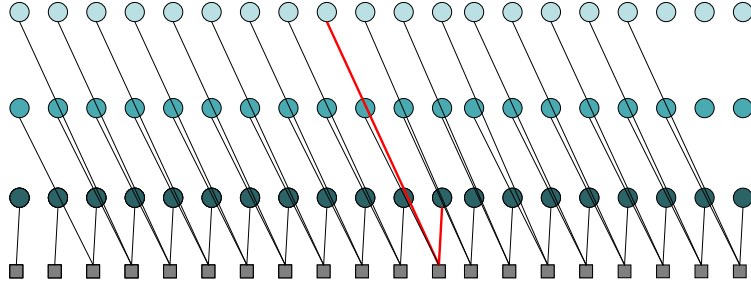


Figure 3.4: Subgraph of **Example 1** with second parity-check equation

When the message a check node sends to a bit nodes is zero, there is no information about the value of the bit node and the bit node can not update.

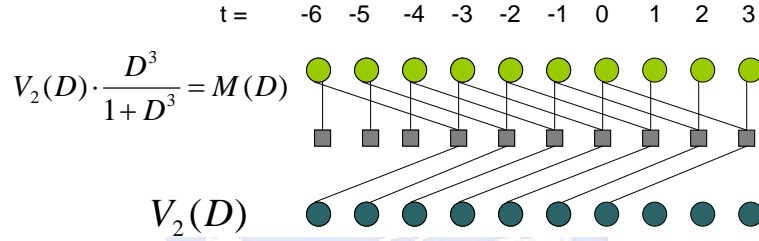


Figure 3.5: Subgraph of **Example 1** with relations between  $\mathbf{M}(D)$  and  $\mathbf{V}_0(D)$

In first iteration, the check nodes in Figure 3.5 are failed due to each check node is linked by two bit nodes for different time instants of  $\mathbf{M}(D)$ . The outgoing message from each check node is zero. Luckily, the check equations

$$\begin{aligned} \mathbf{V}_0(D) \cdot 1 + \mathbf{V}_1(D) \cdot D + \mathbf{V}_2(D) \cdot \frac{D^3}{1+D^3} &= 0 \\ \Rightarrow \mathbf{V}_0(D) \cdot 1 + \mathbf{V}_1(D) \cdot D + \mathbf{M}(D) &= 0 \end{aligned}$$

can help update the bit nodes for each time instant of  $\mathbf{M}(D)$  with extrinsic information. The bit nodes for each time instant of  $\mathbf{M}(D)$  now have nonzero LLR. We say that those bit nodes are recovered. In the next iteration, all the check nodes are no longer failed.

Here comes the question that is there always possible that the bit nodes for dummy variable  $\mathbf{M}(D)$  can be recovered? The answer is not.

**Example: 2** A good BER performance LDPC convolutional code [7]

$$\mathbf{H}(D) = \begin{bmatrix} D & D^{252} & D^{354} & D^{377} & D^{279} \\ D^{20} & D^{409} & D^{344} & D^{383} & D^{107} \\ D^{400} & D^{181} & D^{144} & D^{82} & D^{35} \end{bmatrix}$$

For this parity-check matrix, we try to generate another equivalent code with rational parity-check matrix. For example,

$$\mathbf{H}(D) = \begin{bmatrix} \frac{D}{1+D^{100}} & \frac{D^{252}}{1+D^{100}} & \frac{D^{354}}{1+D^{100}} & \frac{D^{377}}{1+D^{100}} & \frac{D^{279}}{1+D^{100}} \\ D^{20} & D^{409} & D^{344} & D^{383} & D^{107} \\ D^{400} & D^{181} & D^{144} & D^{82} & D^{35} \end{bmatrix}$$

Observing the first row of equivalent parity-check matrix, if we replace  $\mathbf{V}_0(D) \cdot \frac{D}{1+D^{100}}$  by a dummy variable  $\mathbf{M}(D)$ , we have a parity-check equation

$$\mathbf{M}(D) + \mathbf{V}_1(D) \cdot \frac{D^{252}}{1+D^{100}} + \mathbf{V}_2(D) \cdot \frac{D^{354}}{1+D^{100}} + \mathbf{V}_3(D) \cdot \frac{D^{377}}{1+D^{100}} + \mathbf{V}_4(D) \cdot \frac{D^{279}}{1+D^{100}} = 0$$

and another relation equation about  $\mathbf{M}(D)$  and  $\mathbf{V}_0(D)$

$$\mathbf{M}(D) = \mathbf{V}_0(D) \cdot \frac{D}{1+D^{100}}$$

If we multiply the two equations by the denominator,  $1+D^{100}$ , we have

$$\begin{cases} \mathbf{M}(D) \cdot (1+D^{100}) + \mathbf{V}_1(D) \cdot D^{252} + \mathbf{V}_2(D) \cdot D^{354} + \mathbf{V}_3(D) \cdot D^{377} + \mathbf{V}_4(D) \cdot D^{279} & = 0 \\ \mathbf{M}(D) \cdot (1+D^{100}) + \mathbf{V}_0(D) \cdot D & = 0 \end{cases}$$

That means all the check nodes linked to the bit node representing time instant  $t$  of  $\mathbf{M}(D)$  must also connect to the bit node representing time instant  $(t-100)$ . The check nodes can not recover any bit node for different time instant of  $\mathbf{M}(D)$  because the messages are always zero in iterations.

To be free from the failure of recovery, we refer the two bit nodes for time instant  $t$  and  $(t-100)$  of  $\mathbf{M}(D)$  as a super node that stores the LLR of the value equally to the XOR of the values of bit nodes at time instant  $t$  and  $(t-100)$ . By doing so, we have super nodes in different time instant and the super nodes can be updated now.

After super nodes are induced, we can do SPA in decoding now. In the first iteration, the bit nodes for codeword variable  $\mathbf{V}_0(D), \mathbf{V}_1(D), \mathbf{V}_2(D), \mathbf{V}_3(D), \mathbf{V}_4(D)$  pass extrinsic information to the super nodes which then can be updated now. In the first iteration, the outgoing messages from the check nodes linked to the super nodes are all zeros except the message to the super nodes because that is the extrinsic information from the bit nodes for  $\mathbf{V}_0(D), \mathbf{V}_1(D), \mathbf{V}_2(D), \mathbf{V}_3(D), \mathbf{V}_4(D)$ . After first iteration, all the bit nodes in Tanner graph can be updated as usual. The decoding stops when either all parity-check equations are satisfied or the maximum number of iterations is reached.

### 3.3 Simulation Results

In this section, we present the simulation results of the realization for LDPC convolutional codes with rational parity-check matrix.

**Example 1:** A LDPC-CC with small memory.

$$\mathbf{H}(D) = \begin{bmatrix} 1 & D & \frac{D^3}{1+D^3} \\ D^3 & D^2 & 1 \end{bmatrix}$$

From previous method, the rational entry would be expand to infinite series (direct I realization), or the parity-check matrix would be multiplied by  $(1 + D^3)$  (direct II realization).

$$\mathbf{H}(D) = \begin{bmatrix} 1 & D & D^3 + D^6 + D^9 + \dots \\ D^3 & D^2 & 1 \end{bmatrix}$$

$$\mathbf{H}(D) = \begin{bmatrix} 1 + D^3 & D + D^4 & D^3 \\ D^3 & D^2 & 1 \end{bmatrix}$$

With our method, we induce a dummy variable  $\mathbf{M}(D)$  to represent the rational entry, which we may see the codeword polynomial become

$$\mathbf{V}'(D) = \left( \mathbf{V}_0(D) \ , \ \mathbf{V}_1(D) \ , \ \mathbf{V}_2(D) \ , \ \mathbf{M}(D) \right)$$

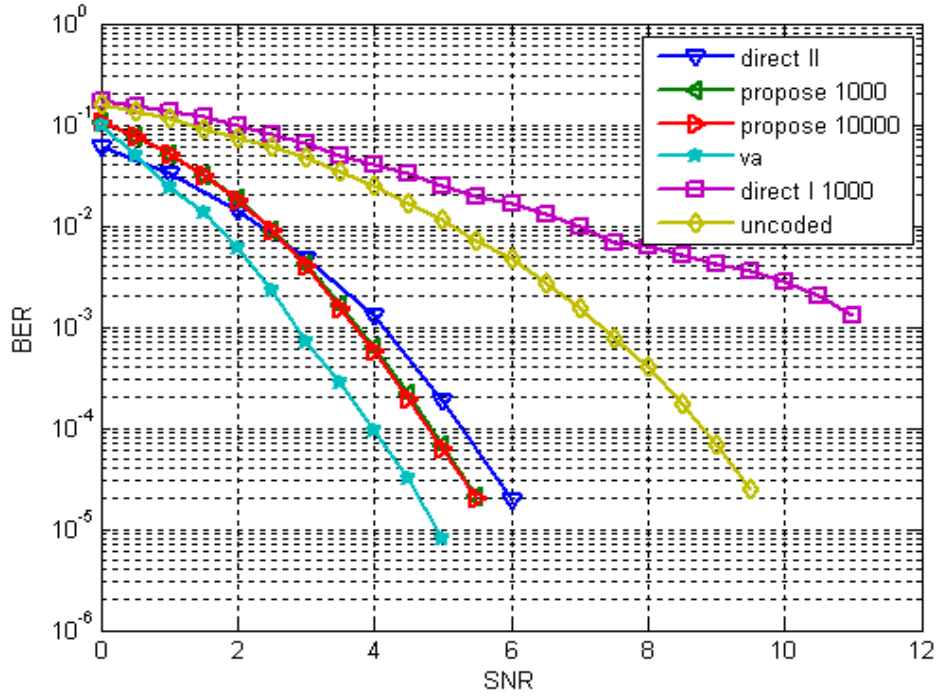


Figure 3.6: Performance Results of example 1.

And the parity-check matrix would become (another realization)

$$\mathbf{H}'(D) = \begin{bmatrix} 1 & D & 0 & 1 \\ D^3 & D^2 & 1 & 0 \\ 0 & 0 & D^3 & 1 + D^3 \end{bmatrix}$$

And we have

$$\mathbf{V}'(D) \cdot \mathbf{H}'^T(D) = \mathbf{0}$$

The performance is in Figure 3.6 with maximum iteration 50 for all simulations. We can observe that the performances of proposed realization with termination length 1000 and 10000 have almost 1dB improvement than that of direct II realization. The performance of direct I realization is even poorer than that of uncoded system.

**Example 2:** A LDPC-CC with large memory.

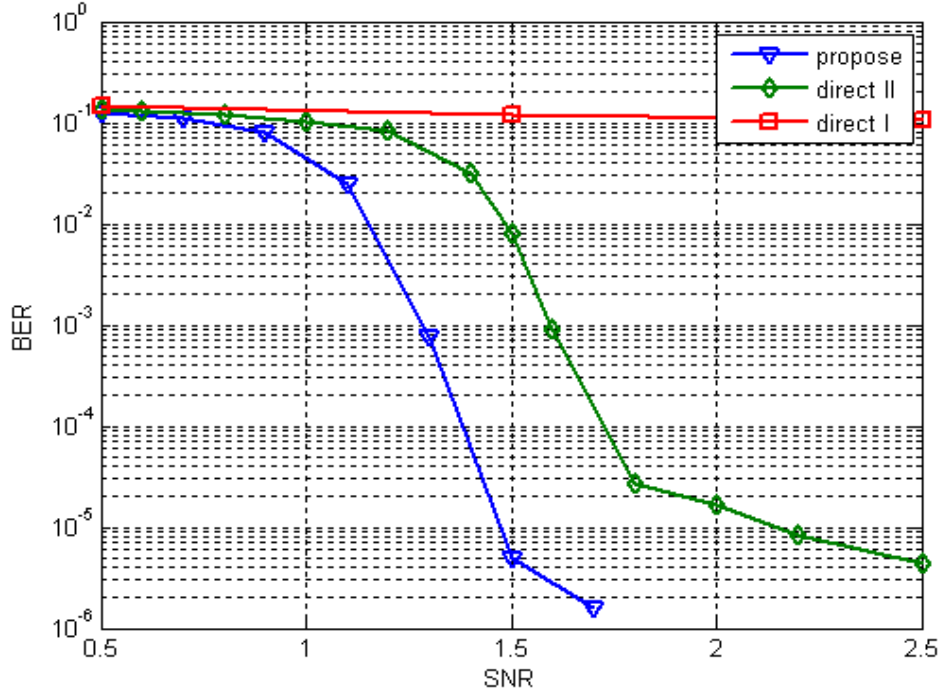


Figure 3.7: Performance Results of example 2.

$$\mathbf{H}(D) = \begin{bmatrix} D & D^{252} & D^{354} & D^{377} & D^{279} \\ D^{20} & D^{409} & D^{344} & D^{383} & D^{107} \\ D^{400} & D^{181} & D^{144} & D^{82} & \frac{D^{35}}{1+D^{138}} \end{bmatrix}$$

The proposed realization

$$\mathbf{H}(D) = \begin{bmatrix} D & D^{252} & D^{354} & D^{377} & D^{279} & 0 \\ D^{20} & D^{409} & D^{344} & D^{383} & D^{107} & 0 \\ D^{400} & D^{181} & D^{144} & D^{82} & 0 & 1 \\ 0 & 0 & 0 & 0 & D^{35} & 1 + D^{138} \end{bmatrix}$$

The direct II realization

$$\mathbf{H}(D) = \begin{bmatrix} D & D^{252} & D^{354} & D^{377} & D^{279} \\ D^{20} & D^{409} & D^{344} & D^{383} & D^{107} \\ D^{400} + D^{538} & D^{181} + D^{319} & D^{144} + D^{282} & D^{82} + D^{220} & D^{35} \end{bmatrix}$$

The performance is in Figure 3.7 with maximum iteration 50 for both simulations. We can observe that the performance of proposed realization has about 0.7 dB improvement



at BER  $1e^{-5}$  than that of direct realization II. This illustrates that a LDPC convolutional code with rational parity-check matrix can work fine by proposed decoding algorithm.

**Example 3:** A LDPC-CC with large memory.

$$\mathbf{H}(D) = \begin{bmatrix} D & D^{252} & D^{354} & D^{377} & D^{279} \\ D^{20} & D^{409} & D^{344} & D^{383} & D^{107} \\ D^{400} & D^{181} & D^{144} & D^{82} & D^{35} \end{bmatrix}$$

In this example, we try to find an equivalent parity-check matrix with rational entries. For example

$$\mathbf{H}(D) = \begin{bmatrix} D & D^{252} & D^{354} & D^{377} & D^{279} \\ D^{20} & D^{409} & D^{344} & D^{383} & D^{107} \\ \frac{D^{400}}{1+D^{100}} & \frac{D^{181}}{1+D^{100}} & \frac{D^{144}}{1+D^{100}} & \frac{D^{82}}{1+D^{100}} & \frac{D^{35}}{1+D^{100}} \end{bmatrix}$$

We replace the  $\mathbf{V}_3(D) \cdot \frac{D^{82}}{1+D^{100}}$  with  $\mathbf{M}(D)$  and have

$$\begin{aligned} \mathbf{H}'(D) &= \begin{bmatrix} D & D^{252} & D^{354} & D^{377} & D^{279} & 0 \\ D^{20} & D^{409} & D^{344} & D^{383} & D^{107} & 0 \\ \frac{D^{400}}{1+D^{100}} & \frac{D^{181}}{1+D^{100}} & \frac{D^{144}}{1+D^{100}} & 0 & \frac{D^{35}}{1+D^{100}} & 1 \\ 0 & 0 & 0 & \frac{D^{82}}{1+D^{100}} & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} D & D^{252} & D^{354} & D^{377} & D^{279} & 0 \\ D^{20} & D^{409} & D^{344} & D^{383} & D^{107} & 0 \\ D^{400} & D^{181} & D^{144} & 0 & D^{35} & 1+D^{100} \\ 0 & 0 & 0 & D^{82} & 0 & 1+D^{100} \end{bmatrix} \end{aligned}$$

with codeword polynomial becomes

$$\mathbf{V}'(D) = \left( \mathbf{V}_0(D) \ , \ \mathbf{V}_1(D) \ , \ \mathbf{V}_2(D) \ , \ \mathbf{V}_3(D) \ , \ \mathbf{V}_4(D) \ , \ \mathbf{M}(D) \right)$$

The performance is in Figure 3.8 with maximum iteration 50 for all simulations. We can observe that there are performance loss for our realization. Therefore, we try another realization with the dummy variable replacing the whole column, i.e.,

$$\mathbf{H}'(D) = \begin{bmatrix} D & D^{252} & D^{354} & 0 & D^{279} & D^{295}(1+D^{100}) \\ D^{20} & D^{409} & D^{344} & 0 & D^{107} & D^{301}(1+D^{100}) \\ D^{400} & D^{181} & D^{144} & 0 & D^{35} & (1+D^{100}) \\ 0 & 0 & 0 & D^{82} & 0 & (1+D^{100}) \end{bmatrix}$$

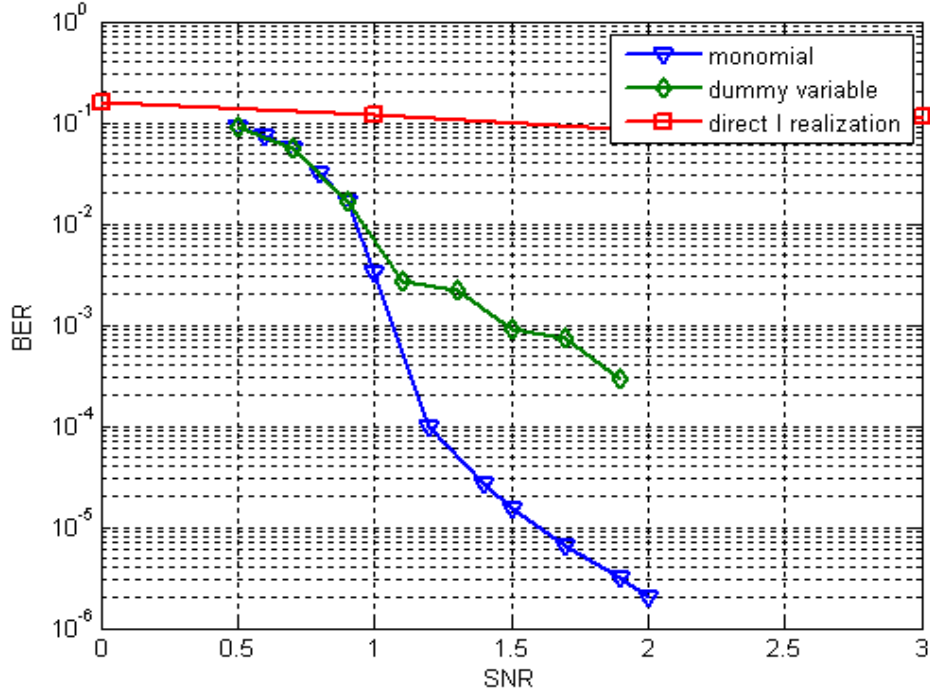


Figure 3.8: Performance Results of example 3.

The performance is in Figure 3.9 with maximum iteration 50 for all simulations. We can see with this realization, the performance of the parity-check matrix is comparable with that of the monomial one.

**Example 4:** A LDPC-CC with large memory [11].

$$\mathbf{H}(D) = \begin{bmatrix} D + D^{144} & 0 & D^{58} + D^{123} & 0 & D^{199} \\ D^{196} & D^{161} & D^{185} & D^{54} & D^{199} \\ 0 & D + D^{144} & 0 & D^{58} + D^{123} & D^{199} \end{bmatrix}$$

This time, we directly use dummy variable  $\mathbf{M}(D)$  to replace the last column and we have

$$\mathbf{H}'(D) = \begin{bmatrix} D + D^{144} & 0 & D^{58} + D^{123} & 0 & 0 & 1 + D^{123} \\ D^{196} & D^{161} & D^{185} & D^{54} & 0 & 1 + D^{123} \\ 0 & D + D^{144} & 0 & D^{58} + D^{123} & 0 & 1 + D^{123} \\ 0 & 0 & 0 & 0 & D^{199} & 1 + D^{123} \end{bmatrix}$$

The performance is in Figure 3.10 with maximum iteration 50 for all simulations. We can

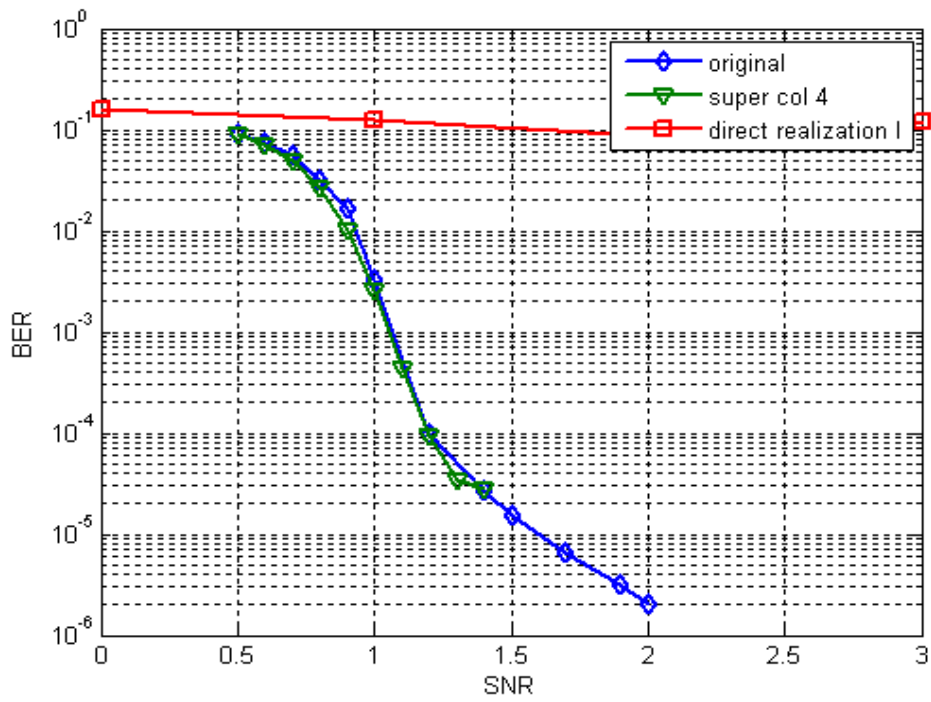


Figure 3.9: Performance Results of example 3.

observe that the performance of the rational parity-check matrix is comparable with that of the monomial one.

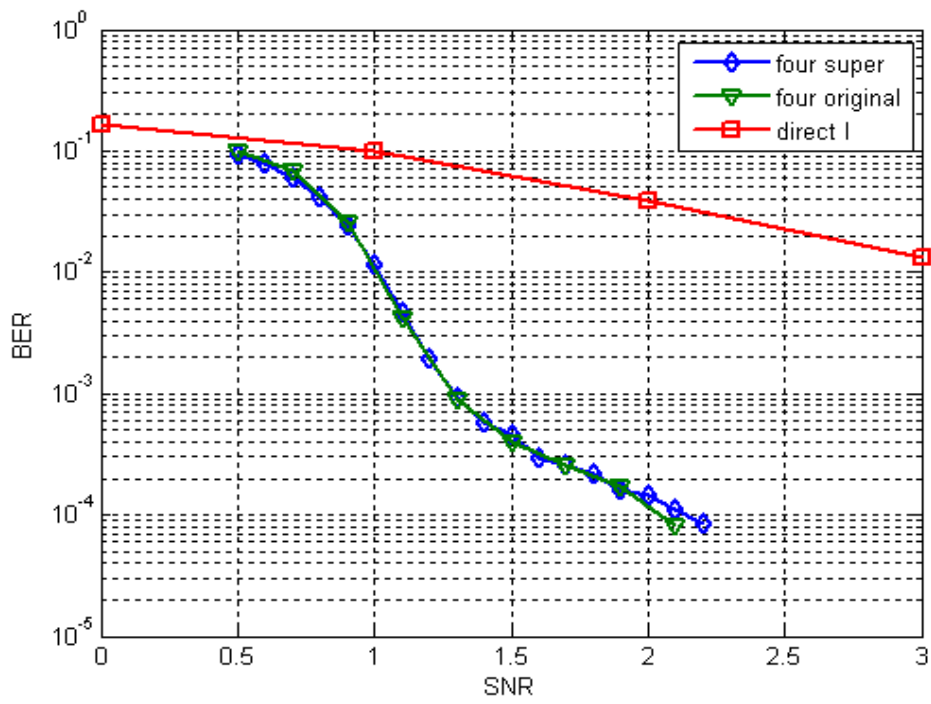


Figure 3.10: Performance Results of example 4.

## Chapter 4

# Bit-Flipping for LDPC Convolutional Codes

The SPA utilizes the soft value of received sequence and updates the LLR of each bit node during iterations. There is bit-flipping (BF) decoding algorithm that deals with the hard value of the received sequence because of the concern of complexity, power consumption, and the implementation of the decoder. In this chapter, we propose a modified bit-flipping decoding algorithm for LDPC convolutional codes, and the BER performance is better than that of previous works.

The decoder for a rate  $R = b/c$  LDPC convolutional code with memory  $m_s$  is equipped with  $I$  processors corresponding the number of maximum iterations. Each processor cares about  $c \cdot (m_s + 1)$  bit nodes and  $b \cdot (m_s + 1)$  check nodes. The  $I$  processors form a decoding window that slide through the Tanner graph of the LDPC convolutional code. After an initial delay, the outputs of the decoding window are continuously popped out. The parallel processor and continuous outputs are the feature of the decoding for LDPC convolutional codes.

There are many modified bit-flipping decoding algorithms for different type of LDPC block codes such as using gradient decent method to decide whether to flip the value of the bit or not in decoding LDPC codes [9]. The modified algorithms performs better than Gallager bit-flipping decoding algorithm does. However, the decoder must collect all the information of bits and compute the objective function to determine the behavior of the

decoder in the next iteration. For this reason, the modified bit-flipping decoding algorithms for LDPC block codes are not suitable for LDPC convolutional codes because each processor of the LDPC convolutional decoder only sees a part of bit nodes, and the processors deal with the information locally. That means the bit-flipping decoding algorithms for LDPC convolutional code must consider the structure of the codes and try to keep the good feature of the decoder.

## 4.1 Previous Works for Modified Bit Flipping Algorithms for Decoding LDPC Convolutional Codes

There is a modified BF algorithm for decoding LDPC convolutional codes [10]. For a column weight  $w_c = 3$  LDPC convolutional code, the algorithm assigns each processor of the LDPC convolutional decoder a threshold with threshold pattern "3-2-3-2-...", i.e., the first processor is assign a threshold by 3, the second one is assign a threshold by 2, the third one is assign a threshold by 3 and so on.

By assigning each processor a threshold, the processor does not need to care about the information of all the bits. The processor counts the number of unsatisfied equations the bit involved and flips the bit if the unsatisfied number is greater than the threshold.

Figure 4.1 illustrates the performance of Gallager bit-flipping decoding algorithm and decoder with threshold pattern. The performance of the modified algorithms has about 3 dB improvement than that of Gallager bit-flipping decoding algorithm at BER  $1e^{-5}$ .

There is some pattern such as "3-3-2-3-3-2-...", "3-3-3-2-3-3-3-2-..." are tested as well, but the performance of the pattern "3-2-3-2-3-2-..." has the most significant improvement.

## 4.2 Proposed Bit Flipping Algorithms for LDPC Convolutional Codes

The previous work assigns each processor a threshold by user-defined threshold pattern before decoding. The processor does not care about the real situation of bits; for example, the

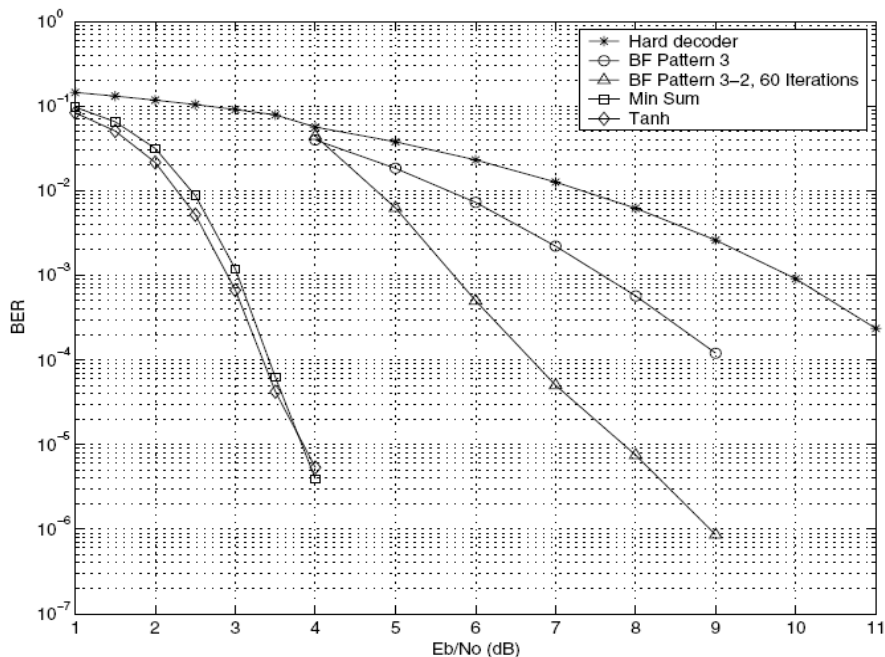


Figure 4.1: The Performance Comparison between Gallager Bit Flipping and Modified Algorithms for a (128,3,6) LDPC Convolutional Code

average number of unsatisfied equations for a bit is decreasing when the SNR is increasing. If the number of unsatisfied equations is one for most bits, the processor can not do anything under this circumstance; besides, the further iterations are wasted because there are no longer bits get flipped.

Instead of predefine a threshold pattern, we want the processors change their threshold over time. We conjecture that the greater the number of unsatisfied equations a bit involved, the more likelihood the error occurs. Besides, when SNR is high, the average number of unsatisfied equations for a bit is small. If the predefined threshold pattern does not consider the case that the maximum number of unsatisfied equations is smaller than any value in the pattern, there would be no bits to be flipped and no update for further iterations. Considering on this case, we propose an idea that makes each processor flip the most probable error bits. Each iteration, we want to flip the bits with greatest number of unsatisfied equations among all bits.

Applying this idea to the LDPC convolutional decoder, there is a problem to solve.

Each processor sees only part of bits, it can not decide whether the number of unsatisfied equations for these bits is the greatest number among all bits or not. Each processor sees only locally maximum number of unsatisfied equations. This difficulty may make each processor flip bits erroneously.

To solve this problem, we let the threshold of processor can be increased if the number of unsatisfied equations for bits in the influence of that processor is larger than the threshold. For each processor, the threshold is initially assign a small value larger than one. When the decoding window slides through the Tanner graph, there are bits leaving the influence of a processor and entering into the influence of another processor at each time instant. The processor see new nodes and compute the syndrome for the bits that is going to leave. If the number of unsatisfied equations for leaving bits is greater than the threshold, those bits are flipped by the processor and the processor increases the threshold to the value equally to the number of unsatisfied equations the leaving bits involved.

The initial value is greater than one because we observe that the bits with one unsatisfied equation are correct in most cases. There is error bit involved in the same parity-check equation and makes the nonzero syndrome. Hence, if we set the threshold equal to one, we may flip the correct bits very often, which results worse correcting behavior.

### 4.3 Simulation Results

In this section, we present the results of proposed bit-flipping decoding algorithms for LDPC convolutional codes. The results are compared with previous works. Our performances have improvements than that of previous works in every case.

**Example 1:** Bit-flipping decoding for a LDPC-CC with parity-check matrix given by

$$\mathbf{H}(D) = \begin{bmatrix} D & D^{252} & D^{354} & D^{377} & D^{279} \\ D^{20} & D^{409} & D^{344} & D^{383} & D^{107} \\ D^{400} & D^{181} & D^{144} & D^{82} & D^{35} \end{bmatrix}$$

The performance results are in Figure 4.2 with maximum iteration 50 for all simulations. The column weight of this code is three. According to the idea of the previous work, we



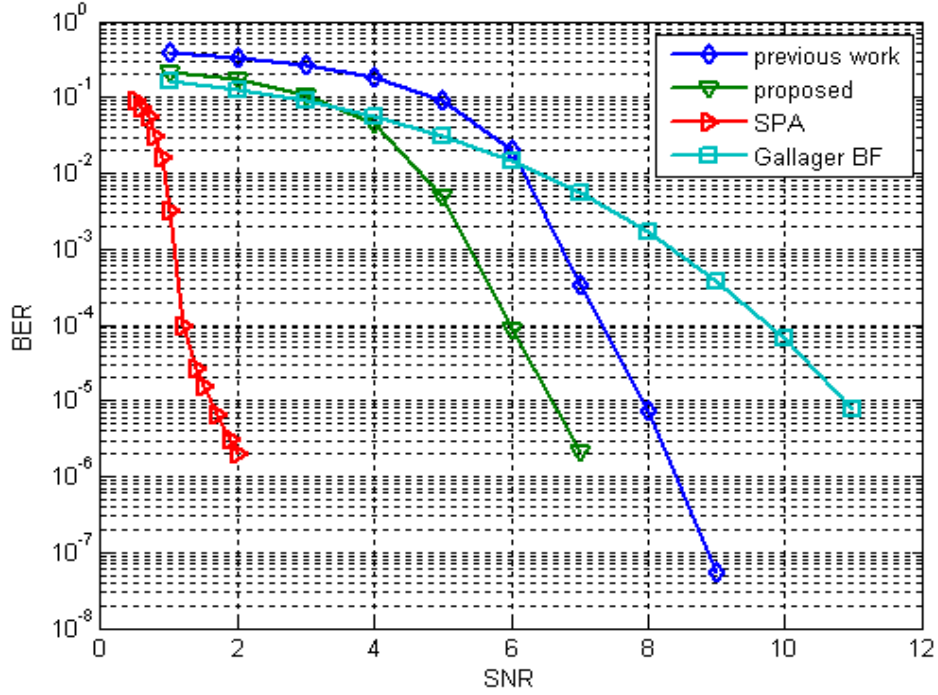


Figure 4.2: Bit-Flipping For LDPC Convolutional Codes in example 1.

define the threshold pattern "3 – 2 – 3 – 2 – ...", and we set the threshold equal to three for Gallager BF algorithm. We have tried to set the threshold equal to two for Gallager BF algorithm, but it works worse than threshold 3 does. We can observe that our proposed algorithm has almost 2 dB improvement than that of previous work and is 4dB improvement than that of Gallager bit-flipping decoding.

**Example 2:** Bit-flipping decoding for a LDPC-CC with parity-check matrix given by

$$\mathbf{H}(D) = \begin{bmatrix} D + D^{144} & 0 & D^{58} + D^{123} & 0 & D^{199} \\ D^{196} & D^{161} & D^{185} & D^{54} & D^{199} \\ 0 & D + D^{144} & 0 & D^{58} + D^{123} & D^{199} \end{bmatrix}$$

The performance results are in Figure 4.3 with maximum iteration 50 for all simulations. The column weight of this code is three. According to the idea of the previous work, we define the threshold pattern "3 – 2 – 3 – 2 – ...", and we set the threshold equal to three for Gallager BF algorithm. We have tried to set the threshold equal to two for Gallager

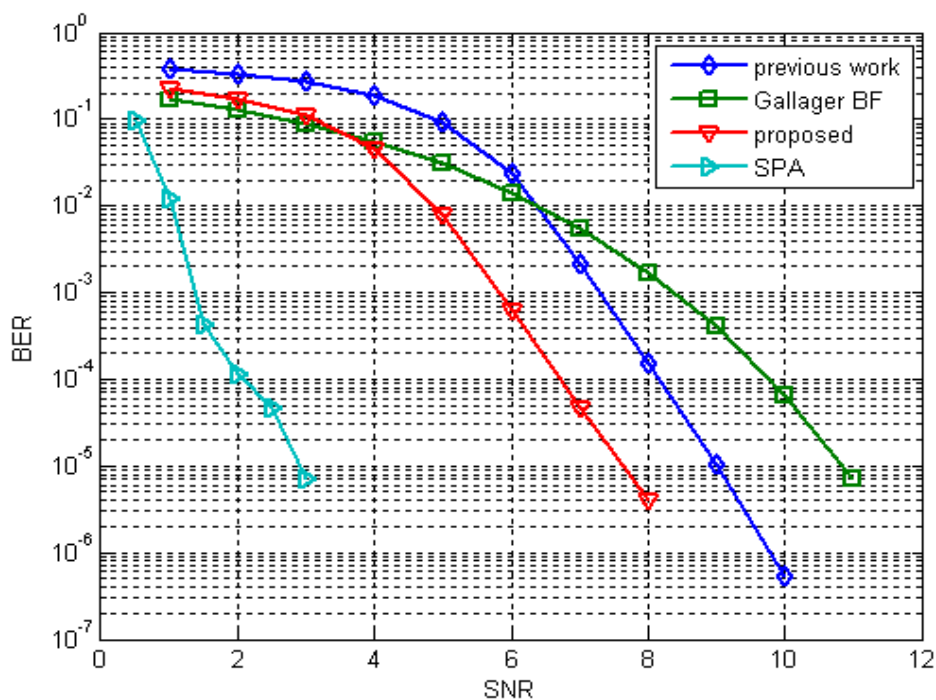


Figure 4.3: Bit-Flipping For LDPC Convolutional Codes in example 2.

BF algorithm, but it works worse than threshold 3 does. We can observe that our proposed algorithm has about 1.5 dB improvement than that of previous work and 3 dB improvement than that of Gallager bit-flipping decoding.

**Example 3:** Bit-flipping decoding for a LDPC-CC with parity-check matrix given by

$$\mathbf{H}(D) = \begin{bmatrix} D + D^2 & 0 & D^4 + D^8 & 0 \\ D^5 & D^9 & D^{10} & D^{20} \\ 0 & D^{25} + D^{19} & 0 & D^7 + D^{14} \end{bmatrix}$$

The performance results are in Figure 4.4 with maximum iteration 50 for all simulations. The column weight of this code is three. According to the idea of the previous work, we define the threshold pattern "3 - 2 - 3 - 2 - ...", and we set the threshold equal to three for Gallager BF algorithm. We have tried to set the threshold equal to two for Gallager BF algorithm, but it works worse than threshold 3 does. We can observe that our proposed algorithm has about 1.7 dB improvement than that of previous work and almost 4 dB than

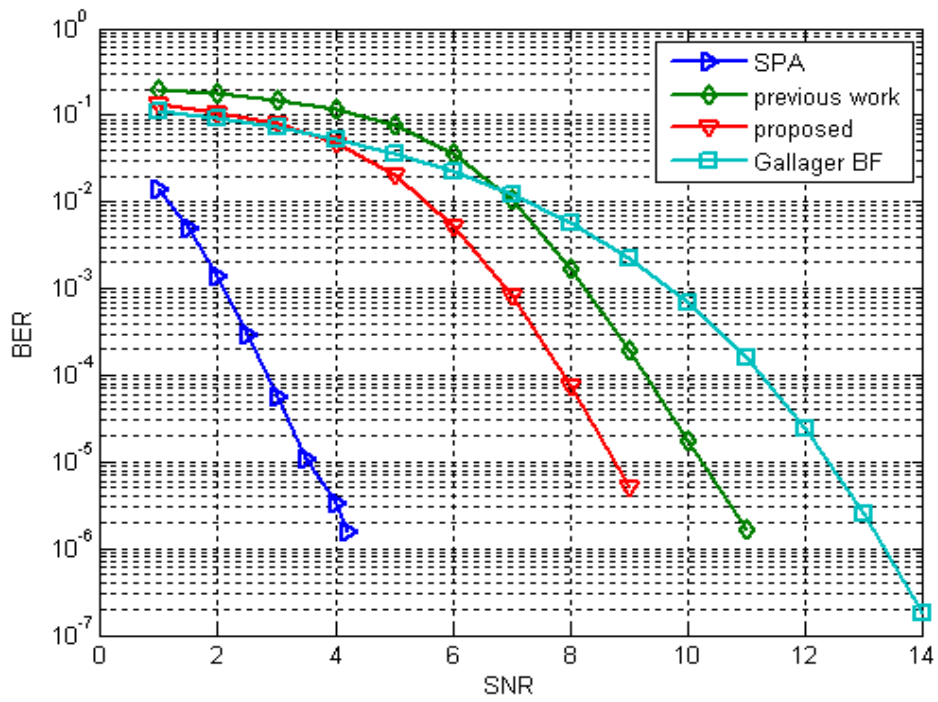
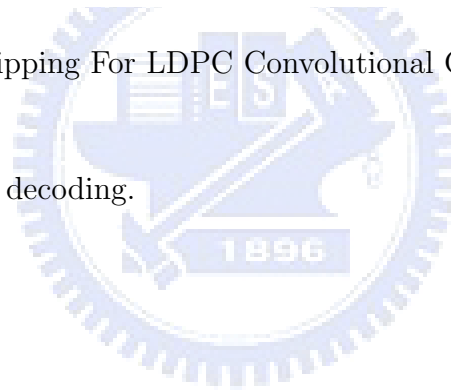


Figure 4.4: Bit-Flipping For LDPC Convolutional Codes in example 3.

that of Gallager bit-flipping decoding.



# Chapter 5

## Conclusion

In traditional, many people construct LDPC convolutional codes from QC-LDPC codes. Therefore, only zero, monomial or binomial parity-check matrix for a LDPC convolutional code is discussed. In ML decoding, the performance of equivalent parity-check matrices with polynomial entries or rational entries is the same. We want to figure out that is there any possibility that the performance of rational parity-check matrix is better than that of the equivalent polynomial parity-check matrix in suboptimal decoding such as SPA.

If we want to use Tanner graph for the graph decoding for decoding LDPC convolutional codes with rational parity-check matrix, there is modification must be done to prevent cycles of length four. In our works, we propose a realization of Tanner graph for rational parity-check matrix and the BER performance is better than that of the parity-check matrix with rational directly expanded into polynomial. We also generate rational parity-check matrices from equivalent monomial ones. The decoding results are comparable after we modified our decoding method.

We also discussed the hard decision decoding algorithms for LDPC convolutional codes. Previous works are done by predefining a threshold pattern for each processor of LDPC convolutional decoder. We proposed an algorithm that can make each processor adaptive to the situations of bits. In decoding process, each processor can update its own threshold without increasing complexity. Besides, the BER performance is better than previous works.

# Bibliography

- [1] R. G. Gallager, “Low-Density Parity-Check Codes,” *IRE Trans. on Inform. Theory*, vol. 8, pp. 21-28, Jan. 1962.
- [2] Marc P. C. Fossorier, “Quasi-Cyclic Low-Density Parity-Check Codes From Circulant Permutation Matrices,” *IEEE Trans. on Inform. Theory*, vol. 50, NO. 8, Aug. 2004.
- [3] R. Smarandache and P. O. Vontobel, “On Regular Quasi-Cyclic LDPC Codes from Binomials,” in *Proc. IEEE Intern. Symp. on Inform. Theory*, Chicago, IL, USA, June 27 - July 2 2004, p. 274.
- [4] S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications, 2nd edition*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 2004.
- [5] Alberto Jiménez Felström and Kamil Sh. Zigangirov, “Time-Varying Periodic Convolutional Codes With Low-Density Parity-Check Matrix,” *IEEE Trans. on Inform. Theory*, vol. 45, NO. 6, Sep. 1999.
- [6] A. Sridharan, “Design and Analysis of LDPC Convolutional Codes,” Ph.D. Dissertation, University of Notre Dame, Notre Dame, Indiana, 2005.
- [7] R. M. Tanner, D. Sridhara, A. Sridharan, T. E. Fuja, and D. J. Costello, “LDPC Block and Convolutional Codes Based on Circulant Matrices,” *IEEE Trans. on Inform. Theory*, vol. 50, no. 12, pp. 2966-2984, Dec. 2004.
- [8] D. J. C. MacKay and M. C. Davey, “Evaluation of Gallager codes for short block length and high rate applications,” in *Codes, Systems and Graphical Models*; volume 123 of

IMA Volumes in Mathematics and its applications;, B. Marcus and J. Rosenthal, Eds.  
New York: Springer- Verlag, 2000, pp. 113-130.

- [9] T. Wadayama, K. Nakamura, M. Yagita, Y. Funahashi, S. Usami and I. Takumi, “Gradient Decent Bit Flipping Algorithms for Decoding LDPC codes,” arXiv:0711.0261v2, Apr. 2008.
- [10] Xin Sheng Zhou, Bruce F. Cockburn and Stephen Bates, “Improved Iterative Bit Flipping Decoding Algorithms for LDPC Convolutional Codes,” *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM'07)*, Victoria, BC, Canada, August 22-24, 2007, pp. 541-544.
- [11] Chi-Jen Wu, Yi-Chun Chou, Chung-Hsuan Wang and Chi-chao Chao, “New Construction of LDPC Convolutional Codes,” in *Proc. 2008 IEEE Int. Symp. on Inform. Theory*, Toronto, ON, Canada, Jul. 2008, pp. 1040-1044.

