

國立交通大學

電信工程學系

碩士論文

低密度校驗碼之信度傳播排程及其性能分析

Belief Propagation Based Decoding Schedules for  
Low-Density Parity Check Codes and their  
Behavior Analysis

研究生：王詩堯

指導教授：蘇育德教授

中華民國 96 年 7 月

# 低密度校驗碼之信度傳播排程及其性能分析

Belief Propagation Based Decoding Schedules for Low-Density

Parity Check Codes and their Behavior Analysis

研究生：王詩堯

Student: Shihyao Wang

指導教授：蘇育德 教授

Advisor: Prof. Yu T. Su

國立交通大學

電信工程學系碩士班

碩士論文

A Thesis

Submitted to Department of Communication Engineering

Collage of Electrical and Computer Engineering

National Chiao Tung University

in Partial Fulfillment of Requirements

for the Degree of

Master of Science

in Communication Engineering

July 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年七月

# 低密度校驗碼之信度傳播排程及其性能分析

學生：王詩堯

指導教授：蘇育德 教授

國立交通大學

電信工程學系碩士班

## 中文摘要

低密度校驗碼最常用的解碼方式是信度傳播演算法，它利用遞迴的方式達到近似最佳的效能，但缺點是需要大量的遞迴次數，而且，硬體的實現也是一大困難。信度傳播演算法很適合採用完全平行的處理，但若要達到此目的，則需要大量的記憶體、處理器及複雜的連接繞線網路，目前的技術尚無法解決此問題，硬體實作仍須採取分群來進行串連式處理。因此，學術上提出了兩種新的演算法去增加收斂的速度以及減少記憶體的使用，一為垂直式重組信度傳播演算法，另一為水平式重組信度傳播演算法。然而，這兩種演算法並未考慮低密度校驗碼本身既有的迴圈效應，為了解決此問題，我們提出一套新的排程，盡可能地降低迴圈的影響。根據模擬的結果，在 IEEE 802.11n 這套標準下，此排程的確讓錯誤率減少了 0.5 ~ 0.2 dB，其優點在遞迴次數少的情況下更加顯而易見。此外，我們亦提出了一套分析解碼演算法收斂行為的方法，並將分析結果與電腦模擬的解碼效能做比較，進而利用這套方法來解釋不同的解碼排程及不同碼的收斂行為與框錯誤率性能表現。

# Belief Propagation Based Decoding Schedules for Low-Density Parity Check Codes and their Behavior Analysis

Student: Shihyao Wang

Advisor: Yu T. Su

Department of Communications Engineering  
National Chiao Tung University

## Abstract

Low density parity check (LDPC) codes form a class of sparse graph codes that offer powerful error-correcting capability. For decoding LDPC codes, the belief propagation (BP) or sum-product algorithm (SPA) is usually used. However, implementation of a BP-based LDPC decoder often requires large memory space and very high degree of parallelism using many component soft-in soft-output (SISO) decoding units and complex interconnect network to perform message passing from variable nodes to check nodes or vice versa. An obvious solution for a relatively long code is to divide a decoding iteration into several serial sub-iterations in which a sub-iteration performs only part of the complete parallel message-passing operation.

There are several architectures and decoding schedules for serial implementation of LDPC decoders. We are interested in two of the more popular ones, namely, the horizontal shuffled BP (HSBP) and the vertical shuffled BP (VSBP) algorithms. It has been shown that, with a proper architecture and scheduling, such parallel-serial decoding methods not only require less storage space but also give faster convergence and, sometimes, better performance. A shortcoming of these existing schedules is that they are more concerned with reduction of memory and interconnect complexities and do not consider the short cycle effect which is the dominant factor limiting a BP-based algorithm's bit error rate (BER) performance.

We propose a simple scheduling approach for reducing the short-cycle effect in a BP-based decoding algorithm. The impact of a short cycle can be lessened if one alternates the decoding schedule so that the cycle length can be effectively extended. The HSBP and VSBP algorithms partition the check or variable nodes into several groups where a group consists of (almost) the same number of consecutive nodes according to the natural order of the parity-check matrix and carry out the BP process group-by-group. Our algorithm groups the check nodes according to the number of short cycles a node is involved. Message-passing to a group with more short cycles is given lower priority in our decoding schedule. The resulting decoding algorithm is referred to as the *cycle-based* HSBP (or VSHP) algorithm.

Another major contribution of this thesis is the development of a convenient and effective method to explain and predict both the performance and the convergence behavior of a candidate LDPC code and the corresponding decoding algorithm/schedule.



We define a function called *message profile* that describes the composition of the extrinsic information associated with a bit node. This function measures how much each bit node has contributed to the extrinsic information of a bit from the beginning of the decoding process up to the end of a given iteration. The normalized correlation spread (NCS) of a bit, defined as the mean-to-root-mean-squared ratio of its message profile, is then used to evaluate the degree of local flooding uniformity of a bit node in a particular iteration. The NCS of a bit node usually converge in just a few iterations and more importantly, all NCS' converge to the same steady-state value though not uniformly. Numerical results indicate that a BP-based algorithm converges when the NCSs of all bits converge, i.e., the convergence of the message-profile functions is consistent with convergence of the decoder. Furthermore, it is found that the decoder performance is directly related to the common steady state value—the larger it is, the better the BER performance becomes.



## 致謝

首先，要感謝我的指導教授—蘇育德老師，在我研究所的求學過程中悉心地指導與教誨，並且適時的給予實用的建議與鼓勵。從老師的身上，我看到了一位真正的學者對於研究的認真態度與無比熱忱，這是讓我由衷地欽佩和感動的。在老師的指導下，讓我初窺到學術殿堂的門徑，也讓我學習到做研究應有的態度與方法，實在是獲益良多。此外，我也要感謝電子工程學系張錫嘉教授的指導，讓我對通訊的領域有更深一層的瞭解。

感謝實驗室的所有夥伴們，這兩年來的朝夕相處。謝謝延修、淵斌、昌明、彥志、人仰學長在學術研究上的協助與指導；謝謝士瑋學長在我心情煩躁時適時給予溫暖；謝謝琬瑜、泰翔、和炳全，陪我一起度過充滿著酸甜苦辣的研究生活；謝謝汀華、千雅、坤昌、佐翰、郁文、易霖等學姊弟妹們的熱情，讓我在這裡總是感到歡樂，不會孤單。



最後，更要感謝我的父母與其他親人好友們，謝謝你們永遠給我無限的支持與鼓勵，讓我有信心有勇氣去面對各種挑戰。因為有你們，是讓我不斷向前邁進的原動力！

謹將此論文獻給所有愛我與我愛的人

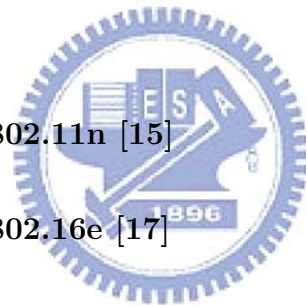
2007年7月 新竹交大

# Contents

<b>Chinese Abstract</b>	<b>i</b>
<b>English Abstract</b>	<b>ii</b>
<b>Acknowledgements</b>	<b>iv</b>
<b>Contents</b>	<b>v</b>
<b>List of Figures</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Low-Density Parity Check Codes</b>	<b>5</b>
2.1 Factor Graphs . . . . .	5
2.2 The Belief Propagation Algorithm . . . . .	7
<b>3 Shuffled Iterative Decoding of LDPC Codes</b>	<b>9</b>
3.1 Multi-stage Factor Graphs of LDPC codes . . . . .	9
3.2 Shuffled Iterative Decoding Algorithms . . . . .	10
3.2.1 Vertical Shuffled Belief Propagation Algorithm . . . . .	11
3.2.2 Horizontal Shuffled Belief Propagation Algorithm [11] . . . . .	12
<b>4 A Novel Scheduling Method for the HSBP Algorithm</b>	<b>15</b>
4.1 A Decoding Schedule for Reducing Short Cycle Effects . . . . .	15
4.2 Searching for Length-4 Cycles . . . . .	17



<b>5</b>	<b>Message Flow Distribution and Decoder Behaviors</b>	<b>21</b>
5.1	Profile of the Conventional BP Algorithm . . . . .	21
5.2	Profile of the VSBP Algorithm . . . . .	23
5.3	Profile for the HSBP Algorithm . . . . .	24
<b>6</b>	<b>Numerical Results and Discussion</b>	<b>28</b>
6.1	Error Rate Performance Comparison . . . . .	28
6.2	Profile Comparison . . . . .	31
6.3	Comparison between LDPC Codes in 802.11n and 802.16e . . . . .	39
<b>7</b>	<b>Conclusions</b>	<b>49</b>
	<b>Bibliography</b>	<b>49</b>
	<b>Appendix</b>	<b>51</b>
<b>A</b>	<b>Specification of IEEE 802.11n [15]</b>	<b>52</b>
<b>B</b>	<b>Specification of IEEE 802.16e [17]</b>	<b>54</b>

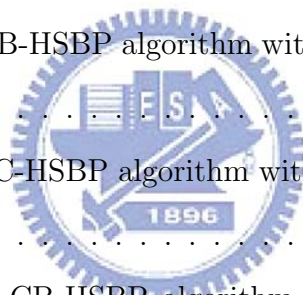


# List of Figures

2.1	Two classes of factor nodes. . . . .	6
2.2	A typical factor graph. . . . .	6
2.3	A factor graph of a (7,3) code. . . . .	7
3.1	A MSFG that describes a conventional BP decoding (schedule) of a (7,3) code. . . . .	10
3.2	The MSFG representation of a VSBP decoding schedule of a (7,3) code. . . . .	12
3.3	A MSFG that describes the HSBP decoding schedule of a (7,3) code. The dash lines represent the (message) flooding that are by-passed. . . . .	14
4.1	Flow chart of the proposed searching algorithm that computes the number of length- $2k$ cycles. . . . .	18
4.2	MSFG representations of a cycle-factor based (a) and conventional (b) HSBP algorithms for an LDPC code with $G=2$ . . . . .	20
5.1	Message flow of bit node $V_1$ at the first iteration. . . . .	27
6.1	Package error rate performance of the (240, 120) QC-LDPC code with $I_{max} = 5, 10, G=2$ and different decoding schedules. . . . .	29
6.2	Package error rate performance the (480, 240) QC-LDPC code with $I_{max} = 5, 10$ and $G=2$ . . . . .	30
6.3	Package error rate performance of the (1080, 540) QC-LDPC code with $I_{max} = 5, 10$ and $G=2$ . . . . .	30

6.4	Package error rate performance of the (480,240) QC-LDPC code with $I_{max} = 5, 10$ and $G=4$ . . . . .	31
6.5	Package error rate performance of the (1080, 480) QC-LDPC code with $I_{max} = 5, 10$ and $G=4$ . . . . .	32
6.6	Profile NCR behavior of the C-BP schedule for the (480,240) QC-LDPC code. . . . .	33
6.7	Profile NCR behavior of the C-HSBP algorithm with $G=2$ in decoding the (480,240) QC-LDPC code. . . . .	33
6.8	Profile NCR behavior of the CB-HSBP algorithm with $G=2$ in decoding the (480,240) QC-LDPC code. . . . .	34
6.9	Profile NCR behavior of the C-VSBP algorithm with $G=2$ for the (480,240) QC-LDPC code. . . . .	34
6.10	Profile mean of the C-BP algorithm in decoding the (480,240) QC-LDPC code. . . . .	35
6.11	Profile mean of the C-HSBP algorithm with $G=2$ in decoding the (480,240) QC-LDPC code. . . . .	35
6.12	Profile mean of the CB-HSBP schedule with $G=2$ in decoding the (480,240) QC-LDPC code. . . . .	36
6.13	Profile spread (standard deviation) of the C-BP algorithm in decoding the (480,240) QC-LDPC code. . . . .	36
6.14	Profile spread (standard deviation) of the C-HSBP algorithm with $G=2$ in decoding the (480,240) QC-LDPC code. . . . .	37
6.15	Profile spread (standard deviation) behavior of the CB-HSBP algorithm with $G=2$ for the (480,240) QC-LDPC code. . . . .	37
6.16	Profile mean and spread behaviors of the C-HSBP algorithm with $G=2$ in decoding the (480,240) QC-LDPC code. . . . .	38

6.17	Profile mean and spread behaviors of the C-HSBP algorithm with $G=2$ in decoding the (480,240) QC-LDPC code. . . . .	38
6.18	Profile mean and spread behaviors of the CB-HSBP algorithm with $G=2$ for the (480,240) QC-LDPC code. . . . .	39
6.19	Profile NCR behavior of the C-HSBP algorithm with $G=4$ for the (480,240) QC-LDPC code. . . . .	40
6.20	Profile NCR behavior of the CB-HSBP algorithm with $G=4$ for the (480, 240) QC-LDPC code. . . . .	40
6.21	Profile NCR behavior of the C-VSBP algorithm with $G=4$ for the (480, 240) QC-LDPC code. . . . .	41
6.22	Profile mean of the C-HSBP algorithm with $G=4$ for decoding the (480,240) QC-LDPC code. . . . .	41
6.23	Profile mean of the CB-HSBP algorithm with $G=4$ for the (480, 240) QC- LDPC code. . . . .	42
6.24	Profile spread of the C-HSBP algorithm with $G=4$ for the (480, 240) QC- LDPC code. . . . .	42
6.25	Profile spread of the CB-HSBP algorithm with $G=4$ for the (480, 240) QC-LDPC code. . . . .	43
6.26	Profile mean and spread behaviors of the C-HSBP algorithm with $G=4$ for the (480, 240) QC-LDPC code. . . . .	43
6.27	Profile mean and spread behaviors of the CB-HSBP algorithm with $G=4$ for the (480, 240) QC-LDPC code. . . . .	44
6.28	Profile NCR behavior of the code derived from the base model matrix of IEEE 802.11n. . . . .	45
6.29	Profile NCR behavior of the code derived from the base model matrix of IEEE 802.16e. . . . .	45



6.30	The PER performance of the (480,240) QC-LDPC code with the C-BP decoder. . . . .	46
6.31	Profile NCR behavior of the BP algorithm for the (480, 240) QC-LDPC code in 802.16e. . . . .	46
6.32	Profile mean of the BP algorithm for the (480, 240) QC-LDPC code in 802.16e. . . . .	47
6.33	Profile spread of the BP algorithm for the (480, 240) QC-LDPC code in 802.16e. . . . .	47
6.34	Profile mean and spread behaviors of the conventional BP algorithm for the (480, 240) QC-LDPC code in 802.16e. . . . .	48





# Chapter 1

## Introduction

Belief propagation (BP) [5] is an efficient iterative algorithm for computing marginals of multivariate functions on a graphical model most commonly used in communication theory and artificial intelligence. The algorithm is supposed to solve inference problems that require the evaluation of some marginal (density) functions and/or find their extremes. Pearl [1] formulated this algorithm on trees in 1982, and Kim and Pearl (in 1983) [2] on polytrees. Pearl [3] has suggested this algorithm as an approximation for general (loopy) networks. Since then it had been shown that many known algorithms like the forward-backward algorithm, the Viterbi algorithm, Kalman filter, Gallager's algorithm for decoding low density parity check (LDPC) codes, etc., are all special instances of the BP algorithm (now also known as the sum-product algorithm).

The BP algorithm is an efficient inference algorithm on trees and has demonstrated empirical success in numerous applications that involve loopy networks including LDPC codes, turbo codes, free energy approximation, and satisfiability. It is commonly used in pairwise Markov random fields (MRFs with a maximum clique size of 2), Bayesian networks, and factor graphs.

We are interested in the application of the BP algorithm, regarded as a distributive algorithm, for decoding LDPC codes [6]. There are two kinds of functions, check node and bit node functions, in the bipartite factor graph associated with an LDPC code. Both functions generally apply maximum a posteriori (MAP) rule to provide extrinsic

information. At each decoding round, all bit nodes generate and flood the extrinsic information to check nodes. Then all check nodes use the incoming information as a priori information and update extrinsic information and flood the information for all bit nodes as a priori information in the next iteration. In general, the more iterations (or decoding rounds) processed, the more information gathered and thus the less error rate achieved.

BP decoding on the bipartite graph representing an LDPC code is naturally suited for parallel processing. However, except for short codes, hardware implementation of a parallel decoder requires large memory, high computational complexity and very complicated interconnection. A practical alternative would be to serialize the decoding process so that an iteration is divided into many cascaded sub-iterations. The vertical shuffled BP (VSBP) algorithm [9, 13] proposed in 2004 divides, at each iteration, all bit nodes into multiple groups and then sequentially processes these groups. Because the decoder passes the updated bit-to-check messages from the former groups, it can obtain more reliable information and thus reduce the required number of iterations compared with the BP algorithm at the fixed maximum number of iterations ( $I_{max}$ ). Moreover, it sends the updated information right to the corresponding nodes, eliminating the need of temporary storage.

Another serial iterative decoding algorithm is called the horizontal shuffled BP (HSBP) algorithm [10]. The HSBP algorithm decouples the computation of check node functions into multiple groups and performs BP on each group sequentially. The extrinsic information just updated in a group is then forwarded to the corresponding nodes in the immediate neighboring groups so that more reliable information is used to compute the new log-likelihood ratio. As a result, the required number of iterations is reduced and, given a moderate  $I_{max}$ , the HSBP algorithm outperforms the BP algorithm.

For the above two SBP algorithms, the partition of bit or check nodes in the factor

graph of an LDPC code is carried out according to the original natural order determined by the parity-check matrix. The resulting decoding schedules do not make any effort to lessen the impact of short cycles which makes the BP algorithm sends highly correlated message back to a node in just a few (2 or 3) iterations. Since the less correlated the messages used in updating a node's extrinsic information the more reliable the updated information becomes, short cycles are obviously undesirable. We propose a new decoding schedule that suppresses the detrimental effect of short cycles. The schedule is based on a node partition criterion that counts the number of short cycles a node is involved, assigns those nodes with the same cycle number to the same group and prioritizes the decoding of those groups with the smallest cycle number.

To understand and predict both the performance and the convergence behavior of the proposed decoding schedule, we develop an analytic framework based on a function which describes the composition of the extrinsic information associated with a bit at a given time. This function, which we call the message profile, measures how much each bit node has contributed to the extrinsic information of a bit. The contribution is traced through a given schedule (decoding path) that starts with the initial message computation at the contribution node and ends with the receiving node at a given iteration. Hence the profile is a function of both time and constituent nodes. At a given time we define the normalized correlation spread (NCS) of a bit as the mean-to-root-mean-squared ratio of its message profile. This parameter is then used to evaluate the degree of local flooding uniformity of a bit node up to a particular iteration. The NCS regarded a function of time (iteration) has the surprising but desired properties that it converges fast and the family of NCS' associated with all constituent nodes converge to the same steady-state value though not uniformly. Numerical results indicate that a BP-based algorithm converges when all NCS' converge, i.e., the decoder will not converge until all message profiles converge. Hence the convergence of the decoder is consistent with that of the NCS family. We further found that the decoder performance is directly related to the

common steady state value—the larger it is, the better the BER performance becomes.

We also apply the proposed analytic tool to predict the transient and steady state behavior of established LDPC codes and decoding schedules. The simulation results verify the correctness of our prediction and indicate that it can be applied to any LDPC code and the corresponding decoding algorithm/schedule.

The rest of this thesis is organized as follows. Chapter 2 gives a brief overview of the LDPC code, its factor graph representation and the associated BP algorithm. In Chapter 3 we first introduce a graph representation called multi-stage factor graphs (MSFGs) for describing the time evolution of a decoding algorithm's message-passing process. We then review two SBP algorithms and construct their MSFGs to trace the decoding paths and message-passing flows. Different decoding schedules can thus be easily visualized and their effects assessed. The new scheduling approach is presented in the following chapter and the proposed analytic method for predicting the convergence and performance of an LDPC decoder is detailed in Chapter 5. We provide some numerical examples and discuss the behavior trends in Chapter 6. The last chapter contains some concluding remarks and suggests some topics for future research.

# Chapter 2

## Low-Density Parity Check Codes

LDPC codes belong to the class of linear block codes and were invented by Gallager in 1960 [6]. The name of the code was derived from the characteristic of the corresponding parity check matrix  $H$  because the number of 1's is much smaller than that 0's in  $H$  or equivalently, the percentage (density) of 1's in  $H$  is low. An iterative decoding algorithm that gives near-maximum-likelihood performance was also proposed by Gallager.

For an LDPC code to have effective error-correcting capability, it must have a large minimum distance which, as Gallager had shown, implies the code length must be long enough. But even with Gallager's algorithm, decoding of long LDPC codes was impractical in the early 1960's. Hence, LDPC codes are almost totally forgotten until they were rediscovered by MacKay and Neal [7] in 1996. By that time Gallager's algorithm had become implementable and the code had since become a subject of intensive research as well as commercial interest.

### 2.1 Factor Graphs

Nowadays, the structure and decoding of an LDPC code are usually described and analyzed by a graphic representation called factor graphs. Factor graphs are generalization of the so-called Tanner graphs which were proposed by Michael Tanner to create larger codes from smaller ones using recursive techniques. Tanner's approach, in turn, was a generalization of a technique Elias developed for product codes.

A factor graph is a bipartite graphic representation [8] that expresses the structure of factorization  $g(x_1, \dots, x_n) = \prod_j f_j(X_j)$ , where  $f_j$  has the subset of all variables  $(x_1, \dots, x_n)$  in  $g$ . A factor graph can be divided into two groups. One is inclusive of variable nodes for variable  $x$  and the other contains of factor nodes for local functions  $f$ . Any two nodes from different groups are connected by a edge if  $x_i$  is an argument of  $f_j$ . In Fig. 2.1, there are two classes of factor nodes. The factor node in Fig. 2.1

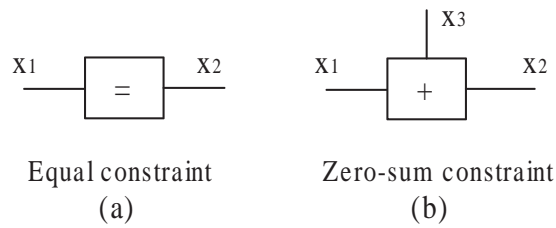


Figure 2.1: Two classes of factor nodes.

(a) has the equal constraint and its local function is  $\delta(x_1 - x_2)$ . The other in Fig. 2.1 (b) has the zero-sum constraint and its local function is  $\delta(x_1 + x_2 + x_3)$ . Then we give an example which factor graph is showed in Fig. 2.2. The function  $g$  is represented by  $g(x_1, \dots, x_6) = \delta(x_1 + x_2 + x_5)\delta(x_2 + x_3 + x_6)\delta(x_1 + x_3 + x_4) \prod_{j=1}^6 f(y_j|x_j)$ .

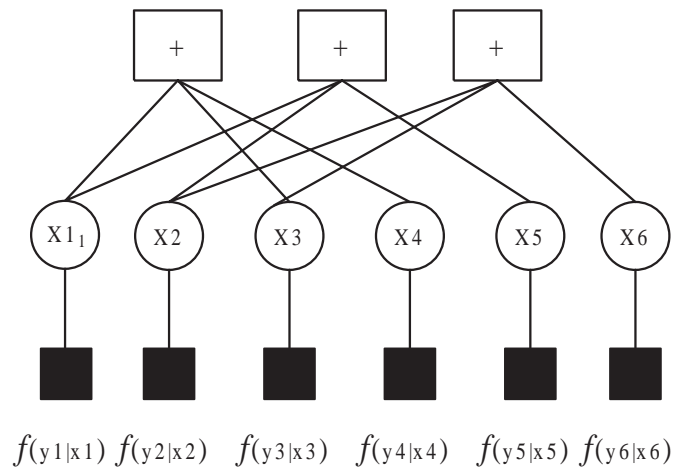


Figure 2.2: A typical factor graph.

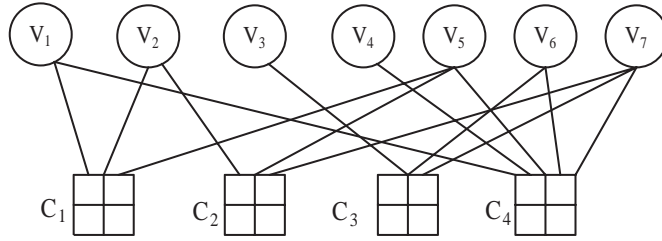


Figure 2.3: A factor graph of a (7,3) code.

## 2.2 The Belief Propagation Algorithm

LDPC codes are often decoded by using the belief propagation (BP) or Gallager algorithm [5, 8] and, for some long codes with large number of decoding iterations, the error rate performance can be very close to that predicted by Shannon. The operation of the BP algorithm used in the decoding of a LDPC codes is conveniently described by the corresponding factor graph. A factor graph of an LDPC code can be partitioned into two sparse bipartite subgraphs. The upper subgraph contains  $N$  nodes,  $(V_1, V_2, \dots, V_N)$ , called *bit nodes* and the lower subgraph contains  $M$  nodes,  $(C_1, C_2, \dots, C_M)$ , called *check nodes*. The function in the check nodes have to satisfy the relation  $H^T V = 0$ . The function of the bit nodes in a BP decoder is to combine the information sent from the check nodes. We depict the factor graph of a (7,3) code in Fig. 2.3.

For convenience of reference, we use the same notations as those used in [5, 8]. For an  $M \times N$  parity check matrix  $\mathbf{H} = [H_{mn}]$ , we denote the set of all bit nodes connecting to check node  $m$  by  $\mathcal{N}(m) = \{n : H_{mn} = 1\}$  and the set of all check nodes connecting to bit node  $n$  by  $\mathcal{M}(n) = \{m : H_{mn} = 1\}$ . Assume a codeword  $\{w_n\}$  is sent by using binary phase shift keying (BPSK) signals over an additive white Gaussian noise (AWGN) channel with zero mean and variance  $N_0/2$  and then let  $\{y_n\}$  be the received signals.

Let  $F_n$  be the log-likelihood ratio (LLR) of bit node  $n$  and  $\{\varepsilon_{mn}^i\}$  be the message which is sent from check node  $m$  to bit node  $n$ ,  $\{z_{mn}^i\}$  the message passed from bit node

$n$  to check node  $m$  at the  $i$ th iteration. We also denote by  $\{z_n^i\}$  the a posteriori LLR of bit node  $n$ .

For the conventional BP algorithm, all values of check-to-bit messages  $\varepsilon_{mn}^i$  at the  $i$ th iteration are updated by using the values of bit-to-check messages  $z_{mn}^{i-1}$  at the  $(i-1)$ th iteration. Furthermore, all new values of bit-to-check messages  $z_{mn}^i$  are updated by passing the values of newly updated check-to-bit messages  $\varepsilon_{mn}^i$ . The detailed BP algorithm is described below.

1. Initialization: set  $i = 1$  and the maximum number of iterations  $= I_{max}$ . Initialize

$$F_n = 4y_n/N_0. \text{ For each } m, n, \text{ set } z_{mn}^0 = F_n.$$

2. • At check nodes:  $\forall m, 1 \leq m \leq M$ , and each  $n \in \mathcal{M}(m)$ , compute

$$\tau_{mn}^i = \prod_{n' \in \mathcal{N}(m) \setminus n} \tanh\left(\frac{z_{mn'}^{i-1}}{2}\right) \quad (2.1)$$

$$\varepsilon_{mn}^i = \log \frac{1 + \tau_{mn}^i}{1 - \tau_{mn}^i} \quad (2.2)$$

• At bit nodes:  $\forall n, 1 \leq n \leq N$ , and each  $m \in \mathcal{M}(n)$ , calculate

$$z_{mn}^i = F_n + \sum_{m' \in \mathcal{M}(n) \setminus m} \varepsilon_{m'n}^i \quad (2.3)$$

$$z_n^i = F_n + \sum_{m' \in \mathcal{M}(n)} \varepsilon_{m'n}^i \quad (2.4)$$

3. Make hard decision on  $\{z_n^i\}$  to obtain a tentative decoded codeword  $\{D_n^i\}$ , i.e. if  $z_n^i > 0$ ,  $D_n^i = 0$ , otherwise  $D_n^i = 1$ . If  $\mathbf{HD}^i = 0$ , i.e. all syndromes equal to zero, or the maximum number of iterations  $I_{max}$  is reached, stop operation and  $\{D_n^i\}$  is the decoded codeword. Otherwise, set  $i = i + 1$  and go to Step 2.



# Chapter 3

## Shuffled Iterative Decoding of LDPC Codes

### 3.1 Multi-stage Factor Graphs of LDPC codes

Multi-stage factor graphs (MSFG) [10] is a useful trellis-like graph for representing message-passing process of an iterative decoder. Such a graphic representation is used to investigate the effect of various decoding algorithms/schedules. One can easily distinguish a decoding procedure (path) in an MSFG from other paths or find candidate paths of different lengths from bit node  $V_i$  to bit node  $V_j$ . We also use the MSFG for search cycles of a code and determine its *girth* (the length of the shortest cycle).

**Example 1** For the code with the parity check matrix  $H$  given below, the MSFG representation of the BP decoder is shown in Fig. 3.1.

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

In Fig. 3.1, the message originated from bit node  $V_1$ , after travelling through  $C_4$ ,  $V_5$ ,  $C_4$ , will return to  $V_1$  again. Hence the message generated at bit node  $V_1$  goes back to itself after only 2 (time) stages and the corresponding path forms a short cycle of length 4, which is the *girth* of the LDPC code.

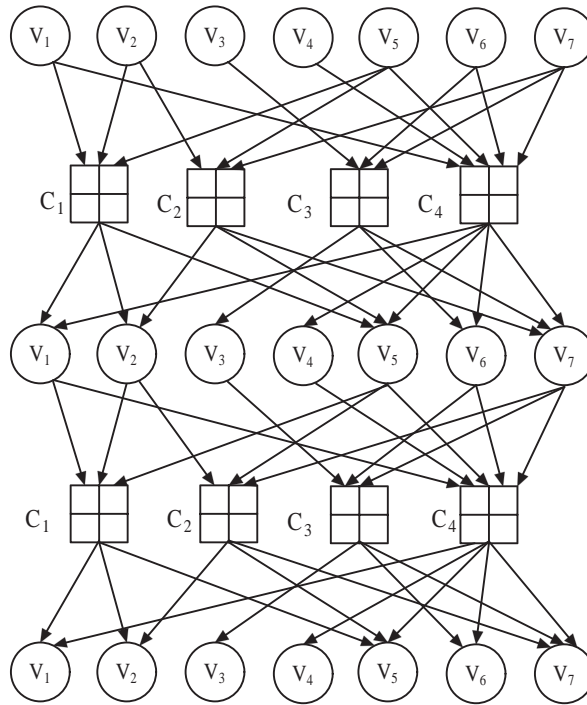


Figure 3.1: A MSFG that describes a conventional BP decoding (schedule) of a  $(7,3)$  code.

## 3.2 Shuffled Iterative Decoding Algorithms

The BP algorithm, as described by its MSFG or factor graph representation, can be processed in fully parallelism with no memory contention concern. However, such a fully parallel implementation requires large number of processing units, high computing complexity, large memory space and very complicate routing network connecting the processors and memory banks [16]. The codeword length of the LDPC code used is generally large and it is very difficult and costly if not impractical to realize a fully parallel decoder for any LDPC code longer than, say, 500. It is natural to have a serial architecture that serializes the decoding process by dividing a single iteration into several sub-iterations such that a reasonable number of processing units can perform the message computing and passing over a limited number nodes in each sub-iteration. BP within a group of nodes is performed in parallel but groups are processes serially. The parallel-

serial architecture is often referred to as shuffled decoding. Depending on whether bit nodes or check nodes are partitioned, we have the vertical shuffled BP algorithm or the horizontal shuffled BP algorithm. In the ensuing discussion we use the same notations as those defined in Chapter 2.

### 3.2.1 Vertical Shuffled Belief Propagation Algorithm

The vertical shuffled belief propagation (VSBP) algorithm [6, 9, 13] divides all bit nodes into several groups and decoding is carried out in group-by-group manner. The conventional BP algorithm uses all bit-to-check messages  $z_{mn}^{i-1}$  obtained at the  $(i-1)$ th iteration to update the values of check-to-bit messages  $\varepsilon_{mn}^i$  at the  $i$ th iteration. It then use the newly updated  $\varepsilon_{mn}^i$  to produce  $z_{mn}^i$ . However, for the VSBP algorithm, after a sub-iteration is performed on a group, some bit-to-check messages become available and can be sent to the neighboring group for subsequent decoding sub-iteration. hence, instead of  $z_{mn}^{i-1}$ , all latest updated bit-to-check messages  $z_{mn}^i$  can be used [6].

More specifically, in each group, the VSBP algorithm applies the newly updated bit-to-check messages  $z_{mn}^i$  (obtained in the previous group) to generate for the nodes within the group new values of the check-to-bit messages in parallel as the conventional BP algorithm does. We summarize the VSBP algorithm as follows and depict the corresponding MSFG in Fig. 3.2.

1. Initialization: Set  $i = 1$  and the maximum number of iterations= $I_{max}$ . Initialize

$$\forall m, n, \text{ set } z_{mn}^0 = F_n = 4y_n/N_0.$$

2. For  $1 \leq g \leq G$ ,

- At check nodes:  $\forall n, 1 \leq n \leq N$ , and each  $m \in \mathcal{M}(n)$ , calculate

$$\tau_{mn}^i = \prod_{\substack{n' \in \mathcal{N}(m) \setminus n \\ n' \leq g \cdot N_G}} \tanh\left(\frac{z_{mn'}^i}{2}\right) \prod_{\substack{n' \in \mathcal{N}(m) \setminus n \\ n' > g \cdot N_G}} \tanh\left(\frac{z_{mn'}^{i-1}}{2}\right) \quad (3.1)$$

$$\varepsilon_{mn}^i = \log \frac{1 + \tau_{mn}}{1 - \tau_{mn}} \quad (3.2)$$

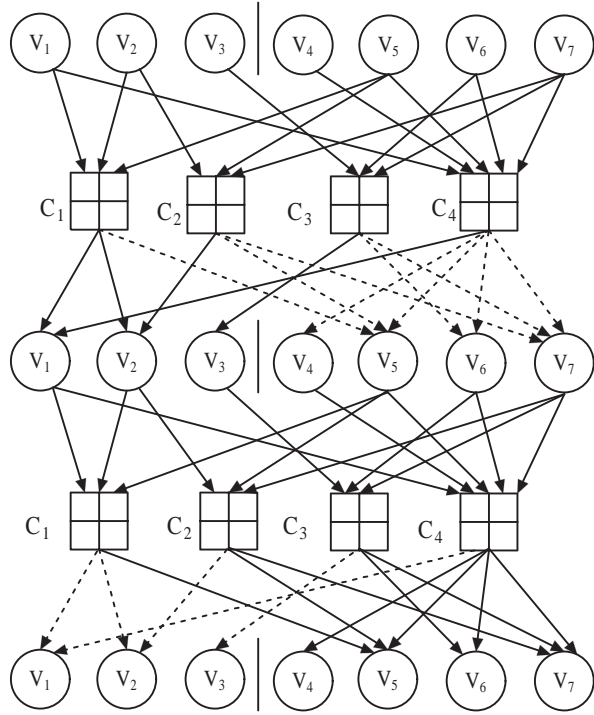


Figure 3.2: The MSFG representation of a VSBP decoding schedule of a (7,3) code.

- At bit nodes:  $\forall n, (g-1) \cdot N_G + 1 \leq n \leq g \cdot N_G$ , and each  $m \in \mathcal{M}(n)$ , calculate

$$z_{mn}^i = F_n + \sum_{m' \in \mathcal{M}(n) \setminus m} \varepsilon_{m'n}^i \quad (3.3)$$

3. At bit nodes:  $\forall n, 1 \leq n \leq N$ , and each  $m \in \mathcal{M}(n)$ , calculate

$$z_n^i = F_n + \sum_{m' \in \mathcal{M}(n)} \varepsilon_{m'n}^i \quad (3.4)$$

4. Make hard decisions on  $\{Z_n^i\}$  to obtain a tentative decoded codeword  $\{D_n^i\}$ . If  $\mathbf{HD}^i = 0$ , or the maximum number of iterations  $I_{max}$  is reached, stop decoding and output  $\{D_n^i\}$ . Otherwise, set  $i = i + 1$  and go to Step 2.

### 3.2.2 Horizontal Shuffled Belief Propagation Algorithm [11]

In contrast to the VSBP algorithm, the horizontal shuffled belief propagation (HSBP) algorithm partitions the check nodes into several groups and serialize the belief updates

in a group-by-group manner. The check node processors can use some bit-to-check messages that are produced in the previous sub-iteration to generate more updated LLR estimates. Before starting to update certain check-to-bit messages  $\varepsilon^i_{mn}$  at the  $i$ th iteration, we update all values of bit-to-check messages  $z^i_{mn'}$  for all bit nodes  $n'$  which are connected to check node  $m$  but exclusive of bit node  $n$ . The newly updated messages for bit node  $n$  are used to generate updated bit-to-check node messages, replacing  $z^i_{mn}$  by  $z_{mn}$ . We summarize the HSBP algorithm in followings. The corresponding MSFG is shown in Fig. 3.3.

1. Initialization: set  $i = 1$  and the maximum number of iterations= $I_{MAX}$ . Initialize

$$F_n = 4y_n/N_0. \text{ For each } m, n, \text{ set } z_{mn} = F_n.$$

2. Let  $G$  be the number of groups and  $M_G$  be the number of check nodes in a group.

For  $1 \leq g \leq G$ ,

- At check nodes:  $\forall m, (g-1) \cdot M_G + 1 \leq m \leq g \cdot M_G$  and each  $n \in \mathcal{N}(m)$ , calculate

$$\tau_{mn}^i = \prod_{n' \in \mathcal{N}(m) \setminus n} \tanh\left(\frac{z_{mn'}}{2}\right) \quad (3.5)$$

$$\varepsilon_{mn}^i = \log \frac{1 + \tau_{mn}^i}{1 - \tau_{mn}^i} \quad (3.6)$$

- At bit nodes:  $\forall n, 1 \leq n \leq N$ , and each  $m \in \mathcal{M}(n)$ , calculate

$$z_{mn} = F_n + \sum_{\substack{m' \in \mathcal{M}(n) \setminus m \\ m' \leq g \cdot M_G}} \varepsilon_{m'n}^i + \sum_{\substack{m' \in \mathcal{M}(n) \setminus m \\ m' > g \cdot M_G}} \varepsilon_{m'n}^{i-1} \quad (3.7)$$

3. At bit nodes:  $\forall n, 1 \leq n \leq N$ , and each  $m \in \mathcal{M}(n)$ , calculate

$$z_n^i = F_n + \sum_{m \in \mathcal{M}(n)} \varepsilon_{mn}^i \quad (3.8)$$

4. Make hard decisions on  $\{z_n^i\}$  to render a temporary decoded codeword  $\{D_n^i\}$ . If  $\mathbf{HD}^i \mathbf{i}^T = 0$  or the maximum number of iterations  $I_{max}$  is reached, stop decoding and output  $\{D_n^i\}$ . Otherwise, set  $i = i + 1$  and go to Step 2.

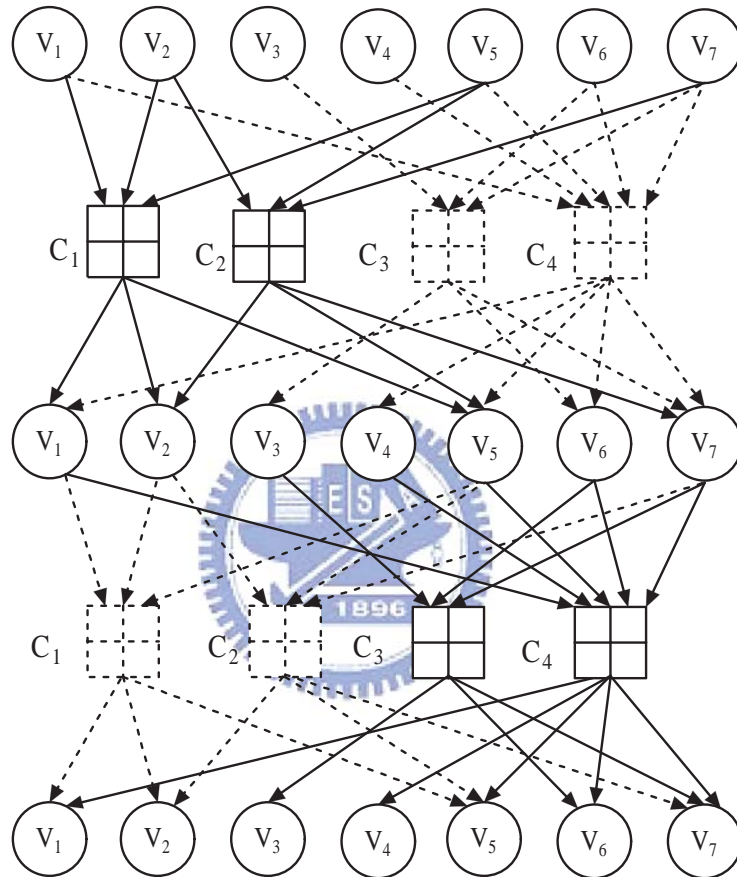


Figure 3.3: A MSFG that describes the HSBP decoding schedule of a (7,3) code. The dash lines represent the (message) flooding that are by-passed.

# Chapter 4

## A Novel Scheduling Method for the HSBP Algorithm

The HSBP algorithm directly groups check nodes by the row ordinal. Obviously this grouping method does not consider the short cycle effects. [13] points out the cycle effect and weights the effect by the shortest cycle and girth to measure its effect. We follow a similar line of thinking to develop a new decoding schedule and apply it to the HSBP algorithm to reduce the influence of the short cycles. Since the basic HSBP structure remains intact, the proposed decoding schedule is in fact equivalent to a new partition of the check nodes or simply a rearrangement of the check nodes.

### 4.1 A Decoding Schedule for Reducing Short Cycle Effects

A cycle (loop) in a graph is a closed path which can be represented by a finite-length sequence of vertices (nodes) in the graph. Expressing the finite-sequence as a vector, then a cycle is equivalent to a vector whose first and last entries are identical. A cycle (loop) of length  $2k$  is thus of the form  $\ell = (V_{n_1}, C_{m_1}, V_{n_2}, C_{m_2}, \dots, C_{m_k}, V_{n_{k+1}})$ , where  $n_i \neq n_j$  unless  $i = 1$  and  $j = k + 1$ , and  $m_i \neq m_j$  for all  $i \neq j$ .

We are concerned about the number and the length of cycles a check node is involved. Each cycle passes through several check nodes. The more cycles a check node is involved, the more correlated extrinsic information it will send back to its neighboring bit nodes.

The length of a cycle determines the time elapsed for a bit node to receive a message in which the information originated from itself is embedded. The correlation between the original message and the returning message is in inverse proportion to the length of the path through which the original message has traveled for the BP (sum-product) algorithm treats all incoming messages equally, i.e., it does not give special weight to any incoming message in producing a outgoing message.

The length of a cycle  $\ell$  is denoted by  $|\ell|$  and the set of all length- $2(k+1)$  cycles is denoted by  $\mathcal{L}_k$ . Let  $\gamma_i(j)$  be the number of length- $2(j+1)$  cycles which pass through check node  $i$ , i.e.,  $\gamma_i(j) = |\{\ell | i \in \ell, |\ell| = 2(j+1)\}|$ . Define the cycle factor of node  $i$  as

$$w_i = \sum_{j=1}^L f\left(\frac{1}{j+1}\right) \gamma_i(j)$$

where  $L < L_{max}$  and  $L_{max}$  is the maximum cycle length for the code of concern. For a block with  $M$  parity-check bits (i.e.,  $M$  check nodes) the maximum cycle length  $L_{max}$  is upper-bounded by  $2M$ .  $f(\cdot)$  is a linear or nonlinear weighting function and the argument  $1/(j+1)$  reflects the fact that the degree of correlation is in reverse proportion to the cycle length. The cycle factor  $w_i$  thus represents the average degree of correlation of node  $i$  with its neighboring bit nodes.

We summarize our cycle-factor-based HSBP algorithm in the following.

1. Initialization:  $1 \leq i \leq M, 1 \leq j \leq L, \gamma_i(j) = 0$ .
2. Calculate  $\gamma_i(j)$ :  $\forall j$ , search all cycles with length  $2(j+1)$ . If there are  $k$  length- $2(j+1)$  cycles including check node  $i$ ,  $\gamma_i(j) = k$ .
3. Reorder the parity check matrix: rearrange the rows of the parity check matrix in increasing order of  $w_i$ .
4. Perform the HSBP decoding algorithm on bipartite factor graph generated by the the rearranged parity check matrix.

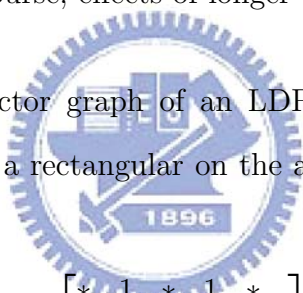


The issue of selecting appropriate  $L$  and  $f(\cdot)$  is beyond the scope of this thesis. For simplicity, we choose  $L$  to be such that  $2(L+1)$  is the girth (length of the shortest cycle) of the code. Hence  $L = 1$  or  $2$  when the girth is 4 or 6.

## 4.2 Searching for Length-4 Cycles

The proposed algorithm requires that all cycles of the same length be found. This section describe an efficient algorithm to search for all length- $2k$  cycles. To begin with, we define an  $M \times L$  matrix  $\mathbf{\Gamma}_{M \times L} = [\gamma_i(j)]$  and an  $M \times 1$  matrix  $\mathbf{\Delta} = [\delta_i]$ , where  $M$  is the number of check nodes and  $L$  indicates that the maximum length of a cycle is  $2(L+1)$ . As searching all cycles requires enormous computing complexity we develop an algorithm for finding all length- $2k$  cycles. Only length-4 and length-6 cycles are considered in subsequent discourse; effects of longer cycles are not considered.

A length-4 cycle on the factor graph of an LDPC code corresponds to four non-zero entries which constitutes a rectangular on the associated parity check matrix. An exemplary placement is



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

We exploit this and similar properties to check if there exists any length-4 cycle. The flow chart shown in Fig. 4.1 uses the above fact (and its extensions) to search for all length- $2k$  cycles. For an  $M \times N$  parity check matrix, the complexity of our algorithm is at most  $O\left(C_2^{d_{c,max}} \times C_2^{d_{v,max}}\right)$ , where  $d_{c,max}$  and  $d_{v,max}$  are the maximum degrees corresponding to all check nodes and bit nodes. For a  $(d_c, d_v)$  regular LDPC code, the searching complexity becomes  $O\left(C_2^{d_c} \times C_2^{d_v}\right)$ .

**Example 2** Consider the LDPC code with the following parity check matrix

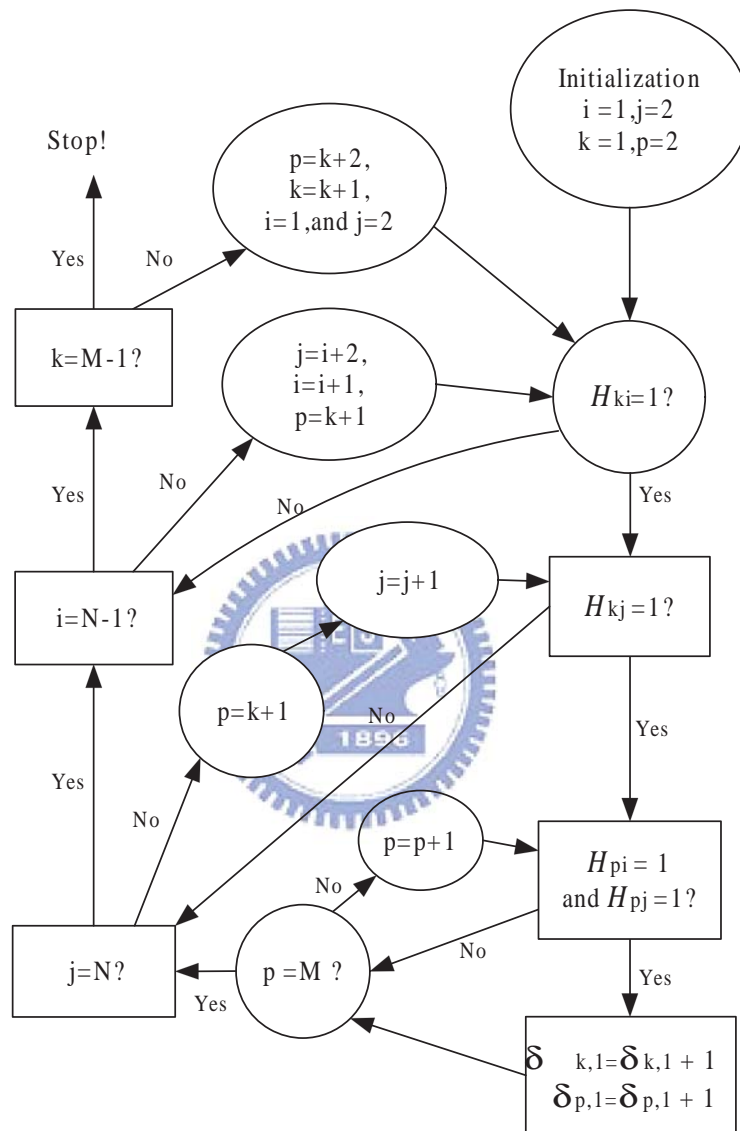


Figure 4.1: Flow chart of the proposed searching algorithm that computes the number of length- $2k$  cycles.

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

It is straightforward to show that all possible length-4 cycles are given by

1.  $(V_1, C_1, V_5, C_4, V_1)$ ,
2.  $(V_2, C_1, V_5, C_2, V_2)$ ,
3.  $(V_5, C_2, V_7, C_4, V_5)$ ,
4.  $(V_6, C_3, V_7, C_4, V_6)$ .

Since  $\mathbf{\Gamma}_{4 \times 1} = [2 \ 2 \ 1 \ 3]^T$  and  $\mathbf{\Delta}_{4 \times 1} = [1 \ 1 \ 0.5 \ 1.5]^T$ , we obtain the reordered parity check matrix as

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

The corresponding MSFG is shown in the left part of Fig. 4.2. In the first stage,  $C_1$  and  $C_3$  receive information from  $\{V_1, V_2, V_5\}$  and  $\{V_3, V_6, V_7\}$ , meaning the information these two check nodes receive is uncorrelated if information from different bit nodes are uncorrelated. On the contrary, for the original HSBP algorithm whose MSFG is shown in the right part of Fig. 4.2,  $C_1$  and  $C_2$  receive information from  $\{V_1, V_2, V_5\}$  and  $\{V_2, V_5, V_7\}$  so that both of them are influenced by messages from nodes  $\{V_2, V_5\}$ . This simple example shows that the proposed method is capable of extending the cycle length and reduces the dependence among different message flows in the course of an iterative decoding process.

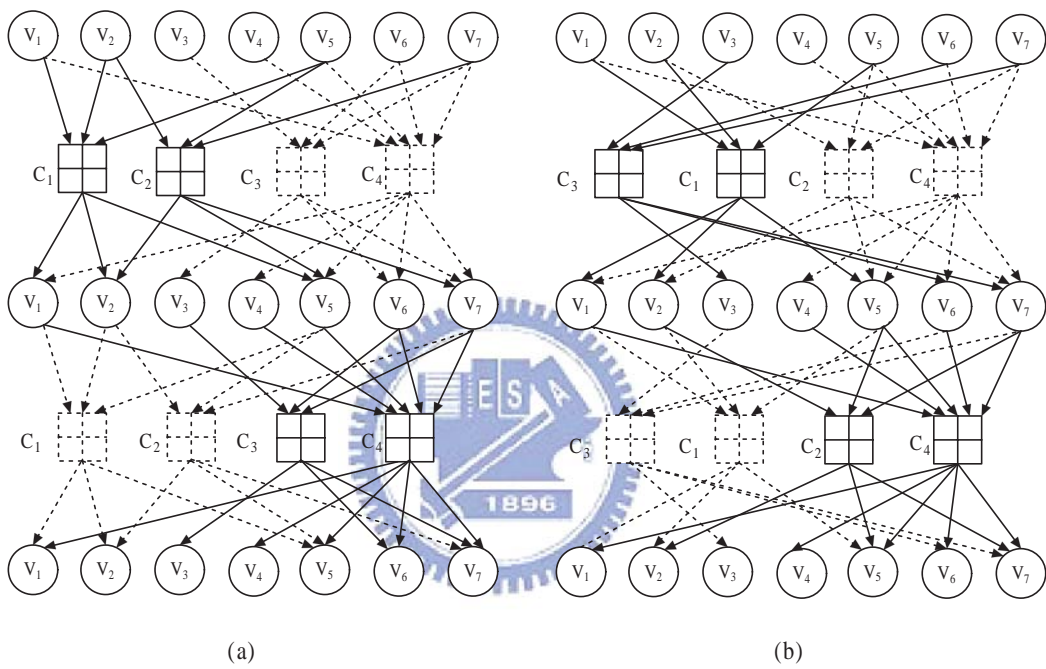


Figure 4.2: MSFG representations of a cycle-factor based (a) and conventional (b) HSBP algorithms for an LDPC code with  $G=2$ .

# Chapter 5

## Message Flow Distribution and Decoder Behaviors

An iterative decoder's convergence behavior can be conveniently analyzed by computing the corresponding EXIT chart. The EXIT chart, however, is based on the assumptions of long codes and Gaussian distribution of the output extrinsic information. For a serial decoding schedule with small number of outputs per sub-iteration, the prediction by EXIT chart may not be accurate. We propose an alternative analytic tool to predict the convergence behavior and the high SNR error rate performance of iteratively decoded LDPC codes. Regarding the MSFG of the code as a time-evolving network, we trace each and every message flows over the network along their designated paths (schedules). We assume that the *pipes* (branches) connecting nodes to their neighbors have unlimited capacity so that the flow size within any pipe can be any positive number. We analyze the composition of the flow in a branch, compute its size at each stage (iteration), evaluate the time(iteration)-dependent flow distribution. Finally we use a critical statistical parameter associated with the distribution to describe the convergence and steady state behavior of the code.

### 5.1 Profile of the Conventional BP Algorithm

For a parity check matrix  $\mathbf{H} = [H_{mn}]$ , we denote, as was defined before, the set of all bit nodes connecting to check node  $m$  by  $\mathcal{N}(m) = \{n : H_{mn} = 1\}$  and the set of all

check nodes connecting to bit node  $n$  by  $\mathcal{M}(n) = \{m : H_{mn} = 1\}$ . We also need the following definitions.

1.  $c_{mn}(j; i)$  = the *amount of contribution* bit node  $V_j$  has made and is embedded in the message passed from check node  $m$  to bit node  $n$  at the  $i$ th iteration.  
= the average amount of flow within the message flow passed from  $C_m$  to  $V_n$  at the  $i$ th iteration that was originated from  $V_j$  at earlier iterations.
2.  $b_{nm}(j; i)$  = the *amount of contribution* bit node  $V_j$  has made and is embedded in the message passed from bit node  $n$  to check node  $m$  at the  $i$ th iteration.  
=the average amount of flow within the message flow passed from  $V_m$  to  $C_n$  at the  $i$ th iteration that was originated from  $V_j$  at earlier iterations.
3.  $p_n(j; i)$  = the *bit node message flow profile* of  $V_n$  at the  $i$ th iteration.

$p_n(j; i)$ , regarded as a function of the bit node parameter  $j$ , quantifies the composition of the contributions of various bit node inputs up to the  $i$ th iteration. These contributions are embedded in all check-to-bit messages passed to  $V_n$  at the  $i$ th iteration.

We further define the *average message flow size* that is incident to  $V_n$ ,  $\mu(n, i)$ , and the *average total flow size*,  $m_i$ , at iteration  $i$  as

$$\mu(n, i) = \frac{1}{N} \sum_j p_n(j, i) \quad (5.1)$$

$$m_i = \sum_n \mu(n, i) \quad (5.2)$$

To quantify the contributions, we need to trace the information generated by a given input bit at various stages through the message path it has travelled. As the channel values associated with the bit nodes are independent and identically distributed (i.i.d.), we assign equal (contribution) capacity to all initial messages generated by bit nodes. The above definitions are thus defined and computed in an iterative manner as follows.

1. Initialization: Set  $i=1$  and  $b_{nm}(j; 0) = \delta(j - n)$ ,  $1 \leq j \leq N$ , where

$$\delta(n) = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases} . \quad (5.3)$$

2. • At check nodes:  $\forall n, 1 \leq n \leq N$ , and each  $m \in \mathcal{M}(n)$ , compute

$$c_{mn}(j; i) = \frac{\sum_{n' \in \mathcal{N}(m) \setminus n} b_{n'm}(j; i-1)}{\deg(m) - 1}, \quad 1 \leq j \leq N, \quad (5.4)$$

where  $\deg(m)$  = the degree of check node  $m$ , and the normalization factor is used to satisfy the flow conservation principle.

- At bit nodes:  $\forall n, 1 \leq n \leq N$ , and each  $m \in \mathcal{M}(n)$ , compute

$$b_{nm}(j; i) = \delta(j - n) + \sum_{j, m' \in \mathcal{M}(n) \setminus m} c_{m'n}(j, i) \quad (5.5)$$

$$p_n(j; i) = \delta(j - n) + \sum_{j, m \in \mathcal{M}(n)} c_{mn}(j; i) \quad (5.6)$$

3.  $\forall n, 1 \leq j \leq N$ , compute the profile mean and variance

$$\mu(n, i) = \frac{\sum_j p_n(j; i)}{N} \quad (5.7)$$

$$\sigma^2(n, i) = \frac{\sum_j [b_n(j; i) - \mu(n, i)]^2}{N} \quad (5.8)$$

## 5.2 Profile of the VSBP Algorithm

The weight distribution obtained by using the VSBP algorithm is modified from that by using the conventional BP algorithm. The iterative procedure for computing the corresponding flow densities and related statistics is summarized below.

1. Initialization: Set  $i=1$  and  $b_{mn}(j; 0) = \delta(j - n)$ ,  $1 \leq j \leq N$ .

2. Let  $G$  be the number of groups and  $N_G$  be the number of bit nodes in a group.

- For  $1 \leq g \leq G$ ,

- At check nodes:  $\forall n, (g-1)N_G \leq n \leq gN_G$ , and each  $m \in \mathcal{M}(n)$ , compute

$$c_{mn}(j; i) = \frac{c_{mn}^i(j) + c_{mn}^{i-1}(j)}{\deg(m) - 1}, \quad (5.9)$$

where

$$c_{mn}^i(j) = \sum_{\substack{n' \in \mathcal{N}(m) \setminus n \\ n' \leq g \cdot N_G}} b_{mn'}(j; i) \quad (5.10)$$

$$c_{mn}^{i-1}(j) = \sum_{\substack{n' \in \mathcal{N}(m) \setminus n \\ n' > g \cdot N_G}} b_{mn'}(j; i-1) \quad (5.11)$$

– At bit nodes:  $\forall n, (g-1) \cdot N_G + 1 \leq n \leq g \cdot N_G$ , and each  $m \in \mathcal{M}(n)$ , calculate

$$b_{mn}(j; i) = \delta(j-n) + \sum_{m' \in \mathcal{M}(n) \setminus m} c_{m'n}(j; i-1) \quad (5.12)$$

• At bit nodes:  $\forall n, 1 \leq n \leq N$  and each  $m \in \mathcal{M}(n)$ , calculate

$$p_n(j; i) = \delta(j-n) + \sum_{m \in \mathcal{M}(n)} c_{mn}(j; i) \quad (5.13)$$

3.  $\forall n, 1 \leq j \leq N$ , compute the profiles's mean and variance

$$\mu_n(i) = \frac{\sum_j p_n(j; i)}{N} \quad (5.14)$$

$$\sigma_n^2(i) = \frac{\sum_j [p_n(j; i) - \mu_n(i)]^2}{N} \quad (5.15)$$

### 5.3 Profile for the HSBP Algorithm

The flow profiles of the HSBP algorithm are obtained by a iterative algorithm similar to that for the VSBP algorithm.

1. Initialization: Set  $i=1$  and  $b_{mn}(j, 0) = \delta(j-n)$ ,  $1 \leq j \leq N$ .

2. Let  $G$  be the number of groups and  $M_G = |G|$ .

• For  $1 \leq g \leq G$ ,

– At check nodes:  $\forall m, (g-1)M_G + 1 \leq m \leq gM_G$ , and each  $n \in \mathcal{N}(m)$ , calculate

$$c_{mn}(j, i) = \frac{\sum_{n' \in \mathcal{N}(m) \setminus n} b_{mn'}(j, i)}{\deg(m) - 1} \quad (5.16)$$



- At bit nodes:  $\forall m, (g-1)M_G + 1 \leq m \leq gM_G$ , and each  $n \in \mathcal{N}(m)$ , calculate

$$b_{mn}(j, i) = \delta(j - n) + \sum_{\substack{m' \in \mathcal{M}(n) \setminus m \\ m' \leq gM_G}} c_{m'n}(j; i) + \sum_{\substack{m' \in \mathcal{M}(n) \setminus m \\ m' > gM_G}} c_{m'n}(j; i - 1) \quad (5.17)$$

- At bit nodes:  $\forall n, 1 \leq n \leq N$ , and each  $m \in \mathcal{M}(n)$ , calculate

$$p_n(j; i) = \delta(j - n) + \sum_{m \in \mathcal{M}(n)} c_{mn}(j; i) \quad (5.18)$$

3.  $\forall n, 1 \leq j \leq N$ , calculate

$$\mu(n, i) = \frac{\sum_j p_n(j; i)}{N} \quad (5.19)$$

$$\sigma^2(n, i) = \frac{\sum_j [p_n(j; i) - \mu(n, i)]^2}{N} \quad (5.20)$$

**Example 3** Consider an LDPC code with the parity matrix  $H$

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

The first two stages of the corresponding MSFG is shown in 3.1. The following two matrices show the message flow at the first two iterations when conventional BP algorithm is used. The value at the  $(V_a, V_b)$  position, (the row  $V_a$  and the column  $V_b$ ), equals to  $\{bw_{a,b}^i\}$ , the times which the bit node  $V_b$  passes through the bit node  $V_a$ . The value at the  $(V_a, \mu)$  position equals to  $\mu_a^i$ , the mean of the message flow profile from all bit nodes for the bit node  $V_a$ . Moreover, the value at the  $(V_a, \sigma^2)$  position equals to  $\sigma_a^2$ , the variance of the profile mean from all bit nodes for the bit node  $V_a$ . We take the value of  $\mu_a^i/\sigma_a^i$  to analyze the profile for the bit node  $V_a$ . If the values of  $\mu_a^i/\sigma_a^i$  for all  $a$  are average distributed at the  $i$ th iteration, it means that the decoding algorithm starts to be balanced. The less the value  $i$  is, the high the convergent speed of the error rate

performance becomes. For example, according to the value at the  $(V_4, V_5)$  in the first matrix, the bit node  $V_5$  from the first stage to the second stage will totally pass through the bit node  $V_4$  0.25 times. As the values of  $\mu_a^i/\sigma_a^i$  are unevenly distributed across  $a$ , the corresponding decoding process has not reached its steady state yet.

Iteration 1:

	$V_1$	$V_2$	$V_3$	$V_4$	$V_5$	$V_6$	$V_7$	$\mu$	$\sigma^2$	$\mu/\sigma$
$V_1$	1	0.5	0	0.25	0.75	0.25	0.25	0.428571	0.102041	1.34164
$V_2$	0.5	1	0	0	1	0	0.5	0.428571	0.173469	1.02899
$V_3$	0	0	1	0	0	0.5	0.5	0.285714	0.132653	0.784465
$V_4$	0.25	0	0	1	0.25	0.25	0.25	0.285714	0.0969388	0.917663
$V_5$	0.75	1	0	0.25	1	0.25	0.75	0.571429	0.137755	1.5396
$V_6$	0.25	0	0.5	0.25	0.25	1	0.75	0.428571	0.102041	1.34164
$V_7$	0.25	0.5	0.5	0.25	0.75	0.75	1	0.571429	0.0663265	2.2188

Iteration 2:

	$V_1$	$V_2$	$V_3$	$V_4$	$V_5$	$V_6$	$V_7$	$\mu$	$\sigma^2$	$\mu/\sigma$
$V_1$	1.25	1.125	0.25	0.375	1.125	0.5	1.125	0.82143	0.15561	2.0823
$V_2$	1.125	1.5	0.25	0.5	1.25	0.75	1.125	0.92857	0.16901	2.2587
$V_3$	0.25	0.25	1	0.25	0.5	0.625	0.625	0.5	0.06696	1.9322
$V_4$	0.375	0.5	0.25	1	0.5	0.375	0.5	0.5	0.04911	2.2563
$V_5$	1.125	1.25	0.5	0.5	2	0.875	1.25	1.0714	0.23151	2.2268
$V_6$	0.5	0.75	0.625	0.375	0.875	1.25	0.875	0.75	0.07143	2.8062
$V_7$	1.125	1.125	0.625	0.5	1.25	0.875	1.5	1	0.10714	3.0551

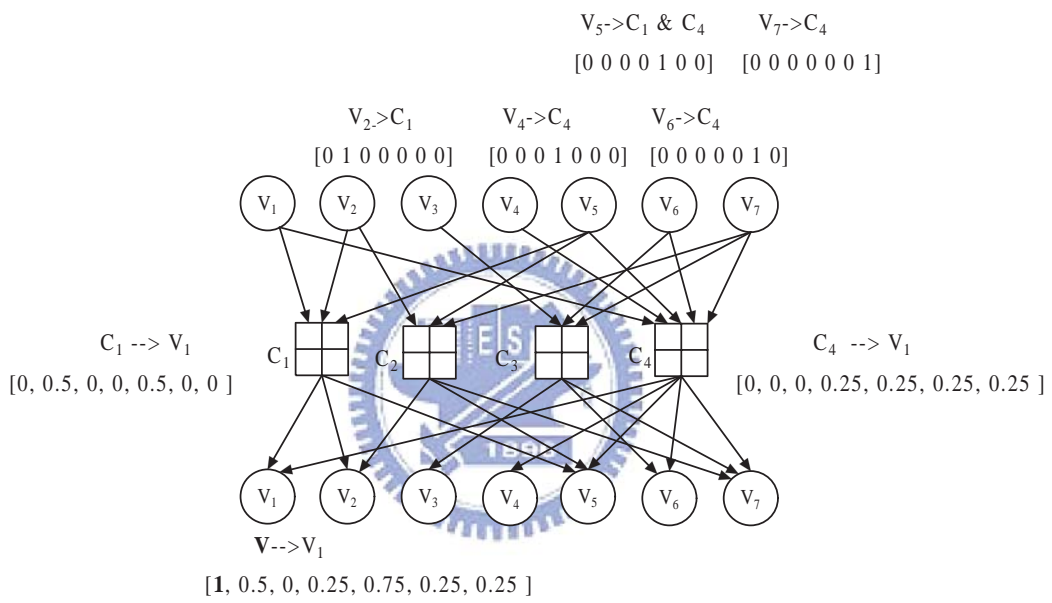


Figure 5.1: Message flow of bit node  $V_1$  at the first iteration.

# Chapter 6

## Numerical Results and Discussion

We report some numerical results derived from computer simulation in this chapter. Cycle-based and conventional decoding schedules for HSBP and VSBP decoders with  $G = 2, 4$  are considered as well as conventional BP decoders. For notational brevity, we use the abbreviations, C-BP C-HSBP, C-VSBP for conventional BP, HSBP, and VSBP algorithms (schedules), and CB-HSBP, CB-VSBP for cycle-based HSBP and VSBP algorithms (schedules). We first check the error rate performance and examine the corresponding profile behaviors. A relation between the error rate performance to the evolution of profiles is then firmly established. Finally, we use our analytic tool to predict the performance of the codes derived from expanding the base model matrix defined in the IEEE 802.11n specification [15] and those from the base model given in the IEEE 802.16e specification [17]. Computer simulations are performed to validate our prediction. Only  $L = 2$  for the CB- HSBP algorithm is discussed in this thesis.

### 6.1 Error Rate Performance Comparison

The base model matrix in the IEEE 802.11n specification is used to create rate  $1/2$  quasi-cyclic LDPC (QC-LDPC) codes with different code length. In Fig. 6.1, the decoder's packet error rate (PER) performance of  $(240, 120)$  QC-LDPC code with  $I_{max} = 5, 10$  and  $G = 2$  is presented. The cycle-based HSBP schedule outperforms the other schedules when  $I_{max} = 5$ . For  $I_{max} = 10$ , cycle-based and conventional HSBP schedules

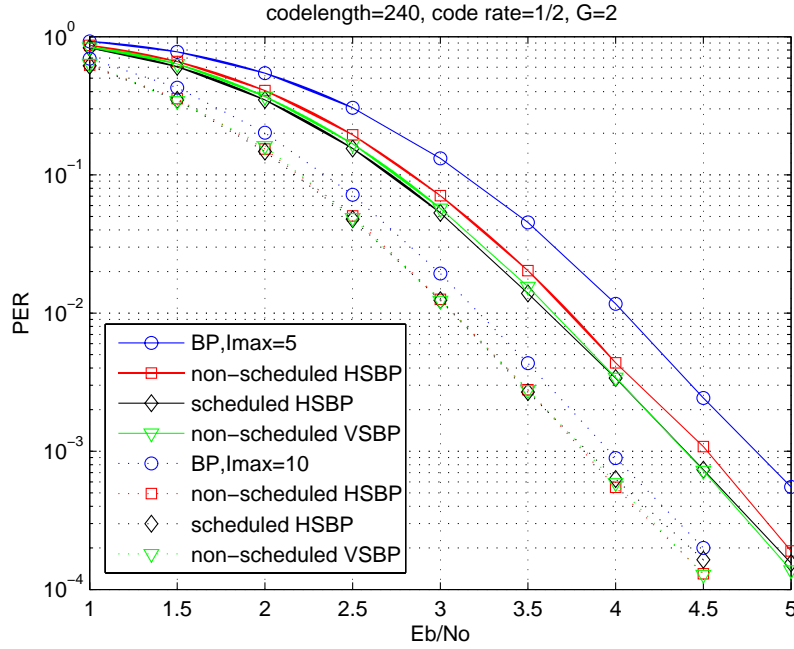


Figure 6.1: Package error rate performance of the (240, 120) QC-LDPC code with  $I_{max} = 5, 10$ ,  $G=2$  and different decoding schedules.

give similar performance but still outperform the conventional BP algorithm.

If the (480, 240) QC-LDPC code is used, Fig. 6.2 indicates that the PER performance of the cycle-based schedules surpass all others for  $I_{max} = 5$  and 10 with the conventional BP schedule yields the worst PER. Similar performance trend can be noticed for the (1080, 540) QC-LDPC code with  $I_{max} = 5, 10$  and  $G = 2$ ; see Fig. 6.3,

The (240, 120) code has a short code length and requires less iterations to reach convergence. Therefore, at  $I_{max} = 10$ , the PER performance for both HSBP schedules are about the same. In contrast, the (480, 240) and (1080, 540) codes have longer code length and the proposed schedule enjoys the edge of having a faster convergence rate.

The effect of the node partition can be found in Figs. 6.4 and 6.5 where the performance of various schedules with  $I_{max} = 5, 10$  and  $G = 4$  for decoding the (480, 240) and (1080, 540) QC-LDPC codes are shown. The simulation results show that the proposed

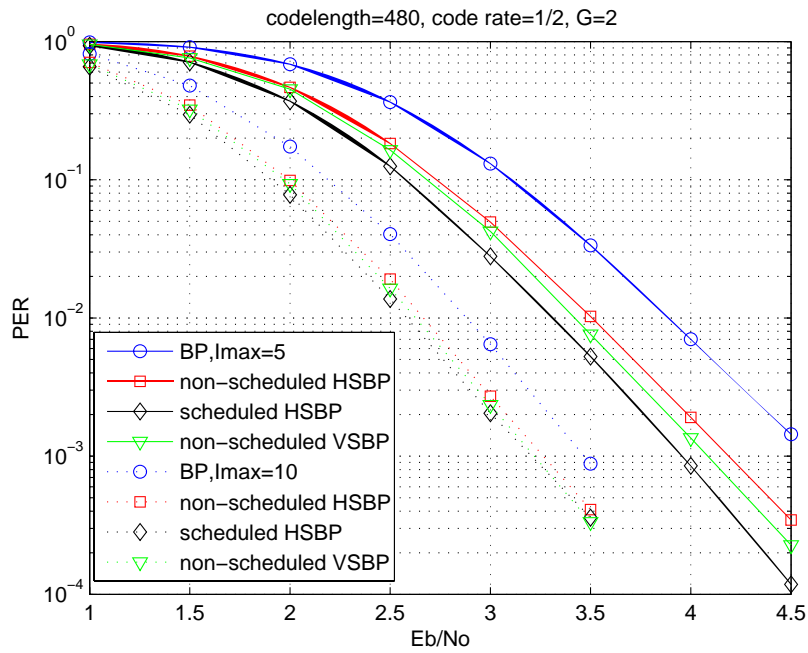


Figure 6.2: Package error rate performance the (480, 240) QC-LDPC code with  $I_{max} = 5, 10$  and  $G=2$ .

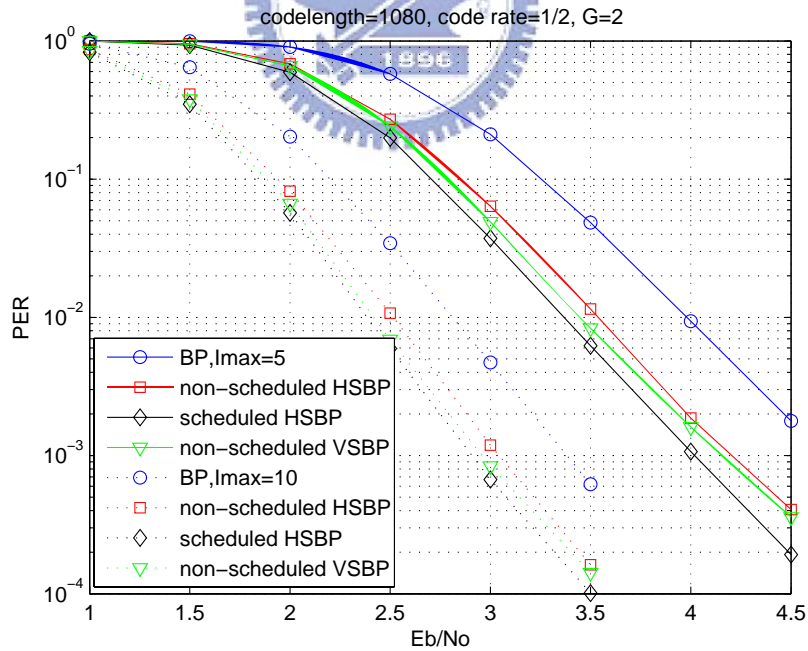


Figure 6.3: Package error rate performance of the (1080, 540) QC-LDPC code with  $I_{max} = 5, 10$  and  $G=2$ .

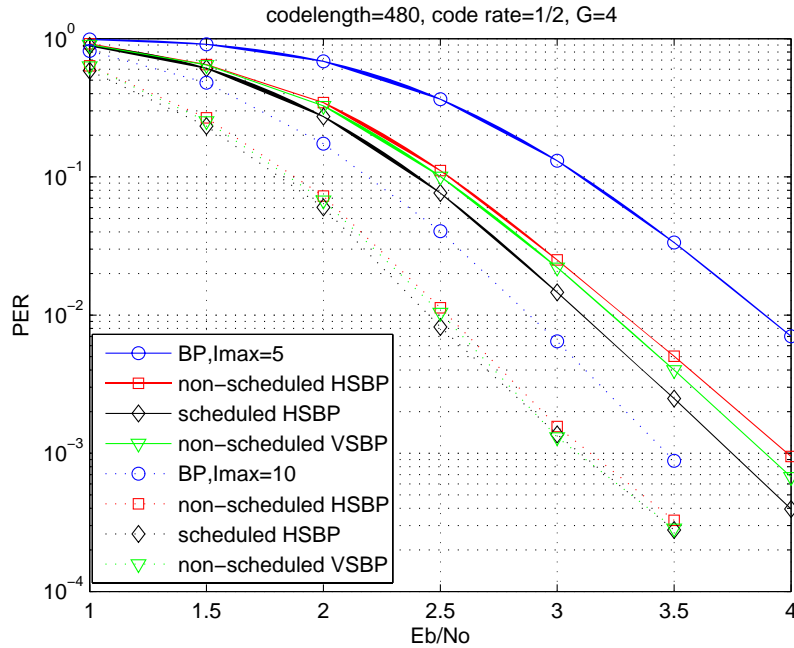


Figure 6.4: Package error rate performance of the (480,240) QC-LDPC code with  $I_{max} = 5, 10$  and  $G=4$ .

CB-HSBP schedule outperforms the others. Besides, the PER performance of the C-VSBP schedule is still better than that of the C-BP schedule. It is clear that although with a larger  $G$  the cycle effects might become more severe, the proposed schedule still maintain its effectiveness in combating the short cycle effect. From Figs. 6.1-6.5, we find that, for code length 480 and 1080,  $G = 4$  always offer a better PER performance and there is a 0.1-0.3 dB gain at  $PER = 10^{-3}$ . These results show that a larger number of node groups benefits the CB-HSBP schedules in both PER and convergence behaviors.

## 6.2 Profile Comparison

To investigate the convergence behavior of various schedules based on the corresponding profile behaviors, we define the normalized concentration ratio (NCR) by  $\gamma_c = (\sigma_n(i)/\mu_n(i))^{-1}$  which measures the variation of the profile, i.e., whether the composition of an incoming message is widely spread across all nodes or it is concentrated

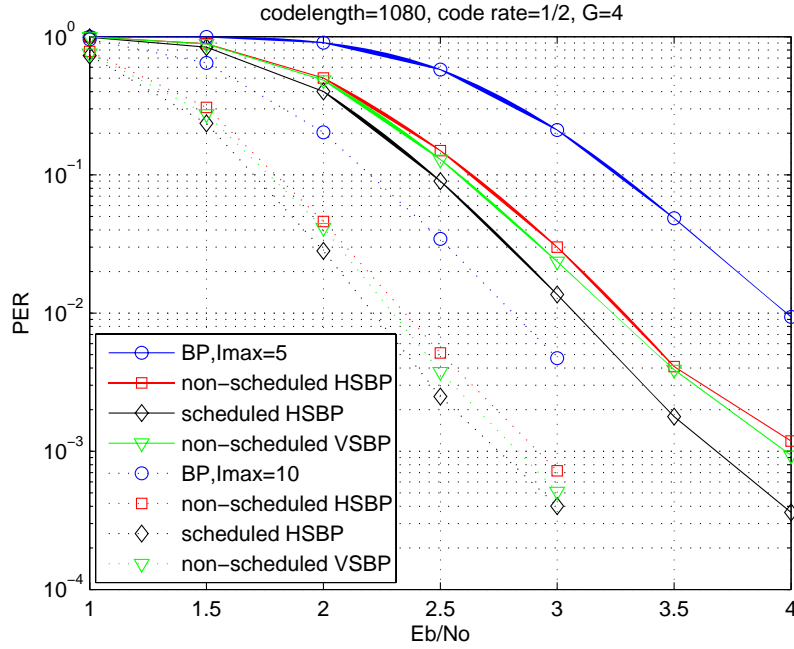


Figure 6.5: Package error rate performance of the (1080, 480) QC-LDPC code with  $I_{max} = 5, 10$  and  $G=4$ .

within a few nodes only. Shown in Figs. 6.6, 6.7, 6.8 and 6.9 are the NCR behaviors of various BP schedules in decoding the (480, 240) QC-LDPC code. We compare Figs. 6.7, 6.8 and 6.9 with Fig. 6.6 and notice that the C-BP and C-VSBP schedules has slower convergence rates but their steady state values are larger than those of various serial schedules with  $G = 2$  except for the C-VSBP schedule. Therefore, the CB-HSBP schedule would outperform the other schedules when the number of iteration is small. Similarly, the NCR analysis indicates that C-VSBP schedule is better than the C-HSBP schedule and the C-BP gives the worst PER performance.

Figs. 6.19, 6.20 and 6.21 show the profile NCR behaviors of the C- and CB-HSBP, and the C-VSBP algorithms with  $G = 4$  for decoding the same code. These results show that the NCR of the C-HSBP and C-VSBP schedules with  $G = 4$  converge at the same iteration. Then, we observe the convergent values of the profiles of these two decoding algorithms. The smaller convergent value is that of the C-HSBP algorithm and the other



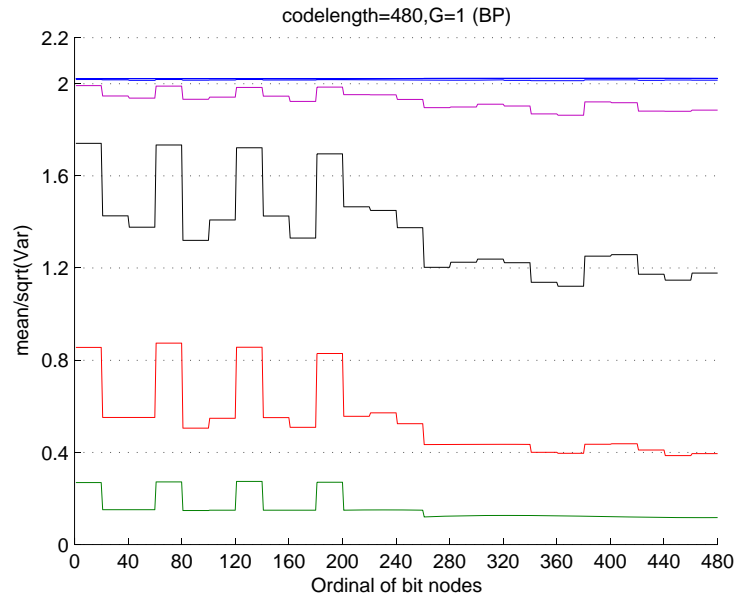


Figure 6.6: Profile NCR behavior of the C-BP schedule for the (480,240) QC-LDPC code.

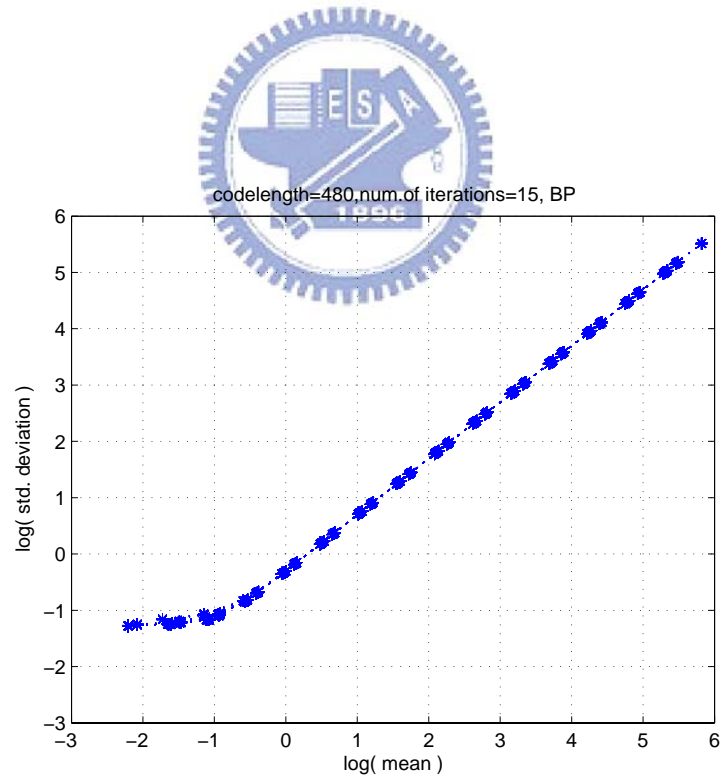


Figure 6.7: Profile NCR behavior of the C-HSBP algorithm with  $G=2$  in decoding the (480,240) QC-LDPC code.

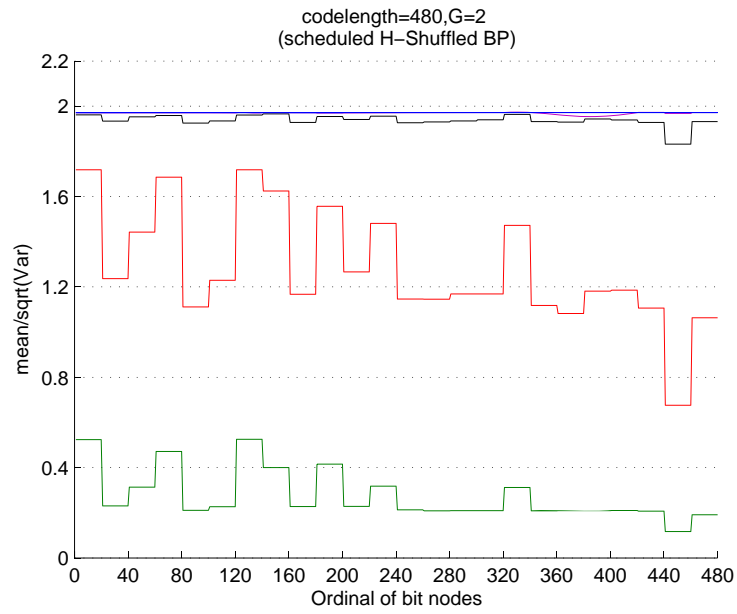


Figure 6.8: Profile NCR behavior of the CB-HSBP algorithm with  $G=2$  in decoding the (480,240) QC-LDPC code.

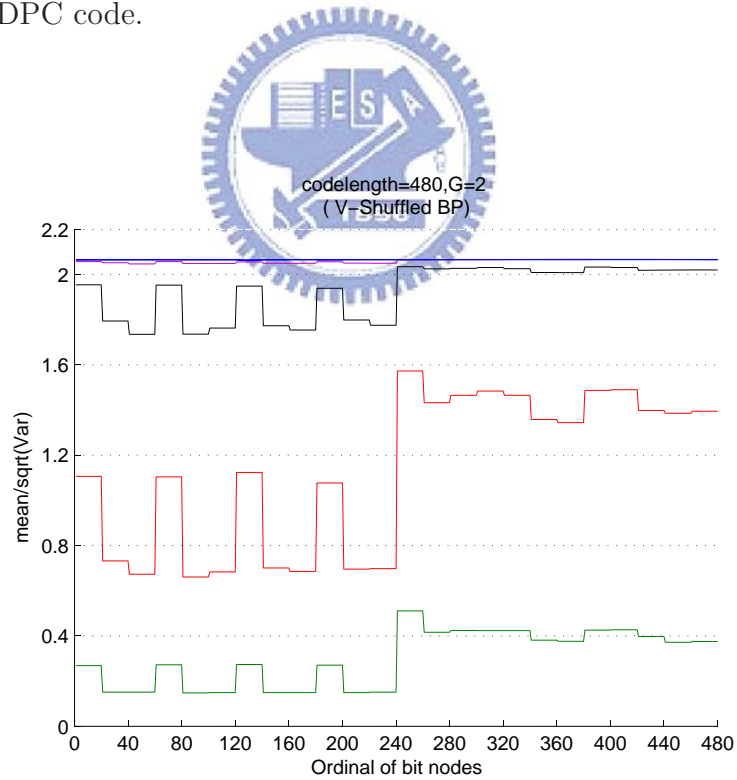


Figure 6.9: Profile NCR behavior of the C-VSBP algorithm with  $G=2$  for the (480,240) QC-LDPC code.

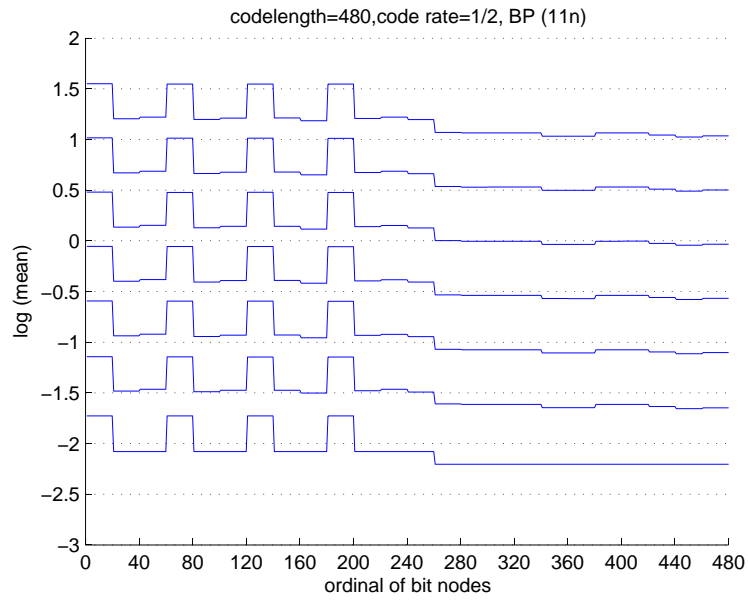


Figure 6.10: Profile mean of the C-BP algorithm in decoding the (480,240) QC-LDPC code.

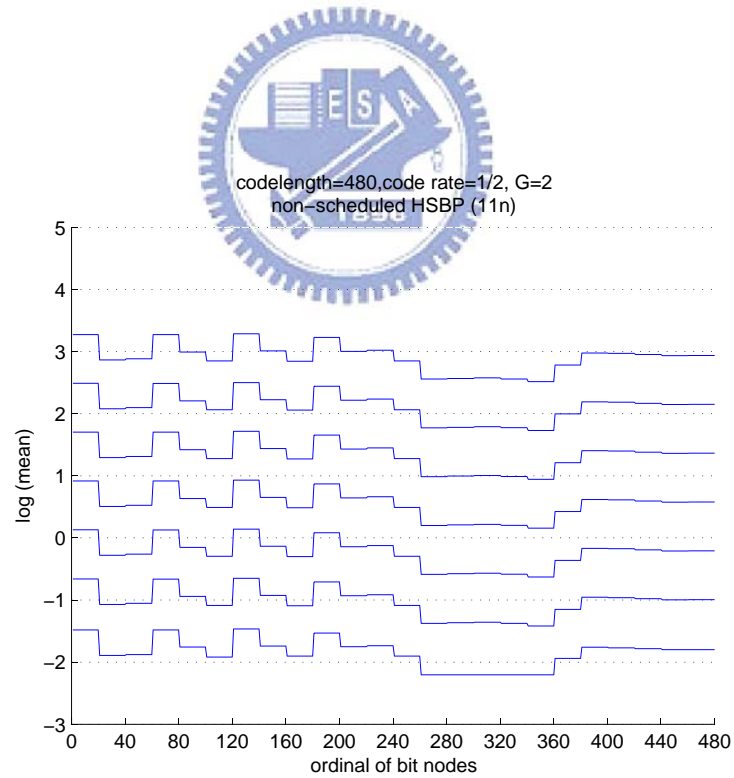


Figure 6.11: Profile mean of the C-HSBP algorithm with  $G=2$  in decoding the (480,240) QC-LDPC code.

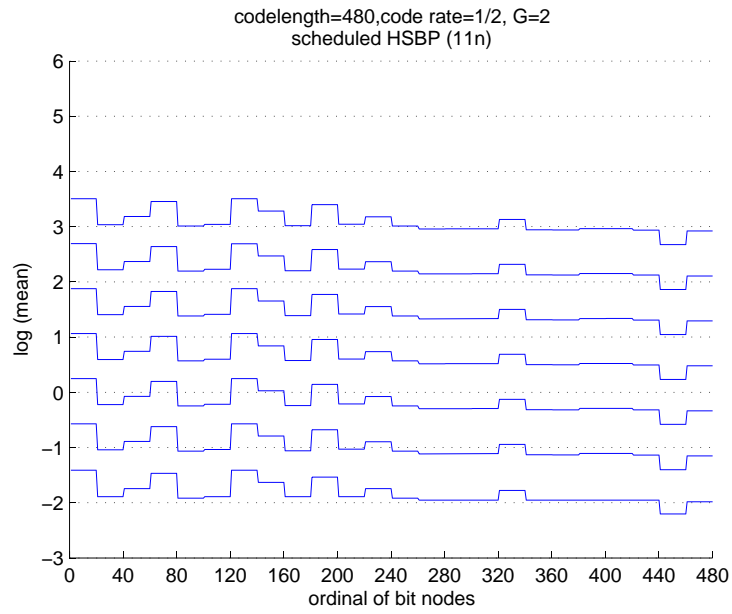


Figure 6.12: Profile mean of the CB-HSBP schedule with  $G=2$  in decoding the (480,240) QC-LDPC code.

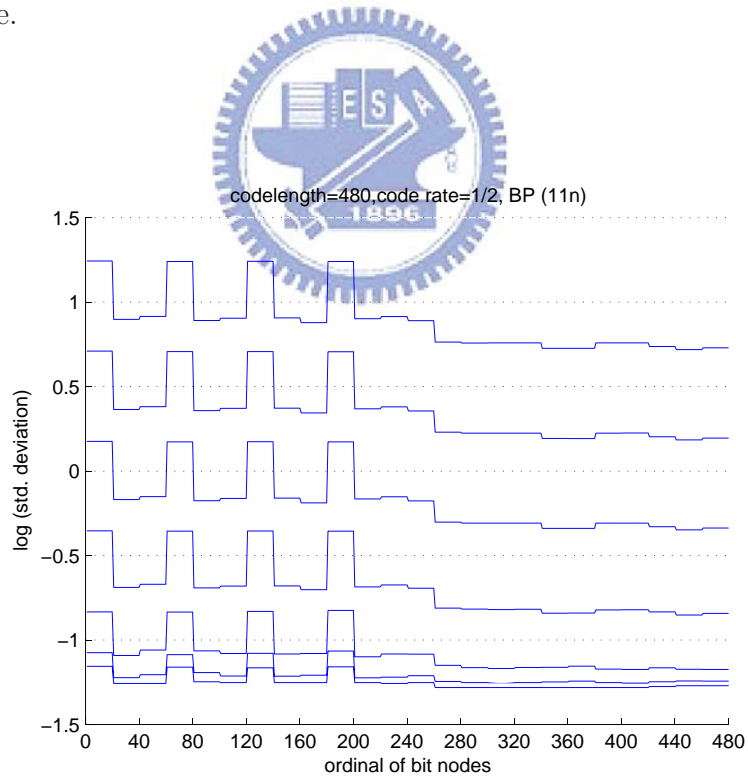


Figure 6.13: Profile spread (standard deviation) of the C-BP algorithm in decoding the (480,240) QC-LDPC code.

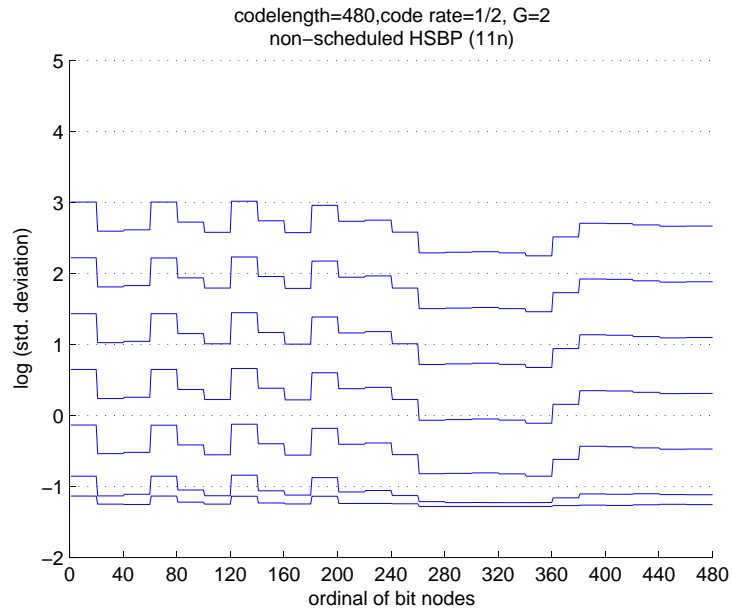


Figure 6.14: Profile spread (standard deviation) of the C-HSBP algorithm with  $G=2$  in decoding the (480,240) QC-LDPC code.

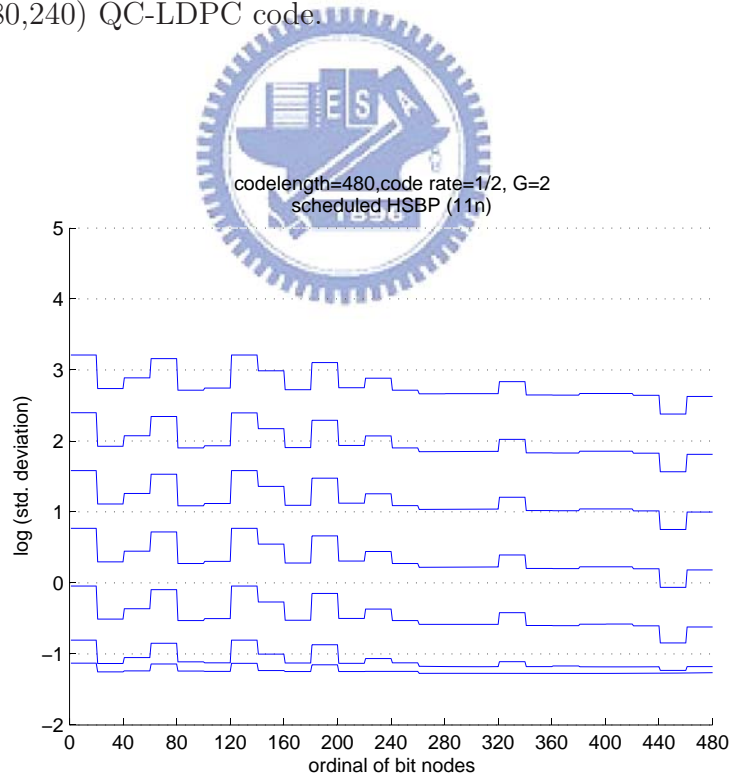


Figure 6.15: Profile spread (standard deviation) behavior of the CB-HSBP algorithm with  $G=2$  for the (480,240) QC-LDPC code.

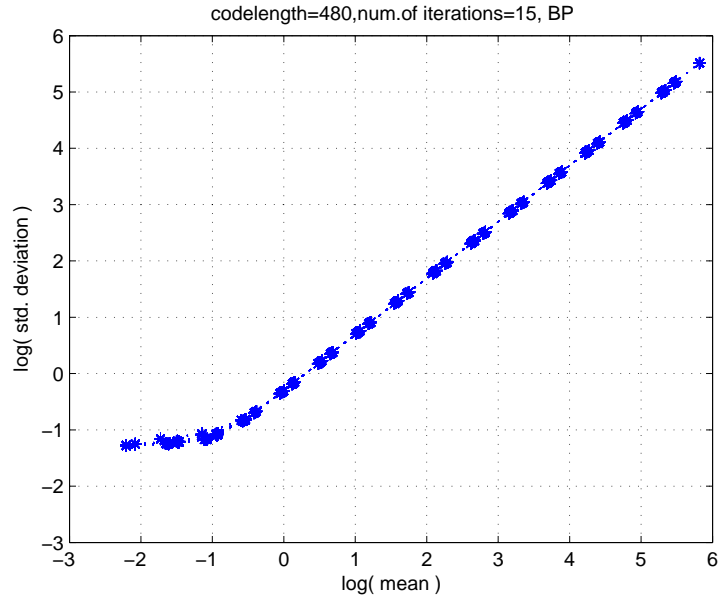


Figure 6.16: Profile mean and spread behaviors of the C-HSBP algorithm with  $G=2$  in decoding the (480,240) QC-LDPC code.

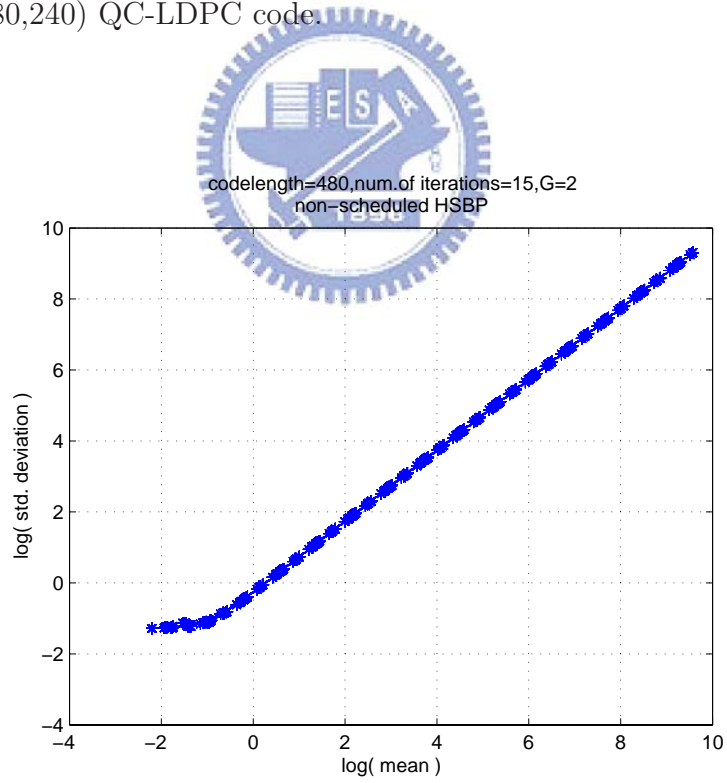


Figure 6.17: Profile mean and spread behaviors of the C-HSBP algorithm with  $G=2$  in decoding the (480,240) QC-LDPC code.

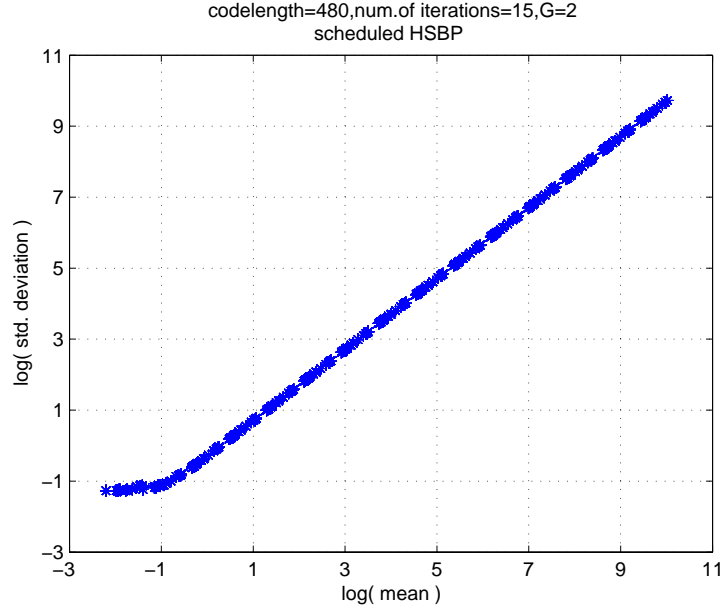


Figure 6.18: Profile mean and spread behaviors of the CB-HSBP algorithm with  $G=2$  for the (480,240) QC-LDPC code.

is that of the C-VSBP algorithm. In our opinion, we regard that the convergent speed of the error rate performance for the C-VSBP algorithm is faster than that for the C-HSBP algorithm. Moreover, the CB-HSBP algorithm requires the fewest number of iterations, so the CB-HSBP algorithm outperforms others. Actually, the simulation results about the error rate performance correlate to those about the profiles of the three decoding schedules.

### 6.3 Comparison between LDPC Codes in 802.11n and 802.16e

We now apply the above NCR analysis to check the performance of two LDPC codes used in two IEEE standards. The two (24,12) QC-LDPC codes are derived from the base model matrices in IEEE 802.11n and IEEE 802.16e, respectively. Since the numerical NCR behaviors of these two LDPC codes indicate that the 16e code has larger steady state value, we predict the code derived from 802.16e outperforms that derived from

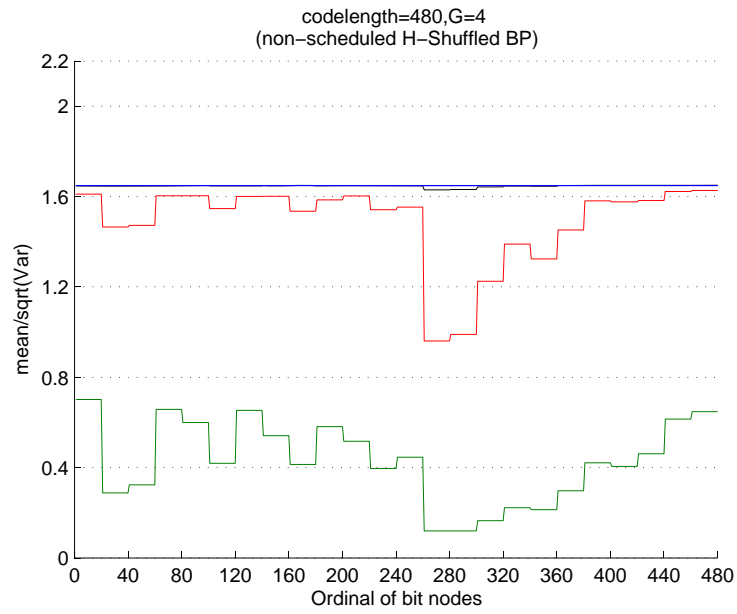


Figure 6.19: Profile NCR behavior of the C-HSBP algorithm with  $G=4$  for the  $(480,240)$  QC-LDPC code.

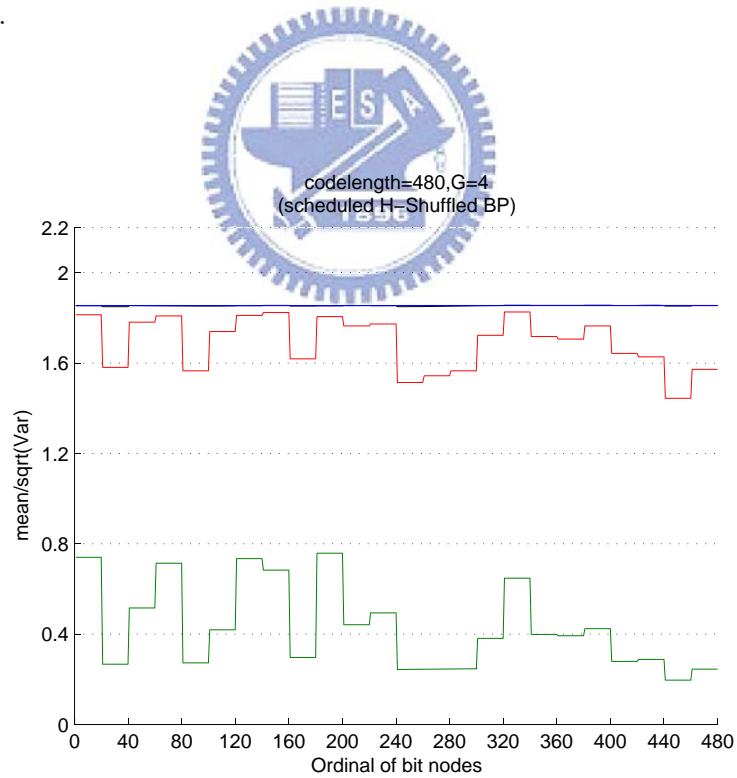


Figure 6.20: Profile NCR behavior of the CB-HSBP algorithm with  $G=4$  for the  $(480,240)$  QC-LDPC code.



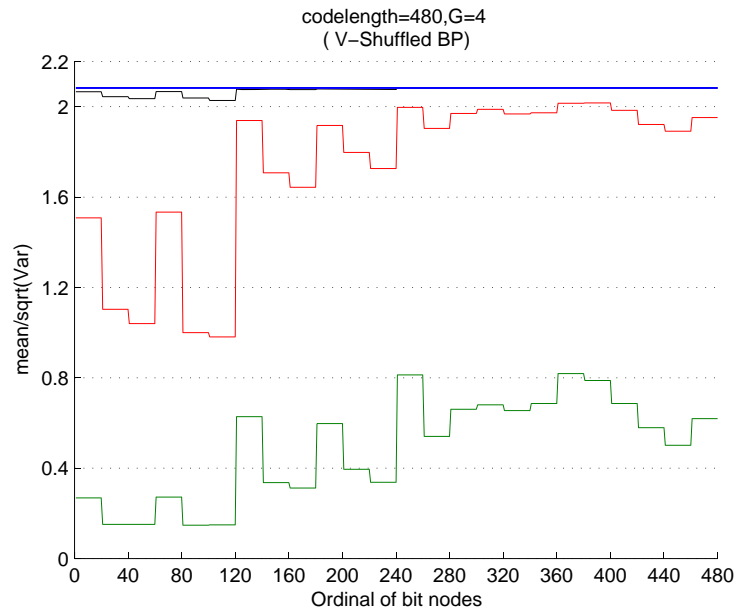


Figure 6.21: Profile NCR behavior of the C-VSBP algorithm with  $G=4$  for the  $(480, 240)$  QC-LDPC code.

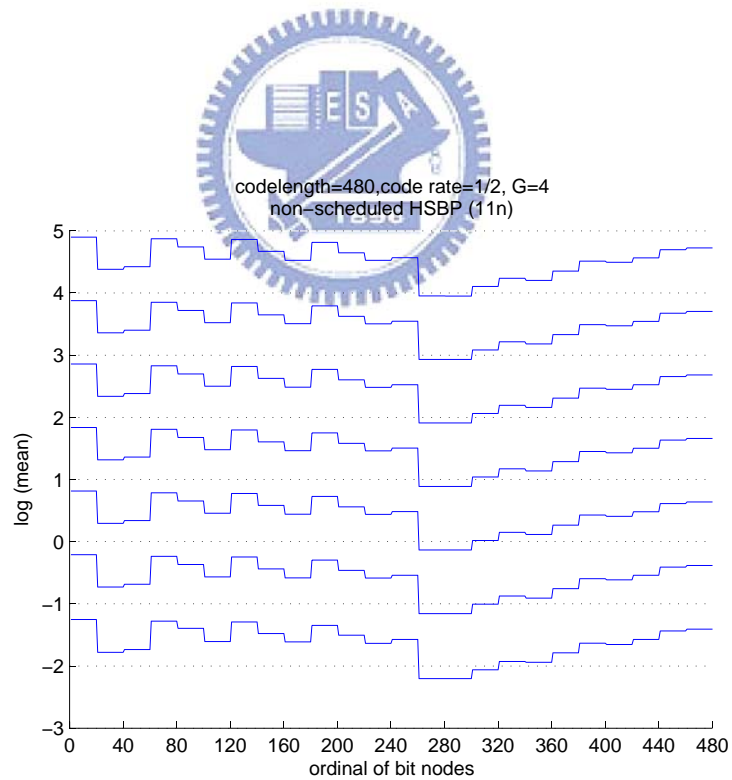


Figure 6.22: Profile mean of the C-HSBP algorithm with  $G=4$  for decoding the  $(480, 240)$  QC-LDPC code.

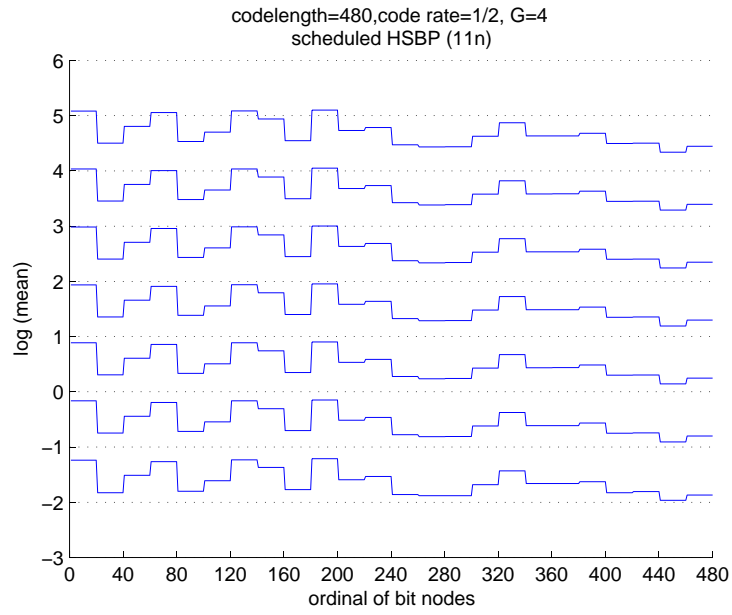


Figure 6.23: Profile mean of the CB-HSBP algorithm with  $G=4$  for the  $(480, 240)$  QC-LDPC code.

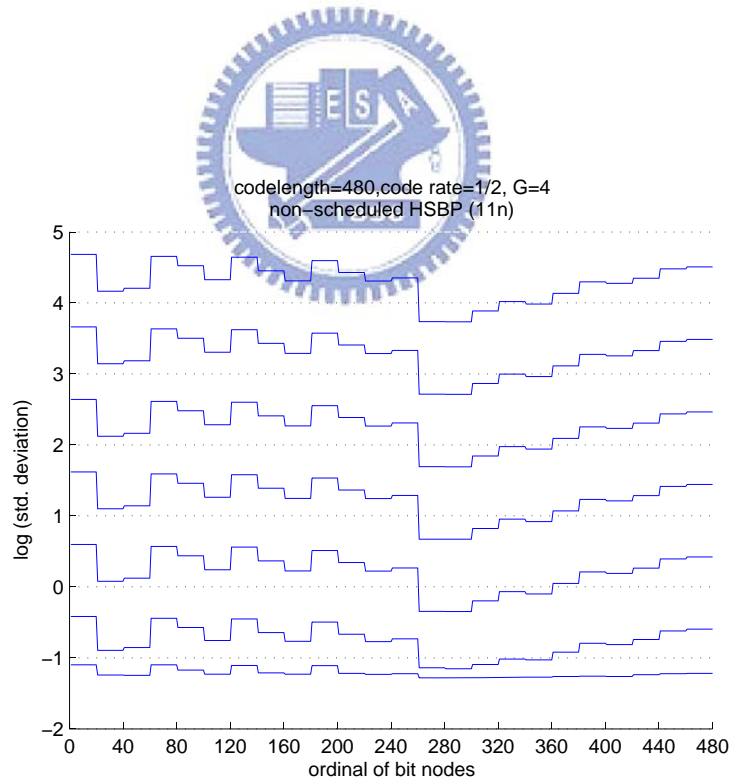


Figure 6.24: Profile spread of the C-HSBP algorithm with  $G=4$  for the  $(480, 240)$  QC-LDPC code.

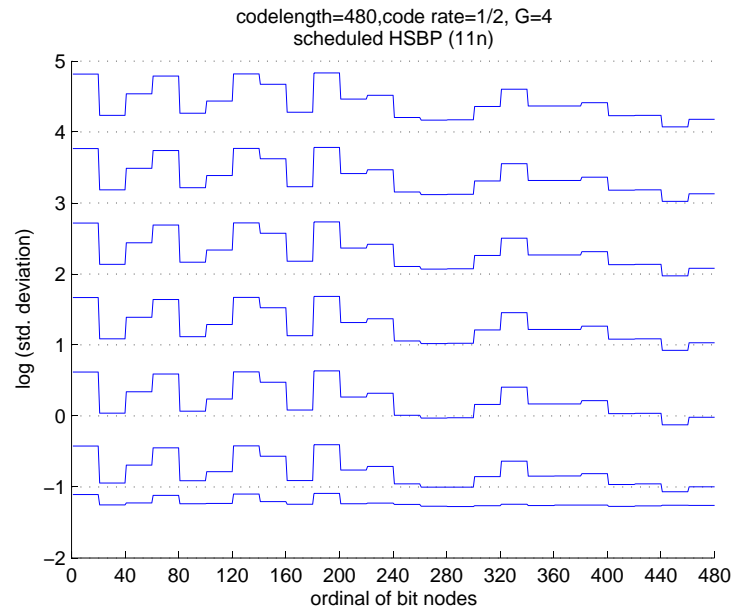


Figure 6.25: Profile spread of the CB-HSBP algorithm with  $G=4$  for the  $(480, 240)$  QC-LDPC code.

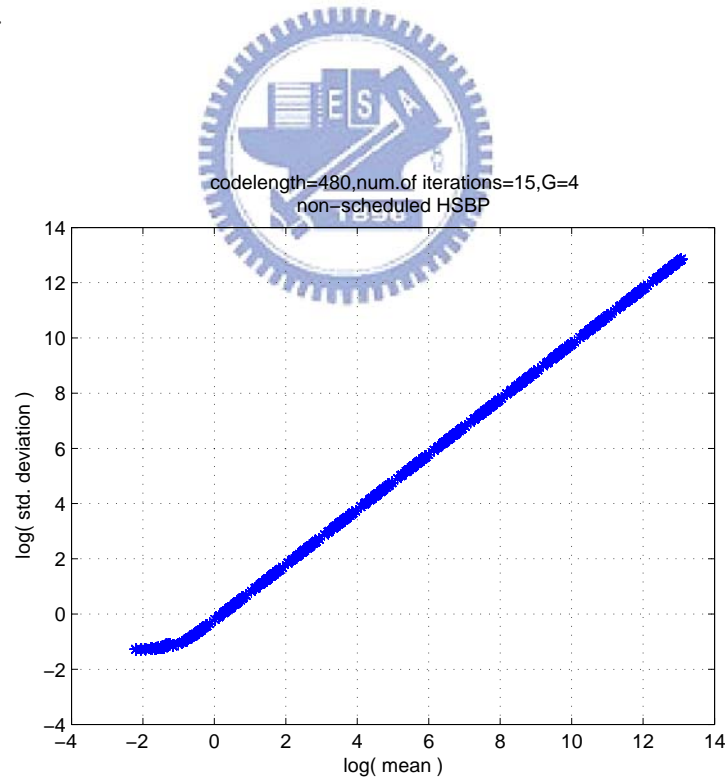


Figure 6.26: Profile mean and spread behaviors of the C-HSBP algorithm with  $G=4$  for the  $(480, 240)$  QC-LDPC code.

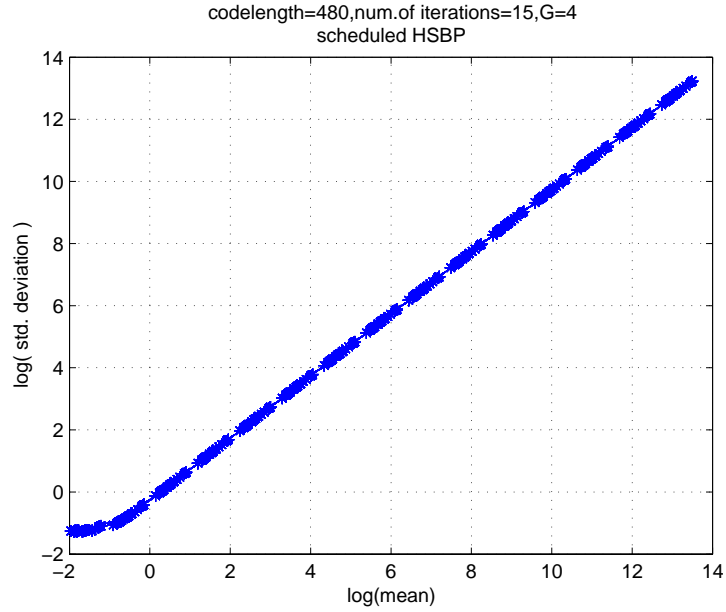


Figure 6.27: Profile mean and spread behaviors of the CB-HSBP algorithm with  $G=4$  for the (480, 240) QC-LDPC code.

802.11n. This prediction is confirmed by Figs. 6.28 and 6.29.

In Fig. 6.30, we examine the performance of two (480,240) QC-LDPC codes derived from expanding the base model matrices in 802.11n and 802.16e, respectively. The results show that the 802.16e code outperforms the 802.11n code. Figs. 6.6 and 6.31 show the profile NCR of the two (480,240) QC-LDPC codes. The steady state value of the former code is smaller than that of the later code. As a result, we predict that the 16e code would surpass the 11n code. This prediction is validated by Fig. 6.30. The above discussion leads us to conclude that one can find a good base model matrix by using the proposed NCR analysis and then expand the base matrix to obtain a QC-LDPC code with the desired code length.

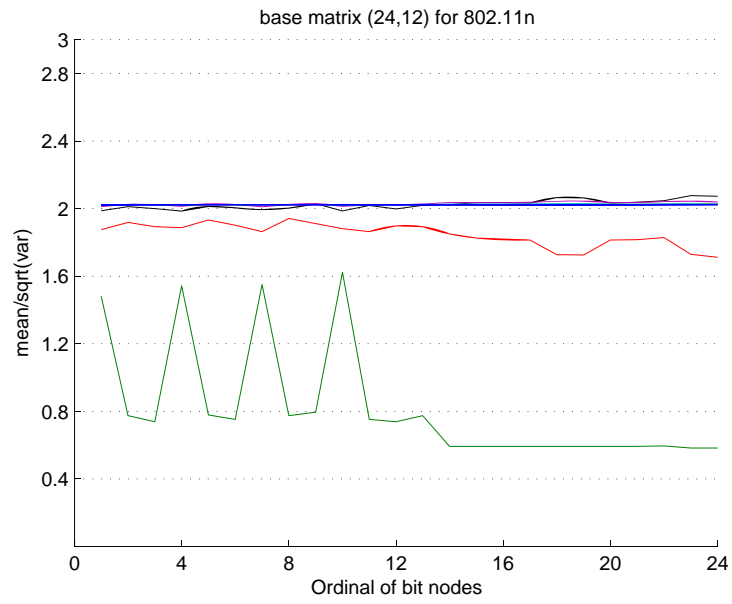


Figure 6.28: Profile NCR behavior of the code derived from the base model matrix of IEEE 802.11n.

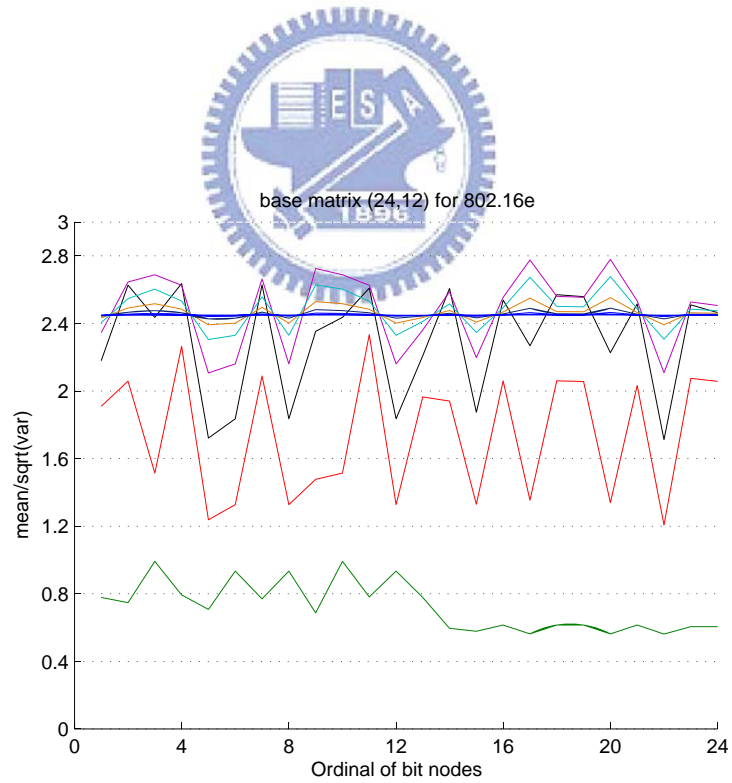


Figure 6.29: Profile NCR behavior of the code derived from the base model matrix of IEEE 802.16e.

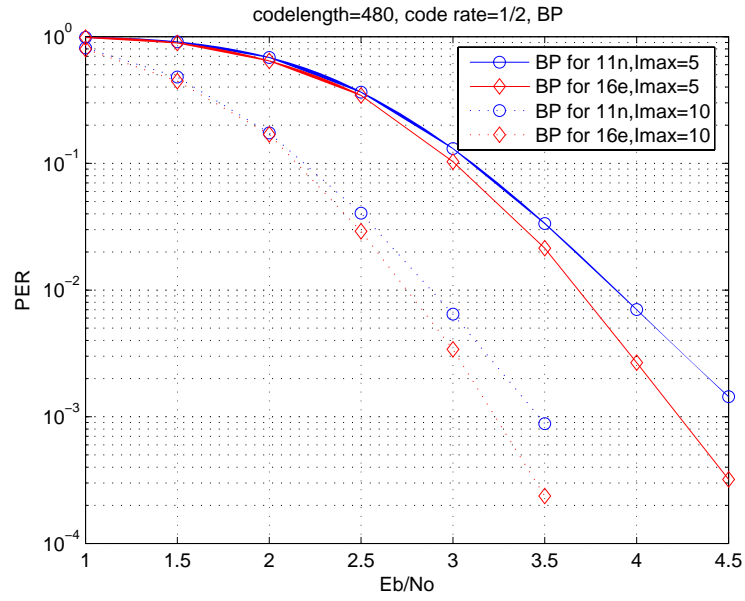


Figure 6.30: The PER performance of the (480,240) QC-LDPC code with the C-BP decoder.

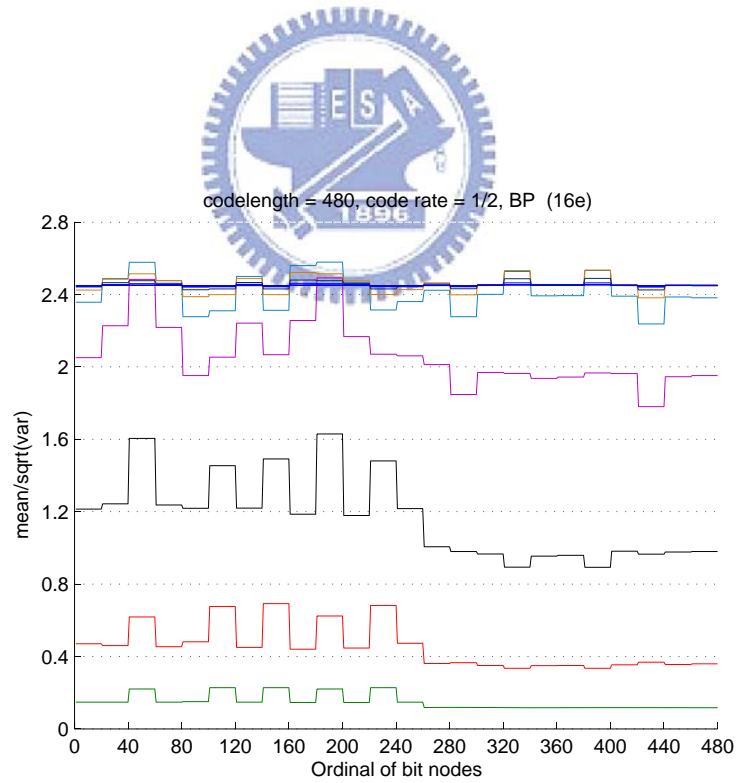


Figure 6.31: Profile NCR behavior of the BP algorithm for the (480,240) QC-LDPC code in 802.16e.

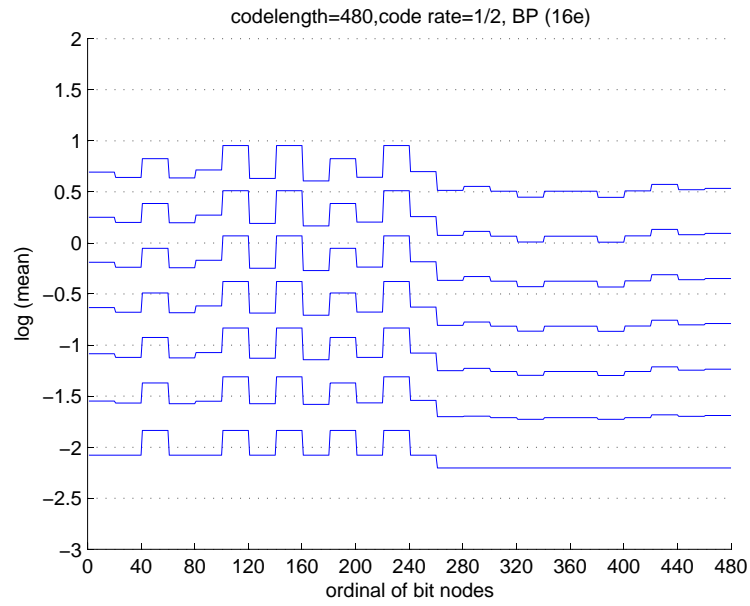


Figure 6.32: Profile mean of the BP algorithm for the (480, 240) QC-LDPC code in 802.16e.

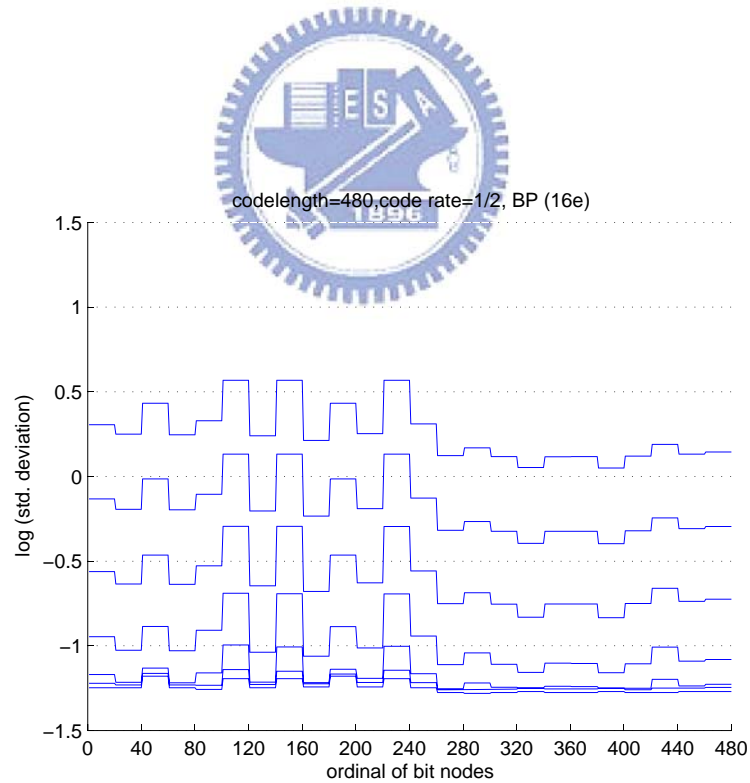


Figure 6.33: Profile spread of the BP algorithm for the (480, 240) QC-LDPC code in 802.16e.

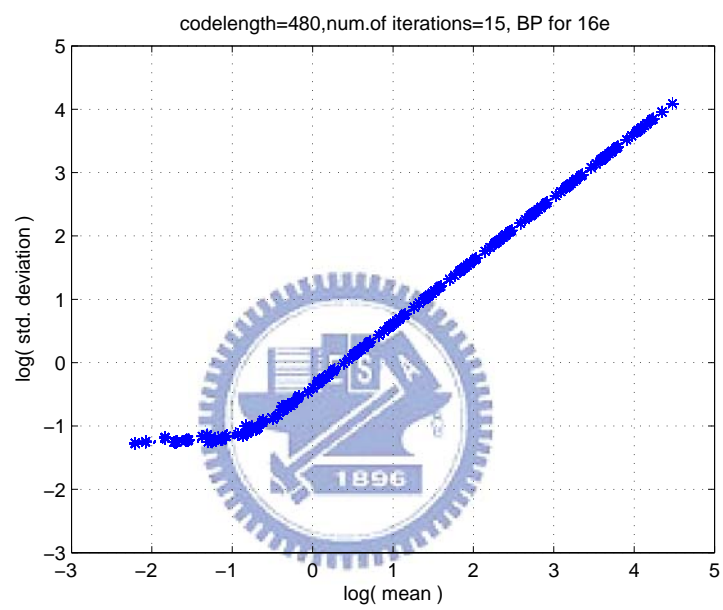


Figure 6.34: Profile mean and spread behaviors of the conventional BP algorithm for the (480, 240) QC-LDPC code in 802.16e.



# Chapter 7

## Conclusions

We propose a simple and systematic method to modify the conventional HSBP and VSBP schedules so that short cycles effect can be minimized. The new schedules are very effective in combating the cycle effects. They achieve improved PER performance with a more rapid convergence rate and reduced computation complexity. We also present a computational efficient framework to analyze the convergence and steady state behaviors of various LDPC decoding schedules. We establish consistent relations between the profile NCR behavior and the PER performance trend. The convergence rate of the NCR is similar to that of the corresponding decoding schedule and the steady state NCR value is a useful indicator for the PER performance. Since the evaluation of a profile NCR is far simpler than error rate performance estimation by computer simulation, the proposed analytic tool can be used to construct good LDPC codes and decoding schedules.

# Bibliography

- [1] J. Pearl, “Reverend Bayes on inference engines: A distributed hierarchical approach,” Proc. American Association of Artificial Intelligence National Conference on AI, Pittsburgh, PA, 133–136, 1982.
- [2] J. H. Kim, and J. Pearl, “A computational model for combined causal and diagnostic reasoning in inference systems,” Proceedings IJCAI-83, Karlsruhe, Germany, 190–193, 1983.
- [3] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, (Revised Second Printing) San Francisco, CA: Morgan Kaufmann.
- [4] Y. Weiss, “Correctness of Local Probability Propagation in Graphical Models with Loops,” *Neural Computation*, 2000.
- [5] R. J. McEliece, D. J. C. MacKay, J.-F. Cheng, ”Turbo decoding as an instance of Pearl’s ”belief propagation” algorithm”, *IEEE JSAC*, vol. 16, no. 2, pp. 140-152, Feb. 1998.
- [6] R. G. Gallager, *Low-density parity-check codes*, Cambridge, MA: MIT Press, 1963.
- [7] D. J.C. MacKay and R. M. Neal, “Near Shannon limit performance of low density parity check codes,” *Electronics Letters*, vol. 33, no. 6, pp. 457-458, Mar. 1997
- [8] F. R. Kschischang, B. J. Frey, H.-A. Loeliger, ”Factor graphs and the sum-product algorithm,” *IEEE Trans. on Inform. Theory*, vol. 47, no. 2, pp. 498-519, Feb. 2001.

- [9] J. Zhang, M. P. C. Fossorier, "Shuffled Iterative Decoding," *IEEE Trans. Commun.* vol. 53, no. 2, pp. 209-213, Feb. 2005.
- [10] Y.-X. Zheng, Y. T. Su, "Decoding inter-block permuted turbo codes based on multi-stage factor graphs," in *Proc. 4th Intern'l Symp. Turbo Codes & Related Topics*, Munich, German, Apr. 2006.
- [11] M. M. Mansour and N. R. Shanbhag, "High throughput LDPC decoders," *IEEE Trans. VLSI Systems*, vol. 11, pp. 976-996, Dec. 2003.
- [12] Hua Xiao and Amir H. Banihashemi, "Graph-Based Message-Passing Schedules for Decoding LDPC Codes", in *IEEE Trans. Commun.* VOL. 52, NO. 12, pp. 454-458, Sep. 2005.
- [13] M. M. Mansour and N. R. Shanbhag, "Turbo decoder architecture for low-density parity-check codes," in *Proc. Global Telecommun. Conf.*, Nov. 2002, pp. 1383-1388.
- [14] J. Zhang, Y. Wang, M. P. C. Fossorier, J.S. Yedidia, "Replica shuffled iterative decoding," in *Proceedings. ISIT 2005*, pp. 454-458, Sep. 2005.
- [15] "11-04-0889-04-000n-tgnsync-proposal-technical-specification" IEEE 802.11n Contribution, Mar. 2005
- [16] E. Sharon, S. Litsyn, and J. Goldberger, "An efficient message-passing schedule for LDPC decoding," in *Electrical and Electronics Engineers in Israel, 2004. Proceedings*, pp. 223-226, Sept. 2004.
- [17] IEEE 802.16e-04/78:Optional B-LDPC coding for OFDMA PHY.

# Appendix A

## Specification of IEEE 802.11n [15]

The parity check matrices  $H$  of the encoding procedure are derived from one of the base parity check matrices  $H_b$ . In 802.11n specification, specified below. One base model matrix is defined per code rate. Size of a base parity check matrix is denoted as  $M_b \times N_b$ .  $N_b$ , the number of columns in the base model matrix, is fixed for all code rates,  $N_b = 24$ .  $M_b$ , the number of rows in the base model matrix, depends on the code rate as follows:  $M_b = N_b \times (1 - R)$ . Parity check matrix  $H$  of size  $M \times N$  is generated by expanding the base model matrix for the selected rate,  $H_b$ ,  $z$ -times:  $z = N/N_b = M/M_b$ . The expansion operation is defined by element values of the base model matrix. Each non-negative base model matrix element,  $s$ , is replaced by a  $z \times z$  identity matrix,  $I_z$ , cyclically shifted to the right  $s' = smod(z)$  times. Each negative number (-1) in the base model matrix is replaced by a  $z \times z$  zero matrix,  $0_{z \times z}$ . For the codeword of size 576 bits,  $z = 24$  and for the codeword of size 1728 bits,  $z = 72$ .

We use the following base model matrix for code rate = 1/2 to derive each LDPC code used in our research.

- base model matrix specification:

Code rate 1/2:  $M_b \times N_b = 12 \times 24$



# Appendix B

## Specification of IEEE 802.16e [17]

The LDPC code is based on a set of one or more fundamental LDPC codes. Each of the fundamental codes is a systematic linear block code. Using the described methods in section 8.4.9.2.5.3 [17] Code Rate and Block Size Adjustment, the fundamental codes can accommodate various code rates and packet sizes. Each LDPC code in the set of LDPC codes is defined by a matrix  $H$  of size  $m - by - n$ , where  $n$  is the length of the code and  $m$  is the number of parity check bits in the code. The number of systematic bits is  $k = n - m$ . The matrix  $H$  is defined as:

$$H = \begin{pmatrix} P_{0,0} & P_{0,1} & P_{0,2} & \dots & P_{0,n_b-2} & P_{0,n_b-1} \\ P_{1,0} & P_{1,1} & P_{1,2} & \dots & P_{1,n_b-2} & P_{1,n_b-1} \\ P_{2,0} & P_{2,1} & P_{2,2} & \dots & P_{2,n_b-2} & P_{2,n_b-1} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ P_{m_b-1,0} & P_{m_b-1,1} & P_{m_b-1,2} & \dots & P_{m_b-1,n_b-2} & P_{m_b-1,n_b-1} \end{pmatrix}$$

where  $P_{i,j}$  is one of a set of  $z - by - z$  permutation matrices or a  $z - by - z$  zero matrix.

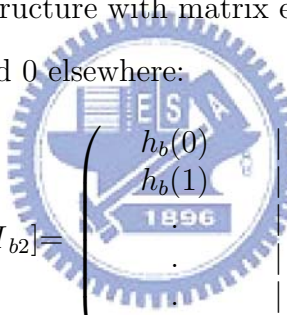
The matrix  $H$  is expanded from a binary base model matrix  $H_b$  of size  $m_b - by - n_b$ , where  $z$  is an integer  $\geq 1$ . The base model matrix is expanded by replacing each 1 in the base model matrix with a  $z - by - z$  permutation matrix, and each 0 with a  $z - by - z$  zero matrix. The base model matrix size  $n_b$  is an integer equal to 24.

The permutations used are circular right shifts, and the set of permutation matrices contains the  $z \times z$  identity matrix and circular right shifted versions of the identity matrix. Because each permutation matrix is specified by a single circular right shift, the

binary base model matrix information and permutation replacement information can be combined into a single compact model matrix  $H_{bm}$ . The model matrix  $H_{bm}$  is the same size as the binary base model matrix  $H_b$ , with each binary entry  $(i,j)$  of the base model matrix  $H_b$  replaced to create the model matrix  $H_{bm}$ . Each 0 in  $H_b$  is replaced by a blank or negative value (e.g., by  $-1$ ) to denote a  $z \times z$  all-zero matrix, and each 1 in  $H_b$  is replaced by a circular shift size  $p(i, j) \geq 0$ . The model matrix  $H_{bm}$  can then be directly expanded to  $H$ .

$H_b$  is partitioned into two sections, where  $H_{b1}$  corresponds to the systematic bits and  $H_{b2}$  corresponds to the parity-check bits, such that  $H_b = [(H_{b1})_{m_b \times k_b} \mid (H_{b2})_{m_b \times m_b}]$ .

Section  $H_{b2}$  is further partitioned into two sections, where vector  $h_b$  has odd weight, and  $H_{b2}$  has a dualdiagonal structure with matrix elements at row  $i$ , column  $j$  equal to 1 for  $i = j$ , 1 for  $i = j + 1$ , and 0 elsewhere:



$$H_{b2} = [h_b \mid H_{b2}] = \left( \begin{array}{c|cccc} h_b(0) & 1 & \cdot & \cdot & \cdot \\ h_b(1) & 1 & 1 & \cdot & 0 \\ \cdot & \cdot & 1 & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 0 & 1 & 1 \\ h_b(m_b - 1) & \cdot & \cdot & \cdot & 1 \end{array} \right)$$

The base model matrix has  $h_b(0) = 1$ ,  $h_b(m_b - 1) = 1$ , and a third value  $h_b(j)$ ,  $0 < j < (m_b - 1)$  equal to 1. The base model matrix structure avoids having multiple weight  $-1$  columns in the expanded matrix.

In particular, the non-zero sub-matrices are circularly right shifted by a particular circular shift value. Each 1 in  $H_{b2}$  is assigned a shift size of 0, and is replaced by a  $z \times z$  identity matrix when expanding to  $H$ . The two located at the top and the bottom of  $h_b$  are assigned equal shift sizes, and the third 1 in the middle of  $h_b$  is given an unpaired shift size.

