

國立交通大學

電信工程學系

碩士論文

低密度校正檢測碼用於正交頻域多工系統之
能量-載波分配與解碼排程

Power/subcarrier allocation and decoding
schedule for LDPC coded OFDM systems

研究生：顏佐翰

指導教授：蘇育德

中華民國九十七年九月

低密度校正檢測碼用於正交頻域多工系統之能量-載波
分配與解碼排程

Power/subcarrier allocation and decoding schedule for
LDPC coded OFDM systems

研 究 生：顏佐翰

Student：Johann Yan

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 論 文

A Thesis

Submitted to Department of Communication Engineering

College of Electrical and Computer Engineering

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Communication Engineering

September 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年九月

低密度校正檢測碼用於正交頻域多工系統之能量-載波分配與解碼排程


學生：顏佐翰

指導教授：蘇育德

國立交通大學

電信工程學系碩士班

摘 要



現今低密度校驗碼的錯誤更正技巧與謝農極限已經能相當接近。信賴度傳輸這種演算法用於低密度校驗碼的解碼上，是相當有效的。但是、對於某些低密度校驗碼的架構上的使用，傳統的信賴度傳輸演算法卻無法提供一個有效率的解法。所以我們提供兩種分群的方法來提升效能，第一個方法是基於我們能有效收集到的獨立信號而定，第二個方法是基於抵抗錯誤能力的不同而分群。藉著這兩種方法，我們可以針對不同的使用方式而得到益處。

Power/subcarrier allocation and decoding schedule for LDPC coded OFDM systems

Student : Johann Yan Advisor : Yu T. Su

Institute of Communication Engineering
National Chiao Tung University

Abstract

The low density parity check (LDPC) codes are powerful error-correcting codes that, thanks in part to the belief propagation (BP) decoding algorithms, offers near-Shannon Limit performance when the code length is sufficiently (but finite) long. The BP algorithms refer to a class of iterative algorithms that passes probabilistic (reliability) messages on a graph that describes the probabilistic (Markovian) relations amongst the associated random variables. With proper message-updating rules and message-passing schedule, a BP algorithm can efficiently compute the a posterior probability (APP) or likelihood function needed in maximum likelihood (ML) or APP decoding.

Two parallel-serial decoding algorithms, namely, the horizontal shuffled BP (HSBP) and the vertical shuffled BP (VSBP) algorithms, have been proposed in the literature. They partition either 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 in group-by-group manner. Three design issues for the resulting parallel-serial decoder arise: the degree of parallelism (the cardinality of a group), the partition rule (which nodes should be in the same group)


and the associated message-passing schedule. All of these three design concerns affect the decoder complexity, convergence speed and the error rate performance.

The basic per-sub-iteration message-passing behavior of a shuffled BP algorithm is determined by the corresponding submatrices of the parity check matrix H . That of a shuffled BP using a partition which is not based on the natural order can be described by a permuted version of the original H . An all-zero column (row) in a sub-matrix implies that the corresponding nodes will undergo no information update. It is thus desired that there be as few all-zero columns (rows) in a sub-matrix as possible.

We present two partition criteria; the first criterion is based on the innovation rate (new uncorrelated information collected per sub-iteration) while the second one is based on the bit nodes' normalized correlation spreads (NCS) which are used to measure the degree of local flooding uniformity of a bit node in each iteration. As the NCS also reveals the unequal error protection (UEP) nature of an irregular LDPC code, the second partition actually divides the bit nodes into groups with different error rate performance. In a multicarrier transmission system, such a UEP property can be exploited to improve the overall performance by using a proper power and subcarrier allocation in carrying out BP decoding. Numerical simulation indicates that both approaches yield improved error rate and convergence performance.

致謝

兩年來的研究生生活，非常感謝的得到很多人的幫助與鼓勵，也由於大家的幫助與鼓勵，才有今天這本論文的完成。首先我要感謝蘇育德老師，跟老師學習的兩年間，不只是學術上的心得，更重要的是待人處事的啟發，令我受益良多。我相信老師兩年的教導，我將受用一輩子。



再來要感謝我的家人，在我就學期間不斷的支持與鼓勵，陪著我走過每一段低潮。其次要感謝論文撰寫期間，一起並肩作戰的伙伴們，易霖、郁文、千雅，這一路上的大家，讓我有繼續拼下去的動力。最後要感謝一路陪我走來的好朋友們，811 研究室的學弟妹小石、Max、Leodo、Nuts、鄭瑩、Tofar。學長：小明、人仰、坤昌、彥志、家偉。和大學時期的好朋友們，謝謝你們的支持與友情贊助，讓我一次又一次的衝過論文瓶頸。

謝謝大家。

Contents

English Abstract	i
Contents	iii
List of Figures	v
1 Introduction	1
2 Low-Density Parity-Check Code	2
2.1 Concept of factor graph	2
2.2 Constraint node of factor graph	3
2.3 Belief propagation	4
2.3.1 Message from check node to variable node	5
2.3.2 Message from variable node to check node	7
3 Shuffled Iterative decoding	9
3.1 Vertical shuffled belief propagation	9
3.2 Horizontal shuffled belief propagation	12
4 Group scheduling methods for shuffled belief propagation algorithm	15
4.1 Message passing for several sub-iterations	15
4.2 Group schedule method	18
4.3 Difference between schedule of VSBP and HSBP	20
4.4 Performance gain divination from EXIT chart	21

5	Message flow distribution and noise resisting	27
5.1	Message flow for BP algorithm	27
5.2	Comparing EXIT chart to message flow	31
5.3	Power allocation based on message flow	32
5.4	Channel mapping based on message flow	33
6	Simulation result	36
6.1	Error rate performance comparison	36
6.2	Message flow and distinct performance of different bits	39
6.3	Power allocation and channel mapping	45
7	Conclusion	51
A	Structure for a (96,48) LDPC code	54
B	Base matrix of IEEE 802.11n and 802.16e	55



List of Figures

2.1	A factor graph for $f_A(x_1, x_4)f_B(x_4)f_C(x_2, x_3)f_D(x_4, x_5)$	3
2.2	(a) is equality constraint node and (b) is zero-sum constraint node	3
2.3	Tanner graph of example 2	4
2.4	Idea of message passing in check node	6
2.5	Idea of message passing in variable node	7
3.1	parallel architecture	10
3.2	serial architecture	10
3.3	multi-stage factor graph representation of VSBP algorithm	11
3.4	multi-stage factor graph representation of HSBP algorithm	13
4.1	MSBP factor graph of (8,4) LDPC code	17
4.2	Statistics for (480,240) QC-LDPC code generated by 802.11n specification with traditional BP with transmitted $x=1$	21
4.3	Statistics for (480,240) QC-LDPC code generated by 802.11n specification with VSBP, $G=6$ with transmitted $x=1$	22
5.1	message flow for example 7	30
5.2	model of a multipath channel	34
5.3	a channel response example of frequency selective channel	35
6.1	BER performance of the (480,240) QC-LDPC code with $I_{max}=1$, VSBP $G=2,4,6$	37

6.2	BER performance of the (480,240) QC-LDPC code with $I_{max}=3$,VSBP	
	G=2,4,6.	38
6.3	BER performance of the (480,240) QC-LDPC code with $I_{max}=1$,HSBP	
	G=2,4.	39
6.4	BER performance of the (480,240) QC-LDPC code with $I_{max}=3$,HSBP	
	G=2,4.	40
6.5	FER performance of the (480,240) QC-LDPC code with $I_{max}=5,10$, G=2.	40
6.6	FER performance of the (480,240) QC-LDPC code with $I_{max}=5,10$, G=4.	41
6.7	Message flow behavior of the (480,240) 802.11n specification QC-LDPC	
	code.	41
6.8	BER performance of different bits for (480,240) 802.11n specification QC-	
	LDPC code.	42
6.9	Message flow behavior of the (480,240) 802.16e specification QC-LDPC	
	code.	43
6.10	BER performance of different bits for (480,240) 802.16e specification QC-	
	LDPC code.	43
6.11	Message flow behavior of the (96,48) QC-LDPC code.	44
6.12	BER performance of different bits for (96,48) QC-LDPC code.	44
6.13	BER of 802.11n specification QC-LDPC code with power allocation at $I=5$.	45
6.14	magnitude response for frequency selective channel with small difference	
	of magnitude.	47
6.15	802.11n specification performance for frequency selective channel with	
	small difference of magnitude.	47
6.16	magnitude response for frequency selective channel with large difference	
	of magnitude.	48
6.17	performance for frequency selective channel with large difference of mag-	
	nitude.	49

6.18	magnitude response for frequency selective channel with difference of magnitude between Fig 6.14 and Fig 6.16.	49
6.19	performance for frequency selective channel with difference of magnitude between Fig 6.14 and Fig 6.16.	50
B.1	Base matrix of 802.11n	55
B.2	Base matrix of 802.16e	56



Chapter 1

Introduction

Usually the using of LDPC code always is processed by BP algorithm and conventional BP algorithm does not provide an efficiency method which is based on the character of one code to achieve another performance gain. And now our focal point of this thesis is improving the efficiency of a code by grouping it into several small units to analysis. In chapter 2 we introduce some definition and theorem of LDPC code and chapter 3 shows the idea of shuffled BP and how does shuffled BP can speed-up the convergence.

We will use two kinds of grouping method to enhance our performance. First, in chapter 4 with the concept of shuffled BP, we add another processing which focus on the amount of independent information we can get from a sub-iteration and hope after scheduling, more independent information can be achieved. Second, we employ a simple method to find different noise resisting ability of every bit and then group them. This will be introduced in chapter 5. We still use EXIT chart to verify our assumption, then in chapter 6 shows our simulation result. And the latest chapter shows our conclusion and some future work.

Chapter 2

Low-Density Parity-Check Code

Low-density parity-check (LDPC) code was invented by Gallager in 1962 [1] and the processor of computer was not strong enough to support the complexity computation at that time, so it was ignored until Mackey[2] rediscovered it. LDPC code is a linear block code which is specified by a very sparse parity check matrix with small numbers of 1's and 0's else. The decoding algorithm of LDPC code can be represented by Factor Graph and is proved to achieve a capacity near Shannon limit when block length is large .

2.1 Concept of factor graph

Factor graphs[3][4] are a straightforward generalization of the Tanner graphs and are introduced to describe the families of LDPC codes. Graphical models such as factor graphs support a general trend to iterative processing.

A factor graph is a diagram that represents the factorization of function $g(x_1, \dots, x_n) = \prod_j f_j(X_j)$ where X_j is a subset of x_1, x_2, \dots, x_n and $f_j(X_j)$ is a function having the elements X_j as arguments. A factor graph contains two kinds of node, variable node for each variable x_i and factor node for each local function f_j . There is one edge that connects a variable node x_i with a factor node f_j if x_i is an argument of f_j .

Example 1(A simple factor graph): Let $g(x_1, x_2, x_3, x_4, x_5)$ be a function with 5

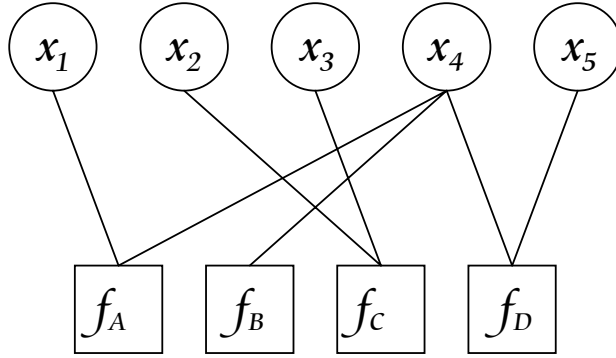


Figure 2.1: A factor graph for $f_A(x_1, x_4)f_B(x_4)f_C(x_2, x_3)f_D(x_4, x_5)$

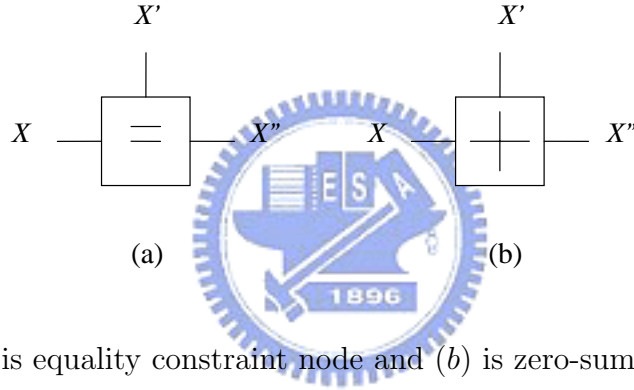


Figure 2.2: (a) is equality constraint node and (b) is zero-sum constraint node

variables ,and then assume that g can be expressed as a product of 5 factors:

$$g(x_1, x_2, x_3, x_4, x_5) = f_A(x_1, x_4)f_B(x_4)f_C(x_2, x_3)f_D(x_4, x_5) \quad (2.1)$$

Where $X_A = \{x_1, x_4\}$, $X_B = \{x_4\}$, $X_C = \{x_2, x_3\}$, $X_D = \{x_4, x_5\}$, and then Fig.2.1 shows the factor graph of equation (2.1).

2.2 Constraint node of factor graph

A factor graph contains two kinds of constraint nodes: equality constraint node and zero-sum constraint node. Show in Fig.2.2:

The local function of equality constraint node is

$$f_=(x, x', x'') \triangleq \delta(x - x')\delta(x - x'') \quad (2.2)$$

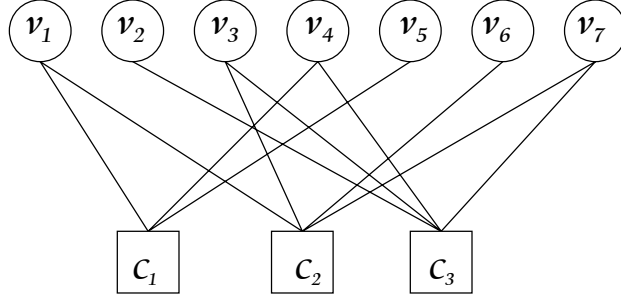


Figure 2.3: Tanner graph of example 2

The local function of zero-sum constraint node is

$$f_+(x, x', x'') \triangleq \delta(x + x' + x'') \quad (2.3)$$

where $\delta(\cdot)$ denote Delda function and $\delta(x) = 1$ when $x = 0$.

Example 2(An example of LDPC code): If we have a parity check matrix

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

If $X = (v_1, v_2, v_3, v_4, v_5, v_6, v_7)$ is a codeword and syndrom is $S = (C_1, C_2, C_3)$, than from the characteristic of linear code we have $HX^T = S^T = [C_1, C_2, C_3]^T = [0 \ 0 \ 0]^T$. The representation of three check nodes is described as

$$C_1 : x_1 \oplus x_4 \oplus x_5 = 0$$

$$C_2 : x_1 \oplus x_3 \oplus x_6 \oplus x_7 = 0$$

$$C_3 : x_2 \oplus x_3 \oplus x_4 \oplus x_7 = 0$$

The Tanner graph of this parity check matrix which has 7 variable nodes and 3 check nodes is shown in Fig.2.3.

2.3 Belief propagation

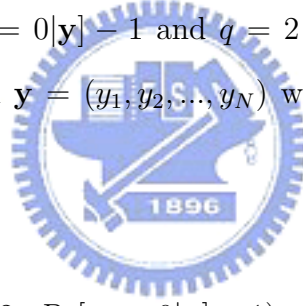
In this section we will introduce the decoding algorithm of LDPC code. The decoding method is called belief propagation, or message passing algorithm which is a iterative

algorithm. The main concept of the algorithm is that the information can be updated by passing current information from variable nodes to check nodes and from check nodes to variable nodes. Information from the variable node n to the check node m is computed by collecting all information which is neighbor to the variable node n except to that connects with check node m , and the computation of information from check node m to variable node n is similar to it.

For the convenience, we define the likelihood of binary random variable x : $L(x)$ as $Pr[x = 0]/Pr[x = 1]$ and conditional likelihood of x : $L(x|\mathbf{y})$ is defined as $Pr[x = 0|\mathbf{y}]/Pr[x = 1|\mathbf{y}]$.

2.3.1 Message from check node to variable node

If we assume $p = 2 \cdot Pr[x_1 = 0|\mathbf{y}] - 1$ and $q = 2 \cdot Pr[x_2 = 0|\mathbf{y}] - 1$ where x_1, x_2 are binary random variables and $\mathbf{y} = (y_1, y_2, \dots, y_N)$ where y_1, y_2, \dots, y_N are continuous random variables, then



$$\begin{aligned}
pq &= (2 \cdot Pr[x_1 = 0|\mathbf{y}] - 1) \cdot (2 \cdot Pr[x_2 = 0|\mathbf{y}] - 1) \\
&= 4 \cdot Pr[x_1 = 0|\mathbf{y}] \cdot Pr[x_2 = 0|\mathbf{y}] - 2 \cdot Pr[x_1 = 0|\mathbf{y}] - 2 \cdot Pr[x_2 = 0|\mathbf{y}] + 1 \\
&= 2 \cdot Pr[x_1 = 0|\mathbf{y}] \cdot Pr[x_2 = 0|\mathbf{y}] + 2 \cdot Pr[x_1 = 0|\mathbf{y}] \cdot Pr[x_2 = 0|\mathbf{y}] \\
&\quad - 2 \cdot Pr[x_1 = 0|\mathbf{y}] - 2 \cdot Pr[x_2 = 0|\mathbf{y}] + 1 \\
&= 2 \cdot Pr[x_1 = 0|\mathbf{y}] \cdot Pr[x_2 = 0|\mathbf{y}] + 2 \cdot (1 - Pr[x_1 = 1|\mathbf{y}]) \cdot (1 - Pr[x_2 = 1|\mathbf{y}]) \\
&\quad - 2 \cdot (1 - Pr[x_1 = 1|\mathbf{y}]) - 2 \cdot (1 - Pr[x_2 = 1|\mathbf{y}]) + 1 \\
&= 2 \cdot Pr[x_1 = 0|\mathbf{y}] \cdot Pr[x_2 = 0|\mathbf{y}] + 2 \cdot Pr[x_1 = 1|\mathbf{y}] \cdot Pr[x_2 = 1|\mathbf{y}] - 1 \\
&= 2 \cdot Pr[x_1 \oplus x_2 = 0|\mathbf{y}] - 1
\end{aligned}$$

so from above we can extend it as

$$2 \cdot Pr[x_1 \oplus \dots \oplus x_l = 0|\mathbf{y}] - 1 = \prod_{i=1}^l (2 \cdot Pr[x_i = 0|\mathbf{y}] - 1) \quad (2.5)$$

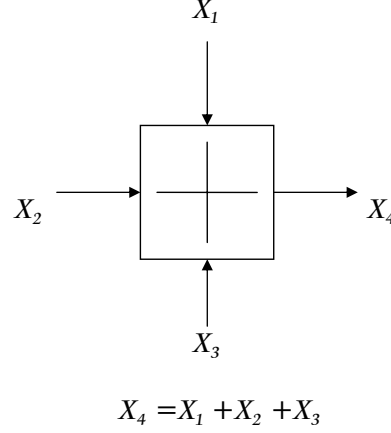


Figure 2.4: Idea of message passing in check node

where x_1, \dots, x_l are all binary random variables. And because

$$\begin{aligned} L(x_i|\mathbf{y})/(1 + L(x_i|\mathbf{y})) &= \frac{Pr[x_i = 0|\mathbf{y}]/Pr[x_i = 1|\mathbf{y}]}{(Pr[x_i = 0|\mathbf{y}]/Pr[x_i = 1|\mathbf{y}]) + 1} \\ &= \frac{1}{1 + (Pr[x_i = 1|\mathbf{y}]/Pr[x_i = 0|\mathbf{y}])} \\ &= \frac{1}{Pr[x_i = 0|\mathbf{y}]} \end{aligned}$$

, the equation $2Pr[x_i = 0|\mathbf{y}] - 1$ can be represented as $(L(x_i|\mathbf{y}) - 1)/(L(x_i|\mathbf{y}) + 1)$. If we define $l = \ln L(x_i|\mathbf{y})$, then $2Pr[x_i = 0|\mathbf{y}] - 1 = \tanh(l/2)$. Therefore we can get

$$\ln L(x_1 \oplus, \dots, \oplus x_l | \mathbf{y}) = \ln \frac{1 + (\prod_{i=1}^l \tanh(l_i/2))}{1 - (\prod_{i=1}^l \tanh(l_i/2))} \quad (2.6)$$

where l_i is $L(x_i|\mathbf{y})$.

From Fig 2.4 we know the information of X_4 is just the sum of X_1, X_2, X_3 and $Pr(X_4) = Pr(X_1 \oplus X_2 \oplus X_3)$, so equation (2.6) can implement the message from the check node to the variable node. For an $M \times N$ parity check matrix $H = [H_{mn}]$, we define 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 $\mathcal{M}(n) = \{m : H_{mn} = 1\}$. If we denote the message from variable node to check node at the $(i-1)$ -th iteration as $z_{mn'}^{i-1}$, then the message from check node to variable node at the i -th iteration ε_{mn}^i can

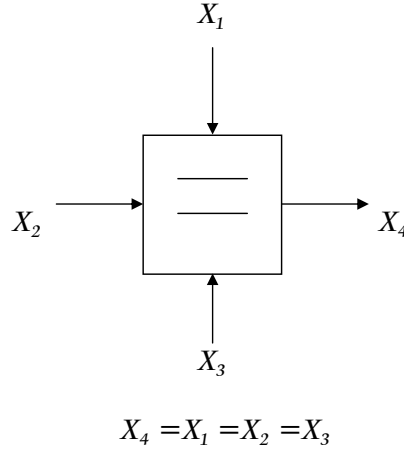


Figure 2.5: Idea of message passing in variable node

be described as

$$\varepsilon_{mn}^i = \ln \frac{1 + \prod_{n' \in \mathcal{N}(m) \setminus n} \tanh(z_{mn'}^{i-1}/2)}{1 - \prod_{n' \in \mathcal{N}(m) \setminus n} \tanh(z_{mn'}^{i-1}/2)}$$

2.3.2 Message from variable node to check node

If $Pr[x_1 = 0|\mathbf{y}] = p$ and $Pr[x_2 = 0|\mathbf{y}] = q$ where x_1, x_2 are binary random variables, the probability $Pr[x_1 = x_2 = 0|\mathbf{y}] = (p \cdot q)/n$ and the probability $Pr[x_1 = x_2 = 1|\mathbf{y}] = (1-p) \cdot (1-q)/n$ where n is a normalization argument to make sum of the two probability be one. And $L(x_1 = x_2|\mathbf{y}) = L(x_1) \cdot L(x_2)$ then $\ln L(x_1 = x_2|\mathbf{y}) = \ln L(x_1) + \ln L(x_2)$. So we extend it to the case with several variable nodes:

$$\ln L(x_1 = x_2 = \dots = x_l) = \ln L(x_1) + \ln L(x_2) + \dots + \ln L(x_l) \quad (2.7)$$

where x_1, \dots, x_l are all binary random variables. From Fig 2.5: $Pr(X_4) = Pr(X_1 = X_2 = X_3)$. Then the information from variable nodes to check nodes can be implemented by equation (2.7). If we denote the check-to-variable message at the i -th iteration as z_{mn}^i ,

then

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

where $F_n = 4 \cdot y_n / N_0$, and the initial value $z_{mn}^0 = F_n$

We summarize detailed BP as below:

1. Initialization: set $i = 1$ and the max number of iteration is I_{max} . For each m, n , initialize $z_{mn}^0 = F_n = 4 \cdot y_n / N_0$.
2. • At check nodes: $\forall m, 1 \leq m \leq M$, and each $n \in \mathcal{N}(m)$

$$\tau_{mn}^i = \prod_{n' \in \mathcal{N}(m) \setminus n} \tanh\left(\frac{z_{mn'}^{i-1}}{2}\right) \quad (2.8)$$

$$\varepsilon_{mn}^i = \ln \frac{1 + \tau_{mn}^i}{1 - \tau_{mn}^i} \quad (2.9)$$

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

$$z_{mn}^i = F_n + \sum_{m' \in \mathcal{M}(n) \setminus m} \varepsilon_{m'n}^i \quad (2.10)$$

$$z_n^i = F_n + \sum_{m' \in \mathcal{M}(n)} \varepsilon_{m'n}^i \quad (2.11)$$

3. Make hard decision z_n^i to obtain a outcome R_n^i with $z_n^i > 0$ if $R_n^i = 0$, else $R_n^i = 1$. If the outcome satisfies $H \cdot (R_n^i)^T = 0$ or achieves the maximum number of iteration I_{max} , then stop operation and choose R_n^i as the decoded codeword. Otherwise set $i = i + 1$ then go to Step 2.

Chapter 3

Shuffled Iterative decoding

The BP algorithm is a multi-iteration parallel procedure that updates information from variable nodes to check nodes and check nodes to variable nodes until achieving max iteration number or the value of syndrome being zero. However the parallel implementation architectures (shown in Fig 3.1) of decoders for these codes require a lot of processing elements and complex interconnection between variable nodes and check nodes [5] due to the sparse structure of the underlying Tanner graph. It is nature to have a serial architecture that serializes and distributes the parallel algorithm among a small number of processing elements (shown if Fig 3.2) and the information between variable nodes and check nodes is saved in memory. Different with the parallel architecture, the computation of information in serial architecture has data dependencies on information and parallel architecture relies on current computation of information. So serial architecture can achieve better error performances and faster convergence than parallel architecture algorithm.

3.1 Vertical shuffled belief propagation

The vertical shuffled belief propagation algorithm [1] [6] splits all the variable nodes into some groups and each full iteration of the decoding process consists of many sub-iterations.

The standard BP algorithm uses all variable-to-check information at $(i - 1)$ -th iter-

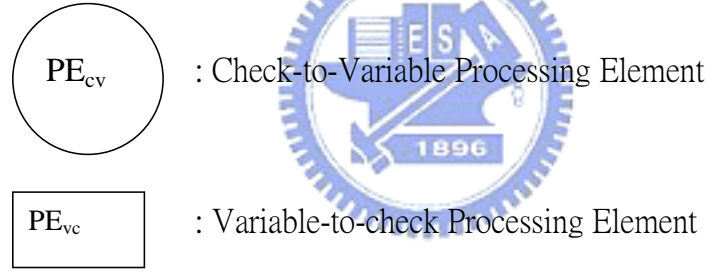
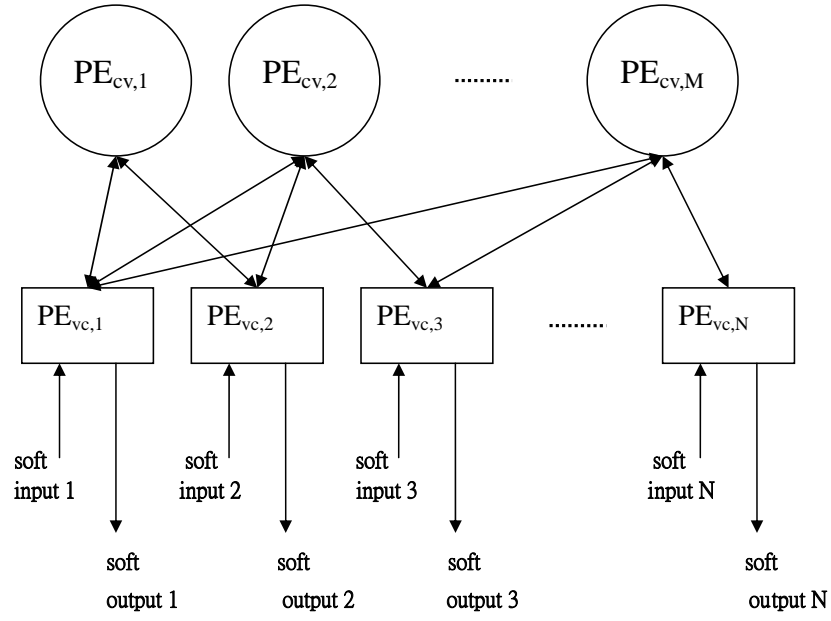


Figure 3.1: parallel architecture

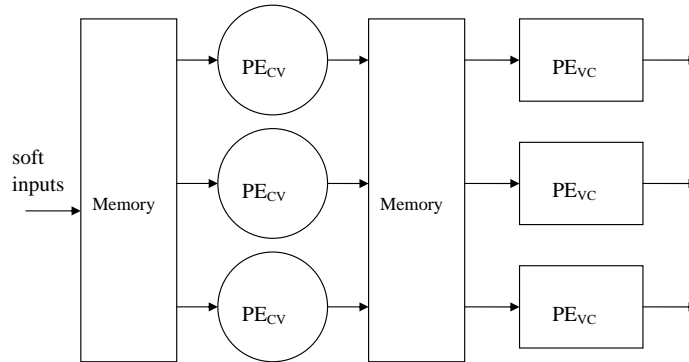


Figure 3.2: serial architecture

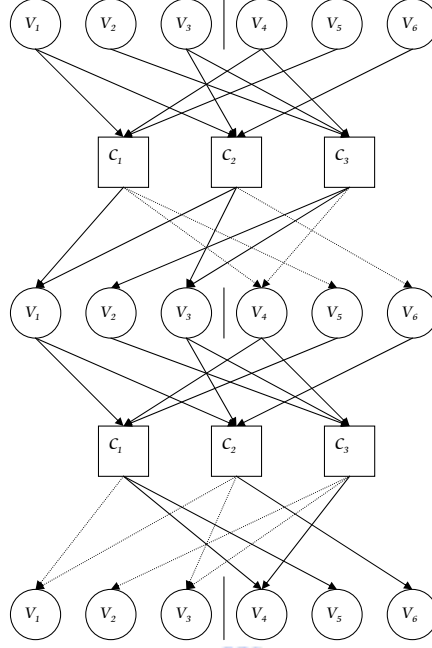


Figure 3.3: multi-stage factor graph representation of VSBP algorithm

ation z_{mn}^{i-1} to update the check-to-variable information at i -th iteration ε_{mn}^i . However the information of variable nodes to check nodes at i -th iteration z_{mn}^i is based on the computation of some check-to-variable information ε_{mn}^i , and from equation (2.9) ε_{mn}^i is also related to z_{mn}^{i-1} . Then we can use it to compute remaining values ε_{mn}^i .

For example: assume that we have a parity check matrix

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

and we split columns into 2 groups, every group contains 3 columns. Then the representation of VSBP algorithm in this example can be shown in Fig 3.3. The graph shows that the check-to-variable information obtained from first sub-iteration can be used to update the variable-to-check information at second sub-iteration.

So the concept of VSBP algorithm is that every sub-iteration can achieve available information from previous sub-iteration and sends newer information to next sub-iteration to have better error rate performance. The following is the step of VSBP algorithm:

1. Initialization: set $i = 1$ and the max number of iteration is I_{max} . For each m, n , initialize $z_{mn}^0 = F_n = 4 \cdot y_n / N_0$.
2. For $1 \leq g \leq G$. G is the number of group and N_G is the number of columns in group G .

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

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

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

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

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

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

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

4. Make hard decision z_n^i to obtain a outcome R_n^i with $z_n^i > 0$ if $R_n^i = 0$, else $R_n^i = 1$. If the outcome satisfies $H \cdot (R_n^i)^T = 0$ or achieving the maximum number of iteration I_{max} , then stop operation and choose R_n^i as the decoded codeword. Otherwise set $i = i + 1$ then go to Step 2.

3.2 Horizontal shuffled belief propagation

Same as VSBP algorithm, HSBP[7] algorithm splits check nodes into several groups and decoding is carried out in a group-by-group manner. HSBP algorithm is similar to VSBP algorithm. First we collect all the variable-to-check information z_{mn}^i (check m belong to k -th group) to update information of check nodes to variables nodes ε_{mn}^i . And then at next time sub-iteration, the computation of variable-to-check information z_{mn}^i

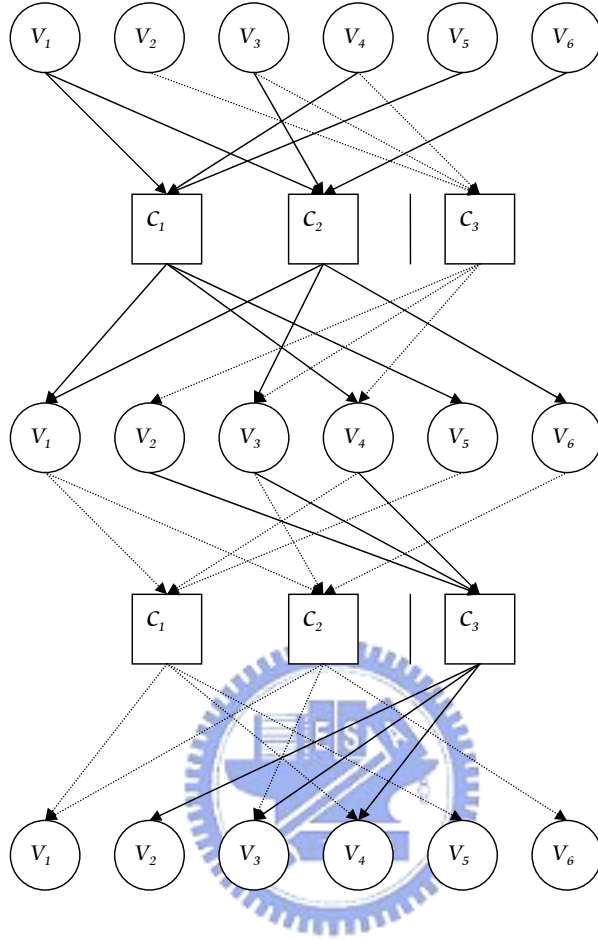


Figure 3.4: multi-stage factor graph representation of HSBP algorithm

(check m belong to $(k+1)$ -th group) is updated by ε_{mn}^{i-1} and ε_{mn}^i from early sub-iteration

For example: we use the same parity check matrix in previous VSBP example and separate rows into 2 groups. The first group contain 1 - st row and 2 - nd row, and other rows are in group 2. Fig 3.4 shows the representation. The idea of HSBP is to use the newest information to accelerate convergence.

Summarizing HSBP as following:

1. Initialization: set $i = 1$ and the max number of iteration is I_{mzx} . For each m, n , initialize $z_{mn}^0 = F_n = 4 \cdot y_n / N_0$.

2. For $1 \leq g \leq G$. G is the number of group and M_G is the number of rows in group G .

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

$$\tau_{mn}^i = \prod_{n' \in \mathcal{N}(m) \setminus n} \tanh\left(\frac{z_{mn'}^{i-1}}{2}\right) \quad (3.5)$$

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

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

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

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

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

4. Make hard decision z_n^i to obtain a outcome R_n^i with $z_n^i > 0$ if $R_n^i = 0$, else $R_n^i = 1$.

If the outcome satisfies $H \cdot (R_n^i)^T = 0$ or achieving the maximum number of iteration I_{max} , then stop operation and choose R_n^i as decoded codeword. Otherwise set $i = i + 1$ then go to Step 2.

Chapter 4

Group scheduling methods for shuffled belief propagation algorithm

The shuffled belief propagation algorithm introduced in chapter 3 shows that grouping parity check into several small sub-matrixes can help to reduce the complexity, processing elements of decoding and the most important speed-up the convergence of error rate. But how to group parity check matrix to achieve better speed of convergence will become a significant problem. The shuffled belief propagation algorithms include Vertical SBP and Horizontal SBP, and there exist some different decoding methods so the scheduling method would have some distinction and we will discuss them in the follows.

4.1 Message passing for several sub-iterations

The main idea of shuffled iterative decoding is that the more independent information can be updated, the more reliable extrinsic information and more correct decoding result we can have. More independent information achieving in every sub-iteration can avoid message dispersing un-uniformly, un-uniform messages might make some bits receiving higher reliability and others lower reliability and after several time iteration, this situation always result in weak convergence speed. So the question now is that find a schedule to implement this kind of independent situation.

In fact satisfying this condition is very complex because we must consider the con-

necting relation between check nodes and variable nodes for every sub-iteration. Taking an example that for HSBP we divided parity check matrix into two groups, and an iteration time will contain two sub-iterations. Every check group has path connecting with variable nodes. If there are no many intersection numbers of the first and second group variables, after first sub-iteration of i -th iteration the updated check-to-variable message will just depend on those variable-to-check information which connects with those variable nodes in first group. At second sub-iteration we will lose the relation of information when we want to update variable-to-check messages. Of course if we want to collect more independent information, the decoding processing of sub-iteration must get more information from more variable nodes.

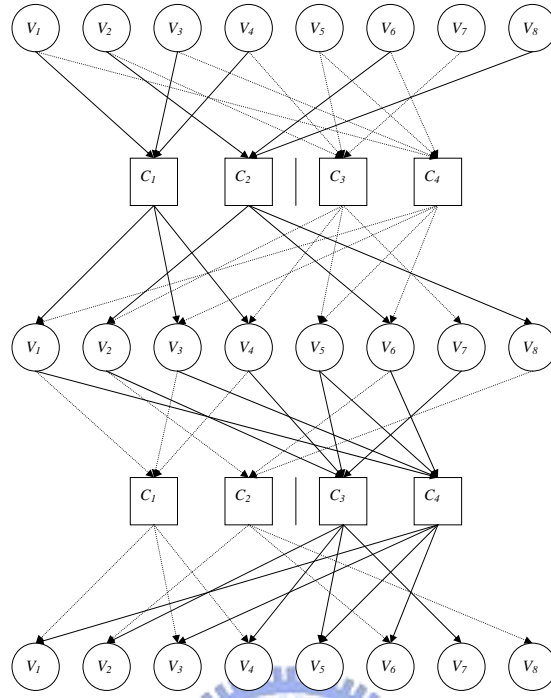
Example 3(*Group shuffled belief propagation*): Considering the parity check matrix:



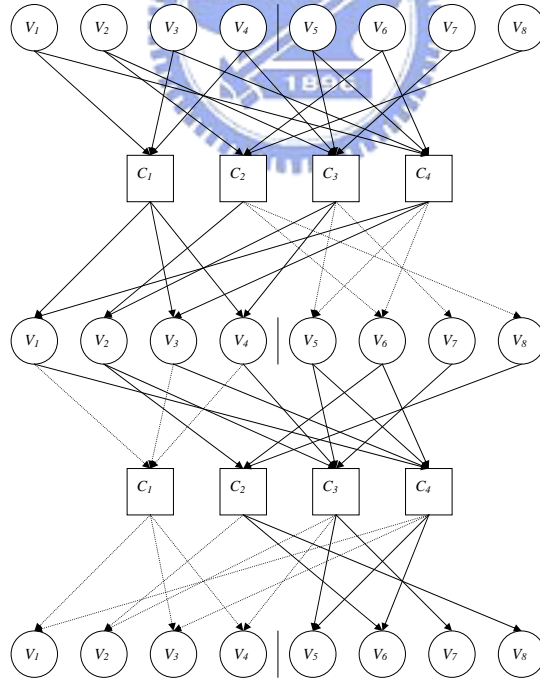
$$H = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

The MSBP factor graph of HSBP algorithm is shown in Fig 4.1(a). From Fig 4.1(a) we know that the first check group connects with V_1, V_2, V_3, V_4, V_6 and V_8 , so at first sub-iteration just some check-to-variable messages can be updated and the messages from V_5, V_7 can't be obtained. But second sub-iteration we also need messages from V_5, V_7 to update check-to-variable message. Because at first sub-iteration messages to V_5, V_7 can't be updated, we must use the messages updated from previous iteration for second sub-iteration at this iteration. This situation will cause that we could not get the newest information to update check-to-variable message at second sub-iteration and wouldn't speed-up the convergence.

With the same idea, Fig 4.1(b) shows the VSBP factor graph of this code. At second sub-iteration there are no check-to-variable messages obtained from check 1, and this situation will bring the un-uniform result to slow down the convergence. If the variable nodes in second group have connection with check 1, the processing can include the



(a) MSBP factor graph of HSBP algorithm



(b) MSBP factor graph of VSBP algorithm

Figure 4.1: MSBP factor graph of (8,4) LDPC code

variable-to-check 1 messages which are not used at second sub-iteration in the example, and then we will get more information.

In order to avoid the situation above, the easiest thought is that every check group must connect with almost all variable nodes for HSBP and every variable group must connect with almost all check nodes for VSBP. This situation not only can get more independent information at every sub-iteration time but also get newer message which is updated form previous sub-iteration time.

4.2 Group schedule method

Now considering that how to prevent the situation in example 3. Continuous to example 3: at first sub-iteration of HSBP the reason that there are no messages from V_5, V_7 to group 1 is V_5, V_7 doesn't participate with the check nodes in group 1. $H = [h_1^T, h_2^T]^T = [h'_1, h'_2]$, where

$$h_1 = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

$$h_2 = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

$$h'_1 = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

$$h'_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

h_1 is a sub-matrix of H and provide the message passing of first sub-iteration for HSBP, and the same as h_2 . h'_1 provide the message passing of first sub-iteration for VSBP, and the same as h'_2 . Because V_5, V_7 have no any connection with group 1, the 5—th column and 7—th column of h_1 (the check nodes of group 1 for HSBP) are zeros. Similar, from h'_2 the first row is zero, and it represent that there are no messages from check node 1 to those variable nodes in group 2 at second sub-iteration for VSBP. The all

zero sub-matrix columns for HSBP and all zero sub-matrix rows for VSBP result in the un-uniform situation which cause information can't be used efficiency.

In order to prevent all rows/columns zero, we must find a group schedule method before splitting parity check matrix. Taking an example that permuting the 4—th column of h'_1 with the first column of h'_2 , then all rows of the two matrixes will not be zeros. But with a parity check matrix, the algorithm to implement the result above is not easy and full searching seems not to be a good and efficiency method. So we try to find a simple method to nearly implement it, the following is for VSBP and the same concept as HSBP:

1. Define group number **group** and the number in a group **k**, and max number of iteration I_{max} . Initialize iteration=1.
2. Initialize group index $i = 1$. Iteration $> I_{max}$ finish the processing.
3. If $i=\mathbf{group}$, then iteration=iteration+1 and goto step 2, else goto step 4.
4. Initialize group index $ii = 1$. If j —th row of group i is all zero, then searching the column α with smallest column weight in group i . Else $i = i + 1$, then goto step 3. Initialize *counter*=1.
5. Finding the column β with nonzero j —th row in group ii and $ii \neq i$. Checking the value of ii , if $ii = \mathbf{group}$ goto step 8.
6. If there is no such column β , then $ii = ii + 1$ and redo step 5.
7. Permutating column α and column β . Checking if the number of nonzero rows in group i after permutating is less than before and the same as group ii . If the number of nonzero rows in group i and ii is all less than before, $i = i + 1$ goto step 3. Else recover the permutation and then $ii = ii + 1$, $counter = counter + 1$, goto step 5.

8. If $counter = \mathbf{group}$, searching the column β with second(third, forth,... \mathbf{k} -th) small column weight in group i and let $ii = 1$, then goto step 5. Else and no such column β then $i = i + 1$ and goto step 3.

Finishing this algorithm we can get H' produced by H and we hope that comparing to H there are less all zero columns in H' after separating them into sub-matrixes which provide message passing.

4.3 Difference between schedule of VSBP and HSBP

After implement the algorithm, we must pay attention to some details. Group of VSBP is based on variable node and HSBP is based on check node, this is the difference between them. For VSBP i -th sub-iteration can get check-to-variable messages which pass to those variable belonging to group i . Different sub-iteration time in a full iteration always cause different belief degree. The later we update the message the more reliability message we can have.

Because codeword contain information bits and parity check bits, and information bits are more important than parity check bits, in order to achieve better performance the group with information bits must be processed latest. So for VSBP we first split bits of codeword into two group, first group is parity check bits and second group is information bits. Then using the algorithm described in section 4.2 to schedule them.

But the group of HSBP is check node based, in fact almost every check nodes would connect with nearly the same information bit numbers. Therefore we don't have another processing like VSBP, just implement the group schedule algorithm.

Example 4 (*complete VSBP example of group schedule method*) Let a codeword $\mathbf{c} = (i_1, i_2, \dots, i_k, p_1, \dots, p_{n-k})$ where i_1, \dots, i_k means information bits and p_1, \dots, p_{n-k} is parity checks bits. If parity check matrix $\mathbf{H} = [h_1, h_2, \dots, h_k, h_{k+1}, \dots, h_n]$, h_1, \dots, h_k correspond to information bits and else parity check bits, then taking h_1, \dots, h_k and h_{k+1}, \dots, h_n into different group. Define a matrix $P_{mtn} = [G_1, G_2]$, $G_1 = [h_{k+1}, h_{k+2}, \dots, h_n]$ and $G_2 =$

$[h_1, \dots, h_k]$. Assuming we want to achieve VSBP by 4 groups, then dividing G_1 and G_2 into 2 small groups (if $k = n/2$). Let $g_{1,a}, g_{1,b}$ and $g_{2,a}, g_{2,b}$ be the two small groups divided from G_1 and G_2 . Then use group schedule method to implement G_1 and G_2 . The most important is that we can't permute the column in group $g \subset G_1$ with column in the group $g \subset G_2$, because it will cause that information bits couldn't get the newest extrinsic information.

4.4 Performance gain divination from EXIT chart

Now before simulation, we take the concept of EXIT chart[12] to estimate that if

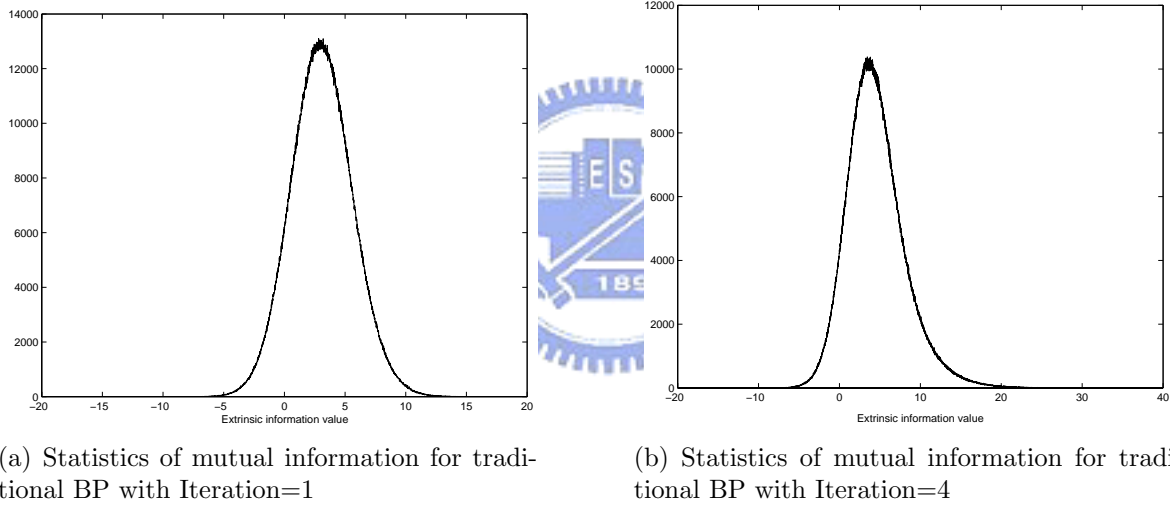


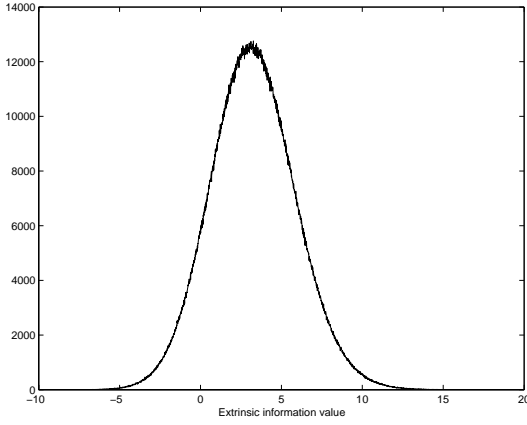
Figure 4.2: Statistics for (480,240) QC-LDPC code generated by 802.11n specification with traditional BP with transmitted $x=1$

our algorithm can improve speed of convergence. Usually we will assume that a priori input A can be modeled by applying an independent Gaussian random variable n_A with variance σ_A^2 and zero mean in conjunction with the known transmitted x .

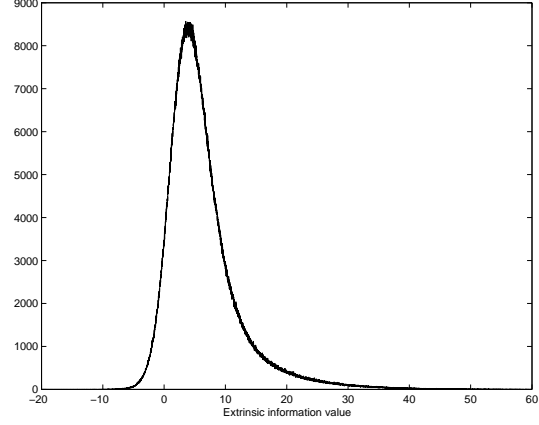
$$A = \mu_A \cdot x + n_A \quad (4.1)$$

where $\mu_A = \sigma_A^2/2$. With equation above the conditional pdf of A can be written as

$$P_A(\zeta|X=x) = \frac{e^{-((\zeta - ((\sigma_A^2/2) \cdot x)^2)/2\sigma_A^2)}}{\sqrt{2\pi}\sigma_A} \quad (4.2)$$



(a) Statistics of mutual information for VSBP with Iteration=1, G=6



(b) Statistics of mutual information for VSBP with Iteration=4, G=6

Figure 4.3: Statistics for (480,240) QC-LDPC code generated by 802.11n specification with VSBP, G=6 with transmitted $x=1$

mutual information $I_A = I(X; A)$ is

$$I_A = \frac{1}{2} \sum_{x=-1,+1} \int_{-\infty}^{\infty} P_A(\zeta|X=x) \cdot \log_2 \frac{2 \cdot P_A(\zeta|X=x)}{P_A(\zeta|X=-1) + P_A(\zeta|X=+1)} d\zeta \quad (4.3)$$

taking equation (4.2) into equation (4.3) we can get

$$I_A(\sigma_A) = 1 - \int_{-\infty}^{\infty} \frac{e^{-((\zeta - (\sigma_A^2/2))^2)/2\sigma_A^2)}}{\sqrt{2\pi}\sigma_A} \log_2(1 + e^{-\zeta}) d\zeta \quad (4.4)$$

when we want to draw EXIT chart, first we will define a I_A value and get σ_A from equation (4.4), then taking this σ_A into equation (4.1) to form a set of a priori input. Finally send a priori input and intrinsic information into decoder to achieve I_E , an example shown in Fig 4.1. But now there is a problem that we all assume a priori input as a Gaussian random variable. In the processing of decoding we will use I_E in previous step as I_A in this step and I_E may not has a Gaussian distribution especially for scheduling BP. Because scheduling BP would separate a full iteration into several sub-iterations and the message updated at later sub-iteration will be more reliable, the total extrinsic information PDF would turn aside Gaussian. Once I_E is not Gaussian distribution and we would not use Gaussian approximation to implement I_A which is equal to I_E in previous step. Fig 4.2 and Fig 4.3 show the statistics of I_E with 5000

codeword pattern which is send from bit nodes to check nodes, Y-axis means the numbers locates at x (extrinsic information value). We can see that at iteration=4 I_E of VSBP is not already Gaussian, so we use decoding algorithm to plot EXIT chart neither Gaussian approximation.

the following is a EXIT chart table for G=4 HSBP.

1 dB	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
IE_{i-VC}^g	0.603338	0.694625	0.740615	0.770734	0.792035	0.807281	0.818275	0.826249
IE_{VC}^g	0.594851	0.667506	0.707433	0.735937	0.758411	0.776163	0.790108	0.800967
IE_{CV}^g	0.069809	0.126099	0.180043	0.237488	0.297182	0.352668	0.402199	0.443413
IE_{i-VC}	0.599266	0.687808	0.732027	0.760926	0.781573	0.796804	0.80853	0.817505
IE_{VC}	0.589721	0.660188	0.698271	0.72481	0.745497	0.762259	0.776169	0.787574
IE_{CV}	0.066822	0.117586	0.163079	0.209308	0.257665	0.304959	0.349137	0.388982

2 dB	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
IE_{i-VC}^g	0.704862	0.844271	0.914014	0.950083	0.968754	0.978315	0.983514	0.986445
IE_{VC}^g	0.692075	0.808452	0.878958	0.923572	0.95081	0.966495	0.975527	0.980829
IE_{CV}^g	0.151169	0.332077	0.523086	0.683455	0.79643	0.865705	0.90634	0.930486
IE_{i-VC}	0.699321	0.83166	0.898798	0.936373	0.9581	0.970593	0.977968	0.982432
IE_{VC}	0.684403	0.792933	0.858301	0.901901	0.931656	0.951406	0.964107	0.972338
IE_{CV}	0.141169	0.293694	0.451728	0.594971	0.710391	0.795219	0.852686	0.889952

mutual information of bit-to-check/check-to-bit for G=4 HSBP at $E_b/N_0=1$ and 2dB

where IE_{i-VC}^g means mutual information of information bit nodes send to checks nodes for group scheduling method and IE_{i-VC} is traditional method without scheduling, the same, IE_{VC}^g / IE_{CV}^g is mutual information of bit nodes to check nodes/check nodes to bit nodes for group scheduling method and $i = n$ in the table means n -th iteration. We always focus on BER or FER performance, so the most important for us is information bits not codeword bits. Maybe there is a case that the value IE_{i-VC} in our algorithm is bigger than the IE_{i-VC} without scheduling but IE_{VC} inverse. The value which can reflect BER or FER is IE_{i-VC} and this is the reason we choosing additional IE_{i-VC} to analysis. Comparing EXIT chart table we can find that according to increasing iteration number, the corresponding mutual information will be enhanced. And the table above

also reveal that the mutual information which information bit nodes send to check nodes for group scheduling always larger than the mutual information without scheduling under the same iteration. Sometime Exit chart verify that why our algorithm can speed up the convergence of performance. The following shows the complete EXIT chart table for different cases.

1 dB	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
IE_{i-VC}^g	0.585746	0.67344	0.718868	0.748854	0.771095	0.788092	0.801126	0.811241
IE_{VC}^g	0.581934	0.651335	0.689472	0.716273	0.737715	0.755601	0.770465	0.782752
IE_{CV}^g	0.061804	0.108829	0.151101	0.194284	0.240457	0.288675	0.334817	0.3765
IE_{i-VC}	0.586216	0.673041	0.71701	0.745941	0.767016	0.783044	0.795765	0.805811
IE_{VC}	0.579862	0.648083	0.685314	0.711004	0.730961	0.747448	0.761492	0.773436
IE_{CV}	0.060901	0.106005	0.144723	0.183119	0.222733	0.263708	0.303933	0.341878

2 dB	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
IE_{i-VC}^g	0.676562	0.806036	0.87894	0.923314	0.949935	0.965895	0.975254	0.981082
IE_{VC}^g	0.671018	0.777417	0.845639	0.894011	0.927116	0.94933	0.963518	0.972494
IE_{CV}^g	0.127032	0.265894	0.420488	0.572855	0.698155	0.790517	0.852712	0.892073
IE_{i-VC}	0.674178	0.804371	0.875159	0.917493	0.943574	0.960185	0.970632	0.977236
IE_{VC}	0.66831	0.770944	0.83456	0.879629	0.91213	0.935526	0.951963	0.963243
IE_{CV}	0.124467	0.253178	0.387236	0.518155	0.633238	0.726419	0.797796	0.848137

mutual information of bit-to-check/check-to-bit for G=2 HSBP at $E_b/N_0=1$

and 2dB

1 dB	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
IE_{i-VC}^g	0.652909	0.70058	0.73251	0.756394	0.775104	0.790137	0.80223	0.811945
IE_{VC}^g	0.632109	0.671163	0.698723	0.720514	0.738787	0.754679	0.768461	0.780301
IE_{CV}^g	0.054664	0.095990	0.133426	0.171447	0.212175	0.255747	0.300071	0.342928
IE_{i-VC}	0.647367	0.696525	0.729314	0.753797	0.77334	0.78865	0.800594	0.810228
IE_{VC}	0.633566	0.673305	0.701239	0.723463	0.742494	0.758496	0.771869	0.783154
IE_{CV}	0.063825	0.105162	0.142222	0.180133	0.22122	0.264898	0.308427	0.349028

2 dB	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
IE_{i-VC}^g	0.771652	0.849569	0.901349	0.934633	0.955667	0.968392	0.976175	0.98106
IE_{VC}^g	0.74261	0.809612	0.860294	0.898823	0.927242	0.947108	0.960637	0.969674
IE_{CV}^g	0.10753	0.223848	0.358886	0.503866	0.63483	0.73698	0.810739	0.861254
IE_{i-VC}	0.762402	0.842303	0.896452	0.931802	0.953623	0.966889	0.975121	0.980343
IE_{VC}	0.747058	0.817552	0.870524	0.909096	0.935548	0.95313	0.964716	0.972321
IE_{CV}	0.13099	0.251863	0.387202	0.527521	0.651693	0.748292	0.817742	0.864729

mutual information of bit-to-check/check-to-bit for G=2 VSBP at $E_b/N_0=1$
and 2dB

1 dB	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
IE_{i-VC}^g	0.664242	0.7159	0.749304	0.77362	0.792015	0.805942	0.816083	0.823683
IE_{VC}^g	0.640015	0.682989	0.712814	0.736291	0.75584	0.772292	0.785619	0.79634
IE_{CV}^g	0.060833	0.110264	0.156447	0.205976	0.258813	0.313228	0.364553	0.409898
IE_{i-VC}	0.658139	0.7115	0.745783	0.770686	0.78957	0.803393	0.813841	0.821728
IE_{VC}	0.64203	0.6857	0.715791	0.739548	0.759042	0.774552	0.787033	0.797027
IE_{CV}	0.071736	0.12059	0.165758	0.213341	0.263942	0.313988	0.360252	0.40162

2 dB	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
IE_{i-VC}^g	0.789965	0.875166	0.92636	0.955281	0.970881	0.979346	0.984003	0.986783
IE_{VC}^g	0.75548	0.830789	0.885675	0.924593	0.950092	0.965726	0.974922	0.980442
IE_{CV}^g	0.12387	0.272566	0.447494	0.618162	0.751003	0.837415	0.889112	0.919233
IE_{i-VC}	0.778951	0.865693	0.918861	0.949346	0.966573	0.976128	0.9818	0.984984
IE_{VC}	0.760888	0.839262	0.893799	0.929402	0.951766	0.965413	0.973838	0.978982
IE_{CV}	0.151256	0.300959	0.463108	0.614346	0.732856	0.815202	0.86904	0.902527

mutual information of bit-to-check/check-to-bit for G=4 VSBP at $E_b/N_0=1$
and 2dB

1 dB	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
IE_{i-VC}^g	0.669661	0.723362	0.757573	0.7815	0.799659	0.812891	0.82252	0.829341
IE_{VC}^g	0.643778	0.688673	0.719704	0.743614	0.76358	0.779763	0.792745	0.802738
IE_{CV}^g	0.064490	0.117763	0.169117	0.223607	0.281114	0.337498	0.388619	0.431895
IE_{i-VC}	0.662277	0.717874	0.753155	0.778057	0.796315	0.809356	0.818959	0.826206
IE_{VC}	0.645373	0.691148	0.722534	0.746908	0.766331	0.781382	0.793297	0.802659
IE_{CV}	0.075910	0.128478	0.177978	0.230345	0.285135	0.337183	0.383892	0.424279

2 dB	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
IE_{i-VC}^g	0.798766	0.886061	0.934489	0.960367	0.973659	0.981154	0.98498	0.987468
IE_{VC}^g	0.761586	0.839725	0.894382	0.931616	0.954627	0.9687	0.976738	0.981726
IE_{CV}^g	0.133524	0.296723	0.483439	0.654254	0.776509	0.853796	0.899671	0.926987
IE_{i-VC}	0.786138	0.876658	0.928476	0.956363	0.971277	0.979557	0.983924	0.986857
IE_{VC}	0.767115	0.849887	0.904365	0.938013	0.958131	0.970205	0.977125	0.981688
IE_{CV}	0.162687	0.326971	0.500232	0.652337	0.76571	0.840914	0.887562	0.916437

mutual information of bit-to-check/check-to-bit for G=6 VSBP at $E_b/N_0=1$
and 2dB

Scanning over those EXIT chart table, the common phenomenon is that no matter what kind of cases, the value IE_{i-VC}^g is larger than IE_{i-VC} all the time. Although the improvement seems not apparent enough for us to distinguish IE_{i-VC}^g and IE_{i-VC} , mutual information with scheduling indeed excels mutual information without scheduling. In fact we can't find a relation between tiny improvement of mutual information and performance enhancement, but this is also not our emphasis, we just want to confirm that rising speed of convergence with group scheduling will be reflected in the EXIT chart. With this EXIT chart result we can estimate the simulation performance with group scheduling in the following chapter would be better than without scheduling.



Chapter 5

Message flow distribution and noise resisting

Usually when we analysis the error rate performance of a code, a codeword is always taken into the smallest processing unit (we always don't take process on bits). Hence each bit within a codeword is regarded as having equal noise resisting ability, and generally we doesn't take extra processing on bits in codeword.

It is impossible that every bit has the same ability of noise resisting for LDPC code because each bit is composed of different numbers of information bits (one bit which is composed of many information bits usually has better noise resisting ability). If the error protection ability is apparent in different bits, the traditional codec algorithm should be modified for this special characteristic, and it would be introduced in this chapter.

5.1 Message flow for BP algorithm

The most important thing in this section is to search an efficiency algorithm to quantize the quantity of noise resisting ability at different bits.

Use a simple example to express it: assume a codeword containing 4 bits, and use a vector to represent codeword bits contribution to one bit. Considering one vectors $[0.5, 0.3, 0.1, 0.1]$, the first element (contribution from first bits in a codeword) of the vector dominant the $\{+1, -1\}$ outcome and can't provide a objective solution. Then the method to find contribution vector is introduced in the next. The contribution coming

from all bits in a codeword can also provide a rough and easy analysis of decoding. The more uniform information getting from all bits, the more correct result we have.

For a parity check matrix $\mathbf{H} = [H_{mn}]$ as we defined before, the set of bits participating in check m is $\mathcal{N}(m) = \{n : H_{mn} = 1\}$ and the set of checks in which bit participates as $\mathcal{M}(n) = \{m : H_{mn} = 1\}$. Some definition are also needed:

1. c_{mn}^i = the amount of contribution vector which is made by variable nodes passing from check node m to variable node n at i -th iteration.
2. b_{mn}^i = the amount of contribution vector which is made by variable nodes passing from variable node n to check node m at i -th iteration.
3. p_n^i = the complete contribution vector of V_n at i -th iteration.

The value we want to find is p_n^i . Similar to the message flow of BP algorithm, we could define the relation of c_{mn}^i , b_{mn}^i , p_n^i and process of message travel. As the channel values associated with the variable nodes are independent and identically distributed (i.i.d.), we assign equal contribution to all initial messages generated by variable nodes. With the definitions above, message flow can be computed as 4 steps:

1. Initialization: Set $b_{mn}^0 = [\delta(1-n), \delta(2-n), \dots, \delta(N-n)]$, $1 \leq n \leq N$, then let $i = 1$ and δ defined as

$$\delta(n) = \begin{cases} 1, & \text{if } n = 0 \\ 0, & \text{if } n \neq 0 \end{cases}$$

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

$$c_{mn}^i = \frac{\sum_{n' \in \mathcal{N}(m) \setminus n} b_{mn'}^{i-1}}{\deg(m) - 1} \quad (5.1)$$

$\deg(m)$ in equation (5.1) is the number of variable nodes connecting with check m , and the purpose of dividing $\deg(m)$ is to normalize the value of c_{mn}^i to make sum of every element in this vector be one.

3. At variable node: $\forall n, 1 \leq n \leq N$, and each $m \in \mathcal{M}(n)$, then compute

$$b_{mn}^i = \frac{[\delta(1-n), \delta(2-n), \dots, \delta(N-n)] + \sum_{m' \in \mathcal{M}(n) \setminus m} c_{m'n}^i}{deg(n) - 1} \quad (5.2)$$

$$p_n^i = \frac{[\delta(1-n), \delta(2-n), \dots, \delta(N-n)] + \sum_{m \in \mathcal{M}(n)} c_{mn}^i}{\gamma} \quad (5.3)$$

the definition and purpose of $deg(n)$ in equation (5.2) are very similar to $deg(m)$, and γ is also to normalize p_n^i .

4. $\forall n, 1 \leq n \leq N$, compute mean and variance and after the processing set $i = i + 1$.

Define $b_{j,n}^i$ is j -th element of vector p_n^i . If $i < I_{max}$, goto step 2.

$$\mu_n^i = \frac{\sum_{j=1}^N b_{j,n}^i}{N} = \frac{1}{N} \quad (5.4)$$

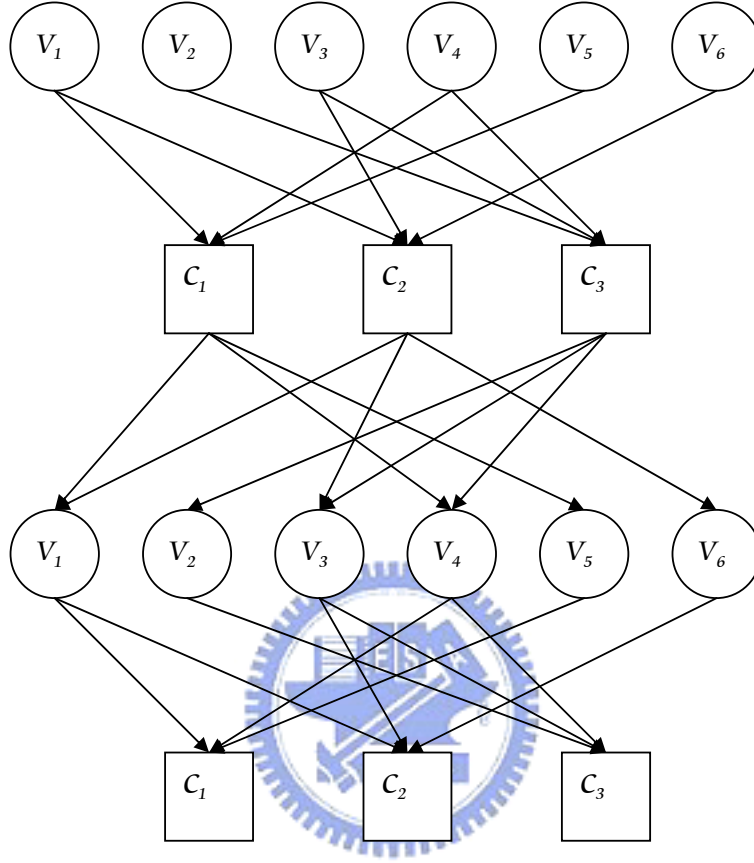
$$(\sigma_n^i)^2 = \frac{\sum_{j=1}^N [b_{j,n}^i - \mu_n^i]^2}{N} \quad (5.5)$$

Finally, the quantization of contribution can be introduced by the value μ_n^i / σ_n^i . Because μ_n^i is fixed, we just focus on the value σ_n^i . If the difference of every element in the same vector is not much, this vector would be uniform and under this condition the value $(\sigma_n^i)^2$ is always small. Since $(\sigma_n^i)^2$ is positive, small value of $(\sigma_n^i)^2$ implies large value of $1/\sigma_n^i$. The more uniform vector we have, the larger $1/\sigma_n^i$ value we can get, and this is the reason why using the value to quantize uniformity of a vector. Because the noise is not considered on our design, we expect that at high SNR we achieve the desired analytic result we have.

Example 5 (*example of message flow*): For a parity check matrix H of LDPC code:

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

with this matrix, Fig 5.1 shows the first two iteration message flow, and the following two tables are contribution vector set of first and second iteration. The tuple (V_a, V_b) (V_a is the label of first column in the two tables and V_b is the label of first row in the two tables) denote the contribution from V_b to V_a , so each row of table represent contribution of all variable nodes for V_a and σ^2 is the variance of vector V_a .



first step:

$$V_1 \rightarrow C_1 \& C_2 = [1, 0, 0, 0, 0, 0]$$

$$V_3 \rightarrow C_2 = [0, 0, 1, 0, 0, 0]$$

$$V_4 \rightarrow C_2 = [0, 0, 0, 1, 0, 0]$$

$$V_5 \rightarrow C_1 = [0, 0, 0, 0, 1, 0]$$

$$V_6 \rightarrow C_2 = [0, 0, 0, 0, 0, 1]$$

Second step:

$$C_1 \rightarrow V_1 = [0, 0, 0, 0.5, 0.5, 0]$$

$$C_2 \rightarrow V_1 = [0, 0, 0.5, 0, 0, 0.5]$$

Third step:

$$V_1 \rightarrow C_1 = [0.5, 0, 0.25, 0, 0, 0.25]$$

Figure 5.1: message flow for example 7

	V_1	V_2	V_3	V_4	V_5	V_6	σ^2	$1/\sigma$
V_1	0.3333	0	0.1666	0.1666	0.1666	0.1666	0.0556	4.24264
V_2	0	0.5	0.25	0.25	0	0	0.2083	2.19089
V_3	0.1666	0.1666	0.3333	0.1666	0	0.1666	0.0556	4.24264
V_4	0.1666	0.1666	0.1666	0.3333	0.1666	0	0.0556	4.24264
V_5	0.25	0	0	0.25	0.5	0	0.2083	2.19089
V_6	0.25	0	0.25	0	0	0.5	0.2083	2.19089

iteration = 1

	V_1	V_2	V_3	V_4	V_5	V_6	σ^2	$1/\sigma$
V_1	0.3333	0.0833	0.125	0.125	0.0833	0.1667	0.1667	5.11682
V_2	0.125	0.5	0.125	0.125	0.0625	0.0625	0.138	2.69171
V_3	0.125	0.1667	0.3333	0.125	0.0833	0.1667	0.0382	5.11682
V_4	0.125	0.1667	0.125	0.3333	0.1667	0.0833	0.0382	5.11682
V_5	0.125	0.0625	0.125	0.125	0.5	0.0625	0.138	2.69171
V_6	0.125	0.0625	0.125	0.125	0.0625	0.5	0.138	2.69171

iteration = 2

5.2 Comparing EXIT chart to message flow

Before this section, we must jump to Fig 6.7 at section 6.2 first. Fig 6.7 show the result of message flow of QC-LDPC generated by 802.11n specification. Message flow in previous section illustrates the method to search different noise resisting ability in different variable nodes. Now we observe the amount of mutual information in different bit to forecast this phenomenon roughly. The following shows mutual information for traditional BP algorithm and IE_{b-VC} denotes IE of variable nodes to check nodes subject to those bits corresponding to Fig 6.7 with value $1/\sqrt{Var}$ larger than 150.

1 dB	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
IE_{b-VC}	0.563895	0.68092	0.735431	0.769741	0.793753	0.811279	0.824541	0.834421
IE_{VC}	0.563327	0.628319	0.664278	0.68898	0.70823	0.724211	0.738177	0.750516

2 dB	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
IE_{b-VC}	0.643299	0.80696	0.884653	0.930109	0.956167	0.970733	0.978948	0.983486
IE_{VC}	0.642671	0.736195	0.795915	0.840562	0.875558	0.903072	0.924362	0.940556

mutual information of bit-to-check/check-to-bit for traditional BP at $E_b/N_0=1$ and 2dB

From the table above we can see that in every iteration IE_{b-VC} is always better than IE_{VC} . Larger IE_{b-VC} shows that given a set of a priori input, those bits with better performance can get more mutual information than other bits and imply that those variable nodes in Fig 6.7 with larger $1/\sqrt{\text{Var}}$ value have better error correcting ability. EXIT chart is another method to explain unequal error protection and its result seems to coincide with that of message flow.

5.3 Power allocation based on message flow

After considering message flow and expecting the phenomenon of unequal noise resisting, it arise a question: is there any processing to further improve the performance? Because different variable has distinct error protection, it is instinctive to add a weight to some variables.

We consider that how does the weight be attached. There are many kinds of weighting method and are used in several decoding algorithm, for example iterative decoding of turbo product codes by Pyndiah[8]. An easy weighting processing is to give a heavy weight to those variable-to-check messages which come from the variable nodes with better performance, and slight weight to messages come from another variables with worse performance. It looks like a good method but this assumption may seem to clash with the principle of BP algorithm in equation (2.10) of section 2.3.

We focus on how to seek a efficiency method to put concept of weighting into decoding without clashing with the principle of BP algorithm. But from equation (2.8) to equation (2.10), the messages ε_{mn}^i and z_{mn}^i all depend on messages ε_{mn}^{i-1} and z_{mn}^{i-1} , and any adjustment will cause aberration of BP at next iteration for iterative decoding algorithm. So we choose F_n as the tuning argument.

There are several reasons for choosing F_n to adjust; first, the value of F_n doesn't vary

with iteration. Second, equation (2.10) shows z_{mn}^i is composed of F_n , and in other words, adjustment of F_n sometimes can represent that messages are weighted. Third, all the messages are consisted of F_n after step by step iterations and imply idea of weighting.

Since $F_n = 4 \cdot y_n / N_o$ and N_o is fixed, only the remaining part y_n can be tuned. Since $y_n = x_n + \mathcal{AWGN}$, y_n is related to x_n where x_n is a codeword bit after modulation. Let $\mathbf{x} = [x_1, x_2, \dots, x_N]$ be modulated codeword vector. The value of x_n depend on total transmitting power. Therefore, allocating power to each x_n would produce variation of y_n under fixed power constrain. As long as a adequately power allocation method can be found, we can further improve the performance.

The value of x_n is changed, so F_n of decoding algorithm must be modified. Let A_n be the amplitude of x_n , and $A_n = 1$ is the condition without power allocation.

$$\begin{aligned}
 F_n &= \ln \frac{Pr(x_n = +A_n | y_n)}{Pr(x_n = -A_n | y_n)} \\
 &= \ln \frac{e^{-\frac{(y_n - A_n)^2}{N_o/2}}}{e^{-\frac{(y_n + A_n)^2}{N_o/2}}} \\
 &= \frac{4 \cdot A_n \cdot y_n}{N_o}
 \end{aligned}$$

At receiver, we must use the modified F_n to replace with the F_n without power allocation.

5.4 Channel mapping based on message flow

Before starting this section, we briefly introduce frequency selective channel. In wireless telecommunications, multipath is the propagation phenomenon that results in radio signals' reaching the receiving antenna by two or more paths (Fig 5.2). The effects of multipath include constructive and destructive interference, and phase shifting of the signal. When signal is viewed in the frequency domain, the parameter of concern is the channel's coherence bandwidth, B_c , which is a measure of the transmission bandwidth

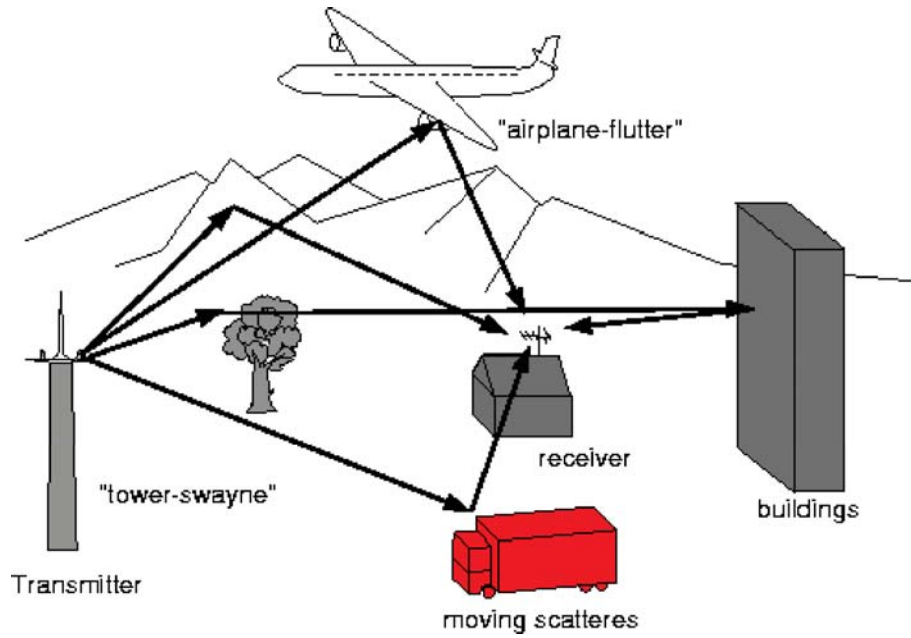


Figure 5.2: model of a multipath channel

for which signal distortion across the channel becomes noticeable. A multipath channel is said to be frequency-selective if the coherence bandwidth of the channel is smaller than the bandwidth of the signal. Fig 5.3 shows a channel response example for frequency-selective channel.

Next, we enter the emphasis on this section. If the performance indeed is improved using power allocation, we must discuss that what does power allocation brings to this system. The method of weighting mentioned in previous section is just a idea, and for system the concept of weighting actually forms different E_b/N_o at different bit position. Different variable nodes have different noise resisting ability and performance, therefore distinct E_b/N_o can be used to improve the performance of those variable nodes with worse BER.

Generally, there are several methods achieving distinct E_b/N_o , if all the channel gains are the same, power allocation seems to be the sole solution, but when channel response is not flat we are not necessary using power allocation. Assuming one channel has channel gain $|H|$, then we can write $Y = |H| * X + W$ (if we assume the phase

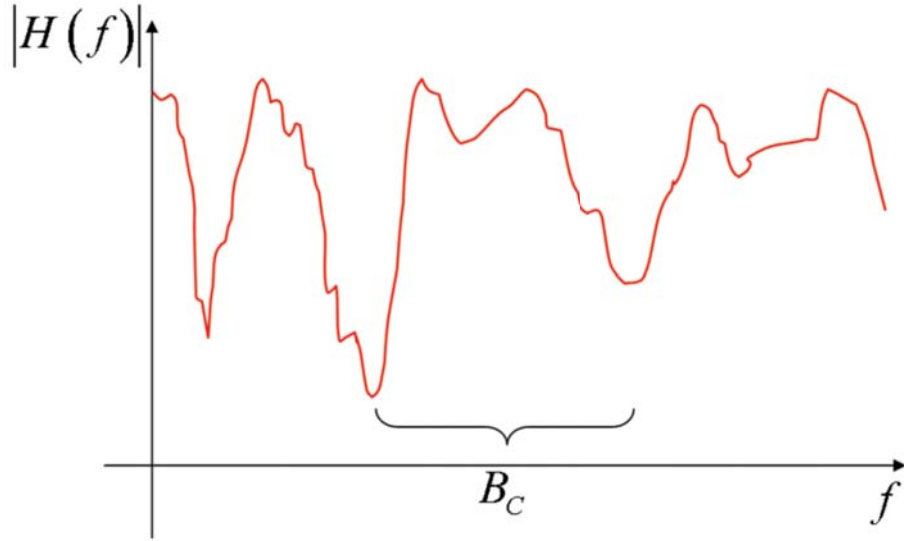


Figure 5.3: a channel response example of frequency selective channel

response is known) where X is the signal, Y the received signal and W is noise. We know $|H|$ is like an adjustable argument, comparing with $|H| = 1$ if $|H| > 1$, this means X is amplified and the equivalent E_b/N_o is enhanced, therefore if $|H| < 1$ implies E_b/N_o is reduced.

Finally, like power allocation we just find out all the channel gains, and choosing the channels with higher channel gains to transmit those bits with better or worse performance for different requirement.

Chapter 6

Simulation result

In this chapter we report some results from computer simulation. We use the codes derived from expanding the base model matrix defined in the IEEE 802.11n[9] specification and those from the base model given in the IEEE 802.16e[10] specification to complete our analysis. First, we compare shuffled method and group schedule method for VSBP and HSBP (with different group $G=2,4,6$) with conventional BP decoders. For convenient we use C-BP, C-VSBP, and C-HSBP to represent conventional BP, VSBP and HSBP. The same, G-VSBP, G-HSBP represent group schedule method for VSBP and HSBP. Second, we perform the message flow of BP algorithm to estimate distinct noise resisting ability of different bits in a codeword, then we put the result into channel mapping algorithm and compare the performance of channel mapping with performance of non-mapping .

6.1 Error rate performance comparison

We use the quasi-cyclic LDPC (QC-LDPC) code with code rate= $1/2$, codeword length=480, created by the base model matrix in the IEEE 802.11n specification as fundamentality of analysis. Fig 6.1 and Fig 6.2 present the bit error rate (BER) performance with $I_{max}=1, 3$ for $G=2, 4$ and 6 . They show that our algorithm can help to speed up the convergence of VSBP if we compare it with conventional shuffled BP algorithm. Fig 6.3 and Fig 6.4 also display similar results in the case of HSBP.

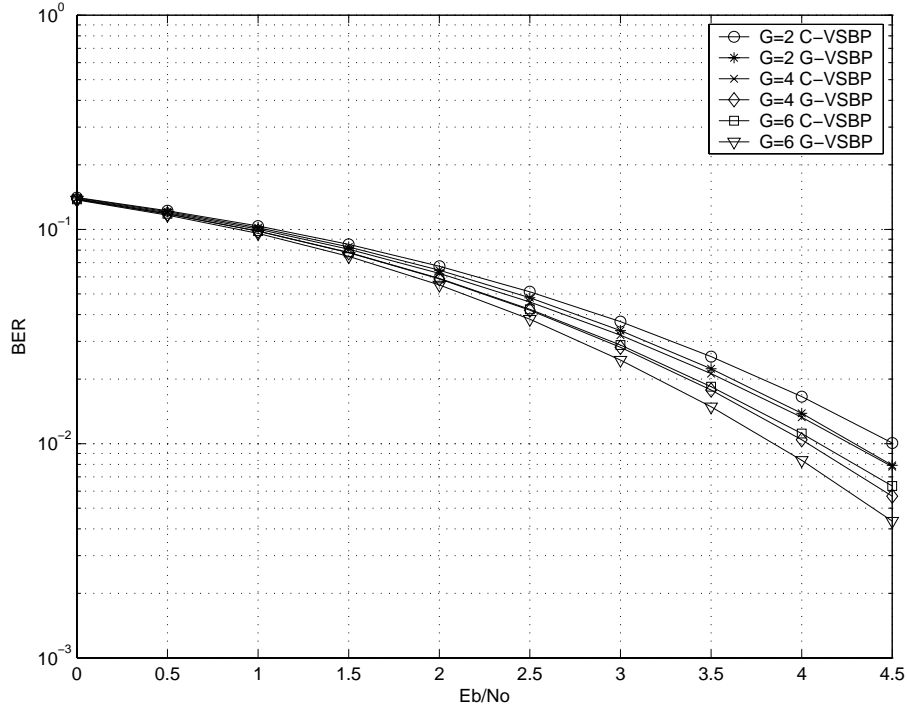


Figure 6.1: BER performance of the (480,240) QC-LDPC code with $Imax=1$, VSBP $G=2,4,6$.

Taking Fig 6.2 and Fig 6.4 into compare, for fixed SNR and G , the gap between C-HSBP and G-HSBP seems to be larger than the gap between C-VSBP and G-HSBP. We explain it: in i -th sub-iteration, just some check-to-variable messages which connect with variable nodes in i -th group be updated for VSBP. A complete iteration include several sub-iterations. The variable nodes in the later group (they almost are information bits) usually receive reliable information updated from previous sub-iterations, so the variable nodes in different groups will get different information reliability. Because different variable nodes in different groups during one iteration would get distinct reliability and lead to un-uniform information, the un-uniform information will pass to next iteration and become more un-uniform after several iteration. However, for HSBP, almost every check-to-variable messages which pass from the check nodes in i -group will convey to information bits, so in every sub-iteration we can get messages from check nodes to information bits and compare to VSBP it has more uniform information dispersion.

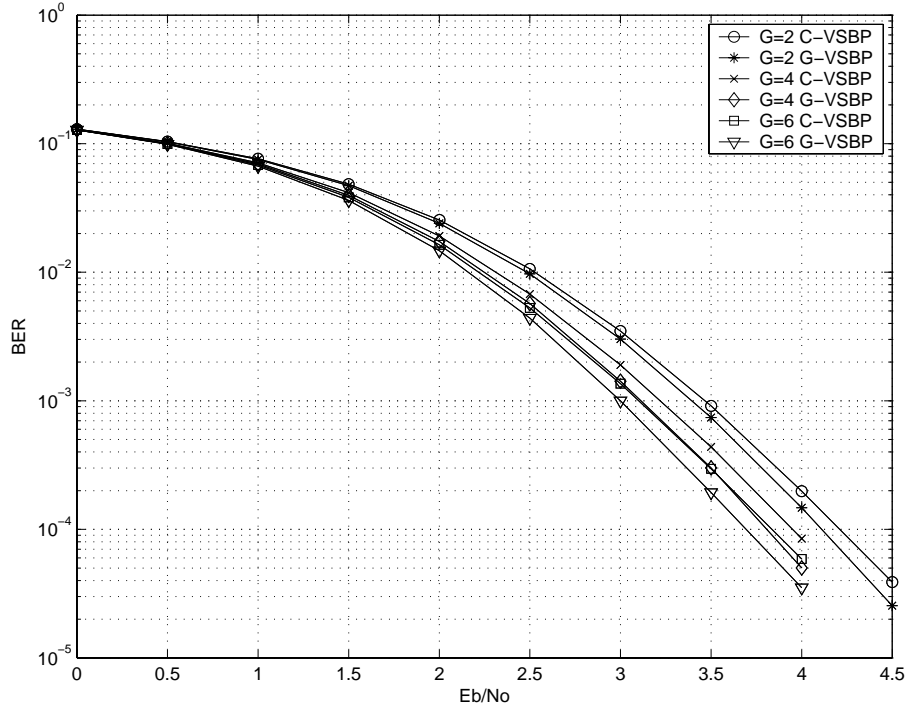


Figure 6.2: BER performance of the (480,240) QC-LDPC code with $I_{max}=3$, VSBP $G=2,4,6$.

Then, we use frame error rate (FER) performance to make a comparison with another schedule method CB-HSBP [11]. CB-HSBP means the schedule: the more cycles a check node suffered, the more anterior group we put it into. Fig 6.5 and Fig 6.6 show the convergence rate for G-HSBP is better than C-HSBP and BP, and the more group we divide the larger performance gap under fixed SNR we have.

Then we discuss the reason that the convergence rate of G-HSBP is better than CB-HSBP. For LDPC code we know girth (smallest length of cycle) would affect the final BER performance, however in every iteration the performance be enhanced relies on the amount of extrinsic information we can get between different iterations. Therefore, in order to speed up the convergence we must focus on extrinsic information neither girth length.

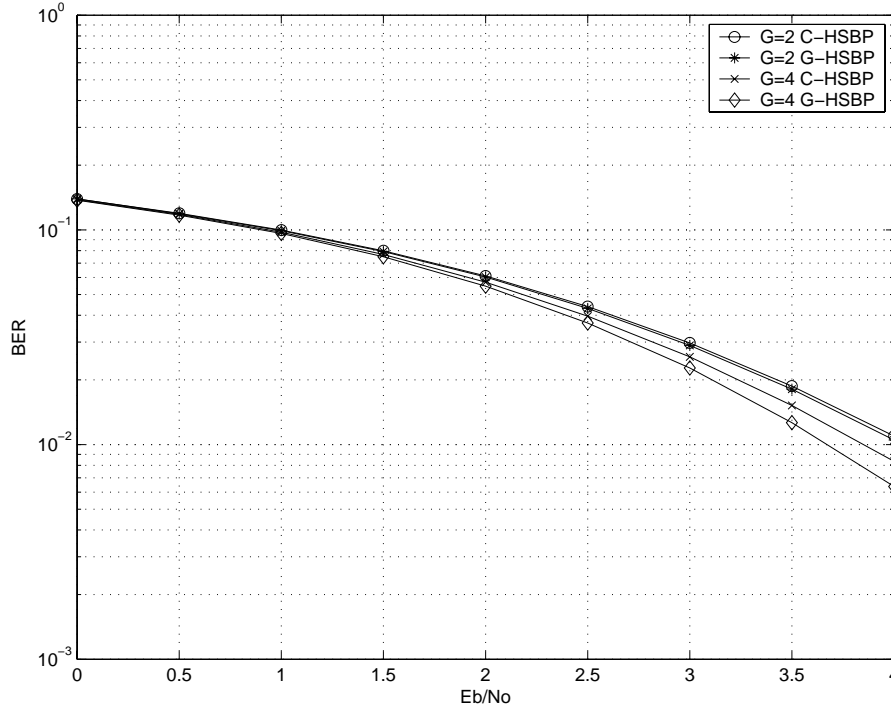


Figure 6.3: BER performance of the (480,240) QC-LDPC code with $I_{max}=1$, HSBP $G=2,4$.

6.2 Message flow and distinct performance of different bits

Fig 6.7 is the result of message flow (mentioned at Chapter 5) for code created from IEEE 802.11n specification with $L=480$, x-axis represents the position of codeword bits and y-axis is the value $1/\sqrt{Var}$ which is used to guess the noise resisting behavior. If one bit has higher $1/\sqrt{Var}$, sometimes it implies this bit suffers uniform contribution from other bits and has better error resisting ability. After several iterations, $1/\sqrt{Var}$ value becomes larger and we expect that numbers of error would be reduced.

The codeword split into 2 different groups is based on the value of Fig 6.7 at $I_{max} \geq 4$. We put the value larger than 150 in a group and other values in another group, then we observe the performance of different group. Fig 6.8 shows the BER of high $1/\sqrt{Var}$

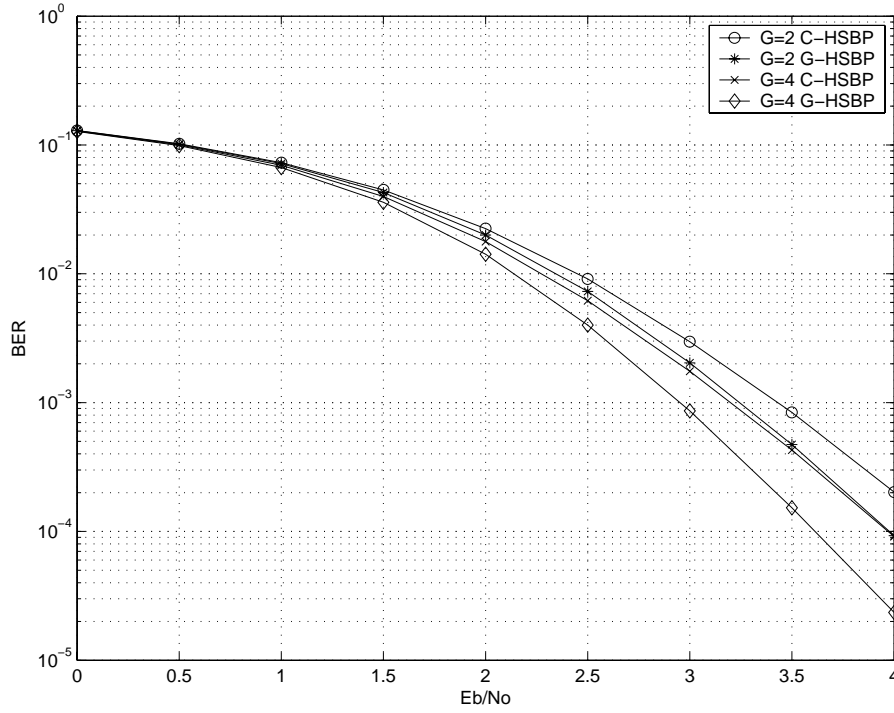


Figure 6.4: BER performance of the (480,240) QC-LDPC code with $Imax=3$, HSBP $G=2,4$.

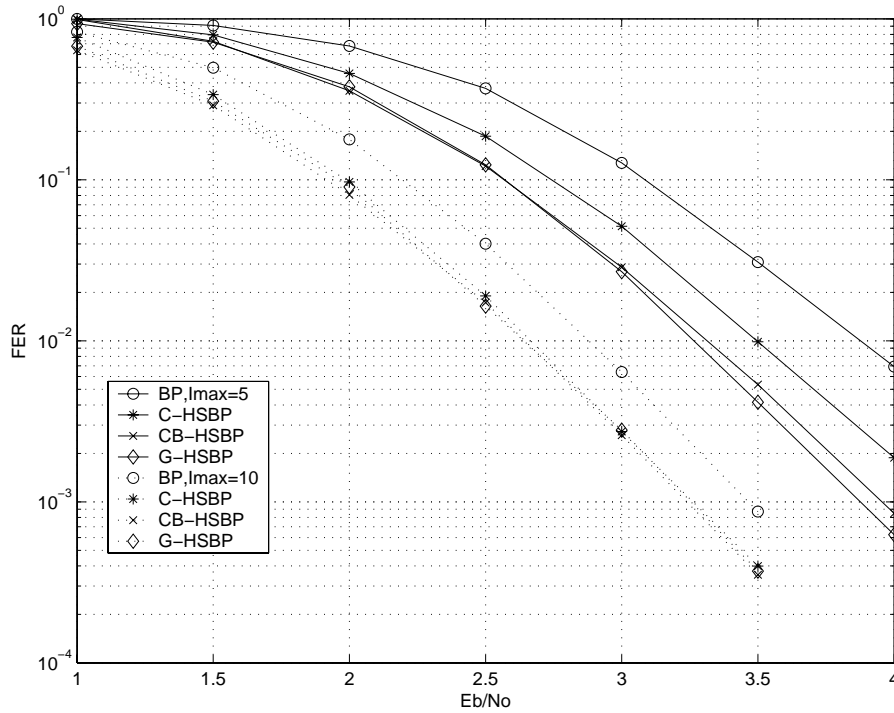


Figure 6.5: FER performance of the (480,240) QC-LDPC code with $Imax=5,10$, $G=2$.

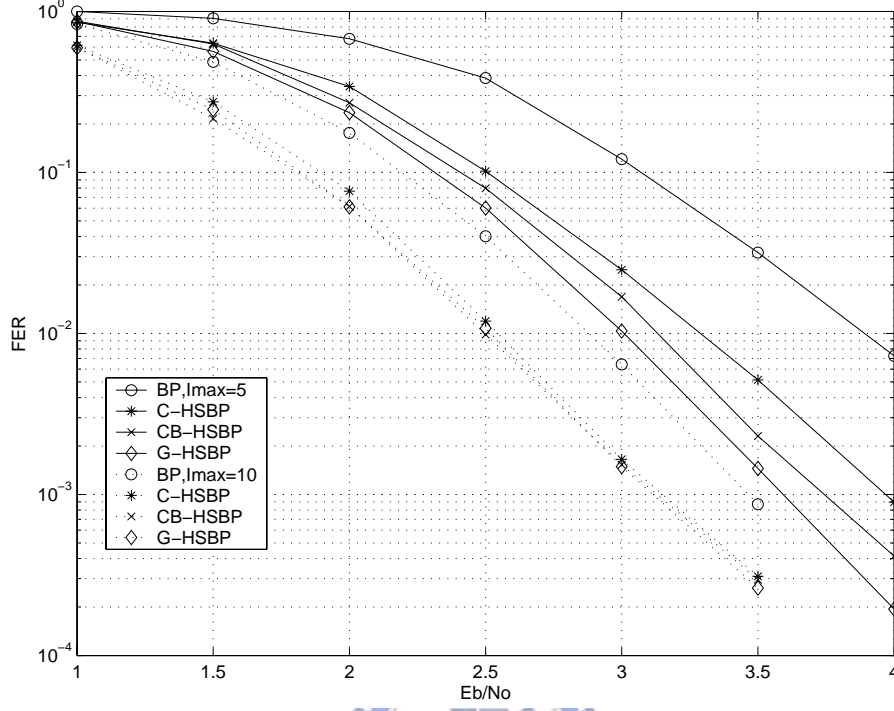


Figure 6.6: FER performance of the (480,240) QC-LDPC code with $I_{max}=5,10$, $G=4$.

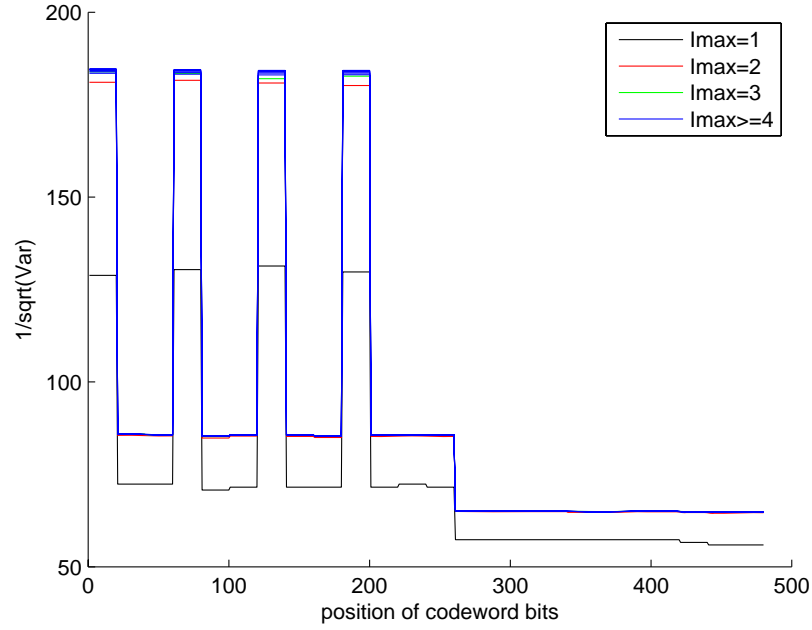


Figure 6.7: Message flow behavior of the (480,240) 802.11n specification QC-LDPC code.

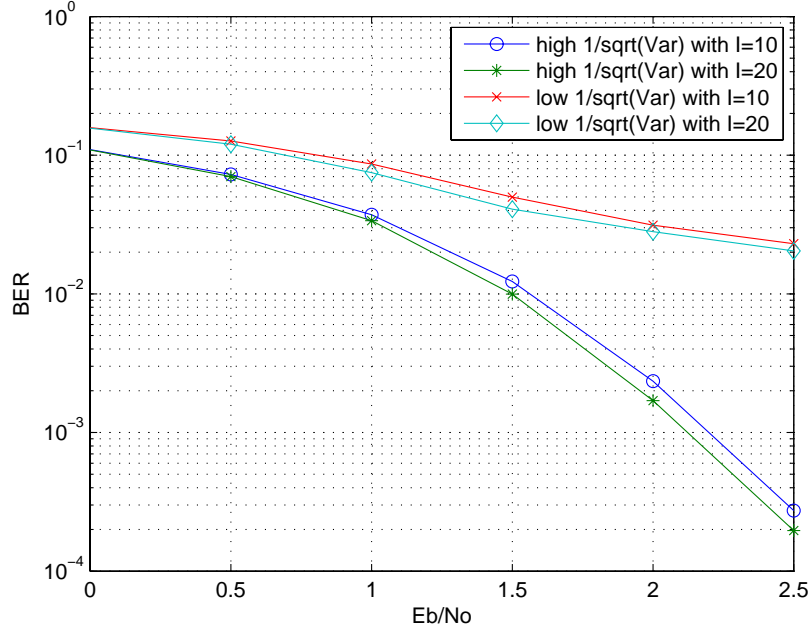


Figure 6.8: BER performance of different bits for (480,240) 802.11n specification QC-LDPC code.

and BER of low $1/\sqrt{\text{Var}}$ with different iteration, and as E_b/N_0 be enhanced the performance gap between the two groups would become larger. Compare Fig 6.7 with Fig 6.8, like we expect, those bits with high value in Fig 6.7 has better performance as shown in Fig 6.8. Since the difference of performance almost be more than 1dB after several iterations at high E_b/N_0 , the noise resisting ability of every bit in the same codeword may not be the same and has apparent distinction.

Fig 6.9 and Fig 6.11 show the message flow of another two LDPC codes, one is created from IEEE 802.16e specification with $L=480$ and another is shown in the Appendix A. Then Fig 6.10 and Fig 6.12 is their performance, for 802.16e we choose the value larger than 140 as high $1/\sqrt{\text{Var}}$ but in (96, 48) code the difference of $1/\sqrt{\text{Var}}$ is not apparent so we choose position (17 ~ 64) which have relative higher $1/\sqrt{\text{Var}}$. From Fig 6.7 to Fig 6.12 we see that the result of message flow could sometime predict the performance of the three code. The difference of high $1/\sqrt{\text{Var}}$ and low $1/\sqrt{\text{Var}}$

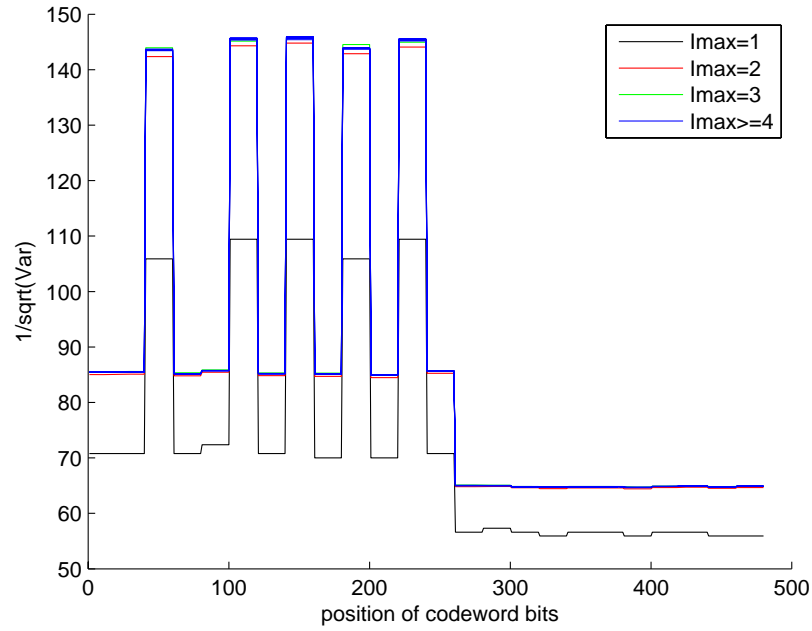


Figure 6.9: Message flow behavior of the (480,240) 802.16e specification QC-LDPC code.

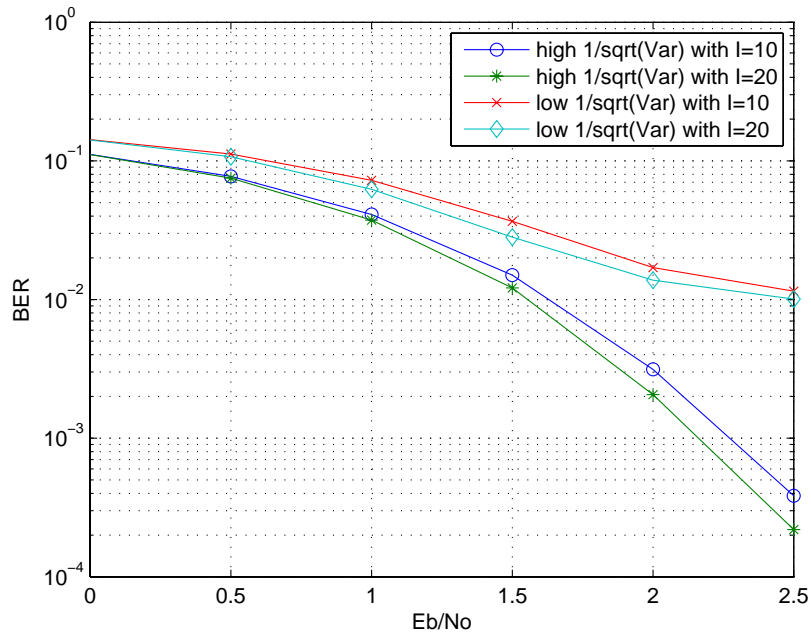


Figure 6.10: BER performance of different bits for (480,240) 802.16e specification QC-LDPC code.

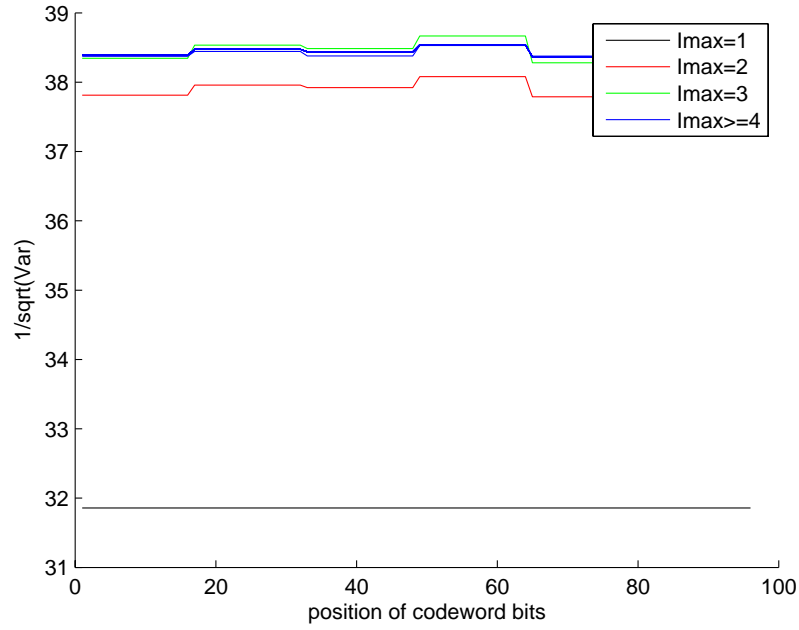


Figure 6.11: Message flow behavior of the (96,48) QC-LDPC code.

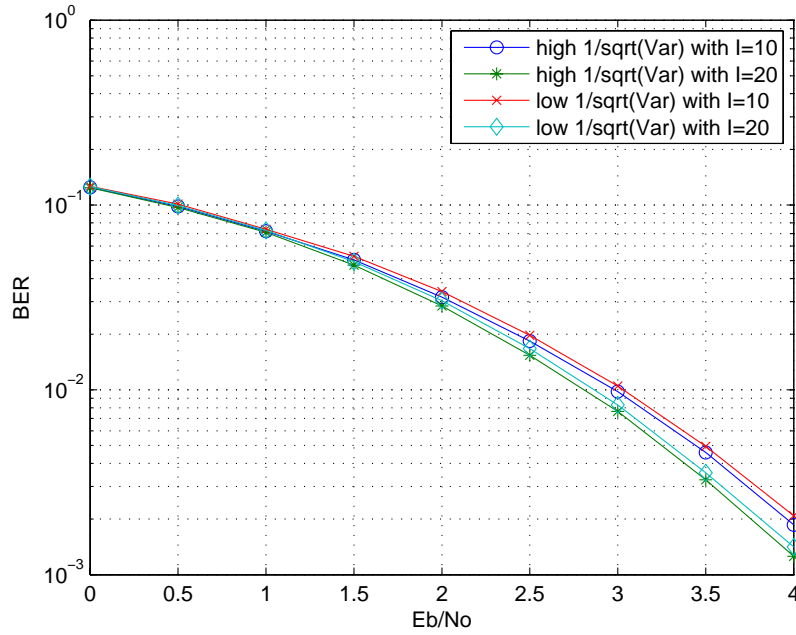


Figure 6.12: BER performance of different bits for (96,48) QC-LDPC code.

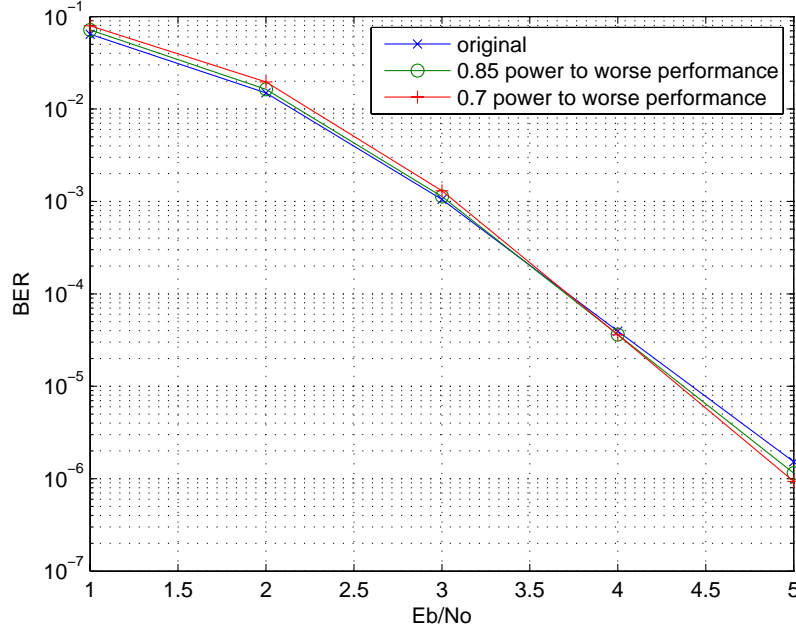


Figure 6.13: BER of 802.11n specification QC-LDPC code with power allocation at $I=5$.

would show the distinction of performance which is between the group with better performance and worse performance. There are larger $1/\sqrt{\text{Var}}$ difference in Fig 6.7, so the performance gap of high $1/\sqrt{\text{Var}}$ value and low $1/\sqrt{\text{Var}}$ value would be large. The same, Fig 6.11 has lower $1/\sqrt{\text{Var}}$ distinction and the performance gap would be small. In fact the $1/\sqrt{\text{Var}}$ difference of Fig 6.11 is very small and the corresponding performance gap is nearly negligible. Message flow can help us predict different error correcting ability of codeword bits, but small difference of convergence value is useless.

6.3 Power allocation and channel mapping

In this section we use the result from Fig 6.7 and Fig 6.8, and allot 0.85/0.7 times original power to those bits with higher $1/\sqrt{\text{Var}}$ in Fig 6.7 and the remaining power to other bits with worse performance.

Fig 6.13 shows the performance of power allocation and we can see that at 10^{-6} we

have nearly 0.15dB gain at $I=5$, so power allocation indeed improves the performance at high SNR. But now we face several questions about power allocation: First, the gain seems not large and performance with power allocation algorithm is better than original case at high SNR. Second, at transmitter the amplifier must change the amplitude frequently and it is almost impossible in current system. As we mentioned before, performance which can be improved is conjectured that every bits suffer different E_b/N_0 . So we extend the concept to channel mapping for frequency selective channel.

Because the channel response is not flat for frequency selective channel, different subcarrier would achieve different channel gain and suffers different E_b/N_0 equivalently. Assuming that we know the channel response of k -th subcarrier, we can write $Y_k = H_k \times X_k + N_k$ and estimate transmitting signal as $X_k + N_k/H_k$. Although transmitting power is equal in every bits, the term N_k/H_k will affect the relative E_b/N_0 . Large H_k will compress N_k and forms relative high E_b/N_0 , vice versa. By this idea we can achieve the same result of power allocation. The following shows three simple examples about channel mapping.

Fig 6.14 shows one channel response with small magnitude difference, the corresponding performance is in Fig 6.15. We can find that performance could be enhanced at high SNR when using channel mappings,. Then we discuss that why there is better performance at high SNR but worse performance at low SNR? Our explanation is that those bits with better performance always has good performance at high SNR if we put them in the subcarrier with small H_k . Compare to the case: we put bits in the subcarrier with large H_k , the numbers of error may be increased. Those bits with worse performance may have meliorative performance at high SNR and then reduce the number of error if we put them in the subcarrier with large H_k . Assuming that the error which is reduced is larger than the error which is increased and total error number is lower than Hamming distance, we could get correct codewords after several iterations. Fig 6.16 ~ Fig 6.19 show other frequency selective channels and performance cases. The three cases

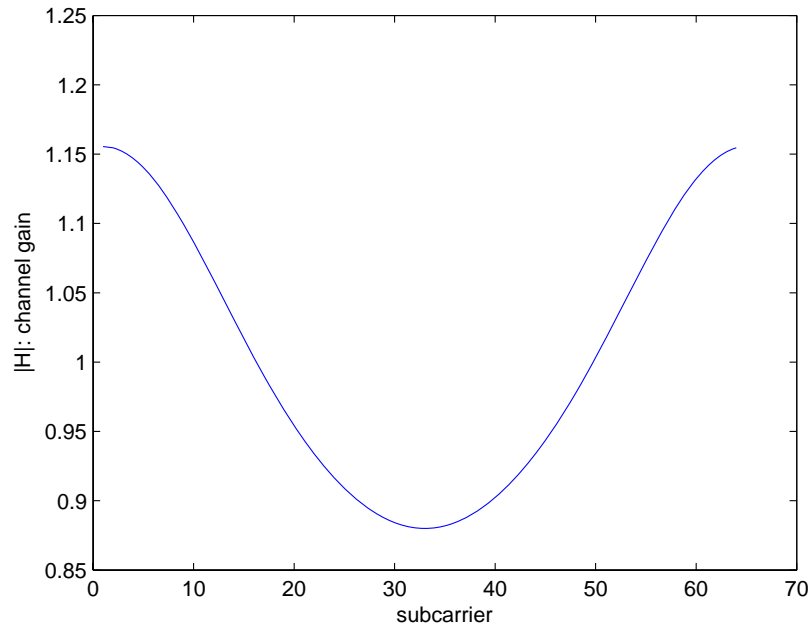


Figure 6.14: magnitude response for frequency selective channel with small difference of magnitude.

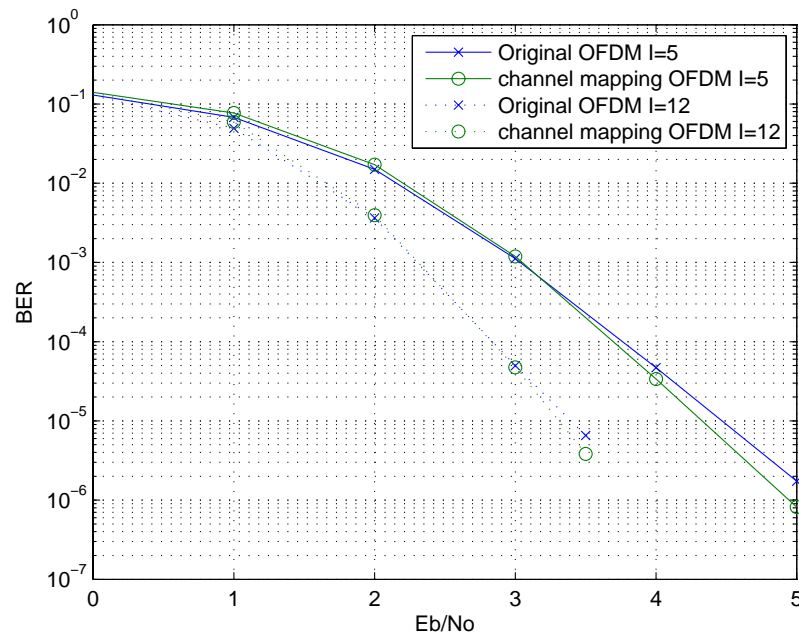


Figure 6.15: 802.11n specification performance for frequency selective channel with small difference of magnitude.

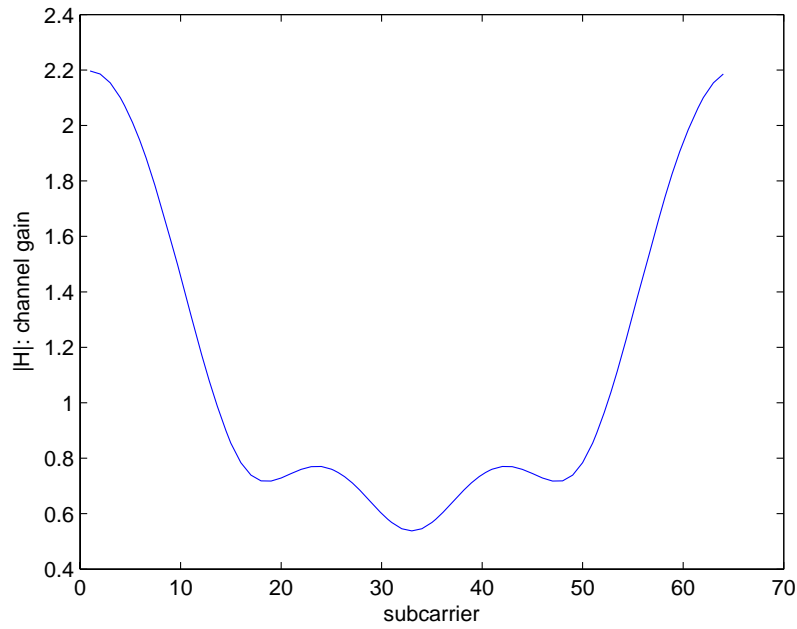


Figure 6.16: magnitude response for frequency selective channel with large difference of magnitude.

reveal that channel mapping can improve the performance of channel at high SNR and usually there are better improvements to those channels which have large difference of magnitude in frequency selective channels. Because LDPC codes created from 802.11n and 802.16e specification all have apparent bits with different noise resisting ability, we can use this skill to implement power allocation and channel mapping.

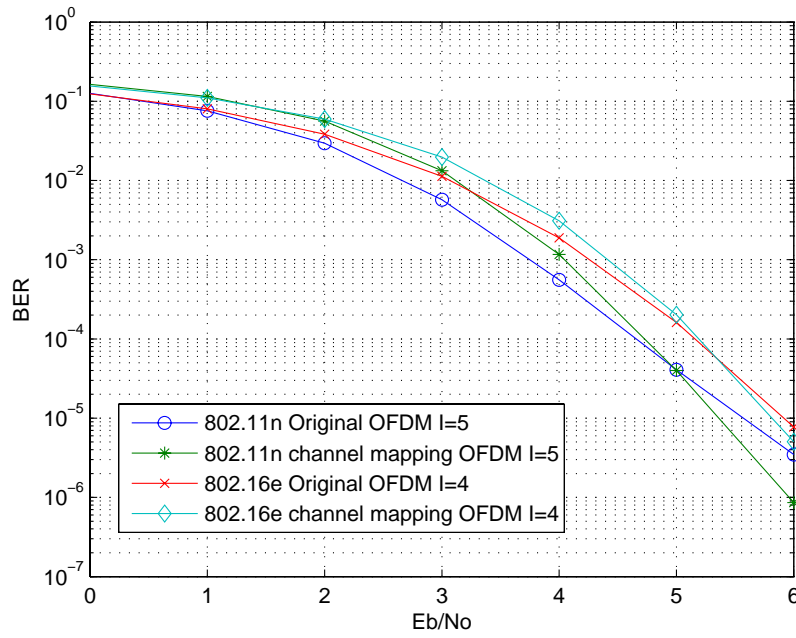


Figure 6.17: performance for frequency selective channel with large difference of magnitude.

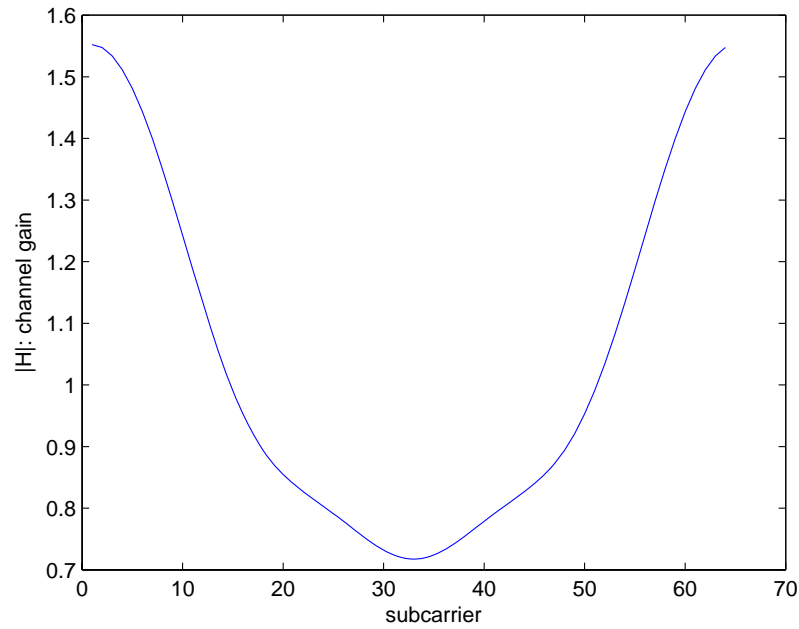


Figure 6.18: magnitude response for frequency selective channel with difference of magnitude between Fig 6.14 and Fig 6.16.

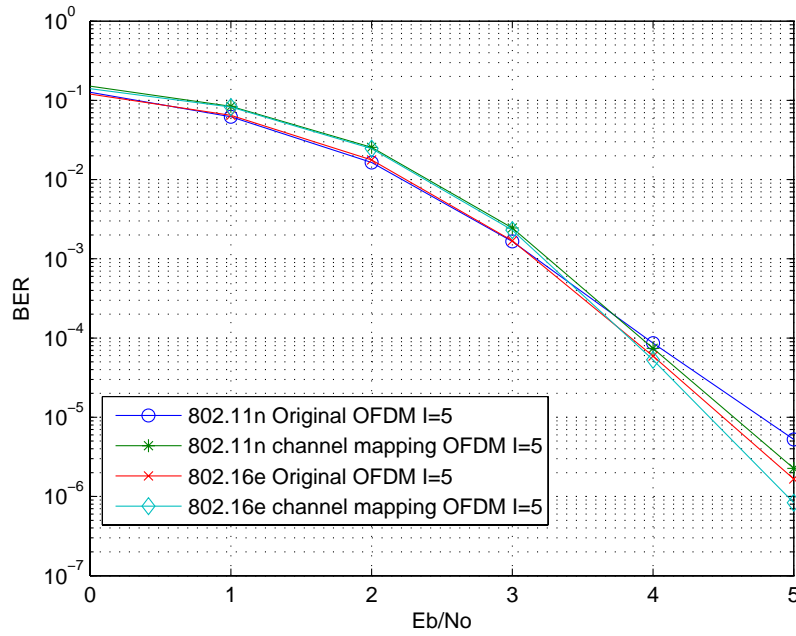


Figure 6.19: performance for frequency selective channel with difference of magnitude between Fig 6.14 and Fig 6.16.

Chapter 7

Conclusion

We provide two nodes partition approaches for BP decoding of LDPC codes. The first partition is related to the partition of the corresponding parity check matrix H and is based on the amount of uncorrelated information a node received to update its reliability per sub-iteration. The resulting shuffled BP decoding algorithms are shown to yield enhanced convergence and error rate performance when compared with conventional HSBP or VSBP algorithms. The second method is based on a parameter which measures the error protection a bit node received. We take advantage of the UEP nature of the irregular LDPC code and proposed a subcarrier and power allocation method that offers performance improvement.

We do not claim any optimality about the proposed partition and believe that an optimal partition and the corresponding decoding schedule do exist. A near-optimal partition based on our first criterion is one which results in submatrices with no all-zero rows or columns. An efficient algorithm to find such a partition remains unavailable.

The proposed subcarrier mapping and power allocation are based on the degrees of error protection. Since the ratio between the number of higher error protection bits and that of less-protected bits is not an integer, it is difficult to design a shuffled BP algorithm. We believe that by suitably combining the first partition approach, a shuffled BP algorithm that takes into account resource allocation strategy can be designed and further improved performance can be obtained.

Bibliography

- [1] R. G. Gallager, "Low-density parity-check codes," Cambridge, MA: MIT Press, 1963.
- [2] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 32, no. 18, pp. 1645-1646, Aug. 1996.
- [3] F.R. Kschischang, B.J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inform. Theory*, vol. 47, pp. 498-519, Feb. 2001.
- [4] B.J. Frey, F.R. Kschischang, H.-A. Loeliger, and N. Wiberg, "Factor graphs and algorithms," *Proc. 35th Allerton Conf. Communications, Control, and Computing*, Allerton House, Monticello, IL, Sept. 29-Oct. 1, 1997, pp. 666-680.
- [5] 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.
- [6] J. Zhang, M. P. C. Fossorier, "Shuffled Iterative Decoding," *IEEE Trans. Commun.* vol. 53, no. 2, pp. 209-213, Feb. 2005.
- [7] M. M. Mansour and N. R. Shanbhag, "High throughput LDPC decoders," *IEEE Trans. VLSI Systems*, vol. 11, pp. 976-996, Dec. 2003.
- [8] Ramesh Mahendra Pyndiah, "Near-Optimum Decoding of Product Codes: Block Turbo Codes" *IEEE Transaction on communication*, VOL. 46, NO. 8, August 1998.

- [9] "11-04-0889-04-000n-tgnsync-proposal-technical-specification", IEEE 802.11n Contribution, Mar. 2005.
- [10] IEEE 802.16e-04/78:Optional B-LDPC coding for OFDMA PHY.
- [11] Shihyao Wang, "Belief Propagation Based Decoding Schedules for Low-Density Parity Check Codes and their Behavior Analysis", NCTU, Hsin Chu, 2007.
- [12] Stephan ten Brink, "Convergence Behavior of Iteratively Decoded Parallel Concatenated Codes," *IEEE Transaction on communication*, VOL 49, NO. 10, October 2001.



Appendix A

Structure for a (96,48) LDPC code

The following is a (96,48) LDPC code used in Chapter 6, the elements of row vectors means the "1" position. EX: **5:** 1 3 7 represent H at row 5 has three "1" in column 1, 3, 7 and other columns are all "0".



1: 2 29 33 51 68 84	2: 3 30 34 52 69 85	3: 4 31 35 53 70 86
4: 5 32 36 54 71 87	5: 6 17 37 55 72 88	6: 7 18 38 56 73 89
7: 8 19 39 57 74 90	8: 9 20 40 58 75 91	9: 10 21 41 59 76 92
10: 11 22 42 60 77 93	11: 12 23 43 61 78 94	12: 13 24 44 62 79 95
13: 14 25 45 63 80 96	14: 15 26 46 64 65 81	15: 16 27 47 49 66 82
16: 1 28 48 50 67 83	17: 16 31 46 61 75 92	18: 1 32 47 62 76 93
19: 2 17 48 63 77 94	20: 3 18 33 64 78 95	21: 4 19 34 49 79 96
22: 5 20 35 50 80 81	23: 6 21 36 51 65 82	24: 7 22 37 52 66 83
25: 8 23 38 53 67 84	26: 9 24 39 54 68 85	27: 10 25 40 55 69 86
28: 11 26 41 56 70 87	29: 12 27 42 57 71 88	30: 13 28 43 58 72 89
31: 14 29 44 59 73 90	32: 15 30 45 60 74 91	33: 10 25 39 56 70 85
34: 11 26 40 57 71 86	35: 12 27 41 58 72 87	36: 13 28 42 59 73 88
37: 14 29 43 60 74 89	38: 15 30 44 61 75 90	39: 16 31 45 62 76 91
40: 1 32 46 63 77 92	41: 2 17 47 64 78 93	42: 3 18 48 49 79 94
43: 4 19 33 50 80 95	44: 5 20 34 51 65 96	45: 6 21 35 52 66 81
46: 7 22 36 53 67 82	47: 8 23 37 54 68 83	48: 9 24 38 55 69 84

Base matrix of IEEE 802.11n and 802.16e

depth process is in [10] and

in the following.

0	0	-1	0	-1	0	-1	-1	-1	0	-1	-1	1	0	-1	-1	-1	-1	-1	-1
29	-1	0	26	-1	-1	0	-1	0	-1	-1	-1	0	0	-1	-1	-1	-1	-1	-1
-1	-1	-1	21	0	-1	17	-1	-1	38	-1	0	-1	0	0	-1	-1	-1	-1	-1
43	-1	-1	30	-1	-1	-1	0	-1	41	0	-1	-1	-1	0	0	-1	-1	-1	-1
5	-1	1	-1	-1	20	35	-1	1	2	-1	-1	-1	-1	-1	0	0	-1	-1	-1
-1	46	-1	-1	-1	-1	22	-1	40	8	-1	-1	0	-1	-1	-1	0	0	-1	-1
-1	-1	-1	9	-1	-1	18	13	-1	35	-1	27	-1	-1	-1	-1	0	0	-1	-1
2	-1	44	-1	-1	-1	27	-1	-1	25	18	-1	-1	-1	-1	-1	-1	0	0	-1
33	35	-1	29	-1	-1	16	-1	-1	-1	30	-1	-1	-1	-1	-1	-1	0	0	-1
-1	-1	-1	4	-1	-1	-1	15	17	-1	-1	-1	-1	-1	-1	-1	-1	-1	0	-1
5	-1	-1	19	-1	14	-1	-1	-1	-1	11	-1	-1	-1	-1	-1	-1	-1	-1	0
10	-1	-1	-1	21	-1	18	8	-1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1	0

Figure B.1: Base matrix of 802.11n

簡歷

姓名：顏佐翰

出生地：台中縣清水鎮

生日：1984/05/06

學歷：台中縣清水國小

台中縣沙鹿國中

台中市台中一中

交通大學電信工程系

交通大學電信工程研究所

