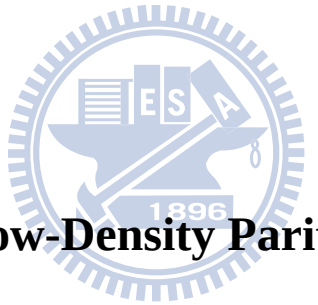
國立交通大學
電機與控制工程學系

碩　士　論　文

以優先權為基礎之消去迴圈演算法
建構低密度同位元檢查碼



Constructing Low-Density Parity-Check Codes by Priority Based Cycle Elimination Algorithm

研 究 生：連昶翔
指導教授：董蘭榮 博士

中 華 民 國 九 十 八 年 十 月

以優先權為基礎之消去迴圈演算法
建構低密度同位元檢查碼

Constructing Low-Density Parity-Check Codes by Priority Based Cycle Elimination Algorithm

研 究 生：連昶翔　　Student：Chang-Hsiang Lien

指導教授：董蘭榮 博士　　Advisor：Lan-Rong Dung

國 立 交 通 大 學

電 機 與 控 制 工 程 學 系

碩 士 論 文

A Thesis
Submitted to Department of Electrical and Control Engineering
College of Electrical Engineering and Computer Science
National Chaio-Tung University
in Partial Fulfillment of the Requirements
for the Degree of Master
in
Electrical and Control Engineering
October 2009
Hsinchu, Taiwan, Republic of China

中 華 民 國 九 十 八 年 十 月

以優先權為基礎之消去迴圈演算法
建構低密度同位元檢查碼

研 究 生：連昶翔　　指導教授：董蘭榮 博士

國立交通大學電機與控制工程學系

中文摘要

低密度同位元檢查碼具有優異的解碼能力及硬體實現的低複雜度，近年來受到廣大討論與研究，其中一部分研究為設計具有較大周長或是消除更多的短迴圈的同位檢查矩陣以得到更好的解碼效能。本論文提出以優先權為基礎的消去迴圈演算法，統計每個元素包含的迴圈數量高低排列優先順序，如此可有效率打斷迴圈的連結，故本改良式演算法能大幅降低建構矩陣的運算量。此外設計低編碼長度的位元檢查矩陣時因為優先權的機制能更有效率打斷迴圈，因此相較其它兩者類似架構的演算法能消除更多的迴圈，得到效能上的增進，尤其是在較高訊號雜訊比的情況下差異較為明顯，適用於低功耗或低運算量等通訊系統。

Constructing Low-Density Parity-Check Codes by Priority Based Cycle Elimination Algorithm

Student：Lien Chang-Hsiang Advisor：Lan-Rong Dung

Institute of Electrical and Control Engineering National Chiao-Tung University

Abstract

In recent years, Low-Density Parity-Check Codes have attracted a lot of attention and discussion due to great decoding ability and low complexity of hardware implementation. Some research focuses on performance improvement by designing high performance coding with large girth. In this thesis, we propose a priority based cycle elimination algorithm. It is efficient to eliminate cycle by setting the priority based on the number of dependent cycles. As shown in the results, the proposed algorithm can significantly reduce the complexity in operation. It can also construct high-performance codes and eliminate more cycles than traditional approaches for short code-length applications. Comparing with the other algorithms, the proposed algorithm can have better decoding performance, especially in high SNR environment; hence, our algorithm can satisfy the requirement of low-power communication systems.

誌　謝

這篇論文的完成要感謝許多人，首先感謝老師兩年多來的細心指導與督促，讓我確定研究的方向，多方面意見的交流使我改進不足的地方，給我許多寶貴的意見而能繼續研究下去。

再來感謝實驗室的夥伴，謝謝學弟們幫忙我許多事情，使我更能專心的研究省去一些生活上的不便，謝謝致翰、嘉鴻以及智勝，在課業及生活中給予我許多的幫助，並留下許多美好的回憶。

最後我要感謝我的家人，一直在背後照顧、栽培我的父親與母親以及可愛的弟弟，感謝你們一路支持我、使我無後顧之憂專心完成學業，在此獻上最深的敬意與感謝。

98 年 10 月

目　錄

圖 目 錄

表 目 錄

第一章　導論

本章節介紹論文主題之相關背景及研究的動機，並在論文組織中簡述各章節內容。

1.1 研究背景與動機

在通訊技術日益進步的現代，人們越來越倚靠通訊所帶給人類的便利性。無論是最近熱門的無線網路系統、WI-FI、WiMAX、新一代手機通訊系統等技術的發展一日千里，因為這部分擁有廣大的商機及市場，眾多研究機構及廠商投資大量人力及金錢進行研發工作，而隨著傳輸速度增加及通訊品質的要求，尤其對於高速無線系統來說環境所造成雜訊干擾甚大，有時還會受到頻率範圍的限制對整體通訊品質受到影響，因此大家都專注在如何讓無線傳輸的品質變好，主要的解決辦法就是使用錯誤更正碼提升傳輸效率及錯誤更正能力。

在傳輸系統中資料傳輸速度有其上限值稱為通道容量，只要傳輸的訊號強度及速度不超過此值均可以使用適當的編、解碼達到良好的錯誤更正能力。1948年夏農(Shannon)發表重要的通道理論[1]，在傳輸過程中如果速率小於通道容量，就可以達到很小的錯誤率，雖然未指出哪種編碼方式的效能可以達到很小的錯誤率，但此理論成為指標也發展了許多的編碼方式[2]，這些編碼主要分成兩大類，一類為線性區塊碼[3](Linear Block Codes)、另一類為迴旋碼[4](Convolutional Code)，線性區塊碼目前較為熱門有循環碼[5](Cyclic Codes)、里德-所羅門碼[6](Reed-Solomon Codes)、低密度同位檢查碼(Low-Density Parity-Check Codes)[7]，而迴旋碼中熱門有渦輪碼[8](Turbo Codes)，其中又以低密度同位檢查碼跟渦輪碼最受重視。低密度同位檢查碼簡稱 LDPC Codes 早在 1962 年就由 Gallager 提出，當時因為電腦與超大型積體電路不發達，因此無

法負擔複雜運算量高的 LDPC Codes，直到近年來電腦運算發達的現代才逐漸被重視，並發現他擁有優異的錯誤更正能力[9][10]，開始有許多人踏入研究 LDPC Codes 的領域。

LDPC Codes 解碼方式類似 Turbo Codes，以重覆遞迴更新每一位元的機率資訊來找出最正確的 codes，具有高錯誤更正率、硬體實現度高且可平行化、Low error-floor 等特性，同時 LDPC 屬於線性區塊碼，一次能處理幾千甚至上萬個位元數，再利用解碼平行化的實現[11]，所以能達到很高的吞吐量[12]，但缺點為編碼的複雜度高，現在有幾篇已發表的論文如[13]使編碼的複雜度變低，加速編碼的速度。

構成一個好的 LDPC 有兩條重要的因素，一為矩陣中元素 1 的分佈[14]，另一為短迴圈(short cycle)的影響，因為 LDPC 解碼方式為重覆遞迴去逼近於正確的碼字，解碼的過程與訊息傳遞之間的獨立性有很大的關係，在已有的文獻指出 LDPC 要有良好的更正能力必須讓訊息傳送盡可能的獨立不受干擾，所以研究為如何設計一個具有高周長(Girth)的位元檢查矩陣為本論文研究的動機之一，另外 BLOCK-LDPC 的架構在硬體實現上較為容易且能平行化增加產出，結合這兩項因素形成消去迴圈演算法的主要理論與架構。

本篇論文主要探討建構 LDPC Codes 的演算法設計，改良原有消去迴圈演算法(Cycle Elimination Algorithm)，由於 CE 演算法本身為近似暴力法的方式實現，故運算量及複雜度高，且演算法運作特性使然在低擴展倍數下打斷迴圈的效果有限，因此我們提出以 CE 演算法為基礎加入優先權的概念而改良的演算法，找尋該元素擁有最大關聯性也就是包含最多迴圈優先動作，如此可在每次動作時有效率的打斷更多迴圈，不僅大量減低演算法的運算複雜度，在設計低編碼長度小於 4000 位元的矩陣能消除更多的短迴圈進而增加解碼效能，模擬結果也顯示相較於其它兩種演算法效能上的改進，所以適用於行動或無線通訊系統為了低運算量及省電而採取低編碼長度的解碼應用。

1.2 論文組織

本篇論文將以下的章節劃分為五個章節，茲分述如下：

第一章簡述錯誤更正碼的發展，說明 LDPC 碼的特性與研究背景，最後提出本論文的動機與方向。

第二章則介紹本論文之背景知識作，內容共分為三個部份。第一節介紹 LDPC 的基本原理與特性；第二節介紹 LDPC 的兩種編法方式，一種為傳統式編碼，複雜度會隨著矩陣大小而有明顯的改變，另一種為低運算量的編碼方式，尤其針對長矩陣的編碼能有效減低運算複雜度。

第三章介紹傳統式消去迴圈演算法，除了說明演算法流程之外，並分析演算法的運算複雜度與優缺點。

第四章介紹改良式消去迴圈演算法，詳盡說明演算法的步驟與要點，此外以兩個範例矩陣證明具有快速消除迴圈的效果，也簡化大量的運算複雜度，最後提出以較小的編碼長度的解碼效能略優於傳統式的消去迴圈演算法。

第五章為結論與未來研究方向之提議。

第二章　低密度同位檢查碼

本章將介紹 LDPC 碼的基本概念，包含了 LDPC 的特性、建構同位檢查矩陣的種類以及編碼、解碼的方式。

2.1 低密度同位檢查碼簡介

此節敘述 LDPC 基本概念及特性，並介紹最早所提出的 Gallager Codes 架構。

2.1-1 基本概念及特性介紹

低密度同位檢查碼(LDPC Codes)是一種線性區塊碼，因此在編碼及解碼時會將一連串的來源訊息分段處理，每一段的位元長度為 K。編碼時將此 K bits 與生成矩陣 $G_{K\times N}$ 相乘後得到長度為 K+M 的碼字(Code word)，其中 M bit 為檢查位元，我們可以表示為:

$$v_{1\times N} = s_{1\times K} G_{K\times N} \qquad (1)$$

編碼後的資料 v 經由通道傳輸到接收端，再利用同位檢查矩陣 H (Parity-Check matrix) 檢查收到的碼是否正確並加以修正。

H 矩陣分成正規矩陣與非正規矩陣(regular codes)，所謂正規矩陣指 H 矩陣中每一行及每一列中 1 的個數相同，若一個 m x n 大小 H 矩陣中每列含有 k 個 1、每行含有 j 個 1 則定義為 (n,j,k)-regular LDPC Codes，其中 j、k 稱為行權重 (column weight) 及列權重 (row weight) ，並不是每個 H 矩陣都有相同的權重，而這些矩陣為非正規矩陣(irregular codes)。

為了方便我們了解 LDPC 運作的情形，Tanner[15]提出了 Tanner Graph 也稱為 Bipartite Graph，以圖 2-1 的(7,3,3) 正規檢查矩陣為例，它的 Tanner

Graph 表示為圖 2-2，每一列均可用一個節點代表稱做檢查節點(check node)，每一行均可用一個節點代表稱做位元節點(check node)，而檢查節點與位元節點中間的連線稱做 edge，這些 edge 將各節點的訊息互相傳輸。

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}_{M \times N}$$

圖 2-1 (7, 3, 3) 正規檢查矩陣

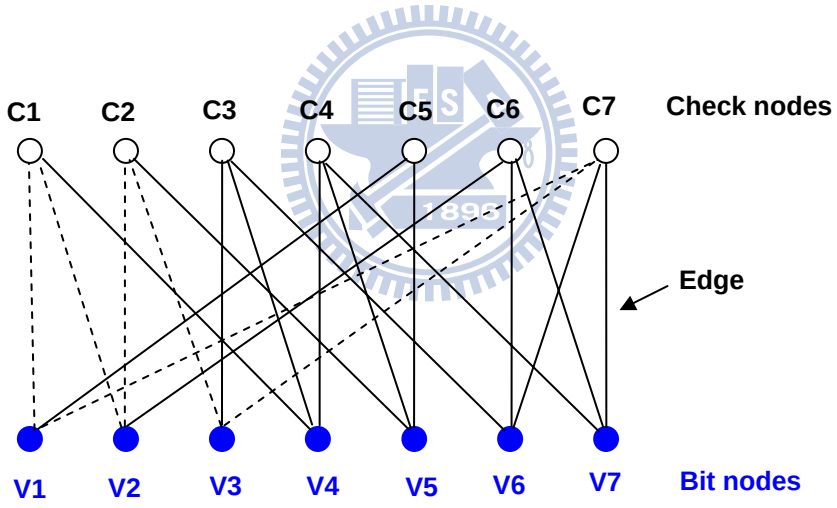


圖 2-2 (7, 3, 3) 正規檢查矩陣之 Tanner Graph

迴圈 (Cycle) 指從一個節點出發，經過幾條 edge 之後再回到同一個節點，而這個路徑是封閉的且同一節點不能通過兩次以上，如圖 2-2 中虛線所示，從 V1 節點出發經由 edge 通過 C1、V2、C2、V3、C7 共六點及六條 edge 回到 C1，這是一條封閉的 Cycle 且因為通過六個點我們稱為 Cycle-6，一個矩陣所有的 Cycle 中最小 Cycle length 稱作 Girth。

LDPC 的同位檢查矩陣除了影響編碼的複雜度之外，對解碼效能的影響更是巨大，所以我們一般說建構 LDPC Codes 就是指設計它的 H 矩陣。通常 H 矩陣設計上有兩個重點，一為矩陣的大小，因為矩陣的大小、行數等於 Code length，在 Gallager[7]文獻中提到，一個亂數產生的(n, j, k)矩陣最小距離(minimum distance) $d_{\min} \geq N\sigma(j,k)$ 其中 $\sigma(j,k)>0$，所以當 code length N 越長時最小距離也跟著變大，也就是有較好的效能；另一為 Cycle length 的大小，Tanner 在[14]文獻中指出，如果 LDPC Codes 有越大的 Girth 時代表解碼時能有更多獨立的遞迴運算使錯誤更正能力增加。因此 H 矩陣中短 Cycle 是盡量避免的，這部分也是本論文的重點，利用改良的演算法使短 Cycle 消除。

2.1-2 Gallager Codes

前文曾提到 LDPC Codes 共分為 regular 及 irregular 兩種，Gallager LDPC 則是屬於 regular 隨機分配的矩陣，具有以下定義及特性。一個 (n, j, k)-regular 的矩陣含有 nj / k 個列，矩陣大小為 M x N，而它的檢查位元(parity bit)長度為 k=M-N，此矩陣的排列架構如式(2)表示：

$$H = \begin{bmatrix} H_1 \\ H_2 \\ \vdots \\ H_j \end{bmatrix} \qquad (2)$$

對於每一個子矩陣 H_i , $i = 1,2,...,j$ 大小為 p x (p x k)、列權重為 k 及行權重為 1。第一個子矩陣 H_1 的分佈似階梯狀，如圖 2-3 所示在第一列中有連續 3 個 1 排列，因此 1 的排列範圍為 $(i-1)k+1 \text{ to } ik$, $for\ i = 1,2,...,p$ ，此處 k=3，其它部分的子矩陣是配合第一列所做的隨機排列矩陣。

$$H_1 = \begin{bmatrix} 1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,1\,1\,1\,0\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,1\,1\,1\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1 \end{bmatrix}$$

圖 2-3 Gallager LDPC Codes H1 的區塊矩陣

2.1-3 Mackay Codes [16]

Mackay 提出一個隨機產生位元檢查矩陣的方法，建構矩陣時為一行一行疊上去，如第一行元素 1 的位置是隨機的，但第二行開始元素 1 的位置不能與第一行相同以免造成 cycle-4 的迴圈產生，而之後的行數每疊上去都會檢查之前行數 1 的位置避免在同一列，因此所建構而成的矩陣完全沒有 cycle-4 的迴圈存在，可在解碼時增加錯誤更正能力以提升效能。

2.2 LDPC 編碼方式介紹

LDPC 為線性區塊碼，因此可利用高斯消去法產生 G，但因為此種方式隨著 code length 增加而需要大量的運算及複雜度，所以部分研究針對 LDPC 編碼提出有效率的編碼方法，以下的內容也會一併介紹。

2.2-1 傳統編碼方式

令原始訊息為 $u=[u_0,u_1,...,u_{k-1}]$ ，經由生成矩陣 G 得到碼字 $c=[c_0,c_1,...,c_{n-1}]$，可表示為:

$$c=u \cdot G \qquad (3)$$

其中 G 的大小為 k x n。因為 H 矩陣與 G 矩陣維度互相正交使相乘之後為零，故 H 的大小為(n-k) x n，所以可由以下公式推導出 $c \cdot H^T = 0$。

$$\begin{aligned} & G \cdot H^T = 0 \\ & \Rightarrow uG \cdot H^T = 0 \ (\because u \neq 0) \\ & \Rightarrow c \cdot H^T = 0 \end{aligned} \tag{4}$$

如果 H 矩陣為列滿秩則由高斯消去法得到

$$H_{M \times N} \leftrightarrow H_{sys} = \left[I_{(M-N) \times N} \mid P_{(M-N) \times N} \right] \tag{5}$$

其中 I 為單位矩陣，因為 $G \cdot H^T = 0$，故

$$\begin{aligned} & G_{sys} \cdot H^T{}_{sys} = 0 \\ & \Rightarrow G_{sys} \cdot \left[\frac{I_{(M-N) \times N}}{P^T{}_{(M-N) \times N}} \right] = 0 \\ & \Rightarrow G_{sys} = \left[P^T{}_{(M-N) \times N} \mid I_{(M-N) \times N} \right] \end{aligned} \tag{6}$$

在剛剛的高斯消去法中得到矩陣不一定是標準的[I | P]格式，還需要經過行與行之間的置換，所以產生的 $G_{sys} \leftrightarrow G$ 轉換成正確的 G 也要重做行之間的置換。

2.2-2 Richardson 編碼演算法 [13]

H 矩陣經高斯消去法可得到如圖 2-4 的矩陣分配，但不是任意矩陣均可由高斯消去法加上行轉換形成此種矩陣，因此 Richardson 提出一種有效率的編碼適用於任何矩陣，且減少矩陣運算的複雜度及運算量。

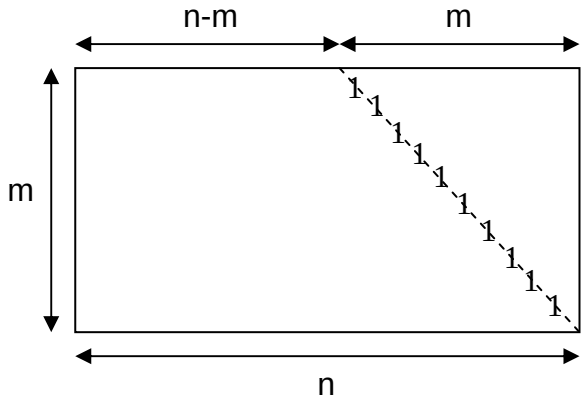


圖 2-4 H 矩陣的下三角矩陣格式

首先將 H 矩陣經過行與列的排列之後形成圖 2-5 的形式，此形式表示為

$$H = \begin{bmatrix} A & B & T \\ C & D & E \end{bmatrix} \tag{7}$$

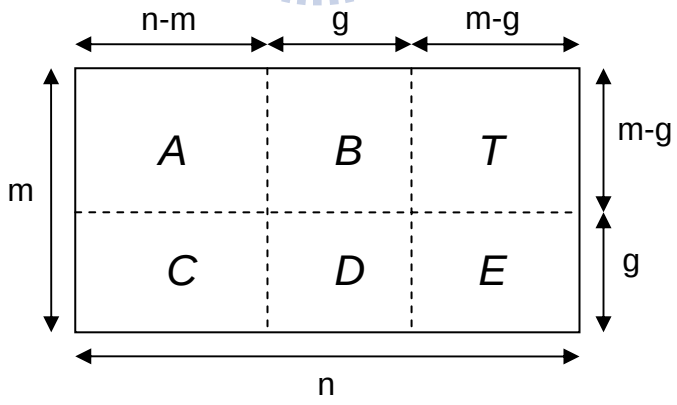


圖 2-5 Richardson 定義之矩陣格式

矩陣中各子矩陣的大小上圖所標示，其中 T 為下三角矩陣，將此矩陣的左邊乘以下述的子矩陣

$$\begin{bmatrix} I & 0 \\ -ET^{-1} & I \end{bmatrix} \cdot \begin{bmatrix} A & B & T \\ C & D & E \end{bmatrix} \tag{8}$$

得到

$$\begin{bmatrix} A & B & T \\ -ET^{-1}A+C & -ET^{-1}B+D & 0 \end{bmatrix} \tag{9}$$

令編碼完的碼字分成三部分 $c=(s,p_1,p_2)$，s 為長度(m-n)資訊位元、p1 及 p2 分別為長度 g 與(m-g)的檢查位元，因為 $c\cdot H^T=0$，所以將 c 與式(9)相乘得到兩條式子

$$\begin{cases} As^T+Bp_1^T+Tp_2^T=0 \\ (-ET^{-1}A+C)s^T+(-ET^{-1}B+D)p_1^T=0 \end{cases} \tag{10}$$

設 $r=-ET^{-1}B+D$ 為非奇異矩陣可逆，代入式(10)運算得

$$p_1^T=-r^{-1}(-ET^{-1}A+C)s^T \tag{11}$$

p_1^T 得到後再代回式(10)得

$$p_2^T=-T^{-1}(As^T+Bp_1^T) \tag{12}$$

有了 p_1,p_2 就可代回原來分段的碼字中得到 c，即為編碼後的結果，整體的運算複雜度列於表 2-1 與表 2-2。

算式	內容	複雜度
As^T	稀疏矩陣相乘	$O(n)$
$T^{-1}[As^T]$	$T^{-1}[As^T]=y^T \Leftrightarrow [As^T]=Ty^T$	$O(n)$
$-E[T^{-1}As^T]$	稀疏矩陣相乘	$O(n)$

Cs^T	稀疏矩陣相乘	$O(n)$
$[-ET^{-1}As^T]+[Cs^T]$	相加	$O(n)$
$-r^{-1}[-ET^{-1}As^T+Cs^T]$	密集矩陣相乘	$O(g^2)$

表 2-1 p_1 複雜度分析

算式	內容	複雜度
As^T	稀疏矩陣相乘	$O(n)$
Bp_1^T	稀疏矩陣相乘	$O(n)$
$[As^T]+[Bp_1^T]$	相加	$O(n)$
$-T^{-1}[As^T+Bp_1^T]$	$-T^{-1}[As^T+Bp_1^T]=y^T \Leftrightarrow -[As^T+Bp_1^T]=Ty^T$	$O(n)$

表 2-2 p_2 複雜度分析

2.3 LDPC 解碼方式介紹

LDPC 的解碼方式採用遞迴運算(Iterative Decoding)，利用多次重覆傳送位元節點與檢查節點之間的訊息使解碼結果逼近原始資料，通常遞迴的次數越多則解碼值越正確，此外為了方便硬體實現及簡化運算複雜度，解碼時大多採用 LLR 的形式[17]。本節將介紹常用的三種解碼演算法，這些演算法均屬於軟性決策(Soft decision)解碼而有優秀的錯誤更正能力。

2.3-1 Sum-Product Algorithm [18]

Tanner Graph 的圖形解碼使我們容易了解 LDPC 解碼的過程，在說明整體演算法流程之前，先舉簡單的例子表達訊息傳遞及推導機率公式的過程，其中部分過程參考[19]。

這邊有一個 2x4 的位元檢查矩陣：

$$H = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \tag{13}$$

設收到的碼字 code word 為 c=(c1, c2, c3, c4)，要判斷是否為正確的碼字則使之與位元檢查矩陣相乘得到

$$\begin{aligned} & c \cdot H^T = 0 \\ & \Rightarrow (c1, c2, c3, c4) \cdot \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} = 0 \\ & \Rightarrow \begin{cases} c1 \oplus c4 = 0 & (equation\ S1) \\ c1 \oplus c2 \oplus c3 = 0 & (equation\ S2) \end{cases} \end{aligned} \tag{14}$$

其中符號⊕代表 modulo-2 加法運算，所得到的兩條方程式稱為位元檢查方程式(parity check equation)，而方程式的數量與列的數量相同。

位元檢查矩陣由 Tanner Graph 表示如下

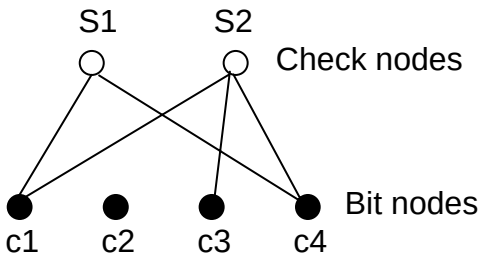
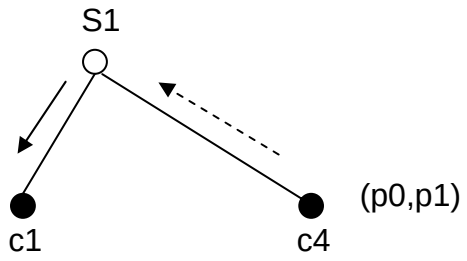


圖 2-6　式 2-13 之 Tanner Graph

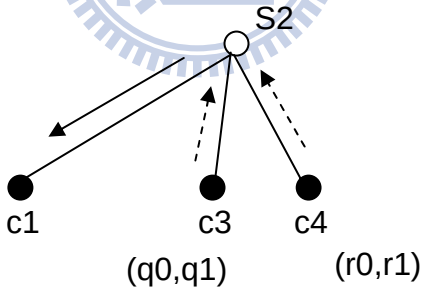
S1 跟 S2 分別代表兩條位元檢查方程式，c1~c4 代表 code word 中 4 個位元的點，首先我們對 c1 做單一位元的解碼：

位元 c4 的 priori-probability 為 0 與 1 的機率是 p0 與 p1，其中 p0+p1=0。因為透過 S1 的傳遞給 c1 訊息的位元節點有 c4，所以經由 S1 方程式 $c1 \oplus c4 = 0$ 可以得到



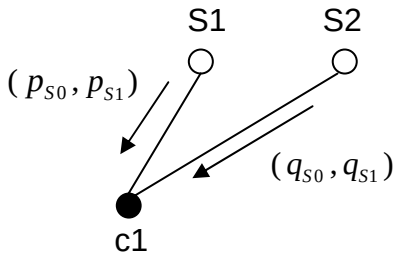
$$\begin{cases} P(c1=0) = P(c4=0) = p_0 \\ P(c1=1) = P(c4=1) = p_1 \end{cases} \tag{15}$$

同理透過 S2 的位元節點有 c3、c4，經由 S2 方程式 $c1 \oplus c2 \oplus c3 = 0$ 得到



$$\begin{cases} P(c1=0) = P(c2 \oplus c3 = 0) \\ = P(c2=0)P(c3=0) + P(c2=1)P(c3=1) = q_0 r_0 + q_1 r_1 \\ P(c1=1) = P(c2 \oplus c3 = 1) \\ = P(c2=1)P(c3=0) + P(c2=0)P(c3=1) = q_1 r_0 + q_0 r_1 \end{cases} \tag{16}$$

因為 S1 及 S2 都有傳遞機率訊息給 c1，因此 c1 要整合這兩組資訊，我們由式(17)表示：



$$\begin{cases} P(c1=0) \in P(S1=0 \text{ and } c1=0)P(S1=0 \text{ and } c1=0) = p_{S0}q_{S0} \\ P(c1=1) \in P(S1=0 \text{ and } c1=1)P(S1=0 \text{ and } c1=1) = p_{S1}q_{S1} \end{cases} \quad (17)$$

其中 $p_{S0} = p_0$, $p_{S1} = p_1$, $q_{S0} = q_0 r_0 + q_1 r_1$ and $q_{S1} = q_1 r_0 + q_0 r_1$ 。

以 S2 檢查節點來看，它要整合 c3 及 c4 的機率資訊，所以我用下式代表 c1 得到的機率：

$$pS2(q_0, q_1, r_0, r_1) = (q_0 r_0 + q_1 r_1, q_1 r_0 + q_0 r_1) \quad (18)$$

因為任一點 0 與 1 的機率總合為 1 也就是 $p_0 + p_1 = 1$ ，為了簡化運算採用 LLR(Log-Likelihood Ratio)的型式

$$L(p_0, p_1) = \ln \frac{p_0}{p_1} = \ln \lambda \quad (19)$$

代入式(18)得

$$pS2(L_1, L_2) = pS2(L_1 \oplus L_2) = \ln \frac{q_0 r_0 + q_1 r_1}{q_1 r_0 + q_0 r_1} = \ln \frac{1 + \frac{q_0 r_0}{q_1 r_1}}{\frac{r_0}{r_1} + \frac{q_0}{q_1}}$$

$$= \ln\frac{1+\lambda_1\lambda_2}{\lambda_1+\lambda_2} = \ln\frac{1+e^{L_1}e^{L_2}}{e^{L_1}+e^{L_2}} = \ln\frac{e^{-\frac{L_1+L_2}{2}}+e^{\frac{L_1+L_2}{2}}}{e^{\frac{L_1-L_2}{2}}+e^{-\frac{L_1-L_2}{2}}}$$
$$= \ln(\cosh(\frac{L_1+L_2}{2})) - \ln(\cosh(\frac{L_1-L_2}{2})) \quad (20)$$
$$= 2\tanh^{-1}(\tanh(\frac{L_1}{2})\times\tanh(\frac{L_2}{2}))$$

式(20)可轉成另一種型式

$$pS2(L_1,L_2) = 2\tanh^{-1}(\tanh(\frac{L_1}{2})\times\tanh(\frac{L_2}{2})) \quad (21)$$
$$= sign(L_1)sign(L_2)\phi(\phi(|L_1|)+\phi(|L_2|))$$

其中 $\phi(x) = -\ln\left(\tanh(\frac{x}{2})\right) = \ln\left(\frac{e^x+1}{e^x-1}\right)$ and $\phi(\phi(x))=x$ (22)

以上為 S2 檢查節點搜集其它位元節點的資訊，經由數學運算統計機率後再傳給該解碼的位元節點，然而 S1 與 S2 都有連線至 c1，因此 c1 會收到這兩個檢查節點的訊息，與(18)類似的概念得到下式

$$pc1(p_{S0},p_{S1},q_{S0},q_{S1}) = \left(\frac{p_{S0}q_{S0}}{p_{S0}q_{S0}+p_{S1}q_{S1}}, \frac{p_{S1}q_{S1}}{p_{S0}q_{S0}+p_{S1}q_{S1}}\right) \quad (23)$$

將式(19)代入

$$pc1(L_1,L_2) = \ln(\lambda_1\lambda_2) = \ln\lambda_1 + \ln\lambda_2 = L_1 + L_2 \quad (24)$$

故將兩個檢查節點的機率值相加即是綜合 S1、S2 的機率。

式(21)與式(24)均為統計運算兩個節點的機率值，將此兩個公式擴展成整合多數節點的機率值，由式(20)擴展得到

$$
\begin{aligned}
&pS(L_1 \oplus L_2 \oplus \ldots \oplus L_n) \\
&= pS(pS(\ldots pS(pS(L_1 \oplus L_2) \oplus L_3)\ldots) \oplus L_n) \\
&= sign(L_1)sign(L_2)\cdots sign(L_n)\phi(\phi(|L_1|)+\ldots+\phi(|L_n|)) \\
&= \prod_{i=1}^{n} sign(L_i) \times \phi(\phi(\sum_{i=1}^{n}|L_n|))
\end{aligned}
\tag{25}
$$

此為檢查節點從 n 個位元節點統計的機率值，接著要統計多個檢查節點至單一位元節點的機率值，由式(24)擴展得到

$$
\begin{aligned}
&pc(L_1, L_2, \ldots, L_n) = \ln(\lambda_1\lambda_2\ldots\lambda_n) \\
&= \ln\lambda_1 + \ln\lambda_2 + \ldots + \ln\lambda_2 = L_1 + L_2 + \ldots + L_n
\end{aligned}
\tag{26}
$$

此為位元節點從 n 個檢查節點統計的機率值，有了上述的概念之後，接下來介紹 Sum-Product Algorithm 的解碼流程，以編號作為演算法之順序。

(1) 初始化 (Initialization)

設

$$
L_n = \ln\frac{P(y_n \mid x_n = 0)}{P(y_n \mid x_n = 1)} = \frac{2}{\sigma^2}y_n \tag{27}
$$

其中 P(a|b)表示 b 位元從傳送端出發、在接收端收到 a 位元時為 0 與 1 的機率，σ^2 代表雜訊參數，所以接收到的 codeword 每個位元機率可表示為 $L_1 \sim L_n$。

當 H 矩陣中 $H_{m,n}=1$ 時 設 $q_{m,n} = L_n$ ，即對應矩陣中元素為 1 的機率值。

(2) 計算位元節點至檢查節點的機率資訊

每個檢查節點均會收到所連結位元節點的機率資訊，設這些機率值為 $q_{m,n}$ ，但因為統計這些機率不包括本身的機率，所以表示為

$$r_{m,n} = \underset{n'\in L(m)\backslash n}{pS}(\sum \oplus q_{m,n'})$$
$$= sign(q_{m,n}) \prod_{n'\in L(m)} sign(q_{m,n'}) \times \phi(\phi(\sum_{n'\in L(m)} |q_{m,n'}|) - \phi(|q_{m,n}|)) \quad (28)$$
$$where \ \ \phi(x) = -\ln\left(\tanh(\frac{x}{2})\right) = \ln\left(\frac{e^x+1}{e^x-1}\right) \text{ and } \phi(\phi(x))=x$$

(3) 計算檢查節點至位元節點的機率資訊

每個位元節點接收來自所連結檢查節點的機率資訊。

$$q_{m,n} = pc(\underset{m'\in M(n)\backslash m}{pc}(r_{m',n}), L_n) = L_n + \sum_{m'\in M(n)\backslash m} r_{m',n} \quad (29)$$

(4) 計算位元節點本身的機率資訊與解碼

$$q_n = pc(\underset{m\in M(n)}{pc}(r_{m,n}), L_n) = L_n + \sum_{m\in M(n)\backslash m} r_{m,n} \quad (30)$$

得到的 q_n 進行判斷：$c_n = \begin{cases} 1 & if \ \ q_n \geq 0 \\ 0 & if \ \ q_n < 0 \end{cases}$

(5) 重複遞迴運算

如果解出來的 codeword 滿足 $c \cdot H^T = 0$ 則解碼結束，否則重覆從步驟(2)開始直到解出正確的碼或達到遞迴次數的上限值。

2.3-2 Min-Sum Algorithm

由式(20)可改寫成

$$
\begin{aligned}
& pS2(L_1, L_2) = pS2(L_1 \oplus L_2) \\
& = \ln(\cosh(\frac{L_1 + L_2}{2})) - \ln(\cosh(\frac{L_1 - L_2}{2})) \\
& = \left|\frac{L_1 + L_2}{2}\right| - \left|\frac{L_1 - L_2}{2}\right| + \ln\frac{1 + e^{-|L_1 + L_2|}}{1 + e^{-|L_1 - L_2|}} \\
& = sign(L_1) \times sign(L_2) \times \min(|L_1|, |L_2|) + \ln\frac{1 + e^{-|L_1 + L_2|}}{1 + e^{-|L_1 - L_2|}} \\
& \approx sign(L_1) \times sign(L_2) \times \min(|L_1|, |L_2|)
\end{aligned}
\tag{31}
$$

此式將複雜的運算式 ϕ 簡化成最小值運算，雖然有效降低運算量但也失去部份的效能，因為 $\phi(x)$ 中 x 值越小時 ϕ 值越大，若同時有 4 個最小值大小排列為 x4>x3>x2>x1，則算式只會選擇 x1 卻失去 x2、x3、x4 帶來效能偏差，所以通常 Min-Sum Algorithm 與 Sum-Product Algorithm 在 code length 越長時效能影響越大。

2.3-3 Min-Sum Correction-Factor [20]

為了改善 Min-Sum Algorithm 的效能損失，在式(31)中省略的部分

$$
\ln\frac{1 + e^{-|L_1 + L_2|}}{1 + e^{-|L_1 - L_2|}} = \ln(1 + e^{-|L_1 + L_2|}) - \ln(1 + e^{-|L_1 - L_2|}) \tag{32}
$$

此參數稱為 Correction-Factor，可經由查表法 LUT 算出。

加入此項參數後能修正數值，使解碼效能更接近 Sum-Product Algorithm。

Min-Sum Algorithm 的解碼流程如下：

(1) 初始化 (Initialization)

設

$$L_n = \ln \frac{P(y_n \mid x_n = 0)}{P(y_n \mid x_n = 1)} = \frac{2}{\sigma^2} y_n \tag{33}$$

(2) 計算位元節點至檢查節點的機率資訊

$$\begin{aligned} r_{m,n} &= \underset{n' \in L(m) \backslash n}{pS} \left(\sum \oplus q_{m,n'} \right) \\ &= sign(q_{m,n}) \prod_{n' \in L(m)} sign(q_{m,n'}) \times \min_{n' \in L(m)} \{ |q_{m,n}| \} \end{aligned} \tag{34}$$

(3) 計算檢查節點至位元節點的機率資訊

$$q_{m,n} = pc\left(\underset{m' \in M(n) \backslash m}{pc} (r_{m',n}), L_n \right) = L_n + \sum_{m' \in M(n) \backslash m} r_{m',n} \tag{35}$$

(4) 計算位元節點本身的機率資訊與解碼

$$q_n = pc\left(\underset{m \in M(n)}{pc} (r_{m,n}), L_n \right) = L_n + \sum_{m \in M(n) \backslash m} r_{m,n} \tag{30}$$

得到的 q_n 進行判斷：$c_n = \begin{cases} 1 & if \quad q_n \geq 0 \\ 0 & if \quad q_n < 0 \end{cases}$

(5) 重複遞迴運算

如果解出來的 codeword 滿足 $c \cdot H^T = 0$ 則解碼結束，否則重覆從步驟(2)開始直到解出正確的碼或達到遞迴次數的上限值。

第三章 傳統消去迴圈演算法 (Cycle Elimination)

一般來說 SPA 演算法具有優異的解碼能力，在解碼的過程中因為訊息經由 edge 互相傳遞而造成訊息的相依關係，再加上 LDPC 解碼屬於遞迴式的演算法，希望訊息傳遞時盡可能的獨立，如果在位元檢查矩陣中含有許多短迴圈(cycle)會使得收斂速度變慢且解碼效能變低，因此設計具有長迴圈架構的位元檢查矩陣能使訊息之間相依關係減低進而獲得效能提升。

此章將介紹傳統消去迴圈演算法(Cycle Elimination) [21]，採用搜尋短迴圈並打斷迴圈的機制產生矩陣，並同時擴展每個元素形成 Block-LDPC 矩陣格式有利於 VLSI 硬體上編碼與解碼的實現[22][23]。

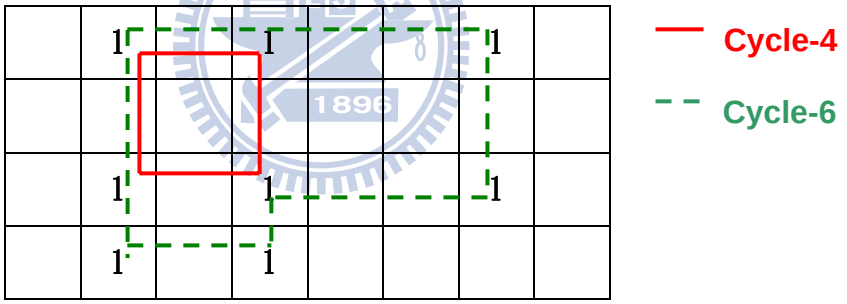


圖 3-1 Cycle 表示方法

3.1 基本矩陣與擴展

矩陣中最小的 cycle 定義為 Girth，一般而言當 Girth=8 以上有較好的解碼效能，也就是矩陣中不會形成如圖 3-1 所示 cycle-4 或 6。目前有幾種演算法能增加矩陣中 Girth 長度，如 Progressive Edge-Growth(PEG)[24]、Bit-Filling [25]等演算法，但產生的矩陣中非零的元素亂數排列，對實現硬體上較為困難複雜度也相對增加。

為了利於硬體實現並採用平行化的概念以達到高吞吐的輸出，產生位元檢查矩陣時採用 Block-LDPC[26]架構，如圖 3-2 所示先準備 Ms x Mn 大小的基本矩陣 (Base matrix)，再按照 code length 所需的大小展開成 p · Ms x p · Mn，其中每個 0 展開為 p x p 大小的零矩陣，每個 1 展開為 p x p 大小的單位矩陣。

擴展基本矩陣除了對硬體實現有好處外，同時也能對效能有所助益，因為擴展後的矩陣 Girth 一定大於或等於基本矩陣的 Girth，此特性也是支持消去迴圈演算法的主要理論，經由擴展的過程中位移單位矩陣達到打斷 Cycle 的效果。

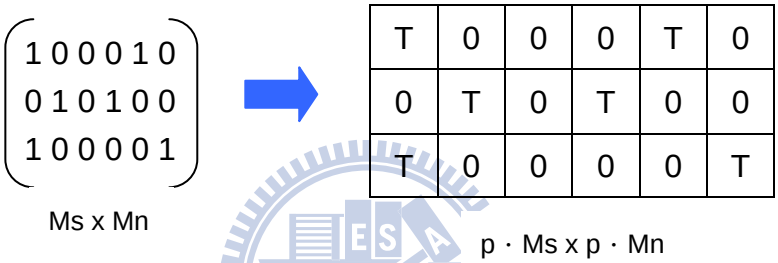


圖 3-2 擴展矩陣示意圖

3.2 形成迴圈之條件與限制

由上一節得知用位移單位矩陣打斷短迴圈，而形成迴圈中每一個元素位移情況以圖 3-3 舉例說明。

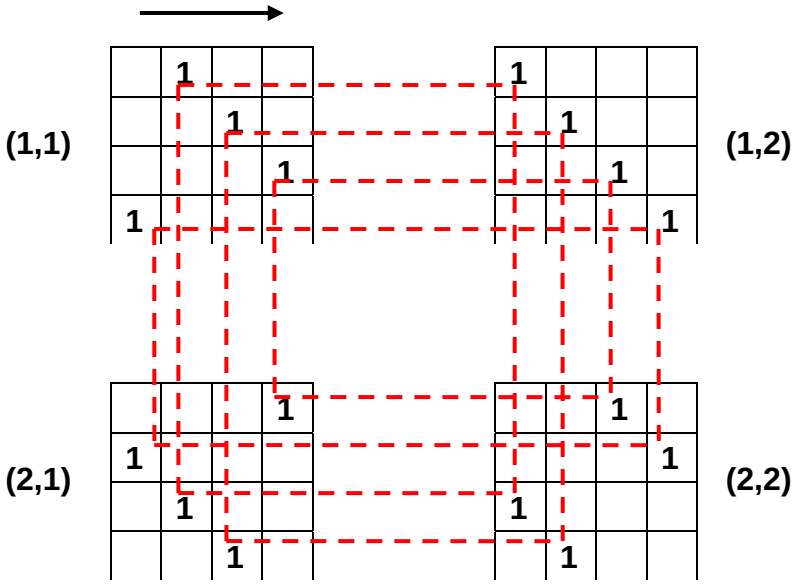


圖 3-3 擴展矩陣之後迴圈情況

以圖 3-3 為例，將基本矩陣中每個元素擴展成 4 x 4 的小區塊矩陣，每個區塊都是單位矩陣位移之後形成的，P 為向右位移量，故 P(1, 1)=1、P(1, 2)=0、P(2, 1)=3、 P(2, 2)=2，得到上圖形成 4 條長度為 4 的迴圈，此種迴圈通稱為 cycle-4，同理長度為六的迴圈通稱為 cycle-6 以此類推，而任何矩陣中最小的迴圈為 cycle-4，為了探討各元素位移量與形成迴圈之間的關係，用數學公式來推導證明如下。

將同一行中兩個元素歸類一組，如 P(1, 1)和 P(2, 1)為一組、P(1, 2)和 P(2, 2)為一組，在[27]中定義：

$$\Delta P_{i,j->i,m} = P_{i,j} - P_{i,m} \tag{31}$$

，則

$$\Delta P_{1,1->2,1} = P_{1,1} - P_{2,1} = 3-1 = 2 \tag{32}$$

$$\Delta P_{2,2->1,2} = P_{2,2} - P_{1,2} = 0-2 = -2 \tag{33}$$

，最後把式(32)和式(33)相加得到

$$\sum_V(\Delta P)=P_{1,2}-P_{2,2}=(2)+(-2)=0 \qquad (34)$$

，式 3-5 中的 V 代表 vertical shift-value drops。

上述是以 cycle length=4 為例，ΔP 相減的順序如圖 3-4 所示，從左上方的元素點出發順時鐘或逆時鐘繞回原點，保持方向一致性，不管是 Cycle 4、6、8 都相同。

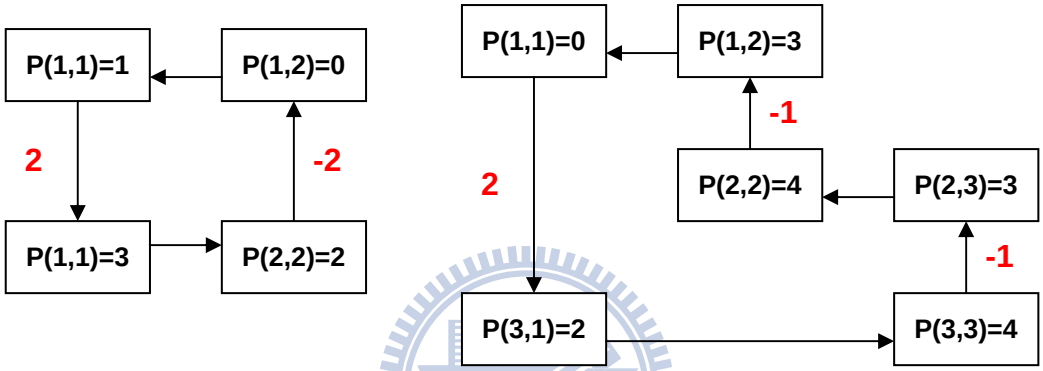


圖 3-4 計算位移量的方向示意圖

因此我們可以直接用數學計算位移量和形成迴圈之間的關係。首先如圖 3-5 所示把展開之後的方塊矩陣切成兩部份，每一個 1 的坐標都可表示為下式：

$$\begin{cases}(i,i+P-0//L), & \text{當 } 0\le i<L-P \text{ (左上方)}\\ (i,i+P-L), & \text{當 } L-P\le i\le L-1 \text{ (右下方)}\end{cases} \qquad (35)$$

，將式(35)合併成

$$(i,i+P-0//L) \qquad (36)$$

，對應圖 3-6 中表示為

$$(s_i, s_i + P_{i,1} - 0 // L) \tag{37}$$

，又得知 H(1,2)與 H(2,1)有相同的行，H(1,1)與 H(g,2)有相同的行，以此類推得到下列的式子：

$$\begin{aligned} s_1 + P_{1,2} - (0 // L) &= s_2 + P_{2,1} - (0 // L) \\ s_2 + P_{2,2} - (0 // L) &= s_3 + P_{3,1} - (0 // L) \\ &\vdots \\ s_g + P_{g,2} - (0 // L) &= s_1 + P_{1,1} - (0 // L) \end{aligned} \tag{38}$$

，將上式加總得到

$$(P_{1,2} - P_{2,1}) + (P_{2,2} - P_{3,1}) + \ldots + (P_{g,2} - P_{1,1}) = (y - x)(-L) \tag{39}$$

，整理得

$$\sum\nolimits_V (\Delta P) = k \cdot L \ (k = 0, \pm 1, \pm 2, \ldots, \pm g) \tag{40}$$

上式為判斷位移量與迴圈的準則稱做 Cycle constraint，其中 g 值為 cycle 長度的一半，例如 cycle length=6 則 g=3。

等式成立的話則會形成迴圈，因此以下的內容若稱滿足 constraint 則表示等式不成立，不滿足 constraint 則表示等式成立。

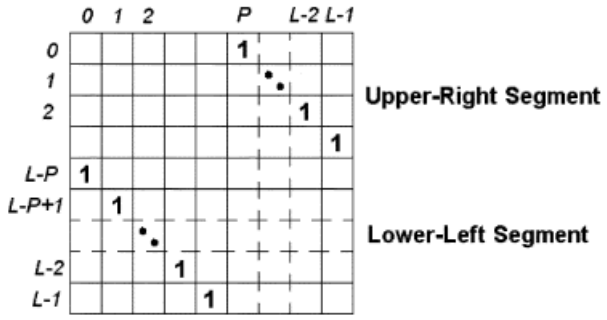


圖 3-5 方塊矩陣之元素坐標（節錄自[21] Fig.5）

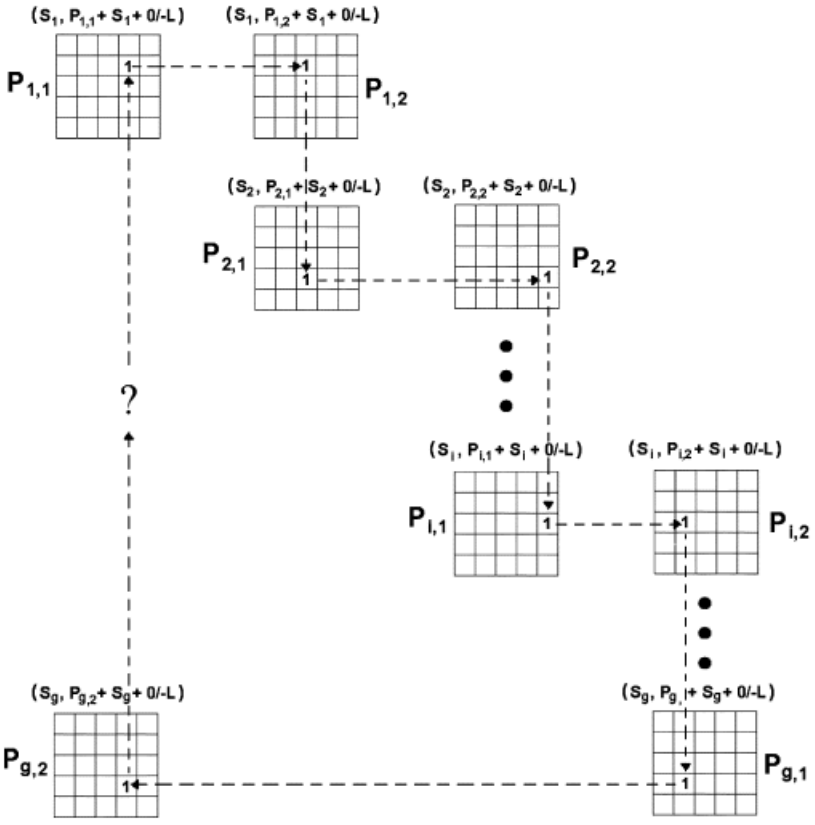


圖 3-6 各方塊矩陣迴圈連結示意圖(節錄自[21] Fig.6)

3.3 傳統消去迴圈演算法

這裡以條列式介紹傳統的消去迴圈演算法，並分析該演算法之優缺點及改進的目標，並在下一章描述改良的演算法。

3.3-1 演算法流程

首先準備好基本矩陣，矩陣中每個 1 含有一個清單，清單是用來儲存式(40)的 constraint，用來判斷是否有打斷或重新回復 Cycle。

接著開始搜尋 Cycle length 為 4、6、8 之迴圈，從最左邊的行(column)開始往右搜尋，並將每一個找到的迴圈記錄在清單中以便之後拿來判斷使用，以程式模擬結果得知 Cycle length 越大其個數增加的越快。

當找出任何迴圈時，開始對此迴圈中每個 1 做 constraint 的判斷以打斷迴圈，因為擴展成 L X L 的區塊矩陣中位移量限制為[0 , L-1]，為了使位移增加的速度減緩，先將每個 1 的初始位移值儲存，每個 1 檢查自己的 constraint 是否符合，若不符合則位移值加 1（$P_{i,j}=P_{i,j}+1$）直到滿足所有 constraint，找出此迴圈中位移總合最小的 1 保留此值，其它的 1 回存原值。以圖 3-7 為例，P 代表總位移值、括弧內代表此迴圈檢查 constraint 所增加的值，選擇最小的總位移量(36)並回存其它值。

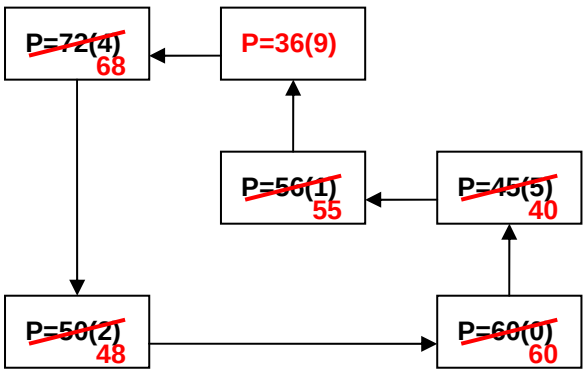


圖 3-7 傳統消去迴圈演算法位移動作之圖例

3.3-2 演算法分析與比較

傳統消去迴圈演算法是找到一個迴圈就會進行位移的動作，以消去目前找到的迴圈再進行下一個迴圈消除，此種作法類似暴力法，將所有的迴圈跑過一遍並檢查迴圈中每個元素的 constraint，再做比較位移最小值的工作，因此運算量非常可觀，下表為模擬圖 4-8 的矩陣的結果。

	Cycle-4	Cycle-6	Cycle-8
執行迴圈總量	81	924	17811
平均單一迴圈檢查 constraint 數量	5.54	142.14	4262.44
總合	448.74	131337.36	75918319

表 3-1 估計運算的 constraint 總量

表 3-1 為 code rate=1/2、18 x 36 非正規基本矩陣，其中包含 117 個 1。該演算法從最左邊的行(column)開始往右動作，可避免重覆的迴圈運算，由上表觀察出 cycle 8 數量比 cycle 4、6 高出許多，且因為每執行一次消去迴圈位移的動作，就必須把迴圈中每個元素 1 的 constraint 都計算過一遍，所以增加不少運算量。例如 cycle-6 就要檢查 6 個點，相對增加 6 倍 constraint 的運算，故在 cycle-8 中平均每 cycle 需檢查 3253 個 constraint。

表 3-2 分別列出在 cycle 4、6、8 中每個 constraint 乘法與加法的總數，若分別乘上表 3-1 constraint 的數量即為整體演算法的運算量，因此如果把 constraint 定在 cycle-8 將會耗費相當多的運算。

雖然傳統消去迴圈演算法的計算方式需要大量運算量及時間，但是位移打斷迴圈的效果全面且仔細。

	Cycle-4	Cycle-6	Cycle-8
加法數	3	5	7
乘法數	5	7	9

表 3-2 每條 constraint 的加、減法總量

第四章 改良式消去迴圈演算法

本章節將介紹改良式演算法，根據元素包含的迴圈數高低取得優先次序，執行位移打斷迴圈的動作，並充分利用每次能打斷 Cycle 的機會。我將以不同的基本矩陣做說明，除了大幅改善位移消去之運算量及時間，在低擴展倍數時也能有效的將迴圈數量減到最低。

4.1 搜尋迴圈與演算法流程

此節介紹搜尋迴圈的方式，詳細說明改良式消去迴圈演算法的流程與架構，並在下一節分析演算法的模擬結果。

4.1-1 搜尋迴圈方法

首先準備基本矩陣，先找出所有元素 1 儲存各種資訊於陣列中，如表 4-1 這些包含上、下、左、右 1 的個數、坐標及位移資訊，而演算法中不管是搜尋迴圈或是權重機制都是從左上方的點開始往右搜尋到底後換列，如圖 4-1 所示，直到所有的點都被搜尋過，如此一來保持演算法執行的方向統一避免重覆，以下用條列式的方式介紹搜尋方法與順序，並以圖 4-2 輔助說明。

(x, y)坐標
上端個數. 坐標
下端個數. 坐標
左端個數. 坐標
右端個數. 坐標
Cycle 個數. 坐標
位移量
位移旗標

表 4-1 每個元素為 1 之陣列

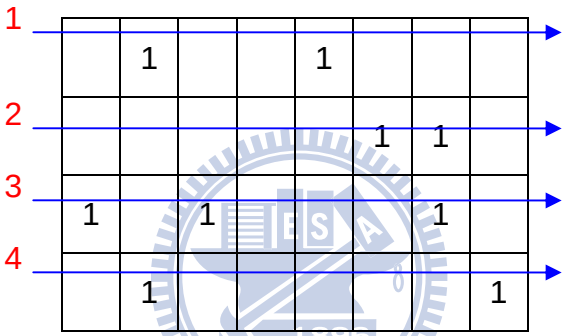


圖 4-1 搜尋時之方向示意圖

1. 從左上方的原點 P(1,1)出發，往右及往下搜尋找到 P(1,2)及 P(4,1)並標記為第一層，其中 P(1,2)與原點有相同的列數、P(4,1)與原點有相同的行數。
2. 從 P(1,2)往上或下搜尋得到 P(2,2)，而 P(4,1)往右搜尋得到 P(4,4)，將 P(2,2)及 P(4,4)標記為第二層，其中 P(1,2)與 P(2,2)有相同的行數、P(4,1)與 P(4,4)有相同的列數。
3. 從 P(2,2)往右或左搜尋得到 P(2,3)，而 P(4,4)往上或往下搜尋得到 P(3,4)，將 P(2,3)及 P(3,4)標記為第三層，其中 P(2,2)與 P(2,3)有相同的列數、P(3,4)與 P(4,4)有相同的行數。
4. 最後搜尋全部的點，若有一點 P(X,Y)與 P(1,2)有相同的行數並且與 P(4,1)有相同的列數則形成 Cycle-4；若有一點 P(X,Y)與 P(2,2)有相同的列數並

且與 P(4,4)有相同的行數則形成 Cycle-6；若有一點 P(X,Y)與 P(2,3)有相同的行數並且與 P(3,4)有相同的列數則形成 Cycle-8。

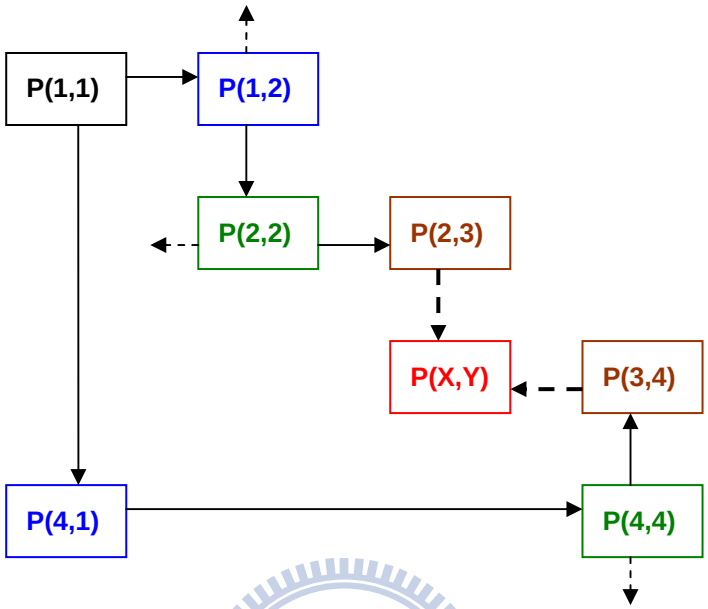


圖 4-2 搜尋迴圈示意圖

搜尋迴圈時有兩點需要特別注意:

(1) 在階層搜尋時，必須避免不同階層搜尋到相同的點，這種現在尤其在 cycle-8 時更容易發生。如圖 4-3 所示，空心的點為原點，在往右搜尋時會先找到 A 點，而同樣地從原點依序往下、往右及往上也會找到 A 點，造成同一點搜尋兩次的情況，因此在搜尋過程中設定所有點的坐標均不相等即可。

(2) 搜尋的方向不能超過起始點左側的行數，如圖 4-4 以空心點當原點時，搜尋到第三層也就是虛線框起來的部分，這兩點行數已小於原點；而以 B 點當原點搜尋時也會搜尋到一樣的迴圈，造成同一個迴圈被搜尋至兩次，所以要以原點的行數做標準，任何搜尋的路徑只能在原點的右半側。

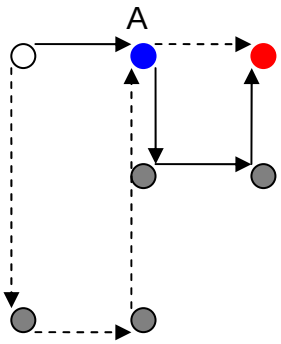


圖 4-3 造成同一點搜尋兩次之範例

此行數為基準時已搜尋過

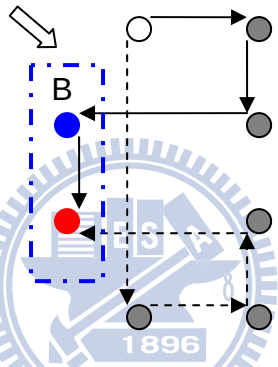


圖 4-4 搜尋時路徑超過原點左半邊之範例

4.1-2 演算法流程與架構

前一章介紹了傳統式迴圈演算法，雖然過程是循序式的將迴圈打斷，但運算量及複雜度很高，所以我們提出改良式消去迴圈演算法，除了有效打斷消除迴圈之外，依照每一點包含迴圈的數量訂為權重值，按權重高低消去迴圈以簡化運算量。

優先權消去迴圈演算法流程：

(1) 找出基本矩陣中為 1 的元素，並創造一個空陣列儲存各種資訊。

(2) 清空元素 1 中全部參數的初始值。

(3) 從最左上角的 1 開始，找出所有 1 的（x,y）坐標及上、下、左、右四個方向為 1 的個數及坐標，並儲存在陣列中供搜尋迴圈使用。

(4) //搜尋 Cycle-4、6、8，同時將每條迴圈的坐標及資訊存入陣列。

(5) 從原點 H(i,j)往右、往下找到兩點 H(i,k)、H(m,j)，若有一點坐標為 H(m,k) 則此 4 點形成 cycle-4。

(6) for H(i,k)上方及下方的 1 為 H(p,k)、H(m,j)右方的 1 為 H(m,n)，若有一點坐標為 H(p,n) 則此 6 點形成 cycle-6。

(7) for H(p,k)右方及左方的 1 為 H(p,r)、H(m,n)上方及下方的 1 為 H(s,n)，若有一點坐標為 H(s,r) 則此 8 點形成 cycle-8。

(8) end

(9) end

(10) //優先權一次式消去迴圈法

(11) while（元素 1 最大迴圈數>0）

(12) if ((某一點元素迴圈數==最大迴圈數)&&(位移旗標==0))

(13) for（該元素所有可位移的值 P =（1~L-1））

(14) 檢查該元素包含的 constraint，分別記錄滿足 constraint 及不滿足的數量(這邊定義參數為 rig 與 wro)。

(15) if（rig > wro）

(16) 比較該元素所有的位移值並取 P 使(rig-wro)有最大值。

(17) else

(18) 該元素位移值 P=0 ；

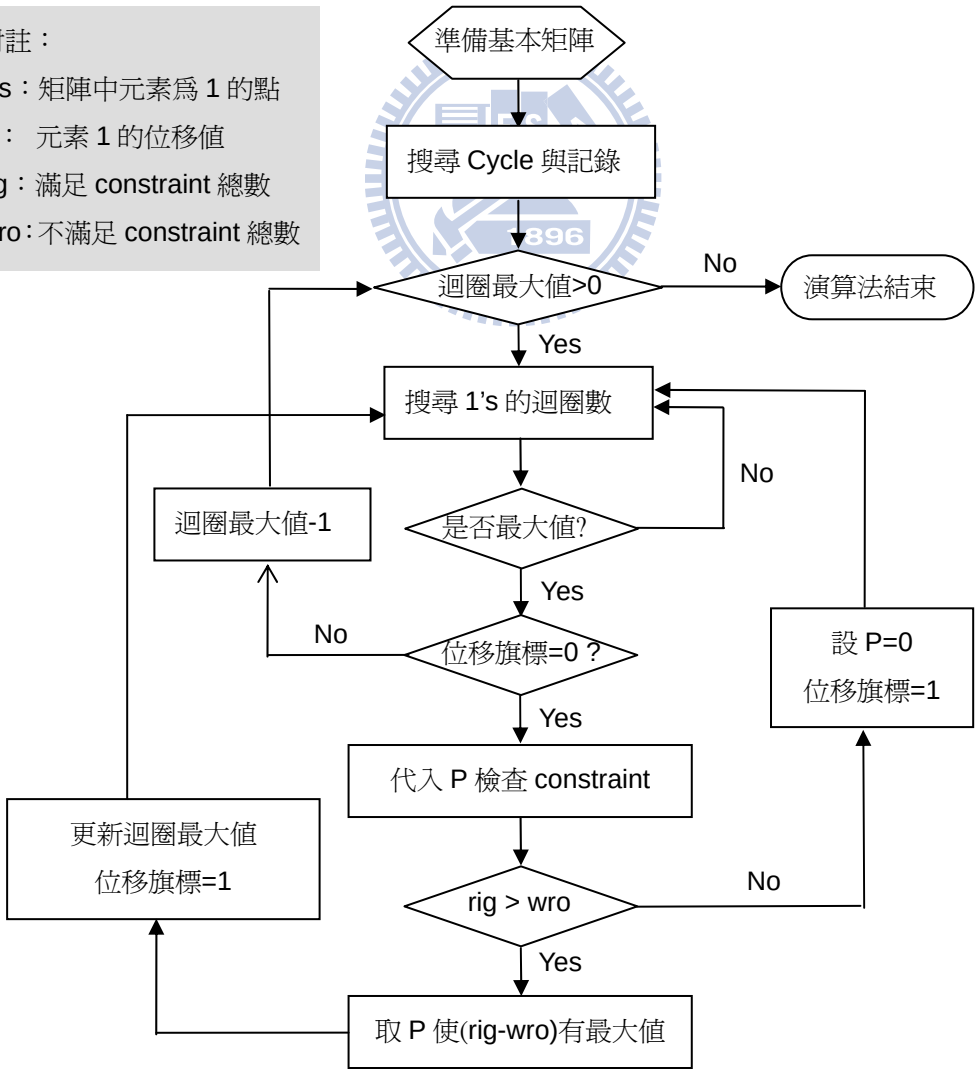
(19) end

(20) 該元素的位移旗標設為 1；

```
(21)        更新基本矩陣的最大迴圈數；
(22)      elseif ((某一點元素迴圈數==最大迴圈數)&&(位移旗標==1))
(23)        跳過並尋找下一個含有相同最大迴圈數之元素；
(24)      else
(25)          最大迴圈數=最大迴圈數-1；
(26)      end
(27)    end
(28) end
```

演算法流程圖：

附註：
1's：矩陣中元素爲 1 的點
P： 元素 1 的位移值
rig：滿足 constraint 總數
wro：不滿足 constraint 總數



演算法分段以行數說明與分析如下：

(1)~(3)行：首先準備一個基本矩陣，所有在矩陣中的 1 均產生如表 4-1 的清單並將初始值歸零。如圖 4-1 從矩陣的左上角開始搜尋 1 的位置記錄坐標，並同時找出上下左右四個方向的 1，將其坐標與數量儲存。因為每個元素包含許多 cycle，每條 cycle 都會轉成 constraint 存於清單。

(4)~(9)行：每一點包含 4 個方向的資訊，我們利用此資訊尋找迴圈。在 4.1-1 節說明了搜尋方法，但此方法只適用於專門搜尋 Cycle-4、6、8 的情況，cycle 的基本定義為從原點出發繞過一條封閉的路徑回到原點，且每一個節點不能通過第 2 次，所以提出兩點方法避免找到錯誤或重覆的 cycle。然而這邊所搜尋是未擴展之前的基本矩陣，經由模擬結果證明圖 4-3 的情形在擴展矩陣之後有可能造成新的 cycle-8，如果我未考慮及搜尋到此種迴圈，即便程式結果指出已打斷所有迴圈，新的擴展矩陣還會有未打斷的 cycle 產生。為了使程式能完全打斷所有的迴圈，提出兩條輔助方法：

(1) 以圖 4-5 當作範例，若藍色 AB 共點則勢必造成紅色 CD 共點，反之亦然；這並不符合構成迴圈的形式，因為當搜尋至 A、B 點時下一步是往水平方向搜尋而找到 C、D 點，找到 C、D 點後下一步往垂直方向搜尋，若 C、D 共點的話則無法找到搜尋的路徑，所以除了讓 A、B 和 C、D 不共點之外取消其它避免共點的限制。

(2) 在[13]文獻中提到所有搜尋的點的行數必須大於原點的行數(d > j)， 但這種說法並不周延，我的模擬實驗出現了一種情況，以圖 4-6 所示藍點為原點，各點的數字指移位量，在此種情形下原點被搜尋到兩次，利用式 3-11 得到 $\sum \Delta P = 0 + (14) + (14) + (-28) = 0$ ，代表在擴展矩陣後會形成新的 cycle-8，並不符合作者演算法中的描述，因此我修改為所有點的行數必須大於或等於原點的行數。

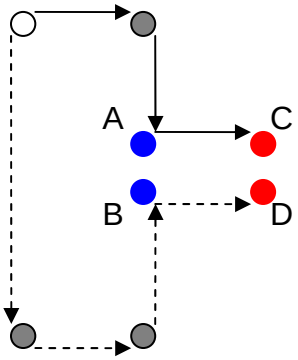


圖 4-5 若 A、B 共點則 C、D 共點將無法找到搜尋路徑之圖例

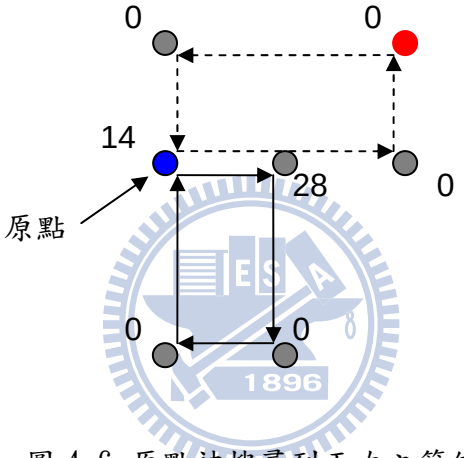


圖 4-6 原點被搜尋到兩次之範例

(10)~(28)行：權重指元素 1 所包含迴圈總數，這也是本演算法主要的精神，依照每個 1 不同權重來執行迴圈消去的優先次序。當初的構想是當該元素擁有最高權重時，能一次打斷迴圈的數量必會高於其它元素。首先找出具有最高權重的元素，接著把所有的位移值（1~L-1）代入 constraint 表，統計各別滿足及不滿足 constraint 的總數，首要條件為滿足的總量必須大於不滿足的總量，再來選擇讓兩者相差最大的位移值，也就是該位移值能打斷更多的迴圈、更少重新連結的迴圈，充份利用每一次打斷的機會。另外再準備 1 張表單，把滿足的 constraint 搬到此表單、不滿足的維持在原表單，如圖 4-7 所示方便處理迴圈資訊與數量。若滿足的總量小於不滿足的總量，則該元素位移值設為零，繼

續往下一個擁有高優先權的元素動作，最後只要有判斷執行該元素時均拉起位移旗標表示已處理過。每執行完一次位移動作時，程式將重新整理取得最新的優先權順序，再執行位移動作盡可能打斷所有的迴圈。因為整體演算法執行到最後才會將基本矩陣擴展，不可能每次統計最新優先權時執行搜尋迴圈的動作，於是在這邊使用一個技巧，直接讀取圖 4-7 中未被打斷迴圈的數量，就可快速且精準地得到各元素所剩餘的迴圈數。

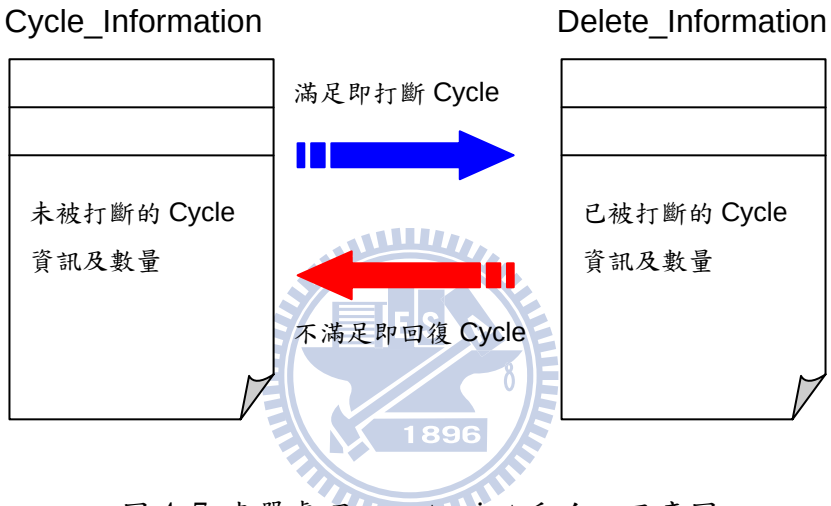


圖 4-7 表單處理 constraint 動向之示意圖

4.2 模擬結果與分析

以下將以非正規矩陣(irregular matrix)做模擬與分析，在[29]、[30]提出設計良好的非規則矩陣解碼效能比規則矩陣好，而規則矩陣的好處為簡化設計硬體的複雜度，不過隨著超大型積體電路的進步，現在的應用大多採用非規則矩陣以達到更好的效能。

模擬矩陣的參數性質列於表 4-2，採用[24]文獻中的方法建構而成的基本矩陣，此矩陣與表 3-1 所做分析模擬之矩陣相同，因此稍後有兩種演算法模擬的數據比較。

Code Rate	1/2
Variable node degree	2, 3, 7
Check node degree	5, 6
Base matrix size	18x36

表 4-2 非正規基本矩陣各種參數

圖 4-8 為 code rate=1/2 之非規則矩陣，大小為 18x36，位元節點的權重分別為{ 2, 2, …, 2, 3, …, 3, 7, …, 7 }，矩陣中共有 117 個 1，可觀察出矩陣的右半段 1 的分佈較為密集，表示這些點含有大量的迴圈，由程式分析顯示每一個元素 1 含有 cycle-4、6、8 平均值達到 1268。當基本矩陣能擴展的倍數越大，即代表有更多的位移值選擇而打斷更多的迴圈，模擬顯示擴展倍數 L 需要 235 方能打斷所有迴圈。

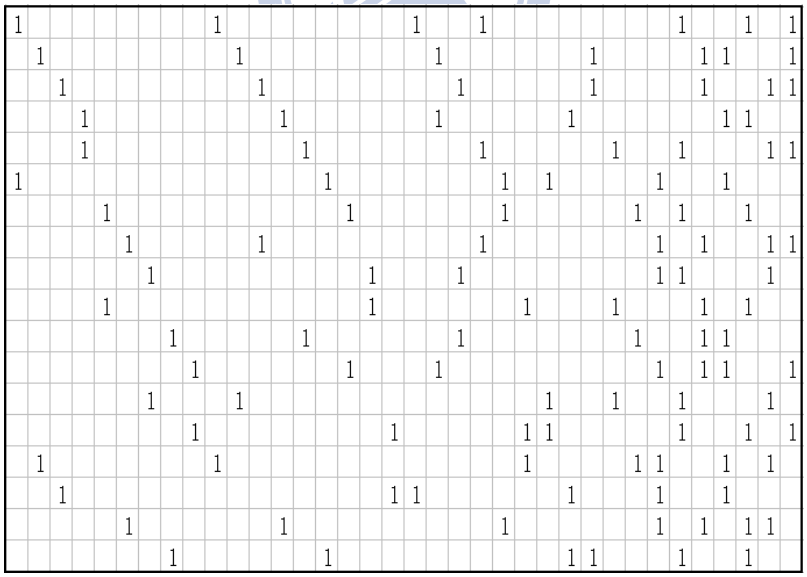


圖 4-8 Code Rate=1/2 Irregular Matrix

迴圈類型	Cycle-4	Cycle-6	Cycle-8
總數	324	5,544	142,488
每元素平均值	2.77	47.38	1217.84

表 4-3 Code Rate=1/2 矩陣中各種迴圈總數與平均

為什麼優先權的機制能更有效率打斷迴圈？因為當一個元素為 1 的點具有最高迴圈數量時，代表能一次打斷迴圈的數量越多，迴圈的總數是固定的，如果能在每次位移動作時能打斷最多的迴圈，不僅加快迴圈消去的速度，也能減少執行元素位移動作的次數。圖 4-9 為每次執行完該元素位移動作後，所有元素之最大迴圈值的數據，開始執行 10 次之內的最大迴圈值有明顯減少，故每次動作都能打斷最多的迴圈數量，即可看出採用權重優先打斷迴圈的方法收到良好的成效，而超過 20 次後因為最大迴圈值變低而打斷的迴圈數量有限，因此曲線較為平緩，總共花費 53 次執行遞迴次數將所有 Cycle-4、6、8 消除。

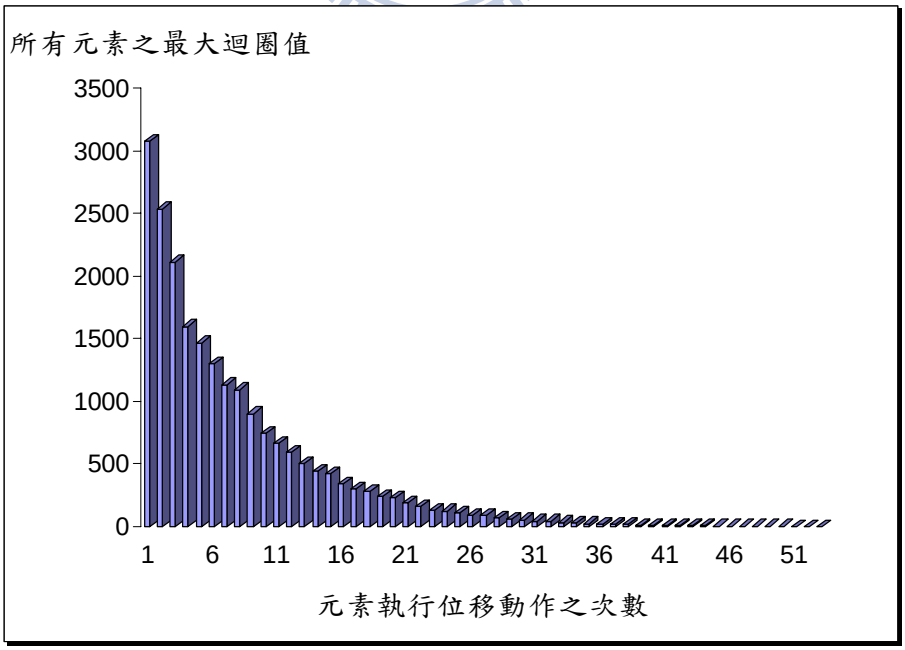


圖 4-9 每次執行位移動作後所有元素最大迴圈值之長條圖

圖 4-10 為每次執行位移動作後每個元素的迴圈平均值，在前十次元素位移消去的動作時，所有剩下迴圈的平均值已從 1268 降至 361，下降的幅度達 71.5%，再次證明以優先權打斷的機制獲得了成效，除此之外執行元素位移的次數減少後，運算量相對減低許多，在表 3-1 已討論過相同矩陣利用傳統迴圈消去演算法所花費的運算量，而這邊也將確切的運算量列於表 4-4，由於表 3-1 是以 constraint 的數量做為基準，而搜尋迴圈的方法差不多，故這邊也是以 constraint 的數量當作比較的單位。

原作者的演算法每遇到迴圈就搜尋此迴圈中所有的 1 並檢查 constraint 以減低快速增加的位移量，但帶來了大量的運算量，而以權重優先的方式先找出最大迴圈值的元素動作，僅動用了 53 個元素還不到總個數的一半就可打斷所有的迴圈，相對減低檢查 constraint 的數量達到 72%，若再乘上每個 constraint 的加法數及乘法數將會更可觀，結果列於表 4-5。

另外也將電腦模擬時間列於表 4-4，程式由 Matlab 撰寫實現，電腦主機為 Intel Core2-Quad Q6600，擴展的倍數為 235 倍。

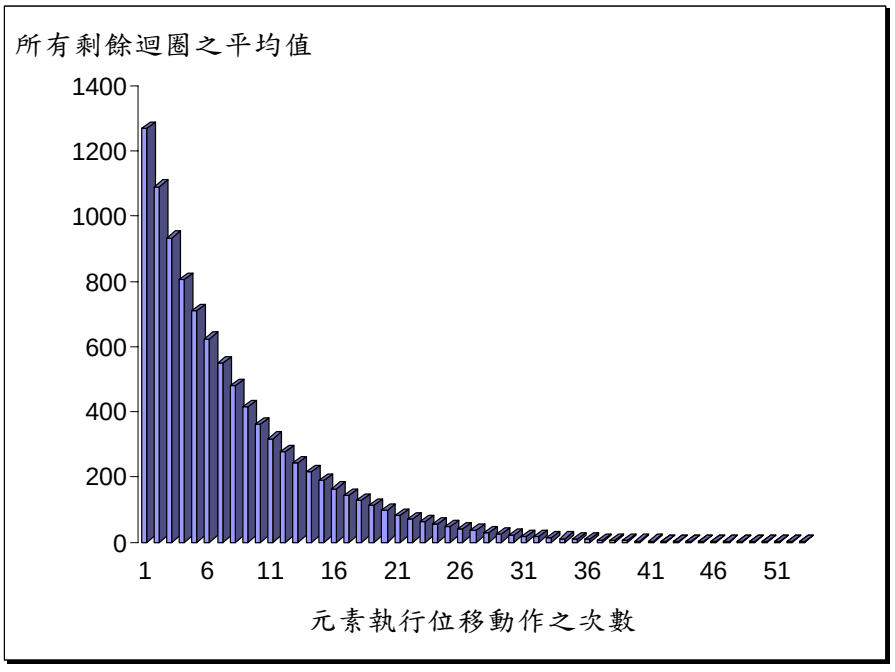


圖 4-10 每次執行位移動作後所有剩餘迴圈平均值之長條圖

	總 constraint 運算量	實際運算時間
傳統式	76,050,105	5575 秒
改良式	21,328,164	404 秒
改進比例(%)	72%	92%

表 4-4 兩種演算法之比較列表

每條 constraint	傳統式	改良式
加法數	532,068,806	149,297,148
乘法數	684,164,027	191,953,476

表 4-5 兩種演算法之乘法數與加法數

前文提到本矩陣靠近右側具有較密集的 1，因此右側將會產生大量的迴圈，如果當基本矩陣的擴展值有一定的限制時，以原作者的演算法一定從最左側的行開始消去，消去到一半時可能將所有擴展值的選擇用完，反而最需要打斷的右側卻無法進行打斷迴圈的動作，所以原作者的演算法遇到右側有密集的 1 且擴展範圍有限時會遇到這方面的問題。反觀以迴圈的總數排定優先次序，對於越密集的 1 所形成的迴圈優先打斷，整體效果會更好。表 4-6 為隨機取右側 9 個元素在不同擴展倍數剩餘的迴圈數與原始未被打斷的數據比較，證明能有效打斷這些迴圈，因此本演算法適用於低擴展倍數，也就是編碼長度短的應用，將在下一節說明與分析。

坐標/擴展倍數	未擴展	50 倍	100 倍	150 倍
(2,32)	2,482	18	4	1
(2,33)	2,114	18	4	2
(4,33)	1,564	14	3	0

(6,33)	1,627	15	1	0
(10,32)	1,840	14	0	0
(11,32)	1,783	15	0	0
(11,33)	1,587	10	1	0
(5,12)	2,817	24	6	0
(12,33)	2,450	23	3	1

表 4-6 本演算法在 1 分佈密集區域執行後剩餘的迴圈數量

4.3 適用於低編碼長度之說明與效能模擬

4.3-1 解碼長度與遞迴次數之間的影響

LDPC 為遞迴式(Iterative)的解碼機制，經由反覆傳送機率資訊將收到的碼更逼近於正確的碼，然而遞迴的次數與解碼長度有很大的關係[30]，如圖 4-12 所示，模擬通道為 AWGN，使用的解碼演算法是 SPA，H 矩陣為碼率 1/2 的非正規矩陣，紅色實線與藍色虛線分別代表 Code length 為 7200bits 與 3600bits。

在長度 7200、三種遞迴次數均造成明顯的效能差異；而在長度 3600、遞迴次數為 30 與 50 的效能差異減低，但在遞迴次數為 10 時解碼效能很差，因此在越長的解碼長度時除了帶來大量的運算，也需要更多的遞迴次數達到解碼收斂的效果，其中遞迴次數與耗電量成正比，如遞迴次數 50 次比 30 次多出 1.6 倍的耗電量，因此改良式消去迴圈演算法應用於低編碼長度實現模擬與分析為主要目標。

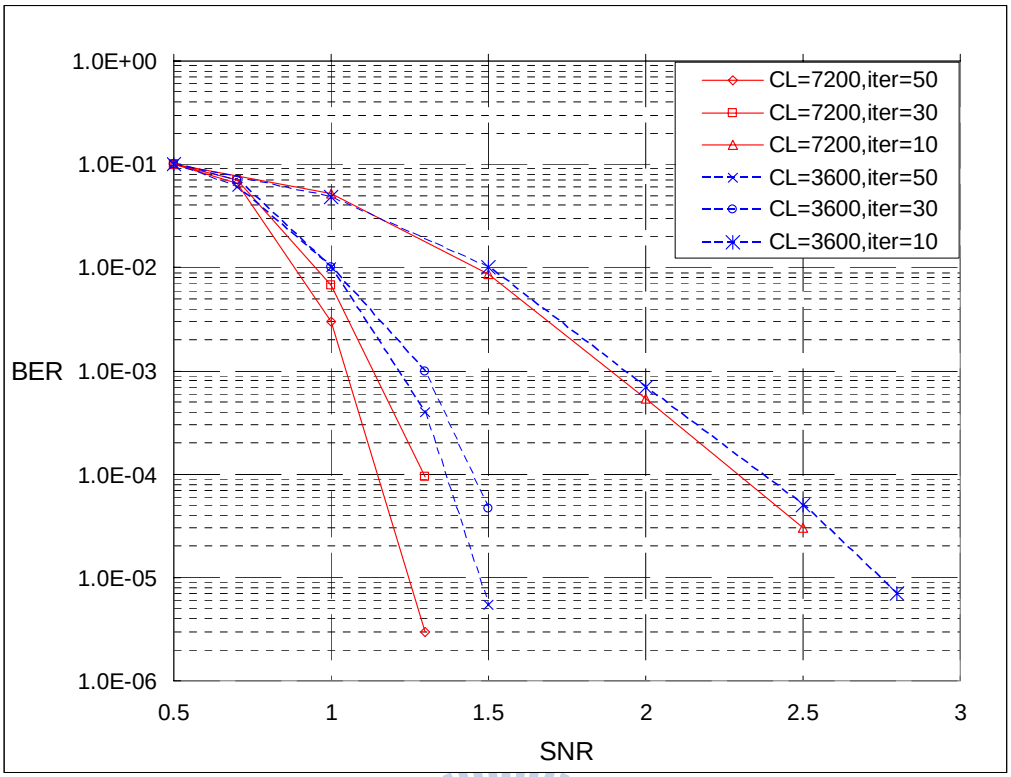


圖 4-11 不同編碼長度下遞迴次數對解碼效能的影響

4.3-2 周長與短迴圈對解碼效能之模擬

在 PEG 演算法[24]文獻中提出如果位元檢查矩陣具有較多的長迴圈，則在解碼時能使每次遞迴動作之間更為獨立以減低錯誤率；另外在[25]文獻中提出短迴圈數量越少則有較好的錯誤更正能力；還有在[31]文獻中指出如果在 Tanner Graph 出現太多短迴圈則使 MPA 解碼的效能變差。因此這邊利用改良式消去迴圈演算法產生三種不同周長(Girth)矩陣，模擬結果比較於下圖。

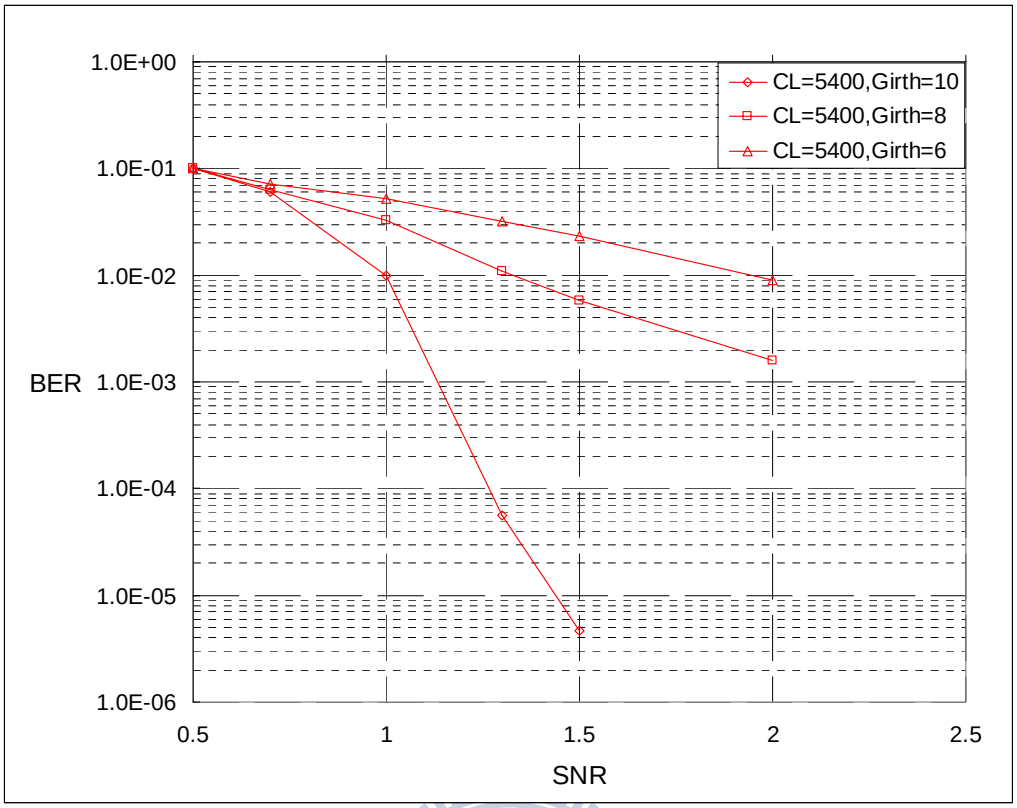


圖 4-12 不同周長的位元檢查矩陣對解碼效能的影響

模擬採用的基本矩陣與圖 4-8 相同，矩陣擴展的倍數為 150 倍，遞迴次數為 50 次，分別產生消除 cycle-4、6、8 的矩陣，另外註明因為全部消除 cycle-8 迴圈需要擴展 230 倍使編碼長度達 8280，為了減低運算量及對照低編碼長度的情況，故此處採用展開 150 倍的矩陣且 cycle-8 幾乎全部消除。由圖 4-13 得知消除 cycle-8 的矩陣效能比其它兩者優異，因為 cycle-4 與 6 保留太多 cycle-8 的迴圈，經擴展矩陣之後產生更多迴圈而使短迴圈效果影響更明顯。另外參考[32]文獻中模擬非正規矩陣的效能圖，得知 Girth-6、8 的效能接近，而 Girth-10 的效能與其它兩者差異大。

4.3-3 傳統與改良式演算法在低編碼長度之模擬比較

綜合前兩小節的論述，產生一個低編碼長度的同位檢查矩陣以減少需要的遞迴次數及運算量，達到低功耗的目的。此外因為改良式演算法擁有優先權的特

性，在低擴展倍數仍然能有效打斷短迴圈進而增進解碼效能，所以這邊將與傳統式演算法進行分析與模擬。

模擬使用的演算法除了傳統與改良式消去迴圈演算法外，加入 ZP 演算法[33]做為參考，ZP 演算法同為擴展基本矩陣的方式建構位元檢查矩陣，而每個元素 1 的位移量是由位移的公式算出:

$$P_{x,y} = T^{u}(I) \text{ , where } u = ((x-1)\cdot y) \bmod L \qquad \text{（式 4-1）}$$

模擬的方式分成兩部分，因為演算法在擴展矩陣 30 倍時能將所有 cycle-4、6 消除，所以一類擴展倍數低於 30，以不同的倍數統計剩下 cycle-4、6 的總數，再做效能比較分析；另一類擴展倍數介於 30 與 200，以不同的倍數統計剩下 cycle-4、6、8 的總數，最後做效能分析。所有的模擬比較都是建立在相同編碼長度的基礎上，而遞迴次數隨著編碼長度調整。

(1) 編碼長度小於 1000 之模擬與分析：

此擴展倍數均小於 30，主要分析兩者演算法在編碼長度小於 1000，針對 cycle-4、6 消除迴圈的情形以及效能結果。

圖 4-14 為不同倍數下剩餘短迴圈的數量，得知在擴展倍數小的時候，傳統演算法尚未執行至右半邊矩陣的元素就已經達到位移上限值，因此未打斷的短迴圈數量很多，而這些短迴圈同時對效能造成影響。

模擬基本矩陣為 18x36 大小的非正規矩陣，擴展倍數為 10 與 20 次、遞迴次數設為 10 與 20 次。圖 4-15 中兩者效能差異並不大，因為打斷迴圈的效果有限，加上 Girth-6 與 Girth-8 在非正規矩陣中效能差異不明顯，只有在 SNR=3 也就是低錯誤率時效能差距較大。

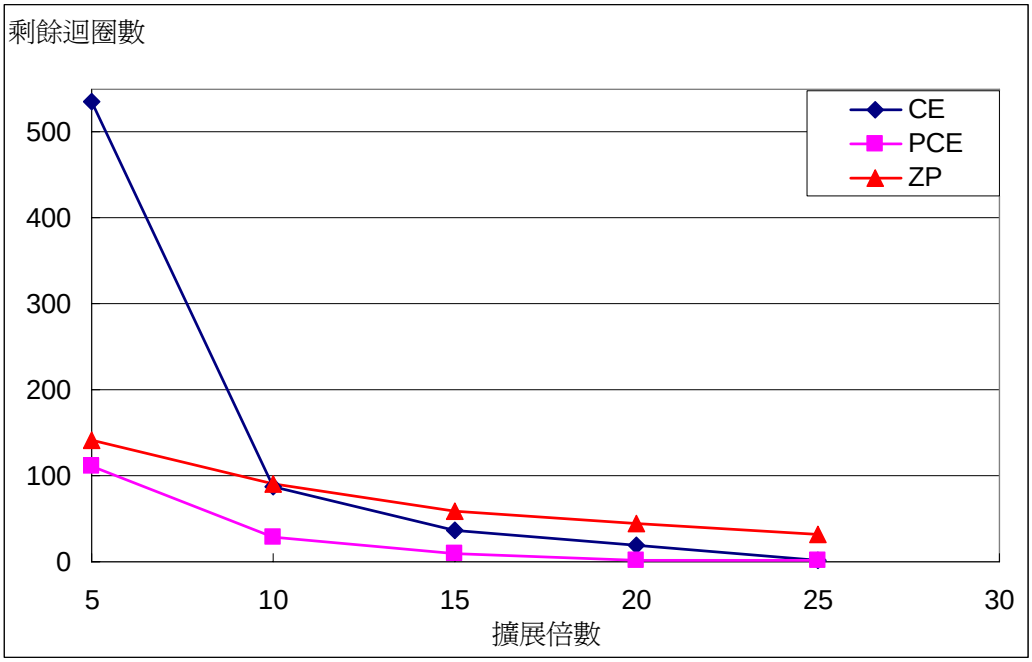


圖 4-13 不同擴展倍數下所剩餘迴圈總量

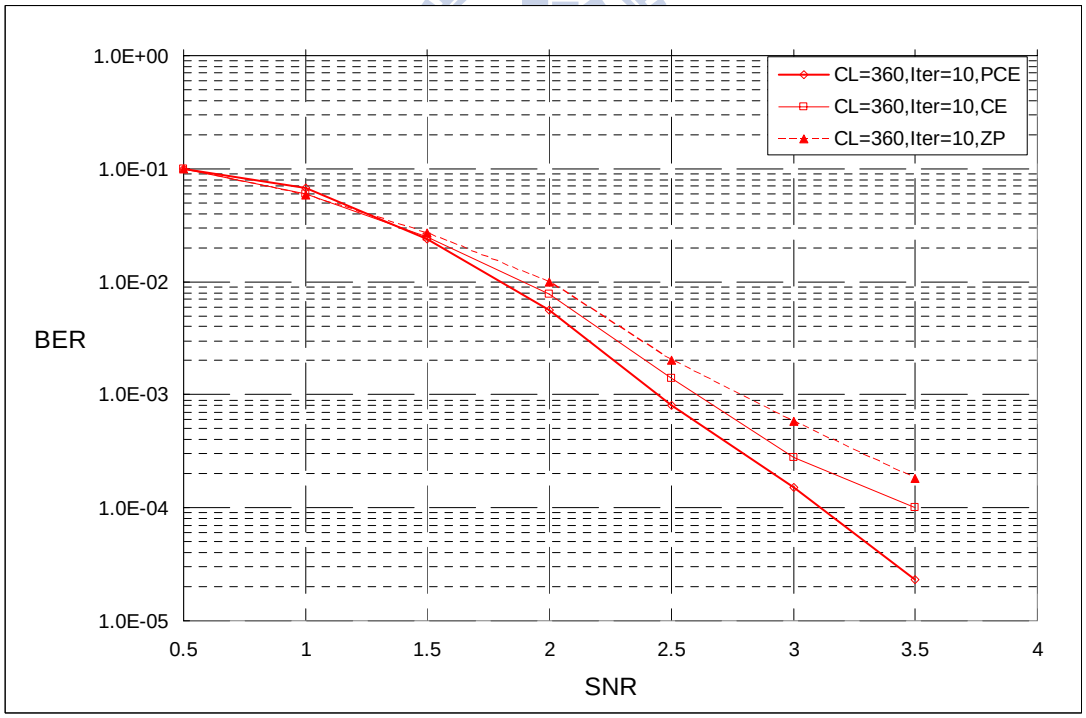


圖 4-14 PCE 與 CE 演算法編碼長度為 360 之效能圖

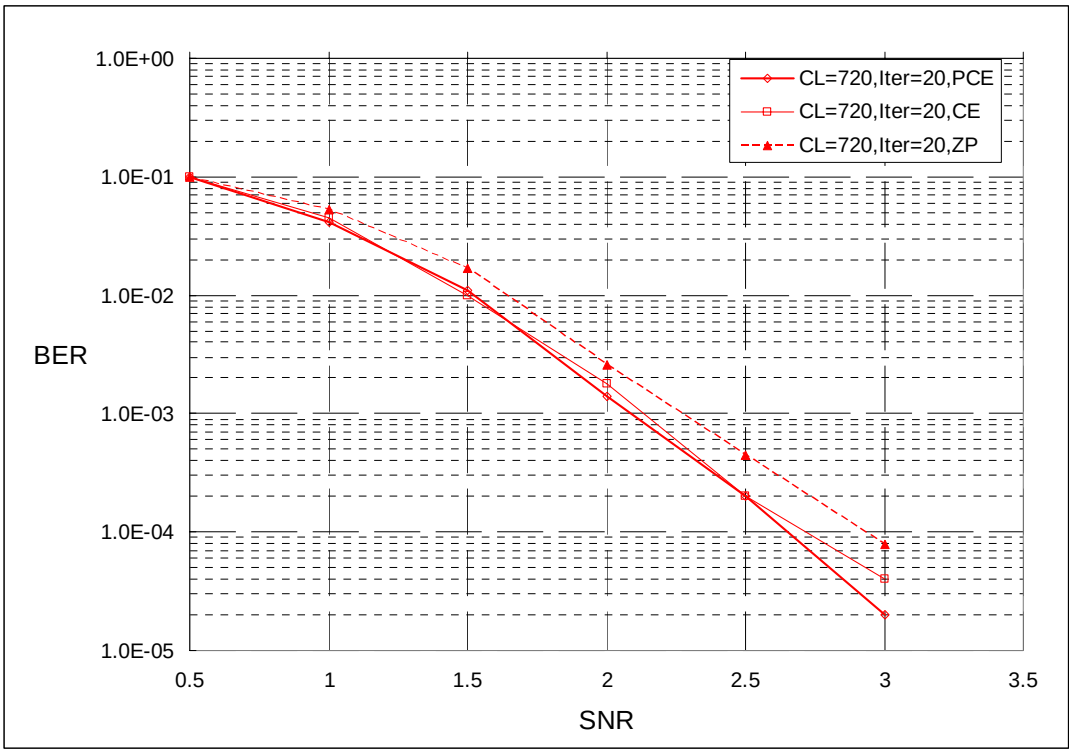


圖 4-15 PCE 與 CE 演算法編碼長度為 720 之效能圖

(2) 編碼長度於 1000 至 4000 之模擬與分析：

圖 4-16 為不同擴展倍數下所剩餘短迴圈總量，由於 cycle-8 數量非常多，所以傳統消去迴圈演算法在擴展 60 倍時只能執行左半邊少許元素的位移動作而留下大量的迴圈，反之改良式演算法直接對最高優先權動作消除許多迴圈，其中擴展 50 倍之內已消除所有 cycle-4、6 迴圈，因此圖 4-16 也表示剩餘 cycle-8 的數量。

模擬所使用的基本矩陣同上例，擴展的矩陣倍數為 50 與 100，遞迴次數為 30 次。在圖 4-17 中 SNR 大於 1.5 時，長度 1800 的效能曲線有明顯差異，因為擴展 50 倍時傳統演算法所留下 cycle-8 的迴圈數比改良式演算法高出許多，再次驗證高 SNR、低錯誤率時短迴圈的影響更大[34]。此外若為了省電採用低遞迴次數解碼，在圖 4-18 所示遞迴次數為 15 次也能得到解碼效能的改進；在擴展 100 倍時兩者演算法所留下的迴圈數較接近，因此圖 4-19 反應效能上也比較相近，但改良式演算法能消除更多的短迴圈以增進解碼效能。

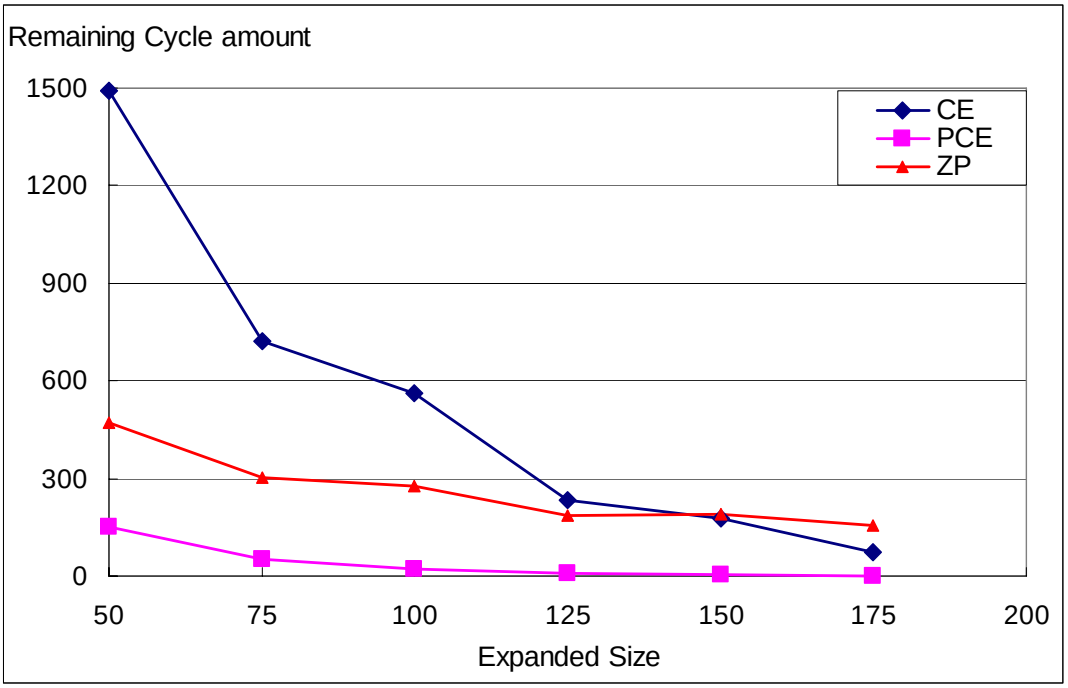


圖 4-16 不同擴展倍數下所剩餘迴圈總量

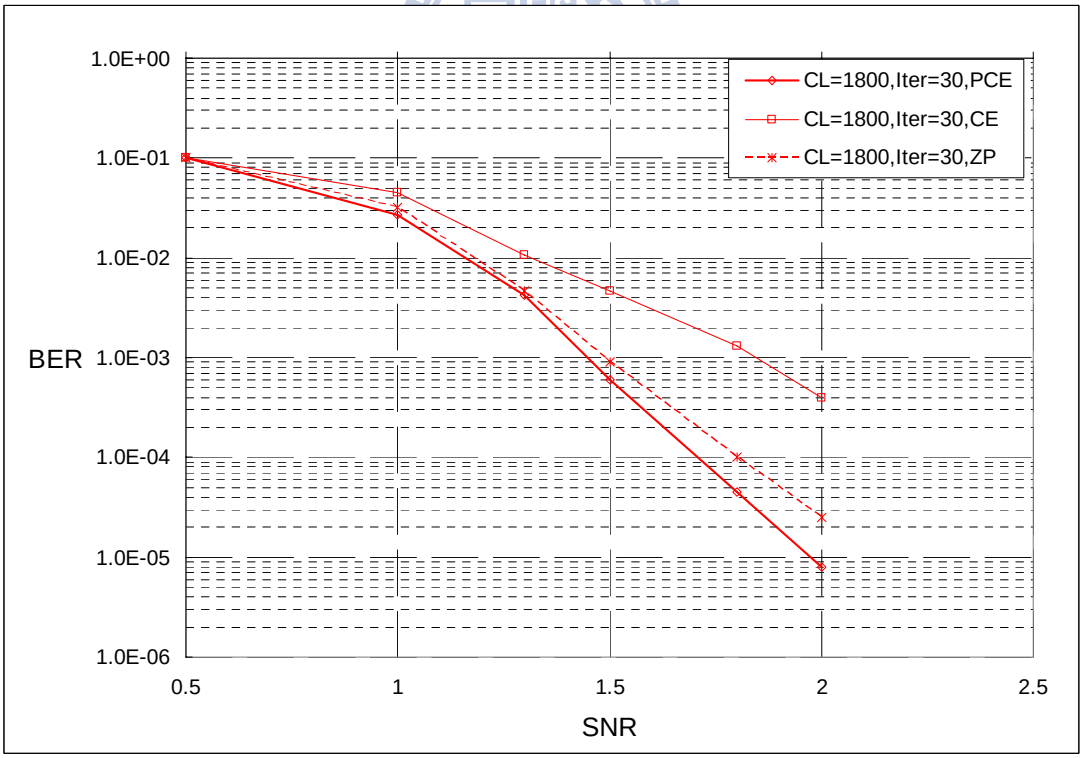


圖 4-17 PCE 與 CE 演算法編碼長度為 1800 之效能圖

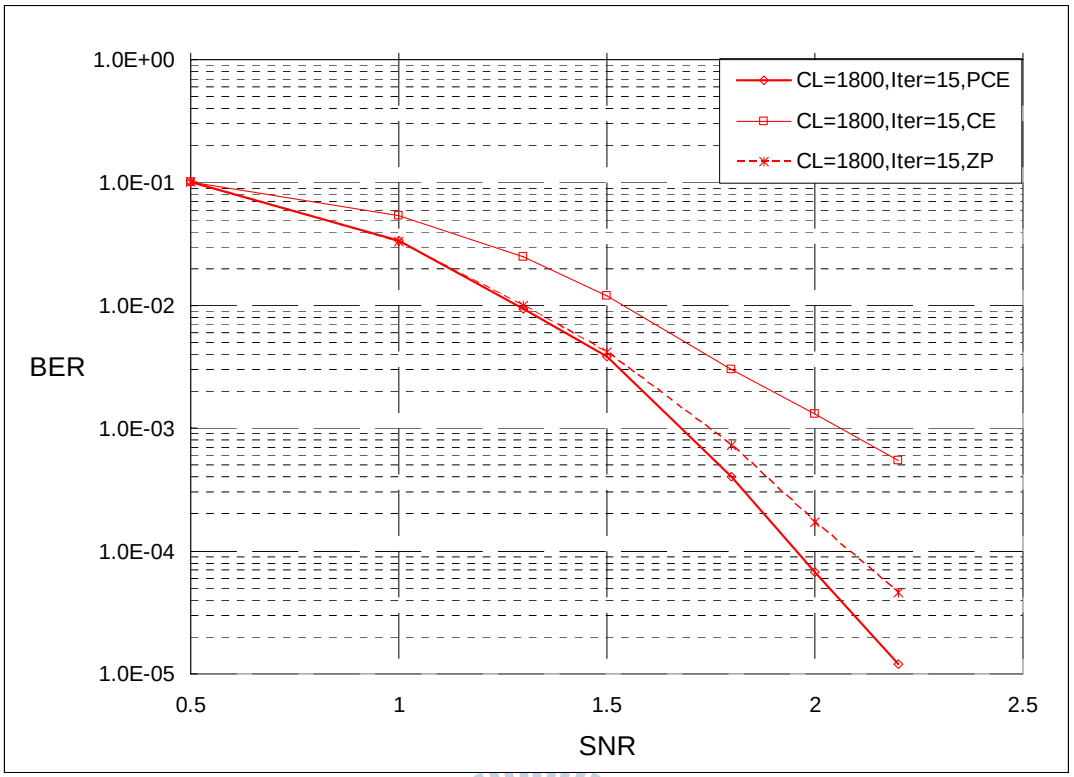


圖 4-18 PCE 與 CE 演算法編碼長度為 1800 之效能圖

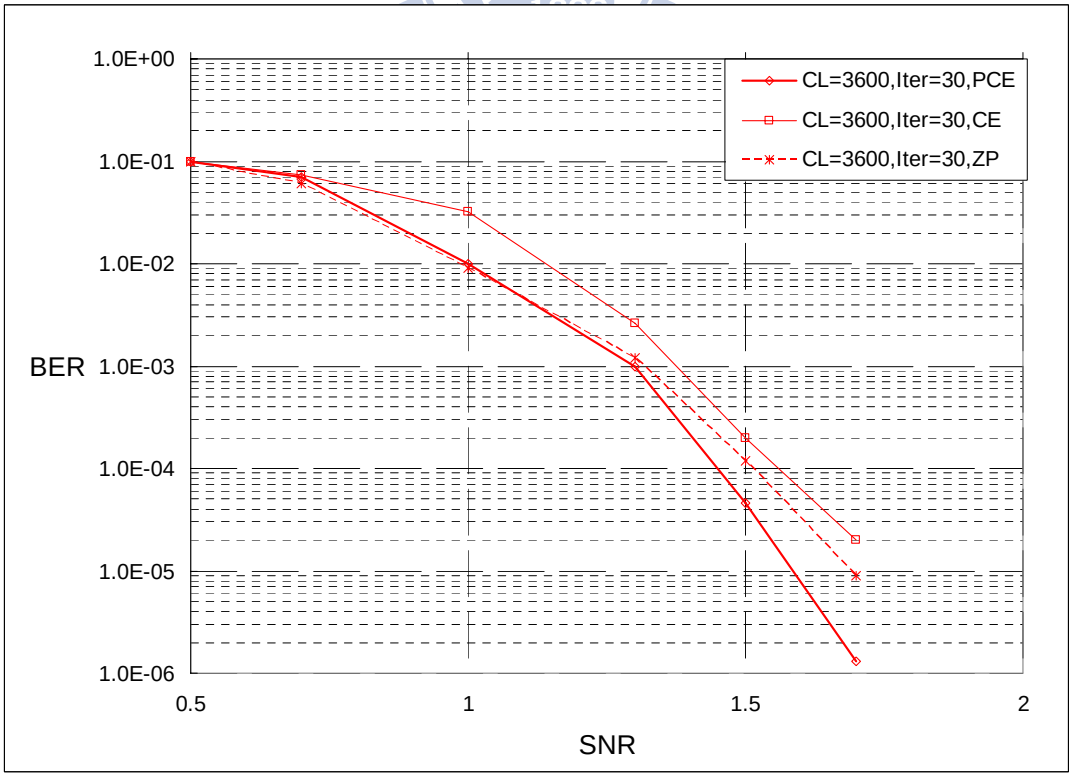


圖 4-19 PCE 與 CE 演算法編碼長度為 3600 之效能圖

第五章 結論

本篇論文探討 LDPC Codes 設計並適用於低編碼長度之應用，所建構的位元檢查矩陣採用 Block-LDPC[17]的形式有利於硬體上實現，主要針對兩方面做演算法改良，一方面改良式演算法比傳統式演算法節省大量運算及複雜度，另一方面在不同的擴展倍數均能有效打斷迴圈的連結以增進解碼效能。

改良式演算法以每個元素 1 的迴圈總數決定優先順序，從最高優先權開始消除迴圈動作，充分利用每次打斷迴圈的機會故減低了演算法遞迴次數，尤其在前幾次遞迴動作時有效減少大量迴圈，所以在消去相同迴圈總數時，運算量相較傳統演算法減低約 70~80%，以 Matlab 為模擬平台之模擬時間減少約 90%，均可證明改良式演算法能節省大量運算及複雜度。

在移動通訊或無線通訊的應用中[35]，若使用 LDPC 當作編、解碼的媒介而採用的矩陣大多為短編碼長度的矩陣，因為長編碼矩陣除了帶來大量運算，也需要更多次數的遞迴解碼使效能收斂，應用上更為耗電。因此我們著重在產生一個好的 LDPC 矩陣，再加上優先權為基礎之特性，在前幾次遞迴運算時就能打斷大量迴圈，能越早消除大量迴圈則在低擴展倍數時越有利，因為遞迴次數的增加讓位移值的選擇越來越少。所以模擬編碼長度在 4000 位元以下的矩陣作為解碼矩陣，解碼的效能上均比其它兩種相同架構的演算法要來得優異，尤其在低錯誤率時效能增進更明顯，突顯出低編碼長度時能夠有效率消除迴圈，適用於省電或低運算量等通訊系統。

參考文獻

[1] C. E. Shannon, “A Mathematical Theory of Communication,” Bell Syst. Tech. J., pp. 379-423 (Part 1); pp. 623-56 (Part 2), July 1948.

[2] Shu Lin, D. J. Costello, “Error Control Coding 2nd ed.,” Person-Prentice Hall, 2004.

[3] E. R. Berlekamp, Algebraic Coding Theory, McGraw-Hill, New York, 1968; Rev. ed., Aegean Park Press, Laguna Hills, N. Y., 1984.

[4] P. Elias, “Coding for Noisy Channels,” IRE Conv. Rec., p. 4:37-47, 1955.

[5] E. Prange, “Cyclic Error-Correcting Codes in Two Symbols,” AFCRC-TN-57, 103, Air Force Cambridge Research Center, Cambridge, Mass., September 1957.

[6] I. S. Read and G. Solomon, “Polynomial Codes over Certain Fields,” J. Soc. Ind. App. Math., 8:300-304, June 1960.

[7] R. G. Gallager, “Low-Density Parity-Check Codes”. Cambridge, MA: MIT Press, 1963.

[8] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes,” IEEE Intl. Conf. Commun., pp. 1064-70, Geneva, Switzerland, May 1993.

[9] D.J.C Mackay and R.M Neal, ”Near Shannon limit performance of low-density parity-check codes,” Electron. Lett. vol.32, pp.1645-1646, Aug. 1996.

[10] N. Wiberg, “Codes and Decoding on General Graphs,” IEEE Conf. Inf. Theory, p.468, Sep. 1995.

[11] A. Delvarathinam, E. Kim, and G. Choi, “Low-Density Parity-Check Decoder Architecture for High Throughput Optical Fiber Channels,” IEEE Intl. Conf. Comp. Design, pp. 520–525, 2003.

[12] A. J. Blanksby and C. J. Howland, “A 690-mW 1-Gb/s 1024-b, rate-1/2 Low-Density Parity-Check Code Decoder,” IEEE Trans. Solid-State Circuits, vol. 37, no.3, pp. 404–412, Mar. 2002.

[13] T. Richardson and R. Urbanke, “Efficient encoding of Low-Density Parity-Check Codes,” IEEE Trans. Inf. Theory, vol. 47, no. 2, pp. 638–656, Feb. 2001.

[14] T. Richardson, A. Shokrollahi, and R. Urbanke, “Design of Capacity Approaching Low-Density Parity-Check Codes, “IEEE Trans. Inf. Theory, vol.47, no.2, pp. 619-637, Feb. 2001.

[15] R. MICHAEL TANNER, “A Recursive Approach to Low Complexity Codes,” IEEE Trans. Inf. Theory, vol.27, no.2, pp.533-547, Sep. 1981.

[16] D. J. C. MacKay, “Good error-correcting codes based on very sparse matrices,” IEEE Trans. Inf. Theory, vol. 45, no. 2, pp. 399–431, Mar. 1999.

[17] M. Chiani, A. Conti, and A. Ventura, “Evaluation of Low-Density Parity-Check Codes Over Block Fading Channels,” IEEE Conf. Commun., vol.3 pp. 1183–1187, June. 2000.

[18] F. R. Kschischang, B. J. Frey, and H. A. Loeliger, “Factor Graphs and the Sum-Product Algorithm,” IEEE Trans. Inf. Theory, vol. 47, pp498-519, Feb 2001.

[19] 朱元志, 低密度對偶檢查碼之改進以及其解碼器之超大型積體電路實現, 國立交通大學論文, 2005.

[20] Xiao-Yu Hu, E. Eleftheriou, D. M. Arnold, A. Dholakia,” Efficient Implementations of the Sum-Product Algorithm for Decoding LDPC,” IEEE Conf. Globe Telecom, pp. 1036-1036E, Nov. 2001.

[21] Lei Yang, Hui Liu, C.-J Richard Shi, “Code Construction and FPGA Implementation of a Low-Error-Floor Multi-Rate Low-Density Parity-Check Code Decoder,” IEEE Trans. Circuit and Systems, vol. 53, no. 4, Apr. 2006.

[22] D. E. Hocevar, “LDPC Code Construction with Flexible Hardware Implementation,” IEEE Conf. Commun., vol.4, pp. 2708–2712, May 2003.

[23] K. Shimizu, T. Ishikawa, N. Togawa, T. Ikenaga, S. Goto, “Partially-Parallel LDPC Decoder Based on High-Efficiency Message-Passing Algorithm,” IEEE Conf. Com., pp. 503-510, Oct. 2005.

[24] X. Y. Hu, E. Eleftheriou, and D. M. Arnold, “Progressive Edge-Growth Tanner Graphs,” IEEE Conf. Global Telecomm., vol. 2, pp.995–1001, Nov.2001.

[25] J. Campello, D. S. Modha, and S. Rajagopalan, “Designing LDPC Codes Using Bit-Filling,” IEEE Conf. Com., pp. 55–59, June 2001.

[26] H. Zhang and T. Zhang, “Design of VLSI Implementation-Oriented LDPC Codes,” IEEE Conf. Technol., vol. 1, pp. 670–673, Oct. 2003.

[27] L.Yang, H. Liu, and C.-J. R. Shi, “Cycle Elimination Method to Construct VLSI Oriented LDPC Codes,” IEEE Technol. Conf., pp. 522–526, Sep. 2005.

[28] D. J. C. Mackay, S. T. Wilson, and M. C. Davey, “Comparison of Constructions of Irregular Gallager Codes,” IEEE Trans. Comm., vol. 47, pp.1449–1454, Oct. 1999.

[29] T. Tian, C. Jones, J. D. Villasenor, and R. D.Wesel, “Construction of Irregular LDPC Codes with Low Error Floors,” IEEE Conf. Com., vol. 5, pp. 3125–3129, May 2003.

[30] T. Richardson, “Error Floors of LDPC Codes,” IEEE Conf. Commun., pp. 1426–1435, Oct. 2003.

[31] H. Zhang and T. Zhang, “Block-LDPC: A Practical LDPC Coding System Design Approach,” IEEE Trans. Circuits and Systems, vol. 52, no. 4, Apr. 2005.

[32] Campello J., Modha D.S., “Extended Bit-Filling and LDPC Code Design,” IEEE Conf. Global Telecomm., vol. 2, pp. 25-29, Nov. 2001.

[33] T. Zhang and K. K. Parhi, “VLSI Implementation-Oriented (3,k)-Regular Low-Density Parity-Check Codes,” IEEE Conf. Signal Process, pp. 25–36, Sep. 2001.

[34] J. M. F. Moura, Jin Lu, Haotian Zhang, “Structured Low-Density Parity-Check Codes,” IEEE Trans. Signal Process, vol. 21, pp.42-55, Jan. 2004.

[35] IEEE 802.16e-2005, “IEEE Standard for Local and Metropolitan Area Network - Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems – Amendment 2: Physical and Medium Access Control layers for Combined Fixed and Mobile Operation in Licensed Bands,” 2005.