國立交通大學

電機與控制工程學系

碩　士　論　文

用於無線通訊渦輪解碼之適應性運算量控制

Adaptive Iteration Control of Turbo Decoding for WiMAX Applications

研 究 生：謝博仁

指導教授：董蘭榮 博士

中 華 民 國 九 十 七 年 十 月

用於無線通訊渦輪解碼之適應性運算量控制

Adaptive Iteration Control of Turbo Decoding for WiMAX Applications

研 究 生：謝博仁　　Student：Po-Jen Hsieh

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

Eletrical and Control Engineering

October 2008

Hsinchu, Taiwan, Republic of China

中 華 民 國 九 十 七 年 十 月

用於無線通訊渦輪解碼之適應性運算量控制

研 究 生：謝博仁　　指導教授：董蘭榮 博士

國立交通大學電機與控制工程學系

摘要

渦輪解碼是一種利用遞迴的方式來完成解碼的動作，而解碼次數與通道的傳輸品質有相當大的關係，當通道品質相當好時，渦輪解碼可以很快的解碼出正確的資料，然而通道環境惡劣時，渦輪解碼幾乎都得解到所預設的最大解碼次數才會停止。而與其最後才發現解碼是失敗的，到不如及早停止解碼。因此我們提出一個能夠估測封包解碼失敗的方法，並且在盡可能不影響錯誤率的前提下，盡快停止無意義的遞迴動作，以節省不必要的功率消耗，並能縮短解碼時間，提早重傳。其中我們利用一個簡單的方法觀察解碼資料事後機率，來做為判斷是否要放棄解碼的機制，我們稱此機制為預棄技術(Early Give-up)。

另外，基於重傳的封包資料相同的情況下，我們可以利用上一筆解碼失敗所留下來的資訊，做為新的解碼開始的初始值，使得整體平均的解碼遞迴次數能夠再下降，根據模擬結果，結合以上兩種方法，在吵雜的通道環境下我們可以省下最多約80%的遞迴次數。

Adaptive Iteration Control of Turbo Decoding for WiMAX Applications

Student：Po-Jen Hsieh Advisor：Lan-Rong Dung

Institute of Electrical and Control Engineering National Chiao-Tung University

Abstract

Turbo Decoding is a kind of iterative decoding process.The number of iterations required to correctly decode the packet depend on the channel condition.Turbo Decoding can obtain correct information quickly when the channel condition is good enough.When the channel is noisy, turbo decoding always needs a lot of the number of iterations which is close to the preset maximum iterations.So we propose a method which can predict that the packet will not be decoded correctly,giving up the decoding process earlier and request data to be re-transmitted immediately. We judge whether the packet is a failed pattern based on observations from the average absolute value of a posteriori probability of turbo decoding. The techbique called “Early Give-up”.

Besides, we can reuse the prior MAP information of the failed process based on the assumption of correlation between same packets transmitted at different times. Simulation results shows that the average iterations required to decode a packet can be reduced up to 80% under bad channel conditions.

誌謝

本篇論文得以順利完成，首先要感謝的是我的指導教授──董蘭榮教授，太老實且領悟力又很差的我，總是讓老師很頭痛，但老師還是不厭其煩的指導我，教我做事的方法與態度，如何冷靜有邏輯的思考問題，讓我學到了很多書上學不到的事。

同時，我要感謝我的爸媽，沒有你們辛辛苦苦的栽培，不會有今天的我，我能夠受到這麼好的教育，完完全全都要感謝你們。還有要感謝我的哥哥，大姊以及二姊，有你們的陪伴與支持，是我努力做好我自己的動力。

另外，也感謝實驗室的學長們──穎毅、騰轟、學之、文豪、俊衛、峻轍、志惟、仕捷、信承，你們予我分享了許多經驗還有幫助，也給了我很多快樂的回憶。還有感謝陪伴兩年的同學們──詠麟、仁德斯、小康、Kong，在課業與生活上的互相扶持、分擔紓解彼此的壓力，我覺得很幸運能夠認識你們。當然也要感謝實驗室的學弟妹們和兩位助理──邱哥、單哥、智聖、小嘉鴻、貞如、乃禎、惟茵，因為有了你們使得枯燥煩悶的實驗室快樂許多。

謹將此論文獻給所有關心我的人，在此致上最深的謝意。

章節目錄

圖目錄

第一章 緒論

渦輪解碼(Turbo Decoding)自 1993 年被提出後[1]，由於其有著接近於通道容量的仙儂極限(Shannon limit)的優異錯誤更正能力，在通道編碼的領域中，一直是個很熱門的研究主題，同時也被廣泛的應用在各個標準協定中，如 3GPP、DVB-RCS、IEEE 802.16 等等。

然而渦輪解碼耗費龐大的運算量，並且需要大量的記憶體，導致功率上的消耗相對的也不小，而這缺點對於可攜式的行動設備內有限的電池電量造成很大的負擔。除此之外，渦輪解碼需要較長的解碼時間，如何增加整體解碼的吞吐量也是門重要的課題。由於渦輪解碼是種以遞迴的方式來做解碼的方法，所以若能夠省下無意義的遞迴次數，相對的就省下更多的功率消耗，不僅如此，也能夠增加整體的吞吐量(thoughput)。

對於傳統的作法而言，遞迴次數是固定成某個數值，然而這種作法並不能適性通道品質的變化，在大部分的情形下，這都不是很實際的作法，而在如何省下遞迴次數方面，已經有大量的參考文獻研究著這方面的問題[7][24]，這些方法被稱為early-termination，而這些方法的精髓普遍在於如何去判斷所解碼的資料已經正確被解出，以便停止多餘的遞迴。這些方法在資料的訊雜比較大時，解碼的資料能夠很快的被判斷出已經收斂且解碼成功，便可以省下很多的遞迴次數，然而在通道狀態相當吵雜時，這些方法所提出的收斂條件便無法達到，以致於所需要的遞迴次數便幾乎等於所預設的最大的遞迴次數，但最後解碼可能一樣是失敗的。因此我們對此提出一個能夠估測封包解碼失敗的方法，且在儘量不影響錯誤率的前提下，盡快停止無意義的遞迴動作，以節省不必要的功率消耗。

另外基於重傳的封包資料相同的情況下，我們還能夠使用上一筆解碼失敗所留下來的資訊做為新的一筆資料解碼的初始值[22]，使得整體平均的解碼遞迴次數能夠再下降。

1-1 研究動機

對於行動式的裝置來說，由於有限的電池容量，所以功率消耗的議題非常的重要，對於渦輪解碼這種解碼方式來說，省下遞迴次數便等於省下功率消耗，而我們希望能夠適應通道的狀態有效率的做解碼，並且在盡量不損失錯誤率效能的前提下盡可能地減少功率的消耗。

1-2 章節規劃

本篇論文的章節架構如下：

第一章：序論

第二章：渦輪解碼技術

在此第二章中，我們簡單的介紹渦輪解碼中所使用的 MAP 演算法，以及 log-MAP 演算法如何對 MAP 演算法做複雜度上的改善，還有渦輪編碼器以及解碼器的架構與概念做說明，而最後我們會再介紹渦輪碼應用在 IEEE 802.16 上的規格。

第三章：渦輪解碼中的預棄與狀態再利用技術

此章節是我論文的主體，我將會就這兩種技術的概念與想法以及一些問題如何去克服做討論。

第四章：模擬結果比較

本章中總結綜合第三章的分析，歸納出一個最有效的方法，並對各種案例作模擬。

第五章：總結與未來展望

第二章 渦輪解碼技術

渦輪碼最早於 1993 年由 Berrou, Glavieux 以及 Thitimajshima 所提出[1]，是通道編碼領域中的非常重大的突破，它有著非常優異的錯誤更正能力，相當接近於通訊理論上薛儂極限值(Shannon limit)，而理論上在白色高斯雜訊(AWGN)的通道下，當訊雜比(Eb/No)為 0.7dB 時，此系統的錯誤更正能力能夠使得位元錯誤率(Bit Error Rate)達到 10^{-5}。渦輪碼在近年來被廣泛應用在無線通訊領域上，包含數位影像廣播(DVB-RCS)、第三代行動電話(3GPP)，以及 WiMAX 等等之應用，而本論文是利用 IEEE 802.16 中所制訂的 Convolutional Turbo Code 的規格作為研究的平台。

2-1 渦輪碼的編碼架構

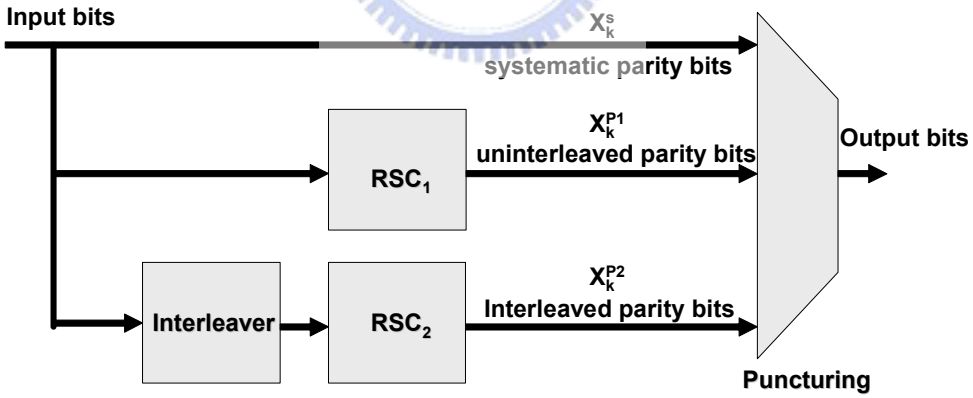

圖 1 渦輪編碼器之架構圖

渦輪碼的編碼器如圖 1 所示主要由兩個相同的 RSC(Recursive Systematic Convolutional)編碼器，以及一個交錯器(interleaver)所組成。欲編碼的資料流(input bits)會經由第一個 RSC 編碼器做編碼，而得到一組平等位元資料(uninterleaved parity bits)，另一方面，資料流同時會經過交錯器，得到一

組相對於原始的資料流有著相當低的關連性的資料，再經過相同的 RSC 編碼器做編碼，得到第二組平等位元資料(interleaved parity bits)，所以基本上渦輪碼的原始碼率(Coding Rate)為 1/3，而最後全部的三組資料會再經由 Puncturer 根據所需要的較高的碼率，利用特定的 puncturing table 去選擇特定所要傳輸的位元。

2-2 渦輪碼的解碼架構

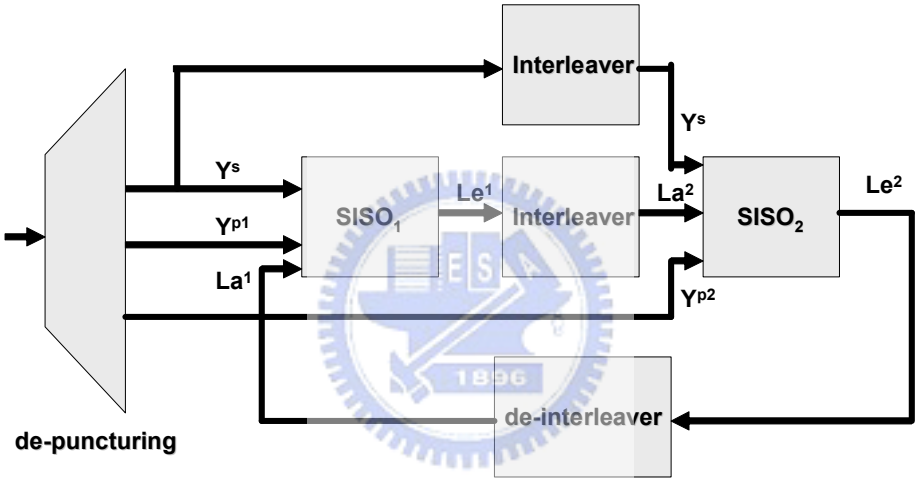

圖 2 渦輪解碼器之架構圖

渦輪解碼器如圖 2 所示主要由兩個 SISO(Soft-Input Soft-Output)解碼器，以及兩個交錯器和反交錯器(de-interleaver)所組成。De-puncturing 的作用在於將被 puncture 掉的位元填回來，以便得到一個完整的 codeword。

渦輪解碼的概念如下，首先 SISO 解碼器會利用從通道中收到的 systematic bits 所對應的值 Y^s，以及 uninterleaved parity bits 所對應的值 Y^{p1}，以及一組事前資訊(a priori information)La^1，來計算每個位元為 1 或 0 的機率的比值，之後便產生了一組外部資訊(extrinsic information)，再經過了交錯器之

後，便成為第二組 SISO 解碼器所需要的事前資訊 La^{2}，接著第二組 $SISO_2$ 解碼器便利用 La^{2} 以及 systematic bits 經過交錯器所得到的值，和 interleaved parity bits 所對應的 Y^{p2} 來計算每個位元為 1 或 0 的機率的比值，一樣地，$SISO_2$ 解碼完後，會產生一組外部資訊供 $SISO_1$ 在下一次的解碼用。而在做完一次 $SISO_1$ 以及 $SISO_2$ 的解碼，這樣便算是完成了一次的渦輪解碼的遞迴，而渦輪解碼便是利用這樣的遞迴關係反覆的解碼，越多次的遞迴能夠得到越高的錯誤更正能力。另外值得一提的是，在第一次遞迴的開始，La^{1} 並還沒有辦法從 $SISO_2$ 解碼器那獲得外部資訊，所以基本上，第一次解碼的開始 La^{1} 會被設成零。

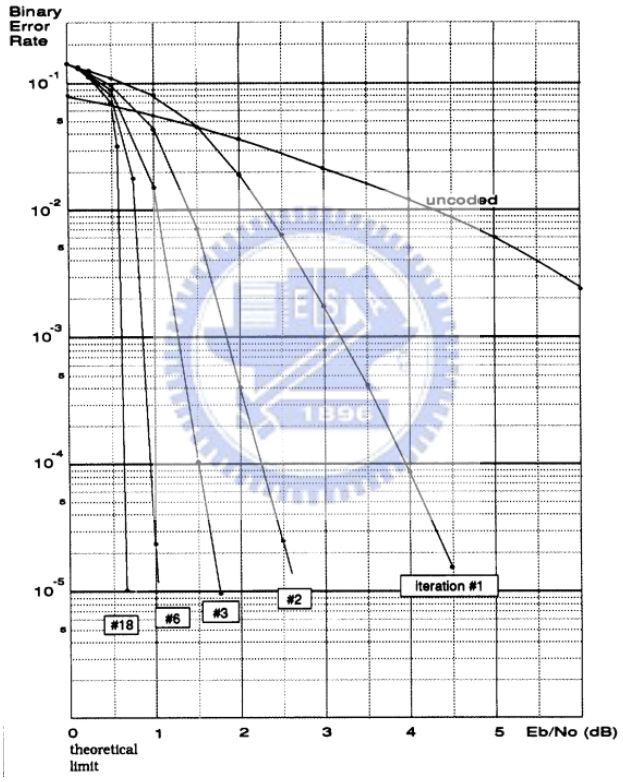

圖 3 [1]中所預期的渦輪解碼的位元錯誤率效能

從圖 3 中可看到渦輪碼的錯誤更正能力與解碼所使用的遞迴次數有很大的正比關係，當所使用的遞迴次數等於 18 次時，在 Eb/No 約等於 0.7dB 時，錯誤率便可達到 10^{-5}。

對於渦輪解碼的 SISO 解碼器，能夠使用最大事後機率 MAP(Maximum a posteriori)解碼器或是 SOVA(Soft-output Viterbi algorithm)解碼器來完成，而在此篇論文中，所選擇的是以 MAP 解碼器來實現。

2-3 最大事後機率演算法(MAP Algorithm)

MAP 演算法在 1974 年由 Bahl, Cocke, Jelinek 以及 Raviv 所提出，所以亦稱為 BCJR 演算法，它是種軟式輸入軟式輸出(Soft Input Soft Output)的解碼方式。MAP 演算法的精神在於利用所有從通道所接收到對應於每個位元的軟性值(soft value) R_1^N ，去估算每個位元 d_k 為 1 和 0 的事後機率的比值，即 LLR(log-likelihood ratio)，若 LLR 的值大於零便解碼為 1，反之便解碼為 0。

$$LLR(d_k) = \ln \frac{\Pr\{d_k = 1 \mid R_1^N\}}{\Pr\{d_k = 0 \mid R_1^N\}} \tag{1}$$

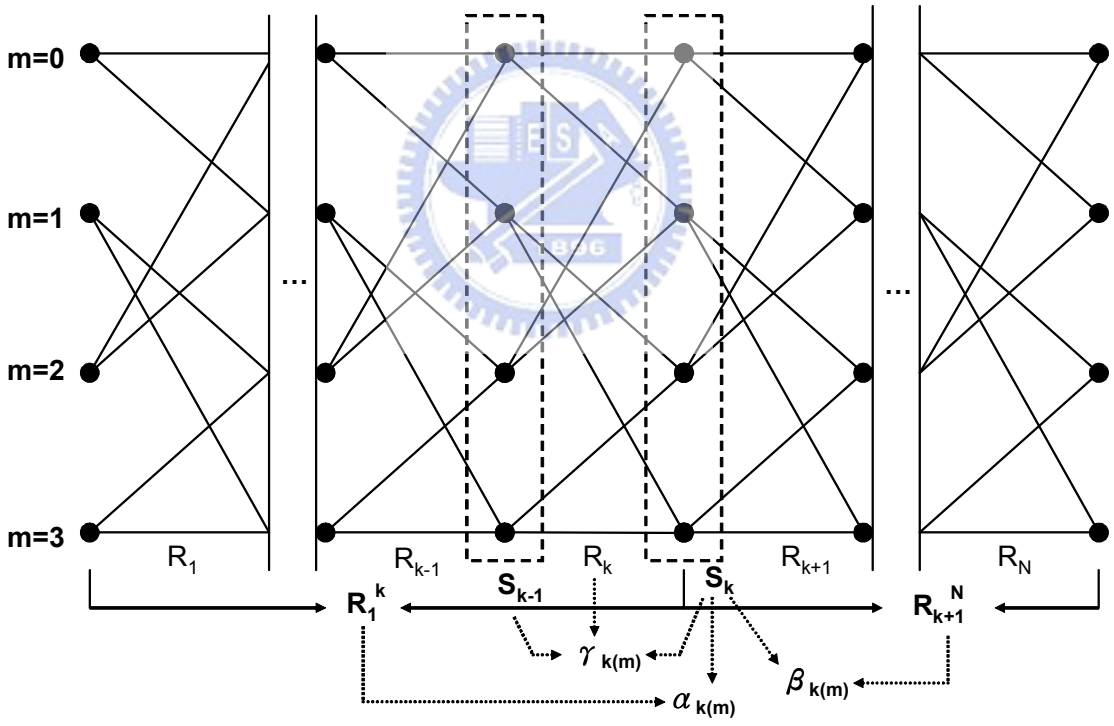

圖 4 在 MAP 演算法中各個路徑值的相關性(參考自[3])

而如何計算(1)式中所示的 $LLR(d_k)$ ，MAP 演算法其實利用到了格子狀態圖(trellis state diagram)來做為整個機率推導的基礎，如圖 4 所示。

我們觀察到(1)式中的 $\Pr\{d_k = i \mid R_1^N\}$ 可利用格子狀態圖表示成

$$\Pr\{d_k = i \mid R_1^N\} = \sum_m \sum_{m'} \Pr\{d_k = i, S_k = m, S_{k-1} = m' \mid R_1^N\} \quad (2)$$

其中(2)式中的 m 與 m' 所指的是當 $d_k = i$ 時，所有可能會從狀態 m' 跳到狀態 m 的集合。而(2)式可以分解如下

$$\begin{aligned}
&\sum_m \sum_{m'} \Pr\{d_k = i, S_k = m, S_{k-1} = m' \mid R_1^N\} \\
&= \sum_m \sum_{m'} \frac{\Pr\{d_k = i, S_k = m, S_{k-1} = m', R_1^N\}}{\Pr\{R_1^N\}} \\
&= \sum_m \sum_{m'} \frac{\Pr\{d_k = i, S_k = m, S_{k-1} = m', R_1^{k-1}, R_k, R_{k+1}^N\}}{\Pr\{R_1^N\}} \\
&= \sum_m \sum_{m'} \frac{\Pr\{d_k = i, S_k = m, R_k, R_{k+1}^N \mid S_{k-1} = m', R_1^{k-1}\} \Pr\{S_{k-1} = m', R_1^{k-1}\}}{\Pr\{R_1^N\}}
\end{aligned} \quad (3)$$

其中(3)式的最後由於當狀態到達 S_{k-1} 後，後面的所有狀態與編碼的輸出值都已和 R_1^{k-1} 沒有關係，所以

$$\begin{aligned}
\Pr\{d_k = i, S_k = m, R_k, R_{k+1}^N \mid S_{k-1} = m', R_1^{k-1}\} &= \Pr\{d_k = i, S_k = m, R_k, R_{k+1}^N \mid S_{k-1} = m'\} \\
&= \frac{\Pr\{d_k = i, R_k, R_{k+1}^N, S_k = m, S_{k-1} = m'\}}{\Pr\{S_{k-1} = m'\}}
\end{aligned} \quad (4)$$

其中 $\Pr\{d_k = i, R_k, R_{k+1}^N, S_k = m, S_{k-1} = m'\}$ 可以拆解如下

$$\begin{aligned}
&\Pr\{d_k = i, R_k, R_{k+1}^N, S_k = m, S_{k-1} = m'\} \\
&= \Pr\{R_{k+1}^N \mid d_k = i, R_k, S_k = m, S_{k-1} = m'\} \Pr\{d_k = i, R_k, S_k = m, S_{k-1} = m'\}
\end{aligned} \quad (5)$$

一樣地，當狀態 S_k 到達 m 後，之後的狀態轉移與輸出皆與 $d_k = i, R_k, S_{k-1} = m'$ 這三項沒有關係，所以 $\Pr\{R_{k+1}^N \mid d_k = i, R_k, S_k = m, S_{k-1} = m'\} = \Pr\{R_{k+1}^N \mid S_k = m\}$ (6)

(5)式中的 $\Pr\{d_k = i, R_k, S_k = m, S_{k-1} = m'\}$ 可做類似的拆解如下

$$\Pr\{d_k = i, R_k, S_k = m, S_{k-1} = m'\} = \Pr\{d_k = i, R_k, S_k = m \mid S_{k-1} = m'\} \Pr\{S_{k-1} = m'\} \quad (7)$$

所以綜合(2)~(7)式，我們可以得到

$$\Pr\{d_k = i \mid R_1^N\} = \sum_m \sum_{m'} \frac{\Pr\{R_{k+1}^N \mid S_k = m\} \Pr\{d_k = i, R_k, S_k = m \mid S_{k-1} = m'\} \Pr\{S_{k-1} = m', R_1^{k-1}\}}{\Pr\{R_1^N\}}$$

(8)

再將(8)套入(1)中得到

$$\begin{aligned} LLR(d_k) &= \ln \frac{\sum_m \sum_{m'} \Pr\{R_{k+1}^N \mid S_k = m\} \Pr\{d_k = 1, R_k, S_k = m \mid S_{k-1} = m'\} \Pr\{S_{k-1} = m', R_1^{k-1}\}}{\sum_m \sum_{m'} \Pr\{R_{k+1}^N \mid S_k = m\} \Pr\{d_k = 0, R_k, S_k = m \mid S_{k-1} = m'\} \Pr\{S_{k-1} = m', R_1^{k-1}\}} \\ &= \ln \frac{\sum_m \sum_{m'} \Pr\{S_{k-1} = m', R_1^{k-1}\} \Pr\{d_k = 1, R_k, S_k = m \mid S_{k-1} = m'\} \Pr\{R_{k+1}^N \mid S_k = m\}}{\sum_m \sum_{m'} \Pr\{S_{k-1} = m', R_1^{k-1}\} \Pr\{d_k = 0, R_k, S_k = m \mid S_{k-1} = m'\} \Pr\{R_{k+1}^N \mid S_k = m\}} \end{aligned}$$

(9)

其中我們再定義

$$\alpha_{k-1}(m') = \Pr\{S_{k-1} = m', R_1^{k-1}\} \quad (10)$$

$$\gamma_k^i(m', m) = \Pr\{d_k = i, R_k, S_k = m \mid S_{k-1} = m'\} \quad (11)$$

$$\beta_k(m) = \Pr\{R_{k+1}^N \mid S_k = m\} \quad (12)$$

將(10)~(12)式代入(9)式中，最後得到 MAP 演算法實際上所計算的式子

$$LLR(d_k) = \ln \frac{\sum_m \sum_{m'} \alpha_{k-1}(m') \gamma_k^1(m, m') \beta_k(m)}{\sum_m \sum_{m'} \alpha_{k-1}(m') \gamma_k^0(m, m') \beta_k(m)} \quad (13)$$

如圖 4 所示，$\alpha_k(m)$ 實際上利用到第 1 個到第 k 個時間點從通道中所收到的 $R_1^k (= R_1, R_2, ..., R_k)$ 值，以及狀態 S_k 的機率去計算，而以下將會推導 $\alpha_k(m)$ 實際上可由前一個時間點的所有 $\alpha_{k-1}(m')$ 來遞迴算出，所以 $\alpha_k(m)$ 稱為前向路徑狀態值 (Forward State Metrics)。

$$\begin{aligned}
\alpha_k(m) &= \Pr\{S_k = m, R_1^k\} = \sum_{m'=0}^{M-1} \Pr\{S_{k-1} = m', S_k = m, R_1^k\} = \sum_{m'=0}^{M-1} \Pr\{S_{k-1} = m', S_k = m, R_1^k\} \\
&= \sum_{m'=0}^{M-1} \Pr\{S_{k-1} = m', S_k = m, R_1^{k-1}, R_k\} = \sum_{m'=0}^{M-1} \Pr\{S_{k-1} = m', S_k = m, R_1^{k-1}, R_k\} \\
&= \sum_{m'=0}^{M-1} \Pr\{S_k = m, R_k \mid S_{k-1} = m', R_1^{k-1}\} \Pr\{S_{k-1} = m', R_1^{k-1}\} \\
&= \sum_{m'=0}^{M-1} \Pr\{S_k = m, R_k \mid S_{k-1} = m'\} \Pr\{S_{k-1} = m', R_1^{k-1}\} \\
&= \sum_{m'=0}^{M-1} \Pr\{S_{k-1} = m', R_1^{k-1}\} \Pr\{S_k = m, R_k \mid S_{k-1} = m'\} \\
&= \sum_{m'=0}^{M-1} \alpha_{k-1}(m') \gamma_k(m', m)
\end{aligned} \tag{14}$$

另外，如圖 4 所示，$\beta_k(m)$ 實際上利用到第 k+1 個到第 N 個時間點從通道中所收到的 $R_{k+1}^N (= R_{k+1}, R_{k+2}, ..., R_N)$ 值，以及狀態 S_k 的機率去計算，而以下將會推導 $\beta_k(m)$ 實際上可由後一個時間點的所有 $\beta_{k+1}(m')$ 來遞迴算出，所以 $\beta_k(m)$ 稱為後向路徑狀態值(Backward State Metrics)。

$$\begin{aligned}
\beta_k(m) &= \Pr\{R_{k+1}^N \mid S_k = m\} = \sum_{m'=0}^{M-1} \Pr\{S_{k+1} = m' R_{k+1}^N \mid S_k = m\} = \sum_{m'=0}^{M-1} \frac{\Pr\{S_{k+1} = m' R_{k+1}^N, S_k = m\}}{\Pr\{S_k = m\}} \\
&= \sum_{m'=0}^{M-1} \frac{\Pr\{S_{k+1} = m' R_{k+1}, R_{k+2}^N, S_k = m\}}{\Pr\{S_k = m\}} \\
&= \sum_{m'=0}^{M-1} \frac{\Pr\{R_{k+2}^N \mid S_{k+1} = m' R_{k+1}, S_k = m\}}{\Pr\{S_k = m\}} \Pr\{S_{k+1} = m', R_{k+1}, S_k = m\} \\
&= \sum_{m'=0}^{M-1} \frac{\Pr\{R_{k+2}^N \mid S_{k+1} = m', R_{k+1}, S_k = m\}}{\Pr\{S_k = m\}} \Pr\{S_{k+1} = m', R_{k+1} \mid S_k = m\} \Pr\{S_k = m\} \\
&= \sum_{m'=0}^{M-1} \Pr\{R_{k+2}^N \mid S_{k+1} = m'\} \Pr\{S_{k+1} = m', R_{k+1} \mid S_k = m\} \\
&= \sum_{m'=0}^{M-1} \beta_{k+1}(m') \gamma_{k+1}(m, m')
\end{aligned} \tag{15}$$

而在(14)以及(15)式中，有個很重要的事情如何決定α與β的初始值(即$\alpha_0(m)$及$\beta_N(m)$)，對於一般的渦輪編碼的系統而言，兩個 RSC 編碼器的起始狀態皆為零狀態，所以對於此種系統而言，前向狀態路徑值的初始值的設定如下

$$\alpha_0(m)=\begin{cases}1, m=0\\0, m\neq 0\end{cases} \tag{16}$$

而 RSC 編碼器的最終狀態，對於一般的系統而言，在編碼時會多送一筆額外的資料，稱為 Termination bits，這些位元的目的在於將編碼的最終狀態設成零狀態，以便能夠讓後向狀態路徑值有明確的初始值如下

$$\beta_N(m)=\begin{cases}1, m=0\\0, m\neq 0\end{cases} \tag{17}$$

現在問題剩下如何去計算(11)式中的$\gamma_k^i(m',m)$，觀察此式我們可以看出這個值與$S_{k-1}=m'$狀態轉換到$S_k=m$狀態，以及發生此狀態轉變時，所相對應的輸入位元d_k，還有所收到的R_k有關，所以由於這個值與發生在格子狀態圖上的每條分支有關，$\gamma_k^i(m',m)$稱為分支路徑值(Branch Metrics)。

而(11)式中，$R_k=(y_k^s,y_p^s)$，y_k^s所代表的是對應於系統位元(systematic bits)從通道中所收到的值，而y_p^s所代表的是對應於平等位元(parity bits)從通道中所收到的值，所以$\gamma_k^i(m',m)$可以再拆解如下

$$\begin{aligned}\gamma_k^i(m',m)&=\Pr\{d_k=i,R_k,S_k=m\mid S_{k-1}=m'\}\\&=\frac{\Pr\{d_k=i,R_k,S_k=m,S_{k-1}=m'\}}{\Pr\{S_{k-1}=m'\}}\\&=\frac{\Pr\{R_k\mid d_k=i,S_k=m,S_{k-1}=m'\}}{\Pr\{S_{k-1}=m'\}}\Pr\{d_k=i,S_k=m,S_{k-1}=m'\}\\&=\frac{\Pr\{R_k\mid d_k=i,S_k=m,S_{k-1}=m'\}}{\Pr\{S_{k-1}=m'\}}\Pr\{d_k=i\mid S_k=m,S_{k-1}=m'\}\Pr\{S_k=m,S_{k-1}=m'\}\end{aligned}$$

$$= \Pr\{R_k \mid d_k = i, S_k = m, S_{k-1} = m'\} \Pr\{d_k = i \mid S_k = m, S_{k-1} = m'\} \Pr\{S_k = m \mid S_{k-1} = m'\}$$

(18)

其中 $\Pr\{d_k = i \mid S_k = m, S_{k-1} = m'\}$，若由 $S_{k-1} = m'$ 狀態轉變成 $S_k = m$ 狀態時輸入的位元 $d_k = i$，此條分支存在於格子點狀態圖上，則 $\Pr\{d_k = i \mid S_k = m, S_{k-1} = m'\} = 1$，若不存在，則 $\Pr\{d_k = i \mid S_k = m, S_{k-1} = m'\} = 0$。

所以，對於 $\Pr\{d_k = i \mid S_k = m, S_{k-1} = m'\} = 1$ 的分支，即

$$\gamma_k^i(m', m) = \Pr\{R_k \mid d_k = i, S_k = m, S_{k-1} = m'\} \Pr\{S_k = m \mid S_{k-1} = m'\}$$
$$= \Pr\{y_k^s \mid d_k = i, S_k = m, S_{k-1} = m'\} \Pr\{y_k^p \mid d_k = i, S_k = m, S_{k-1} = m'\} \Pr\{S_k = m \mid S_{k-1} = m'\}$$

(19)

其中因為 d_k 所編碼出的系統位元只和 d_k 本身有關，而與第 k 個時間點的狀態如何轉移無關，所以 $\Pr\{y_k^s \mid d_k = i, S_k = m, S_{k-1} = m'\} = \Pr\{y_k^s \mid d_k = i\}$ (20)

而(19)式變成

$$\gamma_k^i(m', m) = \Pr\{y_k^s \mid d_k = i\} \Pr\{y_k^p \mid d_k = i, S_k = m, S_{k-1} = m'\} \Pr\{S_k = m \mid S_{k-1} = m'\}$$
$$\equiv \Pr\{y_k^s \mid d_k = i\} \gamma_k^i(y_k^p, m', m)$$

(21)

其中 $\Pr\{S_k = m \mid S_{k-1} = m'\}$ 對應到前面所提到的事前資訊(a priori information)，而在解碼的開始，此項機率並沒有額外的資訊來做參考，所以 $\Pr\{d_k = 1\} = \Pr\{d_k = 0\} = 0.5$。

所以綜合以上的推導，(13)式又進一步的變成

$$LLR(d_k) = \ln \frac{\sum_m \sum_{m'} \alpha_{k-1}(m') \Pr\{y_k^s \mid d_k = 1\} \gamma_k^1(y_k^p, m', m) \beta_k(m)}{\sum_m \sum_{m'} \alpha_{k-1}(m') \Pr\{y_k^s \mid d_k = 0\} \gamma_k^0(y_k^p, m', m) \beta_k(m)}$$

$$= \ln \frac{\Pr\{y_k^s \mid d_k = 1\}}{\Pr\{y_k^s \mid d_k = 0\}} + \ln \frac{\sum_m \sum_{m'} \alpha_{k-1}(m')\gamma_k^1(y_k^p, m', m)\beta_k(m)}{\sum_m \sum_{m'} \alpha_{k-1}(m')\gamma_k^0(y_k^p, m', m)\beta_k(m)} \quad (22)$$

(22)式中有個很重要的觀念，我們可以把 $LLR(d_k)$ 分成三項 LLR 值的相加，即

$$LLR(d_k) = L_{systematic}(d_k) + L_{apriori}(d_k) + L_{extrinsic}(d_k) \quad (23)$$

(a) $L_{systematic}(d_k)$：系統資訊(systematic information)，其代表的是根據每個時間點所收到的系統位元它是傳送 1 或 0 的機率比值，

即 $\ln \frac{\Pr\{y_k^s \mid d_k = 1\}}{\Pr\{y_k^s \mid d_k = 0\}}$

(b) $L_{apriori}(d_k)$：事前資訊(a priori information)，其代表即 $\Pr\{S_k = m \mid S_{k-1} = m'\}$ 的機率，而此項值會代入另一個 MAP 解碼器所算出的 $L_{extrinsic}(d_k)$，一般來說，在解碼的開始，此項值被設定為零。

(c) $L_{extrinsic}(d_k)$：外部資訊(extrinsic information)，此項 LLR 值被用來當作另一個 MAP 解碼器的事前資訊。

最後，簡述一下 MAP 解碼器整個解碼的流程

STEP 1：根據(16)，(17)式初始化前向狀態路徑值以及後向狀態路徑值。

STEP 2：在每個時間點，對於所收到的 R_k，利用(21)算出分支路徑值，並利用(14)的遞迴關係式去計算前向狀態路徑值。

STEP 3：當所有的 R_k 已經都收到，便可利用(15)的遞迴關係式計算出後向狀態路徑值，並且可以根據(13)或(22)式去決定出每個位元的 LLR 值，

STEP 4：對於渦輪解碼而言，利用(23)式計算出 $L_{extrinsic}(d_k)$ 後，即

$$L_{extrinsic}(d_k) = LLR(d_k) - L_{systematic}(d_k) - L_{apriori}(d_k) \quad (25)$$

再把這項 LLR 值送進另一個 MAP 解碼器中作為 $L_{apriori}(d_k)$，另一個解碼器會再根據經過交錯器所得到的 R_k，利用 STEP1~STEP4，反覆地解碼。

STEP 5：在每次 MAP 解碼器解碼後，都可以得到對應於每個位元的 LLR 值，而此時我們可以對任意一組 MAP 解碼器所解出來的 LLR 值去根據(24)式做 Hard Decision，算出 d_k 的值。

$$d_k = \begin{cases} 1, LLR(d_k) > 0 \\ 0, LLR(d_k) < 0 \end{cases} \tag{24}$$

最後再將以上整個 MAP 演算法的運算與相關性畫成流程圖，如圖 5 所示。

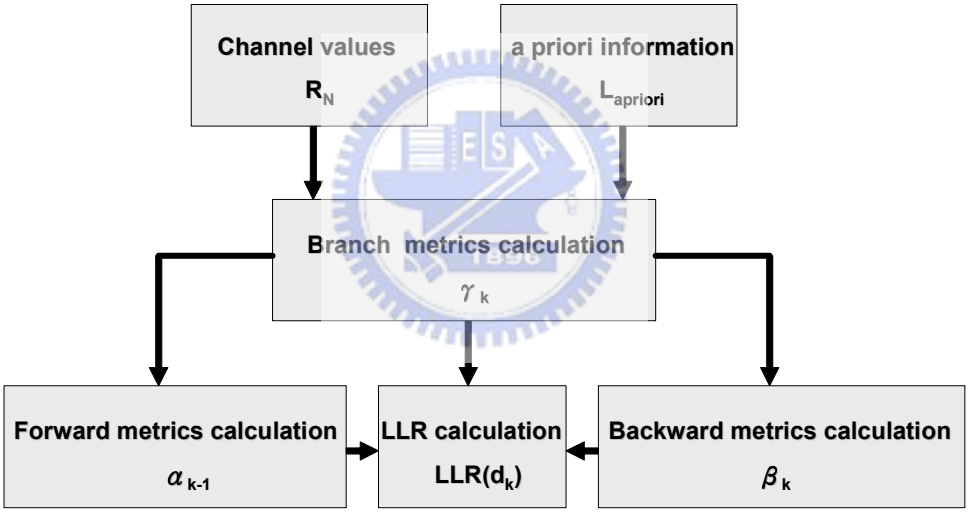

圖 5 MAP 演算法的運算流程圖

2-4 LOG-MAP 演算法

儘管 MAP 演算法有相當優異的且接近薛儂極限的錯誤更正能力，但是 MAP 的原始演算法需要使用到大量的指數，乘法，加法等運算，這對於硬體實現而言複雜度太高，所以有學者便試想是否能將原本使用乘法的 MAP 演算法，實現在對數

領域上，想辦法讓乘法運算轉換成加法運算，而在 1995 年由 Robertson, Villebrun, 以及 Hoehes 提出了 Log-MAP 演算法[4]。

Log-MAP 演算法是對原始的 MAP 演算法做複雜度的改善，而理論上計算的結果是一樣的。Log-MAP 主要精隨是先將前向狀態路徑值，後向狀態路徑值，以及分支路徑值都取對數運算，如下

$$\gamma'^{i}_{k}(m',m) = \ln \gamma^{i}_{k}(m',m)$$
$$= \ln \Pr\{y^{s}_{k} \mid d_k = i\} + \ln \Pr\{y^{p}_{k} \mid d_k = i, S_k = m, S_{k-1} = m'\} + \ln \Pr\{S_k = m \mid S_{k-1} = m'\} \quad (25)$$

$$\alpha_k'(m) = \ln \alpha_k(m) = \ln\{\sum_{m'=0}^{M-1} \alpha_{k-1}(m')\gamma_k(m',m)\}$$
$$= \ln\{\sum_{m'=0}^{M-1} \exp(\alpha'_{k-1}(m')) \exp(\gamma'_k(m',m))\} \quad (26)$$
$$= \ln\{\sum_{m'=0}^{M-1} \exp(\alpha'_{k-1}(m') + \gamma'_k(m',m))\}$$

而 $\alpha_k'(m)$ 的初始值設定如下，

$$\alpha_0'(m) = \begin{cases} 0, m = 0 \\ -\infty, m \neq 0 \end{cases} \quad (27)$$

$$\beta'_k(m) = \ln \beta_k(m) = \ln\{\sum_{m'=0}^{M-1} \beta_{k+1}(m')\gamma_{k+1}(m,m')\}$$
$$= \ln\{\sum_{m'=0}^{M-1} \exp(\beta'_{k+1}(m') + \gamma'_{k+1}(m,m'))\} \quad (28)$$

而 $\beta'_k(m)$ 的初始值設定如下，

$$\beta'_N(m) = \begin{cases} 0, m = 0 \\ -\infty, m \neq 0 \end{cases} \quad (29)$$

將(25)，(26)，(28)式代入(13)式中得到

$$LLR(d_k) = \ln \frac{\sum_m \sum_{m'} \alpha_{k-1}(m')\gamma_k^1(m,m')\beta_k(m)}{\sum_m \sum_{m'} \alpha_{k-1}(m')\gamma_k^0(m,m')\beta_k(m)}$$

$$= \ln\{\sum_m \sum_{m'} \alpha_{k-1}(m')\gamma_k^1(m,m')\beta_k(m)\} - \ln\{\sum_m \sum_{m'} \alpha_{k-1}(m')\gamma_k^0(m,m')\beta_k(m)\}$$

$$= \ln\{\sum_m \sum_{m'} \exp(\alpha'_{k-1}(m') + \gamma'^1_k(m,m') + \beta'_k(m))\} - \ln\{\sum_m \sum_{m'} \exp(\alpha'_{k-1}(m') + \gamma'^0_k(m,m') + \beta'_k(m))\}$$

（30）

從(30)式中，我們發現雖然原本(13)式中的乘法變成了加法運算，但是卻也多出了一項非線性的指數運算，所以到此為止，複雜度其實並沒有減少。然而，我們可以利用到一個關係式

$$\max^*(x,y) = \ln(e^x + e^y) = \max(x,y) + \ln(1 + e^{-|x-y|}) \quad （31）$$

且此關係式可以擴展成 3 個以上的變數，只要將兩兩變數做計算，如下所示

$$\max^*(x,y,z) = \ln(e^x + e^y + e^z) = \max^*(\max^*(x,y),z) \quad （32）$$

將 max*函數代入(26)和(28)中可以得到

$$\alpha_k'(m) = \ln\{\sum_{m'=0}^{M-1} \exp(\alpha'_{k-1}(m') + \gamma'_k(m',m))\}$$
$$= \max_{m'}^*(\alpha'_{k-1}(m') + \gamma'_k(m',m)) \quad （33）$$

$$\beta'_k(m) = \ln\{\sum_{m'=0}^{M-1} \exp(\beta'_{k+1}(m') + \gamma'_{k+1}(m,m'))\}$$
$$= \max_{m'}^*(\beta'_{k+1}(m') + \gamma'_{k+1}(m,m')) \quad （34）$$

再將 max*函數代入(30)得到

$$LLR(d_k) = \max_{m,m'}^*(\alpha'_{k-1}(m') + \gamma'^1_k(m,m') + \beta'_k(m))$$
$$- \max_{m,m'}^*(\alpha'_{k-1}(m') + \gamma'^0_k(m,m') + \beta'_k(m)) \quad （35）$$

從(33)~(35)可以看到，log-MAP 演算法所計算的前向狀態路徑值，後向狀態路徑值，以及 LLR(d_k)的運算已經變成了加法運算，以及兩個變數輸入的 max 函數的運算，和 $\ln(1+e^{-|x-y|})$ 的計算，而最後這項函數可使用查表法或是線性趨近的方法來計算，而此函數在|x-y|大於某個值時，$\ln(1+e^{-|x-y|})$ 便變得很小，所以也有種 MAP 演算法的變型稱為 log-max MAP 演算法，即是將 $\ln(1+e^{-|x-y|})$ 直接忽略掉，而 max*函數也就直接用 max 函數來取代，當然相對的必須付出約 0.3dB 的效能上的降低[4]。

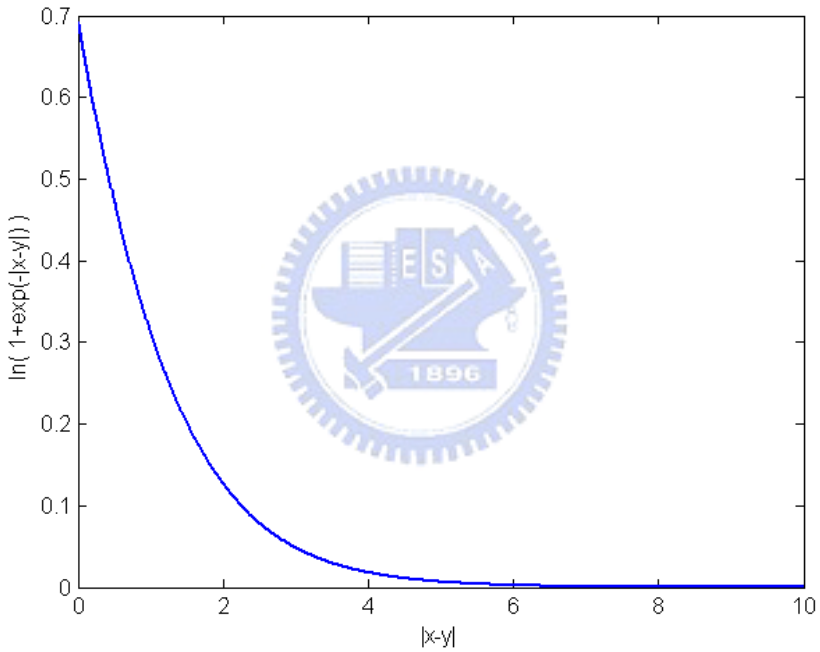

圖 6 max(x,y)與 max*(x,y)間的誤差

觀察圖 6 可以發現，在大於約 2.5 之後，誤差的值其實已經小於 0.05，而之後的誤差也是以指數的方式下降，而我們可以在誤差較大的部分，利用一條直線去趨近它，在這篇論文中，我便是以線性趨近的方法去計算 max*(x,y)，詳細方法以及其所造成的誤差如下所示

其中 TJIAN = 2.50681740420944

AJIAN = -0.2490416319543

$$\max\_linear(x,y) = \begin{cases} x & , \quad (x-y) > TJIAN \\ x + AJIAN \times (x-y-TJIAN) & , \quad 0<(x-y)<TJIAN \\ y - AJIAN \times (x-y+TJIAN) & , \quad -TJIAN<(x-y)<0 \\ y & , \quad (x-y) < -TJIAN \end{cases} \tag{36}$$

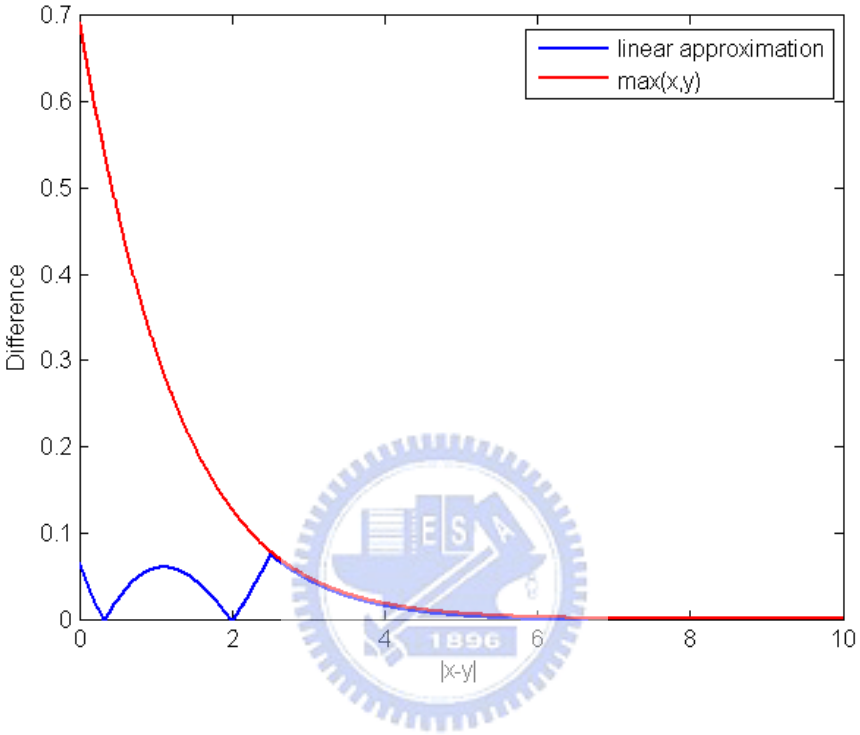

圖 7 線性趨近方法與忽略 $\ln(1+e^{-|x-y|})$ 的方法的誤差比較

圖 7 裡面表示我們利用線性趨近的方式可以使得原本直接忽略所造成的最大等於 0.7 的誤差降低至不管在任何情況下都可以小於 0.05。

2-4-1 渦輪解碼的終止技術(Termination Techniques)

在前面的章節已經討論過渦輪解碼的遞迴次數與錯誤率有相當大的關係，而如何去決定該用多少次的遞迴去解碼，對於傳統的作法而言，它會預設一個最大的遞迴次數，而不管解碼是否已經成功，一樣都得等到預設的遞迴次數達到了，才會停止解碼，這顯然是個很不實際的方法，因為在訊雜比較高時，實際上渦輪

解碼能夠很快地就解出正確的資料，於是多餘的遞迴次數只是多付出了額外的功率消耗。於是有大量的文獻研究如何去判斷渦輪解碼已經解出正確的資料，以便盡快的停止解碼，這些方法稱為終止技術(termination techniques)。

終止技術又分為兩大類，一類的方法試著利用渦輪解碼中能夠觀察到的資訊去判斷解碼的收斂性[5]，[6]，[7]。另一類方法在渦輪編碼之前加入額外的錯誤控制碼(error control coding)，例如循環冗餘校驗碼(Cyclic redundancy check)，並利用這些編碼去判斷每次遞迴後解碼結果的正確性。而在此篇論文中，即是利用後者的方法做為判斷解碼正確性的方法。而加入循環冗餘校驗碼的渦輪編碼器的架構圖如圖 8 所示

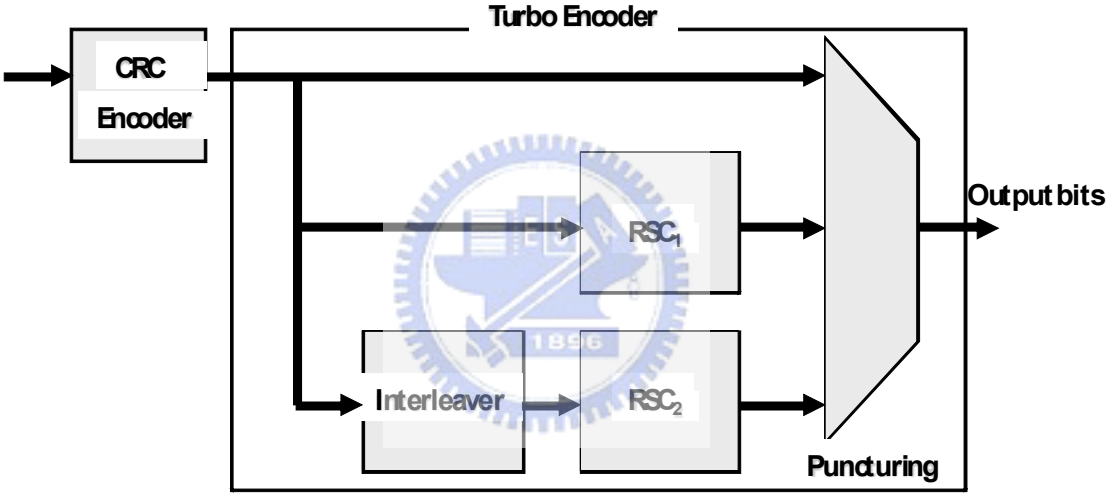

圖 8 Turbo-CRC 編碼器

而圖 9 是加入循環冗餘校驗碼渦輪解碼器的架構圖，主要差別在於多了”CRC Decoder”這塊硬體，它會在每次遞迴之後，先針對 $SISO_2$ 每個位元的 LLR 輸出做 hard decision(也可使用 $SISO_1$ 的 LLR 輸出，而在此篇論文中，選擇的是 SISO2 的輸出)，之後對於所加入的循環冗餘校驗碼做解碼，若通過了錯誤偵測，則停止解碼，並將 hard decision 的結果輸出。而實際上，循環冗餘校驗碼會有一定的機率會將錯誤的資料誤判成正確的資料[8]，只要當所加上去雜訊的位元剛好能夠整除循環冗餘校驗碼的生成多項式(generator polynomial)時，便會造成循環冗餘校驗碼解碼的錯誤，然而越長的循環冗餘校驗碼發生偵測失敗的機率就越低[20]。

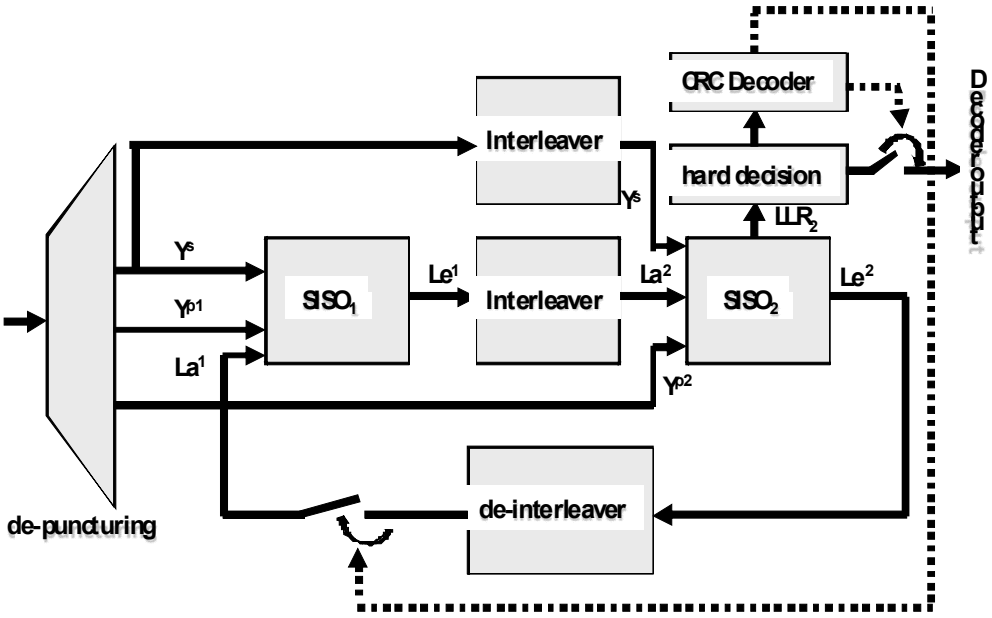

圖 9 Turbo-CRC 解碼器

2-5 渦輪碼在 IEEE 802.16 上的應用

在本小節中,我們將描述制訂在 IEEE 802.16 中的迴旋渦輪碼(Convolutional Turbo Code)的規格，以及如何產生子封包(subpacket)，而在此篇論文中，所選用來研究的渦輪碼系統即是 IEEE 802.16 中所制訂的規格，而此種規格可以被用來支援 HARQ(Hybrid Automatic Repeat Request)傳輸，而在這裡我們先就 HARQ 做個簡單的介紹。

2-5-1 混合式自動重傳請求(Hybrid Automatic Repeat Request)

HARQ 是種結合自動重傳請求(Automatic Repeat-reQuest，ARQ)以及前向錯誤更正碼(Forward error correction,FEC)的通訊機制[9]，在只有 ARQ 的系統中，我們只要對傳送的資料加入錯誤偵測碼，接收端靠著這些碼偵測是否有錯誤而決定重新傳送與否，其中錯誤偵測碼通常都不長，碼率(coding rate)能夠維

持的很高，所以此系統在訊雜比很高的通道下，能有很高的吞吐量(thoughput)，然而在通道很吵雜的情況下，當資料發生錯誤時只能要求重送，造成吞吐量大大的降低。在另一方面，只有 FEC 的系統中，當訊雜比較低時，錯誤更正碼能夠盡可能的更正錯誤，讓整體的錯誤率降低，然而在訊雜比較高時，資料可能只有少部分發生錯誤甚至沒有錯誤，而在這樣的情況下，加入太長的錯誤更正碼很明顯的是對頻寬的浪費，所以 HARQ 便是希望能夠適當的結合這種系統的優點的一種機制，除此之外，在 HARQ 系統中，我們會希望解碼失敗的資料並不直接或完全被丟棄，而是能夠結合重傳之後的封包，讓錯誤率降低或是吞吐量增加，而 HARQ 到目前為止大致上分成三大類：

TYPE I HARQ：只是單純地結合 FEC 與 ARQ 兩種機制，在每次的重送中，都同時送出了錯誤偵測碼和全部的錯誤更正碼，而對於各次重傳所收到的封包，能夠使用一些封包結合(packet combining)的方法，例如 Chase 所提出的 Code Combining[10]，它能夠非常有效的提升效能。

TYPE II HARQ：對於 FEC 的部分，並不在一開始就送出所有的錯誤更正碼，而是隨著重傳的次數而慢慢的釋出更多的冗餘(redundancy)位元，讓錯誤更正能力慢慢提升，如此一來便能適應通道狀態的變化，此種方法稱為 Incremental redundancy [11][12][13]，而 IEEE 802.16 迴旋渦輪碼所支援的 HARQ 傳輸便是屬於這種形式。

TYPE III HARQ：一般來說，TYPE II HARQ 每次所傳送的子封包，單獨是無法解碼的，必須要結合其他筆資料才有辦法解碼，而對於 TYPE III HARQ 而言，主要特點便是其所傳送的每一筆子封包都有能夠個別解碼的能力，也就增加了解碼成功的機會[14]。

2-5-2 IEEE 802.16 中渦輪編碼的規格

在 IEEE 802.16 的迴旋渦輪編碼器中，使用雙二位元循環遞迴系統迴旋碼(duo binary Circular Recursive Systematic Convolutional code)作為其組成碼(constituent code)，如圖 10 所示，其中 A,B 代表資料輸入的位元向量，而輸入編碼器的資料區塊假設為 k 個位元也就是 N 對位元，則 $8 \le N/4 \le 1024$，而以下的式子定義此編碼器的電路連結(使用十六進位表示法及多項式表示)

迴授連結： 0xB， $1+D+D^3$

平等位元 Y 輸出： 0xD， $1+D^2+D^3$

平等位元 W 輸出： 0x9， $1+D^3$

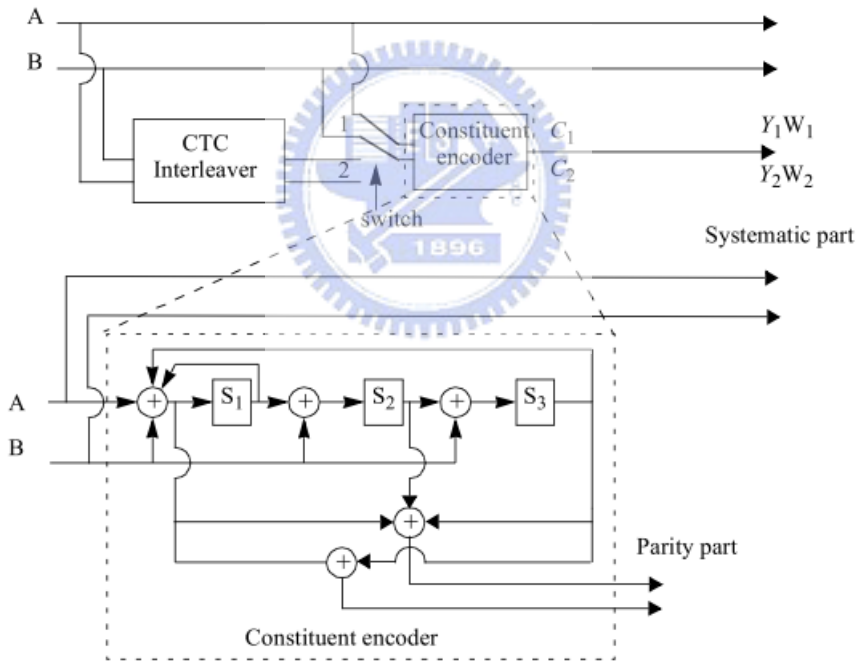

圖 10 迴旋渦輪編碼器(參考至[15])

在此渦輪編碼器中，使用 Tail-Biting 的技術[16][17][18]，其與傳統的渦輪編碼器將編碼的初始狀態與終止狀態設定為零狀態有所不同，在傳統的作法中，由於必須讓編碼的終止狀態回到零狀態，所以還必須送出一段稱為終結位元(termination bits)的多餘資料，而我們試著想省掉這些位元所造成在碼率上面

的損失，而 Tail-Biting 便是其中一種方法，在這種編碼器中，我們設定編碼的初始狀態與終止狀態為同樣的狀態，稱之為循環狀態(circulation states)，而如何決定此狀態必須根據所要編碼的資料而有所改變，在 IEEE 802.16 中，它提供了一種查表的方法讓我們決定編碼的初始狀態，步驟如下

STEP 1: 將編碼器初始狀態設定為零狀態，開始輸入欲編碼的資料編碼，假設最後的終止狀態為 $S0_{N-1}$，利用圖 11 的查找表(lookup table)找出所對應的初始狀態。(對於兩個組成編碼器皆須做以上的步驟，因為經過交錯器後的資料在編碼過程中的狀態轉變是和原本是完全不一樣的。)

STEP 2：將編碼器初始狀態設定為 STEP 1 中所查到的狀態後，再將資料重新編碼一次，而最後的終止狀態便會和初始狀態相同。

N_{mod_7}	$S0_{N-1}$							
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	0	6	4	2	7	1	3	5
2	0	3	7	4	5	6	2	1
3	0	5	3	6	2	7	1	4
4	0	4	1	5	6	2	7	3
5	0	2	5	7	1	3	4	6
6	0	7	6	1	3	4	5	2

圖 11 循環狀態查找表(參考至[15])

以下討論子封包的產生(subpacket generation)的相關技術，在資料經過碼率=1/3 的迴旋渦輪編碼後，之後會經過通道交錯器(channel interleaver)將資料打亂，最後在依照所需要的碼率做 puncturing 的動作，而這些一系列的動作

的流程圖如圖 12 所示，其中通道交錯器的動作又細分為三個步驟 symbol separation，subblock interleaving，以及 symbol grouping，而 puncturing 的動作在這裡又稱為 symbol selection。最後會產生一個或數個子封包，而這裡所產生的子封包可以使用在 HARQ 傳輸中，下面針對各個功能做介紹。

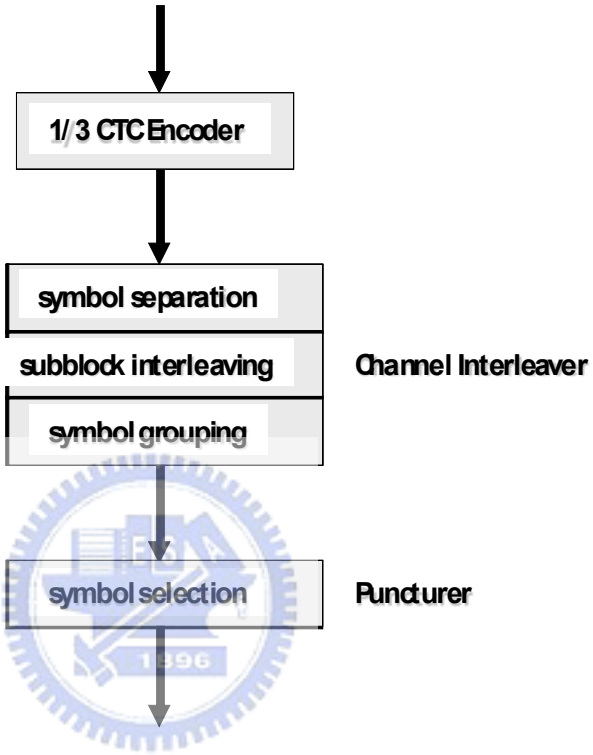

圖 12 產生子封包的流程圖(參考至[15])

通道交錯器(Channel Interleaver)的各個步驟的動作：

(a) Symbol separation：這步驟將碼率為 1/3 的迴旋渦輪編碼器的所有輸出分成六個子區塊 A, B, Y_1, Y_2, W_1, W_2 。

(b) Subblock interleaving：在各個子區塊中，各自做內部的交錯排列。

(c) Symbol grouping：最後再將所有區塊經過交錯器打亂後的資料收集起來，其中 Y_1, Y_2 與 W_1, W_2 會再經過交替的排列，整個通道交錯器的流程圖如圖 13 所示。

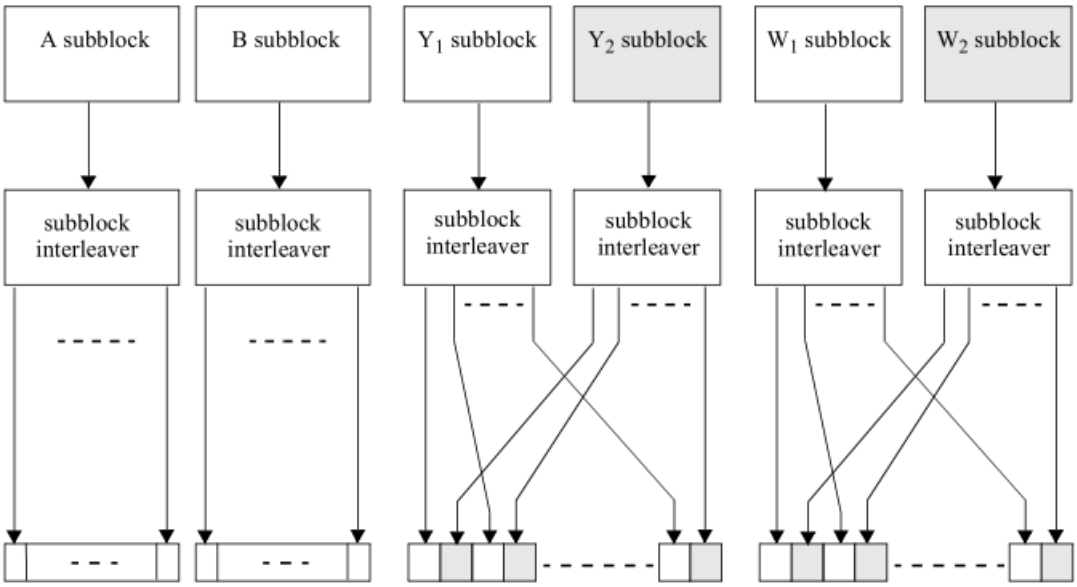

圖 13 通道交錯器的架構圖(參考至[15])

最後，執行 symbol selection 以產生所需要的子封包，symbol selection 可以從通道交錯器所產生的序列中任意的地方開始選取一段連續的位元，而對於第一次傳輸，必須從第一個位元開始選取，也就是必須包含系統位元(systematic bits)的部分，而當沒有使用 HARQ 傳輸時，第一次傳輸所產生的子封包就是輸出的 codeword。而子封包的長度決定於所需要的碼率，而碼率的選擇又決定於通道的狀態，下面將說明子封包的位元是如何被選擇的，我們先定義幾個參數：

k：子封包的索引。第一次傳輸時，k=0，每下一個子封包，k 的值會被加一。而當沒有使用 HARQ 傳輸時，k=0。

N_{EP}：進入迴旋渦輪編碼器的資料區塊的位元數。

N_{SCH}：子通道(subchannel)的個數。

m_k：第 k 個子封包傳輸時所使用的調變的階數。

（當使用 QPSK 時，m_k=2。當使用 16-QAM 時，m_k=4。當使用 64-QAM 時，m_k=6）

$SPID_k$：第 k 個子封包 subpacket ID（第一個子封包的 $SPID_k$ = 0）

我們將通道編碼器輸出的所有位元從第一個到最後一個編號為 0~(N-1)號，而第 k 個子封包的第 i 個位元的編號可由下面的的公式得到：

$$S_{k,i} = (F_k + i) \bmod (3 \times N_{EP}) \qquad (37)$$

其中 $i = 0,1,...,L_K - 1$

$$L_K = 48 \cdot N_{SCHk} \cdot m_k$$

$$F_k = (SPID_k \cdot L_k) \bmod (3 \cdot N_{EP})$$

第一次傳輸的子封包包含系統位元的部分，因此當不使用 HARQ 傳輸時，這個子封包便可作為傳輸的 codeword。到此為止，子封包已經完整的被產生出來。在這篇論文中，我們並未使用 HARQ 傳輸方式去討論研究的主題，然而 HARQ 傳輸所能夠提供的優異的性能是非常值得探討的。

第三章 渦輪解碼中的預棄與狀態再利用技術

3-1 研究動機

根據[21][22][23]等相關文獻的模擬，渦輪解碼解出正確的資料所需要的遞迴次數會隨著通道的訊雜比的減少而增加，而在非常吵雜的通道下，所須的遞迴次數幾乎等於所預設的最大遞迴次數，但在此情況下(如圖 14 中 Eb/No 在 0~0.6dB 時，解碼次數幾乎等於 10 次)，大部分的封包最後解碼的結果卻是失敗的。這個方法很顯然並不實際，所以針對這點讓我們有了個想法，既然最後解碼的結果是失敗的，那何不即早放棄解碼，以節省無意義的解碼遞迴次數。而在[25]的論文中他提到了一種預先放棄解碼的方法簡稱為預棄技術(Early Give-up technique)，然而在模擬分析之後發現，這方法在資料區塊較短時，錯誤率會漸漸上升，而在此篇論文中，主要便是探討如何不增加太多的遞迴次數的前提下，能夠降低預棄技術的錯誤率。

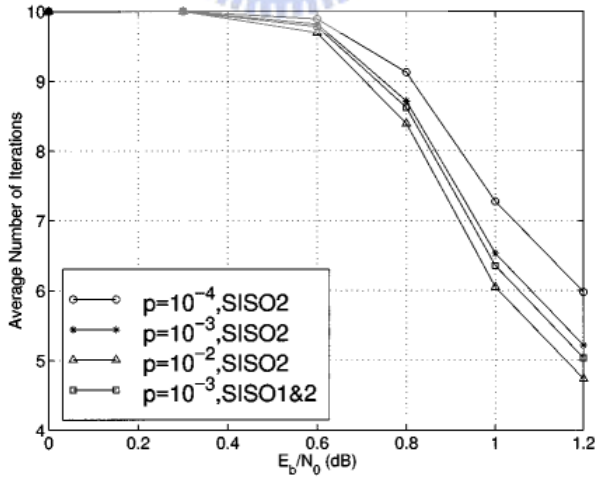

圖 14 訊雜比與渦輪解碼遞迴次數的關係(參考至[21])

預棄技術的觀念是基於觀察渦輪解碼中的資訊，當觀察到解碼似乎有解碼失敗的趨勢時，便立即停止解碼，如此一來，所省下的遞迴次數等效的能夠節省多

餘的功率消耗，更可以縮短整體解碼的時間延遲，而能提早要求傳送端重傳一筆新的封包，提升通訊服務的品質(Quality of Service,QoS)。

預棄技術能夠省下可能會解碼失敗的封包所浪費掉的遞迴次數，而另一方面，當在收到一筆重傳的相同封包時，其實我們可以利用上一筆解碼失敗的資料解碼後所留下來的資訊做為新的一筆封包渦輪解碼時的初始值[22]，而這個方法我們稱之為狀態再利用技術(State Reuse technique)，而在結合[22]中的方法至預棄技術的過程中，需要做一些小小的修正，以讓錯誤率方面有更好的效能，而模擬證明，這方法能讓渦輪解碼遞迴次數有效的減少，甚至能讓錯誤率也有所改善。

3-2 預棄技術的概念與模擬結果

前面有提到預棄技術是基於觀察封包是否有解碼失敗的趨勢，決定是否要放棄解碼，而根據[12]中的論述，我們發現對於大部分解碼能夠成功的封包而言，其在渦輪解碼時，隨著每次遞迴次數的增加，LLR 值的期望值會單調地遞增，而從此觀點出發，我們似乎可以將”LLR 值的期望值是否單調遞增”做為判斷封包能否成功解碼的條件。圖 15 與圖 16 是在[12]中，它分析大量的封包解碼的情形後，根據各個封包在每次遞迴後所剩下來的錯誤位元數以及解碼的情形分成五大類：

(a) F1：error-free，fast convergence

解碼成功，且在解碼時很快就更正了所有的錯誤。

(b) F2：error-free，slow convergence

解碼成功，但在解碼時更正錯誤的速度較慢，花了較多的遞迴次數。

(c) F3：few errors

解碼失敗，最後留下了少數的錯誤位元數。

(d) F4：many errors

解碼失敗，最後還存在著許多錯誤位元數。

(d) F5：oscillating errors

錯誤位元數在每次遞迴中不停的震盪。

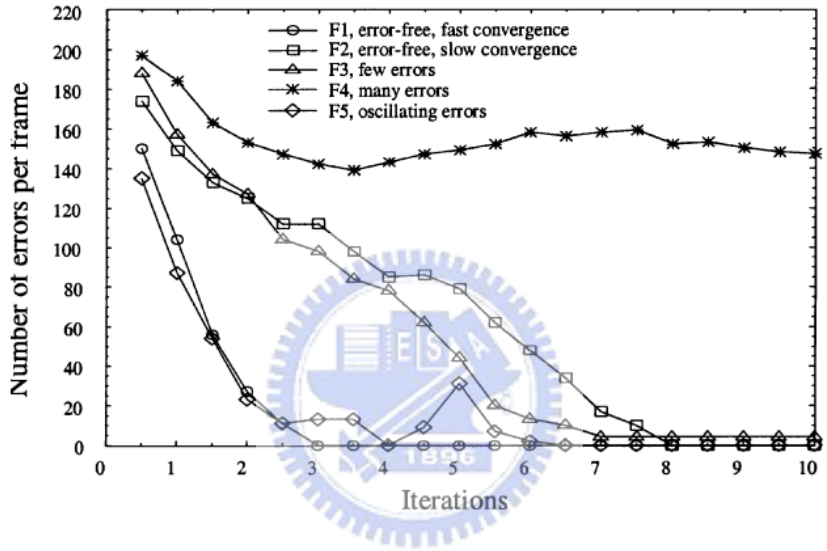

圖 15 每次遞迴後所剩下的錯誤位元數(參考自[12])

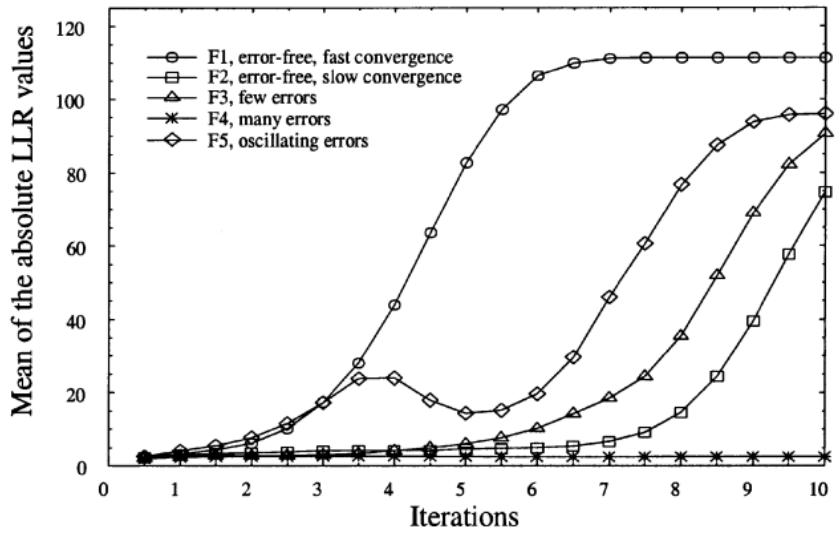

圖 16 每次遞迴後 LLR 的絕對值的平均值(參考自[12])

由圖 16 中看到對於最後解碼失敗且錯誤數目很多的這種封包(F4)，LLR 的絕對值的平均值是在每次的遞迴間是不停的來回震盪的，也就是說無法單調遞增，所以我們以及[25]的預棄技術便以此做為放棄解碼的指標的基礎。然而若單純以此準則來做判斷條件，可能將部分沒有單調遞增但最後能夠解碼成功的封包誤判，造成錯誤率的上升，而這些現象我們會在後面慢慢作分析以及改善。

而根據[12]這篇論文中，它有提到一些理論觀點來證明為什麼 LLR 的絕對值的平均值會有這樣的趨勢。假設所要傳送的資料 $x \in \{-1,+1\}$ 的可能性相同的(equally likely)，另外假設介於渦輪編碼器，實體通道，渦輪解碼器間的 metachannel 的通道模型為加成性白色高斯雜訊(AWGN)的通道，且其期望值為零(zero mean)，變異數為 σ_m^2，接收到的值為 z。

$$LLR(x) = \ln \frac{\Pr\{x=+1 \mid z\}}{\Pr\{x=-1 \mid z\}} = \ln \frac{f(z \mid x=+1)}{f(z \mid x=-1)} = \frac{2}{\sigma_m^2} z \qquad (38)$$

其中，$f(\cdot|\cdot)$ 表示條件機率密度函數(conditional probability density function)。對(38)式的左右各取絕對值與期望值的計算變成，

$$E[|LLR(x)|] = \frac{2}{\sigma_m^2} E[|z|] \qquad (39)$$

而透過遞迴次數增加，渦輪解碼器會試圖減少 σ_m^2，所以當 $\sigma_m^2 \downarrow$ 則 $E[|LLR(x)|] \uparrow$。

3-2-1 編碼資料區塊大小的影響

由上一小節所得到的想法，[25]便是將停止解碼的條件設定為”若 LLR 的絕對值的平均值未單調遞增則停止解碼”，換句話說，我們會在每次解碼的遞迴後，便去計算 LLR 的絕對值的平均值，即

$$E[|LLR(x)|] = \frac{1}{N} \sum_{i=1}^{N} |LLR(x)| \qquad (40)$$

若發現這次遞迴後的 $E[|LLR(x)|]$ 小於上次遞迴的值，那便立即停止解碼。而以下是不同的資料區塊大小以及不同的碼率所得的錯誤率及平均每一筆封包解碼出正確的資料時所需遞迴次數。而以下模擬圖中，Magic Genie 的方法[7]指的是在每次遞迴後所解出來的資料，在模擬階段，我們可以和真的所送出的資料作比較，若已經相同，則馬上停止解碼，而在解碼失敗時，遞迴次數便等於最大的解碼遞迴次數，所以這方法所得到的遞迴次數等於是解出正確的一筆資料所需要的最小遞迴次數。

在圖 17~28 中，不論是位元錯誤率或是平均遞迴次數的模擬圖裡，我將資料區塊大小(Block size)標示於圖形的上方，碼率(Code rate)也標示在圖形的上方。

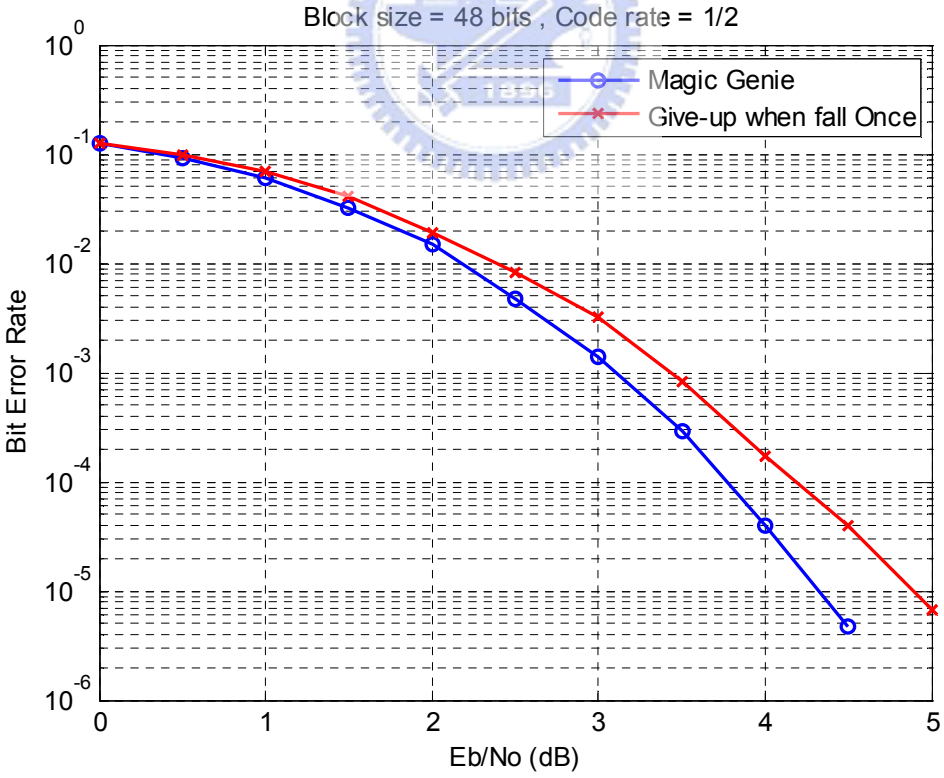

圖 17 位元錯誤率

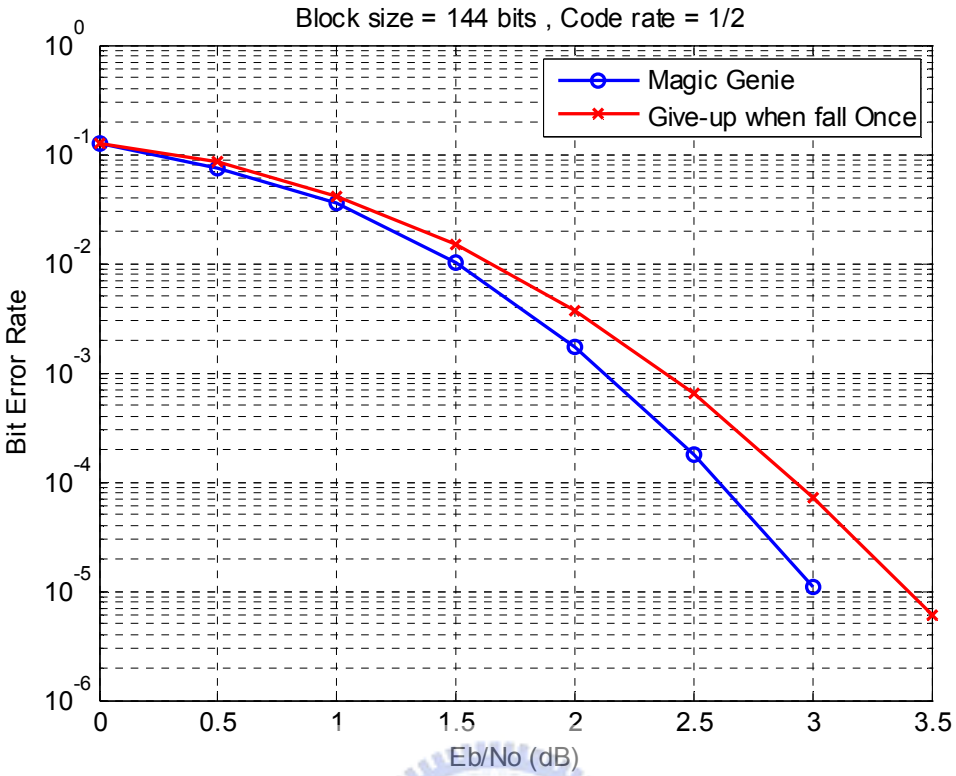

圖 18 位元錯誤率

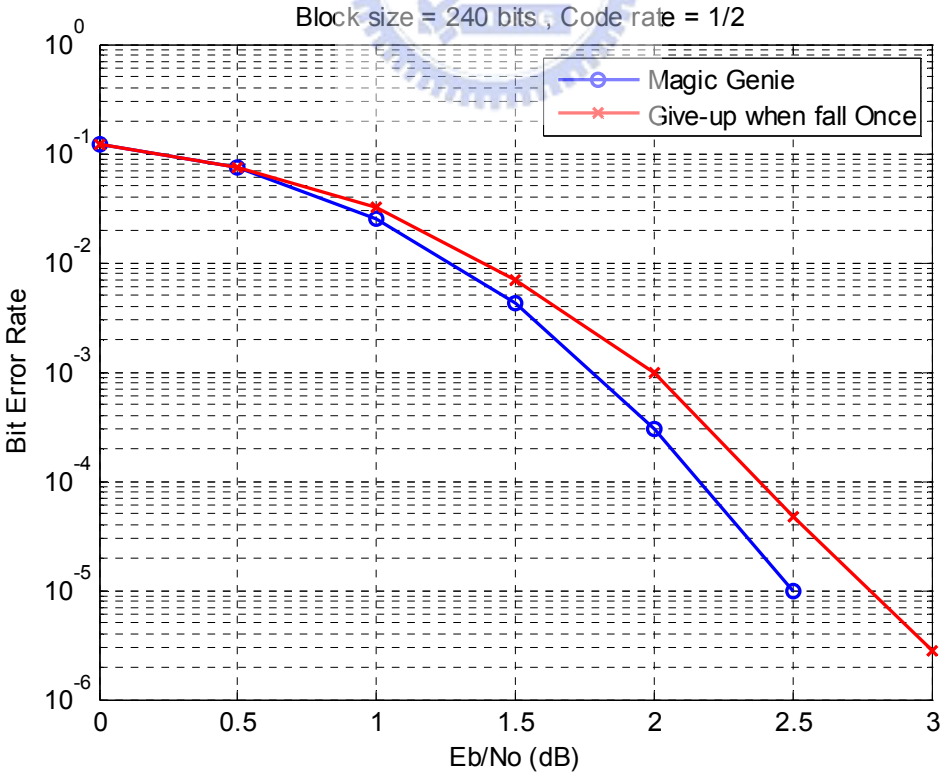

圖 19 位元錯誤率

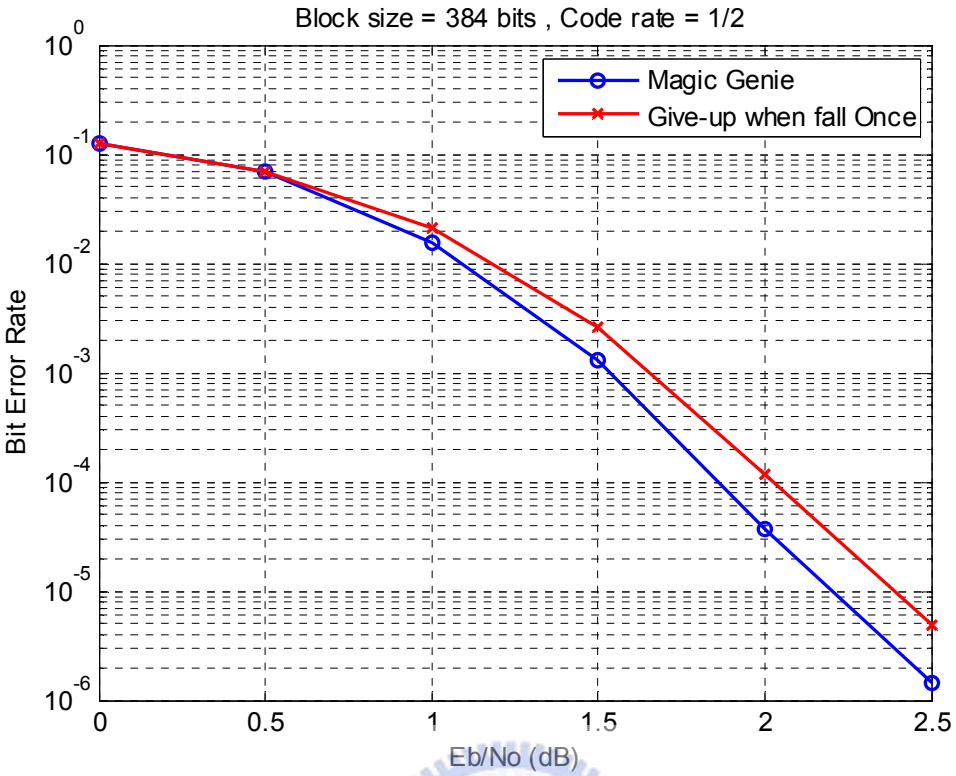

圖 20 位元錯誤率

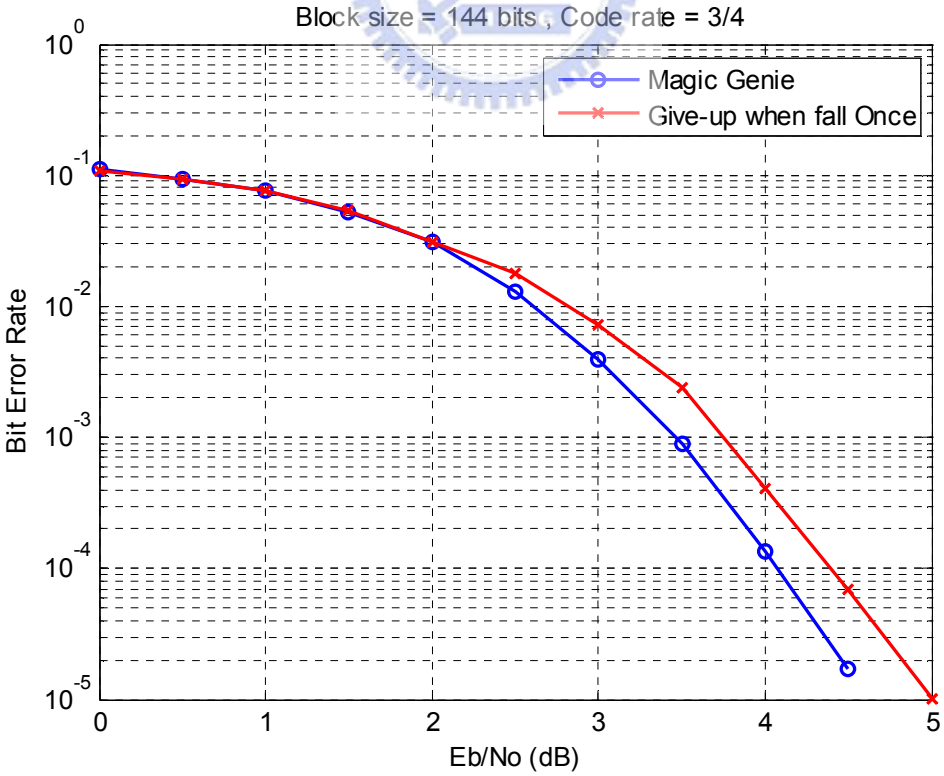

圖 21 位元錯誤率

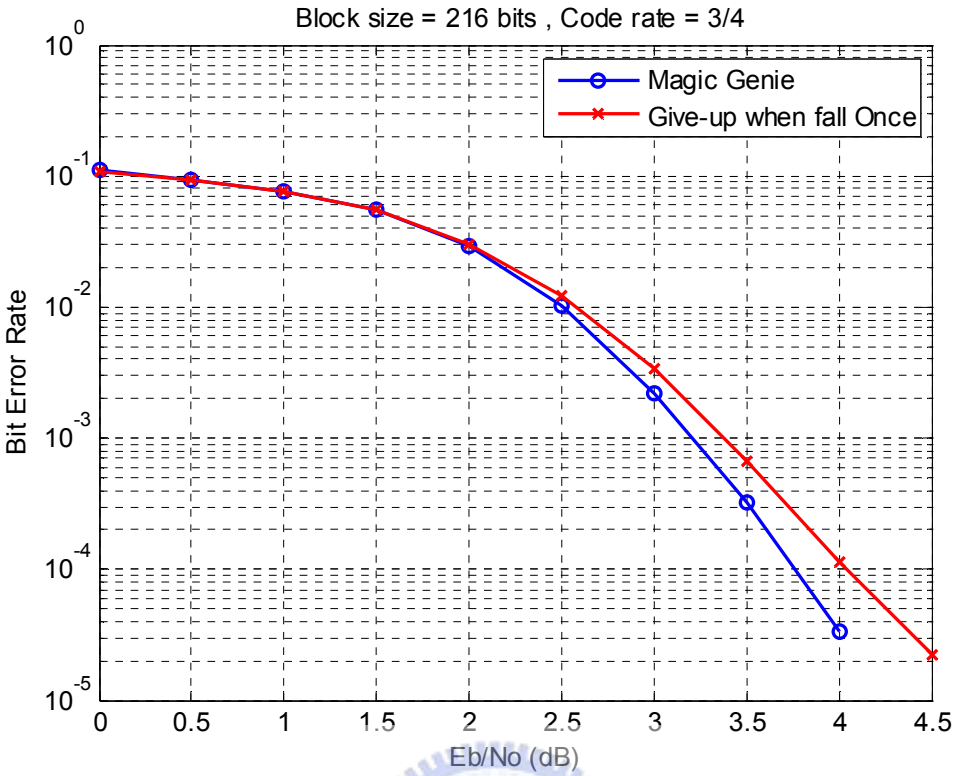

圖 22 位元錯誤率

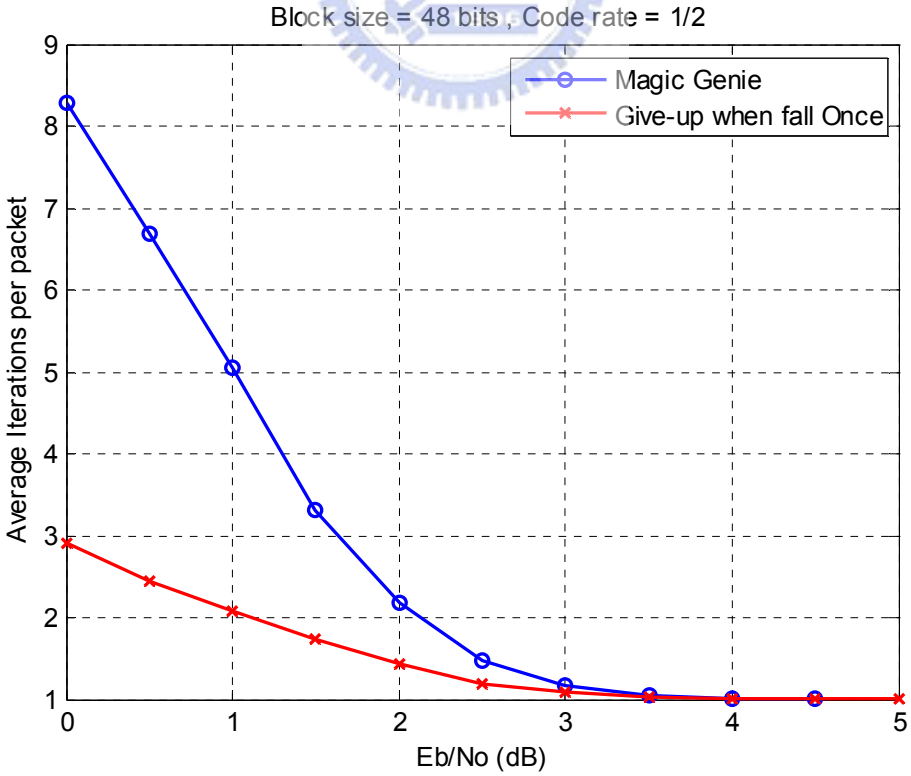

圖 23 平均遞迴次數

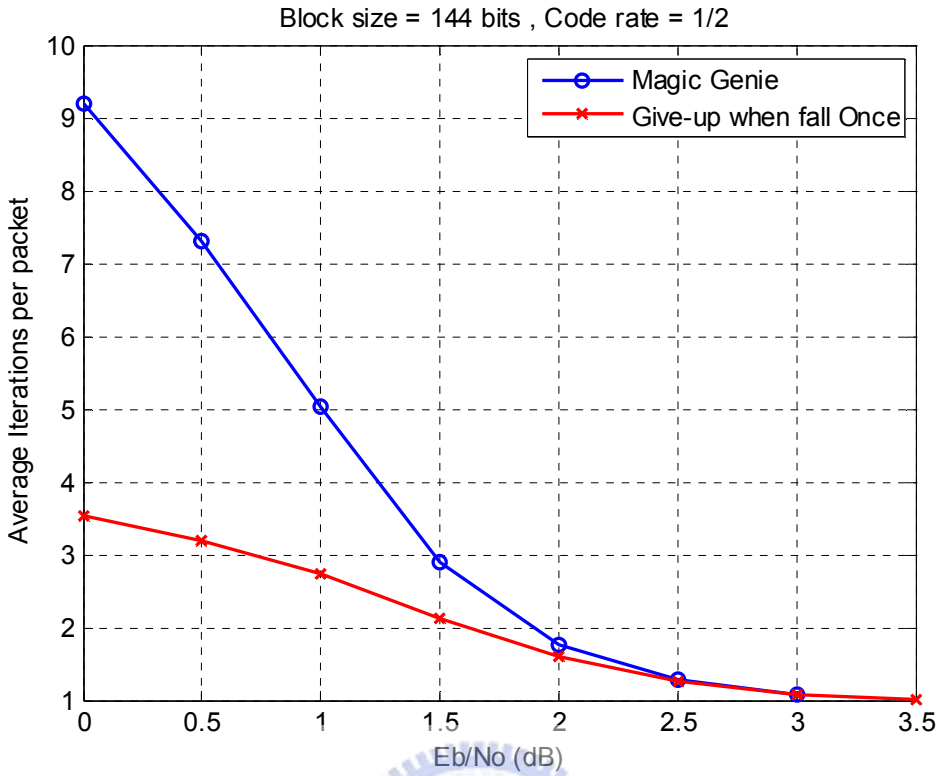

圖 24 平均遞迴次數

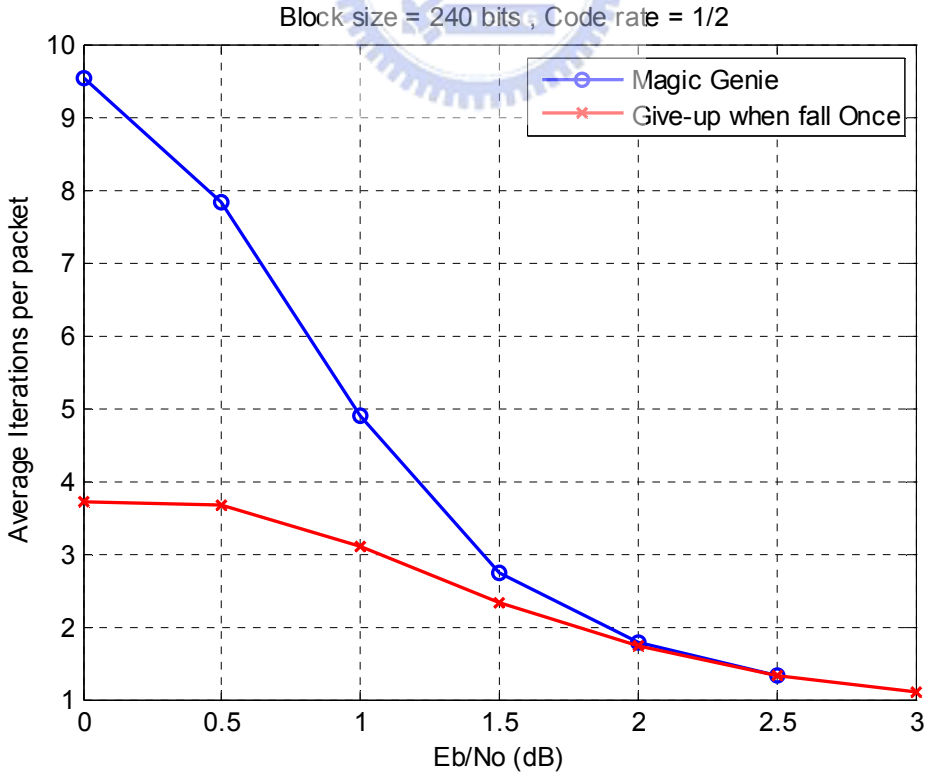

圖 25 平均遞迴次數

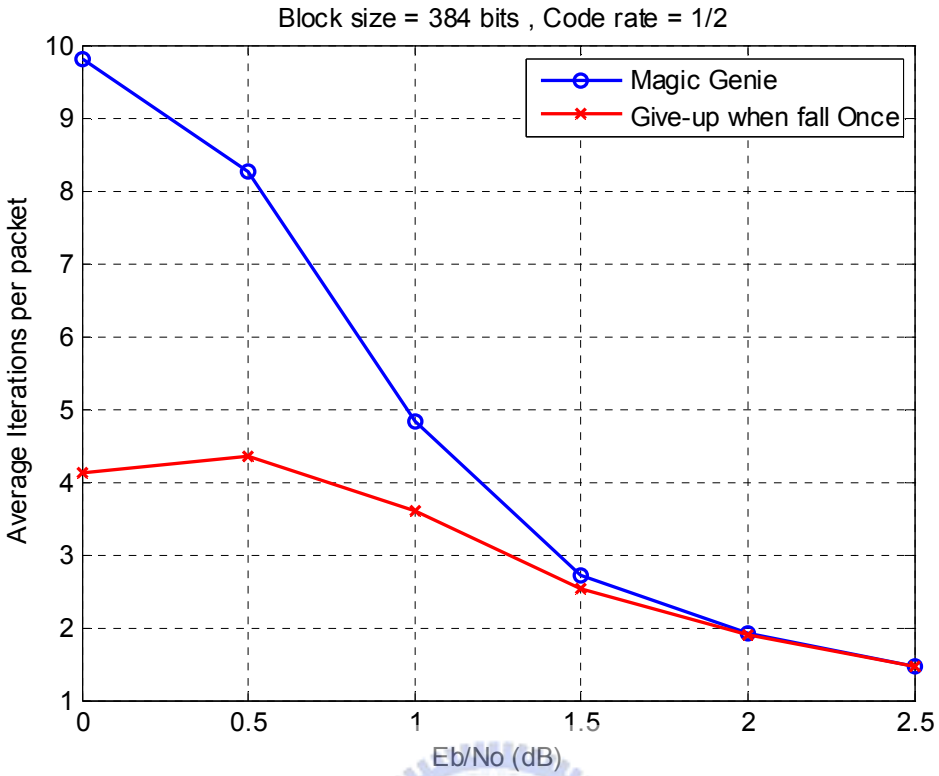

圖 26 平均遞迴次數

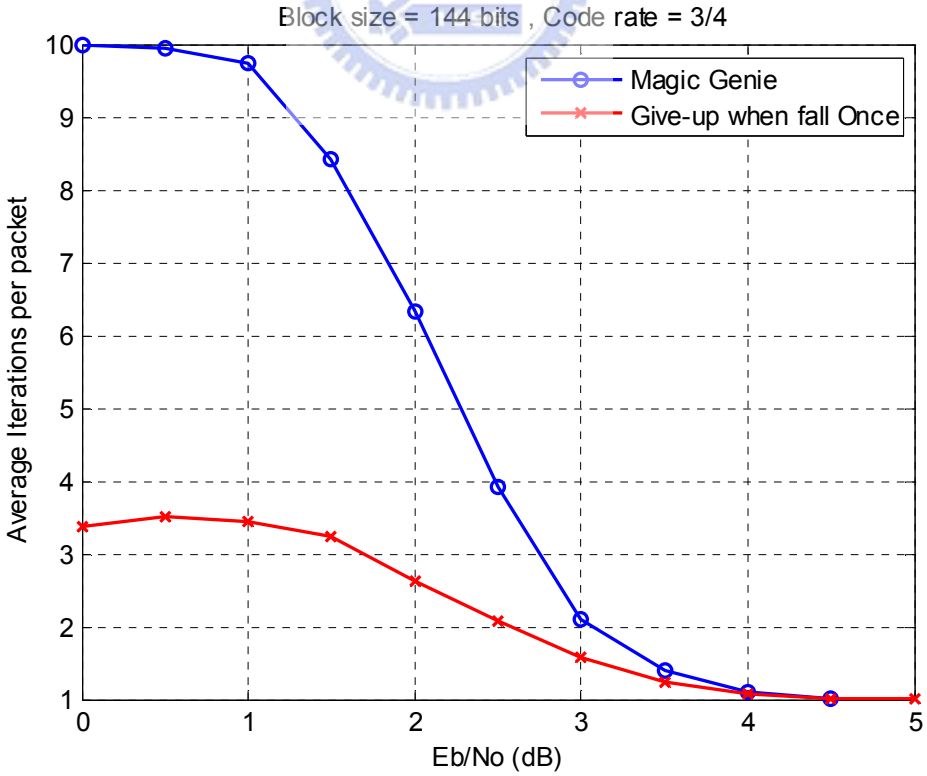

圖 27 平均遞迴次數

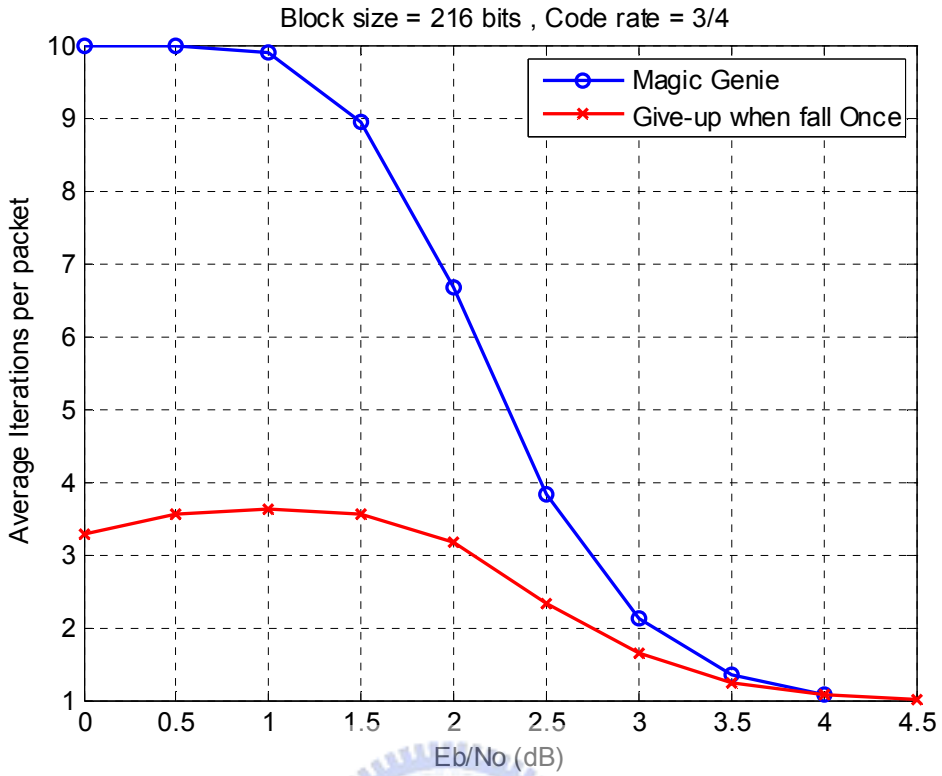

圖 28 平均遞迴次數

由圖 23~26(碼率皆是 0.5)的平均遞迴次數的模擬圖來看，使用預棄的方法在約 Eb/No 等於 0~1dB 時，相較於 magic genie 的方法約可省下 60%左右的遞迴次數，而在 Eb/No 等於 1.5dB 時，隨著資料區塊大小升高，解碼的能力也上升，解碼會失敗的封包便越來越少，所以在這個訊雜比下，圖 23(區塊大小=48 bits)，還可以省下 57%左右，而圖 24(區塊大小=144 bits)剩下 33%左右，而圖 25(區塊大小=240 bits)只剩下 16%左右，圖 26(區塊大小=384 bits)只剩下不過 5%。而當區塊大小變大，解碼能力上升，所以會在 Eb/No 越小的值時，magic genie 和預棄的遞迴次數就貼在一起了，因為已經沒有會失敗的封包可以被放棄，例如圖 23 在 3dB 之後預棄技術沒辦法省下什麼遞迴次數，圖 24，圖 25 則是在 2dB 之後，圖 26 則比 2dB 還小一點。而錯誤率方面，圖 17(區塊大小=48 bits)在 BER=10^{-5} 的情況下，和 magic genie 的差異約 0.5dB，而隨著區塊大小的上升，圖 18(區塊大小=144 bits)在 BER=10^{-5} 下，差異約 0.4dB，圖 19(區塊大小=240 bits)

在 BER=10^{-5}下，差異約 0.25dB，圖 20(區塊大小=384 bits)在 BER=10^{-5}下，差異約 0.2dB，也就是說，隨著區塊大小的增加，我們可以利用這個機制來估算封包會解碼失敗的準確度越來越高，所以和 magic genie 間的差距也就越來越小，這有可能是因為當計算$|LLR(x)|$的平均值時所使用的樣本數越來越多，所得到的平均值便能越接近理論上的期望值$E[|LLR(x)|]$，所以我們推論在使用較長的資料區塊大小的渦輪碼時，利用剛剛簡單的預棄判斷條件在錯誤率上面的效能損失會越來越小，所以我們將研究重心放在資料區塊大小較小的部分。

對於圖 27~28(碼率皆等於 3/4)而言，圖 27(區塊大小=144 bits)的平均遞迴次數在 0~3dB 間約可省下 65%~25%，圖 28(區塊大小=216 bits)的平均遞迴次數在 0~3dB 間也約可省下 65%~25%。而錯誤率方面，圖 27 和圖 28 在 BER=10^{-4}的情況下，錯誤率約比 magic genie 高了 0.2dB 左右。

所以使用以上的預棄技術的方法，在 BER=10^{-5}時，最多會有 0.5dB 的損失，在高訊雜比時，不但遞迴次數沒有任何優勢，錯誤率甚至還升高了，所以現在必須針對如何降低誤判的機率這點想辦法作改進。

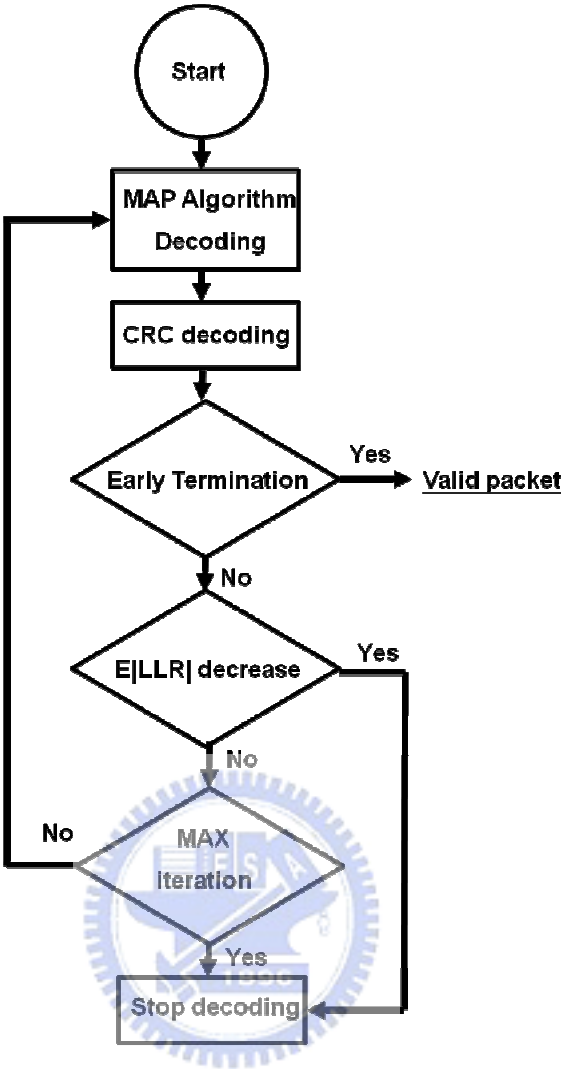

圖 29 以非單調遞增為預棄條件的解碼流程圖

而我們將[25]的預棄技術的整個解碼過程化成流程圖來解釋，如圖 29 所示，一開始我們接收到資料後，便開始使用 MAP 解碼器做解碼，在每次解碼遞迴後，便將 $SISO_2$ 所得到的 LLR 值做 Hard decision 以得到所有的二進位數值，而再對這些位元裡面所加入的 CRC 碼做錯誤偵測檢查，若發現 CRC 已經正確，便視為解碼已經成功，立即停止解碼。而若發現 CRC 解碼未正確，表示資料可能還有錯誤，則再判斷 E|LLR|是否比上次遞迴的數值來的小，如果是則便猜測這是一筆解碼無法成功的封包，便立即停止解碼。若 E|LLR|未減少，則繼續判斷是否已經到達最大的解碼次數，若是則停止解碼，否則繼續回到 MAP 解碼器解碼。

3-2-2 如何降低預棄技術的誤判率

上一小節中，我們知道[25]將解碼過程中 $E[|LLR(x)|]$ 未單調遞增的封包放棄解碼，然而模擬結果發現，在資料區塊大小較小的封包在訊雜比較高的情況下，容易發生誤判的事件，造成本來能夠解碼成功的封包卻在解碼成功前被強迫停止，使得整體錯誤率的上升。

而要如何盡量避免誤判發生這種解碼情況的封包，於是我們有了個想法：若能夠解碼成功的封包絕大部分會有單調遞增的情形，換句話說，這些封包的 $E[|LLR(x)|]$ 應該會隨著遞迴次數的增加而朝向較高的值發展，而這種間若不小心值減少了，我們是不是可以在給它機會做解碼，期望它能夠在未來的遞迴中能夠解碼成功。所以我們將預棄的條件修改成以下的方法，首先我們會先定義一個最大值的參數稱為 MAX_FALL，這個值代表的是在解碼的遞迴間，我們能容許 $E[|LLR(x)|]$ 的值下降幾次，即 $E[|LLR(x)|]$ 的值沒有大於前一次遞迴的值的次數，當下降的次數已經等於 MAX_FALL 次，或是已經到達預先定義的最大解碼次數，則便立即停止解碼，整體詳細的解碼流程圖繪製在圖 30 中。

在此說明一下在圖 29 及圖 30 裡，為何將 CRC 解碼以及判斷提早終結(Early Termination)的條件判斷放在預棄條件判斷的前面的是因為這能夠避免掉若這筆資料其實已經沒有錯誤，但是卻湊巧在此次遞迴後，$E[|LLR(x)|]$ 剛好減少且滿足了預棄的條件，而被誤判的機會，除非這筆資料的錯誤剛好是 CRC 解碼所無法辨認的，但是這樣的機率相當小，所以這樣的解碼流程安排，理論上能比將預棄條件判斷擺在提早終結條件判斷的前面有較低的錯誤率。

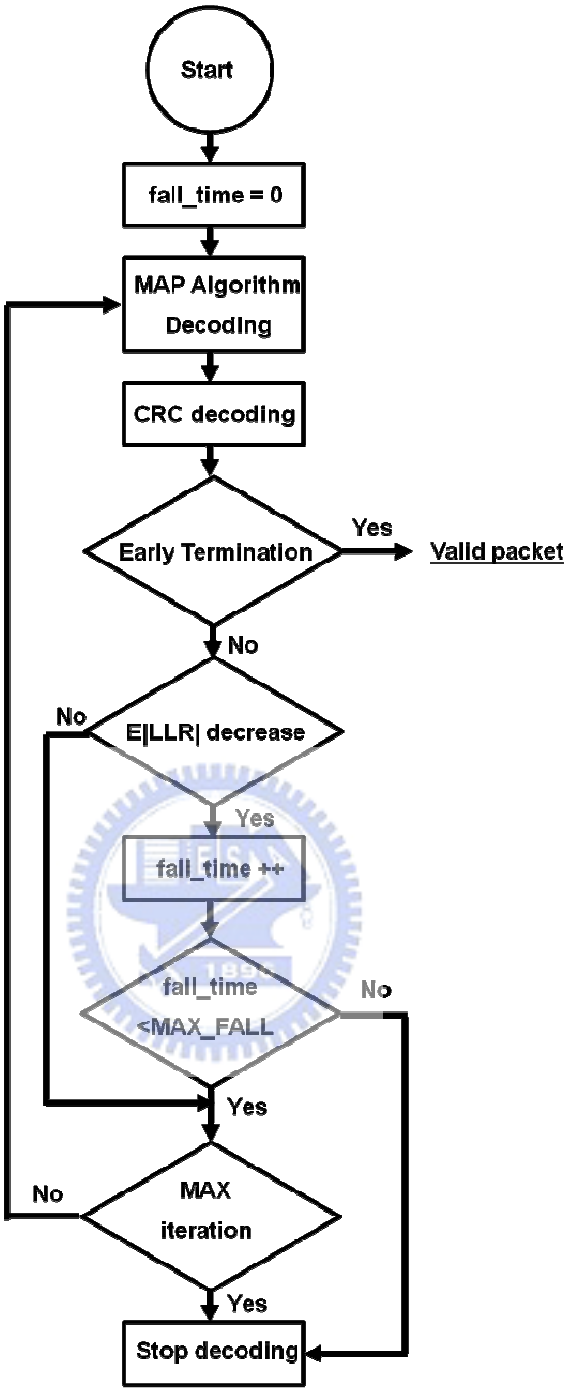

圖 30 以不同的下降次數為預棄條件的解碼流程圖

而底下我們對於不同的 MAX_FALL 設定做模擬，在此模擬三種不同的例子，MAX_FALL=1,2,3，而分別對應的模擬圖上的 Give-up when fall once，Give-up when fall twice，以及 Give-up when three times。而當 MAX_FALL=1 時，便等效於圖 29 也就是[25]的方法。

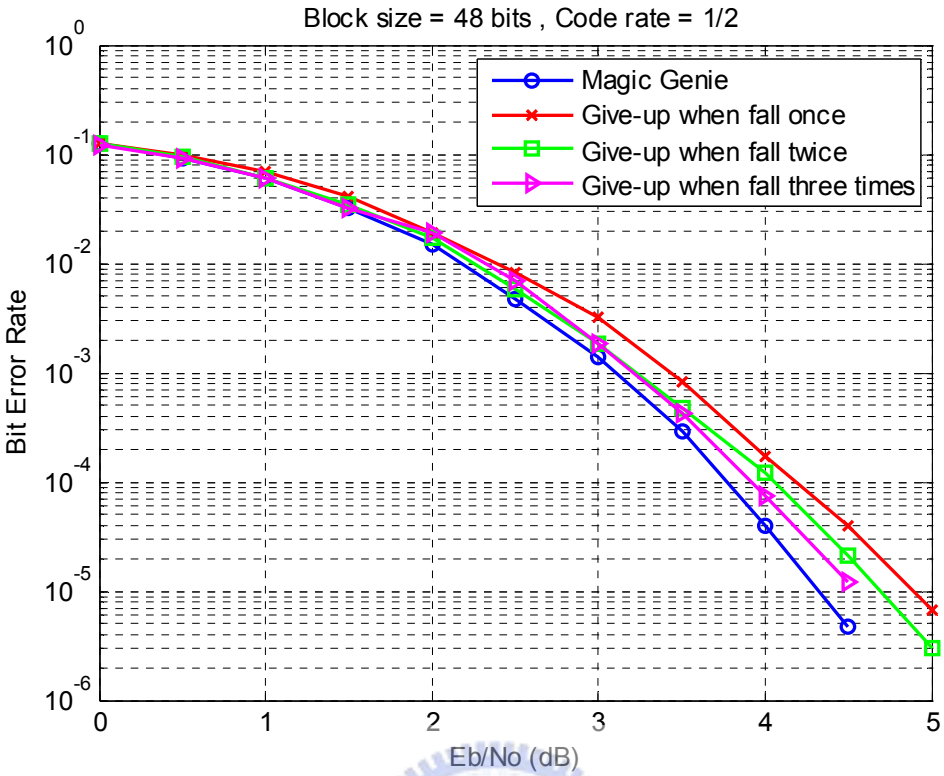

圖 31 位元錯誤率

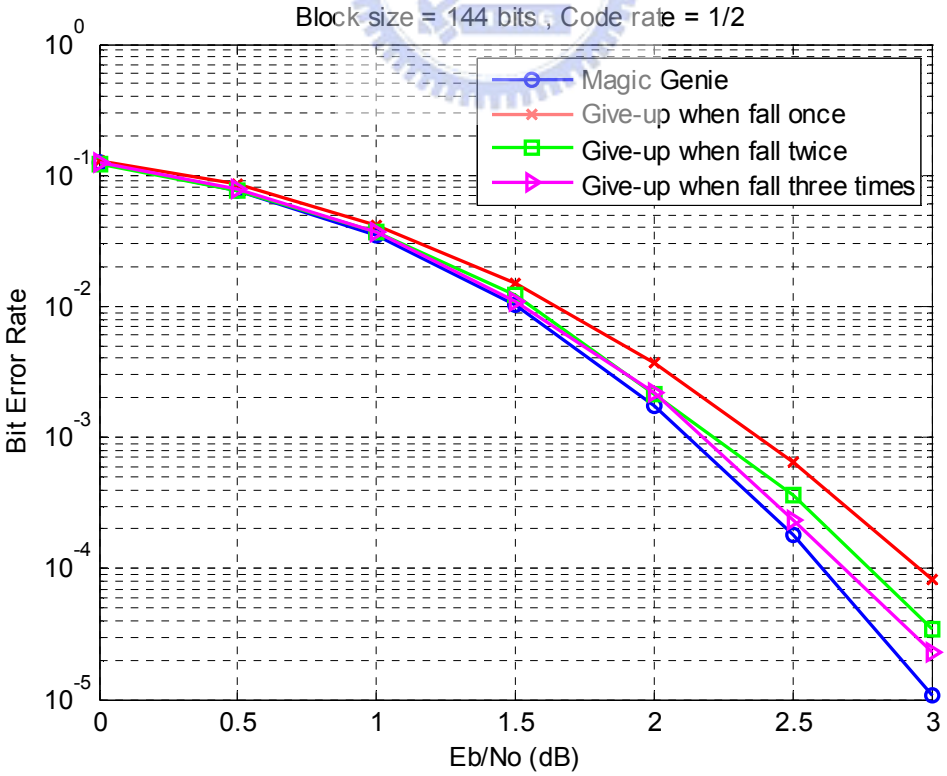

圖 32 位元錯誤率

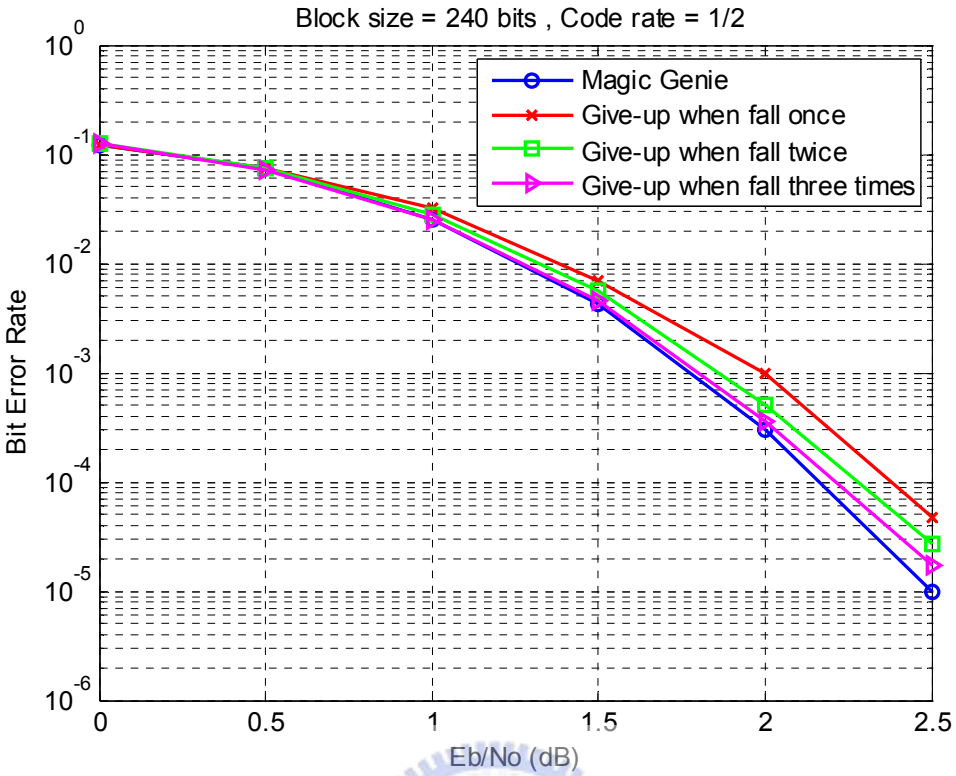

圖 33 位元錯誤率

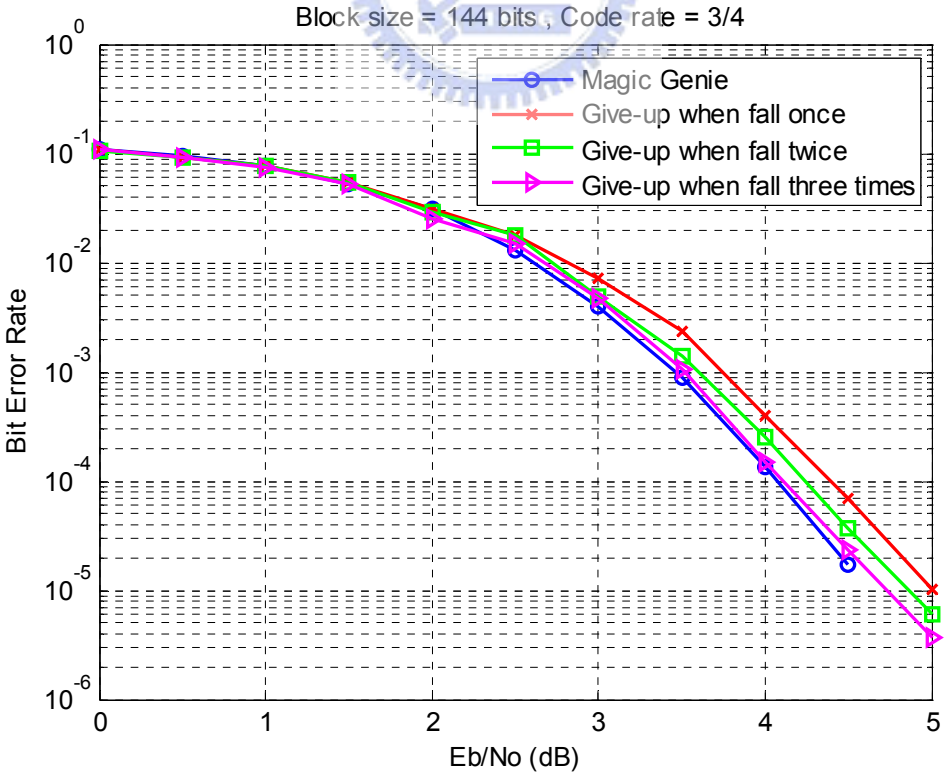

圖 34 位元錯誤率

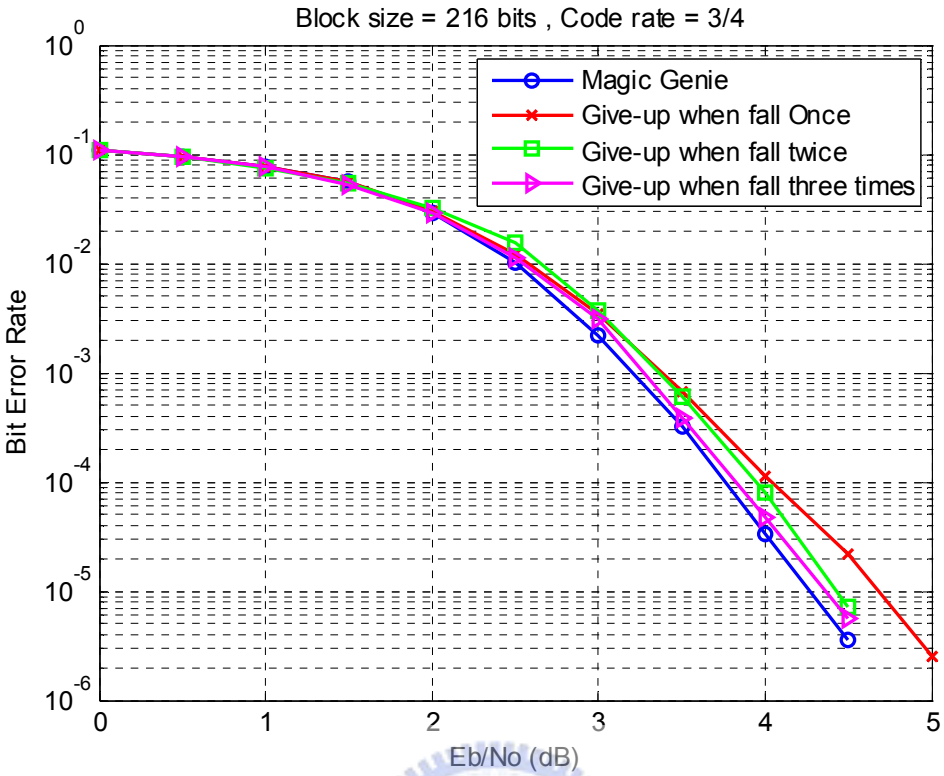

圖 35 位元錯誤率

在錯誤率方面，圖 31~35 中，若是選用設定落下次數等於 3 次的部分，在 BER=10^{-5}，的情況下，和 magic genie 大概最多只有 0.1dB 的差距，而且隨著區塊大小便大，一樣有便準的趨勢，而對於設定落下 2 次的部分，在 BER=10^{-5}下，最多大約有 0.2dB 的差距，而對於落下次數等於 1 次的部分，便和圖 17~22 有完全一樣的結果。

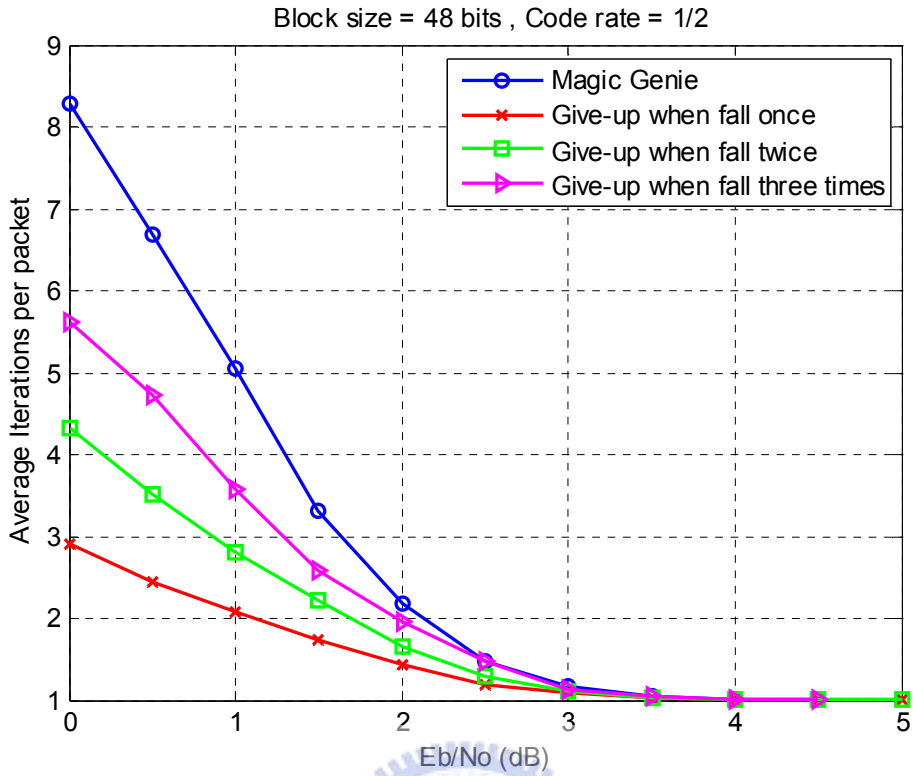

圖 36 平均遞迴次數

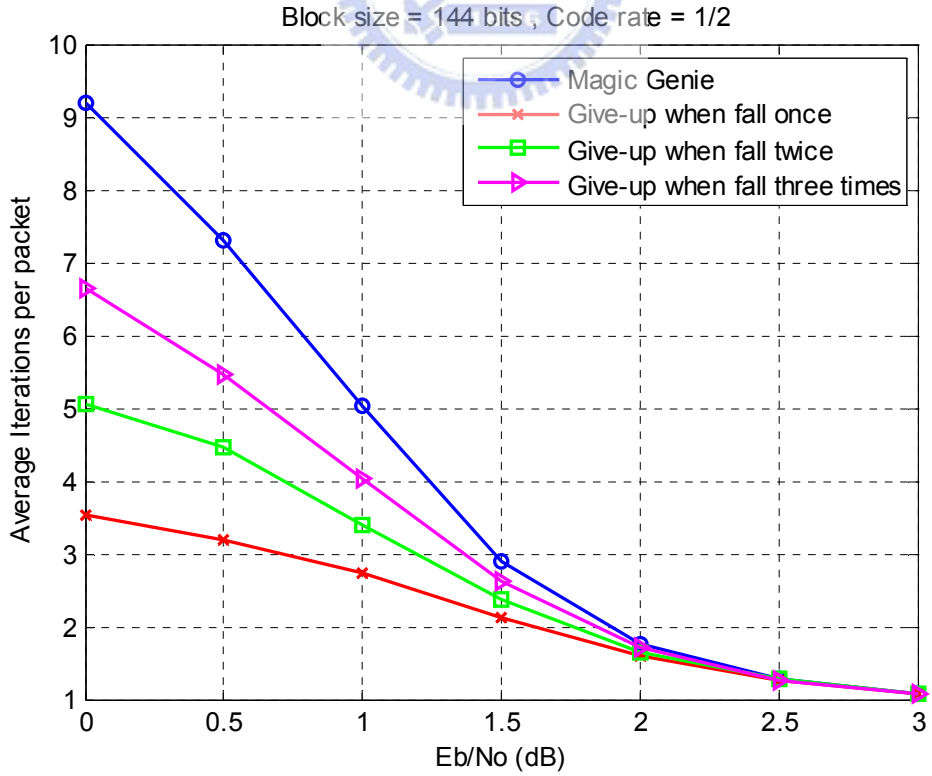

圖 37 平均遞迴次數

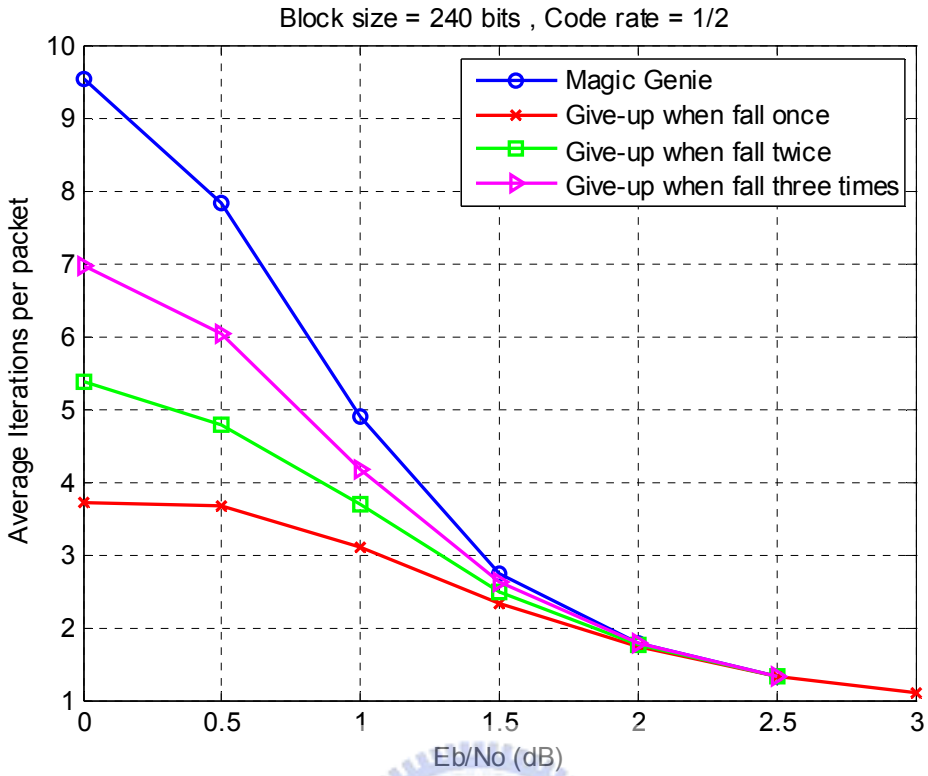

圖 38 平均遞迴次數

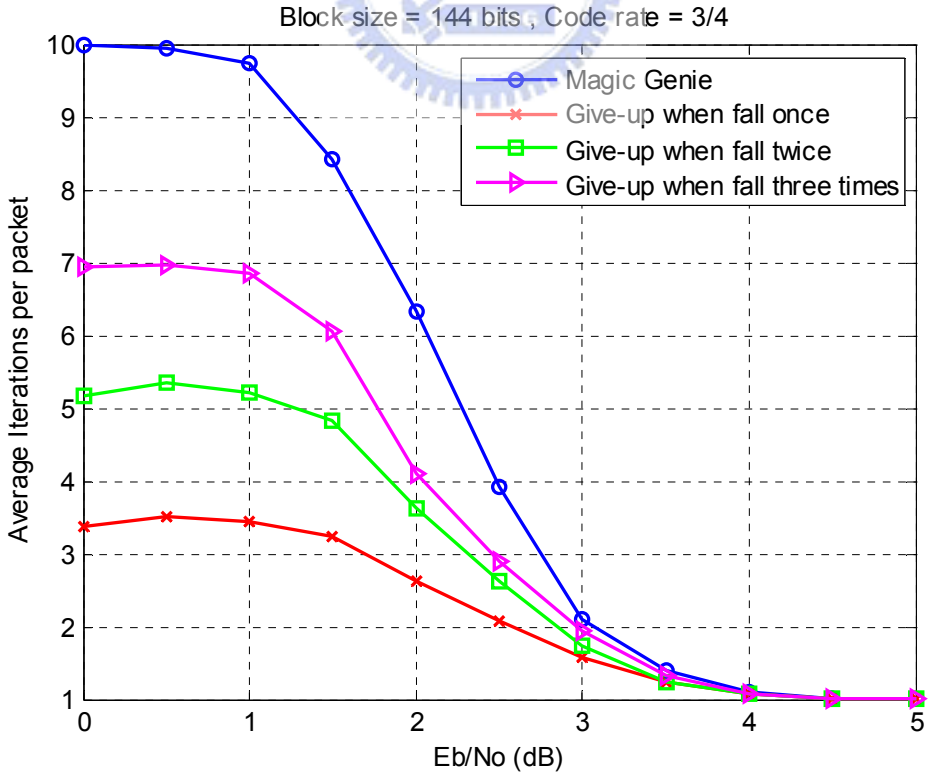

圖 39 平均遞迴次數

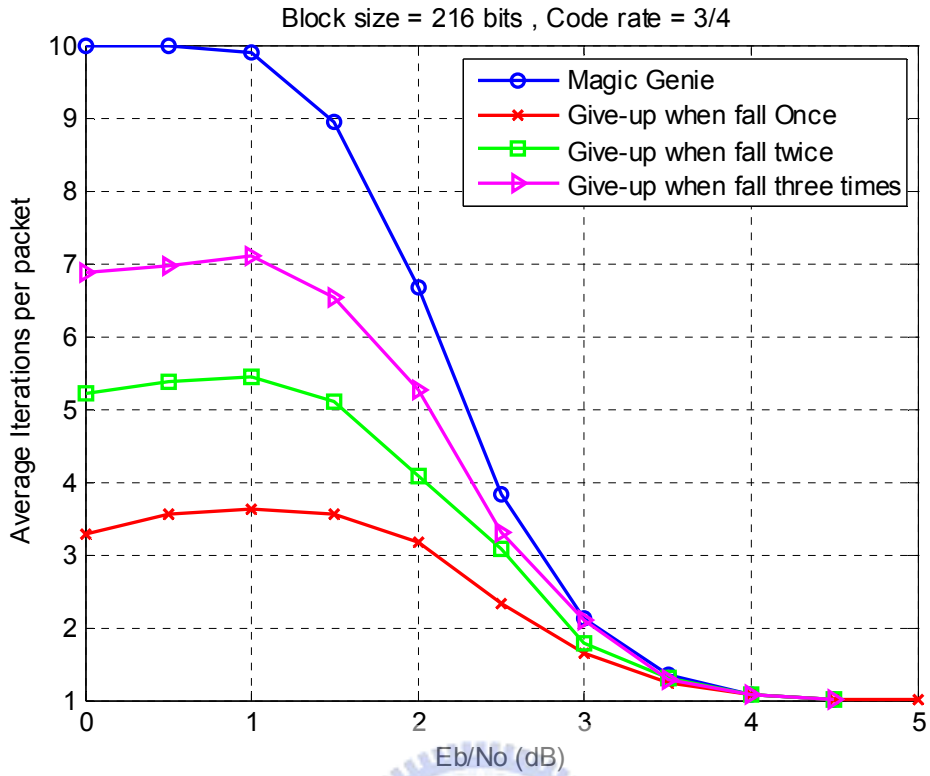

圖 40 平均遞迴次數

而看看圖 36~40 的部分，使用較大的 MAX_FALL 值，也就是給予封包越多的機會做解碼的確能讓錯誤率有效的降低，然而在平均遞迴次數的效能圖中又發現，其在低訊雜比(0~0.5dB)的部分，MAX_FALL 每增加 1，整體平均的遞迴次數也增加了約 1~1.5 次，這是因為我們讓真正能夠解碼成功的封包多一點的機會去解碼，但相對的也浪費了遞迴次數給解不開的封包。另外，在 MAX_FALL=3 的情形下，錯誤率在訊雜比較高的部分依然有一定的損失，這是因為我們只能保證在 $E[|LLR(x)|]$ 減少了三次之內，封包能夠解碼成功，若為其他的情形下，此封包便會被迫停止解碼，而如果我們將 MAX_FALL 設定成更大的數值，那基本上便沒有辦法節省多少遞迴次數。

模擬發現，利用以上的這種修正方法避免不了遞迴次數與錯誤率之間的嚴重拉扯，所以這讓我們又開始尋找在渦輪解碼中是不是有其他的更好的判斷方法能夠去估測封包是會解碼失敗的。而讓我們再看看 $E[|LLR(x)|]$ 的趨勢圖。

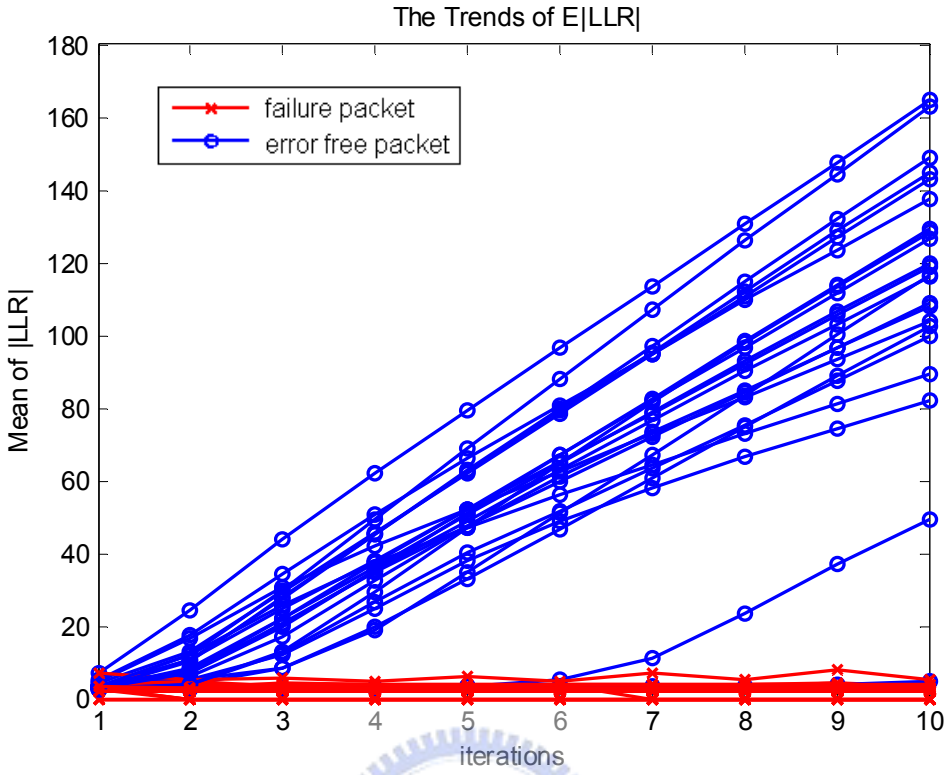

圖 41 LLR 絕對值的平均值的趨勢

圖 41 的模擬圖是根據資料區塊大小為 48 bits，碼率為 1/2，訊雜比(Eb/No)在 0dB 下所模擬的，而每一條線，所代表的便是一個封包它在 10 次的解碼遞迴中 E|LLR|的值，而藍色的線代表的是這筆封包在 10 次遞迴裡面，解碼能夠成功的封包，相反的，紅色的線代表的是在 10 次遞迴裡面這筆封包解碼失敗。

而可以看出來，對於解碼失敗的封包(紅色的部分)而言，它的 $E[|LLR(x)|]$ 值幾乎都是在很小的地方來回震盪，而根據模擬也發現其他的資料區塊大小或是碼率的封包也有一樣的趨勢，所以我們猜想是否可以設定一個臨界值，而將預棄的條件修正為當 $E[|LLR(x)|]$ 值發生減少的情形時，我們去判斷 $E[|LLR(x)|]$ 是否小於這個臨界值，若是則我們判斷它應該是屬於圖 41 底部的那些解不開的封包，便立即停止解碼，若否，則繼續解碼，整個流程圖如圖 42 所示，其中我們所改變的是在於判斷完 E|LLR|是否降低之後，多加了判斷它是否小於我們所設定的臨界值，而要注意的是，這邊用來判斷是否小於臨界值的 E|LLR|，我們是取上一

次遞迴的 $E|LLR|_{previous}$，而這是因為模擬發現利用 $E|LLR|_{previous}$ 來判斷是否小於臨界值的方法比用這次遞迴之後的 $E|LLR|$ 來做判斷能有較低的錯誤率，因為有些其實能夠解碼成功的資料的 $E|LLR|$，可能會在某次遞迴後突然降低很多，甚至降到了臨界值以下，所以我們可以用前一筆的值來做判斷以避免掉這方面的誤判。

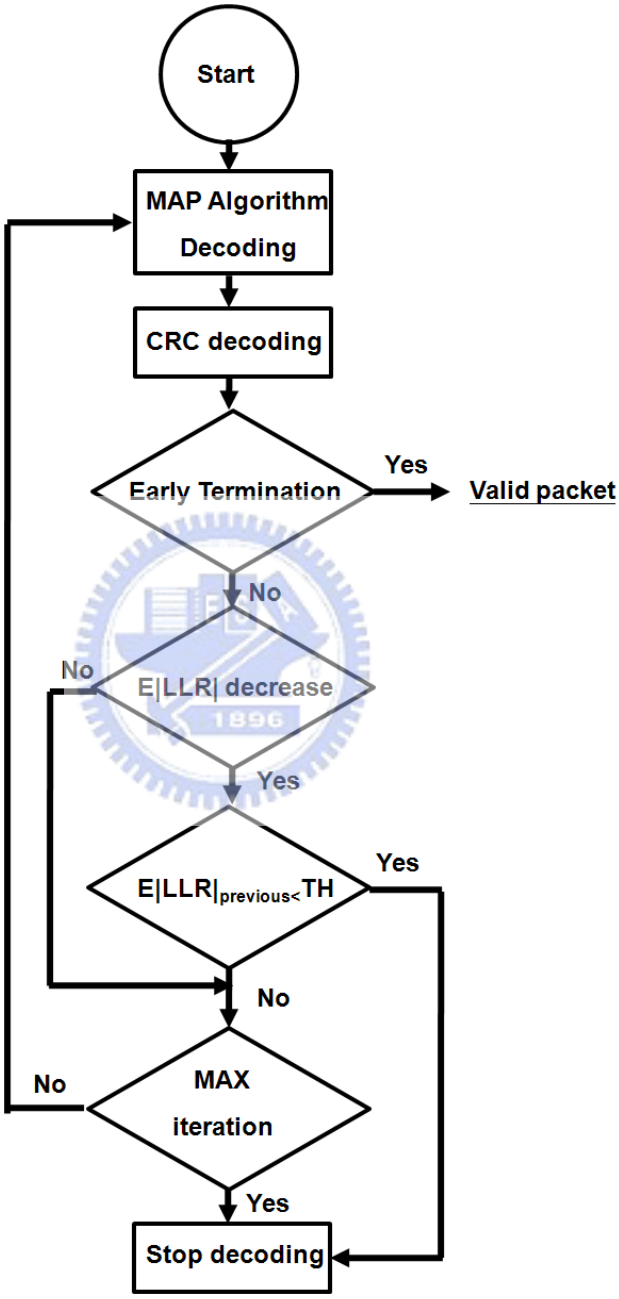

圖 42 以非單調遞增以及臨界值為預棄條件的解碼流程圖

而為了得知這個觀點應該是有用的，我以觀察類似圖 41 的模擬圖的方式，先定義出一個粗糙的臨界值並代入模擬。

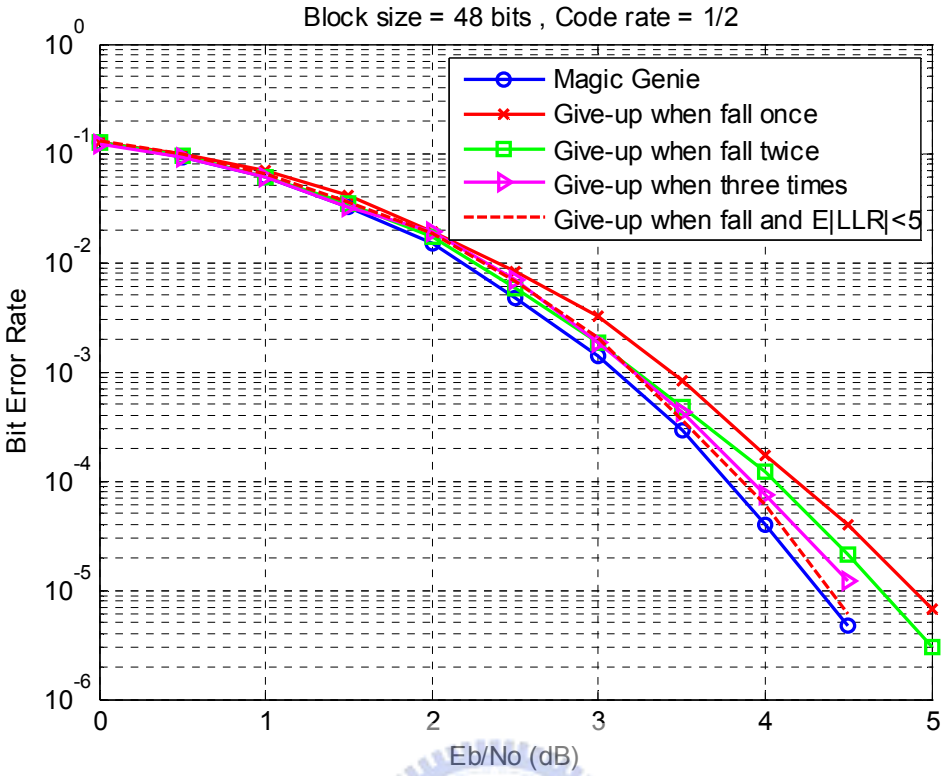

圖 43 以非單調遞增與臨界值判斷為預棄條件的效能比較

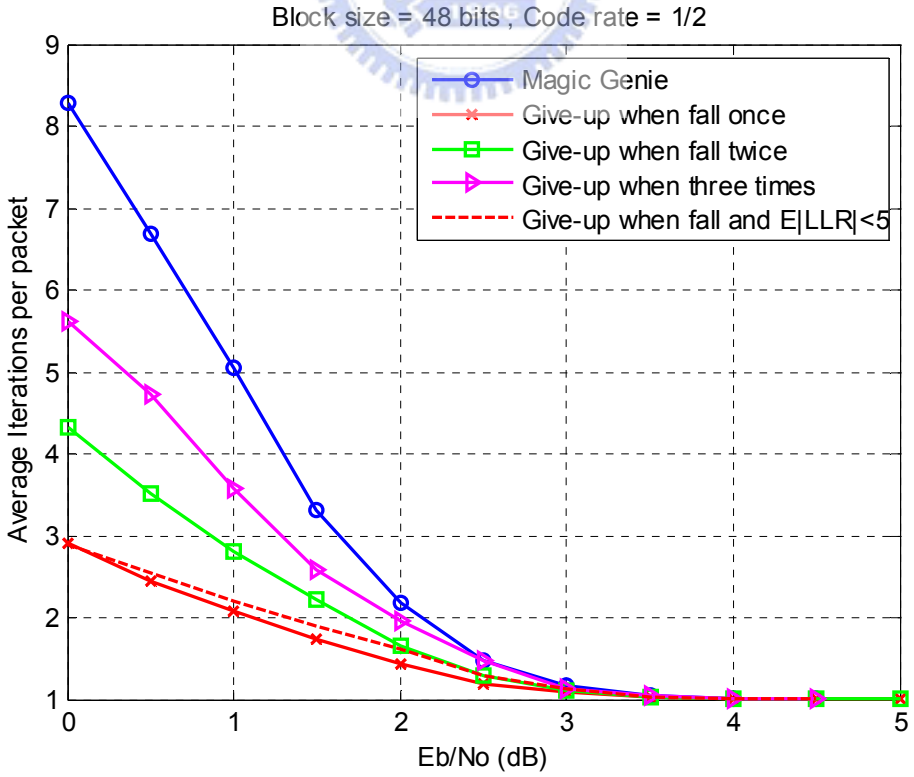

圖 44 以非單調遞增與臨界值判斷為預棄條件的效能比較

在圖 43，圖 44 中，我們比較使用臨界值的方法和不同降低次數的方法作比較，結果發現它在錯誤率的效能能夠很貼近 magic genie 的方法，甚至比下降三次就放棄解碼的方法來的低(在圖 43 中的 BER=$^{-5}$的情形下，與 magic genie 的差異剩下了約 0.05dB 左右)，而遞迴次數方面卻只比下降一次就放棄解碼的方法多付出了一些(從 Eb/No=0~3dB 間，最多多了 0.2 次遞迴)，所以這方法不論在通道訊雜比低或高的情形下，都能有很不錯的表現。而剩下的問題便是如何去決定此臨界值，我們先看圖 45，其中我分別設定了三個不同的臨界值 3，5 與 7，而模擬結果可以看到當臨界值訂的越高時，錯誤率和平均遞迴次數都會漸漸接近下降一次就放棄解碼的效能，因為會有越多的封包在 $E[|LLR(x)|]$ 發生下降時被直接放棄掉，反之。若臨界值訂的越低，則錯誤率和平均遞迴次數便會漸漸接近 magic genie 的方法。在此篇論文中，我調整臨界值的準則是基於讓位元錯誤率在大於等於 10^{-5} 時和 magic genie 的方法在訊雜比上相差不超過 0.1dB 為原則，而盡可能的降低平均遞迴的次數。

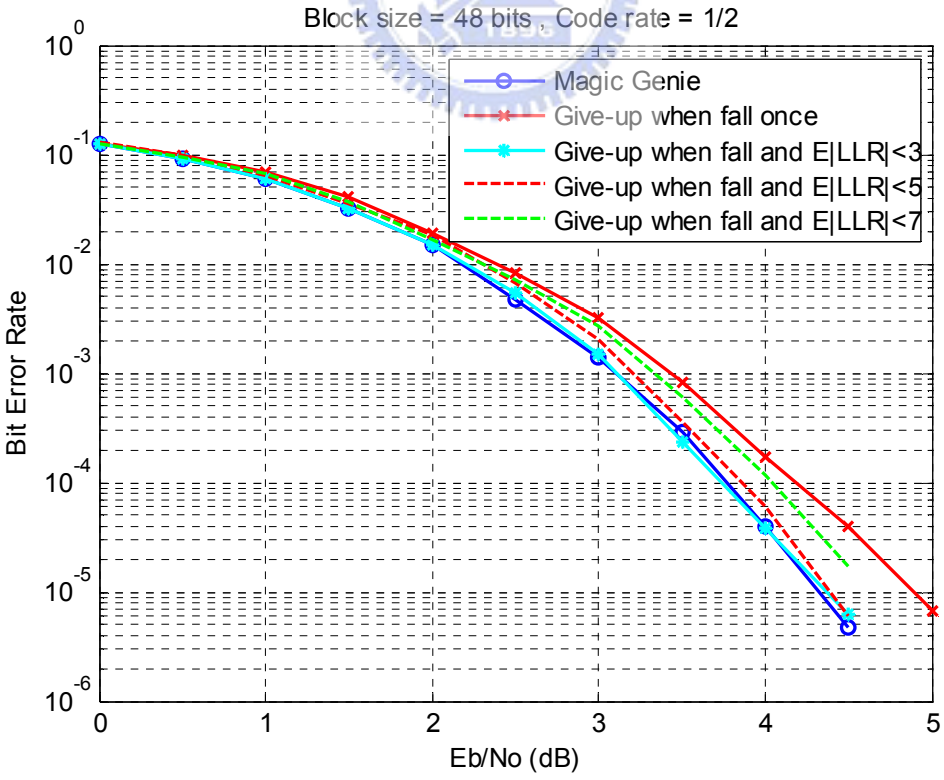

圖 45 不同的臨界值在位元錯誤率上的差異

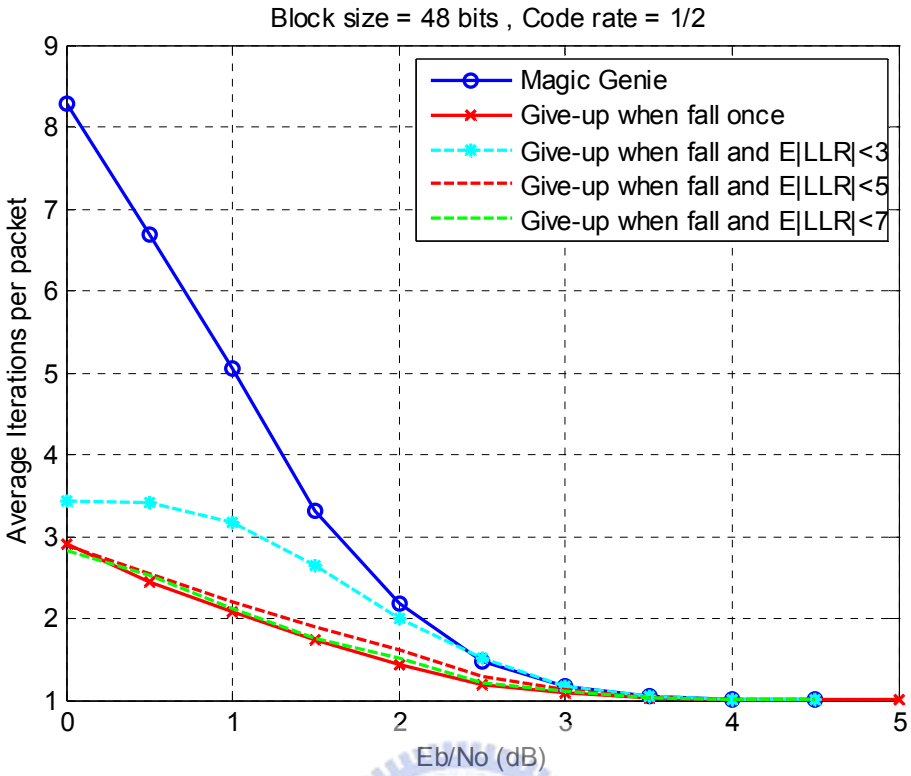

圖 46 不同的臨界值在平均遞迴次數上的差異

由圖 46 中可以看到，臨界值為 3 的遞迴次數在 Eb/No=0~3dB 間，最多約比下降一次就放棄的方法多了 1 次的遞迴次數，然而當臨界值設定為 5 時，便只比下降一次就放棄的方法在 Eb/No=0~3dB 間多了約 0.2 次遞迴，然而它們之間的錯誤率差別在圖 45 中可以看到卻不到 0.05dB。

而根據模擬的統計，對於相同的碼率而言，各種區塊大小的最佳化臨界值是差不多的，而當碼率上升時，臨界值也必須往上調整。換句話說，高碼率時要能解碼成功，$E[|LLR(x)|]$ 的門檻必須較高，而這原因或許可以從(22)式來推論

$$LLR(d_k) == \ln\frac{\Pr\{y_k^s \mid d_k = 1\}}{\Pr\{y_k^s \mid d_k = 0\}} + \ln\frac{\sum_m\sum_{m'}\alpha_{k-1}(m')\gamma_k^1(y_k^p, m', m)\beta_k(m)}{\sum_m\sum_{m'}\alpha_{k-1}(m')\gamma_k^0(y_k^p, m', m)\beta_k(m)}$$

對於高碼率的 codeword 來說，所收到的平等位元(parity bits)數目較少，所以在所有位元中 $\gamma_k^i(y_k^p, m', m)$ 這項有值的數目也較少，而每個時間點資料的

LLR 值其實是由所有時間點的路徑值所遞迴累積起來的，即高碼率的 codeword 其 $|LLR(x)|$ 等於是由少部分的 $\gamma_k^i(y_k^p, m', m)$ 資訊來推算，所以就算 $E[|LLR(x)|]$ 已經達到和低碼率時一樣的臨界值，準確率卻比較低。所以若將我們將低碼率時的臨界值放到高碼率來使用，等於是放任部分解不開的封包繼續解碼，所以會造成遞迴次數的放費。

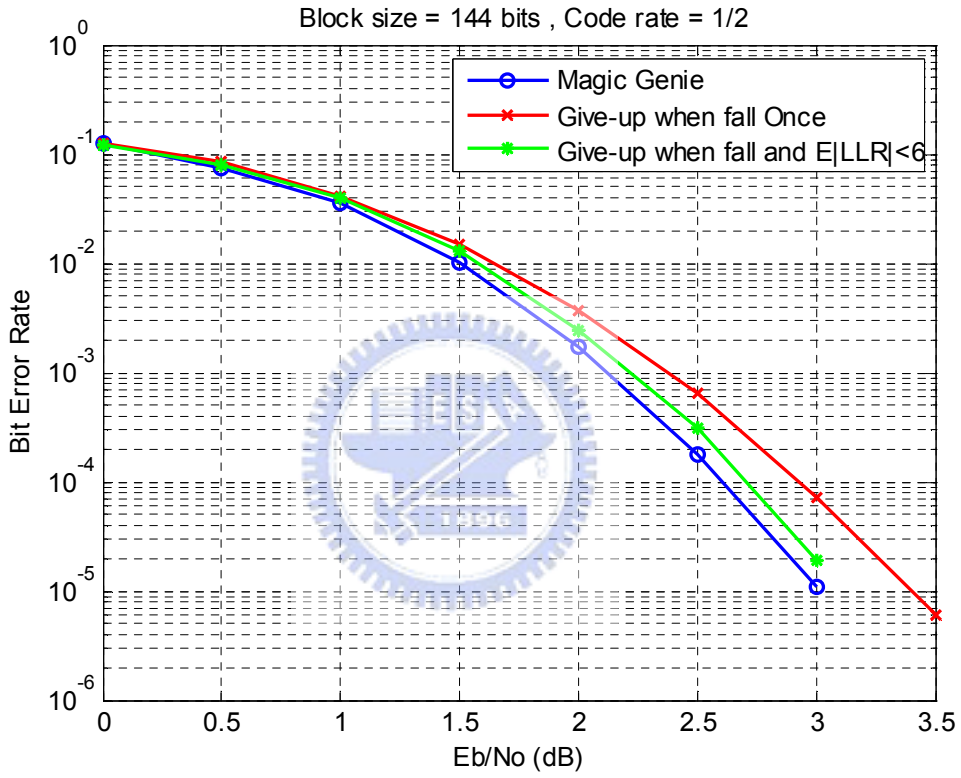

圖 47 最佳臨界值的位元錯誤率

圖 47 是以不讓所設定的臨界值跑出來的錯誤率在 10^{-5}dB 時差異超過 0.1dB 的原則所調出來的，可以看到約在 BER=10^{-5}dB 時，Eb/No 的差異比下降一次就放棄的方法改善了 0.3dB 左右。

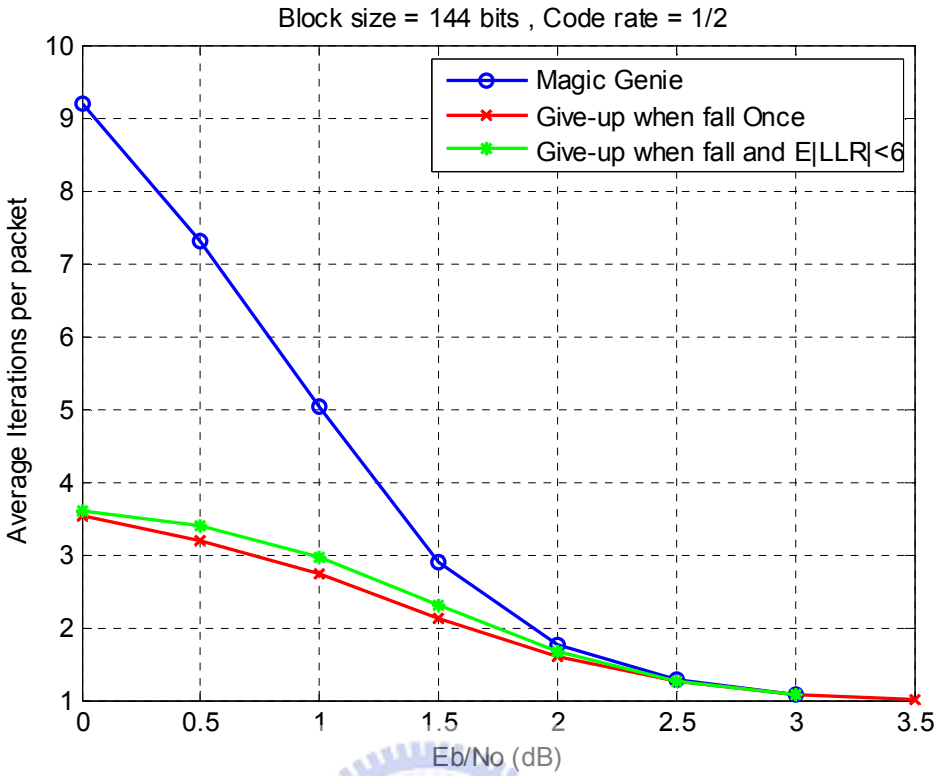

圖 48 最佳臨界值的效能

而圖 48 中可以看到，在 Eb/No=0~3dB 間，只要多付出約 0.2 次的遞迴次數，便可以達到如在圖 47 中和 magic genie 間錯誤率的差異只剩下 0.1dB。

在圖 49 中，其中 Eb=1~3dB 這段中，我們看到遞迴次數因為臨界值設定不佳而增加了約 1.5 次之多，然而若臨界值調整的好，例如 th=9 時，遞迴次數在同樣的區段卻只增加了約 0.4 次的遞迴，然而這兩個臨界值所產生的錯誤率在圖 50 中可以看出來在 BER=10^{-2} 的情況下，Eb/No 約只差了 0.05dB 而已。

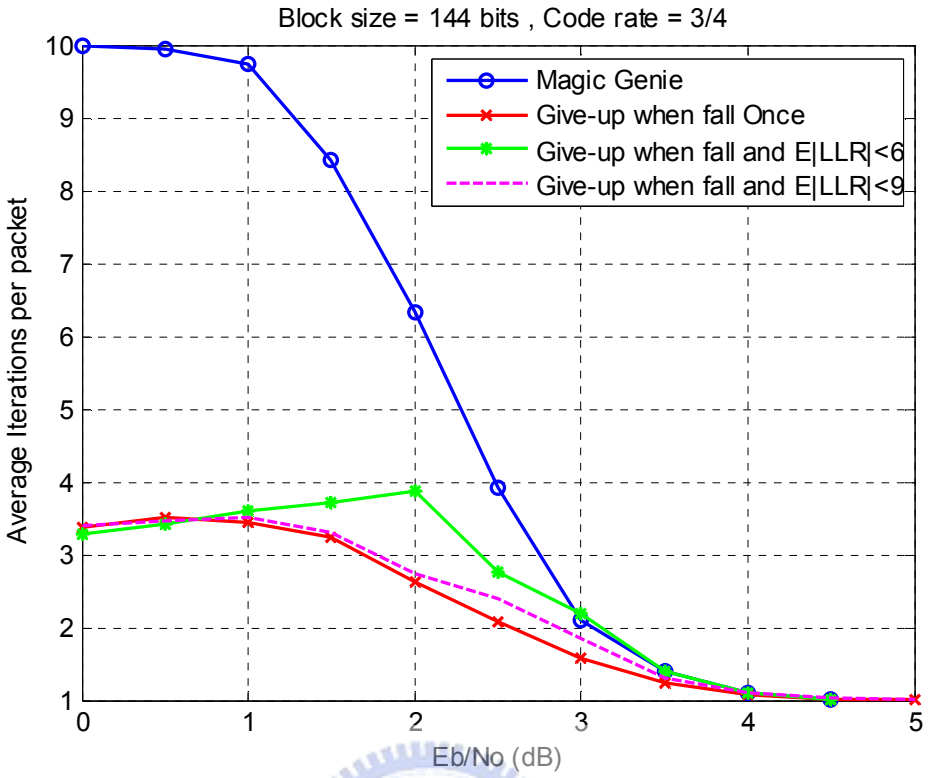

圖 49 最佳臨界值的效能

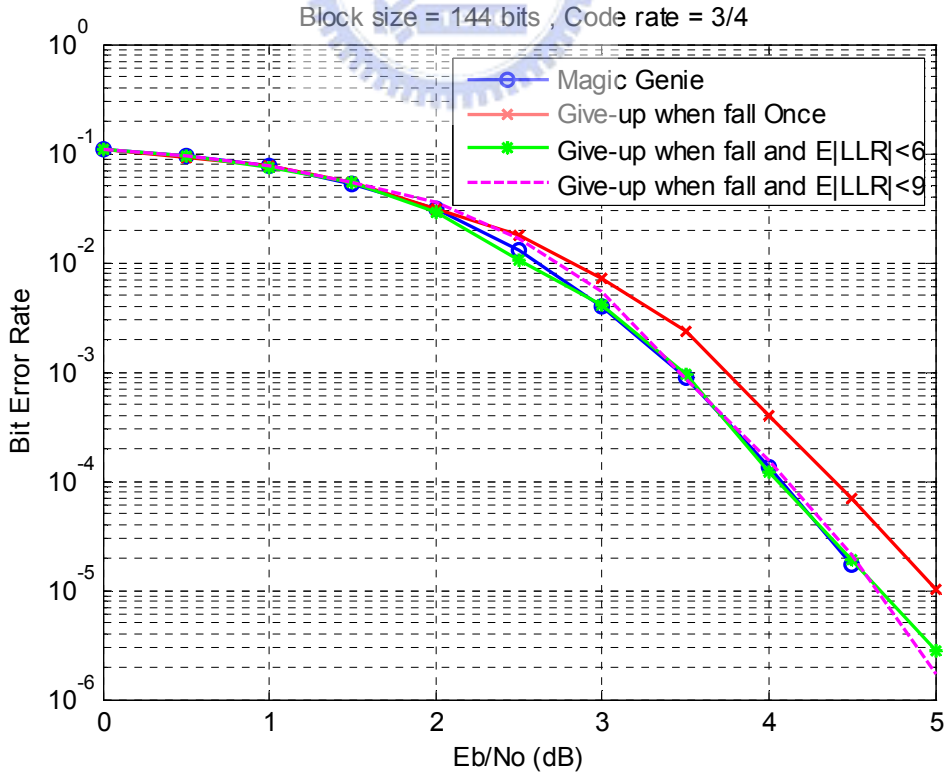

圖 50 最佳臨界值的效能

3-3 狀態再利用技術(State Reuse Technique)

在 3-2 裡，我們已經提出了一個能夠有效的節省渦輪解碼的遞迴次數的方法，而其精神是在於去猜測封包是否有解碼失敗的可能，若有便盡快的停止解碼，以達到節省遞迴數目的目的。然而，在解碼失敗後，必須重傳相同的資料再繼續解碼，假設重傳封包的資料是和前一筆失敗的資料是一模一樣的，而差別只是在於兩筆資料在傳送時所加入的通道雜訊有所不同，所以在我們想雖然前一筆資料解碼失敗了，但是是否有留下什麼資訊能夠幫助新的一筆資料作解碼，而讓我們回顧一下第二章所提到的渦輪解碼架構圖，如圖 51 所示。

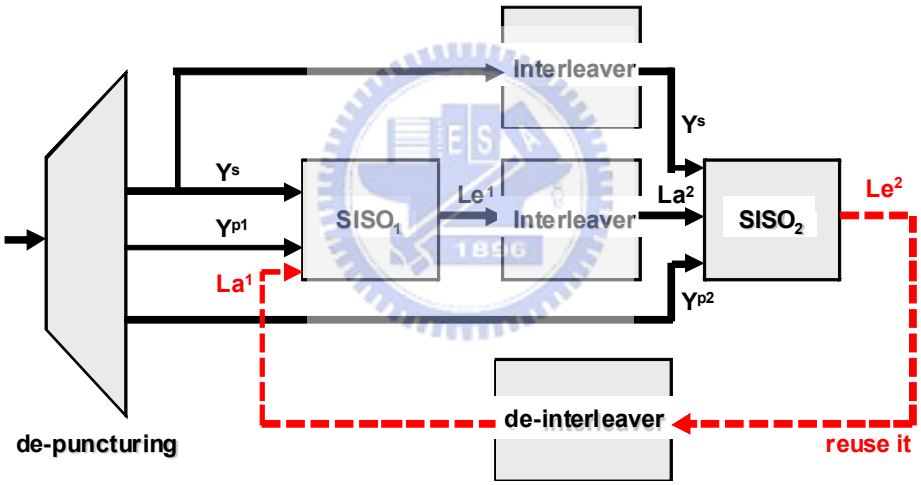

圖 51 渦輪解碼架構圖

我們知道渦輪解碼器中的兩個 SISO 解碼器會互相傳遞外部資訊(extrinsic information)當作另一個解碼器的事前資訊(a priori information)，而在 $SISO_1$ 解碼器第一次解碼的開始，$SISO_2$ 並還沒有辦法提供外部資訊供它做參考，所以在傳統的作法中，便將 La^1 設為零。然而，[25]中提出一種作法，假設現在有筆封包解碼失敗，停止解碼後，重新接收了一筆相同的資料，那在上一筆封包停止解碼前所算出的 Le^2 是否可以拿來作為 La^1 的初始值而非毫無頭緒的設成零，如圖 51 中紅色路徑所示。

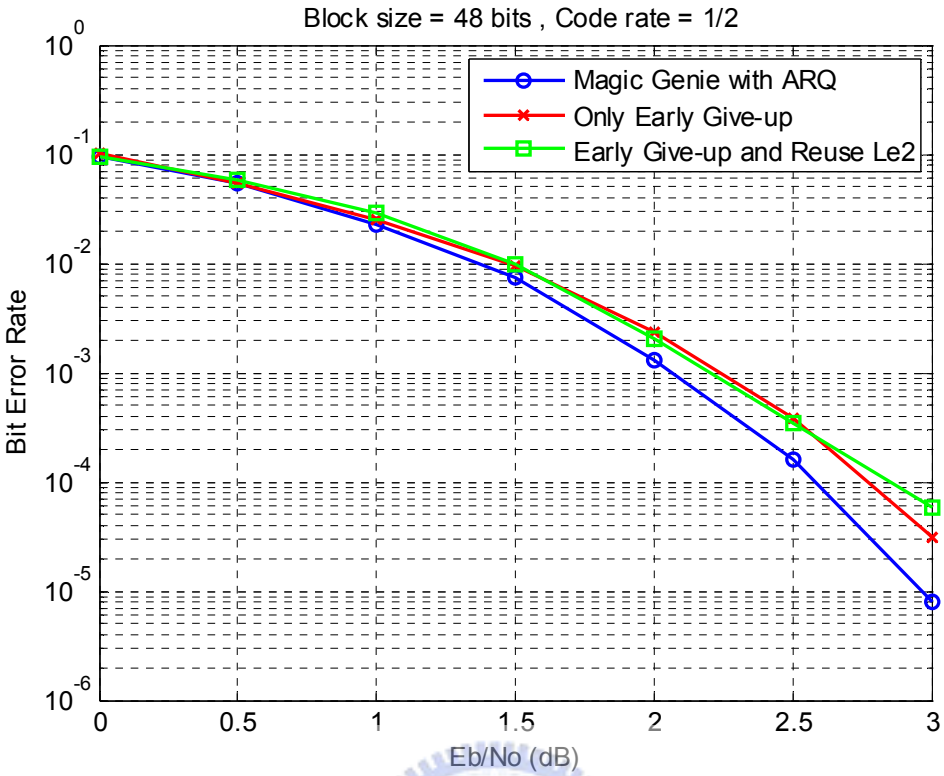

圖 52 使用前一筆封包的 Le^{2} 做為 La^{1} 的效能圖

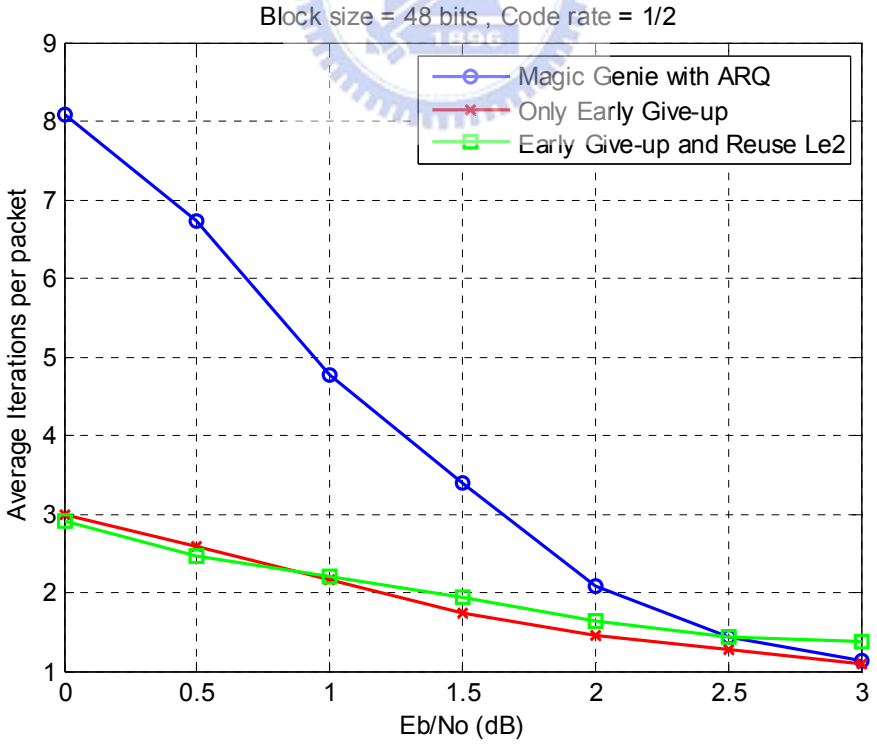

圖 53 使用前一筆封包的 Le^{2} 做為 La^{1} 的效能圖

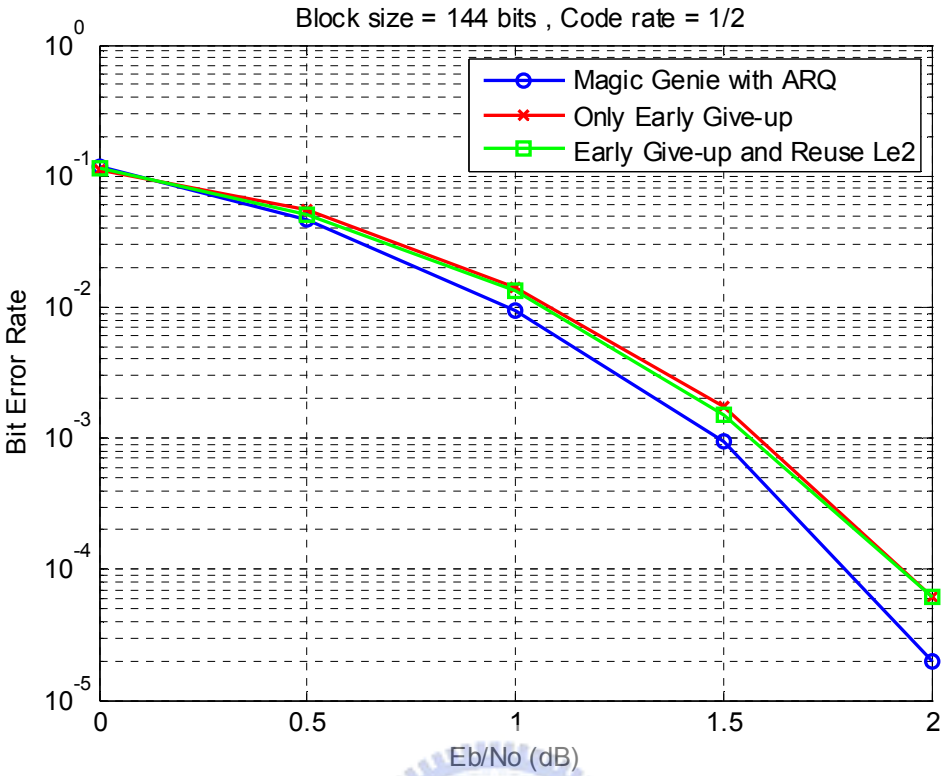

圖 54 使用前一筆封包的 Le^{2} 做為 La^{1} 的效能圖

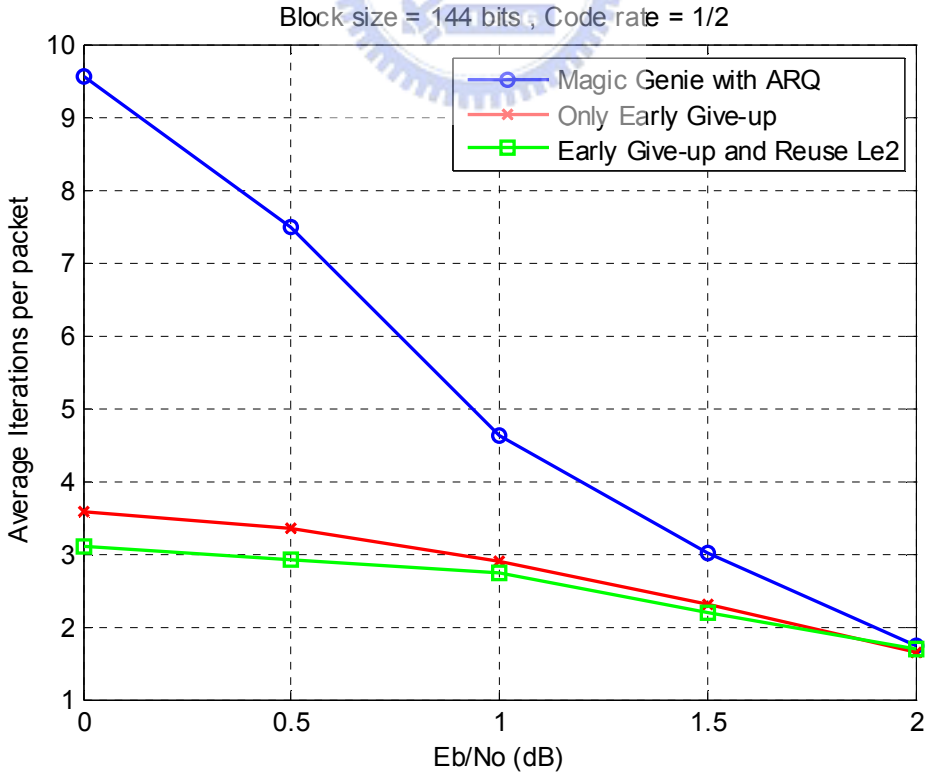

圖 55 使用前一筆封包的 Le^{2} 做為 La^{1} 的效能圖

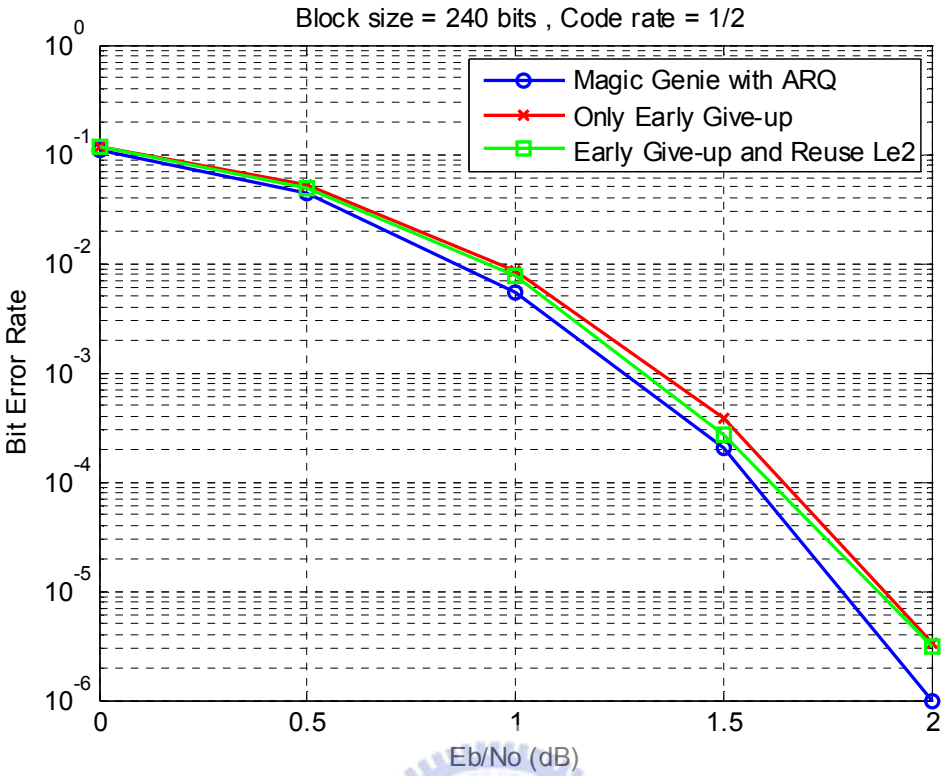

圖 56 使用前一筆封包的 Le^2 做為 La^1 的效能圖

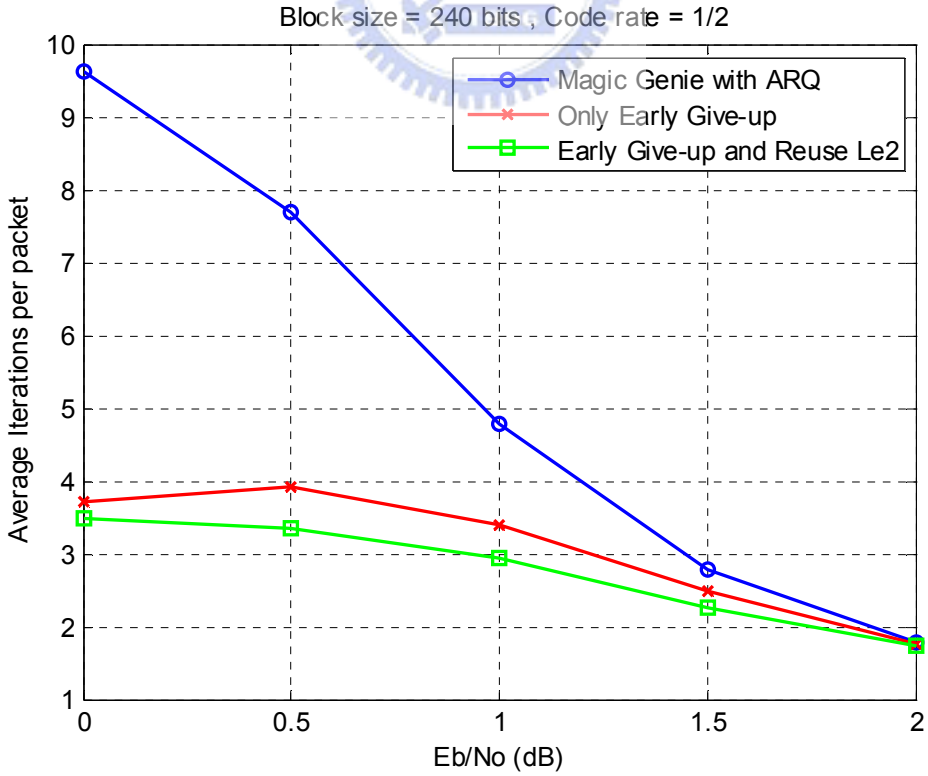

圖 57 使用前一筆封包的 Le^2 做為 La^1 的效能圖

圖 52~57 中，”Magic Genie with ARQ”指的是在第一筆資料的解碼中，我們使用了 Magic Genie 方法作解碼，而除非解出了正確的資料或是到達了最大的解碼次數，否則我們不會停止解碼，而當資料解碼錯誤時，會要求重送一筆資料再重新解碼，而第二筆資料的解碼中，一樣是使用 Magic Genie 的方法來做解碼，所以”Magic Genie with ARQ”所得到的便是我們所能達到最低的錯誤率極限，而“Only Early Give-up”指的是在第一筆資料的解碼中，我們只使用了只利用 3-2 節所討論的最佳的欲棄技術，而當在第一筆資料解碼錯誤時，會要求重送資料，而在第二筆資料解碼時，一樣是只利用到 3-2 節所討論的最佳預棄技術。而”Only Early Give-up and Reuse Le2”指的是除了使用最佳的預棄技術外，在解碼重傳的封包時，利用上一筆資料解碼留下來的 Le^2 做為解碼的初始值。而也就是說對於以上模擬，是對於每個不同的封包，我們給予最多重傳一次的機會，而所計入的錯誤位元數是在重傳了一筆新的資料後還是解碼失敗的情形下所留下來的錯誤數目。而平均遞迴次數的部分則是單指”所有重傳的封包”的平均解碼遞迴次數，如此我們才能看出在使用了 Le^2 當作初始值後是否有得到任何好處。

由圖 55，圖 57 我們觀察到使用上一筆封包的 Le^2 做為解碼的初始值的方法，能夠省下最多約 0.5 次的遞迴次數，但我們發現資料區塊太短的情況下，甚至沒有改善(如圖 53 所示)，我想這應該是因為由於我們所使用的 Le^2 畢竟是一筆解碼失敗的封包，所以其中必定有些位元所指的方向是錯誤的，而這些少部分的錯誤初始值對於較長的碼而言，能夠很快的被其他的位元所修正回來，而太短的碼便可能反而需要比原來更多的遞迴次數去修正這些錯誤。所以這方法對於區塊越長的封包而言似乎有越顯著的改善。

且在錯誤率的模擬圖方面，圖 52，54，56 看出來，錯誤率並沒有什麼樣的改善。

那除了可以使用 Le^2 做為新一筆資料的事前機率外，是否還有其他的資訊能夠被使用的呢？而觀察圖 51 的渦輪解碼架構圖可以看到，在各個節點間都散佈著各種不同的 LLR 值，而這些值只要能夠用來提供 $\Pr\{S_k = m \mid S_{k-1} = m'\}$ 這個機率究竟是朝向那個方向，如果方向正確了，那應該就能夠加速解碼的收斂速度。根據[22]的研究中，其並非使用 Le^2 做為起始的 La^1，而是使用 LLR_2 做為 La^1 的起始值，而在 $SISO_1$ 做完第一次解碼產生 Le^1 後，$SISO_2$ 第一次解碼的 La^2 設定並非為 Le^1 而是 Le^1 加上前一筆封包所留下來的 LLR_2，所以現在必須針對這兩種作法做效能上的比較。

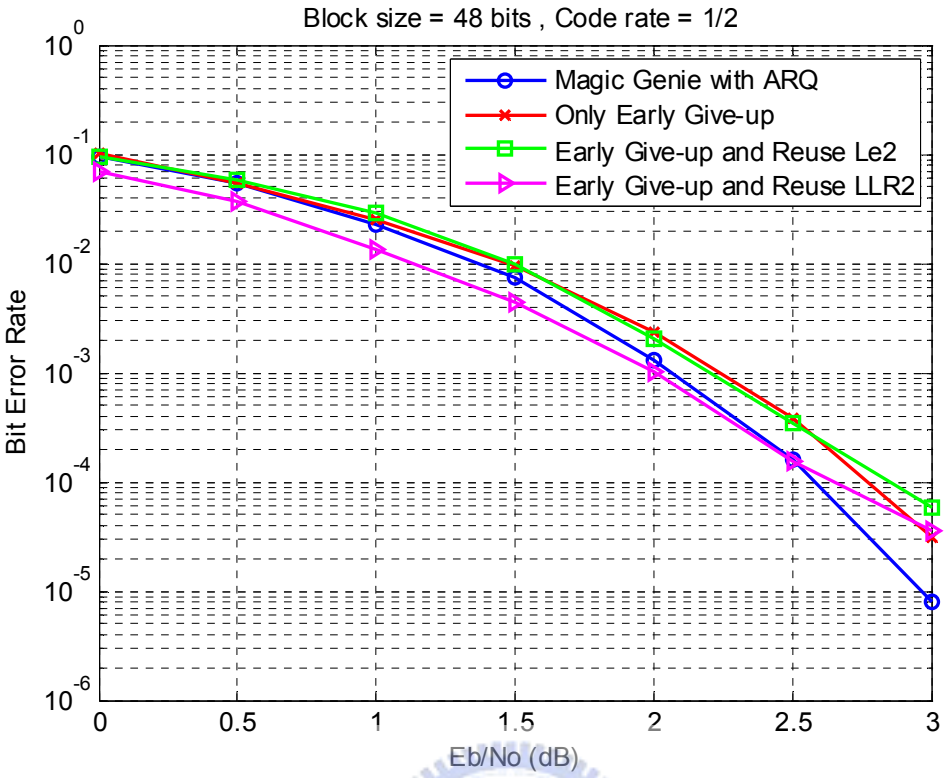

圖 58 使用不同的 La^{1} 起始值的效能比較

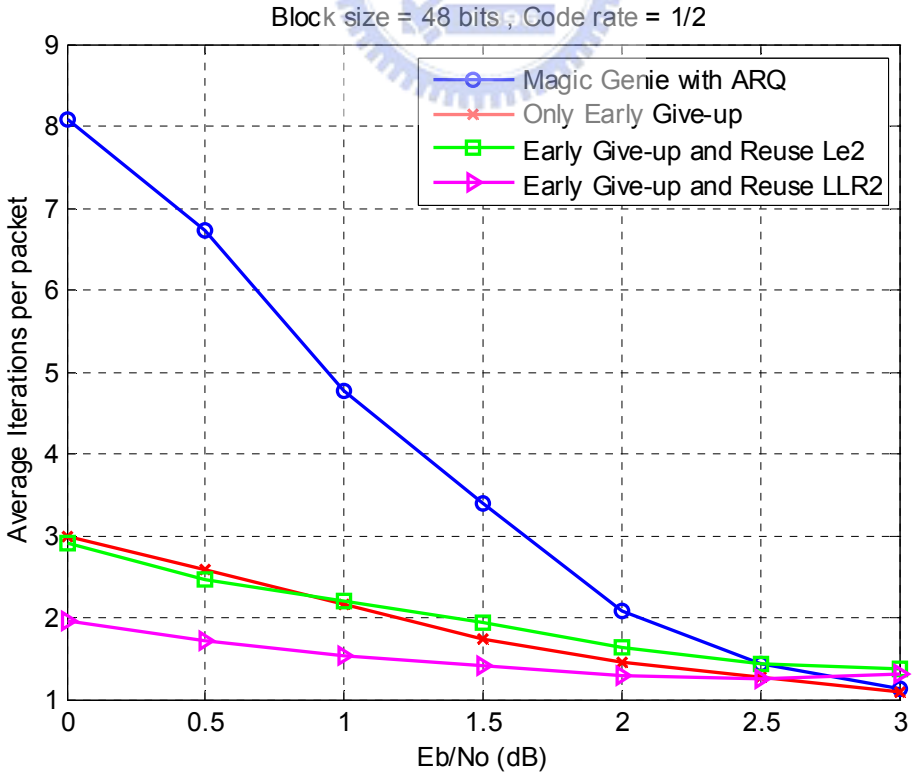

圖 59 使用不同的 La^{1} 起始值的效能比較

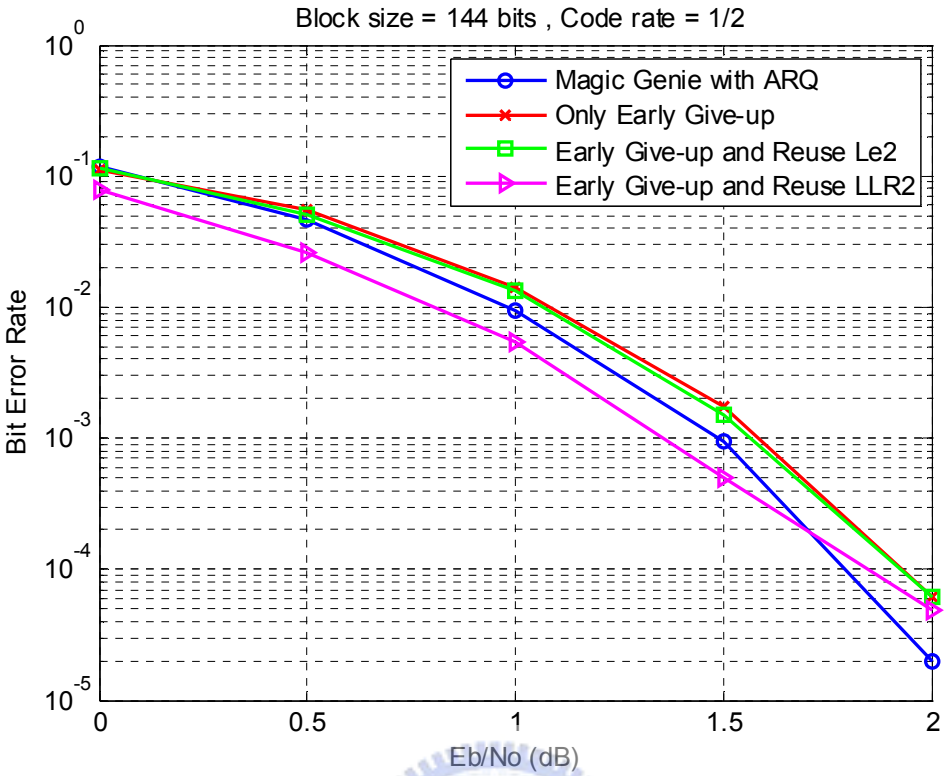

圖 60 使用不同的 La^1 起始值的效能比較

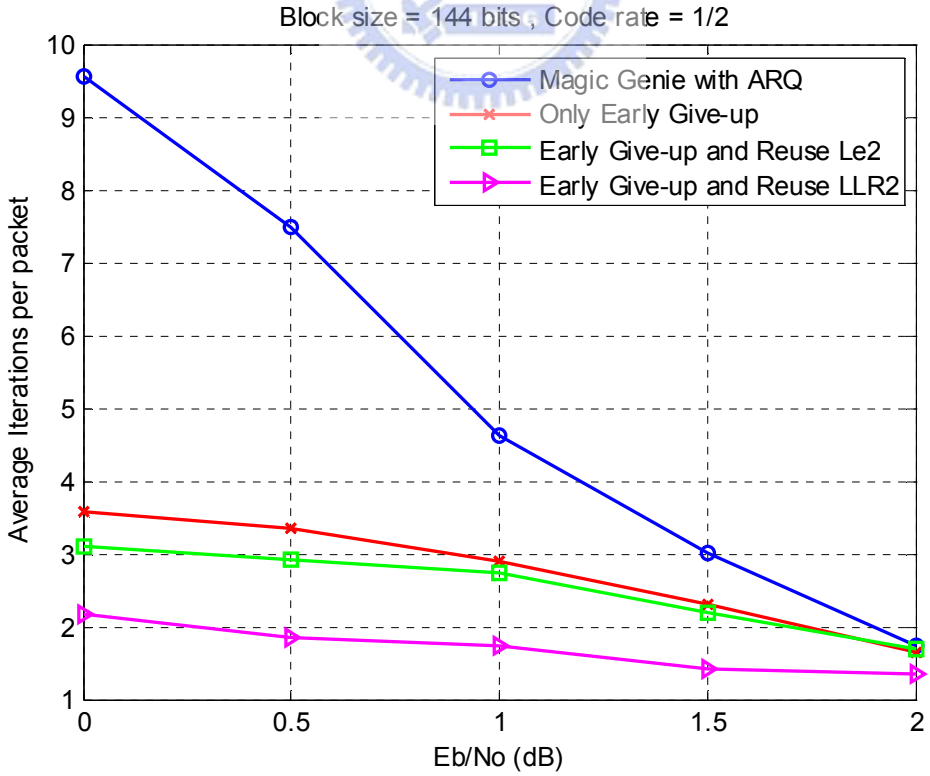

圖 61 使用不同的 La^1 起始值的效能比較

由圖 59，圖 61 的模擬結果中，可以看出使用 LLR_2 做為初始值的方法能讓平均遞迴次數有約有比原本只有預棄的方法下約有 50%減少，且除了 Eb/No 在圖 59 的 3dB 時沒有改善以外，其他的通道狀態下都有改善，而在錯誤率方面，看圖 58 和圖 60 可知對於較長的碼甚至能比不使用狀態再利用的 magic genie 方法有更低的錯誤率，而在[22]這篇論文中已經預期錯誤率能夠有所善，然而此篇論文中並沒有提到區塊大小對於這個結果的影響。

而使用 LLR^2 之所以能比 Le^2 有較好的效果，可能是因為 $LLR^2(d_k)$ 基本上包含了所有時間點的系統位元及平等位元的資訊，而 $Le^2(d_k)$ 只與部分位元的資訊有關係，所以 LLR^2 應該會比 Le^2 來的準確。

儘管在訊雜比較低的情況下(在圖 58 的 2.5dB 以下及圖 60 的 1.5dB 以下)，以上的方法能讓錯誤率降低，但是觀察圖 58 的位元錯誤率的圖會發現在 En/No = 3dB 時，錯誤率卻反而比 magic genie 來的高，而這個現象可能與所使用的 $E|LLR^2|$ 的大小有所關聯。

當封包解碼失敗時，$E|LLR_2|$ 值有可能已經累積成一個很大的值，如圖 62 和圖 63 所示(每一條線所代表的是一筆封包解碼失敗時，E|LLR|的趨勢圖)，而因為封包解碼失敗所以留下來的 LLR^2 裡面必定有些錯誤的方向性，所以若我們將太大的 LLR^2 值拿來做新一筆資料解碼的初始值，可能沒有辦法將這些錯誤的方向性的位元更正回來，使得錯誤率上升，所以我們將狀態再利用的條件修正成只有當上一筆資料解碼失敗且 $E|LLR^2|$ 小於預棄條件的臨界值時，才進行狀態再利用。由圖 64 與圖 65 可以看到，雖然我們將狀態再利用的使用條件修正之後在訊雜比較低時多付出了一點點遞迴次數(不到 0.1 次的遞迴)，但是換來的是錯誤率上能夠有所改善(在圖 64 的 2dB 時和圖 66 中的 1.75dB 的確能夠降到和 magic genie with ARQ 的方法一樣低了)。

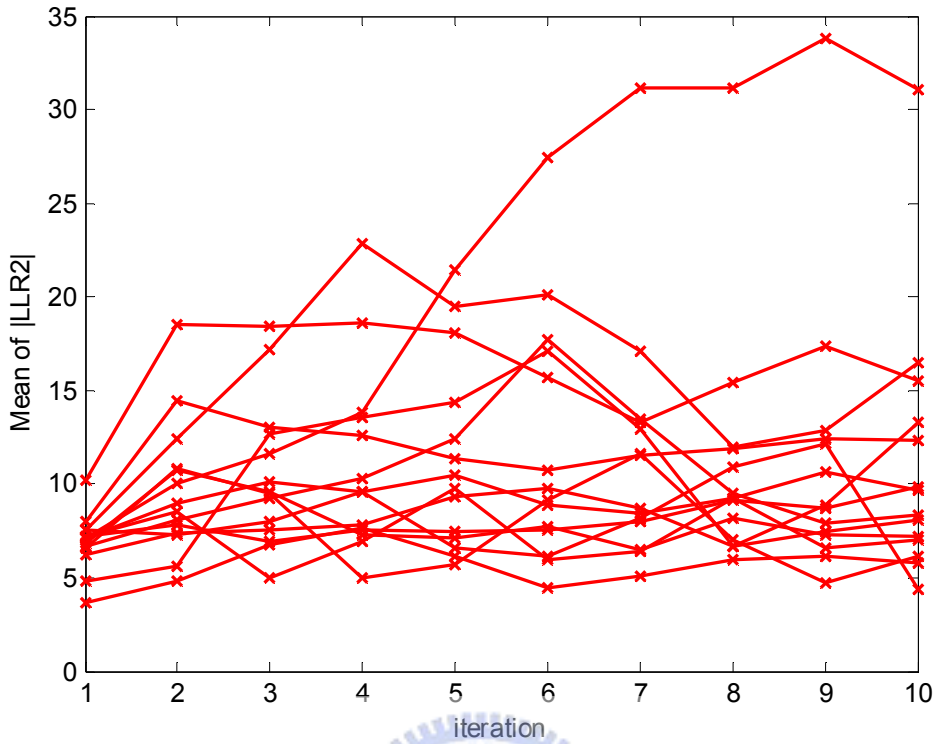

圖 62 解碼失敗時 $E|LLR^2|$ 的趨勢

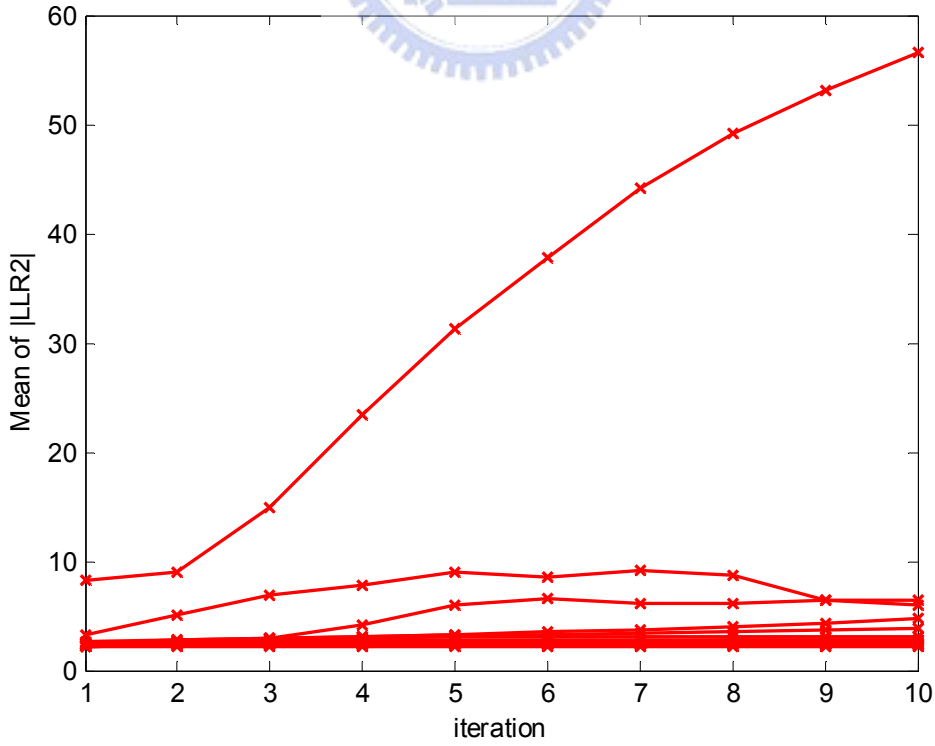

圖 63 解碼失敗時 $E|LLR^2|$ 的趨勢

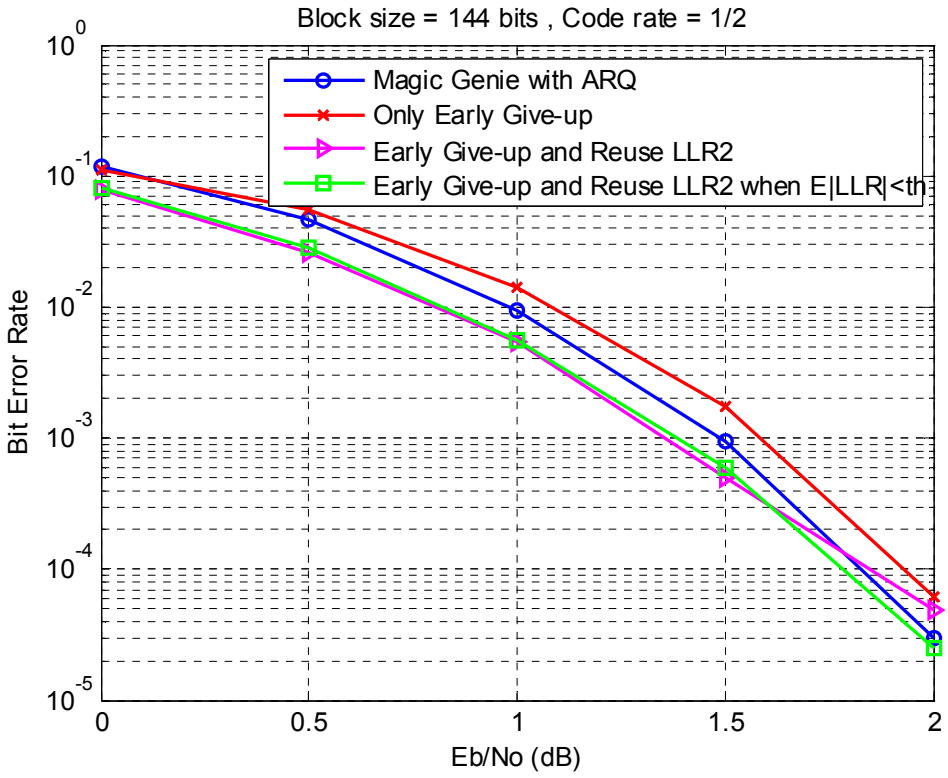

圖 64 只有在 $E|LLR^2|$ 小於預棄條件的臨界值時才使用狀態再利用機制

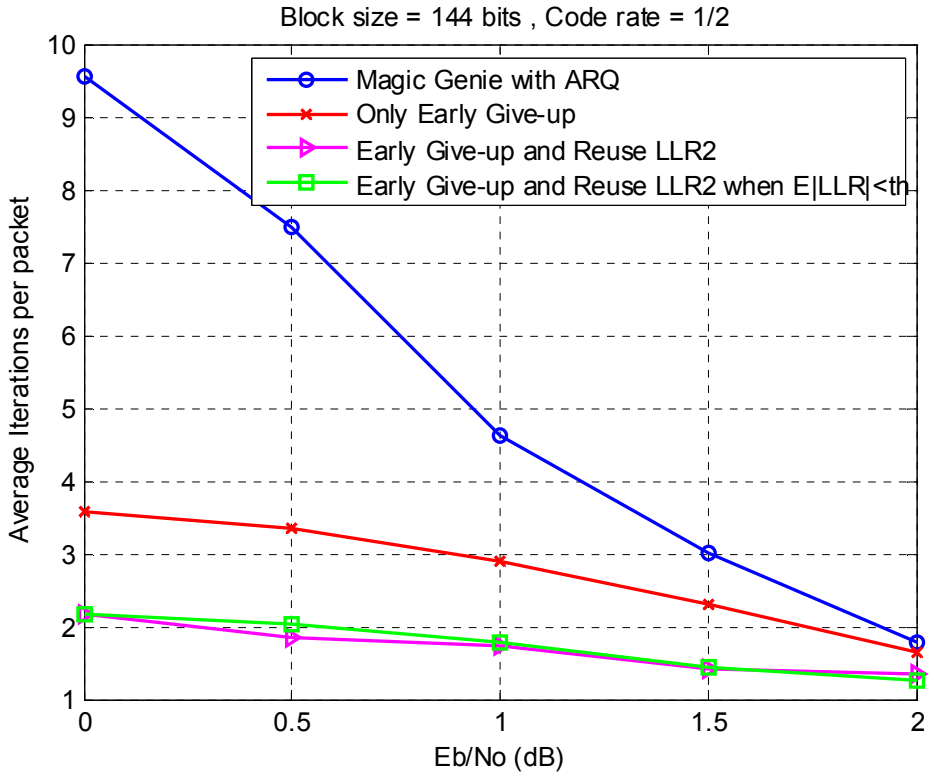

圖 65 只有在 $E|LLR^2|$ 小於預棄條件的臨界值時才使用狀態再利用機制

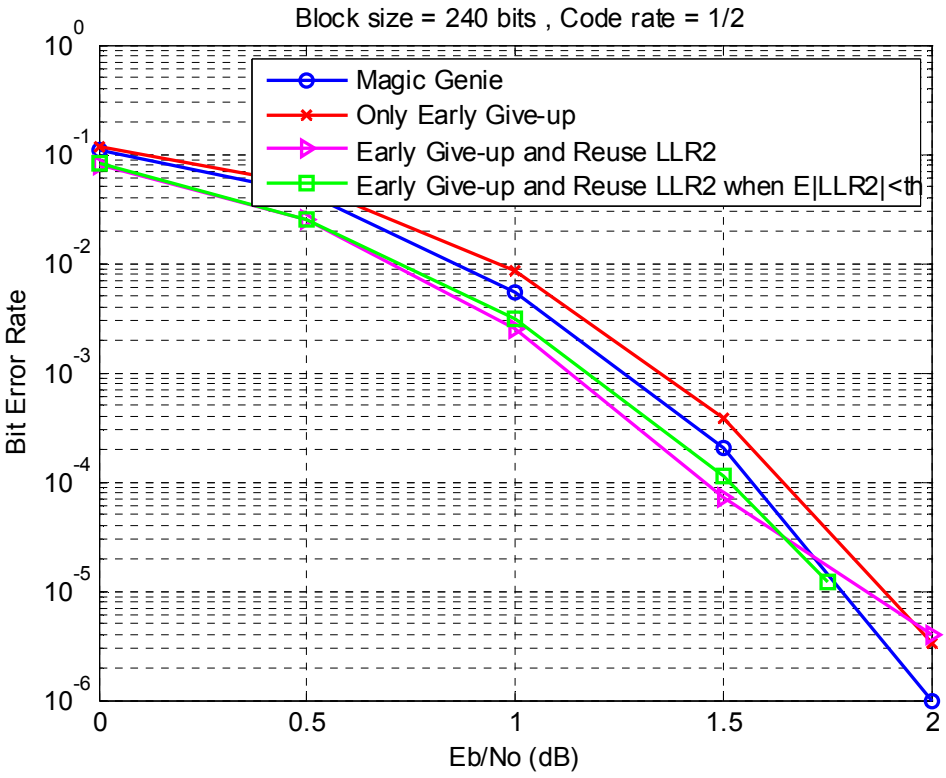

圖 66 只有在 $E|LLR^2|$ 小於預棄條件的臨界值時才使用狀態再利用機制

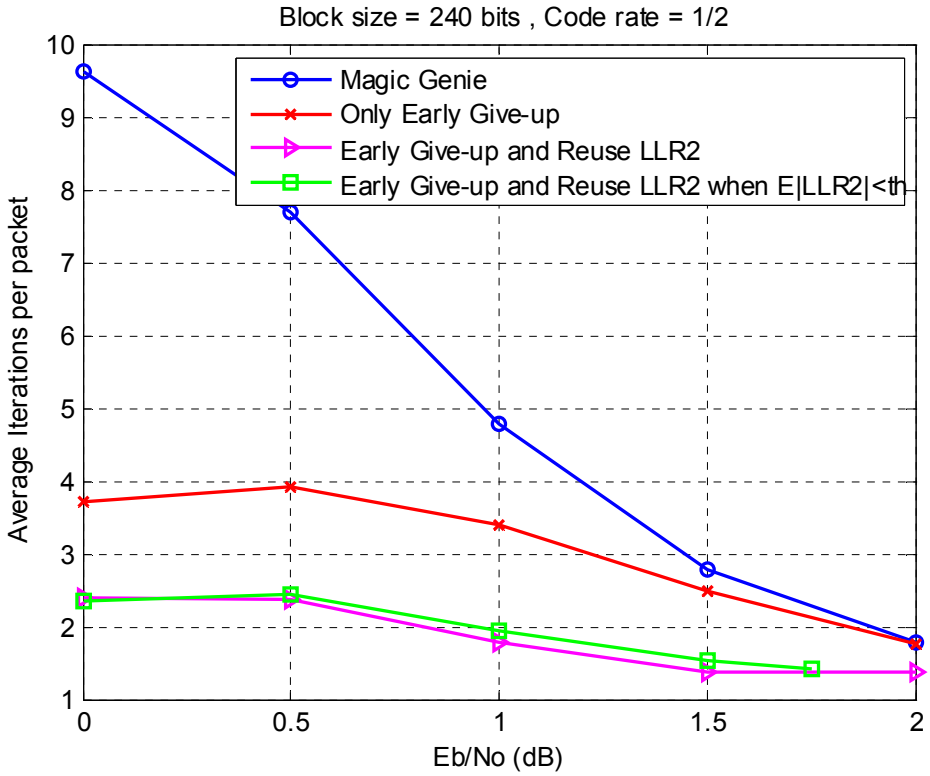

圖 67 只有在 $E|LLR^2|$ 小於預棄條件的臨界值時才使用狀態再利用機制

第四章 模擬結果比較

總結第三章的討論與分析，我們最終可以得到如下的兩種技術的結合能讓整體的效能最佳：

(1) 預棄技術：在每次遞迴後，觀察 $E|LLR^2|$ 是否有減少，若有且小於某個臨界值則立即放棄解碼。

(1) 狀態再利用技術：重傳收到新的資料後，只有在所留下來的 $|LLR^2|$ 的平均值小於預棄技術所設定的臨界值時，才將它設定成解碼初始值，否則一樣設定成 0。

以下圖 68~77 中，是針對四種方法所產生的不同的錯誤率以及平均的遞迴次數的分析，其中

(1) Magic Genie：

所指的是在模擬時，當此筆封包在我們所設定的最大遞迴次數(在此設定為 10 次)內能夠解碼成功，則會在成功的那次遞迴時停止，而所記入的遞迴次數便為所停下的那次遞迴決定，而當此筆資料在 10 次遞迴之後，解碼還是失敗的，則遞迴次數便記為 10 次，而決定封包是否解碼成功是利用在模擬方法時我們已經可以預知正確資料為何所以可以比對每次所解出來的資料和真正資料間的正確性，所以此方法稱為 Magic Genie[7]，由以上解釋可知，Magic Genie 方法所得到的遞迴次數對於解碼能夠成功的封包而言為真正所須的最小遞迴數，所以 Magic Genie 方法可以說是 early termination 方法的最佳解，除此之外，Magic Genie 方法是除非封包已經解碼成功否則不會停止，所以這個方法所算出的錯誤率代表解碼錯誤率的最低極限。

(2) Give-up when fall once：

所指的是使用[25]中或是 3-2-1 小節所提到的預棄技術的方法，此方法在發現 E|LLR|小於上次遞迴的值時，便視此封包無法正確解碼，於是立即停止解碼，所以若解碼失敗時所記入的解碼次數便為放棄解碼的那次遞迴數目，而非最大的解碼次數(10 次)。而當解碼資料在某次遞迴後，CRC 便解碼正確時，系統便視這筆資料已經沒有錯誤，而停止解碼動作，所以所記入的遞迴次數便是 CRC 解碼正確的那次，而此方法會誤判一些原本能夠正確解碼的封包，所以在訊雜比較高時，這些誤判的封包所造成的錯誤率上升的影響便漸漸反應出來。

(3) Give-up when fall and E|LLR|<th：

所指的是我們所修正的預棄技術，也就是在判斷 E|LLR|是否下降後，必須符合臨界值的要求才會真正放棄解碼，所以方法的所得到的平均遞迴次數一定會比[25]中來的多一些。

(4) Early Give-up lower bound

這條界線所指的是假如我們在收到資料後，便有種方法可以得知最後解碼結果會失敗，而我們就不做任何解碼，所以根本不需要遞迴次數，而在解碼能夠成功時，所須的遞迴次數便和 Magic Genie 一樣，也就是說這條 lower bound 所代表的是預棄技術的最佳效能。

以下便依照各種情況下位元錯誤率以及每個封包平均解碼的次數做模擬與討論。

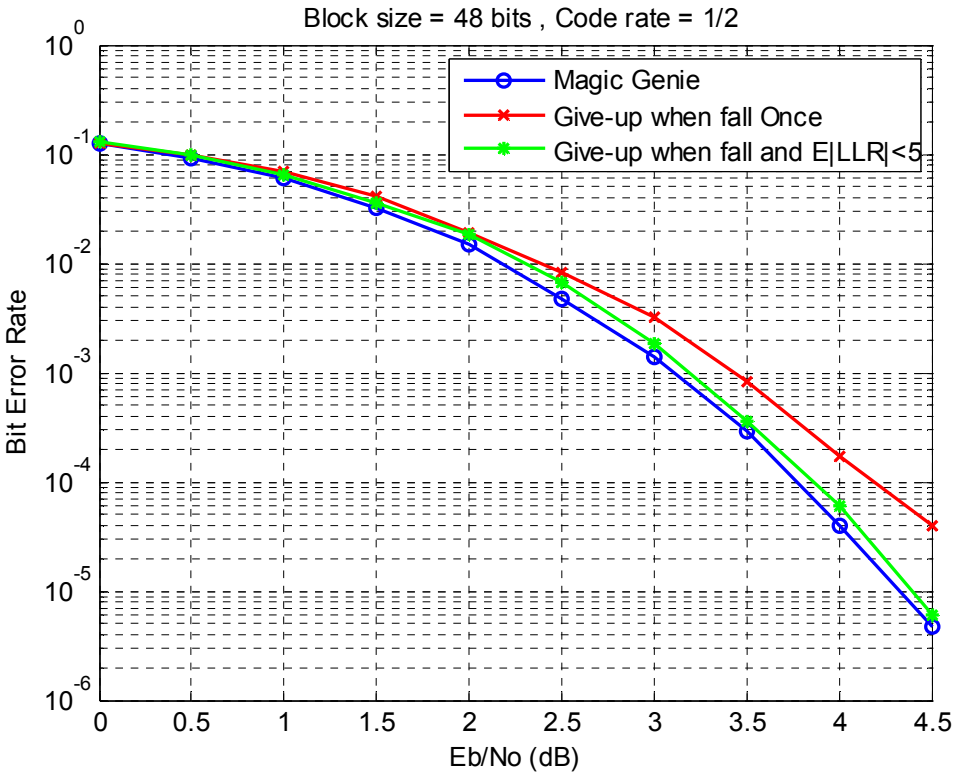

圖 68 位元錯誤率

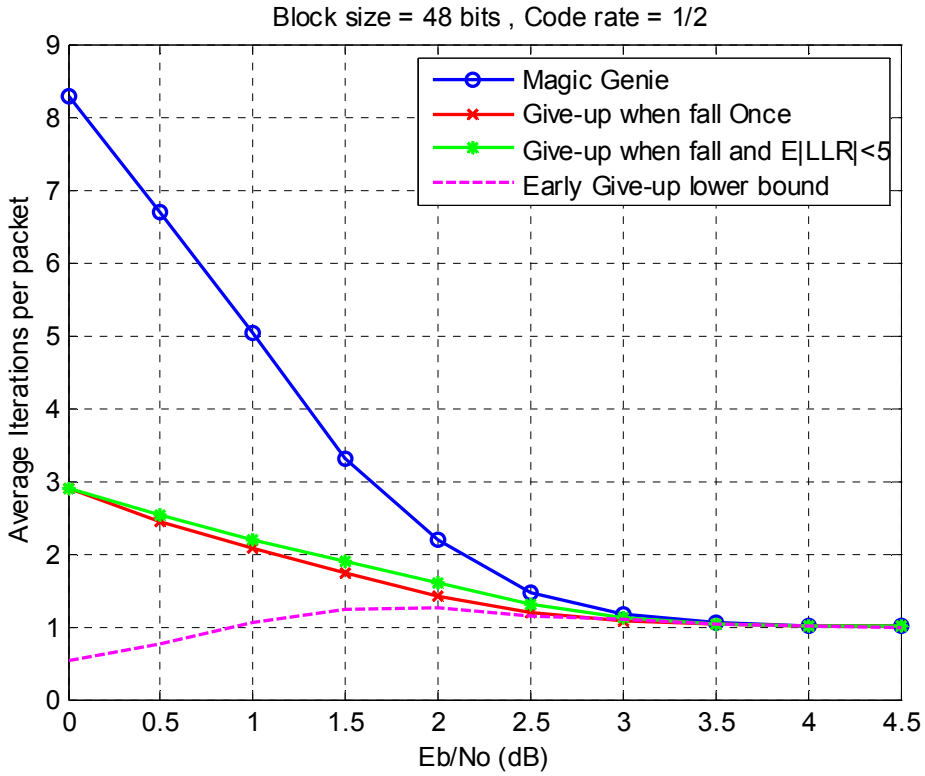

圖 69 平均遞迴次數

在圖 68，圖 69 中，我們看到在區塊大小為 48 bits 如此短的封包下，在修正了預棄技術的放棄解碼的條件以後，能夠在最多約增加 0.2 次的遞迴次數的情況下，讓位元錯誤率在 4×10^{-5} 時約改善了 0.4dB，然而在越低的錯誤率下，這之間的差距應該又越大。

另一方面，我們看到我們所提出的預棄技術在訊雜比越低時，它所使用的平均遞迴次數與預棄技術的最佳值間差距越大，這是因為在低訊雜比時，封包大部分是解碼失敗的，然而我們的預棄技術還是會試圖的讓部分的解不開的封包進行解碼。而實際上我們所用的預棄技術對於解碼會失敗的封包所需的遞迴次數的極限值為 2 次遞迴，這是因為必須判斷是否比上一次遞迴時來的小，所以最快發生放棄解碼的時間點便是在第二次遞迴時，所以此方法便至少需要兩次的遞迴次數。

然而在訊雜比越來越高時，真正解碼會失敗的封包越來越少，所以我們所使用的預棄技術所需的平均遞迴次數便與最佳值間的差距縮小了。根據圖 69 上的結果，我們所使用的預棄技術在 Eb/No=0dB 時約可省下 60%的遞迴次數，而錯誤率上在 BER=10^{-5} 時的損失最多大約在 0.1dB 左右。

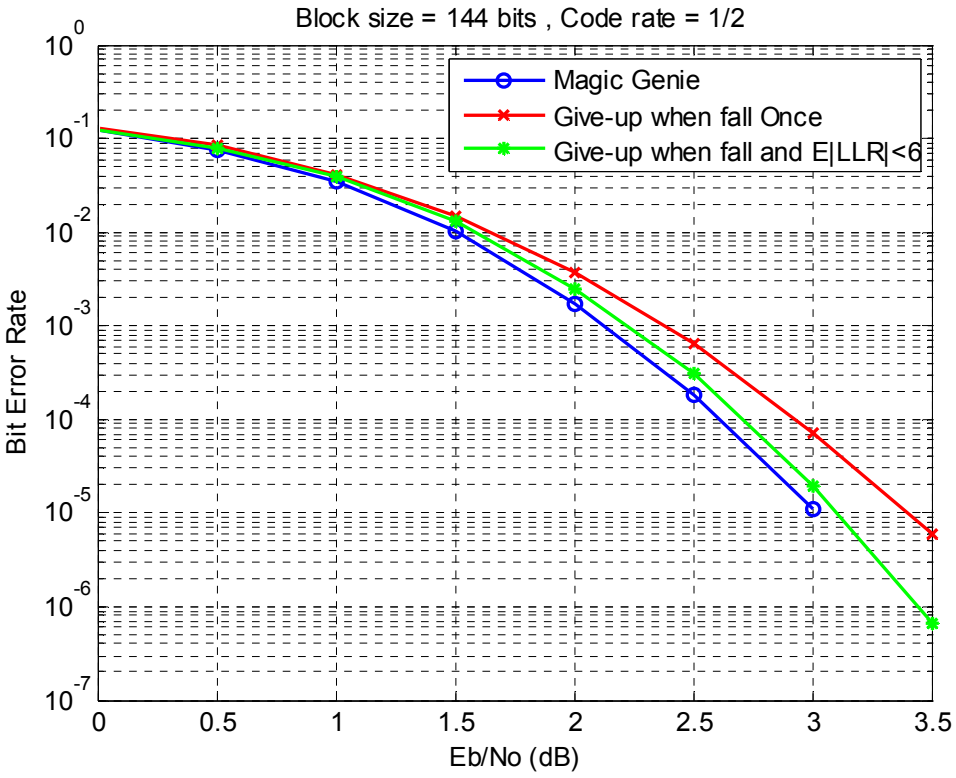

圖 70 效能比較圖

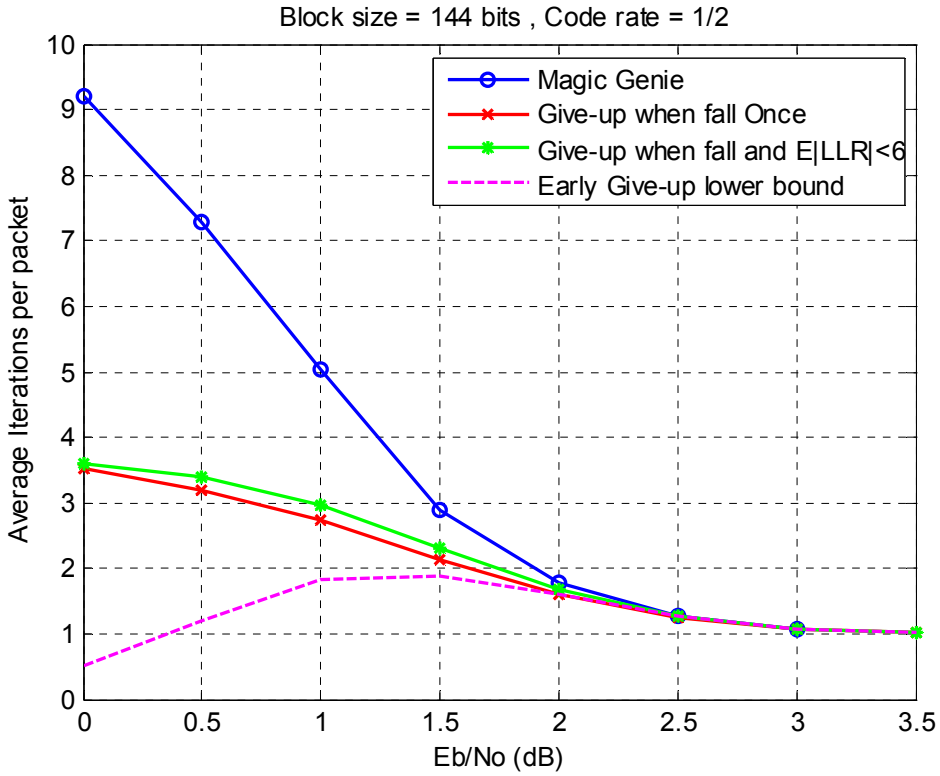

圖 71 效能比較圖

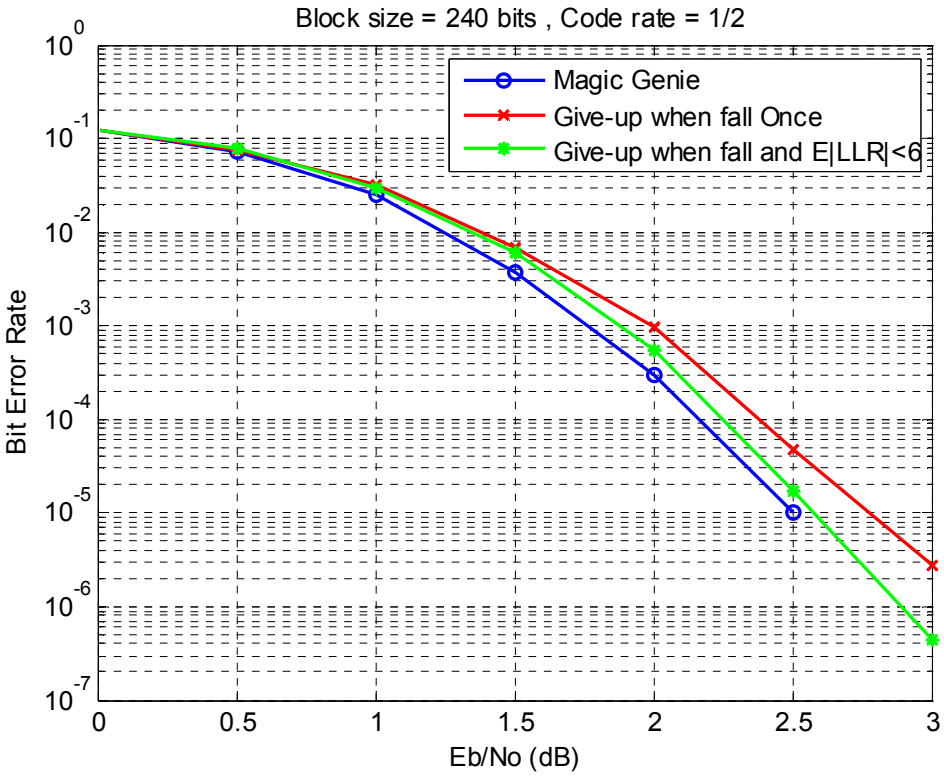

圖 72 效能比較圖

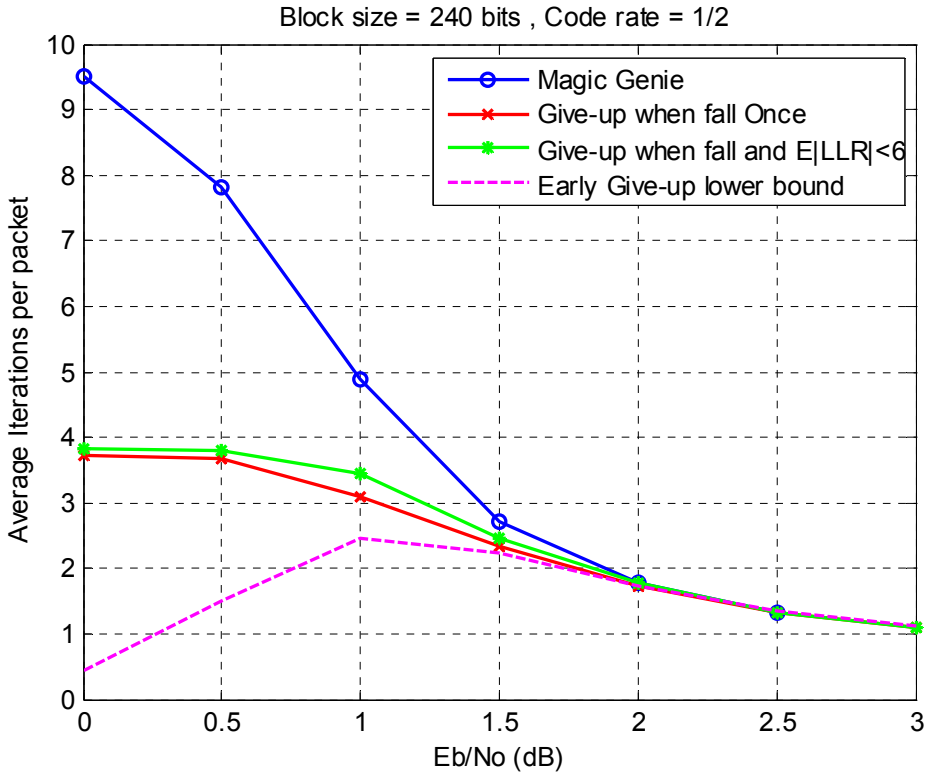

圖 73 效能比較圖

隨著資料區塊大小的增加，我們看到在圖 71 中，資料區塊大小由 48 bits 提升到 144 bits，而所能提供的 E|LLR|的樣本數便多了，所以[25]中的預棄方法所得到的錯誤率也能夠較準一些，所以我們所使用的預棄方法的錯誤率和他的方法的差距也就減少了一點，在位元錯誤率為 10^{-5} 時，我們所使用的預棄方法約可改善 0.25dB 左右，而平均遞迴次數最多約增加了 0.2 次左右。

而當資料區塊大小繼續增加到 240 bits 時(如圖 73)，我們發現錯誤率在 10^{-5} 時的改善又降到了 0.2dB 左右，所以資料區塊大小越長，[25]的方法能夠越準確的猜測出無法正確解碼的封包。

而再觀察我們所使用的預棄技術的臨界值，根據統計發現同樣碼率所使用的最佳臨界值的大小差不多(碼率=1/2→th=6)，但是對於太短的碼，如圖 68 中所示，臨界值設的比其他較長的碼來的小一些(碼率=1/2→th=5)才能夠較接近於我們所設定的錯誤率(與 magic genie 方法不超過 0.1dB)

而在圖 74~77 中，由於碼率的降低，所設定的最佳臨界值也如第三章的分析必須上升(碼率=3/4→th=9)，使得遞迴次數能夠壓低。另外我們看到，在碼率較低的情況下，我們所使用的預棄技術能夠相當的準確地估測會解碼失敗封包，所以在錯誤率上面我們發現和 magic genie 幾乎是貼在一起的，而遞迴次數也是最多大約增加了 0.2 次左右。

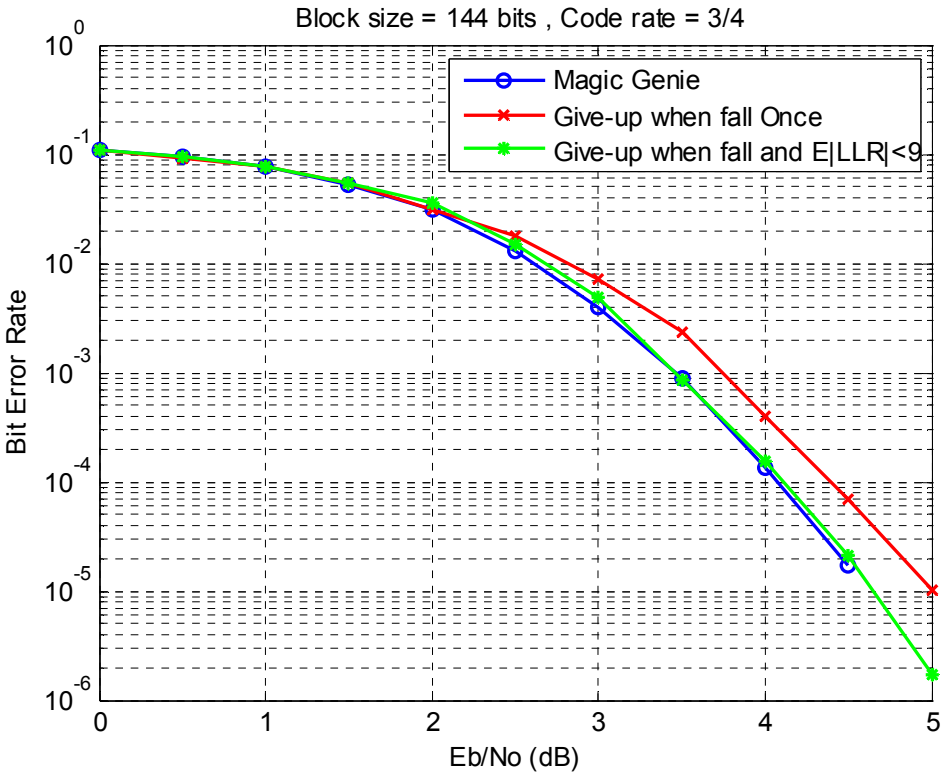

圖 74 效能比較圖

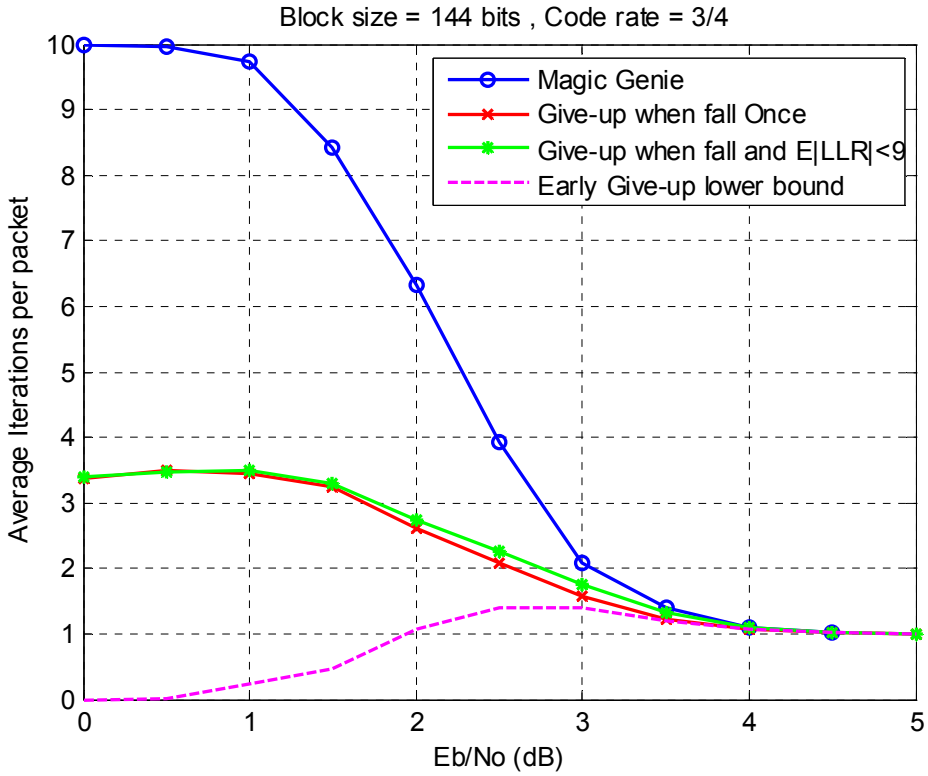

圖 75 效能比較圖

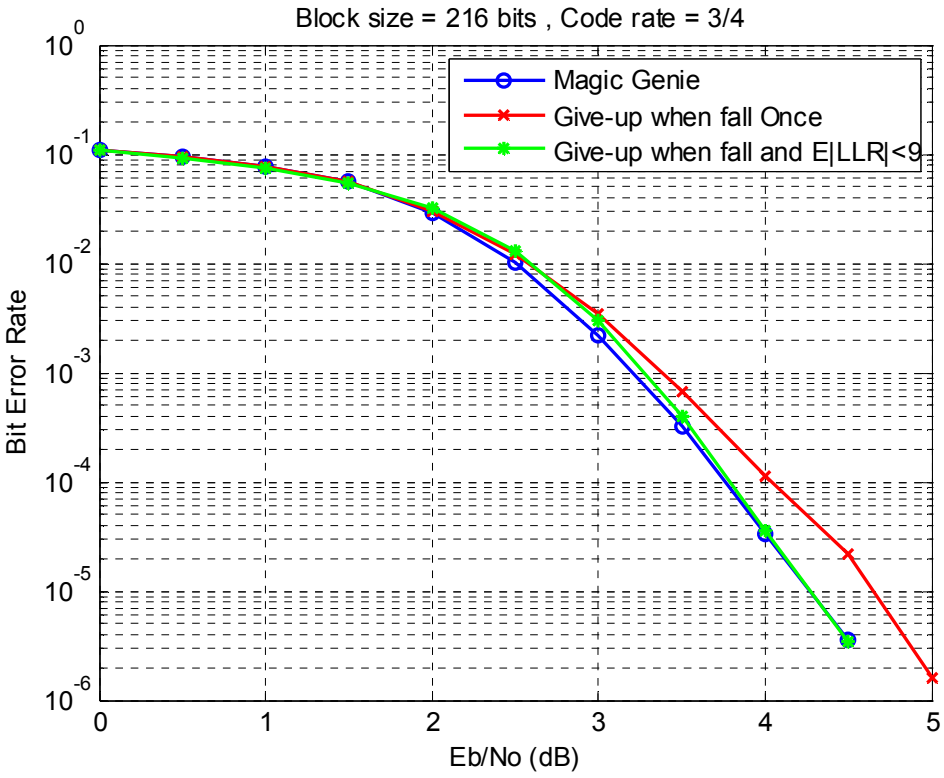

圖 76 效能比較圖

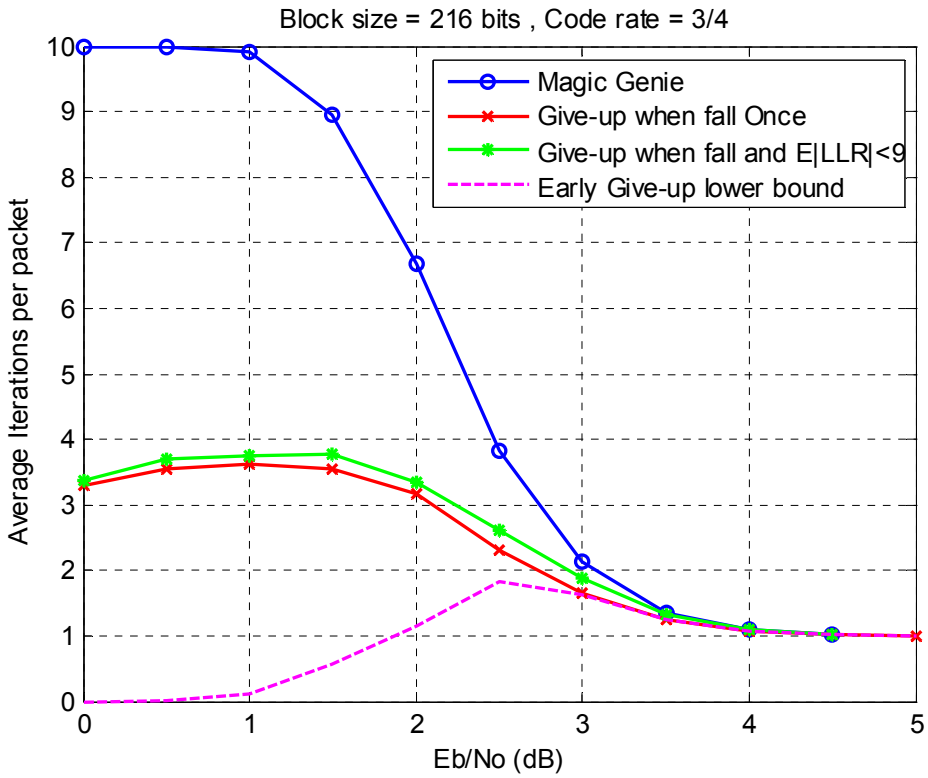

圖 77 效能比較圖

而接下來我們必須針對有重傳的情況下的解碼效能作分析，而我設定每一筆資料若解碼錯誤，最多能被重傳一次，在圖 78~87 的模擬中，”Magic Genie with ARQ”所指的是在每個不同資料的第一筆資料解碼時，我們使用 magic genie 的方法作解碼，若解碼失敗了，我們會傳重一筆一模一樣的資料再進行第二次的 magic genie 解碼，如果第二筆資料還是錯了，便統計剩下多少錯誤位元，而計算位元錯誤率便是利用第二次解碼還是失敗的錯誤位元數來計算。

而”Proposed Method”所指的是當在解碼每一個資料的第一筆資料時，使用了判斷未單調遞增且小於臨界值的預棄技術，而當解碼失敗時，我們會重傳封包，並且在解重傳封包時，若之前的失敗封包解碼後的 $E|LLR^2|$ 小於預棄條件的臨界值時，則使用 LLR^2 來做為新一筆資料的解碼初始值，同時第二筆封包一樣是有啟動預棄技術的。而以下的模擬中，位元錯誤率指的是只計算第二筆資料還錯誤的情形下的位元錯誤率。而在平均遞迴次數方面，這裡所指的是只有重傳後的封包解碼時所需要的遞迴次數，只統計這項值是因為我們想關心的是使用了狀態再利用的技術對於整體而言改進了多少，而若把第一筆和第二筆資料的解碼次數加在一起的話，便較看不出這項值的改善。

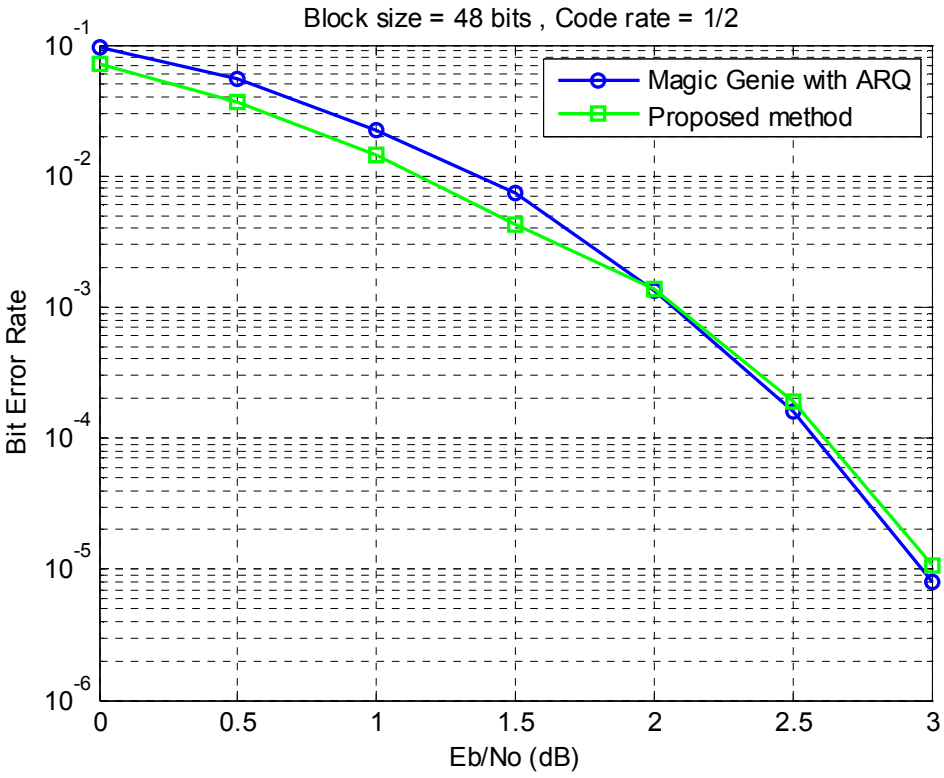

圖 78 效能比較圖

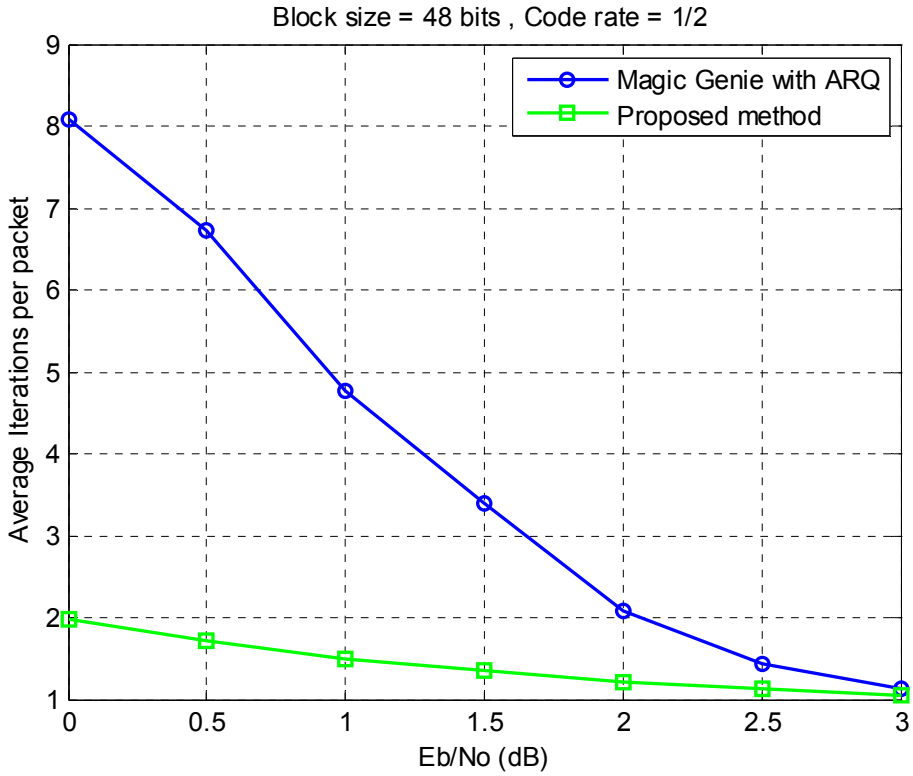

圖 79 效能比較圖

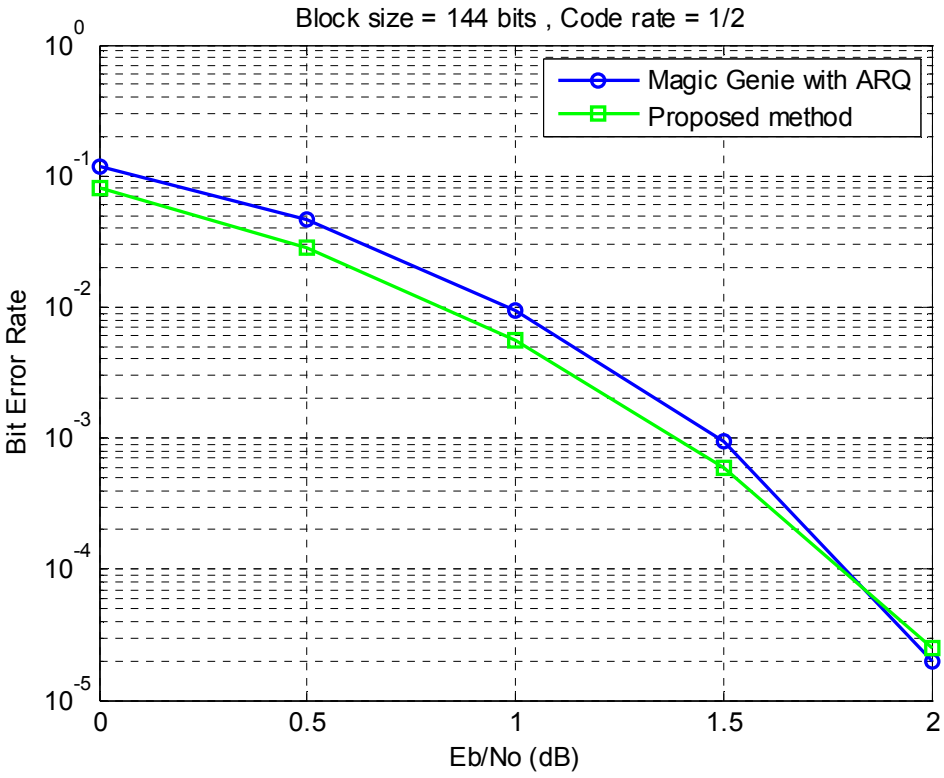

圖 80 效能比較圖

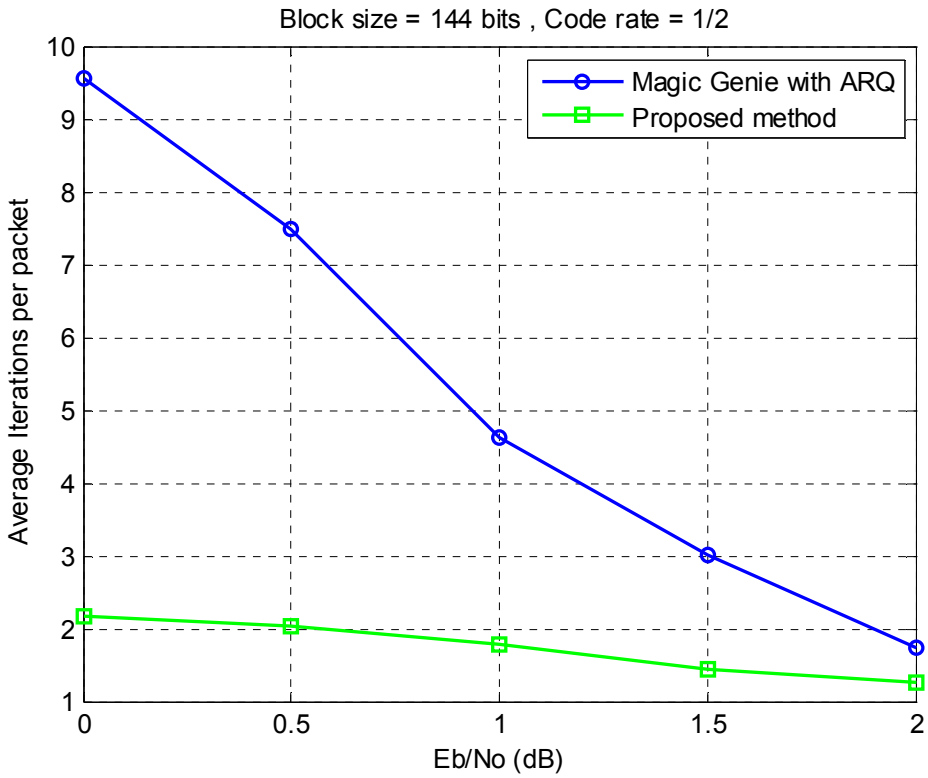

圖 81 效能比較圖

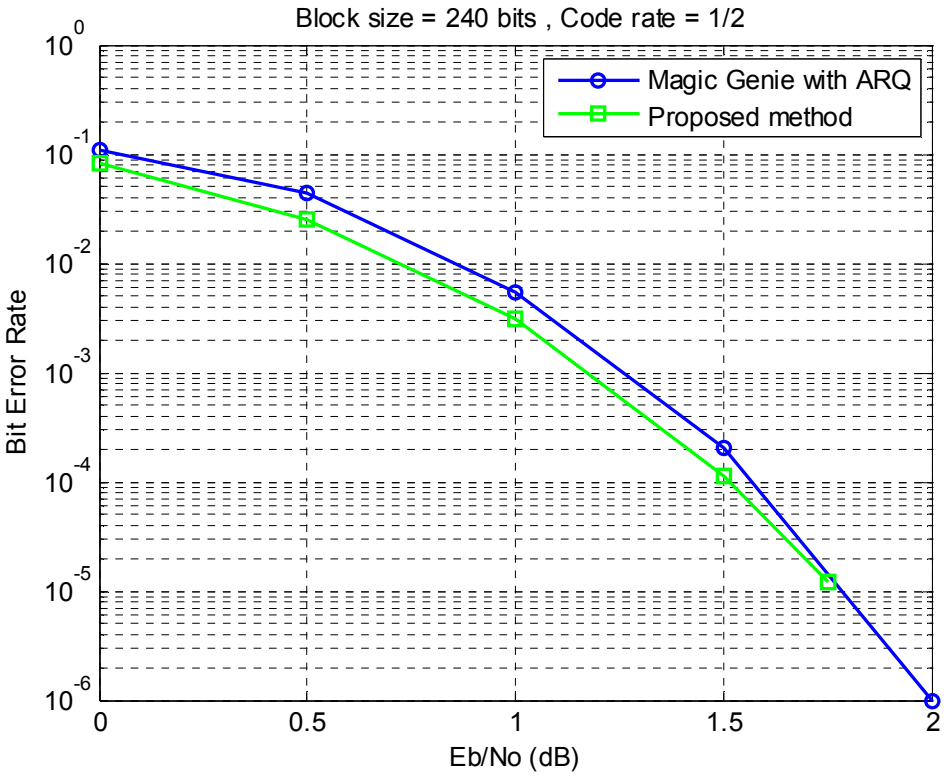

圖 82 效能比較圖

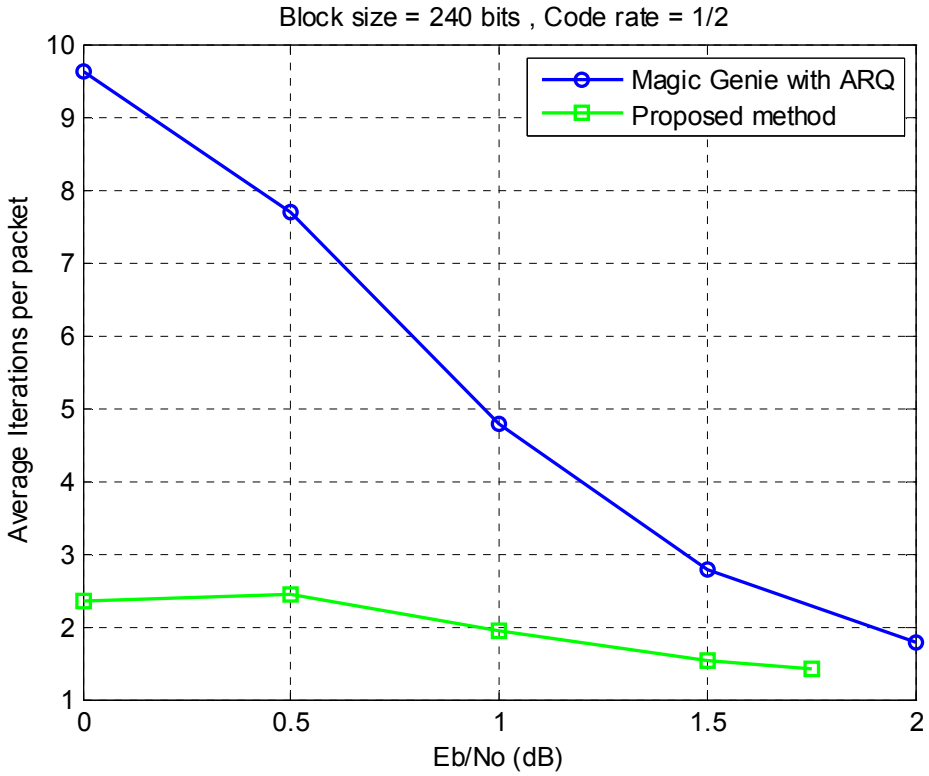

圖 83 效能比較圖

圖 78，圖 79 是根據資料區塊大小為 48 bits，碼率是 1/2 所做的模擬，在錯誤率方面，對於較低的訊雜比(Eb/No=0~2dB 間)，我們提出的方法甚至比有使用 ARQ 的 magic genie 機制來的好，但在訊雜比漸漸便高時(Eb/No>2dB 時)，改善的幅度便越來越小，甚至變差，這有可能是因為對於判斷是否要使用失敗封包的 LLR^2 來做為新一筆資料的解碼初始值時,所使用的臨界值可能也需要做大量的模擬分析來調整出一個對於這個問題的最佳臨界值,而或是實際上不能單純只由 $E|LLR^2|$ 的大小來做為是否能夠再利用的準則。

然而，觀察圖 78，圖 80，圖 82 可以發現兩條線在位元錯誤率上面所交叉的點，隨著資料區塊大小越來越長時，便越來越往右移，也就是兩者的差異越來越小。另外，在平均遞迴次數方面和圖 79，圖 81，圖 83 比較可以發現，重傳的這筆封包，由於有使用 LLR^2 來當作解碼初始值，可以省下的 70%做又的遞迴次數。

在圖 84，圖 86 中，可以看到對於碼率較低的碼而言，狀態再利用所得到的好處能夠大大的增加，甚至最多能改善 0.4dB 左右。而因為有使用狀態再利用技術，在重傳時所需要的平均解碼次數最多約可減少 80%左右。

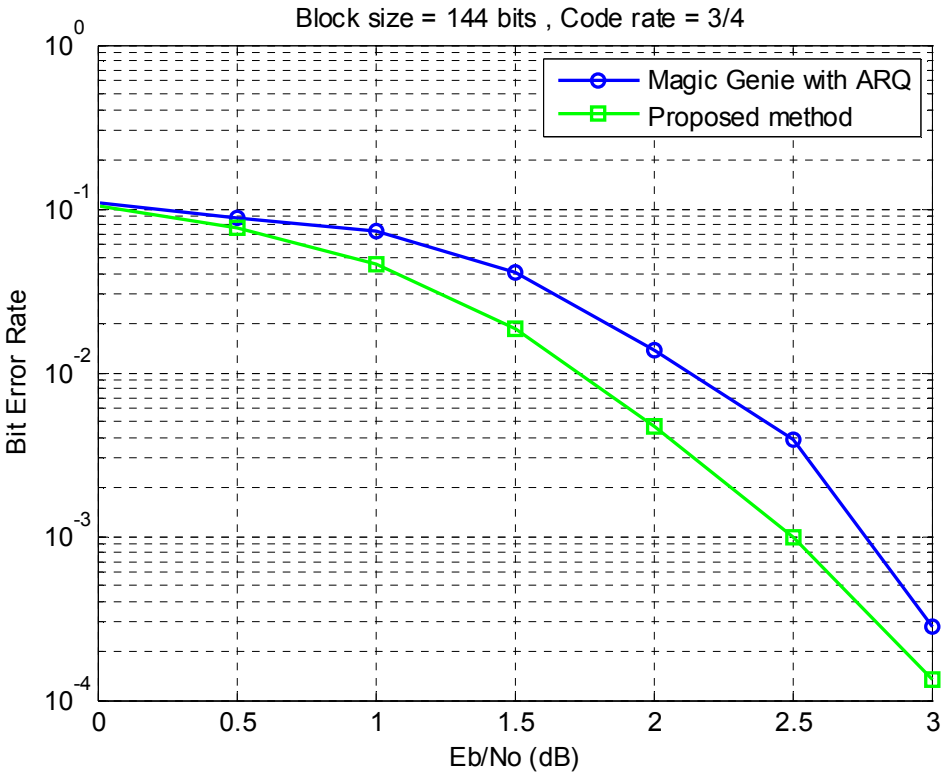

圖 84 效能比較圖

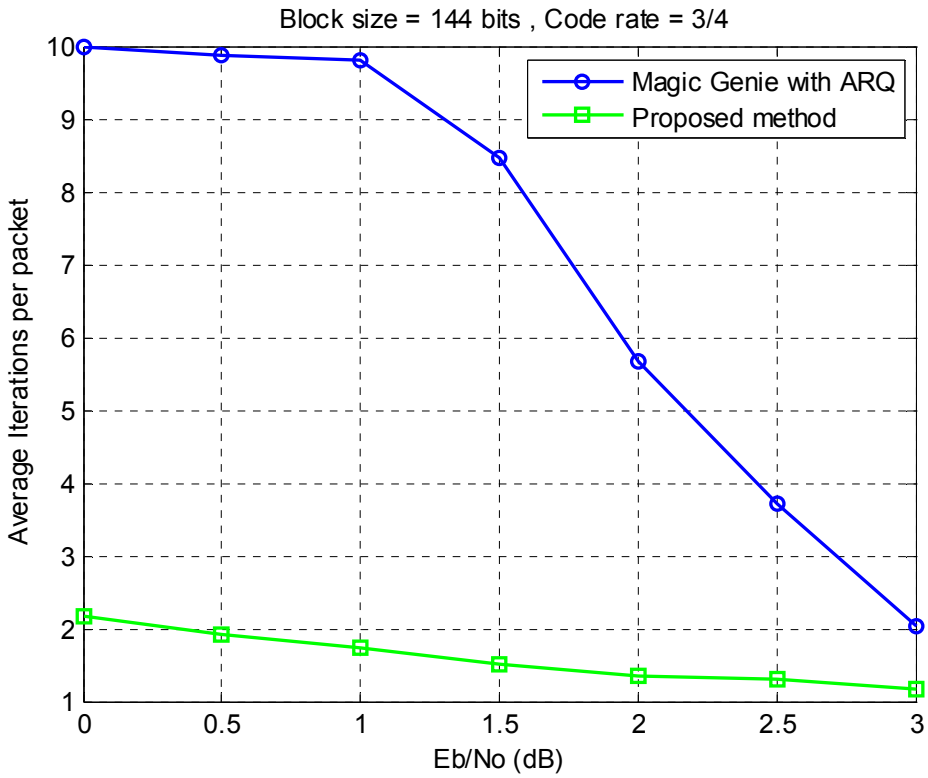

圖 85 效能比較圖

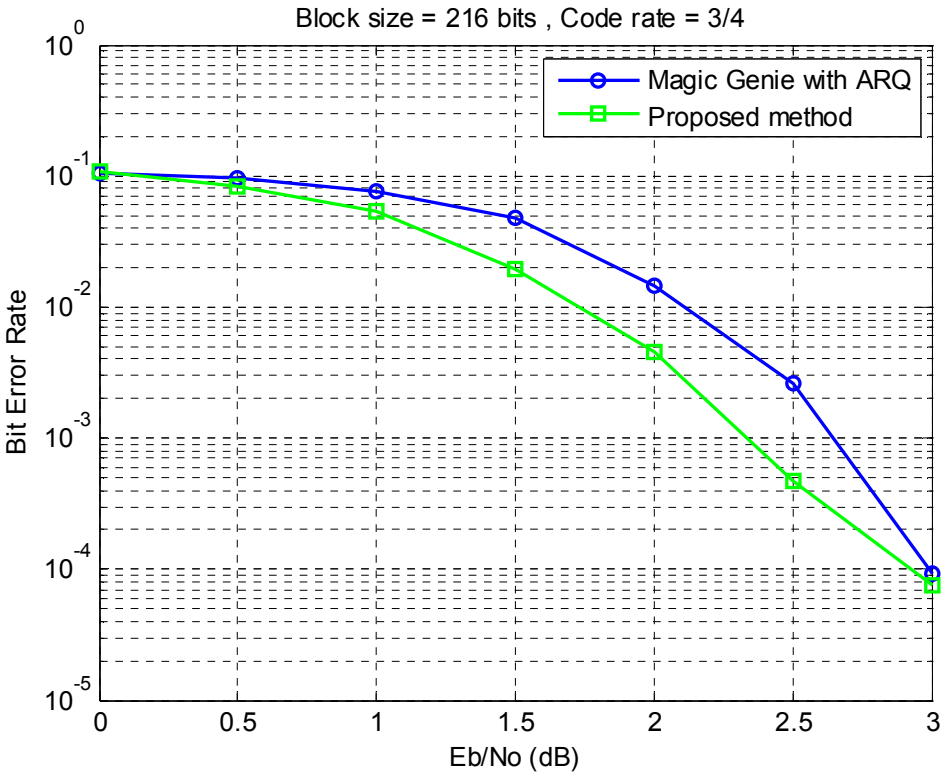

圖 86 效能比較圖

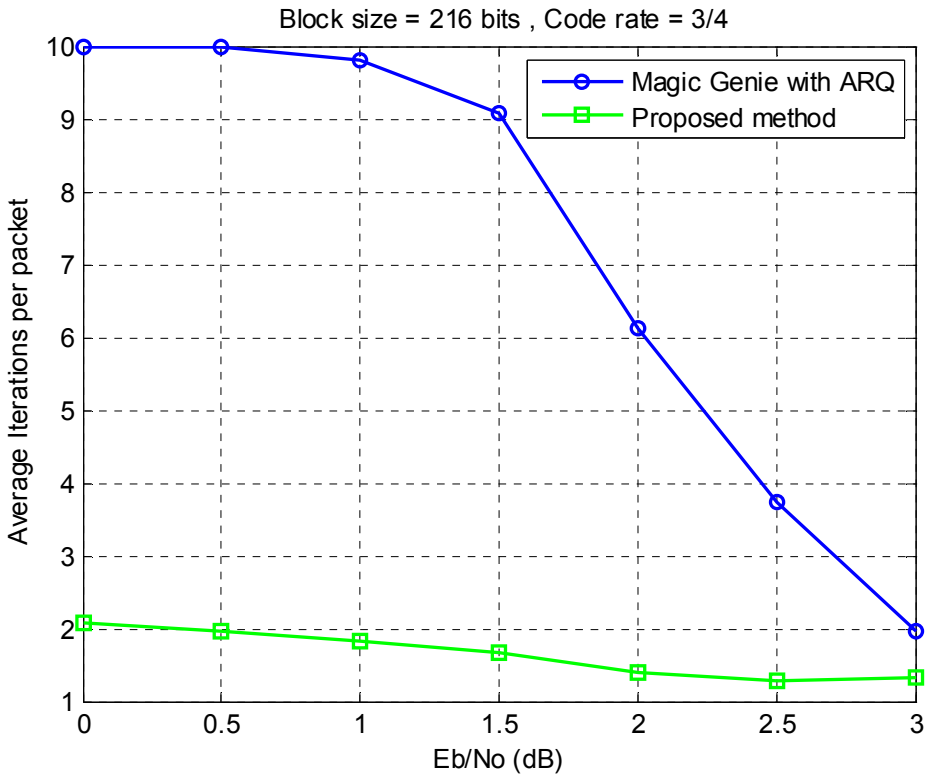

圖 87 效能比較圖

第五章 結論與未來展望

在渦輪解碼時，資料所經過的通道環境與其真正所需要的遞迴次數有很大的關係，當通道相當惡劣時，遞迴的次數便趨近於我們所預設的最大次數，然而可能最後解碼還是失敗的，所以根據這點我們利用觀察 $SISO_2$ 的 LLR 值的方法來估測封包是否有解碼失敗的可能，而提早放棄解碼節省無意義的遞迴。

另外基於解碼失敗而重新傳送的封包，我們能將它所留下來的 LLR^2 做為新的一筆資料解碼時的初始值(La^1)，而模擬證明結合了以上兩種辦法在加成性白色高斯雜訊的通道下，能夠在錯誤率幾乎沒有損失的前提下，省下可觀的遞迴次數。

而在 IEEE 802.16 的規範下，能夠支援混合式自動重傳請求(Hybrid Automatic Repeat Request)，而對於這種封包格式不固定的傳輸方式，預棄以及狀態再利用技術是否還能有效的發揮作用，另外如何結合不同子封包的資料，以達到最好的效能，相信是很值得探討的。

參考文獻

[1] C. Berrou, A. Glavieux and P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: Turbo-codes,” in Proc. ICC ‘93, Geneva, Switzerland, pp. 1064－1070, May 1993.

[2] L. Bahl, J. Cocke, F. Jelinek and J. Raviv, “Optimal decoding of linear codes for minimizing symbol error rate,” IEEE Trans. on Information Theory, vol. 20, pp. 284-287, May 1974.

[3] Alexandre Giulietti, Bruno Bougard and Liesbet Van der Perre , Turbo Codes Desirable and Designable , Kluwer Academic Publishers , 2004.

[4] P. Robertson, E. Villebrun and P. Hoeher, “A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain,” in Proc. IEEE Int. Conf. Communications (ICC ‘95), pp. 1009-1013., 1995.

[5] Nam Yul Yu, Min Goo Kim, Yong Serk Kim and Sang Uoon Chung , “Efficient stopping criterion for iterative decoding of turbo codes,” ELECTRONKS LETTERS 9th , Vol.39 , No.1 , pp. 73-75, January 2003.

[6] Wei Jiang and Daoben Li, “Two Efficient Stopping Criteria for Iterative Decoding,” Communications and Networking in China. ChinaCom’06. First International Conference, pp.1-4, 2006.

[7] A. Matache, S. Dolinar and F. Pollara, “Stopping Rules for Turbo Decoders,” TMO Progress Report, Aug. 15, 2000. http://tmo.jpl.nasa.gov/tmo/progress_report

[8] A. Shibutani, H. Suda and F. Adachi, “Reducing average number of turbo decoding iterations,” IEE Electronic Letters, vol. 35, pp. 701-702, Apr. 1999.

[9] Shu Lin and Daniel J. Costello, Error Control Coding , Prentice-Hall ,

pp. 1174-1197, 2004.

[10] D. Chase, "Code combining--A maximum-likelihood decoding approach for combining an arbitrary number of noisy packets," IEEE Trans. Commun., vol. COM-33, pp. 385-393, May 1985.

[11] Xiu Chundi, Li Yonghui and Fan Yuezu, "A hybrid ARQ scheme using RCPT codes and its performances over rayleigh fading channel," Proc of IEEE Globecom 2002, Vol.2, pp. 1517-1521, Nov 17-21, 2002, Taiwan.

[12] Y. Wang and S. Lin, "A modified selective-repeat type-II hybrid-ARQ system and its performance analysis," IEEE Trans. Communications, Vol. COM-31, pp. 593-607, May 1983.

[13] A. Dholakia, M. A. Vouk, and D. L. Bitzer, "A variable-redundancy hybrid ARQ scheme using invertible convolutional codes," in IEEE 44th Vehicular Technology Conference (VTC '94), vol. 3, pp. 1417-1420, June 1994.

[14] M. Wissem, B. Hatem and S. Mohamed, "Performance comparison of type Ⅰ, Ⅱ and Ⅲ hybrid ARQ schemes over AWGN channels," IEEE International Conference on Industrial Technology, pp.1417-1421 ,2004.

[15] IEEE 802.16TM-2004, IEEE Standard for Local and Metropolitan Area Networks, Part 16: Air interface for fixed broadband wireless access systems, October 2004.

[16] J.B. Anderson and S.M. Hladik, "Tail-biting MAP Decoders," IEEE Journal in Selected Areas in Communication, VOL. 16, NO. 2, pp.297-302, February 1998.

[17] H. H. Ma and J. K. Wolf, "On tail-biting convolutional codes," IEEE

Trans. Commun., vol. 34, pp. 104－111, Feb. 1986.

[18] S.B. Im, M.G. Kim and H.J. Choi , “An Efficient Tail-biting MAP Decoder for Convolutional Turbo Codes in OFDM Systems,” IEEE Region 10 Conference, TENCON 2004 Volume B, 21-24, Vol. 2, pp.589 -592, Nov. 2004.

[19] Y. Wu, B. D. Woerner and W. J. Ebel, “A Simple Stopping Criterion for Turbo Decoding,” IEEE Communications Letters, vol. 4, pp. 258－260, Aug. 2000.

[20] TODD K.MOON, Error Correction Coding ,WILEY , pp.147-150, 2005.

[21] Y. Wu, B. D. Woerner and W. J. Ebel, "A simple. stopping criterion for turbo decoding", IEEE Commun. Lett. vol. 4, no. 8, pp. 258-260, Aug. 2000.

[22] K. R. Narayanan and G. L. Stuber, “A novel ARQ technique using the turbo coding principle,” IEEE Communication. Letter, Vol.1, pp.49-51, Mar.1997

[23] P. Salmela, T. Järvinen, T. Sipilä and J. Takala, "On allocation of turbo decoder iterations," in proceedings of the 14th IEEE 2003 International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC 2003), Beijing, China, 7-10 , pp. 157-160, Sep. 2003.

[24] R. Y. Shao, S. Lin and Marc.P.C. Fossorier, “Two simple stopping criteria for turbo decoding,” IEEE Trans. Communications, vol. 47, pp. 1117－1120, Aug.1999.

[25] 連樹德，「利用預棄技術和狀態再利用實現低功率渦輪解碼器」，國立交通大學，碩士論文，民國94年。