以 Radix-4 為基礎之 SOVA 解碼演算法

Radix-4 SOVA Based Decoding Algorithm

洪偉文　林義楠

Wei-Wen Hung　Yi-Nan Lin

摘要

渦輪碼在 1993 年被 Berrou 等人提出後，由於其優異的改錯能力，目前已被廣泛的應用到通訊系統中，如第三代行動通訊系統 (3G)、太空通訊…等。渦輪碼之傳統(MAP)解碼器在複雜度及時間需求均較其他系統高，因此近來軟式輸出維特比演算法(SOVA)被提出來，它可以達到降低其系統複雜度及維持高解碼效益的好處。本論文提出一個改良式的 SOVA 演算法來進行解碼，使用 Radix-4 的結構來替代原本的 Radix-2，利用此一架構的好處在於計算 ML 最佳的路徑時，可以節省大約一半的路徑存活所需記憶體的儲存空間及運算時之延遲時間。在節省所需之記憶體空間及解碼所耗費之延遲時間下，其相對的解碼效益在位元錯誤率(BER)為 10^{-4} 下，約只下降了 0.4dB。

關鍵詞：SOVA, turbo codes, radix-4

ABSTRACT

The coding technique of turbo codes was proposed by Berrou et al., in 1993. Since its excellent performance, the channel coding has been widely used in error control for many communication applications, such as third-generation (3G) mobile radio systems and deep space communications. A conventional (MAP) turbo decoder requires much more decoding complexity and time consumption than those decoders of other coding scheme. Recently, the soft-output Viterbi algorithm (SOVA) for decoding a code has been introduced and applied to iterative turbo decoding system. Owing to the properties of low system complexity and high decoding speed, it has been recommended to decode the turbo code for a portable radio communication system. In conventional SOVA-based turbo decoder, it has been implemented with a radix-2 trellis. The throughputs of SOVA decoders have traditionally been limited by the implementation of the radix-2 structure due to a one step recursion that processes one bit per clock cycle. In this paper, we try to explore an architecture based on radix-4 trellis SOVA decoder. The simulation results show that the proposed architecture can achieve about halved time latency for computing metrics and up to near twice times symbol throughput than that of radix-2 structure. In coding gain performance aspect, it has a slight degradation of coding gain about 0.4 dB than a conventional SOVA at 10^{-4} BER; nevertheless it is suitable to be applied to decode turbo codes for a mobile phone.

Keywords： SOVA, turbo codes, radix-4

1. 前言

由於渦輪碼的優異改錯能力，使得許多學者都致力於研究其解碼演算法，而一般現在常看到的演算法，大概分成兩類，一類是 MAP，一類則

洪偉文 明志科技大學電子工程系教授
林義楠 明志科技大學電子工程系講師

是 SOVA，雖然 SOVA 沒有 MAP 那麼好的改錯能力但是因為低運算複雜度，所以依然獲得很多學者的青睞。本論文提出一個改良式的 SOVA 演算法來進行解碼，利用 Radix-4(R4)的結構來替代傳統之 Radix-2(R2)SOVA，此架構(R4)是在每二個時間位元之每個狀態下計算出 1 個前向的對數機率累積值(Forward Metric) ，如此即可省去在 R2 之結構下每一個時間位元之每個狀態下均需計算出 1 個前向對數機率累積值(forward metric)的運算量，因此可比傳統之 SOVA(R2)具有較少的狀態推移之對數機率累積的運算量。在格子圖狀態縮減為原架構之 1/2 的情況下，我們在計算推估 ML 時，對於存活路徑及決策度量差等最佳路徑資訊的儲存亦可節省到一半的記憶體空間，因此，亦可提升 2 倍的解碼速率。本文將會分成下列幾個章節，第二章將會簡單介紹傳統的 R2-SOVA，在第三章則是介紹 R4 的 SOVA，第四章會做個比較並且秀出模擬的結果，最後做個總結。

2. SOVA[1][2][5]

它是由傳統的維特比演算法推衍而來，亦能適用於渦輪碼的解碼系統裡，其在硬體實現上會比 MAP 的簡單，性能只是略差於 MAP 一些。

SOVA 比傳統的 VA 多了兩個修正的部份，一個是在格子圖選擇最有可能性的路徑時，路徑度量的計算增加了事先資訊項（Priori Information）的考慮，另一個是對於每一個解碼位元提供一個是事後 LLR(Postiori Loglikehood Ratio)的軟式輸出值(Soft Value)。

2.1 度量值的計算

$$M(\underline{S}_k^s) = M(\underline{S}_{k-1}^{s'}) + \frac{1}{2}u_k L(u_k) + \frac{L_c}{2}\sum_{l=1}^{n} y_{kl}x_{kl} \quad (1)$$

$M(\underline{S}_k^s)$ 定義為度量我們可以看到隨著時間的增加利用更新累加的方式計算出度量，其中(1)式右邊第一項 $M(\underline{S}_{k-1}^{s'})$ 表示時間 k-1 時 $\underline{S}_{k-1}$ = s’的度量，第三項 $\frac{L_c}{2}\sum_{l=1}^{n} y_{kl}x_{kl}$ 為其關連性項的累加與通道可靠度值乘積， L_c 是通道特性，第二項 $\frac{1}{2}u_k L(u_k)$ 是事前資訊項， u_k 取決於此推移的系統性項 $x_{kl}(x_{kl} = u_{kl})$ ，由於這項為事先提供，對於欲解碼位元 u_k 為+1 或是-1 的機率，其可信任的程度之資訊，我們亦稱之為來源可靠度，我們將第二項跟第三項合併起來用一個 $\ln(\gamma_k(s',s)) \equiv \Gamma_k(s',s)$ 來表示。

此度量的計算比維特比演算法的度量之算法，多考慮一個事前資訊項 $\frac{1}{2}u_k L(u_k)$ ，這是第一個修正的部分。

2.2 度量差的計算

再來是 SOVA 對於每個解碼位元，如何能提供一個事後對數似然率(LLR)的軟式輸出值。假設在二元的格子圖中，我們依據(1)式算出每一條分支度量，而每個狀態點會有兩條分之度量合併進來，而分支度量較高 $M(\underline{S}_k^s)$ 是為路徑 $\underline{S}_k^s$ 的，較小的那條 $M(\hat{\underline{S}}_k^s)$ 是為路徑 $\hat{\underline{S}}_k^s$ 的，我們選取較高的分支當作這個狀態點的度量，並且選路徑 $\underline{S}_k^s$ 為存活路徑；另一條為共存路徑，當時間 k 時於格子圖上的狀態 s，我們定義其度量差 Δ_k^s ，亦稱之為可靠度：

$$\ln[\frac{p(correct)}{1-p(correct)}] = \Delta_k^s = M(\underline{S}_k^s) - M(\hat{\underline{S}}_k^s) \geq 0 \quad (2)$$

則此度量差隱含著是否做了正確決策的可信任之程度的度量。

當在格子圖中找出最有可能性的路徑，我們便要依循著此路徑，對每一個位元取出其位元決策的可靠度之對數似然率，所以在找出最有可能性的路徑後，並在時間點 k ~ k + δ 間之最大似然路徑與所有交錯之共存路徑間的度量差必須先計算出來，接著再做一個追溯的步驟。

2.3 對數似然率的計算跟定義

接著我們就來討論追溯的步驟， 假設 l 表示相對時間 k，往後其為第幾條共存路徑，l = k ~ k + δ。從時間點 l 時的共存路徑往回追溯到時間 k 時的狀態，記錄其促成其從前一狀態推移現在的狀態的位元 u_k^l ，我們將每一條追溯回來的 u_k^l 和最有可能性的路徑之位元 u_k 比較，若有不同則紀錄 e_{k-1}^k=-1，此時 e_{k-1}^k 的對數似然率值為最有可能性的路徑上時間 l 時的度量差值。若相同則記錄 e_{k-1}^k=+1， e_{k-1}^k 的對數似然率值為無窮大，其中 e_{k-1}^k：從時間 l 時的共存路徑追溯到時間 k，其位元與最有可能性的路徑之位元 u_k 的相似情況。

$$L(e_k^l)=\ln\left[\frac{p(e_k^l=+1)}{p(e_k^l=-1)}\right]=\begin{cases}\infty & u_k^l=u_k\\ \Delta_k^s & u_k^l\neq u_k\end{cases} \qquad (3)$$

那麼將所有從共存路徑回溯回來產生錯誤的情況加起來：

$$e_k=\bigoplus_{l=0}^{\delta} e_k^l \qquad (4)$$

那麼對於我們所要決策的位元 u_k ，其對數似然率就等於 u_k 乘於錯誤的 L-值

$$L(\underline{u}_k \mid y) = u_k * \bigoplus_{l=k}^{k+\delta} e_k^l$$

$$L(\underline{u}_k \mid y) = u_k * \bigoplus_{l=k}^{k+\delta} e_k^l$$

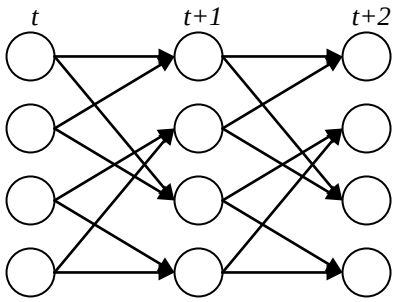
$$L(\underline{u}_k \mid y) \approx u_k * \underset{\substack{l=k\sim k+\delta \\ u_k\neq u_k^l}}{Min} \Delta_k^l \qquad (5)$$

(5)式便是事後 LLR 值，也是我們用來決策這個位元的可靠度的一個相當重要的依據，他是由 u_k 乘上時間從 k ~ k + δ 中 $U_k^l \neq U_k$ 所找出的度量差中最小的值，根據(2)式，我們可以得知度量差越小表示我們判斷錯誤的機率越大，所以當 $L(\underline{u}_k \mid y)$ 出來的值越大，表示我們判斷錯誤的可能也會縮小。

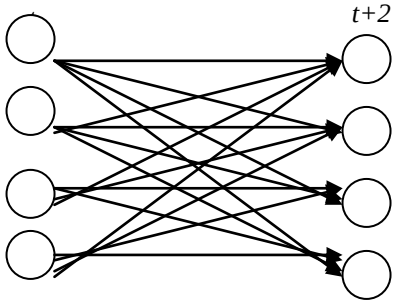
SOVA 其擁有目前所有解碼演算法中， 複雜度最低，相較於 Max-log-MAP 演算法約少將近一半，而其性能表現也不差，約只差 MAP 演算法約 0.6dB[5]。

3. Radix-4 SOVA[3][4]

本文提出一個新的架構來使得 SOVA 的運算量再降低，且在不影響錯誤率不大的情況下。傳統的 SOVA 在 M ＝ 2 時，利用的是 Radix-2 的架構(如圖一所示)來作運算，即每個狀態只會有 2 條路徑到達；而本文所提之 Radix-4 的架構(如圖二所示)，則是將兩個的時間點的 Radix-2 做合併的動作，因而會使得每一個時間點下之每一個狀態均會有 4 條路徑可以到達。雖然每一個計算分支機率累積值的複雜度會是傳統的兩倍，但每二個時間點卻可以少掉一個分支機率累積值的運算量。因此，在解碼的處理時間也會減少一半。



圖一 Radix-2



圖二 Radix-4

在計算 ML 路徑時，跟傳統 SOVA 無異，只是現在每個 State 將會有 4 條路徑同時進入，但是同樣的只選擇一條度量值最大的留下，當作我們的存活度量，以下是本文所提出 R4-SOVA 的演算法。

以 M = 2 為例(時間 k = 1, 2, 4, …, N), N：碼字位元的長度

Step 1：計算 Branch 與 State Metric 值

For k = 1 to N step 2

$$\Gamma_k(s',s) = \frac{1}{2}\sum_{l=1}^{2} u_{kl} L(u_{kl}) + \frac{L_c}{2}\sum_{l=1}^{2n} y_{kl}\cdot x_{kl},\ n=2$$

$$M(\underline{S}_k^i) = Max\begin{Bmatrix} M(\underline{S}_k^m) \equiv Max[M(\underline{S}_{k-1}^0), M(\underline{S}_{k-1}^2), \\ M(\underline{S}_k^n) \equiv Max[M(\underline{S}_{k-1}^1), M(\underline{S}_{k-1}^3) \end{Bmatrix}$$

End.

Step 2：建立 ML-path，藉由倒回追溯，找出最佳路徑 ML-path 並儲存相關的資訊、計算出決策度量差 Δ_k^s 、建立 1/2 的 5 倍限制長度之修改序列(Update Sequence;5δ)，並解出 t ~ t + l 時間點的位元碼。

For k = N to 1 step -2

ML-path-finding(k); //最佳路徑 ML-path

$\Delta_k^s = |M(\underline{S}_k^m) - M(\underline{S}_k^n)|$; //決策度量差

UpdateSquence(5δ); //5 倍限制長度修改序列

$$\overline{L}(\underline{u}_k \mid y) = \begin{bmatrix} u_{k1} & 0 \\ 0 & u_{k2} \end{bmatrix} \cdot \begin{bmatrix} \underset{l=k\sim k+\delta/2}{\underset{u_{k1}\neq u_{k1}^l}{Min}} \Delta_k^l \\ \underset{l=k\sim k+\delta/2}{\underset{u_{k2}\neq u_{k2}^l}{Min}} \Delta_k^l \end{bmatrix};$$ //解碼值

$$\overline{L}_e(u_k) = \overline{L}(\underline{u}_k \mid y) - \overline{L}(u_k) - L_c \bar{y}_{ks}$$;//額外資訊量

End.

Step 3：完成解碼器之解碼後，經由隨機亂數交錯器交錯 $\overline{L}_e$ 。

$\overline{L}(u_k) =$ Inter($\overline{L}_e$); //交錯後的事先額外資訊量

Step 4：goto Step 1., 進行解碼器 2 的解碼。反覆作疊代(Iteration)，一般設定疊代次數最大為 8 次，才停止解碼。

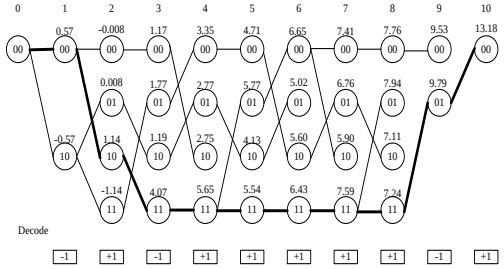
3.1 R2 與 R4-SOVA 的解碼實例[3][4]

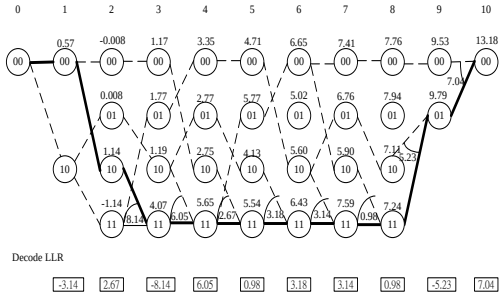
以下我們利用一個例子，並以圖示的方法來解釋 R2-SOVA 跟 R4-SOVA 如何做解碼運算。若傳送端發送 8 個位元，在$(7, 5)_8$的編碼器及經由一隨機交錯器來產生第二組同位元，其傳輸率為 1/3，經由 BPSK 調變後送出，並在 AWGN 的模擬雜訊通道下，接收端收到為一組實數值的向量，如下表一所示。

表一 模擬實例

Input Bit	Systematic Bit	Parity Bits 1	Parity Bits 2	Received Sequences
0	0	0	0	(-1.47, 1.30, -1.53)
1	1	1	1	(1.82, -0.34, 0.83)
0	0	1	0	(-0.24, 0.94, -3.22)
1	1	0	1	(1.48, -1.14, -1.13)
1	1	0	0	(-0.02, -0.94, -2.73)
1	1	0	1	(1.18, -2.48, 0.21)
1	1	0	0	(0.71, 1.07, -1.32)
1	1	0	1	(3.40, -0.26, 2.76)
*	0	1	0	(-0.33, 1.75, -0.28)
*	1	1	0	(1.65, 3.59, -0.53)

(1). R2-SOVA 解碼：經第 1 次解碼後可得：



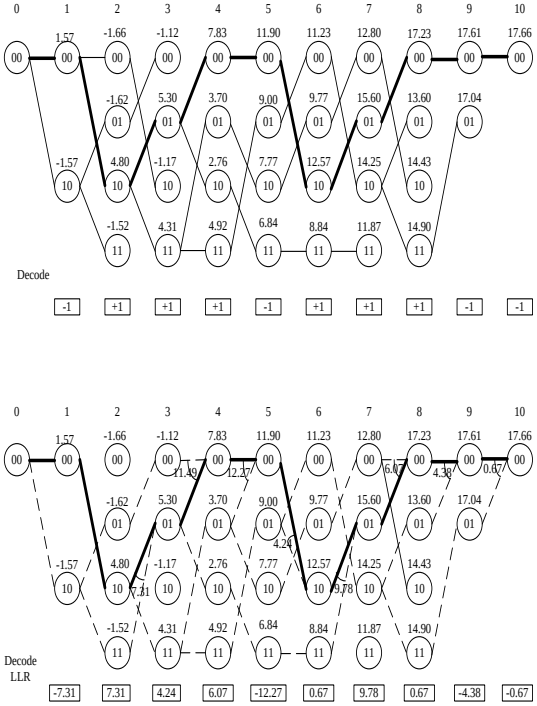


圖三 R2-SOVA Decoder-1 1st iteration 解碼後的數據

表二 R2-SOVA Decoder-1 1st iteration 解碼後的數據

Trellis Stage k	Decoded Bit uk	Δ_k^s	L1	Le1	xs1
1	-1	—	-3.14	-0.408715	0
2	1	—	2.67	1.51592	1
3	-1	8.14	-8.14	-4.04078	0
4	1	6.05	6.05	2.88199	1
5	1	2.67	0.98	2.45779	1
6	1	3.18	3.18	1.40929	1
7	1	3.14	3.14	2.73823	1
8	1	0.98	0.98	1.68608	1
9	-1	5.23	-5.23	0	0
10	1	7.04	7.04	0	1

經由 R2-SOVA 第 2 次解碼後可得：



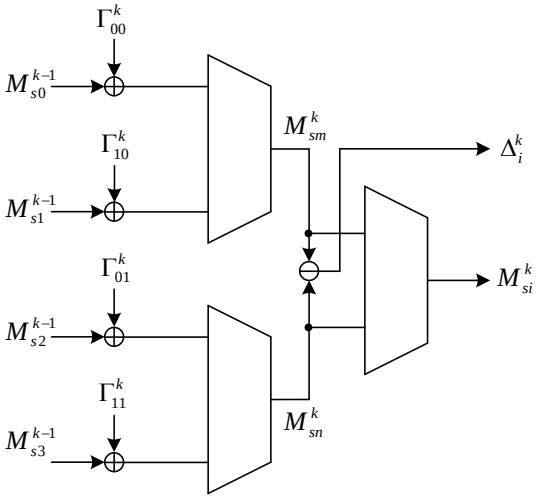
圖四 R2-SOVA Decoder-2 1st iteration 解碼後的數據

表三 R2-SOVA Decoder-2 1st iteration 解碼後的數據

Trellis Stage k	Decoded Bit uk	Δ_k^s	L2	Le2	xs2
1	-1	—	-7.31	-4.17	0
2	1	—	7.31	4.12	1
3	1	7.31	4.24	3.26	1
4	1	11.4	6.07	2.92	1
5	-1	12.2	-12.27	-4.12	0
6	1	4.24	0.67	-1.99	1
7	1	9.78	9.78	3.73	1
8	1	6.07	0.67	-0.3	1
9	-1	4.38	-4.388	0	0
10	-1	0.67	-0.67	0	0

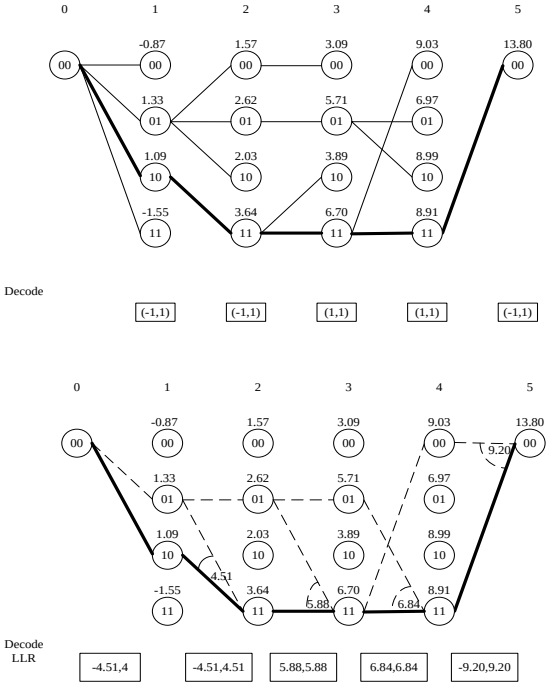
(2). R4-SOVA 的解碼：

下圖為 R4 分支機率累積值之運算單元。



圖五 R2-SOVA 分支機率累積值的運算單元

經由 R4-SOVA 經第 1 次解碼後可得：

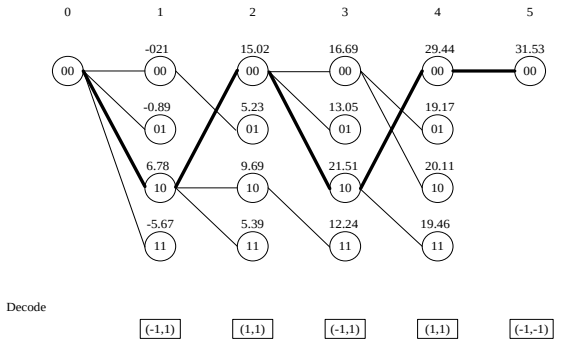


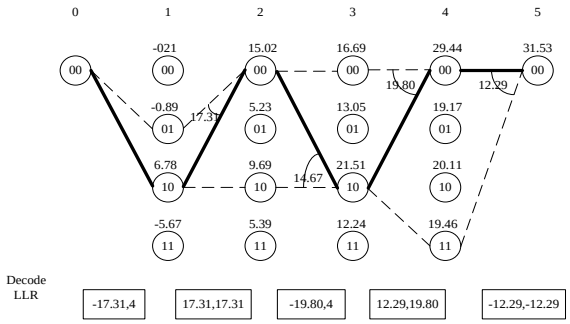
圖六 R4-SOVA Decoder-1 1st iteration 解碼後的數據

表四 R4-SOVA Decoder-1 1st iteration 解碼後的數據

Trellis Stage k	Decoded Bit uk	Δ_k^s	L1	Le1	xs1
1	-1	—	-4.51	-2.55	0
2	1		4	1.57	1
3	-1	4.51	-4.51	-4.18	1
4	1		4.51	2.54	1
5	1	5.88	5.88	5.91	0
6	1		5.88	4.3	1
7	1	6.84	6.84	5.89	1
8	1		6.84	2.29	1
9	-1	9.2	-9.2	0	0
10	1		9.2	0	0

經第 2 次解碼後可得：





圖七 R4-SOVA Decoder-2 1st iteration 解碼後的數據

表五 R4-SOVA Decoder-2 1st iteration 解碼後的數據

Trellis Stage k	Decoded Bit uk	Δ_k^s	L2	Le2	xs2
1	-1	—	-17.31	-12.79	0
2	1		4	2.43	1
3	1	17.31	17.31	10.47	1
4	1		17.31	10.47	1
5	-1	14.67	-19.8	-15.28	0
6	1		4	1.57	1
7	1	19.8	12.29	7.77	1
8	1		19.8	13.91	1
9	-1	12.29	-12.29	0	0
10	-1		-12.29	0	0

由上述之解碼過程，我們可得以下是兩個結構的比較分析

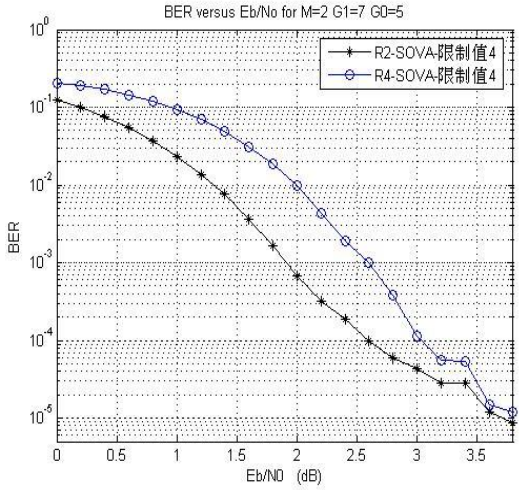
表六 R2 與 R4-SOVA 效能比較

Radix	Speedup	Complexity	Survivor-MM
2	1	1	1
4	2	2	1/2

由上表(六)可以得知雖然 R4 的計算複雜度約為 R2 的二倍，但是其解碼的速率卻可以提升一倍且計算存活路徑所需的記憶體也大概可以減少一半的空間。

4. 模擬結果

若傳送端發送 10M 個位元，在$(7, 5)_8$ 的編碼器及且經由一隨機亂數的交錯器來產生第二組同位元，在傳輸率為 1/3 下，經由 BPSK 調變後送出去，並在有 AWGN 的模擬雜訊通道下，其解碼的結果可得到如下圖八之情形，亦即在位元錯誤率(BER)在 10^{-4} 下約只下降 0.4dB。



圖八 模擬結果

5. 結論

使用 R4 來替代原本的 R2 的結構，即向前瞻看一個時間位元點。利用此一方法在計算 ML 路徑時可以節省大約一半存活路徑所需記憶體的儲存

空間及運算的延遲時間，但相對在節省所需之空間及解碼時間下，其解碼效益約下降了 0.4dB 在位元錯誤率為 10^{-4} 以下時。

6. 參考文獻

[1] Joachim Hagenauer and Peter Hoeher A Viterbi Algorithm with Soft-Decision Outputs and its Application IEEE 1989.

[2] L.Hanzo T.H.Liew B.L.Yeap Turbo Coding,Turbo Equalisation and Space-Time Coding For Transmission over Fading Channels.

[3] Thomas,C. Bickerstaff, M.A. Davis, L.M. Prokop, T. Widdup, B. Gongyu Zhou, Garrett, D. Nicol, C. Integrated circuits for channel coding in 3G cellular mobile wireless systems Communications Magazine, IEEE Volume 41, Issue 8, Aug. 2003 Page: 150 – 159.

[4] Nakamura, E.B. Uehara, G.T. Chu, C.W.P. Shu Lin, A 755 Mb/s Viterbi decoder for the RM (64, 35, 8) subcode Solid-State Circuits Conference, 1999. Digest of Technical Papers. ISSCC. 1999 IEEE International 15-17 Feb. 1999 Page: 342 – 345.

[5] 戴俊彥 改良式 SOVA 解碼演算法 A Modified Sova Decoding Algorithm 長庚大學電機工程研究所 碩士論文.

[6] Ghrayeb, A. Chuan Xiu Huang, Improvements in SOVA-based decoding for turbo-coded storage channels Magnetics, IEEE Transactions on Volume 41, Issue 12, Dec. 2005 Page: 4435 – 4442.

[7] J. G. Proakis，Digital Communication，3rd Edition. New York ： McGraw-Hill，1995.