A Maximun-likelihood decoding algorithm for parallel concatenated convolutoinal codes

Prepared by Hsin-Chi Hsu

Directed by Prof. Po-Ning Chen

In Partial Fulfillment of the Requirements

For the Degree of
Master of Science

Department of Communication Engineering

National Chiao Tung University

Hsinchu, Taiwan 300, R.O.C.

E-mail: u8713524@cc.nctu.edu.tw

June 10, 2000

Abstract

In this thesis, we apply a code-tree based algorithm (MLSDA) proposed in [1] to the decoding of turbo codes. Unlike the iterative decoding algorithm proposed in [3], which is a suboptimal decoding algorithm, the MLSDA turns out to be a maximum-likelihood decoding algorithm for turbo codes. We also investigate the error floor phenomenon[2] for short turbo codes based on the new ML decoding algorithm. Surprisingly, the error floor phenomenon for the code is resulted by the iterative decoding algorithm not the distance spectrum of the code itself. By the simulation results, the iterative decoding algorithm proposed originally performs badly on these short turbo codes.

Acknowledgements

I would like to thank Dr. Po-Ning Chen and Dr. Yunghsiang S. Han for their encouragement and guidance. This work would not have been possible without their advice and commitments.

I would like to thank my dear laboratory mates Chia-Feng, Yi-Hung, Chia-Lung, Shang-Pin. They provide much fun on countless boring days. I would also like to thank the members of Network Technology Laboratory and my friends.

Finally, I give the greatest respect to my family and girl friend for their continued support during research.

Contents

List of Figures

Chapter 1

Introduction and Background

In 1993, codes with near-capacity performance, named turbo codes, were reported. turbo codes can achieve the bit error rate (BER) only 0.7 dB away from capacity limit with a reasonable complexity of decoding and with large block length[3]. Although the iterative decoding performed on the turbo codes [3] is not an ML decoding algorithm, it provides a feasible decoding complexity to decoding long length codeword. Even through the turbo codes with iterative decoding have excellent performance at low signal-to-noise ratios (SNRs), the BER of these codes can't be lower at moderate SNRs. This phenomenon is called "error floor" and is shown in Figure1.1 [2]. One can see that the BER decreases rapidly at E_b/N_0 from $0.8dB$ to $2.0dB$ but the slope of the curve is almost flat beyond $2.0dB$. In [2], the "error floor" phenomenon was shown to be a consequence of the relatively low free distance of the code. For the $(37, 21, 65536)$ turbo code, the free distance asymptote is given by [2]

$$P_{free} = \frac{3 \times 2}{65536} Q\left(\sqrt{6\frac{E_b}{N_0}}\right). \tag{1.1}$$

In [1], authors have proposed a soft-decision sequential decoding algorithm for time-invariant convolutional codes which are antipodally transmitted over the additive white Gaussian noise (AWGN) channel. furthermore, authors defined a new metric for the sequential decoding algorithm, and presented a new *maximum-likelihood soft-decision sequential*

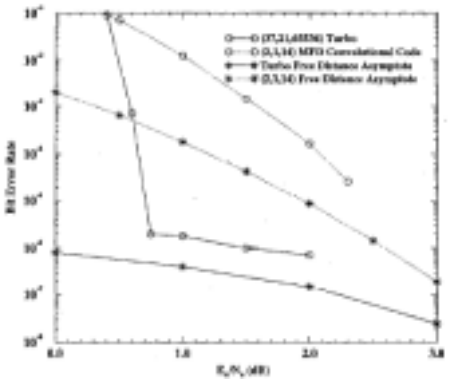

Figure 1.1: Simulation results for the (37,21,65536) turbo code, R=1/2, and the free distance asymptote.

decoding algorithm (MLSDA) based on the new metric.

In this thesis, we apply the algorithm (MLSDA) to the decoding of turbo codes. Unlike the iterative decoding algorithm proposed in [3], which is a suboptimal decoding algorithm, the algorithm turns out to be a maximum-likelihood decoding algorithm for turbo codes. We also investigate the error floor phenomenon [2] for short turbo codes based on the new ML decoding algorithm. Surprisingly, the error floor phenomenon for the codes is caused by the iterative decoding algorithm not the distance spectrum of the code itself. By the simulation results, the iterative decoding algorithm proposed originally performs badly on the short turbo codes.

The thesis is organized as follows:

In Chapter 2, the Background of turbo coeds is briefly presented. Definition and Notations for convolutional codes are first given, and then the encoder structure of turbo coder is introduced. Also an upper bound which explain of the performance of turbo codes at a primary level is discussed. Finally, we give the code tree representation for turbo codes. Chapter 3 is devoted to applying MLSDA to decode the turbo codes. The Tree-based MLSDA for

convolutional codes is introduced, and then tree-based MLSDA for the turbo codes is given. Chapter 4 includes the simulation results. We also compare the performance between the MLSDA and iterative decoding (traditional BCJR decoding) algorithm for some short length codes. Chapter 5 concludes our report and points out some observation.

Chapter 2

Introduction to Turbo Codes

In this chapter, we introduce the convolutional codes, the structure of turbo encoder and then describe the tree representation of the turbo codes.

I Definitions and Notations for Convolutional Codes

The following definitions and notations are defined similarly to [6] and [7].

Let $\boldsymbol{C}$ be a binary (n, k, m) convolutional code, where m is the *memory order* which is defined as the maximum number of shift-register stages from the encoder input to the encoder output.

Let

$$R \stackrel{\triangle}{=} k/n \quad \text{and} \quad N \stackrel{\triangle}{=} n(L+m)$$

be respectively the *code rate* and the *code length* of $\boldsymbol{C}$, where L represents the length of the information sequence. When a convolutional code is transformed into its equivalent linear block code, its *effective code rate* [6] (for block code) becomes

$$\tilde{R} \stackrel{\triangle}{=} \frac{kL}{N} = \frac{kL}{n(L+m)}.$$

As anticipated, $\tilde{R} \approx R$ when $L \gg m$.

Next, we consider a structural representation of codewords for convolutional codes. A *code tree* of an (n, k, m) convolutional code represents every codeword as a path over the tree. For an information sequence of length L, the code tree consists of $(L + m + 1)$ levels. The leftmost node at level 0 is called the *origin node*. There are exactly 2^k branches leaving each node at the first L level. For those nodes at levels L through $(L + m)$, there is only one branch leaving each node. The 2^{kL} rightmost nodes at level $(L + m)$ are called *terminal nodes*. Throughout, a path from the origin node to the terminal node will be called a *code path*.

II Structure of Turbo Encoder

In the design of turbo codes, a special kind of convolutional encoder named recursive systematic convolutional (RSC) encoder is used. *Systematic* means that part of the encoded bits is the same as the input bits. *Recursive* means that the registers in the generator has a feedback loop.

The encoder of turbo codes is a parallel concatenation of several RSC encoders which are separated by interleavers. Fig. 2.1 shows the structure of a specific turbo encoder which is a concatenation of two identical (37,21) RSC encoders. This is the most famous one which Berrou et al. proposed in 1993 [3].

The interleaver is used to rearrange the bits in the input sequence such that the two constituent encoders are operating on two input sequences consisting of the same bits, but in different order. A block interleaver can be described in term of an $N \times M$ matrix. These interleavers are characterized by a process which the data is written into the rows of a matrix of memory elements and read out along the columns. A reverse block interleaver, which is a modified block interleaver, reads the date into the rows of a matrix of memory elements and read out reversely along the columns [5]. In figure 2.2, an example with $M = N = 3$ is

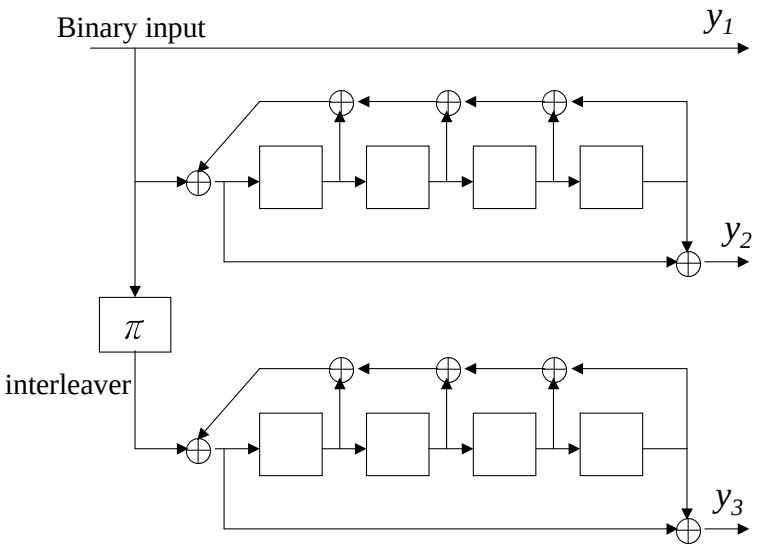

Figure 2.1: Turbo Code Encoder with Component Code (37,21)

given to demonstrate how block and reverse block interleavers work.

A block interleaver can be further characterized by a permutation which maps the positions of input sequence to those of output sequence, so can a reverse block interleaver. For example, the permutations associated with the block interleaver and the reverse block interleaver given in Figure 2.2 are

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 4 & 7 & 2 & 5 & 8 & 3 & 6 & 9 \end{pmatrix}$$

and

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 7 & 4 & 1 & 8 & 5 & 2 & 9 & 6 & 3 \end{pmatrix}$$

respectively.

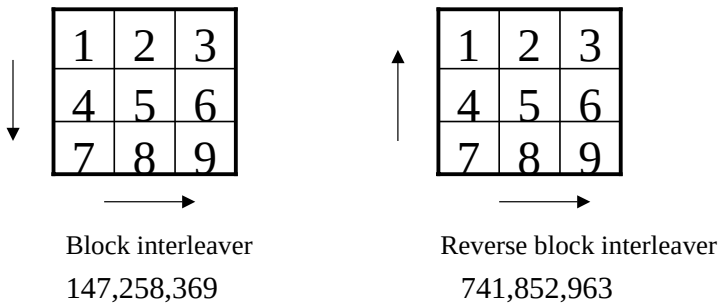

Figure 2.2: A simple example of block interleaver and reverse block interleaver

III The performance bound for turbo codes

In this section, we give a performance bound for convolutional codes which is also valid for turbo codes. If the information length for a turbo encoder is L and rate is $1/3$, any codeword of this code is with length $N = 3L$.

By using the union bound, the bit error rate(BER) of a finite length convolutional code with maximum-likelihood (ML) decoding over the AWGN channel with SNR of E_b/N_0 can be bounded above [2] by

$$P_b \leq \sum_{i=1}^{2^L} \frac{w_i}{L} Q\left(\sqrt{d_i \frac{2R_c E_b}{N_0}}\right), \tag{2.1}$$

where w_i is the Hamming weight of the i^{th} information sequence, d_i is the total Hamming weight of the i^{th} codeword, and $Q(.)$ is the complimentary error function. Let the number of codewords with Hamming weight d be denoted as N_d, and the total information weight of codewords with Hamming weight d as W_d. Define the *average information weight per codeword* as:

$$\tilde{w}_d = \frac{W_d}{N_d}. \tag{2.2}$$

Then the upper bound given above can be re-written as:

$$P_b \leq \sum_{d=d_{free}}^{3L} \frac{N_d \tilde{w}_d}{L} Q\left(\sqrt{d_i \frac{2R_c E_b}{N_0}}\right). \tag{2.3}$$

The Equation 2.3 can be used to estimate the BER of a turbo code with short length when an ML decoder is applied to.

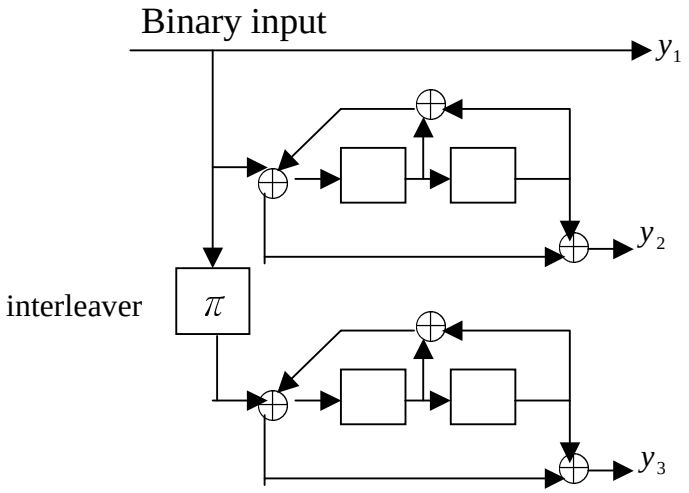

Figure 2.3: The turbo encoder with component code (7,5)

In order to consider the situation that the same input bit has been encoded twice, we first

combine the two information sequence $I = I_1I_2I_3I_4I_5$ and $\bar{I} = I_3I_1I_4I_5I_2$ according to the time the input bit fed into the encoder. Consequently, we can represent the two sequences as ordered pairs like $(I_1I_3), (I_2I_1), (I_3I_4), (I_4I_5)$, and (I_5I_2). These ordered pairs will determine the branches out of a particular nodes in the code tree. At the origin node of the code tree, the pair (I_1I_3) are not determined, so they could be $(00), (01), (10)$, and (11). Thus, there are four paths generated from the orgin node. At level one of the code tree , the new pair (I_2I_1) is considered. Since input bit I_1 in the new pair has been determined previously, not all possible values can be assigned to I_1. Consequently, the number of branches goes out from the nodes at level two of the code tree may be less than four. For example, considering the path generated from the path associated with ordered pair (10) in Figure 2.4, the successor of this path only could be (01), and (11). Furthermore, one may see that the numbers of successors can only be 1, 2 or 4 which is determined by the number of input bits in the new pair previously assigned . If both new input bits are previously assigned, there would be one valid successor; if only one of the two new input Bits is determined, there would be two valid successors; if none of the two new input bits is assigned, there would be four valid successors. The code tree for the code given in Figure 2.3 is presented in Figure 2.5. In this figure, I and $\bar{I}$ in the branch label $I\bar{I}(c_1c_2c_3)$ represents the pre- and post-interleaved input bits, respectively; $(c_1c_2c_3)$ is the corresponding output sequence.

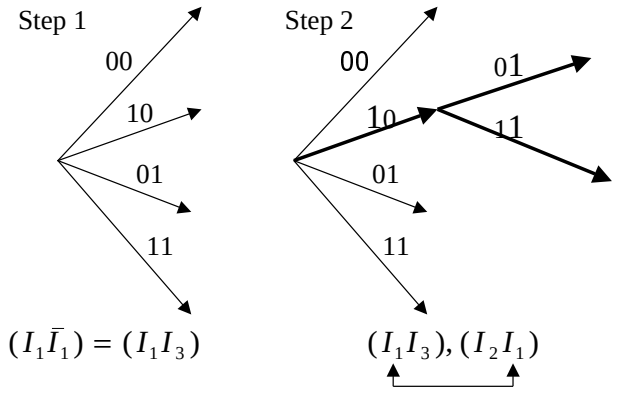

Figure 2.4: The generation of the code tree of a turbo code

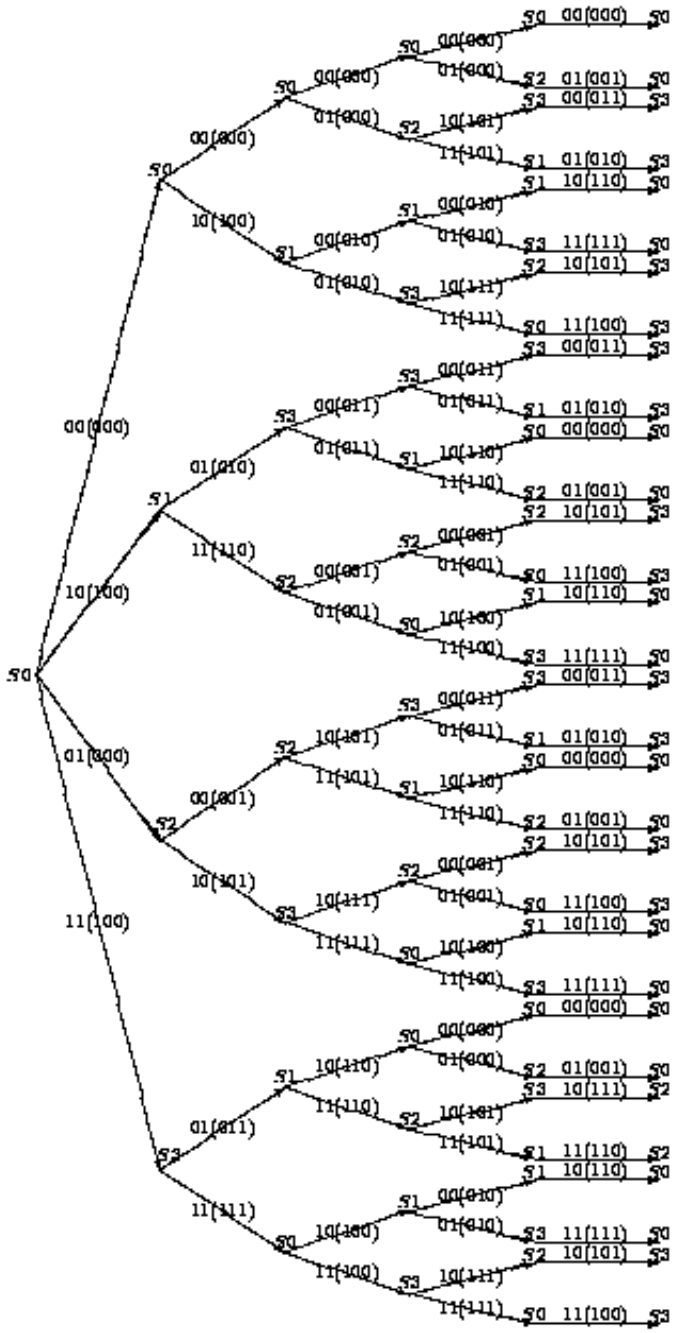

Figure 2.5: The code tree of a turbo code. The interleaver permute $I_1 I_2 I_3 I_4 I_5$ into $\bar{I}_1 \bar{I}_2 \bar{I}_3 \bar{I}_4 \bar{I}_5 = \bar{I}_3 \bar{I}_1 \bar{I}_4 \bar{I}_5 \bar{I}_2$.

Chapter 3

Maximum-likelihood Soft-Decision Sequential Decoding Algorithm for Turbo Codes

In this chapter we present how to apply the MLSDA proposed in [1] to turbo codes which are represented by their code trees.

I Tree-based Maximum-likelihood Soft-Decision Sequential Decoding

Let $\boldsymbol{v} \stackrel{\triangle}{=} (v_0, v_1, \ldots, v_{N-1})$ represent a binary codeword of an (n, k, m) convolutional code, where, as defined in the previous chapter, $N \stackrel{\triangle}{=} n(L+m)$ is the code length and L is the length of the information sequence. Denote a portion of codeword $\boldsymbol{v}$ by

$$\boldsymbol{v}_{(a,b)} \stackrel{\triangle}{=} (v_a, v_{a+1}, \ldots, v_b).$$

For ease of notations, we also let $\boldsymbol{v}_{(b)} \stackrel{\triangle}{=} \boldsymbol{v}_{(0,b)}$. Assume that a binary codeword is antipodally transmitted over the AWGN channel, and induce a received vector $\boldsymbol{r} \stackrel{\triangle}{=} (r_0, r_1, \ldots, r_{N-1})$. In other words,

$$r_j = (-1)^{v_j}\sqrt{\mathcal{E}} + \lambda_j,$$

where $\mathcal{E}$ is the signal energy per channel bit, and λ_j is a noise sample of a Gaussian process with single-sided noise power per hertz N_0. The variance of λ_j is $N_0/2$, and the signal-to-noise ratio (SNR) for the channel is $\gamma \triangleq \mathcal{E}/N_0$. In order to account for the code redundancy of different code rates, we will use the signal-to-noise ratio per information bit ($\mathrm{SNR_b}$), denoted by γ_b, as

$$\gamma_b \triangleq \frac{\mathcal{E}_b}{N_0} = \frac{N}{KL}\gamma = \frac{1}{\tilde{R}}\gamma.$$

The *maximum-likelihood decoding rule* (MLD rule) [9, 10, 11] for a time-discrete memoryless channel can be formulated as

$$\text{estimate } \widehat{\boldsymbol{v}} = \boldsymbol{v}^* \quad \text{for} \quad \boldsymbol{v}^* \in \boldsymbol{C},$$

if

$$\sum_{j=0}^{N-1} \left(\phi_j - (-1)^{v_j^*}\right)^2 \leq \sum_{j=0}^{N-1} \left(\phi_j - (-1)^{v_j}\right)^2 \quad \text{for all } \boldsymbol{v} \in \boldsymbol{C}, \tag{3.1}$$

where

$$\phi_j \triangleq \ln \frac{Pr(r_j|0)}{Pr(r_j|1)}.$$

We next define a new path metric based on the second form of the MLD rule.

Definition 3.1 *For any path $\boldsymbol{x}_{(\ell n-1)}$ ending at level ℓ in a tree, we define the metric associated with it as*

$$M\left(\boldsymbol{x}_{(\ell n-1)}\right) \triangleq \sum_{j=0}^{\ell n-1} (y_j \oplus x_j)|\phi_j|. \tag{3.2}$$

The term $(y_j \oplus x_j)|\phi_j|$ in (3.2) is called the *bit metric*, and is denoted by $M(x_j)$. We remark that the path metrics defined here are equivalent to those given in [13] and [14], which were originally defined over code trees of block codes.

Following the definition of the path metric, we define the metric of a code path $\boldsymbol{v}$ by

$$M(\boldsymbol{v}) \triangleq \sum_{j=0}^{N-1} (y_j \oplus v_j)|\phi_j|.$$

It is clear that $\boldsymbol{e} = \boldsymbol{y} \oplus \boldsymbol{v} \in E(\boldsymbol{s})$. Thus, finding an $\boldsymbol{e}^*$ in the second form of the MLD rule is equivalent to finding a code path with the smallest metric in the trellis. We herein propose a sequential decoding algorithm using the new metric. As a result, the sequential decoding algorithm based on the new metric becomes an ML decoder. We therefore call it the *maximum-likelihood sequential decoding algorithm* (MLSDA).

<The tree-based MLSDA>

Step 1. Load the Stack with the origin node whose metric is assigned to be zero.

Step 2. Compute the metrics of the successors of the top path in the Stack, delete the top path from the Stack.

Step 3. Insert the new paths into the Stack, and reorder the Stack according to ascending metric values.

Step 4. If the top path in the Stack ends at the terminal node in the tree, the algorithm stops; otherwise go to Step 2.

For illustrating the MLSDA clearly, the block diagram of it is shown in Figure 3.1 .

II MLSDA for Turbo Codes

We can generate the code tree of the turbo code as presented in previous chapter, then we apply the tree-based MLSDA to search the optimal path which is the path with the smallest metric on the code tree. Because the limit of the memory, we must limit the size of the

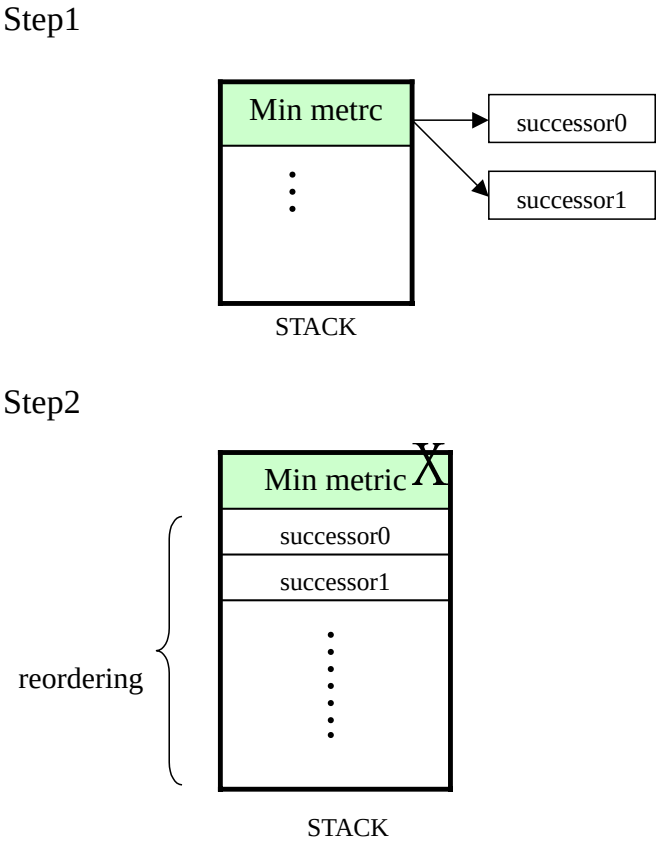

Figure 3.1: The block diagram of MLSDA

stack. If the size of stack is larger than the threshold previously defined, we deleted the path with maximum metric. The following is the modified tree-based MLSDA for turbo codes.

<The tree-based MLSDA for turbo Codes>

Step 1. Load the Stack with the origin node whose metric is assigned to be zero.

Step 2. Compute the metrics of the successors of the top path in the Stack, it might be one or two or four successors. Delete the top path from the Stack.

Step 3. Insert the new paths into the Stack, and reorder the Stack according to ascending metric values.

Step 4. If the stack size exceed the Threshold, delete the path with the maximum metric in the stack.

Step 5. If the top path in the Stack ends at the terminal node in the tree, the algorithm stops; otherwise go to Step 2.

Chapter 4

Simulation Results

In this chapter, we present simulation results of MLSDA for turbo codes with short length, and compare to those of original turbo decoder algorithm for the same codes.

I Simulation environment

During simulation, the component codes of turbo encoder are chosen to be 16-state RSC's with generator (37,21) whose structure was shown in Figure 2.1. Due to the limitation of computing power, the information length L is set to be 36. Consequently, the length of any codeword is 108. The simulations has been done for the block interleaver and the reverse block interleaver. The channel model chosen is the additive white Gaussian channel(AWGN) described in Chapter 3. The complete simulation model is shown in Figure 4.1 and two decoding algorithms used for the same turbo code are MLSDA and the iterative decoding algorithm proposed in [3].

II Simulation results

The simulations have been done for the block interleaver and reverse interleaver. For all samples run in the simulation, the size of stack has been limited to be 4000000 nodes. For

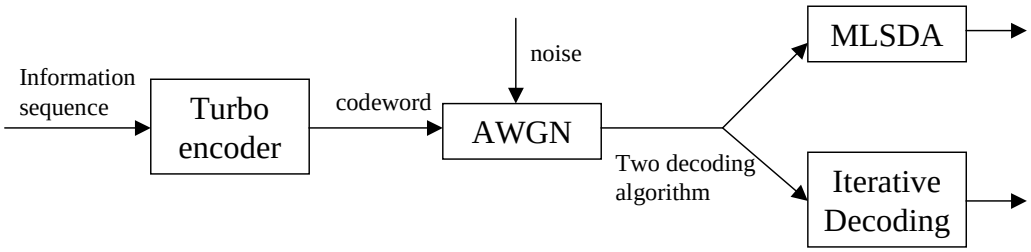

Figure 4.1: Simulation Model

comparison purpose, the upper bound for the BER of turbo codes given in Equation 2.3 are calculated and plotted in all figures.

II.1 Block interleaver

In order to calculate the upper bound given in Equation 2.3 we need to find the distance spectrum of the codes. The spectrum for distances up to 12 is given in Table 4.1.

d	N_d	W_d
3	1	1
6	2	4
7	2	2
9	3	3
10	25	50
11	13	31
12	13	34

Table 4.1: The distance spectrum of the turbo encoder with L=36,BI,(37,21)

In this table, N_d denotes the number of codewords with Hamming weight d, and W_d the total information weight of codewords with Hamming weight d. The free distance of this code (d_{free}) is 3.

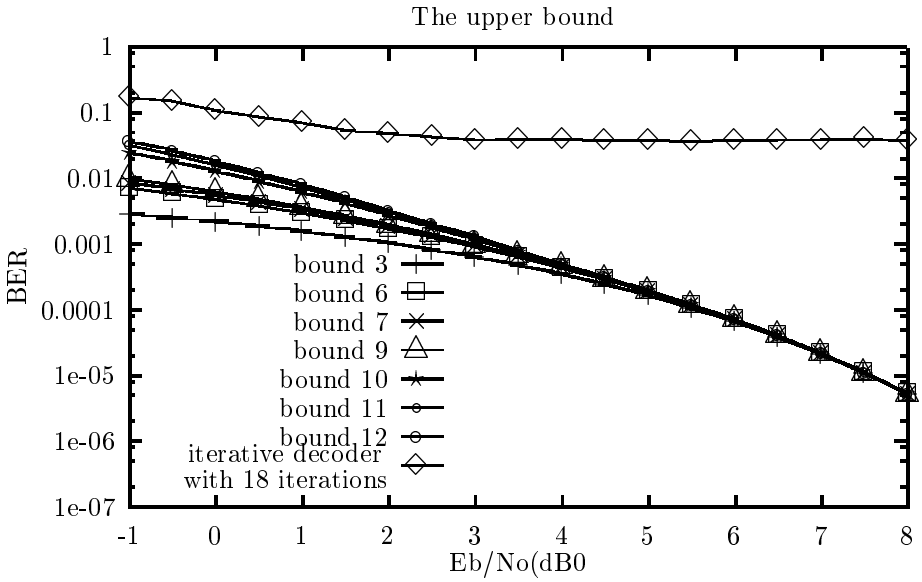

Figure 4.2: The upper bound of the turbo code with $L = 36$, BI, $(37, 21)$

Figure 4.2 shows the upper bound of the turbo encoder with $L = 36$, BI, $(37, 21)$. Even though Equation 2.3 should be calculated based on whole distance spectrum (the summation from d_{free} to $3L$), because of the complexity of computing the spectrum, we simply use the spectrum from distance d_{free} to 12. In Figure 4.2, the "bound i" curve represents the result that Equation 2.3 is calculated based only on the spectrum from distance d_{free} to i. Since the curves plotted in the figure are very close to each other from $5dB$ to $8dB$, it seems that the d_{free} dominates the performance bound for SNRs higher than $5dB$; however, when SNRs is below $5dB$, the d_{free} doesn't dominate the performance of the code.

Figure 4.3 shows the simulation results for MLSDA and the iterative decoder proposed in [3] with 18 and 24 iterations. From the results, one can see that the performance of turbo codes with 18 iterations is almost the same as that with 24 iterations. It means that 18 iterations is enough for the iterative decoder to almost reach its best performance. Furthermore, the "error floor" phenomenon occurs about at $3dB$ for the code. When SNR is larger than $3dB$, for all samples simulated, MLSDA acts like a real ML decoding algorithm

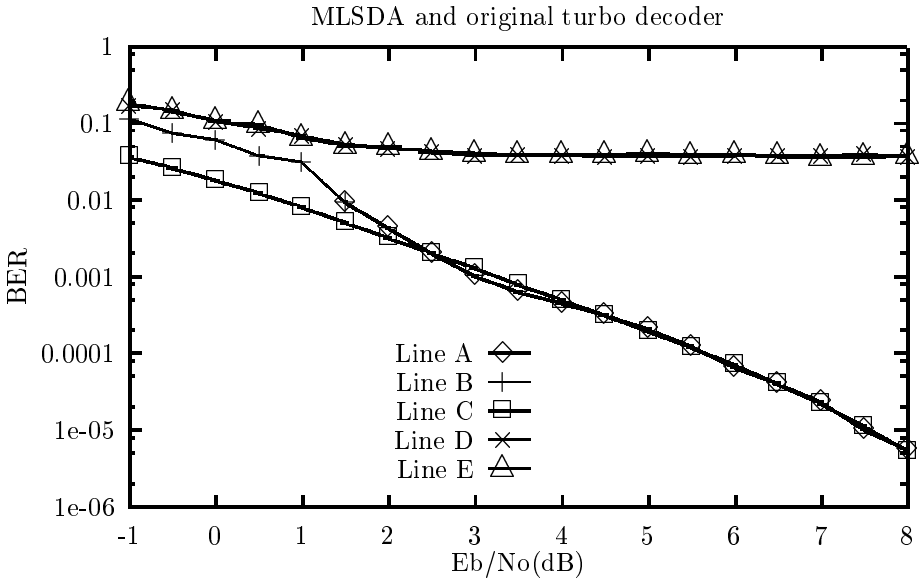

Figure 4.3: Performance of MLSDA and original turbo decoder. The Line A is the MLSDA without stack size limitation. The Line B is MLSDA with limited stack size. The Line C is bound 12. The Line D is the iterative decoding with 18 iterations.The Line E is the iterative decoding with 24 iterations.

since it has never reached the limit of stack size. On the other hand, when SNR is below $3dB$, MLSDA becomes a suboptimal decoding algorithm because of the limitation of stack size.

It has been conjectured that the "error floor" phenomenon is caused by the distance spectrum of the turbo code proposed in [3] (with information length 65536 and a nonuniform interleaver). That is, the iterative decoding for this code has little contribution to the "error floor" phenomenon. Surprisingly, for the short length code simulated, the "error floor" phenomenon is resulted mainly by the iterative decoder chosen and is not related to the distance spectrum of the code. This is conformed by the fact that ML decoder have much better BER than the iterative decoder when SNR is greater than $3dB$.

In Figure 4.4 we investigate the computational complexity of the MLSDA which is defined as the numbers of metrics evaluated. The worst complexity and the average complexity are

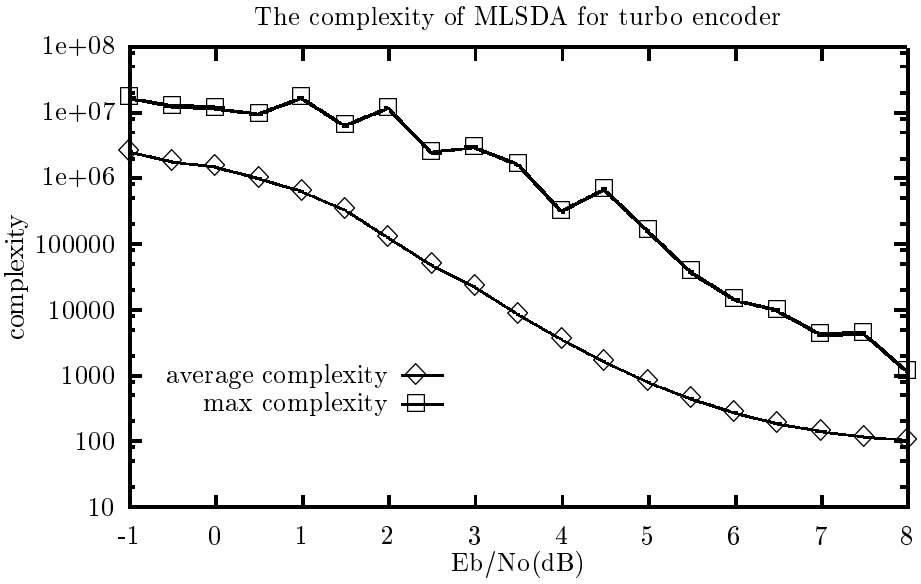

Figure 4.4: The complexity of MLSDA for turbo encoder with BI

plotted according to different SNRs.

The average complexity at $7.5dB$ is about 115 which is very close to the low bound of complexity 108. In Figure 4.4, one can see that the average complexity is below 1000 when E_b/N_0 is above $5dB$. The complexity of the MLSDA grows exponentially as E_b/N_0 is below $5dB$. The worst complexity is about 10-100 times larger than the average complexity.

II.2 Reverse block interleaver

The reverse block interleaver (RBI) was first introduced in [5]. For low bit-error rates (BER's), the performance of the turbo code with RBI is better than that obtained with a random interleaver. It also yields a short decoding delay and reduced decoding complexity (in terms of memory) [5]. In this subsection, we investigate the performance of the turbo code given in Subsection II.1 associated with RBI instead of with block interleaver.

In Table 4.2, N_d denotes the number of codewords with Hamming weight d, and W_d the total information weight of codewords with Hamming weight d. The free distance of the

d	N_d	W_d
5	1	1
8	1	1
9	3	4
10	2	3
11	2	3
12	3	6
13	9	17
14	12	31
15	14	65

Table 4.2: The distance spectrum of the turbo encoder with L=36,RBI,(37,21)

turbo encoder with $L = 36$, RBI, $(37, 21)$ is 5.

Figure 4.5 shows the upper bound of the turbo encoder with $L = 36$, RBI, $(37, 21)$. As stated before, even though Equation 2.3 should be calculated based on whole distance spectrum (the summation from d_{free} to $3L$), because of the complexity of computing the spectrum, we simply use the spectrum from distance d_{free} to 15. In Figure 4.5, the " $d = i$" curve represents the result that Equation 2.3 is calculated based only on the spectrum from distance d_{free} to i. Since the curves plotted in the figure are very close to each other from $3dB$ to $6dB$, it seems that the d_{free} dominates the performance bound for SNRs higher than $3dB$. But when SNR is below $5dB$, the d_{free} doesn't dominate the performance of the code.

Figure 4.6 shows the simulation results for MLSDA and the iterative decoder proposed in [3] with 18 and 24 iterations. From the results, one can see that the performance of turbo codes with 18 iterations is almost the same as that with 24 iterations. It means that 18 iterations is enough for the iterative decoder to almost reach its best performance. Furthermore, the "error floor" phenomenon occurs about at $3dB$ for the code. When SNR is larger than $3dB$, for all samples simulated, MLSDA acts like a real ML decoding algorithm since it has never reached the limit of stack size. On the other hand, when SNR is below $3dB$, MLSDA becomes a suboptimal decoding algorithm because of the limitation of stack size.

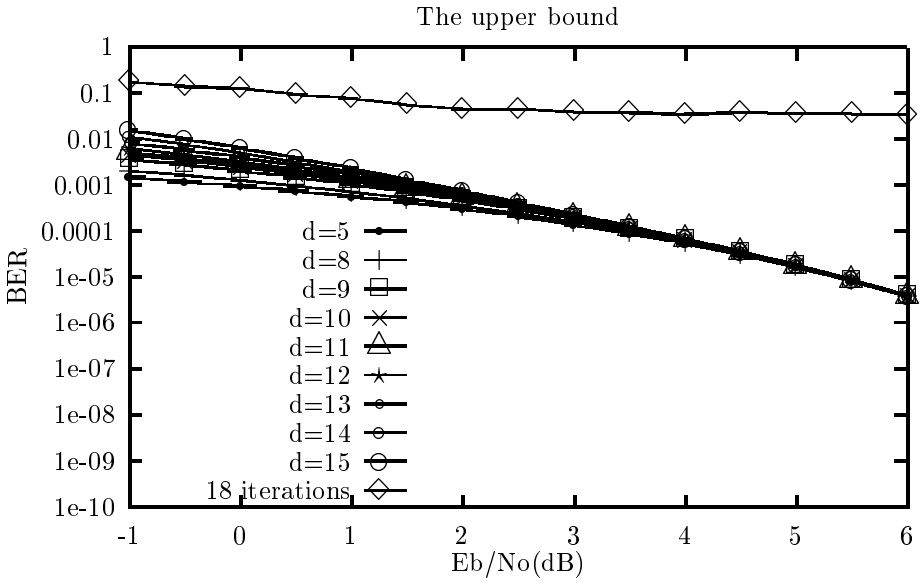

Figure 4.5: The upper bound of the turbo encoder with $L = 36$, RBI, $(37, 21)$

As the results given in the previous subsection, for the short length code simulated, the "error floor" phenomenon is caused mainly by the iterative decoder chosen and is not related to the distance spectrum of the code. This is conformed by the fact that ML decoder have much better BER than the iterative decoder when SNR is greater than $3dB$.

In Figure 4.7 we investigate the computational complexity of the MLSDA. The worst complexity and the average complexity are plotted according to different SNRs.

In Figure 4.7, one can see that the average complexity is below 1000 when E_b/N_0 is above $5dB$. The complexity of the MLSDA grows exponentially as E_b/N_0 is below $5dB$. The worst complexity is about 10-100 times larger than the average complexity.

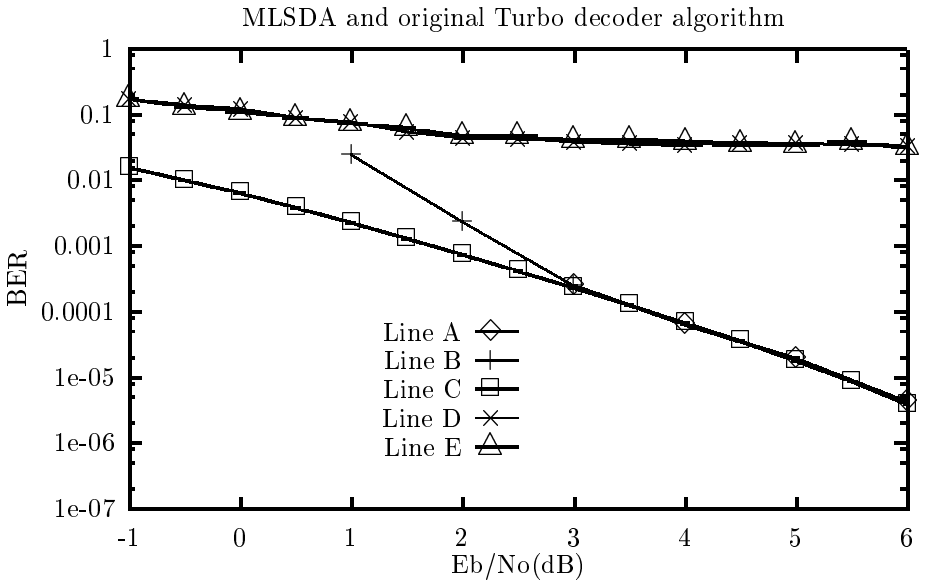

Figure 4.6: Performance of MLSDA and original turbo decoder. The Line A is the MLSDA without stack size limitation. The Line B is MLSDA with limited stack size. The Line C is bound 12. The Line D is the iterative decoding with 18 iterations.The Line E is the iterative decoding with 24 iterations.

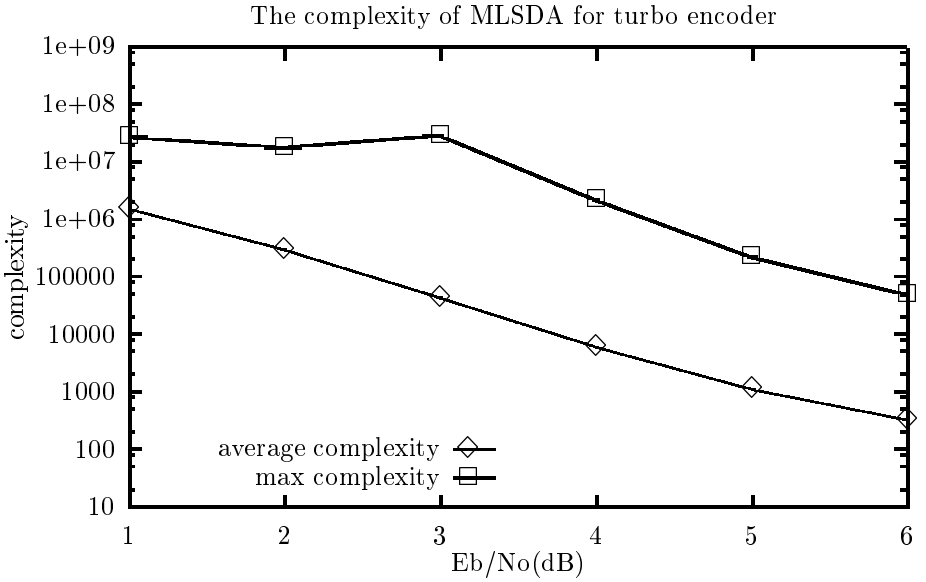

Figure 4.7: The complexity of MLSDA for turbo encoder with RBI

Chapter 5

Conclusions

The original turbo decoding algorithm-BCJR iterative decoding algorithm is not a maximum-likelihood decoding algorithm. When SNR is high, MLSDA proposed here performs as an ML decoding algorithm. At low SNRs, the computational complexity grows exponentially for MLSDA and the size of stack easily grows to exceed the limitation . Under this situation, MLSDA is no more an ML decoder. It was pointed out that the error floor phenomenon is mainly resulted by the first spectrum term of the codeword [2] for the turbo codes given in [3]. It means that the error floor phenomenon is induced by the encoder structure. However, for the short length codes simulated, the "error floor" phenomenon is caused mainly by the iterative decoder chosen and is not related to the distance spectrum of the codes. This is conformed by the fact that ML decoder have much better BER than the iterative decoder when SNR is greater than $3dB$.

Bibliography

[1] Yunghsiang S. Han and Po-Ning Chen, "Maximum-likelihood soft-decision sequential decoding algorithms for convolutional codes," presented at the recent results session of *the 1998 IEEE International Symposium on Information Theory*, Cambridge, MA, USA, August, 1998.

[2] Lance C. Perez, Jan Seghers,Daniel J. Costello, *A Distance Spectrum Interpretation of Turbo Codes*, *IEEE TRANS. Inform. Theory*, pp.1698–1709, NOVEMBER 1996.

[3] S. Berrou, A. Glavieux,and P. Thitimajshima, *Near Shannon Limit error-correcting coding and decoding: Turbo-Codes(1)*, *IEEE TRANS. Com.*, pp.1261–1271, October 1993.

[4] S. Berrou, A. Glavieux, *Near Optimum Error Correcting Coding and Decoding: Turbo-Codes*, *IEEE Int. Conf. on Communication*, pp.1064-1070, October 1993.

[5] Hanan Herzberg, *Multilevel Turbo Coding with Short Interleavers*, *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS*, pp.303-309, FEBRUARY

[6] S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1983.

[7] P. R. Chevillat and D. J. Costello, Jr., "An analysis of sequential decoding for specific time-invariant convolutional codes," *IEEE Trans. Inform. Theory*, pp. 443–451, July 1978.

[8] S. B. Wicker, *Error Control Systems for Digital Communication and Storage*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1995.

[9] Y. Be'ery and J. Snyders, "Optimal soft decision block decoders based on fast Hadamard transform," *IEEE Trans. Inform. Theory*, pp. 355–364, 1986.

[10] A. Vardy and Y. Be'ery, "More efficient soft-decision decoding of the Golay codes," *IEEE Trans. Inform. Theory*, pp. 667–672, 1991.

[11] A. Vardy and Y. Be'ery, "Bit-level soft decision decoding of Reed-Solomon codes," *IEEE Trans on Communications*, pp. 440–445, 1991.

[12] J. Snyders and Y. Be'ery, "Maximum likelihood soft decoding of binary block codes and decoders for the Golay codes," *IEEE Trans. Inform. Theory*, pp. 963–975, September 1989.

[13] L. Ekroot and S. Dolinar, "A* decoding of block codes," *IEEE Trans on Communications*, pp. 1052–1056, September 1996.

[14] Y. S. Han, "A new treatment of priority-first search maximum-likelihood soft-decision decoding of linear block codes," *IEEE Trans. Inform. Theory*, pp. 3091–3096, November 1998.

[15] N. J. Nilsson, *Principle of Artificial Intelligence*, Palo Alto, CA: Tioga Publishing Co., 1980.