國立交通大學

電信工程學系碩士班

碩 士 論 文

循序解碼演算法於有限堆疊下之路徑移除策略

Path Deletions for Finite Stack-Size Sequential-Type Decoding Algorithms

研 究 生：王晨屹

指導教授：陳伯寧　教授

中 華 民 國 九 十 八 年 七 月

循序解碼演算法於有限堆疊下之路徑移除策略

Path Deletions for Finite Stack-Size Sequential-Type Decoding Algorithms

研 究 生：王晨屹　　Student：Chen-Yi Wang

指導教授：陳伯寧　教授　　Advisor：Po-Ning Chen

國 立 交 通 大 學

電 信 工 程 學 系 碩 士 班

碩 士 論 文

A Thesis

Submitted to Institute of and Communication Engineering

College of Electrical and Computer Engineering

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Communication Engineering

July 2009

Hsinchu, Taiwan, Republic of China

中華民國九十八年七月

循序解碼演算法於有限堆疊下之路徑移除策略

研究生：王晨屹　　指導教授：陳伯寧　教授

國立交通大學

電信工程學系碩士班

中文摘要

本篇碩士論文針對循序解碼演算法的特有實作限制進行研究，即路徑堆疊容量有限。在路徑堆疊容量有限的前提下，當路徑堆疊超過上限時的有效路徑移除策略對於系統效能甚至解碼複雜度相當重要。基於此一研究背景，我們論文針對即時輸出解碼結果模式下的循序解碼演算法提出了若干有效路徑移除策略。而當系統容許「離線」輸出解碼結果時 — 意即系統允許接收到所有接收向量值後才開始進行解碼 — 我們提出了二階段解碼結構為基礎的路徑移除策略。簡言之，我們提出於「反向階段」使用M演算法來估計啟發函數值(heuristics function)以作為「順向階段」解碼所用之解碼量度參數。由於M演算法可由硬體電路實現，因此我們論文中僅考慮順向階段的解碼複雜度。模擬結果顯示，我們所提出的兩階段路徑移除策略在順向階段的解碼複雜度不僅優於使用費諾測度的堆疊演算法，更低於 Sikora 與 Costelo 於 2008 年所提出的二階段超級碼(supercode)解碼器[1]。

[1] M. Sikora and D. J. Costello, Jr., “Supercode heuristics for tree search decoding,” in *Proc. IEEE Inform. Theory Workshop*, Porto, Portugal, pp. 411-415, May 2008.

Path Deletions for Finite Stack-Size Sequential-Type Decoding Algorithms

Student: Chen-Yi Wang Advisor: Prof. Po-Ning Chen

Institute of Communication Engineering

National Chiao Tung University

Abstract

In this thesis, we focus on a specific practical constraint on sequential-type decoding algorithms, namely, the finite stack size. Under such a finite-stack-size limitation, the path deletion policy that is effective when the stack exceeds its limit becomes essential in performance and decoding complexity. At this background, we proposed several path deletion schemes for sequential-type decoding algorithms that can produce decoding outputs in an on-the-fly or instantaneous fashion. When ``off-line" decoding is allowed, where the decoding process starts after the reception of the entire received word, we proposed an alternative path deletion scheme based on a two-pass decoding structure. To be specific, the backward pass estimates the heuristic function in terms of the *M*-algorithm for use of the forward decoding search. As the *M*-algorithm can be hardware-implemented, only the computational complexity of the forward pass is accounted for. Simulations show that the computational complexity of the forward pass not only outperforms the stack algorithm with Fano metric but also is smaller than that of the two-pass supercode decoder proposed by Sikora and Costello in 2008 [1].

[1] M. Sikora and D. J. Costello, Jr., “Supercode heuristics for tree search decoding,” in *Proc. IEEE Inform. Theory Workshop*, Porto, Portugal, pp. 411-415, May 2008.

Acknowledgements

I am deeply grateful to everyone who helps me to finish the thesis.

To my advisor, Professor Chen, for your patient instruction and brilliance guidance throughout the research. This work would not been possible without your advise and commitment. I learn a lot from you. It is my pleasure to be advised by you.

To Professor Han, thanks for your opinions and suggestions in our meeting. You make me understand more about the research topic.

To Shin-Lin, thanks for your brilliant guidance through the research. This work cannot be finished without you. I learn so much from you.

To my lab mates, you always listen to my problem, give me much help and more advises, and encourage me when I am depressed.

And thanks National Chiao-Tung University for providing such good environment and such many resources for me.

In the end, I would like to dedicate this thesis to my family for their support, love and encouragement all the time.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In modern communication systems, Viterbi decoder is one of the most popular decoder for convolutional codes. However, one limitation of the prevalent Viterbi decoder is that its decoding complexity grows exponentially with code constraint length. This prevents its practical use for codes of long constraint length. In contrast, the computational complexity of the sequential decoding algorithm is independent of code constraint length. One can thus adopt the sequential decoding search when codes of long constraint length is recommended in certain applications.

The most well-known sequential-type decoding algorithm is perhaps the stack algorithm with Fano metric [3][5][11]. This algorithm, although suboptimal in performance, is efficient in computational complexity for medium to high signal-to-noise ratios (SNRs). In 2002, another metric for use by the sequential-type decoding algorithm was derived based on the Wagner rule [4]. It has been proved that adopting the new metric in place of the Fano metric in the sequential decoding search guarantees optimal performance.

A common problem of sequential-type decoding algorithms is that the memory consumption grows with the length of information sequence. In 2007, Shieh, Chen and Han proposed a method, named "Early Elimination", to alleviate this problem [8]. Specifically, the au-

thors proposed to directly eliminate the top path that is Δ-trellis-level prior to the farthest one among all paths that have been expanded thus far by sequential search. In the same work, the authors empirically showed that taking Δ to be around three times code constraint length suffices to achieve near optimal performance.

In this thesis, we attack the same problem from different standpoint. We assume in the first place that the memory of the system has an upper limit. Therefore, some paths need to be deleted when the stack is full. Several path deletion schemes are subsequently proposed and examined.

The stack algorithm can be treated as a special case of the "best-first" algorithm A for tree search in the computer science literature [7]. In its design, the algorithm A uses a cost function μ to guide the search through the graph, where μ consists of the accumulated cost g and the heuristic function h. With the help of a proper heuristic function, the decoding complexity can be reduced significantly.

By further assuming that the decoding process can be launched after the reception of the entire received vector, which we termed *off-line decoding*, a two-pass sequential-type decoding algorithm is also proposed. In 2008, Sikora and Costello proposed a two-pass decoder [9], where the backward pass generates proper heuristic function values, followed by the forward pass that sequentially search the codeword with the lowest cost. Two heuristic functions are designed in their paper. The first one is obtained by performing the Viterbi algorithm backwardly on a super-code trellis. This design can secure the maximum-likelihood performance; however, the decoding complexity for small to medium SNRs is larger than that of the stack algorithm with Fano metric. In order to improve the decoding complexity, a second heuristic function is proposed in the same paper, which is generated in a fashion similar to the backward stage of the BCJR algorithm. With negligible performance degradation, the decoding complexity in the forward pass becomes smaller than that of the stack algorithm

with Fano metric for all SNRs. The high computational complexity of the BCJR-like operations, as well as the high memory requirement to store the heuristic function, however becomes a burden for systems with limited computation power and memory space.

In this thesis, we proposed a new method to generate the heuristic function in the backward pass. In our design, the M-algorithm [1] is executed over the backward code tree from the single terminal node back to the 2^L start nodes, where L is the length of the information sequence. It keeps only the best M paths among the $2^k \times M$ successor paths at each level, when (n, k, m) convolutional codes are considered. Among the M best paths ending at the same level, only the smallest path metric is stored, which will be used as the heuristic function value of all nodes in this level in the forward pass. As such, the memory requirement for the heuristic function is greatly reduced when it is compared with the second heuristic function in [9]. Obviously, as M increases, so does the accuracy[1] of the heuristic estimate. Our simulation results indicate that near optimal performance can be achieved by using small M. Also, same as the second heuristic function in [9], our design outperforms the stack algorithm with Fano metric in decoding complexity for all SNRs simulated.

The rest of the thesis is organized as follows. The background of this work is briefed in Chapter 2. The proposed path deletion schemes are presented in Chapter 3. The M-algorithm based heuristics estimation method for use by the two-pass sequential-type decoder is introduced in Chapter 4. Simulation results are summarized and remarked in Chapter 5. Chapter 6 concludes the paper and addresses some future work.

[1]Theoretically, the best heuristic function that can guarantee the finding of the optimal path in the forward pass should take the values of the smallest path metric among *all* paths ending at the same level. Here, we use the smallest path metric among the $2^k \times M$ successors paths instead for binary (n, k, m) convolutional codes. Hence, the "accuracy" of the heuristic function grows as M increases.

Chapter 2

Preliminaries

In this chapter, the system model, the Wagner-rule based metric in [4], the supercode heuristics in [9], as well as the algorithms A and A* are introduced.

2.1 System Models

We consider a binary (n, k, m) convolutional code $\boldsymbol{C}$ with finite input information sequence of $k \times L$ bits, followed by $k \times m$ zeros to clear the encoder memory. Denote a codeword of $\boldsymbol{C}$ by $\boldsymbol{v} \triangleq (v_0, v_1, \ldots, v_{N-1})$, where each $v_j \in \{0, 1\}$, and $N \triangleq n(L + m)$. Assume that $\boldsymbol{v}$ is transmitted over a binary-input time-discrete memoryless channel with channel output $\boldsymbol{r} \triangleq (r_0, r_1, \ldots, r_{N-1})$. Define the hard-decision sequence $\boldsymbol{y} \triangleq (y_0, y_1, \ldots, y_{N-1})$ corresponding to $\boldsymbol{r}$ as:

$$y_j \triangleq \begin{cases} 1, & \text{if } \phi_j < 0; \\ 0, & \text{otherwise,} \end{cases}$$

where

$$\phi_j \triangleq \log \frac{\Pr(r_j|v_j = 0)}{\Pr(r_j|v_j = 1)},$$

and $\Pr(r_j|v_j)$ denotes the channel transition probability of r_j given v_j.

By the Wagner rule, the maximum-likelihood decoding output $\hat{\boldsymbol{v}}$ with respect to the

received vector $\boldsymbol{r}$ satisfies:

$$\hat{\boldsymbol{v}} = \boldsymbol{y} \oplus \boldsymbol{e}^*, \tag{2.1}$$

where $\boldsymbol{e}^*$ is the error pattern with the smallest $\sum_{j=0}^{N-1} e_j|\phi_j|$ among all patterns in $\{0,1\}^N$ satisfying $\boldsymbol{e}\mathbf{H}^T = \boldsymbol{y}\mathbf{H}^T$, and $\mathbf{H}$ is the parity check matrix of the equivalent (N, kL) block code of $\boldsymbol{C}$. Here, "$\oplus$" is the bit-wise exclusive-or operation, and superscript "T" denotes the matrix transpose operation. Based on (2.1), the authors in [4] proved that the sequential search result can be made maximum-likelihood if the Fano metric in the conventional sequential decoding algorithm is replaced by a metric defined as:

$$\nu\left(\boldsymbol{z}_{(\ell n-1)}\right) \triangleq \sum_{j=0}^{\ell n-1} \nu(z_j), \tag{2.2}$$

where $\boldsymbol{z}_{(\ell n-1)} \triangleq (z_0, z_1, \ldots, z_{\ell n-1}) \in \{0,1\}^{\ell n}$ represents the label of a path ending at level ℓ in the convolutional code tree[1] (or trellis), and the bit metric is defined as

$$\nu(z_j) \triangleq (y_j \oplus z_j)|\phi_j|. \tag{2.3}$$

In the rest of this thesis, the bit metric (2.3) is adopted if not particularly specified.

2.2 Algorithm A and Algorithm A*

The algorithms A and A* are sequential-type graph search algorithms. These two algorithms separate the decoding metric $\mu(x_\ell)$ associated with node x_ℓ (at level ℓ) into two portions:

$$\mu(x_\ell) = g(x_\ell) + h(x_\ell), \tag{2.4}$$

where $g(x_\ell)$ accumulates the cost from the start node to node x_ℓ, and $h(x_\ell)$ estimates the remaining cost from node x_ℓ to the terminal node. Function h is called the *heuristic function*.

[1]For clarity, an exemplified $(2, 1, 2)$ convolutional code tree is illustrated in Figure 2.1.

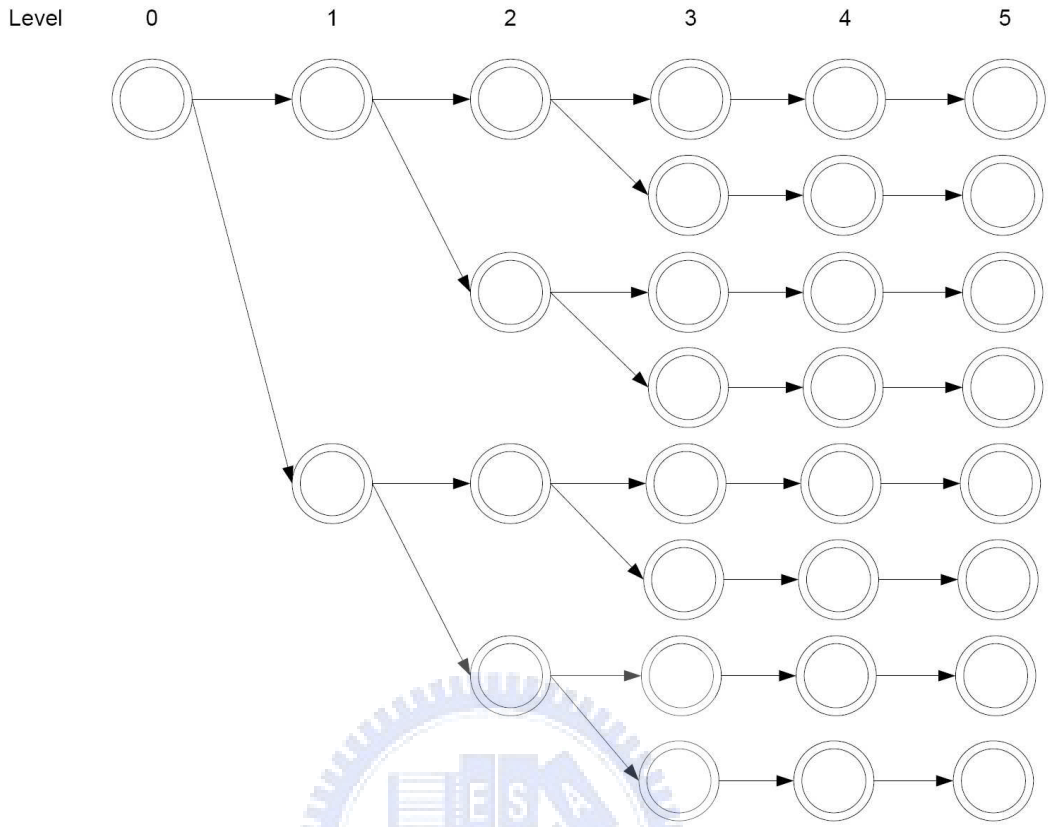


Figure 2.1: An exemplified $(2, 1, 2)$ convolutional code tree with information length $L = 3$.

Suppose $h_{\text{ideal}}(x_\ell)$ is the best cost from node x_ℓ to the respective terminal node. In other words, if $h_{\text{ideal}}(x_\ell)$ is used as the heuristic function in graph search, the algorithm A guarantees the finding of the best solution with minimum search complexity. As such, the search process goes all the way from the start node to the terminal node without diverging from the best path. Nevertheless, such a genie-assisted heuristic function can only be obtained with undue complexity.

The algorithm A*, a variation of the algorithm A, restricts the selections of heuristic function $h^*(x_\ell)$ to

$$h^*(x_\ell) \leq h_{\text{ideal}}(x_\ell). \tag{2.5}$$

It can be shown that as long as the heuristic function satisfies (2.5), the algorithm A*

guarantees to find the best path. In addition, it can be shown that for all heuristic functions satisfying (2.5), the sequential search using a larger heuristic function will expand less number of nodes, and hence, has less decoding complexity.

2.3 Supercode Heuristics of Sikora and Costello

In 2008, Sikora and Costello proposed a two-pass decoder [9], of which the backward pass generates the heuristic function values for use by the forward pass that sequentially search the codeword with the lowest cost.

The main idea of Sikora and Costello's work is to perform the backward pass not on the actual trellis but on a supercode $\boldsymbol{S}$ with a simplified trellis representation $\mathbb{S}$. Since the code $\boldsymbol{C}$ is a proper subset of the supercode $\boldsymbol{S}$, the best cost from node x_ℓ to the respective terminal node $h_{\boldsymbol{S}}(x_\ell)$ must be no larger than the best cost from node x_ℓ to the respective terminal node in the actual code $\boldsymbol{C}$. Hence, $h_{\boldsymbol{S}}(x_\ell)$ satisfies the algorithm A* condition of (2.5), which indicates that the maximum-likelihood performance is still guaranteed. By the proposed proper simplification in the supercode trellis $\mathbb{S}$, performing the Viterbi algorithm backwardly on $\mathbb{S}$ is much simpler than performing the Viterbi algorithm on the actual trellis. However, the decoding complexity for small to medium SNRs is nonetheless larger than that of the stack algorithm with Fano metric.

In order to improve the decoding complexity, a second heuristic function is proposed in the same paper, which is generated in a fashion similar to the backward stage of the well-known BCJR algorithm. The second heuristics function proposed by Sikora and Costello is given by:

$$h^{\alpha}_{\boldsymbol{S}}(\boldsymbol{v}^s_\ell) \;=\; \alpha \log_2 \sum_{(v_0,v_1,\ldots,v_{\ell n-1},\tilde{v}_{\ell n},\ldots,\tilde{v}_{N-1})\in \boldsymbol{S}} 2^{\sum_{i=\ell n}^{N-1}[\log_2 P(r_i|\tilde{v}_i)-R_s+R]/\alpha},$$

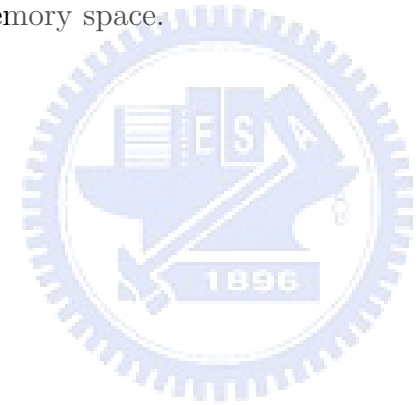
where $\boldsymbol{v}^s_\ell \triangleq (v_0, v_1, \ldots, v_{\ell n-1}) \in \{0,1\}^{\ell n}$ represents the label of a path ending at level ℓ in

the supercode $\boldsymbol{S}$, and R_s is the rate of the supercode $\boldsymbol{S}$. This heuristic function will then be used together with

$$g(x_\ell) = \sum_{i=0}^{\ell n-1} \log_2 P(r_i|v_i)$$

in the forward pass.

By varying the parameter α between 0 and 1, they found a trade-off between the decoding complexity in the forward pass and the performance degradation. With the proper α, the decoding complexity in the forward pass can then be made smaller than the stack algorithm with Fano metric for all SNRs under negligible performance degradation. However, the high computational complexity of the BCJR-like operations, as well as the high memory requirement to store the heuristic function, becomes a burden for systems with limited computation power and memory space.



Chapter 3

Path Deletion Schemes for Limited Stack-Size Sequential-Type Decoding Algorithm

By assuming a limited stack size for sequential-type decoding algorithm, some of the paths have to be deleted when the stack exceeds its upper limit. A deletion scheme therefore is essential such that the paths that are more unlikely to be the maximum-likelihood one should be dropped first. In this chapter, we will introduce and examine several deletion schemes.

3.1 Path Deletion Based on ML Path Metric

In 2002, a metric for use by the sequential-type decoding algorithm was derived based on the Wagner rule [4]. The authors in [4] also proved that applying this metric in place of Fano metric in the sequential decoding search guarantees ML performance. A straightforward path deletion scheme is thus to drop the one with the largest ML path metric among all paths currently in the stack. We refer this scheme as the *ML metric-based path deletion scheme*. For completeness, the decoding procedure with ML metric-based deletion scheme is illustrated below.

Step 1. Load the stack with the path consisting of the single start node x_0, whose ML metric $\mu(x_0) = 0$. Set $\ell = 0$.

Step 2. Extend the top path in the stack to obtain its successor paths. Compute the ML path metric of each successor path. Delete the top path from the stack.

Step 3. Insert the successors into the stack and reorder the stack according to ascending ML path metric values μ. If the stack size exceeds its limit, recursively eliminate the one whose ML metric is the largest until the stack size $\leq$ limit.

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

3.2 Path Deletion Based on Path Levels

In 2007, the authors in [8] proposed a computational complexity reduction method for their previously proposed sequential-type decoding algorithm [4]. Specifically, their scheme drops the paths of which the levels are smaller than the largest level being reached by the sequential search minus a level threshold Δ. The intuition behind their scheme is that the path with the smallest level among all paths currently in the stack usually tends to accumulate large ML metrics after reaching the explored level (since the ML path metric is non-decreasing along the path it traverses). Hence, we may for convenience simply delete the path with the smallest level when the stack exceeds its limit. Notably, the authors in [8] do not assume a finite stack size in their analysis and simulation, but apply their method to reduce the computational complexity. In this thesis, we re-examine their idea by adding a finite stack size assumption. We refer this scheme as the *Level-based path deletion scheme*. For completeness, the decoding procedure with level-based deletion scheme is illustrated below.

Step 1. Load the stack with the path consisting of the single start node x_0, whose ML metric $\mu(x_0) = 0$. Set $\ell = 0$.

Step 2. Extend the top path in the stack to obtain its successor paths. Compute the ML path metric of each successor path. Delete the top path from the stack.

Step 3. Insert the successors into the stack and reorder the stack according to ascending ML path metric values μ. If the stack size exceeds its limit, recursively eliminate the one whose level is the smallest until the stack size $\leq$ limit.

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

3.3 Path Deletion Based on Fano Metric

The ML metric-based path deletion scheme in Section 3.1 drops the path with the largest path metric when the stack is full. However, the path metrics that the deletion scheme is based only accumulate those branch metrics that have been traversed thus far, and no prediction on the future route is performed. Now if we can replace the ending level by a metric that includes future route prediction, and use it as a base for path deletion, the performance may be improved.

Based on Massey's analysis in [6], Fano metric in [3] can be regarded as an average of the future branch metrics, which have not been traversed. With this in mind, Fano metric may be a good candidate to serve as the metric that the deletion scheme is based. We refer this scheme as the *Fano metric-based path deletion scheme*. For completeness, the decoding procedure with Fano metric-based deletion scheme is illustrated below.

Step 1. Load the stack with the path consisting of the single start node x_0, whose ML

metric $\mu(x_0) = 0$. *Set* $\ell = 0$.

Step 2. Extend the top path in the stack to obtain its successor paths. Compute the ML path metric and Fano path metric of each successor path. Delete the top path from the stack.

Step 3. Insert the successors into the stack and reorder the stack according to ascending ML path metric values μ. *If the stack size exceeds its limit, recursively eliminate the one with the largest Fano path metric until the stack size* $\leq$ *limit.*

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

3.4 Summary

The path deletion schemes introduced in this chapter are designed to target the goal that the path that is the most unlikely to belong to the final ML decision is located and dropped when the stack is full. Simulations in Section 5.1 show that given a finite stack size, the Fano metric-based deletion scheme has better performance than the other two schemes. As far as the computational complexity (i.e., the average number of ML metric computations per information bit, or the average number of path expansions per information bit) is concerned, the level-based deletion scheme becomes superior over the other two. Here, we exclude the additional burden to maintain two indices for the stack in level-based and Fano metric-based deletion schemes, which may not be practical. In order to alleviate such additional burden, we will introduce an alternative decoding algorithm that use the same metric in path expansion and path deletion, for which both the error rate and computational complexity are comparable to the best among the three schemes just introduced, but without the burden to maintain additional index for the stack.

Chapter 4

A Novel Two-Pass Sequential-type Decoding Algorithm

This chapter will introduce a novel two-pass sequential-type decoding algorithm to alleviate the decoding burden in both computational complexity and memory management. The proposed algorithm is designed based on the well-known algorithm A [7]. In short, we will devise an appropriate heuristic estimate in order to reduce decoding complexity with negligible error performance loss. In addition, the heuristic estimate will simultaneously suit the need for path deletion when the stack is full.

4.1 Heuristics Analysis

In this section, we analyze the heuristic function for the additive white Gaussian noise (AWGN) channels. Our analysis will hint that a one-pass on-the-fly decoding procedure may not be able to result in a *good* heuristic estimate that can improve the complexity without sacrificing the performance.

For AWGN channels, the ML decision upon the reception of a received vector $\boldsymbol{r} \triangleq$

$(r_0, r_1, \ldots, r_{N-1})$ is given by:

$$\begin{aligned}
\hat{\boldsymbol{v}} &= \arg\min_{\boldsymbol{v}\in\boldsymbol{C}} \|\boldsymbol{r}-\boldsymbol{v}\|^2 \\
&= \arg\min_{\boldsymbol{v}\in\boldsymbol{C}} \sum_{i=0}^{N-1} (r_i - v_i)^2 \\
&= \arg\min_{\boldsymbol{v}\in\boldsymbol{C}} \left(\sum_{i=0}^{N-1} r_i^2 - \sum_{i=0}^{N-1} 2r_i v_i + \sum_{i=0}^{N-1} v_i^2 \right) \\
&= \arg\min_{\boldsymbol{v}\in\boldsymbol{C}} \sum_{i=0}^{N-1} (-r_i v_i)\,.
\end{aligned}$$

The maximum-likelihood decision remains unchanged by adding a constant, independent of the codeword $\boldsymbol{v}$; hence, a constant is added to make non-negative the decision criterion as

$$\begin{aligned}
\hat{\boldsymbol{v}} &= \arg\min_{\boldsymbol{v}\in\boldsymbol{C}} \sum_{i=0}^{N-1} (-r_i v_i) \\
&= \arg\min_{\boldsymbol{v}\in\boldsymbol{C}} \sum_{i=0}^{N-1} |r_i v_i| - \sum_{i=0}^{N-1} r_i v_i \\
&= \arg\min_{\boldsymbol{v}\in\boldsymbol{C}} \sum_{i=0}^{N-1} (|r_i| - r_i v_i).
\end{aligned}$$

Since we desire an on-the-fly decoding metric, the accumulated path metric corresponding to path $\boldsymbol{v}_\ell$ is:

$$g(\boldsymbol{v}_\ell) = \sum_{i=0}^{\ell-1} (|r_i| - r_i v_i).$$

Then, it is clear that g can be recursively computed as:

$$g(\boldsymbol{v}_{\ell+1}) = g(\boldsymbol{v}_\ell) + (|r_\ell| - r_\ell v_\ell)$$

It is known that in order to maintain optimality through sequential decoding search, it suffices to have the heuristic function satisfying:

$$\mu(\boldsymbol{v}_\ell) = g(\boldsymbol{v}_\ell) + h(\boldsymbol{v}_\ell) \leq \min_{\boldsymbol{v}'_N = (v_0, v_1, \ldots, v_{\ell-1}, v'_\ell, v'_{\ell+1}, \ldots, v'_{N-1}) \in \boldsymbol{C}} [g(\boldsymbol{v}'_N) + h(\boldsymbol{v}'_N)]$$

in addition to $h(\boldsymbol{v}'_N) = 0$. This results in:

$$h(\boldsymbol{v}_\ell) \leq \min_{\boldsymbol{v}'_N=(v_0,v_1,\ldots,v_{\ell-1},v'_\ell,v'_{\ell+1},\ldots,v'_{N-1})\in\boldsymbol{C}} \left[\sum_{i=\ell}^{N-1}(|r_i| - r_i v'_i)\right].$$

Without the knowledge of the code structure, we obtain:

$$h(\boldsymbol{v}_\ell) \leq \min_{(v'_\ell,v'_{\ell+1},\ldots,v'_{N-1})\in\{\pm 1\}^{N-\ell}} \left[\sum_{i=\ell}^{N-1}(|r_i| - r_i v'_i)\right] = 0.$$

As a result, the largest heuristic function is the zero-heuristics function. This hints that as aforementioned, an on-the-fly decoding procedure may not be able to result in a heuristic estimate that can improve the complexity without sacrificing the performance. We therefore turn our efforts to the off-line decoding algorithm in the next section, for which a heuristic estimate is obtained through the knowledge of all receptions.

4.2 M-algorithm-Based Heuristics Estimate

In this section, we will introduce our proposed *M-algorithm-based heuristics estimation method (MHEM)*. Similar two-pass decoding idea has been appeared in [9]. Here, the key difference of our work from [9] is that a new heuristic function is designed. Besides, our two-pass decoding is performed respectively on the forward and backward code trees rather than the supercode in [9].

The procedure to obtain our heuristic estimate h in the first pass can be described as follows. Initially, we set the heuristic function value of the single terminal node on the backward code tree as $h_{L+m} = 0$. Then, we execute the M-algorithm from the single terminal node at level $L + m$ back to the 2^L start nodes at level 0, using the metric defined in (2.2). The heuristic function h_ℓ is the the minimum path metric value among the M survivor path metrics at level ℓ. It should be noted that our heuristic function, when it is applied to the forward decoding pass, may not lead to the ML decision especially for small M. In the most

extreme case, where $M = 2^L$, the optimal performance is actually guaranteed. We will show by simulations that near optimal performance can be achieved for small to moderate M.

To be specific, we give an example of the backward pass in Figure 4.1. The heuristic function values resulted are listed in Table 4.1.

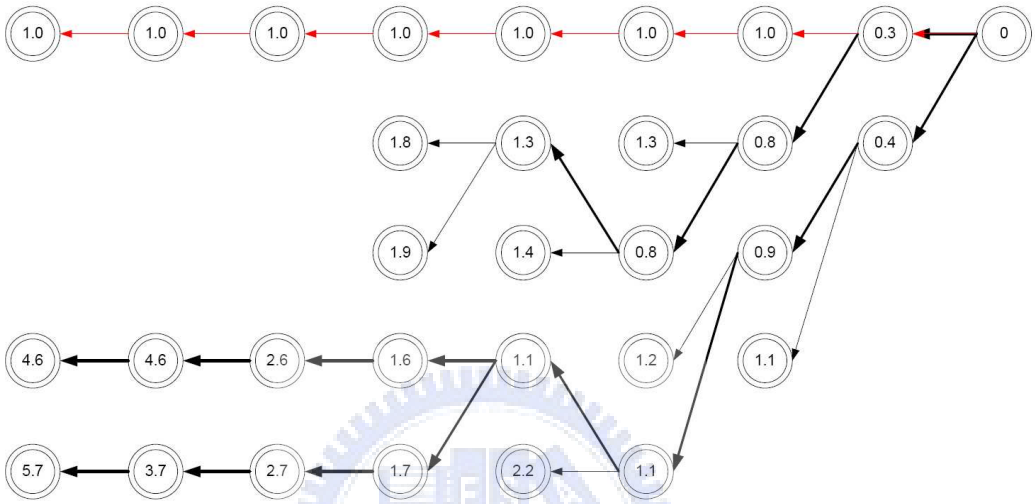


Figure 4.1: An example of M-algorithm-based heuristic function generation for $(2, 1, 3)$ convolutional code of length $N = n(L + m) = 2(5 + 3) = 16$. The maximum-likelihood code word path is marked in red.

Table 4.1: The heuristic function values resulted from Figure 4.1.

Level ℓ	0	1	2	3	4	5	6	7	8
$h_\ell(M = 2)$	4.6	4.6	2.6	1.6	1.1	0.8	0.8	0.3	0.0
$h_\ell(M = 2^L)$	1.0	1.0	1.0	1.0	1.0	0.8	0.8	0.3	0.0

After we obtain the heuristic function, the forward pass performs the usual stack algorithm with metric $\mu(x_\ell)$ defined in (2.4), where $g(x_\ell)$ is the accumulated metric from the start node to node x_ℓ according to (2.2), and $h(x_\ell) = h_\ell$ that is obtained from the previous backward pass. For clarity, we summarize the decoding procedure as follows.

Backward pass:

Step 1. *Set $\ell = L + m$ and $h_\ell = 0$. Let the path consisting of the single terminal node on the backward code tree be the single survivor path for M-algorithm at level ℓ.*

Step 2. *Backwardly extend all successors of the survivor paths at level ℓ. Recursively calculate the path metrics of the successor paths according to (2.2). Set $h_{\ell-1}$ to be the minimum path metric among all successor paths.*

Step 3. *Retain the M successor paths with the smallest path metrics as the new survivor paths, and let $\ell = \ell - 1$.*

Step 4. *If $\ell = 0$, the backward pass stops; otherwise go to Step 2.*

Forward pass:

Step 1. *Load the stack with the path consisting of the single start node x_0, whose metric $\mu(x_0) = g(x_0) + h_0 = h_0$. Set $\ell = 0$.*

Step 2. *Extend the top path in the stack to obtain its successor paths. Compute the path metric of each successor path according to (2.4). Delete the top path from the stack.*

Step 3. *Insert the successors into the stack and reorder the stack according to ascending path metric values μ. If the stack size exceeds its limit, recursively eliminate the one whose μ metric is the largest until the stack size $\leq$ limit.*

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

Chapter 5

Simulation Results

In this chapter, we will examine our proposed schemes in Chapters 3 and 4. The channel model simulated is the additive white Gaussian noise (AWGN) channel.

5.1 The Path Deletion Schemes in Chapter 3

In this section, we examine the path deletion schemes we proposed in Chapter 3. Also examined is the path deletion scheme that uses the Fano metric to select the top path to be extended, and to delete the path when the stack is full. This scheme is referred to as *Fano metric delete by Fano metric* in the legend. By following similar naming rule, the *Fano metric-based path deletion scheme* is referred to as *MLSDA delete by Fano metric*, and the *level-based path deletion scheme* is briefed as *MLSDA delete by path level* in the legend. The *ML metric-based path deletion scheme* is likewisely referred to as *MLSDA delte by Wagner metric*.

We compare these path deletion schemes using $(2, 1, 8)$ convolutional codes with generator polynomial [457,755] (in octal) and $(2, 1, 12)$ convolutional codes with generator polynomial [17663,11271] (in octal). Simulations for $(2, 1, 8)$ convolutional code are summarized in Figures 5.1–5.9.

Figures 5.1–5.6 show that Fano metric-based path deletion scheme is the best in error performance when the stack size is upper-bounded by 2^6-1 and 2^8-1. The ML metric-based path deletion scheme performs apparent worse than all the other schemes, and also requires the largest computational complexity. More specifically, we observe from Figure 5.1 that the word error rate (WER) of the *MLSDA delete by Fano Metric* performs 0.5 dB better than the *MLSDA delete by level* when the stack size is limited to 2^6-1. In comparison with the *Fano metric delete by Fano metric*, *MLSDA delete by Fano Metric* is still 0.2 dB superior in WER performance. However, Figure 5.3 shows that the computational complexity of the *MLSDA delete by Fano Metric* is around two times larger than both *Fano metric delete by Fano metric* and *MLSDA delete by level*. This leads to a tradeoff between error rate and computational complexity when one wishes to select among these schemes.

When the stack size is enlarged to a big size, e.g., $2^{16}-1$, Figures 5.7–5.9 show that near optimal performance can be achieved by all three schemes we proposed in Chapter 3. Yet, with a large stack size, the average computational complexity of the three proposed schemes will become much larger than *Fano metric delete by Fano metric*.

For (2,1,12) convolutional codes with generator polynomial [17663,11271] (in octal), we observe from Figures 5.10–5.15 that the proposed *MLSDA delete by Fano Metric* is still the best in error performance when stack size is limited to $2^{12}-1$, but its computational complexity is again higher than *Fano metric delete by Fano metric* and *MLSDA delete by level*. Figures 5.13–5.15 indicates that optimality can be achieved when the stack size further increases up to $2^{16}-1$.

From the above simulations, we conclude that *Fano metric-based path deletion scheme* can achieve better performance when the memory saving is critical in the system design.

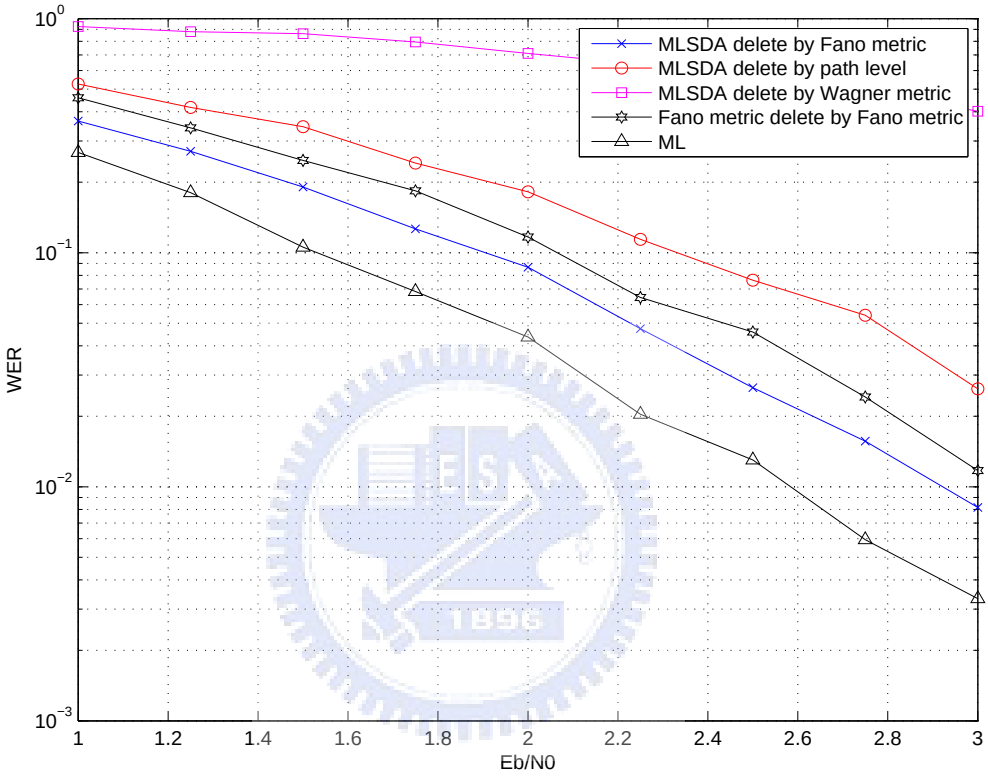


Figure 5.1: Word error rate (WER) performance of path deletion schemes for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is 2^6-1, and the information sequence length $L = 100$.

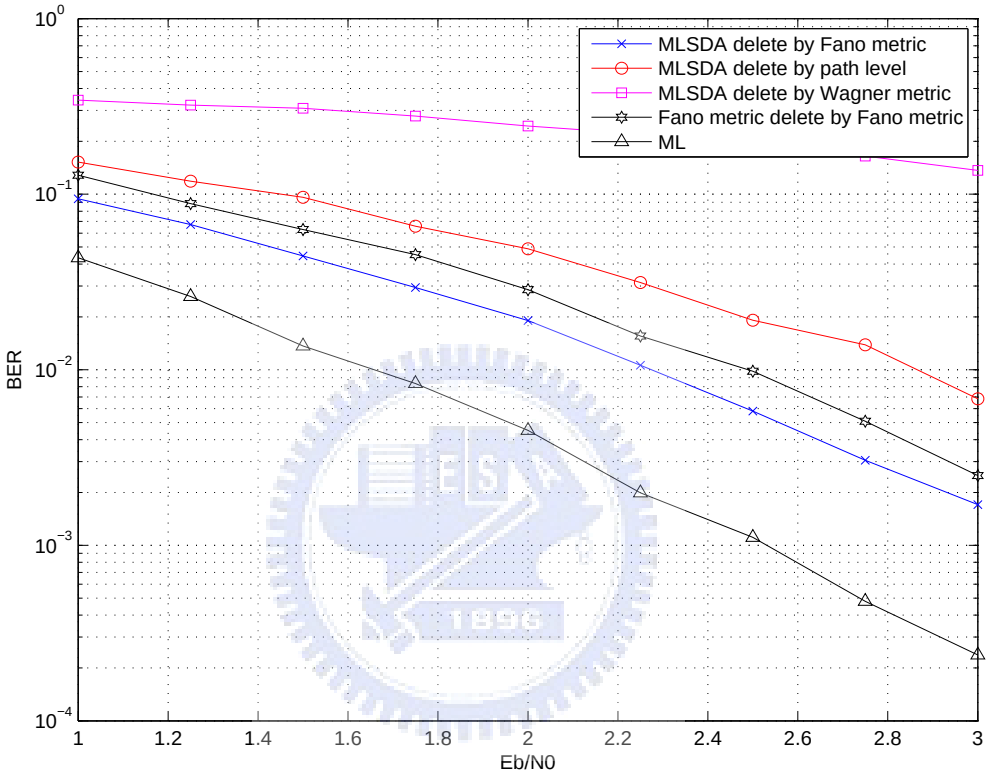


Figure 5.2: Bit error rate (BER) performance of path deletion schemes for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is 2^6-1, and the information sequence length $L = 100$.

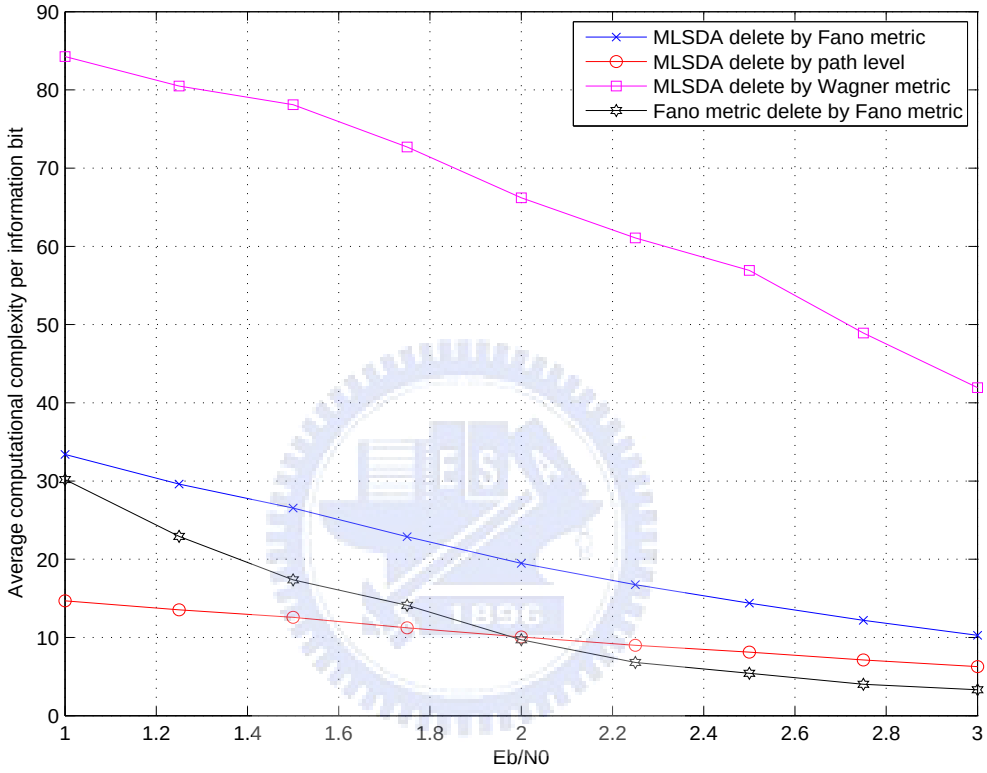


Figure 5.3: Average computational complexity per information bit of path deletion schemes for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $2^6 - 1$, and the information sequence length $L = 100$.

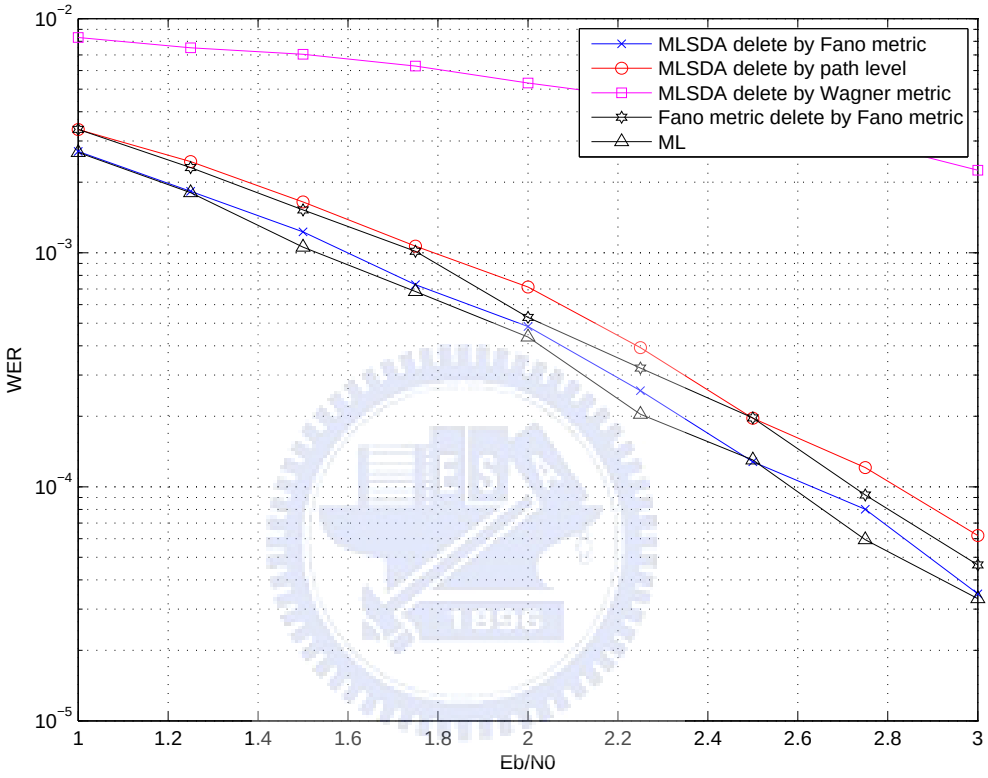


Figure 5.4: Word error rate (WER) performance of path deletion schemes for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $2^8 - 1$, and the information sequence length $L = 100$.

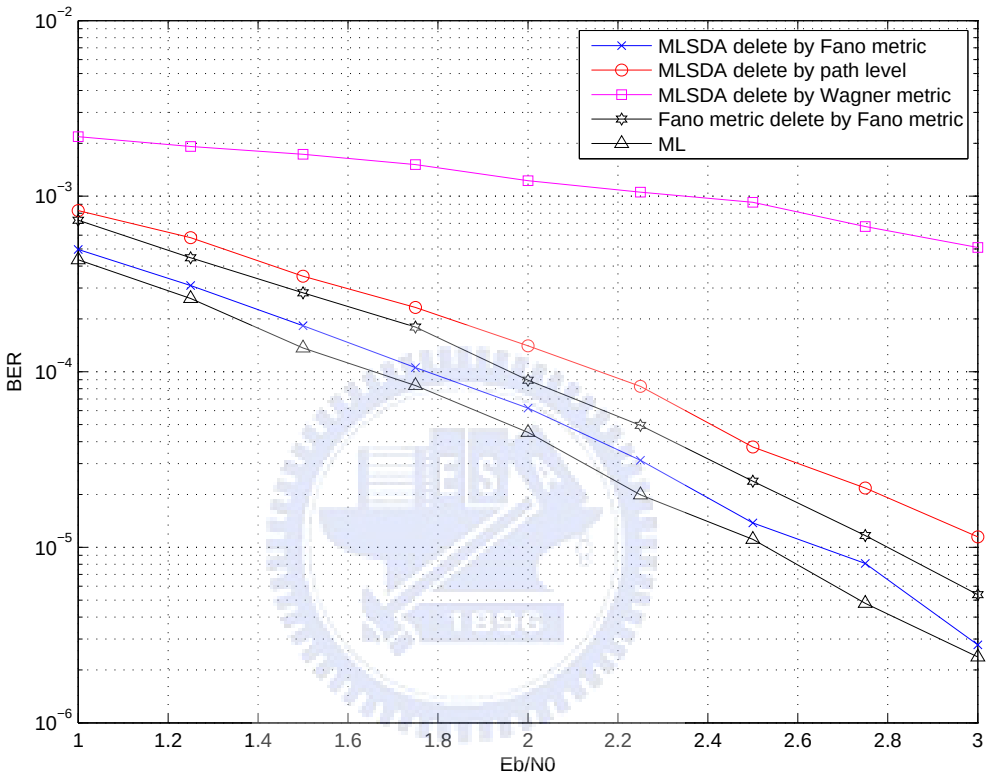


Figure 5.5: Bit error rate (BER) performance of path deletion schemes for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $2^8 - 1$, and the information sequence length $L = 100$.

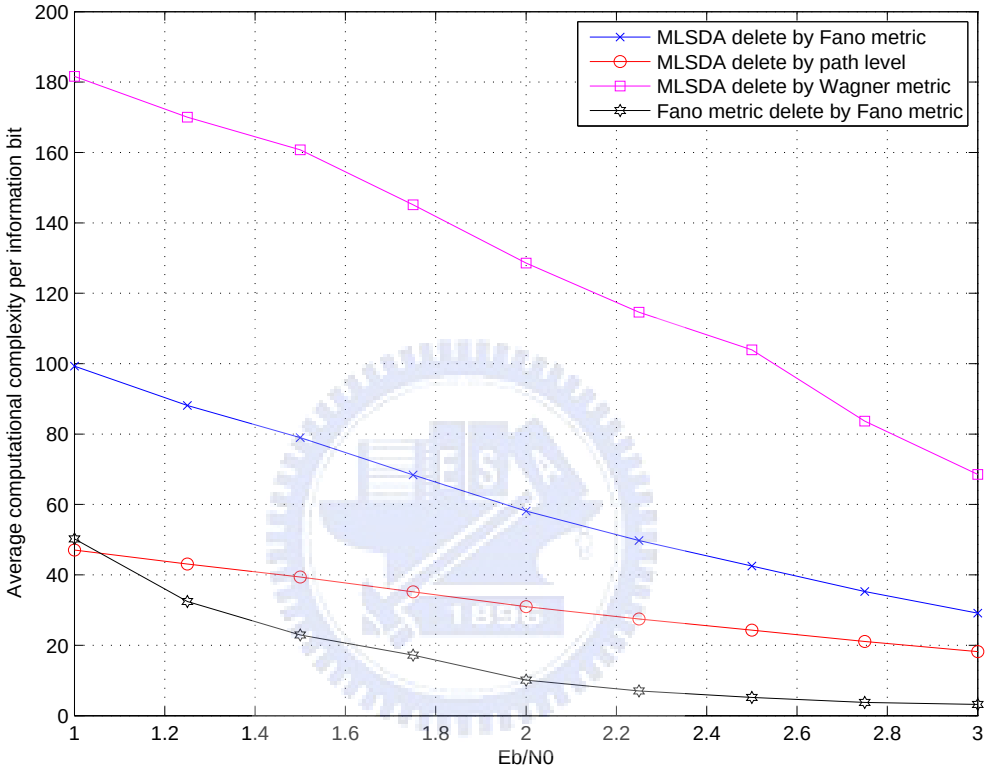


Figure 5.6: Average computational complexity per information bit of path deletion schemes for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $2^8 - 1$, and the information sequence length $L = 100$.

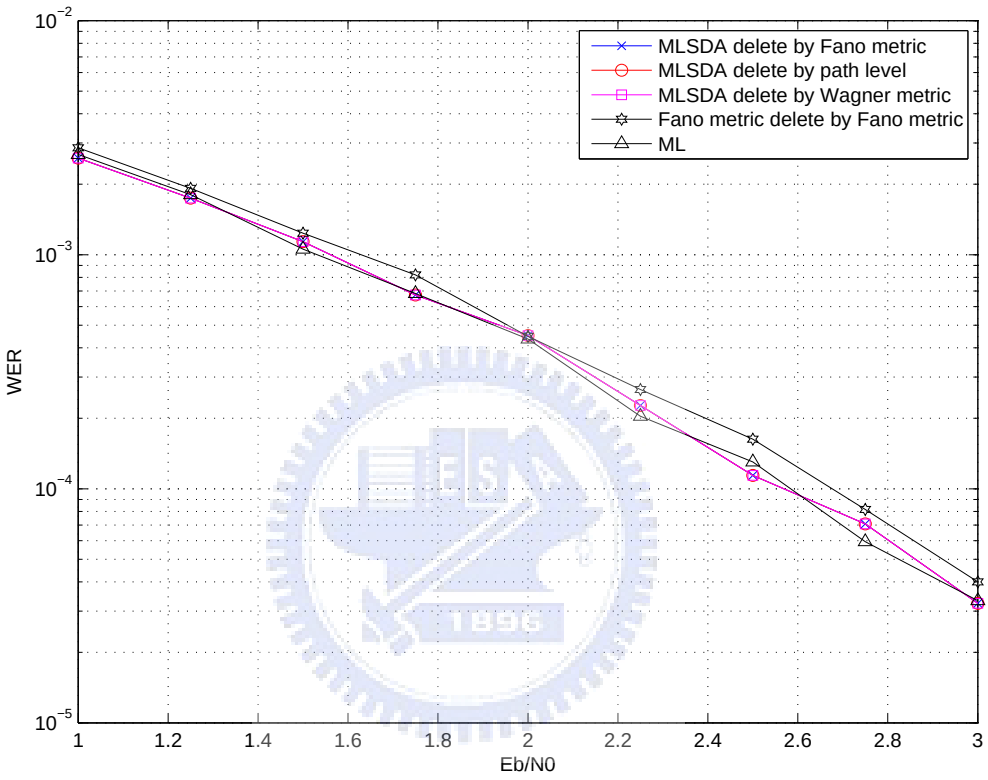


Figure 5.7: Word error rate (WER) performance of path deletion schemes for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $2^{16} - 1$, and the information sequence length $L = 100$.

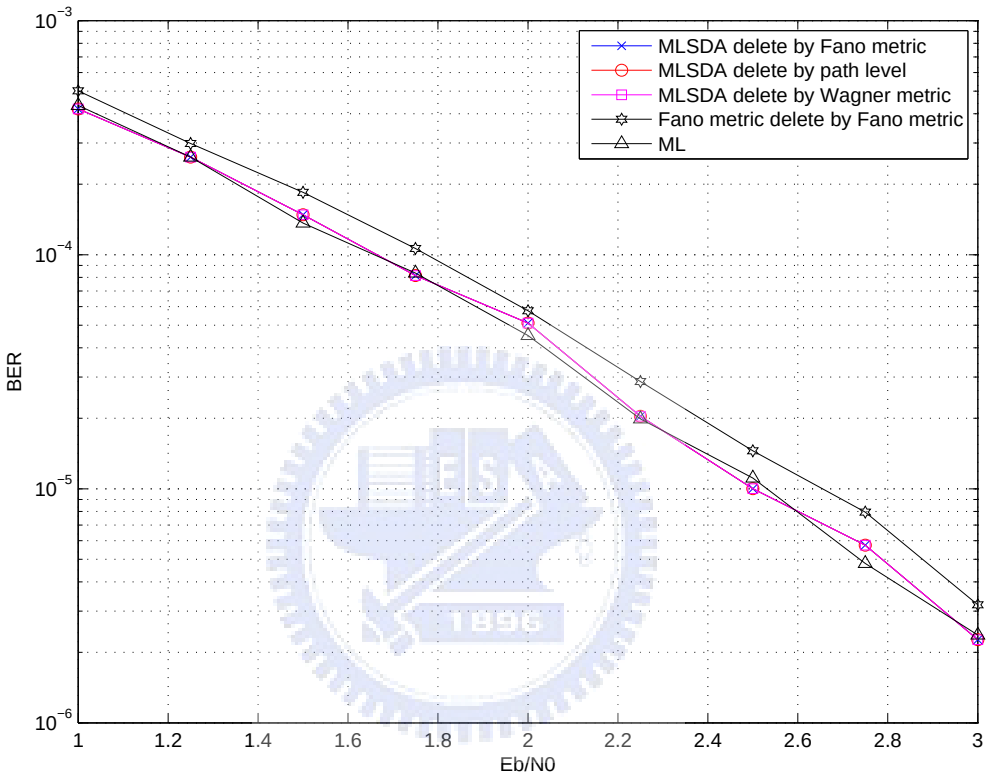


Figure 5.8: Bit error rate (BER) performance of path deletion schemes for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $2^{16}-1$, and the information sequence length $L = 100$.

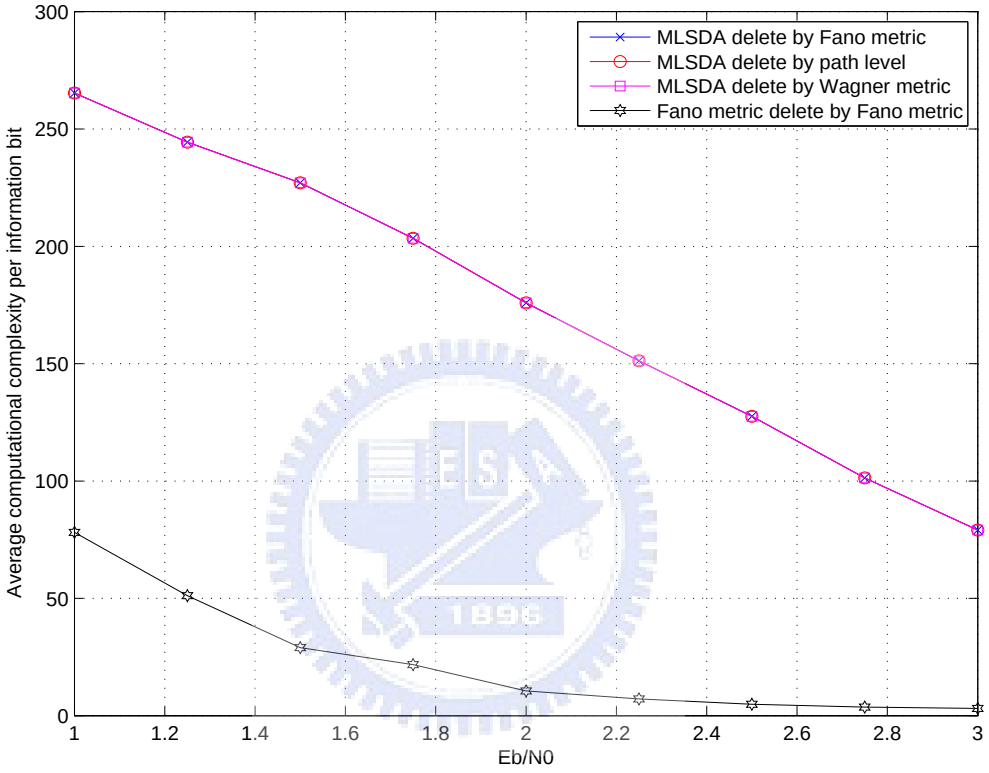


Figure 5.9: Average computational complexity per information bit of path deletion schemes for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $2^{16}-1$, and the information sequence length $L = 100$.

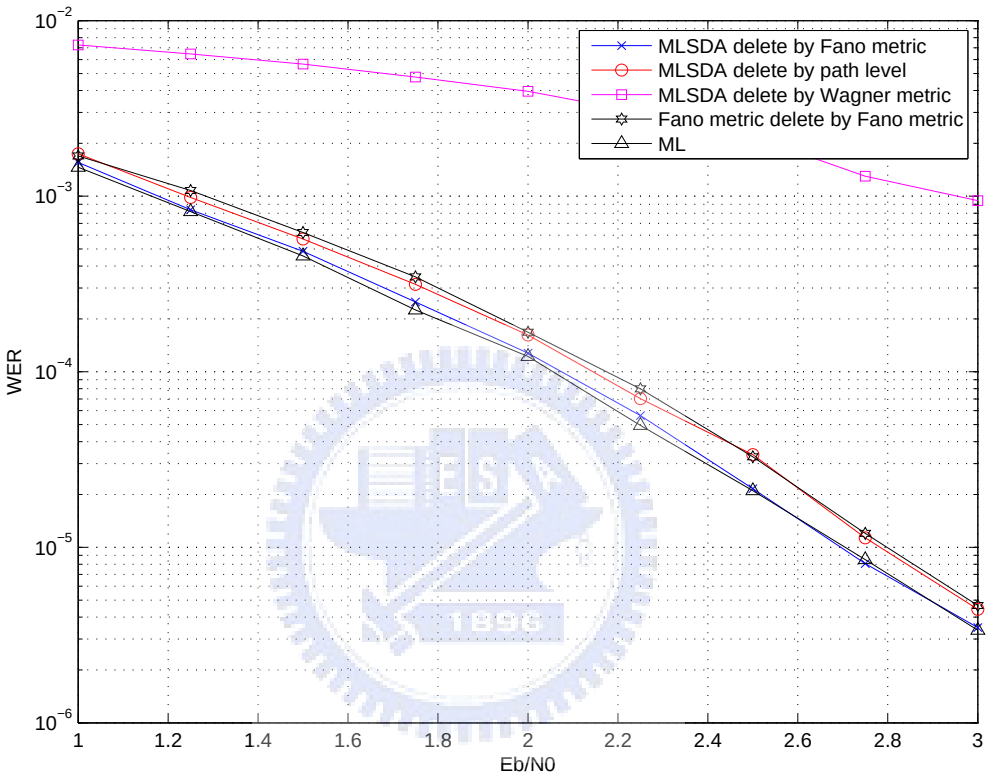


Figure 5.10: Word error rate (WER) performance of path deletion schemes for (2,1,12) convolutional code with generator polynomial [17663,11271]. The stack size is $2^{12} - 1$, and the information sequence length $L = 100$.

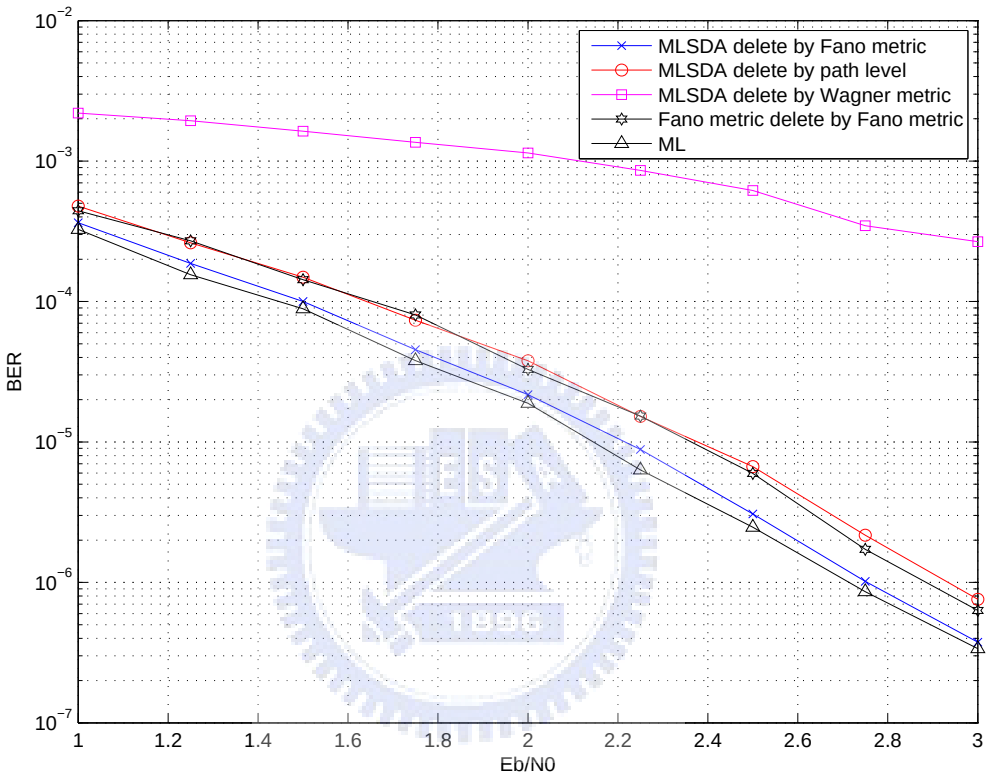


Figure 5.11: Bit error rate (BER) performance of path deletion schemes for (2,1,12) convolutional code with generator polynomial [17663,11271]. The stack size is $2^{12} - 1$, and the information sequence length $L = 100$.

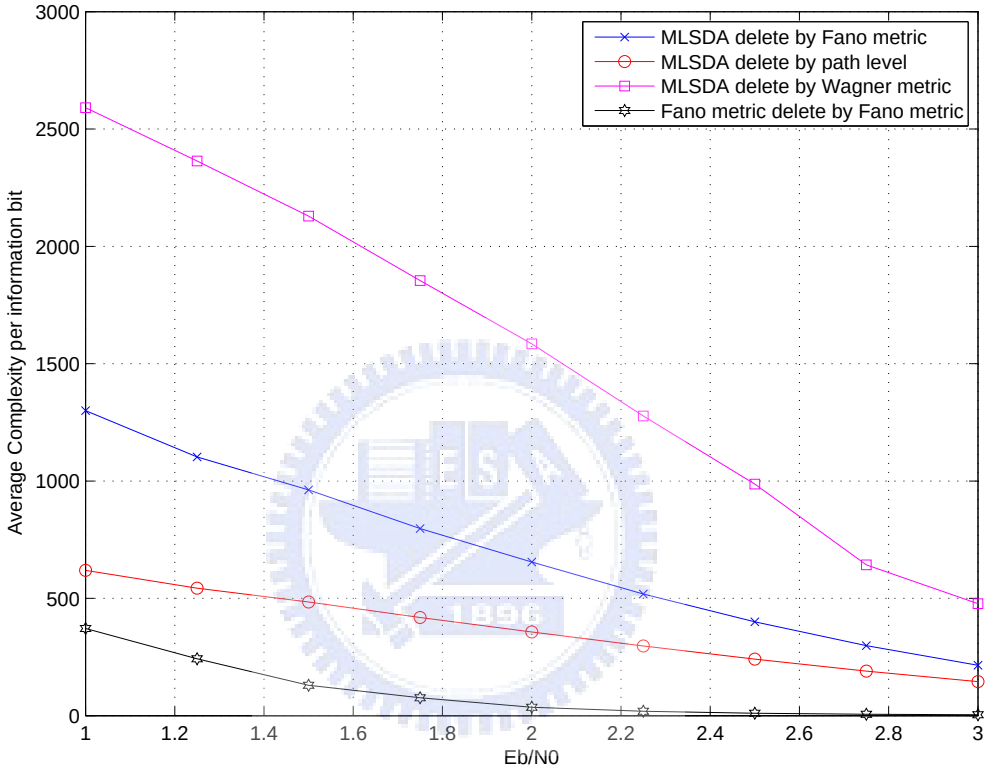


Figure 5.12: Average computational complexity per information bit of path deletion schemes for (2,1,12) convolutional code with generator polynomial [17663,11271]. The stack size is $2^{12} - 1$, and the information sequence length $L = 100$.

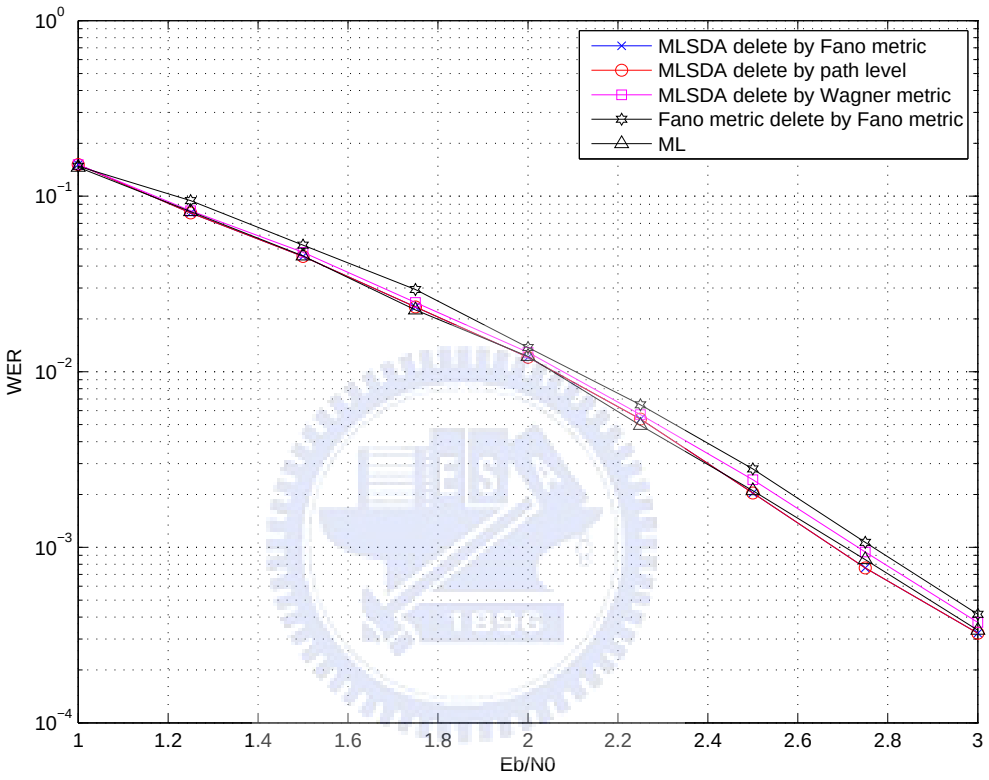


Figure 5.13: Word error rate (WER) performance of path deletion schemes for (2,1,12) convolutional code with generator polynomial [17663,11271]. The stack size is $2^{16} - 1$, and the information sequence length $L = 100$.

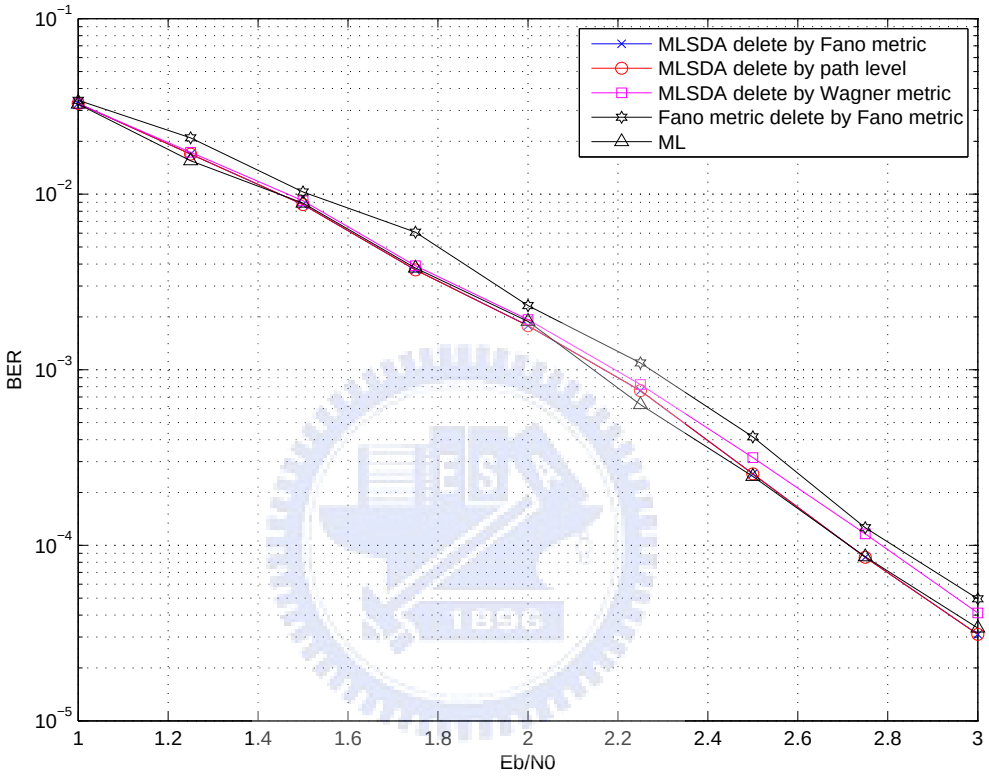


Figure 5.14: Bit error rate (BER) performance of path deletion schemes for (2,1,12) convolutional code with generator polynomial [17663,11271]. The stack size is $2^{16} - 1$, and the information sequence length $L = 100$.

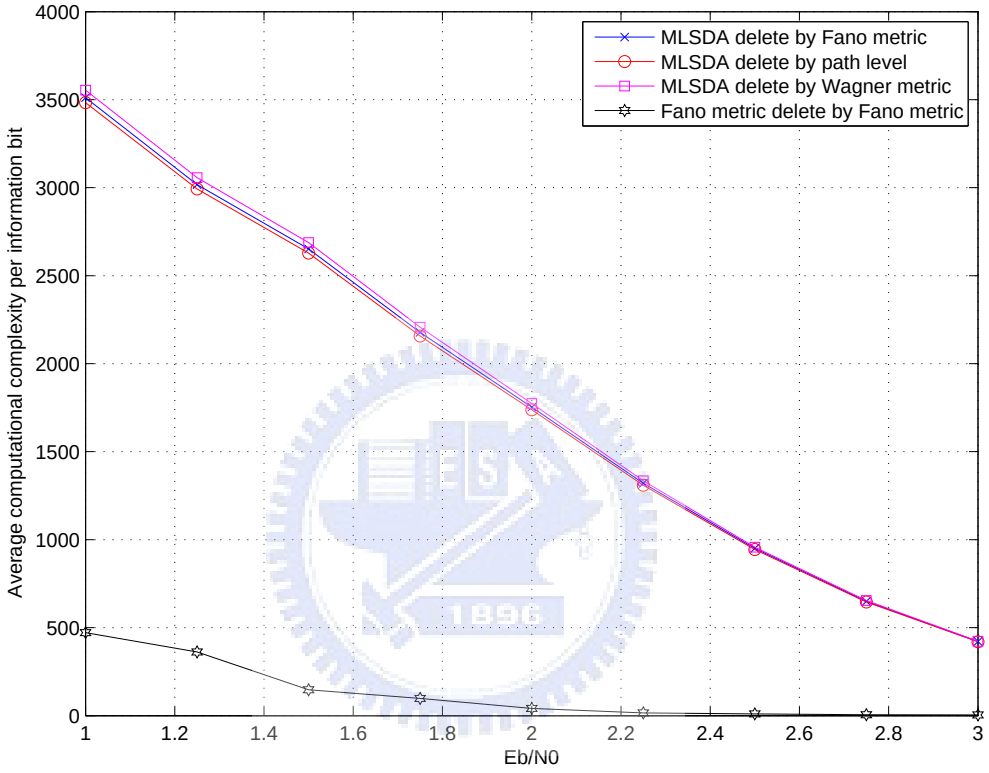


Figure 5.15: Average computational complexity per information bit of path deletion schemes for (2,1,12) convolutional code with generator polynomial [17663,11271]. The stack size is $2^{16} - 1$, and the information sequence length $L = 100$.

5.2 The Two-Pass Sequential-Type Decoding Algorithm in Chapter 4

In this section, we turn to examine the performance of the *M-algorithm-based heuristics estimation method (MHEM)*. Since the routine work in the backward pass can be hardware-implemented [2][10], only the computational complexity in the forward pass is recorded in the following simulations. Same premise is also made in [9].

From Figures 5.16–5.18, we found that the performance of the MEHM-enhanced two-pass decoder is limited by the small stack size $2^4 - 1$, i.e., the performance cannot be improved by increasing M. In comparison with *Fano metric delete by Fano metric*, the MHEM-enhanced two-pass decoder with small stack size is still 0.5 dB better at WER= 10^{-1}. Moreover, the computational complexity of the MHEM-enhanced two-pass decoder is 2/3 times lower than *Fano metric delete by Fano metric* at Eb/N0 = 2.0 dB. Since the small stack size in these simulations limits the performance, and since the small stack size also controls the average computational complexity, M becomes an irrelevant parameter in the system.

By further increasing the stack size up to $2^6 - 1$, Figures 5.19–5.21 show that the larger M is, the better the performance. Interestingly, the computational complexities for all M simulated in these three figures are lower than the *Fano metric delete by Fano metric*.

For stack sizes of $2^8 - 1$ and $2^{10} - 1$, Figure 5.22–Figure 5.27 show that taking $M = 64$ has less than 0.1 dB improvement over $M = 32$ in error performance. Hence, taking $M = 32$ is sufficient in the sense that no apparent improvement can be obtained by further increasing M. In comparison with *Fano metric delete by Fano metric*, our simulation indicates that it has comparable performance with our ($M = 32$)-MHEM-enhanced two-pass decoder, but requires two times larger computational complexity at Eb/N0= 2.0 dB.

The same phenomenon can be observed in Figures 5.28–5.51, in which the lengths of

information sequence increase to $L = 200$ and $L = 400$. In other words, these figures show that no apparent performance improvement can be obtained after $M > 32$, and the average computational complexity of *Fano metric delete by Fano metric* is much higher than the $(M = 32)$-MHEM-enhanced two-pass decoder.

We also investigate the relation between M and the stack size. We found from Figures 5.52–5.54 that the proposed MHEM-enhanced two-pass decoder is more insensitive to the stack size than the *Fano metric delete by Fano metric*.

Figures 5.55–5.63 continue to examine the proposed MHEM-enhanced two-pass decoder using $(2, 1, 12)$ convolutional codes with generator polynomial [17663,11271] (in octal). For length of information sequence $L = 200$, we observe that the performance improvement of $M = 256$ over $M = 128$ is less than 0.1 dB, and the stack size to have less than 0.1 dB performance degradation from the $(M = 128)$-MHEM-enhanced two-pass decoder to the optimal performance is $2^{10} - 1$.

We again do the same examination for $(2, 1, 16)$ convolutional code with information length $L = 100$. The results are summarized in Figures 5.64–5.66. The simulations show that with $M = 512$ and stack size of $2^{12} - 1$, the performance degradation with respect to the ML performance can be made less than than 0.3 dB. Moreover, our $(M = 512)$-MHEM-enhanced two-pass decoder can perform 0.2 dB better than the *Fano metric delete by Fano metric* with seemingly smaller decoding complexity.

All the previous simulations consider code word length of hundred bits. What if the code word length extends to thousand bits? This query is answered by the following simulations. We re-examine our decoder for $(2, 1, 8)$ convolutional codes with generator polynomial [457,755] (in octal) by taking $L = 2048$. The results are summarized in Figures 5.67–5.69 It can be observed from Figure 5.67 that the performance degradation with respect to ML decoding is negligible for $M = 64$. Even for $M = 32$ and $M = 16$, the performance degradations

are still less than 0.1 dB and 0.3 dB, respectively. In these simulations, the performances for the low-complexity suboptimal decoding algorithm proposed in [9] respectively using 16- and 32-state supercodes are also illustrated. The parameters α are set as 0.94 and 0.9 for 16- and 32-state supercodes, respectively, as suggested in Table I in [9].[1] The figure indicates that the performance degradations of Sikora and Costellos's algorithm can also be made less than 0.1 dB with the parameter α chosen.

We observe from Figure 5.69 that our decoder still has much less computational complexity than the *Fano metric delete by Fano metric* (which is now termed as *Fano metric* in the legend). In particular, the decoding complexity of our algorithm remains small at low SNR whereas that of the stack algorithm with Fano metric become almost unbounded when the code word length is large. In comparison with the low-complexity suboptimal decoding algorithm using 32-state supercode in [9], the average decoding complexity of our algorithm ($M = 32$) is about 2/3 times lower at $E_b/N_0 = 1.5$ dB.

It is worth mentioning that the backward M-algorithm in the first pass may have already generated reliable output. This can be examined through the error detection coding technique such as CRC. In such case, no forward pass is necessary. In Figure 5.69, we observe that the decoding complexity can be further reduced, e.g., from 2.54 down to 0.54 per information bit at $E_b/N_0 = 2.5$ dB, provided that perfect error detection scheme is employed.

In the end, it is interesting to note from Figures 5.25 and 5.67 that when the stack size is sufficiently large, the selection of M that can achieve near optimal performance is independent of the code word length, which is 32 in both cases. Hence, the factor that controls the M selected for near optimal performance is more relevant to the code itself, but not on the code word length.

[1]It should be noted that in the forward pass, Sikora and Costello's algorithm extends only *one* code bit in each iteration, instead of *two* code bits (i.e., one tree branch for half-rate codes) per iteration as our proposed algorithm does. Since it requires two code bits to identify a branch on the tree of half-rate codes, we will count two (code bit) extensions on the same branch as one (branch) metric computation in Figure 5.69.

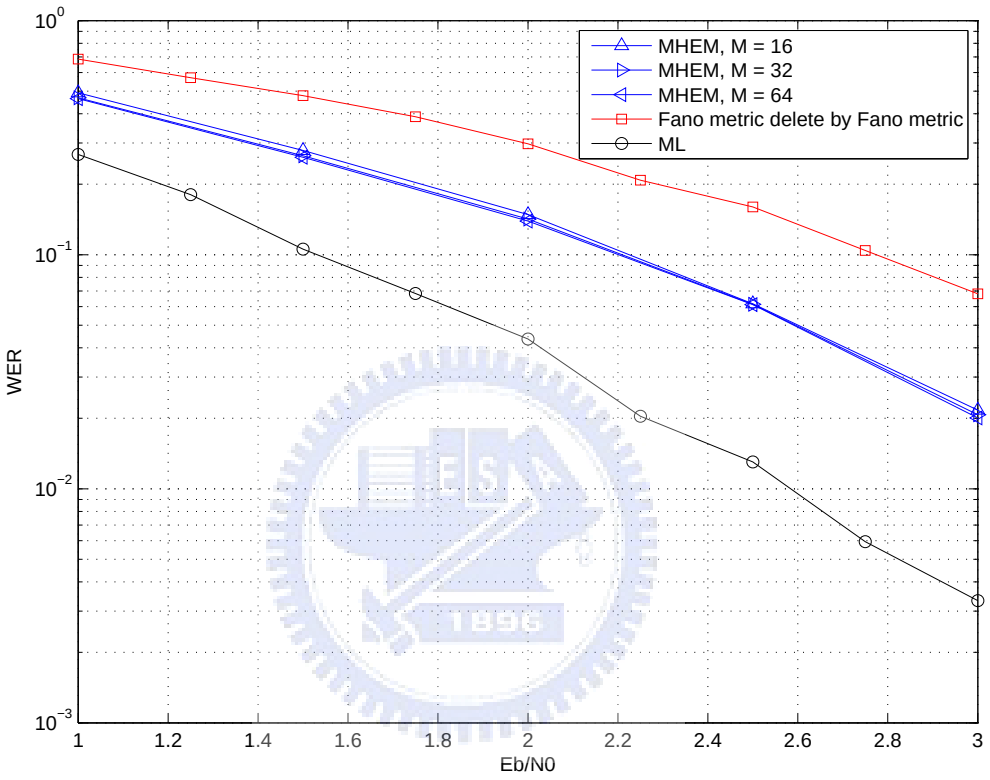


Figure 5.16: Word error rate (WER) performance of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $= 2^4 - 1$, and the information sequence length $L = 100$.

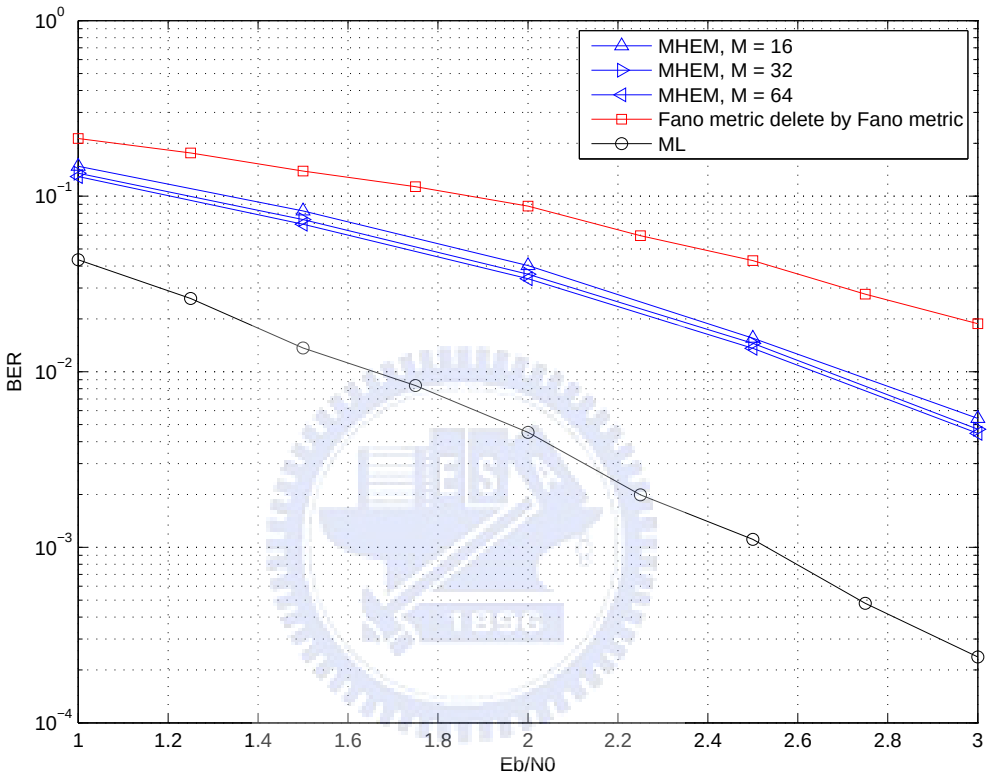


Figure 5.17: Bit error rate (BER) performance of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $= 2^4 - 1$, and the information sequence length $L = 100$.

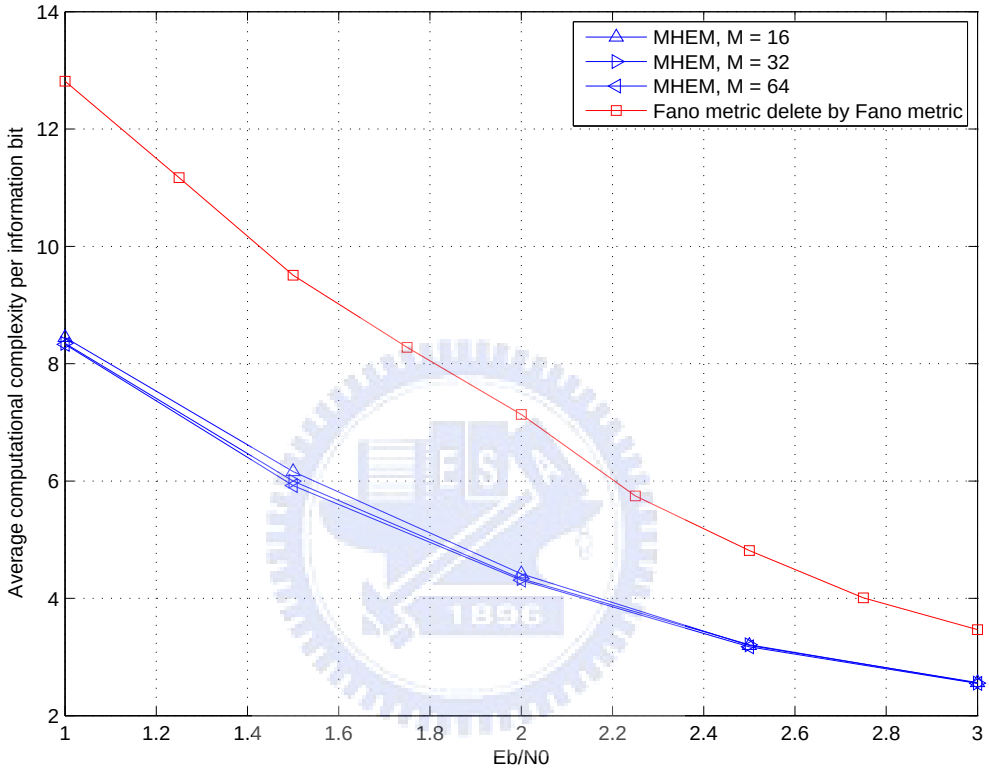


Figure 5.18: Average computational complexity per information bit of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $2^4 - 1$, and the information sequence length $L = 100$.

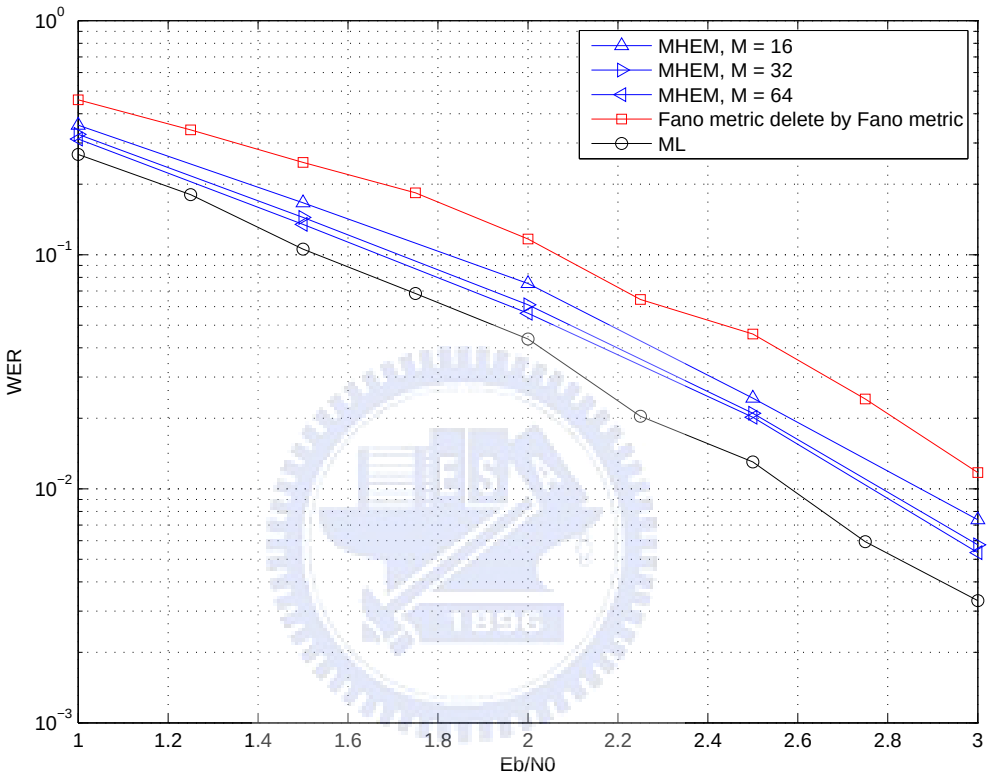


Figure 5.19: Word error rate (WER) performance of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $= 2^6 - 1$, and the information sequence length $L = 100$.

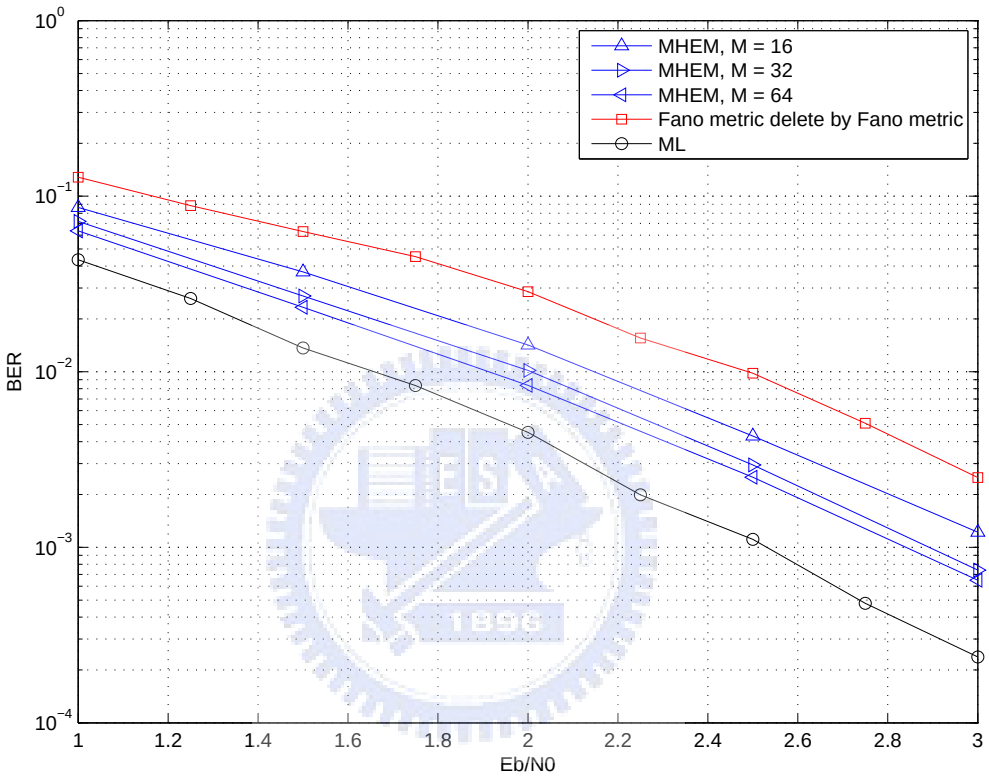


Figure 5.20: Bit error rate (BER) performance of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $= 2^6 - 1$, and the information sequence length $L = 100$.

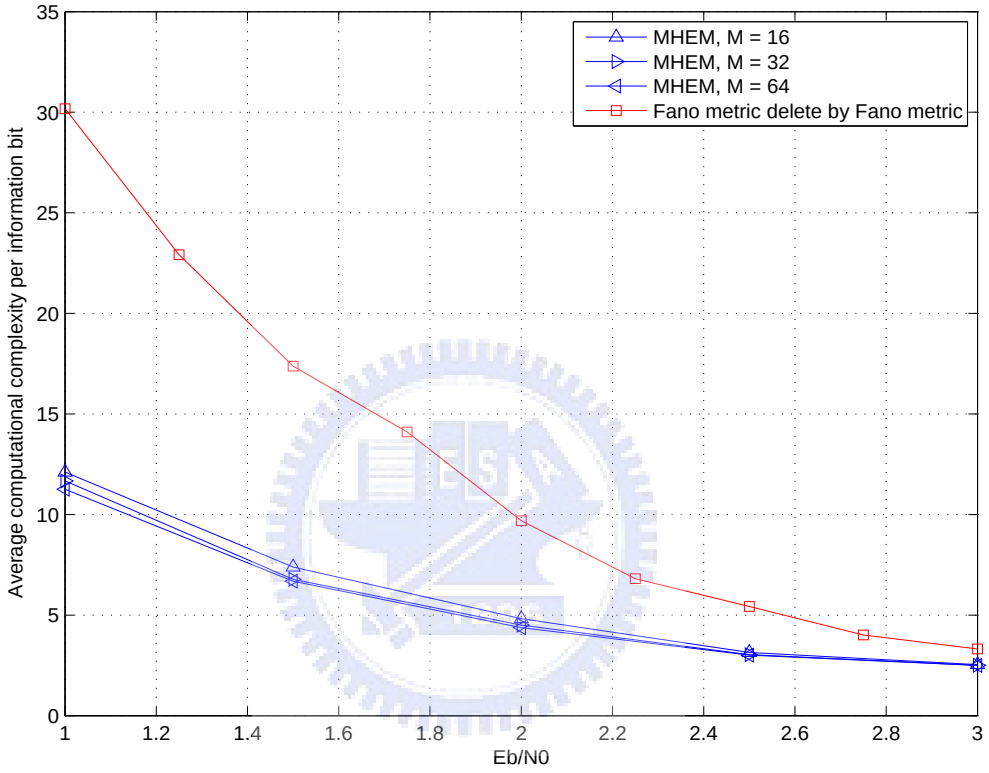


Figure 5.21: Average computational complexity per information bit of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $2^6 - 1$, and the information sequence length $L = 100$.

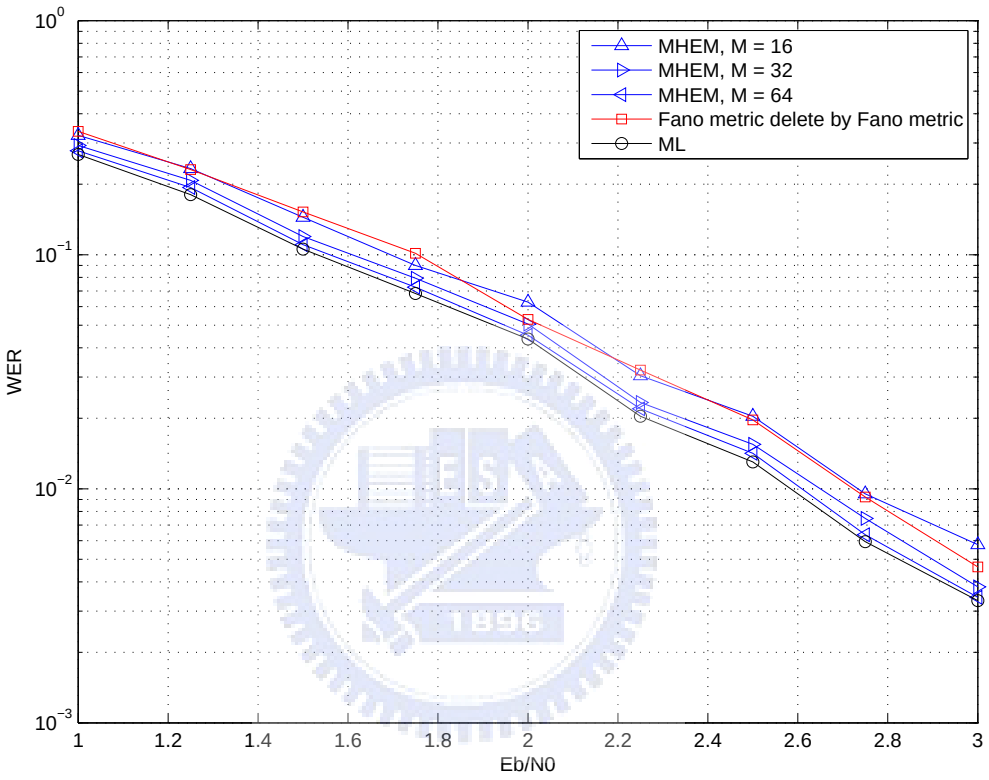


Figure 5.22: Word error rate (WER) performance of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $= 2^8 - 1$, and the information sequence length $L = 100$.

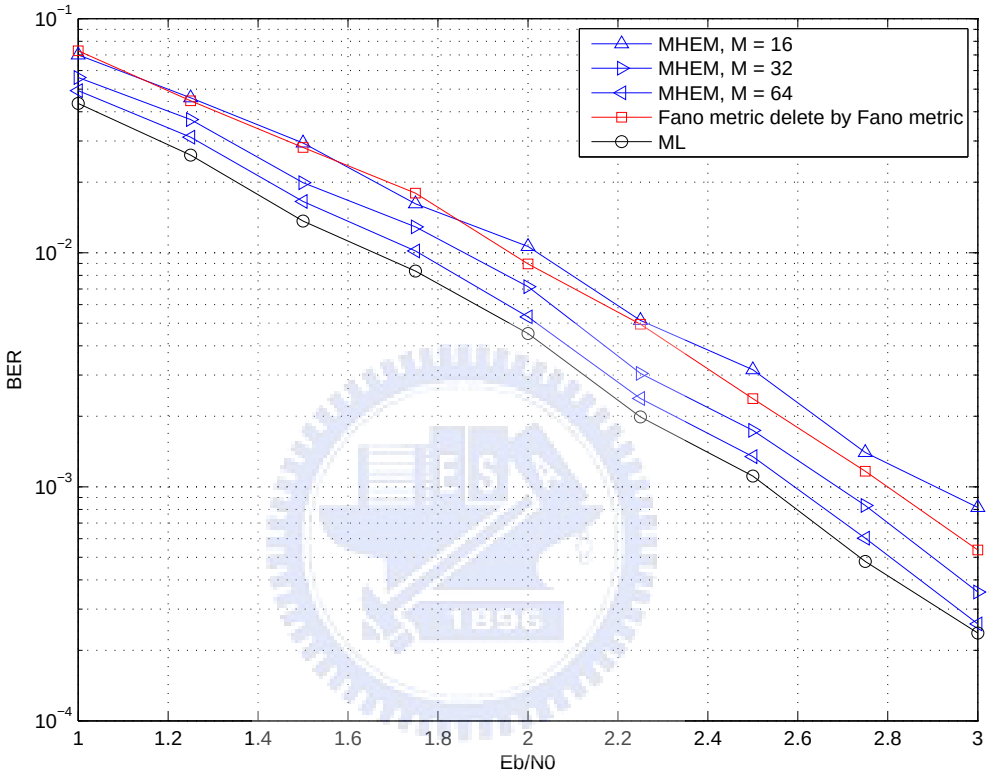


Figure 5.23: Bit error rate (BER) performance of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $= 2^8 - 1$, and the information sequence length $L = 100$.

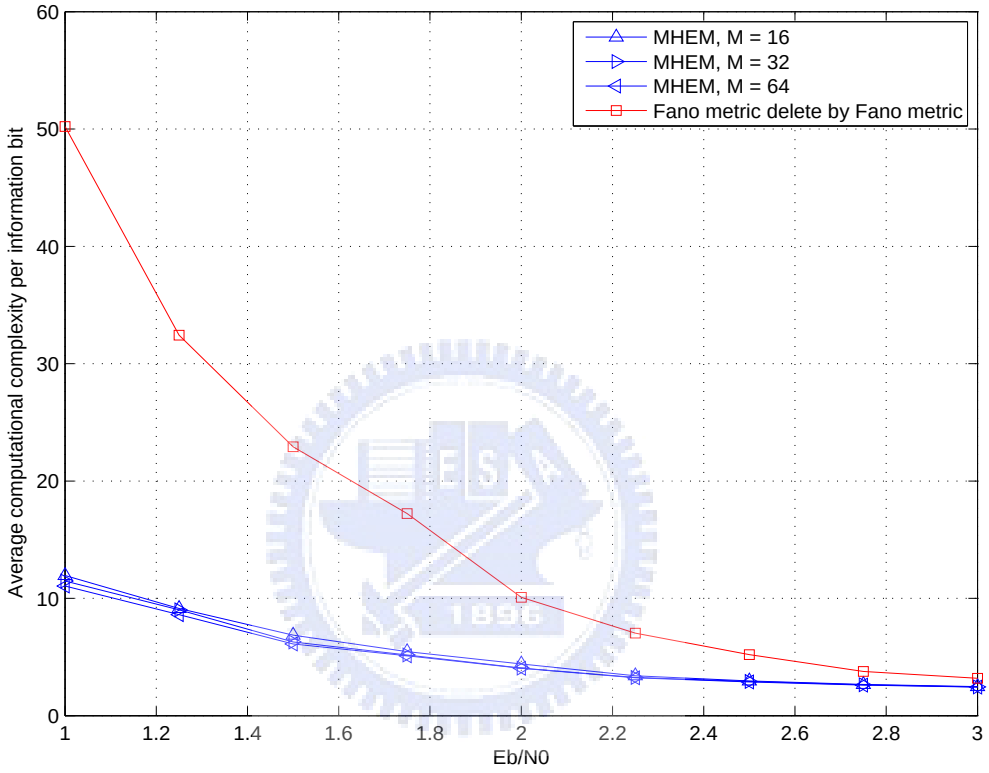


Figure 5.24: Average computational complexity per information bit of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $2^8 - 1$, and the information sequence length $L = 100$.

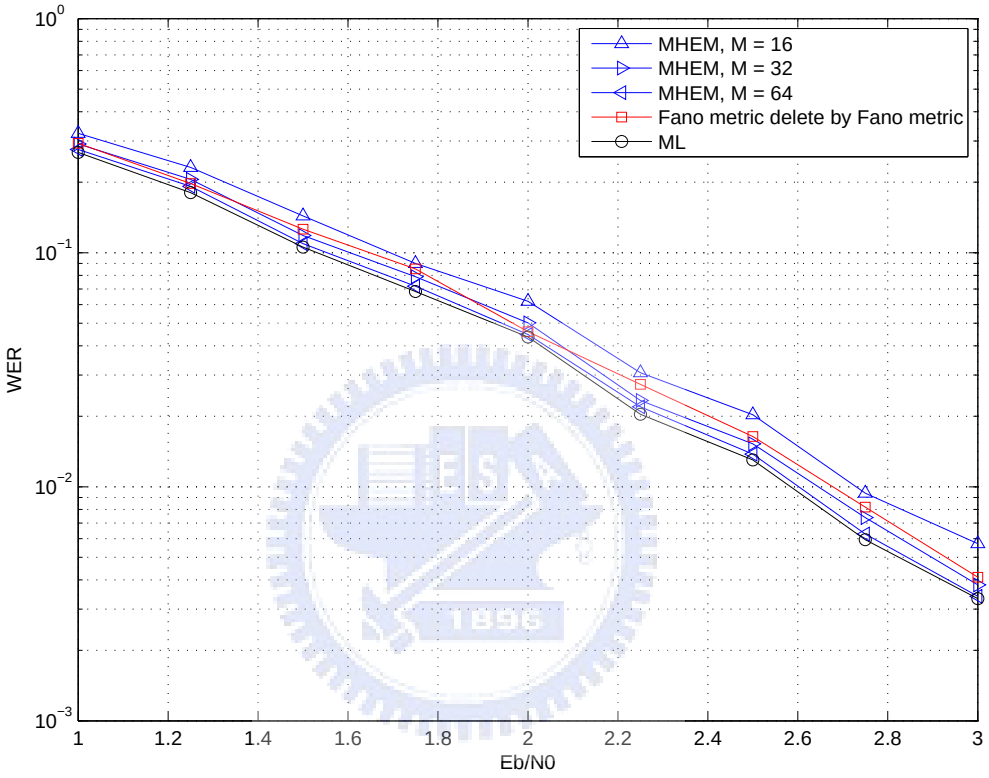


Figure 5.25: Word error rate (WER) performance of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $= 2^{10} - 1$, and the information sequence length $L = 100$.

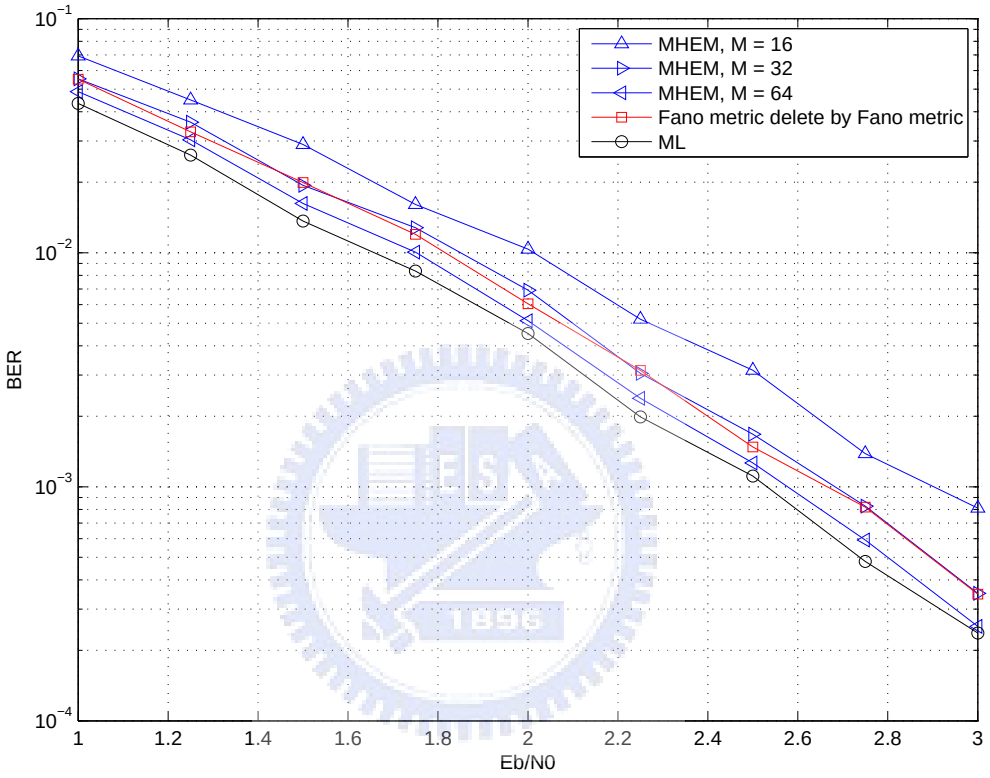


Figure 5.26: Bit error rate (BER) performance of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $= 2^{10} - 1$, and the information sequence length $L = 100$.

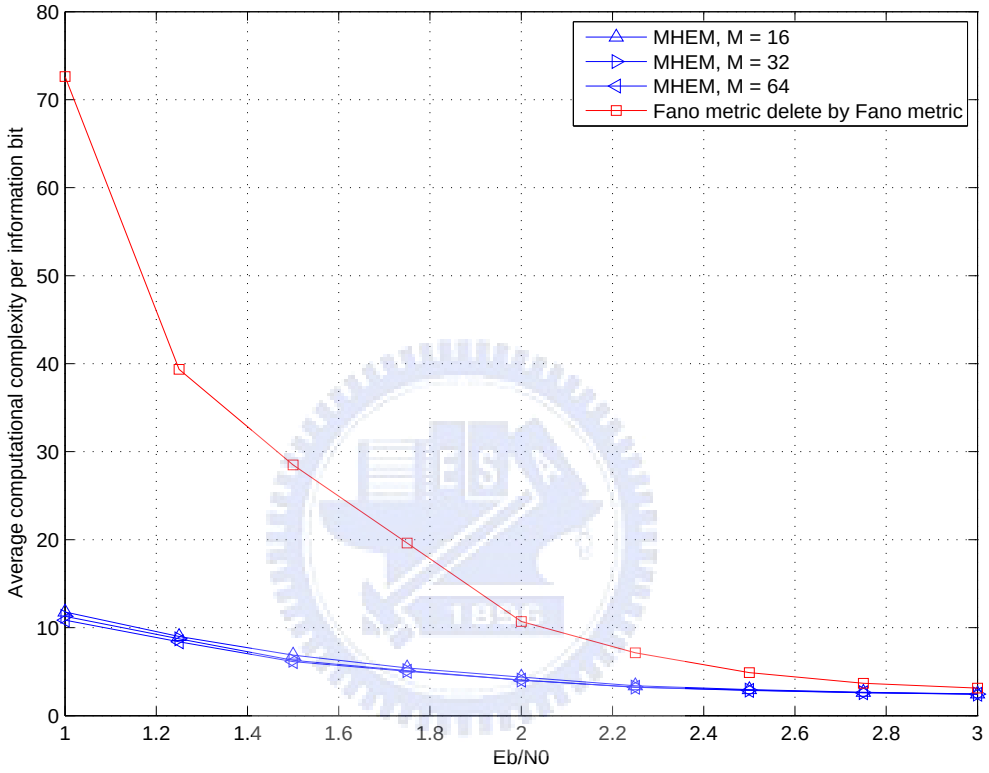


Figure 5.27: Average computational complexity per information bit of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $2^{10} - 1$, and the information sequence length $L = 100$.

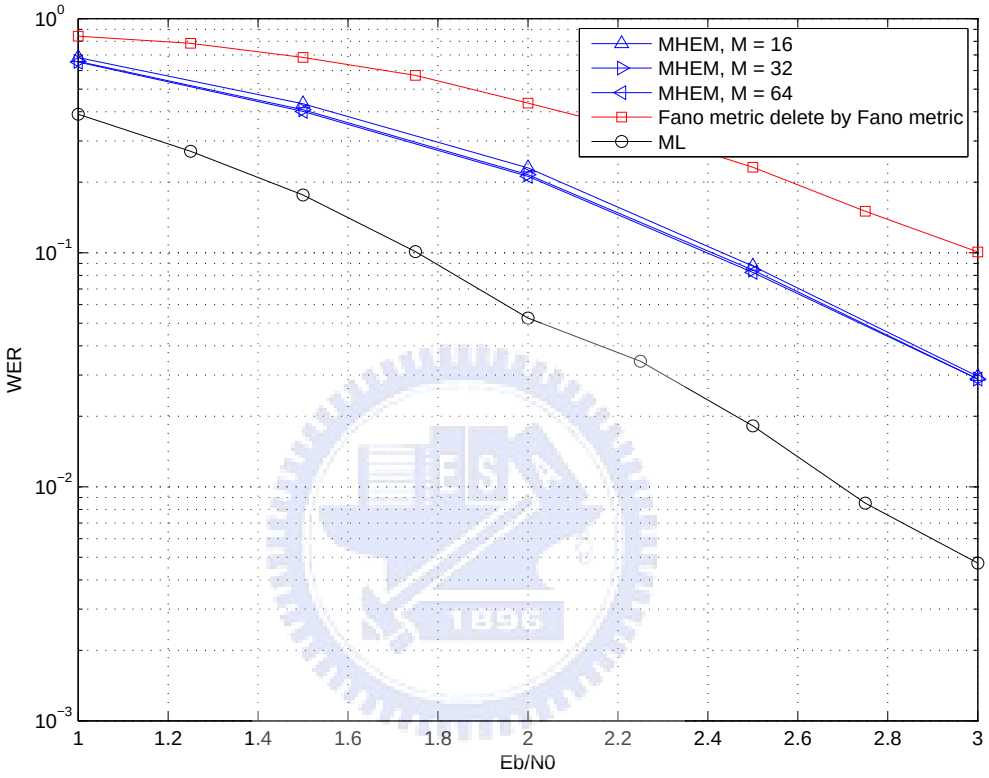


Figure 5.28: Word error rate (WER) performance of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $= 2^4 - 1$, and the information sequence length $L = 200$.

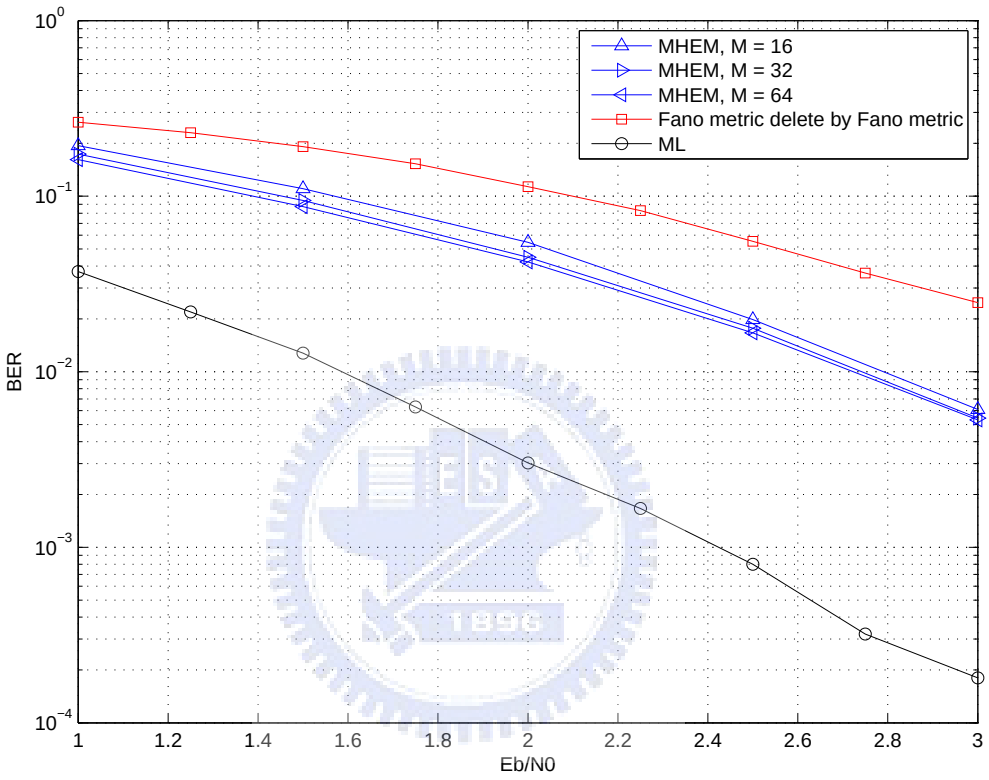


Figure 5.29: Bit error rate (BER) performance of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $= 2^4 - 1$, and the information sequence length $L = 200$.

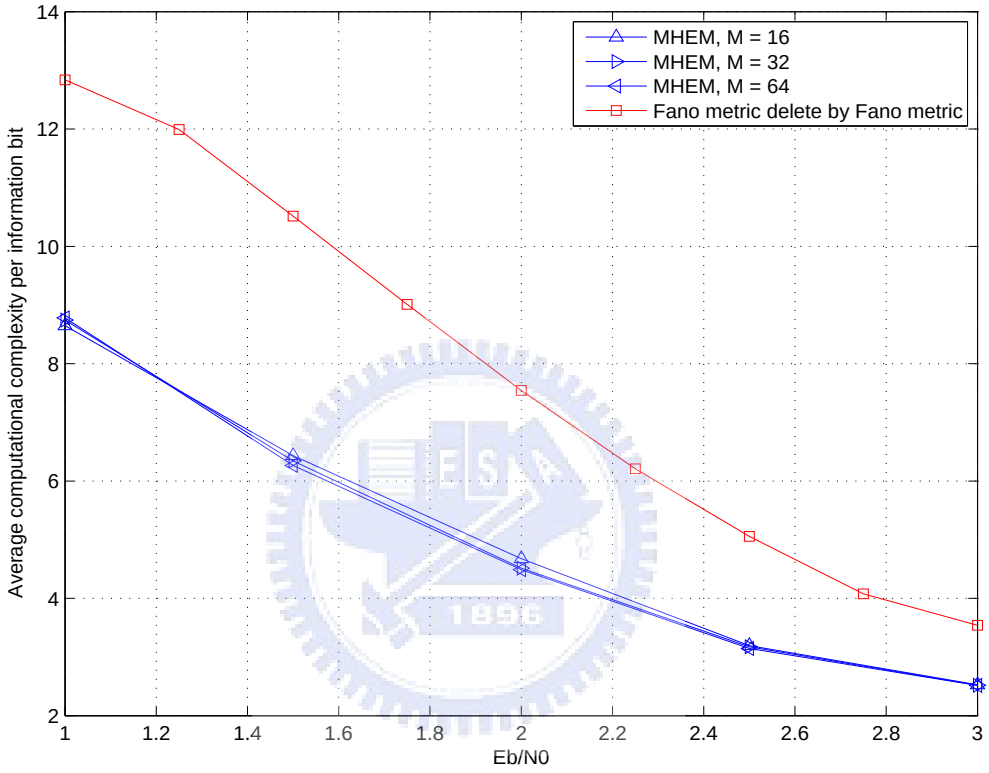


Figure 5.30: Average computational complexity per information bit of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $2^4 - 1$, and the information sequence length $L = 200$.

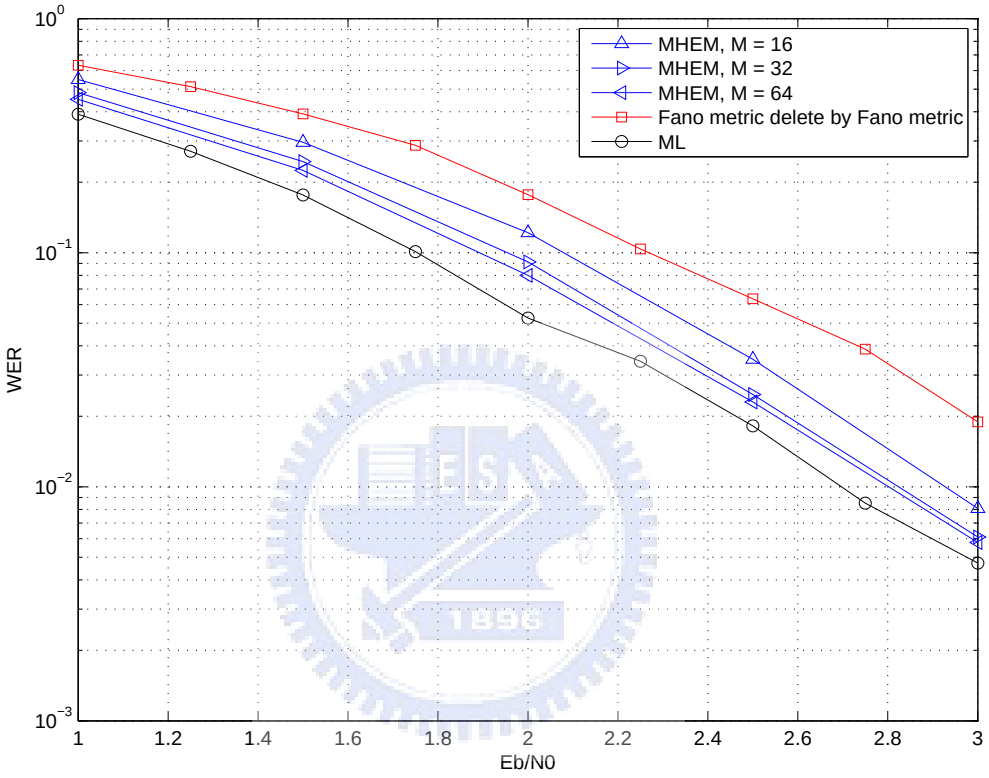


Figure 5.31: Word error rate (WER) performance of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $= 2^6 - 1$, and the information sequence length $L = 200$.

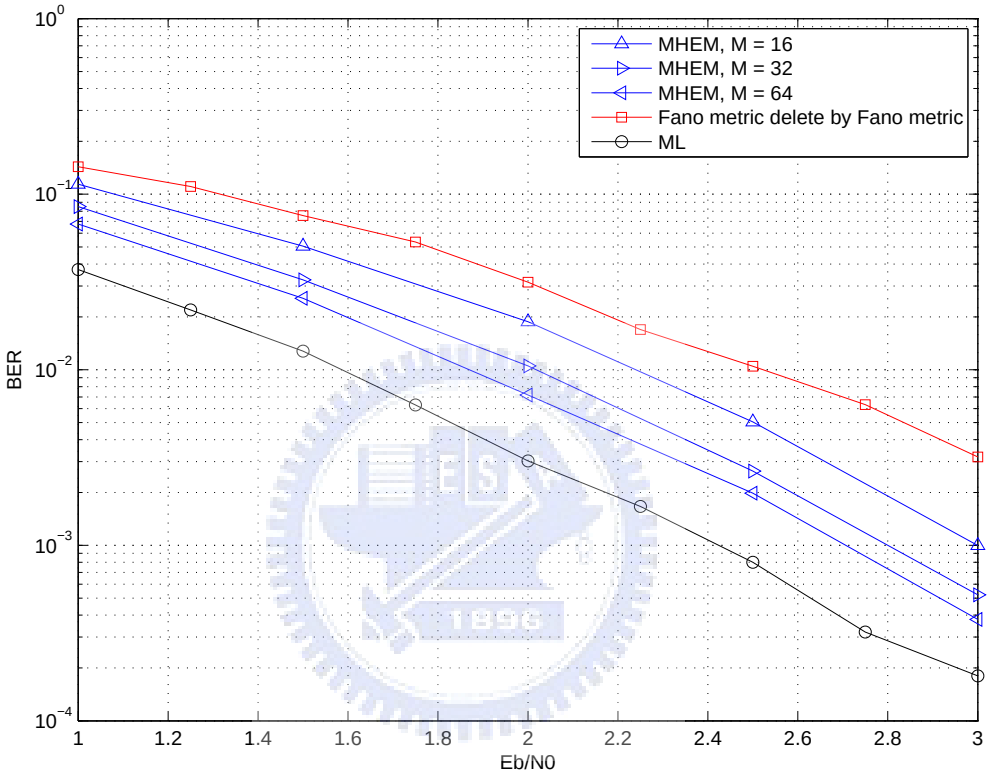


Figure 5.32: Bit error rate (BER) performance of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $= 2^6 - 1$, and the information sequence length $L = 200$.

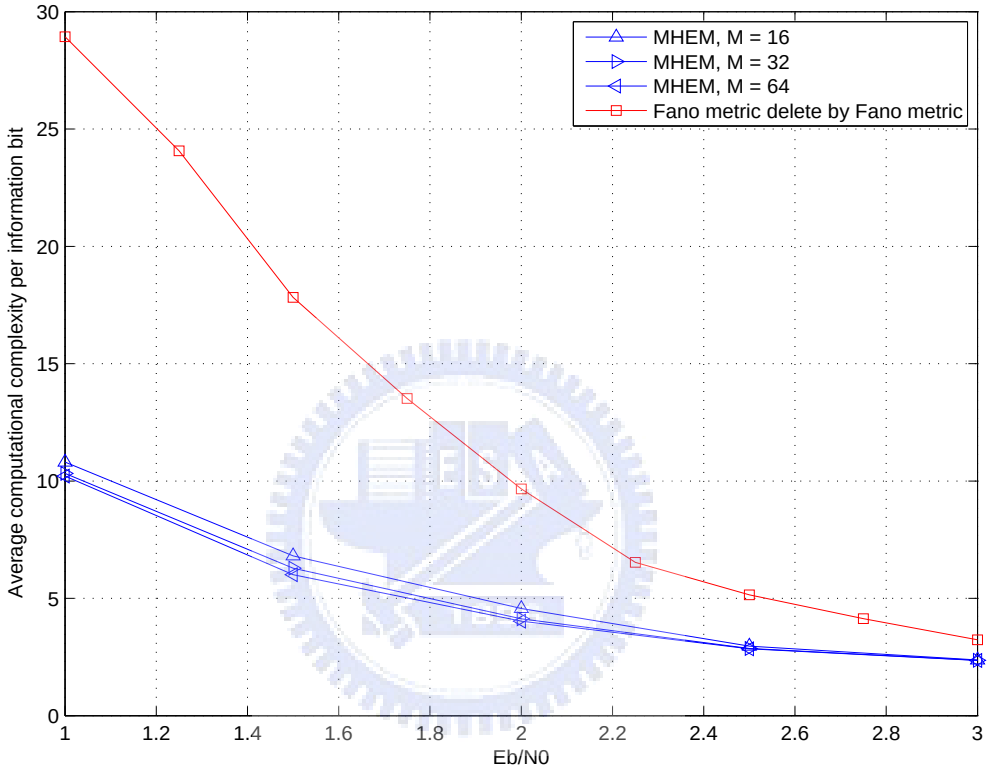


Figure 5.33: Average computational complexity per information bit of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $2^6 - 1$, and the information sequence length $L = 200$.

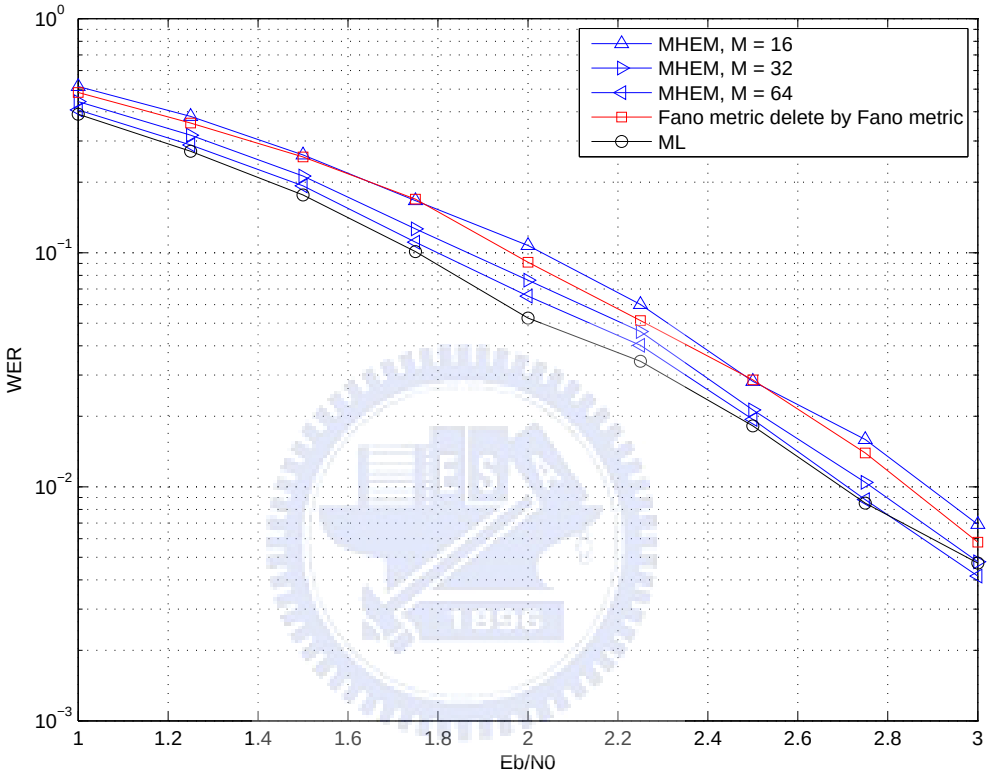


Figure 5.34: Word error rate (WER) performance of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $= 2^8 - 1$, and the information sequence length $L = 200$.

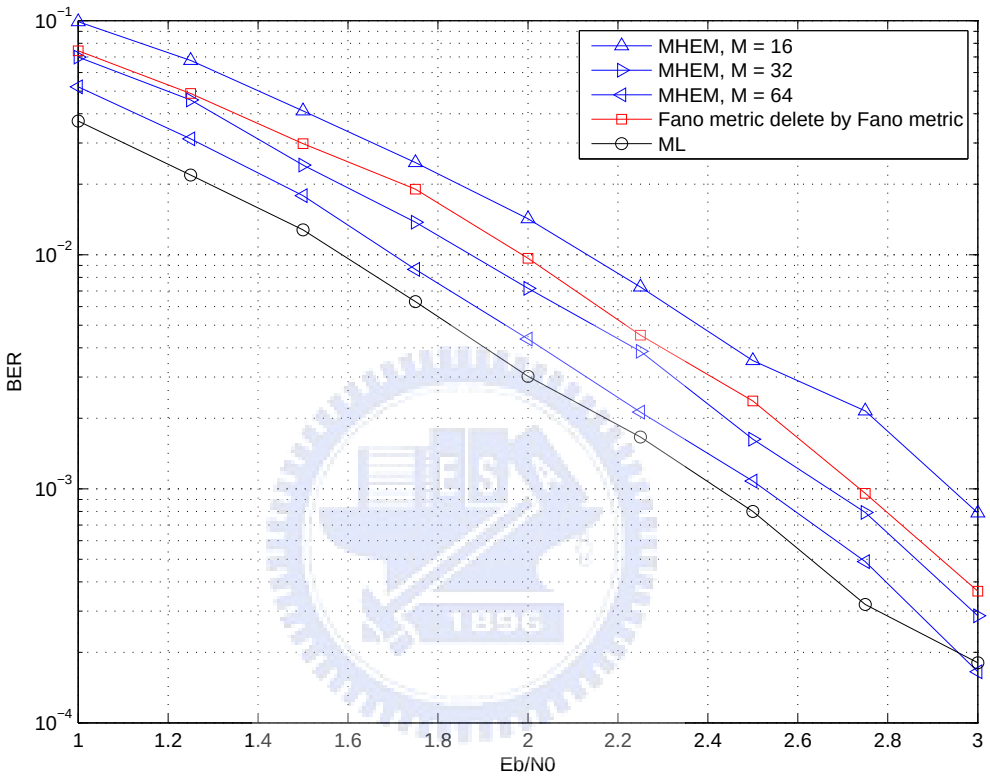


Figure 5.35: Bit error rate (BER) performance of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $= 2^8 - 1$, and the information sequence length $L = 200$.

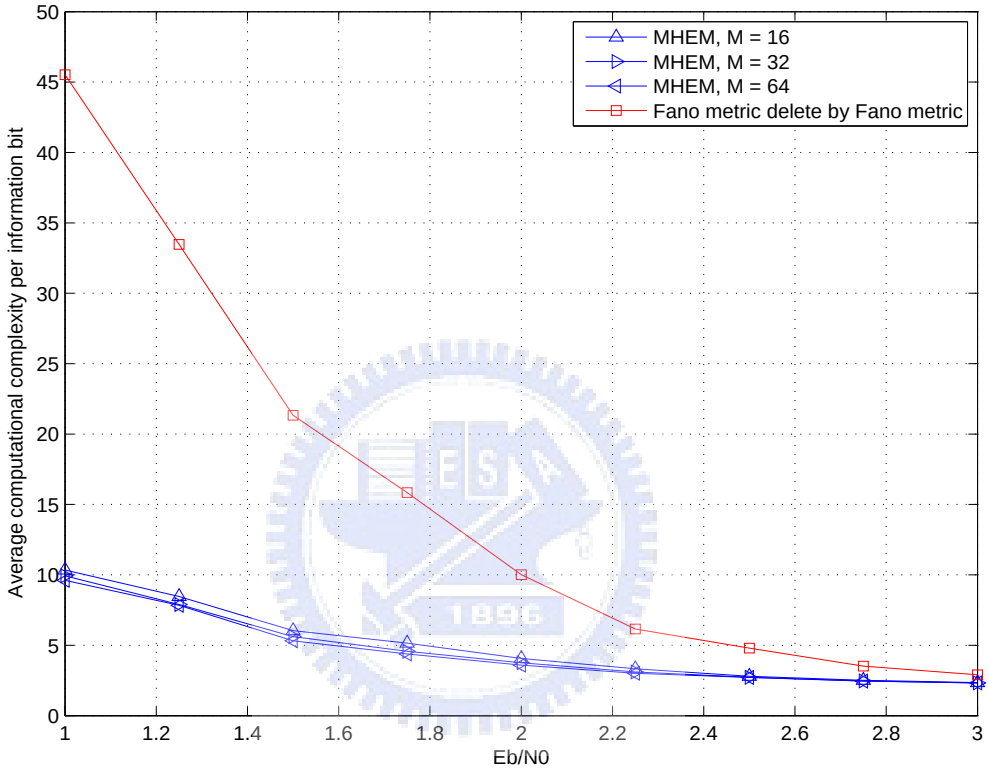


Figure 5.36: Average computational complexity per information bit of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $2^8 - 1$, and the information sequence length $L = 200$.

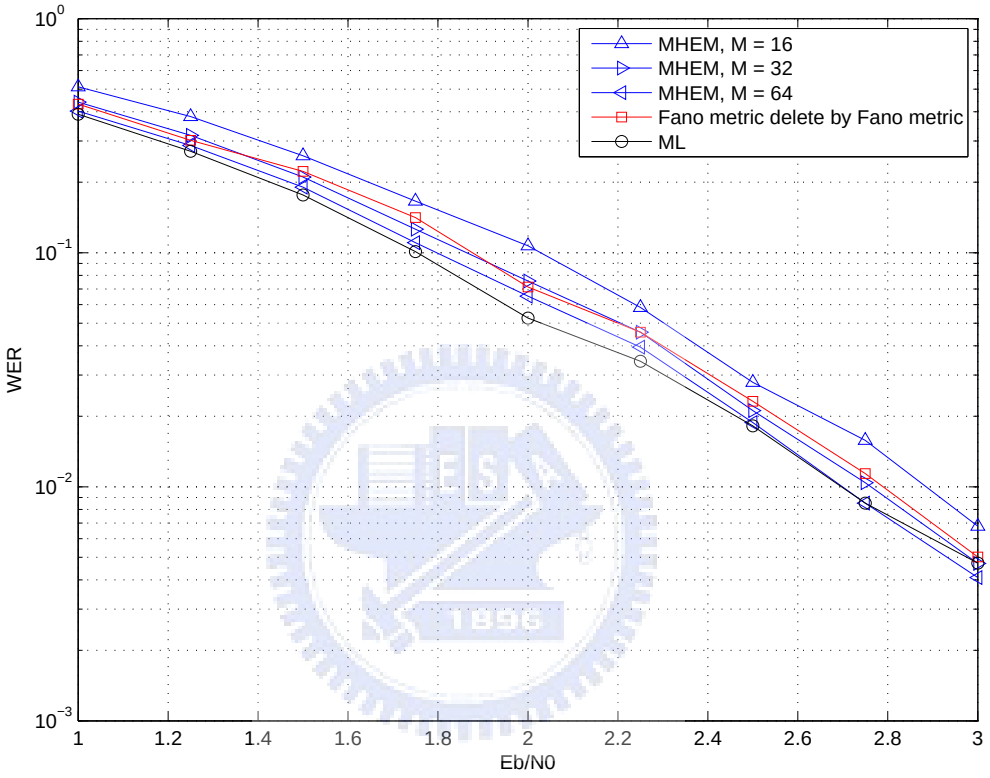


Figure 5.37: Word error rate (WER) performance of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $= 2^{10} - 1$, and the information sequence length $L = 200$.

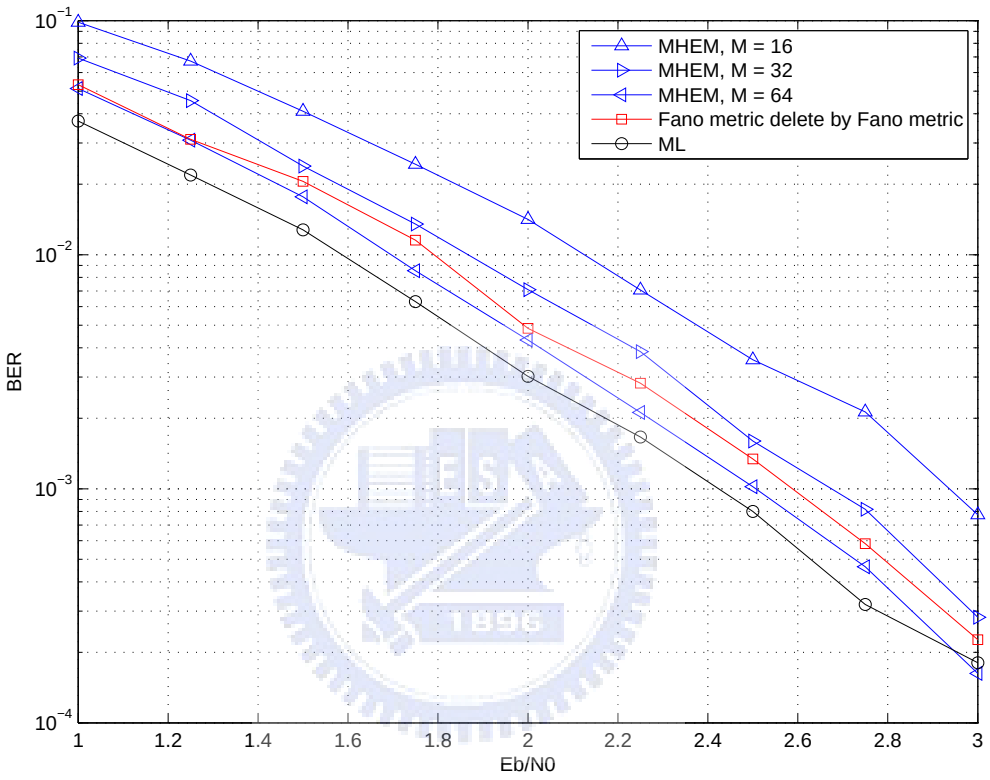


Figure 5.38: Bit error rate (BER) performance of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $= 2^{10} - 1$, and the information sequence length $L = 200$.

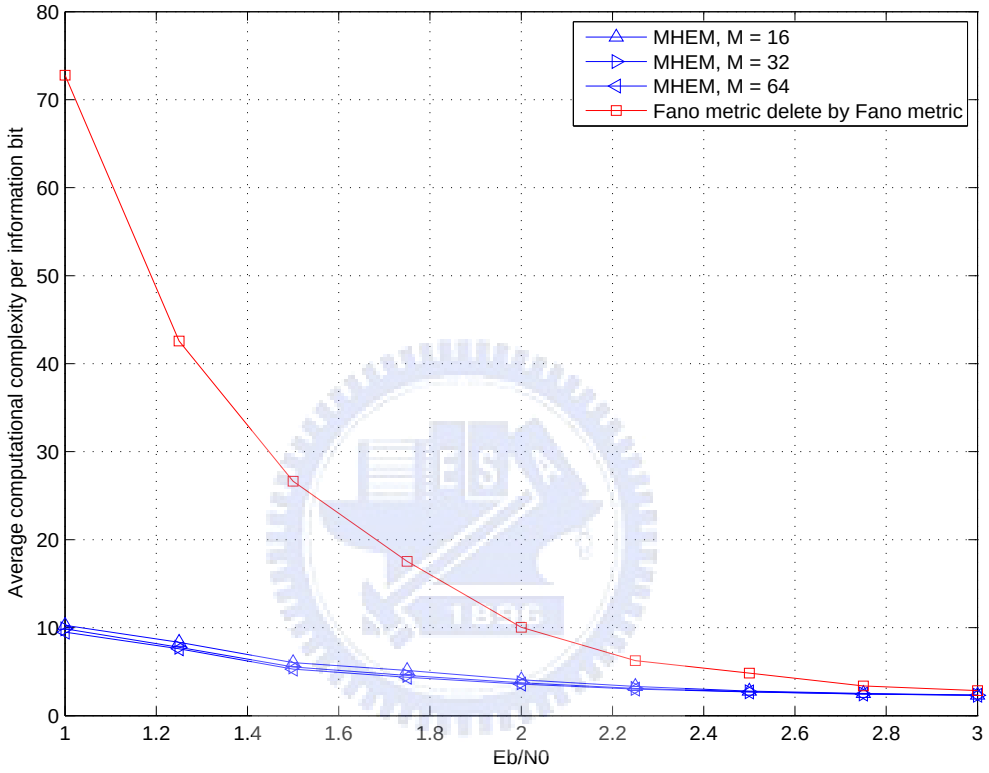


Figure 5.39: Average computational complexity per information bit of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $2^{10} - 1$, and the information sequence length $L = 200$.

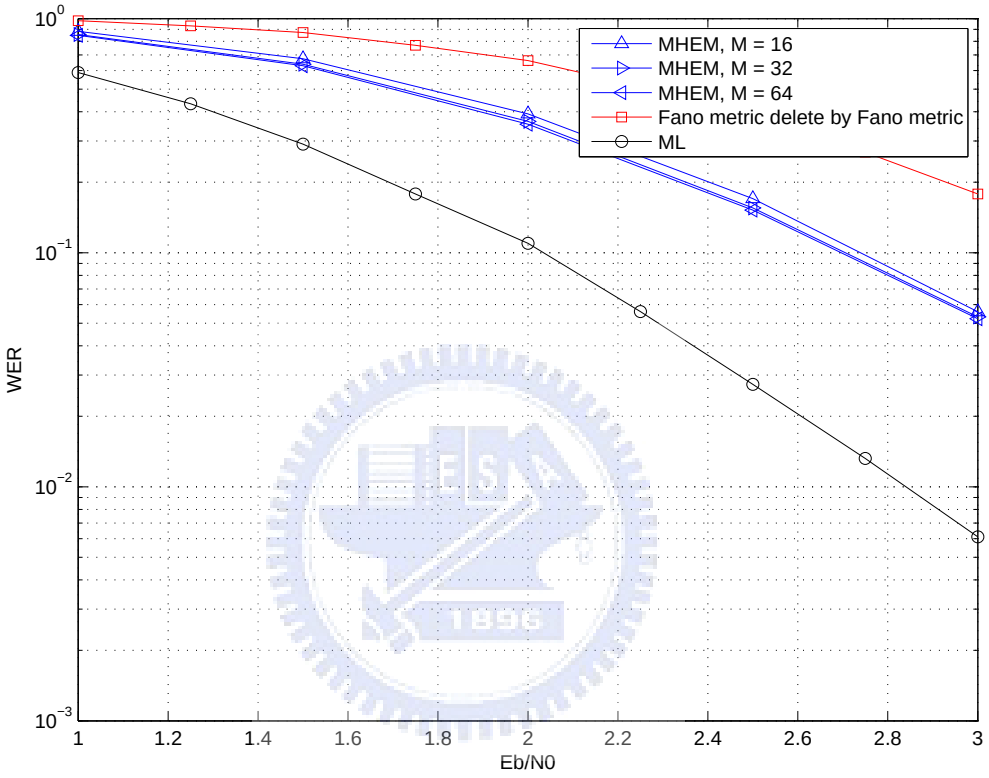


Figure 5.40: Word error rate (WER) performance of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $= 2^4 - 1$, and the information sequence length $L = 400$.

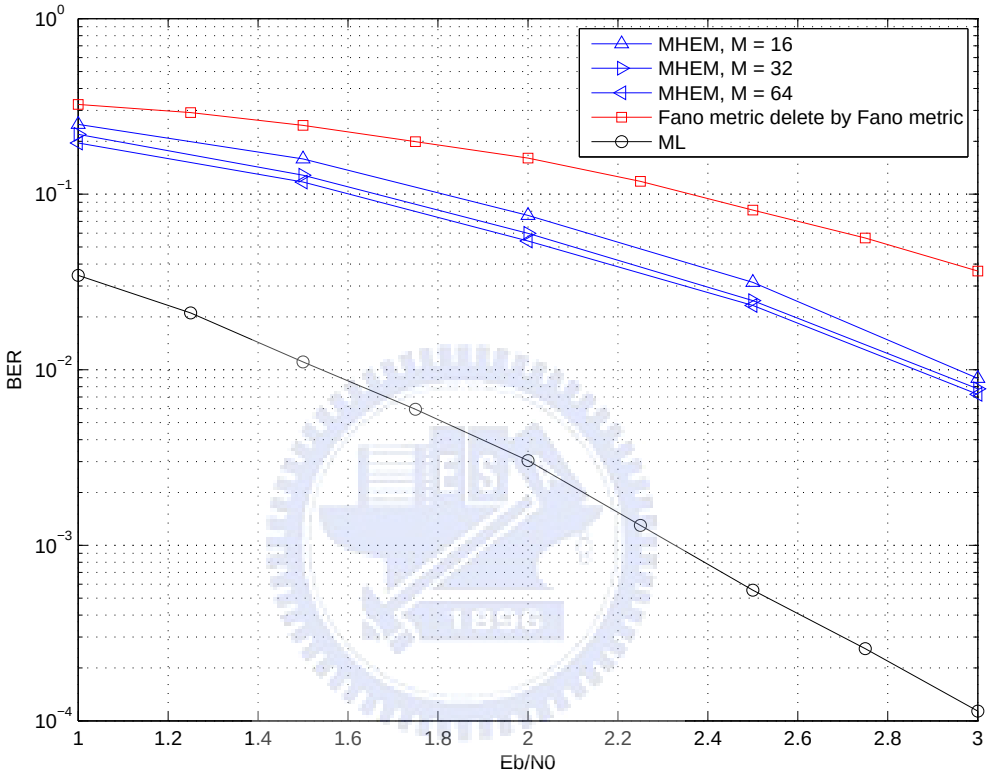


Figure 5.41: Bit error rate (BER) performance of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $= 2^4 - 1$, and the information sequence length $L = 400$.

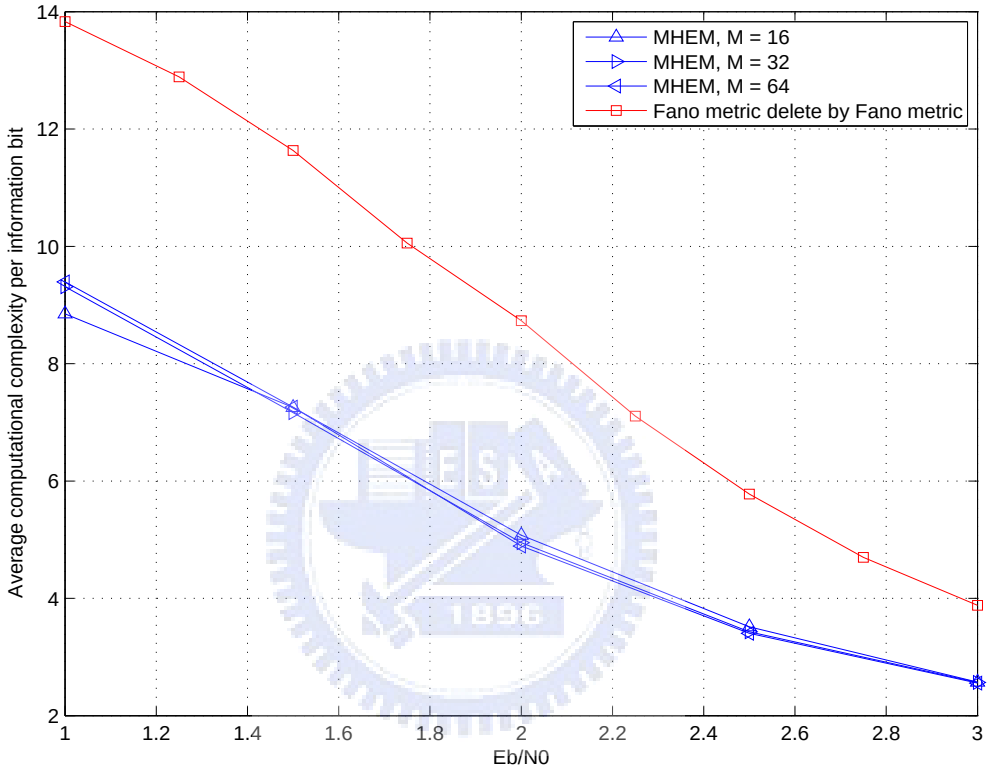


Figure 5.42: Average computational complexity per information bit of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $2^4 - 1$, and the information sequence length $L = 400$.

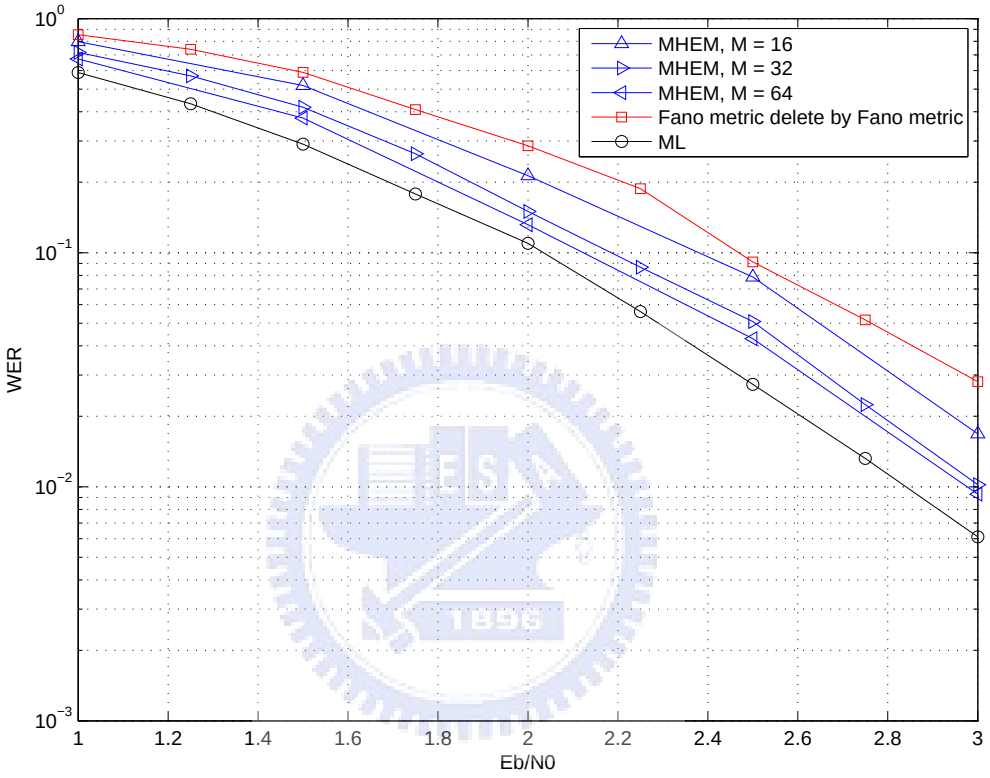


Figure 5.43: Word error rate (WER) performance of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $= 2^6 - 1$, and the information sequence length $L = 400$.

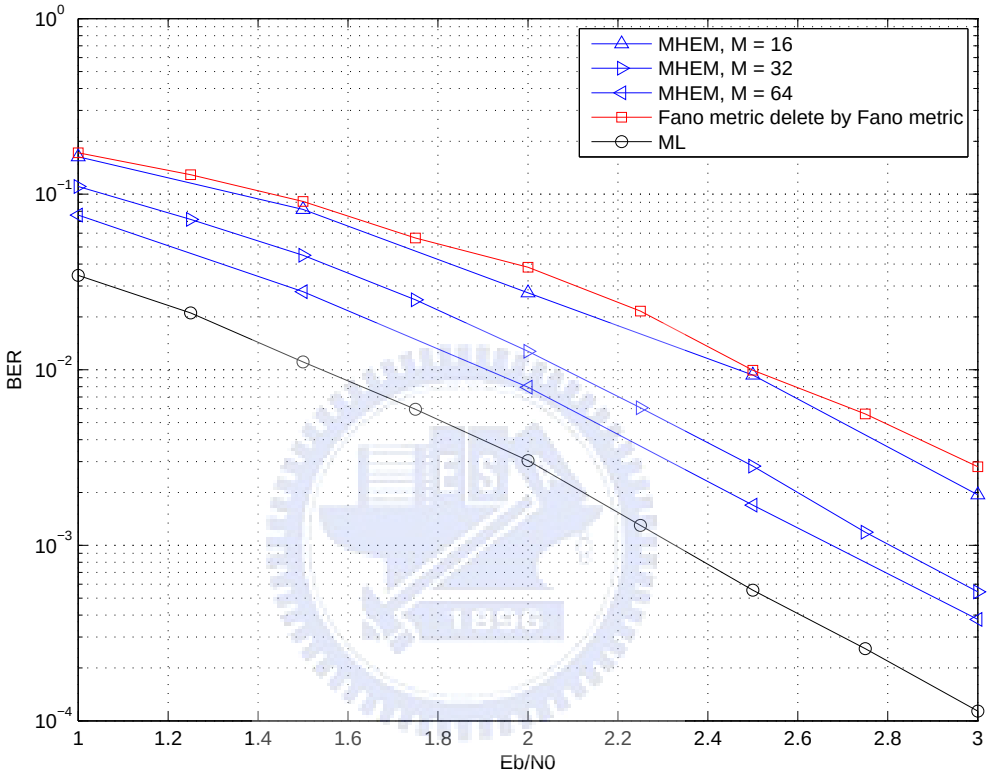


Figure 5.44: Bit error rate (BER) performance of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $= 2^6 - 1$, and the information sequence length $L = 400$.

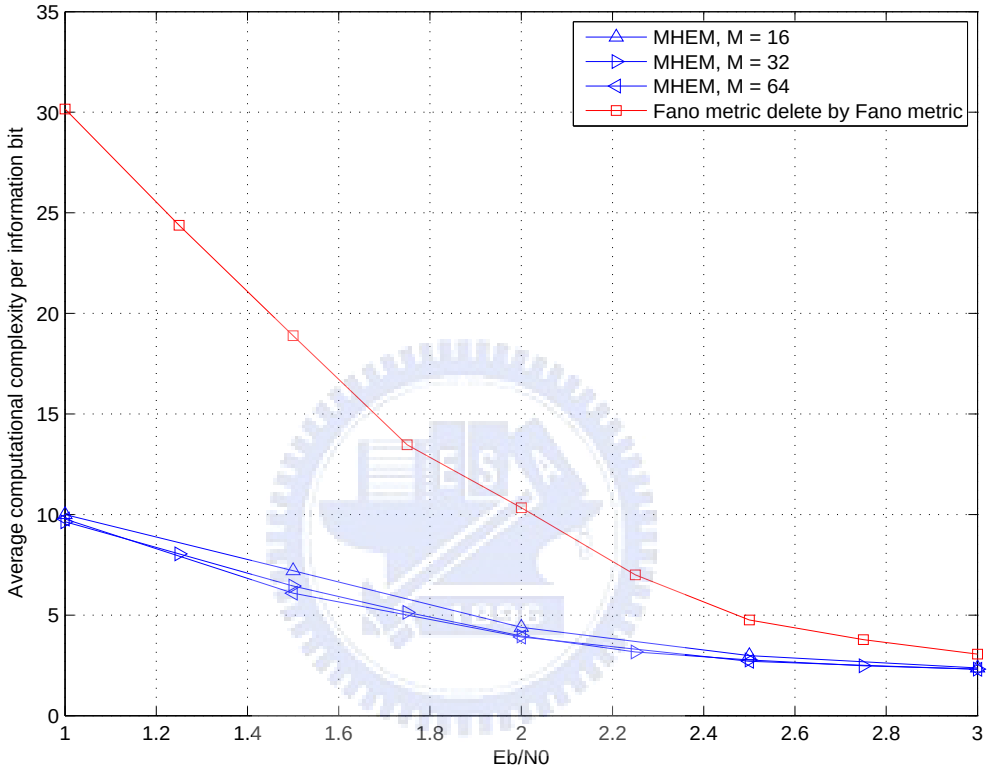


Figure 5.45: Average computational complexity per information bit of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $2^6 - 1$, and the information sequence length $L = 400$.

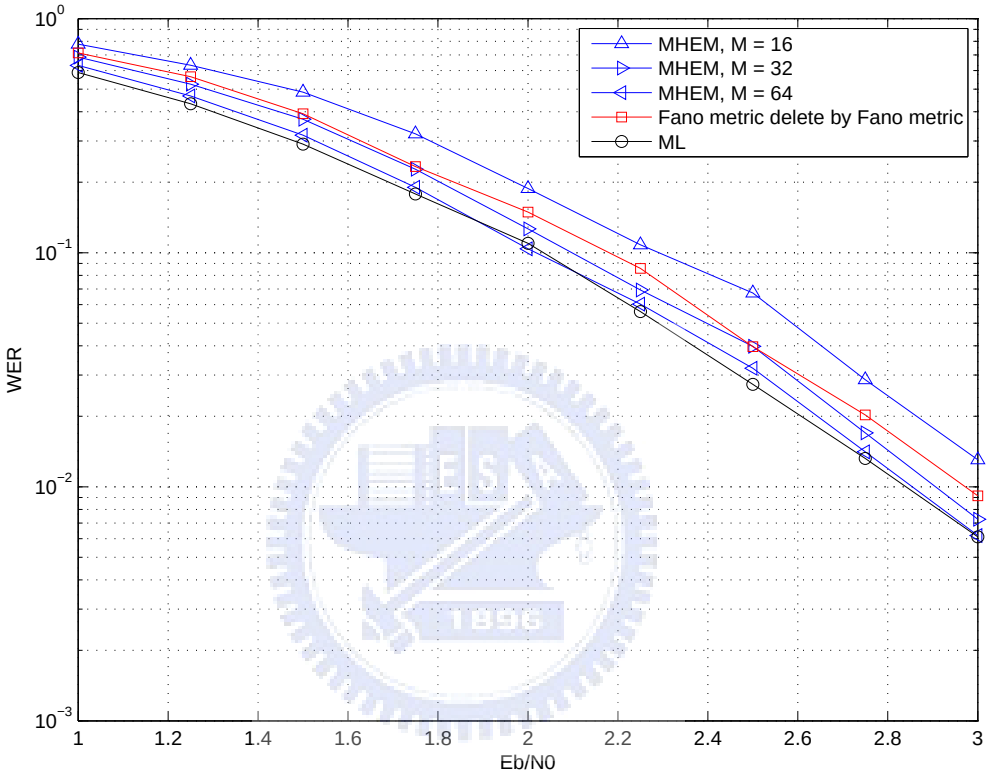


Figure 5.46: Word error rate (WER) performance of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $= 2^8 - 1$, and the information sequence length $L = 400$.

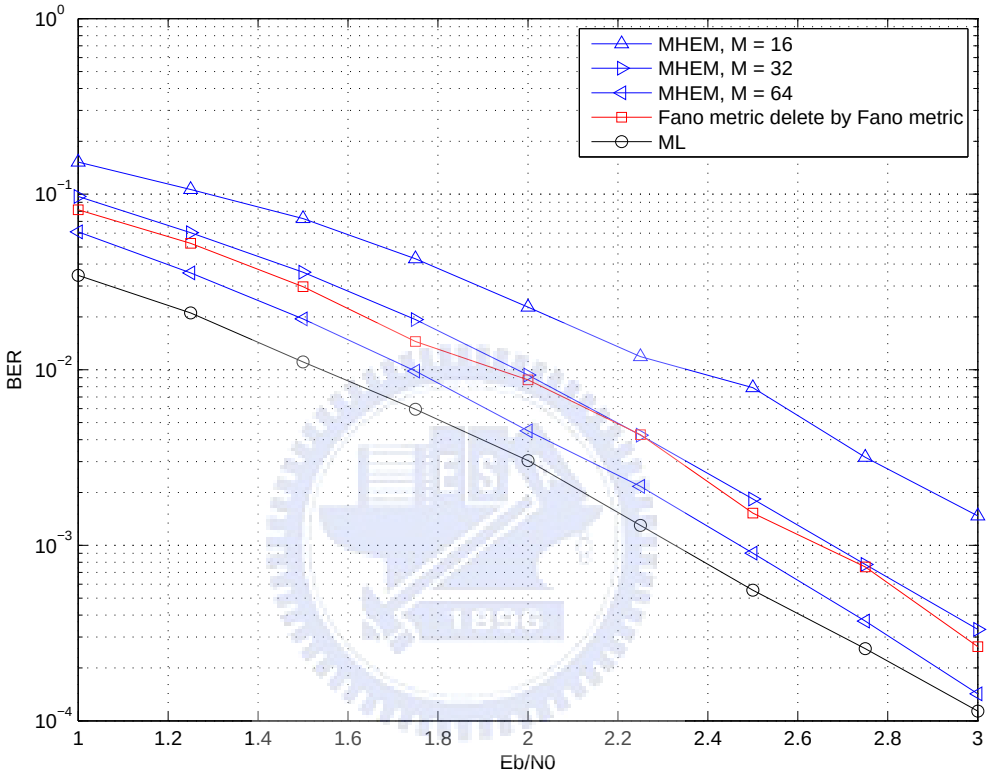


Figure 5.47: Bit error rate (BER) performance of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $= 2^8 - 1$, and the information sequence length $L = 400$.

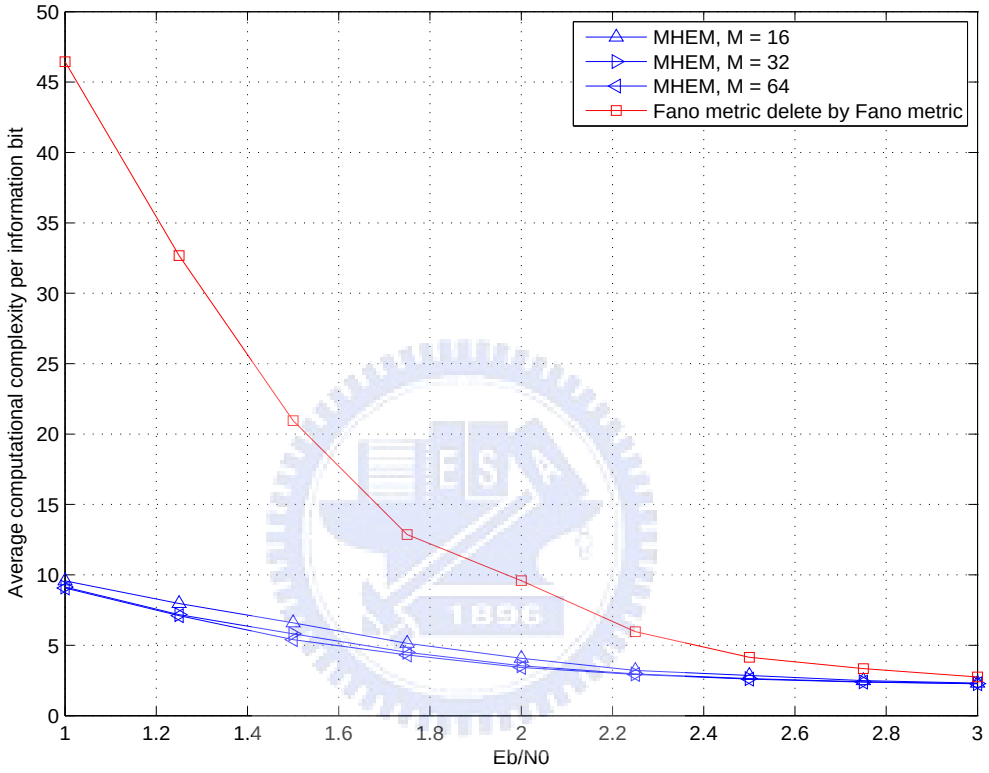


Figure 5.48: Average computational complexity per information bit of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $2^8 - 1$, and the information sequence length $L = 400$.

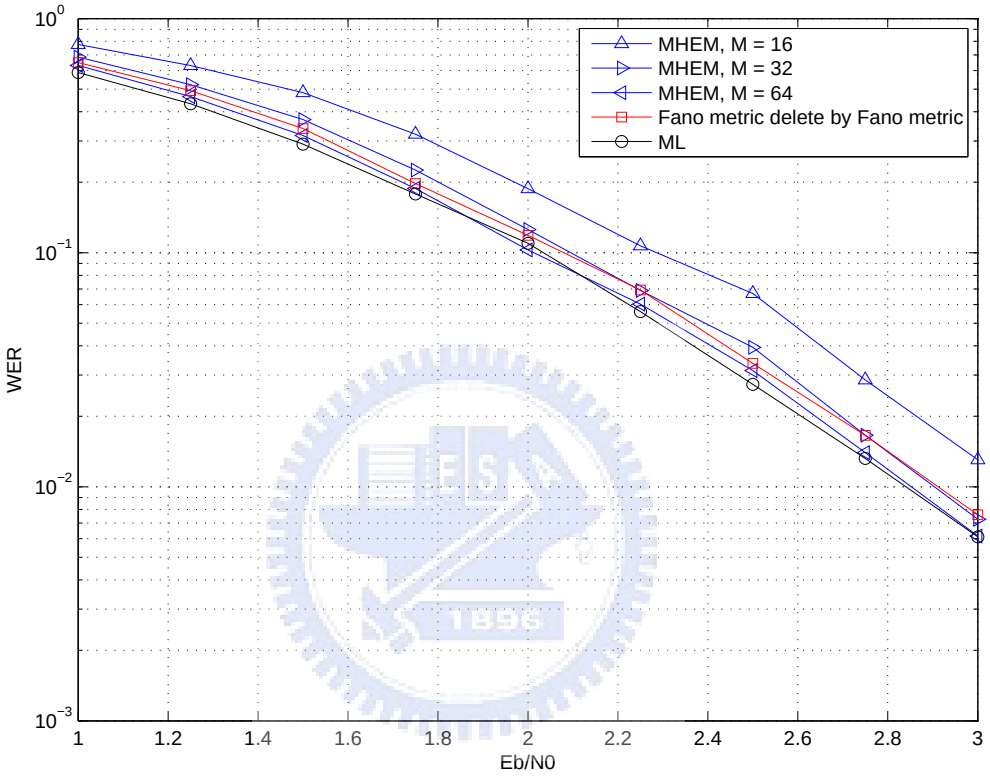


Figure 5.49: Word error rate (WER) performance of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $= 2^{10} - 1$, and the information sequence length $L = 400$.

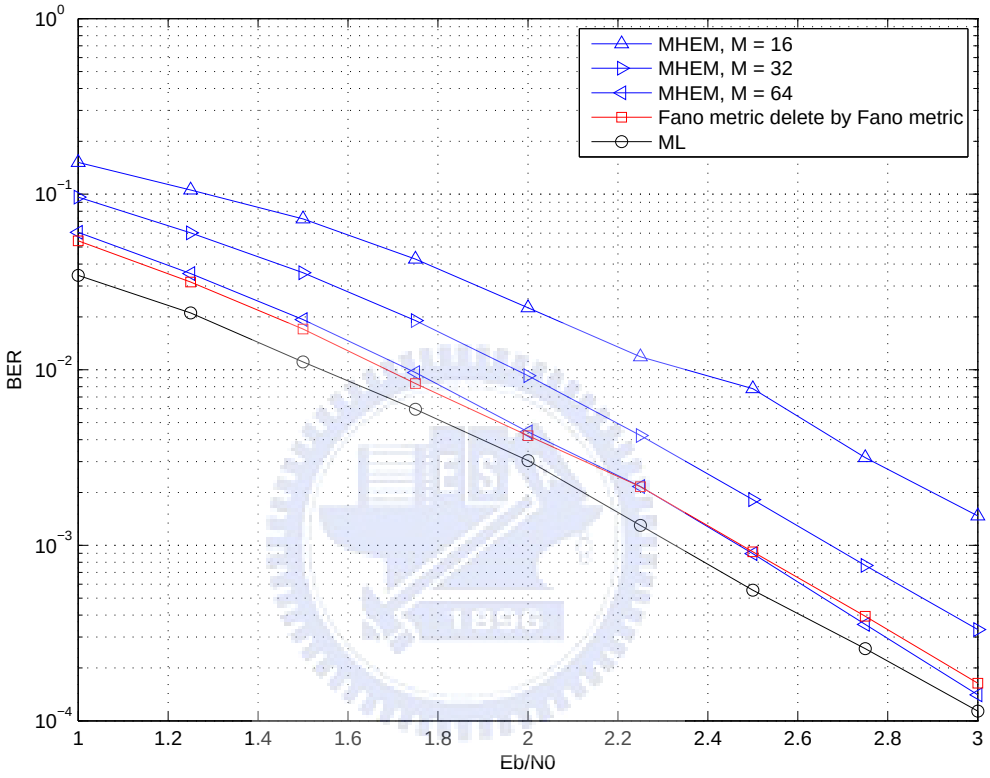


Figure 5.50: Bit error rate (BER) performance of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $= 2^{10} - 1$, and the information sequence length $L = 400$.

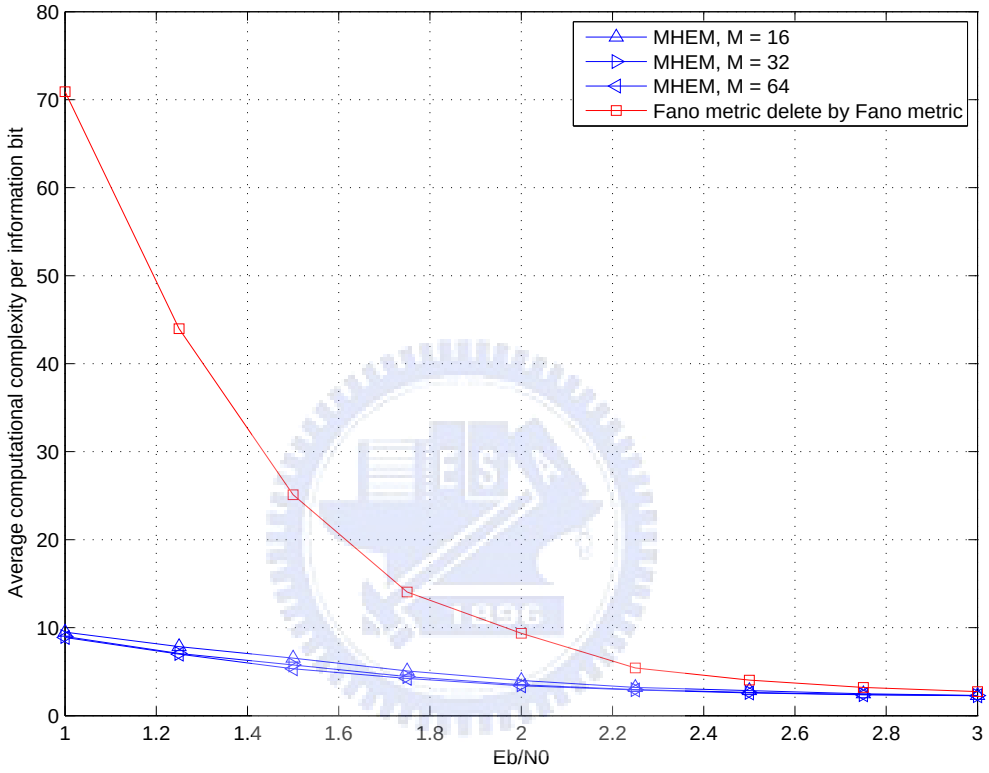


Figure 5.51: Average computational complexity per information bit of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $2^{10} - 1$, and the information sequence length $L = 400$.

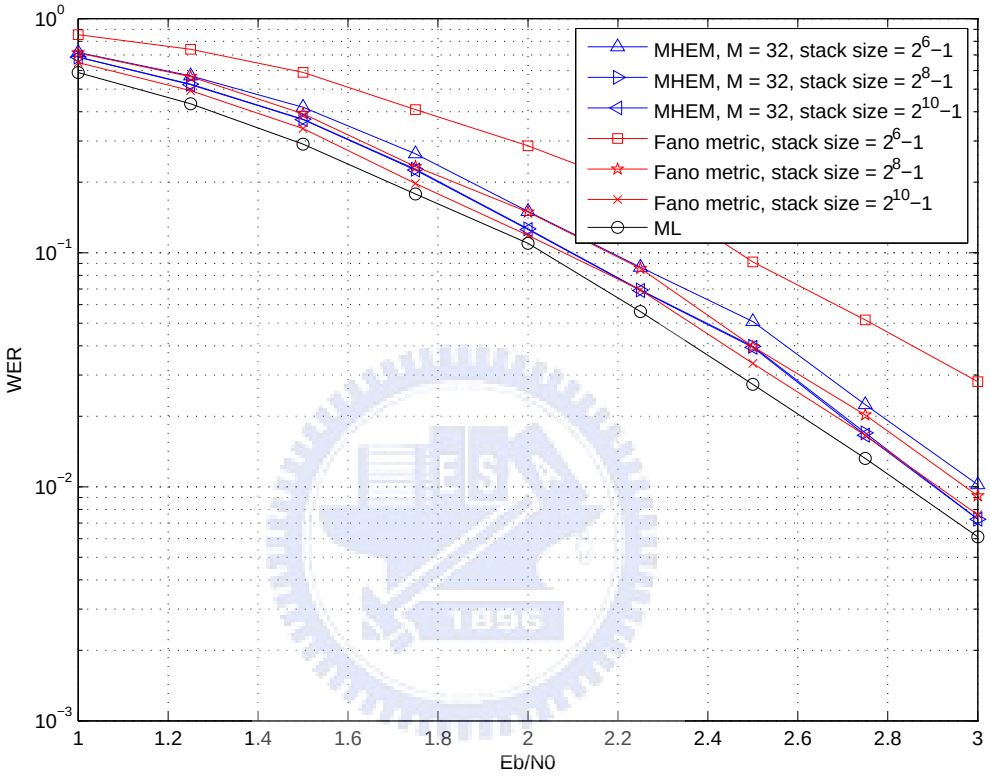


Figure 5.52: Word error rate (WER) performance of MHEM-enhanced two pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. Here, $M = 32$ and the information sequence length $L = 400$.

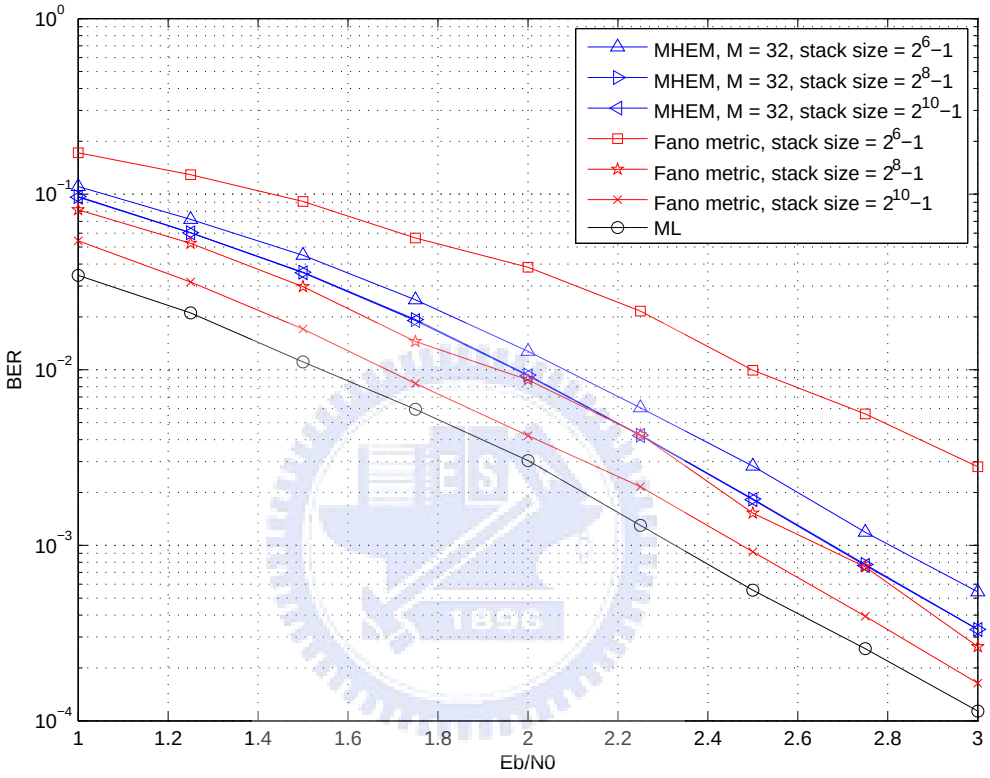


Figure 5.53: Bit error rate (BER) performance of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. Here, $M = 32$ and the information sequence length $L = 400$.

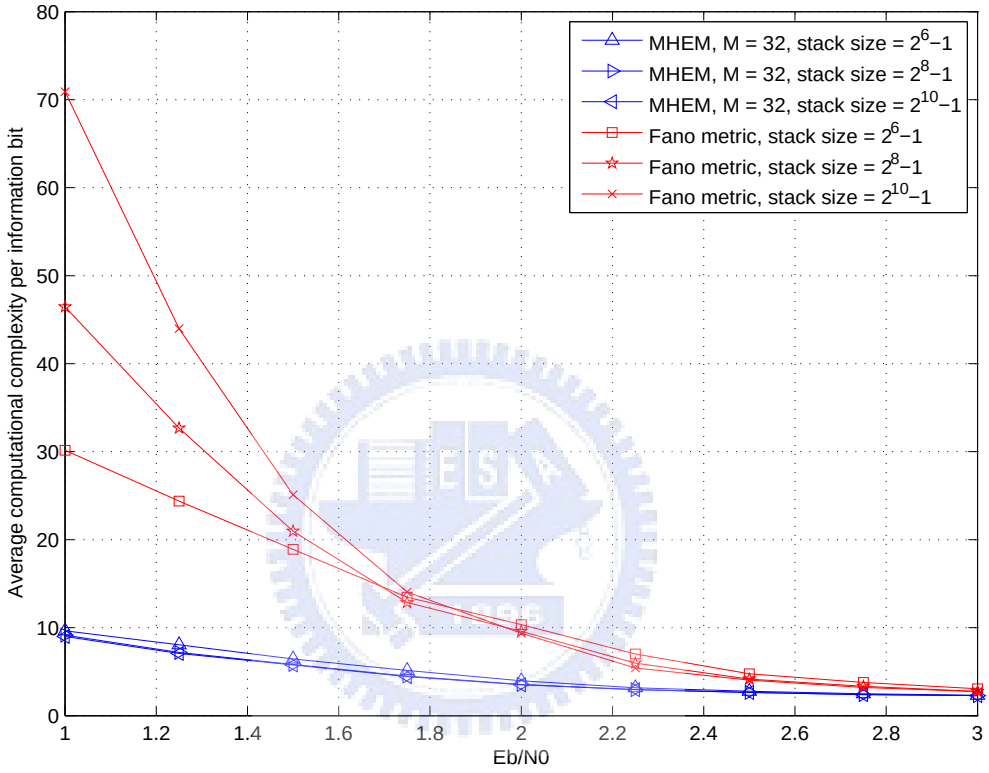


Figure 5.54: Average computational complexity per information bit of MHEM-enhanced two-pass decoder for (2,1,8) convolutional code with generator polynomial [457,755]. Here, $M = 32$, and the information sequence length $L = 400$.

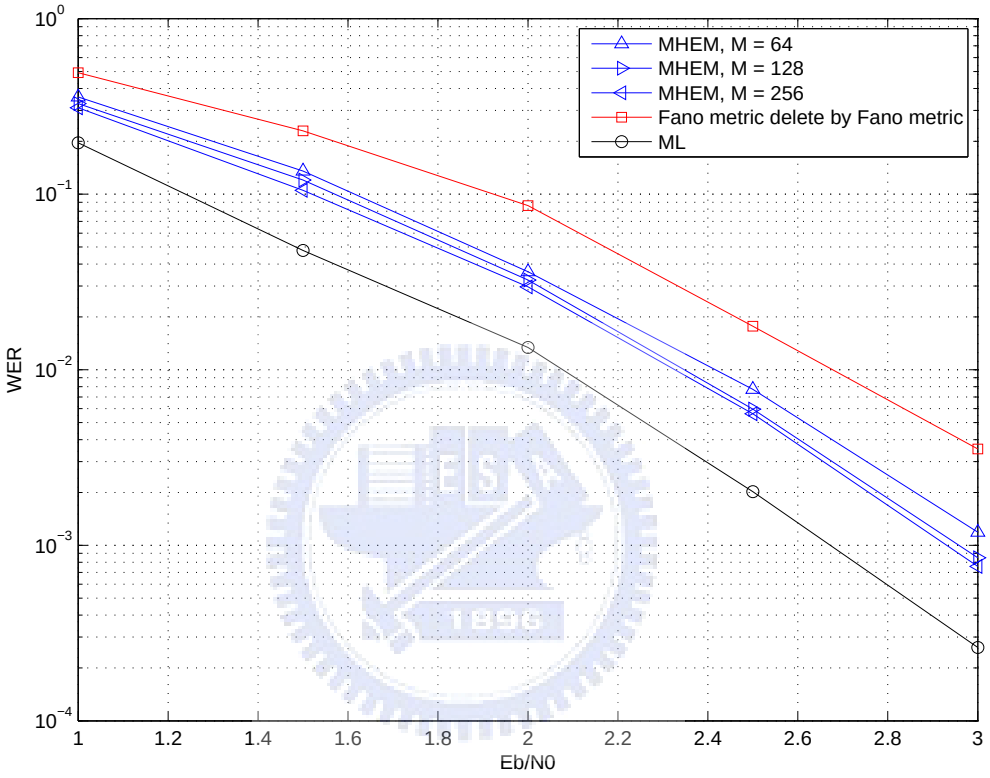


Figure 5.55: Word error rate (WER) performance of MHEM-enhanced two-pass decoder for (2,1,12) convolutional code with generator polynomial [17663,11271]. The stack size is 2^8-1, and the information sequence length $L = 200$.

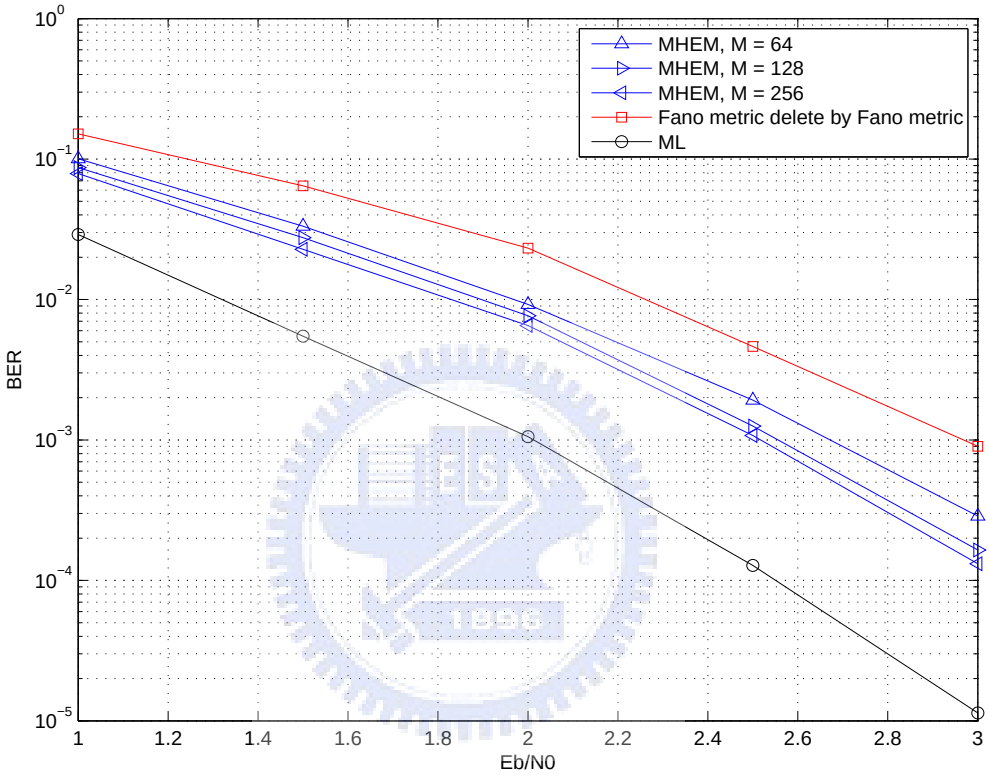


Figure 5.56: Bit error rate (BER) performance of MHEM-enhanced two-pass decoder for (2,1,12) convolutional code with generator polynomial [17663,11271]. The stack size is 2^8-1, and the information sequence length $L = 200$.

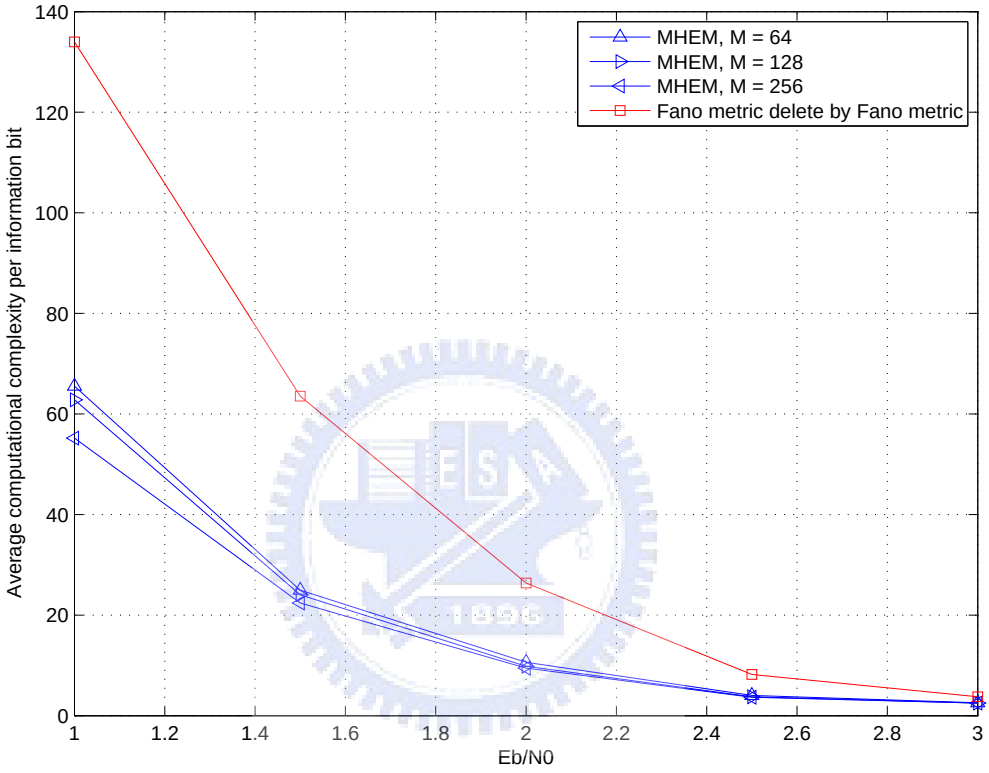


Figure 5.57: Average computational complexity per information bit of MHEM-enhanced two-pass decoder for (2,1,12) convolutional code with generator polynomial [17663,11271]. The stack size is $2^8 - 1$, and the information sequence length $L = 200$.

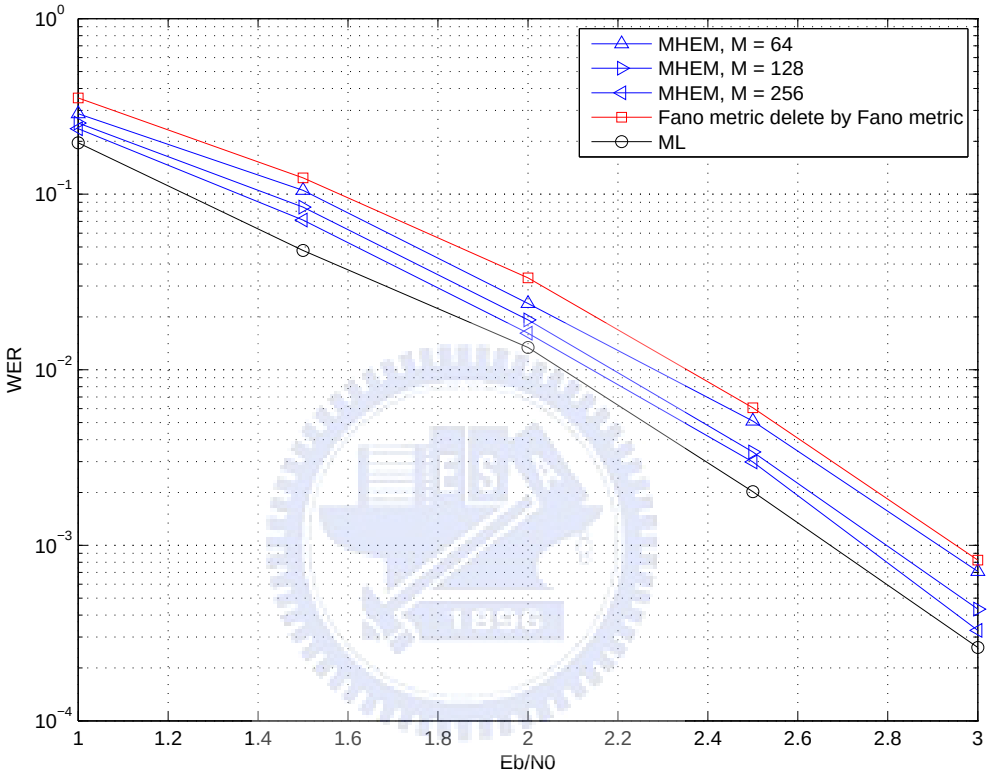


Figure 5.58: Word error rate (WER) performance of MHEM-enhanced two-pass decoder for (2,1,12) convolutional code with generator polynomial [17663,11271]. The stack size is $2^{10} - 1$, and the information sequence length $L = 200$.

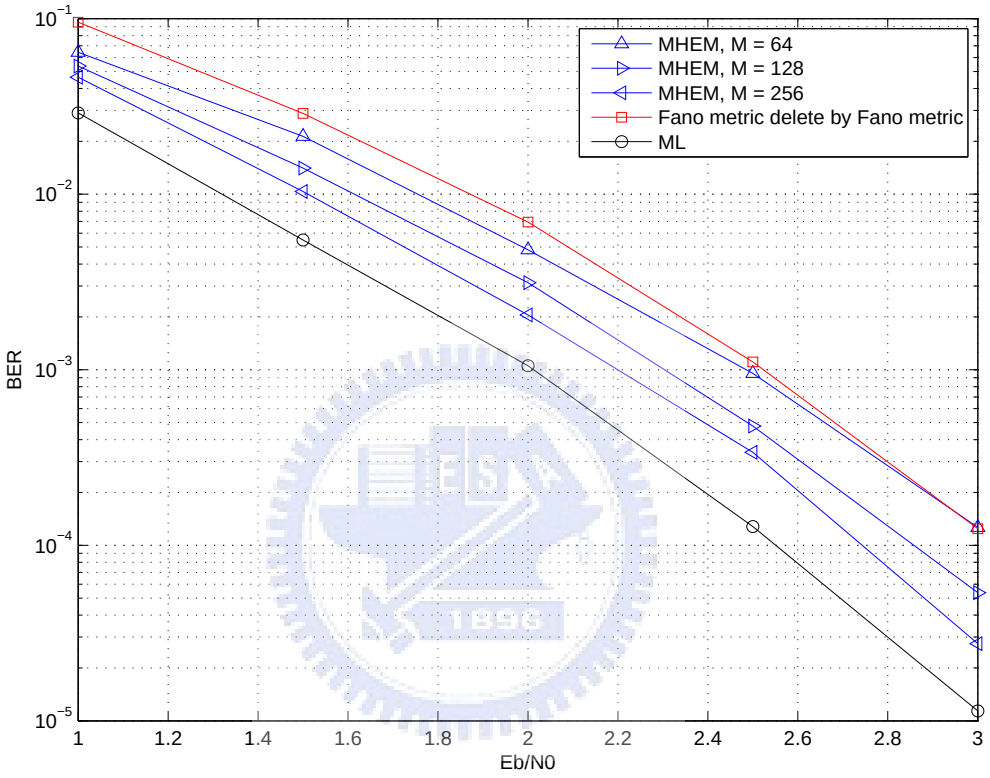


Figure 5.59: Bit error rate (BER) performance of MHEM-enhanced two-pass decoder for (2,1,12) convolutional code with generator polynomial [17663,11271]. The stack size is $2^{10}-1$, and the information sequence length $L = 200$.

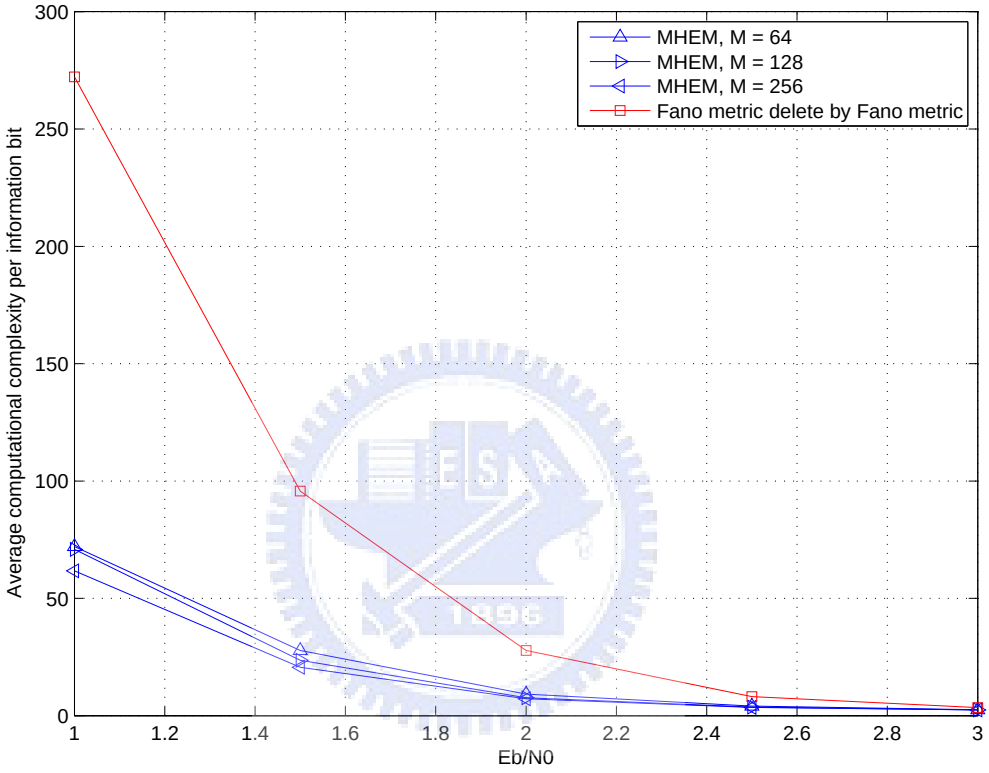


Figure 5.60: Average computational complexity per information bit of MHEM-enhanced two-pass decoder for (2,1,12) convolutional code with generator polynomial [17663,11271]. The stack size is $2^{10} - 1$, and the information sequence length $L = 200$.

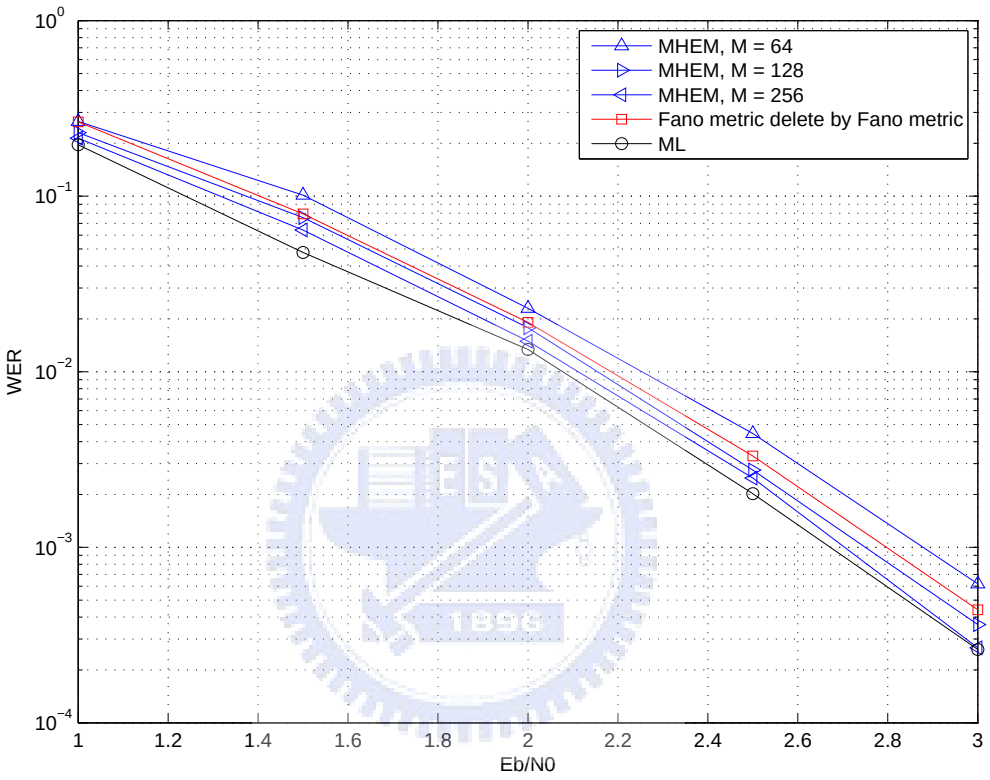


Figure 5.61: Word error rate (WER) performance of MHEM-enhanced two-pass decoder for (2,1,12) convolutional code with generator polynomial [17663,11271]. The stack size is $2^{12} - 1$, and the information sequence length $L = 200$.

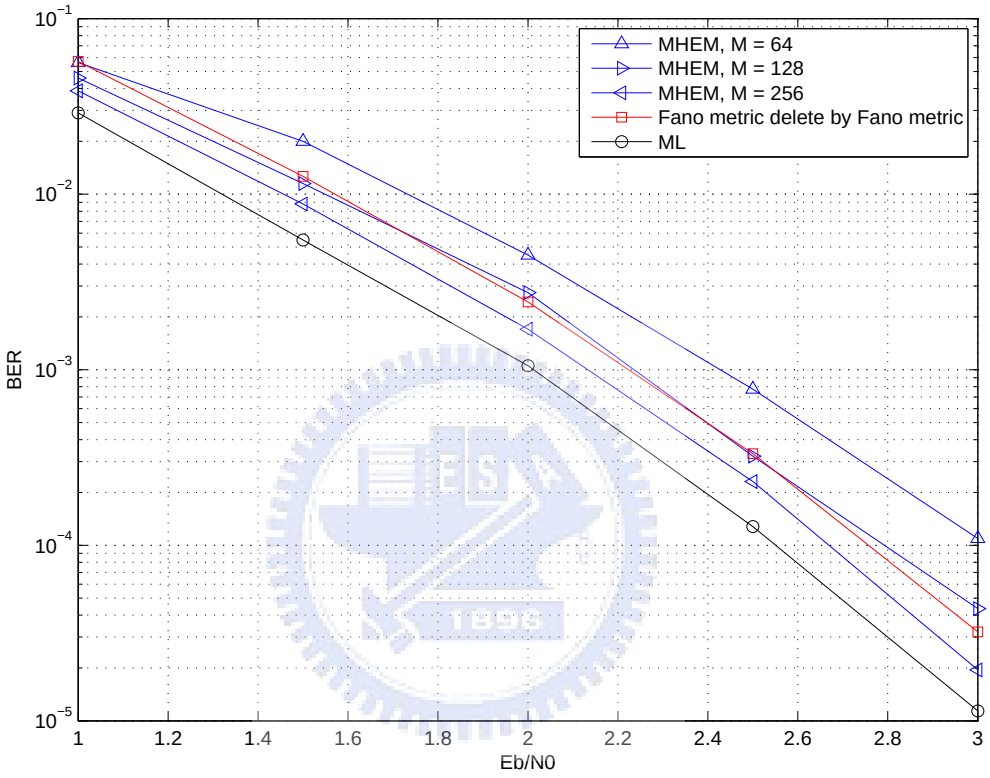


Figure 5.62: Bit error rate (BER) performance of MHEM-enhanced two-pass decoder for (2,1,12) convolutional code with generator polynomial [17663,11271]. The stack size is $2^{12}-1$, and the information sequence length $L = 200$.

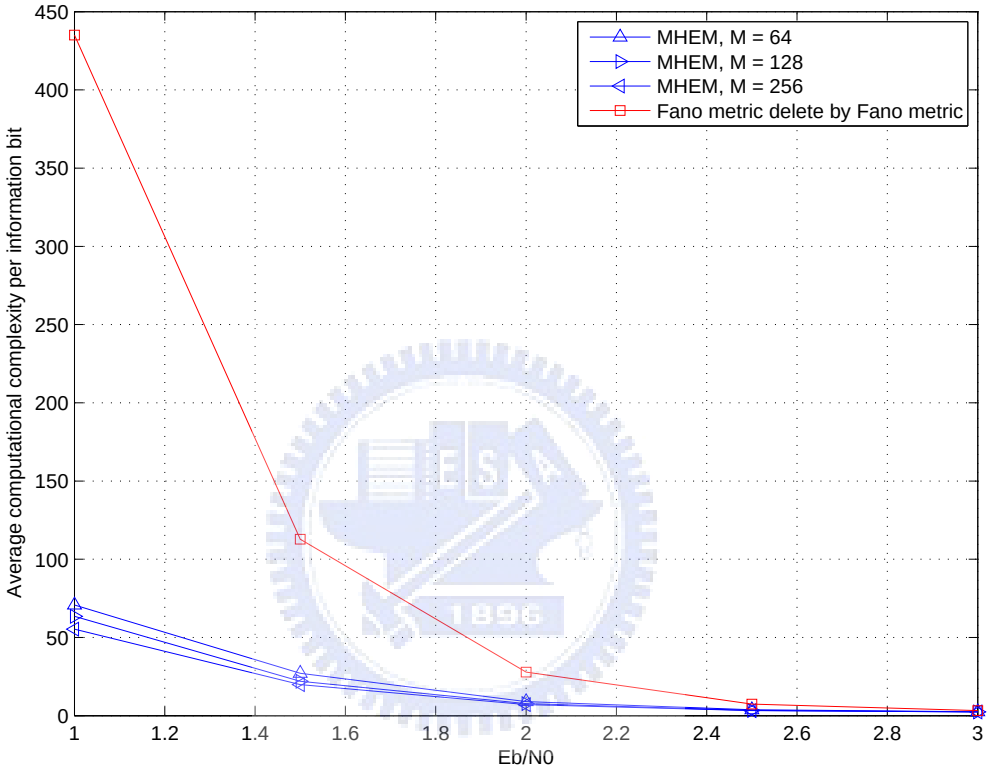


Figure 5.63: Average computational complexity per information bit of MHEM-enhanced two-pass decoder for (2,1,12) convolutional code with generator polynomial [17663,11271]. The stack size is $2^{12} - 1$, and the information sequence length $L = 200$.

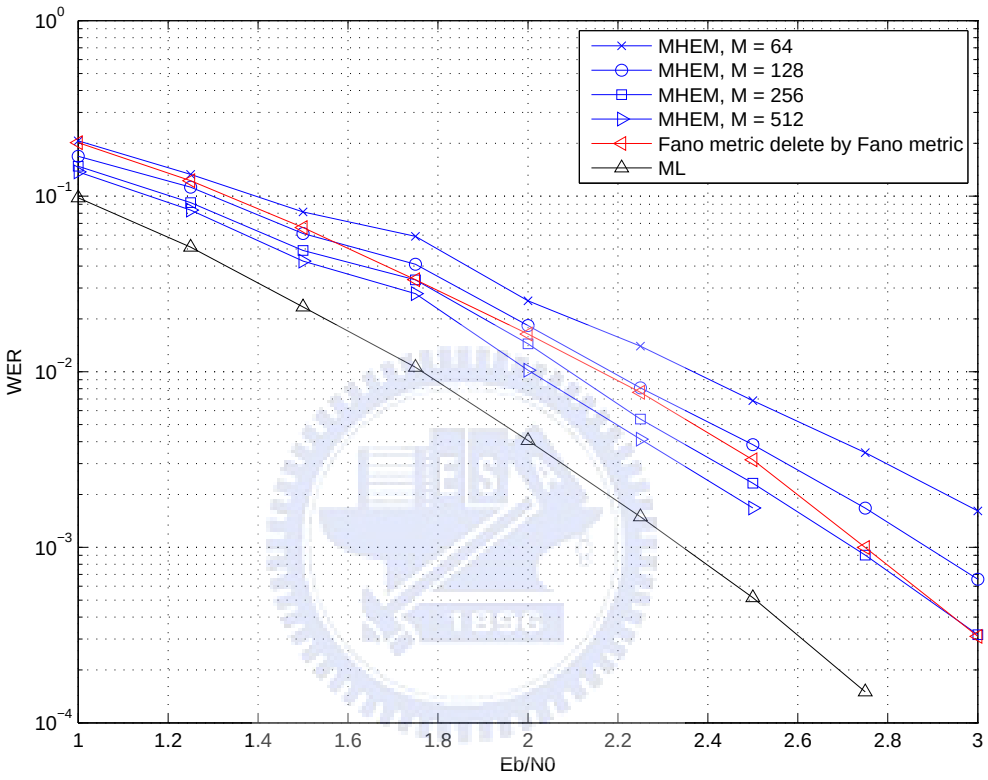


Figure 5.64: Word error rate (WER) performance of MHEM-enhanced two-pass decoder for (2,1,16) convolutional code with generator polynomial [715022,514576]. The stack size is $2^{12} - 1$, and the information sequence length $L = 100$.

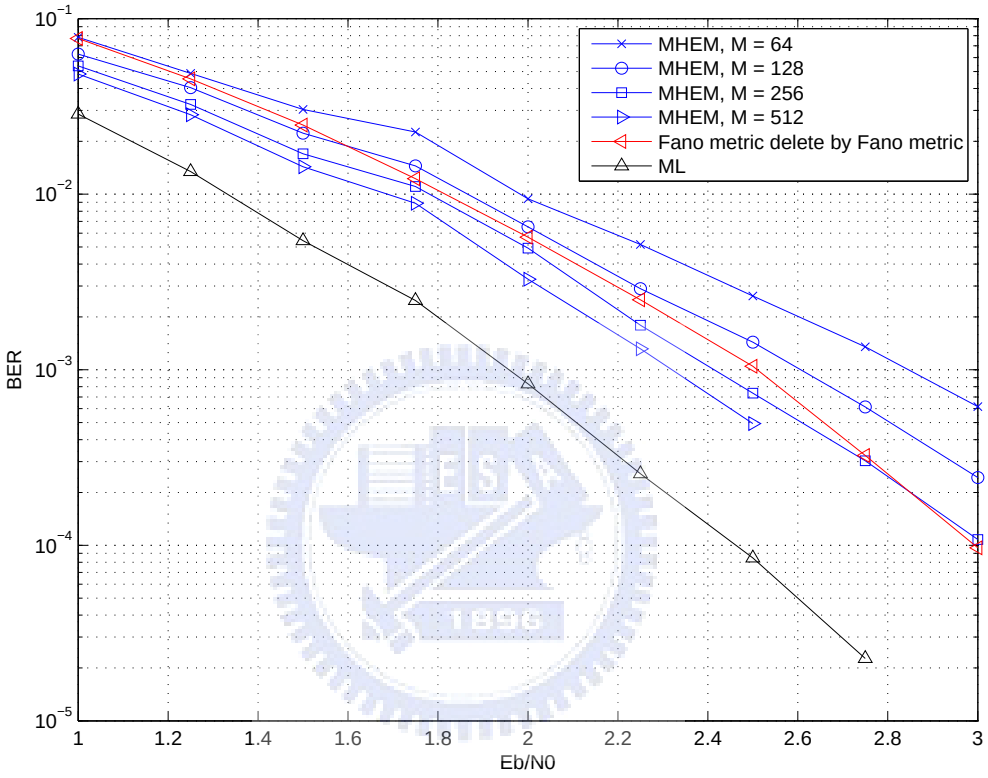


Figure 5.65: Bit error rate (BER) performance of MHEM-enhanced two-pass decoder for (2,1,16) convolutional code with generator polynomial [715022,514576]. The stack size is $2^{12} - 1$, and the information sequence length $L = 100$.

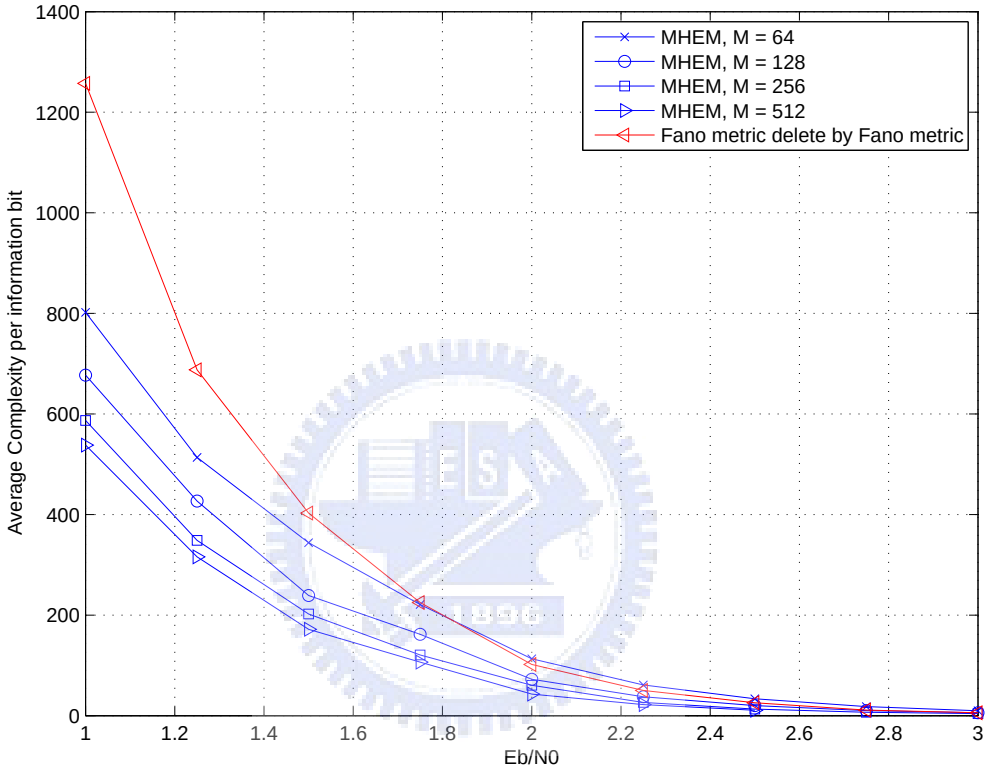


Figure 5.66: Average computational complexity per information bit of MHEM-enhanced two-pass decoder for (2,1,16) convolutional code with generator polynomial [715022,514576]. The stack size is $2^{12} - 1$, and the information sequence length $L = 100$.

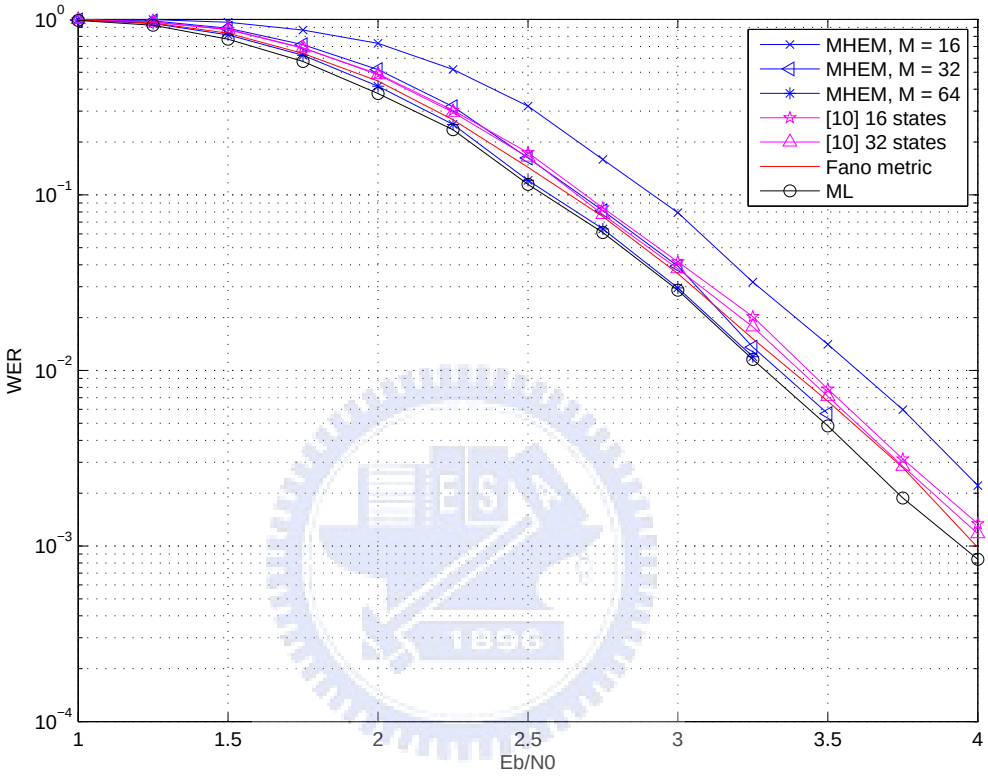


Figure 5.67: Word error rate (WER) performance of MHEM-enhanced two-pass decoder, and low-complexity suboptimal decoding algorithm in [9] for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $2^{18} - 1$, and the information sequence length $L = 2048$.

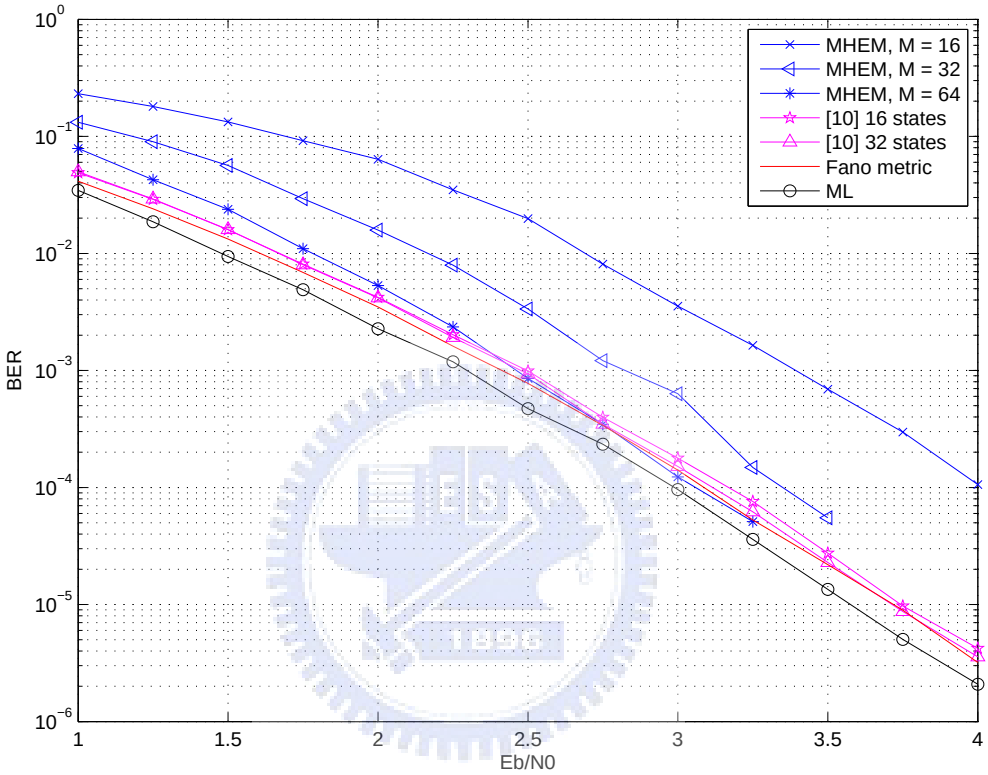


Figure 5.68: Bit error rate (BER) performance of MHEM-enhanced two-pass decoder, and low-complexity suboptimal decoding algorithm in [9] for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $2^{18} - 1$, and the information sequence length $L = 2048$.

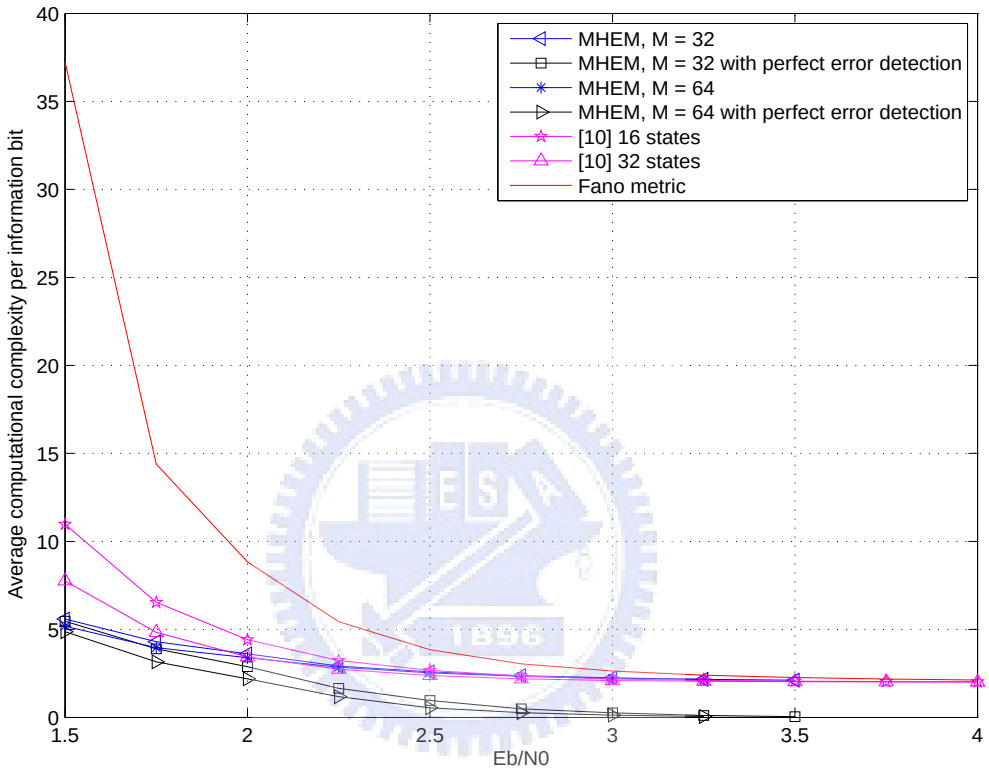


Figure 5.69: Average computational complexity per information bit of MHEM-enhanced two-pass decoder, and low-complexity suboptimal decoding algorithm in [9] for (2,1,8) convolutional code with generator polynomial [457,755]. The stack size is $2^{18} - 1$, and the information sequence length $L = 2048$.

Chapter 6

Concluding Remark and Future work

In this thesis, subject to the practical consideration of finite stack size, we first proposed several path deletion schemes for use of sequential-type decoding algorithm. Afterwards, considering the possibility of off-line decoding, in which the decoding process launches after the reception of entire received vector, we further improved these path deletion schemes by proposing an alternative heuristics estimation method based on two-pass decoding structure. It is empirically shown that the computational complexity of the forward pass not only considerably improves over the stack algorithm with Fano metric under the premise that the backward M-algorithm can be hardware-implemented, but also is smaller than that of the two-pass supercode decoder proposed in [9].

At the current stage, the appropriate M is still obtained through simulations. It would be of practical interest if M can be identified analytically for a given code parameters as well as structure.

References

[1] J. B. Anderson and S. Mohan, "Sequential coding algorithms: A survey and cost analysis," *IEEE Trans. Commun.*, vol. COM-32, pp. 169-176, February 1984.

[2] P. A. Bengough and S. J. Simmons,"Sorting-based VLSI architectures for the M-algorithm and T-algorithm trellis decoders," *IEEE Trans. Commun.*, vol. 43(2/3/4), pp. 514–522, April 1995.

[3] R. Fano, "A heuristic discussion of probabilistic decoding," *IEEE Trans. Inform. Theory*, vol. IT-9, no. 2, pp. 64–73, April 1963.

[4] Y. S. Han, P.-N. Chen and H.-B. Wu, "A maximum-likelihood soft-decision sequential decoding algorithm for binary convolutional codes," *IEEE Trans. Commun.*, vol. 50, no. 2, pp. 173–178, February 2002.

[5] F. Jelinek, "A fast sequential decoding algorithm using a stack," *IBM J. Res. Develop.*, vol. 13, pp. 675–685, November 1969.

[6] J. L. Massey, "Variable-length codes and the Fano metric," *IEEE Trans. Inform. Theory*, vol. 18, no. 1, pp. 196–198, 1972.

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

[8] S. L. Shieh, P.-N. Chen, Y. S. Han,"Reduction of computational complexity and sufficient stack size of the MLSDA by early elimination," in *IEEE International Symposium on Information Theory*, Nice, France, pp. 1671-1675, June 2007.

[9] M. Sikora and D. J. Costello, Jr., "Supercode heuristics for tree search decoding," in *Proc. IEEE Inform. Theory Workshop*, Porto, Portugal, pp. 411-415, May 2008.

[10] S. J. Simmons, "A bitonic-sorter based VLSI implementation of the M-algorithm," in *IEEE Pacific Rim Conference on Communications, Computer ans Signal Processing*, pp. 337–340, June 1989.

[11] K. Zigangirov, "Some sequential decoding procedures," *Probl. Peredach. Inform.*, vol. 2, no. 4, pp. 13–15, 1966.