A State-Reduction Viterbi Decoder for Convolutional Codes with Large Constraint Lengths

Prepared by Sun-Sen Wang

Advisory by Prof. Po-Ning Chen

In Partial Fulfillment of the Requirements

For the Degree of
Master of Science

Department of Communications Engineering

National Chiao Tung University

Hsinchu, Taiwan 300, R.O.C.

E-mail: u8913522@cc.nctu.edu.tw

June, 2002

Abstract

A popular combination in modern coding system is the convolutional encoder and the Viterbi decoder [5]. With a proper design, they can jointly provide an acceptable performance with feasible decoding complexity. In such a combination, a tradeoff on the error performance and the decoding complexity resides on the choice of the code constraint length. Specifically, the probability of Viterbi decoding failure decreases exponentially as the code constraint length increases. However, an increment of code constraint lengths also exponentially increases the computational effort of the Viterbi decoder. Nowadays, the implementation technology on the Viterbi decoder can only accommodate convolutional codes with a constraint length no greater than nine, which somehow limits the achievable error performance.

On the other hand, the construction of convolutional codes with very large constraint lengths are now possible in both theory and practice, yet Monte Carlo simulations of their resultant maximum-likelihood performance is technically infeasible. The authors of [10] then presented a new simulation technique called Important Sampling, which can accurately estimate the maximum-likelihood error performance of convolutional codes with constraint length up to 24 or higher. The authors proved by Important Sampling simulations that the error performance of convolutional codes with certain constraint length can actually be close to the Shannon limit although no feasible decoder can decode such codes.

In this thesis, we propose a reduced-state Viterbi decoder with fixed decoding complexity for use of codes with large constraint lengths. Since, by [10], the maximum-likelihood error

performance of codes with large constraint length is very good, a degradation due to the sub-optimal state reduction at the decoder still provides an acceptably good performance. Performance impact from choosing different decoder parameters, such as state size and sliding window size, are also examined in this thesis.

Contents

List of Figures

Chapter 1

Introduction

From the known principle that the maximum-likelihood error rate of convolutional codes decreases exponentially with the code constraint length, it is reasonably anticipated that a very good system performance can be achieved by using a long-constraint-length convolutional code and its respective maximum-likelihood decoder. This anticipation is quantitatively confirmed through the Important Sampling simulations [10], where a convolutional code with constraint length 24 or higher can achieve 10^{-5} bit error rate at a signal-to-noise ratio per information bit around 1 dB. Although the idea of using long-constraint-length codes is possible in theory, the high computational complexity of maximum-likelihood decoders of long-constraint-length convolutional codes, such as Viterbi algorithm, makes the idea infeasible in practice.

Since, by [10], the maximum-likelihood error performance of codes with large constraint length is already close to the Shannon limit, which is a limit that any code cannot surpass, a small performance degradation due to some reduction in decoding complexity may still give a good performance. This leads us to the developement of a sub-optimal Viterbi decoding algorithm with reduced complexity, which can operate for codes with long constraint lengths. Certainly, the complexity reduction has to catch the main statistics of the noise in order not to introduce large performance degradation.

A straightforward step to reduce the decoding complexity of the Viterbi algorithm is to cut back the *states* examined during the decoding process. This is because the number of *states* examined is the key factor that affects the Viterbi decoding complexity. However, how to structurally remove the states during the decoding process so that only a little performance degradation is resulted still requires certain research efforts. There has been several approaches proposed in the literature. As an example, Feldmann and Harris proposed an adaptive algorithm [4], which uses a technique called *path segment match*. The adaptive algorithm collects several path metrics, and maps them into a path segment metric. The decoding process therefore becomes a testing based on the path segment metric match. The authors showed that with appropriate algorithm parameters, the adaptive algorithm can perform closely to the conventional Viterbi algorithm under a smaller computational complexity.

Another method is the adaptive Viterbi algorithm [7], which was proposed by Chan and Haccoun. The algorithm only keeps a certain amount of most-likely states, which are the states whose respective path metrics are below a given threshold. By a proper decision threshold, the algorithm also performs closely to the conventional Viterbi algorithm with a reduction in computational complexity.

The rest of the thesis is organized as follows. In Chapter 2, some existing complexity-reduction Viterbi algorithms are introduced in more details, and their pro and cons are presented. Another well-known algorithm, named M-algorithm, will also be introduced, and a discussion on the M-algorithm from the aspect of randomized algorithm will be given. In Chapter 3, we will explicitly describe our proposed state-reduction Viterbi algorithm, extending from the M-algorithm. Simulation results will be provided in Chapter 4. Chapter 5 concludes the thesis.

Chapter 2

Background

2.1 Adaptive Viterbi algorithms

As we have mentioned in the previous chapter, two adaptive reduced-complexity Viterbi algorithms have been proposed. For clarity, we will refer to the one proposed by Feldmann and Harris as the *segment-match Viterbi algorithm* [4], and reserve the name of the *adaptive Viterbi algorithm* for the one in [7].

The main focus of these algorithms is to adjust the number of states preserved depending on the variation of channel noises such that it would prevent the loss of the correct path. The key difference of these algorithms is on their approaches to adjust the states. Some researchers combine several states into one to reduce the metric computation [4], and some researchers set a threshold on the path metric to statistically eliminate unlikely efforts [7]. But none of these algorithms can hard-ensure the amount of reduction in computational effort. In other words, the reduction in computational complexity is statistically dependent on the channel noises. As anticipated, these algorithms can achieve decent performance as the conventional Viterbi decoding algorithm with a marked reduction in computational complexity, if a proper system parameter has been selected. Nonetheless, these algorithms provide feasible methods to overcome the problem of high decoding complexity so that

convolutional codes with long constraint lengths can be decoded.

Take the segment-match Viterbi algorithm as an example. It seems that this algorithm can greatly reduce the number of metric computations. However, from another aspect, it also introduce extra complexity due to segment-match. Additionally, the segment-match Viterbi algorithm requires a certain number of well-selected system parameters in order to be well-performed, and these well-selected system parameters may need to be re-selected when the channel statistics, such as noise power, is changed. These complexities in its mechanism (that is, parameters adjustment and segment match) may require additional decoding effort.

As for the adaptive Viterbi algorithm, the average number of states preserved during the decoding process is less than the conventional Viterbi decoding algorithm. However, if channel becomes too good (that is, the channel is very unnoisy), the number of states preserved would be incredibly increased. Hence, when the adaptive Viterbi algorithm is applied to codes with long-constraint length in order to attain a good error performance, its computational complexity may suddenly grow at high signal-to-noise ratio. Some additional amendment action is therefore required for the situation of high signal-to-noise ratio.

2.2 Randomized Algorithm

To overcome the problems of the adaptive algorithms, a new aspect in state reduction may need to take. First of all, if we can fix the number of states examined during the decoding stage, the computational complexity becomes a constant, independent of the channel noise as well as the code constraint length. The question that naturally follows is how to select the visited states in a structural way such that the resultant algorithm is simple and easily implemented. We will come back to this issue in the next section.

Instinctively, the states being selected should be the states which are more likely to

be part of the correct path. This observation matches the basic idea of the well-known *randomized algorithm* [12]. Hence, a section on *randomized algorithm* [12] for supplying a potentially new aspect is introduced.

The concept behind the randomized algorithm has been proposed for quite a while, and has recently found applications in many areas. Originally, randomized algorithms were used to solve the problems from computational theories. A common example is to measure an area in terms of a simple mechanism that can tell whether a point is inside the area or not. By uniformly generating points over a square that covers the targeted area, the targeted area can be estimated simply by the product between the area of that square and the relative frequency of these random points hitting inside the targeted area. This example suggests that by simply incorporating a random number generator, an accurate estimate of the true solution of a sophisticated problem may be obtained by executing a very simple algorithm.

At the first glance of the previous example, it seems that the randomized algorithm may need to take a long executing time to obtain an accurate estimate of the relative frequency. However, many a sophisticate problem in which the randomized algorithm found effective originally needs exponential computation steps to find the exact answer; hence, introducing the randomized algorithm for these problems provides an easy and speedy way to find an accurate estimate of the exact answer.

For clarity, the randomized quick-sort (RandQS) algorithm that is used to sort a sequence of numbers is introduced here [12].

<Algorithm RandQS>

Input: A set of numbers, S.

Output: The elements of S in ascending order.

Step 1. Choose an element y uniformly at random from S: every element in S has equal

probability of being chosen.

Step 2. By comparing each element of S with y, determine the set S_1 of elements smaller than y and set S_2 of elements larger than y.

Step 3. Recursively sort S_1 and S_2. Output the sorted version of S_1, followed by y, and then the sorted version of S_2.

The difference between original QuickSort algorithm and algorithm RandQS is the way to choose the element y. Notably in Step 1, algorithm RandQS chooses an element y uniformly at random from S; hence, it is a randomized algorithm.

A formal definition of the randomized algorithm is that the randomized algorithm is an algorithm that makes random choices during execution. As learned from algorithm RandQS, a proper incorporation of random selection simplifies the work of sorting. It can be derived that its expected execution time is equal to that of deterministic QuickSort algorithm, and with certain probability, the RandomQS could run faster than the QuickSort.

It should be pointed out that in algorithm RandomQS, no matter which y is chosen, the algorithm always gives the correct solution. However, it is not necessary for the randomized algorithms to give the correct answer. As a result, there are two types of randomized algorithms: Las Vegas algorithms, and Monte Carlo algorithms. As anticipated, the former type of randomized algorithms can provide the correct solution, while the latter type of randomized algorithms only gives an estimate of the correct answer.

Although we somehow envisioned that the concept of the randomized algorithms could be useful in state reduction of the Viterbi algorithm, we did not successfully incorporate a random number generator (according to certain distribution) in our proposed algorithm. We therefore defer it as a future work, and provide this section for those who are interested in this research line.

2.3 M-Algorithm

A key question in this thesis is how to properly select (part of) the states to be examined during the decoding process so that the computational complexity of the Viterbi algorithm can be a feasible constant, independent of the channel statistics. According to the randomized algorithm, these states should be chosen randomly according to some distribution. The next question that follows is what the proper distribution is for state selection. In algorithm RandQS, the point y is selected according to a uniform distribution. However, it can be anticipated that uniformly drawing states may not result in a well-performed state-reduction Viterbi algorithm. This is because the information of the "most likely" pathes becomes independent of the state selection, when uniform state-selection distribution is taken. Indeed, the states examined should be drawn according to their probabilities to be the correct path. In extreme case, we can just sort the states according to such probabilities, and only visit a pre-specified fixed number of those whose being-the-correct-path probabilities are higher. This is exactly the idea behind the M algorithm [1, 14, 13, 15, 8, 2, 3].

The basic principle of the M-algorithm [1] is to keep the M most likely paths among those paths that end at the same level of a trellis, where a "more likely" path is the path with "larger path metric." These M most likely paths, or equivalently states, are extended to the next level of the trellis, yielding at most $2^k \times M$ states for (n, k, m) convolutional code. Again, only M most likely states are kept, and the remaining states are deleted. At the end of the trellis, the M algorithm selects the best path with the largest path metric as the Viterbi algorithm does, and outputs the respective path labels of the located best path.

The M-algorithm is widely used in applications such as CPM [8], because of its feasible complexity and acceptable performance. The complexity of the M algorithm is determined by M. Other than the proportional increasing of the number of examined states with respect

to M, additional sorting overhead for selecting the M most likely paths is required. However, it has been shown [9] that the average sorting effort required can be made roughly linear with M, e.g., by slotting the paths in "buckets" according to a coarse quantization of their metrics as first proposed by Jelinek for the stack algorithm [6]. The results then justify the usage of the M algorithm.

A certain number of papers on the M algorithm have been published [14, 13, 15, 8, 2, 3]. Some papers focused on the choice of minimal M under which the M algorithm can asymptotically achieve the maximum-likelihood decoding performance. However, similar to most of the state-reduction Viterbi algorithms, the M algorithm is in general suboptimal in performance. It simply keeps M paths with larger path metrics, and have no mechanism to examine whether the correct path is among them. In case all vestiges of the correct path has been removed, the algorithm has no information to trace back to the correct path. This is the case called the catastrophic correct path loss.

In the next chapter, we will discuss our state-reduction algorithm on the basis of the M algorithm.

Chapter 3

A state-reduction Viterbi decoding algorithm and its performance

3.1 A state-reduction Viterbi decoding algorithm

Our proposed state-reduction Viterbi decoding algorithm happened to be the same as the M algorithm in its procedure. The main difference is that we are perhaps the first one to apply the M algorithm to soft-decision decoding of convolutional codes with very long constraint lengths. The performance impact of selecting different algorithmic parameters, such as M, constraint lengths of the applied code, and Viterbi sliding windows, are investigated.

The algorithm for decoding an (n, k, m) convolutional code can be described as follows.

<State-reduction Viterbi decoding algorithm>

Step 1. Set $\ell = 0$, and set the single original state (or node) of the trellis as the only surviving state (or node) at level 0.

Step 2. Extending all the surviving states (or nodes) at level ℓ to the next level yields the set of visited states (or nodes) at level $\ell + 1$.

Step 3. For each visited state at level $\ell + 1$, keep only one surviving path that enters the

state (or node) by following the conventional Viterbi algorithm.

Keep at most M states at level $\ell + 1$, which correspond to the M largest surviving path metric.

Step 4. $\ell = \ell + 1$.

Step 5. If ℓ is the last level of the trellis, output the information bit labels that correspond to the (single) surviving path, and stop the algorithm; else go to Step 2.

In the next section, simulations of the reduced state Viterbi algorithm are presented, and dominant model parameters, such as constraint lengths, sliding windows and M, on performance are investigated.

3.2 Simulation results

As aforementioned, we will apply the algorithm to convolutional code with very long constraint length, because a little degradation to the very good maximum-likelihood performance owing to state-reduction may still yield an acceptable performance. It should be a reasonable presumption that the convolutional codes with longer constraint length can endure larger degradation due to the usage of suboptimal decoder.

3.2.1 Simulation models

Ever since the invention of the Viterbi algorithm, the problem of constructing good codes appropriate for the Viterbi algorithm has been examined. The problem becomes much more significant once a sub-optimal Viterbi is employed. There are several decisive factors for convolutional codes appropriate for maximum-likelihood Viterbi decoding (and the additive

white Gaussian noise). Two frequently used decisive factors are *free distance* and *distance profile*.

In our simulations, we use two convolutional codes. One is a $(2, 1, 23)$ convolutional code with generator 51202215, 66575563(octal) and free distance 24. Another is a $(2, 1, 16)$ convolutional code with generator 715022, 514576(octal) and free distance 18 [11]. The information length taken is of 1200 bits. Each experiment is repeated 10,000 times. Hence, there is 1.2×10^7 information bits for each experiment, which can give a resolution of bit error rates up to 10^{-5}.

3.2.2 Degradation due to reduction of maintained states, M

Performance degradation due to state-reduction is first examined. A $(2, 1, 6)$ convolutional code with generators 133,171 (octal) is decoded respectively by the conventional 64-state Viterbi and the state-reduction Viterbi with $M = 32$ under AWGN channels. From Fig. 3.1, the suboptimal performance by the M-algorithm with $M = 32$ approaches the maximum-likelihood performance of Viterbi, especially when E_b/N_0 is moderately large. Even at low SNR, only 0.25 dB degradation is observed due to the elimination of half of the states. This result implies that performance remains almost intact, even if half of trellis states are neglected. An interpretation of perfect performance at high SNR is that the metric of the correct path tends to be relatively larger than that of incorrect paths at higher SNR, so the probability of the correct path located at those states with larger surviving path metrics should be higher.

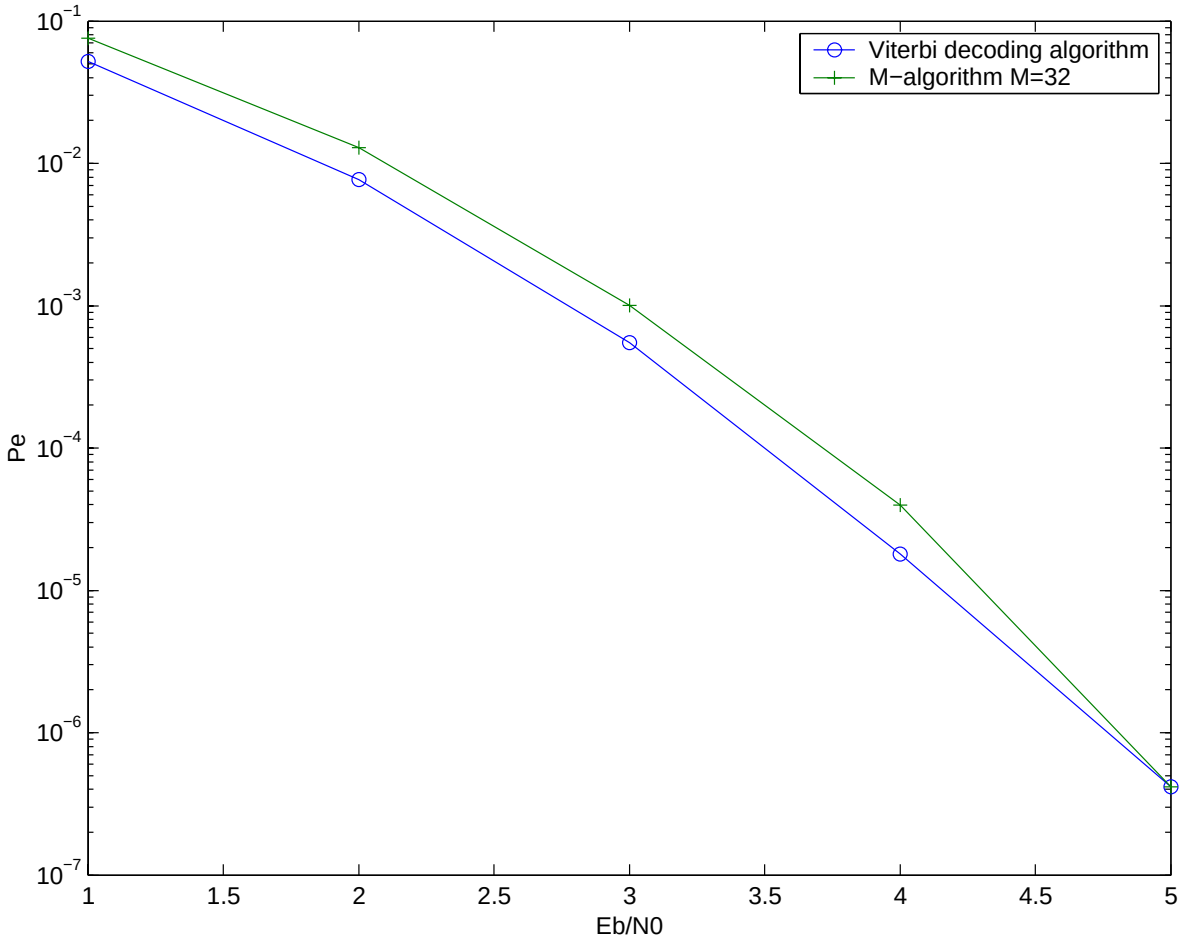

Figure 3.1: BERs of conventional Viterbi and M-algorithms with $M = 32$ for the (2,1,6) convolutional code with generators 133,171 (octal) under AWGN channels.

3.2.3 Performance impact of code constraint length

Next, we use the maximum-likelihood performance of $(2, 1, 6)$ convolutional code as a reference, and examine the M-algorithm performances of $(2, 1, 16)$ and $(2, 1, 23)$ convolutional codes. In this simulation, M is fixed as 64 such that the decoding complexity of the M-algorithm is comparable to the 64-state Viterbi algorithm. The objective is to investigate whether benefits can be gained by adopting a combination of convolutional codes with long constraint length and state-reduction Viterbi algorithm.

Figs. 3.2 and 3.3 summarizes our simulations. Similar trend to Fig. 3.1 is observed, where the state-reduction performance of $(2, 1, 16)$ and $(2, 1, 23)$ codes is closer to the maximum-likelihood performance of $(2, 1, 6)$ code as SNR is higher. At $E_b/N_0 = 5$ dB, the M-algorithm even yields no decoding error for both $(2, 1, 16)$ and $(2, 1, 23)$ convolution codes within our simulation trials. This, to some extent, hints that when $E_b/N_0 \geq 5$ dB, the reduced-state performance can achieve the maximum-likelihood performance for Viterbi with comparable complexity.

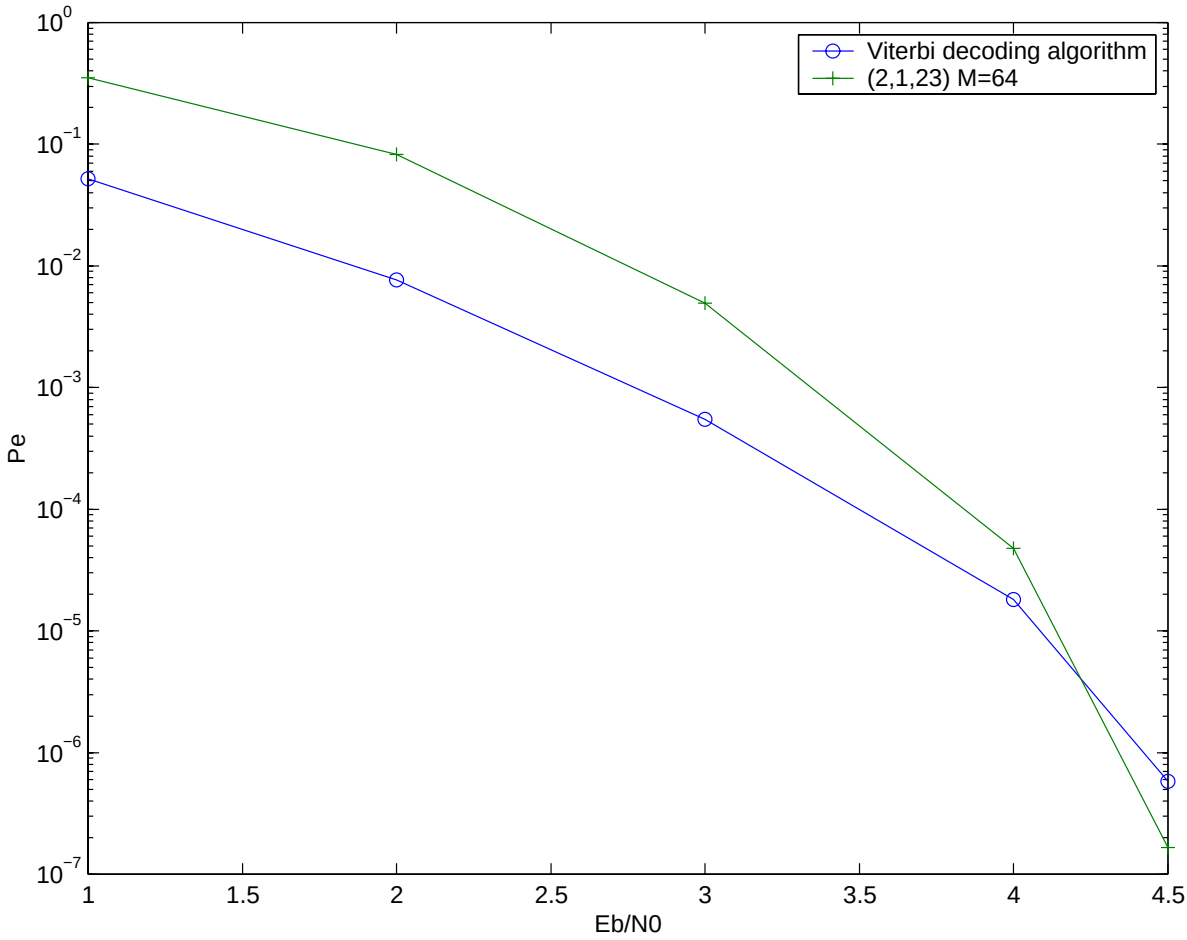

Figure 3.2: BERs of (2,1,6) code decoded by Viterbi algorithm and (2,1,23) code decoded by M-algorithm with $M = 64$ under AWGN channels.

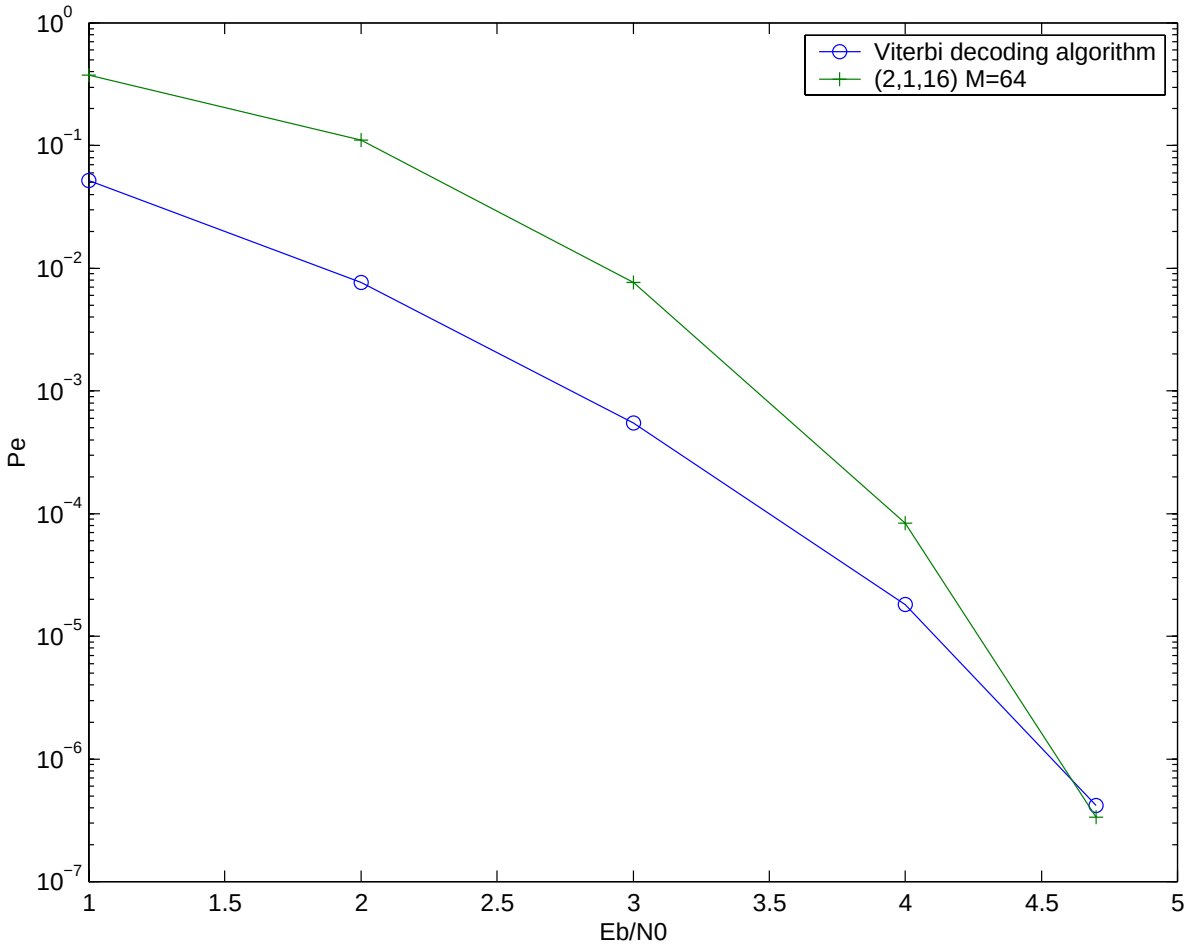

Figure 3.3: BERs of (2,1,6) code decoded by Viterbi algorithm and (2,1,16) code decoded by M-algorithm with $M = 64$ under AWGN channels.

3.2.4 Performance impact of sliding window

In this subsection, we investigate the performance degradation of the M-algorithm due to practical sliding window constraint.

In practical applications, the sliding window determines the decision delay. In order to secure the performance, the sliding window is suggestively to be five times the memory order m of the applied (n, k, m) convolutional code. This ratio was obtained under no state-reduction in Viterbi decoding.

When part of the states is neglected as does by M-algorithm, no conclusive result on the necessary size of sliding window has been established. Experiments are therefore conducted to explore the performance impact of sliding window to state-reduction Viterbi decoding.

Figure 3.4 illustrates the BERs obtained by the 64-state M-algorithm for $(2, 1, 23)$ convolutional code. Two sliding windows are simulated, which are 32 and 120. The two resultant curves almost coincide, which indicates that taking the sliding window to be five times the order of the maintained states (that is, sliding window size is greater than $5 \times \log_2(M)$) seems sufficient. To further examine the observation, another simulation on 32-state M-algorithm is conducted. The result is depicted in Fig. 3.5. This figure, again, confirms that taking $W = 28 \geq \log_2(32)$ suffices to achieve the performance of taking $W = 120$. Similar simulations on (2,1,16) convolution codes, which are summarized in Figs. 3.6 and 3.7, also ensure the same conclusion.

The above conclusion justifies the practical usage of the M-algorithm, since the decoding performance only depends on the order of the maintained states rather than the applied code constraint length. So applying it to codes with long constraint length in order to obtain some performance gain becomes fairly feasible in practice.

Further simulations on the sliding window effect are subsequently conducted. Results

obtained by the 64-state M-algorithm for $(2, 1, 23)$ convolutional code are summarized in Figs. 3.8–3.10. Figures 3.11–3.17 illustrate the results obtained by the 32-state M-algorithm for $(2, 1, 23)$ convolutional code. Figures 3.14–3.19 show the same results for (2,1,16) convolutional code. These results, although they confirm our previous conclusion, hint that a little larger ratio of the sliding window against $\log_2(M)$ may be necessary at lower SNR.

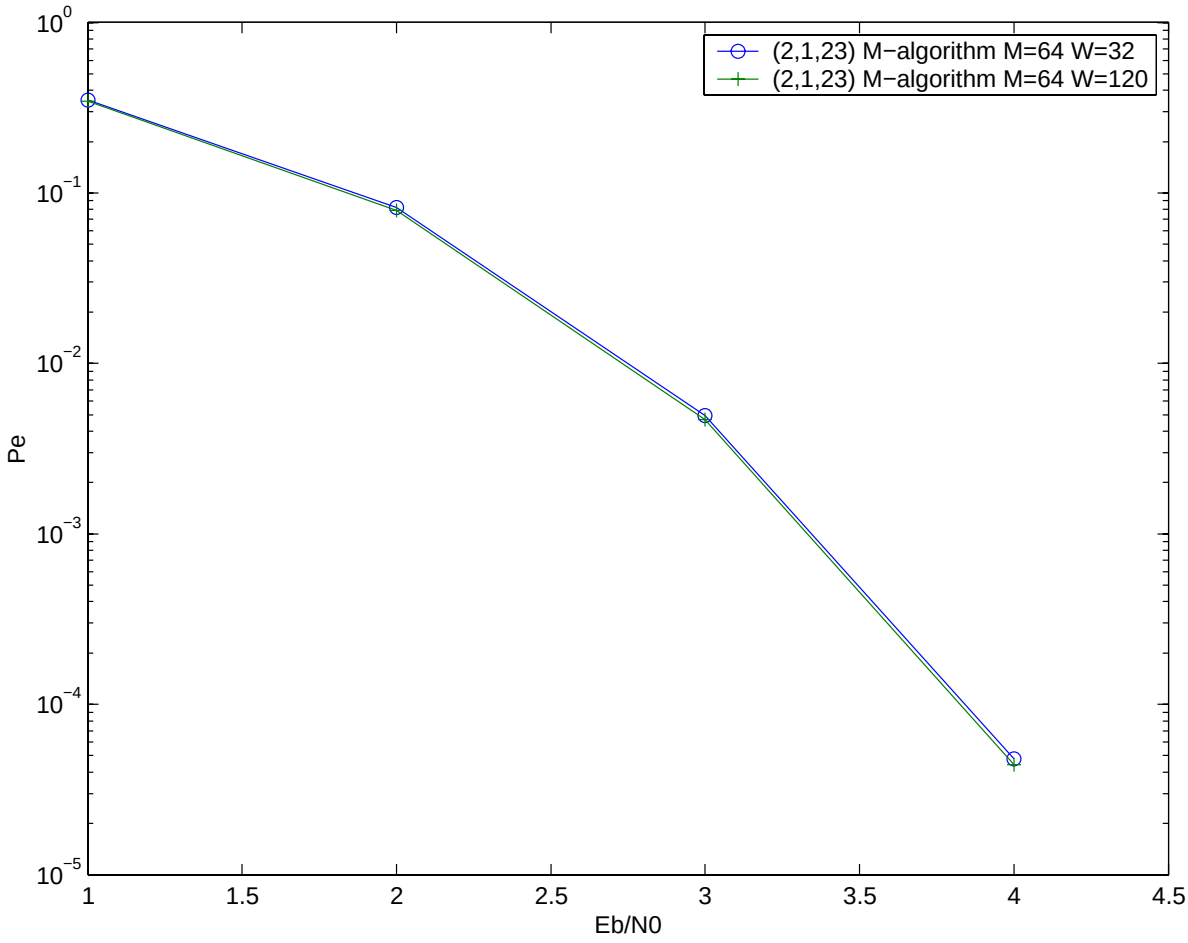

Figure 3.4: BERs obtained by the 64-state M-algorithm with sliding windows being 32 and 120 for (2,1,23) convolutional code under AWGN channels.

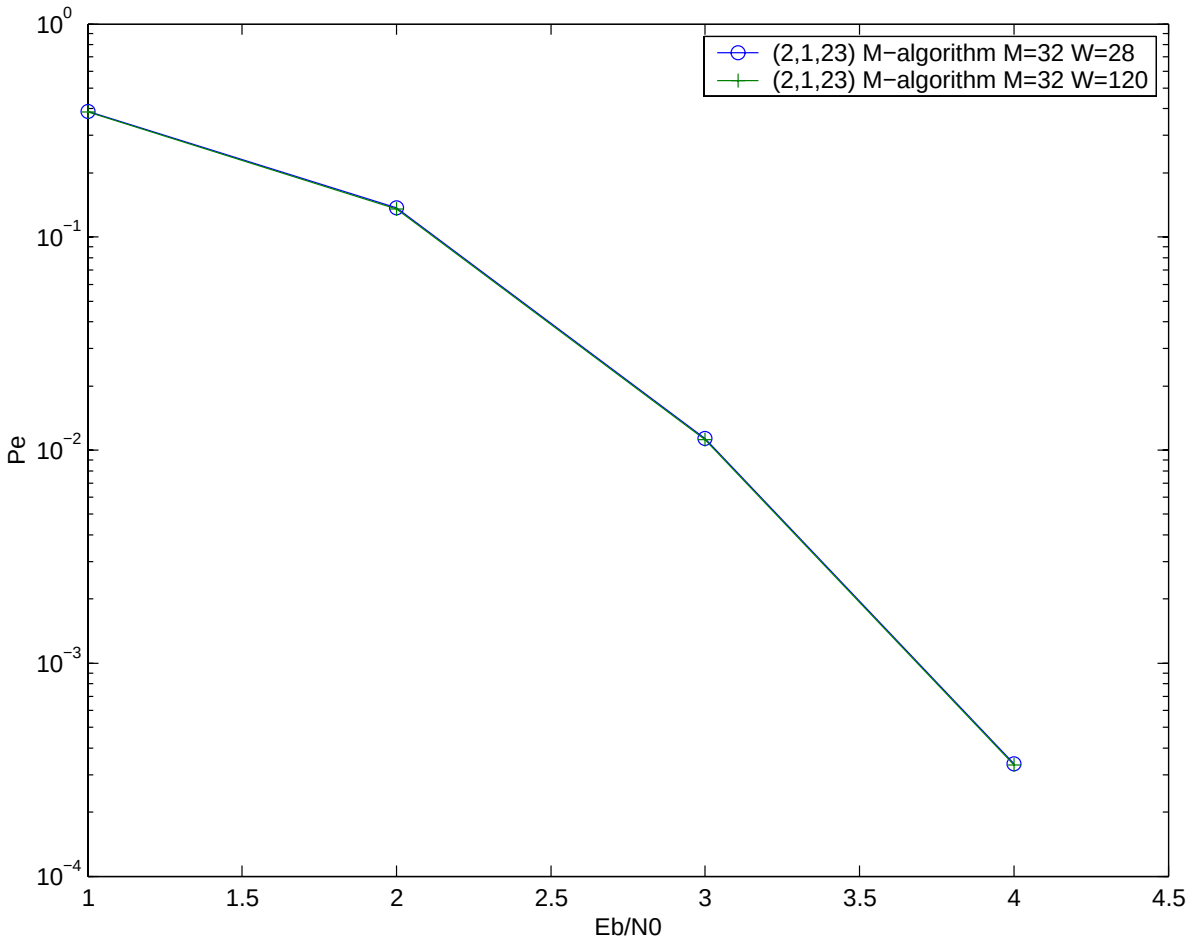

Figure 3.5: BERs obtained by the 32-state M-algorithm with sliding windows being 28 and 120 for (2,1,23) convolutional code under AWGN channels.

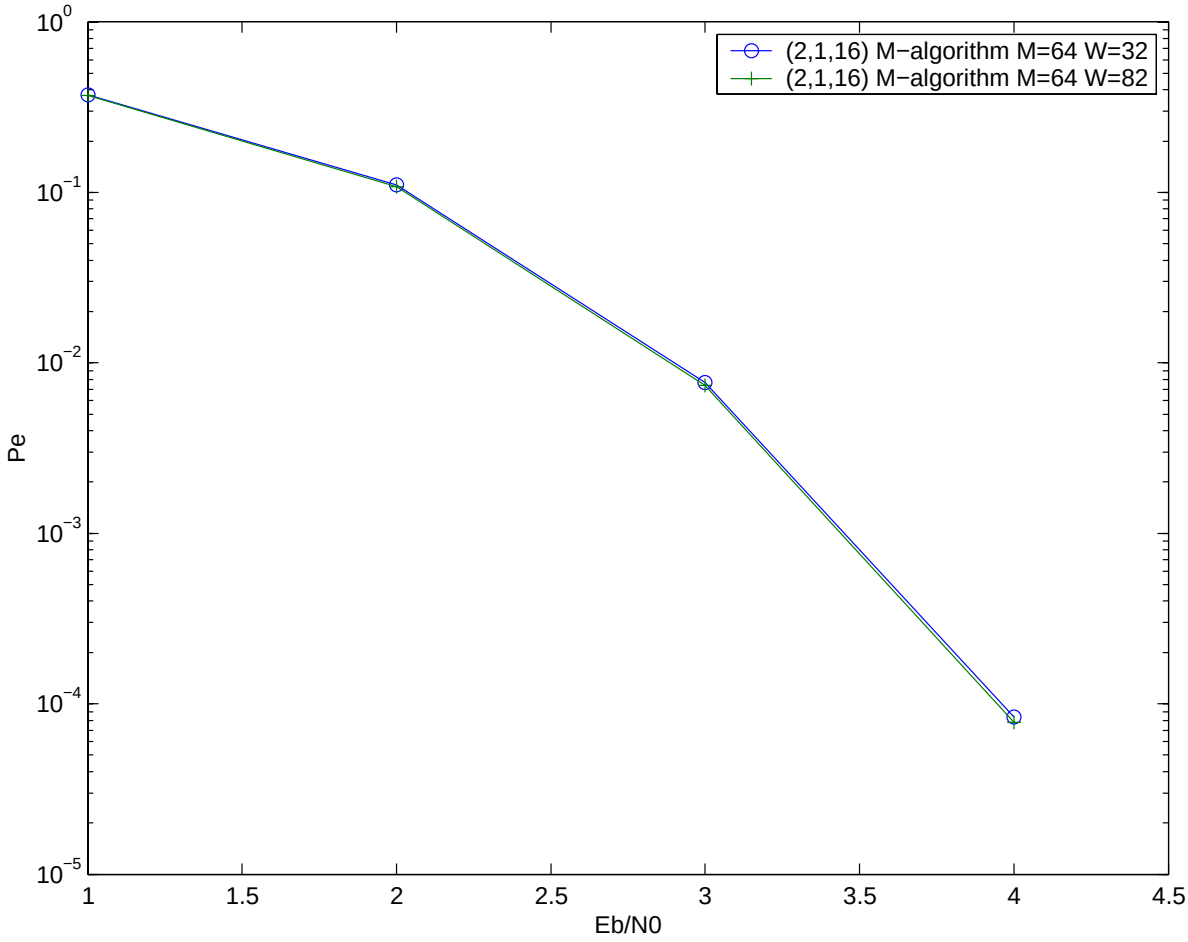

Figure 3.6: BERs obtained by the 64-state M-algorithm with sliding windows being 32 and 82 for (2,1,16) convolutional code under AWGN channels.

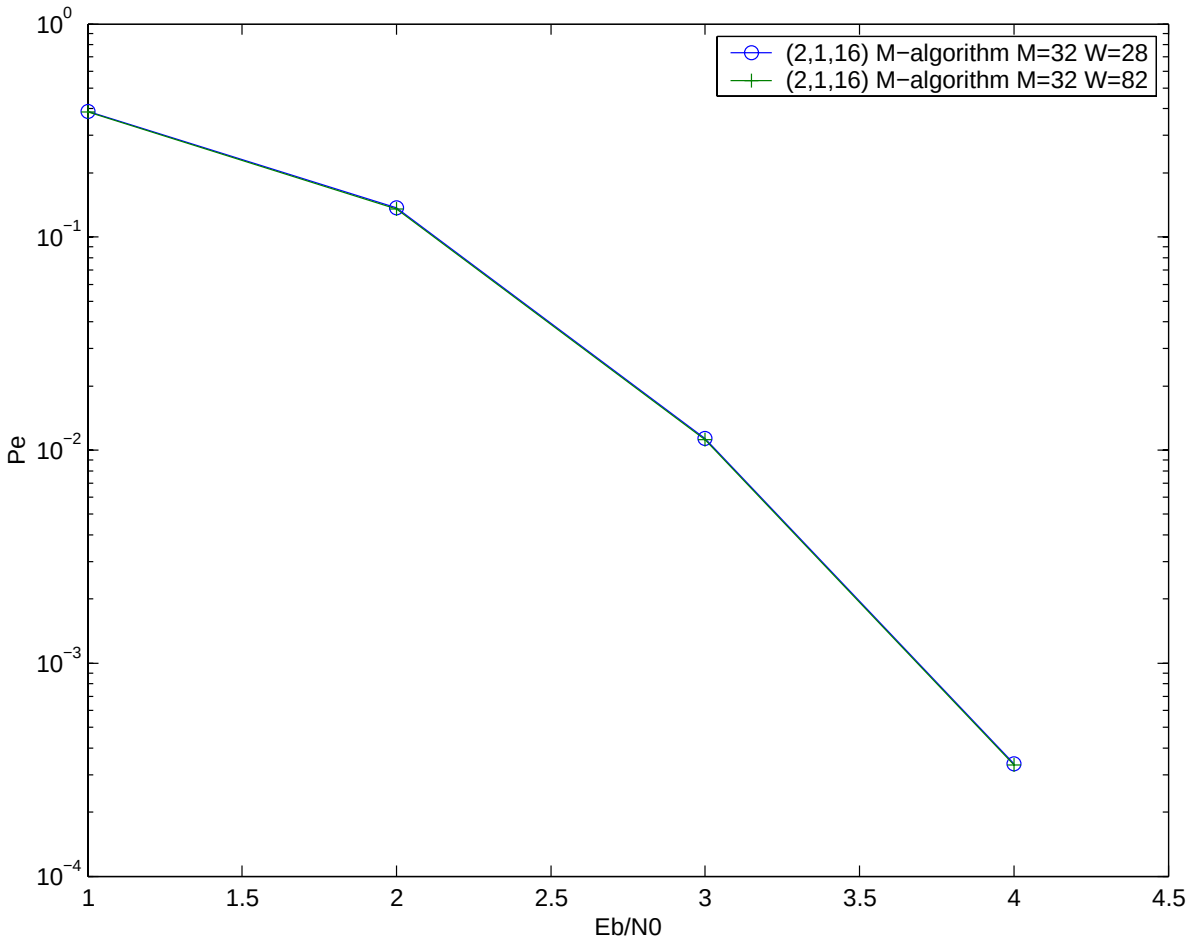

Figure 3.7: BERs obtained by the 32-state M-algorithm with sliding windows being 28 and 82 for (2,1,16) convolutional code under AWGN channels.

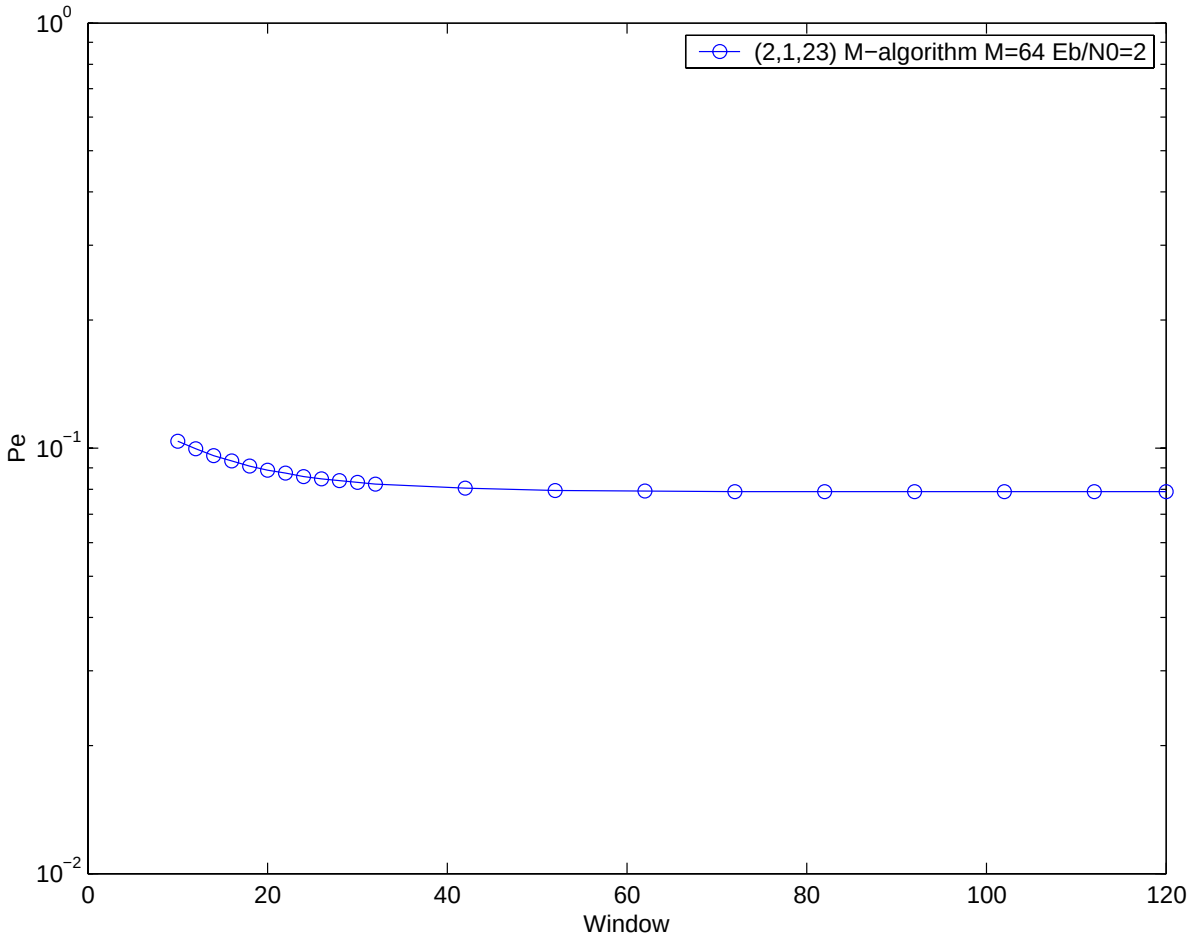

Figure 3.8: BERs obtained by the 64-state M-algorithm with various sliding windows (W = 10–120) for (2,1,23) convolutional code under AWGN channels with $E_b/N_0 = 2$ dB.

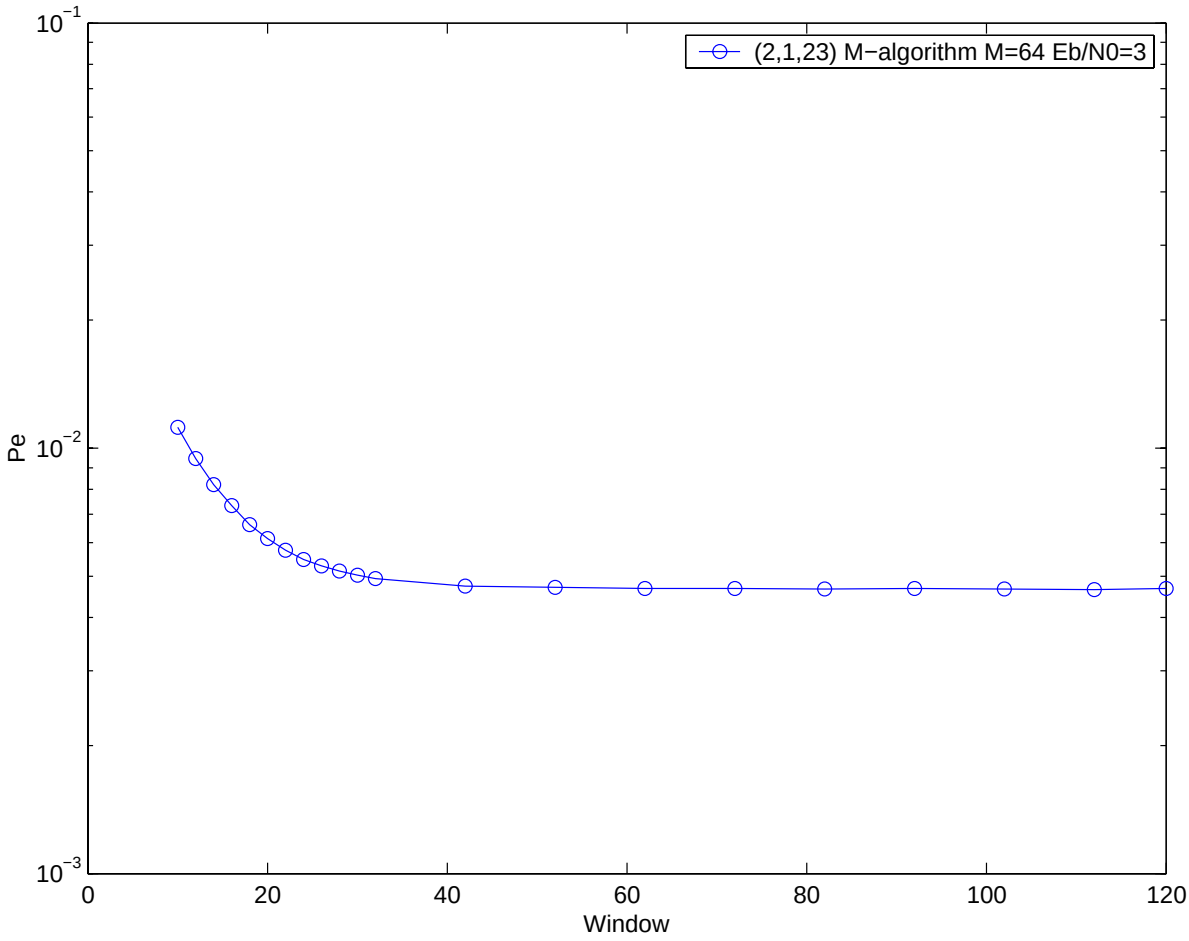

Figure 3.9: BERs obtained by the 64-state M-algorithm with various sliding windows (W = 10–120) for (2,1,23) convolutional code under AWGN channels with $E_b/N_0 = 3$ dB.

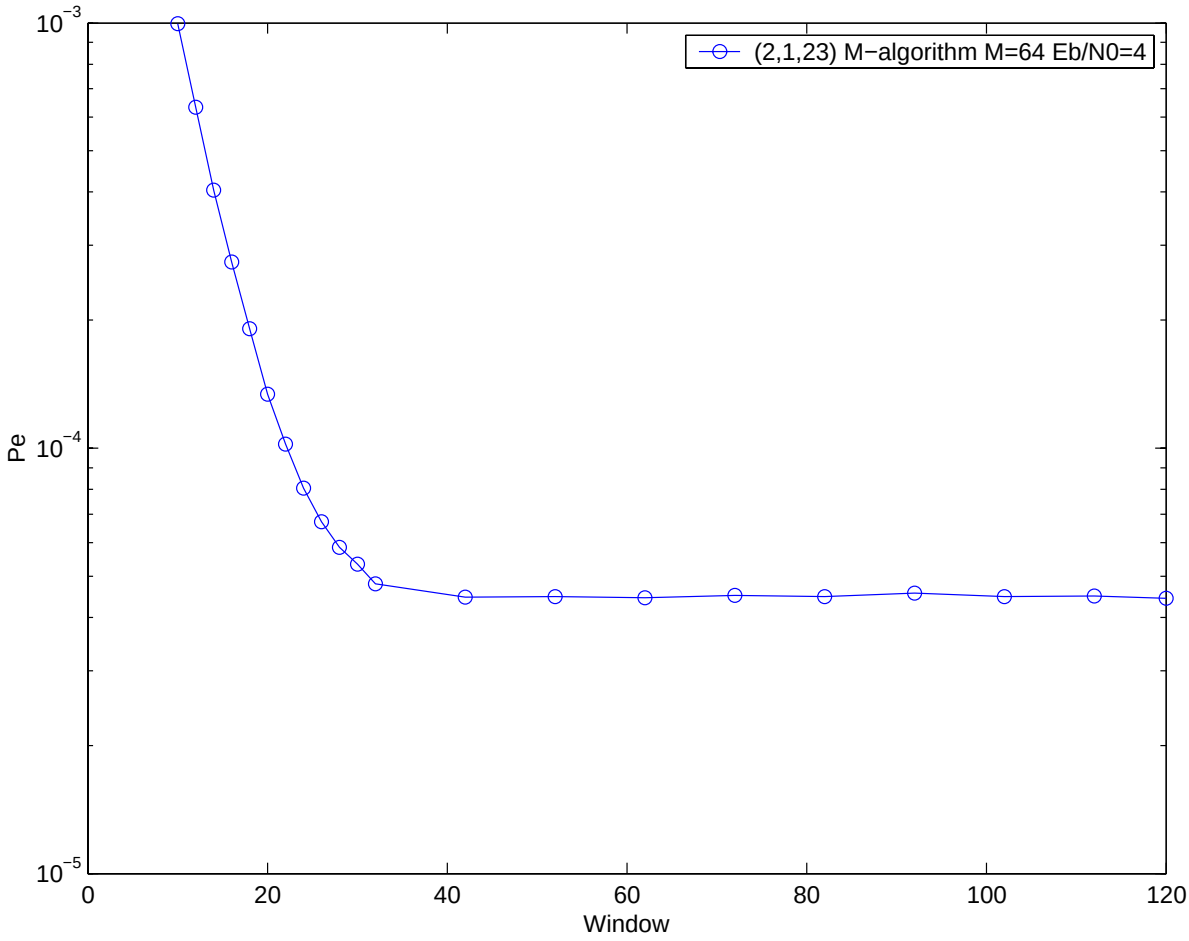

Figure 3.10: BERs obtained by the 64-state M-algorithm with various sliding windows (W = 10–120) for (2,1,23) convolutional code under AWGN channels with $E_b/N_0 = 4$ dB.

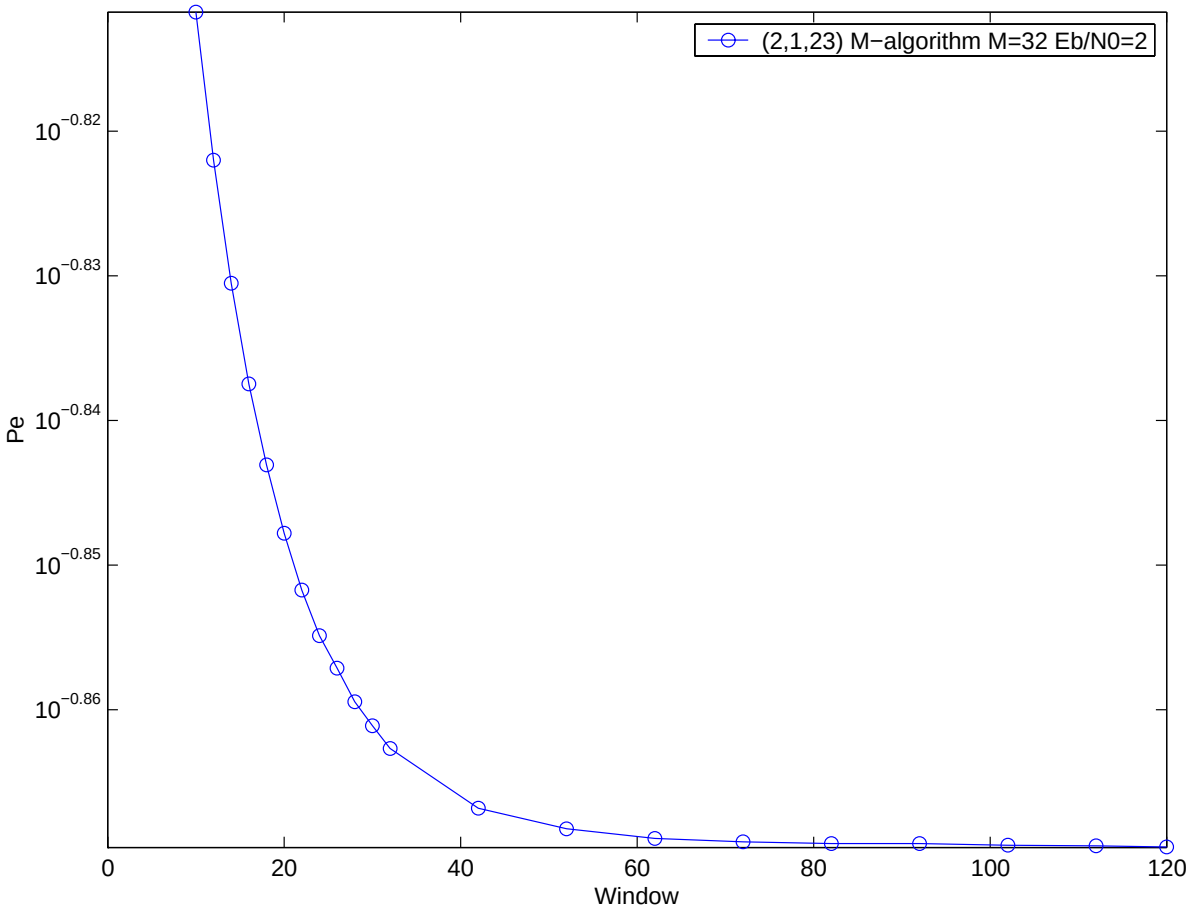

Figure 3.11: BERs obtained by the 32-state M-algorithm with various sliding windows (W = 10–120) for (2,1,23) convolutional code under AWGN channels with $E_b/N_0 = 2$ dB.

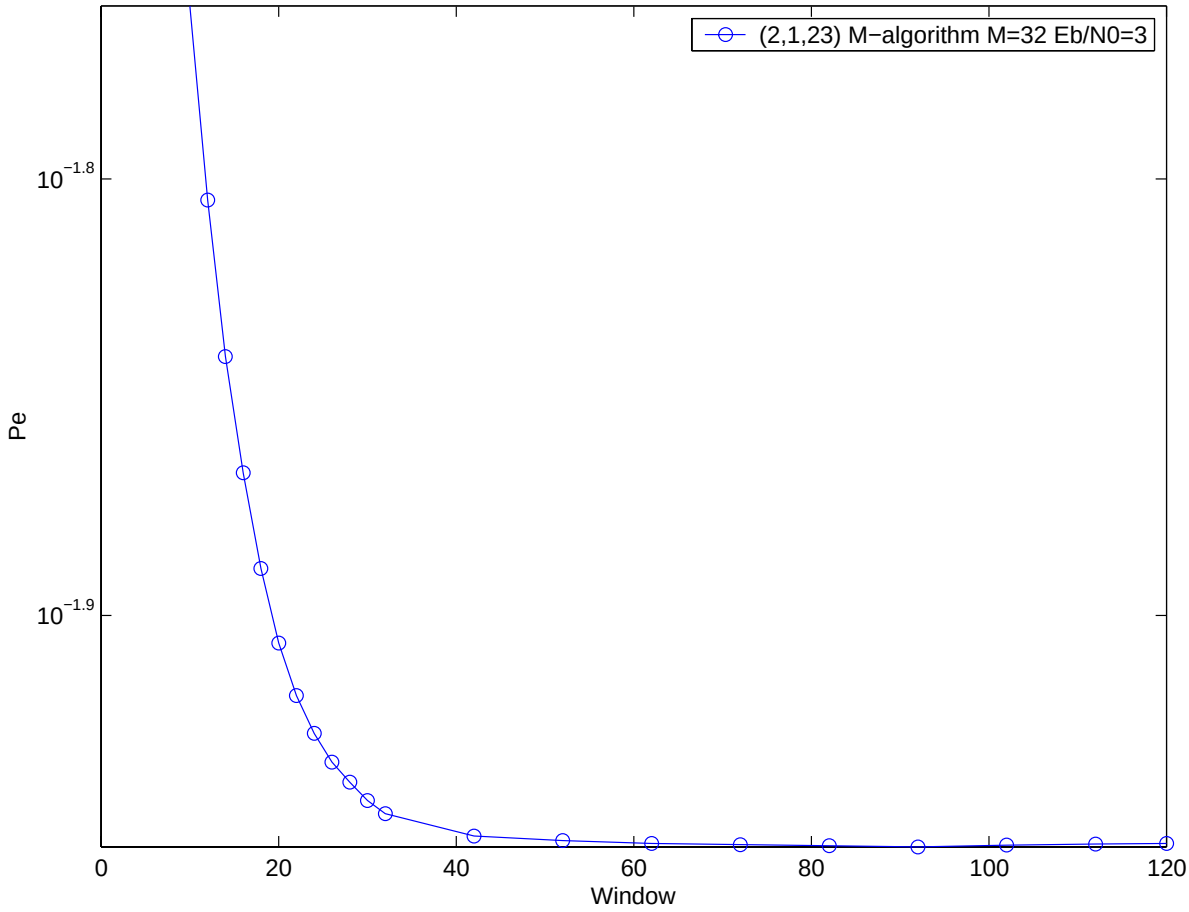

Figure 3.12: BERs obtained by the 32-state M-algorithm with various sliding windows (W = 10–120) for (2,1,23) convolutional code under AWGN channels with $E_b/N_0 = 3$ dB.

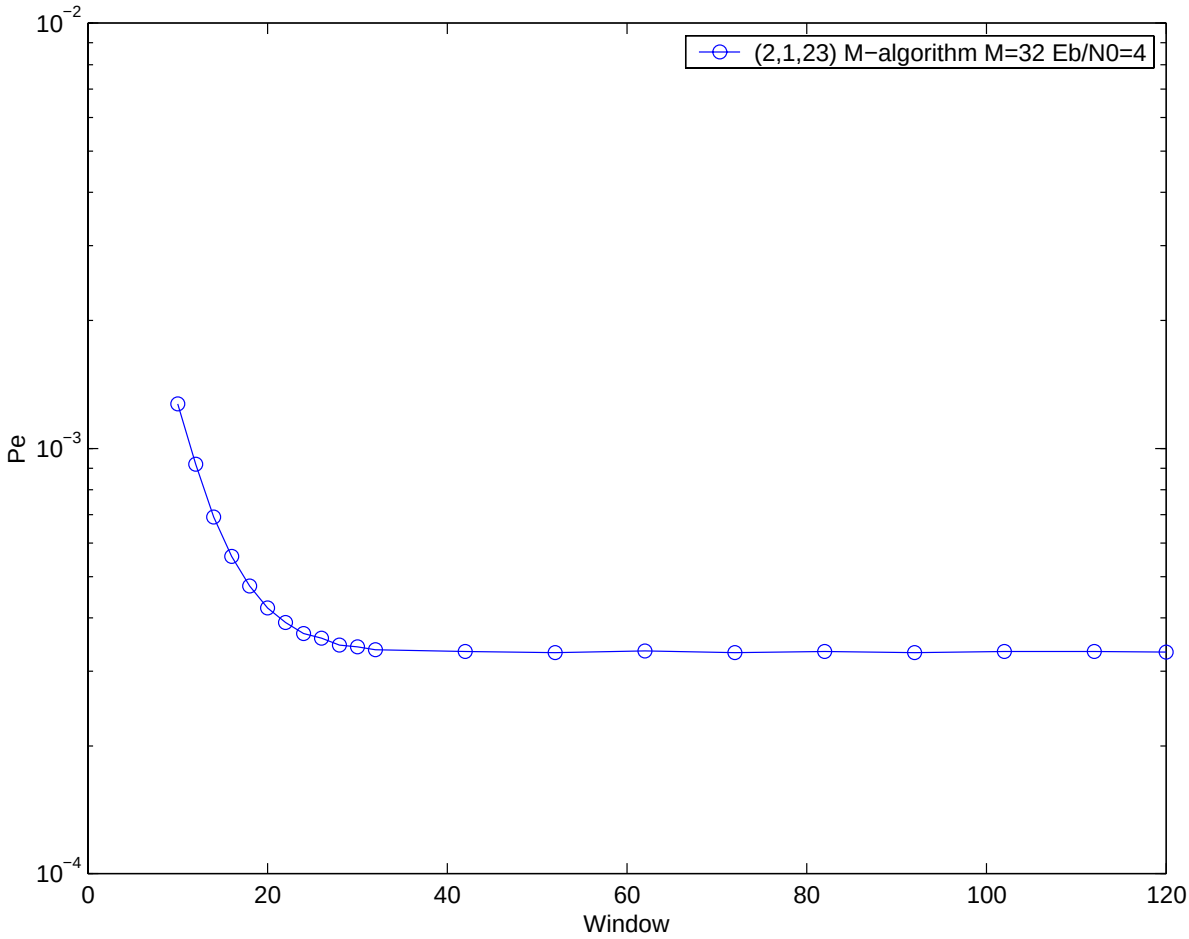

Figure 3.13: BERs obtained by the 32-state M-algorithm with various sliding windows (W = 10–120) for (2,1,23) convolutional code under AWGN channels with $E_b/N_0 = 4$ dB.

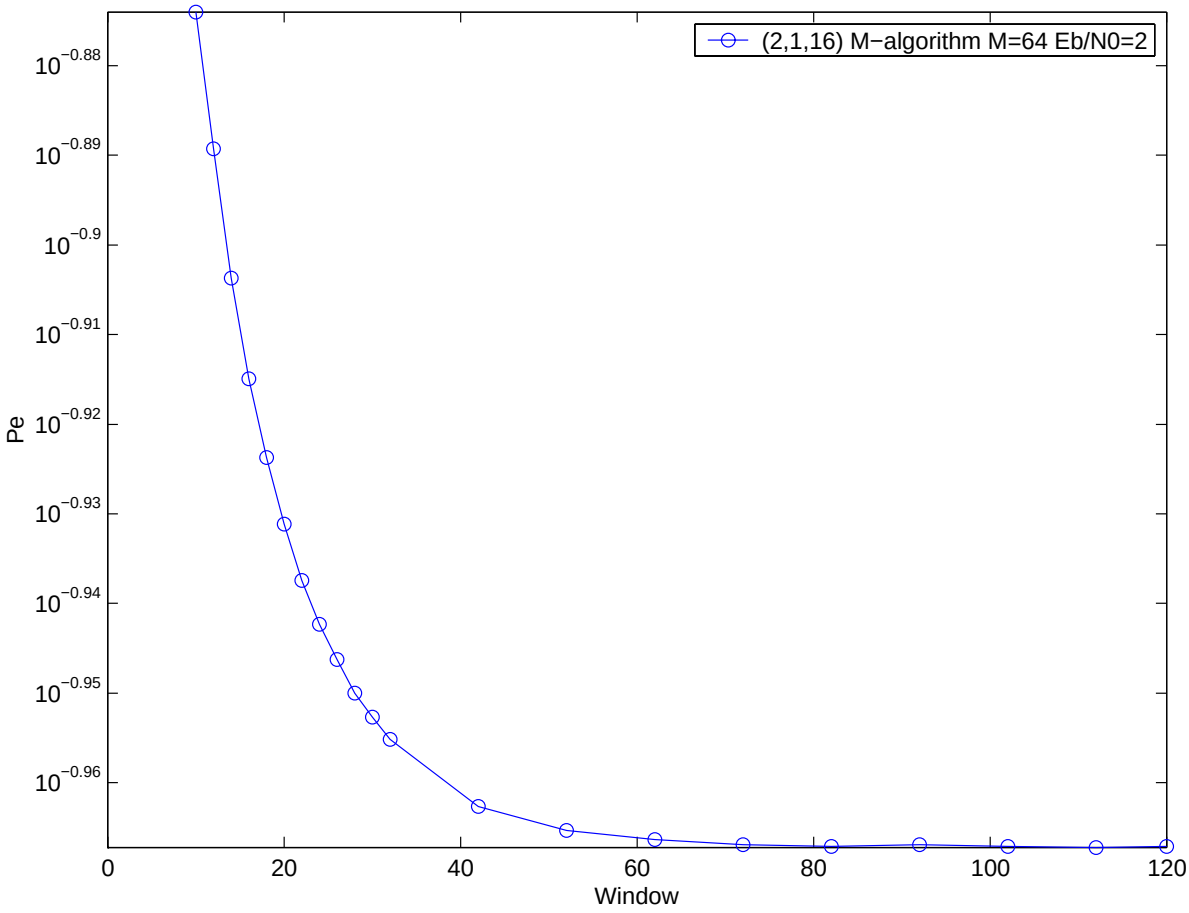

Figure 3.14: BERs obtained by the 64-state M-algorithm with various sliding windows (W = 10–120) for (2,1,16) convolutional code under AWGN channels with $E_b/N_0 = 2$ dB.

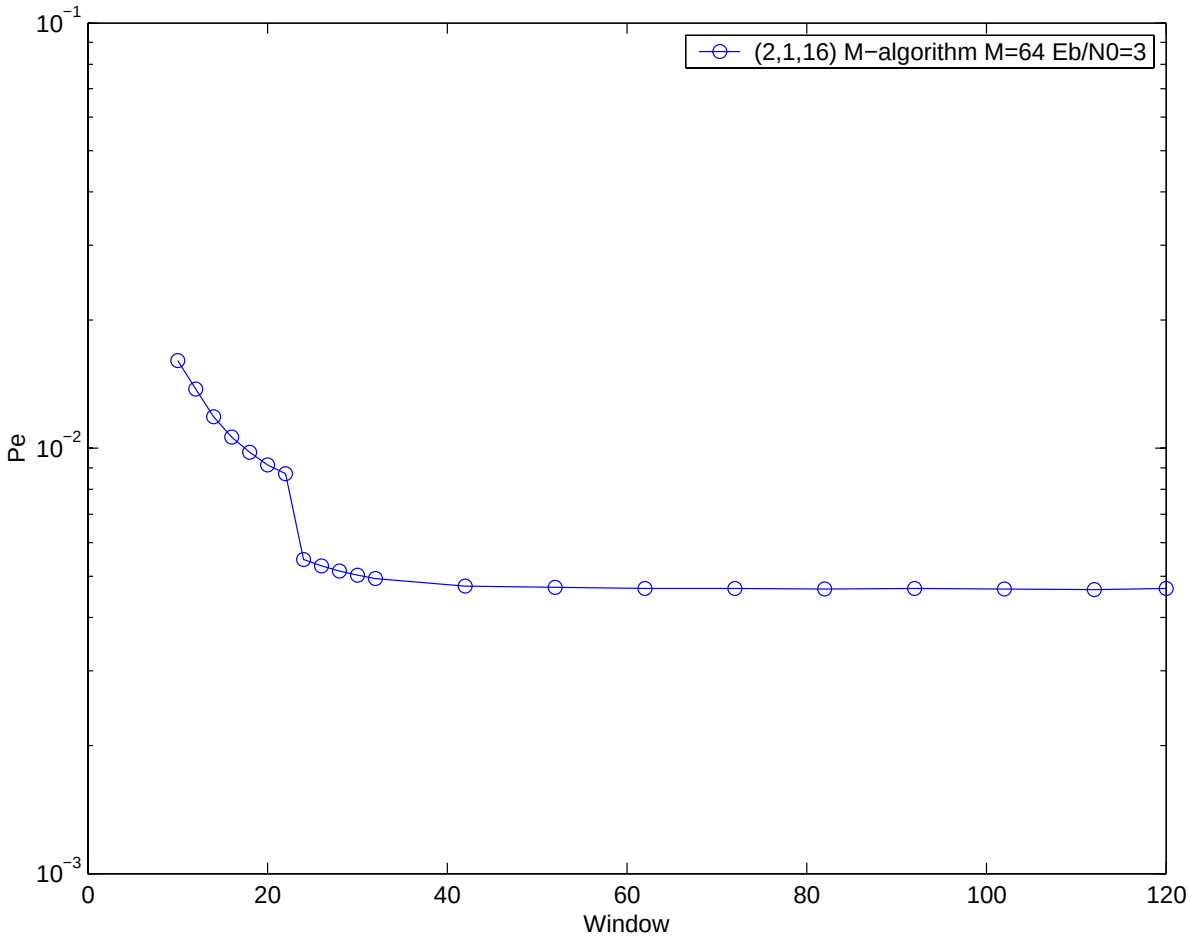

Figure 3.15: BERs obtained by the 64-state M-algorithm with various sliding windows (W = 10–120) for (2,1,16) convolutional code under AWGN channels with $E_b/N_0 = 3$ dB.

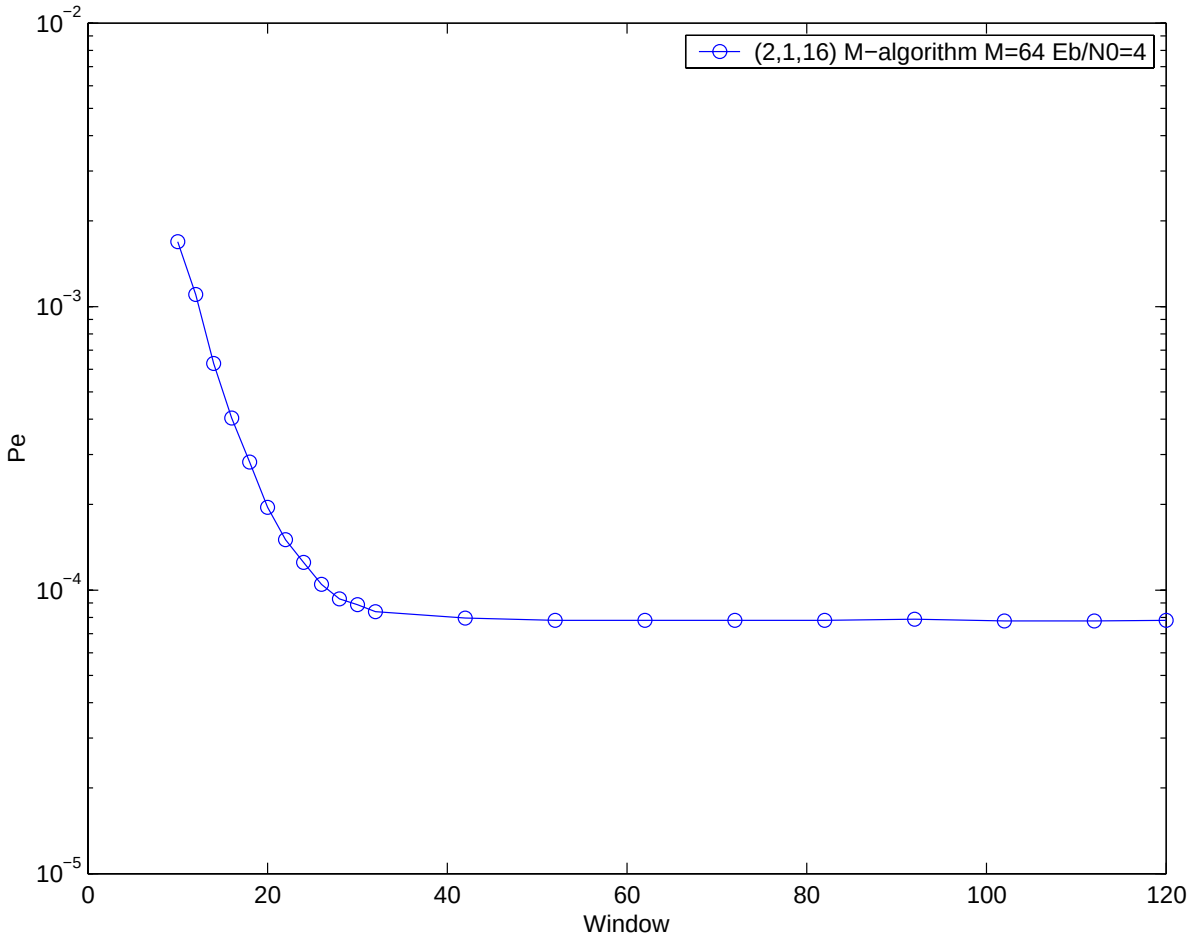

Figure 3.16: BERs obtained by the 64-state M-algorithm with various sliding windows (W = 10–120) for (2,1,16) convolutional code under AWGN channels with $E_b/N_0 = 4$ dB.

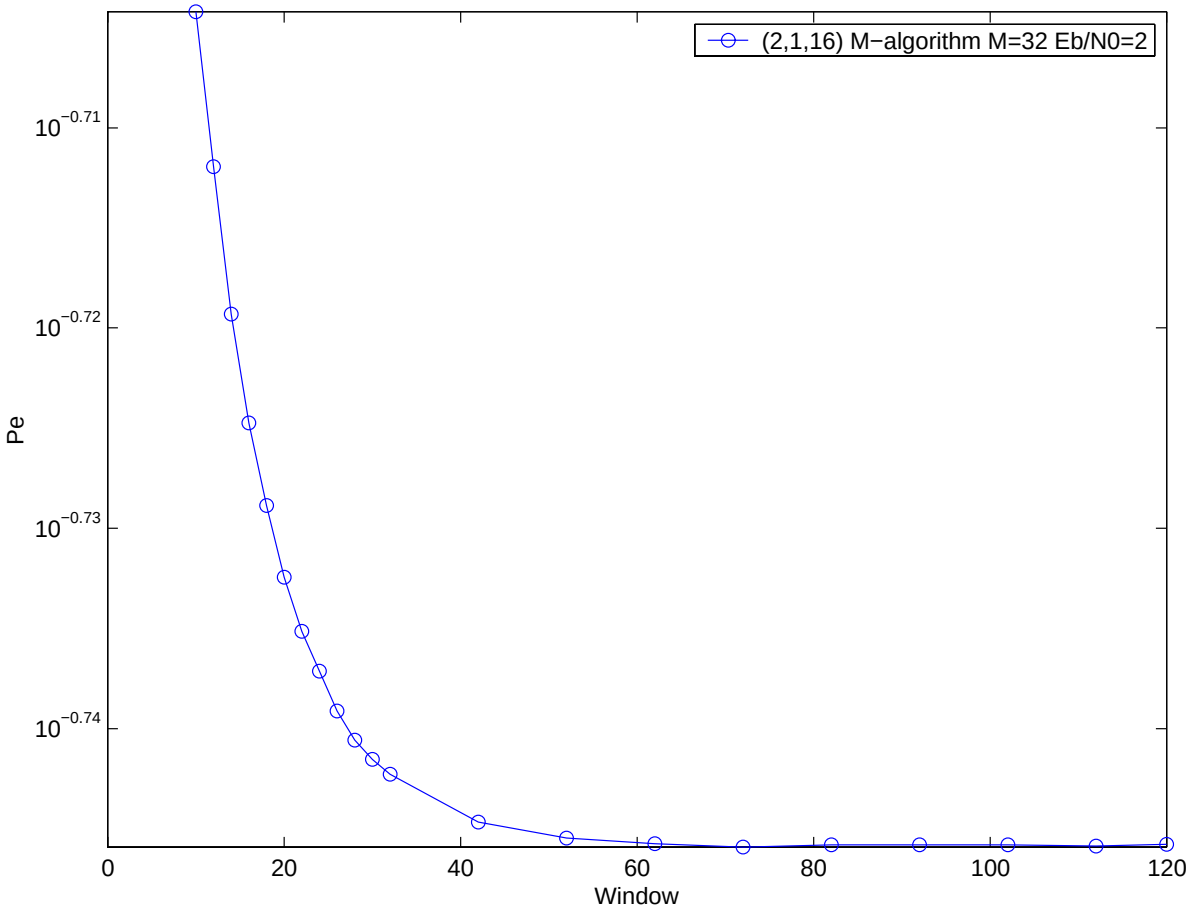

Figure 3.17: BERs obtained by the 32-state M-algorithm with various sliding windows (W = 10–120) for (2,1,16) convolutional code under AWGN channels with $E_b/N_0 = 2$ dB.

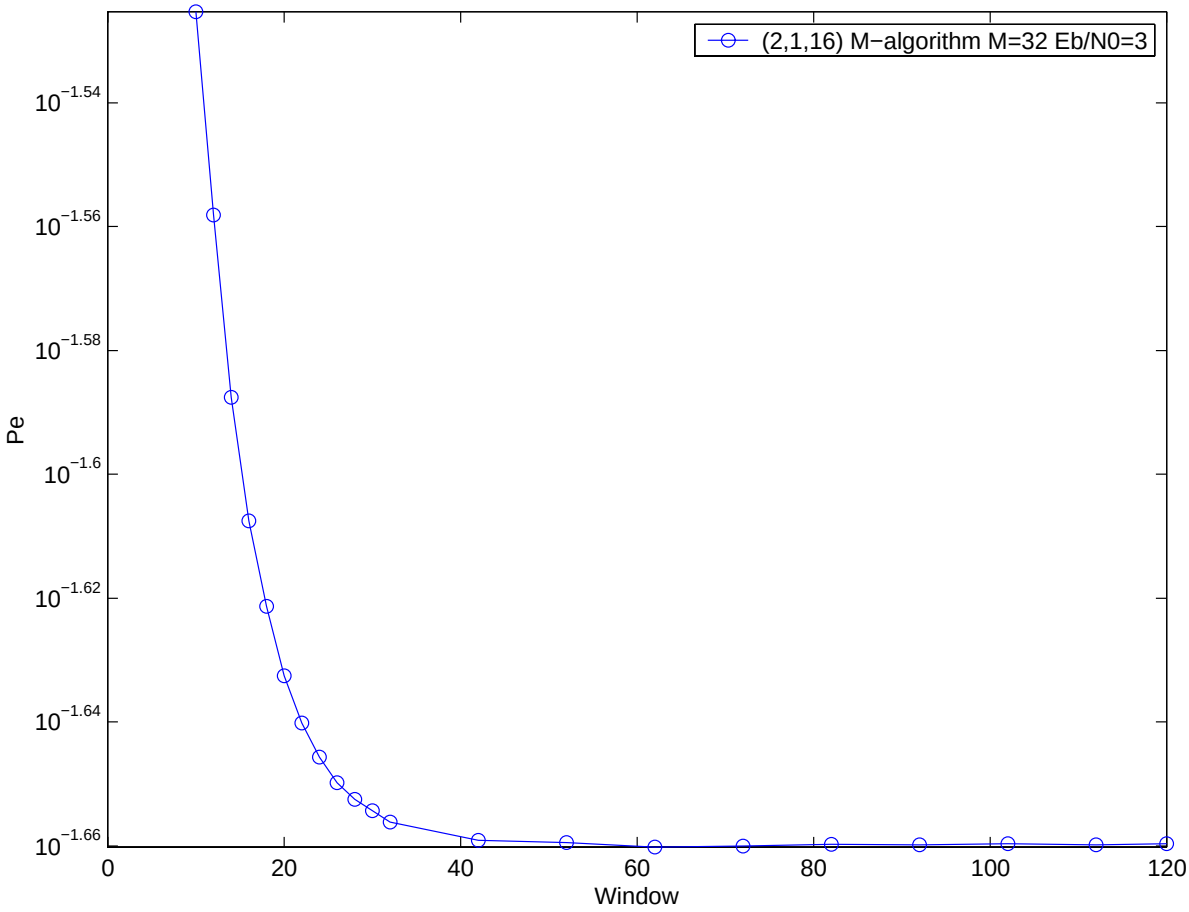

Figure 3.18: BERs obtained by the 32-state M-algorithm with various sliding windows (W = 10–120) for (2,1,16) convolutional code under AWGN channels with $E_b/N_0 = 3$ dB.

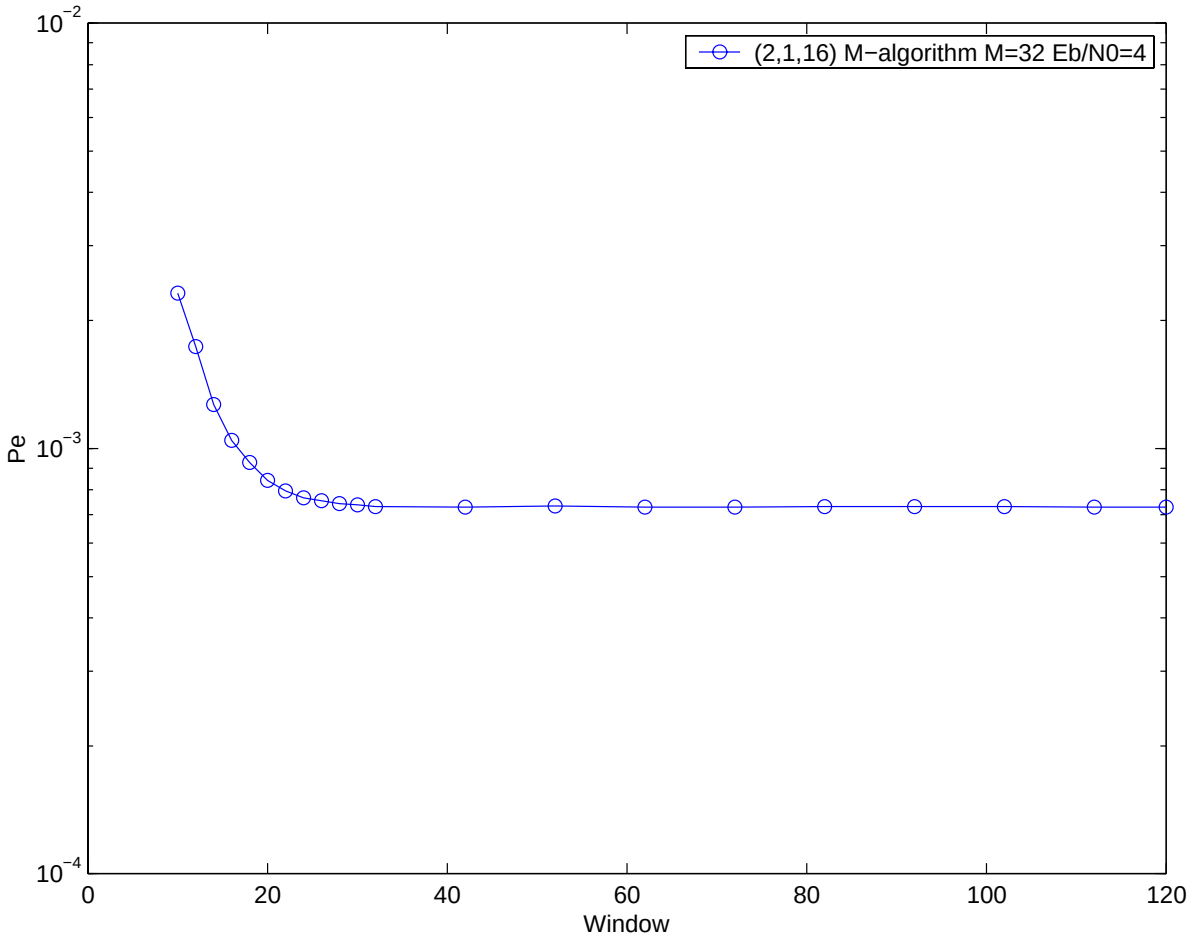

Figure 3.19: BERs obtained by the 32-state M-algorithm with various sliding windows (W = 10–120) for (2,1,16) convolutional code under AWGN channels with $E_b/N_0 = 4$ dB.

3.2.5 Performance impact of code constraint length

In the final subsection, we turn to the key considered factor in the thesis, i.e., the performance impact of code constraint length.

Figure 3.20 displays the BERs obtained by the state-reduction M-algorithm for (2,1,16) and (2,1,23) convolutional codes. The figure apparently indicates that employing codes with longer constraint length yields a better M-algorithm performance. An interesting future work here is to study the ultimate performance improvement by taking a very, very long constraint length. Although the decoding of such a code by the state-reduction M-algorithm is feasible, how to systematically construct convolutional codes with very long constraint length is perhaps another research subject. We therefore defer the work as a future research.

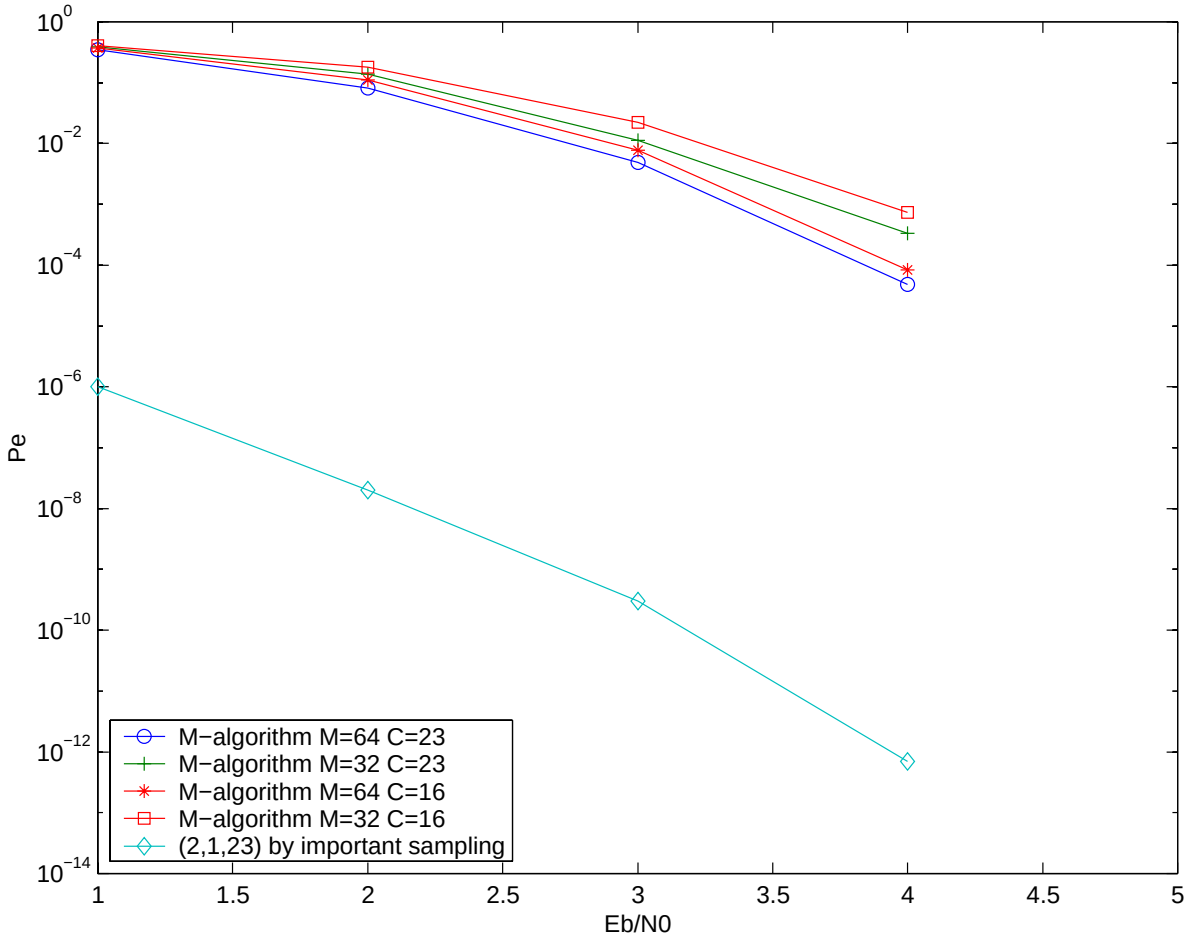

Figure 3.20: BERs obtained by the M-algorithm for (2,1,16) and (2,1,23) convolutional codes under AWGN channels.

Chapter 4

Conclusions

The code constraint length has been a key tradeoff between the maximum-likelihood performance and complexity of Viterbi decoding. Due to the high decoding complexity of Viterbi for codes with long constraint length, conventional Viterbi decoding can not benefit from the seemingly low bit error rates that exponentially decay with code constraint lengths. This motivates us to research on the reduction of the complexity of Viterbi decoding. The basic concept behind is that a proper complexity reduction may yield a suboptimal but still acceptable performance for codes with very long constraint length.

Some algorithms have been proposed for this purpose, such as those adaptive algorithms introduced in Chapter 2. These algorithms reduced the decoding complexity of Viterbi by adaptively adjust the number of preserved states according to the channel noise level. Although their simulation results only indicate a small performance degradation due to adaptive state reduction, dependence upon the chosen algorithmic parameters and noise level induces unexpected variation in system performance. Extra complexity may also be introduced in mechanism-related decision making. We therefore focus our research on the M-algorithm, whose complexity is fixed and independent of the noise level. Notably, the M-algorithm was originally simulated over BSC channels. In this thesis, we extend its usage to AWGN channels, and specifically for codes with long constraint lengths.

Our simulation results show that by taking a sufficiently long code constraint length, the M-algorithm can achieve the maximum-likelihood performance of Viterbi decoding of comparable decoding complexity, especially at high SNR. Simulation results also imply that the sliding window that must be considered in practical implementation of M-algorithm is sufficiently taken to be five times larger than the order of the maintained states (i.e., $\log_2(M)$). A future work of this thesis is to further extend the constraint length, and investigates the ultimate attainable performance of the M-algorithm.

Bibliography

[1] Anderson, John B. and Mohan, Seshadri, *Source and Channel Coding: An Algorithmic Approach*, Kluwer Academic Publishers: London, pp. 297–306, 1991.

[2] Balachandran, K., Anderson, J. B., Finamore, W. A. and Pimentel, C. J.-L., "Better rate 1/2 coded continuous phase modulation schemes and the M-algorithm," *IEEE Trans. Inform. Theory*, vol. 39, no. 5, pp. 1687–1694, Sept. 1993.

[3] Belzile, J. and Haccoun, D., "A Markovian evaluation of the frame error probability for the M Algorithm ," *Prof. of IEEE Internaltional Symposium on Information Theory*, pp. 267–267, 1993.

[4] Chan, F. and Haccoun, D., "Adaptive Viterbi decoding of convolutional codes over memoryless channels," *IEEE Trans. commun.*, vol. 45, no. 11, pp. 1389–1400, Nov. 1997.

[5] Elias, P., "Error-free coding," *IRE Trans. Inform. Theory.*, vol. IT-4, pp. 29–37, Sept. 1954.

[6] F.Jelinek, "Fast sequential decoding algorithm using a stack," *IBM J.Res.Develop.*, pp. 675–685, Nov. 1969.

[7] Feldmann, C. and Harris, J. H., "A constraint-length based modified Viterbi algorithm with adaptive effort," *IEEE Trans. Commun.*, vol. 47, no. 11, pp. 1611–1614, Nov. 1999.

[8] Finamore, W. A., Pimentel, C. J. L. and Anderson, J. B., "M-algorithm decoding of convolutional code combined with CPM," *Telecommunications Symposium, 1990. ITS '90 Symposium Record., SBT/IEEE International*, pp. 514–518, 1990.

[9] J. B. Anderson and S. Mohan, "Senquential coding algorithm:A survey and cost analsis," *IEEE Trans. Commun.*, vol. COM-32, no. 6, pp. 169–176, Feb 1984.

[10] Letaief, K. B., Muhammad, K, "An efficient new technique for accurate bit error probability estimation of ZJ decoders," *IEEE Trans. Commun.*, vol. 43, no. 6, pp. 2020–2027, June 1995.

[11] Lin, Shu and Costello, Daniel J., Jr., *Error Contol Coding: Fundamentals and Applications*, pp. 329–337, Prentice Hall: Englewood Cliffs, 1983.

[12] Motwani, Rajeev and Raghavan, Prabhakar, *Randomized Algorithm*, Cambridge University Press, pp. 3–7, 1995.

[13] Pottie, G. J. and Taylor, D. P., "A comparison of reduced complexity decoding algorithms for trellis codes ," *IEEE Jouranl on Selected Area in Communication*, vol. 7, no. 9, pp. 1369–1380, Dec. 1989.

[14] Sauer, W. and Rupprecht, W., "A suboptimum maximum likelihood detector for severely distorted data signals using a sorting breadth-first strategy," *IEEE International Symposium*, Circuits and Systems, vol. 1, pp. 127–130, 1998.

[15] Simmons, S. J., "Breadth-first trellis decoding with adaptive effort," *IEEE Trans. Commun.*, vol. 38, no. 1, pp. 3–12, Jan. 1990.