# REDUCED COMPLEXITY DECODING AND RELATIVE PERFORMANCE OF TURBO CODES

by

**CHI WA LEONG**

A Thesis Presented to the Faculty of Graduate Studies
in Partial Fulfilment of the Requirements
for the degree of

**MASTER OF SCIENCE**

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg Manitoba
R3T 2N2 Canada

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION PAGE

# REDUCED COMPLEXITY DECODING AND RELATIVE PERFORMANCE OF TURBO CODES

BY

CHI WA LEONG

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirements of the degree of

MASTER OF SCIENCE

# ABSTRACT

The thesis has two main topics, both related to turbo codes. The first one deals with the design of reduced complexity decoding of turbo codes. In particular, two new decoders are designed using ideas from the M-algorithm: one to decode the constituent code and the other the entire turbo code. Both decoders are found to be suitable only for short turbo codes with information block size less than 300 bits. For these turbo codes the two decoders have a lower complexity with a small degradation in error performance when compared to the original decoder used to decode the turbo codes. Therefore they are recommended in applications where short turbo codes are used and a small processing delay is required. The second topic of the thesis is whether the turbo code is superior to any other code. A partial answer is found by comparing short BCH codes to turbo codes with similar parameters. It is found that the BCH codes are more attractive than the turbo codes in both decoding complexity and error performance. This implies that traditional block codes are better than turbo codes with similar parameters.

# ACKNOWLEDGEMENT

I am particularly indebted to Dr. Edward Shwedyk for his endless guidance and encouragement throughout the course of my research work. In addition I would like to thank him for his excellent teaching in many of my undergraduate and graduate courses. Special thanks also go to Zhou, one of my colleagues, who provided suggestions on how to construct the code tree for turbo codes. I am also grateful to the Department of Electrical and Computer Engineering and the University of Manitoba for their financial supports and providing an enthusiastic environment to learn.

Last but not least, I would like to thank my family in Macau since without them I would not have been able to come to Canada to study in the first place.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF ABBREVIATIONS

| | |
|---|---|
| BCH | Bose, Chaudhuri, Hocquenghem |
| BCJR | Bahl, Cocke, Jelinek, Raviv |
| BER | bit error rate |
| DSP | digital signal processing |
| HDI | read-in-horizontally-and-read-out-diagonally interleaver |
| i.i.d. | independent and identically distributed |
| MAP | maximum *a posteriori* |
| MC | M-algorithm on constituent code |
| MW | M-algorithm on whole turbo code |
| PRI | pseudo-random interleaver |
| SISOM | soft-input-soft-output M-algorithm |
| VA | Viterbi algorithm |

# CHAPTER 1 INTRODUCTION

## 1.1 PROBLEM FORMULATION

In 1948 Shannon proved in his famous paper that coding can be used to increase the reliability of a transmission system. In the decades afterwards the development of coding has gone through many major steps, which include the invention of many new codes and the corresponding decoding algorithms. In general, codes can be classified into two categories: convolutional codes and block codes. Convolutional codes are suitable for, but not limited to, transmission systems where data are transmitted in packets of semi-infinite length. On the other hand, block codes are only suitable for systems where data are partitioned into packets of finite length. The most recent breakthrough in coding has been caused by the introduction of a new kind of block coding scheme called turbo code.

In 1993 Berrou, Glavieux and Thitimajshima introduced the turbo code and an iterative decoding algorithm [11]. The turbo code is classified as a block code because information symbols have to be partitioned into blocks of finite length before encoding can be done. The turbo code used in [11] consists of simple components but gives surprisingly good error performance when decoded by an iterative decoding algorithm. The core of the iterative decoding algorithm is the BCJR algorithm designed by Bahl, Cocke, Jelinek and Raviv in 1974 which is able to produce optimal[1] output symbol probabilities

1. In the thesis, "optimal" means "optimal in the maximum *a posteriori* sense".

[4]. The BCJR algorithm consists of one forward and one backward pass through the trellis and therefore requires a lot of computations. The iterative decoding algorithm is made up of several modules of the BCJR algorithm and iteration of the modules is organized in such a way that more iterations produce better results. However, iteration of the BCJR algorithm results in high computational complexity, making turbo codes not suitable for many practical applications. A lot of effort has been devoted to finding reduced complexity decoding algorithms for turbo codes; two important ones are described below.

**Soft-Output Viterbi Algorithm (SOVA)**: the Viterbi algorithm is an optimal decoding algorithm for convolutional codes and produces optimal sequence estimation in the form of hard outputs [21]. In [17] Hagenauer modified the VA so that it is able to produce output symbol probabilities in a suboptimal way but with total complexity less than that of the BCJR algorithm.

**Reduced-Search BCJR Algorithm**: Anderson in [2] proposed a method to simplify the BCJR algorithm. His method is to use the original BCJR algorithm except that only a certain number of branches are considered in the trellis. This results in fewer computations than the BCJR algorithm, which looks at all branches. Simulation results are not yet given but the computations are reduced.

The thesis will also be devoted to the subject of reduced complexity decoding of turbo codes. In particular, two new approaches are considered. The first one is similar to the reduced-search BCJR algorithm mentioned above but ideas from the M-algorithm are borrowed. The M-algorithm was originally a reduced complexity decoding

algorithm for convolutional codes [3]; with some modifications it can also be used to produce symbol probabilities. The second approach unifies the components of a turbo code into a single entity and a single module of the original M-algorithm is applied to decode the code. No iteration is necessary in this case and complexity is thus reduced. Comparisons between the first method and the original iterative decoding algorithm as well as between the second method and the original iterative decoding algorithm will be made in terms of the error performance and the computational complexity.

Further, the thesis also sheds some light on the question "If a traditional block code of the same length as the turbo code in [11] (i.e., 2x65536 bits) is used, will it out perform or perform as well as the turbo code?" The answer to this question is important because the decoding of traditional block codes can be done in a single stage rather than the repetitive steps of the iterative decoding algorithm, making the traditional block code an attractive alternative to turbo codes. However, a complete answer to this question is difficult because of the fact that a low complexity decoder for such long block codes is hard to construct. In the thesis a partial answer to this question is given by comparing a short block code to a turbo code with similar parameters. In particular, BCH codes are chosen because BCH codes of popular rate such as 1/2 or 1/3 exist. Moreover, the minimum distance and the number of minimum distance codewords are either known or can be approximated, making it easier if error bounds are to be derived. To decode the BCH codes, an algorithm that accepts soft inputs from the channel is chosen because it is potentially more powerful than a hard input algorithm [21]. In particular, the M-algorithm is used because of its availability from the first part of the thesis. From the final comparison between the BCH codes and the turbo codes in terms of their error performance and com-

plexity, some speculations and generalizations are made to answer the question.

## 1.2 OUTLINE OF THE THESIS

Chapter Two contains the background information necessary to understand the part of the thesis devoted to reduced complexity decoding of turbo codes. This material includes a description of the important components of a turbo code, the principles of the original iterative decoding algorithm used in [11] and that of the original M-algorithm. Chapter Three contains the details on the two approaches to reduced complexity decoding of turbo codes, their simulation results, computational complexity, their comparison to the original iterative decoding algorithm and a small summary on the investigation done in this Chapter. Chapter Four starts with the introduction of how the comparison of block codes and turbo codes will be made. Then it continues with how the comparisons can be made theoretically and experimentally. In particular, the experimental part contains simulation results and the computational complexity of both codes. A short summary on the investigation finishes Chapter Four. As the final chapter, Chapter Five contains the conclusions of the thesis.

There are in total five appendices. Appendix A contains the metric derivation for all the decoders in the thesis. Appendix B contains the definition of the signal-to-noise ratio used in the thesis. In Appendix C is shown how a generator polynomial in octal representation can be converted back to the polynomial form. Appendix D shows how soft decision decoding can be done on block codes when the M-algorithm is used. Lastly, Appendix E illustrates how computational complexity of decoding an extended block code can be obtained when the M-algorithm is used to decode the code.

# CHAPTER 2 BACKGROUND INFORMATION

## 2.1 THE ENCODER OF A TURBO CODE

Although the structure of a turbo code encoder can be generalized as in [14], in this thesis only the turbo code with two constituent codes is considered. Specifically, a turbo code where both constituent codes are identical recursive systematic convolutional codes, of rate 1/2, memory-4 with generator polynomials $G_1$=37 and $G_2$=21 is chosen for the study. The reason for selecting this code is that it has been demonstrated by many experimental results [11][26][30][32] to have a robust performance. The encoder of the code is shown in Figure 2.1.1.

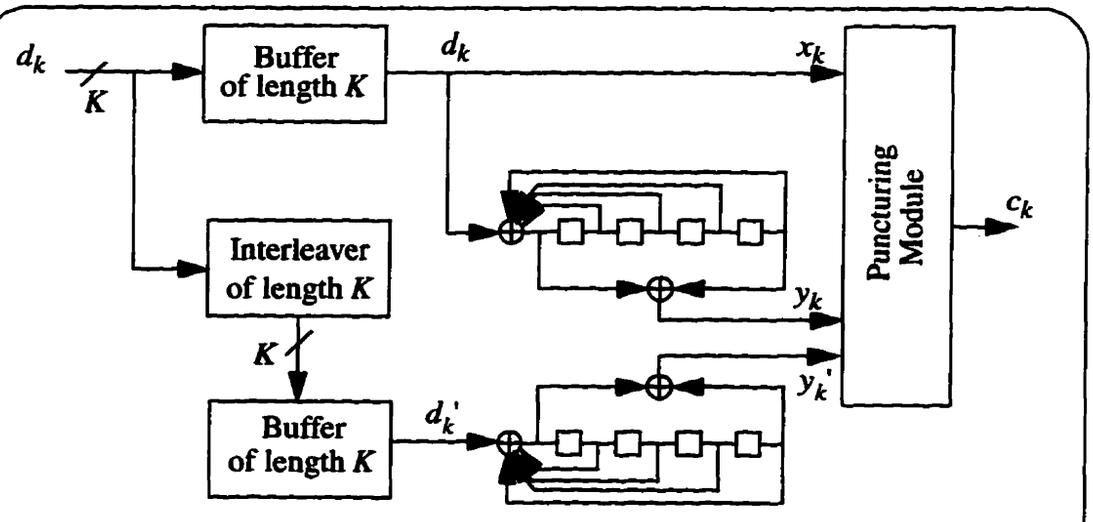


**Figure 2.1.1:** The structure of the encoder of the selected turbo code. Notice that $d_k$ represents the original information sequence, $d_k'$ represents the interleaved information sequence while $c_k$ represents the encoded sequence.

Besides the constituent encoders, the turbo code encoder consists of two buffers, one interleaver and one puncturing module. Two kinds of interleavers are considered: the pseudo-random interleaver and the read-in-horizontally-and-read-out-diagonally interleaver. The former is a very popular interleaver which has been proved both experimentally [1][11][14][26] and theoretically [9][10][26][32] to be a major component contributing to the excellent performance of turbo codes. The latter has a more regular interleaving pattern than the pseudo-random one and is much easier to implement in both software and hardware. Both interleavers are discussed in detail in Section 2.2. The puncturing module considered in the thesis is of either rate 1/2 or rate 1/3. These rates are chosen because many coding schemes in actual applications are of these rates. The puncturing module is discussed further in Section 2.3.

## 2.2 THE INTERLEAVER

The function of an interleaver is to reorder the position of each information bit into a new position. In fact the interleaver can be thought of as a mapping on the time index, $k$, of a block of information bits. If the mapping is denoted as $f(k)$ and the block of information bits to the input of the interleaver as $d_k$, $1 \le k \le K$, then the information bits at the output of the interleaver will be $d_{f(k)}$. Based on this notation the two interleavers of interest are discussed next.

The first one is the pseudo-random interleaver (PRI). For a pseudo-random interleaver $f(k)$ does not have a regular pattern. When a pseudo-random interleaver is designed for a turbo code encoder, $f(k)$ is determined beforehand in software before it is implemented in actual hardware. During encoding the mapping is thus fixed for all the

information bits. An algorithm for generating $f(k)$ for a pseudo-random interleaver of size $K$ is shown in Figure 2.2.1.

The second type of interleaver is the read-in-horizontally-and-read-out-diagonally interleaver (HDI). The mapping $f(k)$ of a HDI has a more regular pattern. Figure 2.2.2 illustrates the mapping of a HDI of length 9. Note that one requires the length of an HDI to be a complete square of an integer.

There are a lot of results published for different interleaver designs but they are not related to the thesis. Some of them can be found in [5], [6], [12] and [18].

```
(1)   VAR          f[1..K]: integer array of size K;
(2)                k, exch: integers

(3)   BUILT-IN FUNCTION
(4)                unran(seed): returns a uniformly distributed (real)
                                random variable between 0 and 1;
(5)                swap(a,b): interchange the values between integer
                              variable a and integer variable b;
(6)                ceil(r): return the smallest integer larger than or
                            equal to the real variable r;

(7)   MAIN         for (k=1; k<=K; k++)
(8)                   f[k]=k;
(9)                for (k=1; k<=K; k++) {
(10)                  exch=ceil(K*unran(-1758));
(11)                  swap(f[k], f[exch]);
(12)               }
(13)  END
```

**Figure 2.2.1:** The algorithm to generate the mapping $f(k)$ for a pseudo-random interleaver of length $K$. The random number generator in line (4) can be found in all popular software systems. The functions specified in line (5) and line (6) are directly available from the system or can be written by oneself. Also notice that the *seed* can be any value but in line (10) an arbitrarily negative integer value is chosen for it.

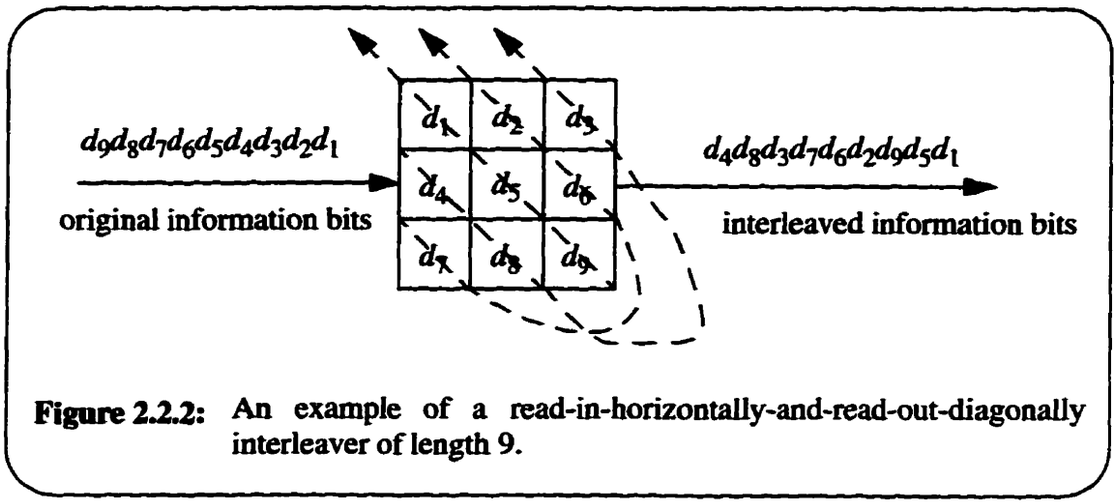


**Figure 2.2.2:** An example of a read-in-horizontally-and-read-out-diagonally interleaver of length 9.

## 2.3 THE PUNCTURING MODULE

The purpose of puncturing is to remove certain encoded bits so that the overall rate of the code can be increased to a specified value. The puncturing pattern can be represented by a matrix. For the turbo code in Figure 2.1.1 the first pattern of interest has the following matrix representation:

$$\begin{array}{c} \\ \text{odd time instant} \\ \text{even time instant} \end{array} \begin{array}{c} \begin{array}{ccc} x_k & y_k & y_k' \end{array} \\ \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \end{array} \qquad (2.3.1)$$

where a 1 indicates that the corresponding encoded bit is sent while a 0 indicates that the corresponding encoded bit is ignored. Also notice that the superscript ' denotes an output due to interleaving. As a result the overall rate of the code is 1/2.

The second pattern of interest has the following representation:

$$\begin{matrix} & x_k & y_k & y_k' \\ \text{odd time instant} & 1 & 1 & 1 \\ \text{even time instant} & 1 & 1 & 1 \end{matrix} \cdot \qquad (2.3.2)$$

Obviously the overall rate of the code is 1/3. In fact, this is equivalent to no puncturing at all. To be consistent with the previous case a puncturing module will still be considered to be present in such a case.

The interleaver and the puncturing module are important elements of a turbo code. When they are combined with the constituent encoders, these elements give each turbo code distinctive characteristics. Sometimes these characteristics can be used as a measure of comparison among the turbo codes. One characteristic that can be used in this way is the effective free distance of the turbo code. This is the only one that is of interest in the thesis and will be discussed in the next section. Discussions on other characteristics of a turbo code can be found in [7], [13], [28] and [34].

## 2.4 THE EFFECTIVE FREE DISTANCE

The effective free distance ($d_{free}^{eff}$)of a turbo code is the minimum Hamming weight of the codeword caused by a weight 2 information sequence. It plays a role similar to that of the free distance for a convolutional code [10][15] and therefore it is an important parameter of the turbo code. The effective free distance is always larger than or equal to the free distance. If $d_{free}$ is the free distance of a turbo code and $t$ is the correction capability of the code, then

$$d_{free}^{eff} \geq d_{free} = 2t + 1. \qquad (2.4.1)$$

As a matter of fact for many turbo codes the two distances are equal to each other. An upper bound for $d_{free}^{eff}$ can be found in [15] but when an exact value is needed, one must resort to a computer search. The effective free distance is particularly useful when the M-algorithm is used to decode the entire turbo code. In Section 3.7 a computer program is written to find out the effective free distance of some of the turbo codes.

Before the reduced complexity decoding can be discussed it is necessary to understand how the original algorithm works. The original iterative decoding algorithm used in [11] is the next topic of discussion.

## 2.5 THE ORIGINAL ITERATIVE DECODING ALGORITHM FOR TURBO CODES

The original iterative decoding algorithm is the first decoding algorithm designed for turbo codes and is still the one that achieves the best results in terms of BER [30]. In fact, Moher [25] has argued with the ideas of cross-entropy minimization that this iterative decoding algorithm is optimal in the maximum *a posteriori* sense as the number of iterations approaches infinity. In many practical situations, very close to optimal results can be achieved only after a finite number of iterations. McEliece in [39] argued that the iterative algorithm used in [11] is not optimal by giving counter examples but he also pointed out that the probability of such examples occurring is very small. Because of this, the decoder with the original iterative decoding algorithm is called the MAP decoder in the thesis.

The core of the MAP decoder is the BCJR algorithm. Consider the turbo code in Figure 2.1.1 with an overall code rate equal to 1/2. Since there are two constituent

codes in the encoder, there are two modules of the BCJR algorithm in the MAP decoder.

Assume antipodal signalling ($0 \rightarrow -1$ and $1 \rightarrow 1$) is used and the channel is an AWGN channel. Then the received sampled values $X_k$, $Y_k$ and $Y_k'$, $1 \le k \le N$ from the channel can be expressed as:

$$X_k = (2x_k - 1) + n_{1k},$$

$$Y_k = (2y_k - 1) + n_{2k},$$

$$Y_k' = (2y_k' - 1) + n_{3k} \qquad (2.5.1)$$

where $n_{1k}$, $n_{2k}$ and $n_{3k}$ are independent zero mean Gaussian random variables of variance $\sigma^2$. Also assume that the discrete random variables $d_k$ and $d_k'$ in Figure 2.1.1 are either 0 or 1 with equal probability and that they are independent and identically distributed (i.i.d.) for each time instant $k$. The details of the decoder are shown in Figure 2.5.1.

Notice that the input sequence $X_k'$ to the second decoder is set to 0 (an erasure for antipodal signalling with signal points at $\pm 1$). This is because only the parity bits of the second constituent code are transmitted. Also every other position of the received sequence $Y_k$ and $Y_k'$ is set to 0 because they are punctured. In general, for any output bits of the encoder not transmitted at a particular time instant, the corresponding channel observation inputs to the decoder are set to 0 (an erasure) at that time instant.

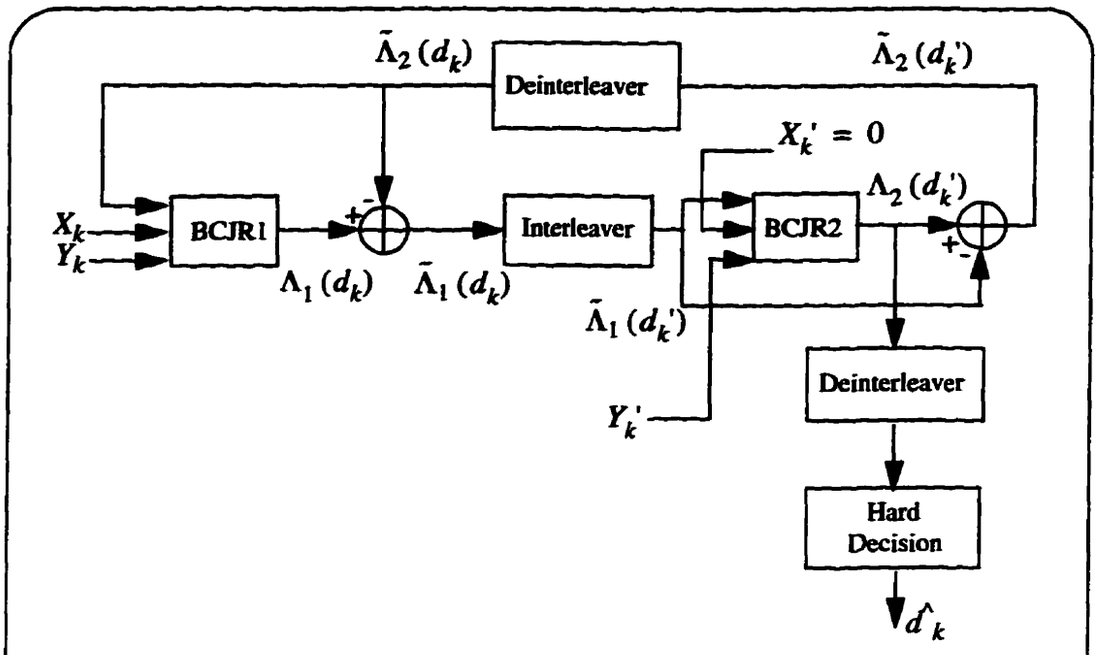


**Figure 2.5.1:** A detailed depiction of the MAP decoder.

In order to apply the BCJR algorithm to decode each constituent code, it is necessary to construct the trellis of the constituent codes. Because both constituent codes are identical, their trellises are also the same. Further, except for branch labelling the trellis structure of a recursive systematic convolutional code is the same as that of a nonrecursive convolutional code of the same memory order [11]. Since the trellis structure of a nonrecursive convolutional code is well known, the trellis of the constituent code in consideration can be obtained easily.

Based on the received sampled values from the channel the algorithm BCJR1 calculates the log likelihood ratios for the first code:

$$\Lambda_1(d_k) = Log \frac{Pr\{d_k = 1 \mid X_k, Y_k\}}{Pr\{d_k = 0 \mid X_k, Y_k\}}$$

$$= Log \frac{\sum_m \lambda^1{}_k(m)}{\sum_m \lambda^0{}_k(m)} \tag{2.5.2}$$

where $m$ represents the state of the trellis and

$$\lambda^i{}_k(m) = \alpha^i{}_k(m)\,\beta_k(m)\,; \tag{2.5.3}$$

$$\alpha^i{}_k(m) = \frac{\sum_{m'}\sum_{j=0}^{1} \gamma_i(R_k, m', m)\,\alpha^j{}_{k-1}(m')}{\sum_{m''}\sum_{m'}\sum_{i'=0}^{1}\sum_{j=0}^{1} \gamma_{i'}(R_k, m', m'')\,\alpha^j{}_{k-1}(m')} \tag{2.5.4}$$

with initial conditions $\alpha^i{}_0(0) = 1$, $\alpha^i{}_0(m) = 0$ for all $m \neq 0$, $i = 0,1$;

$$\beta_k(m) = \frac{\sum_{m'}\sum_{i=0}^{1} \gamma_i(R_{k+1}, m, m')\,\beta_{k+1}(m')}{\sum_{m''}\sum_{m'}\sum_{i=0}^{1}\sum_{j=0}^{1} \gamma_i(R_{k+1}, m', m'')\,\alpha^j{}_k(m')} \tag{2.5.5}$$

with initial conditions $\beta_N(0) = 1$, $\beta_N(m) = 0$ for all $m \neq 0$

and finally

$$\gamma_i(R_k, m', m) = p(R_k | d_k = i, S_k = m, S_{k-1} = m') \cdot q(d_k = i | S_k = m, S_{k-1} = m') \cdot$$

$$\pi(S_k = m | S_{k-1} = m') \tag{2.5.6}$$

where $R_k = (X_k, Y_k)$.

The $\alpha^i{}_k(m)$ 's and $\gamma_i(R_k, m', m)$ 's can be calculated in a forward recursion through the trellis of the constituent code. After they are completed the $\beta_k(m)$ 's can then be calculated by a backward recursion through the trellis.

For each branch in the trellis $p(.|.)$ is the transition probability of the discrete Gaussian memoryless channel. Conditioned on $(d_k = i, S_k = m, S_{k-1} = m')$, the samples $X_k$ and $Y_k$ are two uncorrelated Gaussian random variables and therefore $p(.|.)$ can be split into two terms:

$$p(R_k|d_k = i, S_k = m, S_{k-1} = m') = p(X_k|d_k = i, S_k = m, S_{k-1} = m') \cdot p(Y_k|d_k = i, S_k = m, S_{k-1} = m') \qquad (2.5.7)$$

$$= \frac{1}{\sqrt{2\pi}\sigma}\exp\left[\frac{(X_k - \pm 1)^2}{2\sigma^2}\right] \cdot \frac{1}{\sqrt{2\pi}\sigma}\exp\left[\frac{(Y_k - \pm 1)^2}{2\sigma^2}\right]. \qquad (2.5.8)$$

Because there are no parallel branches in the trellis, $q(d_k = i|S_k = m, S_{k-1} = m')$ is equal to 1 for each branch. The term $\pi(.|.)$ depends on the *a posteriori* information produced by the second decoder. For the first iteration, there is no *a posteriori* information from the second decoder and therefore $\pi(S_k = m|S_{k-1} = m') = 1/2$ for each trellis branch. For all subsequent iterations, the first encoder can utilize the a posteriori information $\tilde{\Lambda}_2(d_k)$ provided by the second decoder as the *a priori* probability of $d_k$. Thus for a subsequent iteration,

$$\pi(S_k = m|S_{k-1} = m') = Pr\{d_k = i|\text{decoder2}\}. \qquad (2.5.9)$$

If the transition from $m'$ to $m$ at time $k$ corresponds to $d_k$=1, then (2.5.9) equals

$$= \frac{\exp\left[\tilde{\Lambda}_2(d_k)\right]}{1 + \exp\left[\tilde{\Lambda}_2(d_k)\right]}; \qquad (2.5.10)$$

otherwise, if the transition from $m'$ to $m$ at time $k$ corresponds to $d_k$=0, (2.5.9) equals

$$= 1 - \frac{\exp\left[\tilde{\Lambda}_2(d_k)\right]}{1 + \exp\left[\tilde{\Lambda}_2(d_k)\right]}. \qquad (2.5.11)$$

After the likelihood ratios for all the $d_k$'s are calculated, they can be passed to the second constituent decoder which can utilize this information as the *a priori* probabilities. Before doing so it is important to make sure that whatever is being passed to the second decoder must be independent of any other information previously generated by itself; otherwise a positive feedback loop may exist. Thus it is necessary to subtract $\tilde{\Lambda}_2(d_k)$ from $\Lambda_1(d_k)$ and the result can then be passed to the second decoder. An interleaver, which has the same interleaving pattern as the one used in the encoder, is inserted in between the adder and the second decoder to make sure that the likelihood ratios are in the correct order before they can be used by the second decoder.

For the second decoder (2.5.2) to (2.5.11) can still be used to calculate the likelihood ratios $\Lambda_2(d_k)$ as long as the following changes are kept in mind:

1) $d_k$ is changed to $d_k'$ in all equations;

2) if the second constituent encoder is not terminated, it will be necessary to modify the initial conditions for $\beta$ in (2.5.5) to:

$$\beta_N = \alpha_N^{\,1}(m) + \alpha_N^{\,0}(m) \quad \text{for all } m;$$

3) $Pr\{d_k = i | \text{decoder2}\}$ is changed to $Pr\{d_k' = i | \text{decoder1}\}$ in (2.5.9);

4) $\tilde{\Lambda}_2(d_k)$ is changed to $\tilde{\Lambda}_1(d_k')$ in (2.5.10) and (2.5.11).

After all the likelihood ratios $\Lambda_2(d_k')$ have been calculated, they can be passed to the first decoder in a way similar to the one described above in order to start the next iteration.[1] The situation is clearly shown in Figure 2.5.1.

When enough iterations have been done, likelihood ratios $\Lambda_2(d_k')$ can be passed to a deinterleaver to get $\Lambda_2(d_k)$ and then a hard decision can be made in the following way:

$$\hat{d}_k = 1 \text{ if } \Lambda_2(d_k) > 0$$

$$\hat{d}_k = 0 \text{ if } \Lambda_2(d_k) < 0.$$

After that the final decision for each symbol is obtained.

After the discussion of the MAP decoder, it is necessary to discuss the M-algorithm from which ideas will be borrowed to design the new decoders. Therefore the M-algorithm is discussed next.

## 2.6 THE M-ALGORITHM

The M-algorithm was originally proposed in [3] as a reduced complexity decoding algorithm for a trellis code or a tree code. It accepts soft inputs from the channel and produces hard estimates (0 or 1) on a coded sequence. The algorithm is classified as a sequential decoding algorithm; however, in contrast to the stack algorithm, the M-algorithm is a breadth-first search algorithm which does not allow any back-tracking from the

1. One iteration of the MAP decoder equals one pass through the module BCJR1 plus one pass through the module BCJR2.

present level of the tree to a previous level. A practical sliding window version of the M-algorithm [3] is shown in Figure 2.6.1.

Repeat **Step 1-3** for each tree level:

| | |
|---|---|
| **Step 1** | (Path Extension) Extend all stored paths to the next depth, creating new paths from each stored path; |
| **Step 2** | (Dropping) Drop all but the $M$ best paths (in metric sense); |
| **Step 3** | (Branch release) If the paths have reached the decision depth, release as output the first branch of the best path. |

**Figure 2.6.1:** The sliding window version of the M-algorithm.

Although the algorithm does not look at every possible node in a code tree, results close to maximum likelihood sequence estimation can still be achieved when the storage size ($M$) and the decision depth ($L_D$) is large enough. For any $t$-error correcting code with $\log_2 b$ input bits and $a$ output bits, the minimum value of $M$ and $L_D$ can be found by using a random coding argument [3]. Table 2.6.1 lists the minimum value of $M$ and $L_D$ for some commonly used convolutional codes.

As is pointed out in [3], the storage requirement of the M-algorithm is significantly less than that of the Viterbi algorithm (VA) while the degradation in error performance of the M-algorithm is negligible compared to maximum likelihood sequence estimation using VA. The next Chapter demonstrates how the M-algorithm can be used to decode turbo codes.

**TABLE 2.6.1:** Theoretical decision depth and storage size for $t$-error correcting codes of rate $R$. The code rate $R$ equals $(\log_2 b)/a$.

| Rate $R$ | $(a, b)$ | $L_D$ | min $M$= $(2^{1-R}-1)^{-t}$ |
|---|---|---|---|
| 1/4 | 4,2 | $2.33t$ | $(1.47)^t$ |
| 1/3 | 3,2 | $3.83t$ | $(1.70)^t$ |
| 1/2 | 2,2 | $9.09t$ | $(2.41)^t$ |
| 2/3 | 3,4 | $10.8t$ | $(3.85)^t$ |
| 3/4 | 4,8 | $12.0t$ | $(5.89)^t$ |
| 7/8 | 8,128 | $14.6t$ | $(11.05)^t$ |

# CHAPTER 3 REDUCED COMPLEXITY DECODING OF TURBO CODES

## 3.1 INTRODUCTION

Reduced complexity decoding can be designed either for the constituent code individually or for the overall turbo code. In this chapter the M-algorithm is used to design reduced complexity decoders for both cases.

## 3.2 REDUCED COMPLEXITY DECODING FOR CONSTITUENT CODES: DECODER STRUCTURE

In order to apply reduced complexity decoding to constituent codes it is necessary to design an algorithm which accepts soft inputs and produces soft outputs on the information bits so that the algorithm can replace the BCJR algorithm. One way is to modify the original M-algorithm in Figure 2.6.1 so that at the end the algorithm will output the $M$ best paths in storage together with the final path metrics. Then the log-likelihood ratios of the information bits can be approximated from the $M$ path metrics. For it to accept soft inputs it is also necessary to modify the branch metric calculation in the original M-algorithm so that the *a priori* information is taken into account. Because intermediate results are not needed, the sliding window of the original algorithm can be eliminated and all the results will be obtained at the end of the algorithm. However, the requirement that $M$ needs to be larger than the minimum value specified in Table 2.6.1, must still be followed. One should also keep in mind that the $M$ paths produced by the M-algorithm are

not necessarily the $M$ best paths of the constituent code because not all possible paths through the tree have been looked at. In the case that $M$ is sufficiently large, most of the $M$ paths produced by the M-algorithm should coincide with the $M$ best paths of the constituent code. Depending on the value of M, the computational complexity of this algorithm can be more or less than that of the BCJR algorithm. Discussions about the computational complexity of the algorithm are postponed until Section 3.4. To make it easier to refer to the new algorithm it will be called the soft-input-soft-output M-algorithm (SISOM). The complete decoder with two modules of the SISOM algorithm will be called the MC (M-algorithm on constituent code) decoder. For the turbo code in Figure 2.1.1 with an overall code rate=1/2 the corresponding MC decoder is shown in Figure 3.2.1.

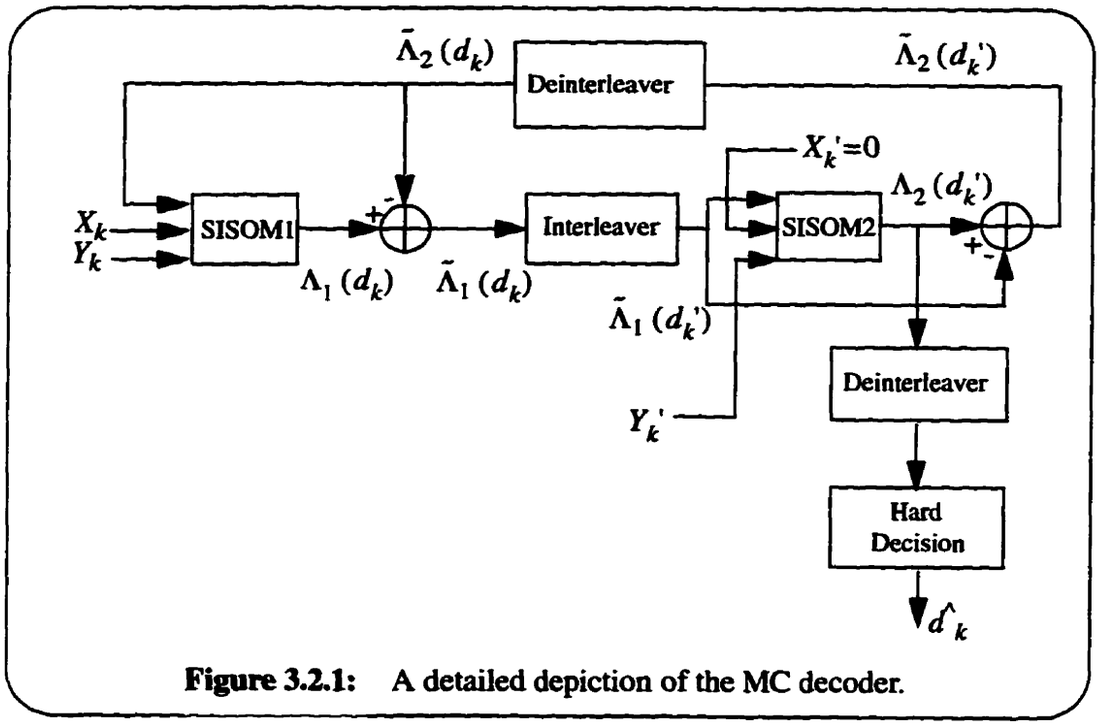


**Figure 3.2.1:** A detailed depiction of the MC decoder.

In order to make use of the *a priori* information available at the input of the constituent decoder it is necessary to modify the branch metric calculation for the SISOM

algorithm. In the following the new metric (derived in Section A.1 of Appendix A) for the first SISOM module is given; the one for the second module is straightforward. It is assumed that antipodal signalling of $\pm 1$ is used, that the encoder is the one in Figure 2.1.1 and that the decoder is the one in Figure 3.2.1. In the log domain, the new branch metric for path $i$ at time instant $k$ is:

$$BM_{i,k} = \log P\left( d_k \middle| \tilde{\Lambda}_2(d_k) \right) - \left( \frac{(X_k - \pm 1)^2}{2\sigma^2} + \frac{(Y_k - \pm 1)^2}{2\sigma^2} \right) \qquad (3.2.1)$$

where $\pm 1$ are branch labels of the tree, $\sigma^2$ is the noise variance and $\tilde{\Lambda}_2(d_k)$ is the *a priori* information of the information bit $d_k$ provided by the previous stage of decoding. $\tilde{\Lambda}_2(d_k)$ is also called the extrinsic information and is usually in the form of a log-likelihood ratio. In that case,

$$P\left( d_k = 1 \middle| \tilde{\Lambda}_2(d_k) \right) = \frac{\tilde{\Lambda}_2(d_k)}{1 + \tilde{\Lambda}_2(d_k)} \text{ and} \qquad (3.2.2)$$

$$P\left( d_k = 0 \middle| \tilde{\Lambda}_2(d_k) \right) = \frac{1}{1 + \tilde{\Lambda}_2(d_k)}. \qquad (3.2.3)$$

The total path metric for path $i$ can then be calculated as

$$PM_i = \sum_{k=1}^{K} BM_{i,k}. \qquad (3.2.4)$$

After the pass through the SISOM algorithm is completed and the $M$ paths with their path metrics in the log domain are ready, the metrics can be converted to the *a posteriori* probabilities and then the log-likelihood ratio $\Lambda_1$ of the corresponding information bit $d_k$ can be calculated. Let the set of information sequence associated with the $M$

best paths in storage be $\{\hat{d}_i,\ 1 \le i \le M\}$. Then the *a posteriori* log-likelihood ratio of the bit $d_k$ can be calculated as follows:

$$\Lambda_1(d_k) = \log \frac{\sum_{\hat{d}_i : d_{ik} = 1} \exp(PM_i + C)}{\sum_{\hat{d}_i : d_{ik} = 0} \exp(PM_i + C)} \tag{3.2.5}$$

where $C = K \cdot \log \frac{1}{2\pi\sigma^2}$ is a constant.

After $\Lambda_1(d_k)$ have been calculated for all $k$, $\tilde{\Lambda}_2(d_k)$ must be subtracted from $\Lambda_1(d_k)$ to obtain $\tilde{\Lambda}_1(d_k)$ and the results are passed to the interleaver and then to the second decoder as shown in Figure 3.2.1. Another SISOM algorithm is then applied to the second constituent code and the whole procedure can be iterated in the same way that the BCJR algorithm in the MAP decoder is iterated. To be consistent with the MAP decoder, it is assumed that one iteration of the MC decoder equals one pass through the module SISOM1 plus one pass through the module SISOM2. The performance of the MC decoder will be demonstrated by computer simulations in the next section.

## 3.3 MC DECODER: SIMULATION RESULTS

The first turbo code chosen for simulation is a short length turbo code. Longer codes will be considered later. Consider the turbo code in Figure 2.1.1 with parameters $K$=64, interleaver=PRI, overall code rate=1/2 and both constituent encoders left open. The number of iterations of the MC decoder is set to 10 and is then run on the turbo

code with various values of $M$. In particular $M$ is varied over 2, 16 and 64. The results are shown in Figure 3.3.1. The result of the MAP decoder with 10 iterations is also shown in the figure. The signal-to-noise ratio (SNR) is defined as $E_b/N_0$, whose full definition is given in Appendix B. It can be noticed from the results that the error performance of the MAP decoder is better than that of the MC decoder but the performance of the MC decoder improves with increasing value of $M$. In particular for $M$=64 the loss is less than 0.5dB at any SNR.

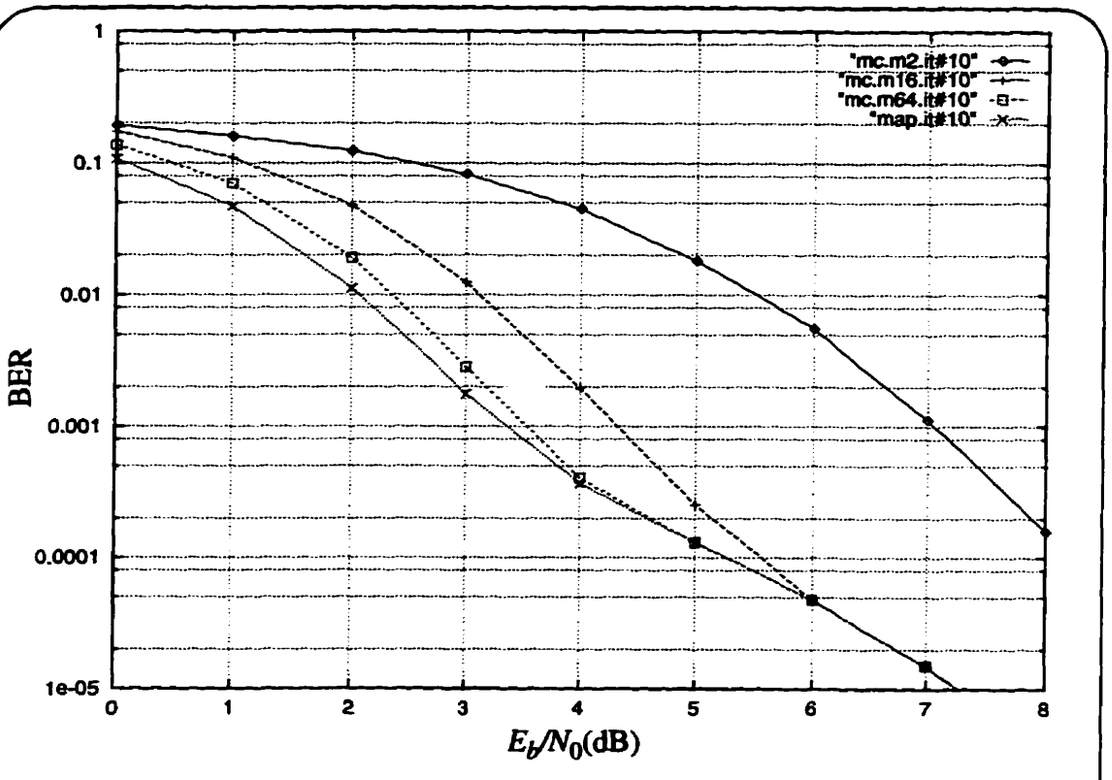


**Figure 3.3.1:** The error performance of the MC decoder and the MAP decoder on a turbo code with a pseudo-random interleaver of size 64. All the results are obtained after 10 iterations of the decoders.

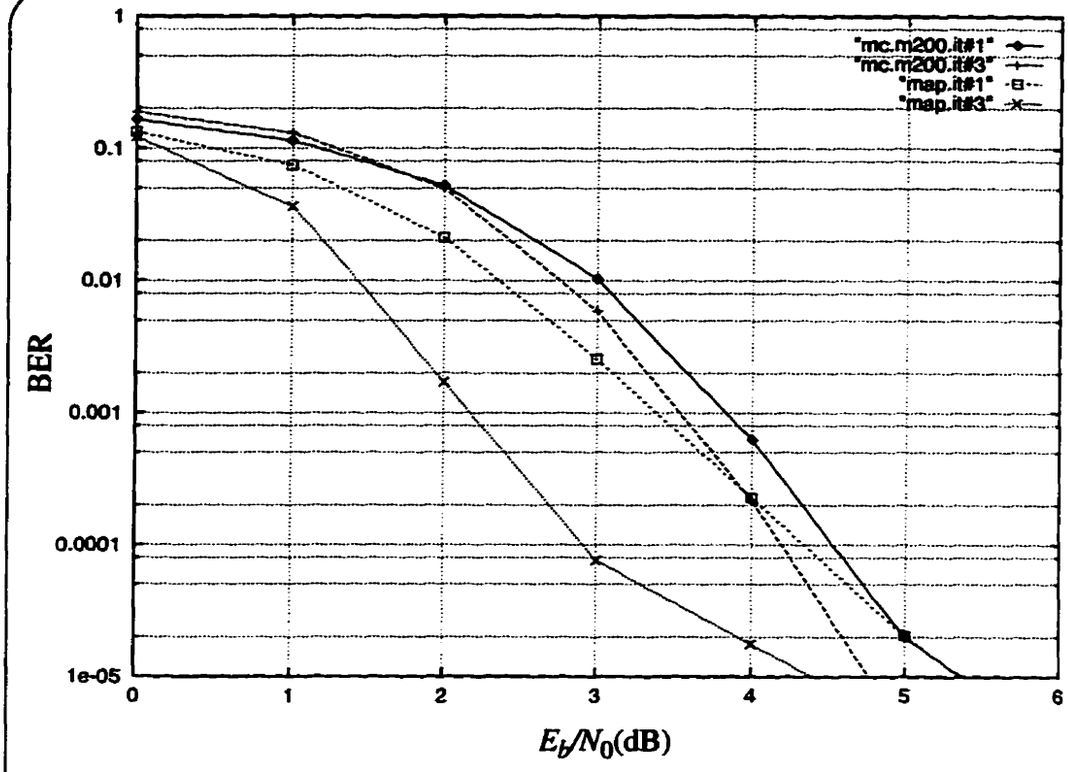


**Figure 3.3.2:** The error performance of the MC decoder and the MAP decoder on a turbo code with a pseudo-random interleaver of size 256.

The second example is a longer turbo code. Consider the one in Figure 2.1.1 with parameters $K$=256, interleaver=PRI, overall code rate=1/2 and both constituent encoders left open at the end of each information block. Then $M$ is chosen to be 200 and the MC decoder is run on the turbo code. The results of the first and third iteration are shown in Figure 3.3.2. The performance of the second iteration of the MC decoder is omitted because it is very close to that of the third iteration. The performance of the MAP decoder on the same turbo code is also shown in the same figure. The result of the second iteration is again omitted for the same reason. For the first iteration, the results show that the MC decoder has a loss of about 0.5dB at an error rate larger than $10^{-3}$ when compared

to the MAP decoder. The loss decreases for smaller BER and larger SNR. In particular, when SNR is larger than or equal to 5dB, both decoders have the same error performance. For the third iteration, the MC decoder has a loss of about 1.5dB at an error rate larger than $10^{-4}$ when compared to the MAP decoder. The loss decreases for smaller error rate and higher SNR. Although it is not shown in the figure, it can be speculated that as SNR reaches 5dB, the error performance of the MC decoder with three iterations is as good as that of the MAP decoder with the same number of iterations.

When a turbo code with $K$=900 is chosen for the simulation, it is found that not even a storage amount in the order of 1000 paths is enough for the MC decoder to produce an error performance that is as good as those in the previous two cases. This suggests that the SISOM algorithm in the MC decoder is not able to handle longer turbo codes. The reason is that for $K$=900, the total number of possible paths through the trellis is $2^{900}$. For the SISOM algorithm to retain enough paths to calculate a good approximation of the *a posteriori* probabilities, the amount of storage must be very large and therefore not practical for many situations. Thus the MC decoder is not suitable for long turbo codes.

As a summary, the MC decoder has a small degradation in error performance when compared to the MAP decoder for short turbo codes with $K$<300. The suitability of the MC decoder for these turbo codes still requires justification from the view point of complexity.

## 3.4 MC DECODER: HARDWARE COMPUTATIONAL COMPLEXITY

In this section the hardware computational complexity of the MC decoder

is considered. In particular, the total number of operations required for a decision to be made on an information bit is calculated. The operations include table look-ups, additions and multiplications which are involved in the metric and log-likelihood calculations. In order to compare the results with those of the MAP decoder obtained in [1] and [32] the conventions used in those two papers are followed.

Because the MC decoder consists of two identical SISOM algorithm modules, it is only necessary to calculate the complexity of a single SISOM algorithm and then multiply it by 2 and then by the number of iterations to get the final complexity of the decoder. An equivalent way is to compare the complexity of the SISOM algorithm to the BCJR algorithm.

In situations where it is not necessary to take into account the *a priori* information and to calculate the log-likelihood ratios, the arithmetic involved in the M-algorithm can be approximated by integer operations [21][32]. Unfortunately this is not the case and real number arithmetic is necessary in the SISOM algorithm. Therefore the following operations are in terms of real number operations.

Consider the turbo code in Figure 2.1.1 with an overall code rate=1/2 and no trellis termination in both constituent codes. The number of table look-ups required to calculate the metrics is calculated as follows. The sum within the bracket in (3.2.1) can be obtained by a single table look-up. Let $j = \left\lceil \log_2 \frac{M}{2} \right\rceil$ where $(j+1)$ is the level of the tree wide enough to have at least $M$ paths; $K$ = the block size of the information bits. In the case where no trellis termination is necessary for both constituent encoders, $K$ equals the size of the interleaver. Then the number of table look-ups due to metric calculation is

$$n_T^m = 2 + 2 \cdot 2 + 2 \cdot 2^2 + \ldots + 2 \cdot 2^j + 2M(K-j-1)$$

$$= 2^{j+2} - 2 + 2M(K-j-1) . \qquad (3.4.1)$$

Similarly the number of additions required to calculate all path metrics is

$$n_A^m = 2n_T^m = 2^{j+3} - 4 + 4M(K-j-1) . \qquad (3.4.2)$$

During the operation of the SISOM algorithm it is necessary to find the $M$ best paths among those that are already extended. Before the tree has fully grown to the size of $M$ paths, there is no need to do a sorting. Once the tree has grown to more than $M$ paths, however, it is necessary to perform a sorting on those extended paths in order to get the best $M$ ones. Assume an insertion sorting routine is used and it is necessary to sort the sequence of metrics in descending order. The number of comparisons required to perform the sorting depends on the pattern of the unsorted sequence. The fortuitous case is that the sequence is already in the correct order which requires only the minimum number of comparisons. In this best case situation the total number of comparisons is denoted as $\left(n_C^m\right)_B$, which is

$$\left(n_C^m\right)_B = (K-j+1)(2M-1) . \qquad (3.4.3)$$

On the other hand, in the worst case the initial sequence is in complete reverse order and it is necessary to perform a maximum number of comparisons. The total number of comparisons in this worst case situation is denoted as $\left(n_C^m\right)_W$, which is

$$\left(n_C^m\right)_W = (K-j+1)\left(2M^2 - M\right). \qquad 3.4.4)$$

In order for a microprocessor to perform a comparison, a subtraction of the

two values is done and based on whether the result is negative, zero or positive the arithmetic-logic-unit of the microprocessor will determine whether the first number is larger, equal or smaller than the second number. Thus a comparison is equivalent to a subtraction. Because a subtraction is equivalent to an addition, so is a comparison.

It is important to point out that the purpose of using the insertion sorting routine in the SISOM algorithm is that it is easy to obtain the exact expressions for the best and worst case computational complexity. In actual implementation, however, a more efficient sorting routine called quicksort is used [29]. The exact expression for the complexity of this routine is difficult to find. However, compared to the average complexity of the insertion sorting routine, which is $O(s^2)$, the average complexity of the quicksort routine has an order of $O(s \log s)$ when $s$ is the size of the sequence to be sorted. Moreover, during the extension phase of the SISOM algorithm the paths from the old list that have the largest partial metric can be extended first and then assigned to the initial positions of the new list. In this way it is very probable to end up with the minimum or close to the minimum computation for the next sorting. With these two considerations it can be conjectured that (3.4.3) is more likely than (3.4.4) to reflect the actual number of comparisons to be done in SISOM algorithm.

Finally, to calculate the log-likelihood ratio for all information bits, according to (3.2.5), it is required to perform

$$n_A^l = 2KM \text{ additions and}$$

$$n_M^l = K \text{ divisions (equivalent to multiplications).} \qquad (3.4.5)$$

From (3.4.1) to (3.4.5) one can calculate the number of operations required

for each information bit in one pass of the SISOM algorithm. They are summarized in the following:

$$n_{T/bit} = \frac{n_T^m}{K} = \frac{2^{j+2} - 2 + 2M(K-j-1)}{K}, \tag{3.4.6}$$

$$(n_{A/bit})_B = \frac{n_A^m + \left(n_C^m\right)_B + n_A^l}{K}$$

$$= \frac{2^{j+3} + (6M-1)(K-j-1) + M(4+2K) - 6}{K}, \tag{3.4.7}$$

$$(n_{A/bit})_W = \frac{n_A^m + \left(n_C^m\right)_W + n_A^l}{K}$$

$$= \frac{2^{j+3} + M(2M+3)(K-j-1) + 2M(2M-1) + 2KM - 4}{K} \tag{3.4.8}$$

and

$$n_{M/bit} = \frac{n_M^l}{K} = 1. \tag{3.4.9}$$

On the other hand, the number of operations required for each information bit in one pass of the BCJR algorithm is summarized as follows [1][32]:

$$n_{T/bit} = 2 \cdot 2^{\nu}, \tag{3.4.10}$$

$$n_{A/bit} = 4 \cdot 2^{\nu} \tag{3.4.11}$$

and

$$n_{M/bit} = 6 \cdot 2^{\nu} + 1 \tag{3.4.12}$$

where $\nu$ is the memory of the constituent code. Here, $\nu$ is equal to 4. Note that the per information bit complexity of the BCJR algorithm depends only on $\nu$ and not on the block

size $K$ of the turbo code. Because the performance of a turbo code when decoded by the MAP decoder becomes better when the block size ($K$) of the code is increased, it can be speculated that the performance of a turbo code when the MAP decoder is used can be improved without paying any penalty for the per information bit complexity of the decoding algorithm. Therefore the MAP decoder is recommended for applications where a long turbo code is required.

On the other hand, by looking at (3.4.6) to (3.4.12) it can be noticed that the complexity of the SISOM algorithm increases with $M$. If a long code is used rather than a short one, the long code must have a larger correction capability ($t$) in order to achieve a performance as good as that of the short code. However, $M$ increases exponentially with $t$ and for the long code a very large $M$ is required. The value of $M$ can be so large that the number of computations required becomes impractical. Therefore the SISOM algorithm is only suitable for decoding short turbo codes, which explains the failure of the MC decoder when it is used to decode the turbo code with $K$=900. In the following the computational complexity of the SISOM and BCJR algorithm is calculated for the short turbo codes simulated in Section 3.3.

Based on (3.4.6) to (3.4.12) one pass of the SISOM algorithm is compared to one pass of the BCJR algorithm. First consider the turbo code mentioned in the first part of Section 3.3, which is a code with $K$ equal to 64. Only the MC decoder with $M$ equal to 64 is considered because its performance is very close to that of the MAP decoders. The computational complexity of the corresponding SISOM and BCJR algorithm are listed in Table 3.4.1. All operations listed are in terms of real number operations.

**TABLE 3.4.1:** A comparison of the computation complexity of one pass of the SISOM algorithm and one pass of the BCJR algorithm for $K$=64.

| | SISOM ($M$=64) | BCJR |
|---|---|---|
| Table Look-ups | 118 | 32 |
| Additions (Best Case) | 483 | 64 |
| Additions (Worst Case) | 7984 | - |
| Multiplications | 1 | 97 |

Notice that the worst case additions for the SISOM algorithm is much worse than that of the best case additions. As already explained one will get close to the best case performance in most situations; thus it is only necessary to pay attention to the latter. Also notice that the SISOM algorithm for $M$=64 requires 86 more table look-ups than the BCJR algorithm. In most DSP chips a table look-up can be accomplished in one machine cycle. Thus not counting other operations the SISOM algorithm is 86 machine cycles slower than the BCJR algorithm. Another thing to notice for the SISOM algorithm is that it requires 419 more real number additions per information bit than the BCJR algorithm while the latter requires 96 more multiplications per information bit than the former. On a general purpose DSP chip, a real number multiplication requires more circuitry and more machine cycles than a real number addition. If the DSP performs an addition 10 times faster than a multiplication, then the SISOM algorithm will out perform the BCJR algorithm in complexity. For most DSP this is not an unreasonable assumption and therefore the SISOM algorithm for $M$=64 should out perform the BCJR algorithm.

Now consider the turbo code simulated in the second part of Section 3.3, which is a code with $K$=256. The SISOM algorithm with $M$=200 is considered. The computational complexity of the SISOM and the BCJR algorithm are listed in Table 3.4.2. Again all operations are listed in terms of real number operations.

**TABLE 3.4.2:** A comparison of the computational complexity of one pass of the SISOM algorithm and one pass of the BCJR algorithm for $K$=256.

| | SISOM ($M$=200) | BCJR |
|---|---|---|
| Table Look-ups | 390 | 32 |
| Additions (Best Case) | 1568 | 64 |
| Additions (Worst Case) | 79108 | - |
| Multiplications | 1 | 97 |

In this case the SISOM algorithm for $M$=200 requires 358 more table look-ups and about 1500 more additions than the BCJR algorithm. On the other hand, the BCJR algorithm still requires 96 more multiplications than the SISOM algorithm. Depending on the technology, the SISOM algorithm may still out perform the BCJR algorithm. For example, if the DSP chip processes an addition 20 times faster than a multiplication, then the SISOM algorithm will slightly out perform its counterpart in complexity.

By looking at the error performance and the complexity of the SISOM algorithm, it can be concluded that the MC decoder, which consists of two SISOM algorithm modules, is suitable for turbo codes with $K$<300. In particular, for $K$=64 the MC decoder with $M$=64 and for $K$=256 the MC decoder with $M$=200 have has less or comparable complexity than the MAP decoder with small degradation in error performance.

Although the MC decoder is not suitable for decoding longer codes, the M-algorithm can be used to decode the entire structure in one pass. Thus it is interesting to see whether this new approach can be used to overcome the problem of decoding long turbo codes. Before this approach is discussed, it is interesting to see another measure of the complexity of the SISOM algorithm. This measure is the number of nodes visited by the algorithm. Because the operations performed at each node in the SISOM algorithm are different from those in the BCJR algorithm, the number of nodes visited by the algorithm is not a suitable measure of the actual complexity. However, it does provide another point of view of the same issue and can act as a secondary measure.

## 3.5 NUMBER OF NODES VISITED

In this section the number of nodes visited by the SISOM algorithm and the BCJR algorithm are calculated. The SISOM algorithm is considered first. Without any termination of the constituent encoders the number of nodes visited by one pass of the SISOM algorithm is

$$N_V = 2 + 2^2 + \ldots + 2^{j+1} + 2M(K-j-1)$$

$$= 2^{j+2} - 2 + 2M(K-j-1) \qquad (3.5.1)$$

in which $j = \left\lceil \log_2 \frac{M}{2} \right\rceil$ where $(j+1)$ is the level of the tree wide enough to have at least $M$ paths and $K$ = the information block size. Thus the number of nodes visited per information bit is

$$N_{V/bit} = \frac{2^{j+2} - 2 + 2M(K-j-1)}{K}. \qquad (3.5.2)$$

On the other hand for the BCJR algorithm under the same situation, i.e., no constituent encoders are terminated, the number of nodes visited by one pass of the algorithm (including both forward and backward recursion) is

$$N_V = 2\left(2+2^2+\ldots+2^{\nu}+2^{\nu+1}(K-\nu)\right)$$

$$= 2\left(2^{\nu+1}-2+2^{\nu+1}(K-\nu)\right)$$

$$= 2^{\nu+2}-4+2^{\nu+2}(K-\nu)\,. \qquad (3.5.3)$$

The number of nodes visited per information bit is therefore

$$N_{V/bit} = \frac{2^{\nu+2}-4+2^{\nu+2}(K-\nu)}{K}\,. \qquad (3.5.4)$$

Based on (3.5.2) and (3.5.4) the number of nodes visited by the two algorithms can be calculated for the codes which have been used in the simulations.

For the turbo code used in the first part of Section 3.3.3, which is a turbo code with $K$=64, the number of nodes visited by the SISOM algorithm for $M$=64 is listed in Table 3.5.1. The number of nodes visited by the BCJR algorithm for the same turbo code is also listed. It can be noticed from the table that the number of nodes visited by the SISOM algorithm is almost double that of its counterpart. Therefore in this case the BCJR algorithm outperforms the SISOM algorithm.

**TABLE 3.5.1:** The number of nodes visited by one pass of the SISOM algorithm and one pass of the BCJR algorithm for $K$=64.

| | SISOM ($M$=64) | BCJR |
|---|---|---|
| # of Nodes Visited/ Information Bit | 118 | 61 |

For the turbo code used in the second part of Section 3.3, which is a code with $K$=256, the number of nodes visited by both algorithms are listed in Table 3.5.2. From there it can be noticed that the number of nodes visited by the SISOM algorithm is far more than that of the BCJR algorithm. Again in terms of the number of nodes visited the BCJR algorithm outperforms the SISOM algorithm.

**TABLE 3.5.2:** The number of nodes visited by one pass of the SISOM algorithm and one pass of the BCJR algorithm for $K$=256.

| | SISOM ($M$=200) | BCJR |
|---|---|---|
| # of Nodes Visited/ Information Bit | 389 | 63 |

Although the complexity of the SISOM algorithm is more than that of the BCJR algorithm in terms of the number of nodes visited, it does not actually reflect the computation time required by the algorithm. Thus this measure should only be looked at as a secondary measure of complexity.

## 3.6 REDUCED COMPLEXITY DECODING FOR THE ENTIRE TURBO CODE: DECODER STRUCTURE

If the code tree of a turbo code can be constructed one can treat it as if it were a normal tree code and apply a sequential decoding algorithm to decode it. The particular algorithm of interest is the M-algorithm. In this section the M-algorithm mentioned in Section 2.6 with few modifications is used to decode the turbo code. The M-algorithm is a reduced complexity decoding algorithm when $M$ is not too large and whether this is the case for turbo codes will be looked at in Section 3.8.

First it is necessary to construct the code tree for a turbo code. Consider again the turbo code in Figure 2.1.1. Because of the presence of the interleaver the elements of the input sequence $d_k$ and $d_k'$ at the input to the two constituent encoders can be dependent or independent at each time instant $k$. If they are dependent at time instant $k$ there will be one or two new branches coming out of the nodes at the previous level of the tree. However, if $d_k$ and $d_k'$ are completely independent at $k$, there will be four branches coming out of the previous nodes. Whether they are dependent or not at $k$ can be found out by evaluating the functions $f()$ and $g()$ at $k$ and comparing the results with the value of $k$ itself. The functions $f()$ and $g()$ are respectively the forward and inverse mapping associated with an interleaver. The relationships between $f()$, $g()$ and the dependence of the inputs at time $k$ are summarized in Table 3.6.1. With that table a computer program can be written to generate the tree of the turbo code.

After the code tree is constructed it is necessary to discuss the structure of the decoder. The block diagram of the decoder is shown in Figure 3.6.1 and the decoder will be referred to as the MW decoder (M-algorithm on the whole turbo code). Note that when compared to the MAP decoder in Figure 2.5.1 or the MC decoder in Figure 3.2.1 the MW decoder is much simpler because only one module of the M-algorithm is necessary and iterations are not needed.

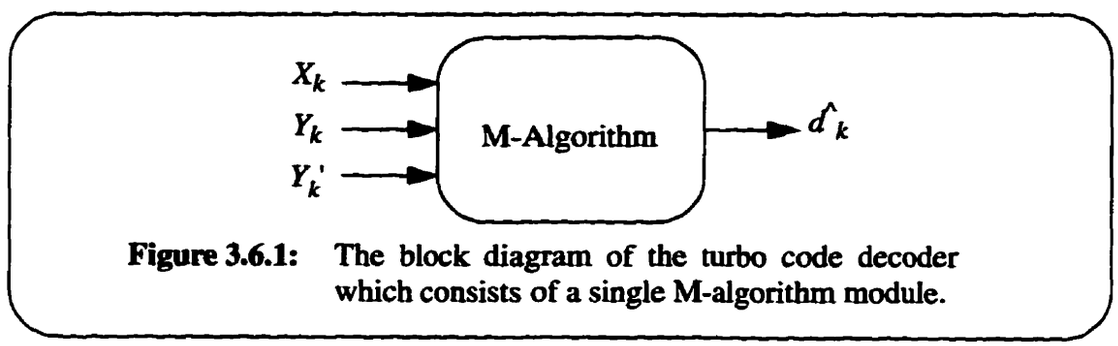


**Figure 3.6.1:** The block diagram of the turbo code decoder which consists of a single M-algorithm module.

**TABLE 3.6.1:** The relationships between $k$, $f()$, $g()$ and the dependence of the inputs at time instant $k$.

| Case | Conditions | Number of New Branches | Corresponding Inputs to Both Constituent Encoders at time $k$ |
|---|---|---|---|
| I | $f(k)=g(k)=k$ | 2 | 1) $d_k=d_k'=0$<br>2) $d_k=d_k'=1$ |
| II | $f(k)<k$ and $g(k)<k$ | 1 | 1) $d_k=d_{f(k)}'$ and $d_k'=d_{g(k)}$ |
| III | $f(k)<k$ and $g(k)>k$ | 2 | 1) $d_k=d_{f(k)}'$ and $d_k'=0$<br>2) $d_k=d_{f(k)}'$ and $d_k'=1$ |
| IV | $f(k)>k$ and $g(k)<k$ | 2 | 1) $d_k=0$ and $d_k'=d_{g(k)}$<br>2) $d_k=1$and $d_k'=d_{g(k)}$ |
| V | $f(k)>k$ and $g(k)>k$ | 4 | 1) $d_k=d_k'=0$<br>2) $d_k=0$ and $d_k'=1$<br>3) $d_k=1$ and $d_k'=0$<br>4) $d_k=d_k'=1$ |

Before the MW decoder can be used to decode the turbo code it is necessary to have a proper metric, in particular, a metric which can be used as a measure to decide which paths to retain during the running of the M-algorithm. A suitable metric is derived in Section A.2 of Appendix A and is given below:

$$BM_{i,k} = -\left( \frac{(X_k - \pm 1)^2}{2\sigma^2} + \frac{(Y_k - \pm 1)^2}{2\sigma^2} + \frac{(Y_k' - \pm 1)^2}{2\sigma^2} \right). \qquad (3.6.1)$$

With the code tree and the metric in hand one is almost ready to use the MW decoder to decode the turbo code. Because the sliding window requirement of the original M-algorithm is not important, it can be ignored and all of the estimates are produced at the end of the algorithm. However, the minimum storage requirement is still important and it is nec-

essary to specify the minimum value of $M$ in order to have close to optimal results. The minimum value of $M$ can be calculated for a turbo code when its effective free distance is known. In the following section a computer program is written to obtain the effective free distances and these values will be used to calculate the value of $M$ for the simulation of the MW decoder.

## 3.7 MW DECODER: SIMULATION RESULTS

Again the simulation starts with a short turbo code. Consider the turbo code in Figure 2.1.1, with parameters interleaver=PRI of size 64, overall code rate=1/3 and termination of the first encoder. In order to calculate the required minimum storage requirement, it is necessary to know the effective free distance of the code. A computer program is written to obtain $d_{free}^{eff}$ of this turbo code and it is found to be 10. Thus the correction capability ($t$) of the code is less than 5. Although the overall code rate is 1/3, the code rate at a particular time instant can be 0/3 (according to Case II of Table 3.6.1), 1/3 (Case I, III and IV of Table 3.6.1) or 2/3 (Case V of Table 3.6.1). The overall code rate of a code will be called the global rate while the code rate at a particular time instant will be called the local rate. For a turbo code the local rate is a function of time. To be on the safe side it is necessary to take the maximum local rate and maximum error correction capability into account when the minimum storage requirement of the M-algorithm is calculated. For this particular turbo code the maximum local rate is 2/3. Since $t<5$, it can be calculated from Section 2.6 that the minimum value of $M$ is $(3.85)^5$=846. The value of $M$ is allowed to be 1000 and the MW decoder is used to decode the turbo code. The results are shown in Figure 3.7.1. Also shown in the figure are the results obtained when the MAP decoder is

used to decode the same turbo code. For the MAP decoder, the results after the first and tenth iteration are shown. It can be observed that the MW decoder fails to decode this turbo code.

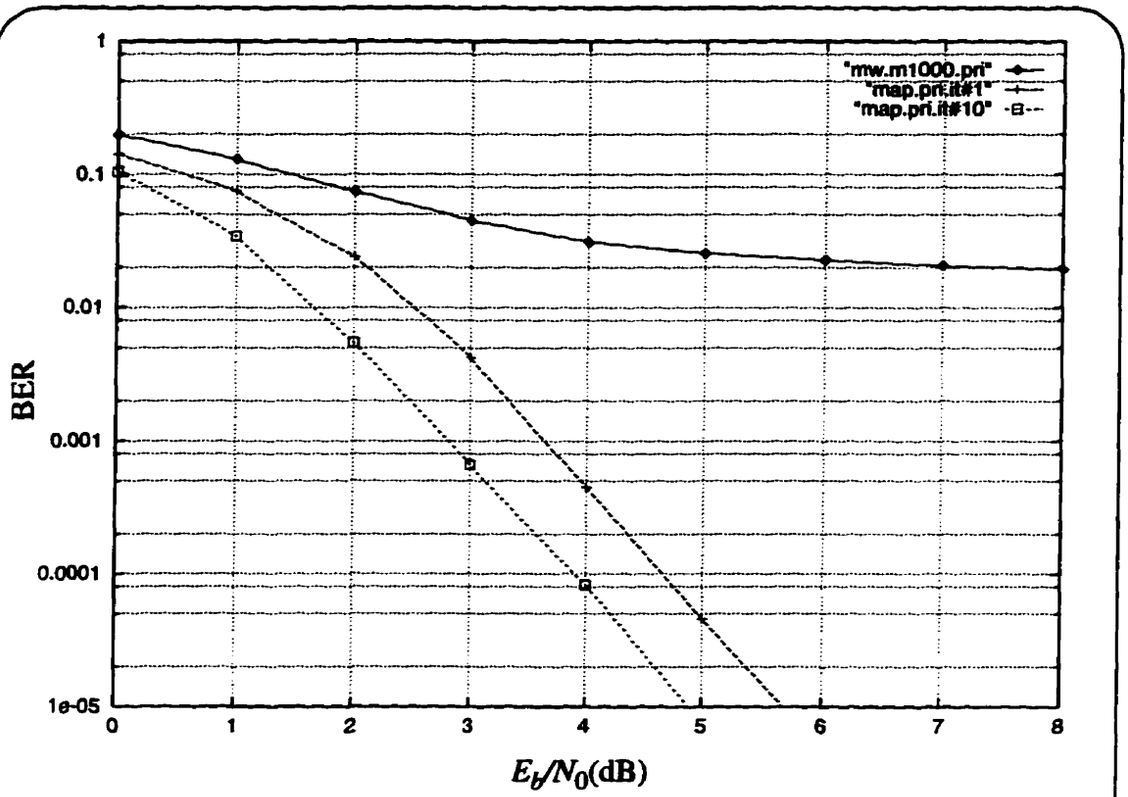


**Figure 3.7.1:** The error performance of the MW decoder when it is used to decode a turbo code with a pseudo-random interleaver of size 64. In this case $M$ is set to 1000 which is already larger than the minimum required value. Also shown are the results of the MAP decoder with number of iterations equal to 1 and 10.

Because of the poor performance the interleaver is changed to HDI and the other parameters are kept unchanged. By using the same program the effective free distance of the turbo code is found to be 8. The correction capability of this code is therefore less than 4. For the maximum local rate of 2/3, the minimum value of $M$ is $(3.85)^4$≈219. The MW decoder with $M$=250, 500 and 1000 is used respectively to decode the turbo code

and the results are shown in Figure 3.7.2. Repeated in the same figure are the results of the MAP decoder from Figure 3.7.1. Notice that the performance of the MW decoder improves suddenly when the PRI is replaced by the HDI in the turbo code. This is because the original M-algorithm is designed for code trees with a regular growth pattern. The code tree of the turbo code with a pseudo-random interleaver is too irregular for the M-algorithm to keep track of the correct path at each time instant. For the turbo code with a read-in-horizontally-and-read-out-diagonally interleaver, the tree is still irregular but to a lesser degree such that the M-algorithm performs better.

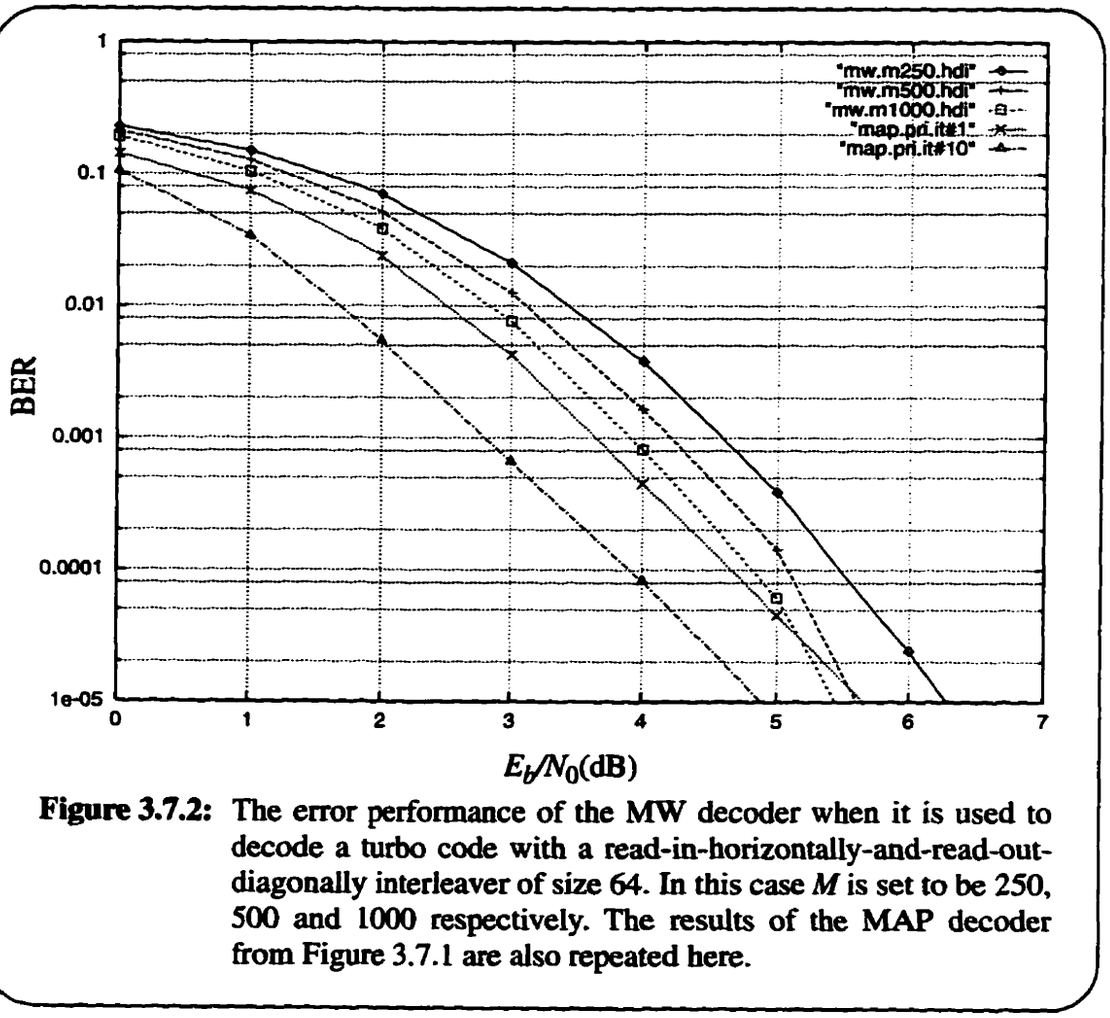


**Figure 3.7.2:** The error performance of the MW decoder when it is used to decode a turbo code with a read-in-horizontally-and-read-out-diagonally interleaver of size 64. In this case $M$ is set to be 250, 500 and 1000 respectively. The results of the MAP decoder from Figure 3.7.1 are also repeated here.

From Figure 3.7.2 it can be noticed that as $M$ is doubled from 250 to 500 the gain achieved is less than 0.5dB for an error rate less than $10^{-4}$. When $M$ is doubled from 500 to 1000 the gain is even smaller. This shows that as M is increased over 250 the gain in return is diminishing. Thus it can be concluded that the minimum storage requirement for $M$ is quite accurate.

The performance of the MW decoder in Figure 3.7.2 can also be compared to that of the MAP decoder. Although not shown in the figure, it can be speculated that the MW decoders with $M$=500 and $M$=1000 achieve performance close to that of the MAP decoder with ten iterations at a SNR of 6dB. For SNR less than 5db the MW decoder with $M$=1000 has a loss of about 1 dB when compared to the MAP decoders.

For the next example the MW decoder is used to decode a longer turbo code. In particular, consider the turbo code in Figure 2.1.1 with parameters interleaver=HDI of size 256, overall code rate=1/3 and termination of the first constituent encoder. The value of $M$ is chosen to be 1000 which is an arbitrary large number and the results are shown in Figure 3.7.3. Also shown in the same figure are the results of the MAP decoder on the same turbo code except with a pseudo-random interleaver. It can be noticed from the figure that even for a large $M$ it requires a fairly large SNR for the MW decoder to approach the MAP decoder in error performance. The reason for the poor performance is similar to that given in Section 3.3 and 3.4. In particular, a longer block code requires a larger error correction capability ($t$) than a short block code in order to achieve the same error probability. However, the minimum storage requirement of the M-algorithm increases exponentially with $t$ and the value of $t$ necessary to achieve a low error probability may require an intolerable storage requirement. Therefore the MW decoder is

only suitable for short turbo codes.

For short turbo codes the MW decoder has performance close to that of the MAP decoder. However, in order for the MW decoder to be useful, its complexity must be less than that of its counterpart. Therefore the next topic of discussion will be the computational complexity of the MW decoder.

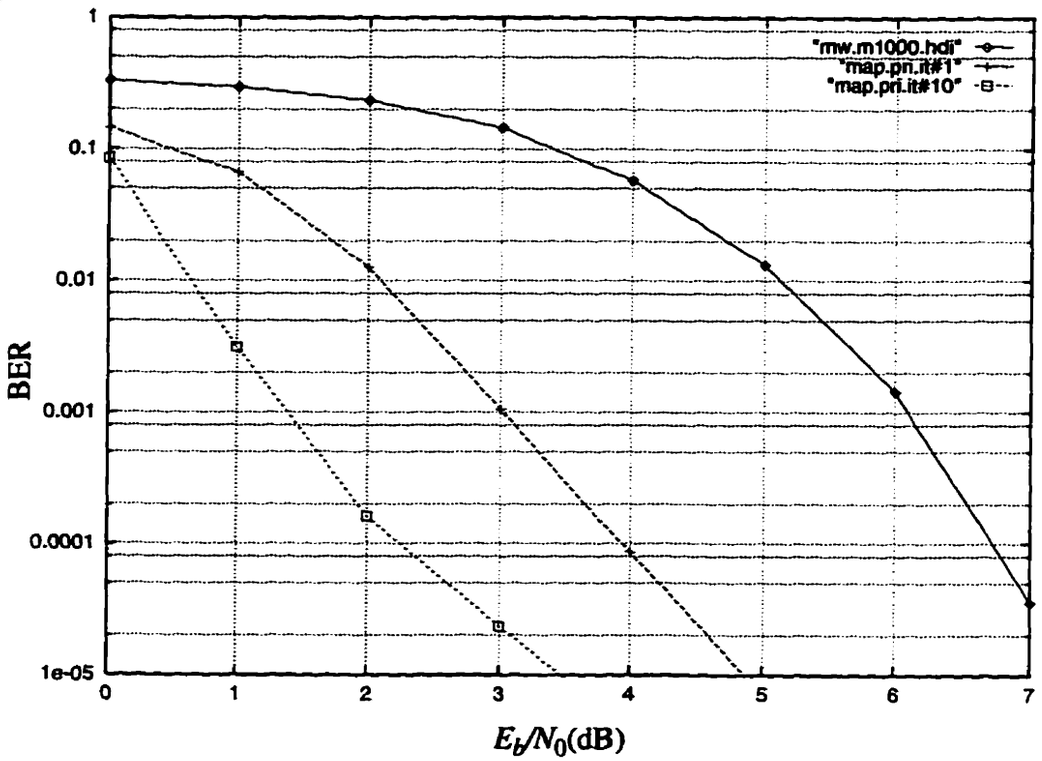


**Figure 3.7.3:** The error performance of the MW decoder when it is applied to a turbo code with a read-in-horizontally-and-read-out-diagonally interleaver of size 256. In this case $M$ is set to be 1000 which is already a large value. The results of the MAP decoder on a turbo code with a pseudo-random interleaver are also shown in the same figure. The number of iterations is allowed to be 1 and 10.

## 3.8 MW DECODER: HARDWARE COMPUTATIONAL COMPLEXITY

As in Section 3.4 the complexity of the MW decoder in terms of the number of operations required for each information bit is considered. The conventions used in [1] and [32] are once again followed so that the results for the MW decoder can be compared to those of the MAP decoder. Also consider the turbo code in Figure 2.1.1 with an overall code rate=1/3, interleaver size=64 and with the first encoder terminated. This implies if $v$ is the memory of the first encoder $v$ bits will be used to terminate the first encoder and the information block size will be equal to $(64-v)$. Because the constituent code is of memory 4, $K$ is equal to 60. In fact, this is the second turbo code chosen for simulation in Section 3.7.

In the case when the M-algorithm is applied directly to decode the turbo code it is not necessary to worry about the *a priori* information and the log-likelihood ratios. Therefore the arithmetic involved in the MW decoder can be approximated by integer operations [21][32]. In the following the upper letter "$N$" with appropriate subscripts or superscripts is used to indicate the number of integer operations required for the MW decoder. As already known the code tree of a turbo code is irregular and it is very difficult to calculate the exact complexity of searching through such a kind of tree. In this situation the code tree of the turbo code is approximated with a binary code tree of length $K+v$ where the first $K$ time steps correspond to the $K$ information bits. The total number of nodes or branches in this binary code tree is therefore the same as that of the turbo code. As a result the approximation should be accurate enough.

First of all the number of table look-ups required is calculated. The sum of

the three terms in (3.6.1) can be obtained in a single table look-up. Let $j = \left\lceil \log_2 \frac{M}{2} \right\rceil$, where $(j+1)$ is the level of the tree wide enough to have at least $M$ paths. Then the number of table look-ups due to metric calculation is

$$N_T = 2 + 2(2) + 2\left(2^2\right) + \ldots + 2(2)^j + 2M(K-j-1) + vM$$

$$= 2^{j+2} - 2 + 2M(K-j-1) + vM. \tag{3.8.1}$$

Similarly the number of additions required to calculate all the path metrics is

$$N_A = N_T = 2^{j+2} - 2 + 2M(K-j-1) + vM. \tag{3.8.2}$$

With the same argument as in Section 3.4 the total number of comparisons in this best case situation is

$$(N_C)_B = (K-j+1)(2M-1) + (M-1) \tag{3.8.3}$$

and in the worst case situation

$$(N_C)_W = (K-j+1)\left(2M^2 - M\right) + \frac{M(M-1)}{2}. \tag{3.8.4}$$

As reasoned in Section 3.4, (3.8.3) is more likely to reflect the actual number of comparisons to be made by the MW decoder.

From (3.8.1) to (3.8.4) the number of operations required for each information bit can be calculated. They are

$$N_{T/bit} = \frac{N_T}{K}$$

$$= \frac{2^{j+2} - 2 + 2M(K-j-1) + vM}{K}, \tag{3.8.5}$$

$$(N_{A/bit})_B = \frac{N_A + (N_C)_B}{K}$$

$$= \frac{2^{j+2} + (4M-1)(K-j-1) + M(5+v) - 5}{K} \qquad (3.8.6)$$

and

$$(N_{A/bit})_W = \frac{N_A + (N_C)_W}{K}$$

$$= \frac{2^{j+2} + M(2M+1)(K-j-1) + 2M(2M-1) + \frac{M}{2}(M-1) + vM - 2}{K}. \qquad (3.8.7)$$

Based on (3.8.5) to (3.8.7) and (3.4.10) to (3.4.12) the comparison between the MW decoder and the MAP decoder can be made. For the turbo code mentioned above, the value of each operation for both decoders is summarized in Table 3.8.1.

**TABLE 3.8.1:** A comparison of the computational complexity of the MW decoder and that of the MAP decoder for $K$=60.

| | MW ($M$=1000) (integer) | MAP (1 iteration) (real) | MAP ($n$ iterations) (real) |
|---|---|---|---|
| Table Look-ups | 1767 | 64 | 64$n$ |
| Additions (Best Case) | 3517 | 128 | 128$n$ |
| Additions (Worst Case) | 1742559 | - | - |
| Multiplications | 0 | 194 | 194$n$ |

Note that the MW decoder requires only integer operations while the MAP

decoder requires real number operations. For table look-ups and additions, it seems that the MW decoder requires more operations than the MAP decoder in mere numbers. In terms of multiplications, the MW decoder requires none while the MAP decoder requires at least hundreds of them. For a popular general purpose DSP (e.g., a TMS32010), a real number operation is about 60 times slower than an integer operation [20][38]. In such a case the MW decoder is much more efficient than the MAP decoder with even one iteration. Because the MW decoder with $M$=1000 has performance approaching to that of the MAP decoder with 10 iterations at SNR larger than 6dB and a small loss at SNR less than 6dB, it can be concluded that for turbo codes with $K$<100 and in applications where processing delay is important the MW decoder is more suitable than the MAP decoder.

## 3.9 NUMBER OF NODES VISITED

In the previous section the number of operations required for both MW and MAP decoders have been looked at. In this section the number of nodes visited by the two decoders will be calculated and compared. During the calculation the code tree of a turbo code will also be approximated by a binary code tree of the same length.

First consider the MW decoder. For the same code used in the previous section the number of nodes visited by the decoder is

$$N_V = 2 + 2^2 + \ldots + 2^{j+1} + 2M(K-j-1) + vM$$

$$= 2^{j+2} - 2 + 2M(K-j-1) + vM \qquad (3.9.1)$$

in which $j = \left\lceil \log_2 \frac{M}{2} \right\rceil$ where $(j+1)$ is the level of the tree wide enough to have at least $M$ paths. Thus the number of nodes visited per information bit is

$$N_{V/bit} = \frac{2^{j+2} - 2 + 2M(K-j-1) + vM}{K}. \qquad (3.9.2)$$

On the other hand when the MAP decoder is used to decode the same turbo code, the number of nodes visited by one iteration of the MAP decoder (including one pass through the BCJR1 and one pass through BCJR2) is

$$N_V = 2 \cdot 2\Big(2\big(2 + 2^2 + \ldots + 2^v\big) + 2^{v+1}(K-v)\Big)$$

$$= 2^{v+4} - 16 + 2^{v+3}(K-v). \qquad (3.9.3)$$

The number of nodes visited per information bit in one iteration is therefore

$$N_{V/bit} = \frac{2^{v+4} - 16 + 2^{v+3}(K-v)}{K}. \qquad (3.9.4)$$

Based on (3.9.2) and (3.9.4) one can calculate the number of nodes visited by the two algorithms.

For the turbo code mentioned above, the number of nodes visited by the MW decoder for the case $M$=1000 is listed in Table 3.9.1. The number of nodes visited by the MAP decoder for the same turbo code is also listed there.

**TABLE 3.9.1:** The number of nodes visited by the MW decoder and the MAP decoder.

| | MW ($M$=1000) | MAP (10 iteration) | MAP ($n$ iteration) |
|---|---|---|---|
| # of Nodes Visited/ Information Bit | 1767 | 1230 | 123$n$ |

It can be noticed from the table that the number of nodes visited by the MW decoder with $M$=1000 is 537 more than that of the MAP decoder with 10 iterations.

Therefore in terms of the number of nodes visited the MW decoder has a greater complexity than that of the MAP decoder. As mentioned before, however, this can only be used as a secondary measure of complexity.

## 3.10 SUMMARY

In this chapter the issue of reduced complexity decoding of turbo codes has been discussed. In particular, two methods based on the original M-algorithm have been derived that are particularly useful for short turbo codes: the one on the constituent level is suitable for $K<300$; the one on the entire turbo code is suitable for $K<100$. In terms of error performance, the first method has negligible degradation when compared to the MAP decoder. In terms of complexity, whether it out performs its counterpart or not depends on how the DSP handles the multiplications. But for most DSP chips the complexity of the MW decoder is either lower than or comparable to that of the MAP decoder. For the second method, the error performance has a coding loss of about 1dB when compared to the MAP decoder at low SNR (SNR approximately less than 5dB) and approaches its counterpart for moderate and high SNR (SNR>5dB). The complexity of the second method, however, is much less than that of the MAP decoder. Therefore these two decoders are recommended for applications where short turbo codes are used and decoder processing delay is important.

This is the end of the discussion on reduced complexity decoding of turbo codes. In the next chapter the question proposed in Chapter One will be partially answered by comparing the performance of turbo codes and BCH codes.

# CHAPTER 4 COMPARISON OF TURBO CODES AND BLOCK CODES

## 4.1 INTRODUCTION

In order to shed light on the question "If a traditional block code of the same length as the turbo code in [11] is used, will it out perform or perform as well as the turbo code?", a comparison on some short turbo codes and BCH codes will be made in this chapter. Based on these results speculation on and generalization to any longer codes will also be given. Because long BCH codes are difficult to decode, only short codes of the same rate are chosen for the comparison.

Two turbo codes are chosen to be compared with the BCH codes. They are chosen because BCH codes of similar parameters can be found. Both turbo codes have the same configuration as the one in Figure 2.1.1. The first one has a pseudo-random interleaver of size 64, global rate of 1/2 and the first encoder terminated. This code will be called **TC1**. The second code is the same as the first one except for the pseudo-random interleaver is of size 256. This code will be called **TC2**. These turbo codes will be decoded by the MAP decoder discussed in Section 2.5, which is the one used by Berrou *et al* in [11] to produce astounding results.

For the BCH codes two rate 1/2 codes are chosen and they are listed in Table 4.1.1. The generator polynomials in Table 4.1.1 are from [21]. They are given in an octal representation. The example shown in Appendix C illustrates how the octal representation can be converted into the polynomial form.

**TABLE 4.1.1:** The BCH codes chosen for the comparison. For each code the parameters are also listed in detail.

| | $N$ | $K$ | $R$ | $t$ | $g(x)$ |
|---|---|---|---|---|---|
| **C1** | 63 | 30 | 0.476 | 6 | 157464165547 |
| **C2** | 255 | 131 | 0.514 | 18 | 2157133314715101512612502774421420241654 71 |

## 4.2 QUICK COMPARISON

A quick comparison of the BCH codes with the turbo codes can be made when theoretical error performance approximation of the BCH codes can be obtained. For any binary ($N$, $K$) BCH code used in an AWGN channel with antipodal signalling and maximum likelihood decoding, the error performance in terms of BER can be approximated when the minimum distance and the number of minimum distance codewords of the code are known. The expression for the approximate BER is

$$BER \approx A_{d_{min}} Q\left(\sqrt{2R\frac{E_b}{N_0}d_{min}}\right) \tag{4.2.1}$$

where from [23]

$$A_{d_{min}} \approx \frac{1}{2^{N-K}}\binom{N}{d_{min}} \tag{4.2.2}$$

is the number of minimum distance codewords. Moreover, $R = \frac{K}{N}$ is the global code rate and $Q(x) = \frac{1}{\sqrt{2\pi}}\int_x^\infty e^{-y^2/2}dy$ is the usual definition of the $Q$ function. The approximation in (4.2.1) is not accurate for all SNRs, neither is the one in (4.2.2) for all values of $N$.

However, they can provide a rough idea of the error performance.

The $d_{min}$ error approximation of **C1** and the simulated performance of **TC1** are shown in Figure 4.2.1. Although **TC1** is a (128, 60) code, which is longer than **C1**, a (63, 30) code, it can be noticed that the maximum likelihood performance of **C1** is better than the simulation results of **TC1**, even after 10th iteration of the MAP decoder. Therefore if the approximation is accurate **C1** will be preferable to **TC1**.

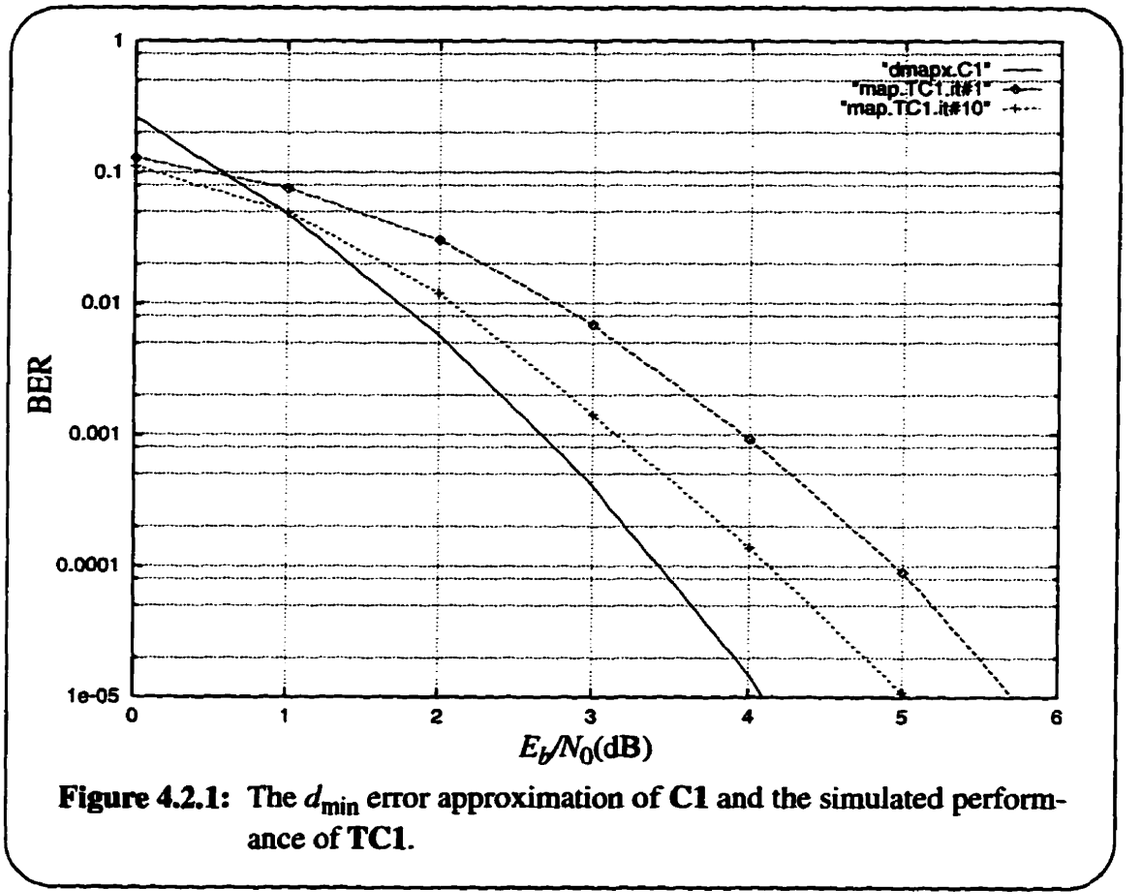


**Figure 4.2.1:** The $d_{min}$ error approximation of **C1** and the simulated performance of **TC1**.

A similar situation occurs for **C2** and **TC2**. The $d_{min}$ error approximation of **C2** and the simulated performance of **TC2** are plotted in Figure 4.2.2. It can be noticed that the maximum likelihood performance of **C2** is better than the performance of **TC2**.

Note that **C2** is only a (255, 131) code while **TC2** is a (512, 252) code. Therefore if the approximation is accurate, **C2** is also preferable to **TC2**.

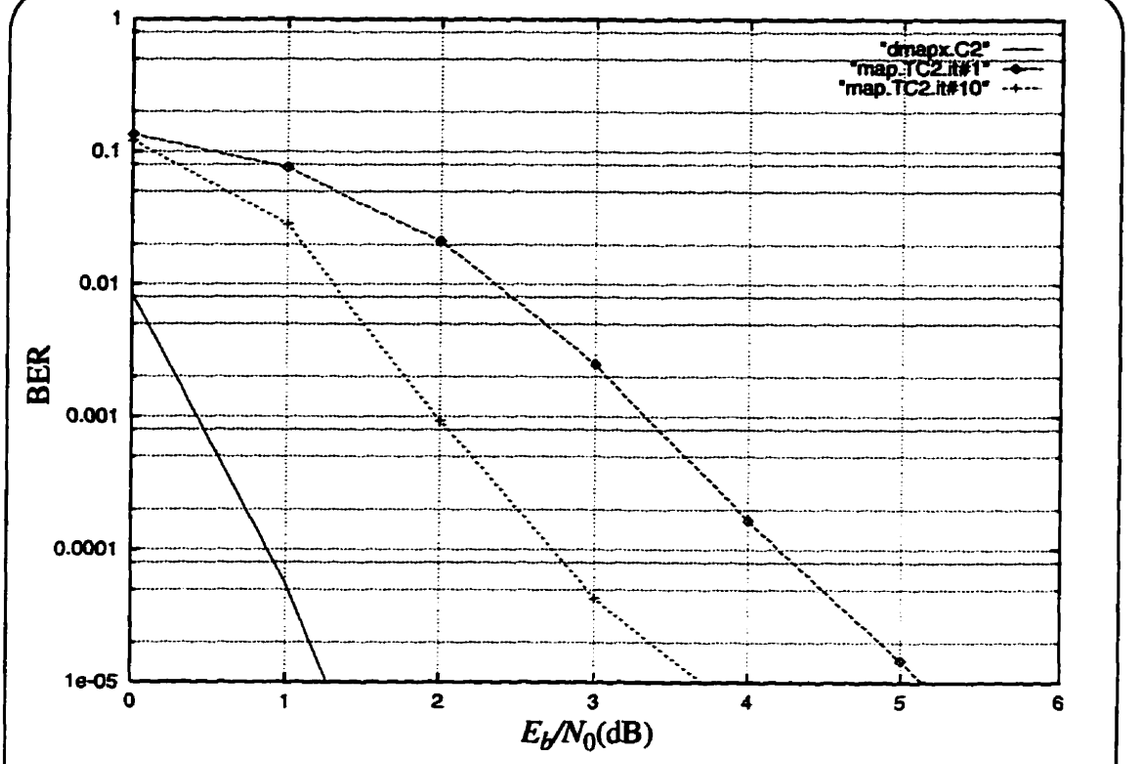


**Figure 4.2.2:** The $d_{min}$ error approximation of **C2** and the simulated performance of **TC2**.

Whether the performance of the BCH code is better than that of the turbo code is still subject to verification by simulation. However, the results from this section do give a quick indication of the relative performance of the codes.

## 4.3 COMPARISON BY SIMULATION RESULTS AND COMPUTATIONAL COMPLEXITY

In this section the simulated performance of the BCH code and the com-

plexity of decoding it will be obtained. In particular, the M-algorithm will be used to decode the BCH code. The M-algorithm is chosen because it is available from the first part of the thesis and it has a performance close to maximum likelihood decoding if enough storage requirement is specified.

When the M-algorithm is used to decode the BCH code, soft decision decoding can be used. In particular, if a code tree can be constructed for the BCH code, the M-algorithm can be made to accept soft inputs from the channel and decode the code by searching through the tree. Indeed, by following the approach in [35], one can construct a code tree for the BCH code. However, the resultant code tree has a maximum local rate of rate 1 and is thus not suitable for the M-algorithm to search through. A solution is found by extending **C1** to a new code **EC1** and **C2** to **EC2** so that each new code has a maximum local rate of 1/2. The parameters of **EC1** and **EC2** are listed with those of **C1** and **C2** in Table 4.3.1. The details on how soft decision decoding is applied to block codes, how the code tree is generated in [35], and how the BCH codes **C1** and **C2** are extended to **EC1** and **EC2** is described in Appendix D.

**TABLE 4.3.1:** The BCH codes **C1** and **C2** with their extended versions **EC1** and **EC2.**

| | $N$ | $K$ | $R$ | $t$ | $g(x)$ |
|---|---|---|---|---|---|
| **C1** | 63 | 30 | 0.476 | 6 | 157464165547 |
| **EC1** | 93 | 30 | 0.323 | - | Extended version of **C1** |
| **C2** | 255 | 131 | 0.514 | 18 | 215713331471510151261250277442142024165471 |
| **EC2** | 386 | 131 | 0.339 | - | Extended version of **C3** |

Although the original codes **C1** and **C2** are converted to **EC1** and **EC2**, the modification is small. Therefore the comparison between **EC1** and **TC1** as well as **EC2**

and **TC2** should still provide enough evidence on which accurate assessment can be made.

Before simulation can be done it is necessary to calculate the minimum storage requirement for the M-algorithm. First consider the code **EC1**. The value of $M$ for the M-algorithm for **EC1** will be chosen according to the criterion in Section 2.6. Although the extended code **EC1** is able to correct more errors than the original code **C1**, how many more is not known and not easy to find out. Therefore it is only required that **EC1** correct 6 errors, which is the same as the error correction capability of **C1**. The minimum value of $M$ for a 6 error correcting code with maximum local rate of 1/2 is $(2.41)^6$=196. Thus $M$ is chosen to be 200 for the code **EC1**. The simulation results along with that of the **TC1** are shown in Figure 4.3.1. The **TC1** is decoded by the MAP decoder and results after the tenth iteration are shown. Using the equations developed in Appendix E the computational complexity of the both decoders is listed in Table 4.3.2.

It can be noticed from the figure that the performance of **EC1** is very close to that of **TC1** at a SNR less than 4dB. When the SNR is more than 4dB, however, **EC1** catches up with **TC1** and becomes better when the SNR gets higher. Also notice that although the $d_{\min}$ error approximation in Figure 4.2.1 is for **C1**, it is very close to the simulated performance for **EC1**. Because the code structure of **EC1** is not much different from that of **C1** and their performances should be close to each other, it can be concluded the $d_{\min}$ error approximation for **C1** is accurate.

To make a comparison on the complexity one can again assume that a real number operation is about 60 times more complicated than an integer operation. From Table 4.3.2 it can be noticed that the total number of operations required for the M-algorithm to decode **EC1** is less than that for the MAP decoder with any iterations. Because

**EC1** has an error performance very close to that of the turbo code for the SNR less than 4dB and better performance for the SNR larger than 4dB, and because **TC1** is already a longer code than **EC1**, it can be concluded that 1) **EC1** is more attractive than **TC1**, and 2) the M-algorithm is an appropriate algorithm for decoding block codes of length less than 100.

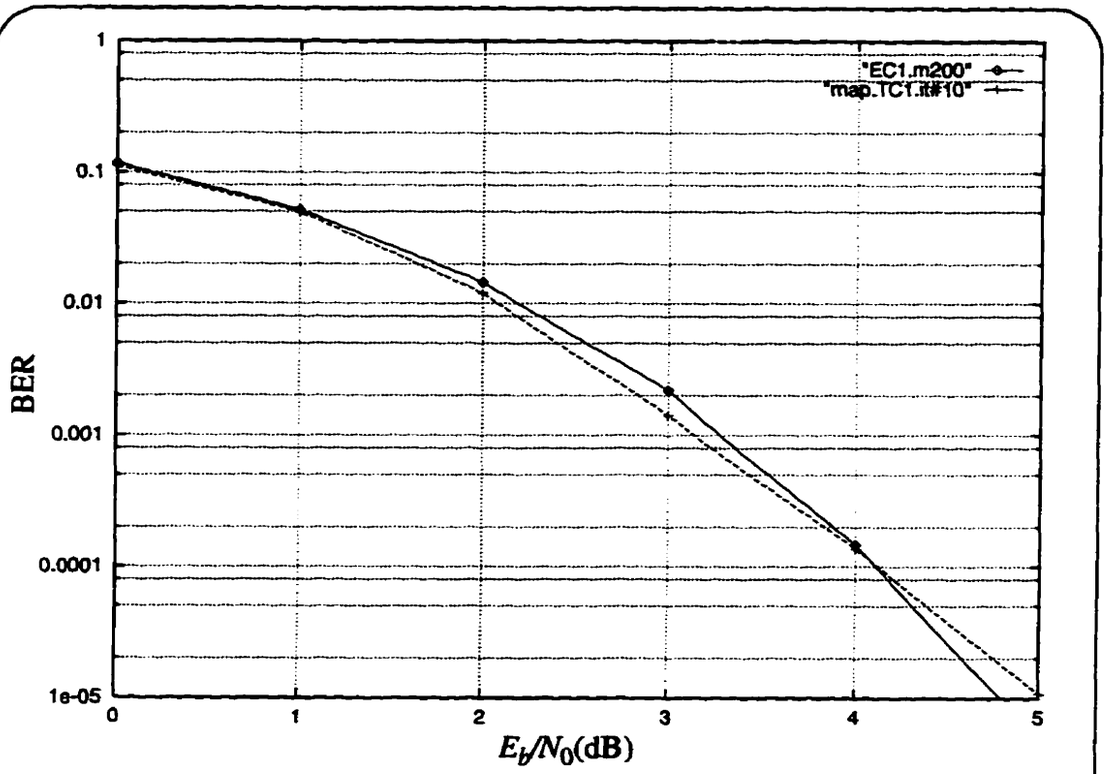


**Figure 4.3.1:** The error performance of the code **EC1** and **TC1**. **EC1** is decoded by the M-algorithm while **TC1** is decoded by the MAP decoder.

**TABLE 4.3.2:** A comparison of the computational complexity of the M-algorithm when it is used to decode the extended code **EC1** and that of the MAP decoder when it is used to decode **TC1**.

| | M-algorithm ($M$=200) (integer) | MAP (10 iterations) (real) | MAP ($n$ iterations) (real) |
|---|---|---|---|
| Table Look ups | 530 | 640 | $64n$ |
| Additions (Best Case) | 856 | 1280 | $128n$ |
| Additions (Worst Case) | 65034 | - | - |
| Multiplications | 0 | 1900 | $190n$ |

To see the performance of a longer code consider the code **EC2** and **TC2**. With similar argument as for **EC1**, **EC2** is chosen to be an 18 error correction code. In this case, however, the value of $M$ should be $(2.41)^{18}$=7521540. Since this is impractically large, two arbitrary values, $M$=200 and $M$=2000 are chosen respectively to see how **EC2** behaves for these two values of $M$. On the other hand, **TC2** is decoded by the MAP decoder with 10 iterations. The simulation results are shown in Figure 4.3.2 and the complexity in Table 4.3.3.

Notice from Figure 4.3.2 that for the SNR less than 4.5dB the extended code with both $M$=200 and $M$=2000 does not perform as well as the turbo code. It is only when the SNR is larger than 4.5dB that the extended code with $M$=2000 starts to catch up with its counterpart. However, neither of the results of the extended code are the full performance of **EC2** because to get that one needs to allow $M$ equal to at least 7521540. This suggests that the M-algorithm is not a suitable algorithm for decoding BCH codes longer than or as long as **EC2**. This in turn implies that if the full power of such long BCH codes

is required a new decoder with low enough complexity must be found. Based on the fact that a fraction of the full power of **EC2** is close to the performance of **TC2**, and that **TC2** is already a longer code than **EC2**, it can be speculated that a BCH code as long as a turbo code will eventually outperform the turbo code when a low complexity decoder is available for the long BCH codes. This speculation is also true for many other block codes because there are many block codes that are better than BCH codes in both performance and code structure.

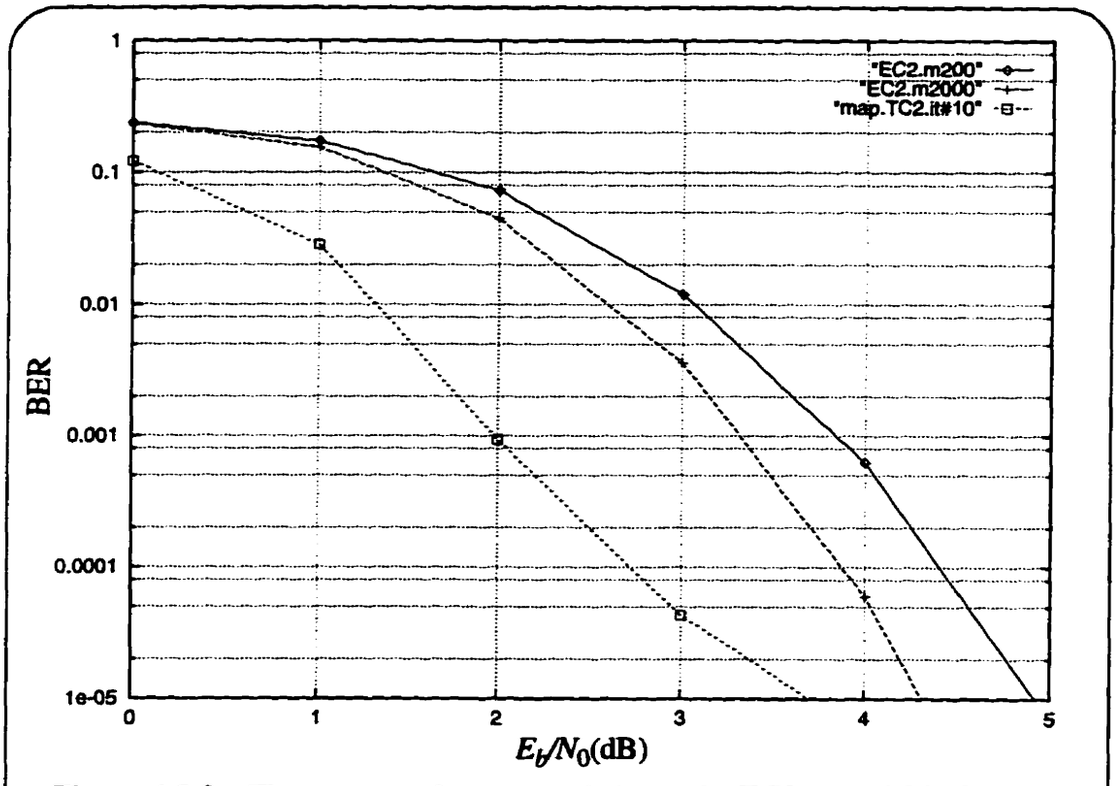


**Figure 4.3.2:** The error performance of the code **EC2** and **TC2**. **EC2** is decoded by the M-algorithm while **TC2** by the MAP decoder. For **EC2** the performance for $M$=200 and $M$=2000 are both shown in the figure.

On the other hand, based on Table 4.3.3, it can be noticed that the complexity of the M-algorithm with $M$=2000 is still less than that of the MAP decoder. Because

**EC2** out performs **TC2** when the SNR is larger than 4.5dB, the M-algorithm is still a suitable candidate as the decoding algorithm for BCH codes of similar length to **EC2** when the system is operating at a SNR larger than 4.5dB.

**TABLE 4.3.3:** A comparison of the computational complexity of the M-algorithm when it is used to decode the extended code **EC2** and that of the MAP decoder when it is used to decode **TC2**.

| | M-algorithm ($M$=2000) (integer) | MAP (10 iterations) (real) | MAP ($n$ iterations) (real) |
|---|---|---|---|
| Table Look ups | 5589 | 640 | $64n$ |
| Additions (Best Case) | 9328 | 1280 | $128n$ |
| Additions (Worst Case) | 7469367 | - | - |
| Multiplications | 0 | 1900 | $190n$ |

## 4.4 SUMMARY

As evident from the simulated performance of **EC1**, **EC2**, **TC1** and **TC2** the BCH code with the same block length as a turbo code would be more powerful than the turbo code. This is subject to the availability of a low complexity decoder that can be implemented for the long BCH codes. Therefore further research that looks for such a decoder would be beneficial. As for short BCH codes the M-algorithm is still appropriate. As reasoned before, the speculation made on BCH codes is also true for many other block codes. Therefore it can be speculated that a block code of similar length to that for a turbo code should out perform the turbo code when subject to maximum likelihood decoding.

# CHAPTER 5 CONCLUSIONS

The thesis can be basically divided into two parts. The first part is devoted to reduced complexity decoding of turbo codes while the second to comparison of block codes to turbo codes. In the first part of the thesis two reduced complexity decoders are investigated; they are respectively the MC decoder and MW decoder. Both decoders are designed based on the M-algorithm and therefore their error performance and the computational complexity are closely related to those of the M-algorithm. The MC decoder is particularly useful for turbo codes with interleaver size less than 300 bits. For these turbo codes the MC decoder is successful in reducing the complexity with a small degradation in error performance when compared to the MAP decoder. For turbo codes with interleaver size larger than 300 the exponential growth of the minimum storage requirement of the M-algorithm is so large that the overall complexity exceeds that of the MAP decoder. Therefore the MC decoder is not suitable for long turbo codes. The MW decoder is also only suitable for short turbo codes, especially those with interleaver of size less than 100. For these kinds of turbo code, the MW decoder has performance close to that of the MAP decoder. In terms of computational complexity, however, the MW decoder requirement is far simpler than the MAP decoder. The only drawback of the MW decoder is its exponential growth of the minimum storage requirement, making it impractical for longer turbo codes.

The investigation in the second part of the thesis is basically a partial answer to the question "If a block code has similar parameters to a turbo code will the

block code outperform the turbo code?" The partial answer is given by comparing short BCH codes to short turbo codes when they are subject to close to optimal decoding. The superior simulated performance and low decoding complexity of the short BCH codes when compared to the turbo codes suggest that the answer of the question is very likely positive. The full answer to the question will not be known until methods of decoding long block codes are found. However, these positive results should lead to motivation for further studies on looking for a practical decoder for long block codes.

# APPENDIX A: METRIC DERIVATION

## A.1 METRIC DERIVATION FOR THE SISOM ALGORITHM

In this section the metric for the SISOM algorithm used to decode the constituent code of a turbo code in a memoryless AWGN channel will be derived. Assume that none of the constituent encoders is terminated. Let the set of all possible input sequences of length $K$ be $\{\vec{d}_i, 1 \le i \le 2^K\}$ where $\vec{d}_i = (d_{i1}, d_{i2}, ..., d_{iK})$ is a $K$-bit sequence and each $d_{ik}$ is an i.i.d. random variable. Also let the received sequence be $\vec{R}=(X_1, Y_1, X_2, Y_2, ..., X_K, Y_K)$. In the maximum *a posteriori* criterion, it is necessary to find the sequence $\vec{d}_i$ that maximizes the *a posteriori* probability $P\left(\vec{d}_i|\vec{R}\right)$, which can be written as

$$P\left(\vec{d}_i|\vec{R}\right) = \frac{f(\vec{R}|\vec{d}_i)\,P(\vec{d}_i)}{f(\vec{R})}. \qquad \text{(A.1.1)}$$

Since $f(\vec{R})$ is constant for all $i$, it can be ignored in the metric calculation. Because the channel is memoryless the first term in the numerator on the RHS of (A.1.1) can be written as

$$f(\vec{R}|\vec{d}_i) = \prod_{j=1}^{K} f(X_j, Y_j|\,(d_{i1}, d_{i2}, ..., d_{iK}))\,. \qquad \text{(A.1.2)}$$

which can be further expanded as

$$f(\vec{R}|\vec{d}_i)=\prod_{j=1}^{K} f(X_j|\,(d_{i1}, d_{i2}, \ldots, d_{iK}))\cdot f(Y_j|\,(d_{i1}, d_{i2}, \ldots, d_{iK}))$$

$$= \prod_{j=1}^{K} \frac{1}{\sqrt{2\pi}\sigma}\exp\left(\frac{(X_j-\pm 1)^2}{2\sigma^2}\right)\cdot\frac{1}{\sqrt{2\pi}\sigma}\exp\left(\frac{(Y_j-\pm 1)^2}{2\sigma^2}\right) \qquad \text{(A.1.3)}$$

where $\sigma^2$ is the white noise variance and antipodal signalling of $\pm 1$ is assumed. For the second factor in the numerator on the RHS of (A.1.1), because the elements in the sequence $\vec{d}_i$ are i.i.d., the probability can be rewritten as

$$P(\vec{d}_i) = \prod_{j=1}^{K} P(d_{ij})\,. \qquad \text{(A.1.4)}$$

This term represents the *a priori* information available to the decoder. For the first constituent code in the first iteration, all information bits $d_k$ is equally likely and therefore

$$P(d_{ij}=0) = P(d_{ij}=1) = \frac{1}{2} \text{ for } 1\le j\le K \text{ and } 1\le i\le 2^K.$$

However, for the first constituent code in all other iterations and for the second constituent code in all iterations, the *a posteriori* information produced by the previous decoder can be used as the current *a priori* information and (A.1.4) can be written as

$$P(\vec{d}_i) = \prod_{j=1}^{K} P(d_{ij}|\text{previous decoder})\,. \qquad \text{(A.1.5)}$$

Combining (A.1.3) and (A.1.5), taking the log of it and ignoring the constant terms, the path metric for path $i$ can be written as

$$PM_i = \sum_{j=1}^{K}\left(\frac{(X_j-\pm 1)^2}{2\sigma^2}\frac{(Y_j-\pm 1)^2}{2\sigma^2} + \log P(d_{ij}|\text{previous decoder})\right). \qquad \text{(A.1.6)}$$

From (A.1.6) it can be noticed that the branch metric for path $i$ at time $j$ is

$$BM_{i,j} = \frac{(X_j - \pm 1)^2}{2\sigma^2} \frac{(Y_j - \pm 1)^2}{2\sigma^2} + \log P\,(d_{ij} | \text{previous decoder})$$

$$= -\left( \frac{(X_j - \pm 1)^2}{2\sigma^2} + \frac{(Y_j - \pm 1)^2}{2\sigma^2} \right) + \log P\,(d_{ij} | \text{previous decoder}) \,. \qquad \text{(A.1.7)}$$

## A.2 METRIC DERIVATION FOR THE M-ALGORITHM APPLIED TO TURBO CODES

In this section the metric for the M-algorithm applied to decode the overall turbo code in a memoryless AWGN channel will be derived. Assume the same setting as in Section A.1 except that the first constituent encoder is terminated with a tail of $v$ bits and that the received sequence is $\vec{R}=(X_1, Y_1, Y_1', X_2, Y_2, Y_2', ..., X_{K+v}, Y_{K+v}, Y_{K+v}')$. Then in maximum *a posteriori* sense, it is necessary to find the sequence $\vec{d}_i$ that maximizes the probability $P\left(\vec{d}_i | \vec{R}\right)$, which can be written as

$$P\left(\vec{d}_i | \vec{R}\right) = \frac{f(\vec{R} | \vec{d}_i)\, P\,(\vec{d}_i)}{f(\vec{R})} . \qquad \text{(A.2.1)}$$

Since both $f(\vec{R})$ and $P\,(\vec{d}_i)$ are constant for all $i$, they can be ignored in the metric calculation. Following the same argument as in Section A.1, the final path metric can be found and for path $i$ it is equal to

$$PM_i = \sum_{j=1}^{K+v} \left( -\frac{(X_j - \pm 1)^2}{2\sigma^2} - \frac{(Y_j - \pm 1)^2}{2\sigma^2} - \frac{(Y_j' - \pm 1)^2}{2\sigma^2} \right). \qquad \text{(A.2.2)}$$

Therefore the branch metric for path $i$ at time instant $j$ is

$$BM_{i,j} = -\frac{(X_j - \pm 1)^2}{2\sigma^2} - \frac{(Y_j - \pm 1)^2}{2\sigma^2} - \frac{(Y_j' - \pm 1)^2}{2\sigma^2}$$
$$= -\left( \frac{(X_j - \pm 1)^2}{2\sigma^2} + \frac{(Y_j - \pm 1)^2}{2\sigma^2} + \frac{(Y_j' - \pm 1)^2}{2\sigma^2} \right). \qquad \text{(A.2.3)}$$

## A.3 METRIC DERIVATION FOR THE M-ALGORITHM APPLIED TO EXTENDED BLOCK CODES

In this section the metric for the M-algorithm used to decode the extended block code in a memoryless AWGN channel will be derived. Assume the same setting as in Section A.2 except that the received sequence is $\hat{R}=(C_1, D_1, C_2, D_2, ..., C_{K+\nu}, D_{K+\nu})$ where all $D_{K+1}, ..., D_{K+\nu}$ are -1. In a maximum *a posteriori* sense, the path metric for path $i$ is

$$PM_i = \sum_{j=1}^{K+\nu} \left( -\frac{(C_j - \pm 1)^2}{2\sigma^2} - \frac{(D_j - \pm 1)^2}{2\sigma^2} \right). \qquad \text{(A.3.1)}$$

From this the branch metric for path $i$ at time instant $j$ is

$$BM_{i,j} = -\frac{(C_j - \pm 1)^2}{2\sigma^2} - \frac{(D_j - \pm 1)^2}{2\sigma^2}$$
$$= -\left( \frac{(C_j - \pm 1)^2}{2\sigma^2} + \frac{(D_j - \pm 1)^2}{2\sigma^2} \right). \qquad \text{(A.3.2)}$$

# APPENDIX B: DEFINITION OF THE SIGNAL-TO-NOISE RATIO

## B.1 DEFINITION

Before simulation can be started one has to choose a proper definition for the signal-to-noise ratio because if a different definition is used, the simulations will produce different results. For the thesis the most common one is chosen so that the results in the thesis can be compared to those in other literature. The definition chosen for the SNR is $E_b/N_0$, which is the energy-per-information-bit-to-one-sided-power-spectral-density ratio. Thus

$$SNR \triangleq \frac{E_b}{N_0} \tag{B.1.1}$$

In the thesis binary coded systems with antipodal signalling of $\pm 1$ operating in an AWGN channel are considered. In particular, when the code is of rate $R$, then

$$\frac{E_b}{N_0} = \frac{1}{2R\sigma^2} \tag{B.1.2}$$

where $\sigma^2$ is the variance of the additive white Gaussian noise.

In the simulation $E_b/N_0$ is a parameter one would like to vary, say from 0dB to 8dB. With (B.1.2) one can calculate the corresponding value of $\sigma^2$, which is

$$\sigma^2 = \frac{1}{2R\left(10^{\frac{\left(\frac{E_b}{N_0}\right)_{dB}}{10}}\right)}. \qquad \text{(B.1.3)}$$

Then the variance of the Gaussian random number generator in the computer program can be set to $\sigma^2$ to get the correct noise value which can then be added to the transmitted signal (either -1 or +1) and the results will be the final sampled, matched-filter output at the receiver.

# APPENDIX C: OCTAL REPRESENTATION OF GENERATOR POLYNOMIALS

## C.1 OCTAL REPRESENTATION TO POLYNOMIAL REPRESENTATION

The generator polynomials found in most textbooks are represented in an octal representation. The example shown in Figure C.1.1 shows how the octal representation can be converted back into a polynomial form.

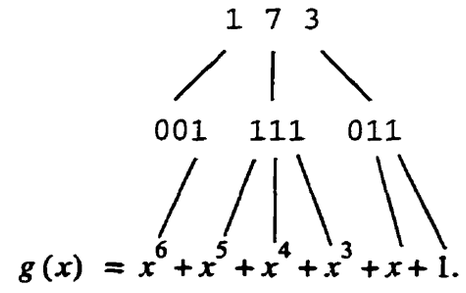


**Figure C.1.1:** This example illustrates how the octal representation 173 represents a generator polynomial $g(x)=x^6+x^5+x^4+x^3+x+1$.

# APPENDIX D: SOFT DECISION DECODING OF BLOCK CODES WITH M-ALGORITHM

## D.1 INTRODUCTION

There have been several attempts in the past to accomplish soft decision decoding of block codes [4][24][33][35][36]. Soft decision decoding here means that the decoder accepts continuous values from the channel and produces hard outputs (0 or 1) as the estimate. In [4], [33], [35] and [36] a trellis of the block code is constructed and standard trellis search algorithms are used to decode the block code. The trellis representation used in these papers follows from the method proposed in [36], which is called Wolf's trellis representation. The search algorithms used in these papers are respectively, the BCJR algorithm [4], simplified Viterbi algorithm [33] and the full Viterbi algorithm [36]. One major disadvantage of the Wolf's trellis representation is that the resultant trellis is more irregular when compared to that of a convolutional code and thus more difficult to implement in both software and hardware. In [35] another trellis representation for block codes is proposed by D. Tempel and E. Shwedyk. Their approach is first to convert a block code encoder to an equivalent feed-forward shift register, which basically can be considered as a convolutional encoder. Except for the branch labelling, the resultant trellis is identical to that of a convolutional code of the same memory order. In [35] the sequential stack algorithm is applied to search through the trellis. In the following Tempel and

Shwedyk's approach is used to construct the encoder of a block code. Then the encoder is modified so that the code tree is suitable for the M-algorithm to search through. From this modified encoder the code tree will be constructed and the M-algorithm will be used to decode the code.

## D.2 CONSTRUCTION OF BLOCK CODE ENCODERS

In the thesis binary BCH codes are of interest and they belong to the class of linear cyclic codes. For any $(N, K)$ linear cyclic code with generator polynomial

$$g(x) = g_0 + g_1 x + g_2 x^2 + \ldots + g_{N-K} x^{N-K}, \qquad \text{(D.2.1)}$$

a feed-forward shift register which produces an equivalent set of codewords to the traditional feedback encoder can be constructed and is shown in Figure D.2.1.

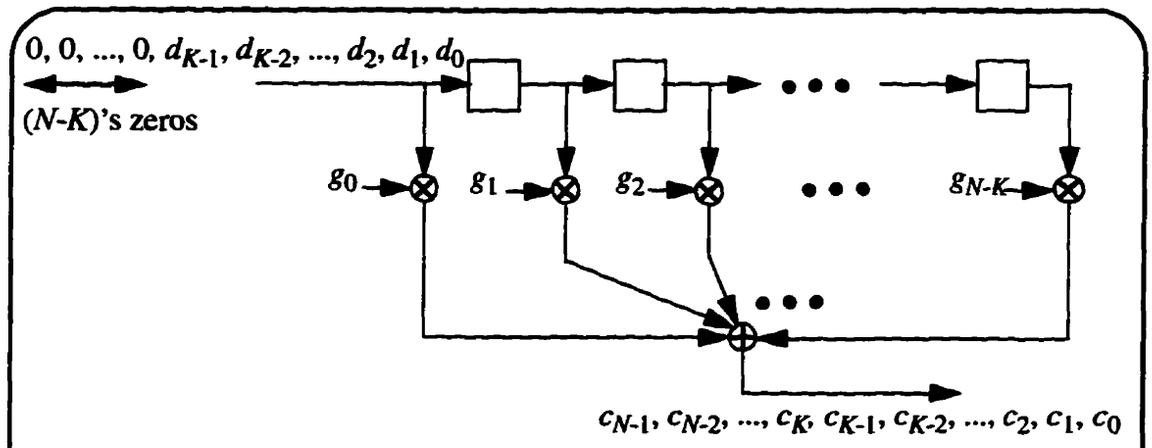


**Figure D.2.1:** Feedforward encoder for a linear cyclic code with generator polynomial $g(x)$.

To encode an information sequence $(d_0, d_1, \ldots, d_{K-1})$ it is necessary to initialize all the elements of the shift register to zero. Then starting from $d_0$ each bit of the information sequence is shifted into the shift register and at the same time the correspond-

ing codeword bit from the output of the summation circuit is collected. After the information sequence is finished a sequence of $(N\text{-}K)$'s zeros is shifted into the shift register to flush out the remaining content of the shift register. The remaining content constitutes the codeword bits $c_K$, ...,$c_{N-2}$ and $c_{N-1}$. After all codeword bits have been collected the encoding for one information sequence is complete.

## D.3 ENCODER MODIFICATION

Consider an $(N, K)$ block code where $N = 2K$, i.e., it is a rate 1/2 code. The code **C1** and **C2** used in Chapter 4 are such examples. For this code the encoder in Figure D.2.1 will generate a code tree of local rate 1 even though the global rate is 1/2. In order for the M-algorithm to work properly, it is necessary to take into account the highest local rate of the tree to calculate the minimum storage requirement for the algorithm. The reason for this is that M-algorithm is not a backtracking algorithm; it has to take precaution from the beginning so that even if unfavourable situations happen at any time the algorithm is still able to track the correct path. According to Table 2.6.1, however, the formula for the minimum storage requirement of the M-algorithm is not valid any more because the formula gives infinity as the result when the rate of 1 is substituted into the formula. The problem for a rate 1 code is that the redundant information is not enough at each time instant for the M-algorithm to accumulate enough distance to separate the correct and incorrect paths. To overcome this problem one has to introduce more redundant information at each step of the tree; therefore the encoder discussed in the previous section has to be modified. As a result of the modification, both the local rate and the global rate of the code will be decreased.

One simple way to modify the encoder is to shift the information bits into the channel at the same time as they are shifted into the encoder. The structure of the new encoder is shown in Figure D.3.1. Increasing the length of a code by introducing more redundancies is called extending the code [21]. The extended code now has a global rate equal to $\frac{K}{N+K} \approx \frac{1}{3}$ and a local rate equal to $\frac{1}{2}$. The encoded bit $c_k'$ of the extended code-word is now equal to

$$c_0, d_0, c_1, d_1, c_2, d_2, ..., c_{K-2}, d_{K-2}, c_{K-1}, d_{K-1}, c_K, c_{K+1}, ..., c_{N-2}, c_{N-1}.$$

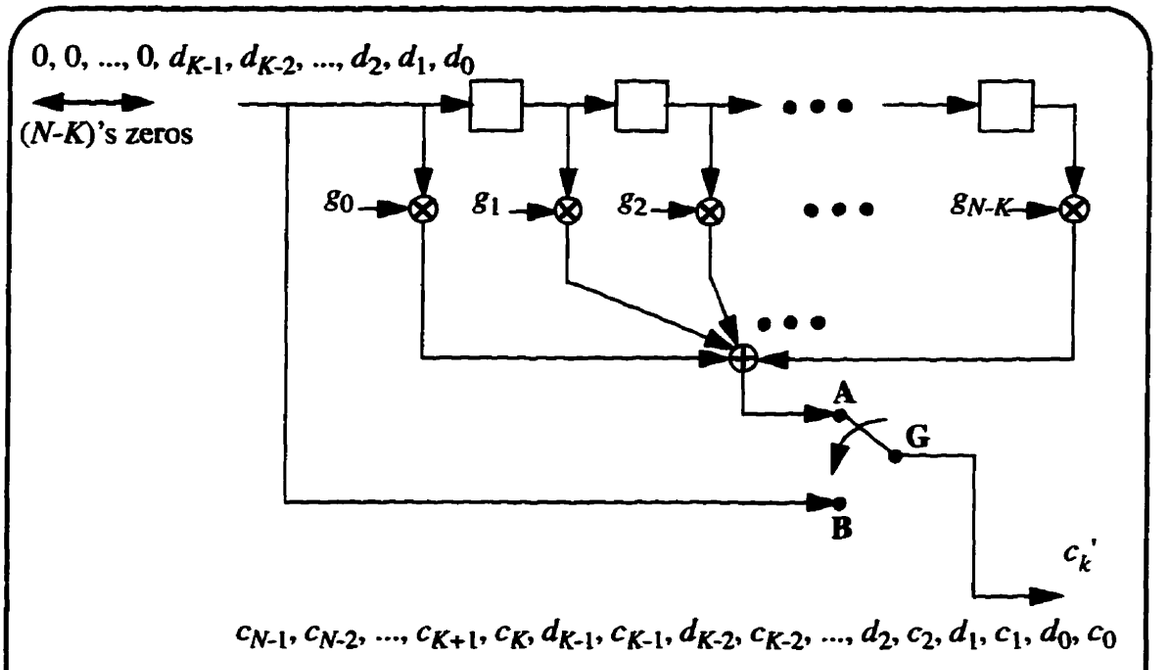


**Figure D.3.1:** The modified encoder for a linear cyclic block code with generator polynomial $g(x)$. Gate **G** will collect encoded bits from both **A** and **B** in each of the first $K$ time instants. In the final $N$-$K$ time instant when the flushing bits are entering the encoder gate **G** will only collect encoded bits from **A**.

It is necessary to have an appropriate decoder to work with the modified encoder. The structure of the new decoder is shown in Figure D.3.2.

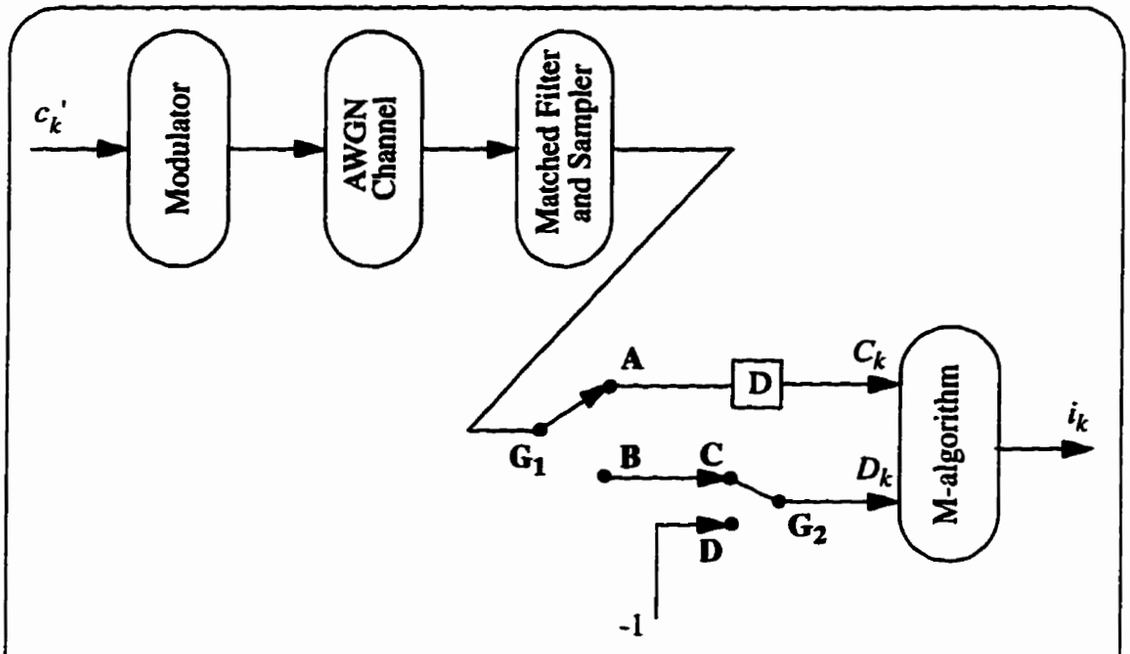


**Figure D.3.2:** The decoder to be used with the modified encoder. For the first $2K$ samples Gate $G_1$ will alternate between **A** and **B** at every other time step. For the last $N$-$K$ samples $G_1$ will stay at **A** only. Moreover, $G_2$ will stay at **C** for the first $2K$ samples and at **D** for the final $N$-$K$ samples.

In order to perform the simulation one should have an appropriate metric. For the extended version of the BCH code the appropriate metric for the path $i$ at time instant $k$ is derived in Section A.3 of Appendix A and is given below:

$$BM_{i,k} = -\left( \frac{(C_k - \pm 1)^2}{2\sigma^2} + \frac{(D_k - \pm 1)^2}{2\sigma^2} \right). \qquad \text{(D.3.1)}$$

# APPENDIX E: COMPUTATIONAL COMPLEXITY OF THE EXTENDED BCH CODES WHEN DECODED BY THE M-ALGORITHM

## E.1 HARDWARE COMPUTATIONAL COMPLEXITY

The complexity of the M-algorithm on the extended BCH codes in terms of the number of operations will be calculated in this section. By following the conventions in [1] and [32] one can compare the results with those of the MAP decoder.

Consider an ($N$, $K$) block code **C** and its extended version **EC**, which is another ($N$+$K$, $K$) code. Although the codeword length of the extended code is $N$+$K$, its code tree is still $N$ steps long. Note that when M-algorithm is applied to decode the block codes, integer arithmetics can be used to approximate the metric calculations.

With $\nu$=$N$-$K$ the expressions in Section 3.8 can be used again for the block codes. In particular, the number of table look-ups per information bit due to metric calculations is

$$N_{T/bit} = \frac{2^{j+2}-2+2M(K-j-1)+(N-K)M}{K} \qquad \text{(E.1.1)}$$

in which $j = \left\lceil \log_2 \frac{M}{2} \right\rceil$ where ($j$+1) is the level of the tree wide enough to have at least $M$ paths.

Similarly one can show that the number of best case additions per informa-

tion bit required to calculate all the path metrics is

$$(N_{A/bit})_B = \frac{2^{j+2} + (4M-1)(K-j-1) + M(5+N-K) - 5}{K}. \qquad \text{(E.1.2)}$$

and the number of worst case additions per information bit is

$$(N_{A/bit})_W = \frac{2^{j+2} + M(2M+1)(K-j-1) + 2M(2M-1) + \frac{M}{2}(M-1) + M(N-K) - 2}{K} \qquad \text{(E.1.3)}$$

As argued in Section 3.8, one can say that (E.1.2) is more likely to reflect the actual number of additions to be done in the M-algorithm.

Based on (E.1.2) to (E.1.3) and (3.4.10) to (3.4.12), one can make a comparison of the M-algorithm and several iterations of the MAP decoder.

## E.2 NUMBER OF NODES VISITED

Just as what was done in Chapter 3, one can make a comparison on the number of nodes visited by the two decoders. In particular, one can calculate how many nodes the M-algorithm visited per information bit in order to decode the extended code and compare it to that of the MAP decoder when the MAP decoder is used to decode a comparable turbo code.

Consider an $(N, K)$ block code **C** and its extended version **EC**, which is another $(N+K, K)$ code. Although the codeword length of the extended code is $N+K$, its code tree is still $N$ steps tall. Therefore the total number of nodes visited by the M-algorithm per information bit in order to decode **EC** is

$$N_{V/bit} = \frac{2+2^2+\ldots+2^{j+1}+2M(K-j-1)+(N-K)M}{K}$$

$$= \frac{2^{j+2}-2+2M(K-j-1)+(N-K)M}{K} \qquad \text{(E.2.1)}$$

in which $j = \left\lceil \log_2 \frac{M}{2} \right\rceil$ where $(j+1)$ is the level of the tree wide enough to have at least $M$ paths.

For the MAP decoder one can use (3.9.4) to calculate the number of nodes visited by the decoder per information bit. For the extended BCH codes and turbo codes used in the simulations in Section 4.3, the number of nodes visited by the M-algorithm and the MAP decoders are summarized in Table E.2.1 and Table E.2.2.

**TABLE E.2.1:** The number of nodes visited by the M-algorithm when it is used to decode the extended BCH code **EC1** and the MAP decoder when it is used to decode **TC1**.

| | M-algorithm ($M$=200) (integer) | MAP (10 iterations) (real) | MAP ($n$ iterations) (real) |
|---|---|---|---|
| # of Nodes Visited/ Information Bit | 530 | 640 | $64n$ |

**TABLE E.2.2:** The number of nodes visited by the M-algorithm when it is used to decode the extended BCH code **EC2** and that of the MAP decoder when it is used to decode **TC2**.

| | M-alg ($M$=200) (integer) | M-alg ($M$=2000) (integer) | MAP (10 iterations) (real) | MAP ($n$ iterations) (real) |
|---|---|---|---|---|
| # of Nodes Visited/ Information Bit | 569 | 5589 | 640 | $64n$ |

It can be noticed from both tables that the number of nodes visited by the M-algorithm is more than that of the MAP decoder. However, since the number of operations required at each node is different for each decoder, the results in Table E.2.1 and Table E.2.2 can only be used as a secondary measure of complexity.

# BIBLIOGRAPHY


[1] J.D.Andersen, "Turbo Coding for Deep Space Applications", Proceedings of 1995 IEEE International Symposium on Information Theory, pp.36, September 1995.

[2] J.B.Anderson and V.Franz, "Reduced-Search BCJR Algorithm", Proceedings of 1997 IEEE International Symposium on Information Theory, pp.230, July 1997.

[3] J.B.Anderson and S.Mohan, "Source and Channel Coding: An Algorithmic Approach", Kluwer Academic Publishers, Norwell, MA, 1991.

[4] L.R.Bahl, J.Cocke, F.Jelinek and J.Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate", IEEE Transactions on Information Theory, vol.IT-20, pp.284-287, March 1974.

[5] A.S.Barbulescu and S.S.Pietrobon, "Terminating the Trellis of Turbo-Codes in the Same State", Electronic Letters, vol.31, no.1, pp.22-23, January 1995.

[6] A.S.Barbulescu and S.S.Pietrobon, "Interleaver Design for Turbo Codes", Electronic Letters, vol.30, no.25, pp.2107-2108, December 1994.

[7] G.Battail, "Construction explicite de bons codes longs", Annales des Telecommunications, vol.44, nos.7-8, pp.392-404, 1989.

[8] G.Battail, Claude Berrou and Alain Glavieux, "Pseudo-Random Recursive Convolutional Coding for Near-Capacity Performance", Proceedings of GLOBECOM'93 Communication Theory Mini-Conference, pp.23-27, Houston, TX, 1993.

[9] S.Benedetto and G.Montorsi, "Unveiling Turbo Codes: Some Results on Parallel Concatenated Coding Schemes", IEEE Transactions on Information Theory, vol.IT-42, pp.409-428, March 1996.

[10] S.Benedetto and G.Montorsi, "Design of Parallel Concatenated Convolutional Codes", IEEE Transactions on Communications, vol.44, pp.591-600, May 1996.

[11] C.Berrou, A.Glavieux and P.Thitimajshima, "Near Shannon Limit Error - Correcting Coding and Decoding: Turbo-Codes", Proceedings of ICC'93, pp.1064-1070, May 1993.

[12] W.J.Blackert, E.K.Hall and S.G.Wilson, "Turbo Code Termination and Interleaver Conditions", Electronics Letters, vol.31, no.24, pp.2082-2084, November 1995.

[13] W.J.Blackert, E.K.Hall and S.G.Wilson, "An Upper Bound on Turbo Code Free Distance", Proceedings of ICC'96, pp.957-961, Dallas, TX, 1996.

[14] D.Divsalar and F.Pollara, "Turbo Codes for PCS Applications", Proceedings of ICC'95, pp.54-59, June 1995.

[15] D.Divsalar and R.J.McEliece, "Effective Free Distance of Turbo Codes", Electronic Letters, vol.32, no.5, pp.445-446, February 1996.

[16] D.Haccoun and M.J.Ferguson, "Generalized Stack Algorithms for Decoding Convolutional codes", IEEE Transactions on Information Theory, vol.IT-21, pp.638-651, November 1975.

[17] J.Hagenauer, E.Offer and L.Papke, "Iterative Decoding of Binary Block and Convolutional Codes", IEEE Transactions on Information Theory, vol.IT-42, pp.429-445, March 1996.

[18] O. Joerssen and H.Meyr, "Terminating the Trellis of Turbo-Codes", Electronic Letters, Vol.30, no.16, pp.1285-1286, August 1994.

[19] P.Jung, "Comparison of Turbo-Code Decoders Applied to Short Frame Transmission Systems", IEEE Journal on Selected Areas in Communications, vol.14, no.3, pp.530-537, April 1996.

[20] K.S.Lin, "Digital Signal Processing Applications with the TMS320 Family, Volume I", Prentice Hall, Englewood Cliffs, NJ, 1987.

[21] S.Lin and D.J.Costello JR., "Error Control Coding: Fundamentals and Applications", Prentice-Hall, Englewood Cliffs, NJ, 1983.

[22] E.Luthi and E.Casseau, "High Rate Soft Output Viterbi Decoder", Proceedings of European Design & Test Conference ED&TC'96, pp.315-319, 1996.

[23] F.J.MacWilliams and N.J.A.Sloane, "The Theory of Error-Correcting Codes", North-Holland Publishing Company, New York, NY, 1977.

[24] K.R.Matis and J.W.Modestino, "Reduced-Search Soft-Decision Trellis Decoding of Linear Block Codes", IEEE Transactions on Information Theory, vol.IT-28, pp.349-355, March 1982.

[25] M.Moher, "Decoding via Cross-Entropy Minimization", Proceedings of GLOBECOM'93, Houston, TX, pp.809-813, November 1993.

[26] L.C.Perez, J.Seghers and D.J.Costello, "A Distance Spectrum Interpretation of Turbo Codes", IEEE Transaction on Information Theory, vol.IT-42, pp.1698-1709, November 1996.

[27] S.S.Pietrobon, "Implementation and Performance of a Serial MAP Decoder for Use in an Iterative Turbo Decoder", Proceedings of 1995 IEEE International Symposium on Information Theory, pp.471, Whistler, BC, September 1995.

[28] R.Podemski, W.Holubowicz, C.Berrou and G.Battail, "Hamming Distance Spectra of Turbo Codes", Annales des Telecommunications, vol.50, nos.9-10, pp.790-797, 1995.

[29] R.Rivest, C.E.Leiserson and T.H.Cormen, "Introduction to Algorithms", The MIT Press, Cambridge MA, 1994.

[30] P.Robertson, "Illuminating the Structure of Code and Decoder of Parallel Concatenated Recursive Systematic (Turbo) Codes", Proceedings of GLOBECOM'93, pp.1298-1304, November 1994.

[31] P.Robertson and T.Worz, "Coded Modulation Scheme Employing Turbo Codes", Electronic Letters, vol.31, no.18, pp.1546-1547, August 1995.

[32] J.Seghers, "On the Free Distance of TURBO Codes and Related Product Codes", Final Report, Diploma Project SS 1995, no.6613, Swiss Federal Institute of Technology, Zurich, Switzerland, August 1995.

[33] J.Snyders and Y.Be'ery, "Maximum Likelihood Soft Decoding of Binary Block Codes and Decoders for the Golay Codes", IEEE Transactions on Information Theory, vol.IT-35, September 1989.

[34] Y.V.Svirid, "Weight Distributions and Bounds for Turbo-Codes", European Transactions on Telecommunications, vol.6, pp.543-556, October 1995.

[35] D.Tempel and E.Shwedyk, "Simple Trellis Encoder for Linear Block Codes", Proceedings of 1994 International Symposium on Information Theory and its Applications, pp.29-30, November 1994.

[36] J.K.Wolf, "Efficient Maximum Likelihood Decoding of Linear Block Codes Using a Trellis", IEEE Transactions on Information Theory, vol.IT-24, pp.76-80, January 1978.

[37] J.Zhou, Private Communication, July 1997.

[38] Texas Instruments, "TMS320 User's Guide", Digital Signal Processor Products, Texas Instrument, 1989.

[39] R.J.McEliece, E.R.Rodemich and J.Cheng, "The Turbo Decision Algorithm", Proceedings of 33rd Allerton Conference on Communications, Control and computing, pp.366-379, Monticello IL, August 1995.