Symbol-Based Turbo Codes for Wireless Communications

by

Mark Bingeman

A thesis

presented to the University of Waterloo

in fulfilment of the

thesis requirement for the degree of

Master of Applied Science

in

Electrical and Computer Engineering

Waterloo, Ontario, Canada, 1998

I hereby declare that I am the sole author of this thesis.

I authorize the University of Waterloo to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the University of Waterloo to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

The University of Waterloo requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

This thesis studies a new method of combining Turbo Codes with various modulation techniques to take advantage of inherent trade-offs between BER performance, code rate, spectral efficiency and decoder complexity. Our new method, which we call *Symbol-Based Turbo Codes*, parses the parallel data streams of the Turbo Code encoder into n-bit *symbols* and maps each symbol to a point in a 2^n-ary signal set. In the case of *Symbol-Based Turbo Codes* with BPSK modulation, the BER performance can be improved while at the same time decreasing the decoder complexity as compared with the traditional Turbo Code.

Symbol-Based Turbo Codes are good candidates for spread spectrum communication systems such as CDMA. Specifically, the Symbol-Based Turbo Code with orthogonal modulation can be used for a non-coherent up-link, or bi-orthogonal modulation for a coherent up-link. Furthermore, Symbol-Based Turbo Codes with either M-ary PSK modulation or BPSK modulation can be used for a coherent down-link.

Acknowledgements

I would like to acknowledge my supervisor Dr. A. K. Khandani for his guidance throughout my research. Thank-you for sharing your insight and enthusiasm about the use of Turbo Codes in PCS applications.

I would like to thank NSERC, ICR and the University of Waterloo for their financial support.

I would also like to thank all my friends at the University of Waterloo Inter-Varsity Christian Fellowship. Your friendship and encouragement has meant a lot to me throughout my years at university.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In a digital wireless communication system, the purpose of the channel code is to add redundancy to the binary data stream to combat the effect of signal degradation in the channel. Ideally, channel codes should meet the following requirements:

1. Channel codes should be high rate to maximize data throughput.
2. Channel codes should have good Bit Error Rate (BER) performance at the desired Signal-to-Noise Ratio (SNR) to minimize the energy needed for transmission.
3. Channel codes should have low encoder/decoder complexity to limit the size and cost of the transceivers.
4. Channel codes should introduce only minimal delays, especially in voice transmission, so that no degradation in signal quality is detectable.

These requirements are very difficult to obtain simultaneously, excellent performance in one requirement usually comes at the price of reduced performance in another.

In 1993, Claude Berrou *et. al* introduced a class of parallel concatenated convolutional codes, known as *Turbo Codes* [1]. They claimed a BER performance of 10^{-5} at an SNR of 0.7 dB for a rate 1/2 binary code. This performance, which was only 0.7 dB from the Shannon limit[1], was a large improvement over other coding methods at the time. Initially, Turbo Codes were received by the coding community with a great deal of skepticism, until other researches were able to reproduce their results [3, 4].

Some researchers have examined combining Turbo Codes with various modulation techniques. However, the majority of these studies focus on high rate, spectrally efficient codes. For example, in 1994, S. Le Goff, A. Glaviux and C. Berrou presented a rate $R = (m - \tilde{m})/m$ Turbo Code in conjunction with M-ary Phase Shift Keying (PSK) or Quadrature Amplitude Modulation (QAM) schemes [5]. These codes processed $m - \tilde{m}$ bits from each of the 3 parallel data stream of the Turbo Code, and punctured the parity bits appropriately before mapping the codeword to a 2^m point PSK or QAM signal set. These codes were decoded using soft demodulation followed by Turbo decoding.

Similarly, R. Pyndiah, A. Picart, and A. Glavieux, applied Turbo decoding to product codes with 16-QAM and 64-QAM modulation to obtain high rate, spectrally efficient codes [6]. For spectral efficiencies greater than 4 bits/s/Hz, their

[1]Claude Shannon originally presented theoretical limits on the performance of a communication system in [2].

codes performed better than the convolutional Turbo Codes with QAM presented by Le Goff *et. al.* This is due to the degradation in the performance of convolutional Turbo Codes when the code is punctured to increase its rate.

Benedetto *et. al* presented a combination of Trellis Coded Modulation (TCM) and Turbo Codes for high rate, spectrally efficient codes in [7]. These codes are formed by a parallel concatenation of two rate $b/(b+1)$ constituent codes which have b systematic bits and one parity bit generated by a recursive convolutional encoder. The outputs of each constituent code are punctured to obtain the desired over-all code rate. 8-PSK, 16-QAM, or 64-QAM modulation schemes are then used. Benedetto *et. al* reported BER performances within 1 dB of the Shannon limit for both 2 and 4 bits/s/Hz throughputs. This was the best BER performance for these throughputs at that time.

The low rate Turbo coding methods primarily focus on deep-space communications and spread spectrum communications where high coding gains are required and lower throughputs can be tolerated. Divsalar and Pollara increased the coding gain of Turbo Codes by parallel concatenating multiple (more than two) Recursive Systematic Convolutional (RSC) encoders for deep-space applications [8]. These codes, which the authors called, *Multiple Turbo Codes*, perform better than the traditional Turbo Codes in terms of BER performance for moderate block sizes (i.e. $N = 4096$) and rates (i.e. $R = 1/4$). For smaller block sizes (i.e. $N = 192$), Multiple Turbo Codes perform worse than the traditional Turbo Codes. Therefore, Multiple Turbo Codes are appropriate for deep-space communications where long delays are tolerable, but not for spread spectrum voice communications where long delays cannot be tolerated.

Other low rate Turbo Coding methods consist of combining Turbo Codes with orthogonal modulation. These codes are mainly suited for spread spectrum communication methods such as Code Division Multiple Access (CDMA) systems where lower code rates can be accommodated in the large spreading factor. Pehkonen and Komulainen presented super-orthogonal Turbo Codes in [9]. In these codes, the state of the RSC encoder and the current input bit are used to select a row of a Walsh-Hadamard matrix (or its complement). The desired code rate is obtained by puncturing on an orthogonal symbol basis (i.e. certain orthogonal symbols are not transmitted).

Naftaili Chayat combined Turbo Codes with orthogonal signaling by augmenting the RSC encoders with more feed-forward outputs. The n parallel outputs of the RSC encoders are mapped to the points of a 2^n-ary orthogonal signal set. As with the previous coding method, the desired rate is obtained by puncturing on an orthogonal symbol basis [10].

This thesis studies a new method of combining Turbo Codes with various modulation techniques. Unlike the codes discussed previously, our codes modulate each parallel data stream independently by parsing each data stream into n-bit sub-blocks which we call *symbols*, and mapping each symbol to a point in a 2^n-ary signal set. Furthermore, the interleaver is restricted to permute sub-blocks. With this restriction in place, the effective encoder operates on a symbol-by-symbol basis. We call these codes *Symbol-Based Turbo Codes*. Trade-offs between the BER performance, code rate, spectral efficiency, and decoder complexity can be made by the selection of different symbol sizes, n, and modulation techniques. In the case of Symbol-Based Turbo Codes with BPSK modulation, the BER performance can be improved while at the same time decreasing the decoder complexity as compared with the traditional Turbo Code. These codes are the subject of a North American patent and have been proposed by NORTEL for integration into the emerging 3G-IS95 standard.

The rest of this thesis is organized as follows: Chapter 2 gives an overview of Turbo Codes and other related background theory. Chapter 3 presents the *Symbol-Based Turbo Code* encoder and decoder structures. Chapter 4 describes the simulation setup. Chapter 5 presents the numerical results of the simulations as well as a discussion on decoder complexity. Finally, Chapter 6 concludes with a summary of results and suggests some future research directions.

Chapter 2

Turbo Codes

2.1 Linear Feedback Shift Register Sequences

2.1.1 Binary Linear Feedback Shift Registers

The Turbo Code encoder is built using Recursive Systematic Convolutional (RSC) encoders. Since RSC codes are governed by Linear Feedback Shift Register (LFSR) theory, we will review the aspects of LFSR sequences which are relevant to Turbo Codes. An extensive study on shift register sequences was made by Golomb in [11].

A binary LFSR is an arrangement of n memory elements which each store a binary variable, $\{0,1\}$. The basic structure of an LFSR is outlined in Figure 2.1. At each step in the sequence, the value in each memory element is shifted one element to the right, and the left-most memory element is calculated as a linear combination of the values in the memory elements during the previous step.

Collectively, the values stored in all the memory elements is called the state of

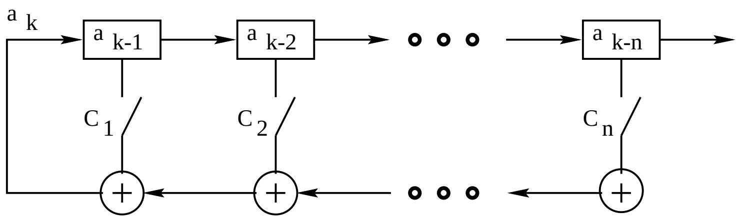


Figure 2.1: Binary Linear Feedback Shift Register

the LFSR. The C_i's are binary variables that indicate the position of the switches. A one indicates a closed switch, and a zero indicates an open switch. From the structure of the LFSR, it can be shown that the generated sequence $\{a_n\}$ satisfies the recursive equation

$$a_k = \sum_{i=1}^{n} c_i a_{k-i} \tag{2.1}$$

with an initial state $a_{-1}, a_{-2}, \ldots, a_{-n}$. The sequence generated by an LFSR will vary depending on its structure and the initial state. Any sequence $\{a_n\} = \{a_0, a_1, a_2, \ldots\}$ can be described by the *generating function* $G(x)$, which for an LFSR sequence is as follows,

$$G(x) = \frac{\sum_{i=1}^{n} c_i x^i (a_{-i}x^{-i} + \ldots + a_{-1}x^{-1})}{1 - \sum_{i=1}^{n} c_i x^i} = \frac{g(x)}{f(x)}. \tag{2.2}$$

The polynomial $f(x) = 1 - \sum_{i=1}^{n} c_i x^i$ is referred to as the *characteristic polynomial* of the sequence $\{a_n\}$ and of the shift register which produced it. Note that $f(x)$ is solely a function of the structure of the LFSR and is independent of the initial state. Furthermore, $f(x)$ is a monic polynomial of degree n, i.e. $c_n = c_0 = 1$.

In an LFSR, the next entry in the sequence is only dependent on the current state. Once any particular state has occurred for the second time, the generated sequence will be periodic from that point on. Therefore, the maximum period of an LFSR sequence is $2^n - 1$, which corresponds to one cycle through each of the $2^n - 1$ non-zero states. These sequences are commonly known as maximum length sequences or *m-sequences*. Golomb showed that the period of an LFSR sequence with characteristic polynomial $f(x)$ is the smallest integer p such that $f(x)$ divides $1 - x^p$ (modulo 2 arithmetic) [11]. This integer p is also called the *exponent* of $f(x)$. A necessary, but not sufficient condition on $f(x)$ to produce an m-sequence is that $f(x)$ must be irreducible. The number of polynomial of degree n which have *maximum exponent* is given by $\phi(2^n - 1)/n$ where $\phi()$ is the Euler ϕ-function.

2.1.2 Generalized Linear Feedback Shift Registers

In a generalized LFSR, each memory element contains a value in $\{0, 1, \ldots, q-1\}$ and all calculations are performed over the finite field $GF(q)$. A sequence, $a_0, a_1, a_2, \ldots$, generated by a generalized LFSR satisfies the linear recurrence relation

$$\sum_{i=0}^{n} c_i a_{k-i} = 0, \quad k = n, n+1, \ldots \quad c_0 = 1, c_n \neq 0, \tag{2.3}$$

where all the elements are taken from a finite field $GF(q)$. This equation can be re-arranged to express a_k in terms of a linear combination of the previous n values of the sequence $\{a_k\}$ as follows,

$$a_k = -c_0^{-1} \sum_{i=1}^{n} c_i a_{k-i}. \tag{2.4}$$

The basic structure of a generalized LFSR is shown in Figure 2.2. The generating function for the sequence produced by a generalized LFSR is $G(x) = g(x)/f(x)$. This is the same as in the binary case, with the exception that $f(x)$ and $g(x)$ have coefficients in $GF(q)$. Similar to the binary case, the period of a generalized LFSR sequence is given by the smallest integer p such that $f(x)$ divides $1 - x^p$ (modulo q arithmetic). The set of all possible sequences generated by an LFSR is called the *solution space* of $f(x)$ and is denoted by $S(f)$.

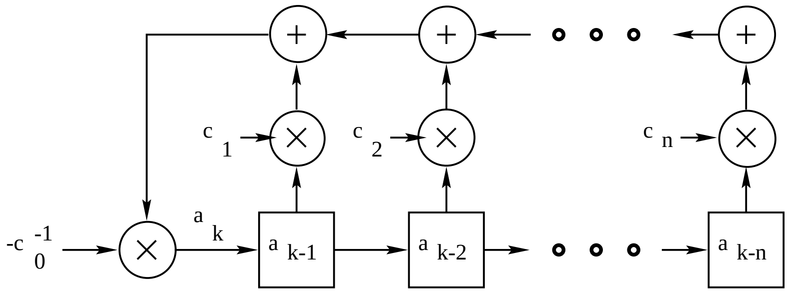


Figure 2.2: Generalized Linear Feedback Shift Register Structure

If a sequence, after the multiplication of each term by an element in $GF(q)$ results in a shift of the original sequence, then this element is called a *multiplier* of the sequence, and the number of positions the original sequence is shifted forward is called the *span* of the multiplier. Furthermore, the set of sequences obtained when a sequence is multiplied by all the non-zero element of $GF(q)$ is called a *block*. Dan Laksov, in [12] states the following two theorems[1] about the relation between multipliers, spans and blocks:

THEOREM 1: The multipliers of the sequences of $S(f)$, where $f(x)$ is irreducible, period p, $e = GCD(p, q-1)$ and $\mu = p/e$, are exactly the elements of

[1]Dan Laksov attained these theorems from the works of Hall and Ward [13, 14, 15].

$GF(q)$ satisfying $x^e = 1$, and have the span $0, \mu, 2\mu, \ldots, (e-1)\mu$.

THEOREM 2: If $f(x)$ is irreducible, then there are t unordered sequences (i.e. sequences which are not shifts of each other) in each block, and there are b blocks, where $t = (q-1)/e$ and $b = (q^n - 1)/(\mu(q-1))$.

Theorem 1 and *2* can be applied to an LFSR with irreducible polynomial $f(x)$ with coefficients in $GF(q)$, which produces a sequence of the maximum period $q^n - 1$, where q is a prime. The following results are easily verifiable:

$$\mu = q^n - 1/(q-1), \quad span = 0, \mu, 2\mu, \ldots, (q-2)\mu, \quad t = 1, \quad b = 1$$

Therefore, the solution space $S(f)$ consists of one block containing one unordered sequence. In other words, $S(f)$ contains each of the $q^n - 1$ phases (or forward shifts) of one unique sequence. Note that the minimum (non-zero) span is $q^n - 1/(q-1)$, which in the binary case is equal to the period $2^n - 1$, but for larger q is less than the period of the sequence.

2.2 Convolutional Encoders

Convolutional encoders can be divided into two main categories, the Non-Recursive Convolutional (NRC) encoder and the Recursive Systematic Convolutional (RSC) encoder. Figure 2.3 shows the structure of both of these encoders. The central component of the NRC encoder is the shift register, which stores previous values of the input bit stream. The outputs are formed by linear combinations of the current and past input values. This particular encoder is non-systematic, which denotes that the systematic (input) data is not directly sent as an output. However NRC encoders can be either systematic or non-systematic. The structure of the NRC encoder in Figure 2.3 can be expressed in terms of the the generator matrix $\mathbf{G} = [G_1(D) \;\; G_2(D)]$ where $G_1(D) = 1 + D + D^2 + D^3 + D^4$, and $G_2(D) = 1 + D^4$. The sequence generated by this NRC encoder will be $[u(D)G_1(D) \;\; u(D)G_2(D)]$, where $u(D)$ is the D-series representation of the input data.

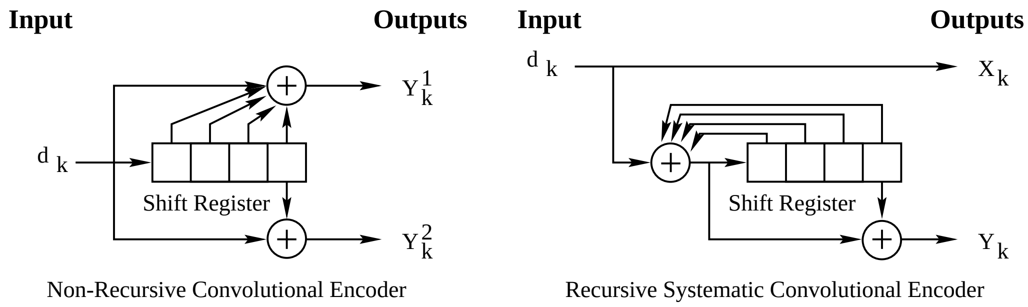


Figure 2.3: NRC and RSC Encoders

Figure 2.3 also shows the structure of the RSC encoder which are commonly used in Turbo Codes. The RSC encoder contains a systematic output, and more importantly, a feedback loop. The generator matrix for the RSC encoder is $\mathbf{G} = [1 \;\; A(D)/B(D)]$ where $A(D) = 1 + D^4$ and $B(D) = 1 + D + D^2 + D^3 + D^4$. $A(D)$ and $B(D)$ are commonly known as the feed-forward and feedback polynomials (or generators) respectively.

The feedback structure of the RSC encoder is encapsulated in the feedback polynomial $B(D)$, which is the same as the characteristic polynomial $f(x)$ described in section 2.1. Therefore, if $B(D)$ has maximum exponent $2^n - 1$, then the period of the output sequence of the RSC encoder will be $2^n - 1$ where n is the degree of $B(D)$. The output sequence generated by this RSC encoder is given by

$[u(D) \;\; u(D)A(D)/B(D)]$. Therefore, an RSC encoder with input sequence $u(D)$ will produce an infinite weight output sequence, with the exception of the the sequences $u(D)$ which are a multiple of the feedback polynomial $B(D)$. Note that this only includes a fraction 2^{-n} of all the possible input sequences. Furthermore, the degree of $u(D)$ must be at least that of $B(D)$ [16].

2.3 Turbo Code Encoder

The traditional Turbo Code encoder is built by concatenating two RSC encoders with an interleaver in between. Usually the systematic output of the second RSC encoder is omitted to increase the code rate. Figure 2.4 shows the Turbo Code encoder used in [1, 3]. Due to the interleaver between the RSC encoders and the nature of the decoding operation, the Turbo Code encoder must operate on the input data stream in N-bit blocks. (Therefore, Turbo Codes are indeed linear block codes.) Although the interleaver appears to play only a minor role in the encoder, in actuality the interleaver is a very important component.

An estimate of the BER performance of a Turbo Code can be made if the codeword weights or code geometry is known. Intuitively, we would like to avoid pairing low-weight codewords from the upper RSC encoder with low-weight words from $\{Y_k^2\}$. Many of these undesirable low-weight pairings can be avoided by properly designing the interleaver. Since the RSC encoders have infinite impulse response, the input sequence consisting of a single one will produce a high-weight codeword. However, two suitably placed 1's in the input sequence can drive the RSC encoder out of the zero state and back to the zero state within a very short span. If the

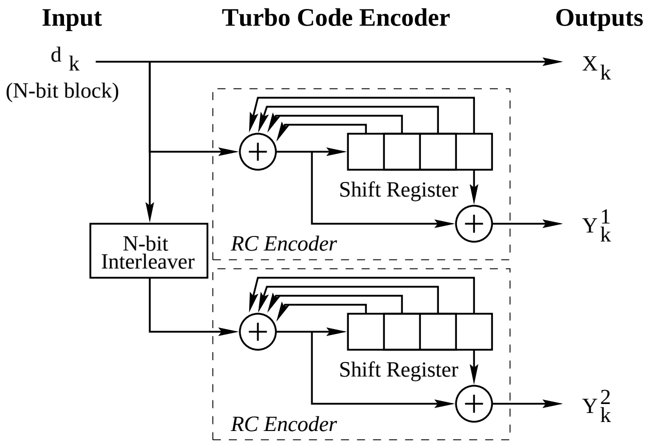


Figure 2.4: Rate 1/3 Turbo Code Encoder

interleaver maps these sequences to similar sequences, the over-all codeword weight will be low.

For moderate SNR values, the weight distribution of the first several low-weight codewords is required. For small interleaver sizes (N), the complete weight distribution can be determined by an exhaustive search using a computer. Calculating (or even estimating) the weight distribution for large N is a very difficult problem. Once the weight distribution is known, an estimate of the bit error rate can be made by using using the union bound [3].

2.4 Upper Bound on BER Performance

An upper bound on the BER performance of an (n, k) systematic block code is given by the union bound,

$$P_b \leq \sum_{w=1}^{k} \sum_{d=d_{min}}^{n} a(w,d) \frac{w}{n} \frac{1}{2} \operatorname{erfc}\left(\sqrt{d \frac{k}{n} \frac{E_b}{N_0}}\right), \tag{2.5}$$

where E_b/N_0 is the signal-to-noise ratio, w is the Hamming weight of the input block, d is the Hamming weight of the codeword, and $a(w,d)$, which is known as the *weight enumeration function*, is the number of such codewords. For simple block codes, $a(w,d)$ can be obtained by an exhaustive search. However, for long block lengths, obtaining $a(w,d)$ can be very difficult.

For parallel concatenated block codes, such as Turbo Codes, two linear systematic codes C_1 and C_2 are linked by an interleaver. In order to obtain the weight enumeration function of such a parallel code, the calculation must take into account each constituent code and the interleaver structure. Since this calculation becomes impractical to perform for even small block lengths, Benedetto and Montorsi introduced an abstract interleaver which they called a *uniform interleaver* in [17]. A uniform interleaver of length k is a probabilistic device which maps a given input word of weight w into all the distinct $\binom{k}{w}$ permutations with equal probability $1/\binom{k}{w}$. This interleaver causes the weight enumeration function of the second code to be independent of the first code. Therefore, the weight enumeration function for the parallel code can be calculated as follows:

$$A_w^{C_p}(Z) = A_w^{C_1}(Z)A_w^{C_2}(Z)/\binom{k}{w}, \tag{2.6}$$

where $A_w^{C_1}, A_w^{C_2}$ are the weight enumeration functions of the constituent codes. Finally, the upper bound on the BER performance of a Turbo Code is obtained by calculating equation 2.5 using $A_w^{C_p}(Z)$.

We present a computational technique to simplify the calculation of the weight enumeration function of the constituent RSC codes.

In the trellis diagram of an RSC code, each path in the trellis represents a transition from a previous state m', to a current state m with an input bit X_k and a parity bit Y_k. We define the *single stage trellis transition matrix* as follows:

$$T_{m',m}(W,Z) = [a_{m',m}(w,j)W^w Z^j], \tag{2.7}$$

where w is the Hamming weight of the input bit X_k, j is the Hamming weight of the parity bit Y_k, and $a_{m',m}(w,j)$ is the number of transitions in the trellis from state m' to state m with input weight w and parity bit weight j. Note that the Hamming weight of the complete codeword is $w + j$.

The weight distribution function for N stages of the trellis (corresponding to an N bit block) is obtained by calculating $T_{m',m}^{N+M}(W,Z)$, where M is the number of memory elements (assuming that M tail bits are added to terminate the trellis). Each entry in $T_{m',m}^{N+M}(W,Z)$ gives the *input redundant weight enumeration function* [17] for all trellis paths that start at state m' and end at state m after $N+M$ transitions. The entry which is of most interest is $T_{0,0}^{N+M}(W,Z)$, which is the input redundant weight enumeration function of all the paths that start and end at the all-zero state.

Since the upper bound on the BER performance is dominated by the lower weight codewords, the calculation of $T_{m',m}^{N+M}(W,Z)$ can be simplified by only enumerating codewords with input and parity Hamming weights below certain thresholds. With $T_{m',m}^{N+M}(W,Z)$ obtained, the remaining calculation is the same as presented in [17]. The input redundant weight enumeration function $T_{0,0}^{k}(W,Z)$ is converted into a weight enumeration function and used in equation 2.6. Finally, the upper bound on the BER performance is obtained by calculating equation 2.5 using $A_w^{C_p}(Z)$.

The upper bound on the BER performance of Turbo Codes are shown in Figure 2.5 and Figure 2.6 for various block lengths (N) and memory sizes (M). Specifically, the bounds for $M = 3$, $G = (15, 13)$ and $M = 4$, $G = (37, 21)$ and $G = (23, 35)$ RSC encoders are shown.

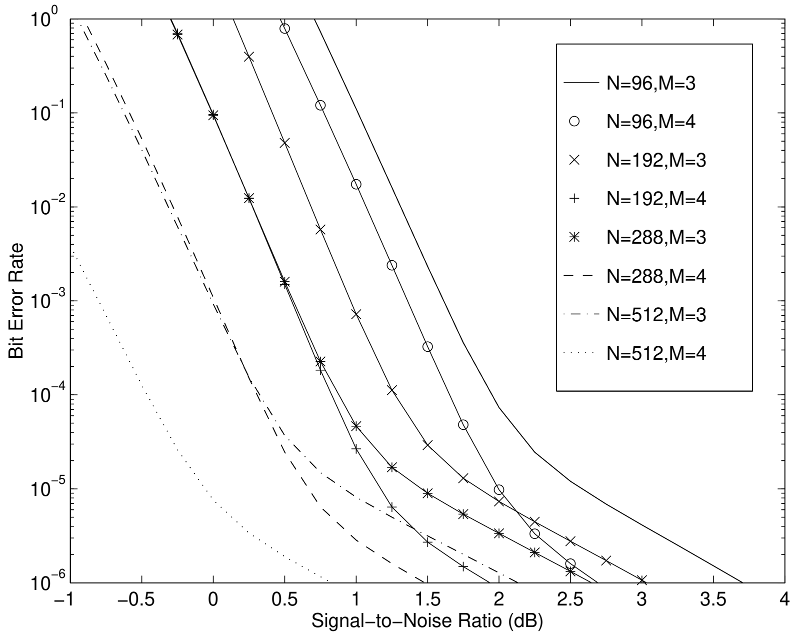


Figure 2.5: Upper Bounds on BER Performance of Turbo Codes for $M = 3$, $G = (15, 13)$ and $M = 4$, $G = (23, 35)$.

The curves in Figure 2.5 and Figure 2.6 have two major segments with greatly different slopes. The shallow sloping portion of the curves, for bit error rates below 10^{-5}, are dominated by a few very low weight codewords. For bit error rates greater than 10^{-5}, the curves show the impact of the first several low-weight codewords.

Figure 2.5 show the upper bounds on the BER performance of Turbo Codes with

$M = 3$ and $M = 4$ memory element RSC encoders. As the block length increases, the difference between these two codes also increase. For example, a BER of 10^{-4} can be obtained for $M = 3, N = 288$ or $M = 4, N = 192$. Therefore, there is a trade-off between block size (and thus interleaver delay) and memory size (which corresponds to decoder complexity).

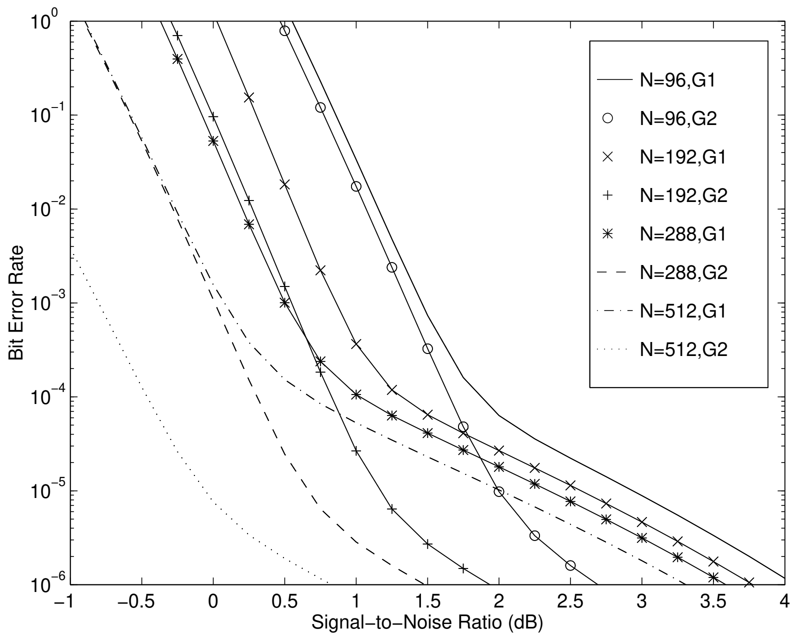


Figure 2.6: Upper Bounds on BER Performance of Turbo Codes for $M = 4$, $G1 = (37, 21)$ and $G2 = (23, 35)$.

Figure 2.6 show the upper bounds on the BER performance of two Turbo Codes with different generators. The first generator was used in [1]. Since the feedback polynomial $(37)_8$ does not produce a maximum length sequence, the upper bound for both the steep and shallow portions of the curves are inferior to the second generator with feedback polynomial $(23)_8$ which does produce a maximum length sequence. This same conclusion was also reached by Benedetto and Montorsi in [17].

2.5 Interleaver Design

The role of the interleaver in a Turbo Code is to modify the pattern of low-weight input sequences which result in a low-weight output after passing through the first Recursive Convolutional (RC) code so that the output weight of the second RC code is high. The major focus of interleaver design has been on breaking weight two input sequences which drives the RC encoder out of the zero state and back into the zero state within a short span resulting in low-weight outputs. For weight two input sequences which cannot be broken, it is desirable that the span of such sequences are large in at least one of the RC codes. Furthermore, the interleaver needs to permute positions near the right end of the block to positions far away from the right edge so as to minimize edge effects.

Consider an RC encoder with transfer function $G(D) = A(D)/B(D)$. From the results of section 2.1, we know that if $B(D)$ has maximum exponent, that G(D) is a periodic sequence with period $p = 2^n - 1$, where n is the degree of $B(D)$. Furthermore, we know that the solution set of G(D) consists of all the $2^n - 1$ phases (or right shifts) of one unique solution, known as an m-sequence. The phases of this m-sequence correspond to the (impulse) response of $G(D)$ when the input sequence, $u(D)$, consists of a single one at position i. Therefore, we label the phases of these sequences by the integers $i = 1, 2, \ldots, p$. It can be shown that the set of phases of an m-sequence (plus the all-zero sequence) constitute a group under binary addition [11]. The order of each element in this group is equal to two, meaning that the sum of each phase with itself results in the all-zero sequence (denoted as the zero phase).

We refer to time positions within the input data block which are congruent to i modulo p as C_i. $C_i = \{t_i : t_i \in [i, N], t_i = i \bmod(p)\}$. If the system is already in phase a, then an impulse at position $t \in C_i$ will result in phase $b = a \oplus i$ at the output of the corresponding RC encoder, where $\oplus$ denotes the group addition of the phases. Note that an impulse at position $t \in C_a$ will result in the zero phase.

In the process of interleaving, if two elements within a given C_i, $i = 1, \ldots, p$, are mapped into two positions within a given C_j, then these positions constitute a weight two, zero-phase sequence which remains zero-phase after interleaving. The interleaver can force the elements of a given C_i, $i = 1, \ldots, p$, to be mapped into different C_j's if and only if the number of positions in each C_i is less than p. This in turn means that the block length N should satisfy $N < p^2$. In practice, $N \gg p^2$ and the interleaver will not be able to break all the weight two input sequences. The fraction of the unbroken weight two sequences within a given C_i is at least $1/p$. The interleaver should be designed such that the span of such unbroken sequences is maximized.

A. Khandani presented an interleaver which is optimal in breaking weight two input sequences and maximizing the span of the unbroken weight two sequences in [18]. This interleaver is obtained by partitioning the input block into sub-blocks of length p and applying a cyclic shift of i positions, $i = 0, 1 \ldots$, to the elements of the i-th sub-block. The effective number of cyclic shifts applied to the i-th sub-block is equal to: $i (\bmod p)$. The BER performance of a Turbo Code with this interleaver performed poorly. Therefore, the BER performance of a Turbo Code is not determined solely by the weight two input sequences.

Also in [18], an alternative interleaver is designed by defining distance functions for each phase and develops a linear objective function which is solved using the Hungarian method [19]. These interleavers result in BER performance superior to that of a random interleaver.

A good review of other non-random and 'semi-random' interleavers was performed by S. Dolinar and D. Divsalar in [20]. Specifically, the authors consider a non-random interleaver which ensures that the minimum distance due to weight two input sequences grows roughly as $\sqrt{2N}$, where N is the block length. They also introduce a 'semi-random' interleaver, which they called *S-random*. This interleaver is designed by randomly selecting integers $i, 1 \leq i \leq N$ without replacement. Each selection is compared with the previous S selections. If the current selection is within a distance $\pm S$ to the previous S selections, then it is rejected. This process has the effect of permuting all positions within a distance of S in the original block, to a distance greater than S in the interleaved block. Although this process is not guaranteed to complete successfully, Dolinar and Divsalar observed that setting $S < \sqrt{N/2}$ usually produces a solution within a reasonable time.

Many other articles about interleaver design were recently presented at the IEEE International Symposium on Turbo Codes and Related Topics in Brest France, September 1997 [21, 22, 23].

2.6 Trellis Termination

In order to decode a linear block code using the trellis representation, it is desirable that the decoder knows both the starting state and finishing state. This gives rise to the need to terminate the trellis of an RSC code. Due to the recursive structure of the RSC encoder, it is not sufficient to merely set the last M information bits to zero (where M is the number of memory elements in the encoder) to drive the encoder to the all-zero state. The most commonly used trellis termination was first presented by Divsalar and Pollara in [4]. Their RSC encoder, shown in Figure 2.7, includes a switch with two positions A and B. The switch is in position A for the first N clock cycles, and in position B for an additional M cycles to drive the encoder to the all-zero state. This method is equivalent to setting the last M information bits equal to the value of the feedback path so that the addition of the feedback and the input is 0 (mod 2).

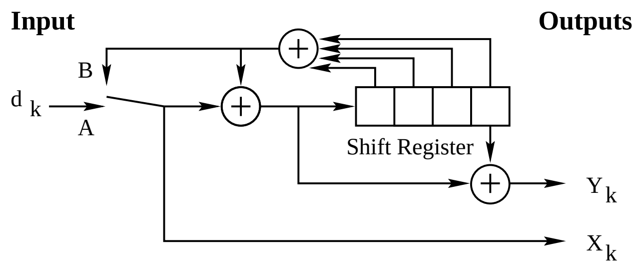


Figure 2.7: RSC Encoder with Trellis Termination

In the Turbo Code encoder, usually one trellis is terminated in this manner and the other trellis is left unterminated. This results in a $(n(N+M), N)$ block code, where the rate of the code (excluding termination) is $1/n$. For large block sizes, the effect of trellis termination on the BER performance and the code rate is negligible. However, for small block sizes, trellis termination becomes much more important.

2.7 Turbo Code Decoder

The Turbo Code decoder is constructed using RSC decoders to iteratively improve the Logarithm of the Likelihood Ratio (LLR) value for each input bit. Claude Berrou *et. al* proposed a modified version of the BCJR algorithm[2] for the RSC decoders in [1]. The BCJR algorithm calculates the *a posteriori probabilities* (APP) of the states and transitions of a Markov source observed though a discrete memoryless channel. Claude Berrou *et. al* modified this algorithm to account for the recursive nature of RSC encoders. Both Patrick Robertson [3] and Peter Jung [25] have introduced further modifications which reduce the complexity and memory storage of the algorithm. Benedetto *et. al* have introduced an alternative Soft-Input Soft-Output (SISO) maximum a posteriori module to decoded parallel and serial concatenated codes in [26]. A sub-optimal decoding algorithm, known as the Soft-Output Viterbi Algorithm (SOVA) can also be used. This algorithm augments the hard-decision outputs of the Viterbi algorithm with soft-output reliability information [27].

A detailed derivation of the modified BCJR algorithm can be found in [1], and will not be repeated here. However, this section will review the basic notation of the modified BCJR algorithm including the changes proposed by P. Robertson in [3].

The RSC encoder processes the input sequence $\{d_1, d_2, \ldots, d_N\}$ (where $d_k \in \{0, 1\}$) and produces two parallel data streams, $\{X_k\}$, $\{Y_k\}$, each of length N. The received data streams are denoted by $R_1^N = (R_1, R_2, \ldots R_N)$, where $R_k = (x_k, y_k)$.

[2] The BCJR algorithm is also known as the Bahl *et. al* algorithm. BCJR refers to each of the authors in [24].

Each R_k represents a 2-tuple of real-valued numbers corresponding to the outputs of the receiver correlators.

The RSC decoder generates an LLR value for each input bit. These LLR values are given by the expression

$$\Delta(d_k) = \log\left(\frac{Pr\{d_k = 1|R_1^N\}}{Pr\{d_k = 0|R_1^N\}}\right), \tag{2.8}$$

where $Pr\{d_k = i|R_1^N\}$ is the APP of the input bit d_k. The LLR values are generated by forward and backward recursions which can heuristically be associated with tracing the most likely path through the trellis in each direction. These recursions are given by the following equations:

$$\alpha_k(m) = \frac{\sum_{m'}\sum_{i=0}^{1}\gamma_i(R_k, m', m)\alpha_{k-1}(m')}{\sum_m\sum_{m'}\sum_{i=0}^{1}\gamma_i(R_k, m', m)\alpha_{k-1}(m')} \tag{2.9}$$

$$\beta_k(m) = \frac{\sum_{m'}\sum_{i=0}^{1}\gamma_i(R_{k+1}, m, m')\beta_{k+1}(m')}{\sum_m\sum_{m'}\sum_{i=0}^{1}\gamma_i(R_{k+1}, m', m)\alpha_k(m')} \tag{2.10}$$

In the above equations, $\gamma_i(R_k, m', m) = Pr\{R_k, S_k = m, d_k = i|S_{k-1} = m'\}$ encapsulates the trellis structure, the APP values of d_k and the channel statistics. The LLR values as a function of α, β and γ are given by the expression

$$\Delta(d_k) = \log\left(\frac{\sum_m\sum_{m'}\gamma_1(R_k, m', m)\alpha_{k-1}(m')\beta_k(m)}{\sum_m\sum_{m'}\gamma_0(R_k, m', m)\alpha_{k-1}(m')\beta_k(m)}\right). \tag{2.11}$$

The Turbo Code decoder proposed by P. Robertson in [3] is shown in Figure 2.8. Each RSC decoder implements the modified BCJR algorithm previously discussed. For an Additive White Gaussian Noise (AWGN) channel, the LLR values generated

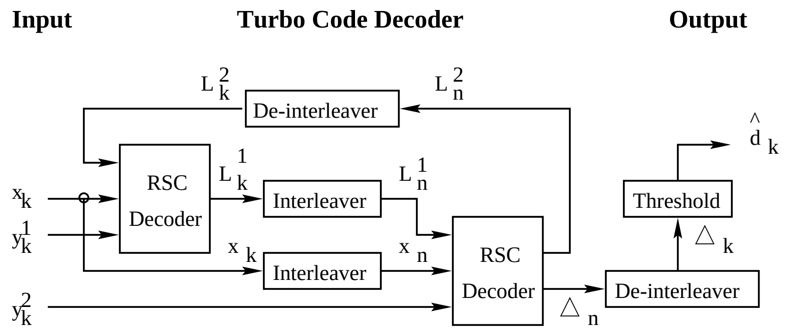


Figure 2.8: Patrick Robertson's Turbo Code Decoder

by the first RSC decoder can be expressed as a function of (three) independent terms, which Berrou *et. al* called intrinsic and extrinsic information [1]:

$$\Delta^1(d_k) = L_k^1 + L_k^2 + \frac{2}{\sigma_n^2} x_k. \tag{2.12}$$

Intrinsic information is the component of the LLR provided by the systematic data stream $\{x_k\}$, while extrinsic information is the component of the LLR provided by the encoded data stream $\{y_k\}$. Therefore, in the previous equation, the last term corresponds to the intrinsic information, where σ_n^2 is the variance of the noise. The remaining two terms, L_k^1, L_k^2 represent the extrinsic information generated by each of the RSC encoders. The L_k^1 term represents the extrinsic information generated by the first RSC decoder during the current iteration, while L_k^2 represents the extrinsic information generated by the second RSC decoder during the previous iteration. In order to prevent the passing of information to the second RSC decoder that was previously provided by that same RSC decoder, only the L_k^1 values are passed to the second RSC decoder. The extrinsic component of the LLR values generated by each RSC decoder are passed between RSC decoders in an iterative manner to improve the LLR values.

The final hard-decision is made by threshold decoding. The sign of the LLR value indicates the bit (0 or 1) and the magnitude of the LLR value indicates the reliability of the decision.

2.8 Turbo Code BER Performance

Figure 2.9 shows the BER performance of the Turbo Code encoder shown in Figure 2.4 over an AWGN channel at different iterations. For short block lengths ($N = 192$), the majority of the improvement in BER performance is obtained after the first 5 or so iterations. For large block lengths ($N = 10000$), the greatest improvement in BER performance occurs during iterations $5 - 15$, after which the incremental improvement is quite small.

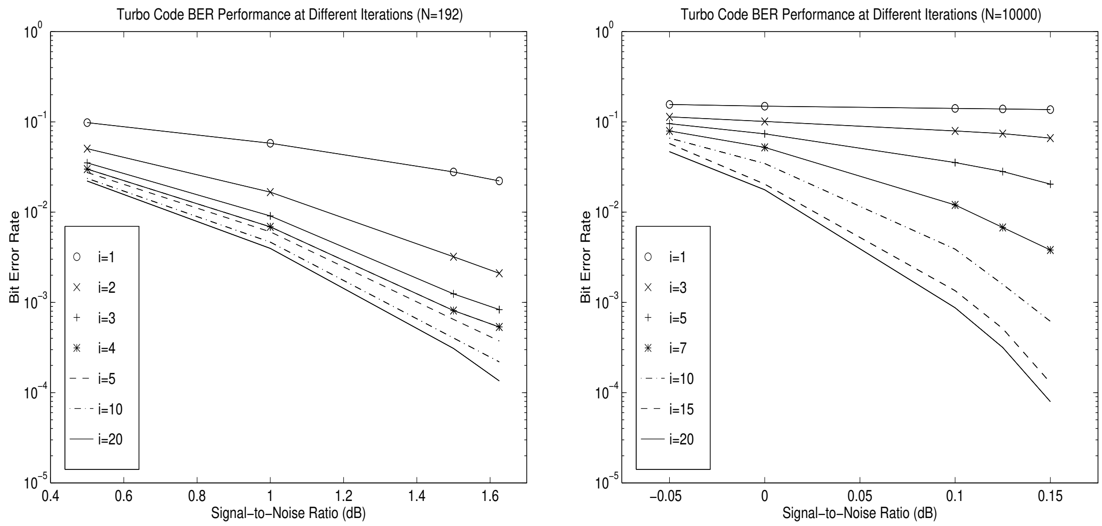


Figure 2.9: BER Performance of Turbo Codes at Different Iterations

Three of the major parameters in Turbo Code design that affect the BER performance are the number of memory elements (M), the block length (N) and the interleaver. Figure 2.10 shows the BER performance of a rate 1/3 Turbo Code ($N = 192$) with various numbers of memory elements. As expected, the BER performance improves with an increase in M. However, this improvement comes at the cost of increased decoder complexity. Figure 2.10 also shows the relationship between SNR and block length (N) for the BERs of $10^{-3}, 10^{-4}$ and 10^{-5}. Notice that for small block lengths ($N \approx 100$) the required SNR to obtain the three bit error rates is quite large. At large block lengths ($N > 10000$) the required SNR is considerably lower, but further increasing the block length only results in a slight reduction in the required SNR. Although large block lengths improve the BER performance, they also introduce long delays which cannot be tolerated in real-time systems with stringent delay specifications.[3]

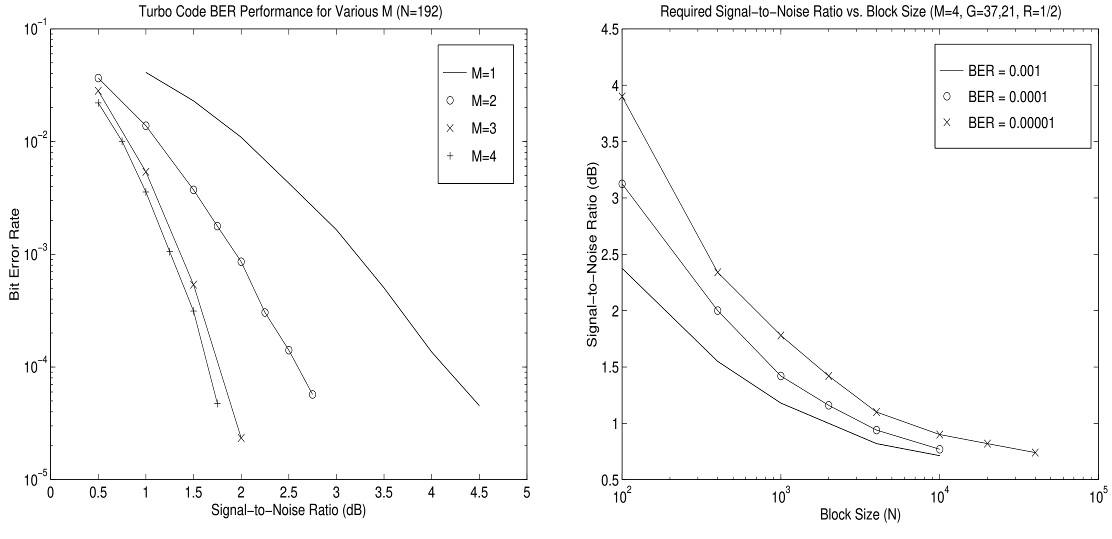


Figure 2.10: BER Performance of Turbo Codes

[3]The BER performance for various M was simulated by the author. The required SNR vs. N was taken from [3].

Chapter 3

Symbol-Based Turbo Codes

Symbol-Based Turbo Codes are our new method for combining Turbo Codes with different modulation techniques [28]. The motivation for this combination is to take advantage of inherent trade-offs between BER performance, code rate, spectral efficiency, and decoder complexity. The main focus of our study has been to produce moderate and low-rate (1/3 - 1/32) codes with good BER performance without dramatically increasing the complexity of the decoder. These codes are well suited for spread spectrum communication systems where the lower rates can be easily accommodated in the large spreading factor. A secondary goal has been to produce spectrally efficient codes which do not suffer the same BER performance degradation of traditional Turbo Codes when punctured to increase their code rates.

Unlike the other methods reviewed in Chapter 1 which use the outputs from all the parallel data streams of the Turbo Code encoder to select a point in a signal space, Symbol-Based Turbo Codes modulate each parallel data stream independently. In this chapter, the generalized structure of the Symbol-Based Turbo Code encoder and decoder are presented.

3.1 Symbol-Based Turbo Code Encoder

The Symbol-Based Turbo Code encoder combines the traditional Turbo code encoder with various modulation schemes by parsing the parallel data streams into n-bit sub-blocks or *symbols*. These symbols are then mapped to a 2^n point signal set. In order to maintain the correspondence between n-bit encoder output symbols and n-bit input symbols, the interleaver must be restricted to operate on a symbol-by-symbol basis. Such an interleaver permutes the input data block in n-bit symbols, but does not change the order of the bits within each symbol. With this restriction in place, the encoder effectively operates over n-bit symbols instead of binary symbols.

The structure of the Symbol-Based Turbo Code encoder is shown in Figure 3.1. The input block is parsed into n-bit input symbols, d_k, and is encoded using a traditional Turbo Code encoder with a restricted interleaver. Each parallel data stream then maps the n-bit encoded symbols to a 2^n point signal set. The over-all code rate is $n/3j$ where n is the symbol size and j is the dimensionality of the signal set. This method can easily be adapted to Turbo Codes which concatenate more than two RSC encoders. Furthermore, each parallel data stream can me modulated with a different modulation scheme. Therefore, there is a great deal of flexibility in designing codes with very high rates as well as very low rates.

By parsing the input data block into n-bit symbols, we are in essence merging (or compressing) n sections of the encoder trellis into one. Figure 3.2 shows two

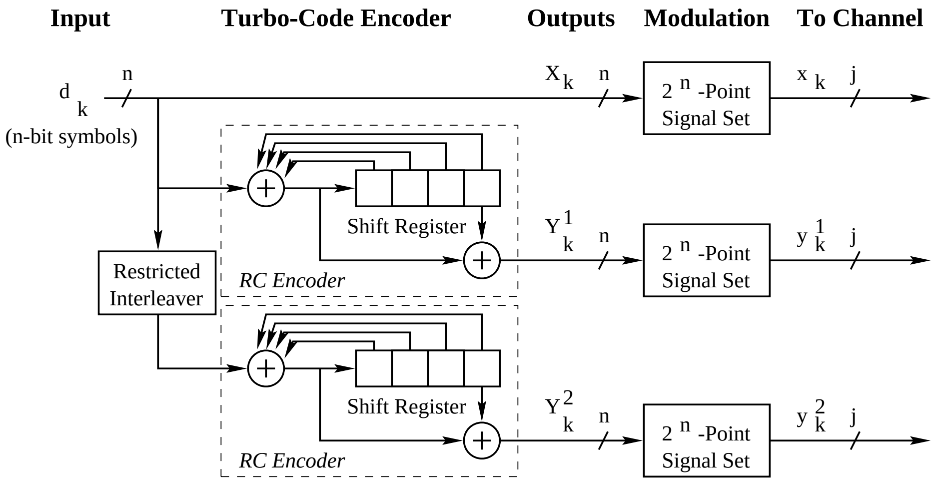


Figure 3.1: Symbol-Based Turbo Code Encoder/Modulator

stages of the RSC trellis with generator $(13, 11)_8$. The solid lines and dotted lines correspond to the input bit of 0 and 1 respectively. Figure 3.2 also shows the trellis diagram when two stages of the original trellis are merged together.

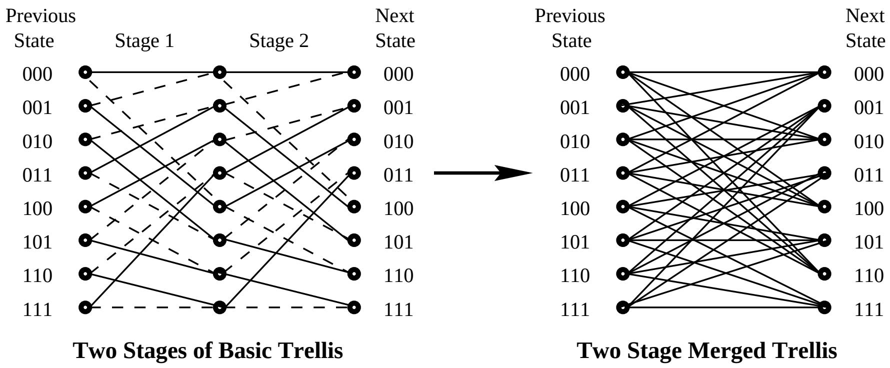


Figure 3.2: Two Stage Basic and Merged Trellis Diagrams

Similarly, Figure 3.3 shows three stages of the basic RSC trellis and the three stage merged trellis. In general, there are 2^n branches leaving each state of an

n-stage merged trellis diagram, which correspond to the 2^n possible n-bit binary values.

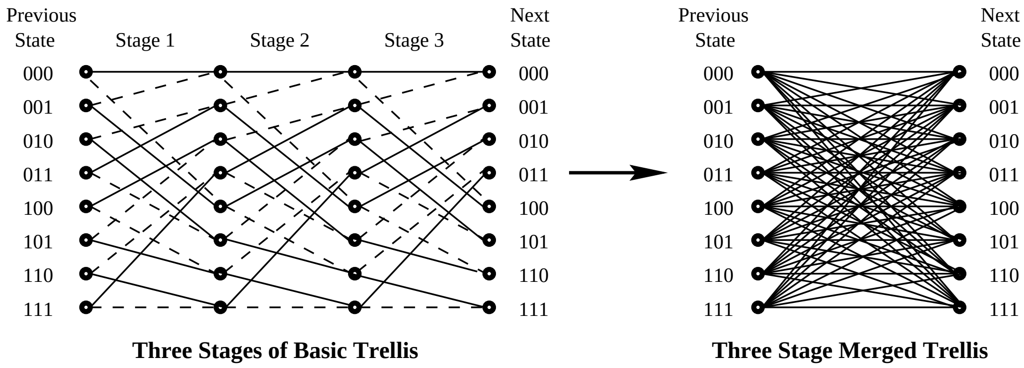


Figure 3.3: Three Stage Basic and Merged Trellis Diagrams

Note that when the number of branches leaving each state is greater than the number of states in the merged trellis, parallel transitions occur. These parallel transitions correspond to short error events which cannot be broken by a symbol-by-symbol interleaver, and consequently, severely degrade the performance of the code. Therefore, the symbol size is limited to $n \leq M$ where M is the number of memory elements in the RSC encoder.

The structure of the Symbol-Based Turbo Code encoder allows for modulation using any arbitrary signal set with 2^n points and J dimensions. We denote the j-th dimension of the signal point associated with the i-th symbol by $m_{i,j}$ for $i = 0, 1, \ldots, 2^n - 1$ (the decimal value of each of the possible n-bit symbols), and $j = 1, 2, \ldots, J$. The J-dimensional modulation vector corresponding to a particular n-bit symbol is denoted by $\vec{m_i} = (m_{i,1}, m_{i,2}, \ldots, m_{i,J})$. Table 3.1 shows the modulation vectors for a few common modulation schemes.

Table 3.1: Various Modulation Vectors for n-bit Symbols

Modulation Scheme	**Dimensions**	**Modulation Vectors**
BPSK	n	$\vec{m}_i = (\pm\sqrt{E_b}, \pm\sqrt{E_b}, \ldots)$
Orthogonal†	2^n	$\vec{m}_0 = (\sqrt{E_s}, 0, 0, \ldots, 0)$, $\vec{m}_1 = (0, \sqrt{E_s}, 0, \ldots, 0), \ldots$
Bi-orthogonal†	2^{n-1}	$\vec{m}_i = (\pm\sqrt{E_s}, 0, 0, \ldots, 0)$, $(0, \pm\sqrt{E_s}, 0, \ldots, 0), \ldots$
M-PSK	2	$\vec{m}_i = (\sqrt{E_s}\cos\theta_i, \sqrt{E_s}\sin\theta_i)$ $\theta_i = 2\pi i/2^n$ ‡
ASK	1	$\vec{m}_i = \pm d, \pm 3d, \pm 5d, \ldots$

† Orthogonal and bi-orthogonal modulation schemes can also be implemented using Hadamard matrices and BPSK modulation.

‡ For larger symbol sizes, θ_i is grey coded to improve BER performance.

3.2 Symbol-Based Turbo Code Decoder

The Symbol-Based Turbo Code decoder structure is very similar to the traditional Turbo Code decoder shown in Figure 2.8. However, the constituent RSC decoders described in section 2.7 operate on a single stage binary RSC trellis. In the Symbol-Based Turbo Code decoder, the constituent RSC decoders are modified to operate on n-bit symbols using the n-stage merged trellis diagram. We introduced the following changes to the modified BCJR algorithm to accomplish this.

Firstly, the LLR values are defined differently to account for the 2^n possible symbol values (instead of just two values, 0 and 1). The new LLR values are defined as follows:

$$\Delta(d_k = i) = \log\left(\frac{Pr\{d_k = i|\vec{R}\}}{Pr\{d_k = 0|\vec{R}\}}\right), \; i = 0, 1, \ldots, 2^n - 1 \tag{3.1}$$

where $\vec{R}$ denotes the received data streams. This definition for the LLR values allows for the easy conversion between symbol LLR values and symbol APP values.

For an Additive White Gaussian Noise (AWGN) channel, the branch metrics are given by [3]:

$$Pr\{x_k|d_k, S_k, S_{k-1}\} = e^{-\frac{1}{N_0}(x_k - b^s(d_k, S_{k-1}, S_k))^2} \tag{3.2}$$

$$Pr\{y_k|d_k, S_k, S_{k-1}\} = e^{-\frac{1}{N_0}(y_k - b^p(d_k, S_{k-1}, S_k))^2} \tag{3.3}$$

where $b^{s|p}(d_k, S_{k-1}, S_k)$ is the modulator output associated with the branch from state S_{k-1} to state S_k at step k if the corresponding input d_k is equal to i. The $b^{s|p}()$ values correspond to the $m_{i,j}$ values discussed earlier. These equations can be generalized for any signal set by calculating them for each dimension and multiplying the resulting terms together. This calculation is based on the assumption that each dimension of the modulation vector $\vec{m}_i$ is independent of all the others, and experience independent AWGN.

Thirdly, the expression for the intrinsic information [1, 3] is also modified to take into account the modulation scheme used. A general expression for the intrinsic information is as follows:

$$\log\left(\frac{Pr\{\vec{x}_k|d_k = i\}}{Pr\{\vec{x}_k|d_k = 0\}}\right) = \frac{1}{N_0}\sum_{j=1}^{J}[2x_{k,j}(m_{i,j} - m_{0,j}) + (m_{0,j}^2 - m_{i,j}^2)], \tag{3.4}$$

where $x_{k,j}$ is the j-th component of the k-th received systematic symbol. Depending on the modulation scheme used, the above expression may be simplified even further.

Lastly, the final hard-decision decoding is performed by selecting the largest LLR value and mapping the index of that LLR value back to an n-bit symbol.

For RSC encoders, whose trellis representation does not have any parallel transitions, the modified BCJR algorithm can be simplified. This is a practical case to consider, because, as noted in section 3.1, parallel transitions in the merged trellis diagram severely degrade the performance of the code. In the modified BCJR algorithm, summations are performed over previous states m' and current states m. For a trellis diagram with no parallel transitions a previous state m' and current state m indicate a unique transition. Since a previous state m' and an input i also uniquely define a current state m and an output Y in any trellis diagram, the branch metrics in equations 3.2, 3.3 can be re-defined in terms of the indices m' and i as follows:

$$Pr\{\vec{x}_k|S_{k-1} = m', d_k = i\} = \prod_{j=1}^{J} \exp\left(\frac{-(x_{k,j} - m_{i,j}(X))^2}{N_0}\right) \tag{3.5}$$

$$Pr\{\vec{y}_k|S_{k-1} = m', d_k = i\} = \prod_{j=1}^{J} \exp\left(\frac{-(y_{k,j} - m_{i,j}(Y))^2}{N_0}\right) \tag{3.6}$$

Furthermore, the forward and backward recursions are also changed so that all the summations are performed over the indices m' and i. This simplification reduces the complexity of the modified BCJR algorithm because all the summations are performed only over trellis branches which actually exist.

A further simplification can be made to the modified BCJR algorithm by examining the γ values. The γ can be expressed as follows by using Bayes rule and the simplification previously discussed:

$$\gamma_i(R_k, m') = Pr\{\vec{x}_k | S_{k-1} = m', d_k = i\} Pr\{\vec{y}_k | S_{k-1} = m', d_k = i\} Pr\{d_k = i\}. \tag{3.7}$$

The first two terms are the branch metrics for the systematic and parity data streams and are given by equations 3.5, 3.6. The third term is the APP of the input symbol d_k. Since the first two terms do not change between iterations, they can be calculated once and merely multiplied by the most current APP values $Pr\{d_k = i\}$.

3.3 Orthogonal Modulation

Symbol-Based Turbo Codes with orthogonal modulation use a 2^n point orthogonal signal set to transmit the parallel data streams. As outlined in Table 3.1, the modulation vectors are $\vec{m}_i = \{0, \ldots, 0, \sqrt{E_s}, 0, \ldots, 0\}$, where E_s is the transmitted energy per n-bit symbol. Clearly $E_s = nE_b$, where E_b is the equivalent energy per bit. The modulation vectors $\vec{m}_i$ are zero in every position except the i-th position, into which all the symbol energy is placed. Equivalently, the rows of a Hadamard matrix H_{2^n} can also be used for the modulation vectors.

The generalized expression for the intrinsic information as given in equation 3.4 can be simplified as follows:

$$\log\left(\frac{Pr\{\vec{x}_k | d_k = i\}}{Pr\{\vec{x}_k | d_k = 0\}}\right) = \frac{2\sqrt{E_s}}{N_0}(x_{k,i} - x_{k,0}) \tag{3.8}$$

The rates of these codes are $n/3(2^n)$ where n is the symbol size. Therefore, the code rates $1/6, 1/6, 1/8, 1/12, 1/19.2, 1/32$ are attainable for $n = 1, 2, \ldots, 6$. These codes are well suited to spread spectrum communication systems where the low code rates can be accommodated in the large spreading factor. For $n = 6$, the resulting code has the same spectral efficiency as the IS-95 up-link code (composed of a rate 1/3 convolutional code and 2^6-ary orthogonal signaling.) A comparison of the BER performance of these codes and a convolutional code with orthogonal signaling with the same structure as the IS-95 up-link code is shown in Chapter 5. The Symbol-Based Turbo Codes cannot be directly compared with the IS-95 Up-link because the IS-95 Up-link consists of non-coherent reception while our study has focused on coherent reception.

3.4 M-ary PSK Modulation

The second Symbol-Based Turbo Code we studied uses M-ary PSK modulation. The modulation vectors are given by $\vec{m}_i = (\sqrt{E_s}\cos\theta_i, \sqrt{E_s}\sin\theta_i)$, where $\theta_i = 2\pi i/2^n$. For larger symbol sizes, the θ_i's are grey coded to improve BER performance.

The generalized expression for the intrinsic information as given in equation 3.4 can be simplified as follows:

$$\log\left(\frac{Pr\{\vec{x}_k|d_k = i\}}{Pr\{\vec{x}_k|d_k = 0\}}\right) = -\frac{2\sqrt{E_s}}{N_0}(x_I(1 - \cos\theta_i) - x_Q \sin\theta_i) \tag{3.9}$$

where x_I, x_Q are the in-phase and quadrature components of the k-th systematic modulation vector.

The rates of these codes are $n/6$ where n is the symbol size. Therefore, the code rates $1/6, 1/3, 1/2, 2/3, 5/6$ are attainable for $n = 1, 2, \ldots, 6$. The BER performance of these codes are compared with convolutional codes with a constraint length $K = 9$ of similar code rates in Chapter 5.

3.5 Bi-orthogonal Modulation

The third Symbol-Based Turbo Code we studied uses bi-orthogonal modulation. The modulation vectors are given by $\vec{m}_i = (0, \ldots, 0, \pm\sqrt{E_s}, 0, \ldots, 0)$. Equivalently, the rows of a Hadamard matrix $H_{2^{n-1}}$ (or the complement of a row) can also be used for the modulation vectors. Table 3.2 shows the mapping between n-bit symbols and bi-orthogonal vectors.

Table 3.2: Mapping Between n-bit Symbols and Bi-orthogonal Vectors

Symbol Value		**Modulation Vectors**		
Decimal	**Binary**	$n = 1$	$n = 2$	$n = 3$
0	000	$(-\sqrt{E_s})$	$(-\sqrt{E_s}, 0)$	$(-\sqrt{E_s}, 0, 0, 0)$
1	001	$(+\sqrt{E_s})$	$(0, -\sqrt{E_s})$	$(0, -\sqrt{E_s}, 0, 0)$
2	010		$(0, +\sqrt{E_s})$	$(0, 0, -\sqrt{E_s}, 0)$
3	011		$(+\sqrt{E_s}, 0)$	$(0, 0, 0, -\sqrt{E_s})$
4	100			$(0, 0, 0, +\sqrt{E_s})$
5	101			$(0, 0, +\sqrt{E_s}, 0)$
6	110			$(0, +\sqrt{E_s}, 0, 0)$
7	111			$(+\sqrt{E_s}, 0, 0, 0)$

If we express the n-bit symbols in terms of the binary digits $b_1 b_2 \ldots b_n$, then the sign of the non-zero component of the bi-orthogonal vector is given by b_1. A one corresponds to a positive sign and a zero to a negative sign. Furthermore, the position in which the symbol energy, E_s, is placed, is given by the expression $b_2 b_3 \ldots b_n$ if $b_1 = 0$, and $\bar{b}_1 \bar{b}_2 \ldots \bar{b}_n$ if $b_1 = 1$, where $\bar{b}_i$ denotes the binary complement of bit b_i. This mapping results in the distance properties, $d^2_{i,j} = 2E_s$ if $i \neq \bar{j}$ and $d^2_{i,\bar{i}} = 4E_s$. This mapping maximizes the BER performance of the code by separating symbols whose binary representations are complements of one another by the largest distance.

The expression for the intrinsic information for this modulation scheme is determined by equation 3.4 and Table 3.2. There is no simplification of this expression because of the mapping required to maximize the BER performance.

The rates of these codes are $n/3(2^{n-1})$ where n is the symbol size. Therefore the code rates $1/3, 1/3, 1/4, 1/6, 1/9.6, 1/16$ are attainable for $n = 1, 2, \ldots, 6$.

3.6 BPSK Modulation

The forth Symbol-Based Turbo Code we studied uses n-dimensional BPSK modulation. The modulation vectors are given by $\vec{m}_i = (\pm\sqrt{E_b}, \pm\sqrt{E_b}, \ldots, \pm\sqrt{E_b})$. The encoding and modulating of this code is identical to the traditional Turbo Code except for the restriction placed on the interleaver as discussed in section 3.1.

The generalized expression for the intrinsic information as given in equation 3.4 can be simplified as follows:

$$\log\left(\frac{Pr\{\vec{x}_k | d_k = i\}}{Pr\{\vec{x}_k | d_k = 0\}}\right) = \frac{4\sqrt{E_b}}{N_0}\sum_{i=1}^{n} x_i b_i \tag{3.10}$$

where the b_i's are the binary digits of the k-th input symbol.

The rates of these codes are the same for any block size. We can change the rate of the code by adding an additional feed-forward generator and then puncturing the resulting encoded bit streams. Figure 3.4 shows an RSC encoder with feedback polynomial $1+D+D^3$, and feed-forward polynomials $1+D+D^2+D^3$ and $1+D^2+D^3$.

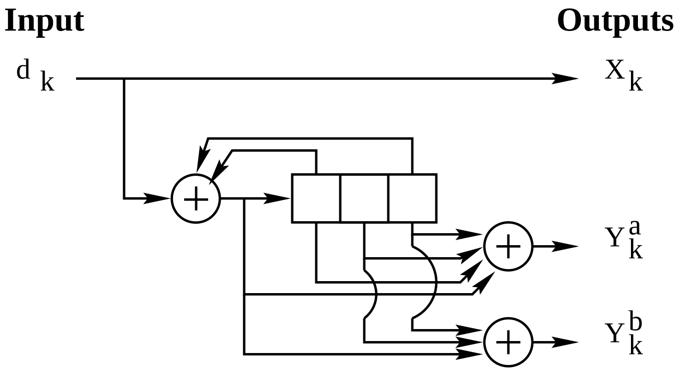


Figure 3.4: Recursive Systematic Convolutional Encoder With Dual Feed-Forward Outputs

This RSC encoder can be used to obtain a Turbo Code with code rates of $1/5, 1/4, 1/3, 1/2$ by puncturing the parallel data streams. In Table 3.3, the puncturing patterns for the various code rates are shown. A one denotes a transmitted bit, and a zero denotes a non-transmitted (or punctured) bit. The puncturing patterns shown in Table 3.3 are used regardless of the symbol size. Other uniform or non-uniform puncturing patterns can also be used.

Table 3.3: Puncturing Patterns for Various Rate Turbo Codes

Code Rate	**Bit Stream**				
	X	Y^{1a}	Y^{1b}	Y^{2a}	Y^{2b}
1/5	1111	1111	1111	1111	1111
1/4	1111	1111	1010	1111	0101
1/3	1111	1111	0000	1111	0000
1/2	1111	1010	0000	0101	0000

3.7 Parity Bits

The inclusion of various parity codes before and/or after the Symbol-Based Turbo Code encoder were also studied. The first Symbol-Based Turbo Code with parity bits studied parses the input data into $n-1$ bit sub-block and adds a single parity bit to each sub-block to create n-bit symbols. These symbol are then encoded using the Symbol-Based Turbo Code encoder. Interleaving is performed on an n-bit basis. The rates of these codes are $(n-1)/3n$ for BPSK modulation. We refer to these codes as Symbol-Based Turbo Codes with *full parity* because the parity bit is passed to all of the parallel data streams. The trellis diagram of the RSC encoder consists of 2^{n-1} branches leaving each state. These branches correspond to all the possible $n-1$ bit sub-blocks. This trellis can also be obtained from the n-stage merged trellis diagram by removing every branch which does not have an even parity input symbol. The decoder uses this modified merged trellis diagram in the constituent decoders.

The second Symbol-Based Turbo Code with parity studied also parses the input data into $n-1$ sub-blocks and calculates a parity bit for each sub-block. However, this parity bit is only passed onto the encoded data streams, as shown in Figure 3.5, and not to the systematic data stream. We call these codes Symbol-Based Turbo Codes with *partial parity*. This structure results in a code rate of $(n-1)/(a+2j)$. When BPSK modulation is used, the code rate becomes $(n-1)/(3n-1)$. Therefore code rates of $1/5, 1/4, 3/11$ are attainable for $n = 2, 3, 4$.

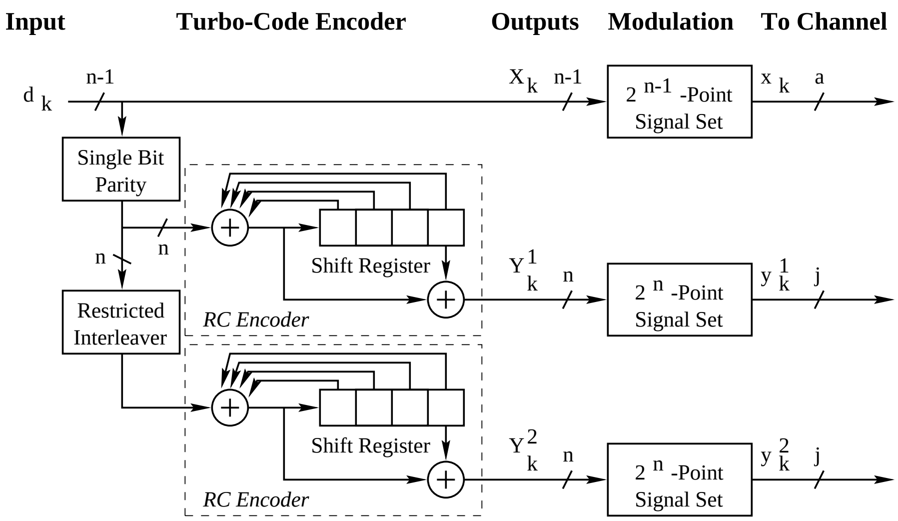


Figure 3.5: Symbol-Based Turbo Code Encoder With Partial Parity

Chapter 4

Simulation Setup

The BER performance of the Symbol-Based Turbo Codes studied were obtained by simulation using an AWGN channel and a Rayleigh fading channel. The discrete model of the AWGN channel is given by the expression $\vec{y}_k = \vec{x}_k + \vec{z}_k$, where $\vec{x}_k$ is the transmitted symbol and $\vec{z}_k$ is a Gaussian random vector with independent and identically distributed (i.i.d.) components with mean zero and variance $N_0/2$. Similarly, the discrete model of the Rayleigh fading channel is given by the expression $\vec{y}_k = a_k\vec{x}_k + \vec{z}_k$, where $\vec{x}_k$ and $\vec{z}_k$ are the same as above, and the a_k's are i.i.d. random variables with a Rayleigh distribution of the form $f(a) = 2ae^{-a^2}$ for $a \geq 0$. For the Rayleigh fading channel, the decoder incorporates the a_k's into the decoding algorithm. In other words, it is assumed that the receiver can determine the multiplicative fading factors. The discrete model for both channels are shown in Figure 4.1.

Most of the simulations were performed using the Symbol-Based Turbo Code encoder shown in Figure 3.1. The RSC encoders have $M = 4$ memory elements and

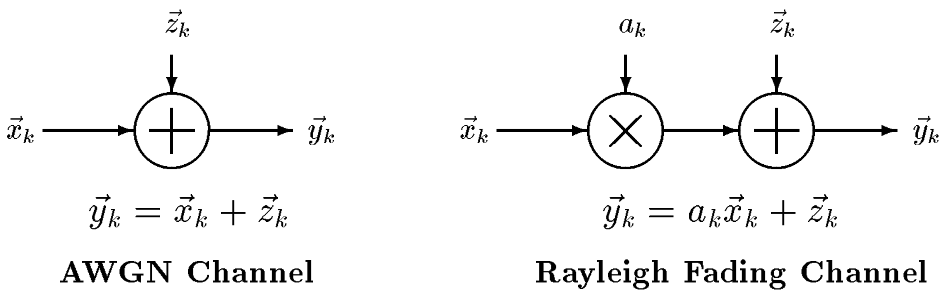


Figure 4.1: AWGN and Rayleigh Fading Channel Models

generator polynomial $(37, 21)_8$. Unless otherwise noted, 20 iterations are performed by the decoder. Assuming a data rate of 9.6 Kbits/sec, the IS-95 delay specification of 20 ms corresponds to a maximum block length of $N = 192$ bits. This block length was used for most of the simulations. However, other block lengths were also simulated for comparison. As discussed previously, when the block size, n, exceeds the number of memory elements, parallel transitions occur in the merged trellis diagram which severely degrade the performance of the code. For symbol sizes of 5 and 6, RC encoders with the value of $M = n$ are used.

Some of the Symbol-Based Turbo Codes were also simulated using the RSC encoder shown in Figure 3.4, which has $M = 3$ memory elements and two feed-forward outputs. In each case, the block size and the code rate are noted.

The Symbol-Based Turbo Codes with orthogonal, bi-orthogonal, M-ary PSK, and BPSK modulation were simulated. For BPSK modulation, the Symbol-Based Turbo Codes were simulated with no parity, full parity and partial parity.

Chapter 5 shows the numerical results of all the simulations performed.

Chapter 5

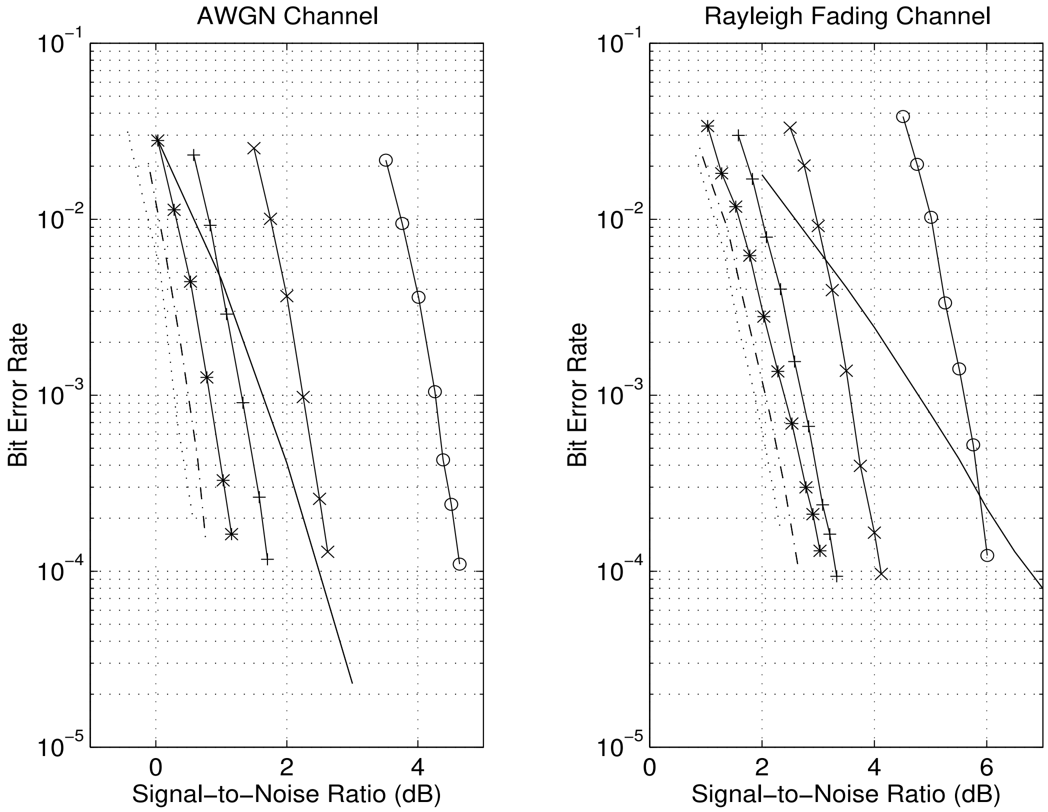
Numerical Results

5.1 Symbol-Based Turbo Codes BER Performance

5.1.1 Orthogonal Modulation

Figure 5.1 shows the BER performance of Symbol-Based Turbo Codes with orthogonal modulation over an AWGN channel and a Rayleigh fading channel for symbol sizes $n = 1, 2, \ldots, 6$. The rates of these codes are $n/3(2^n)$. For a symbol size $n = 6$, the resulting code has the same spectral efficiency as the IS-95 Up-link code (composed of a rate 1/3 convolution code with 2^6-ary orthogonal signaling). In Figure 5.1, the BER performance of the Symbol-Based Turbo Codes are compared with a convolutional code, constraint length $K = 9$, with 2^6-ary orthogonal signaling (i.e. a code with the same structure as the IS-95 Up-link).

The Symbol-Based Turbo Codes cannot be directly compared with the IS-95 Up-link because the IS-95 Up-link consists of non-coherent reception while our



o	Symbol-Based Turbo Code, Symbol Size $n = 1$, Rate 1/6
x	Symbol-Based Turbo Code, Symbol Size $n = 2$, Rate 1/6
+	Symbol-Based Turbo Code, Symbol Size $n = 3$, Rate 1/8
$*$	Symbol-Based Turbo Code, Symbol Size $n = 4$, Rate 1/12
-.-	Symbol-Based Turbo Code, Symbol Size $n = 5$, Rate 1/19.2
...	Symbol-Based Turbo Code, Symbol Size $n = 6$, Rate 1/32
—	Convolutional Code ($K = 9$) with Orthogonal Signaling, Rate 1/32

Figure 5.1: BER Performance of Symbol-Based Turbo Codes ($M = 4$, $N = 192$) With Orthogonal Modulation

study has focused on coherent reception. However, Figure 5.1 does show that the Symbol-Based Turbo Code with orthogonal modulation with symbol size $n = 6$ performs about 1.4 dB better than the convolutional code with orthogonal signaling at a BER of 10^{-3} for an AWGN channel and about 2.9 dB better for a Rayleigh fading channel. This result suggests that the Symbol-Based Turbo Code (symbol size $n = 6$) with non-coherent reception will perform better than the IS-95 Up-link code. However, simulations are required to confirm this.

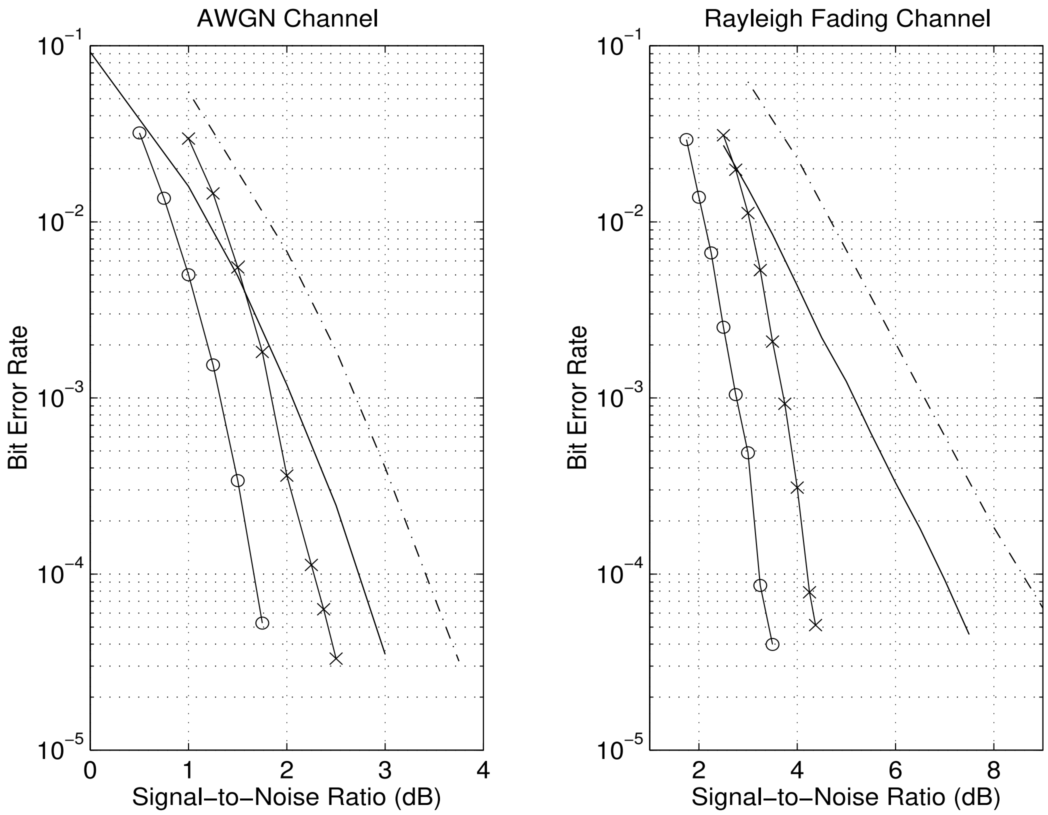
5.1.2 M-ary PSK Modulation

The Symbol-Based Turbo Codes with M-ary PSK modulation have rates of $n/6$, where n is the symbol size. The BER performance of these codes are compared with rate 1/3 and rate 1/2 convolutional codes with constraint length ($K = 9$) for both AWGN and Rayleigh fading channels in Figure 5.2. For a BER of 10^{-3}, the Symbol-Based Turbo Codes with M-ary PSK modulation performs about 0.7 dB better than convolutional codes of the same rate over an AWGN channel, and about 2.2 dB better over a Rayleigh fading channel.

5.1.3 Bi-orthogonal Modulation

The third Symbol-Based Turbo Code simulated uses bi-orthogonal modulation. The rates of these codes are $n/3(2^{n-1})$. Figure 5.3 shows the BER performance of these codes over an AWGN channel.

The Symbol-Based Turbo Code with bi-orthogonal modulation obtains a BER of 10^{-3} at an SNR of 0.68 dB for a symbol size of $n = 4$. For a symbol size of



o Symbol-Based Turbo Code, Rate 1/3
x Symbol-Based Turbo Code, Rate 1/2
— Convolutional Code, Rate 1/3, $K = 9$
-.- Convolutional Code, Rate 1/2, $K = 9$

Figure 5.2: BER Performance of Symbol-Based Turbo Codes ($M = 4$, $N = 192$) With M-ary PSK Modulation and Convolutional Codes ($K = 9$)

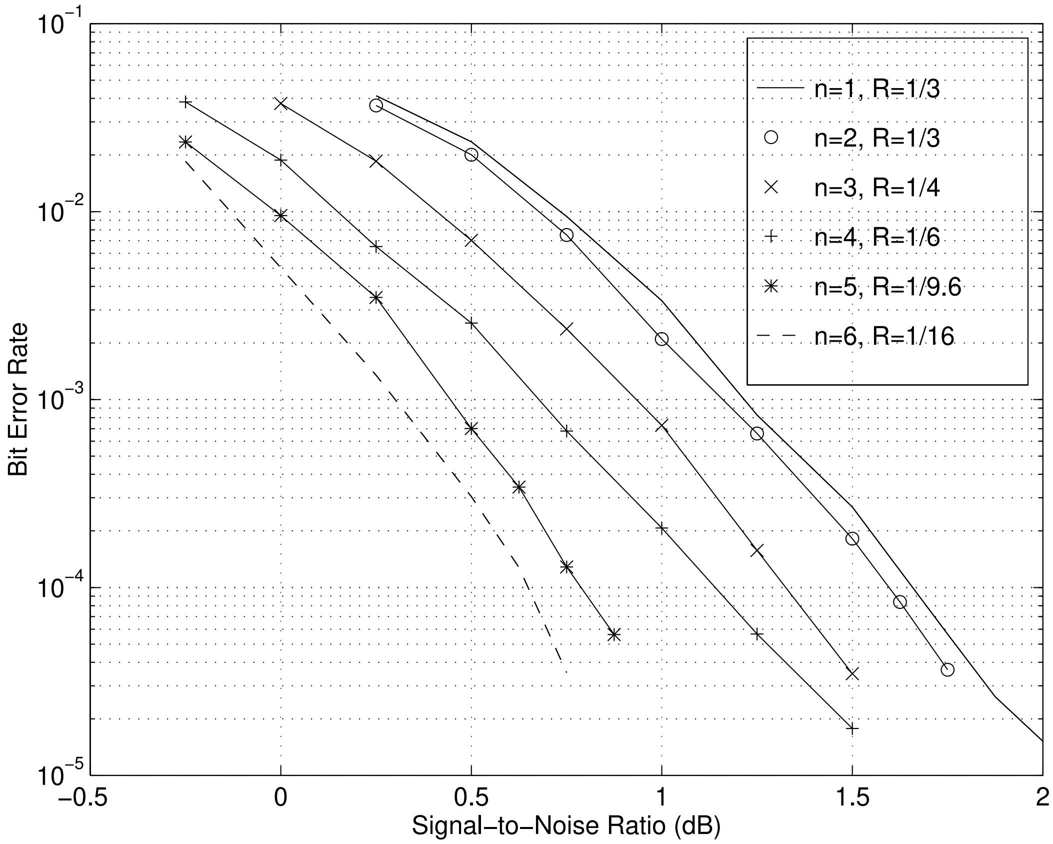


Figure 5.3: BER Performance of Symbol-Based Turbo Codes ($M = 4$, $N = 192$) With Bi-orthogonal Modulation over an AWGN Channel

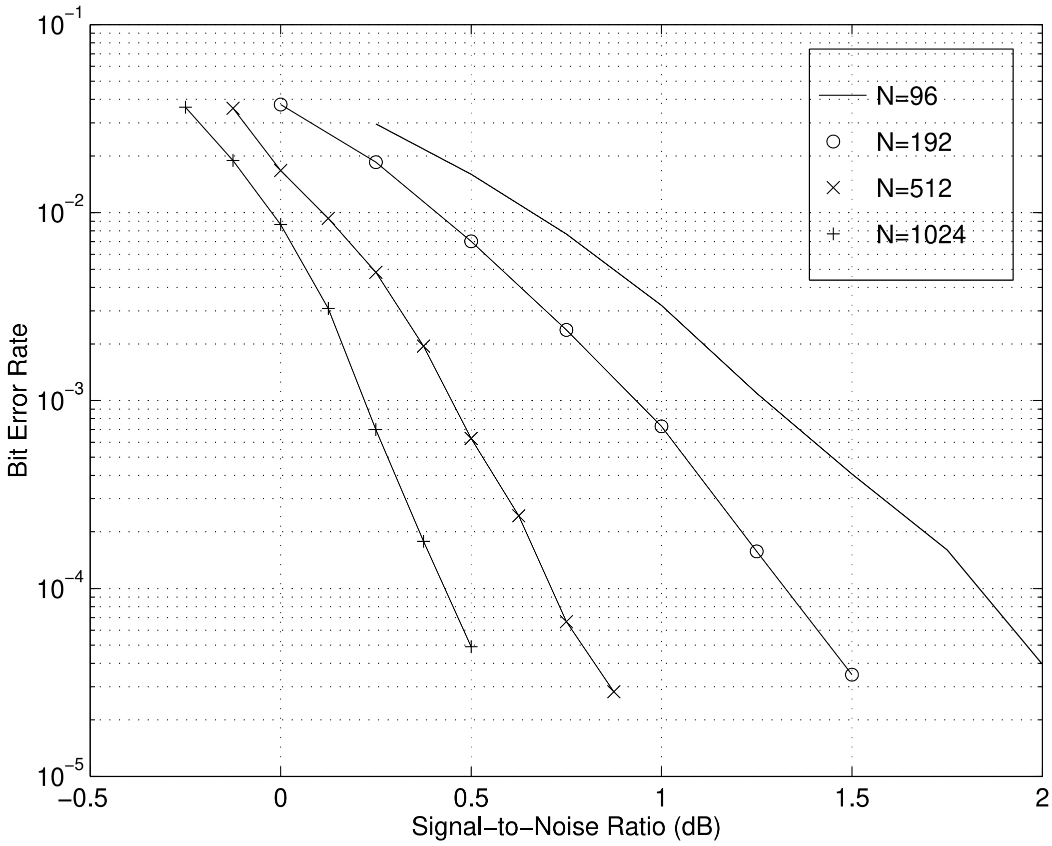


Figure 5.4: BER Performance of the Rate 1/8 Symbol-Based Turbo Code ($M = 4$) With Bi-orthogonal Modulation over an AWGN Channel for Various Block Lengths

$n = 6$, a BER of 10^{-3} is obtained at an SNR of 0.30 dB. As a means of comparison, the Symbol-Based Turbo Code with bi-orthogonal modulation for a symbol size $n = 3$ (rate 1/8) were also simulated for other block lengths, N, and are shown in Figure 5.4. A BER of 10^{-3} is obtained at an SNR of 1.27 dB for $N = 96$, at 0.93 dB for $N = 192$, at 0.45 dB for $N = 512$, and at 0.22 dB for $N = 1024$.

5.1.4 BPSK Modulation

The Symbol-Based Turbo Codes with n-dimensional BPSK modulation were also simulated. The BER performance of these codes improve only slightly (less than 0.1 dB), by increasing the the symbol size from $n = 1$ to $n = 2$. For $n = 3, 4, \ldots$, the BER performance is essentially the same as the $n = 2$ code. However, the strength of these codes can be seen when we consider the 5 and 10 iteration BER performance for symbol sizes $n = 1, 2$. Figure 5.5 shows the BER performance for the Symbol-Based Turbo Code ($M = 4$, $R = 1/3$), with BPSK modulation for $i = 5$, 10 iterations and block sizes $N = 192$, 512. For both block sizes, the Symbol-Based Turbo Code with symbol size $n = 2$ requires about 5 iterations to obtain the same BER performance as the traditional Turbo Code ($n = 1$) with 10 iterations[1].

The same trend is also observed for the $M = 3$, $R = 1/4$ Symbol-Based Turbo Code with BPSK modulation for a block length of $N = 192$, as shown in Figure 5.6. However, for a block length $N = 512$ about 8 iterations of the $n = 2$ code are required to obtain the same BER performance as the $n = 1$ code with 10 iterations.

[1]These codes are the subject of a North American patent and have been proposed by NORTEL for integration into the emerging 3G-IS95 standard.

Therefore, it is recommended, for large block sizes, that RSC encoders with only one feed-forward output be used in these Symbol-Based Turbo Codes.

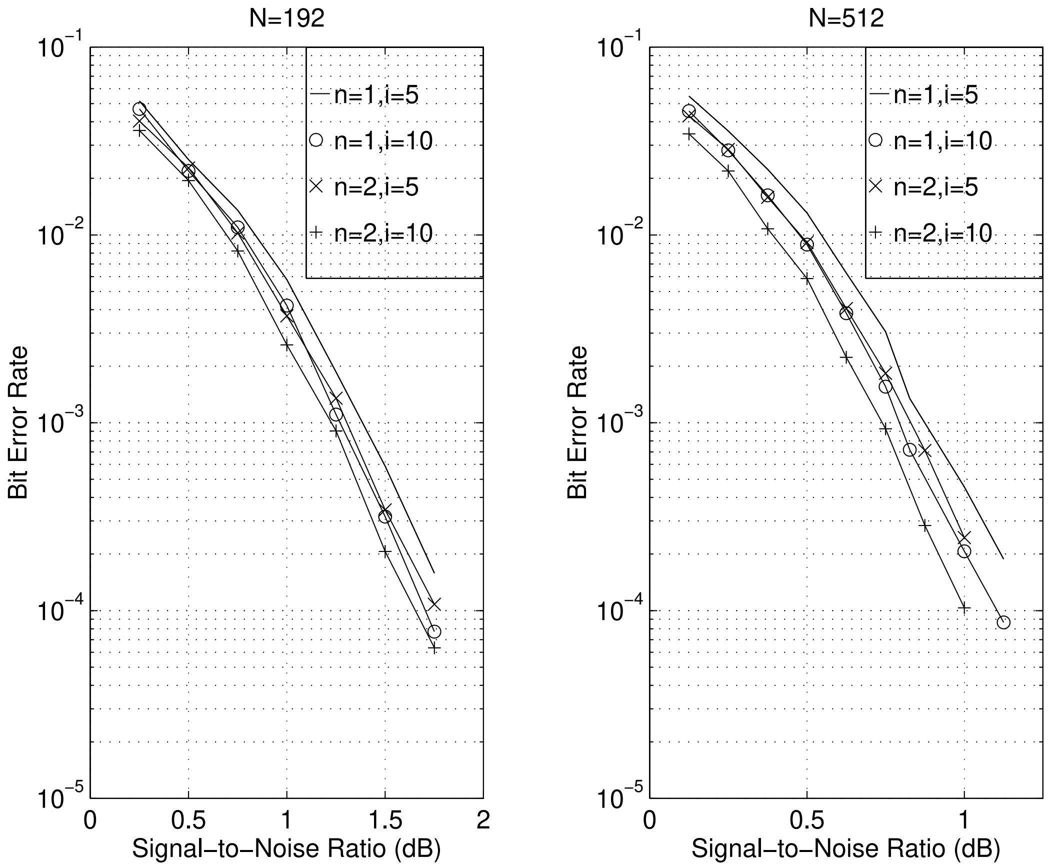


Figure 5.5: BER Performance of Symbol-Based Turbo Code ($M = 4,\ R = 1/3$) with BPSK Modulation for Block Sizes, $N = 192,\ 512$ and $i = 5,\ 10$ Iterations

Figure 5.7 shows the degradation in BER performance of Symbol-Based Turbo Codes with BPSK modulation when less than 20 iterations are performed. Note that as the symbol size increases, the degradation in BER performance decreases. Therefore, we conclude that the the modified BCJR algorithm converges faster as

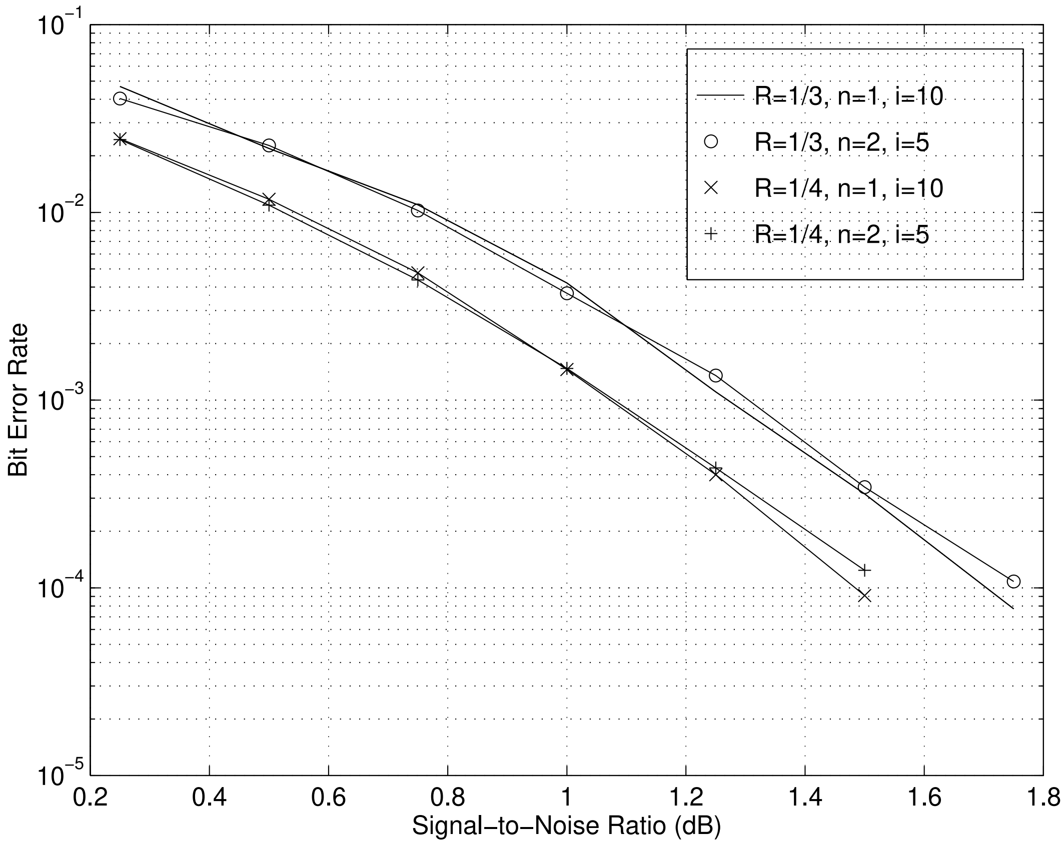


Figure 5.6: BER Performance of Rate 1/3 ($M = 4$) and Rate 1/4 ($M = 3$) Symbol-Based Turbo Codes ($N = 192$) With BPSK Modulation for 5 and 10 Iterations

the symbol size is increased. The same convergence trends are also observed when orthogonal or bi-orthogonal modulation are used. Table 5.1 shows a comparison of the degradation in BER performance for various modulation techniques for symbol size $n = 4$.

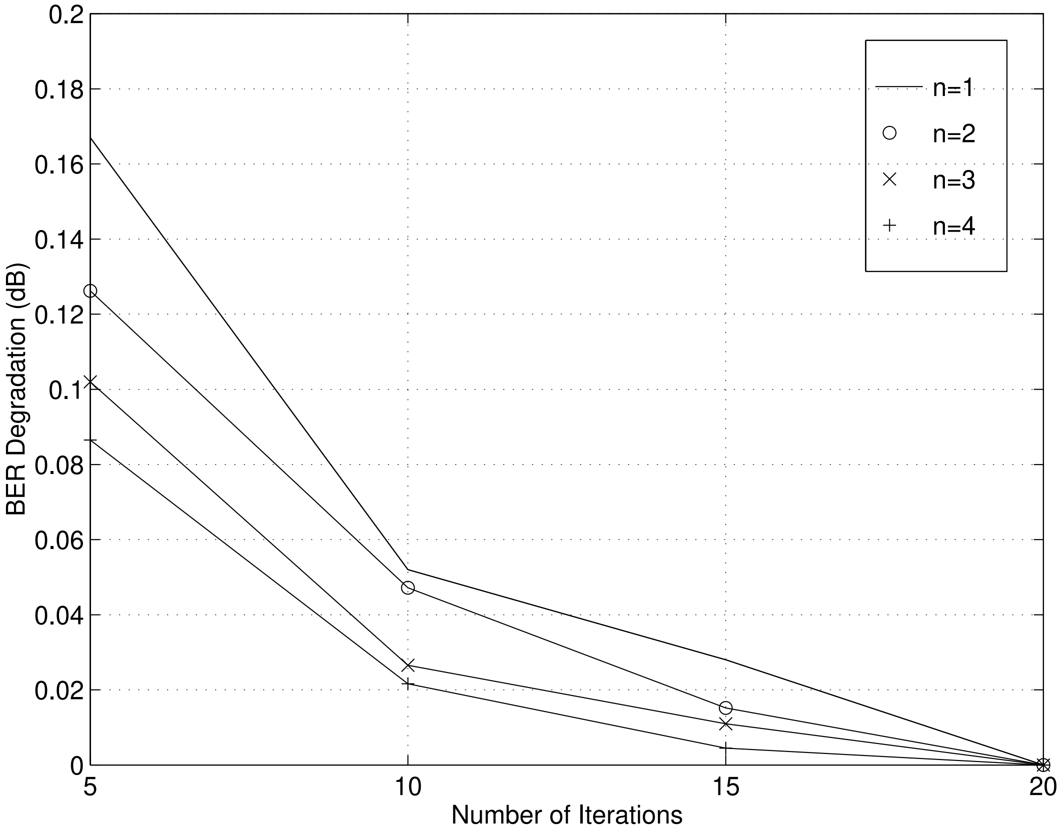


Figure 5.7: Degradation in BER Performance of Rate 1/3 Symbol-Based Turbo Codes ($M = 4$, $N = 192$) With BPSK Modulation vs. Number of Iterations at a BER of 10^{-3}

Table 5.1: BER Performance Degradation of Symbol-Based Turbo Codes as Compared with Symbol Size $n = 4$, $i = 20$ Iterations

Modulation Technique	**BER Degradation (dB)**			
	i=5	**i=10**	**i=15**	**i=20**
Orthogonal	0.0802	0.0200	0.0044	0.0
Bi-orthogonal	0.0701	0.0177	0.0042	0.0
BPSK	0.0865	0.0217	0.0045	0.0

5.1.5 Parity Bits

Symbol-Based Turbo Codes with full parity were the first class of codes we studied which combine Symbol-Based Turbo Codes with parity bit codes. These codes, as discussed in section 3.7, parse the input data into $n-1$ bit sub-blocks and then maps each sub-block to an n-bit symbol by adding a single parity bit. These symbols are then passed through a Symbol-Based Turbo Code encoder. The rates of these codes are $(n-1)/3n$ for BPSK modulation. Figure 5.8 shows the BER performance of these code.

The second second set of codes studied are called Symbol-Based Turbo Code with partial parity. In these codes, the parity bit added to the $n-1$ bit sub-blocks is only passed to the encoded data streams and not to the systematic data stream. Figure 3.5 shows the structure of this encoder. The rates of these codes are $(n-1)/(3n-1)$ for BPSK modulation. The BER performance of these codes is shown in Figure 5.9.

For both parity schemes, the $n = 2$ code is essentially a repetition code. These codes perform worse than the traditional Turbo Code with no parity. For $n = 3, 4$ both parity codes perform about 0.2 dB better than the traditional Turbo Code.

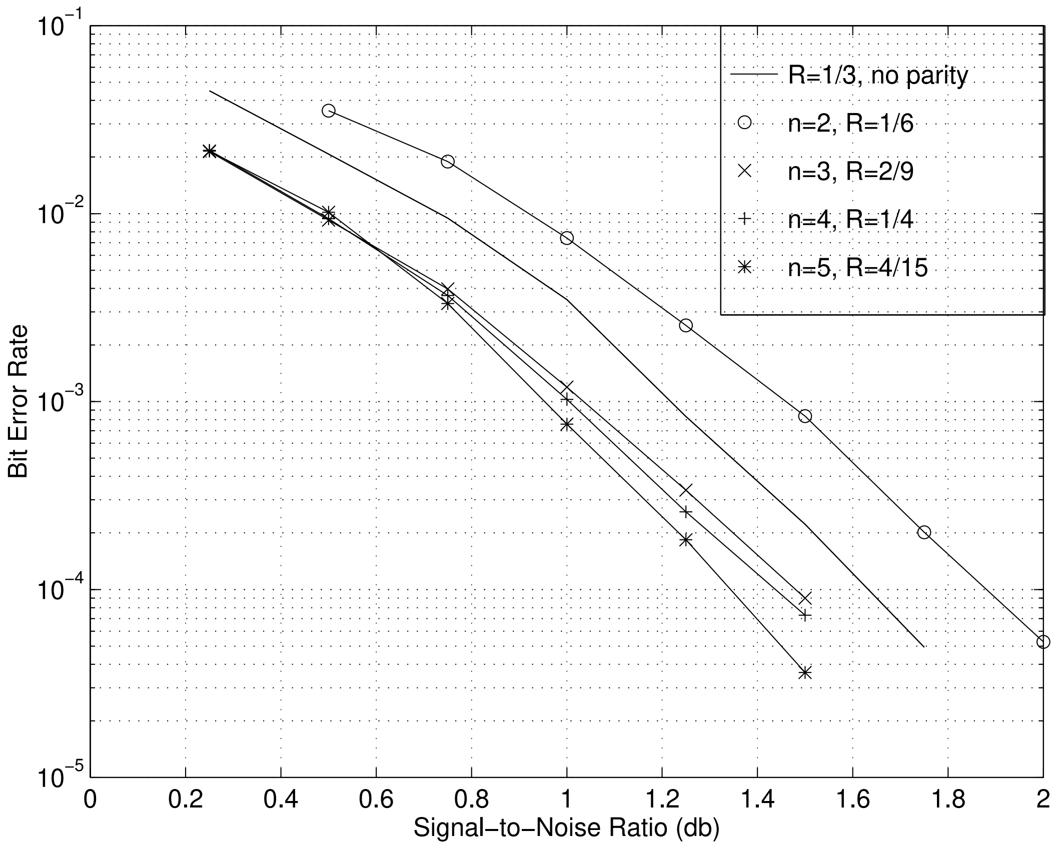


Figure 5.8: BER Performance of Symbol-Based Turbo Codes ($M = 4$, $N = 192$) With BPSK Modulation and Full Parity

However, for a rate $R = 1/4$ code, the full parity code requires a symbol size of $n = 4$, while the partial parity code only needs a symbol size of $n = 3$. Since the decoder complexity is a function of the symbol size n, as will be discussed in section 5.2, the partial parity code can obtain the same BER performance as the full parity code, but with lower decoder complexity.

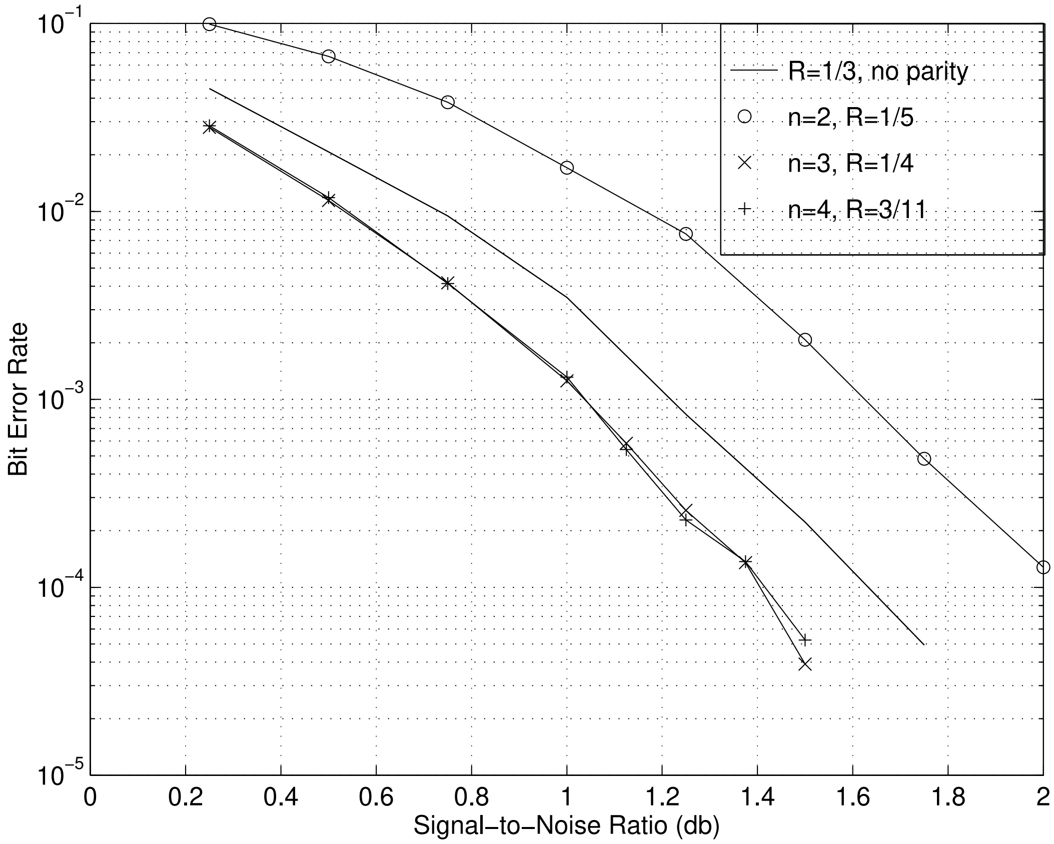


Figure 5.9: BER Performance of Symbol-Based Turbo Codes ($M = 4$, $N = 192$) With BPSK Modulation and Partial Parity

5.2 Complexity and Memory

The computational complexity and the memory requirements of the Symbol-Based Turbo Code decoder are examined in Appendix A. The results of this analysis are discussed in this section.

In Symbol-Based Turbo Codes, the merging of n sections of the RSC trellis result in a reduction in the effective block length by a factor of $1/n$. This shorter effective block length means that fewer $\alpha_k(m)$ values from the forward recursion of the modified BCJR algorithm need to be stored. Similarly, the shorter effective block length means that the modified BCJR algorithm is calculated over fewer stages, thus resulting in a reduction in the number of computations.

On the other hand, the merged trellis diagram has more branches leaving each state (specifically 2^n branches). These extra branches increase the number of calculations required per stage. Furthermore, these extra branches translate into more LLR values that need to be stored. Lastly, the use of a multi-dimensional signal space results in more complex branch metric calculations and requires more memory storage. The memory requirements and computational complexity of the Symbol-Based Turbo Code decoder relative to the traditional Turbo Code, $n = 1$, are shown in Table 5.2.

Table 5.2 shows that the relative RAM (Random Access Memory) requirements of the Symbol-Based Turbo Code decoder are less than the traditional Turbo Code. Furthermore, the ROM (Read Only Memory) used to store the interleaver structure is also reduced by a factor of $1/n$.

The relative computational complexity is roughly given by the expression $2^{n-1}/n$.

Table 5.2: Relative Memory Requirements and Per Iteration Computational Complexity of the Symbol-Based Turbo Code Decoder

	Memory Size (M)	Symbol Size (n)			
		1	2	3	4
Memory Requirements	3	1.00	0.78	0.88	—
	4	1.00	0.69	0.69	0.86
Computational Complexity	3	1.00	1.05	1.45	—
	4	1.00	1.05	1.43	2.18

This result is intuitively satisfying because a merged trellis diagram has 2^n branches leaving each state, and one stage of the trellis corresponds to an n-bit input symbol. The traditional Turbo Code, $n = 1$, has 2 branches leaving each state of the RSC trellis, and one stage of the this trellis corresponds to a single input bit. Thus $(2^n/n)/(2/1) = 2^{n-1}/n$.

Therefore, the computational complexity per iteration for symbol sizes $n = 1, 2$ are approximately the same. However, as shown in Figure 5.5 and Figure 5.6, Symbol-Based Turbo Codes with BPSK modulation and symbol size $n = 2$ require approximately 5 iterations to obtain the same BER performance of the traditional Turbo Code, $n = 1$, with 10 iterations. Overall, Symbol-Based Turbo Codes with symbol size $n = 2$ have an improved BER performance for the same number of decoding iterations (or equivalently a reduction in the number of iterations for the same BER performance), as well as a reduction in the decoder memory requirements as compared with the traditional Turbo Code, $n = 1$, at no extra cost. For larger symbol sizes $n = 3, 4$, the increase in computational complexity is partially compensated by a reduction in the number of iterations required.

Table 5.3: Number of Decoding Operations and Words of RAM Memory Required for the Symbol-Based Turbo Code (5 iterations) and a Convolutional Code ($K = 9$) Per Input Bit

	Memory Size (M)	**Symbol Size (n)**		**Convolutional Code ($K = 9$)**
		1	**2**	
Computations	3	2040	2150	768
RAM Size		16.0	12.5	256
Computations	4	3960	4150	768
RAM Size		24	16.5	256

Table 5.3 compares the computational complexity and memory requirements of the Symbol-Based Turbo Code decoder with the Viterbi decoder for a convolutional code with constraint length $K = 9$.

The Symbol-Based Turbo Code decoder with 5 iterations require more computations than the Viterbi decoder for the constraint length $K = 9$ convolutional code, specifically, 2.7 - 5.4 times more, depending on the symbol size, n, and the number of memory elements, M, in the RSC encoders. However, the Symbol-Based Turbo Code decoder requires much less RAM memory storage than the Viterbi decoder for the constraint length $K = 9$ convolutional code.

Chapter 6

Conclusions

This thesis studies the combination of Turbo Codes with different modulation techniques to take advantage of inherent trade-offs between BER performance, code rate, spectral efficiency, and decoder complexity. The encoder and decoder structure of our coding method, which we call, *Symbol-Based Turbo Codes*, were presented. The BER performance of Symbol-Based Turbo Codes with orthogonal, bi-orthogonal, M-ary PSK, and BPSK modulation were simulated and presented. The decoder memory requirements and computational complexity was also analyzed. The following observations about Symbol-Based Turbo Codes are noted:

- Symbol-Based Turbo Coding is an effective method to combine Turbo Codes with different modulation techniques.
- Low rate Symbol-Based Turbo Codes which use orthogonal or bi-orthogonal modulation display an improvement in BER performance as the symbol size is increased. Furthermore, these codes perform better than constraint length 9

convolutional codes with the same modulation techniques, specifically, 1.4 dB better for an AWGN channel, and 2.9 dB better for a Rayleigh fading channel in the orthogonal modulation case.

- The high-rate Symbol-Based Turbo Codes with M-ary PSK modulation perform about 0.7 dB better than convolutional codes of the same rate and similar decoding complexity over an AWGN channel and about 2.2 dB better over a Rayleigh fading channel.

- The BER performance of the Symbol-Based Turbo Codes studied only degrade 1.25 - 1.5 dB between an AWGN channel and Rayleigh fading channel. In comparison, the convolutional codes displayed a 3 - 4 dB degradation.

- The Symbol-Based Turbo Code decoding algorithm converges faster as the symbol size, n, increases for BPSK, orthogonal and bi-orthogonal modulation techniques.

- The Symbol-Based Turbo Code ($M = 4$, $R = 1/3$) with BPSK modulation and a symbol size $n = 2$ requires only 5 iterations to obtain the same BER performance as the traditional, $n = 1$, Turbo Code with 10 iterations. This trend is observed for both small and large block sizes. The $M = 3$, $R = 1/4$ Symbol Based Turbo Codes also display this feature for small block sizes (i.e. $N = 192$), but not for large block sizes.

- The Symbol-Based Turbo Code decoder requires less memory than the traditional Turbo Code decoder. Specifically, only 69 - 88% of the memory needed by the traditional Turbo Code decoder is required for the Symbol-Based Turbo

Code decoder, depending on the memory size of the RSC encoders (M) and the symbol size (n).

- The relative per iteration computational complexity of the Symbol-Based Turbo Code decoder as compared with the traditional Turbo Code, $n = 1$, is roughly given by the expression $2^{n-1}/n$. Therefore, the per iteration complexity of the $n = 2$ Symbol-Based Turbo Code is approximately the same as the traditional Turbo Code ($n = 1$). However, for symbol size $n = 2$, the code requires fewer iterations. Therefore the over-all complexity and memory requirements of the $n = 2$ code is less than the traditional Turbo Code and is obtained at no extra cost.

- Symbol-Based Turbo Codes with BPSK modulation and partial parity provide a good trade-off between BER performance and computational complexity. For example, the $n = 4$ code with full parity performs about the same as the $n = 3$ partial parity code. Although these codes have the same rate (1/4), the computational complexity of the partial parity code is less than the full parity code.

- Symbol-Based Turbo Codes are good candidates for spread spectrum communication systems such as CDMA. Specifically, the Symbol-Based Turbo Code with orthogonal modulation can be used for a non-coherent up-link, or bi-orthogonal modulation for a coherent up-link. Furthermore, Symbol-Based Turbo Codes with either M-ary PSK modulation or BPSK modulation can be used for a coherent down-link.

Future Research Directions

The future research directions in the study of Symbol-Based Turbo Codes are as follows:

- Examine the BER performance for a more realistic fading channel model (i.e. Jake's model [29].)
- Study the degradation in BER performance when sub-optimal RSC decoders are used.
- Investigate the use of other modulation techniques and/or signal constellations.
- Perform the simulations to obtain the BER performance curves for BERs below 10^{-5} in order to study the shallow portion of the BER curves. It is suspected that the performance in this area of the BER curves will degrade slightly when the symbol size is increased.

The following are a few future research directions for issues related to Turbo Codes in general:

- The modified BCJR algorithm assumes that the receiver can accurately determine the E_b/N_0 ratio of the channel. The BER performance of a Turbo Code needs to be examined for inaccurate estimations of this ratio. For example, and estimation of aE_b/N_0 for $0.25 \leq a \leq 4$.
- The modified BCJR algorithm can accurately decode an RSC code if the path through the trellis consists of a few strong links periodically spaced

throughout the length of the block. A strong link refers to a stage in the trellis where a single path is much more likely than all the other paths combined. If these strong links are placed, say about 20 bits apart, the modified BCJR algorithm can more easily determine the correct path between the strong links. This property can be taken advantage of by intentionally introducing strong links (i.e. by doubling the transmitted energy of every, say 20 bits). The optimal choice of energy increase and span between strong links needs to be examined.

- Implement the modified BCJR algorithm using matrices and take advantage of sparse matrix manipulation techniques.

- Use parity bit codes with Tanner graph representations for the constituent encoders and decode using the forward-backward algorithm for Tanner graphs discussed in [30].

Appendix A

Complexity and Memory Calculations

Consider the Symbol-Based Turbo Code encoder discussed in section 3.1 with symbol size n and constituent RSC encoders with M memory elements. The merged trellis diagram has 2^M states, and 2^n branches leaving each state. The parallel data streams are transmitted using the J-dimensional modulation vectors $\vec{m}_i = (m_{i,1}, m_{i,2}, \ldots, m_{i,J})$.

The modified BCJR algorithm discussed in section 3.2 can be implemented differently depending on the desired trade-off between computational complexity and memory requirements. The decoder can be implemented to minimize the number of computations at the expense of large memory requirements. Alternatively, the decoder can be implemented to minimize the memory requirements at the expense of a great increase in the number of computations. We propose a trade-off which results in memory requirements and computational complexity which are both close to their minimum values.

$\gamma_i(R_k, m')$ Calculations

The $\gamma_i(R_k, m')$ calculations is the primary location in the modified BCJR algorithm where a trade-off between memory storage and computational complexity can be made. As discussed in section 3.2, the $\gamma_i(R_k, m')$ values are given by the following expression:

$$\gamma_i(R_k, m') = Pr\{\vec{x}_k | S_{k-1} = m', d_k = i\} Pr\{\vec{y}_k | S_{k-1} = m', d_k = i\} Pr\{d_k\} \quad \text{(A.1)}$$

The first two terms are the branch metrics for the systematic and parity data streams and the third term is the APP values of the input symbol d_k. For an AWGN channel, the branch metrics are given the following expressions:

$$Pr\{\vec{x}_k | S_{k-1} = m', d_k = i\} = \prod_{j=1}^{J} \exp\left(\frac{-(x_{k,j} - m_{i,j}(X))^2}{N_0}\right) \quad \text{(A.2)}$$

$$Pr\{\vec{y}_k | S_{k-1} = m', d_k = i\} = \prod_{j=1}^{J} \exp\left(\frac{-(y_{k,j} - m_{i,j}(Y))^2}{N_0}\right) \quad \text{(A.3)}$$

A compromise between memory storage and computational complexity can be made if the branch metrics are calculated for each of the parallel data streams $x, y1, y2$ and stored. In the forward and backward equations, the appropriate branch metrics are multiplied together along with the current APP values to form the $\gamma_i(R_k, m')$ values. The calculation of equation A.2, requires 1 subtraction and 1 multiplication for each exponential term, $J-1$ multiplications to form the product, and 1 multiplication to include the common factor e^{N_0}. Equation A.2 needs to be calculated for each of the 2^n possible values of the input symbol d_k. Furthermore, this calculation needs to be performed for each of the parallel data streams. Therefore, the branch metrics require $3(2^n)(3J)$ calculations.

Intrinsic Information

A general expression for the intrinsic information is as follows:

$$\log\left(\frac{Pr\{\vec{x}_k|d_k=i\}}{Pr\{\vec{x}_k|d_k=0\}}\right) = \frac{1}{N_0}\sum_{j=1}^{J}[2x_{k,j}(m_{i,j}-m_{0,j})+(m_{0,j}^2-m_{i,j}^2)] \tag{A.4}$$

This expression requires 3 additions (or subtractions) and 1 multiplication for each term in the summation (assuming that the $m_{i,j}^2$ are pre-calculated), $J-1$ additions to form the sum, and 1 multiplication for the common factor $1/N_0$. Since equation A.4 needs to be calculated for each of the 2^n possible values of the input symbol d_k, the resulting number of computations to calculate the intrinsic information is $(2^n)(5J)$. However, as discussed in sections 3.3 – 3.6, the general expression for the intrinsic information can be further simplified. This simplification reduces the number of calculations to $10(2^n)$ for orthogonal and bi-orthogonal modulation, $5(2^n)$ for M-ary PSK modulation, and $n(2^n)$ for BPSK modulation.

$\alpha_k(m)$ Calculations

The $\alpha_k(m)$ values of the forward recursion are given by the following expression:

$$\alpha_k(m) = \frac{\sum_{m'} \sum_{i=0}^{2^n-1} \gamma_i(R_k, m', m)\alpha_{k-1}(m')}{\sum_m \sum_{m'} \sum_{i=0}^{2^n-1} \gamma_i(R_k, m', m)\alpha_{k-1}(m')} \tag{A.5}$$

This expression requires 3 multiplications to form each term in the numerator because the $\gamma_i(R_k, m', m)$ values consist of 3 terms, and thus require 2 multiplications. To form the sum in the numerator $(2^n - 1)$ additions are required. These calculations must be performed for each of the 2^M states. The calculation of the common denominator term can be omitted to reduce the computational complexity with only a negligible reduction in BER performance. Therefore the total number of calculations required is $3(2^{M+n}) + 2^M(2^n - 1)$.

$\beta_k(m)$ Calculations

The $\beta_k(m)$ values of the backward recursion are given by the following expression:

$$\beta_k(m) = \frac{\sum_{m'} \sum_{i=0}^{2^n-1} \gamma_i(R_{k+1}, m, m')\beta_{k+1}(m')}{\sum_m \sum_{m'} \sum_{i=0}^{2^n-1} \gamma_i(R_{k+1}, m', m)\alpha_k(m')} \tag{A.6}$$

Using a similar analysis as for the $\alpha_k(m)$ calculations, it can be shown that the $\beta_k(m)$ values also require $3(2^{M+n}) + 2^M(2^n - 1)$ calculations if the common denominator term is omitted.

$\Delta_k(i)$ Calculations

The Δ values are given by the expression,

$$\Delta_k(i) = \log\left(\frac{\sum_m \sum_{m'} \gamma_i(R_k, m', m)\alpha_{k-1}(m')\beta_k(m)}{\sum_m \sum_{m'} \gamma_0(R_k, m', m)\alpha_{k-1}(m')\beta_k(m)}\right). \tag{A.7}$$

The central term requires 4 multiplications, and needs to be calculated for each of the branches in the trellis diagram (2^{M+n}). Each summation is performed over all the branches with the same input symbol, thus ($2^M - 1$) additions are required for each of the 2^n input symbol values. Lastly, 2^n divisions are required to complete the expressions. (The log() function is implemented using a look-up table and is not included in the number of calculations required.) Therefore, the $\Delta_k(i)$ values require $4(2^{M+n}) + 2^n(2^M - 1) + 2^n$ calculations.

Miscellaneous Calculations

The remaining calculations in the modified BCJR algorithm deal with the manipulation of the LLR values. Firstly, the conversion of LLR values to APP values requires $2^n - 1$ additions and 2^n divisions. In order to reduce the memory storage requirements, the APP values are calculated in both the forward and backward recursions instead of being calculated once and stored. Secondly, the calculation of the extrinsic component of the LLR values requires $2(2^n)$ subtractions. Thirdly, the final hard-decision decoding is performed by selecting the largest LLR value. This decoding requires 2^n calculations.

Decoder Complexity

All of the analysis of computational complexity in the previous sections is based on the number of computations required per stage of the merged trellis. In order to compare computational complexities, each expression must be divides by n to obtain the number of computations required per input bit.

The calculation of the $\gamma_i(R_k, m')$ values and the final hard decision are both overhead calculations that only need to be performed once. The intrinsic information can also be calculated only once and stored, but this increases the memory storage requirements. If the intrinsic information is calculated each time it is needed, then the memory storage requirements will be reduced with only a 1-4% increase in computational complexity. For this reason, the intrinsic information is not performed as an overhead calculation.

The remaining calculations must be performed by each of the RSC decoders at every iteration. Table A.1 shows the number of computations required for the overhead and per iteration calculations. The number of per iteration calculations is twice the number of calculations required for each RSC decoder. Since the overhead calculations are a function of the the dimensionality of the signal space (J), Table A.1 reflects the values for BPSK modulation. The number of overhead calculations will be slightly higher for other modulation schemes.

Note that the number of overhead calculations required is not a function of the number of memory elements. It is only a function of the modulation scheme and the symbol size. Furthermore, for a decoder that performs $5-10$ iterations, the over-all number of calculations is dominated by the per iteration calculations, thus

Table A.1: Number of Calculations Performed by the Symbol-Based Turbo Code Decoder Per Input Bit

Calculation Type	**Memory Size (M)**	**Symbol Size (n)** 1	2	3	4
Overhead	3	20	38	75	—
Per Iteration		408	430	591	—
Overhead	4	20	38	75	148
Per Iteration		792	830	1135	1727

the number of overhead calculation can ignored without greatly changing the result.

The relative number of per iteration calculations is roughly given by the expression $2^{n-1}/n$. This result is intuitively satisfying because a merged trellis diagram has 2^n branches leaving each state, and consist of n input bits. The traditional Turbo Code, $n = 1$, has 2 branches leaving each state of the RSC trellis, and one stage of the trellis corresponds to a single input bit. Therefore $(2^n/n)/(2/1) = 2^{n-1}/n$. If the common denominator terms are included in the forward and backward recursions, the relative computational complexity very closely follow this expression. Without the common denominator terms, the relative computational complexity only roughly follows this expression.

Memory Requirements

The trade-off between computational complexity and memory requirements presented in this appendix focuses on minimizing the memory requirements as long as doing so will not dramatically increase the computational complexity. Therefore, memory storage is required for the branch metrics, the LLR values from the previous decoder, the $\alpha_k(m)$ values of the forward recursion, and the received systematic data stream for the intrinsic information calculation.

The branch metrics require 2^n units of memory storage per trellis stage for each of the parallel data streams $x, y1, y2$. The LLR values require $2^n - 1$ units of memory storage. (There are 2^n LLR values, but since $\Delta_k(0) = 0$, it does not need to be stored.) The $\alpha_k(m)$ values of the the forward recursion require 2^M units of memory storage.

A trade-off between memory storage and computational complexity can be made by either calculating the intrinsic information once, and storing it, or store the received systematic symbols $\vec{x}_k$ and calculate the intrinsic information when needed. In the first case, the intrinsic information calculation becomes an overhead calculations, but requires 2^n units of memory storage. In the second case, the memory storage for the systematic data stream is J/n units per input bit, and the per iteration computational complexity only increases $1 - 4\%$ depending on the modulation scheme used. The second case was chosen to reduce the memory storage requirements. Therefore, $(1/n)(3(2^n) + (2^n - 1) + 2^M + J)$ units of memory storage are required. Table A.2 shows the memory storage requirements of a Symbol-Based Turbo Code decoder when BPSK modulation is used.

Table A.2: Memory Requirements of the Symbol-Based Turbo Code Decoder Per Input Bit

Memory	**Symbol Size (n)**			
Size (M)	1	2	3	4
3	16.0	12.5	14.0	—
4	24.0	16.5	16.7	20.8

Bibliography

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes (1)", *Proceedings of IEEE International Conference on Communications*, Geneva, Switzerland, pp. 1064-1070, May 1993.

[2] C. E. Shannon, "A Mathematical Theory of Communication", *Bell Syst. Tech. Journal*, vol. 27, pp. 379-423 and 623-656, 1948.

[3] P. Robertson, "Illuminating the Structure of Code and Decoder of Parallel Concatenated Recursive Systematic (Turbo) Codes", *Proceedings of IEEE GLOBECOM'95*, San Francisco, California, pp. 1298-1303, December 1994.

[4] D. Divsalar and F. Pollara, "Turbo Codes for Deep-Space Communications", *The Telecommunications and Data Acquisition Progress Report 42-120, October-December 1994*, Jet Propulsion Laboratory, Pasadena, California, pp. 29-39, February 15, 1995.

[5] S. Le Goff, A. Glavieux, and C. Berrou, "Turbo-Codes and High Spectral Efficiency Modulation", *Proceedings of IEEE International Conference on Com-*

munications, New Orleans, LA. pp. 645-649, May 1-5, 1994.

[6] R. Pyndiah, A. Picart, and A. Glavieux, "Performance of Block Turbo Coded 16-QAM and 64-QAM Modulations", *Proceedings of GLOBECOM'95*, San Francisco, California, pp. 1039-1043.

[7] S. Benedetto, D. Divsalar, G. Montorsi, F. Pollara, "Parallel Concatenated Trellis Coded Modulation", *Proceedings of IEEE International Conference on Communications*, Dallas, Texas USA, pp. 974-978, June 23-27, 1996.

[8] D. Divsalar and F. Pollara, "Multiple Turbo Codes for Deep-Space Communications", *The Telecommunications and Data Acquisition Progress Report 42-121, January-March 1995*, Jet Propulsion Laboratory, Pasadena, California, pp. 66-77, May 15, 1995.

[9] K. Pehkonen and P. Komulainen, "A Superorthongonal Turbo-code for CDMA Applications", *Proceedings of International Symposium on Spread Spectrum Techniques and Applications*, pp. 580-584, 1996.

[10] N. Chayat, "Turbo Codes for Incoherent M-ary Orthogonal Signaling", *Proceedings 19th Convention of Electrical and Electronics Engineers in Israel*, pp. 471-474, 1996.

[11] S.W. Golomb, *Shift Register Sequences*, Holden-Day, San Francisco, CA, 1967.

[12] D. Laksov, "Linear Recurring Sequences Over Finite Fields", *MATH. SCAND.*, vol. 16, pp. 181-196, 1965.

[13] M. Hall, "An Isomorphism Between Linear Recurring Sequences and Algebraic Rings", *Trans. Amer. Math. Soc.*, vol. 44, pp 196-218, 1938.

[14] M. Ward, "The Characteristic Number of a Sequence of Integers Satisfying a Linear Recursion Relation", *Trans. Amer. Math. Soc.*, vol. 33, pp. 153-165, 1931.

[15] M. Ward, "The Distribution of Residues in a Sequence Satisfying a Linear Recursion Relation", *Trans. Amer. Math. Soc.*, vol. 33, pp. 166-190, 1931.

[16] G. Battail, C. Berrou and A. Glavieux, "Pseudo-Random Recursive Convolutional Coding for Near-Capacity Performance", *Proceedings of IEEE GLOBECOM, Communication Theory Mini-Conference*, pp. 23-7, Nov. 1993.

[17] S. Benedetto and G. Montorsi, "Unveiling Turbo Codes: Some Results on Parallel Concatenated Coding Schemes", *IEEE Trans. on Info. Theory*, vol. 42, no. 2, March 1996, pp. 409-428.

[18] A.K. Khandani, "Algebraic Structure of Turbo-code", *Proceedings of 19th Biennial Conference on Communications*, Queens University, Kingston, Ontario, Canada, pp. 70-74, June 1998.

[19] Béla Krekó, *Linear programming,* translated by J. H. L. Ahrens and C. M. Safe, Sir Isaac Pitman & Sons Ltd., 1968.

[20] S. Dolinar and D. Divsalar, "Weight Distributions for Turbo Codes Using Random and Nonrandom Permutations", *The Telecommunications and Data Ac-*

quisition Progress Report 42-122, April-June 1995, Jet Propulsion Laboratory, Pasadena, California, pp. 56-65, August 15 1995.

[21] J. D. Andersen and V.V. Zyablov, "Interleaver design for turbo coding", *Proceedings of International Symposium on Turbo Codes and Related Topics*, Brest, France, pp. 154-6, Sept. 1997.

[22] F. Daneshgaran and M. Mondin, "Design of interleavers for turbo codes based on a cost function", *Proceedings of International Symposium on Turbo Codes and Related Topics*, Brest, France, pp. 255-8, Sept. 1997.

[23] J. Hokfelt and T. Maseng, "Methodical interleaver design for turbo codes", *Proceedings of International Symposium on Turbo Codes and Related Topics*, Brest, France, pp. 212-5, Sept. 1997.

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

[25] P. Jung and M. Nabhan, "Comprehensive Comparison of Turbo-Code Decoders", *Proceedings of IEEE VTC'95*, Chicago, Illinois, pp. 624-628.

[26] S. Benedetto, B. Montorsi, D. Divsalar and F. Pollara, "A Soft-Input Soft-Output Maximum A Posteriori (MAP) Module to Decode Parallel and Serial Concatenated Codes", *The Telecommunications and Data Acquisition Progress Report 42-127, July-September 1996*, Jet Propulsion Laboratory, Pasadena, California, pp. 1-20, November 15 1996.

[27] J. Hagenauer and P. Hoeher, "A Viterbi Algorithm With Soft-Decision Outputs and Its Applications", *Proceedings of GLOBECOM'89*, Dallas, Texas, pp. 47.1.1-47.1.7, November 1989.

[28] M. Bingeman and A. K. Khandani, "Design, Performance Analysis and Decoding of Combined Turbo-code and Modulation", *Proceeding of 19th Biennial Symposium on Communications*, Queens University, Kingston, Ontario, Canada, June 1998.

[29] W. C. Jakes, *Microwave Mobile Communications - second edition*, IEEE Press, New Jersey, USA, pp. 642, 1994.

[30] G. D. Forney Jr., "The Forward-Backward Algorithm", *Proceedings of the Thirty-Fourth annual Allerton conference on communication, control and computing*, Oct. 1997, pp. 432-446.