PUNCTURED PARALLEL AND SERIAL CONCATENATED CONVOLUTIONAL CODES FOR BPSK/QPSK CHANNELS

Omer Fatih Acikel, B.S, M.S.

NMSU-ECE-99-016 December 1999

PUNCTURED PARALLEL AND SERIAL CONCATENATED CONVOLUTIONAL CODES FOR BPSK/QPSK CHANNELS

BY

ÖMER FATİH AÇIKEL, B.S., M.S.

A Dissertation submitted to the Graduate School

in partial fulfillment of the requirements

for the Degree

Doctor of Philosophy

Specialization in: Electrical Engineering

New Mexico State University

Las Cruces, New Mexico

December 1999

"Punctured Parallel and Serial Concatenated Convolutional Codes for BPSK/QPSK Channels," a dissertation prepared by Ömer Fatih Açıkel in partial fulfillment of the requirements for the degree, Doctor of Philosophy, has been approved and accepted by the following:

Timothy J. Pettibone
Dean of the Graduate School

James P. LeBlanc
Chair of the Examining Committee

Date

Committee in charge:

Dr. James P. LeBlanc, Chair

Dr. Douglas S. Kurtz

Dr. Phillip De Léon

Dr. Kwong T. Ng

Dr. William E. Ryan, University of Arizona

DEDICATION

To my wife Patricia,
my parents: Şengül and Fikri
Açıkel, my aunt and uncle
Şenay and Sadettin Erdem,
and my guardian angle
Neal ♡, and to all who
make a difference in
others lives.
♡

ACKNOWLEDGEMENTS

I would like to thank to my advisor Dr. William E. Ryan for his guidance and support through out my research in my graduate studies. I also would like to thank Warner Miller of National Aeronautics and Space Administration (NASA) who supported and encouraged us in our research. I wish to extend thanks to my committee members for their feedback to prepare this disseratation: Dr. James P. LeBlanc, Dr. Douglas S. Kurtz, Dr. Phillip De Léon, and Dr. Kwong T. Ng who supported me in my first year at New Mexico State University.

I would like to thank three special people: Şenay and Sadettin Erdem, and Neal ♡ who believed in me and supported my graduate studies in the US. I am who I am because of my parents, Şengül and Fikri Açıkel: they are my treasure. My lovely wife Patricia's unconditional love, care, and support during my studies made my life a lot easier.

I would like to thank NASA for their support under the Grant 5-1491.

VITA

[redacted]

1984 Graduated from Güzelyurt Kurtuluş High School, Northern Cyprus

1990 B.S. from Istanbul Technical University, Istanbul, Turkey

1996 M.S. from New Mexico State University, Las Cruces, New Mexico

1994–1999 Research Assistant, Klipsch Department of Electrical and Computer Engineering, New Mexico State University, Las Cruces, New Mexico

1997–1999 Teaching Assistant, Klipsch Department of Electrical and Computer Engineering, New Mexico State University, Las Cruces, New Mexico

PUBLICATIONS

Ömer F. Açıkel, William E. Ryan,"High rate serial concatenated convolutional codes on BPSK/QPSK channels," submitted, *IEEE Trans. on Comm.*

Ömer F. Açıkel, William E. Ryan,"Punctured serial concatenated convolutional codes on BPSK/QPSK channels," submitted, *International Conference on Communications*, New Orleans, 2000.

Ömer F. Açıkel, "Implementation issues for high rate turbo codes on BPSK/QPSK channels," accepted, *Globecom*, Rio de Janeiro, Brazil, 1999.

Ömer F. Açıkel and William E. Ryan, "Punctured turbo codes for BPSK/QPSK channels," *IEEE Trans. on Comm.*, vol. 47, no. 9, Sept. 1999.

Ömer F. Açıkel and William E. Ryan, "High rate turbo codes for BPSK/QPSK channels," *International Conference on Communications*, Atlanta, GA, 1998.

Ömer F. Açıkel and William E. Ryan, "Lossless compression of telemetry data," *International Telemetry Conference*, San Diego, CA, 1996.

ABSTRACT

PUNCTURED PARALLEL AND SERIAL CONCATENATED CONVOLUTIONAL CODES FOR BPSK/QPSK CHANNELS

BY

ÖMER FATİH AÇIKEL, B.S., M.S.

Doctor of Philosophy in Engineering

Specialization in Electrical Engineering

New Mexico State University

Las Cruces, New Mexico, 1999

Dr. James P. LeBlanc, Chair

As available bandwidth for communication applications becomes scarce, bandwidth-efficient modulation and coding schemes become ever important. Since their discovery in 1993, turbo codes (parallel concatenated convolutional codes) have been the center of the attention in the coding community because of their bit error rate performance near the Shannon limit. Serial concatenated convolutional codes have also been shown to be as powerful as turbo codes. In this dissertation,

we introduce algorithms for designing bandwidth-efficient rate $r = k/(k+1)$, $k = 2, 3, ..., 16$, parallel and rate 3/4, 7/8, and 15/16 serial concatenated convolutional codes via puncturing for BPSK/QPSK channels.

Both parallel and serial concatenated convolutional codes have an initially steep bit error rate versus signal-to-noise ratio slope (called the "cliff region"). However, this steep slope changes to a moderate slope with increasing signal-to-noise ratio, where the slope is characterized by the weight spectrum of the code. The region after the cliff region is called the "error rate floor" which dominates the behavior of these codes in moderate to high signal-to-noise ratios. Our goal is to design high rate parallel and serial concatenated convolutional codes while minimizing the error rate floor effect. The design algorithm includes an interleaver enhancement procedure and finds the polynomial sets (only for parallel concatenated convolutional codes) and the puncturing schemes that achieve the lowest bit error rate performance around the floor for the code rates of interest.

TABLE OF CONTENTS

LIST OF TABLES xi

LIST OF FIGURES xii

LIST OF ACRONYMS xvi

1 OVERVIEW 1

1.1 Introduction 1

1.2 Channel Coding 3

1.3 Channel Capacity 4

1.4 Bandwidth Efficiency 6

1.5 Convolutional Codes 8

1.5.1 Non-recursive CCs 8

1.5.2 Recursive Systematic CCs 11

1.6 Outline of The Dissertation 16

2 APP DECODING ALGORITHM AND ITS DERIVATIVES 18

2.1 Introduction 18

2.2 The APP Algorithm 20

2.2.1 Modified APP Decoding 24

2.2.2 Iterative (Turbo) Decoding 26

2.3 The Log-MAP Algorithm 30

2.4 Sliding Window-Log-MAP (SW-Log-MAP) Algorithm 33

2.5 The SISO Algorithm 34
2.6 Summary 41
3 PUNCTURED PCCCs FOR BPSK/QPSK CHANNELS 43
3.1 Introduction 43
3.2 Characteristics of PCCCs 45
3.3 The Design Algorithm 47
3.3.1 Design Parameters 48
3.3.2 Algorithm Details 50
3.3.2.1 The Set of RSC Codes 51
3.3.2.2 The Sets $S_t^2(g_1, g_2)$ and $S_t^3(g_1, g_2)$ 51
3.3.2.3 The Interleaver Enhancement Algorithm 52
3.3.2.4 Iterating over (g_1, g_2) and $P(p, q)$ 55
3.4 Design Results 59
3.4.1 $m = 3$ Case 59
3.4.2 $m = 4$ Case 60
3.4.3 Weights of Rates Grater Than 5/6 for $m = 3$ and 4 62
3.5 Computational Complexity Comparison with Riedel's Method . 63
3.6 Applications 65
3.6.1 Low-Orbit-to-Geostationary Satellite Links 66
3.6.2 Rate Compatible Punctured PCCCs (RCP-PCCCs) 67
3.7 Simulation Results 70

3.7.1 $m = 3$ Case 70
3.7.2 $m = 4$ Case 79
3.8 Conclusions 88
4 IMPLEMENTATION ISSUES FOR HIGH RATE PCCCs 91
4.1 Introduction 91
4.2 PCCC Encoders and High Rate PCCCs Considered 92
4.3 Decoding Delay 93
4.4 Quantization 95
4.4.1 Suboptimum Implementations 97
4.4.2 SNR Offset 98
4.5 Results 99
4.6 Conclusions 101
5 PUNCTURED SCCCs FOR BPSK/QPSK CHANNELS 103
5.1 Introduction 103
5.2 Characteristics of SCCCs 105
5.3 The Design Algorithm 109
5.3.1 Design Parameters 109
5.3.1.1 8/8 SCCC 110
5.3.1.2 8/2 and 4/2 SCCCs 111
5.4 Algorithm Details 113
5.4.1 The Sets $R_t^2(o)$ and $R_t^3(o)$ 113

5.4.2 The Interleaver Enhancement Algorithm 115
5.4.2.1 The Interleaver Design Algorithm 117
5.4.2.2 Iterating over $P(p_o, p_i)$ or $P(p_o)$ 119
5.5 Design Results 120
5.5.1 Simulation Results for $r = 3/4$ SCCCs 121
5.5.2 Simulation Results for $r = 7/8$ SCCCs 124
5.5.3 Simulation Results for $r = 15/16$ SCCCs 126
5.6 Performance Comparison Between 8/2 SCCCs and $m = 3$ PCCCs 127
5.7 Conclusions 132
6 CONCLUSIONS AND RECOMMENDATIONS 134
6.1 Conclusions 134
6.2 Suggestions for Future Research 137
REFERENCES 140

LIST OF TABLES

3.1 The design results for rates 2/3, 3/4, and 4/5, $m = 3$. 60

3.2 The design results for rates 2/3, 3/4, and 4/5, $m = 4$. 61

3.3 $d_{2,min}^{PCCC*}$, $d_{3,min}^{PCCC*}$ values for the rates greater than 4/5. 62

3.4 Required E_b/N_0 values in dB at $P_b = 10^{-5}$ for $m = 3$ and 4 (distance from capacity in parentheses). 88

4.1 The optimal step sizes for selected quantization levels to minimize BER. 97

5.1 The interleaver and input block sizes and the rates of the outer and inner encoders for the rates of interest. 107

5.2 The design parameters found for $r = 3/4$ SCCCs along with some other parameters used in BER simulations. 121

5.3 The design parameters found for $r = 7/8$ SCCCs along with some other parameters used in BER simulations. 124

5.4 The design parameters found for $r = 15/16$ SCCCs along with some other parameters used in BER simulations. 126

LIST OF FIGURES

1.1 The basic elements of a digital communication system. 1

1.2 Capacity in bits/channel use vs. E_b/N_0 in dB for BPSK channels. 6

1.3 Rate 1/2 (7,5) NRC encoder. 9

1.4 State diagram of $r = 1/2$ $(7, 5)$ NRC encoder. 10

1.5 Trellis diagram of $r = 1/2$ $(7, 5)$ NRC encoder. 11

1.6 Rate 1/2 (7,5) RSC encoder. 11

1.7 PCCC encoder consists of two rate 1/2 RSC encoders with AWGN channel. 13

1.8 SCCC encoder constructed by a rate 1/2 outer and a rate 1 inner encoders with AWGN channel. 14

2.1 Variables calculated recursively are shown in a trellis diagram. . . 23

2.2 Iterative decoding of PCCCs with component APP decoders. . . . 27

2.3 SISO module. 36

2.4 SCCC decoder with SISO modules. 39

3.1 The outline of the algorithm. 50

3.2 The functionality of the interleaver for the input u_k and the interleaved output u_k^I. 53

3.3 The interleaver enhancement algorithm for weight-two input. . . . 54

3.4 Algorithm for finding $d_{2,min}^{PCCC}$ for each puncturing scheme for a given code. 56

3.5 Code rate assignment on a given E_c/N_0 profile. 68

3.6 Rate decreasing process of RCP-PCCC. Each block represents an RSC parity bit sequence. 69

3.7 Rate 2/3 PCCC BER performance. 71

3.8 Rate 3/4 PCCC BER performance. 72

3.9 Rate 4/5 PCCC BER performance. 72

3.10 Rate 5/6 PCCC BER performance. 73

3.11 Rate 6/7 PCCC BER performance. 73

3.12 Rate 7/8 PCCC BER performance. 74

3.13 Rate 8/9 PCCC BER performance. 74

3.14 Rate 9/10 PCCC BER performance. 75

3.15 Rate 10/11 PCCC BER performance. 75

3.16 Rate 11/12 PCCC BER performance. 76

3.17 Rate 12/13 PCCC BER performance. 76

3.18 Rate 13/14 PCCC BER performance. 77

3.19 Rate 14/15 PCCC BER performance. 77

3.20 Rate 15/16 PCCC BER performance. 78

3.21 Rate 16/17 PCCC BER performance. 78

3.22 Required E_b/N_0 (dB) to achieve 10^{-5} BER for r=2/3 through 16/17 PCCCs ($m = 3$) are compared with capacity for AWGN BPSK/QPSK channel. 79

3.23 Rate 2/3 PCCC BER performance. 80

3.24 Rate 3/4 PCCC BER performance. 80

3.25 Rate 4/5 PCCC BER performance. 81

3.26 Rate 5/6 PCCC BER performance. 81

3.27 Rate 6/7 PCCC BER performance. 82

3.28 Rate 7/8 PCCC BER performance. 82

3.29 Rate 8/9 PCCC BER performance. 83

3.30 Rate 9/10 PCCC BER performance. 83

3.31 Rate 10/11 PCCC BER performance. 84

3.32 Rate 11/12 PCCC BER performance. 84

3.33 Rate 12/13 PCCC BER performance. 85

3.34 Rate 13/14 PCCC BER performance. 85

3.35 Rate 14/15 PCCC BER performance. 86

3.36 Rate 15/16 PCCC BER performance. 86

3.37 Rate 16/17 PCCC BER performance. 87

3.38 Required E_b/N_0 in dB to achieve BER of 10^{-5} for r=2/3 through 16/17 PCCC with $m = 4$ are given to compare with capacity for BPSK/QPSK and AWGN channel. 87

4.1 Performance of rates r=3/4, 7/8, and 15/16 versus sliding window size at a fixed E_b/N_0. 94

4.2 Uniform quantization step size versus BER performance for r=3/4, 8-bit quantization with D=64. 96

4.3 The steps of a suboptimum decoding. 98

4.4 BER performance of rate 7/8 PCCC with Q=8 versus SNR offset in dB for different E_b/N_0 values. 99

4.5 Performance comparison of rates r=3/4, 7/8, and 15/16 for different quantization levels and D. 100

5.1 SCCC encoder constructed by a rate 1/2 outer and inner encoders with AWGN channel. 105

5.2 8/8 SCCC rate 3/4 BER performance. 122

5.3 8/2 SCCC rate 3/4 BER performance. 123

5.4 4/2 SCCC rate 3/4 BER performance. 123

5.5 8/8 SCCC rate 7/8 BER performance. 125

5.6 8/2 SCCC rate 7/8 BER performance. 125

5.7 4/2 SCCC rate 7/8 BER performance. 126

5.8 8/8 SCCC rate 15/16 BER performance. 128

5.9 8/2 SCCC rate 15/16 BER performance. 128

5.10 4/2 SCCC rate 15/16 BER performance. 129

5.11 8/2 SCCC and $m = 3$ PCCC BER performance comparison for rate 3/4. 130

5.12 8/2 SCCC and $m = 3$ PCCC BER performance comparison for rate 7/8. 131

5.13 8/2 SCCC and $m = 3$ PCCC BER performance comparison for rate 15/16. 131

5.14 Error floors for 8/2 SCCCs and $m = 3$ PCCCs for rates 3/4, 7/8, and 15/16. 132

LIST OF ACRONYMS

ACK	Acknowlegment
APP	*a posteriori probability*
ARQ	Automatic Repeat Request
AWGN	Additive White Gaussian Noise
BCJR	Bahl, Cocke, Jelinek, and Raviv
BER	Bit Error Rate
BPSK	Binary Phase Shift Keying
CC	Convolutional Code
FEC	Forward Error Correcting or Control
GEO	Geostationary-Earth-Orbit (Satellite)
LAPP	Log-*a posteriori probability*
LEO	Low-Earth-Orbit (satellite)
Log-MAP	Logarithmic (domain) MAP
MAP	Maximum a Posteriori
ML	Maximum Likelihood
NAK	Negative Acknowledgment
NRC	Non-recursive Convolutional code
PCCC	Parallel Concatenated Convolutional Code
QPSK	Quadrature Phase Shift Keying

RCPC	Rate Compatible Punctured Convlutional codes
RCP-PCCC	Rate Compatible Punctured PCCC
RS	Reed Solomon
RSC	Recursive Systematic Convolutional Code
SCCC	Serial Concatenated Convolutional Code
SISO	Soft Input Soft Output
SOVA	Soft-Output Viterbi Algorithm
SNR	Signal to Noise Ratio
SW-Log-MAP	Sliding Window Log-MAP
TCM	Trellis Coded Modulation
VA	Viterbi Algorithm

1 OVERVIEW

1.1 Introduction

In this chapter, we will cover the essential concepts and definitions to understand why channel coding is important and necessary as a part of a communication system.

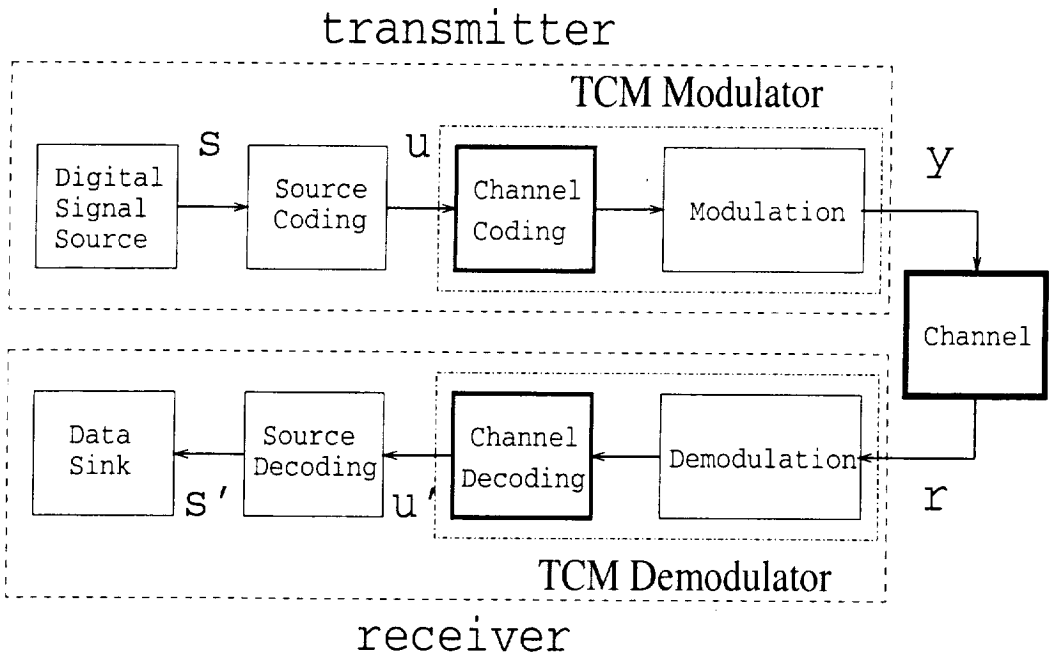

Figure 1.1: The basic elements of a digital communication system.

A block diagram of a typical digital communication system is given in Fig 1.1. The Digital Signal Source can be a combination of an analog source followed by a sampler and an analog-to-digital converter, or it can be a direct digital source. The output of this block, s, takes values from the binary set $\{0, 1\}$. The Source Coding is employed to remove the redundancy in s such that the output u has

the same information as input s with maximum entropy (fewer bits). The next block, Channel Coding, is the primary focus in this dissertation. This block, in contrast to the previous, adds redundancy to u in a systematic way so that by exploitation of this "extra" information, it is possible to detect or even correct some of the errors caused by the channel. The function of the Modulator is to convert the code sequence into a format amenable to transmission over the channel. In some channel coding applications, like trellis coded modulation (TCM), the Channel Coding and Modulation blocks are merged. In this case, at the receiver side, we also have a single block: Demodulation and Channel Decoding. In this dissertation, we study binary Parallel Concatenated Convolutional Codes (PCCCs) and Serial Concatenated Convolutional Codes (SCCCs) which can be designed independently of the modulator when BPSK/QPSK is employed: separate demod/decode is equivalent to combined demod/decode for BPSK/QPSK.

The main goal in designing a digital communication system is to transfer the information s to the Data Sink block in the receiver as reliable as possible at a rate required by the application. Shannon showed in [1] that the information can be transmitted over a channel up to a maximum rate called the *channel capacity* for reliable (arbitrarily small error rate) communication. Although Shannon showed that to achieve such transmission rate one must employ channel coding schemes, he was silent about the structure of these codes. The search for such coding schemes continues to this day. PCCCs and SCCCs are the latest and the most

powerful products of these efforts. These codes can achieve low bit error rates (BER) down to about 10^{-9} at transmission rates very close to capacity limits.

The outline of this chapter as follows. In Section 1.2, a brief history of channel coding is given. In Section 1.3, we review Shannon's channel capacity formula and a capacity expression that we will use later to examine the performance of the PCCCs and SCCCs we designed. In Section 1.4, another important measure of a communication system, *bandwidth efficiency*, is introduced. In Section 1.5, we review the class of Convolutional Codes (CCs) which are employed in the design of PCCCs and SCCCs. Finally in the last Section, we present the outline of this dissertation.

1.2 Channel Coding

Channel coding is a process that adds redundancy to the incoming information stream in such a way that, at the receiver, the Channel Decoder block can detect or correct/control errors caused by the channel. When errors are detected, one possibility for the receiver is to send a retransmit request to the transmitter which in response resends that block of information. These types of systems are called *Automatic Repeat Request* (ARQ). When errors are corrected (hard decision) or controlled (soft decision) at the receiver, then the coding scheme is called *Forward Error Correcting* or *Control* (FEC) coding, respectively. There are also hybrid coding schemes that employ both ARQ and FEC.

In search for codes that achieve transmission rates close to channel capacity, Golay [2] and Hamming [3] were the pioneers for developing practical block codes. In 1955, Elias introduced CCs as an alternative to the block codes in [4]. In 1960's and 1970's, several different algorithms were suggested to decode CCs.

Another milestone in channel coding was the development of concatenated codes by Forney [5]. By connecting two encoder (component codes) in serial, he was able to produce more powerful codes without increasing the decoding complexity beyond the sum of complexities of each decoder (component decoders). In time, the concatenated codes have become a useful tool to produce powerful codes with manageable decoding complexity.

There are three ways to concatenate coding schemes: serial, parallel, and hybrid (both serial and parallel). We focus on parallel and serial concatenations of CCs in this work.

1.3 Channel Capacity

Shannon formulated a quantitative measure for a digital communication system called *channel capacity*, C [bits/s], which is a function of transmitted signal power P [Watts], bandwidth of the channel W [Hz], and power density of additive noise N_0 [Watts/Hz]. For the Additive White Gaussian Noise (AWGN) channel with a bandwidth of W, the channel capacity is given by [1]

$$C = W \cdot \log_2 \left(1 + \frac{P}{WN_0}\right) \text{ [bits/s]} \tag{1.1}$$

(per dimension). Since we assume Nyquist signaling is used, W is clearly defined and related to the transmission rate via $W = R/2$ for baseband and $W = R$ for passband signaling. We recall that the capacity in bits/s is an upper limit on the transmission rate, R, of a communication system for reliable information transfer.

We now review the channel capacity formula for the discrete memoryless channel with discrete input and continuous output [6]. We assume that one- or two-dimensional modulation schemes are used and the AWGN channel is intersymbol-interference free. With perfect synchronization and carrier recovery, the output of the matched filter at discrete time k can be written as

$$r_k = y_k + n_k, \tag{1.2}$$

where y_k is a real or complex discrete signal sent at time k and n_k is an AWGN sample with zero mean and variance σ^2 along each modulation dimension. The average SNR is then equal to [6]

$$\text{SNR} = \frac{E\{|y_k|^2\}}{E\{|n_k|^2\}} = \left\{ \begin{array}{ll} E\{|y_k|^2\}/\sigma^2 & \text{one-dimensional modulation} \\ E\{|y_k|^2\}/2\sigma^2 & \text{two-dimensional modulation} \end{array} \right\}. \tag{1.3}$$

Let us assume that there are N (possibly complex) discrete signal values that y_k can take, $y_k \in \{y^0, y^1, ..., y^{N-1}\}$ with equiprobable occurrence. Then the channel capacity for the discrete input and continuous output channel can be written as [6],

$$C = \log_2(N) - \frac{1}{N} \sum_{k=0}^{N-1} E \left\{ \log_2 \sum_{i=0}^{N-1} \exp\left(- \frac{|y^k + n - y^i|^2 - |n|^2}{2\sigma^2} \right) \right\}, \tag{1.4}$$

where n is real with variance σ^2 for one-dimensional modulation, and is complex with variance $2\sigma^2$ for two-dimensional modulation. The capacity values used in later chapters are found by Monte Carlo averaging of (1.4) for each code rate. The capacity values in bit/channel use for BPSK and QPSK signals in AWGN channels versus E_b/N_0 in dB for reliable communication are given in Fig 1.2. E_b is the information bit energy and $N_0/2$ is the power spectral density of the AWGN (also, $\sigma^2 = N_0/2$).

1.4 Bandwidth Efficiency

The bandwidth efficiency, μ, is a measure of how efficiently a communication system uses its allocated bandwidth. It is defined as

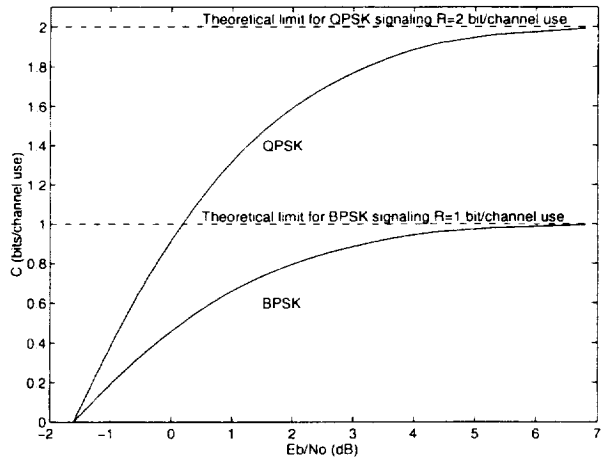

Figure 1.2: Capacity in bits/channel use vs. E_b/N_0 in dB for BPSK channels.

$$\mu = \frac{\text{Bit Rate}}{\text{Channel Bandwidth}} = \frac{R}{W} \qquad \text{[bits/s/Hz]}. \tag{1.5}$$

As there is a theoretical maximum for R, there is also for μ. When the transmission rate is at its maximum, R_{max}, it is equal to channel capacity given in (1.1), i.e.

$$R_{max} = W \cdot \log_2 \left(1 + \frac{P}{WN_0}\right) \text{ [bits/s]}. \tag{1.6}$$

When R_{max} is applied in (1.5), μ takes its maximum value, μ_{max} which is

$$\mu_{max} = \frac{R_{max}}{W} \ . \tag{1.7}$$

Substitution of R_{max} in (1.6) to (1.7) yields the expression for the theoretical maximum bandwidth efficiency for an AWGN channel,

$$\mu_{max} = \log_2 \left(1 + \frac{P}{WN_0}\right). \tag{1.8}$$

Note that $P = \frac{kE_b}{T} = RE_b$ where k is the number of bits per symbol, and T is the symbol duration in seconds. Now by substituting the equality for P in (1.8), we have

$$\mu_{max} = \log_2 \left(1 + \frac{RE_b}{WN_0}\right). \tag{1.9}$$

Note that when the efficiency is at its maximum, $R_{max}/W = C/W = \mu_{max}$. Then the required minimum E_b/N_0 known as *Shannon's bound* for reliable communication with the theoretical bandwidth efficiency becomes

$$\frac{E_b}{N_0} \geq \frac{2^{\mu_{max}} - 1}{\mu_{max}}. \tag{1.10}$$

When a large bandwidth is available, $W \to \infty$, the required E_b/N_0 for reliable communication is the only constraint and can be found by

$$\frac{E_b}{N_0} = \lim_{\mu_{max} \to 0} \frac{2^{\mu_{max}} - 1}{\mu_{max}} = \log_e(2), \tag{1.11}$$

or $E_b/N_0 = -1.59$ dB using the fact that while $W \to \infty$, $\mu_{max} \to 0$.

1.5 Convolutional Codes

An (n, k, m) CC is a k-bit input, n-bit output, and memory size m linear sequential circuit. Since the encoder has memory, the output at any given discrete time will be a function of not only the current input, but the m-bit state of the encoder as well. The rate of an (n, k, m) CC is $r = k/n < 1$ since typically $k < n$. In other words, a rate k/n CC adds $n - k$ redundant bits for every k input bits to form an n-bit codeword. We focus on PCCCs and SCCCs that employ rate 1/2 component CCs with $k = 1$ and $n = 2$.

We considered two groups of CCs: non-recursive convolutional (NRC) and recursive systematic convolutional (RSC) codes. In the following two subsections, we give examples for each as well as some tools to characterize them. When we refer to CCs, we refer to both NRC and RSC codes.

1.5.1 Non-recursive CCs

Fig. 1.3 depicts an NRC (also called feedforward) encoder with $m = 2$. By changing the connections from the memory cells to the modulo-2 adders, it is

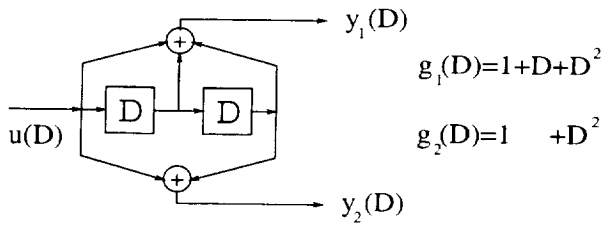

Figure 1.3: Rate 1/2 (7,5) NRC encoder.

possible to define different CCs with the same memory size. Therefore, for each CC, we also supply a generator polynomial set (g_1, g_2) to specify a unique CC. For example, the NRC given in Fig. 1.3 has the generator polynomials $g_1(D) = 1 + D + D^2$ and $g_2(D) = 1 + D^2$ where D represents a single time (clock) delay. Another representation of the generator polynomials which we will use throughout this dissertation is $(g_1, g_2) = (7, 5)$ where representation is octal. Note that since $k = 1$ and $n = 2$, we only need two generator polynomials to define a $(1, 2, m)$ CC.

When a finite length input sequence $u(D)$ is applied to an NRC encoder, the output or the codeword $c(D)$ for input $u(D)$ of the encoder is defined as,

$$\begin{aligned} c(D) = \begin{bmatrix} y_1(D) & y_2(D) \end{bmatrix} &= \begin{bmatrix} u(D)g_1(D) & u(D)g_2(D) \end{bmatrix} \\ &= u(D) \underbrace{\begin{bmatrix} g_1(D) & g_2(D) \end{bmatrix}}_{\mathbf{G}(D)}, \end{aligned} \tag{1.12}$$

where $\mathbf{G}(D)$ is the generator matrix for this NRC encoder. In addition to this mathematical expression for the input-output relation of an NRC code, there are two more well known ways to define an NRC code: the state and trellis diagrams.

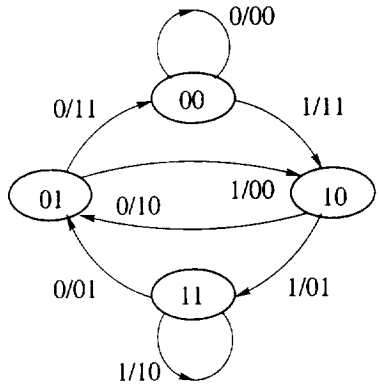

Figure 1.4: State diagram of $r = 1/2$ $(7, 5)$ NRC encoder.

The state diagram for the NRC encoder in Fig. 1.3 is given in Fig. 1.4. Note that since $m = 2$, there are $2^m = 4$ states in the diagram which is the total number of binary combinations for a 2-cell state machine. The transitions from one state to another are indicated by an arrow with "input/output" information. Since our example encoder is a $k/n = 1/2$, for each state transition, there is 1-bit input and 2-bit output. For example, the transition form state 01 to 10 is possible with an input 1 and it produces the output $(y_1, y_2) = (0, 0)$. Note that a state diagram defines all input-output relations of an NRC.

The trellis diagram of the NRC encoder in Fig 1.3 is given in Fig 1.5. As with the state diagram, the trellis diagram of an NRC also indicates the state transitions which are called branches. The trellis also has the time information of an NRC. In other words, the trellis diagram can be used to present the state transitions and outputs of many input bits in the same diagram. In Fig 1.5,

transitions caused by an input 0 are denoted by dashed lines while for input 1 they are denoted by solid lines and produced 2-bit outputs are given for each transition.

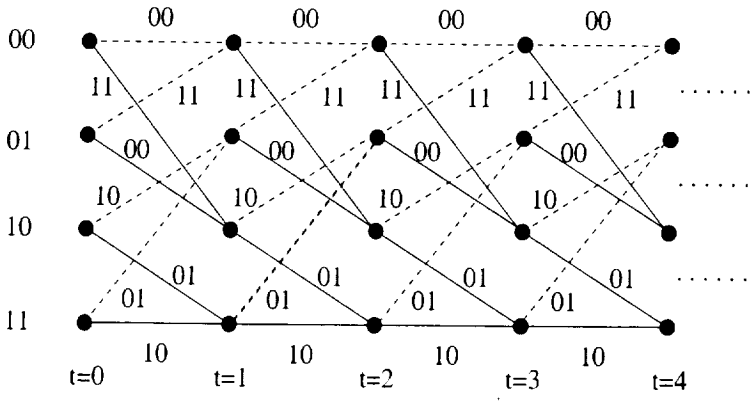

Figure 1.5: Trellis diagram of $r = 1/2$ $(7, 5)$ NRC encoder.

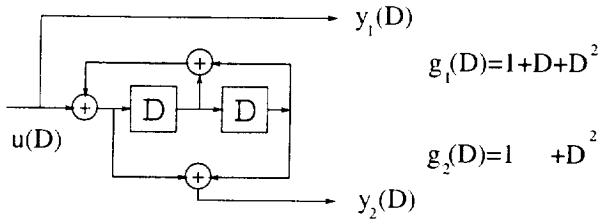

Figure 1.6: Rate 1/2 (7,5) RSC encoder.

1.5.2 Recursive Systematic CCs

If we divide the both sides of the equation (1.12) by $g_1(D)$, we find the input-output relation for the RSC generated by (g_1, g_2) as

$$c(D) = \begin{bmatrix} u(D) & \frac{g_2(D)}{g_1(D)} \end{bmatrix} = u(D) \underbrace{\begin{bmatrix} 1 & \frac{g_2(D)}{g_1(D)} \end{bmatrix}}_{\mathbf{G_s}(D)}, \tag{1.13}$$

where $\mathbf{G_s}(D)$ is the generator matrix for this RSC code. The RSC encoder for this generator matrix with $(g_1, g_2) = (7, 5)$ is given in Fig 1.6 where the polynomial $g_1(D)$ determines the feedback connections. As for an NRC code, an RSC code can be modeled by a state or trellis diagram. RSC and NRC codes with the same (g_1, g_2) and m will produce the same codeword set $\{c(D)\}$, but with different $u(D) \longleftrightarrow c(D)$ mappings. Note that the input $u(D)$ to an NRC encoder and $u(D)g_1(D)$ to an RSC encoder generate the same codeword $c(D)$: $u(D)G(D) = u(D)g_1(D)G_s(D)$. As a consequence, until the discovery of turbo codes[1], there was no reason to choose RSC codes over NRC codes.

The number of non-zero bits in a binary codeword is its *Hamming weight* (or *weight*). The *Hamming distance* (or *distance*) between two binary codewords is the number of locations in which they differ or the weight of the modulo-2 summation of the two codewords. The *minimum distance* of a code, d^H_{min}, is the minimum of the *distances* between all possible pairs of codewords. Since, for linear codes, the sum of any two codewords is another codeword, and the all-zeros vector is always a codeword, their *minimum distance* is also the *distance* minimum weight among non-zero codewords. *Euclidean distance* is a measure of the distance between two codewords in the Euclidian space that is used to represent the transmitted

[1]We will refer parallel concatenated convolutional codes as turbo codes throughout this dissertation.

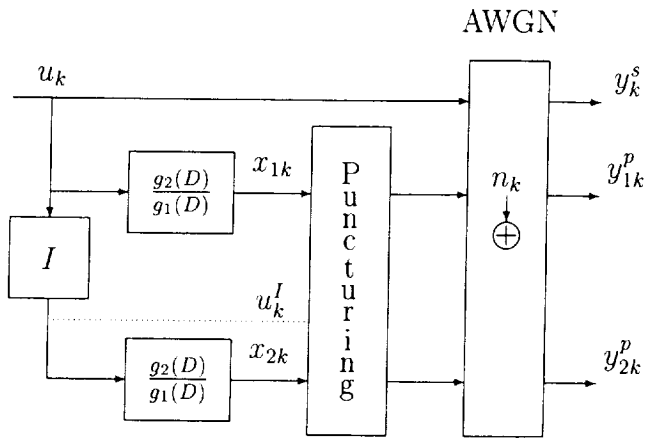

Figure 1.7: PCCC encoder consists of two rate 1/2 RSC encoders with AWGN channel.

signal set. In the case of BPSK/QPSK signaling, the squared *minimum Euclidean distance*, $(d_{min}^E)^2$, of a code is proportional to the *minimum Hamming distance* according to $(d_{min}^E)^2 = d_{min}^H \cdot 4E_c$ where E_c is the channel bit energy. Hence, if a code has large d_{min}^H, it also has large d_{min}^E.

Fig. 1.7 depicts a PCCC encoder with two component RSC encoders [7]. These two rate 1/2 binary encoders are separated by an N-bit pseudo-random interleaver (permuter) which we will denote by I. It will be assumed throughout this dissertation that the two RSC encoders are identical. Although NRC and RSC codes have the same set of codewords, RSC codes generate codewords with low Hamming weight only when the input is divisible by the feedback polynomial $g_1(D)$. Since the codewords with the lowest weight dominate the BER performance of PCCCs, let us assume an input divisible by $g_1(D)$ is encoded by the top RSC encoder and generates a low weight word. Most likely, the interleaved input

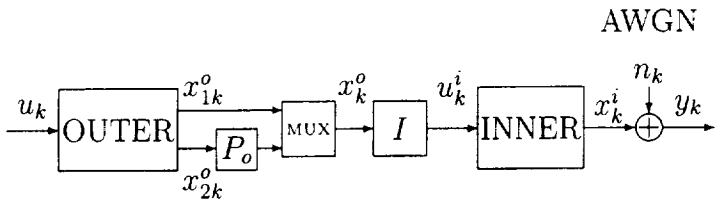

Figure 1.8: SCCC encoder constructed by a rate 1/2 outer and a rate 1 inner encoders with AWGN channel.

to the bottom RSC will not be divisible by $g_1(D)$, generating parity sequence with a large weight. Since the divisibility of both the systematic input and its interleaved version by $g_1(D)$ is rare, the multiplicities of error events for PCCCs is low (e.g., relative to convolutional codes). For a linear block code, the probability of error is approximated for medium to high SNRs by

$$P_b \approx \max_w \frac{n_w \cdot w}{N} Q\left(\sqrt{2rd^H_{w,min}\frac{E_b}{N_0}}\right), \tag{1.14}$$

where w is the encoder input weight, $d^H_{w,min}$ is the minimum Hamming weight for weight-w inputs, n_w is the number of codewords corresponding to weight-w inputs and weight-$d^H_{w,min}$ codewords, r is the code rate, and N is the information word block length. The above approximation shows that low BERs are possible when $d^H_{w,min}$ and N are large, and n_w and w are small. This sheds light on the ability of PCCCs to achieve low BERs even for low SNRs. Since the BER decreases linearly with the interleaver size the term $\frac{1}{N}$ is called "interleaver gain." Perez, et al. [8], called this phenomenon "spectral thinning" and showed that under certain conditions the average number of error events decreases (spectral thinning) to a constant for increasing interleaver size.

A similar argument holds for an SCCC which is depicted in Fig. 1.8. The outer encoder can be either an NRC or RSC encoder, but the inner encoder should be an RSC encoder [9]. The goal is to design an outer encoder that has a large d_{min}. When d_{min} is large, even if the interleaved input is divisible by the feedback polynomial of the inner encoder, it is likely to generate a codeword with large weight after interleaving. It will also reduce the multiplicity of such divisible inputs.

In general, the d_{min} for SCCCs is larger than that of PCCCs for the same number of encoder states. To show this, let us assume the outer and inner encoder of SCCC is the same RSC encoder employed for PCCC. For the SCCCs, the combination of the systematic, $u_k = x^o_{1k}$, and parity, x^o_{2k} sequences of the outer encoder (see Fig. 1.8) is interleaved whereas for the PCCC, only the systematic information sequence of the RSC is interleaved. When $u(D)$ is non-zero, the outer encoder's codeword weight is generally larger than the systematic information weight (except for extreme puncturing). For a randomly generated interleaver, the probability that the PCCC systematic bits will be interleaved to a sequence divisible by $g_1(D)$ is higher than the probability that an SCCC's outer encoder codeword will be interleaved to a word divisible by $g_1(D)$.[2] As a result of this,

[2]In loose terms, for large interleaver sizes, the probability of a weight w input (and its approximately N shifts) to be interleaved to a specific pattern is approximately $N \cdot \frac{1}{\binom{N}{w}} = \left[\frac{N!}{Nw!(N-w)!}\right]^{-1} = \frac{w!}{(N-1)\ldots(N-w+1)} \approx \frac{1}{N^{w-1}}$.

SCCCs have larger d_{min} than PCCCs. Hence, the SCCCs have lower "error floor" than the PCCCs for the same interleaver sizes.

1.6 Outline of The Dissertation

In this dissertation, we examine bandwidth efficient PCCC and SCCC designs for BPSK/QPSK channels. In Chapter 1, we reviewed the field of channel coding and its role in efficient and reliable digital communications, introducing some useful tools like channel capacity and bandwidth efficiency. We also covered NRC and RSC codes as component codes for PCCCs and SCCCs, as well as PCCCs and SCCCs themselves.

In Chapter 2, we briefly introduce the decoding algorithms for CCs. We will focus on an *a posteriori* probability (APP)[3] decoding algorithm for codes modeled with a Markov process proposed by Bahl, et al. [10], that minimizes the bit (or symbol) error rate (BER). We also cover its related algorithms known as Log-MAP and Sliding Window-Log-MAP (SW-Log-MAP) which reduce the complexity of the original APP decoder for hardware implementations.

In Chapter 3, we give the details for the design of bandwidth efficient punctured PCCCs for BPSK/QPSK and AWGN. The design algorithm systematically finds the optimum key parameters of PCCCs to minimize the BER for $m = 3$ and 4. In Chapter 4, an implementation of the design algorithm is given for

[3]The APP algorithm has also been called the MAP algorithm in the literature since it is a MAP algorithm for convolutional codes. However, for the decoding of PCCCs and SCCCs, it is an APP algorithm. We will follow the literature and use MAP and APP interchangeable.

$m = 4$ and rates 3/4, 7/8, and 15/16. Some of the implementation issues such as quantization, SNR estimation offset, and decoding delay are also addressed in this chapter.

In Chapter 5, we describe the details of designing an algorithm for bandwidth efficient high rate SCCCs. The algorithm is similar to that given in Chapter 3 for PCCCs, although there are some modifications due to structural differences between the two codes.

In Chapter 6, we discuss the results found in Chapter 3 through 5, and address suggestions for future research.

2 APP DECODING ALGORITHM AND ITS DERIVATIVES

2.1 Introduction

In 1961, Wozencraft and Reiffen introduced the sequential decoding of CCs in [11]. In later years, the sequential decoding algorithm was improved by Fano [12], Zigangirov [13], and Jelinek [14]. As an alternative to sequential decoding, the Viterbi algorithm (VA) was discovered by Viterbi [15] in 1967. Forney proved that the VA is a maximum likelihood (ML) decoding algorithm for CCs which minimizes the codeword error for AWGN channels in [16] and [17]. The VA is ML in the sense that it selects the most likely codeword over all the possible codewords. In 1974, Bahl, et al. [10], derived an MAP decoding algorithm for Markov processes including CCs that minimizes the BER.

Sequential decoding can achieve low BER with decoding complexity independent from the memory size of CC in high SNR environments. However, for low SNRs, it requires more computational power resulting in longer decoding delays. These long decoding delays eventually cause loss of information since the decoder drops incoming information in order to be able to continue decoding. On the other hand, for the widely used VA, the decoding complexity increases exponentially for a linear increase in the memory size. However, its decoding delay does not change with the SNR fluctuation [18].

Although the VA does not minimize the BER, it has been widely used to decode CCs since it is less complex than the MAP algorithm and their BER performance difference is negligible.

When concatenated codes were first discovered, their decoding was performed by the component decoders each passing hard decisions (either +1 or -1 for binary signalling to represent a 1 or 0 at the encoder, respectively) to the next decoder. For example, consider the serial concatenation the of Reed-Solomon (RS) encoder as the outer (the first) and convolutional encoder as the inner (the second) code. The decoding of this concatenated code was performed first by a soft-decision[4] VA (inner decoder) that passes its hard decisions to the RS decoder (outer decoder). For each input block to be decoded, the VA and RS decoders process their input only once to produce hard-decisions on their codewords.

In 1989, Hagenauer and Hoeher [19] introduced a soft decoding algorithm for concatenated codes. This soft-output Viterbi algorithm (SOVA) allowed the inner decoder to pass soft-decisions to the outer decoder. Although the SOVA is only a modification to the VA, decoders that employ SOVA as component decoders improve the BER performance over the hard-decision component decoders in concatenated schemes.

The idea of iterative (turbo) decoding is first employed in [20]-[22]. In the iterative decoding, not only does the inner decoder pass soft-decisions to the outer

[4]Soft decisions are real numbers taking a range of positive or negative values where the positive sign indicates 1 and the magnitude of the number indicates the reliability: the larger the magnitude, larger the reliability.

decoder, but the outer decoder also passes soft-decisions to the inner decoder. Iterative decoding of PCCCs and SCCCs is different from the previously introduced iterative decoding schemes in several ways. The decoding of PCCCs and SCCCs employs a modified MAP algorithm which also produces soft-decisions and is optimal for minimizing the BER.

In Section 2.2, we will review the MAP decoding algorithm for iterative decoding of PCCCs. In Section 2.3, the MAP decoding algorithm is modified from the original APP algorithm to implement the Log-MAP algorithm. In Section 2.4, the Sliding Window-Log-MAP (SW-Log-MAP) algorithm which reduces the decoding delays of each component APP decoder will be reviewed. The development of the APP algorithm follows [23], [24], and [25]. Finally in Section 2.5, an APP decoding algorithm, called soft-input soft-output (SISO) algorithm employed in the decoding of SCCCs will be covered. The development of the SISO algorithm follows from [26].

2.2 The APP Algorithm

As mentioned, the APP decoding algorithm introduced in [10] needs to be modified for adopting it into an iterative decoding structure and for making it stable, and faster for computer simulations or hardware implementations. First we review the original APP algorithm from [10] and [23]. Then, we introduce the necessary changes to drive the modified APP decoding algorithm, [24] and [27].

Before we start the details of the algorithm, we give the following definitions:

- m is the component RSC memory size.
- S is the set of all 2^m states of the RSC.
- $\mathbf{x}^s = (x_1^s, x_2^s, \ldots, x_N^s) = (u_1, u_2, \ldots, u_N)$ is the RSC input word where N is the word size.
- $\mathbf{x}^p = (x_1^p, x_2^p, \ldots, x_N^p)$ is the parity word generated by each RSC.
- $y_k = (y_k^s, y_k^p)$ is (x_k^s, x_k^p) that is disturbed with AWGN.
- $\mathbf{y}_a^b = (y_a, y_{a+1}, \ldots, y_b)$.
- $\mathbf{y} = \mathbf{y}_1^N = (y_1, y_2, \ldots, y_N)$.

The MAP decoder decides $u_k = +1$ if $P(u_k = +1|\mathbf{y}) > P(u_k = -1|\mathbf{y})$, and decides $u_k = -1$ otherwise. In a compact notation, the MAP decoder calculates the ratio of these two probabilities in the log domain: *log-APP* ratio (LAPP) or soft-decision of u_k, $L(u_k)$ by

$$L(u_k) \triangleq \log\left(\frac{P(u_k = +1|\mathbf{y})}{P(u_k = -1|\mathbf{y})}\right), \tag{2.1}$$

where the log operation is taken in the base e throughout this dissertation for convenience. The decoder makes a hard decision, $\hat{u}_k$, on u_k, if necessary, by

$$\hat{u}_k = \text{sign}\left(L(u_k)\right).$$

Note that the magnitude of the $L(u_k)$ gives information about how reliable the decision $\hat{u}_k$ is. $L(u_k)$ can be written in terms of state transitions as

$$L(u_k) = \log\left(\frac{\sum_{S+} p(s_{k-1} = s', s_k = s, \mathbf{y})/p(\mathbf{y})}{\sum_{S-} p(s_{k-1} = s', s_k = s, \mathbf{y})/p(\mathbf{y})}\right), \tag{2.2}$$

where s is the state of the encoder at discrete time k ($k = 0, 1, 2, \ldots, N$) and u_k is the information bit associated with the state transition from s' (the state at time $k-1$) to s. S is the set of all possible states and $s', s \in S$. The sets S^+ and S^- represent the sets of state transitions ($s' \rightarrow s$) for which $u_k = +1$ and $u_k = -1$, respectively. Note that the term $p(\mathbf{y})$ in the numerator and denominator of (2.2) can be canceled since it is not a function of u_k and the $p(s', s, \mathbf{y})$ term can be written as [23]

$$\begin{aligned} p(s', s, \mathbf{y}) &= p(s', \mathbf{y}_1^{k-1}) \cdot p(s, \mathbf{y}_k | s') \cdot p(\mathbf{y}_{k+1}^N | s) \\ &= \alpha_{k-1}(s')\gamma_k(s', s)\beta_k(s), \end{aligned} \tag{2.3}$$

where $\alpha_k(s) \triangleq p(s_k = s, \mathbf{y}_1^k)$ is computed in a "forward recursion" from

$$\alpha_k(s) = \sum_{s' \in S} \alpha_{k-1}(s')\gamma_k(s', s), \tag{2.4}$$

with the initial conditions $\alpha_0(0) = 1$ and $\alpha_0(s \neq 0) = 0$, assuming that the component RSC starts encoding at zero state.

The probabilities $\beta_k(s) \triangleq p(\mathbf{y}_{k+1}^N | s_k = s)$ are computed in a "backward recursion" from

$$\beta_{k-1}(s') = \sum_{s \in S} \beta_k(s)\gamma_k(s', s), \tag{2.5}$$

with the boundary conditions $\beta_N(0) = 1$ and $\beta_N(s \neq 0) = 0$ assuming that the component RSC encoder is forced to end ("terminate") at the zero state at the end of each block by selecting the last m bits (termination bits) accordingly.

The branch metric $\gamma(s', s)$ in (2.4) and (2.5) is defined as

$$\gamma_k(s', s) \triangleq p(s_k = s, y_k | s_{k-1} = s'). \tag{2.6}$$

We will show how to calculate $\gamma_k(s', s)$ later. Fig. 2.1 shows the relationships between recursively calculated variables in a trellis diagram.

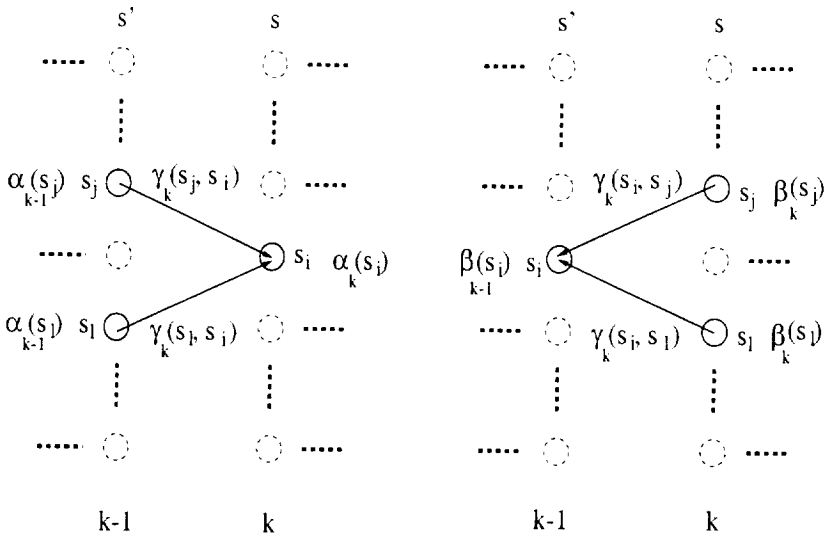

Figure 2.1: Variables calculated recursively are shown in a trellis diagram.

The original APP decoding algorithm given in [10] and [23] calculates LAPP of u_k by combining (2.2) and (2.3)

$$L(u_k) = \log\left(\frac{\sum_{S+}\alpha_{k-1}(s')\gamma_k(s',s)\beta_k(s)}{\sum_{S-}\alpha_{k-1}(s')\gamma_k(s',s)\beta_k(s)}\right), \tag{2.7}$$

since the $p(\mathbf{y})$ terms in (2.2) are canceled. This cancellation causes an unstable algorithm since the algorithm is multiplicative and the recursive variable values decrease exponentially eventually causing an underflow.

In the following subsection, we show a normalization process for $\alpha_k(s)$ and $\beta_k(s)$ variables that prevents their exponential decrease [24], [25]. Also the original BCJR decoding algorithm will be utilized to be used in iterative decoding of PCCCs.

2.2.1 Modified APP Decoding

Instead of canceling the terms $p(\mathbf{y})$ in (2.2), we multiply the numerator and denominator by $p(y_k)$. This means we divide (2.3) by $p(\mathbf{y})/p(y_k|y_1^{k-1}) = p(\mathbf{y}_1^{k-1})p(\mathbf{y}_{k+1}^{N}|\mathbf{y}_1^{k})$. So each term in the numerator and denominator can be written as

$$\frac{p(s',s,\mathbf{y})p(y_k|y_1^{k-1})}{p(\mathbf{y})} = \tilde{\alpha}_{k-1}(s')\gamma_k(s',s)\tilde{\beta}_k(s), \tag{2.8}$$

where

$$\tilde{\alpha}_k(s) = \alpha_k(s)/p(\mathbf{y}_1^k),$$

$$\tilde{\beta}_k(s) = \beta_k(s)/p(\mathbf{y}_{k+1}^{N}|\mathbf{y}_1^{k})).$$

The values $\tilde{\alpha}_k(s)$ can be calculated from $\alpha_k(s)$'s by

$$\tilde{\alpha}_k(s) = \frac{\alpha_k(s)}{\sum_{s\in S}\alpha_k(s)}, \tag{2.9}$$

since $p(\mathbf{y}_1^k) = \sum_{s\in S} \alpha_k(s)$. Now by using (2.4) and (2.9), we can write a recursive equation for $\tilde{\alpha}_k(s)$,

$$\begin{aligned}\tilde{\alpha}_k(s) &= \frac{\sum_{s'} \alpha_{k-1}(s')\gamma_k(s',s)}{\sum_s \sum_{s'} \alpha_{k-1}(s')\gamma_k(s',s)} \\ &= \frac{\sum_{s'} \tilde{\alpha}_{k-1}(s')\gamma_k(s',s)}{\sum_s \sum_{s'} \tilde{\alpha}_{k-1}(s')\gamma_k(s',s)},\end{aligned} \tag{2.10}$$

where the second equality follows after dividing the numerator and denominator by $p(\mathbf{y}_1^{k-1})$.

In order to find a recursive equation for $\tilde{\beta}_{k-1}(s')$ as in (2.5), the term $p(\mathbf{y}_k^N|\mathbf{y}_1^{k-1})$ should be written as

$$\begin{aligned}p(\mathbf{y}_k^N|\mathbf{y}_1^{k-1}) &= \frac{p(\mathbf{y})}{p(\mathbf{y}_1^{k-1})} \\ &= p(\mathbf{y}_1^k) \cdot \frac{p(\mathbf{y}_{k+1}^N|\mathbf{y}_1^k)}{p(\mathbf{y}_1^{k-1})} \\ &= \sum_s \alpha_k(s) \cdot \frac{p(\mathbf{y}_{k+1}^N|\mathbf{y}_1^k)}{p(\mathbf{y}_1^{k-1})} \\ &= \sum_s \sum_{s'} \alpha_k(s')\gamma_k(s',s) \cdot \frac{p(\mathbf{y}_{k+1}^N|\mathbf{y}_1^k)}{p(\mathbf{y}_1^{k-1})} \\ &= \sum_s \sum_{s'} \tilde{\alpha}_{k-1}(s')\gamma_k(s',s) \cdot p(\mathbf{y}_{k+1}^N|\mathbf{y}_1^k),\end{aligned} \tag{2.11}$$

then the recursive equation for $\tilde{\beta}_{k-1}(s')$ can be written by dividing (2.5) by the above equation

$$\tilde{\beta}_{k-1}(s') = \frac{\sum_s \tilde{\beta}_k(s)\gamma_k(s',s)}{\sum_s \sum_{s'} \tilde{\alpha}_{k-1}(s')\gamma_k(s',s)}, \tag{2.12}$$

Note that the $\tilde{\alpha}_k(s)$ and $\tilde{\beta}_k(s)$ have the same initial and boundary conditions as $\alpha_k(s)$ and $\beta_k(s)$, respectively.

Now for this modified APP algorithm, the LAPP can be written as

$$L(u_k) = \log\left(\frac{\sum_{S^+} \tilde{\alpha}_{k-1}(s')\gamma_k(s',s)\tilde{\beta}_k(s)}{\sum_{S^-} \tilde{\alpha}_{k-1}(s')\gamma_k(s',s)\tilde{\beta}_k(s)}\right). \tag{2.13}$$

The computation of $\gamma_k(s',s)$ will be given in the next subsection.

2.2.2 Iterative (Turbo) Decoding

The LAPP for any MAP decoder can be written as the following by using the Bayes' rule

$$L(u_k) = \log\left(\frac{P(\mathbf{y}|u_k=+1)}{P(\mathbf{y}|u_k=-1)}\right) + \log\left(\frac{P(u_k=+1)}{P(u_k=-1)}\right).$$

The second term is called *a priori* information on u_k. For general decoders, u_k is assumed to be $+1$ or -1 equally likely which makes the a priori information equal to zero. In the case of iterative decoding, each decoder (D1 or D2) will produce a priori information for each u_k which is called *extrinsic* or *soft information* to be used for the next decoder (D2 or D1) as *a priori* information. So for the first iteration, D1 passes the extrinsic information on all u_k's to D2 as a priori information. Then D2 passes the extrinsic information to D1 as a priori information. This process continues until an iteration stopping criterion is met. The information passed between the decoders is called extrinsic information since D1 passes this information produced by the decoding of the first encoder's parity bits to D2. D2 also generates extrinsic information produced by the decoding of the second encoder's parity bits to be passed to D1.

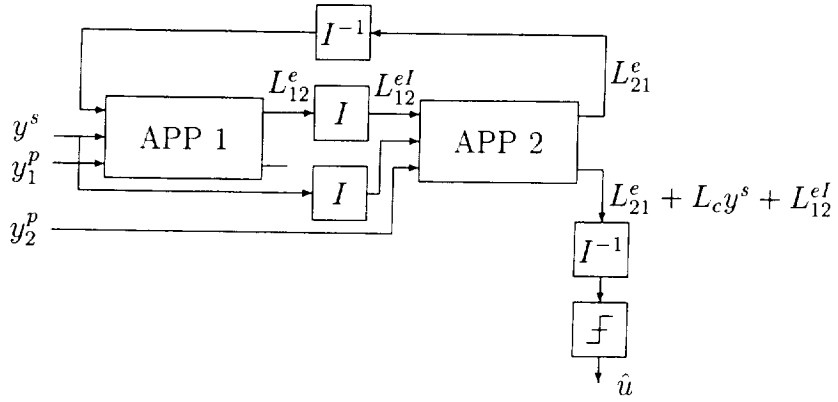

Figure 2.2: Iterative decoding of PCCCs with component APP decoders.

Fig. 2.2 depicts an iterative decoder block diagram using APP decoders for each component encoder. The two interleavers and the deinterleaver ensures each decoder receives the properly ordered systematic, parity, and extrinsic information.

Now we will show how to calculate $\gamma_k(s', s)$ which will lead us to the calculation of the extrinsic information. By using the definition in (2.6) and Bayes' rule

$$\begin{aligned}\gamma_k(s', s) &= p(y_k, s, s')/P(s') \\ &= p(y_k|s, s') \cdot P(s, s')/P(s') \\ &= P(s|s') \cdot p(y_k|s, s') \\ &= P(u_k) \cdot p(y_k|u_k)\end{aligned}$$

where event u_k has a probability assigned when $s' \longrightarrow s$. We define the extrinsic information as

$$L^e(u_k) \triangleq \log\left(\frac{P(u_k = +1)}{P(u_k = -1)}\right). \tag{2.14}$$

Then we can write $P(u_k)$ as

$$P(u_k) = \left(\frac{\exp(-L^e(u_k)/2)}{1+\exp(-L^e(u_k))}\right) \cdot \exp(u_k L^e(u_k)/2)$$
$$= A_k \cdot \exp(u_k L^e(u_k)/2). \tag{2.15}$$

Note that A_k is independent of u_k. We can verify the equation (2.15) by letting $P(u_k = +1) = P_+$ and $P(u_k = -1) = P_-$ for convenience as

$$P(u_k = +1) = \left(\frac{\sqrt{P_-/P_+}}{1+P_-/P_+}\right) \cdot \sqrt{P_+/P_-} = P_+$$
$$P(u_k = -1) = \left(\frac{\sqrt{P_-/P_+}}{1+P_-/P_+}\right) \cdot \sqrt{P_-/P_+} = P_-.$$

The second term in the definition of $\gamma_k(s', s)$ is $p(y_k|u_k)$ which can be written for AWGN channel as

$$p(y_k|u_k) \propto \exp\left(-\frac{(y_k^s - u_k)^2}{2\sigma^2} - \frac{(y_k^p - x_k^p)^2}{2\sigma^2}\right)$$
$$= \exp\left(-\frac{y_k^{s2} + u_k^2 + y_k^{p2} + x_k^{p2}}{2\sigma^2}\right) \cdot \exp\left(\frac{u_k y_k^s + x_k^p y_k^p}{\sigma^2}\right)$$
$$= B_k \cdot \exp\left(\frac{u_k y_k^s + x_k^p y_k^p}{\sigma^2}\right),$$

where we used the fact that $y_k = (y_k^s, y_k^p)$ and $x_k = (x_k^s, x_k^p) = (u_k, x_k^p)$. Note, again, that B_k is independent of u_k. Then we can write

$$\gamma_k(s', s) \propto A_k B_k \cdot \exp\left(\frac{u_k y_k^s + x_k^p y_k^p}{\sigma^2}\right) \cdot \exp(u_k L^e(u_k)/2). \tag{2.16}$$

The term $A_k B_k$ appearing in the numerator and denominator will cancel in (2.12) since it is independent of u_k. We assumed that symbols transmitted over the

AWGN channel are ± 1. Then $\frac{E_s}{N_0/2} = \frac{1}{\sigma^2}$ and $E_s = rE_b$ is the energy per channel symbol. Substituting σ^2 in (2.16) we have

$$\begin{aligned}\gamma_k(s', s) &\propto \exp\left(\frac{1}{2}u_k(L^e(u_k) + L_c y_k^s) + \frac{1}{2}L_c x_k^p y_k^p\right) \\ &= \exp\left(\frac{1}{2}u_k(L^e(u_k) + L_c y_k^s)\right) \cdot \gamma_k^e(s', s) \\ &= C_k \cdot \gamma_k^e(s', s) \end{aligned} \tag{2.17}$$

where $L_c \triangleq \frac{4E_s}{N_0}$ and

$$\gamma_k^e(s', s) \triangleq \exp\left(\frac{1}{2}L_c y_k^p x_k^p\right).$$

$$C_k \triangleq \exp\left(\frac{1}{2}u_k(L^e(u_k) + L_c y_k^s)\right) \cdot \gamma_k^e(s', s).$$

Note that the calculations of $\gamma_k(s', s)$ are independent of forward and backward recursion variables, i.e., $\gamma_k(s', s)$ are the same for iterative decoding in the original and modified APP decoding. Finally, when the $\gamma_k(s', s)$ value is plugged in the LAPP equation, it becomes

$$\begin{aligned} L(u_k) &= \log\left(\frac{\sum_{S^+} \tilde{\alpha}_{k-1}(s') \cdot \gamma_k^e(s', s) \cdot \tilde{\beta}_k(s) \cdot C_k}{\sum_{S^-} \tilde{\alpha}_{k-1}(s') \cdot \gamma_k^e(s', s) \cdot \tilde{\beta}_k(s) \cdot C_k}\right) \\ &= L_c y_k^s + L^e(u_k) + \log\left(\frac{\sum_{S^+} \tilde{\alpha}_{k-1}(s') \cdot \gamma_k^e(s', s) \cdot \tilde{\beta}_k(s)}{\sum_{S^-} \tilde{\alpha}_{k-1}(s') \cdot \gamma_k^e(s', s) \cdot \tilde{\beta}_k(s)}\right). \end{aligned} \tag{2.18}$$

The second equation follows from when C_k is written for $u_k = +1$ and $u_k = -1$, the common terms can be brought out of the log function.

The first term in (2.18) is referred as channel values, the second term is *a priori* information for u_k passed from the previous decoder, and the last term is called

extrinsic information for u_k to pass to the next decoder as a priori information. Then the LAPP calculated for u_k by D1 is

$$L_1(u_k) = L_c y_k^s + L_{21}^e(u_k) + L_{12}^e(u_k), \tag{2.19}$$

where $L_{21}^e(u_k)$ is extrinsic information passed from D2 to D1 and $L_{12}^e(u_k)$ is the extrinsic information D1 passes to D2. In Fig. 2.2, the top output of each APP decoder block shows the extrinsic information passed to the next decoder while the bottom output represents the LAPP value which is the summation of three soft information as in (2.19) and is written explicitly for the second APP decoder.

2.3 The Log-MAP Algorithm

We first modified the original APP decoding algorithm by defining $\tilde{\alpha}_k(s)$ and $\tilde{\beta}_k(s)$. This modification only helped to normalize the exponentially decreasing variables but the modified algorithm is still multiplicative, i.e., the recursive equations to find $\tilde{\alpha}_k(s)$ and $\tilde{\beta}_k(s)$ require multiplications. Also $\gamma_k(s', s)$ calculations still require implementation of the exponential function. For hardware implementations, an addition operation is easier to implement than a multiplication. Implementing the exponential function on the other hand, is possible by large look-up tables. Therefore we introduce the Log-MAP decoding algorithm to convert the multiplications in the original APP decoding into additions. This conversion also removes the necessity to implement a look-up table for the exponential function.

In the Log-MAP algorithm, the new parameters are defined by taking the logarithms of the original APP algorithm parameters in the base e, [28] and [29]. We now replace the $\alpha_k(s)$, $\beta_k(s)$, and $\gamma_k(s',s)$ in (2.4), (2.5), and (2.17) with $\bar{\alpha}_k(s)$, $\bar{\beta}_k(s)$, and $\bar{\gamma}_k(s',s)$ in the Log-MAP algorithm, respectively by

$$\bar{\alpha}_k(s) \triangleq \log\{\alpha_k(s)\} = \log\left\{\sum_{s'\in S} \exp\left[\bar{\alpha}_{k-1}(s') + \bar{\gamma}_k(s',s)\right]\right\} + C_{\bar{\alpha}}, \tag{2.20}$$

with the initial conditions $\bar{\alpha}_0(0) = 0$ and $\bar{\alpha}_0(s \neq 0) = -\infty$.

$$\bar{\beta}_{k-1}(s') \triangleq \log\{\beta_k(s)\} = \log\left\{\sum_{s\in S} \exp\left[\bar{\beta}_k(s) + \bar{\gamma}_k(s',s)\right]\right\} + C_{\bar{\beta}}, \tag{2.21}$$

with the boundary conditions $\bar{\beta}_N(0) = 0$ and $\bar{\beta}_N(s \neq 0) = -\infty$. Finally, the branch metric in (2.6) becomes

$$\bar{\gamma}_k(s',s) \sim \frac{1}{2}u_k[L^e(u_k) + L_c y_k^s] + \frac{1}{2}L_c y_{ik}^p x_{ik}^p. \tag{2.22}$$

$C_{\bar{\alpha}}$ and $C_{\bar{\beta}}$ are necessary constants to avoid over or underflows of $\bar{\alpha}_k(s)$ and $\bar{\beta}_k(s)$ values, but they are not calculated from the normalization factors in the modified APP decoding algorithm.

The Log-MAP algorithm apparently require the summation of exponential functions and the logarithm of the result as seen in (2.20) and (2.21). These operations are transformed to a special operation denoted by max* and defined as [27], [30]

$$\log\{\sum_{i=1}^{n} \exp(a_i)\} = \max^*(a_1, a_2, \ldots, a_n). \tag{2.23}$$

The operator max* with 2 arguments is defined as

$$\max^*(a_1, a_2) \triangleq \max(a_1, a_2) + \log(1 + \exp(-\Delta)), \tag{2.24}$$

where $\Delta = |a_1 - a_2|$. The second term in (2.24) is called *correction term*. If more than two arguments are given, say three, the max* operator is evaluated recursively by

$$\begin{aligned} \max^*(a_1, a_2, a_3) &= \max^*(\max^*(a_1, a_2), a_3) \\ &= \max_{i \in \{1,2,3\}}{}^*(a_i). \end{aligned} \tag{2.25}$$

A sub-optimum version of the max* operation can be achieved by dropping the correction term (called MAX-Log-MAP in [27]). The max operation performs very close to the max* for large SNR values, i.e., Δ is large for large SNR values. Now $\bar{\alpha}_k(s)$ in (2.20) and $\bar{\beta}_k(s')$ in (2.21) can be written with the max* operation by

$$\bar{\alpha}_k(s) = \max_{s' \in S}{}^* \left(\bar{\alpha}_{k-1}(s') + \bar{\gamma}_k(s', s)\right) + C_{\bar{\alpha}}, \tag{2.26}$$

$$\bar{\beta}_{k-1}(s') = \max_{s \in S}{}^* \left(\bar{\beta}_k(s) + \bar{\gamma}_k(s', s)\right) + C_{\bar{\beta}}, \tag{2.27}$$

Finally the LAPP for u_k in (2.18) can be written in terms of $\bar{\alpha}_k(s)$, $\bar{\beta}_k(s')$, and $\bar{\gamma}_k(s', s)$ by

$$L(u_k) = L_c y_k^s + L^e(u_k) + \log\left(\frac{\exp\left(\sum_{S^+} \bar{\alpha}_{k-1}(s') + \bar{\gamma}_k^e(s', s) + \bar{\beta}_k(s)\right)}{\exp\left(\sum_{S^-} \bar{\alpha}_{k-1}(s') + \bar{\gamma}_k^e(s', s) + \bar{\beta}_k(s)\right)}\right). \tag{2.28}$$

and when the max* operation is applied, the LAPP becomes

$$L(u_k) = L_c y_k^s + L^e(u_k)$$

$$+ \max_{S^+}{}^* \left(\bar{\alpha}_{k-1}(s') + \bar{\gamma}_k^e(s', s) + \bar{\beta}_k(s)\right) - \max_{S^-}{}^* \left(\bar{\alpha}_{k-1}(s') + \bar{\gamma}_k^e(s', s) + \bar{\beta}_k(s)\right). \tag{2.29}$$

2.4 Sliding Window-Log-MAP (SW-Log-MAP) Algorithm

In its current form, to calculate $L(u_1)$ with the Log-MAP algorithm, we still need to know $\bar{\beta}_1(s)$ values. However, $\bar{\beta}_1(s)$ are calculated recursively from $\beta_N(s)$ based on the boundary conditions. Therefore, in order to make any LAPP calculation, we need to receive the whole codeword. Benedetto, et al. [26], introduced a method called sliding window decoding to overcome this problem. Pietrobon employed a similar concept to his APP decoder design in [31] (see also Barbulescu [32]). In both, the main purpose is to break the interleaver size down to smaller blocks to avoid long delays D, where $D << N$, and to apply the backward recursion in these smaller blocks. As it will be shown in later chapters, sufficiently large D will result in very small BER performance degradation.

The only difference between the SW-Log-MAP and the Log-MAP algorithm is the boundary conditions and the calculations of backward recursion values. They should be modified in the Log-MAP as [31]

$$\bar{\beta}_{2D+D*j-i}(s') = \max_{s \in S}{}^* \left(\bar{\beta}_{2D+D*j-i+1}(s) + \bar{\gamma}_k(s', s)\right) \tag{2.30}$$

where $j = 0, 1, 2, \ldots, (\frac{N}{D} - 2)$ is the outer loop index and $i = 1, 2, \ldots, 2D - 1$ is the inner loop index. The boundary conditions become

$$\bar{\beta}_{2D+D*j}(s) = 0 \text{ for all } s \text{ and } j = 0, 1, 2 \ldots (\frac{N}{D} - 3)$$

and for the end of the block

$$\bar{\beta}_N(0) = 0 \text{ and } \bar{\beta}_N(s \neq 0) = -\infty.$$

The forward recursion variable $\bar{\alpha}_k(s)$ is calculated as in the Log-MAP algorithm. As soon as $\bar{\beta}_1(s)$ values are calculated, the component APP decoder is ready to calculate the LAPP of u_1, $L(u_1)$. Note that this sliding window approach will only reduce the decoding delay of a component APP decoder. For concatenated schemes, unless a special interleaver is designed, the decoding delay of Log-MAP algorithm and the SW-Log-MAP are similar since each component APP decoder processes the *extrinsic* information received from de/interleaver which generally requires the *extrinsic* information for the whole block to be available at once.

2.5 The SISO Algorithm

The APP algorithm proposed in [10] and its derivatives in the previous sections are developed for the constituent decoders that will not need the *extrinsic* information for parity bits. For PCCCs, the constituent decoders only need to generate and receive the *extrinsic* information for the systematic input bits. However, in the decoding of the SCCCs as shown in Fig 2.4 (see Fig 1.8 for the encoder),

the constituent decoder should be able to generate *extrinsic* information on both input and coded bits. As we will see in detail later in this section, the SISO algorithm produces *extrinsic* information on both input and coded bits. Another advantage of the SISO algorithm is that the transitions (edges) are defined with edge numbers which enables the SISO algorithm to decode the codes with parallel edges.[5]

Before giving the details of the SISO algorithm, we will introduce necessary notation and follow the algorithm details from [26]. We consider a memory size m, rate k_0/n_0 trellis encoder: k_o-bit input and n_0-bit coded symbols. The encoder accepts an input symbol u_k, and generates the coded symbol x_k at time k where $k = 1, 2, \ldots, N$ and N is the block size. Then we can define the input and coded sequences as $\mathbf{u} = (u_1, u_2, \ldots, u_N)$ and $\mathbf{x} = (x_1, x_2, \ldots, x_N)$, respectively. The *a priori* probability of the input sequence $\mathbf{u}$ can be found from the input symbol probability densities by

$$\mathbf{P}(\mathbf{u}; i) = (P_k(u_1; i), P_k(u_2; i), \ldots, P_k(u_N; i)), \tag{2.31}$$

where $P_k(u_k; i) = \mathbf{P}[u_k]$ and i indicates the input sequence.[6] Similarly, for the coded sequence $\mathbf{x}$, the *a priori* probabilities can be written by

$$\mathbf{P}(\mathbf{x}; i) = (P_k(x_k; i), P_k(u_2; i), \ldots, P_k(u_N; i)), \tag{2.32}$$

[5]The parallel edges have transitions starting and ending in the same states.

[6]$P(u)$ represents a function of u, and P[u] represents the probability of event u.

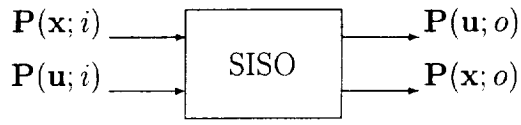

Figure 2.3: SISO module.

where $P_k(x_k; i) = \mathbf{P}[x_k]$. For simplification, we denote $P_k(u_k; i)$ and $P_k(x_k; i)$ by $P_k(u; i)$ and $P_k(x; i)$, respectively.

Although the SISO algorithm can be employed for a time-varying trellis code, for simplicity, we assume that the trellis code is time invariant. Hence, a single trellis section is enough to define a code. A memory size m, rate k_0/n_0 trellis encoder has 2^m states, and 2^{k_0} transitions from each state. Therefore, this code has 2^{k_0+m} edges. The state of the trellis at time k is s' and at time $k+1$ is s, where $s', s \in S = \{s_0, s_1, \ldots, s_{2^m-1}\}$. Then an edge, e, is associated with a pair of states $(s'(e), s(e))$ and $e \in E = \{e_0, e_1, \ldots, e_{2^{k_0+m}-1}\}$. Also associated with each edge e, there is an input symbol $u(e)$ and an output symbol $c(e)$. The pair $(s'(e), u(e))$ always uniquely defines the ending state $s(e)$ which assures that the code is decodable.

SISO decoder module, shown in Fig. 2.3, is a four port device that receives the sequences of probability distributions (or their logarithms) $\mathbf{P}(\mathbf{x}; i)$ and $\mathbf{P}(\mathbf{u}; i)$ and produces the sequences of probability distributions (or their logarithms) $\mathbf{P}(\mathbf{x}; o)$ and $\mathbf{P}(\mathbf{u}; o)$ based on the trellis code constraints.

We will directly give the equations for the Log-SISO (additive SISO) algorithm as the development from the multiplicative SISO to the Log-SISO follows steps similar to the APP algorithm. In order to find the outputs $\mathbf{P}(\mathbf{x}; o)$ and $\mathbf{P}(\mathbf{u}; o)$ from (2.32) and (2.31), respectively, we first find $P_k(x; o)$ and $P_k(u; o)$ for each k from

$$\begin{aligned} P_k(u; o) &= \max_{e:u(e)=u}{}^{*} \left(\alpha_{k-1}[s'(e)] + P_k[u(e); i] + P_k[x(e); i] + \beta_k[s(e)]\right) \\ &= \max_{e:u(e)=u}{}^{*} \left(\alpha_{k-1}[s'(e)] + P_k[x(e); i] + \beta_k[s(e)]\right) \qquad (2.33) \\ P_k(x; o) &= \max_{e:x(e)=x}{}^{*} \left(\alpha_{k-1}[s'(e)] + P_k[u(e); i] + P_k[x(e); i] + \beta_k[s(e)]\right) \\ &= \max_{e:x(e)=x}{}^{*} \left(\alpha_{k-1}[s'(e)] + P_k[u(e); i] + \beta_k[s(e)]\right), \qquad (2.34) \end{aligned}$$

where $k = 1, 2, \ldots, N$. The second equation follows from the fact that $P_k(x; i)$ and $P_k(u; i)$ in the $P_k(c; o)$ and $P_k(u; o)$ calculations, respectively, do not depend on the indices of the max* operation. Hence, they can be removed from the summation. The forward and backward recursion variables $\alpha_k(s)$ and $\beta_k(s)$ can be calculated by

$$\alpha_k(s) = \max_{e:s(e)=s}{}^{*} \left(\alpha_{k-1}[s'(e)] + P_k[u(e); i] + P_k[x(e); i]\right), \; k = 1, \ldots, N \qquad (2.35)$$

$$\beta_k(s) = \max_{e:s'(e)=s}{}^{*} \left(\beta_{k+1}[s(e)] + P_{k+1}[u(e); i] + P_{k+1}[x(e); i]\right), \; k = N-1, \ldots, 0 \qquad (2.36)$$

with the initial conditions $\alpha_0(0) = 0$ and $\alpha_0(s) = -\infty$ for $s \neq 0$ assuming the encoding starts at zero state and the boundary conditions $\beta_N(0) = 0$ and $\beta_N(s) = -\infty$ for $s \neq 0$ assuming the encoding ends at zero state for each block.

These new probabilistic values $P_k(u;o)$ and $P_k(x;o)$ can be seen as improved versions of $P_k(u;i)$ and $P_k(x;i)$, respectively, based on the trellis of the code. In the iterative decoding literature, the soft information $P_k(u;o)$ and $P_k(x;o)$ may be called *extrinsic* information since they are the new information for the next constituent SISO module.

Note that, up to this point, the probabilistic values are found for trellis input and coded symbols u and x, respectively. Since the outer and inner encoders are split by a bit interleaver, we need to find *extrinsic* information for input and output bits so they can be bitwise interleaved/deinterleaved during the decoding. Since we employ bit interleavers, the following assumption holds in the log domain:

$$P_k(u;i) = \sum_{j=1}^{k_0} P_{k,j}(u^j;i) \tag{2.37}$$

$$P_k(x;i) = \sum_{j=1}^{n_0} P_{k,j}(x^j;i) \tag{2.38}$$

where $u^j, x^j \in \{0,1\}$ and they represent the j^{th} bits of the input and coded symbol u and x, respectively. By using the symbol *extrinsic* information output of the SISO module in (2.33) and (2.34), the *extrinsic* information for the j^{th} bit of the input and coded symbols can be written as

$$\begin{aligned} P_k(u^j;o) &= \max_{e:u(e)=u}{}^{*} \left(\alpha_{k-1}[s'(e)] + \sum_{p=1}^{k_0} (P_k[u^p(e);i]) + \sum_{q=1}^{n_0} (P_k[x^q(e);i]) + \beta_k[s(e)] \right) \cdot \\ &= \max_{e:u(e)=u}{}^{*} \left(\alpha_{k-1}[s'(e)] + \sum_{\substack{p=1 \\ p\neq j}}^{k_0} (P_k[u^p(e);i]) + \sum_{q=1}^{n_0} (P_k[x^q(e);i]) + \beta_k[s(e)] \right) \end{aligned} \tag{2.39}$$

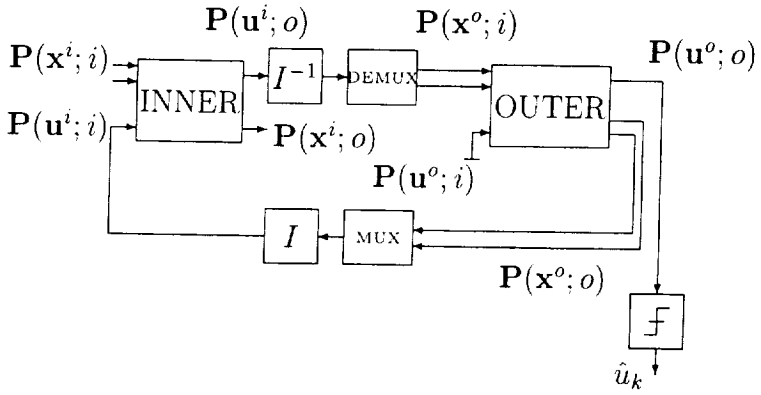

Figure 2.4: SCCC decoder with SISO modules.

$$P_k(x^j; o) = \max_{e:x(e)=x}{}^* \left(\alpha_{k-1}[s'(e)] + \sum_{p=1}^{k_0} (P_k[u^p(e); i]) + \sum_{q=1}^{n_0} (P_k[x^q(e); i]) + \beta_k[s(e)] \right)$$
$$= \max_{e:x(e)=x}{}^* \left(\alpha_{k-1}[s'(e)] + \sum_{p=1}^{k_0} (P_k[u^p(e); i]) + \sum_{\substack{q=1 \\ q \neq j}}^{n_0} (P_k[x^q(e); i]) + \beta_k[s(e)] \right) \tag{2.40}$$

Again, the second equations follow from the fact that when $P_k(u^j; o)$ and $P_k(x^j; o)$ are calculated, $P_k(u^j; i)$ and $P_k(x^j; i)$, respectively, are independent from the indices of the max* operation and can be removed from the summation.

Fig. 2.4 shows the SCCC decoder with SISO modules for the encoder shown in Fig. 1.8. Both inner and outer SISO decoder modules have four ports. Input and coded symbol sequences $\mathbf{u}$ and $\mathbf{x}$ have superscripts i or o to represent inner and outer decoder parameters, respectively. Since only the inner encoder is connected to the channel, only the inner SISO module receives probabilistic values for the coded bits of inner encoder, $\mathbf{P}(\mathbf{x}^i; i)$, from the demodulator. Extrinsic information input for the input bits of the inner decoder, $\mathbf{P}(\mathbf{u}^i; i)$ comes from the code bit

output of the outer SISO module which is all-zeros for the first iteration since it is not available from the outer SISO module yet. The inner SISO module only produces the extrinsic information for the input bits, $\mathbf{P}(\mathbf{u}^i; o)$, which becomes the input to the outer SISO decoder module as the extrinsic information for the coded bit after the deinterleaver and DEMUX. This is because the input sequence to the inner encoder are the code bits of the outer code after the interleaver and MUX. The two input arrows represents the two coded bit sequences since the outer encoder rate is 1/2. The other input to the outer SISO module is the extrinsic information for the input sequence, $\mathbf{P}(\mathbf{u}^o; i)$, which is always all-zero (equiprobable) since there is no extrinsic information is available to outer SISO module for the input bits. The outer SISO module then produces the soft information for the input bits, $\mathbf{P}(\mathbf{u}^o; o)$, which is the information needed to make decisions (hard limiting) for the information bits, u_k's after the last iteration. The soft information output for the coded bits, $\mathbf{P}(\mathbf{x}^o; o)$, of the outer code becomes the extrinsic information for the inner SISO module after the MUX and interleaver. This one cycle completes the first iteration. If more iterations are required, the inner SISO module starts decoding as we explained earlier and the cycle continues until a stopping criterion is met.

As there is a sliding window APP algorithm (SW-Log-MAP), there is also a sliding window SISO algorithm which reduces the decoding delay of the constituent SISO decoder modules. The only difference between the sliding window

Log-SISO (SW-Log-SISO) and the Log-SISO is the calculation of the backward recursion variables. To implement the SW-Log-SISO algorithm, one should use the equation and the boundary conditions in (2.30) for the SW-Log-MAP algorithm and replace the max* operation argument with the backward recursion equation for Log-SISO given in (2.36).

2.6 Summary

In this chapter, we introduced several decoding algorithms for CCs. Among them, the APP decoding algorithm derived by Bahl, et al., is employed in the decoding of PCCCs and SCCCs since it is optimum for minimizing the BER and the algorithm supplies soft decisions for each information bit.

The modified APP algorithm is shown to prevent the exponential growth of recursive parameters in the original APP algorithm. Another important modification is performed on the APP algorithm for allowing the soft decision to be written for iterative decoding.

The Log-MAP algorithm is derived from the original APP algorithm to implement the APP decoding in hardware. A final modification is made to reduce the decoding delay of the original algorithm for component decoder and this latest version is called SW-Log-MAP algorithm.

Finally we introduced the Log-SISO decoding algorithm to be used in the decoding of SCCCs in Chapter 5. The Log-SISO algorithm can produce extrinsic information on both input and coded bits (or symbols) of a Trellis code.

3 PUNCTURED PCCCs FOR BPSK/QPSK CHANNELS

3.1 Introduction

As communications applications increase in prevalence, bandwidth becomes increasingly scarce. Thus, communication systems designers are faced with the design of systems employing bandwidth-efficient modulation and coding schemes. In the satellite communication arena, rate 1/2 coded BPSK or QPSK systems are conventional, so that efficiencies of only 0.5 bps/Hz or 1 bps/Hz are achieved (assuming ideal Nyquist signaling). Trellis-coded M-PSK schemes have been proposed for this application [6] and [33], but carrier recovery can become problematic in this case since the receiver is forced to operate below the recovery loop's threshold [34]. Here, we consider an alternative approach to improving bandwidth efficiency: we design high rate binary PCCCs for BPSK and QPSK channels. BPSK/QPSK carrier recovery loops are capable of operation in the $E_s/N_0 = 0$ dB region [34], and thus carrier recovery is not a problem in principle for coded BPSK/QPSK receivers.

We consider the design of rate $r = k/(k+1)$ $(2 \leq k \leq 16)$ binary PCCCs with constituent encoder memory sizes $m = 3$ and 4 which achieve (theoretical) efficiencies of 1.33 through 1.88 bps/Hz on QPSK channels. Our focus is on PCCCs because these have been shown to achieve near-capacity performance for rates 1/2 and lower [35], [36], and [37]. Most of the previous work in the design of bandwidth-efficient PCCCs have focused on higher-order modulation schemes

such as M-PSK and M-QAM [36], [38], and [39]. Our work is more applicable to situations where carrier recovery is difficult for these higher-order schemes. The algorithm includes the selection of constituent encoder generator polynomials, puncture patterns, and interleavers with the goal of maximizing the minimum codeword weight for weight-two and weight-three inputs.

Hagenauer, et al. [23], gave error rate performance simulation results of several different high rate PCCCs by using punctured convolutional component codes. Riedel [40] derived a method from the earlier work of Hartmann, et al. [41], and Hagenauer, et al. [23], to obtain high-rate PCCCs by using high-rate convolutional component codes. He suggested employing the trellis of the reciprocal dual code of the constituent code at the decoder to reduce its complexity. An (n, k, m) convolutional code has 2^k branches leaving from each state in its trellis whereas its reciprocal dual encoder is an $(n, n-k, m)$ convolutional encoder with 2^{n-k} branches leaving from each state. Riedel used the fact that, for a high rate convolutional code with $k > (n-k)$, the reciprocal dual code has fewer branches leaving each state. Divsalar and Pollara [36] have also considered the design of high rate PCCCs, but their approach uses high rate constituent codes which lead to high branch complexities in the decoder.

Our approach is more classical (but effective) in that we derive high rate codes via puncturing a basic rate 1/3 PCCC composed of two rate 1/2 constituent encoders. Our design method involves a systematic computer search for optimal

constituent encoder generator polynomials, puncture patterns, and interleavers. Optimality is in the sense of the largest minimum codeword weight for weight-two and weight-three inputs and the minimum multiplicites for both.

In the next section, we will review the characteristics of PCCCs to address some of the design parameters involved in the computer search. Section 3.3 then describes the computer search algorithm. Section 3.4 presents the search results for the generator polynomial sets and puncturing schemes. Section 3.5 is devoted to a decoding complexity comparison between Riedel's binary high rate PCCCs and the method described in this chapter by employing the modified APP decoding algorithm. Applications of such high rate PCCCs are given in Section 3.6. Simulation results for the rates of interest on an AWGN channel are given in Section 3.7. Finally Section 3.8 contains concluding remarks.

3.2 Characteristics of PCCCs

Fig. 1.7 depicts a PCCC encoder composed of two RSC encoders. In the design of high rate PCCCs, it is assumed that the two convolutional encoders are identical.

Recall from (1.13) the generator matrix of the RSC encoders is $G_s(D) = \left[1 \quad \frac{g_2(D)}{g_1(D)}\right]$ where $g_1(D)$ and $g_2(D)$, respectively, give the feedback and feedforward connections of the RSC encoder. From this, it is clear that any RSC encoder input will result in a remergent trellis path iff it is divisible by $g_1(D)$. This sheds

light on the utility of the interleaver since if a PCCC encoder input $u(D)$ is divisible by $g_1(D)$, it is unlikely that its permuted version $u^I(D)$ will be divisible as well (and vice versa). Thus, it is unlikely that the two constituent encoders will simultaneously have remergent paths, making larger weight codewords more probable with minimum multiplicities.

It can be shown ([24] and [42]) that the asymptotic bit error probability for a maximum likelihood decoder on an AWGN channel is given by

$$P_b \simeq \max_w \frac{w n_w}{N} Q\left(\sqrt{\frac{2r d_{w,\min}^{PCCC} E_b}{N_0}}\right) \tag{3.1}$$

where $d_{w,\min}^{PCCC}$ is the minimum weight PCCC codeword for weight-w inputs, n_w (multiplicity) is the number of weight-w inputs $u(D)$ resulting in a weight-$d_{w,\min}^{PCCC}$ PCCC codeword, and E_b/N_0 is the user bit energy to one-sided noise power spectral density ratio. The maximizing w in (3.1) is primarily a function of the interleaver and is never equal to one since a weight-one input will lead to non-remergent paths in both RSC encoders. For an "average" interleaver, the maximizing value of w is usually two or three since for $w > 3$ it becomes increasingly unlikely that $u(D)$ and $u^I(D)$ are simultaneously divisible (s.d.) by $g_1(D)$ [24]. Thus, in our code designs below, which include interleaver designs, we produce interleavers which eliminate the more harmful weight-two and weight-three s.d. events. This will be stated more precisely in the next section.

As seen in Fig 1.7, only parity bits are punctured since deletion of systematic bits leads to inferior performance for iterative APP decoders. Thus, to achieve

a code rate of $k/(k+1)$, one parity bit is transmitted for every k information bits presented to the encoder input. We shall assume that the rates of the two constituent encoders after puncturing are the same and that the parity bits to be transmitted must alternate between the two encoders. Therefore, for every $2k$ inputs bits, only two parity bits are saved, one from each of the two constituent encoders. For most codes, we will consider only puncturers which, in effect, partition the parity sequence from each RSC encoder into $2k$-bit blocks, and saves only one bit in each such block. Further, the puncturers are periodic in the sense that the same bit in each $2k$-bit block is saved for both RSC encoders. We will use the notation $P(p,q)$ to indicate a puncturer which saves the p^{th} bit in every $2k$-bit block for the first RSC encoder and the q^{th} bit in every $2k$-bit block for the second RSC encoder, where $1 \leq p, q \leq 2k$. There are clearly $(2k)^2$ such puncturers.

For rates 7/8 and 14/15 for $m = 3$ and rates 5/6, 10/11, and 15/16 for $m = 4$, puncturers with periods other than $2k$ are necessary. We defer treatment of these special cases to Subsection 3.3.2.4.

3.3 The Design Algorithm

We discuss in this section the details of our design algorithm. We first start with the relevant design parameters and then discuss the algorithms employed to optimize these parameters.

3.3.1 Design Parameters

The design parameters are clearly the generator polynomials, (g_1, g_2), the interleaver I, and the puncturer $P(p, q)$. As pointed out above, weight-two and weight-three inputs and their multiplicities, n_2 and n_3, dominate performance, so we would like our optimality criterion to be the maximization of $d^{PCCC}_{2,\min}$ and $d^{PCCC}_{3,\min}$ and the minimization of n_2 and n_3 over the above parameters. For the block sizes of interest in this study, it is computationally impossible to vary (g_1, g_2), I, $P(p, q)$, and the weight-two and -three inputs to find the absolute maximum for $d^{PCCC}_{2,\min}$ and $d^{PCCC}_{3,\min}$ and minimum for n_2 and n_3. Thus, we constrain the above sets in a systematic manner to ensure near optimum values for $d^{PCCC}_{2,\min}$, $d^{PCCC}_{3,\min}$, n_2, and n_3. We will denote these "best" values for $d^{PCCC}_{2,\min}$ and $d^{PCCC}_{3,\min}$ by $d^{PCCC*}_{2,\min}$ and $d^{PCCC*}_{3,\min}$, respectively. Thus,

$$d^{PCCC*}_{i,\min} = \max_{g_1,g_2} \max_{S^i_t(g_1,g_2)} \max_{I} \max_{P(p,q)} d^{PCCC}_{i,\min}, \quad i = 2, 3 \tag{3.2}$$

where $S^2_t(g_1, g_2)$ and $S^3_t(g_1, g_2)$ are defined as

$$S^i_t(g_1, g_2) = \{\text{weight-}i \text{ inputs } u(D) \text{ given } (g_1, g_2)\text{: } (d^{CC}_{u(D)} - i) \leq t\}, \quad i = 2, 3 \tag{3.3}$$

where t is a parity weight threshold set to limit the size of $S^2_t(g_1, g_2)$ and $S^3_t(g_1, g_2)$ (to reduce the search time), and $d^{CC}_{u(D)}$ is the weight of a constituent code codeword for input $u(D)$. Note that equation (3.2) is an abstraction of the design algorithm. Maximization over $S^i_t(g_1, g_2)$ and I is implemented jointly by taking the union of $S^i_t(g_1, g_2)$ for all possible (g_1, g_2) pairs and using the resulting set to enhance a

pseudo-randomly generated interleaver I as detailed in Subsection 3.3.2.3. The maximization over $P(p,q)$ finds the largest minimum codeword weight for the enhanced interleaver and the current (g_1, g_2) pair by applying all possible puncturing schemes. If maximization over $P(p,q)$ finds more than one puncturing scheme with the same $d_{2,\min}^{PCCC}$ and $d_{3,\min}^{PCCC}$, it chooses the puncturing scheme with the minimum n_2 and n_3 (which we will denote by n_2^* and n_3^*). The following subsections elaborate further how the algorithm is implemented.

In addition to constraining the size of the weight-two and weight-three input sets in the optimization equation (3.2) to reduce the search time, we limit the set of interleavers considered. We do this by randomly generating an interleaver and then applying an algorithm to enhance it. The algorithm will be described in Subsection 3.3.2.3.

As for the set of polynomials considered in (3.2), we restrict the polynomials to have memory sizes $m = 3$ and 4. The feedback polynomials for both memory sizes are chosen to be primitive [36] and, thus, our choices for $m = 3$ are limited to $g_1 = 15$ and $g_1 = 13$ and, for $m = 4$, to $g_1 = 31$ and $g_1 = 23$. For the feedforward polynomials, we considered only the polynomials of the form

$$\begin{aligned} g_2(D) &= 1 + g_{21}D + g_{22}D^2 + D^3 \qquad (m = 3) \qquad (3.4) \\ g_2(D) &= 1 + g_{21}D + g_{22}D^2 + g_{23}D^3 + D^4 \qquad (m = 4) \end{aligned}$$

where the coefficients g_{21} and g_{22} are allowed to vary over all two-bit combinations

for $m = 3$ and g_{21}, g_{22}, and g_{23} are allowed to vary over all three-bit combinations for $m = 4$.

Since the set of puncture patterns for a rate $k/(k+1)$ code has cardinality $(2k)^2$, it is possible to vary over the entire set of puncturers in (3.2).

3.3.2 Algorithm Details

Referring to (3.2) above, we may consider (g_1, g_2) to be the outer loop variable, and $S_t^2(g_1, g_2)$ (or $S_t^3(g_1, g_2)$) the next loop variable, and so on. In this section, we discuss how the various parameters are varied to obtain $d_{2,\min}^{PCCC*}$ and $d_{3,\min}^{PCCC*}$, keeping in mind the optimization sequence set forth by the implicit loops in (3.2). For a summary of the design algorithm the reader is referred to Fig. 3.1.

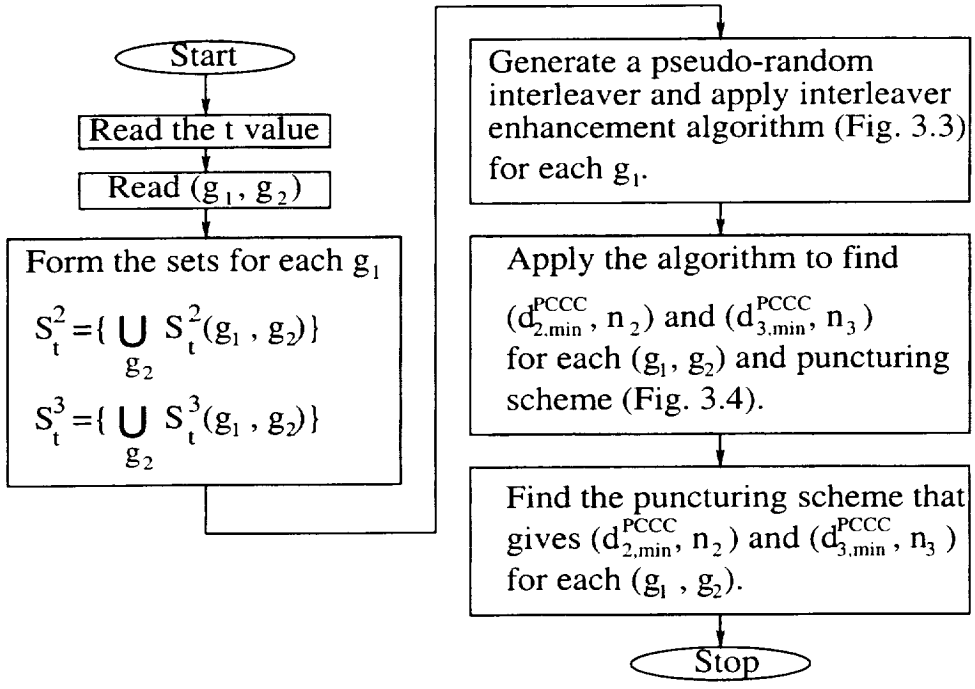

Figure 3.1: The outline of the algorithm.

3.3.2.1 The Set of RSC Codes

From the remarks above, for $m = 3$, $g_1 \in \{15, 13\}$ and for $m = 4$, $g_1 \in \{31, 23\}$. As for g_2, for $m = 3$, $g_2 \in \{11, 13, 15, 17\}$ and for $m = 4$, $g_2 \in \{21, 23, 25, 27, 31, 33, 35, 37\}$, but we can never have $g_1 = g_2$. There are thus, $8 - 2 = 6$ possible RSC codes to be considered for $m = 3$ (minus 2 for the two cases where $g_2 = g_1$), and $16 - 2 = 14$ possible RSC codes for $m = 4$. The set of 6 (or 14) codes can in principle be reduced by identifying equivalent codes in the sense of equivalent weight spectra. For example, a code (g_1, g_2, I) is equivalent to a code (g_1^*, g_2^*, I^*) where the asterisk indicates reciprocal polynomials for g_1 and g_2 and a "reciprocal" interleaver for I. [7] However, in our algorithm below, we will be fixing on one interleaver and then enhancing it for both (g_1, g_2) and (g_1^*, g_2^*), so that there is virtually no chance of the enhanced interleaver $(I')^*$ for (g_1^*, g_2^*) being the reciprocal of the enhanced interleaver I' for (g_1, g_2). As a consequence, we shall consider all 6 (or 14) combinations in our search.

3.3.2.2 The Sets $S_t^2(g_1, g_2)$ and $S_t^3(g_1, g_2)$

We need only discuss $S_t^2(g_1, g_2)$ here as similar comments will hold for $S_t^3(g_1, g_2)$. The set $S_t^2(g_1, g_2)$ is the set of weight-two polynomials $u(D)$ that yield a constituent code parity weight of at most t and is easily generated via computer. We remark that the parameter t should be set sufficiently large so that inputs leading

[7] The reciprocal polynomial, g_1^*, of g_1 is the time-reversed version of g_1. For example, the reciprocal polynomial of $g_1(D) = D^4 + D^3 + 1$ is $g_1^*(D) = D^4 + D + 1$. A reciprocal interleaver here indicates the time-reversed version of an interleaver. The two codes would have equal weight spectra since $u \longleftrightarrow c$ for the first code implies $u^* \longleftrightarrow c^*$ for the second code.

to dominant error events are included. This parameter was found empirically in all cases and must increase with k so as to account for the eventual puncturing that will be applied to the codewords corresponding to the polynomials in S_t^2 and S_t^3.

Because the sets $S_t^2(g_1, g_2)$ and $S_t^3(g_1, g_2)$ are functions of (g_1, g_2), in principle, the design algorithm described to this point would lead to 6 (or 14) separate interleavers as there are 6 (or 14) (g_1, g_2) pairs. But since $S_t^2(g_1, g_2)$ and $S_t^3(g_1, g_2)$ are stronger functions of g_1 than they are of g_2, we considered only two sets for $S_t^2(g_1, g_2)$and $S_t^3(g_1, g_2)$, one each for the two g_1 polynomials under consideration. For example, for a fixed g_1 (and t), we varied g_2 over its 3 (or 7) possibilities and formed the union of all sets S_t^2 generated (similarly, for S_t^3).[8] In this manner, we considered only two enhanced interleavers for all 6 (or 14) codes.

3.3.2.3 The Interleaver Enhancement Algorithm

As mentioned above, the interleaver is first generated pseudo-randomly and then enhanced. The MATLAB program used to generate the initial interleaver is given below (it was run with the default start-up seed: 931316785). Once we have an interleaver generated randomly, the next step is to modify it for weight-two and weight-three inputs according to certain rules which we presently describe.

[8] We dropped the (g_1, g_2) argument from $S_t^2(g_1, g_2)$ and $S_t^3(g_1, g_2)$, since the union sets are only a function of g_1.

```
N=10000;
for i=1:N,
   P(i)=round((N-1)*rand)+1;
   while length(find(P(1:i-1)==P(i))) ~= 0
       P(i)=round((N-1)*rand)+1;
   end
end
```

In Fig. 3.2 the functionality of the interleaver with respect of input bits u_k and the interleaved bits u_k^I is presented where $s < k < t$.

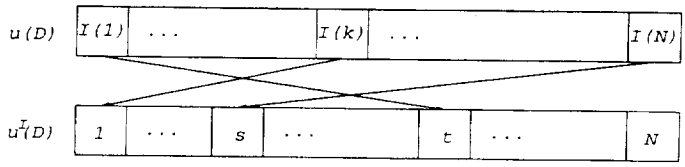

Figure 3.2: The functionality of the interleaver for the input u_k and the interleaved output u_k^I.

a. Weight-Two Input Rules

Fig. 3.3 gives the interleaver enhancement algorithm for weight-two inputs in the set S_t^2. One goal is to iteratively modify the interleaver so that if the interleaver input $u(D) \in S_t^2$, then the interleaver output $u^I(D)$ will not be in the set S_t^2 (since the elements in this set produce relatively low weight RSC codewords). As shown in the figure, this is done as follows. Suppose $u(D) = D^y + D^z \in S_t^2$ where $y < z$. Then if $u^I(D) = D^{I(y)} + D^{I(z)} \in S_t^2$, we first generate a random number in the set

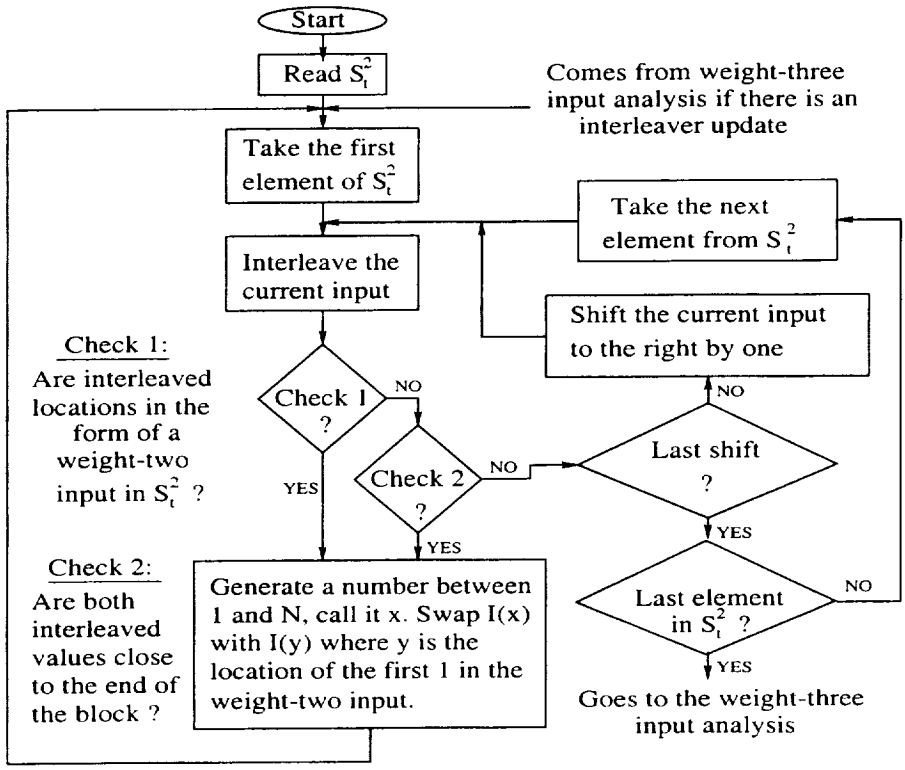

Figure 3.3: The interleaver enhancement algorithm for weight-two input.

$\{1, 2, ..., N\}$, x say. We then swap $I(y)$ and $I(x)$ so that now $u^I(D) = D^{I(x)} + D^{I(z)}$ for the enhanced interleaver.

A second goal is to avoid situations at the interleaver output for which $I(y)$ and $I(z)$ are both "close" (within 150) to the interleaver length N, since in this case $u^I(D)$ is certain to produce a low weight output. This situation is corrected in the same manner.

Note that whenever the interleaver is modified, the algorithm resets itself to the first element in S_t^2 and reiterates. Also, as seen in the figure, when all of the

above conditions are satisfied for all of the elements in S_t^2, the interleaver is sent to the weight-three algorithm for further enhancement.

b. Weight-Three Input Rules

The algorithm for this case is analogous to that of the weight-two case. That is, we would like to eliminate situations for which $u(D) \in S_t^3$ results in $u^I(D) \in S_t^3$ or $I(y)$, $I(z)$, and $I(v)$ are all "close" to N. These situations are corrected using the same random swapping procedure. We check for an additional special case for which $D^{I(y)} + D^{I(z)} \in S_t^2$ and $I(v)$ is close to N $(I(y) < I(z) < I(v))$. [We consider only a subset of S_t^2 (by reducing t) since otherwise the algorithm requires an excessive amount of time.]

Finally, whenever any modification is made, the modifier goes back to the weight-two input modifications starting from the first element of the S_t^2 to ensure that no weight-two rules are violated. The interleaver enhancement algorithm stops when the interleaver can pass through the weight-three algorithm with no modifications.

3.3.2.4 Iterating over (g_1, g_2) and $P(p, q)$

As mentioned above, we consider 6 (or 14) different pairs of polynomials (g_1, g_2), and for each pair there are $(2k)^2$ puncturing patterns to consider. Fig. 3.4 depicts the algorithm for iterating over $P(p, q)$ in the computation of $d_{2,\min}^{PCCC*}$ for a given code specification (g_1, g_2) and enhanced interleaver. (The algorithm for $d_{3,\min}^{PCCC*}$ is identical.) The algorithm will generate a table of puncturing schemes and

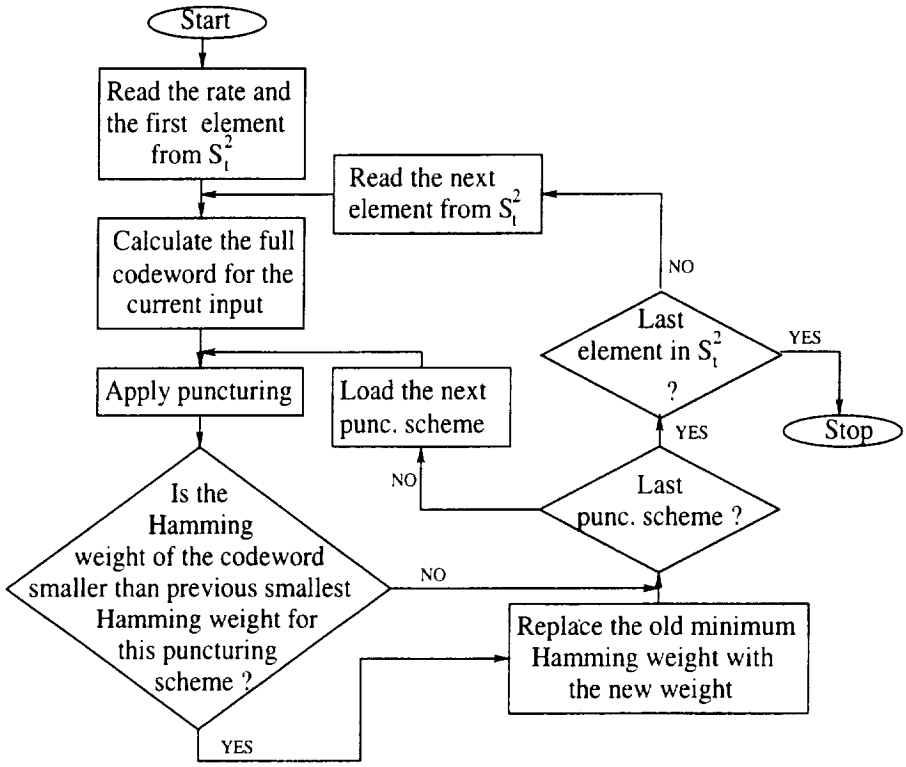

Figure 3.4: Algorithm for finding $d_{2,min}^{PCCC}$ for each puncturing scheme for a given code.

their minimum codeword weights. An analogous algorithm for $d_{3,\min}^{PCCC*}$ produces a similar table.

We found that most of the time the puncturing scheme(s) that gives $d_{2,\min}^{PCCC*}$ is not the same as the puncturing scheme(s) that gives $d_{3,\min}^{PCCC*}$. In those cases, the puncturing scheme is chosen to simultaneously optimize $d_{2,\min}^{PCCC*}$ and $d_{3,\min}^{PCCC*}$. The simultaneous optimization process can be explained with the help of equation (3.1). For a given N, E_b/N_0, and rate r, the error rate P_b depends on wn_w and $d_{w,\min}^{PCCC}$. Thus, in choosing the puncturing pattern, we weight the contributions of

the distances $d_{2,\min}^{PCCC}$ and $d_{3,\min}^{PCCC}$ and the multiplicities $2n_2$ and $3n_3$, choosing in favor the pattern which will minimize P_b for large SNR's.

We also found that, for rates 7/8 and 14/15 for $m = 3$, and rates 5/6, 10/11, and 15/16 for $m = 4$, puncturing with a period of $2k$ gave very poor codes. This can be explained as follows. Note that a single data 1 input will yield a periodic pattern of parity bits at a constituent encoder output (after a brief initial transient which is a function of the initial state of the encoder) [43] and [44]. Further, since the feedback polynomial g_1 is primitive, the period will be maximal length, $2^m - 1$. Now since the encoder is linear, an input of two or more data 1's will yield a sum of shifted versions of periodic patterns, and is essentially
periodic with period $2^m - 1$ (except for the transient responses and phase changes corresponding to each data 1). Now consider a puncture pattern for the $m = 4$, rate 5/6 code, which retains every 10^{th} constituent code parity bit: without loss of generality, let us retain bits 1, 11, 21, 31, 41, etc. In the context of the length-15 periods, we thus retain bits 1 and 11 in the first period, then bit 6 (= 21 mod 15) in the second period, then bits 1 and 11 in the third (1=31 mod 15 and 11=41 mod 15), and so on. Thus, over the whole parity word, we retain only bits 1, 11, and 6 (mod 15), over and over again. But suppose these bits are all zero (recalling the periodicity of the parity sequence). Then the punctured encoder has not increased the Hamming weight beyond that of the information word and we cannot expect a better performance than the uncoded case.

The situation is analogous for the $m = 4$, rate 10/11 code. For the $m = 4$, rate 15/16 code, matters are somewhat worse as only one bit per two length-15 periods is retained (the same bit (mod 15) every time). For comparison, the rate 2/3 code retains 15 different (mod 15) parity bits over the length of the parity word and the rate 3/4 code retains 5 different (mod 15) parity bits.[9] Similar comments hold for the $m = 3$, rate 7/8 and 14/15 codes.

The solution to this problem is to increase the number of "different" saved parity bit locations in every length-$(2^m - 1)$ period. One way to increase the number of different saved parity bit locations is to allocate locations, $\{s_n\}$, for the saved parity bits pseudo-randomly according to

$$\begin{aligned} q_n &= 1 + [(q_{n-1} - 1 + f_m) \bmod 2k] \\ s_n &= 1 + [(q_n - 1 + 2k(n-1) \bmod (2^m - 1)] \end{aligned} \tag{3.5}$$

for $n = 1, 2, ..., \lfloor N/2k \rfloor$ ($\lfloor x \rfloor$ is integer part of x) where $f_3 = 3$ and $f_4 = 7$ for $k = 5$ and 15, and $f_4 = 17$ for $k = 10$. The sequence $\{q_n\}$ represents the saved parity bit locations in every length-$2k$ period (mod 2k) and $\{s_n\}$ represents the saved parity bit locations in every length-$(2^m - 1)$ period (mod $2^m - 1$). In all cases, we initialized $q_0 = 5$.

As an example, for $m = 4$, the sequence $s = (s_1, s_2, s_3, ...)$ is $s = (2, 4, 11, ...)$ when $k = 5$ (rate 5/6). Thus, the following component encoder parity bits are saved: 2, 19, 41,..., $s_n + 15(n-1)$, ... We point out that the period of the sequence

[9] It is easily shown that the number of different parity positions retained within the length-$(2^m - 1)$ period is $Q_m = (2^m - 1)/\gcd[2k, 2^m - 1]$.

s is 30, with the numbers 1, 2, ..., 15 each appearing twice in each period. Thus, we have increased the number of different retained parity bit locations from 3 to 15.

3.4 Design Results

We present in this section code design results, the generator polynomial set and the puncturing scheme for the enhanced interleaver. We treat the $m = 3$ and $m = 4$ cases separately.

3.4.1 $m = 3$ Case

Although we observed that for $m = 3$, the six candidate polynomial sets given in Subsection 3.3.2.1 gave similar minimum codeword weights $d_{2,\min}^{PCCC*}$ and $d_{3,\min}^{PCCC*}$ for all the rates of interest, the polynomial set $(g_1, g_2) = (15, 11)$ resulted in the slightly better $d_{2,\min}^{PCCC*}$ and $d_{3,\min}^{PCCC*}$ for rates 2/3, 3/4, and 4/5. These results are also supported by simulated bit error rate curves in Section 3.7.1 which showed $P_b = 10^{-5}$ was possible at about 0.85 dB from capacity. Table 3.1 summarizes the weights, $d_{2,\min}^{PCCC*}$ and $d_{3,\min}^{PCCC*}$, and the multiplicities, n_2^* and n_3^*, for the $(15, 11)$ code, for the rates 2/3, 3/4, and 4/5.

Search for near-optimum polynomial sets and puncturing schemes became unnecessary for rates greater than 4/5 since the $d_{2,\min}^{PCCC*}$ and $d_{3,\min}^{PCCC*}$ values were almost identical for all puncturing schemes. Therefore, we have chosen the polynomial set $(15, 11)$ as the favored generator polynomial for rates greater than 4/5.

Table 3.1: The design results for rates 2/3, 3/4, and 4/5, $m = 3$.

Rate	t	Selected Poly.	Selected P(p,q)	$(d_{2,\min}^{PCCC*}, n_2^*)$	$(d_{3,\min}^{PCCC*}, n_3^*)$
2/3	60	(15,11)	P(2,1)	(20,2)	(16,1)
3/4	60	(15,11)	P(2,4)	(13,2)	(12,1)
4/5	60	(15,11)	P(2,7)	(10,1)	(9,2)

Simulation results for these rates showed that $P_b = 10^{-5}$ is achievable at about 0.9 dB from capacity with a fixed puncturing scheme $P(2, 2)$ (except for the special rates mentioned earlier).

In summary, the polynomial set $(15, 11)$ can be used for all codes to achieve near-capacity performance for all rates of interest. The puncture pattern $P(2, 2)$ is near optimal for $r > 4/5$, except for the rates 7/8 and 14/15 which require the pseudo-random puncturing scheme.

3.4.2 $m = 4$ Case

Table 3.2 presents the code parameters resulting from the above design procedure for rates $2/3, 3/4$, and $4/5$. We note that the rate 2/3 and 3/4 codes have the same code polynomials, $(g_1, g_2) = (23, 31)$. We also point out that the performance of the rate 4/5 code with these polynomials (instead of $(31, 25)$ as indicated in Table 3.2) is only marginally inferior when the puncture pattern $P(1, 2)$ is used.

As in the $m = 3$ case, searches for near-optimal polynomial sets and puncturing schemes becomes pointless for rates greater than 4/5 because almost all the puncturing schemes give the same $d_{2,\min}^{PCCC*}$ and $d_{3,\min}^{PCCC*}$. Therefore, for rates

Table 3.2: The design results for rates 2/3, 3/4, and 4/5, $m = 4$.

Rate	t	Selected Poly.	Selected P(p,q)	$(d_{2,\min}^{PCCC*}, n_2^*)$	$(d_{3,\min}^{PCCC*}, n_3^*)$
2/3	50	(23,31)	P(3,4)	(25,1)	(20,4)
3/4	60	(23,31)	P(3,5)	(17,3)	(13,1)
4/5	70	(31,25)	P(7,6)	(11,1)	(11,1)

greater than 4/5 (except the special rates we have noted above), we applied the puncturing scheme $P(2,2)$ to the polynomial set $(23,31)$. We chose this polynomial set since we have seen via simulation that it has superior or comparable performance to all other polynomial sets for the rates 2/3, 3/4, and 4/5. This choice turned out to be justified, as our simulations in Section 3.7.2 have demonstrated performance about 0.75 dB from capacity in each case. We also remark that these polynomials result in quasi-transparent[10] PCCCs as they both have odd weight [45].

In summary, we can use the polynomial set $(23,31)$ for all codes, with near-capacity performance in all cases. The puncture pattern $P(2,2)$ is near optimal for the higher rate codes, except for rates 5/6, 10/11, and 15/16, which require a pseudo-random puncturing scheme.

[10]A constituent encoder is quasi-transparent if for an input sequence u with initial encoder state s generates an output sequence c, then u^c with the initial state s^c generates c^c. u^c is conjugate version of u.

Table 3.3: $d_{2,min}^{PCCC*}$, $d_{3,min}^{PCCC*}$ values for the rates greater than 4/5.

	$(d_{2,\min}^{PCCC*}, n_2^*)$		$(d_{3,\min}^{PCCC*}, n_3^*)$	
Rate	$m=3$	$m=4$	$m=3$	$m=4$
5/6	(8,1)	(8,2)	(6,1)	(8,4)
6/7	(7,1)	(8,3)	(7,5)	(6,1)
7/8	(9,1)	(6,1)	(9,1)	(5,2)
8/9	(6,9)	(5,1)	(4,1)	(5,5)
9/10	(5,1)	(5,4)	(4,1)	(5,5)
10/11	(4,1)	(5,6)	(3,1)	(4,5)
11/12	(4,3)	(4,1)	(3,1)	(4,2)
12/13	(5,33)	(4,1)	(4,5)	(4,3)
13/14	(4,8)	(3,1)	(3,2)	(3,1)
14/15	(4,1)	(3,1)	(5,2)	(4,9)
15/16	(3,5)	(3,3)	(3,1)	(3,1)
16/17	(3,4)	(3,2)	(3,2)	(3,5)

3.4.3 Weights of Rates Grater Than 5/6 for $m = 3$ and 4

We have tabulated in Table 3.3 $d_{2,\min}^{PCCC*}$ and $d_{3,\min}^{PCCC*}$ values for rates larger than 4/5. For $m = 3$, the $(15, 11)$ code is used and, for $m = 4$, the $(23, 31)$ code is used, with the fixed puncturing scheme $P(2, 2)$ in both cases (except the special rates). Note in Table 3.3, that for $m = 3$, the rate 7/8 code (one of the special rates) has $d_{2,\min}^{PCCC*} = (9, 1)$ and the rate 6/7 code has $d_{2,\min}^{PCCC*} = (7, 1)$. This unusual behavior, where the higher rate code has a superior weight spectrum, is more pronounced for $d_{3,\min}^{PCCC*}$ between rates 13/14 and 14/15 (special rate). But for $m = 4$, no such situation occurs when one of the special rates is involved. However, this situation arises for $d_{3,\min}^{PCCC*}$ between rates 13/14 and 14/15 of which neither is a special rate.

This is so simply by our favoring the optimization of $(d_{2,\min}^{PCCC}, n_2)$ over $(d_{3,\min}^{PCCC}, n_3)$ in our design.

The improved weight spectra due to pseudo-random puncturing is attributed the fact that the puncturer is designed to increase the number of different parity positions retained within the length-$(2^m - 1)$ period. As mentioned in Section 3.3.2.4, for a rate $k/(k+1)$ PCCC, the number of different parity positions retained within the length-$(2^m - 1)$ period is given by $Q_m = (2^m - 1)/\gcd[2k, (2^m - 1)]$. We expect the rates with high Q_m ($Q_3 \in \{1, 7\}$ and $Q_4 \in \{1, 3, 5, 15\}$) to have larger minimum weight. For $m = 3$, rates 6/7 and 7/8 (the latter with pseudo-random puncturing) have $Q_3 = 7$. In some cases, however, higher Q_m value may not improve the weight spectrum at all. For example, for $m = 4$, rates 11/12 and 12/13 have the same optimum weight with close multiplicity, but $Q_4 = 15$ for rate 11/12 and $Q_4 = 5$ for rate 12/13.

3.5 Computational Complexity Comparison with Riedel's Method

As mentioned in the introductory section, Riedel [40] proposed an alternative method to construct high-rate PCCCs. His method is based on the work of C. Hartmann and L. Rudolph [41] for linear codes, and outlined by J. Hagenauer, et al. [23], for the case when *a priori* information is available and soft decisions are possible. The method in [23] and [41] suggests a reduction in decoding complexity when a (n, k) linear code C is decoded using the trellis of its $(n, n-k)$ dual code

$C^{\perp}$ when $n-k<k$ holds.[11] Riedel showed that by using the reciprocal dual code $\tilde{C}^{\perp}$ of C in the decoding process, its complexity can be reduced. The rate of a PCCC constructed by rate R_1 and R_2 component codes is given by

$$R=\frac{1}{\frac{1}{R_1}+\frac{1}{R_2}-1}$$

where R_1 is the rate of the first (top) RSC encoder and R_2 is the rate of the second (bottom) RSC encoder. Riedel achieves a rate $R=\frac{k}{k+1}$ PCCC by using two identical rate $R_1=R_2=\frac{2k}{2k+1}$ RSC codes. For an RSC code with memory m, Riedel's method requires[12]

$$(1+\frac{3}{2k})\cdot 2^{m+1} \text{ or less} \tag{3.6}$$

multiplications and

$$(1+\frac{1}{2k})\cdot 2^{m+1} \tag{3.7}$$

additions to decode one information bit per iteration for each component decoder. By avoiding the calculation of extrinsic information directly, it is possible to implement Riedel's method with the number of additions and multiplications given in equations (3.6) and (3.7) which are less than given in [40]. These numbers do not include the evaluation of tanh(.) and scaling of the channel values with the estimated channel SNR value, L_c, in (2.17).

[11]Note that this condition always holds for rate $k/k+1$ codes.

[12]The computational complexity calculations for Riedel and the modified APP algorithms were provided to us by Dr. S. Pietrobon. He also pointed out that Riedel's algorithm requires complicated addition circuitry and lookup tables when implemented in the log domain.

The rate $R = \frac{k}{k+1}$ PCCC with two identical rate $R_1 = R_2 = \frac{2k}{2k+1}$ RSC via puncturing, decoded by the modified APP algorithm of [7], [23], and [24] with a component code memory of m will require

$$3 \cdot 2^{m+1} + \frac{1}{2k} \tag{3.8}$$

multiplications and

$$2^{m+2} - 2 \tag{3.9}$$

additions to decode one information bit per iteration.[13] Again these numbers do not include the scaling of the channel values with L_c. Thus, in this multiplicative form, Riedel's method is less complex than the modified APP algorithm for a given rate and m.

As noted in Section 2.3, for hardware implementations, Log-MAP algorithm should be employed which requires the evaluation of the max* operation. However, Riedel's method in log domain requires taking the log of negative numbers which result in complex numbers. This fact requires usage of more complex adders and lookup tables for the max* operator. Thus, in the log domain, Riedel's method is inferior to punctured PCCCs.

3.6 Applications

We now give two specific applications for variable-rate punctured PCCCs: Low-orbit-to-geostationary satellite links and rate compatible punctured PCCCs

[13] These numbers are also supplied by Dr. S. Pietrobon.

(RCP-PCCCs). We remark that Riedel's method [40] is not suitable for variable rate applications since each rate requires a different set of component codes.

3.6.1 Low-Orbit-to-Geostationary Satellite Links

A low-orbit-to-geostationary satellite link is a communication link that allows a low-earth-orbit (LEO) satellite to communicate with a ground station via the geostationary (GEO) satellite assigned to this ground station [46]. In [46], an approach is given for the maximization of the throughput (bits/day) of a link between a LEO (with a non-gimbaled antenna) and a GEO satellite by optimizing certain satellite parameters: signaling rate (R_s), small satellite antenna bandwidth, modulation scheme, and coding scheme. The interested reader is referred to [46] for details.

Since the LEO satellite will be in the view of the GEO satellite for a limited time, with changing angle and link distance between the two, the carrier-power-to-noise density C/N_0 in the link will be time varying. When there is a contact, the $C(t)/N_0$ profile can be approximated as

$$C(t)/N_0 = A_0 \exp(-t^2/2\sigma^2) \tag{3.10}$$

where the peak value A_0 occurs at time $t = 0$, and σ is a function of the duration of the link. For a chosen code bit rate, R_c, an $E_c(t)/N_0$ profile can be obtain from (3.10) as

$$E_c(t)/N_0 = \frac{A_0}{R_c} \exp(-t^2/2\sigma^2) \tag{3.11}$$

where $E_c(t)$ is the time-varying channel bit energy and is equal to $C(t)/R_c$. We assume communication is possible only for $E_c(t)/N_0$ values above 0 dB since carrier and timing recovery is difficult for MPSK schemes when $E_c(t)/N_0 < 0$ [34].

In [46], three coding schemes are considered to maximize throughput. The scheme with a variable rate ($\frac{1}{2}$, $\frac{2}{3}$, $\frac{3}{4}$, and $\frac{4}{5}$) inner convolutional code concatenated with a variable-rate $(255, k)$ outer Reed-Solomon code ($k = 223, 225, ..., 255$) gave the maximum throughput. The efficiency of this scheme relative to capacity is 77%, that is, $\mathcal{T}_{CC-RS}/\mathcal{T}_C = 0.77$ where $\mathcal{T}_{CC-RS}$ is the throughput of this scheme in bits/contact, and $\mathcal{T}_C$ is the theoretical throughput in bits/contact calculated from capacity. When a variable rate punctured PCCC scheme is applied to the same $E_c(t)/N_0$ profile used in [46], that is, equation (3.11) with $A_0 = 2$, the rate assignment shown in Fig. 3.5 is obtained for $m = 4$ and $P(2, 2)$ with a $P_b \leq 10^{-5}$ performance constraint on the probability of error.[14] For this rate assignment, the throughput efficiency becomes $\mathcal{T}_{PCCC}/\mathcal{T}_C = 0.893$, where $\mathcal{T}_{PCCC}$ is the throughput of the variable-rate punctured PCCC scheme.

3.6.2 Rate Compatible Punctured PCCCs (RCP-PCCCs)

Rowitch and Milstein [47] introduced a hybrid FEC/ARQ system with RCP-PCCC codes to enhance the throughput of a nonstationary Gaussian channel. They applied Rate Compatible Punctured Convolutional (RCPC) codes intro-

[14] Figure interpretation: an E_c/N_0 of about 1.1 dB is required to achieve an error rate $P_b \leq 10^{-5}$ for a rate 3/4 code, and similarly for the other rates. The channel bit number represents a unit of time over this time-varying communication link.

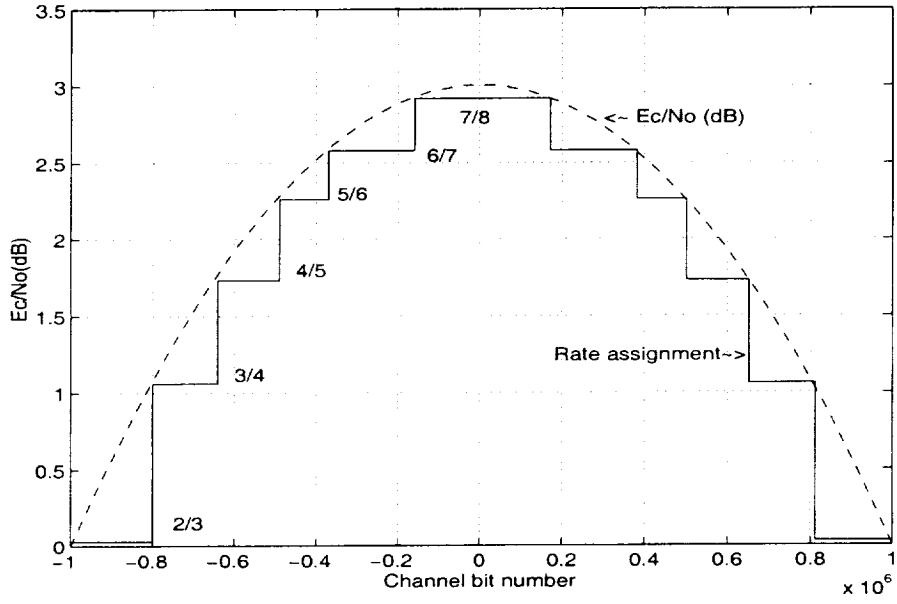

Figure 3.5: Code rate assignment on a given E_c/N_0 profile.

duced by Hagenauer in [48] to PCCCs to maintain a certain performance level by adjusting the rate of the PCCC.

It is possible to apply the hybrid FEC/ARQ system defined in [47] to the high-rate PCCCs outlined in this study with certain limitations on the set of possible rates and puncturing schemes. The set of possible rates can be expressed as

$$V = \left\{ \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{4}{5}, \frac{8}{9}, \frac{16}{17} \right\}$$

Note that except for rate $\frac{1}{3}$, the rates are in the form of $\frac{2^a}{2^a+1}$ for $a = 0, 1, .., 4$. A rate $k/(k+1)$, $P(p,q)$-punctured PCCC is achieved by saving the p^{th} bit from every $2k$-bit parity block in the first encoder, and the q^{th} bit from every $2k$-bit parity block in the second encoder. Therefore, the codes with rates in the set V follow this puncturing rule while allowing the insertion of extra parity bits to

methodically lower the rate. Fig. 3.6 shows how to lower the rate first from $\frac{4}{5}$ to $\frac{2}{3}$, and then from $\frac{2}{3}$ to $\frac{1}{2}$ in one of the parity bit sequences for $P(2,2)$. Note that added parity bits are equally distributed and the distance between any of the two is $2k$ where it is equal to 8 for the rate transition from $\frac{4}{5}$ to $\frac{2}{3}$ and 4 for the rate transition from $\frac{2}{3}$ to $\frac{1}{2}$.

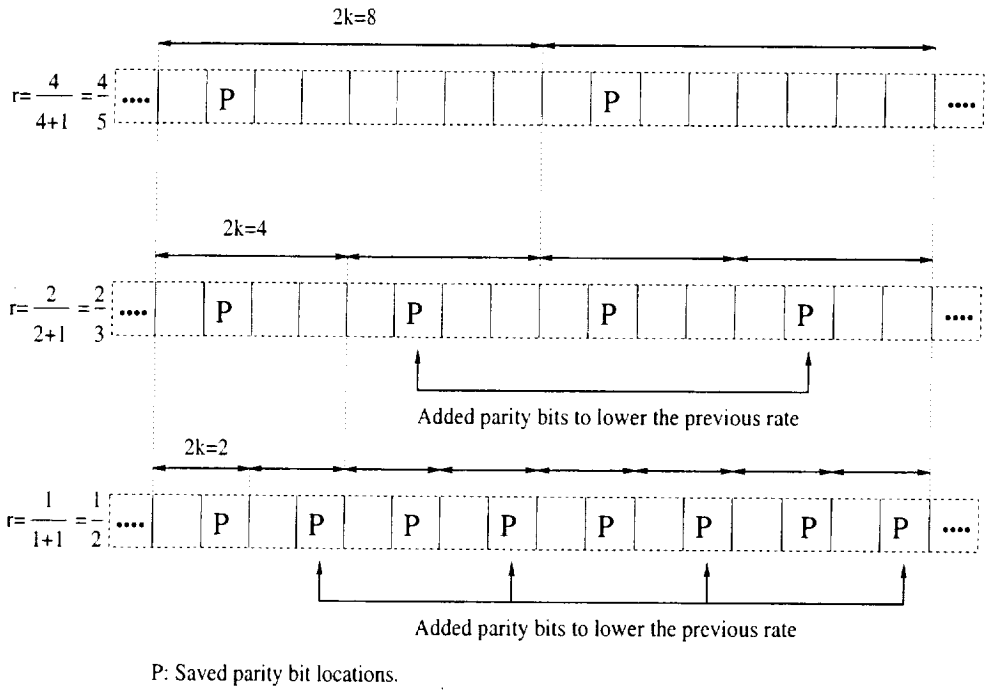

Figure 3.6: Rate decreasing process of RCP-PCCC. Each block represents an RSC parity bit sequence.

In our example, $\frac{16}{17}$ is the highest rate of the hybrid FEC/ARQ which is the initial rate of transmission. As long as the transmitter receives an ACK signal from the receiver, the transmitter keeps sending codewords at this rate by sending N information bits and $N/16$ parity bits: $(N/2)/16$ parity bits from each component

encoder. When the transmitter receives a NAK, it then switches to the next lower rate, $\frac{8}{9}$, by sending $N/16$ extra parity bits. If a second NAK occurs, then $N/8$ additional parity bits are sent to lower the transmission rate from $\frac{8}{9}$ to $\frac{4}{5}$. If the minimum possible rate of $\frac{1}{3}$ has been reached and the transmitter receives a NAK, the transmitter then resets to the highest rate of $\frac{16}{17}$ and retransmits the entire codeword.

3.7 Simulation Results

In this section we give the simulation results for the rates $r = k/k+1$, $k = 2, 3, \ldots, 16$. In all cases, we assume $N = 10,000$ and 15 decoder iterations, where the constituent decoders employ the "block-oriented" modified MAP algorithm [10] (i.e., not a sliding window MAP algorithm [30] and [31]). In our simulations, we terminated only the first RSC encoder to the zero state.

3.7.1 $m = 3$ Case

The following, Figs. 3.7 through 3.21, are the high rate PCCCs BER simulations for rates 2/3 through 16/17, respectively, designed by the algorithm given in this chapter with $m = 3$. We also added rate 3/4, 7/8, and 1/2 convolutional codes' BER performance [49] to compare with rate 3/4, 7/8, and 13/14 PCCCs, respectively. When the rate 3/4 and 7/8 PCCC are employed, a 3 dB gain is possible at BER of 10^{-5} over the same rate convolutional codes as shown in Fig. 3.8

and Fig. 3.12, respectively. In Fig. 3.18 rate 1/2 convolutional code versus rate 13/14 PCCC is given that shows around 10^{-5} and 10^{-6} BER region, the performance of the convolutional code becomes inferior to the PCCC for the same signal power. Therefore using the rate 13/14 PCCC yields around 86% more bandwidth.

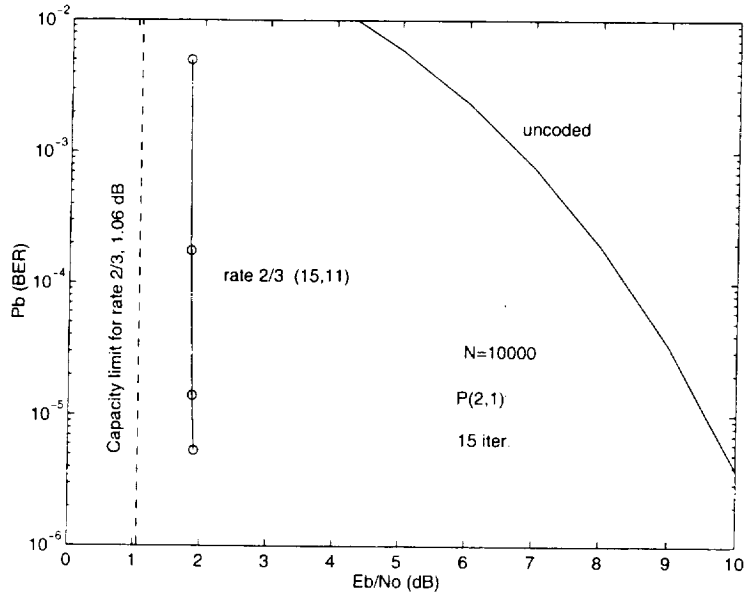

Figure 3.7: Rate 2/3 PCCC BER performance.

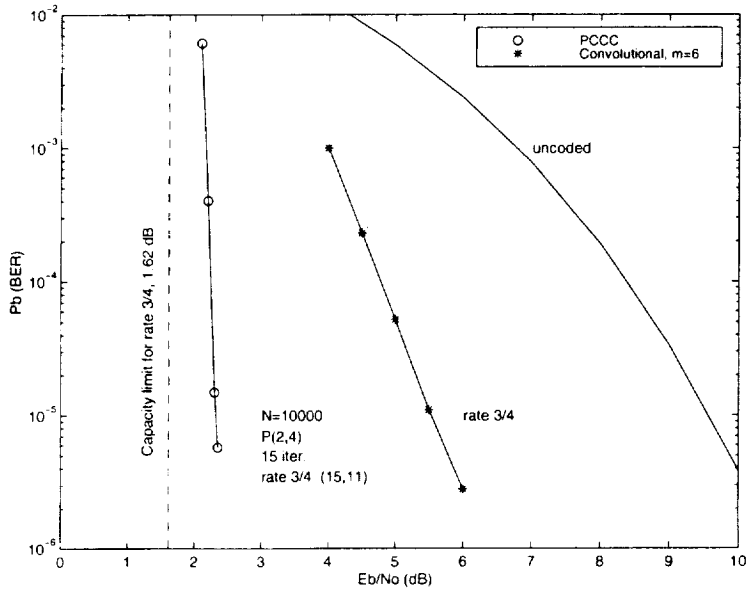

Figure 3.8: Rate 3/4 PCCC BER performance.

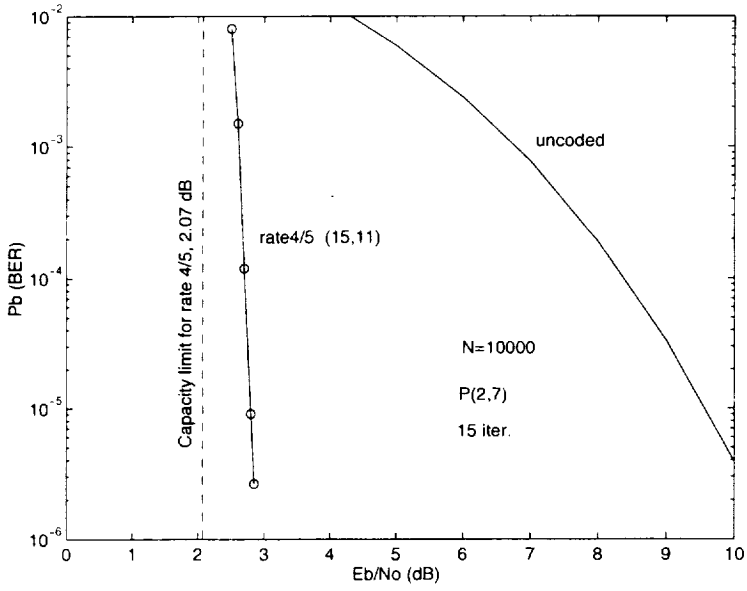

Figure 3.9: Rate 4/5 PCCC BER performance.

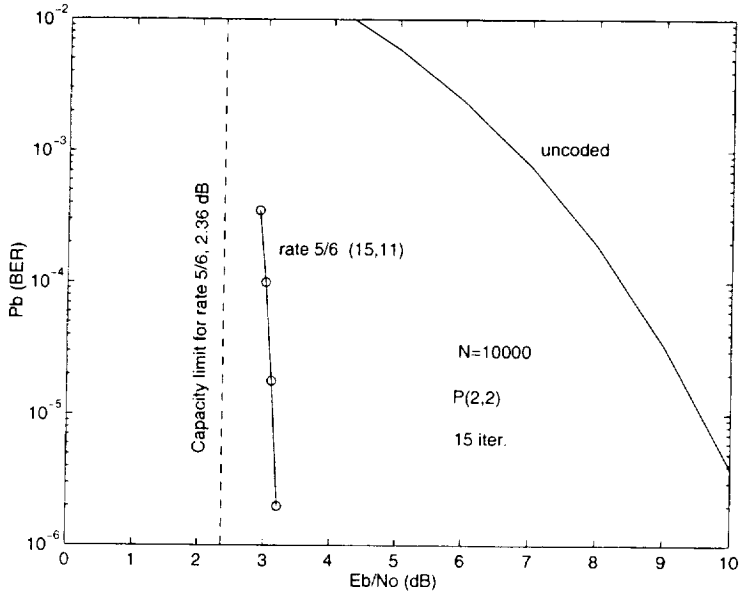

Figure 3.10: Rate 5/6 PCCC BER performance.

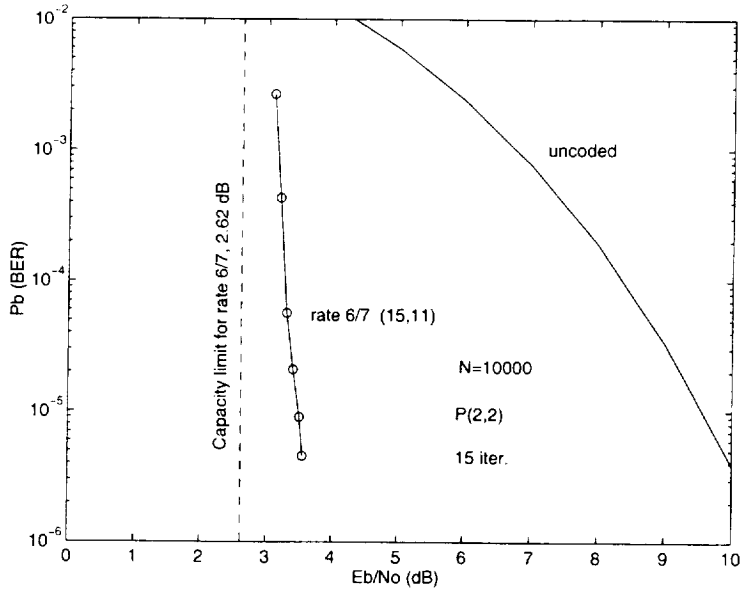

Figure 3.11: Rate 6/7 PCCC BER performance.

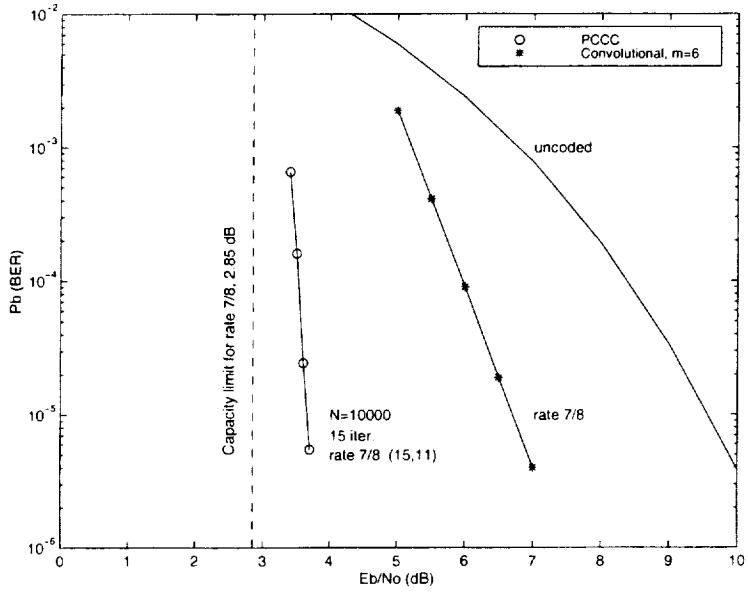

Figure 3.12: Rate 7/8 PCCC BER performance.

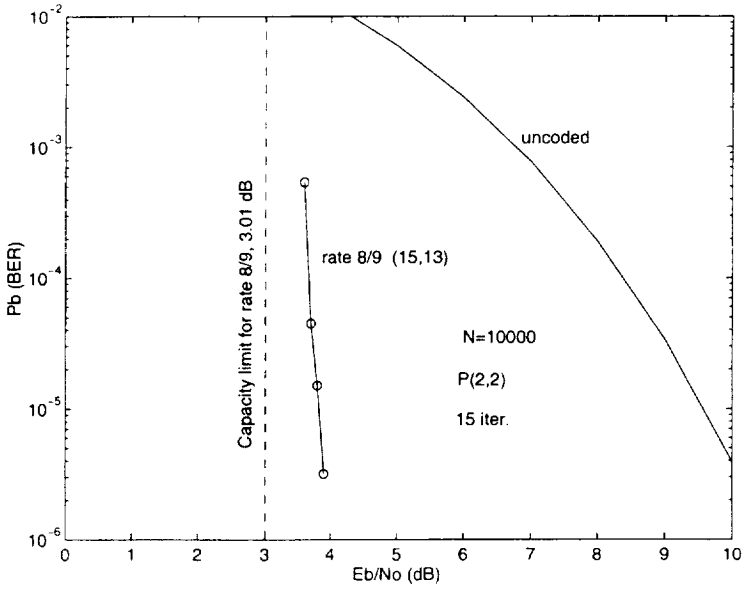

Figure 3.13: Rate 8/9 PCCC BER performance.

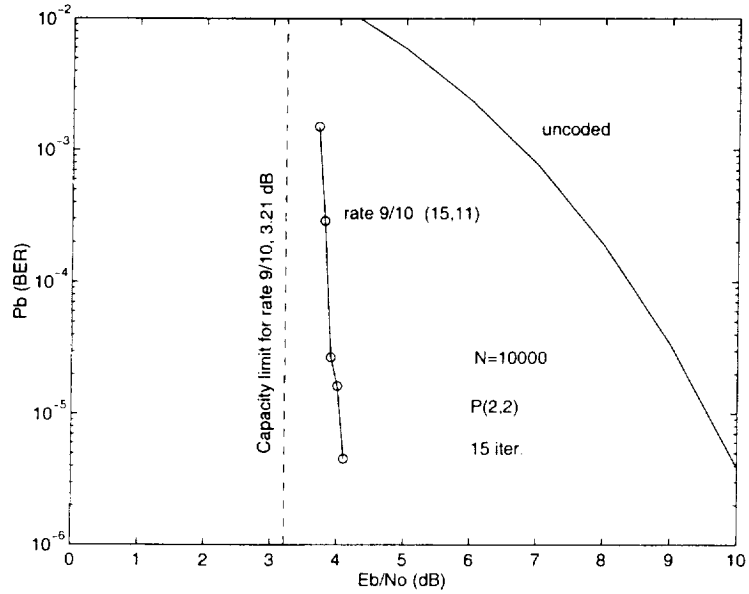

Figure 3.14: Rate 9/10 PCCC BER performance.

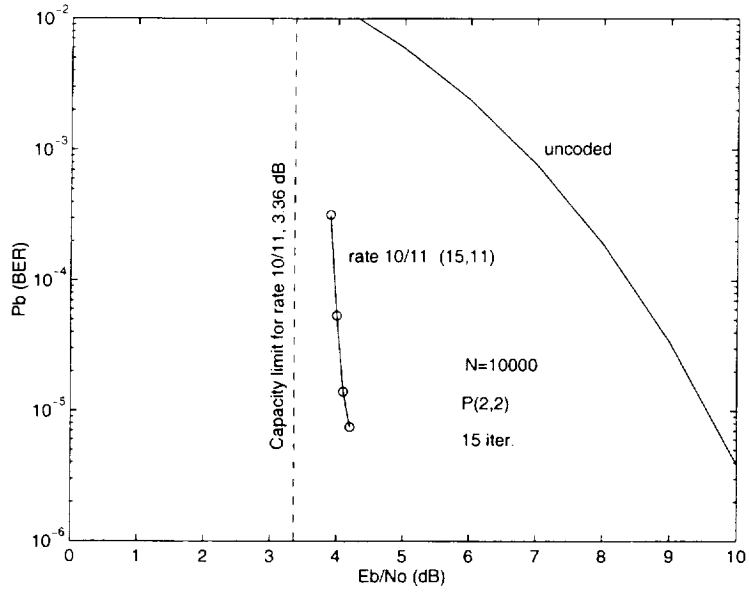

Figure 3.15: Rate 10/11 PCCC BER performance.

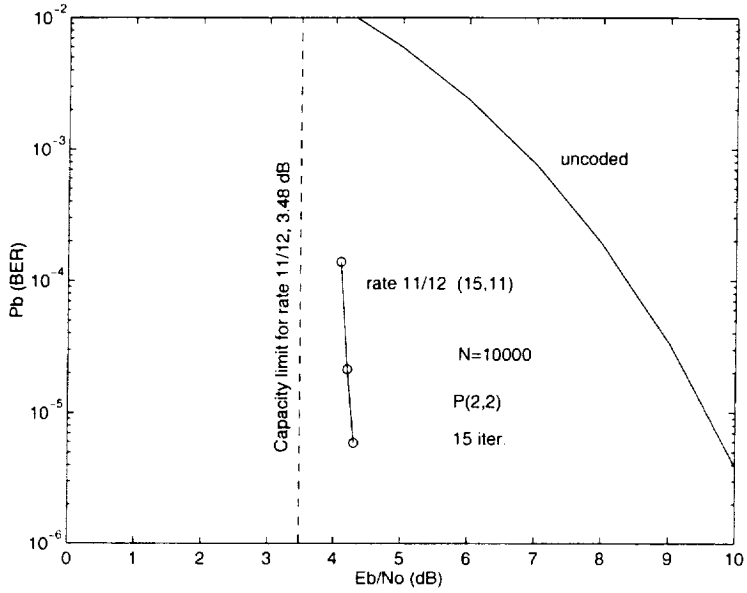

Figure 3.16: Rate 11/12 PCCC BER performance.

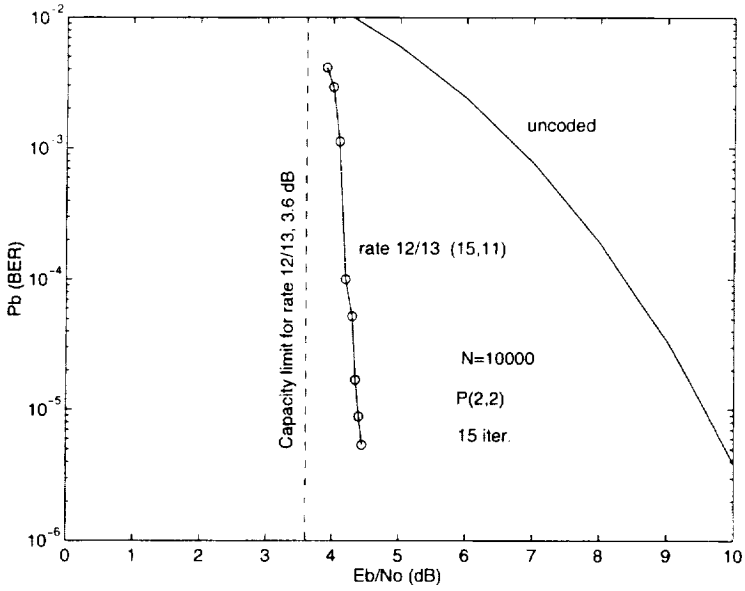

Figure 3.17: Rate 12/13 PCCC BER performance.

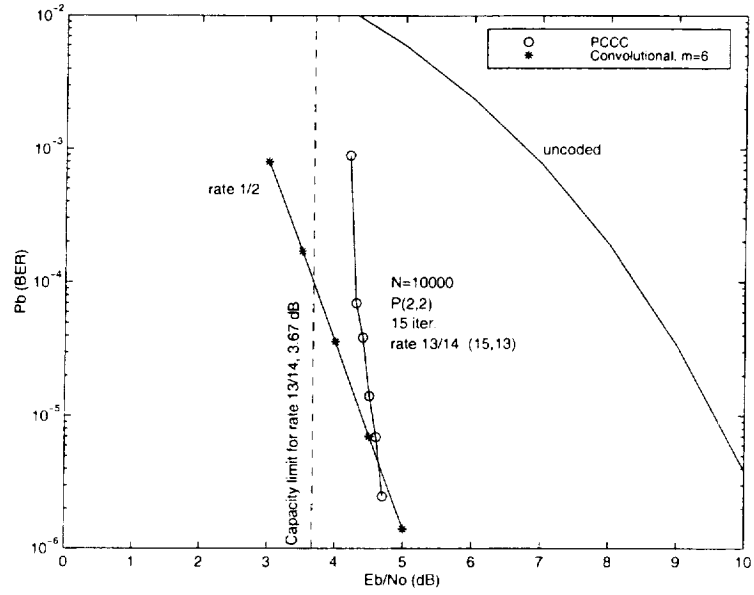

Figure 3.18: Rate 13/14 PCCC BER performance.

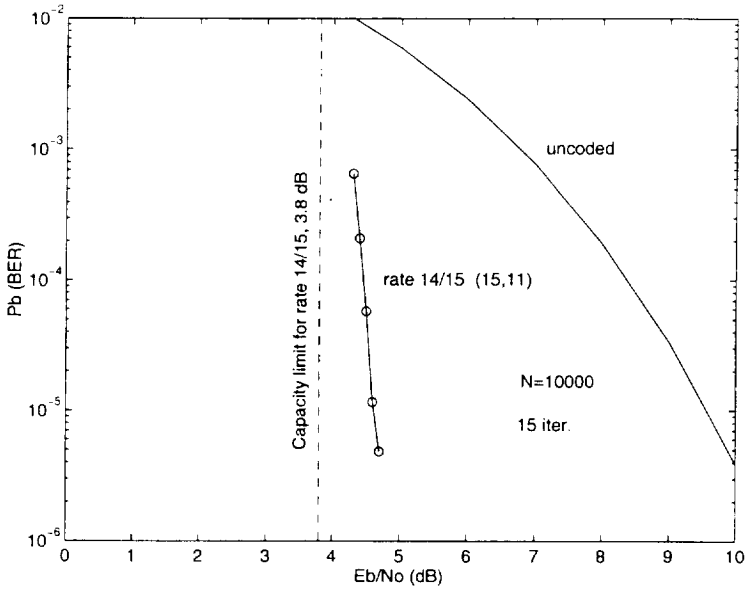

Figure 3.19: Rate 14/15 PCCC BER performance.

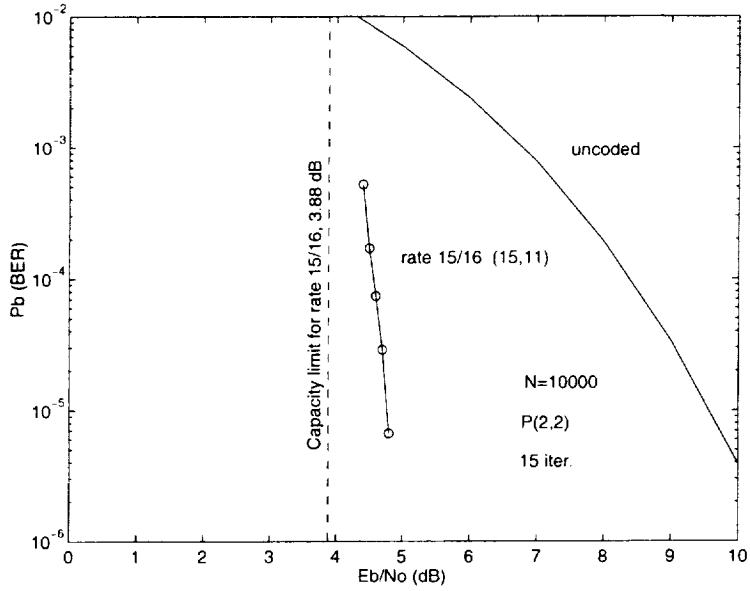

Figure 3.20: Rate 15/16 PCCC BER performance.

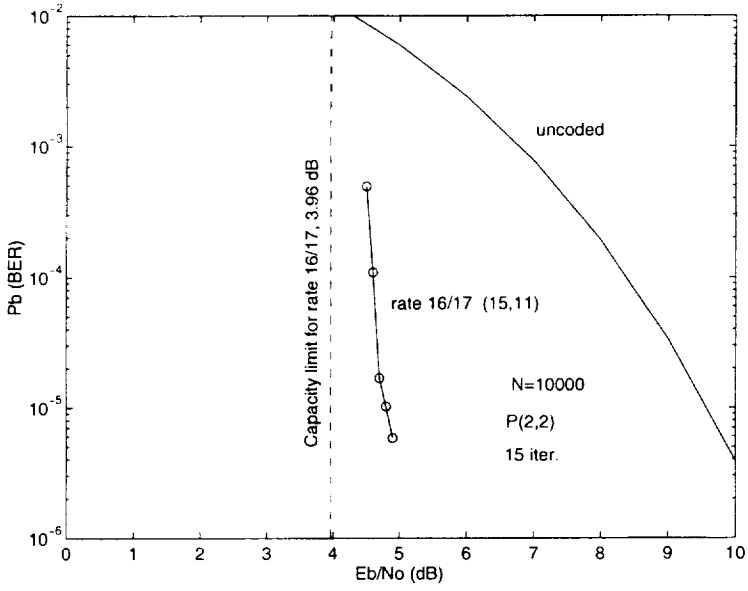

Figure 3.21: Rate 16/17 PCCC BER performance.

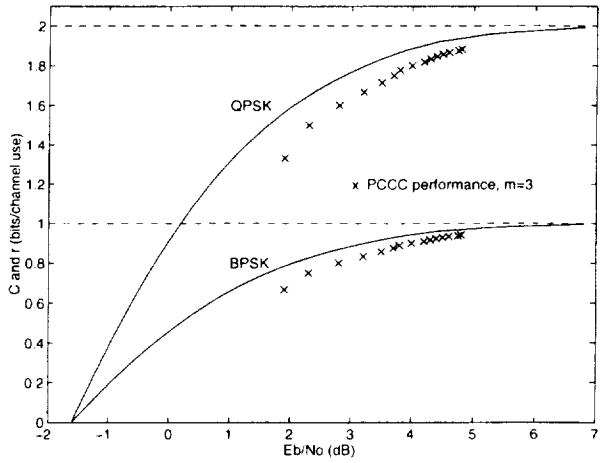

Figure 3.22: Required E_b/N_0 (dB) to achieve 10^{-5} BER for r=2/3 through 16/17 PCCCs ($m = 3$) are compared with capacity for AWGN BPSK/QPSK channel.

3.7.2 $m = 4$ Case

The following, Figs. 3.23 through 3.37, are the high rate PCCCs BER simulations for rates 2/3 through 16/17, respectively, designed by the algorithm given in this chapter with $m = 4$. As in the $m = 3$ case, we added the rate 3/4, 7/8, and 1/2 convolutional codes' BER performance to compare with rate 3/4, 7/8, and 16/17 PCCCs, respectively. When rate 3/4 and 7/8 PCCC are employed, a 3 dB gain is possible at BER of 10^{-5} over the same rate convolutional codes as shown in Fig. 3.24 and 3.28, respectively. In Fig. 3.37 rate 1/2 convolutional code versus rate 16/17 PCCC is given that shows around 10^{-5} and 10^{-6} BER region, the performance of the convolutional code becomes inferior to the PCCC for the same signal power which the later yields around 92% more bandwidth.

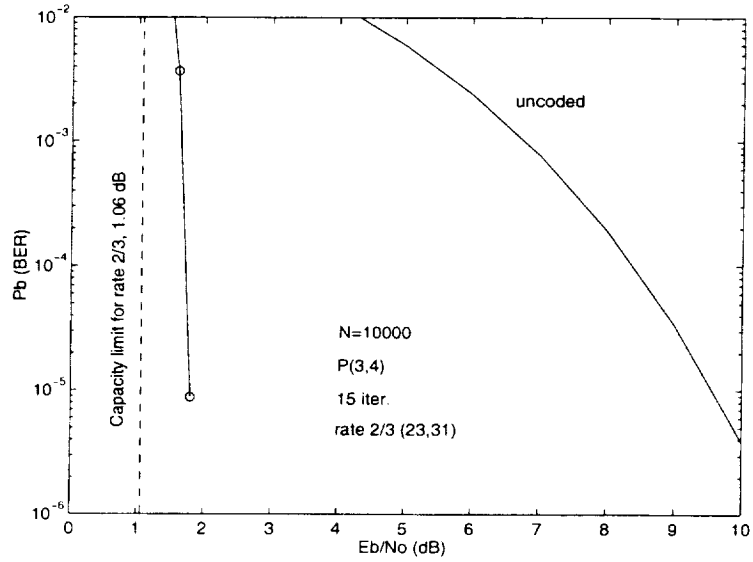

Figure 3.23: Rate 2/3 PCCC BER performance.

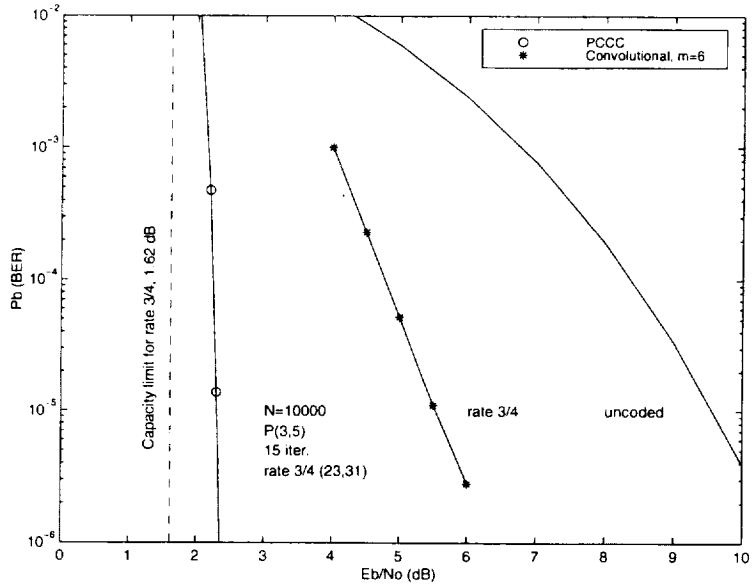

Figure 3.24: Rate 3/4 PCCC BER performance.

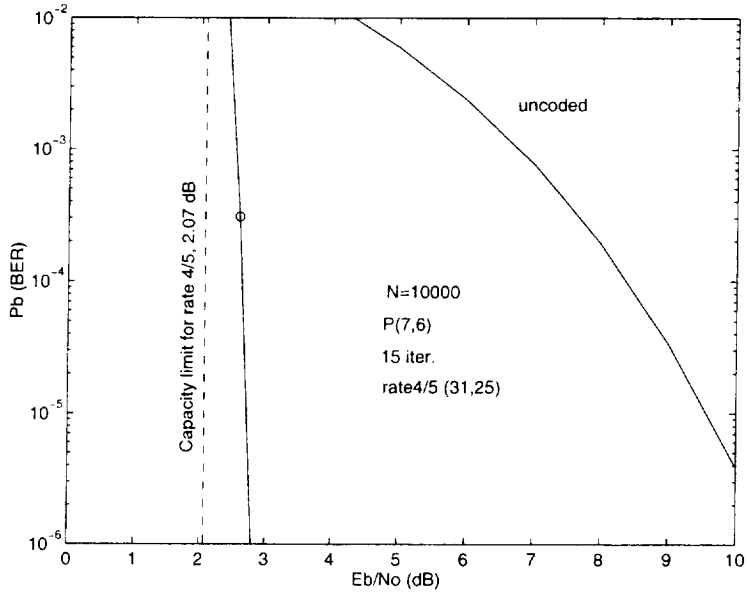

Figure 3.25: Rate 4/5 PCCC BER performance.

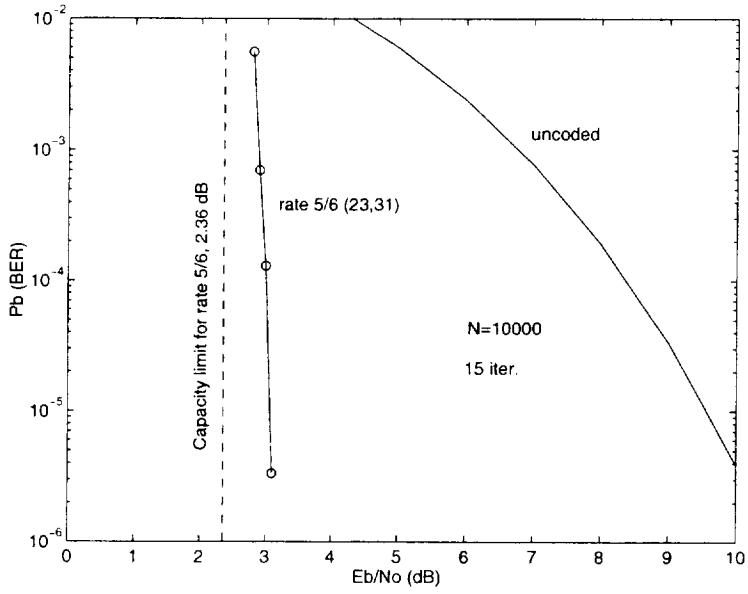

Figure 3.26: Rate 5/6 PCCC BER performance.

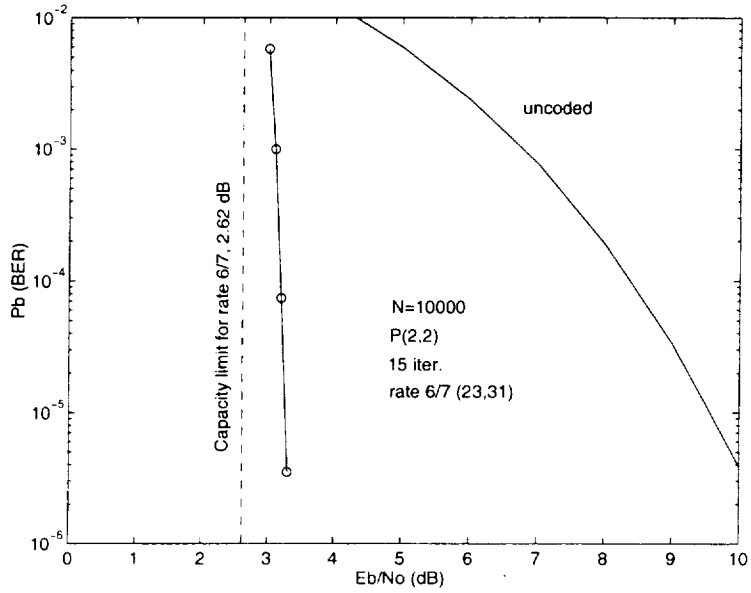

Figure 3.27: Rate 6/7 PCCC BER performance.

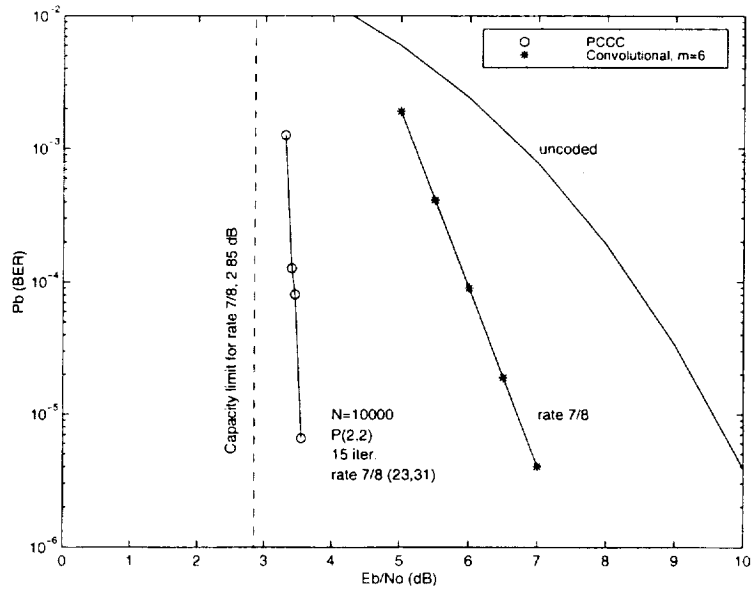

Figure 3.28: Rate 7/8 PCCC BER performance.

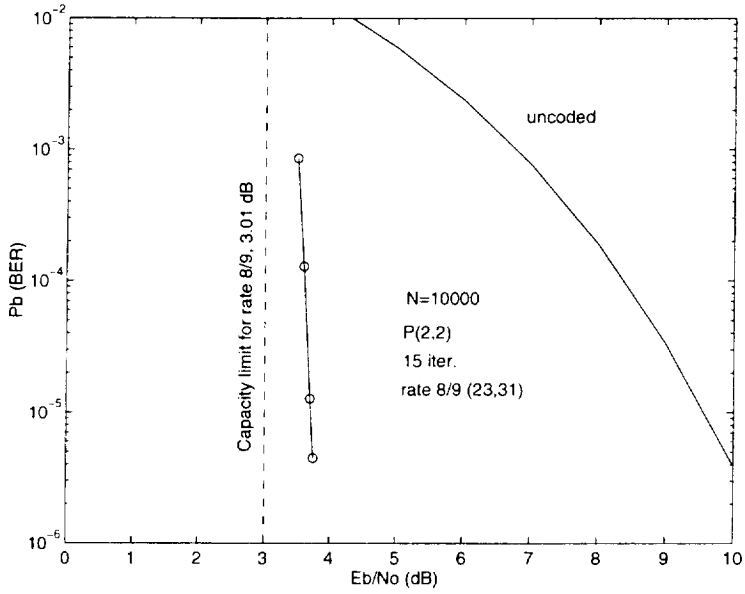

Figure 3.29: Rate 8/9 PCCC BER performance.

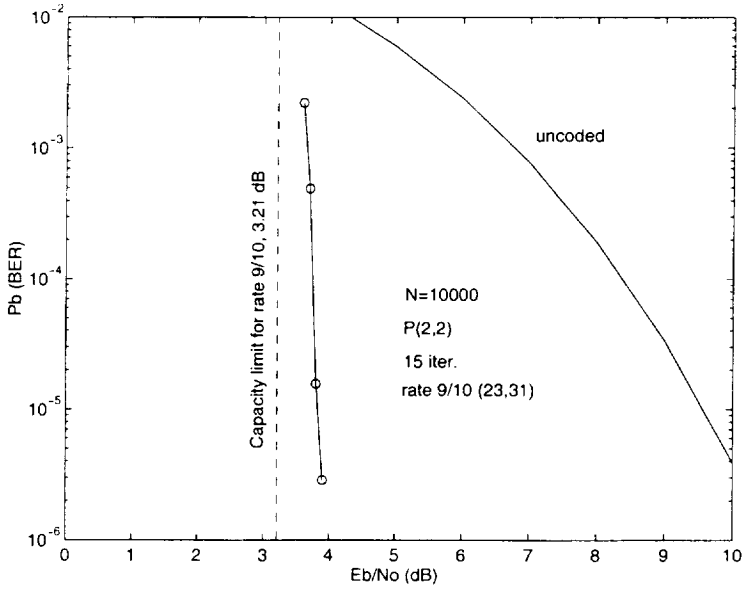

Figure 3.30: Rate 9/10 PCCC BER performance.

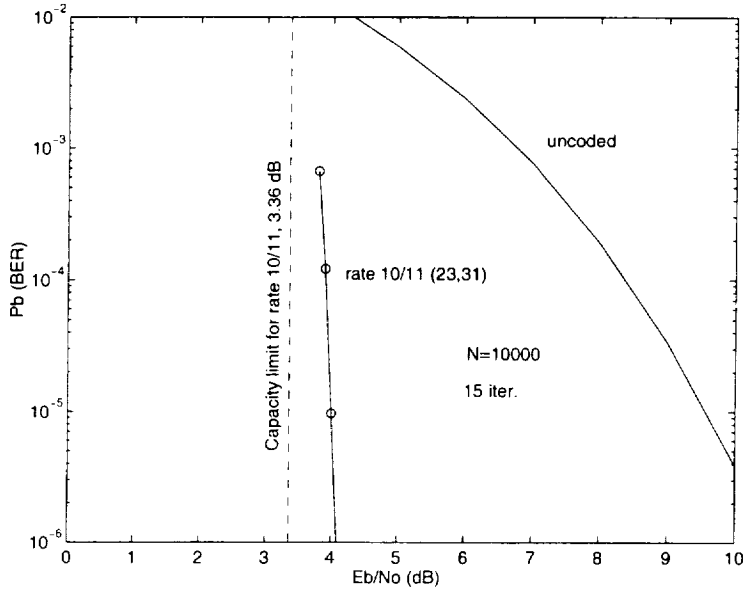

Figure 3.31: Rate 10/11 PCCC BER performance.

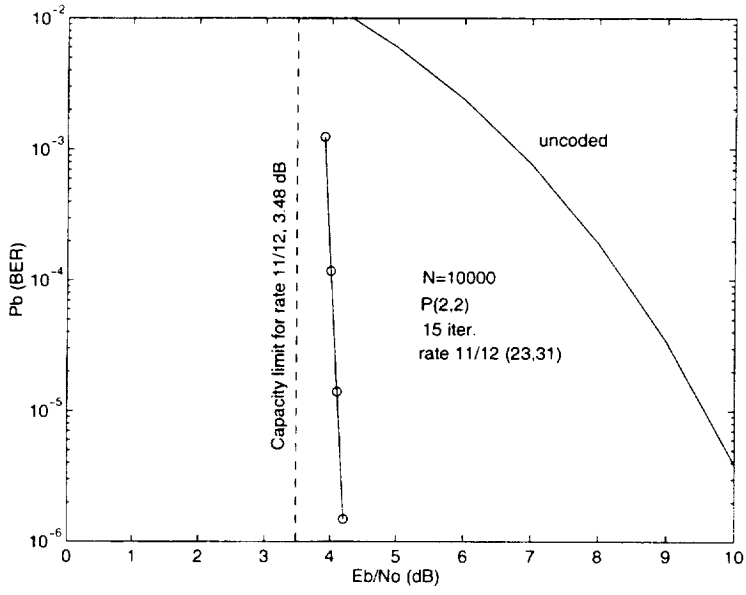

Figure 3.32: Rate 11/12 PCCC BER performance.

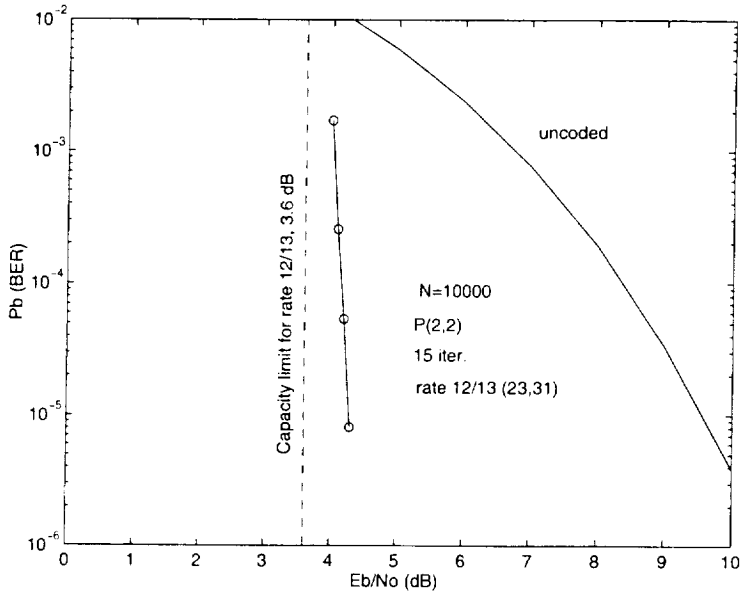

Figure 3.33: Rate 12/13 PCCC BER performance.

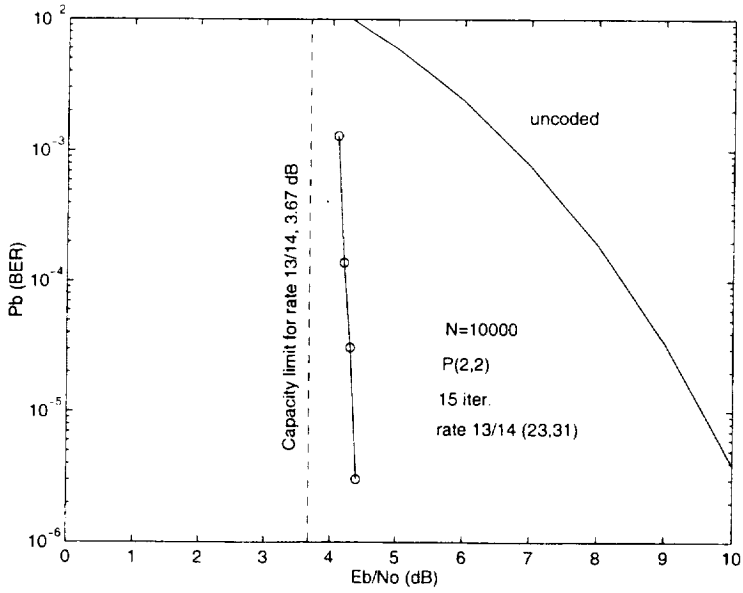

Figure 3.34: Rate 13/14 PCCC BER performance.

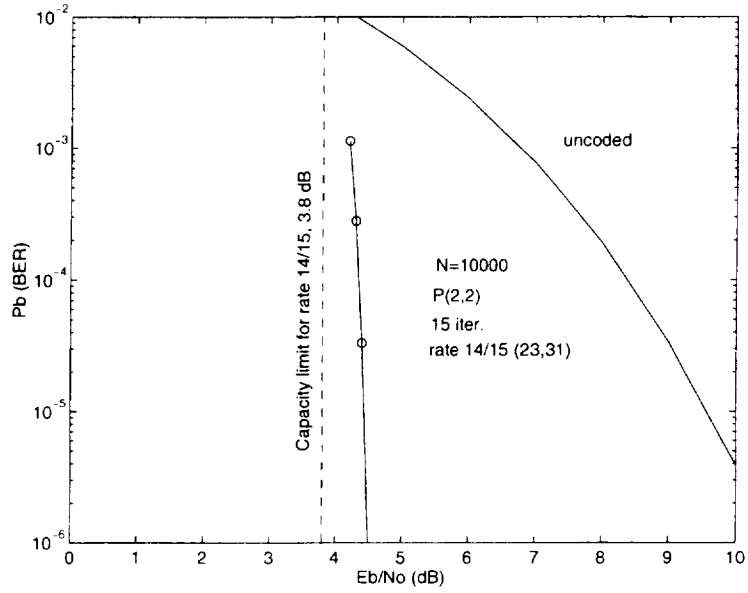

Figure 3.35: Rate 14/15 PCCC BER performance.

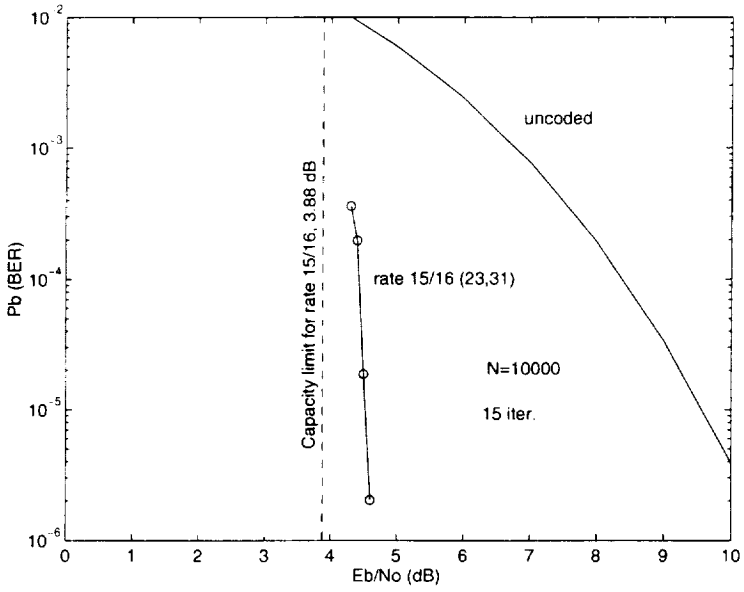

Figure 3.36: Rate 15/16 PCCC BER performance.

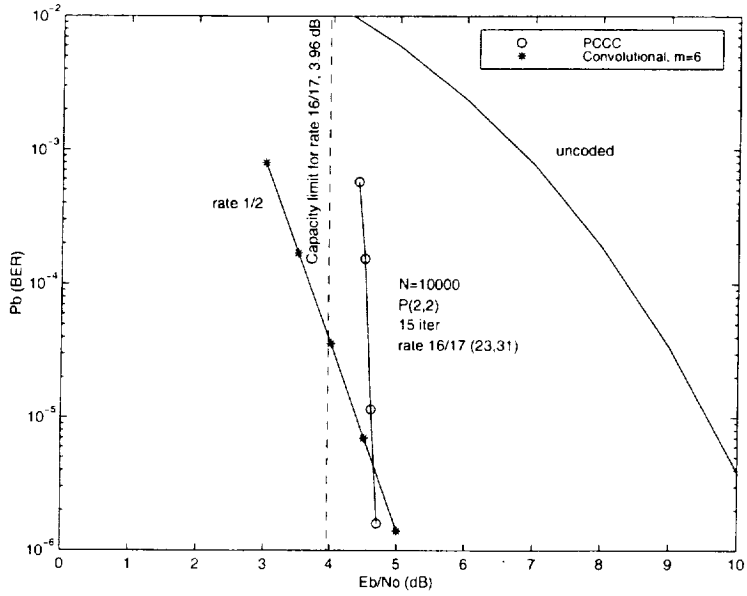

Figure 3.37: Rate 16/17 PCCC BER performance.

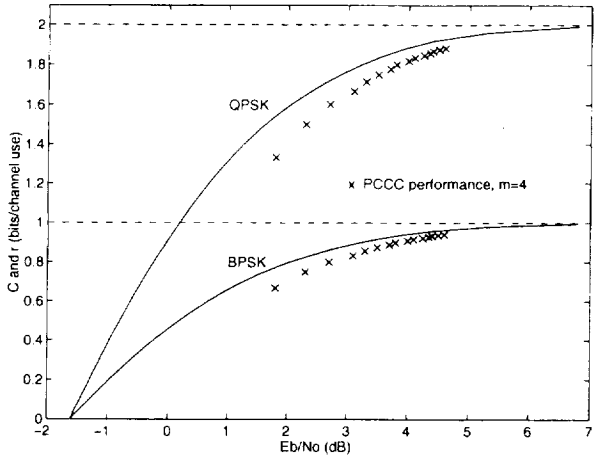

Figure 3.38: Required E_b/N_0 in dB to achieve BER of 10^{-5} for r=2/3 through 16/17 PCCC with $m = 4$ are given to compare with capacity for BPSK/QPSK and AWGN channel.

The required E_b/N_0 values in dB to achieve $P_b = 10^{-5}$ for all rates and $m = 3$ and 4 are given in Table 3.4.

Table 3.4: Required E_b/N_0 values in dB at $P_b = 10^{-5}$ for $m = 3$ and 4 (distance from capacity in parentheses).

Rate	E_b/N_0(dB), $m = 3$	E_b/N_0(dB), $m = 4$
2/3	1.9 (0.84)	1.8 (0.73)
3/4	2.3 (0.70)	2.3 (0.69)
4/5	2.8 (0.74)	2.7 (0.63)
5/6	3.2 (0.79)	3.1 (0.70)
6/7	3.5 (0.87)	3.3 (0.62)
7/8	3.7 (0.80)	3.5 (0.65)
8/9	3.8 (0.81)	3.7 (0.69)
9/10	4 (0.80)	3.8 (0.62)
10/11	4.2 (0.80)	4 (0.64)
11/12	4.3 (0.78)	4.1 (0.65)
12/13	4.4 (0.80)	4.25 (0.70)
13/14	4.5 (0.85)	4.35 (0.65)
14/15	4.6 (0.82)	4.4 (0.61)
15/16	4.75 (0.87)	4.5 (0.63)
16/17	4.8 (0.84)	4.6 (0.65)

3.8 Conclusions

An algorithm for designing near optimum high rate PCCCs via puncturing is given. The algorithm consists of several steps to maximize the special weight-two and -three input codeword weights and to minimize their multiplicities which are known to dominate PCCC performance.

The maximization of the weights and the minimization of the multiplicities are performed over all possible encoder polynomial sets with primitive feedback

polynomials, dominant subsets of the weight-two and -three inputs, a subset of all interleavers, and a number of puncturing schemes as defined in Section 3.2. The schemes are defined by assuming the rates of the constituent encoders are the same after puncturing.

The polynomial sets $(15, 11)$ for $m = 3$ and $(23, 31)$ for $m = 4$ are found to have similar or better performance (BER) over the other sets for rates lower than 5/6. For both $m = 3$ and 4, searches for near-optimal polynomial sets and puncturing schemes which produce maximum codeword weights with minimum multiplicities are pointless for rates higher than 4/5 since all polynomials give similar optimal solutions for all the puncturing schemes.

We found that for the rates such that $Q_m \leq 3$ for $m = 3$, and 4, pseudo-random puncturing is required to achieve a good performance.

Our simulations showed that with the proposed design algorithm for rates $k/(k+1)$ with $2 \leq k \leq 16$, $P_b = 10^{-5}$ is possible within 0.9 dB of capacity for $m = 3$ and within 0.75 dB of capacity for $m = 4$. Thus, we believe our design algorithm, while suboptimum due to a search over limited set of parameters, produced near optimum codes in the 10^{-5} ("cliff") region. As for the "floor" region of the curves, while our searches were nonexhaustive, our simulations have indicated that the floors in each case start somewhere below $P_b = 10^{-7}$. Determining how close $d^{PCCC*}_{2,\min}$ and $d^{PCCC*}_{3,\min}$ (which determine the level of the floor) are to the globally optimum values is a problem of unmanageable complexity. The values reported

here might be improved with alternative puncturing schemes. In this work, we only resorted to pseudo-random puncturing when it was necessary. Although not reported here, we remark that we have used the described technique to design rate 32/33 and 64/65 codes, again achieving $P_b = 10^{-5}$ performance within 1 dB of the capacity limit.

4 IMPLEMENTATION ISSUES FOR HIGH RATE PCCCs

4.1 Introduction

In this chapter, we investigate implementation issues for the SW-Log-MAP algorithm in the design of high rate punctured PCCC decoders in hardware. The original and modified MAP algorithms are computationally complex and have a large decoding delay for long interleaver sizes. By using the suboptimal SW-Log-MAP algorithm, it is possible to design APP decoders that are less complex with short decoding delays. We consider the decoding of rate 3/4, 7/8, 15/16 punctured PCCCs with optimum sliding window sizes and uniform quantization of decoding parameters in order to minimize the BER.

In [40], [36], and in the previous chapter various ways of designing high rate PCCCs were shown, but the results were given for decoding delays on the order of interleaver size (maximum decoding delay) and the MAP or Log-MAP decoding algorithms are applied with "infinite" precision. Benedetto, et al. [30], proposed SW-Log-MAP decoding algorithm to reduce the decoding delays at the expense of a small BER degradation. Pietrobon gave the details of his SW-Log-MAP decoder hardware in [31] and showed the BER results for PCCC decoder for various rates from 1/7 up to 1/2. Robertson, et al. [27], gave results for the quantized Log-MAP decoder. In the latter, the quantized rate 1/2 Log-MAP for 8 iterations with 8-bit uniform quantization of the decoding parameters was inferior by approximately 0.5 dB to rate 1/2 Log-MAP decoder BER performance with "infinite" precision.

We consider the quantized version of the SW-Log-MAP algorithm for the high rate punctured PCCCs given in the previous chapter. Our goal is to find the acceptable decoding delays (sliding window sizes) and quantization levels for rate 3/4, 7/8, and 15/16 PCCCs such that they are feasible and their implementation losses are minimized. The implementation loss is defined as the difference between the E_b/N_0 required to achieve 10^{-5} BER for the quantized PCCC with a short decoding delay and that for infinite precision PCCC with maximum decoding delay.

In the following section, we briefly give the high rate PCCC encoder structure and the characteristics of the PCCCs we considered. Section 4.3 is devoted to decoding delay of APP decoding. Section 4.4 covers quantization of various APP decoder variables, suboptimum implementations and effects of SNR offset on the BER performance. Section 4.5 presents simulation results for various decoding delays and quantization levels for the AWGN channel. Finally, Section 4.6 contains concluding remarks.

4.2 PCCC Encoders and High Rate PCCCs Considered

We applied the design algorithm for high rate PCCCs in the form $k/(k+1)$ given in Chapter 3 to achieve rate 3/4, 7/8, and 15/16 PCCCs for memory size $m = 4$. The following are the characteristics of the PCCCs we considered. Both RCS's started encoding at zero state for each interleaver block and the first RSC

is forced to end encoding at zero state with 4 tail bits. For all the rates of interest, recursive generator polynomial set (23,31) and $N = 10,000$ bit enhanced interleaver are used. For rate 3/4 and 7/8 the puncturing schemes $P(3,5)$ and $P(2,2)$, respectively, are used. Since rate 15/16 is one of the special rates for $m = 4$, we applied the pseudo-random puncturing scheme with $q_0 = 5$, $f_4 = 7$ as given in Chapter 3.

4.3 Decoding Delay

The backward recursion variable $\bar{\beta}_k(s)$ in the APP decoding algorithm can be calculated only after the full codeword is received. Therefore, the decoding delay for the APP algorithm is on the order of the interleaver size, N. It is a well known fact that, reducing the interleaver size to reduce the decoding delay will result in degradation in BER proportional to interleaver size.

Benedetto, et al., [30] and Pietrobon [31] introduced a method to reduce the decoding delay without reducing the interleaver size. In this method, the interleaver is divided into smaller blocks and the decoding algorithm is applied within these smaller blocks. When the decoding delay is D, the size of each block is $2D$. For each block, new initial conditions on $\bar{\alpha}_k(s)$'s and new boundary conditions on $\bar{\beta}_k(s)$'s will be applied. Since there is no information about which state the trellis will be in the beginning and the end of each block, we will assume all $\bar{\alpha}_k(s)$'s and $\bar{\beta}_k(s)$'s are equally probable except for the first and the last blocks. As seen in

Fig. 4.1, the BER degradation with this method is negligible for sufficiently large block sizes.

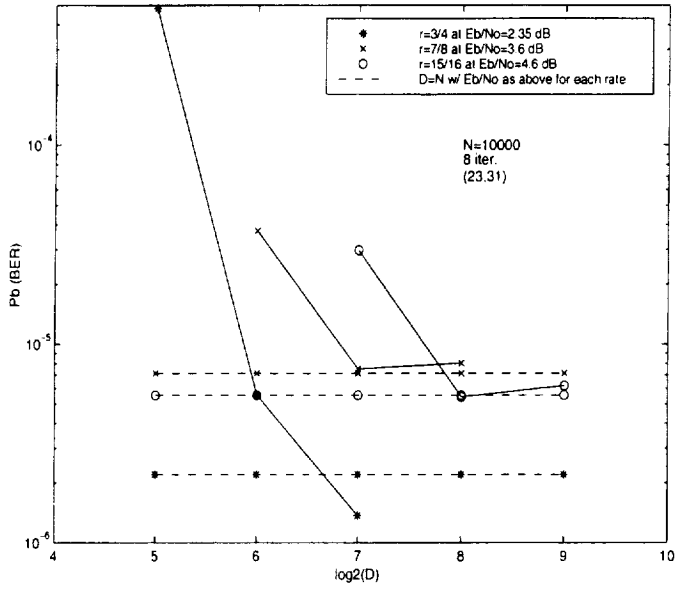

Figure 4.1: Performance of rates r=3/4, 7/8, and 15/16 versus sliding window size at a fixed E_b/N_0.

In Fig. 4.1, we first set E_b/N_0 values for each rate such that for the maximum decoding delay the BER is lower than 10^{-5} with 8 iterations. The horizontal dashed lines show these references for each rate. Then, we applied various decoding delays equal to a power of 2 and show their BER performance in solid lines for each rate. As expected for rates 3/4, 7/8, and 15/16, rate 3/4 required the shortest D to come close to its reference. The reason for this is for a given decoding length D, the rate 3/4 codeword has more saved parity bits than the other rates so it achieves a BER performance closer to its reference. By using the

results in Fig. 4.1, we chose to set D= 64, 128, and 256 for rate 3/4, 7/8, and 15/16, respectively.

4.4 Quantization

Let $y_k = (y_k^s, y_{1k}^p, y_{2k}^p)$ represent the AWGN channel values of the systematic bit and the first and second RSC parity bits at time k after the match filter and the depuncturing circuitry, respectively. We will assume BPSK signalling with perfect timing and carrier recovery. Extension to QPSK signalling is similar. Then any of the three channel bits at time k can be written as:

$$y_k^s = \pm 1 + n_k \text{ with } n_k \sim \eta\left(0, \frac{N_0}{2}\right) \tag{4.1}$$

assuming $E_s = 1$. In order to construct the quantized SW-Log-MAP algorithm, we must quantize the $y_k = (y_k^s, y_{1k}^p, y_{2k}^p), \bar{\alpha}_k(s), \bar{\beta}_k(s), \bar{\gamma}_k(s', s), L^e(u_k)$, and $L(u_k)$ values.

As mentioned earlier, BPSK or QPSK signalling is assumed. Therefore in-phase and quadrature components of the received signal are individually quantized. For a given quantization level Q (bits), there are total 2^Q uniformly quantized values: $2^{Q-1} - 1$ positive and $2^{Q-1} - 1$ negative. The other 2 quantized values are used to represent zero which then has two different binary values.

From our observations, the channel values $y_k = (y_k^s, y_{1k}^p, y_{2k}^p)$ required 6-bit or less quantization levels depending on the signal to noise ratio. For the rest of the parameters, after considering their variances and their equivalent contribution

to calculations of *extrinsic* $L^e(u_k)$ and LAPP, $L(u_k)$, we decided to use the same number of quantization levels. Because of feasibility concerns, we considered three quantization levels for these parameters: Q=4-, 6-, and 8-bit.

For each quantization level, we have searched for the optimum step size, Δ_Q, in order to minimize the BER. Fig. 4.2 shows step size versus BER results for 8-bit quantization for rate 3/4 at $E_b/N_0 = 2.4$ dB. The optimum step sizes for 4-, 6-, and 8-bit quantization levels are listed in Table 4.1.

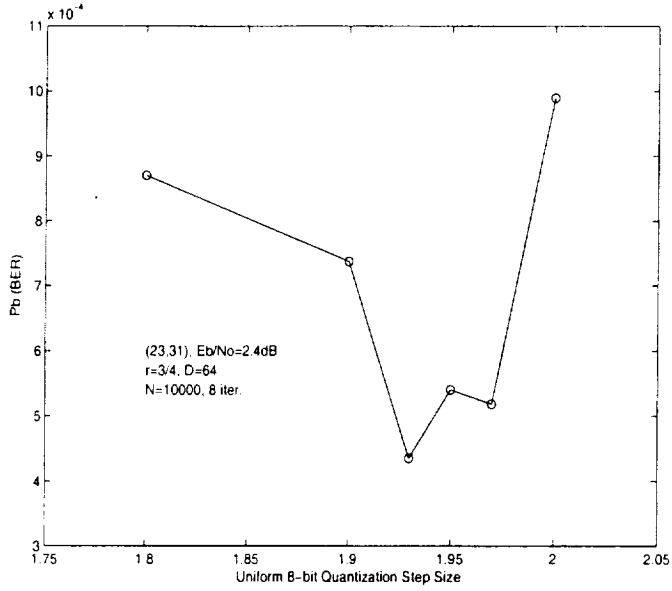

Figure 4.2: Uniform quantization step size versus BER performance for r=3/4, 8-bit quantization with D=64.

The optimum quantization levels found for rate 3/4 are also used for rate 7/8 and 15/16 because of the similar variance in the channel values in the range of

Table 4.1: The optimal step sizes for selected quantization levels to minimize BER.

Q	Δ_Q
4	0.421
6	0.26
8	0.193

E_b/N_0 we operate. This is also justified by the BER simulations of quantized PCCCs in Fig. 4.5.

4.4.1 Suboptimum Implementations

The variables $\bar{\gamma}_k(s', s)$, $\bar{\alpha}_k(s)$, and $\bar{\beta}_k(s)$ may cause a suboptimum implementation due to their quantization. We will describe such a situation for $\bar{\alpha}_k(s)$: a similar scenario is possible for the others. Let us call the maximum and minimum values of $\bar{\alpha}_k(s)$ at time k $\bar{\alpha}_k(s_i)$ and $\bar{\alpha}_k(s_j)$, respectively. If the distance between the two satisfies the following condition,

$$d_{ij} = [\bar{\alpha}_k(s_i) - \bar{\alpha}_k(s_j)] > \frac{2^Q}{2} = 2^{Q-1} \tag{4.2}$$

then the algorithm becomes suboptimum since the distance between $\bar{\alpha}_k(s_i)$ and

$\bar{\alpha}_k(s_j)$ will not be preserved when a constant value is subtracted from each $\bar{\alpha}_k(s)$ to prevent overflow. Fig 4.3. depicts the steps of such a situation.

Our simulations showed that the quantized algorithm becomes suboptimum in the low E_b/N_0 environment only after a large number of iterations. The reason

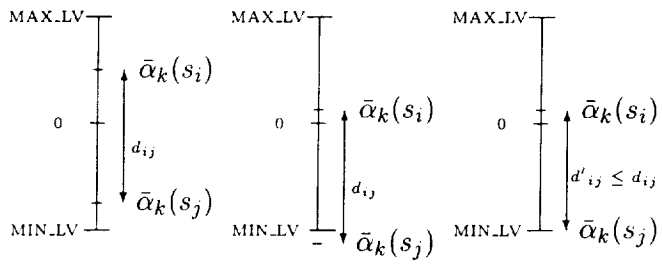

Figure 4.3: The steps of a suboptimum decoding.

is that APP decoders receive noisy channel values and it takes many iterations to find a winning state to satisfy the above condition. For medium E_b/N_0 values, quantized decoding becomes suboptimum after only a few iterations. Finally, in high E_b/N_0 environments, if the algorithm is not suboptimum after the first iteration, it is almost always suboptimum after the second. In all cases we observed, $\bar{\alpha}_k(s)$ or $\bar{\beta}_k(s)$ caused the suboptimality before $\bar{\gamma}_k(s', s)$.

4.4.2 SNR Offset

The APP decoder requires the exact value of a parameter $L_c = 2E_s/N_0$ for optimum BER performance. Overestimating L_c results in the channels values being considered more reliable than they actually are while under estimating L_c reduces the effect of channel values when it should be stronger.

In Fig. 4.4, the effects of L_c offset in dB is given on BER performance for different E_b/N_0 values for rate 7/8 with $Q = 8$. Two important observations can be made from this plot. First, within 2 dB over or underestimation of L_c degrades the BER performance very little. Second, for larger deviations from the true value

of L_c, APP decoders suffer from underestimation much more than overestimation. Similar results on SNR offset sensitivity of APP decoders are reported in [50], [51], and [52].

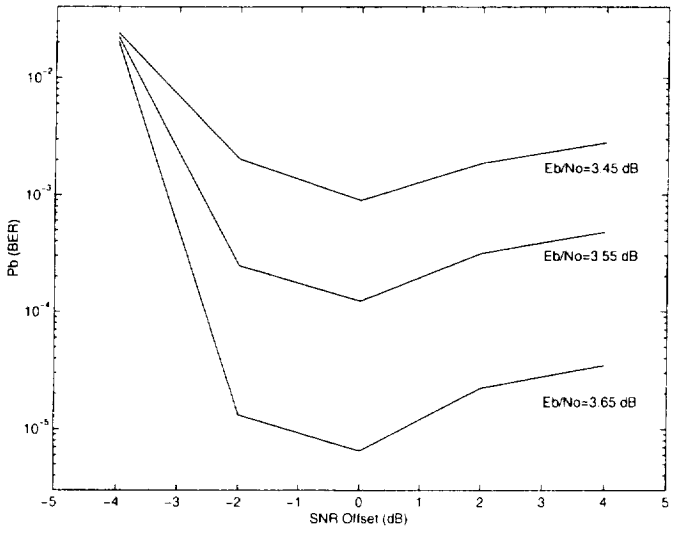

Figure 4.4: BER performance of rate 7/8 PCCC with Q=8 versus SNR offset in dB for different E_b/N_0 values.

4.5 Results

The following simulation results are given for $N = 10,000$-bit interleaver with 8 iterations for all cases. In order to address the different parameters for each simulation, we will use two numbers split by a semicolon in square brackets. The first number will indicate the number of quantization bits; if it is "infinite" precision, we will indicate it with ∞. The second will refer the decoding delay, D. Unless stated otherwise, the simulations for "infinite" precision used the exact

correction term for the max* operation defined in (2.23). For quantized PCCC simulations, correction terms stored in an 8-value lookup table are used.

Fig. 4.5 shows the BER versus E_b/N_0 performance comparisons of $[\infty; 5,000]$ PCCCs to $[Q; D]$ PCCCs decoded by SW-Log-MAP algorithm for rates 3/4, 7/8, and 15/16. The curves with solid lines are the performance of $[\infty; 5,000]$ PCCCs and dashed curves are for $[Q; D]$ PCCCs with Q = 4-, 6-, and 8-bit for all rates and $D = 64, 128$, and 256 for rate 3/4, 7/8, and 15/16, respectively.

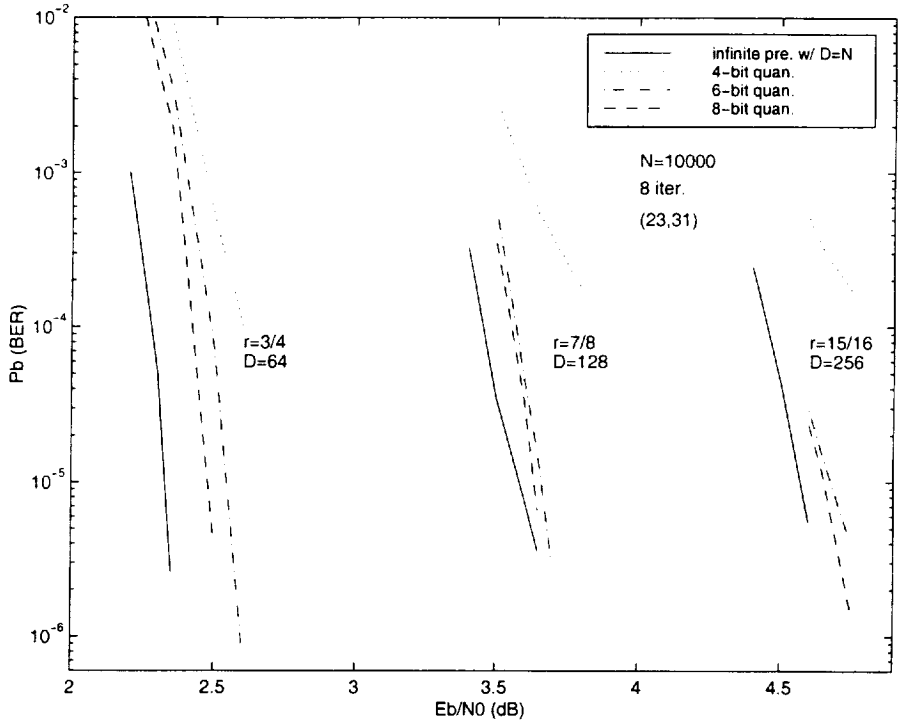

Figure 4.5: Performance comparison of rates r=3/4, 7/8, and 15/16 for different quantization levels and D.

One important result from these curves is that for each rate of interest, it is possible to implement a practical PCCC decoded by SW-Log-MAP with imple-

mentation loss within 0.2 dB at 10^{-5} BER. More specifically, the rate 3/4 [8; 64] PCCC decoded by SW-Log-MAP algorithm shows within 0.2 dB implementation loss. The implementation loss for rate 7/8 [8; 128] PCCC is less than 0.15 dB. Finally, for rate 15/16 [8; 256] PCCC, the implementation loss is within 0.2 dB. With more sacrifice in the implementation loss, about 0.1 dB more for all rates, 6-bit quantization of SW-Log-MAP parameters can be used with the same decoding delays.

4.6 Conclusions

In this chapter, we outlined a process to implement a practical punctured high rate PCCCs of rate 3/4, 7/8, and 15/16 decoded by SW-Log-MAP algorithm. Our goal is to select the minimum decoding delay and uniform quantization level to minimize the implementation loss of such feasible PCCCs.

We first determined what decoding delay should be used for each rate by comparing the performance of PCCCs with maximum delay to PCCCs with various decoding delays at a fixed E_b/N_0 that yields BER between 10^{-5} and 10^{-6}. Then we simulated the punctured PCCC BER performances of each rate with the selected decoding delays and 4-, 6-, and 8-bit quantization levels. Finally we compared these simulations with $[\infty; 5,000]$ PCCC simulations for each rate to observe the implementation loss. We found that the implementation loss for rate 3/4, 7/8, and 15/16 PCCCs with Q=8-bit and 8-value correction term is less than 0.2 dB

for selected D values. If 6-bit quantization is used with the same decoding delays, the implementation loss is within 0.3 dB for all the rates.

These rate 3/4, 7/8, and 15/16 PCCC implementations are suboptimum since the distance between the maximum and minimum values for certain decoding parameters cannot be preserved. Also, with these implementations, SNR offset of within ± 2 dB result in small BER degradation. For larger offset values, overestimating the channel quality shows much less BER degradation than underestimating it.

5 PUNCTURED SCCCs FOR BPSK/QPSK CHANNELS

5.1 Introduction

In Chapter 3, one of the main reasons for designing binary high rate punctured PCCCs was the simplicity and the robustness of the receiver for coded binary signals, specifically, the carrier and timing recovery loops and decoder. It was shown in Chapter 3 that it is possible to design high rate punctured ($r = k/(k+1)$, $k = 2, \ldots, 16$) PCCCs that operate near the Shannon limit. Despite their excellent bit error rate (BER) performance in the "cliff region" (low SNR), PCCCs reach an error floor at high SNRs. A careful interleaver design may lower the error floor, but applications requiring a BER of 10^{-10} or lower is possible only with extremely large interleaver sizes. A large interleaver size, in return, means large decoding delays which is not suitable for high speed applications.

Shortly after the discovery of PCCCs, SCCCs were proposed as an alternative to PCCCs since SCCCs generally have lower error floors [53]. Benedetto, et al. [53] made ensemble performance comparisons between PCCCs and SCCCs with the same constituent encoder memory sizes for rates 1/3 and 1/4 with various interleaver sizes. The comparison showed that the BER performance of SCCCs are inferior to PCCCs for low SNRs (the "cliff region"), but are superior in the floor region. Several SCCC schemes are considered in the literature [54], [55], and [56]. These schemes consider rate 1/2 or lower for AWGN channels. However, in

[57]-[63] higher rate SCCCs are considered when the precoded PR4 channel is the inner code.

In this study, we consider the design of high rate $r = 3/4$, 7/8, and 15/16 binary SCCCs with various constituent encoder memory sizes which achieve (theoretical) efficiencies of 1.33 through 1.88 bps/Hz on QPSK channels. For each rate, we design 8-state outer, 8-state inner (8/8), 8-state outer, 2-state inner (8/2), and 4-state outer, 2-state inner (4/2) SCCCs. We focus on low-memory designs to reduce the decoder complexity for high data rate applications. The design algorithm includes finding the best puncturing patterns and interleavers that maximize the minimum codeword weight for inputs causing the most likely error patterns. For the reasons we discuss in the following section, both inner and outer encoders will be recursive systematic convolutional (RSC) codes in our high rate SCCC designs. In all designs, the feedback polynomial of the RSC codes will be primitive. When a 2-state inner encoder is employed, it is always a rate 1 differential encoder (which is recursive).

For the 4/2 SSSC design, the search for the best polynomial set was not necessary since there is only one 4-state RSC outer encoder with maximum d_{free} and a primitive feedback polynomial. For 8/8 and 8/2 SCCC designs, we did not notice a heavy dependence on the generating polynomial set for the rates of interest. Hence, we did not include a polynomial set search for our high rate SCCC design algorithm.

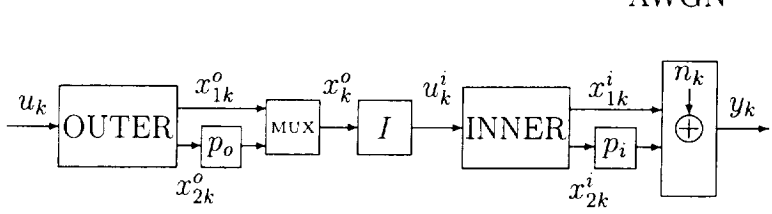

Figure 5.1: SCCC encoder constructed by a rate 1/2 outer and inner encoders with AWGN channel.

In the next section, we review some of the important characteristics of SCCCs which will play an important role in understanding the design algorithm. Section 5.3 explains the computer search algorithm that includes finding the optimum puncture patterns and interleaver designs. Section 5.5 is devoted to the search results and the computer simulations for the rates of interest on AWGN channel. In Section 5.6, BER performance and error floor comparisons between the SCCCs designed here and the PCCCs designed in Chapter 3 are given. Finally, concluding remarks are given in Section 5.7.

5.2 Characteristics of SCCCs

In Fig. 1.8 an SCCC is depicted with a rate 1/2 outer encoder and a rate 1 inner encoder separated by a MUX and a bit interleaver, I. Only the parity sequence of the outer encoder is systematically punctured to achieve higher rates, since the puncturing of the systematic bits results in poor BER performance for RSC encoders. This figure represents the 8/2 and 4/2 SCCCs we consider since the inner encoder for these cases is a two-state, rate $r_i = 1$ differential encoder.

Hence, in order to design a rate $k/(k+1)$ SCCC in this case, the outer encoder should be a rate $r_o = k/(k+1)$ RSC encoder. We will use the puncturing notation $P(p_o)$ for this case, where $1 \leq p_o \leq k$ represents the saved parity bit locations of the outer parity bit sequence in every k-bit block. Note that this also suggests the input block size, L, should be a multiple of k to have an integer interleaver size, N, where $N = L/r_o = L \cdot (k+1)/k$. In Table 5.1, the input block size and the interleaver sizes are given for the rates of interest.[15]

In Fig. 5.1, another SCCC structure we consider is depicted where, in contrast to the previous structure, the rate of the inner RSC encoder is $r_i = 1/2$ (before puncturing). The inner encoder's parity sequence is punctured to achieve higher inner code rates. This SCCC structure will be used in the design of 8/8 SCCCs. A rate $k/(k+1)$ 8/8 SCCC is achieved by employing a rate $r_o = (2k+1)/(2k+2)$ outer RSC code and a rate $2k/(2k+1)$ inner RSC code. The puncturing notation for this case is $P(p_o, p_i)$, where $1 \leq p_o \leq (2k+1)$ and $1 \leq p_i \leq 2k$. In $P(p_o, p_i)$, p_o represents the saved parity bit locations of the outer parity bit sequence in every $(2k+1)$-bit block, and p_i is likewise for the inner RSC encoder in every $2k$-bit block. Again, in order to have an integer interleaver size, the input block size should be a multiple of $(2k+1)$ as in Table 5.1.

Recall from [42] that, because of the presence of the constituent RSC encoders and the interleaver, the BER performance of a PCCC is dominated by codewords

[15]To compare the SCCC and PCCC BER performances later, the input block sizes for SCCCs are chosen to be the closest integer to 10,000, the input block size (and interleaver size) for the PCCCs designed in Chapter 3.

Table 5.1: The interleaver and input block sizes and the rates of the outer and inner encoders for the rates of interest.

States	r	r_o	r_i	L	N
8/8	3/4	7/8	6/7	10003	11432
8/8	7/8	15/16	14/15	10005	10672
8/8	15/16	31/32	30/31	10013	10336
8/2	3/4	3/4	1	9999	13332
8/2	7/8	7/8	1	10003	11432
8/2	15/16	15/16	1	10005	10672
4/2	3/4	3/4	1	9999	13332
4/2	7/8	7/8	1	10003	11432
4/2	15/16	15/16	1	10005	10672

produced by certain weight-two and -three inputs. From this and the fact that PCCCs are linear codes, the BER performance of a PCCC for medium to high SNRs can be approximated by (3.1) Because both the outer and the inner encoders will be RSCs, only certain inputs of weight $w \geq 2$ that are divisible by the feedback polynomial of the encoders will generate low weight codewords. For reasons similar to the PCCC case, the presence of the interleaver between the outer and inner encoder will prevent the certain outer encoder inputs with $w \geq 4$ to dominate the BER performance of SCCCs for large interleaver sizes (even when the parity bits do not add any weight to the outer codeword). Therefore, the performance of SCCCs will be also dominated by certain weight-two and -three inputs. Further, as for PCCCs, SCCCs are linear codes which allows us to use (3.1) to estimate high-SNR BER performance for SCCCs, by replacing $d_{w,min}^{PCCC}$ with $d_{w,min}^{SCCC}$ to denote the minimum SCCC codeword weight caused by weight-w inputs. In summary,

we will use the approximation (3.1) with $w = 2, 3$ to estimate the error floor for our high rate SCCC designs.

We will review the important results given in [53] to shed light on the type of the convolutional codes we should employ to lower the error floor of the high rate SCCCs we design. In order to have an "interleaver gain," the inner encoder should be an RSC code. Lowering the error floor is possible by choosing an inner encoder with a primitive feedback polynomial [42]. In [54] it is shown that when the inner encoder is simply a differential encoder, $\frac{1}{(1+D)}$, the overall SCCC still has good BER performance. Note that when the inner encoder is a differential encoder, the odd-weight inputs will generate large codeword weights since they are not divisible by $(1 + D)$ [53]. Therefore, choosing an RSC inner encoder $\frac{1}{(1+D)}$ will mean that we need only worry about even-weight inputs, and most likely weight-two inputs. However, any even-weight inputs with two ones close to each other will cause a low weight SCCC codeword. Therefore, we consider RSC inner encoders with a primitive feedback polynomial as well. This is done only for 8/8 SCCC design.

Another important result given in [53] was the criterion for choosing the outer encoder. It is suggested in [53] to use a non-recursive convolutional (NRC) encoder as an outer encoder. Although theoretically this is true, we found employing an RSC encoder as an outer encoder results in better BER performance when higher

rate outer encoders are needed via puncturing. Hence, in all SCCC designs, we employed RSC outer encoders with a primitive feedback polynomial.

5.3 The Design Algorithm

In this section, we discuss an algorithm for designing high rate SCCCs. In the following subsection, we introduce the design parameters and, in the subsequent section, we discuss the algorithms used to optimize these parameters.

5.3.1 Design Parameters

In the design of high rate PCCCs, the design parameters were the generator polynomials for the constituent RSC encoders, the interleaver, and the puncturer. For SCCCs, it is only natural that the same parameters play an important role in the performance. However, as we pointed out in Section 5.1, in our search for the high rate SCCCs, we either have only one choice of generator polynomial set or the varying the generator polynomials for the outer and the inner encoders do not greatly affect performance. Hence, the primary design parameters for SCCCs are the interleaver I and the puncturer, $P(p_o)$ or $P(p_o, p_i)$. When the inner encoder is a differential encoder, the design algorithm is different than when it is an RSC encoder with a primitive feedback polynomial. Therefore, we investigate the design of the 8/8 SCCC and 8/2-4/2 SCCCs in different subsections.

5.3.1.1 8/8 SCCC

Since both the inner and outer encoders are RSC codes with primitive feedback polynomials, weight-two and weight-three inputs dominate the BER performance of the SCCC. Hence, the design parameters should be optimized to maximize $d^{SCCC}_{2,\min}$ and $d^{SCCC}_{3,\min}$ and to minimize n_2 and n_3. For the interleaver sizes of interest in this study, it is computationally impossible to vary the interleaver, the puncturer, and the weight-two and weight-three inputs to find the absolute maximum for $d^{SCCC}_{2,\min}$ and $d^{SCCC}_{3,\min}$ and the minimum for n_2 and n_3. Instead, we find near-optimum values for $d^{SCCC}_{2,\min}$, $d^{SCCC}_{3,\min}$, n_2, and n_3 by varying the design parameters in a systematic way. We will denote these near-optimum values for $d^{SCCC}_{2,\min}$ and $d^{SCCC}_{3,\min}$ by $d^{SCCC*}_{2,\min}$ and $d^{SCCC*}_{3,\min}$, respectively. Then,

$$d^{SCCC*}_{w,\min} = \max_{R^w_t(g^o_1,g^o_2)} \max_{S^w_t(g^i_1,g^i_2)} \max_{I} \max_{P(p_o,p_i)} d^{SCCC}_{w,\min}, \quad w = 2,3 \tag{5.1}$$

where $R^w_t(g^o_1, g^o_2)$ is generated from $S^w_t(g^o_1, g^o_2)$ for w =2 and 3, and $S^w_t(g_1, g_2)$ is defined in (3.3)

The generator polynomial sets for the outer and the inner RSC encoders are $(g^o_1, g^o_2) = (13, 17)$ and $(g^i_1, g^i_2) = (15, 11)$, respectively, with $m = 3$ for both. Since the generator polynomial sets are not a parameter of the design algorithm, we write $R^w_t(o) = R^w_t(g^o_1, g^o_2)$ and $R^w_t(i) = R^w_t(g^i_1, g^i_2)$. Note that the design algorithm in (5.1) is an abstraction. The maximization process over $R^w_t(o)$, $R^w_t(i)$ and I is a joint implementation by using these sets to improve a pseudo-randomly generated interleaver I using the process explained in Subsection 5.4.2. The goal of the

maximization over $P(p_o, p_i)$ is to find the largest possible codeword weight for the enhanced interleaver by applying all the possible puncturing schemes. The number of all possible puncturing schemes is a function of the rate of the SCCC and is equal to $(2k+1)\cdot(2k)$ for a rate $k/(k+1)$ SCCC. When the maximization over $P(p_o, p_i)$ results in more than one puncturing scheme with the same $d_{2,\min}^{SCCC}$ and $d_{3,\min}^{SCCC}$, it selects the puncturing scheme with the lowest n_2 and n_3 which will be denoted by n_2^* and n_3^*, respectively. After discussing the outline of the design algorithm for 8/2 and 4/2 SCCCs in the next subsection, we describe the further details of the design algorithm in Subsection 5.4.

5.3.1.2 8/2 and 4/2 SCCCs

In this case, a $\frac{1}{(1+D)}$ differential encoder serves as the inner encoder. Therefore, inner encoder inputs with odd weight will generate SCCC codewords with large weight. Hence, inner encoder inputs with even weight will dominate the BER performance of these SCCCs. We point out that weight-three inputs to the outer RSC can cause a low weight SCCC codeword when the parity sequence of the outer encoder is heavily punctured and there is no weight contribution from the parity sequence to the outer codeword. This happens when the three ones of any of these weight-three inputs are interleaved to near the end of the interleaver block. As mentioned earlier, due to the presence of the interleaver, the possibility of generating the minimum weight SCCC codeword by the inputs with weights larger than two is less likely. Despite this fact, we still assume the BER performance

will be dominated by weight-two and -three inputs. Thus, the design goal will be to maximize $d^{SCCC}_{2,\min}$ and $d^{SCCC}_{3,\min}$ and to minimize n_2 and n_3.

As in the previous case, for the interleaver sizes of interest, it is computationally impossible to vary the interleaver, the puncturer, and the weight-two and -three inputs to find the absolute maximum for $d^{SCCC}_{2,\min}$ and $d^{SCCC}_{3,\min}$ and the minimum for n_2 and n_3. Instead, we find near-optimum values for $d^{SCCC}_{2,\min}$ and $d^{SCCC}_{2,\min}$, and n_2 and n_3, by varying the design parameters in a systematic way. We will denote these near-optimum values for $d^{SCCC}_{2,\min}$ and $d^{SCCC}_{3,\min}$ by $d^{SCCC*}_{2,\min}$ and $d^{SCCC*}_{3,\min}$ respectively. Then,

$$d^{SCCC*}_{w,\min} = \max_{R^w_t(g^o_1,g^o_2)} \max_{I} \max_{P(p_o)} d^{SCCC}_{w,\min}, \quad w = 2, 3. \tag{5.2}$$

For the 8/2 SCCC, the generator polynomial set for the outer RSC encoder is $(g^o_1, g^o_2) = (13, 17)$ (where $m = 3$), and it is $\frac{1}{(1+D)}$ for the inner encoder (where $m = 1$). For the 4/2 SCCC, the generator polynomial set for the outer RSC encoder is $(g^o_1, g^o_2) = (7, 5)$ (where $m = 2$), and it is again $\frac{1}{(1+D)}$ for the inner encoder. Since the generator polynomial sets are not a parameter of the design algorithm for these cases, we write $R^w_t(o) = R^w_t(g^o_1, g^o_2)$. The maximization in (5.2) over $R_t(o)$ and I is performed jointly by using the set to design an interleaver I as discussed in Subsection 5.4.2.1. The maximization over $P(p_o)$ seeks for the largest minimum codeword weight for the designed interleaver by applying all the possible puncturing schemes. For a rate $k/(k+1)$ SCCC, the number of possible puncturing schemes is k. Again, when more than one puncturing scheme results

in the same $d_{2,min}^{SCCC}$ and $d_{3,min}^{SCCC}$, it selects the puncturing scheme with the minimum n_2 and n_3, which will be denoted by n_2^* and n_3^*, respectively.

5.4 Algorithm Details

The design algorithms abstracted in (5.1) and (5.2) consist of two parts. The first part is the production of the necessary weight-two and -three input sets and enhancing or designing an interleaver with the help of these sets. The second part is finding the best puncturing scheme to maximize the minimum SCCC codeword weight and minimize its multiplicity. Hence, we first explain how to produce the sets in (5.1) and (5.2), then discuss the interleaver design algorithm for 8/2 and 4/2 SCCCs in Subsection 5.4.2.1. The interleaver enhancement algorithm for 8/8 SCCC is very similar to the PCCCs given in Subsection 5.4.2. Finally, we discuss the maximization over all puncturing schemes and identify the cases called "special rates" in Chapter 3 which require pseudo-random puncturing.

5.4.1 The Sets $R_t^2(o)$ and $R_t^3(o)$

The sets $R_t^2(o)$ and $R_t^3(o)$ are derived from the sets $S_t^2(g_1^o, g_2^o)$ and $S_t^3(g_1^o, g_2^o)$. In the following, we only discuss weight-two inputs as similar comments will hold for weight-three inputs. In both Figs. 1.8 and 5.1, the outer codeword is sent to the interleaver. Let us assume the worst case where $r_o = k/(k+1)$ is high, so when a weight-two input enters to the outer RSC encoder, all of the saved parity

bits are zero. Then, the outer codeword weight is two, but two separations of the two one bits are possible, depending on their locations at the input.

For example, for the outer RSC encoder with a primitive feedback polynomial with rate $r_o = 3/4$ and $m = 3$, one of the weight-two inputs is $D^j(1 + D^7)$ where j is the shift from the time zero. Let us assume $j = 0$, so that P(3) gives all zero parity bits when the weight-two input $(1 + D^7)$ is applied. Then the interleaver will receive the weight-two input $(1 + D^9)$ as a result of the two inserted parity bits. Now let us assume $j = 2$ and the same puncturing scheme still produces zero parity bit weight. When the input $D^2(1 + D^7)$ is applied, the interleaver will receive the weight-two input $D^2(1 + D^{10})$ since in this case three parity bits have been inserted.

It is these weight-two inputs that are stored in the set $R_t^2(o)$ along with the analogous ones for $w = 3$. As a result of this, when the outer encoder receives a weight-w input in the set $S_t^w(o)$, the inner encoder will see the encoded, interleaved version of a weight-w input in the set $R_t^w(o)$. Hence, in the interleaver enhancement or design process, $R_t^w(o)$ should be used. In Chapter 3, the sets $S_t^w(g_1, g_2)$ with $t = 60$ were employed for the interleaver enhancement process (where the size of the sets increases with t). As we will see in the results section, Section 5.5, the t values for the sets $R_t^w(o)$ will be smaller since for each entry in $S_t^w(o)$, the set $R_t^w(o)$ may have two to four entries which makes the sizes of $S_t^w(g_1, g_2)$ and $R_t^w(o)$ almost equal.

5.4.2 The Interleaver Enhancement Algorithm

This design algorithm is very similar to the algorithm given in Subsection 3.3.2.3 in the design of high rate PCCCs. The difference is that, for SCCC design we have weight-two and weight-three input sets for outer and inner encoder individually, whereas for PCCCs, we needed only one weight-two and weight-three input sets since the RSC encoders were identical. We note that, as seen in Fig. 3.2, for each $k = 1, 2, \ldots, N$, the interleaving process is applied by the following expression: $u_k^i = x_{I(k)}^o$ where x_k^o is the input and u_k^i is the output of the interleaver (see Figs. 1.8 and 5.1).

We first generated a pseudo-random interleaver for each rate since each rate requires an interleaver with different size as a result of setting the input block size to an integer closest to 10,000. Then we enhance it for the weight-two and weight-three inputs by applying the design rules given below.

a. Weight-Two Input Rules:

We check whether the interleaved weight-two input in $S_t^2(o)$ forms an entry in $S_t^2(i)$. In other words, suppose $x^o(D) = (D^y + D^z)$ where $y < z$; then if $x_I^o(D) = (D^{I(y)} + D^{I(z)}) \in R_t^2(o)$, the interleaver output $u^i(D)$ should not be in $S_t^2(i)$. When this rule is violated, the swapping process explained in Section 3.3.2.3 under "Weight-Two Input Rules." is applied.

Another case we want to avoid is when both of the interleaved outputs are close (within 300 bits) to the interleaver length N, a case which causes a low codeword weight. This situation is also corrected with the same swapping process.

We mention that whenever a modification is made, the algorithm restarts from the first element in $R_t^2(o)$ and reiterates. When the all of the conditions above are passed for all the elements in $R_t^2(o)$ without a modification, the interleaver enhancement algorithm continues with the weight-three input rules given below.

b. Weight-Three Input Rules:

This process is also very similar to the algorithm given in Subsection 3.3.2.3 under "Weight-Three Input Rules." Suppose $x^o(D) = (D^y + D^z + D^v)$ where $y < z < v$; then we apply the swapping procedure if $x_I^o(D) = (D^{I(y)} + D^{I(z)} + D^{I(v)}) \in R_t^3(o)$ results in $u^i(D) \in S_t^3(i)$ or if all of the interleaved outputs are close to N. Further, we check if two of the interleaved outputs form a weight-two input in a subset of $R_t^2(o)$ (checking this case with $R_t^2(o)$ takes an excessive amount of time to process) and the third interleaved output is close to N. When this is the case, the swapping procedure is applied.

Whenever a modification is made, the interleaver is sent back to weight-two input modifications starting from the first element in $R_t^2(o)$ to verify whether any of the weight-two input rules are violated. The interleaver enhancement algorithm is completed when the weight-three input rules are verified without any modifications.

5.4.2.1 The Interleaver Design Algorithm

In this case, we pseudo-randomly generate an interleaver, but with some constrains, instead of generating a pseudo-random interleaver and then enhancing it. This is due to fewer and weaker constraints on the interleaver than for the 8/8 SCCC case. We note that the interleaver design algorithm has the flavor of the S-random interleavers introduced in [64] since the goal is to separate ones in any weight-two interleaver outputs. We also eliminate the cases when the three ones of weight-three interleaver inputs are interleaved close to N. In this latter case, it is enough to interleave any two of the ones far from each other to have a differential encoder output with a large weight. For most of the cases in the design of SCCCs, $t = 25$ was the largest t value we could use for the weight-two and three sets of the outer encoders.

The following are the steps for generating an interleaver for 8/2 and 4/2 SCCCs.

1. Let I be an empty set and $Q = \{1, 2, \ldots, N\}$. Set a value for the variable S, where $S \lesssim \frac{\sqrt{N}}{2}$ is used to allow convergence with $R_{25}^2(o)$ and $R_{25}^3(o)$.

2. Pick an element $q \in Q$ that is not in I.

3. When the second non-zero element of every weight-two inputs in $R_t^2(o)$ is interleaved to the current q value, check whether the current q value is at least S locations apart from the S most recently assigned values in I. If the current q value does not satisfy this condition, then go to step 2.

4. Check whether the current q value is at least $3S$ away from N. If it is, then assign the current q value in I and go to step 2. If the current q value is within $3S$ from N, then check whether it is at least S locations apart from the S most recently assigned values in I when at least one of the three spans[16] of the entire weight-three inputs in $R_t^3(o)$ is interleaved to q. If at least one of the spans is interleaved more than S locations apart, then assign the current q value in I and go to step 2. If all of the spans are interleaved less than S locations apart, then go to step 2.

5. If I has less than $N-1$ elements, then go to step 2. If I has $N-1$ elements, then assign the last element left in Q to I as the last element in I and stop.

When it is successfully completed, the algorithm described above will separate the weight-two inputs in $R_t^2(o)$ at least S locations apart. Also, the algorithm will design an interleaver that ensures when all of the non-zero elements of any weight-three input in $R_t^3(o)$ are interleaved within $3S$ away from N, at least one of the interleaved non-zero element is S locations apart from the next closest non-zero element. The only exception to these rules is the last interleaved location as described in item 5 above, which usually prevents convergence if it is enforced by the rules given in items 3 and 4.

[16]The three spans of a weight-three input $(D^a + D^b + D^c)$ where $a < b < c$, are $b-a$, $c-a$, and $c-b$.

5.4.2.2 Iterating over $P(p_o, p_i)$ or $P(p_o)$

For 8/8 SCCC, we puncture the parity bits of both outer and inner encoders $P(p_o, p_i)$ whereas, for 8/2 and 4/2 SCCCs, we only puncture the parity bits of the outer encoder $P(p_o)$ over all the possible schemes to find the maximized $d_{2,min}^{SCCC*}$ and $d_{3,min}^{SCCC*}$ for rates 3/4, 7/8, and 15/16.

As in the design of high rate PCCCs, the puncturing scheme that maximized the $d_{2,min}^{SCCC*}$ is usually different than the puncturing scheme that maximized $d_{3,min}^{SCCC*}$. In such a case, we choose the puncturing scheme that simultaneously maximizes $d_{2,min}^{SCCC*}$ and $d_{3,min}^{SCCC*}$. This simultaneous maximization is a process that weights the contributions of the distances $d_{2,min}^{SCCC*}$ and $d_{3,min}^{SCCC*}$ and information weight $2n_2$ and $3n_3$, and chooses the puncturing scheme that minimizes the P_b expression in (3.1) for large SNRs.

There are special rates that require pseudo-random puncturing rather than a fixed location puncturing. The reason they require pseudo-random puncturing and the identification of such rates for a given memory size m is discussed in detail in Subsection 3.3.2.4 and 3.4.3. We only point out here that for $m = 2$, these rates are of the from $3j/(3j+1)$, and for $m = 3$, these rates are of the form $7j/(7j+1)$. We choose various f_2 and f_3 values with $q_0 = 0$ as defined in Subsection io for pseudo-random puncturing of SCCCs. The values for f_2 and f_3 are given in the results section for each rate.

5.5 Design Results

We give the 8/8, 8/2, and 4/2 SCCC optimum design parameter values and BER simulation results on an AWGN channel for each rate in separate subsections. In each subsection, we supply a table that gives the design parameter values, the number of states and rates of outer and inner encoders, the input block size (L), interleaver size (N), S, t, the optimum puncturing schemes (Pu.Sc.), and finally, $(d_{2,min}^{SCCC*}, n_2^*)$ and $(d_{3,min}^{SCCC*}, n_3^*)$ pairs.[17] When pseudo-random puncturing is applied, it is denoted by "PP" in the puncturing notation.

For the BER simulations, we employed the Log-SISO algorithm for each constituent decoder with 15 iterations and an 8-value look-up table for the max* correction term [53], [27]. We terminated only the outer encoder with m tail bits.

We point out here that both serial and parallel concatenated codes decoded by an iterative fashion occasionally produce an oscillating number of errors with an increasing number of iterations without converging to an error-free decoding. This situation is more prevalent for SCCCs. In order to decode blocks with such behavior with the minimum number of errors possible, we observe the variance (or the mean magnitude) of the soft information generated by the outer decoder for the input sequence. During normal operation, as the outer decoder produces soft information from one iteration to the next, the variance of the soft information generally increases. When an oscillation occurs, there is eventually an increase

[17] The t value for 8/8 SCCCs is a pair: the first value is for $R_t^w(o)$ and the second is for $S_t^w(i)$.

in the number of errors from the previous iteration, and the variance of the soft information drops, showing less confidence for the computed soft information. By monitoring these drops, we are able to stop iterating at the point where there is a minimum number of errors.

5.5.1 Simulation Results for $r = 3/4$ SCCCs

Table 5.2: The design parameters found for $r = 3/4$ SCCCs along with some other parameters used in BER simulations.

States	L	N	S	t	Pu.Sc.	$(d_{2,min}^{SCCC*}, n_2^*)$	$(d_{3,min}^{SCCC*}, n_3^*)$
8/8	10003	11432	300	25,40	$P(PP, 5)$	(9,2)	(4,7)
8/2	9999	13332	70	25	$P(1)$	(110,1)	(33,1)
4/2	9999	13332	56	25	$P(PP)$	(42,1)	(43,1)

Table 5.2 presents the design results for rate 3/4 SCCCs. Note that the 8/8 SCCC outer and the 4/2 SCCC outer encoders are special rates for their memory sizes, and required pseudo-random puncturing with parameters $f_3 = 5$ and $f_2 = 2$, respectively. The optimal weights $d_{2,\min}^{SCCC*}$ and $d_{3,\min}^{SCCC*}$ for the 8/8 SCCC are much lower than those of the 8/2 and 4/2 SCCCs due to heavy puncturing. For example, a rate 3/4 8/8 SCCC is produced by a rate 7/8 outer and a rate 6/7 inner encoder, where as a rate 3/4 8/2 SCCC is produced by a rate 3/4 outer encoder. Also observed in the table is the surprising result that the 4/2 SCCC has a lower error rate floor than the 8/2 SCCC as evidenced by their respective values for $d_{2,min}^{SCCC*}$ and $d_{3,min}^{SCCC*}$.

Figs. 5.2, 5.3, and 5.4 show the BER performance of the designed rate 3/4 SCCCs on an AWGN channel. Despite its higher error rate floor, the 8/8 SCCC has the closest BER performance to the channel capacity limit at BER = 10^{-5} among the all SCCC designs considered for this rate. In fact, its performance is exactly the same as the $m = 3$, $r = 3/4$ PCCC designed in Chapter 3 in this region. The 8/2 and the 4/2 SCCCs have about the same performance in this region which is about 0.2 dB inferior to that of the 8/8 SCCC case. However, both have much lower error floors than 8/8 SCCC as indicated in Table 5.2. Note that the 4/2 SCCC is a simpler code than the 8/2 SCCC with the same BER performance and lower error floor for rate 3/4. We mention also that, in the $P_b = 10^{-5}$ region, the rate 3/4 8/2 and 4/2 SCCCs require 3 dB less SNR than a punctured rate 3/4, $m = 6$ convolutional code decoded by Viterbi algorithm.

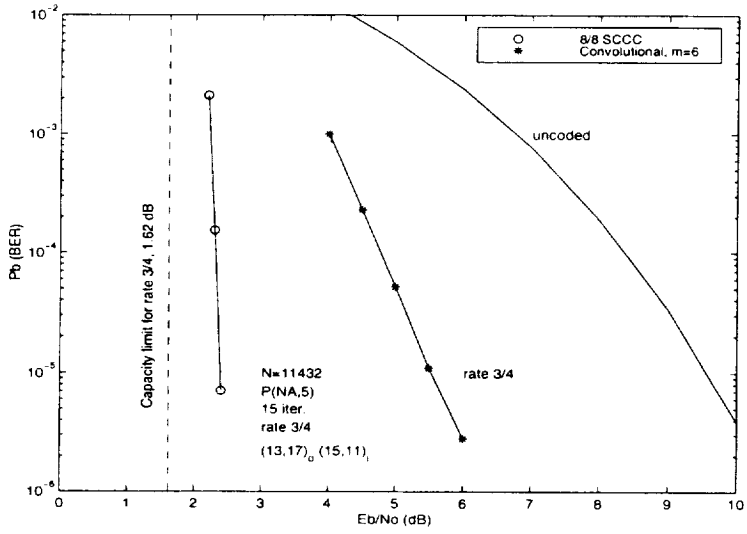

Figure 5.2: 8/8 SCCC rate 3/4 BER performance.

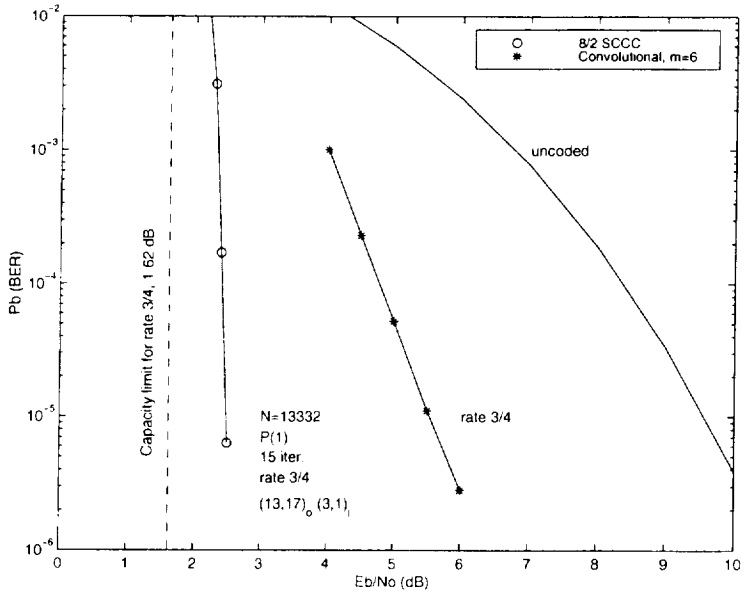

Figure 5.3: 8/2 SCCC rate 3/4 BER performance.

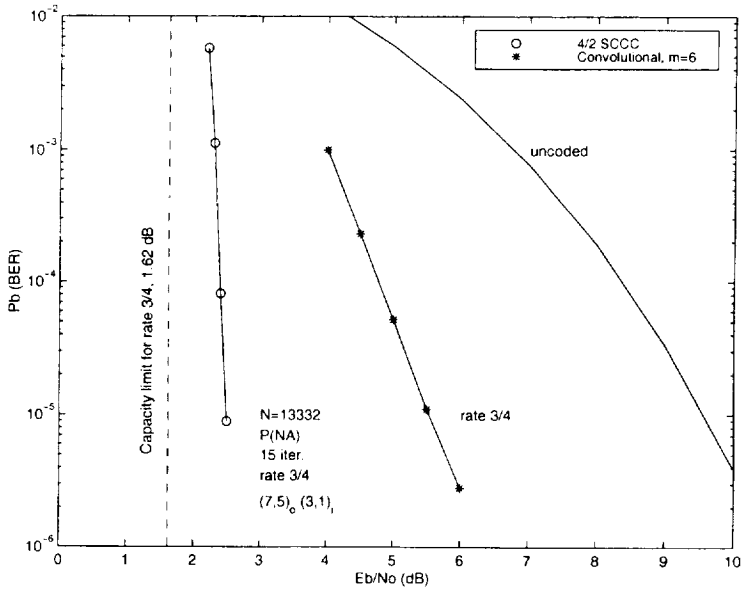

Figure 5.4: 4/2 SCCC rate 3/4 BER performance.

5.5.2 Simulation Results for $r = 7/8$ SCCCs

Table 5.3: The design parameters found for $r = 7/8$ SCCCs along with some other parameters used in BER simulations.

States	L	N	S	t	Pu.Sc.	$(d_{2,min}^{SCCC*}, n_2^*)$	$(d_{3,min}^{SCCC*}, n_3^*)$
8/8	10005	10672	300	23,45	$P(5, PP)$	(5,3)	(4,4)
8/2	10003	11432	60	25	$P(PP)$	(42,1)	(9,1)
4/2	10003	11432	54	25	$P(2)$	(34,1)	(11,1)

Table 5.3 presents the design results for rate 7/8 SCCCs. Note that the 8/8 SCCC inner and the 8/2 SCCC outer encoders are special rates for their memory sizes, and required pseudo-random puncturing with parameters $f_3 = 5$ and $f_3 = 4$, respectively. Again, the optimal weights $d_{2,\text{min}}^{SCCC*}$ and $d_{3,\text{min}}^{SCCC*}$ for 8/8 SCCC are much lower than for 8/2 and 4/2 SCCCs due to heavy puncturing. Again, the 4/2 SCCC has a slightly better error floor than the 8/2 SCCC. We note that for this rate, the 4/2 SCCC interleaver design was completed by avoiding the rules 3 and 4 for the last three q values in Section 5.4.2.1.

Figs. 5.5, 5.6, and 5.7 show the BER performance of the designed rate 7/8 SCCCs on an AWGN channel. For this rate, the BER performance of the 8/8 SCCC and the 8/2 SCCC are identical for the error rate simulated. The 4/2 SCCC performance is inferior by about 0.3 dB to the 8/2 SCCC at a BER of 10^{-5}. The SCCCs offer a 3 dB gain at 10^{-5} over a punctured rate 7/8, $m = 6$ convolutional code as seen in the figure.

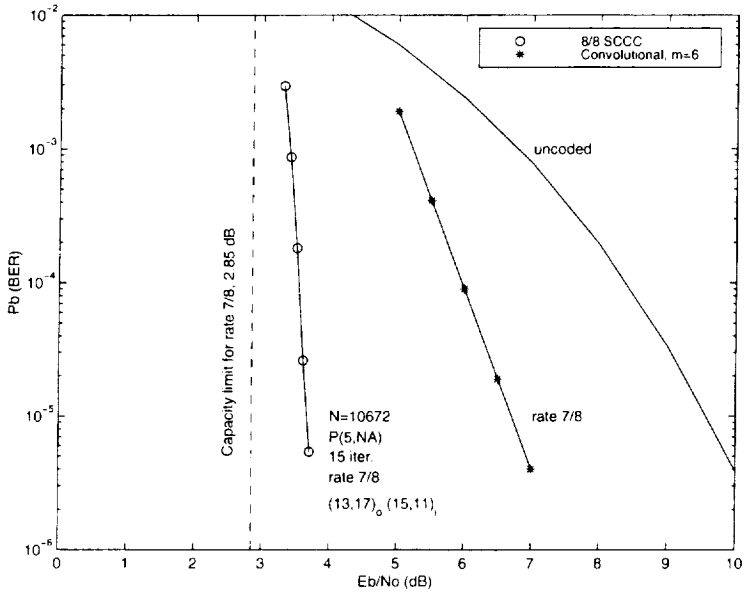

Figure 5.5: 8/8 SCCC rate 7/8 BER performance.

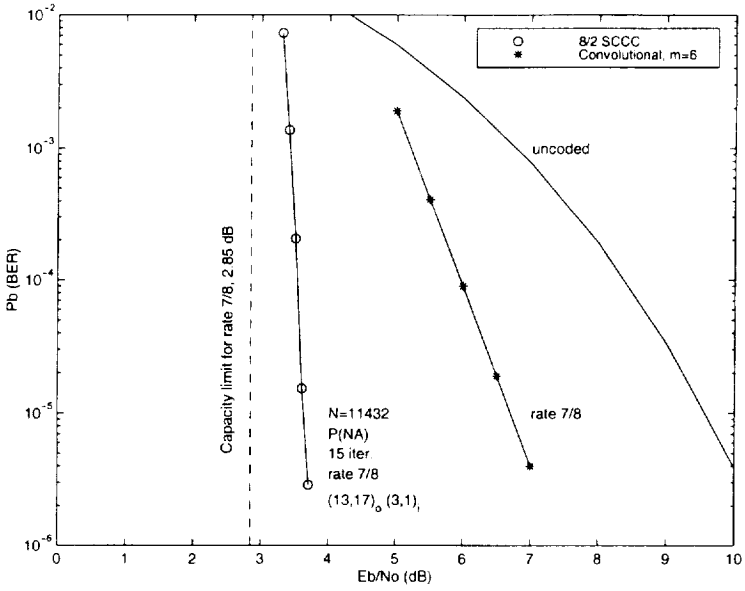

Figure 5.6: 8/2 SCCC rate 7/8 BER performance.

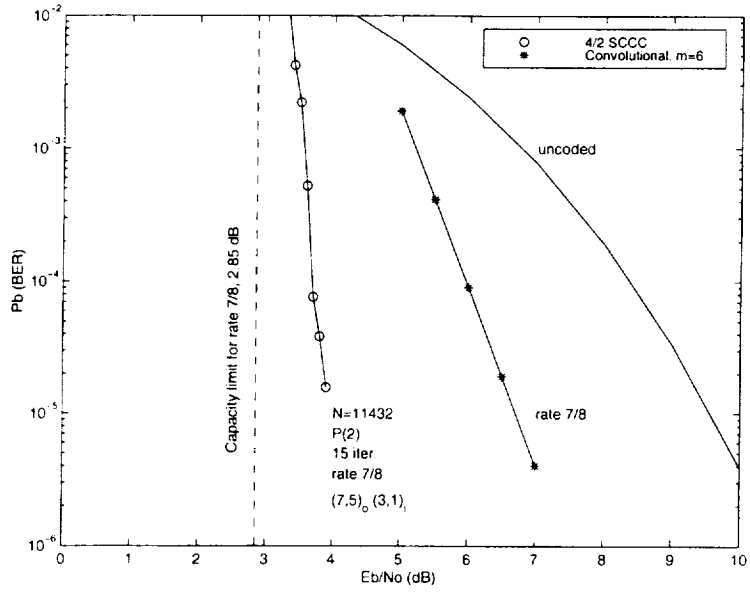

Figure 5.7: 4/2 SCCC rate 7/8 BER performance.

5.5.3 Simulation Results for $r = 15/16$ SCCCs

Table 5.4: The design parameters found for $r = 15/16$ SCCCs along with some other parameters used in BER simulations.

States	L	N	S	t	Pu.Sc.	$(d_{2,min}^{SCCC*}, n_2^*)$	$(d_{3,min}^{SCCC*}, n_3^*)$
8/8	10013	10336	300	21,40	$P(1,2)$	(2,7)	(3,2)
8/2	10005	10672	60	25	$P(7)$	(25,1)	(10,2)
4/2	10005	10672	54	25	$P(PP)$	(17,1)	(9,1)

Table 5.4 presents the design results for rate 15/16 SCCCs. For this rate, only the 4/2 SCCC outer encoder is a special rate, and required pseudo-random puncturing with parameter $f_2 = 1$. The optimal weights $d_{2,\min}^{SCCC*}$ and $d_{3,\min}^{SCCC*}$ for the 8/8 SCCC are at their minimum possible ($d_{2,\min}^{SCCC*} = 2$ and $d_{3,\min}^{SCCC*} = 3$). For

this rate, the 4/2 SCCC has slightly higher error floor than the 8/2 SCCC since the 8/2 SCCC has larger distance parameters.

Figs. 5.8, 5.9, and 5.10 show the BER performance of the designed rate 15/16 SCCCs in AWGN. As for the r=7/8 case, the BER performance of the 8/8 and 8/2 SCCCs for this rate are identical for the region simulated. The 4/2 SCCC performance is inferior by about 0.4 dB to the 8/2 SCCC at 10^{-5}. When the BER performance of the rate 1/2, $m = 6$ convolutional code is compared to that of the rate 15/16 8/2 SCCC, the latter is inferior by only 0.3 dB at a BER of 10^{-5}. However, the SCCC is 85% more bandwidth efficient.

In summary, we find 8/2 SCCCs as the best performer for all rates which have comparable, if not better, performance to the 8/8 SCCCs in the cliff region and have comparable or lower error floors than the 4/2 SCCCs. For r=3/4 applications, the 4/2 SCCC can be a good and simple code since it has comparable BER performance to the 8/8 and 8/2 SCCCs with the lowest error floor for this rate. The 8/8 SCCCs stand as a good performer for BERs down to 10^{-6}, but suffer from high error floors due to heavy puncturing.

5.6 Performance Comparison Between 8/2 SCCCs and $m = 3$ PCCCs

In Figs. 5.11, 5.12, and 5.13, we compare the BER performance of 8/2 SCCCs and PCCCs designed in Chapter 3 with two identical constituent RSC encoders

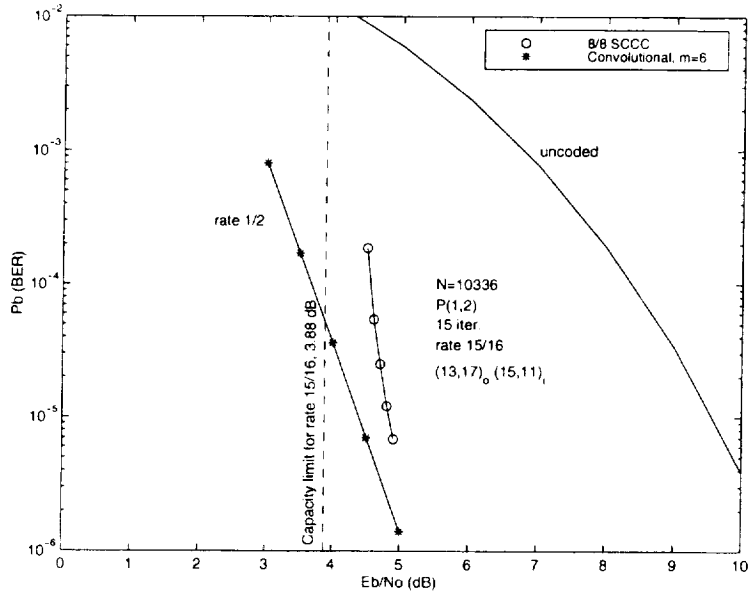

Figure 5.8: 8/8 SCCC rate 15/16 BER performance.

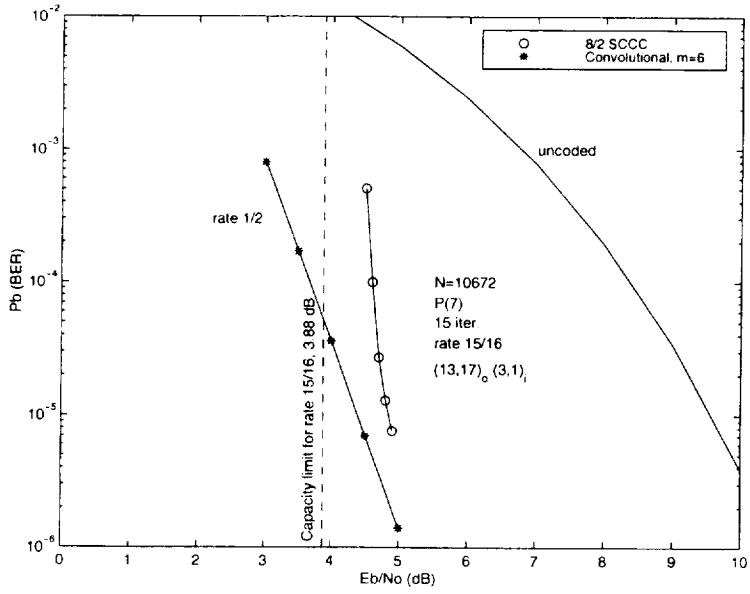

Figure 5.9: 8/2 SCCC rate 15/16 BER performance.

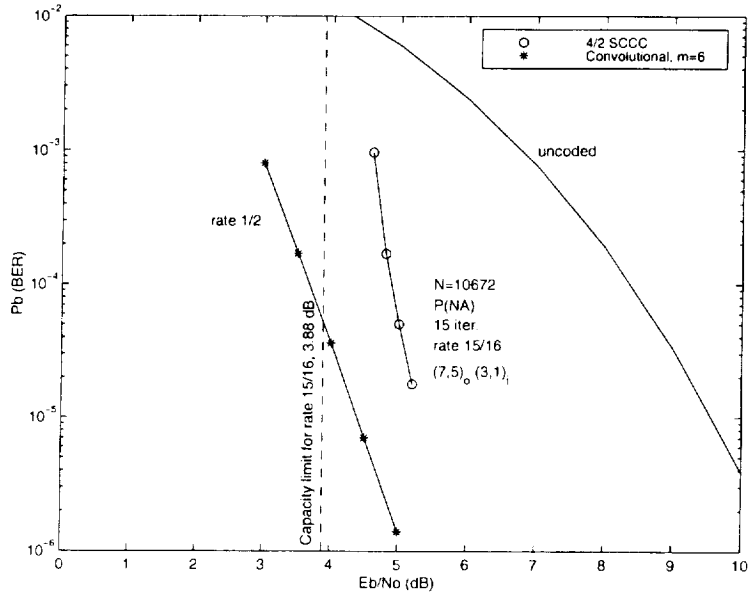

Figure 5.10: 4/2 SCCC rate 15/16 BER performance.

with $m = 3$, for rates 3/4, 7/8, and 15/16. In Fig. 5.14, we show the analytically computed error rate floors for these codes.

In order to find the analytical bounds, we generated all dominant weight-two and -three inputs for the outer encoder for each case (8/8, 8/2, and 4/2 SCCCs) and rate (3/4, 7/8, and 15/16). Then, these inputs and their all possible shifts are fed to SCCCs to observe the minimum codeword weights for each puncturing scheme. Finally, by substituting in the values for $d_{2,min}^{SCCC*}$, $d_{3,min}^{SCCC*}$, $2n_2^*$, and $3n_3^*$ (see Tables 5.2, 5.3, and 5.4) in (3.1) for each case and rate, we were able to plot Fig. 5.14. The same process is applied to plot analytically computed error rate floor for the PCCCs designed in Chapter 3.

In Fig. 5.11, the rate 3/4 performance comparison of the two codes is given which shows the 8/2 SCCC performance is inferior to PCCC's by about 0.2 dB at a BER of 10^{-5}. However, Fig. 5.14 shows that the error rate floor for the rate 3/4 8/2 SCCC is much lower than that of the rate 3/4 PCCC. In Fig. 5.12, the rate 7/8 BER performance comparison shows no difference between the 8/2 SCCC and the PCCC, but the 8/2 SCCC again has a much lower error rate floor than the PCCC as shown in Fig. 5.14. Finally, in Fig. 5.13, the rate 15/16 performance comparison shows almost no difference between the two codes, but again the 8/2 SCCC error rate floor is still much lower than the PCCC as seen in Fig 5.14.

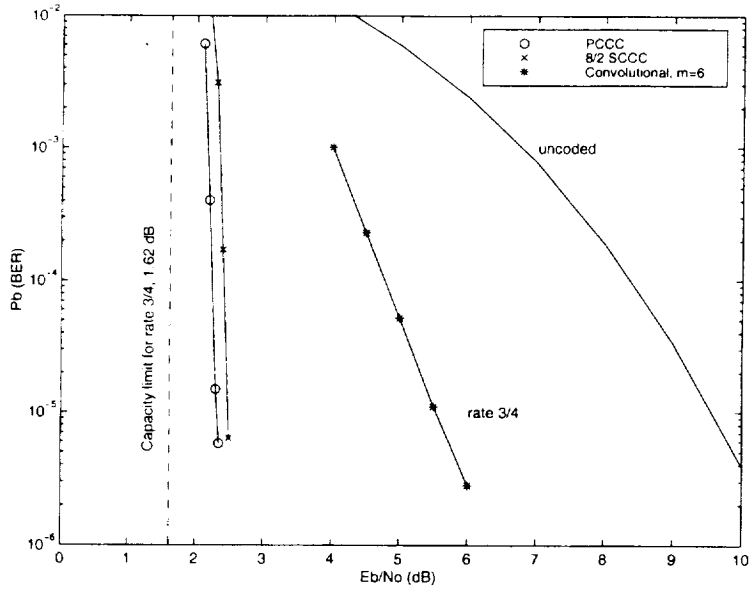

Figure 5.11: 8/2 SCCC and $m = 3$ PCCC BER performance comparison for rate 3/4.

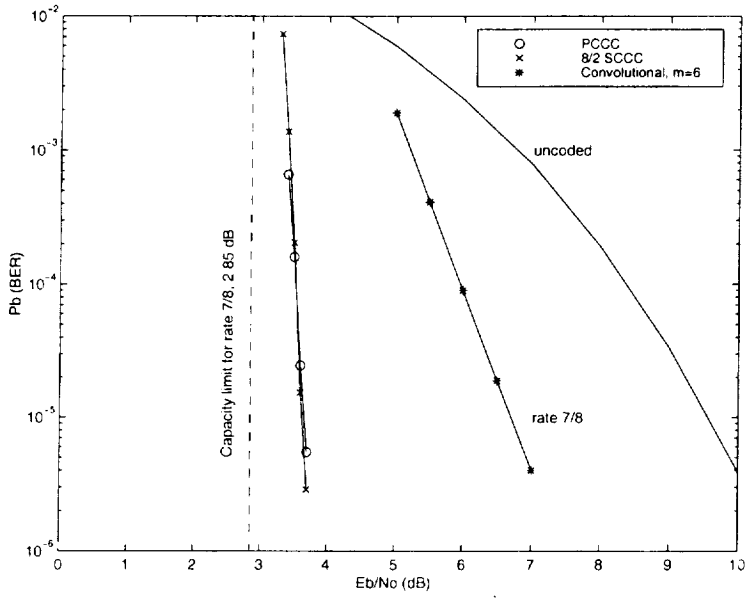

Figure 5.12: 8/2 SCCC and $m = 3$ PCCC BER performance comparison for rate 7/8.

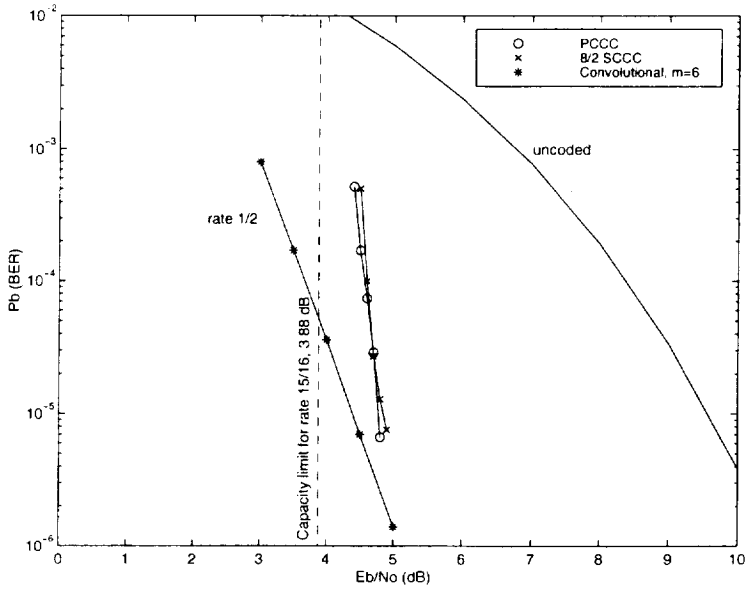

Figure 5.13: 8/2 SCCC and $m = 3$ PCCC BER performance comparison for rate 15/16.

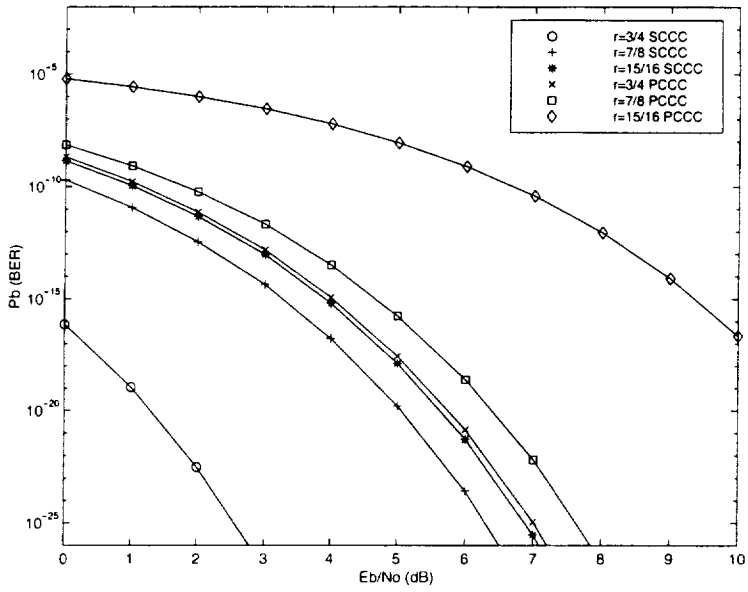

Figure 5.14: Error floors for 8/2 SCCCs and $m = 3$ PCCCs for rates 3/4, 7/8, and 15/16.

5.7 Conclusions

We described an algorithm for designing high rate punctured 8/8, 8/2, and 4/2 SCCCs for rates 3/4, 7/8, and 15/16 on the AWGN channel. The algorithm includes several steps to maximize the worst case weight-two and -three input codeword weights and minimize their multiplicities.

Our simulation results showed that the r=3/4, 8/8 SCCC showed the best BER performance which is only 0.7 dB away from capacity, while the BER performance of the 8/2 and 4/2 SCCCs are identical and their performance is only 0.9 dB away from capacity at BER of 10^{-5}. For this rate, 4/2 SCCC has the lowest

error floor (largest $d_{2,\min}^{SCCC*}$ and $d_{3,\min}^{SCCC*}$). Note that for r=3/4, 4/2 SCCC also has comparable BER performance with much lower error floor over $m = 3$ PCCC which is computationally much more complex than 4/2 SCCC. For the rates 7/8 and 15/16, 8/2 SCCC BER performance with much lower error floor is identical to $m = 3$ PCCC which is computationally more complex than 8/2 SCCC. For rates 7/8 and 15/16, the 8/2 SCCC performance is only 0.8 and 0.9 dB away from capacity at 10^{-5} BER, respectively.

We point out that one of our goals in the design of high rate SCCCs was that they have lower error floors than high rate PCCCs. As our results show, it is possible to design high rate SCCCs that perform as well as PCCCs for low SNRs and perform better at medium to high SNRs.

6 CONCLUSIONS AND RECOMMENDATIONS

6.1 Conclusions

In this dissertation, we described algorithms for designing punctured high rate PCCCs and SCCCs for the AWGN channel and showed design results for selected rates. In Chapter 1, we reviewed some of the essential concepts and definitions to understand the importance of channel coding in digital communication systems. After giving a brief history of channel coding, we described *bandwidth efficiency* and *channel capacity*, the later concept an important parameter for digital communication system as formulated by Shannon. We then reviewed convolutional codes, and focused on non-recursive and recursive systematic convolutional codes which are employed by PCCCs and SCCCs. We introduced some of their important parameters like minimum Hamming distance and their multiplicities which dominates performance at medium to high SNRs. Finally, we gave brief introduction to PCCCs and SCCCs: their structure that leads BER performance close to capacity. In Chapter 2, we reviewed the decoding algorithms for the constituent decoders employed in the iterative decoding of PCCCs and SCCCs. We first gave a brief history of the ML decoding algorithms which minimize codeword error for convolutional codes, namely sequential and VA decoding. Then we introduced a MAP decoding algorithm for convolutional codes called the BCJR algorithm which minimizes the BER. We pointed out that the BCJR is a MAP algorithm for convolutional codes but an APP decoding algorithm for concatenated schemes. We

then introduced the APP decoding algorithm tailored for the iterative decoding of PCCCs. We reviewed the Log-APP algorithm which reduces the computational complexity of the APP, and the SW-Log-APP algorithm which reduces the decoding delay of a single constituent decoder. Finally, we introduced the SISO decoding algorithm which is employed in the iterative decoding of SCCCs.

Chapter 3 is devoted to the design of punctured high rate PCCCs for rates $k/(k+1)$ where $k = 2, 3, \ldots, 16$ with constituent encoder memory sizes $m = 3$, and 4. In this chapter, we explain the structure of the PCCC encoder we considered for our designs, the reason that RSC encoders are employed as constituent encoders, the special weight-two and -three inputs that are more important than any other inputs, and the functionality of the interleaver between the two RSC encoders. We showed that codewords generated by these certain weight-two and -three inputs and their multiplicities dominate the BER for medium to high SNRs. We designed an algorithm that systematically varies the set of dominant special weight-two and -three inputs, the generator polynomial sets, puncturing schemes, and the interleaver to maximize the weights and minimize the multiplicities of these codewords generated by these special inputs. Our BER simulations results showed that it is possible to design punctured high rate PCCCs with $m = 3$ and 4 that their BER performance is within 0.9 and 0.75 dB, respectively, of the capacity limit at 10^{-5} BER. Despite their excellent BER performance, PCCCs have relatively high error floors for smaller interleaver sizes. It is clear that for

applications requiring high reliability (BER $< 10^{-10}$) with fast data transmission rates (requiring small decoding delays, hence, small interleavers), PCCCs are not suitable.

In Chapter 4, we investigate a practical implementation of the punctured high rate PCCCs designed in Chapter 3. We focused on the implementation issues for rates 3/4, 7/8, and 15/16. We addressed quantization, suboptimum implementations (over- and under-flow of decoding parameters), decoding delays, and SNR estimation error issues that effect BER performance of PCCCs. We simulated 4-, 6-, and 8-bit quantization implementation of the SW-Log-APP algorithm and found that 8-bit implementation resulted in less than 0.2 dB implementation loss for all the rates of interest.

In Chapter 5, we gave the details of an algorithm to design punctured high rate SCCCs for rates 3/4, 7/8, and 15/16. Three different types of SCCCs are considered: 8/8, 8/2, and 4/2 SCCCs. Similar to the PCCC design, our goal is to maximize the codeword weight and minimize their multiplicities caused by weight-two and -three inputs. We varied a subset of modified dominant weight-two and -three inputs for outer encoders, a subset of dominant weight-two and -three inputs for the inner encoder, the interleaver, and the puncturing schemes to maximize the minimum codeword weight and minimize their multiplicities to improve their BER for medium to high SNRs. The simulations results showed that it is possible to design simple high rate SCCCs to have comparable or the same

BER performance with lower error floors over $m = 3$ PCCCs for rates 3/4, 7/8, and 15/16. Note that for rate 3/4, a 4/2 SCCC can be employed and a comparable BER performance with much lower error floor can be achieved over $m = 3$ PCCC which is computationally more complex. A similar conclusion can be drawn for the rates 7/8 and 15/16 that 8/2 SCCC has identical BER performance with a much lower error floor than the $m = 3$ PCCC which is computationally more complex.

6.2 Suggestions for Future Research

Throughout this dissertation, we assumed BPSK and QPSK signaling over a AWGN channel. We recommend the study of the design of high rate punctured PCCCs and SCCCs for other types of channels such as Rayleigh or Ricean channels which find applications in today's code division multiple access (CDMA) technology.

In Chapter 4, we showed a way to implement high rate PCCCs designed in Chapter 3. During the implementation process, we chose the same quantization levels for the four most important parameters of the SW-Log-APP algorithm based on their equal importance in the decoding process. We suggest the investigation of the usage of different quantization levels for these parameters to reduce the complexity of the decoder while keeping the BER performance as close to the theory as possible.

In the summer of 1998, rate 1/2, 3/4, 7/8, 9/10, 13/14, and 19/20 PCCCs were designed for Adaptive Broadband (formerly EF Data Corp.) in Tempe, AZ using the technique described here in. In Xilinx (hardware description language by Xilinx Corp.) field programmable gate array (FPGA) design of the codes, Small World's MAP decoders were employed [65]. A detailed report by Eric Jacobsen from Adaptive Broadband on the designs and BER versus E_b/N_0 curves can be found in [66]. In these designs, a Reed-Solomon code is employed as an outer code to lower the error floor. This outer Reed-Solomon code is capable of correcting five 8-bit symbol errors within an interleaver block. As the BER curves in the report indicates, the error floor is lowered by employing the Reed-Solomon codes. Except for rate 1/2 and 3/4 BER (see Figs. A1 and A2 in the report) curves, simulated results agreed with the lab results. The reason for this 0.8 and 0.35 dB difference between the simulated and actual rate 1/2 and 3/4 PCCC performance, respectively, is due to locking problems in the phase detector (carrier recovery). For these rates, the phase detector "sees" the channel symbol energy per AWGN power density, E_s/N_0, of -1.8 and 1.25 dB for rate 1/2 and 3/4, respectively, at BER of 10^{-6} in the simulations. These low values make it difficult to keep the phase loop locked. Adaptive Broadband plans to convert the Xilinx FPGA design to an ASIC design which does not allow any changes in the hardware, but runs about two to five times faster than the FPGA version.

For CDMA applications power is one of the main driving parameters of the system designs. Improvements in DSP reduces the size and the cost of the chips for more sophisticated processes but the power required for data transfer is dominated by the signalling/coding schemes. By employing high rate PCCCs and SCCCs in CDMA applications, it is possible to reduce the required power to keep the same level of reliability. Although for CDMA applications, AWGN channel itself does not represent the channel, Reed-Solomon codes concatenated with high rate PCCCs have around 1.3 dB gain over Reed-Solomon codes concatenated with convolutional codes on an AWGN channel [66]. For this reason, it is important to investigate the performance of the former scheme on fading channels for CDMA applications.

Ungerboeck proposed and analyzed the Trellis coded modulation (TCM) scheme for multilevel signaling [6] to increase the bandwidth efficiency. State-of-art pragmatic (practical) rate $k/(k+1)$ TCMs are designed by employing rate 1/2 convolutional codes [49]. In [39] and [67]-[70], several ways of designing multilevel, bandwidth efficient turbo TCM (TTCM) schemes are described. We suggest to design pragmatic TTCM schemes to improve the bandwidth efficiency further by employing designed high rate PCCC and SCCCs.

REFERENCES

[1] C. E. Shannon, "A Mathematical Theory of Communication," *Bell Syst. Tech. J.*, vol. 27, pp. 379-423, 1948.

[2] M. J. E. Golay, "Notes on digital coding," *Proc. IEEE*, Vol. 37, p.657, 1949.

[3] R. W. Hamming, "Error detecting and error correcting codes, " *Bell Syst. Tech. J.*, Vol. 29, pp. 147-160, 1950.

[4] P. Elias, "Coding for noisy channels," *IRE Conv. Rec.*, Part 4, pp. 37-47, 1955.

[5] G. D. Forney, *Concatenated Codes*, MIT Press, Cambridge, MA, 1966.

[6] G. Ungerboeck, "Channel coding with multilevel/phase signals," *IEEE trans. Comm.*, vol. IT-28, No. 1, Jan 1982.

[7] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: turbo-codes," *Int. Conf. on Comm.*, Geneva, Switzerland, pp. 1064-1070, May 1993.

[8] L. C. Perez, J. Senghers, and D. J. Castello Jr., " A distance spectrum interpretation of turbo codes," *IEEE Trans. on Inf. Theory*, vol. 42, no. 6, pp. 1698-1709, Nov. 1996.

[9] S. Benedetto, G. Montorsi, D. Divsalar, and F. Pollara, "Serial concatenation of interleaved codes: performance analysis, design, and iterative decoding," *TDA Progress Report 42-126*, pp. 1-26, Aug 15, 1996.

[10] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. on Inf. Theory*, pp. 284-287, March 1974.

[11] J. M. Wozencraft and B. Reiffen, *Sequential Decoding* , MIT Press, Cambridge, MA, 1961.

[12] R. M. Fano, "A heuristic discussion of probabilistic decoding," *IEEE Trans. on Inf. Theory*, IT-9, pp. 64-74, April 1963.

[13] K. Zigangirov, "Some sequential decoding procedures," *Probl. Peredachi Inf.*, 2, pp. 13-25, 1966.

[14] F. Jelinek, "A fast sequential decoding algorithm using a stack," *IBM J. Res. and Dev.*, 13, pp. 675-685, November, 1969.

[15] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. on Inf. Theory*, IT-3, pp. 260-269, April 1967.

[16] G. D. Forney, Jr. "The Viterbi algorithm," *Proc. IEEE*, 61, pp.268-278, March 1973.

[17] G. D. Forney, Jr. "Convolutional codes II: maximum likelihood decoding," *Inf. Control*, 25, pp. 222-266, July 1974.

[18] S. Lin and D. J. Costello, Jr. , "*Error control coding: fundamentals and applications*," Prentice-Hall, Englewood Cliffs, New Jersey, 1983.

[19] J. Hagenauer and P. Hoeher, "A viterbi algorithm with soft-decision outputs and its applications," *IEEE Globecom Conf.*, Dallas, TX, pp. 1680-1686, Nov. 1989.

[20] J. Lodge, P. Hoeher, and J. Hagenauer, "The decoding of multidimensional codes using separable MAP 'filters'," in *Proc. 16th Biennial Symp. on Communications*, pp. 343-346, Queen's University, Kingston, Ont., Canada, May 1992.

[21] J. Lodge, R. Young, P. Hoeher, and J. Hagenauer, "Separable MAP 'filters' for the decoding of product and concatenated codes," in *IEEE Int. Conf. on Comm.*, pp. 1740-1745, Geneva, Switzerland, May 1993.

[22] J. Hagenauer and P. Hoeher, "Concatenated Viterbi-decoding," *4th Joint Swedish-Soviet Int. Workshop on Inf. Theory*, pp. 29-33, Gotland, Sweden, Aug. 1989.

[23] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. on Inf. Theory*, vol. 42, no. 2, pp. 429-445, March 1996.

[24] W. Ryan, "A turbo code tutorial," unpublished paper available at http://www.ece.arizona.edu/~ryan

[25] P. Robertson, "Illuminating the structure of code and decoder of parallel concatenated recursive systematic (turbo) codes," *GlobeCom*, pp. 1298-1303, 1994.

[26] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "A soft-input soft-output maximum a posteriori (MAP) module to decode parallel and serial concatenated codes," *TDA Progress Report 42-127*, pp. 1-20, November, 1996.

[27] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," *IEEE Int. Conf. on Comm.*, Seattle, WA, vol. 2, pp. 1009-1013, June 1995.

[28] J. A. Erfanian, S. Pasupathy, and G. Gulak, "Reduced complexity symbol detectors with parallel structures for isi channels," *IEEE Trans. on Comm.*, pp. 1661-1671, February/March/April 1994.

[29] W. Koch and A. Baier, "Optimum and suboptimum detection of coded data distributed by time-varying intersymbol interference," *in Proc. GLOBRCOM '90*, pp. 1679-1684, December 1994.

[30] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Soft-output decoding algorithms in iterative decoding of turbo codes," *TDA Progress Report 42-124*, pp. 63-87, February, 1996.

[31] S. Pietrobon, "Implementation and performance of a turbo/MAP decoder," *Int. Journal of Satellite Comm.*, vol. 16, pp. 23-46, Jan.-Feb. 1998.

[32] S. A. Barbulescu, *Iterative decoding of turbo codes and other concatenated codes*, Ph. D. Dissertation, University of South Australia, Feb 1996.

[33] S. Pietrobon, R. Deng, A. Lafanechére, G. Underboeck, and D. Costello, "Trellis-coded multidimensional phase modulation," *IEEE Trans. on Inf. Theory*, vol. 36, no. 1, pp. 63-89, Jan. 1990.

[34] B. Kopp, *An Analysis of Carrier Phase Jitter in an MPSK Receiver Utilizing MAP Estimation*, Ph.D. Dissertation, New Mexico State University, Las Cruces, May 1994.

[35] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: turbo-codes," *IEEE Trans. on Comm.*, vol. 44, no. 10, pp. 1064-1070, Oct. 1996.

[36] D. Divsalar and F. Pollara, "On the design of turbo codes," *TDA Progress Report 42-123*, pp. 99-121, Nov. 1995.

[37] D. Divsalar and F. Pollara, "Turbo codes for deep-space communications," *TDA Progress Report 42-120*, pp. 29-39, Feb. 1995.

[38] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Parallel concatenated trellis coded modulation," *IEEE Int. Conf. on Comm.*, Dallas, TX, vol. 2, pp. 974-978, June 1996.

[39] P. Robertson and Thomas Wörz, "A novel bandwidth efficient coding scheme employing turbo codes," *IEEE Int. Conf. on Comm.*, Dallas, TX, vol. 2, pp. 962-967, June 1996.

[40] S. Riedel, "MAP decoding of convolutional codes using reciprocal dual codes," *IEEE Trans. Inform. Theory*, vol. 44, no. 3, pp. 1176-1187, May 1998.

[41] C. P. Hartmann and L.D. Rudolph, "An optimum symbol-by-symbol decoding rule for linear codes," *IEEE Trans. on Inf. Theory*, vol. IT-22, no. 5, pp. 514-517, Sept. 1976.

[42] S. Benedetto and G. Montorsi, "Design of parallel concatenated convolutional codes," *IEEE Trans. on Comm.*, vol. 44, no. 5, pp. 591-600, May 1996.

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

[44] W. J. Blackert, E. K. Hall, and S. G. Wilson, "An upper bound on turbo code free distance," *IEEE Int. Conf. on Comm.*, Dallas, TX, vol. 2, pp. 957-961, June 1996.

[45] S. Wilson, Personal Communication, 1996.

[46] W. Ryan, L. Han, and P. Quintana, "Design of a low-orbit-to-geostationary satellite link for maximal throughput," *IEEE Trans. Comm.*, vol. 45, no. 8, pp. 988-996, Aug. 1997.

[47] D. N. Rowitch and L. B. Milstein, "Rate compatible punctured turbo (RCPT) codes in a hybrid FEC/ARQ system," *Globecom*, Phoenix, AZ, vol. 4, pp. 55-59, Nov. 1997.

[48] J. Hagenauer, "Rate compatible punctured convolutional codes (RCPC codes) and their applications," *IEEE Trans. on Comm.*, vol. 36, no. 4, pp. 389-400, April 1988.

[49] Data Sheet by QualComm Corp., "Q1900 Viterbi/Trellis Decoder," available at http://www.qualcomm.com/ProdTech/asic/vtdec.html.

[50] Todd A. Summers and Stephen G. Wilson, "SNR Mismatch and online estimation in turbo decoding," *IEEE Trans. on Comm.*, vol. 46, pp. 421-423, April 1998.

[51] M. Jordan and R. Michols, "The effects of channel characterization on turbo code performance," in *Proc. Milcom'96*, McLean, VA, Oct. 1996.

[52] M. Reed and J. Asenstorfer, "A novel variance estimator for turbo-code decoding," in *Int. Conf. Telecommunications*, Melbourne, Australia, pp. 173-178, April 1997.

[53] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial concatenation of interleaved codes: performance analysis, design, and iterative decoding," *IEEE Trans. on Inf. Theory*, vol. 44, no. 3 pp. 909-926, May 1998.

[54] Krishna R. Narayanan and Gordon L. Stüber "A serial concatenation approach to iterative demodulation and coding," *IEEE Tran. on Comm.*, vol. 47, no. 7, pp. 956-961, July 1999.

[55] P. Hoeher and J. Lodge, " "Turbo DSPSK": iterative differential PSK demodulation and channel decodings," *IEEE Trans. on Comm.*, vol. 47, no. 6, pp. 837-843, June 1999.

[56] M. Peleg and S. Shamai (Shitz), "Iterative decoding of coded and interleaved noncoherent multiple symbol detected DSPSK," *Electron. Lett.*, vol. 33, no. 12, pp. 1018-1020, June 1997.

[57] W. Ryan, "Performance of high rate turbo codes on a PR4-equalized magnetic recording channel," *International Conference on Communications*, pp. 947-951, Atlanta, GA, 1998.

[58] W. Ryan, L. McPheters, and S. McLaughlin, "Combined turbo coding and turbo equalization for PR4-equalized Lorentzian channels," *Conf. on Information Systems and Sciences*, 1998.

[59] W. Ryan, "Optimal code rates for concatenated codes on a PR4-equalized magnetic recording channel," *Proc. Conf. on Inf. Sciences and Systems*, pp. 432-437, Mar. 1999.

[60] L. McPheters, S. McLaughlin, and K. Narayanan, "Precoded PRML, serial concatenated, and iterative (turbo) decoding for digital magnetic recording," *International Magnetics Conference.*, 1999.

[61] L. McPheters, S. McLaughlin, and E. Hirsch, "Turbo codes for PR4 and EPR4 magnetic recording," *Proc. of the 1998 Asilomar Conf. on Computers and Communications*, Pacific Grove, CA, pp. 1778-1782, Nov. 1998.

[62] T. Souvignier, A. Friedmann, M. Öberg, P. Siegel, R. Swanson, and J. Wolf, "Turbo decoding for PR4: parallel versus serial concatenation," *International Conference on Communication*, Vancouver, Canada, 1999.

[63] M. Öberg and P. Siegel, "Performance analysis of turbo-equalized dicode partial-response channels," *Proc. 36th Allerton Conf. on Communication, Control, and Computing*, Sept. 1998.

[64] S. Dolinar and D. Divsalar, "Weight distributions for turbo codes using random and nonrandom permutations," *The JPL TDA Progress Report* 42-121, Apr-June 1995, Aug 15, 1995.

[65] http://www.sworld.com.au/pub/

[66] http://web.nmsu.edu/~oacikel/pub.htm

[67] S. Goff, A. Glavieux, and C. Berrou, "Turbo-codes and high spectral efficiency modulation," *International Conf. on Comm.*, New Orleans, LA, May 1994.

[68] R. Pyndiah, A. Glavieux, and A. Picart, "Near optimum decoding of product codes," *Globecom*, San Francisco, Nov. 1994.

[69] R. Pyndiah, A. Picart, and A. Glavieux, "Performance of block turbo coded 16-QAM and 64-QAM modulations," *Globecom*, Singapore, Nov. 1995.

[70] S. Benedetto, D. Divsalar, G. Montorsi, and F. Polara, " Bandwidth efficient parallel concatenated coding schemes," *Electronic Letters*, vol. 31, Nov. 23 1995.