
TURBO CODES

Principles and Applications

**THE KLUWER INTERNATIONAL SERIES
IN ENGINEERING AND COMPUTER SCIENCE**

TURBO CODES

Principles and Applications

Branka Vucetic

*The University of Sydney
Sydney, Australia*

Jinhong Yuan

*The University of New South Wales
Sydney, Australia*



Springer Science+Business Media, LLC

Library of Congress Cataloging-in-Publication

Vucetic, Branka.

Turbo codes : principles and applications / Branka Vucetic, Jinhong Yuan.

p. cm. -- (The Kluwer international series in engineering and computer science
; SECS 559.)

Includes bibliographical references and index.

ISBN 978-1-4613-7013-0 ISBN 978-1-4615-4469-2 (eBook)

DOI 10.1007/978-1-4615-4469-2

1. Coding theory. 2. Signal theory (Telecommunication) I. Yuan, Jinhong, 1969- II.

Title. III. Series.

TK5102.92. V83 2000

003'.54--dc21

00-033104

Copyright © 2000 by Springer Science+Business Media New York

Originally published by Kluwer Academic Publishers, New York in 2000

Softcover reprint of the hardcover 1st edition 2000

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, mechanical, photo-copying, recording, or otherwise, without the prior written permission of the publisher,
Springer Science+Business Media, LLC.

Printed on acid-free paper.

Contents

List of Acronyms	xi
List of Figures	xiii
List of Tables	xxiii
Preface	xxv
1 Introduction	1
1.1 Digital Communication System Structure	2
1.2 Fundamental Limits	5
2 Block Codes	13
2.1 Block Codes	13
2.2 Linear Systematic Block Codes	15
2.3 Parity Check Matrix	16
2.4 The Minimum Distance of a Block Code	17
2.5 Maximum Likelihood Decoding of Block Codes for a BSC Channel	18
2.6 Maximum Likelihood Decoding of Block Codes for a Gaussian Channel	19
2.7 Weight Distribution of Block Codes	20
2.8 Performance Upper Bounds	23
2.8.1 Word Error Probability Upper Bounds	23
2.8.2 Bit Error Probability Upper Bounds	26
2.9 Coding Gain	28
2.10 Soft Decision Decoding of Block Codes	30
2.11 Trellis Structure of Linear Binary Block Codes	30

3 Convolutional Codes	37
3.1 Introduction	37
3.2 The Structure of $(n,1)$ Convolutional Codes	38
3.3 The Structure of (n,k) Convolutional Codes	43
3.4 Systematic Form	45
3.5 Parity Check Matrix	50
3.6 Catastrophic Codes	51
3.7 Systematic Encoders	53
3.8 State Diagram	58
3.9 Trellis Diagram	60
3.10 Distance Properties of Convolutional Codes	62
3.11 Weight Distribution of Convolutional Codes	63
3.12 Punctured Convolutional Codes	66
4 Turbo Coding Performance Analysis and Code Design	73
4.1 Introduction	73
4.2 Turbo Coding	74
4.2.1 A Turbo Encoder	74
4.2.2 Interleaving	76
4.2.3 Trellis Termination	77
4.2.4 High Rate Turbo Codes	78
4.3 Performance Upper Bounds of Turbo Codes	80
4.3.1 Conditional WEF's of Average Turbo Codes	80
4.3.2 Conditional WEF's of Component Codes .	82
4.3.3 Average Upper Bounds on Bit Error Probability	84
4.3.4 Interleaving Performance Gain	87
4.3.5 Effective Free Distance	90
4.4 Turbo Code Performance Evaluation	92
4.5 Turbo Code Design	98
4.5.1 Turbo Code Design at High SNR's	100
4.5.2 Turbo Code Design at Low SNR's	103
4.5.3 Simulation Results	105
4.6 Serial Concatenated Convolutional Codes	107
4.6.1 A Serial Concatenated Encoder	107
4.6.2 Performance Analysis and Code Design	108

5 Trellis Based Decoding of Linear Codes	117
5.1 Introduction	117
5.2 System Model	118
5.3 Optimization Criteria	120
5.4 The Viterbi Algorithm	122
5.5 The Bidirectional Soft Output Viterbi Algorithm	126
5.6 Sliding Window SOVA	135
5.7 The MAP Algorithm	138
5.8 The Max-Log-MAP Algorithm	149
5.9 The Log-MAP Algorithm	151
5.10 Comparison of Decoding Algorithms	152
6 Iterative Decoding	157
6.1 Optimum Decoding of Turbo Codes	157
6.2 Iterative Decoding of Turbo Codes Based on the MAP Algorithm	159
6.3 The Effect of the Number of Iterations on Turbo Code Performance	164
6.4 The Effect of Interleaver Size on Turbo Code Performance	166
6.5 The Effect of Puncturing Component Codes on Turbo Code Performance	167
6.6 Comparison Between Analytical Upper Bounds and Simulation Results	169
6.7 Asymptotic Behavior of Turbo Codes	170
6.8 Iterative SOVA Decoding of Turbo Codes	171
6.9 Comparison of MAP and SOVA Iterative Decoding Algorithms	177
6.10 Iterative MAP Decoding of Serial Concatenated Convolutional Codes	178
6.11 Iterative SOVA Decoding of Serial Concatenated Convolutional Codes	180
6.12 Serial Concatenated Convolutional Codes with Iterative Decoding	182
6.12.1 The Effect of Interleaver Size and the Number of Iterations on AWGN Channels	182

6.12.2	The Effect of Memory Order on AWGN channels	184
6.12.3	Comparison of MAP and SOVA Decoding Algorithms on AWGN Channels	186
7	Interleavers	193
7.1	Interleaving	193
7.2	Interleaving with Error Control Coding	195
7.3	Interleaving in Turbo Coding	196
7.3.1	The Effect of Interleaver Size on Code Performance	197
7.3.2	The Effect of Interleaver Structure on Code Performance	198
7.3.3	Interleaving Techniques	200
7.4	Block Type Interleavers	200
7.4.1	Block Interleavers	200
7.4.2	Odd-Even Block Interleavers	202
7.4.3	Block Helical Simile Interleavers	204
7.5	Convolutional Type Interleavers	206
7.5.1	Convolutional Interleavers	206
7.5.2	Cyclic Shift Interleavers	208
7.6	Random Type Interleavers	209
7.6.1	Random Interleavers	209
7.6.2	Non-uniform Interleavers	210
7.6.3	S -random Interleavers	211
7.7	Code Matched Interleavers	213
7.8	Design of Code Matched Interleavers	214
7.9	Performance of Turbo Codes with Code Matched Interleavers	220
7.10	Performance of Turbo Codes with Cyclic Shift Interleavers	222
8	Turbo Coding for Fading Channels	231
8.1	Introduction	231
8.2	Fading Channels	232
8.2.1	Multipath Propagation	232
8.2.2	Doppler Shift	232
8.3	Statistical Models for Fading Channels	233

8.3.1	Rayleigh Fading	233
8.3.2	Rician Fading	235
8.4	Capacity of Fading Channels	236
8.5	Performance Upper Bounds on Fading Channels . .	240
8.5.1	Upper Bounds on the Pairwise Error Probability	241
8.5.2	Average Upper Bounds on the Bit Error Probability	246
8.6	Iterative Decoding on Fading Channels	249
8.6.1	Modified MAP Decoding with CSI	252
8.6.2	Modified SOVA Decoding with CSI	254
8.7	Performance Simulation Results on Fading Channels	256
8.7.1	Performance Comparison Between MAP and SOVA Algorithms on Independent Fading Channels	256
8.7.2	Performance Comparison Between Turbo and Serial Concatenated Codes on Independent Fading Channels	256
8.7.3	Performance Comparison Between MAP and SOVA Algorithms on Correlated Fading Channels	258
8.7.4	Performance Comparison Between Turbo and Serial Concatenated Codes on Correlated Fading Channels	260
9	Turbo Trellis Coded Modulation Schemes	263
9.1	Introduction	263
9.2	Binary Turbo Coded Modulation	264
9.2.1	Pragmatic Binary Turbo Coded Modulation	264
9.2.2	Multilevel Turbo Coding	267
9.3	Turbo Trellis Coded Modulation	270
9.3.1	Schemes with Alternate Puncturing of Parity Digits	270
9.3.2	Log-MAP Decoding Algorithm for Turbo Trellis Coded Modulation with Punctured Parity Digits	274

9.3.3	SOVA Decoding Algorithm for Turbo Trellis Coded Modulation with Punctured Parity Digits	278
9.3.4	Performance of Turbo Trellis Coded Modulation with Punctured Parity Digits	279
9.3.5	Schemes with Puncturing of Systematic Bits	282
9.4	I-Q Turbo Coded Modulation for Fading Channels	288
9.4.1	I-Q Coded Modulation Structure	290
9.4.2	The Decoder	292
9.4.3	Performance of I-Q Turbo Coded Modulation on Fading Channels	292
10	Applications of Turbo Codes	297
10.1	Turbo Codes for Deep Space Communications	297
10.2	Turbo Codes for CDMA2000	299
10.3	Turbo Codes for 3GPP	300
10.4	Turbo Codes for Satellite Communications	302
Index		307

List of Acronyms

3GPP	3rd Generation Partnership Project
APP	a posteriori probability
AWGN	additive white Gaussian noise
BER	bit error rate
BPSK	binary phase shift keying
BSC	binary symmetric channel
bps	bits per second
CCSDS	Consultative Committee for Space Data Systems
CDMA	code division multiple access
CRC	cyclic redundancy check
CSI	channel state information
GCD	greatest common divisor
IOWEF	input-output weight enumerating function
IRWEF	input-redundancy weight enumerating function
ISI	intersymbol interference
LLR	log-likelihood ratio

MAP	maximum a posteriori
ML	maximum likelihood
NRC	nonrecursive convolutional
ODS	optimal distance spectrum
PCCC	parallel concatenated convolutional code
PSK	phase shift keying
QAM	quadrature amplitude modulation
RSC	recursive systematic convolutional
SCCC	serial concatenated convolutional code
SER	symbol error rate
SISO	soft-input soft-output
SNR	signal-to-noise ratio
SOVA	soft-output Viterbi algorithm
TCM	trellis coded modulation
TTCM	turbo trellis coded modulation
UEP	unequal error protection
VA	Viterbi algorithm
WEF	weight enumerating function
WER	word error rate

List of Figures

1.1	Model of a digital communication system	2
1.2	Spectral efficiency of various modulation and coding schemes computed for the bit error rate of 10^{-5}	8
2.1	Coded system model	24
2.2	Performance upper bounds for the (7,4) Hamming code	28
2.3	Trellis for the binary (5,3) code	33
2.4	Expurgated trellis for the binary (5,3) code	33
3.1	A rate 1/2 convolutional encoder	38
3.2	A general $(n, 1, \nu)$ convolutional code feedforward encoder	41
3.3	Encoder for a (3, 2, 1) code	46
3.4	The controller canonical form of a rational transfer function $\mathbf{a}(D)/\mathbf{q}(D)$	48
3.5	The observer canonical form of a rational transfer function $\mathbf{a}(D)/\mathbf{q}(D)$	49
3.6	The controller canonical form of the systematic (2, 1) encoder with the generator matrix $\mathbf{G}_1(D)$	49
3.7	The observer canonical form of the systematic (2, 1) encoder with the generator matrix $\mathbf{G}_1(D)$	50
3.8	Nonsystematic encoder in Example 3.9	55
3.9	Systematic encoder in Example 3.9	56
3.10	A systematic encoder with the generator matrix in (3.73)	57
3.11	Observer canonical form of an $(n, n - 1)$ systematic encoder	58

3.12 State diagram for the $(2, 1)$ nonsystematic convolutional encoder from Fig. 3.1	59
3.13 State diagram for the $(2, 1)$ systematic encoder in Fig. 3.7	60
3.14 Trellis diagram for the $(2, 1)$ nonsystematic encoder in Fig. 3.1	61
3.15 Augmented state diagram of Fig. 3.12	64
3.16 Trellis diagram of a rate $2/3$ punctured code produced by periodically deleting symbols from a rate $1/2$ code	67
3.17 Encoder for a rate $2/3$ code	68
3.18 Trellis diagram of a rate $2/3$ code	68
 4.1 A turbo encoder	75
4.2 A rate $1/3$ turbo encoder	77
4.3 Trellis termination	78
4.4 A rate $1/2$ turbo encoder	80
4.5 A compound error path	83
4.6 Bit error probability upper bounds for a turbo code with interleaver size 500	86
4.7 Bit error probability upper bounds for a turbo code with various interleaver sizes	91
4.8 Turbo encoder TC1	94
4.9 Turbo encoder TC2	94
4.10 Distance spectra for component code of TC1 and turbo code TC1 with interleaver sizes of 20 and 50	95
4.11 Bit error probability upper bounds for component code of TC1 and turbo code TC1 with interleaver sizes of 20 and 50	96
4.12 Relative contributions of various distance spectral lines to overall bit error probability for turbo code TC1 with interleaver size 20	97
4.13 Relative contributions of various distance spectral lines to overall bit error probability for turbo code TC1 with interleaver size 50	98
4.14 Distance spectra for turbo codes TC1 and TC2 with interleaver sizes of 20 and 50	99

4.15	Bit error probability upper bounds for turbo codes TC1 and TC2 with interleaver sizes of 20 and 50	100
4.16	Distance spectra for ODS turbo codes with inter- leaver size 40	105
4.17	Bit error probability upper bounds for ODS turbo codes with interleaver size 40	106
4.18	Performance of ODS and BM turbo codes with rate 1/3 and memory order 4 on AWGN channels	107
4.19	A serial concatenated encoder	108
5.1	System model	118
5.2	A convolutional encoder and its graphical represen- tation	127
5.3	The branch metrics in Example 5.1	128
5.4	The survivors and their path metrics in Example 5.1	128
5.5	The branch metrics in Example 5.2	133
5.6	The forward recursion in Example 5.2, the ML path is shown by the thick line	133
5.7	The backward recursion in Example 5.2, the ML path is shown by the thick line	133
5.8	Forward and Backward processing for the simplified SOVA	137
5.9	A rate 1/2 memory order 2 RSC encoder	139
5.10	State transition diagram for the (2,1,2) RSC code .	140
5.11	Trellis diagram for the (2,1,2) RSC code	141
5.12	Graphical representation of the forward recursion .	146
5.13	Graphical representation of the backward recursion	147
5.14	Trellis diagram for the encoder in Example 5.3 . . .	150
5.15	Performance comparison of MAP and SOVA	154
6.1	Basic turbo encoder	158
6.2	An iterative turbo code decoder based on the MAP algorithm	160
6.3	BER performance of a 16 state, rate 1/3 turbo code with MAP algorithm on an AWGN channel, inter- leaver size 4096 bits, variable number of iterations.	164

6.4	BER performance of a 16 state, rate 1/3 turbo code with MAP algorithm on an AWGN channel, interleaver size 16384 bits, variable number of iterations	165
6.5	BER performance of a 16 state, rate 1/3 turbo code with MAP algorithm on an AWGN channel, interleaver size N , the number of iterations 18	167
6.6	BER performance of a 16 state, rate 1/2 turbo code with MAP algorithm on an AWGN channel, interleaver size N , the number of iterations 18	168
6.7	BER performance of a 16 state, rate 2/3 turbo code with MAP algorithm on an AWGN channel, interleaver size N , the number of iterations 18	169
6.8	Simulation result of a 16 state, rate 1/3 turbo code with MAP, interleaver size 1024 bits, variable number of iterations I and the theoretical bound on an AWGN channel.	170
6.9	Simulation result of a 16 state, rate 1/3 turbo code with MAP, interleaver size 1024 bits, the number of iterations 10 and the theoretical bound on an AWGN channel.	172
6.10	An iterative turbo code decoder based on the SOVA algorithm	174
6.11	BER performance of a 16 state, rate 1/3 turbo code with MAP, Log-MAP and SOVA algorithm on an AWGN channel, interleaver size 4096 bits, the number of iterations 18	178
6.12	Iterative MAP decoder for serial concatenated codes	179
6.13	Iterative SOVA decoder for serial concatenated codes	181
6.14	Performance of a rate 1/3 serial concatenated code, with a rate 1/2, 4 state nonrecursive convolutional code as the outer code, a rate 2/3, 4 state recursive convolutional code as the inner code, AWGN channel, SOVA decoding algorithm, various interleaver size N , and the number of iterations 20	184

6.15 Comparison of a rate 1/3, memory order 2 turbo code with interleaver size 4096 bits and a rate 1/3 serial concatenated code with memory order 2 outer code, interleaver size 4096 bits on an AWGN channel, SOVA decoding algorithm, the number of iterations 18	185
6.16 BER performance of a rate 1/3 serial concatenated code with rate 1/2, 4 state outer code and rate 2/3, 4 state inner code with SOVA algorithm on an AWGN channel, interleaver size 4096 bits, variable number of iterations	186
6.17 Comparison of a rate 1/3 turbo code for different memory order with SOVA algorithm on an AWGN channel, interleaver size 1024 bits, the number of iterations 12	187
6.18 Comparison of a rate 1/3 turbo code for different memory order with SOVA algorithm on an AWGN channel, interleaver size 4096 bits, the number of iterations 18	188
6.19 Comparison of a rate 1/3 serial concatenated code for different outer code memory order with SOVA algorithm on an AWGN channel, interleaver size 1024 bits, the number of iterations 12.	188
6.20 Comparison of a rate 1/3 serial concatenated code for different outer code memory order with SOVA algorithm on an AWGN channel, interleaver size 4096 bits, the number of iterations 18	189
6.21 Performance comparison of MAP and SOVA for a rate 1/3 serial concatenated convolutional code	189
7.1 An interleaver device	194
7.2 An interleaver mapping	195
7.3 Distance spectra for a turbo code with various interleaver sizes	198
7.4 Bit error probability upper bounds for a turbo code with various interleaver sizes	199
7.5 A block interleaver	201

7.6	A weight-4 square input pattern of block interleavers	202
7.7	A convolutional interleaver and deinterleaver	207
7.8	A convolutional interleaver with $L = 3$ and $B = 2$	208
7.9	A cyclic shift interleaver	209
7.10	A general m -stage shift register with linear feedback	210
7.11	A weight-2 input sequence pattern	217
7.12	A weight-4 input sequence pattern	218
7.13	BER performance of the 4-state, rate $1/3$, $(1, 5/7)$ turbo code with random, S -random and code matched interleavers on an AWGN channel	221
7.14	BER performance of the 8-state, rate $1/3$, $(1, 17/15)$ turbo code with random, S -random and code matched interleavers on an AWGN channel	222
7.15	BER performance of the 16-state, rate $1/3$, $(1, 33/31)$ turbo code with random, S -random and code matched interleavers on an AWGN channel	223
7.16	BER performance of the 16-state, rate $1/3$, $(1, 33/31)$ turbo code with S -random and cyclic shift interleavers on an AWGN channel	224
8.1	The pdf of Rayleigh distribution	234
8.2	The pdf of Rician distributions with various K	237
8.3	Capacity of independent Rayleigh fading channels with coherent BPSK signalling	239
8.4	Coded system block diagram	241
8.5	Bit error probability upper bound for the 4 state, rate $1/3$ turbo code with interleaver size 100 on independent Rician fading channels with ideal channel state information. The curves are for Rician channels with $K=0, 2, 5, 50$, starting from the top, with the bottom one referring to an AWGN channel.	249
8.6	Bit error probability upper bound for the 4 state, rate $1/3$ turbo code with interleaver size 100 on independent Rician fading channels without channel state information. The curves are for Rician channels with $K=0, 2, 5, 50$, starting from the top, with the bottom one referring to an AWGN channel.	250

8.7	Bit error probability upper bound for the 4 state, rate 1/3 serial code with information size 100 on independent Rician fading channels with ideal channel state information. The curves are for Rician channels with $K=0, 2, 5, 50$, starting from the top, with the bottom one referring to an AWGN channel.	251
8.8	Bit error probability upper bound for the 4 state, rate 1/3 serial code with information size 100 on independent Rician fading channels without channel state information. The curves are for Rician channels with $K=0, 2, 5, 50$, starting from the top, with the bottom one referring to an AWGN channel.	252
8.9	Distance spectrum comparison of the 4 state, rate 1/3 turbo and serial concatenated codes with information size 100	253
8.10	Bit error probability upper bound comparison of the 4 state, rate 1/3 turbo and serial concatenated codes with information size 100 on independent Rayleigh fading channels	254
8.11	Performance comparison of MAP and SOVA, with and without CSI, for the 16 state, rate 1/3 turbo code on an independent Rayleigh fading channel, information size 1024, the number of iterations 8	257
8.12	Performance comparison for the 4 state, rate 1/3 turbo and serial codes on an independent Rayleigh fading channel	258
8.13	Performance comparison of MAP and SOVA, with and without CSI, for the 16 state, rate 1/3 turbo code on a correlated Rayleigh fading channel, the fading rate normalized by the symbol rate is 10^{-2} , information size 1024, the number of iterations 8	259
8.14	Performance comparison for the turbo and serial codes on a correlated Rayleigh fading channel, the fading rate normalized by the symbol rate is 10^{-2} , information size N , the number of iterations I	260
9.1	Pragmatic turbo TCM encoder	265

9.2	Pragmatic turbo TCM decoder	265
9.3	16-QAM with Gray mapping	266
9.4	Multilevel turbo encoder	268
9.5	Multilevel turbo decoder	269
9.6	Turbo TCM encoder with parity symbol puncturing	271
9.7	Example of a turbo trellis coded 8-PSK with parity symbol puncturing	273
9.8	Turbo TCM decoder with parity symbol puncturing	275
9.9	Performance of the rate 2/3, 4-state turbo trellis coded 8-PSK with various interleaver sizes on an AWGN channel, SOVA decoding algorithm, the number of iterations I , bandwidth efficiency 2 bits/s/Hz	280
9.10	Performance of the rate 2/3, 8-state turbo trellis coded 8-PSK with various interleaver sizes on an AWGN channel, SOVA decoding algorithm, the number of iterations I , bandwidth efficiency 2 bits/s/Hz	281
9.11	Performance of the rate 2/3, 16-state turbo trellis coded 8-PSK with various interleaver sizes on an AWGN channel, SOVA decoding algorithm, the number of iterations I , bandwidth efficiency 2 bits/s/Hz	282
9.12	Performance comparison of the Log-MAP and SOVA for the rate 2/3, 8-state turbo trellis coded 8-PSK with interleaver size 1024 on an AWGN channel, bandwidth efficiency 2 bits/s/Hz	283
9.13	Performance comparison of the Log-MAP and SOVA for the rate 3/4, 4-state turbo trellis coded 16-QAM with various interleaver sizes on an AWGN channel, bandwidth efficiency 3 bits/s/Hz	284
9.14	Performance comparison of the Log-MAP and SOVA for the rate 3/4, 8-state turbo trellis coded 16-QAM with various interleaver sizes on an AWGN channel, bandwidth efficiency 3 bits/s/Hz	285
9.15	Performance comparison of the Log-MAP and SOVA for the rate 3/4, 16-state turbo trellis coded 16-QAM with various interleaver sizes on an AWGN channel, bandwidth efficiency 3 bits/s/Hz	286

9.16 Turbo trellis coded 16-QAM with systematic symbol puncturing	287
9.17 Performance comparison of the turbo trellis coded 16-QAM with systematic symbol puncturing and the pragmatic turbo coded 16-QAM with bandwidth efficiency 2 bits/s/Hz and interleaver size 32768 on an AWGN channel, the number of iterations 8	288
9.18 Turbo trellis coded 8-PSK with systematic symbol puncturing	289
9.19 Performance of the turbo trellis coded 8-PSK with systematic symbol puncturing with bandwidth efficiency 2 bits/s/Hz and interleaver size 16384 on an AWGN channel, the number of iterations I	290
9.20 I-Q turbo trellis coded 16-QAM	291
9.21 Performance of the I-Q turbo coded 16-QAM with bandwidth efficiency 2 bits/s/Hz and various interleaver sizes on a Rayleigh fading channel	293
9.22 Performance comparison of the I-Q turbo coded 16-QAM and the pragmatic turbo coded 16-QAM with bandwidth efficiency 2 bits/s/Hz and interleaver size 4096 on a Rayleigh fading channel	294
10.1 CCSDS turbo encoder block diagram	298
10.2 The reverse link turbo encoder for CDMA2000	300
10.3 The turbo encoder for 3GPP	301
10.4 The serial concatenated convolutional encoder for 3GPP	303

List of Tables

2.1	A (6, 3) linear block code	15
3.1	Punctured convolutional codes	70
4.1	Best rate 1/3 turbo codes at high SNR's [14]	103
4.2	Rate 1/3 ODS turbo codes at Low SNR's	104
5.1	Decoder complexity comparison	153
8.1	Channel capacity limits for coherent BPSK	240
9.1	Rate 2/3 turbo trellis coded 8-PSK schemes	279
9.2	Rate 3/4 turbo trellis coded 16-QAM schemes	281

Preface

This book grew out of our research, industry consulting and continuing education courses.

Turbo coding initially seemed to belong to a restricted research area, while now has become a part of the mainstream telecommunication theory and practice. The turbo decoding principles have found widespread applications not only in error control, but in detection, interference suppression and equalization.

Intended for use by advanced students and professional engineers, involved in coding and telecommunication research, the book includes both basic and advanced material. The chapters are sequenced so that the knowledge is acquired in a logical and progressive way. The algorithm descriptions and analysis are supported by examples throughout the book. Performance evaluations of the presented algorithms are carried out both analytically and by simulations.

Basic material included in the book has been taught to students and practicing professionals over the last four years in the form of senior undergraduate or graduate courses, lecture series and short continuing education courses.

Most of the presented material is a compilation of the various publications from the well established literature. There are, however, original contributions, related to decoding algorithms, interleaver design, turbo coded modulation design for fading channels and performance of turbo codes on fading channels. The bidirectional SOVA decoding algorithm, presented in the book, had been developed for soft output detection and originally applied to cellular mobile receivers, but was subsequently modified for decoding of turbo codes. We have published various versions of the algorithm

at a number of conferences, but never as a journal paper, so it has not been widely known. Its distinguishing features are excellent performance, which is only slightly worse than the optimum MAP algorithm, and very low complexity, making it attractive for practical implementation. It should be of particular value to telecommunication system designers.

A great deal of effort has been put into ensuring consistency and continuity between chapters.

Special Thanks

We would like to thank everyone who has been involved in the process of writing, proof reading and publishing this book. In particular we would like to thank Dr Lei Wei, Dr Steven Pietrobon, Dr Adrian Barbulescu, Dr Miroslar Despotovic, Prof Shu Lin, and Prof Dusan Drajic for reading the manuscript and providing valuable feedback. We would also like to thank Dr Akihisa Ushirokawa for constructive discussions and Enrico Vassallo for providing the details on the CCSDS standard.

We are pleased to acknowledge the students' contribution to advancing the understanding of turbo coding. In particular, we thank Wen Feng for her work reported in Chapters 6 and 7, Jade Kim for her work reported in Chapter 6, Mark Tan for his work reported in Chapter 7 and Lei Wan for her comments on Chapters 5 and 6.

We express our appreciation to Wen Feng for providing simulation results as well as to Maree Belleli and Zhuo Chen for typing the manuscript and preparing illustrations for the book.

We owe special thanks to the Australian Research Council, NEC, DSTO, Motorola and other companies, whose support enables graduate students and the staff of Sydney University to pursue continuing research in this important field.

Alex Greene, senior editor, of Kluwer, helped and motivated us during all phases of the preparation of the book.

Finally, we would like to thank our families for providing the most meaningful content in our lives.

Chapter 1

Introduction

Novel communication and information services are being introduced almost daily and the demands for higher data rates and communication capacity continue to grow. This spectacular progress of communication is to a great extent due to consistent performance increase and cost reduction of devices and circuit technology. Such advancements have also been fostered by major theoretical developments. The synergy between components and signal processing techniques is considered to be the main cornerstone of modern communication equipment.

This book is based on two theoretical discoveries that have had a considerable impact on communication systems.

In the first one, Shannon established the fundamental limits on the transmission rates in digital communication systems and motivated the search for coding techniques to approach the capacity limit [3]. In another landmark development, Berrou, Glavieux and Thitimajshima proposed turbo error control coding by which the gap between the capacity limit and practically feasible channel utilization is almost closed [4].

The emphasis in the book is on various aspects of encoding and decoding algorithms, interleaver and code design and performance analysis.

We start by outlining the structure of digital communication systems and proceed by reviewing the channel capacity theorem and its application to various error control coding schemes. Fur-

thermore, basic principles of mathematical and trellis representation, encoding/decoding and performance analysis of block codes are presented.

1.1 Digital Communication System Structure

In order to understand the role of error control coding we present a model of a general communication system, as shown in Fig. 1.1.

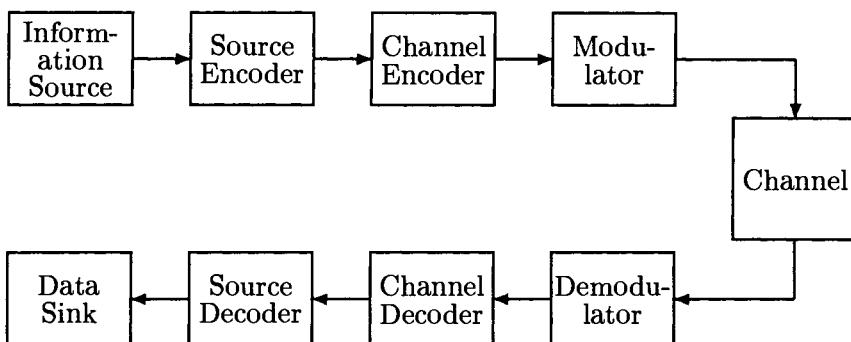


Fig. 1.1: Model of a digital communication system

The task of the transmitter in such a system is to transform the information generated by a source into a form that can withstand the effects of noise over the transmission medium.

An *information source* generates messages bearing information to be transmitted. The messages can be words, code symbols etc. The output of the information source is converted to a sequence of symbols from a certain alphabet. Most often binary symbols are transmitted.

The output of the information source is in general not suitable for transmission as it might contain too much redundancy. For efficiency reasons, the *source encoder* is designed to convert the source output sequence into a sequence of binary digits with minimum redundancy. If the source encoder generates r_b bits per second (bps), r_b is called the *data rate*.

The channel impairments cause errors in the received signal. The *channel encoder* is incorporated in the system to add redundancy to the information sequence. This redundancy is used to minimize transmission errors. The channel encoder assigns to each message of k digits a longer message of n digits called a *codeword*. A good error control code generates codewords which are as different as possible from one another. This makes the communication system less vulnerable to channel errors. Each code is characterized by the ratio $R = k/n < 1$ called the *code rate*. The data rate at the output of the channel encoder is $r_c = r_b/R$ bps. The primary goal of error control coding is to maximize the reliability of transmission within the constraints of signal power, system bandwidth and complexity of the circuitry. It is achieved by introducing structured redundancy into transmitted signals. This usually results in a lowered data transmission rate or increased channel bandwidth, relative to an uncoded system.

The output signal of the channel encoder is not normally suitable for transmission. The *modulator* enables signal transmission over a channel. The main goals of the modulation operation are to match the signal to the channel, enable simultaneous transmission of a number of signals over the same physical channel and increase the speed of information transmission.

The modulator, maps the encoded digital sequences into a train of short analog waveforms suitable for propagation. An M -ary modulator maps a block of l binary digits from the channel encoder into one of the M possible waveforms, where $M = 2^l$. The duration of the modulator output waveform is T sec, and it is referred as the *signalling interval*, while $r_s = 1/T$ is called the *symbol rate*. The minimum signal bandwidth is equal to r_s Hz, where r_s can be expressed as

$$r_s = \frac{r_b}{Rl} \quad (1.1)$$

Modulation can be performed by varying the amplitude, the phase or the frequency of a sinusoidal waveform called the *carrier*.

Channels are transmission media used to carry or store information. Channel examples are wire lines, microwave radio links over free space, satellite links, fiber optic channels, magnetic recording media etc. Very often the term channel is referred to the frequency

range allocated to a particular service such as television or a telephone channel. Two major limitations of real channels are thermal noise and finite bandwidth. In addition, mobile radio channels suffer from multipath propagation, fiber-optic cables from signal dispersion, magnetic recording media are exposed to dust and physical damage.

In the receiver, the demodulator typically generates a binary or analog sequence at its output as the best estimates of the transmitted codeword or the modulated sequence respectively. The channel decoder makes estimates of the actually transmitted message. The decoder process is based on the encoding rule and the characteristics of the channel. The goal of the decoder is to minimize the effect of channel noise.

Based on the source encoding rule the source decoder transforms the sequence at its input into an estimate of the source output sequence and delivers it to the user.

By proper design of the transmitter-receiver system it would be possible to remove or reduce the effects of attenuation and distortion and to minimize the noise effects. The impact of noise cannot be totally removed since we do not have the complete knowledge of the noise.

If the demodulator makes hard decisions, its output is a binary sequence. The subsequent channel decoding process is called *hard decision decoding*. The three blocks consisting of the modulator, the waveform channel and the demodulator can be represented by a *discrete channel*. The input and the output of the discrete channel are binary sequences at r_c bps. If the demodulator output in a given symbol interval depends only on the signal transmitted in that interval and not on any previous transmission, it is said that this channel is *memoryless*. Such a channel is described as by the set of transition probabilities $p(i | j)$ where i denotes a binary input symbol and j is a binary output symbol. The simplest channel model is obtained when the probability of error for binary symbols 0 and 1 are the same and the channel is memoryless. This channel model is known as the *binary symmetric channel* (BSC). The BSC channel is completely described by the transition probability p .

Hard decisions in the demodulator result in some irreversible

information loss. An alternative is to quantize the demodulator output to more than two levels or take samples of the analog received baseband signal and pass it to the channel decoder. The subsequent decoding process is called *soft decision decoding*.

It is important to note that in both above systems coding is performed separately from modulation/demodulation and a performance gain is obtained at the expense of increased signal bandwidth. If bandwidth efficiency is essential a more effective signal design is obtained by combining coding and modulation into a single entity. In this approach binary messages are encoded into signal sequences over a certain modulation signal set. This results in increased noise immunity of the signal without increasing the channel bandwidth. The combined coding and modulation is called *coded modulation*.

The two most frequently used types of codes are block and convolutional codes. The main difference between the two of them is memory of the encoder. In block codes each encoding operation depends on the current input message and is independent on previous encodings. That is, the encoder has no memory of history of past encodings. In contrast, for a convolutional code, each encoder output sequence depends not only on the current input message, but also on a number of past message blocks.

1.2 Fundamental Limits

The signal bandwidth is a measure of its speed. The signals that change quickly in time have large bandwidth. On the other hand, every communication system has a limited bandwidth due to capacitances and inductances which prevent instantaneous change of signals. The system bandwidth B limits the speed of signal variations. Bandwidth limitation of a system is quantified by the *spectral efficiency*, denoted by η , defined as

$$\eta = \frac{r_b}{B} \quad \text{bits/sec/Hz} \quad (1.2)$$

It can be expressed as

$$\eta = \frac{r_s l R}{B} \quad (1.3)$$

where r_s is the symbol rate. As the minimum required bandwidth for a modulated signal is r_s Hz, the maximum spectral efficiency, denoted by η_{\max} , is given by

$$\eta_{\max} = l R \quad (1.4)$$

Another important parameter used to measure the reliability of information transmission in digital communication systems is the *bit error probability*. Power efficiency is captured by the required *bit energy to one sided noise power spectral density ratio*, E_b/N_o , to achieve a specified bit error probability. The *signal-to-noise ratio* (SNR), denoted by S/N , is related to E_b/N_o as

$$\frac{S}{N} = l R \frac{E_b}{N_o} \quad (1.5)$$

For a given channel, there is an upper limit on the data rate related to the signal-to-noise ratio and the system bandwidth. Shannon introduced the concept of *channel capacity*, C , as the maximum rate at which information can be transmitted over a noisy channel. This rate is referred to as the channel capacity and for an additive white Gaussian noise (AWGN) channel it is given by the Shannon-Hartley formula

$$C = B \log_2 \left(1 + \frac{S}{N} \right) \text{ bits/sec} \quad (1.6)$$

Shannon's channel coding theorem guarantees the existence of codes that can achieve arbitrary small probability of error if the data transmission rate r_b is smaller than the channel capacity. Conversely, for a data rate $r_b > C$ it is not possible to design a code that can achieve an arbitrarily small error probability. This fundamental result shows that noise sets a limit on the data rate but not on the error probability as widely believed before. Though the theorem does not indicate how to design specific codes achieving maximum possible data rate at arbitrarily small error probabilities, it motivated the development of a number of error control techniques.

Assuming that the data rate takes its maximum possible value for error-free transmission, equal to the channel capacity C , the maximum spectral efficiency, $\eta_{\max} = \frac{C}{B}$, can be expressed as

$$\eta_{\max} = \log_2 \left(1 + R \cdot l \frac{E_b}{N_0} \right) \quad (1.7)$$

or substituting for η_{\max} from (1.4) we get

$$\eta_{\max} = \log_2 \left(1 + \eta_{\max} \frac{E_b}{N_0} \right) \quad (1.8)$$

The minimum required $\frac{E_b}{N_0}$ for error-free transmission is then given by

$$\frac{E_b}{N_0} = \frac{2^{\eta_{\max}} - 1}{\eta_{\max}} \quad (1.9)$$

If the bandwidth is not limited, it could be expanded to increase the channel capacity. In the limiting case when $B \rightarrow \infty$, or $\eta_{\max} \rightarrow 0$, we get for the minimum $\frac{E_b}{N_0}$

$$\lim_{\eta_{\max} \rightarrow 0} \frac{E_b}{N_0} = \ln 2 = -1.59 \text{ dB} \quad (1.10)$$

The minimum required $\frac{E_b}{N_0}$ for error-free transmission is -1.59 dB.

Fig. 1.2 shows the bandwidth versus power efficiency of modulation and coding for a number schemes for the bit error probability of 10^{-5} on an AWGN channel. They are shown against the Shannon spectral efficiency limit expressed by Eq. (1.9).

The uncoded BPSK modulation achieves bit error probability of 10^{-5} at $\frac{E_b}{N_0}$ of 9.5 dB and spectral efficiency of 1, while the Shannon limit for this case is $\frac{E_b}{N_0} = 10 \log_{10} 1 = 0$ dB. Thus, this scheme is about 9.5 dB worse than the Shannon limit. In other words, the required signal power could be reduced by about 9.5 dB.

The code used in satellite communications as well as in the Voyager mission is the (2, 1, 6) Odenwalder convolutional code [14]. It achieves the bit error probability of 10^{-5} at 4.5 dB with BPSK modulation and soft decision Viterbi decoder giving the spectral efficiency of 0.5 bits/sec/Hz.

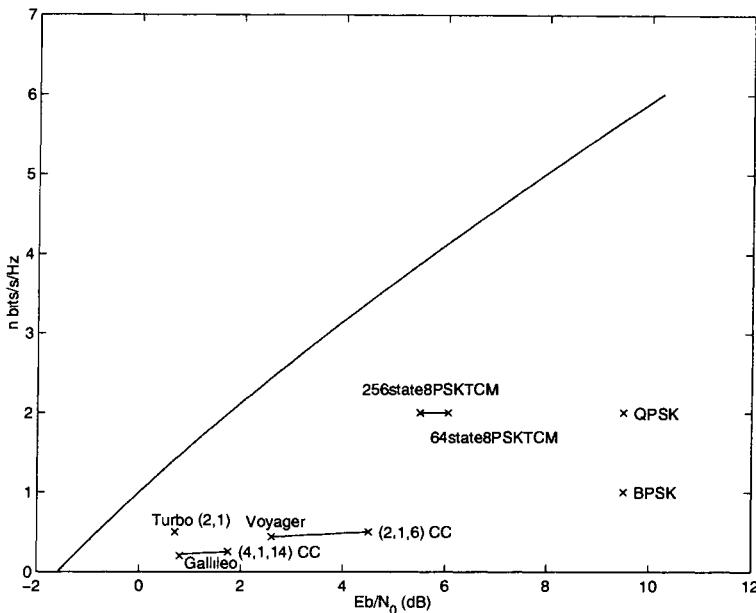


Fig. 1.2: Spectral efficiency of various modulation and coding schemes computed for the bit error rate of 10^{-5}

To improve the performance, the (2,1,6) convolutional code was concatenated with a (255,223) Reed-Solomon code in the space mission of Voyager to Uranus in 1986. The inner convolutional code was decoded by a soft decision Viterbi algorithm and the Reed-Solomon code was decoded by a Massey-Berlekamp hard decision decoder. The concatenated code can achieve a bit error rate of 10^{-5} with an E_b/N_0 ratio of 2.6 dB, with the spectral efficiency of 0.4370.

A more powerful concatenated coding scheme was chosen for the Galileo spacecraft launched in 1989 [13]. The inner (4,1,14) convolutional code [15] was serially concatenated with the (255,233) Reed-Solomon code. The inner code has the bit error rate of 10^{-5} at E_b/N_0 ratio of 1.75 dB with the spectral efficiency of 0.25 bits/sec/Hz while the concatenated coding scheme for the same error rate needs at least an $\frac{E_b}{N_0}$ of 0.8 dB with the spectral efficiency of 0.219.

Trellis coded modulation (TCM) [6] has a desirable property that coding gains are achieved without requiring more bandwidth

than the reference uncoded system with the same spectral efficiency. The asymptotic coding gains for two-dimensional TCM schemes vary from 3 to 6 dB.

For example, a 64-state 8-PSK TCM has a bit error rate of 10^{-5} at an $\frac{E_b}{N_0}$ ratio of 6.05 dB with the spectral efficiency of 2 bits/sec/Hz, gaining 3.5 dB relative to the reference uncoded QPSK, while a 256-state 8PSK TCM gains 4 dB.

Turbo codes with iterative decoding [4] have almost closed the gap between the capacity limit and real code performance. They get a bit error rate of 10^{-5} at an $\frac{E_b}{N_0}$ of 0.7 dB with the spectral efficiency of 0.5 bits/sec/Hz.

Bibliography

- [1] J. M. Wozencraft, and I. M. Jacobs, “*Principles of Communication Engineering*,” John Wiley, 1965.
- [2] R. G. Gallager, “*Information Theory and Reliable Communications*,” John Wiley, 1968.
- [3] C. E. Shannon, “A Mathematical Theory of Communication”, Bell System Technical Journal, Vol. 27, pp. 379-423 (Part One), pp. 623-656 (Part Two), Oct. 1948, reprinted in book form, University of Illinois Press, Urbana, 1949.
- [4] C. Berrou, A. Glavieux and P. Thitimajshima, “Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes”, in *Proc. 1993 Inter. Conf. Commun.*, 1993, pp. 1064-1070.
- [5] J. G. Proakis, “*Digital Communications*”, 2nd Ed. McGraw-Hill, New York, 1989.
- [6] G. Ungerboeck, “Channel Coding with Multilevel Phase Signals”, *IEEE Trans. Inform. Theory*, vol. 28, Jan. 1982, pp. 55-67.
- [7] S. Lin, and D. J. Costello, Jr., “*Error Control Coding: Fundamentals and Applications*”, Prentice-Hall, 1983.
- [8] E. R. Berlekamp, “*Algebraic Coding Theory*”, McGraw-Hill, 1968.
- [9] R. Blahut, “*Theory and Practice of Error Control Codes*”, Addison-Wesley Publishing Company, 1983.

- [10] A. J. Viterbi, and J. K. Omura, “*Principles of Digital Communication and Coding*”, McGraw-Hill, New York, 1979.
- [11] E. Biglieri, D. Divsalar, P. McLane and M. Simon, “*Introduction to Trellis-Coded Modulation*”, Macmillan Publishing Company, 1991.
- [12] C. Heegard and S. Wicker, “*Turbo Coding*”, Kluwer Academic Publishers, 1999.
- [13] L. Swanson, “A New Code for Galileo”, Abstracts 1988, *International Symposium on Information Theory*, p. 94.
- [14] J. P. Odenwalder, “Optimal Decoding of Convolutional Codes”, PhD Thesis, Systems Science Department, University of California, Los Angeles, 1970.
- [15] O. M. Collins, “The Subtleties and Intricacies of Building a Constraint Length 15 Convolutional Encoder”, *IEEE Trans. Commun.*, vol. COM-40, 1992, pp. 1810-1819.

Chapter 2

Block Codes

2.1 Block Codes

The encoder of an (n, k) block code accepts a *message* of k symbols and transforms it into a longer sequence of n symbols called a *codeword*. The important feature of a block code is that a codeword depends only on the current input message and not on the past messages. That is, the encoder is a memoryless device. In general, both messages and codewords can consist of nonbinary symbols. However, block codes with binary symbols are most often used due to implementation complexity constraints.

In an (n, k) block code there are 2^k distinct messages. Since there is one-to-one correspondence between a message and a codeword, there are 2^k distinct codewords. The code rate $R = k/n$ determines the amount of redundancy.

An (n, k) block code is linear if

- (1) component-wise modulo-2 sum of two codewords is another codeword and
- (2) the code contains the all-zero codeword.

In the language of linear algebra an (n, k) linear block code is a k -dimensional subspace of the vector space V_n of all the binary n -

tuples. An (n, k) linear block code can be generated by a set of k linearly independent binary n -tuples $\mathbf{g}_0, \mathbf{g}_1, \dots, \mathbf{g}_{k-1}$. The codewords are obtained as linear combinations of these k n -tuples.

Consequently, the codeword for a message $\mathbf{c} = (c_0, c_1, \dots, c_{k-1})$ can be represented as

$$\mathbf{v} = c_0 \cdot \mathbf{g}_0 + c_1 \cdot \mathbf{g}_1 + \cdots + c_{k-1} \cdot \mathbf{g}_{k-1}$$

The k vectors generating the code $\mathbf{g}_0, \mathbf{g}_1, \dots, \mathbf{g}_{k-1}$ can be arranged as rows of a $k \times n$ matrix as follows

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_{k-1} \end{bmatrix} = \begin{bmatrix} g_{00} & g_{01} & \cdots & g_{0,n-1} \\ g_{10} & g_{11} & \cdots & g_{1,n-1} \\ \vdots & \vdots & \vdots & \vdots \\ g_{k-1,0} & g_{k-1,1} & \cdots & g_{k-1,n-1} \end{bmatrix}$$

The array \mathbf{G} is called the *generator matrix* of the code. Then, the codeword \mathbf{v} for message \mathbf{c} can be written as

$$\mathbf{v} = \mathbf{c} \cdot \mathbf{G} = c_0 \cdot \mathbf{g}_0 + c_1 \cdot \mathbf{g}_1 + \cdots + c_{k-1} \cdot \mathbf{g}_{k-1}$$

Example 2.1 An $(6, 3)$ linear block code can be generated by the generator matrix

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \mathbf{g}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The message $\mathbf{c} = (0\ 1\ 1)$ is encoded as follows:

$$\begin{aligned} \mathbf{v} &= \mathbf{c} \cdot \mathbf{G} \\ &= 0 \cdot (011100) + 1 \cdot (101010) + 1 \cdot (110001) \\ &= (000000) + (101010) + (110001) \\ &= (011011) \end{aligned}$$

Table 2.1 presents the list of messages and codewords for the $(6, 3)$ linear block code.

Table 2.1: A (6, 3) linear block code

Messages (c_0, c_1, c_2)	Codewords ($v_0, v_1, v_2, v_3, v_4, v_5$)
(0 0 0)	(0 0 0 0 0 0)
(1 0 0)	(0 1 1 1 0 0)
(0 1 0)	(1 0 1 0 1 0)
(1 1 0)	(1 1 0 1 1 0)
(0 0 1)	(1 1 0 0 0 1)
(1 0 1)	(1 0 1 1 0 1)
(0 1 1)	(0 1 1 0 1 1)
(1 1 1)	(0 0 0 1 1 1)

2.2 Linear Systematic Block Codes

A linear *systematic* block code has the additional structure that the message sequence is itself part of the codeword. In addition to the k -digit message sequence the codeword contains the $(n - k)$ -digit parity check sequence. This format allows direct extraction of the message from the codeword. The systematic format is shown below

$$\overbrace{(c_0, c_1, \dots, c_{k-1})}^{\text{message}} \longrightarrow \underbrace{v_0, v_1, \dots, v_{n-k-1}}_{\text{parity-check}}, \underbrace{c_0, c_1, \dots, c_{k-1}}_{\text{message}}$$

The generator matrix for a systematic block code has the following form

$$\mathbf{G} = [\mathbf{P} \quad \mathbf{I}_k]$$

\mathbf{I}_k is the $k \times k$ identity matrix and \mathbf{P} is a $k \times (n - k)$ matrix of the form

$$\mathbf{P} = \begin{bmatrix} p_{00} & p_{01} & \cdots & p_{0,n-k-1} \\ p_{10} & p_{11} & \cdots & p_{1,n-k-1} \\ p_{20} & p_{21} & \cdots & p_{2,n-k-1} \\ \vdots & \vdots & \ddots & \vdots \\ p_{k-1,0} & p_{k-1,1} & \cdots & p_{k-1,n-k-1} \end{bmatrix}$$

where $p_{ij} = 0$ or 1.

Example 2.2 The (6,3) code given by Table 2.1 is a systematic code since its generator matrix can be represented in the form

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \mathbf{g}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} = [\mathbf{P} \quad \mathbf{I}_3]$$

2.3 Parity Check Matrix

Let us consider a systematic linear block code specified by its generator matrix in the form

$$\mathbf{G} = [\mathbf{P} \quad \mathbf{I}_k]$$

Given a message sequence \mathbf{c} the codeword is obtained as

$$\mathbf{v} = \mathbf{c} \cdot \mathbf{G}$$

Since the code is systematic the codeword can be written as

$$\mathbf{v} = [\mathbf{a} \quad \mathbf{c}]$$

where

$$\mathbf{a} = \mathbf{c} \cdot \mathbf{P} \tag{2.1}$$

is the parity check consisting of $(n - k)$ digits. Note that $\mathbf{a} + \mathbf{a} = \mathbf{0}$ and combining with (2.1),

$$\mathbf{a} + \mathbf{c} \cdot \mathbf{P} = \mathbf{0} \tag{2.2}$$

The above equation can be rewritten as

$$[\mathbf{a} \quad \mathbf{c}] \cdot \begin{bmatrix} \mathbf{I}_{n-k} \\ \mathbf{P} \end{bmatrix} = \mathbf{0} \tag{2.3}$$

where \mathbf{I}_{n-k} is the $(n - k)$ identity matrix. (2.3) can be written in a more compact form as

$$\mathbf{v} \cdot \mathbf{H}^T = \mathbf{0} \tag{2.4}$$

where

$$\mathbf{H} = [\mathbf{I}_{n-k} \quad \mathbf{P}^T]$$

The $(n - k) \times n$ matrix \mathbf{H} is called the *parity check matrix*. It is used as an another way to specify an $(n - k)$ linear block code.

The parity check matrix is used in error detection to test whether a received vector is a codeword by checking (2.4).

2.4 The Minimum Distance of a Block Code

Let us consider an (n, k) linear block code. The *Hamming weight* (or weight) of a codeword \mathbf{v} is defined as the total number of nonzero components in the codeword. The *Hamming distance*, $d(\mathbf{v}, \mathbf{u})$, between two codewords \mathbf{v} and \mathbf{u} is defined as the number of places in which these two codewords differ. For example, if the two code words are $\mathbf{v} = (100111)$ and $\mathbf{u} = (010001)$ the Hamming distance between them is 4. The Hamming distance between two codewords is identical to the weight of their modulo-2 sum. The *minimum Hamming distance* (or minimum distance) of a code, d_{\min} , is defined as the smallest Hamming distance between two different codewords in the code. The linearity property of block codes requires that the modulo-2 sum of two codewords is another codeword. This implies that the minimum distance of a linear block code is the smallest weight of the nonzero codewords in the code. The importance of the minimum distance parameter is in the fact that it determines error correcting and detecting capability of a code.

Example 2.3 Determine the minimum distance of the $(6,3)$ code given in Example 2.1. A linear block code is defined by the set of codewords that satisfy

$$\mathbf{v} \cdot \mathbf{H}^T = \mathbf{0}$$

Let us express the parity check matrix in the form

$$\mathbf{H} = [\mathbf{h}_0, \mathbf{h}_1, \dots, \mathbf{h}_i, \dots, \mathbf{h}_{n-1}]$$

where \mathbf{h}_i represents the i th column of \mathbf{H} . Then the code can be specified as

$$v_0 \cdot \mathbf{h}_0 + v_1 \cdot \mathbf{h}_1 + \dots + v_i \cdot \mathbf{h}_i + \dots + v_{n-1} \cdot \mathbf{h}_{n-1} = \mathbf{0}$$

where v_i is the i th component of the codeword \mathbf{v} . To satisfy this equation codeword \mathbf{v} must have nonzero components in such positions that corresponding columns of \mathbf{H} sum to the zero vector $\mathbf{0}$.

On the other hand, the minimum distance of the code is the smallest number of nonzero components in a codeword. Therefore, the minimum distance of a linear block code is equal to the smallest number of columns of the matrix \mathbf{H} that sum to $\mathbf{0}$. The parity check matrix \mathbf{H} for the $(6,3)$ code is given by

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

All columns in \mathbf{H} are nonzero and no two of them are the same. Hence, no two or less columns sum to $\mathbf{0}$. Hence, the minimum distance of the code is 3.

2.5 Maximum Likelihood Decoding of Block Codes for a BSC Channel

For a BSC channel the demodulator makes hard decisions and the received sequence \mathbf{r} is a binary sequence. The *maximum likelihood decoding* (MLD) rule for a hard decision decoder over a BSC channel can be stated as follows. For each codeword \mathbf{v} in an (n, k) block code compute the conditional probability

$$P(\mathbf{r} | \mathbf{v})$$

The codeword with the maximum conditional probability is chosen as the estimate of the actually transmitted codeword \mathbf{v} . For a BSC channel with the transition probability p the conditional probability is given by

$$P(\mathbf{r} | \mathbf{v}) = p^{d(\mathbf{r}, \mathbf{v})} (1 - p)^{n-d(\mathbf{r}, \mathbf{v})}$$

where $d(\mathbf{r}, \mathbf{v})$ is the Hamming distance between the vectors \mathbf{r} and \mathbf{v} .

Since for $p < 1/2$, $P(\mathbf{r} | \mathbf{v})$ is a monotonically decreasing function of $d(\mathbf{r}, \mathbf{v})$, we will have

$$P(\mathbf{r} | \mathbf{v}) > P(\mathbf{r} | \mathbf{w})$$

if and only if

$$d(\mathbf{r}, \mathbf{v}) < d(\mathbf{r}, \mathbf{w})$$

Hence the MLD rule for a BSC channel can be obtained by comparing the received vector \mathbf{r} with all codewords and selecting the codeword that has the smallest Hamming distance from \mathbf{r} as the estimate of the actually transmitted codeword. This decoding rule is known as the *minimum distance decoding*. The MLD rule results in the minimum block error probability.

2.6 Maximum Likelihood Decoding of Block Codes for a Gaussian Channel

Let us consider a communication system with an (n, k) block code and binary antipodal modulation. That is, the modulator maps binary symbols v_i to the modulated signals x_i according to the rule

$$x_i = \begin{cases} +1 & v_i = 1 \\ -1 & v_i = 0 \end{cases}$$

where v_i is the i th component in codeword \mathbf{v} . We assume that the signal in the channel is corrupted by additive white Gaussian noise. The i th component of the received sequence r_i is given by

$$r_i = x_i + n_i$$

where n_i is a sample of a Gaussian noise with zero mean and variance σ^2 . The demodulator takes samples of the received sequence

$$\mathbf{r} = (r_0, r_1, \dots, r_i, \dots, r_{n-1})$$

and presents it to the decoder. Since the demodulator does not produce binary outputs the decoder will make soft decision decoding. The maximum likelihood soft decision decoder computes the conditional probability

$$P(\mathbf{r} | \mathbf{v})$$

for each codeword \mathbf{v} and selects the codeword with the largest conditional probability as the estimate of the actually transmitted codeword.

The conditional probability $P(\mathbf{r} | \mathbf{v})$ for a channel with noise affecting each sample being independent Gaussian random variable with zero mean and variance σ^2 is given by

$$\begin{aligned} P(\mathbf{r} | \mathbf{v}) &= \prod_{i=0}^{n-1} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(r_i - x_i)^2}{2\sigma^2}} \\ &= \left(\frac{1}{\sqrt{2\pi\sigma^2}} \right)^n e^{-\sum_{i=0}^{n-1} \frac{(r_i - x_i)^2}{2\sigma^2}} \end{aligned}$$

where x_i is the i th component of the sequence \mathbf{x} obtained by modulating codeword \mathbf{v} . The above expression is a monotonically decreasing function of $\sum_{i=0}^{n-1} (r_i - x_i)^2$ and it is maximized when

$$d_E^2 = \sum_{i=0}^{n-1} (r_i - x_i)^2$$

is minimized. The sum represents the Euclidean distance between the received sample sequence and the modulated codeword. Hence, the MLD rule for a Gaussian channel can be accomplished as follows. The decoder computes the Euclidean distance between the received sample sequence and all modulated codewords and selects a codeword with the minimum Euclidean distance as the estimate of the actually transmitted codeword. However, this method becomes impractical for long codes.

2.7 Weight Distribution of Block Codes

The weight distribution of a block code is useful in computing probabilities of undetected or uncorrected errors. In this section we will consider the weight distribution of a block code. Subsequently, we will apply it in the calculation of word and bit error probability bounds for a block code on an additive white Gaussian noise channel.

Given an (n, k) linear systematic block code, its weight distribution can be expressed by the code *weight enumerating function*

(WEF) [14]. The WEF of a code is given by

$$A(X) = \sum_{i=0}^n A_i X^i \quad (2.5)$$

where A_i is the number of codewords of Hamming weight i and X is a dummy variable. The set

$$\{A_{d_{\min}}, A_{d_{\min}+1}, \dots, A_i, \dots, A_n\}$$

is called the *weight distribution* or the *weight spectrum* of the code.

There is a special class of codes with a binomial weight distribution. These codes are known as *perfect codes*.

The WEF describes the code weight distribution which only depends on the codewords. It does not give the contributions of the input information bits to the total Hamming codeword weight. A more detailed weight profile of a code is described by the *input-output weight enumerating function* (IOWEF) defined as

$$A(W, X) = \sum_{w,x} A_{w,x} W^w X^x \quad (2.6)$$

where $A_{w,x}$ is the number of codewords of weight x generated by input information of weight w . For systematic block codes, we can separate codeword weights into input information weight and parity check information weight and define the *input-redundancy weight enumerating function* (IRWEF) of a code as

$$A(W, Z) = \sum_{w,z} A_{w,z} W^w Z^z \quad (2.7)$$

where $A_{w,z}$ is the number of the codewords of the block code with input information weight w and parity check information weight z . The overall Hamming weight of a codeword is $d = w + z$. The relationship between the WEF coefficients A_i , the IOWEF coefficients $A_{w,x}$ and the IRWEF coefficients $A_{w,z}$ is given by

$$A_i = \sum_{w+z=i} A_{w,z}, \quad A_i = \sum_{x=i} A_{w,x}, \quad A_{w,z} = A_{w,x} \Big|_{z=x-w} \quad (2.8)$$

Furthermore, the IRWEF can be decomposed according to the contributions of distinct input information weights w , as

$$A(W, Z) = \sum_w W^w A_w(Z) \quad (2.9)$$

where $A_w(Z)$ is called the *conditional weight enumerating function*. The conditional WEF describes the parity check weights of the codewords generated by input information of weight w . It is given by

$$A_w(Z) = \sum_z A_{w,z} Z^z \quad (2.10)$$

The relationship between the IRWEF and the conditional WEF can be rewritten as

$$A_w(Z) = \frac{1}{w!} \left. \frac{\partial^w A(W, Z)}{\partial W^w} \right|_{W=0} \quad (2.11)$$

Example 2.4 Weight enumerating functions for a $(7, 4)$ Hamming code

Let us consider a systematic $(7, 4)$ Hamming code with generator matrix

$$\mathbf{G}(D) = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \quad (2.12)$$

The WEF of the code is

$$A(X) = 1 + 7X^3 + 7X^4 + X^7 \quad (2.13)$$

The IOWEF of the code is

$$\begin{aligned} A(W, X) = & 1 + 3WX^3 + WX^4 + 3W^2X^3 + 3W^2X^4 \\ & + W^3X^3 + 3W^3X^4 + W^4X^7 \end{aligned} \quad (2.14)$$

Separating the codeword weights into the input information weight and the parity check information weight, we get for the IRWEF

$$\begin{aligned} A(W, Z) = & 1 + W(3Z^2 + Z^3) + W^2(3Z + 3Z^2) \\ & + W^3(1 + 3Z) + W^4Z^3 \end{aligned} \quad (2.15)$$

The conditional WEF's of the code are

$$\begin{aligned} A_0(Z) &= 1 \\ A_1(Z) &= 3Z^2 + Z^3 \\ A_2(Z) &= 3Z + 3Z^2 \\ A_3(Z) &= 1 + 3Z \\ A_4(Z) &= Z^3 \end{aligned}$$

2.8 Performance Upper Bounds

Weight enumerating functions can be used to calculate code error performance. In particular, the WEF specifies the weight distribution of codewords which determines the word error probability, while both IRWEF and conditional WEF's characterize the input/output code weight profile, which can provide information on bit error probability.

2.8.1 Word Error Probability Upper Bounds

In this performance analysis, we assume binary phase shift keying (BPSK) modulation is used to map each binary symbol into a signal from the $\{-1, 1\}$ modulation signal set. The modulated signals are transmitted over an AWGN channel. The system model is shown in Fig. 2.1.

Let a coded sequence be $\mathbf{v} = (v_0, v_1, \dots, v_{n-1})$, where n is the code length. The modulated sequence is $\mathbf{x} = (x_0, x_1, \dots, x_{n-1})$. At the receiver, the received sequence $\mathbf{r} = (r_0, r_1, \dots, r_{n-1})$ is observed. The received signal at time i can be represented by

$$r_i = x_i + n_i \quad (2.16)$$

where n_i is a noise sample with a zero mean and variance σ^2 . The noise sequence is $\mathbf{n} = (n_0, n_1, \dots, n_{n-1})$. The decoder performs a maximum likelihood sequence decoding which maximizes the probability $P(\mathbf{r} | \mathbf{v})$.

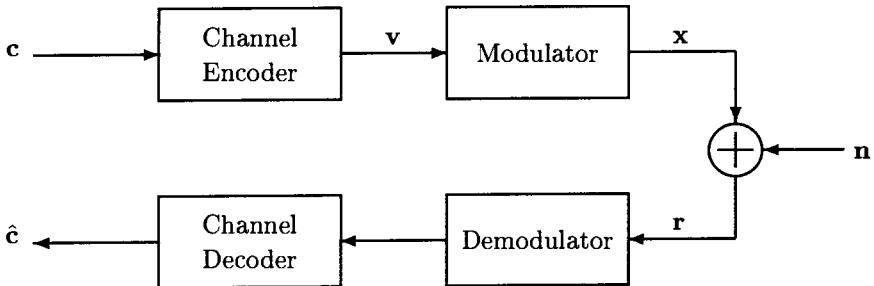


Fig. 2.1: Coded system model

If errors occur in transmission the decoder might choose another code sequence as the decoded sequence. That is, an error event occurs whenever the received sequence is closer in Euclidean distance to another code sequence \hat{v} . The sequence \hat{v} is called an error sequence. The probability that the decoder makes a wrong decision by selecting an error sequence is called the *error event probability* or *pairwise error probability*.

Let the transmitted sequence associated with the code sequence \hat{v} be \hat{x} . The pairwise error probability, denoted by P_d , can be given by

$$\begin{aligned} P_d &= P_r \left\{ \sum_{i=0}^{n-1} |r_i - x_i|^2 \geq \sum_{i=0}^{n-1} |r_i - \hat{x}_i|^2 \right\} \\ &= P_r \left\{ \sum_{i=0}^{n-1} n_i (\hat{x}_i - x_i) \geq 2d \right\} \end{aligned} \quad (2.17)$$

where d is the Hamming distance between the pair codeword sequences. In this analysis, we will introduce a random variable

$$A = \sum_{i=0}^{n-1} n_i (\hat{x}_i - x_i) \quad (2.18)$$

Since n_i is a zero mean Gaussian noise sample with variance σ^2 , the random variable A is a Gaussian variable with zero mean and variance $\sigma_A^2 = 4d\sigma^2$. Thus the pairwise error probability can be

expressed as [3]

$$P_d = Q\left(\sqrt{2dR\frac{E_b}{N_0}}\right) \quad (2.19)$$

where $R = k/n$ is the code rate, E_b is the signal energy per bit, N_0 is the single sided power spectral density of the Gaussian noise and $Q(x)$ is the complementary error function defined by

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-\frac{t^2}{2}} dt \quad (2.20)$$

The word error probability of a code on AWGN channels can be upper-bounded by a union bound that sums contributions from all error events with various Hamming distances. The word error probability union bound is given by

$$\begin{aligned} P_e &\leq \sum_{d=d_{\min}} A_d P_d \\ &= \sum_{d=d_{\min}} A_d Q\left(\sqrt{2dR\frac{E_b}{N_0}}\right) \end{aligned} \quad (2.21)$$

where d_{\min} is the code *minimum Hamming distance*, A_d is the number of error events with Hamming distance d , called the error coefficient.

For linear codes, we may assume the all-zero code sequence is transmitted. The error coefficient A_d will be the number of codewords with weight d . It can be obtained from the code WEF $A(X)$.

Considering the Q -function bound

$$Q(x) \leq \frac{1}{2} e^{-\frac{x^2}{2}}, \quad x \geq 0 \quad (2.22)$$

in (2.21), we obtain an upper bound for the word error probability

$$\begin{aligned} P_e &\leq \sum_{d=d_{\min}} \frac{1}{2} A_d e^{-dR\frac{E_b}{N_0}} \\ &= \frac{1}{2} [A(X) - 1] \Big|_{X=e^{-R\frac{E_b}{N_0}}} \end{aligned} \quad (2.23)$$

As an alternative to (2.22), making use of the inequality

$$Q\left(\sqrt{x+y}\right) \leq Q\left(\sqrt{x}\right)e^{-\frac{y}{2}}, \quad x, y \geq 0 \quad (2.24)$$

a tighter upper bound can be obtained from (2.21) as

$$\begin{aligned} P_e &\leq Q\left(\sqrt{2d_{\min}R\frac{E_b}{N_0}}\right)e^{d_{\min}R\frac{E_b}{N_0}} \sum_{d=d_{\min}} A_d e^{-dR\frac{E_b}{N_0}} \\ &= Q\left(\sqrt{2d_{\min}R\frac{E_b}{N_0}}\right)e^{d_{\min}R\frac{E_b}{N_0}} [A(X) - 1] \Big|_{X=e^{-R\frac{E_b}{N_0}}} \end{aligned} \quad (2.25)$$

2.8.2 Bit Error Probability Upper Bounds

The bit error probability of an (n, k) block code decoded by a maximum likelihood algorithm over an AWGN channel can be upper-bounded by a union bound as

$$\begin{aligned} P_b(e) &\leq \sum_{d=d_{\min}} B_d P_d \\ &= \sum_{d=d_{\min}} B_d Q\left(\sqrt{2dR\frac{E_b}{N_0}}\right) \end{aligned} \quad (2.26)$$

where B_d is the error coefficient, i.e., the average number of bit errors caused by transitions between the all-zero codeword and codewords of weight d ($d \geq d_{\min}$). The error coefficient B_d can be obtained from the code IRWEF. It is given by

$$B_d = \sum_{d=w+z} \frac{w}{k} \cdot A_{w,z} \quad (2.27)$$

B_d determines the contribution of the codewords with the same weight d to the bit error probability. The set of all pairs of (d, B_d) , denoted by

$$\{(d_{\min}, B_{d_{\min}}), (d_{\min} + 1, B_{d_{\min}+1}), \dots, (d_i, B_{d_i}), \dots\}, \quad (2.28)$$

is called the *code distance spectrum*.

Considering the Q -function bound in (2.22), the bit error probability upper bound can be given by

$$\begin{aligned}
 P_b(e) &\leq \sum_{d=d_{\min}} \frac{1}{2} B_d e^{-dR \frac{E_b}{N_0}} \\
 &= \sum_{w=1}^k \frac{w}{2k} W^w A_w(Z) \Big|_{W=Z=e^{-R \frac{E_b}{N_0}}} \\
 &= \frac{W}{2k} \frac{\partial A(W, Z)}{\partial W} \Big|_{W=Z=e^{-R \frac{E_b}{N_0}}} \tag{2.29}
 \end{aligned}$$

In addition, a tighter bit error probability upper bound can be expressed as

$$\begin{aligned}
 P_b(e) &\leq Q \left(\sqrt{2d_{\min} R \frac{E_b}{N_0}} \right) e^{d_{\min} R \frac{E_b}{N_0}} \sum_{d=d_{\min}} B_d e^{-dR \frac{E_b}{N_0}} \tag{2.30} \\
 &= Q \left(\sqrt{2d_{\min} R \frac{E_b}{N_0}} \right) e^{d_{\min} R \frac{E_b}{N_0}} \sum_{w=1}^k \frac{w}{k} W^w A_w(Z) \Big|_{W=Z=e^{-R \frac{E_b}{N_0}}} \\
 &= Q \left(\sqrt{2d_{\min} R \frac{E_b}{N_0}} \right) e^{d_{\min} R \frac{E_b}{N_0}} \frac{W}{k} \frac{\partial A(W, Z)}{\partial W} \Big|_{W=Z=e^{-R \frac{E_b}{N_0}}}
 \end{aligned}$$

Example 2.5 Error performance upper bounds for the $(7, 4)$ Hamming code from Example 2.4

The word error probability upper bound of the code is

$$\begin{aligned}
 P_e &\leq \sum_{d=d_{\min}} \frac{1}{2} A_d e^{-dR \frac{E_b}{N_0}} \\
 &= \frac{7}{2} e^{-3R \frac{E_b}{N_0}} + \frac{7}{2} e^{-4R \frac{E_b}{N_0}} + \frac{1}{2} e^{-7R \frac{E_b}{N_0}} \tag{2.31}
 \end{aligned}$$

By referring to (2.27) and (2.14), the code error coefficients B_d can be computed as

$$\begin{aligned}
 d = 3 &\quad B_d = \frac{1}{4} \times 3 + \frac{2}{4} \times 3 + \frac{3}{4} \times 1 = 3 \\
 d = 4 &\quad B_d = \frac{1}{4} \times 1 + \frac{2}{4} \times 3 + \frac{3}{4} \times 3 = 4 \\
 d = 7 &\quad B_d = \frac{4}{4} \times 1 = 1 \tag{2.32}
 \end{aligned}$$

The bit error probability upper bound is

$$\begin{aligned} P_b(e) &\leq \sum_{d=d_{\min}} \frac{1}{2} B_d e^{-dR \frac{E_b}{N_0}} \\ &= \frac{3}{2} e^{-3R \frac{E_b}{N_0}} + 2e^{-4R \frac{E_b}{N_0}} + \frac{1}{2} e^{-7R \frac{E_b}{N_0}} \end{aligned} \quad (2.33)$$

The word error probability and bit error probability upper bounds are shown in Fig. 2.2.

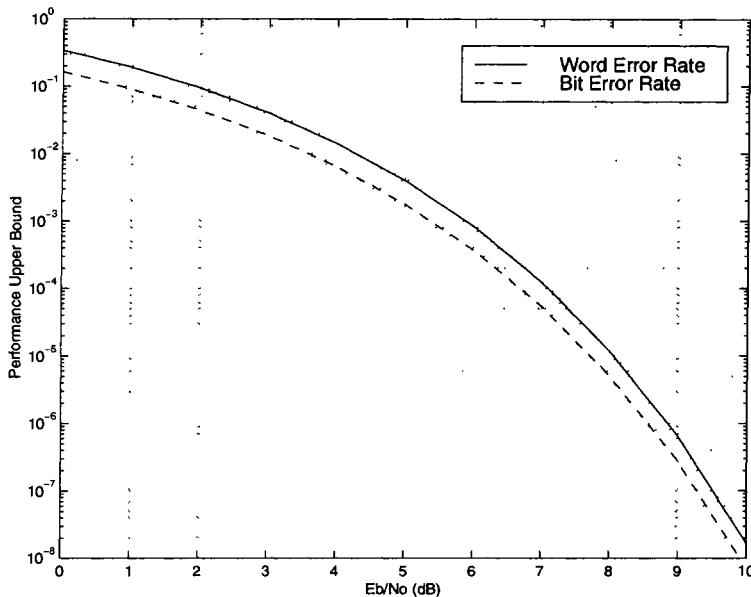


Fig. 2.2: Performance upper bounds for the (7,4) Hamming code

2.9 Coding Gain

Coded systems typically require a less SNR than uncoded systems to achieve the same output bit error probability. This reduction, expressed in decibels, in the required SNR to achieve a specific bit error probability, of a coded system over an uncoded system, is called *coding gain*. The *asymptotic coding gain* of a coded system

with soft decision decoding over an uncoded reference system, if the two systems have the same bandwidth, is defined as

$$G = 10 \log_{10} \frac{d_{E,\min}^2}{d_u^2} \quad \text{dB} \quad (2.34)$$

where d_u is the Euclidean distance of the uncoded system and $d_{E,\min}$ is the minimum Euclidean distance between all modulated codewords. At high SNR's most likely errors that the decoder will make will be coming from the modulated codewords at the Euclidean distance $d_{E,\min}$ from the correct modulated codeword.

However, coded systems often require a higher bandwidth than uncoded systems resulting in different noise variances. The asymptotic coding gain in this case will be

$$G = 10 \log_{10} \frac{d_{E,\min}^2 \sigma_u^2}{d_u^2 \sigma_c^2} \quad (2.35)$$

where σ_c^2 and σ_u^2 are the variances of Gaussian noise for the coded and uncoded systems, respectively.

Example 2.6 Compute the asymptotic coding gain for the (7,4) block code with the minimum Hamming distance of 3.

The minimum Euclidean distance of the uncoded system with binary antipodal modulation is $d_u = 2$. The minimum Euclidean distance of the code combined with binary antipodal modulation is $d_{E,\min} = 2\sqrt{3}$. The coded system with the (7,4) code requires a bandwidth $7/4$ times higher than the bandwidth of the uncoded system. Therefore, the variance of the white additive Gaussian noise of the coded system σ_c^2 is related to the variance of the same noise in the uncoded system σ_u^2 by

$$\sigma_c^2 = 7/4 \cdot \sigma_u^2$$

The asymptotic coding gain is then

$$G = 10 \log_{10} \frac{12}{4} \frac{4}{7} = 2.34 \text{dB}$$

2.10 Soft Decision Decoding of Block Codes

We will present here a look-up table algorithm that performs soft decision maximum likelihood decoding for block codes.

Let us consider an (n, k) binary block code. Assume that the transmitted codeword is \mathbf{v} . In the modulator each binary symbol is mapped into one of two antipodal signals $\{-1, 1\}$. At the receiver, the demodulator only samples the received signal, presenting a sequence of sampled symbols \mathbf{r} to the decoder, where

$$\mathbf{r} = (r_0, r_1, \dots, r_i, \dots, r_{n-1})$$

This algorithm is based on the assumption that for a Gaussian channel most likely error patterns will be closest in the Euclidean distance sense to the received sequence.

The decoding operations can be summarized as follows.

1. Compute Euclidean distances between the received sample sequence \mathbf{r} and each codeword from the code set $\{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_{2^k}\}$, which is stored in a look-up table.
2. Choose a codeword \mathbf{v}_j which is closest to the received sequence \mathbf{r} in Euclidean distance as the estimate of the transmitted codeword.

This algorithm can be used for relatively short block codes only, as its complexity increases with the message length.

2.11 Trellis Structure of Linear Binary Block Codes

In 1974 Bahl, Cocke, Jelinek and Raviv [8] presented a method for representing codewords of a linear block code by a trellis. Later, Wolf [9] introduced an identical trellis and applied the Viterbi algorithm for maximum likelihood decoding of block codes. In 1988 Forney [10] generated a trellis which minimizes the number of nodes at each depth. For this reason this trellis is known as the minimal trellis.

Consider a linear (n, k) code C over a binary field $GF(2)$, with a parity check matrix \mathbf{H}

$$\mathbf{H} = [\mathbf{h}_0, \mathbf{h}_1, \dots, \mathbf{h}_i, \dots, \mathbf{h}_{n-1}] \quad (2.36)$$

where \mathbf{h}_i is an $(n - k) \times 1$ column vector.

Note that the analysis can be easily generalized to include non-binary codes, defined over a finite field $GF(q)$.

The parity check matrix \mathbf{H} characterizes the code C . The code C then consists of all vectors

$$\mathbf{v} = (v_0, v_1, \dots, v_i, \dots, v_{n-1}), \quad v_i \in \{0, 1\} \quad (2.37)$$

such that

$$\mathbf{v}\dot{\mathbf{H}}^T = v_0\mathbf{h}_0 + v_1\mathbf{h}_1 + \dots + v_{n-1}\mathbf{h}_{n-1} = \mathbf{0} \quad (2.38)$$

The code C can also be described by a *trellis*. A trellis is a set of nodes interconnected by unidirectional branches. Every node is assigned an integer l , which is referred to as the *depth* of the trellis, and an $(n - k)$ -tuple, called a *state*, denoted by $S_i(l)$, for a certain integer i . So there will be at most 2^{n-k} states. They are all ordered from 0 to $2^{n-k} - 1$, with 0 referring to the all-zero $(n - k)$ -tuple.

There is only one node at depth 0, denoted by $S_0(0)$ and only one node at depth n , denoted by $S_0(n)$. A *path* in the trellis of length L is a sequence of L branches. Let us denote by I_l the set of node subscripts at depth l , which represents a subset of the integers $\{0, 1, 2, \dots, 2^{n-k} - 1\}$. The trellis for the linear block code C is constructed as follows.

At depth $l = 0$, the trellis contains only one node, which is the all-zero $(n - k)$ -tuple $S_0(0)$.

For each $l = 0, 1, \dots, (n - 1)$ the set of states at depth $(l + 1)$ is obtained from the set of nodes at depth l by the formula

$$S_m(l + 1) = S_i(l) + \alpha_i^j \mathbf{h}_{l+1} \quad (2.39)$$

for all $i \in I_l$, $m \in I_{l+1}$, and $j \in \{0, 1\}$, where α_i^j are binary inputs

$$\alpha_i^j = \begin{cases} 1 & j = 1 \\ 0 & j = 0 \end{cases} \quad (2.40)$$

Therefore, for each i in I_l , connecting branches are formed between node $S_i(l)$ and two nodes at depth $(l + 1)$, for two different binary values $\alpha_i^j \in \{0, 1\}$, according to the above formula. Each branch is labelled by the corresponding value α_i^j .

At depth $l = n$, the final node, $S_0(n)$, is given by

$$S_0(n) = S_0(0) + \sum_{i=0}^{n-1} \alpha_i^j \mathbf{h}_i = 0 \quad (2.41)$$

As both $S_0(n)$ and $S_0(0)$ are the all-zero states, it implies

$$\sum_{i=0}^{n-1} \alpha_i^j \mathbf{h}_i = 0 \quad (2.42)$$

As any codeword satisfies Eq. (2.38), which is identical to Eq. (2.42), it implies that only those α_i^j which constitute a codeword will form paths in the trellis ending in the final all-zero state. Thus every valid path in the trellis corresponds to a codeword.

By the construction method in Eq. (2.39), all possible binary n -tuples, are formed in the trellis. There are total 2^n of them. They include 2^k valid codewords. Thus, every codeword in C is represented in the trellis and it corresponds to a path in the trellis.

Finally, we remove any nodes that do not have a path to the final all-zero state at depth n and all branches leading to these expurgated nodes.

Example 2.7 We illustrate the trellis construction for the binary $(5, 3)$ code with the parity check matrix

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix} = [\mathbf{h}_0 \ \mathbf{h}_1 \ \mathbf{h}_2 \ \mathbf{h}_3 \ \mathbf{h}_4] \quad (2.43)$$

Following the described procedure we obtain the trellises before and after expurgation, as shown in Fig. 2.3 and 2.4.

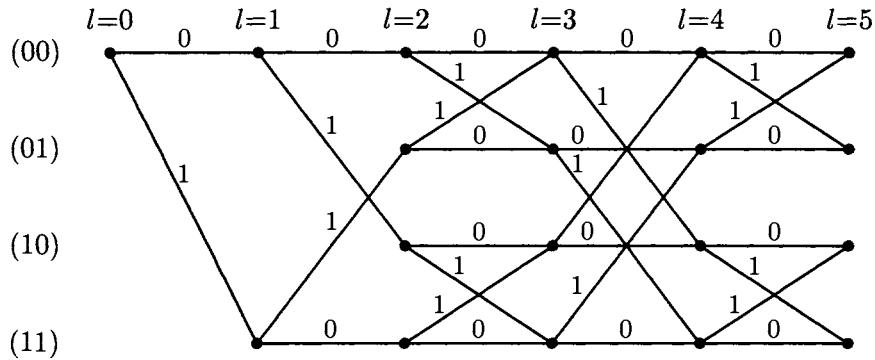


Fig. 2.3: Trellis for the binary (5,3) code

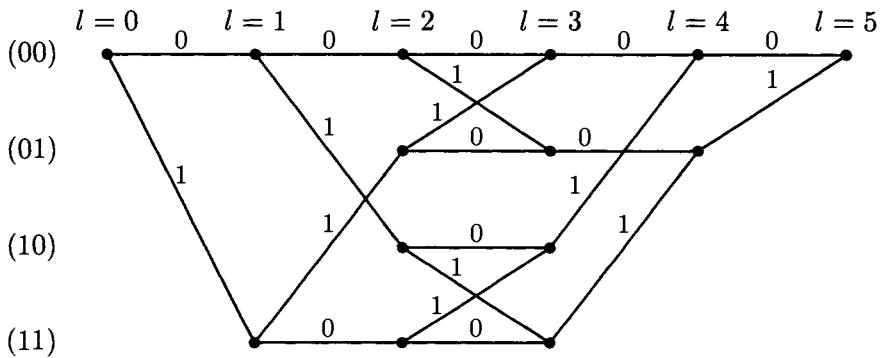


Fig. 2.4: Expurgated trellis for the binary (5,3) code

Bibliography

- [1] J. M. Wozencraft, and I. M. Jacobs, “*Principles of Communication Engineering*,” John Wiley, 1965.
- [2] R. G. Gallager, “*Information Theory and Reliable Communications*,” John Wiley, 1968.
- [3] J. G. Proakis, “*Digital Communications*”, 2nd Ed. McGraw-Hill, New York, 1989.
- [4] S. Lin, and D. J. Costello, Jr., “*Error Control Coding: Fundamentals and Applications*”, Prentice-Hall, 1983.
- [5] E. R. Berlekamp, “*Algebraic Coding Theory*”, McGraw-Hill, 1968.
- [6] R. Blahut, “*Theory and Practice of Error Control Codes*”, Adison-Wesley Publishing Company, 1983.
- [7] A. J. Viterbi, and J. K. Omura, “*Principles of Digital Communication and Coding*”, McGraw-Hill, New York, 1979.
- [8] L. R. Bahl, J. Cocke, F. Jelinek and J. Raviv, “Optimum Decoding of Linear Codes for Minimizing Symbol Error Rate”, *IEEE Trans. Inform. Theory*, vol. IT-13, 1974, pp. 284-287.
- [9] J. K. Wolf, “Efficient Maximum Likelihood Decoding of Linear Block Codes”, *IEEE Trans. Inform. Theory*, vol. IT-24, Jan. 1978, pp. 76-80.
- [10] G. D. Forney, Jr., “Coset Codes–Part II: Binary Lattices and Related Codes”, *IEEE Trans. Inform. Theory*, vol. 34, Sep. 1988, pp. 1152-1187.

- [11] R. J. McEiece, "On the BCJR Trellis for Linear Block Codes", *IEEE Trans. Inform. Theory*, vol. 42, No. 4, July 1996.
- [12] H. Imai, "*Essentials of Error-Control Coding Techniques*", Academic Press, 1990.
- [13] D. Chase, "A Class of Algorithms for Decoding Block Codes with Channel Measurement Information", *IEEE Trans. Inform. Theory*, vol. IT-18, Jan. 1972, pp. 170-182.
- [14] S. Benedetto and G. Montorsi, "Unveiling turbo-codes: Some results on parallel concatenated coding schemes," *IEEE Trans. Inform. Theory*, vol. 42, no. 2, Mar. 1996, pp. 409-428.

Chapter 3

Convolutional Codes

3.1 Introduction

Convolutional codes have been widely used in applications such as space and satellite communications, cellular mobile, digital video broadcasting etc. Its popularity stems from simple structure and availability of easily implementable maximum likelihood soft decision decoding methods.

Convolutional codes were first introduced by Elias [5]. The ground work on algebraic theory of convolutional codes was performed by Forney [6]. In this chapter we present the main results on the algebraic structure of convolutional codes needed in the design of turbo codes. Of particular importance are code structure, encoder realization and trellis representation. We first discuss the structure of convolutional codes and then show how to implement feedforward and feedback convolutional encoders.

The equivalence of encoders is discussed and it is shown how to get equivalent systematic encoders with rational generator matrices from nonsystematic encoders with polynomial generator matrices. Next, we present the finite state machine description of convolution codes and derive the state and trellis diagrams.

In the interest of brevity many details of the theory had to be omitted. The chapter is concluded by the discussion of punctured convolutional codes.

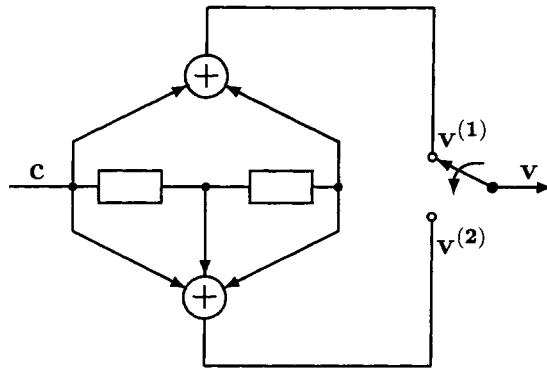


Fig. 3.1: A rate 1/2 convolutional encoder

3.2 The Structure of $(n,1)$ Convolutional Codes

Let us consider a simple rate 1/2 convolutional code generated by the encoder shown in Fig. 3.1. The encoder is a linear feedforward shift register. The input data stream is a binary sequence, given by

$$\mathbf{c} = (\dots, c_{-1}, c_0, c_1, \dots, c_l, \dots) \quad (3.1)$$

where l is a time instant.

The outputs are two sequences $\mathbf{v}^{(1)}$ and $\mathbf{v}^{(2)}$

$$\begin{aligned} \mathbf{v}^{(1)} &= (\dots, v_{-1}^{(1)}, v_0^{(1)}, v_1^{(1)}, \dots, v_l^{(1)}, \dots) \\ \mathbf{v}^{(2)} &= (\dots, v_{-1}^{(2)}, v_0^{(2)}, v_1^{(2)}, \dots, v_l^{(2)}, \dots) \end{aligned} \quad (3.2)$$

where we assume that $v_l^{(i)}$, $i = 1, 2$ are binary symbols.

At time l the input to the encoder is c_l and the output is a code block \mathbf{v}_l

$$\mathbf{v}_l = (v_l^{(1)}, v_l^{(2)}) \quad (3.3)$$

The elements of the output code block are computed as

$$\begin{aligned} v_l^{(1)} &= c_l + c_{l-2} \\ v_l^{(2)} &= c_l + c_{l-1} + c_{l-2} \end{aligned} \quad (3.4)$$

where $+$ denotes modulo-2 addition. Hence, the encoder must store past two information bits. We say that the *memory* of the encoder is $\nu = 2$. The two output sequences are multiplexed by a commutator to form a single code sequence.

$$\mathbf{v} = (\dots, v_{-1}^{(1)}, v_{-1}^{(2)}, v_0^{(1)}, v_0^{(2)}, v_1^{(1)}, v_1^{(2)}, \dots, v_l^{(1)}, v_l^{(2)}, \dots) \quad (3.5)$$

The set of all possible code sequences \mathbf{v} produced by the encoder forms the $(2, 1)$ convolutional code. The connections between the shift register elements and the modulo 2 adders can be conveniently described by the following two generator sequences

$$\begin{aligned}\mathbf{g}^{(1)} &= (g_0^{(1)} g_1^{(1)} g_2^{(1)}) = (101) \\ \mathbf{g}^{(2)} &= (g_0^{(2)} g_1^{(2)} g_2^{(2)}) = (111)\end{aligned}$$

where $\mathbf{g}^{(1)}$ represents the upper and $\mathbf{g}^{(2)}$ the lower connections with the leftmost entry being the connection to the leftmost stage. The term convolutional codes comes from the observation that the i th output sequence, $i = 1, 2$, given by Eq. (3.2), represents the convolution of the input sequence and the i th generator sequence.

$$\mathbf{v}^{(i)} = \mathbf{c} * \mathbf{g}^{(i)}, \quad i = 1, 2 \quad (3.6)$$

where $*$ denotes the convolution operator.

Example 3.1 Let us assume that the input sequence is

$$\mathbf{c} = (1011100 \dots) \quad (3.7)$$

Then the two output sequences are

$$\mathbf{v}^{(1)} = (1001011 \dots) \quad (3.8)$$

$$\mathbf{v}^{(2)} = (1100101 \dots) \quad (3.9)$$

The transmitted code sequence is

$$\mathbf{v} = (11, 01, 00, 10, 01, 10, 11, \dots) \quad (3.10)$$

If the generator sequences $\mathbf{g}^{(1)}$ and $\mathbf{g}^{(2)}$ are interleaved and arranged in the matrix

$$\mathbf{G} = \begin{bmatrix} g_0^{(1)} g_0^{(2)} & g_1^{(1)} g_1^{(2)} & g_2^{(1)} g_2^{(2)} & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & g_0^{(1)} g_0^{(2)} & g_1^{(1)} g_1^{(2)} & g_2^{(1)} g_2^{(2)} & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & g_0^{(1)} g_0^{(2)} & g_1^{(1)} g_1^{(2)} & g_2^{(1)} g_2^{(2)} & \cdots \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & \cdots \\ 0 & 0 & 0 & 0 & \cdots & g_0^{(1)} g_0^{(2)} & g_1^{(1)} g_1^{(2)} & g_2^{(1)} g_2^{(2)} \end{bmatrix}$$

the encoding operation can be represented in a matrix form as

$$\mathbf{v} = \mathbf{c} \cdot \mathbf{G} \quad (3.11)$$

where all operations are modulo 2.

Observe that each row of \mathbf{G} is obtained by shifting the preceding row by $n = 2$ places to the right. The generator matrix is semi-infinite corresponding to the fact that the input sequence may be infinite.

Similarly, we can generate an $(n, 1)$ convolutional code by a feedforward linear shift register shown in Fig. 3.2, which has one input and n output sequences. The code rate is $1/n$.

The encoder is specified by a set of n generator sequences of length $(\nu + 1)$, where ν is the encoder memory, given by

$$\begin{aligned} \mathbf{g}^{(1)} &= (g_0^{(1)} g_1^{(1)} \cdots g_{\nu}^{(1)}) \\ \mathbf{g}^{(2)} &= (g_0^{(2)} g_1^{(2)} \cdots g_{\nu}^{(2)}) \\ &\vdots \\ \mathbf{g}^{(n)} &= (g_0^{(n)} g_1^{(n)} \cdots g_{\nu}^{(n)}) \end{aligned} \quad (3.12)$$

with coefficients in the binary field $GF(2)$. The i th output sequence is obtained by convolving the input message sequence \mathbf{c} and the i th generator sequence $\mathbf{g}^{(i)}$

$$\mathbf{v}^{(i)} = \mathbf{c} * \mathbf{g}^{(i)}, \quad i = 1, 2, \dots, n \quad (3.13)$$

At the time l , the i th output symbol is

$$\begin{aligned} v_l^{(i)} &= c_l g_0^{(i)} + c_{l-1} g_1^{(i)} + c_{l-2} g_2^{(i)} + \cdots + c_{l-\nu} g_{\nu}^{(i)} \\ &= \sum_{j=0}^{\nu} c_{l-j} g_j^{(i)} \end{aligned} \quad (3.14)$$

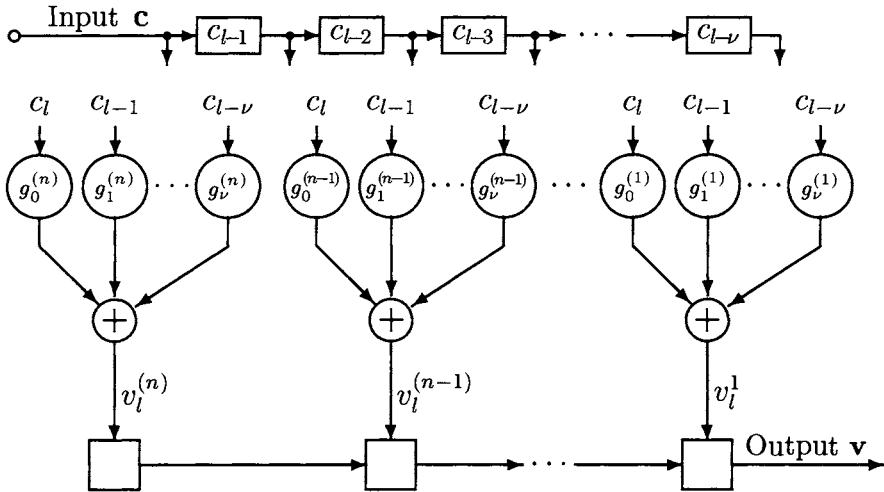


Fig. 3.2: A general $(n, 1, \nu)$ convolutional code feedforward encoder

Let us assume that the input message \mathbf{c} consists of L bits. As it takes ν time instants to move out of the memory, each output sequence consists of $L + \nu$ binary symbols.

We can associate with input and output streams, $c(D), v^{(i)}(D)$, respectively, power series sequences in the form

$$\begin{aligned} \mathbf{c}(D) &= \cdots + c_{-1}D^{-1} + c_0 + c_1D + c_2D^2 + \cdots \\ \mathbf{v}^{(i)}(D) &= \cdots + v_{-1}^{(i)}D^{-1} + v_0^{(i)} + v_1^{(i)}D + v_2^{(i)}D^2 + \cdots \end{aligned}$$

where the symbol D is an indeterminate. The indeterminate D can be considered as a unit delay operator. Its power denotes the number of time units the binary symbol is delayed with respect to the initial binary symbol in the sequence¹.

A feedforward shift register of length ν can be represented by a polynomial of degree at most ν in the field of the register. The coefficients of the polynomial are the shift register taps. These polynomials are called the *generator polynomials*.

¹The delay operator D corresponds to Z^{-1} in sampled data theory, where Z is a complex variable. Operators involving D are called transform domain operators. $\mathbf{c}(D)$ and $\mathbf{v}^{(i)}(D)$ can be regarded as modulo-2 transforms of \mathbf{c} and $\mathbf{v}^{(i)}$, respectively.

For the encoder in Fig. 3.1 the generator polynomials are

$$\begin{aligned}\mathbf{g}_1(D) &= 1 + D^2 \\ \mathbf{g}_2(D) &= 1 + D + D^2\end{aligned}\quad (3.15)$$

where all the operations are modulo-2.

The generator polynomials of a convolutional code are typically arranged into a matrix of polynomials called the *generator matrix*. For the encoder in Fig. 3.1 the generator matrix is

$$\mathbf{G}(D) = \begin{bmatrix} 1 + D^2 & 1 + D + D^2 \end{bmatrix} \quad (3.16)$$

The encoding equations for the encoder in Fig. 3.1 can be now written as

$$\begin{aligned}\mathbf{v}^{(1)}(D) &= \mathbf{c}(D)\mathbf{g}^{(1)}(D) \\ \mathbf{v}^{(2)}(D) &= \mathbf{c}(D)\mathbf{g}^{(2)}(D)\end{aligned}\quad (3.17)$$

After multiplexing the entire code sequence becomes

$$\mathbf{v}(D) = \mathbf{v}^{(1)}(D^2) + D\mathbf{v}^{(2)}(D^2) \quad (3.18)$$

Example 3.2 For the information sequence

$$\mathbf{c}(D) = 1 + D^2 + D^3 + D^4 \quad (3.19)$$

the encoding equation becomes

$$\begin{aligned}\mathbf{v}^{(1)}(D) &= (1 + D^2 + D^3 + D^4)(1 + D^2) = 1 + D^3 + D^5 + D^6 \\ \mathbf{v}^{(2)}(D) &= (1 + D^2 + D^3 + D^4)(1 + D + D^2) = 1 + D + D^4 + D^6\end{aligned}$$

The transmitted code sequence is

$$\begin{aligned}\mathbf{v}(D) &= \mathbf{v}^{(1)}(D^2) + D\mathbf{v}^{(2)}(D^2) \\ &= 1 + D + D^3 + D^6 + D^9 + D^{10} + D^{12} + D^{13}\end{aligned}$$

which agrees with the result in Example 3.1.

For an $(n, 1)$ convolutional code there are n generator polynomials in the generator matrix. They are arranged into a $1 \times n$ matrix of polynomials

$$\mathbf{G}(D) = [\mathbf{g}_1(D) \ \mathbf{g}_2(D) \ \cdots \ \mathbf{g}_n(D)] \quad (3.20)$$

The encoding operation can be represented as a vector matrix product

$$\mathbf{v}(D) = \mathbf{c}(D)\mathbf{G}(D) \quad (3.21)$$

where

$$\mathbf{v}(D) = [\mathbf{v}^{(1)}(D) \ \mathbf{v}^{(2)}(D) \ \cdots \ \mathbf{v}^{(n)}(D)] \quad (3.22)$$

3.3 The Structure of (n, k) Convolutional Codes

In general, the encoder has k input and n output sequences. Each input information sequence is encoded into a code sequence. The set of all possible code sequences generated by the encoder forms an (n, k) convolutional code. As with block codes, the code rate is defined as the ratio $R = k/n$.

Any (n, k) convolutional code is specified by $k \times n$ generator polynomials, which form the generator matrix, $\mathbf{G}(D)$

$$\mathbf{G}(D) = \begin{bmatrix} \mathbf{g}_{11} & \mathbf{g}_{12} & \cdots & \mathbf{g}_{1n} \\ \mathbf{g}_{21} & \mathbf{g}_{22} & \cdots & \mathbf{g}_{2n} \\ \vdots & & & \vdots \\ \mathbf{g}_{k1} & \mathbf{g}_{k2} & \cdots & \mathbf{g}_{kn} \end{bmatrix} \quad (3.23)$$

Consider k input sequences $\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_i, \dots, \mathbf{c}_k$. Each of them can be represented by a power series

$$\mathbf{c}^{(i)}(D) = \cdots c_{-1}^{(i)}D^{-1} + c_0^{(i)} + c_1^{(i)}D + c_2^{(i)}D^2 + \cdots \quad (3.24)$$

for $i = 1, 2, \dots, k$. They can be arranged as a row vector of sequences

$$\mathbf{c}(D) = [\mathbf{c}^{(1)}(D) \ \mathbf{c}^{(2)}(D) \ \cdots \ \mathbf{c}^{(k)}(D)] \quad (3.25)$$

Alternatively, we can write for $\mathbf{c}(D)$ as

$$\begin{aligned}\mathbf{c}(D) &= [c_0^{(1)}, c_0^{(2)}, \dots, c_0^{(k)}] + [c_1^{(1)}, c_1^{(2)}, \dots, c_1^{(k)}] D \\ &\quad + \dots + [c_l^{(1)}, c_l^{(2)}, \dots, c_l^{(k)}] D^l + \dots\end{aligned}$$

Each bracketed block represents an input block. Similarly, the entire output sequence consists of n code polynomials

$$\mathbf{v}(D) = [\mathbf{v}^{(1)}(D) \ \mathbf{v}^{(2)}(D) \ \dots \ \mathbf{v}^{(n)}(D)] \quad (3.26)$$

The encoding operation can now be represented by a vector-matrix product

$$\mathbf{v}(D) = \mathbf{c}(D)\mathbf{G}(D) \quad (3.27)$$

The j th code sequence is computed as

$$\mathbf{v}^{(j)}(D) = \sum_{i=1}^k \mathbf{c}^{(i)}(D) \mathbf{g}_{ij}(D) \quad (3.28)$$

After multiplexing the transmitted code sequence is given by

$$\mathbf{v}(D) = \mathbf{v}^{(1)}(D^n) + D\mathbf{v}^{(2)}(D^n) + \dots + D^{(n-1)}\mathbf{v}^{(n)}(D^n) \quad (3.29)$$

The *memory* of the i th encoder input is given by

$$\nu_i = \max_{1 \leq j \leq n} \{\deg \mathbf{g}_{ij}(D)\} \quad (3.30)$$

The memory of the encoder is defined as the maximum memory of the encoder inputs

$$\nu = \max_{1 \leq i \leq k} \{\nu_i\} \quad (3.31)$$

and the *overall encoder memory* as the sum of the memories of the encoder inputs

$$K = \sum_{i=1}^k \nu_i \quad (3.32)$$

The encoder for an (n, k) convolutional code can be realized by a linear sequential circuit consisting of k feedforward shift registers, the i th of length ν_i , with the outputs formed as modulo-2 sums of the corresponding shift register contents.

Example 3.3 Consider the $(3, 2)$ convolutional encoder shown in Fig. 3.3. The generator matrix for this encoder is

$$\mathbf{G}(D) = \begin{bmatrix} 1+D & D & 1+D \\ D & 1 & 1 \end{bmatrix} \quad (3.33)$$

If the input sequences are

$$\begin{aligned} \mathbf{c}^{(1)}(D) &= 1 + D^2 \\ \mathbf{c}^{(2)}(D) &= 1 + D \end{aligned} \quad (3.34)$$

the encoding operation can be represented as

$$\begin{aligned} \mathbf{v}(D) &= [1 + D^2 \ 1 + D] \begin{bmatrix} 1+D & D & 1+D \\ D & 1 & 1 \end{bmatrix} \\ &= [1 + D^3 \ 1 + D^3 \ D^2 + D^3] \end{aligned} \quad (3.35)$$

After multiplexing the entire code sequence is given by

$$\begin{aligned} \mathbf{v}(D) &= \mathbf{v}^{(1)}(D^3) + D\mathbf{v}^{(2)}(D^3) + D^2\mathbf{v}^{(3)}(D^3) \\ &= 1 + D + D^8 + D^9 + D^{10} + D^{11} \end{aligned} \quad (3.36)$$

The overall memory for the encoder in Fig. 3.3 is 2 and the memory for each encoder input is 1. The number of memory elements required for this type of implementation is equal to the overall memory.

3.4 Systematic Form

In a *systematic* (n, k) convolutional code the first k output sequences are exact replicas of the input information sequences. The generator matrix of a systematic convolutional code is of the form

$$\mathbf{G}(D) = [\mathbf{I} \ \mathbf{P}(D)] \quad (3.37)$$

where \mathbf{I} is a $k \times k$ identity matrix, and $\mathbf{P}(D)$ is a $k \times (n - k)$ matrix of polynomials with elements in $GF(2)$.

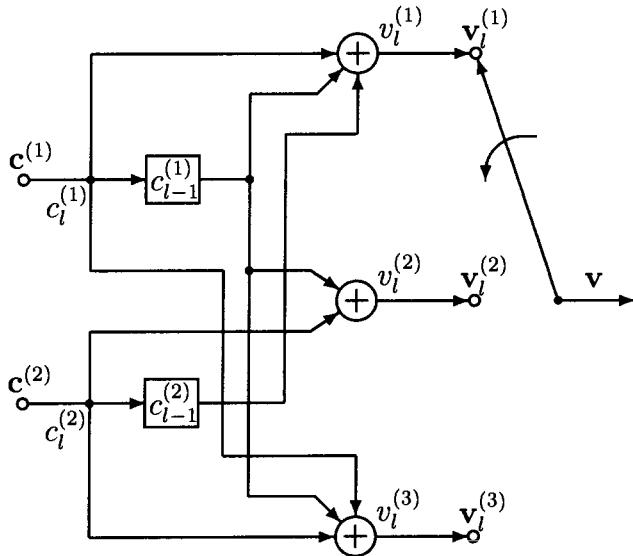


Fig. 3.3: Encoder for a (3, 2, 1) code

For a block code a generator matrix can be transformed into a systematic form by elementary row operations. This is not always possible for convolutional codes.

In order to obtain a systematic form for a given generator matrix of a convolutional code we need first to define *equivalent* generator matrices. Two generator matrices are equivalent if they generate the same convolutional code.

Example 3.4 Consider the generator matrix for the encoder in Fig 3.1.

$$\mathbf{G}(D) = \begin{bmatrix} 1 + D^2 & 1 + D + D^2 \end{bmatrix} \quad (3.38)$$

The code sequence of this encoder is given by

$$\begin{aligned} \mathbf{v}(D) &= \mathbf{c}(D) [1 + D^2 \ 1 + D + D^2] \\ &= \mathbf{c}(D)(1 + D^2) \left[1 \ \frac{1 + D + D^2}{1 + D^2} \right] \\ &= \mathbf{c}'(D) \left[1 \ \frac{1 + D + D^2}{1 + D^2} \right] \\ &= \mathbf{c}'(D)\mathbf{G}_1(D) \end{aligned} \quad (3.39)$$

where

$$\mathbf{c}'(D) = \mathbf{c}(D)\mathbf{T}(D) \quad (3.40)$$

and

$$\mathbf{T}(D) = [1 + D^2] \quad (3.41)$$

Clearly, multiplication of $\mathbf{c}(D)$ by $\mathbf{T}(D)$ generates a scrambled version of the input sequence. Thus, the set of scrambled input sequences, $\mathbf{c}'(D)$, multiplied by the generator matrix $\mathbf{G}_1(D)$ produces the same set of output sequences as the original generator matrix $\mathbf{G}(D)$, where

$$\mathbf{G}(D) = \mathbf{T}(D)\mathbf{G}_1(D) \quad (3.42)$$

We say that these two matrices are equivalent. The set of sequences $\mathbf{c}(D)$ is identical to the set of sequences $\mathbf{c}'(D)$ if and only if $\mathbf{T}(D)$ is invertible. The inverse of $\mathbf{T}(D)$ in this example is given by

$$\mathbf{T}^{-1}(D) = \left[\frac{1}{1 + D^2} \right] \quad (3.43)$$

In general, two encoding matrices $\mathbf{G}(D) = \mathbf{T}(D)\mathbf{G}_1(D)$ and $\mathbf{G}_1(D)$ are equivalent if the matrix $\mathbf{T}(D)$ is invertible.

Matrix $\mathbf{G}_1(D)$ in the previous example is in systematic form. However, its entries are not polynomials, but rational functions. The parity output sequence in Eq. 3.39 is obtained by multiplying the input sequence by the polynomial $(1 + D + D^2)$ and dividing it by the polynomial $(1 + D^2)$. The multiplication operations can be performed by a feedforward shift register and division by a feedback shift register.

In general, the multiplication and division of the input sequence $\mathbf{c}(D)$ by the polynomials $\mathbf{a}(D)$ and $\mathbf{q}(D)$, respectively, as shown below

$$\mathbf{v}(D) = \mathbf{c}(D) \frac{\mathbf{a}(D)}{\mathbf{q}(D)} \quad (3.44)$$

can be performed by the circuits illustrated in Figs. 3.4 and 3.5. Note that $\mathbf{a}(D)$ and $\mathbf{q}(D)$ are the polynomials with binary coefficients

$$\begin{aligned} \mathbf{a}(D) &= a_0 + a_1 D + \cdots + a_\nu D^\nu \\ \mathbf{q}(D) &= 1 + q_1 D + \cdots + q_\nu D^\nu \end{aligned} \quad (3.45)$$

It is assumed that $\mathbf{q}(D)$ is a *delay free* polynomial, i.e. $q_0 = 1$. In other words, it can always be written in this form by pulling out the common form D^i , where i is the delay.

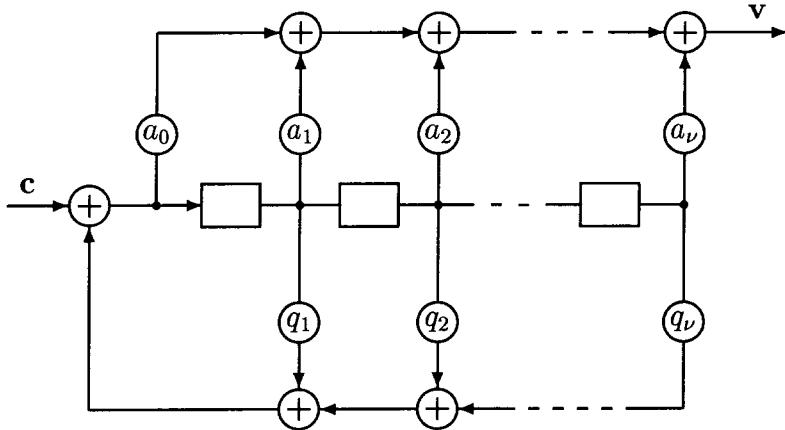


Fig. 3.4: The controller canonical form of a rational transfer function $\mathbf{a}(D)/\mathbf{q}(D)$

The ratio

$$\mathbf{T}(D) = \frac{\mathbf{a}(D)}{\mathbf{q}(D)} \quad (3.46)$$

represents a *rational transfer function*. In general, the generator matrices of convolutional codes have as entries rational transfer functions. The outputs are obtained by multiplying the input sequence with the generator matrix containing rational transfer functions. Given a rational transfer function and the input sequence the multiplication can be performed by linear circuits in many different ways. Fig. 3.4 shows the *controller canonical form* of the rational function in Eq. (3.46).

Another so-called *observer canonical form* of the rational transfer function in Eq. (3.46) is illustrated in Fig. 3.5.

As the circuit in Fig. 3.5 is linear, we have

$$\mathbf{v}(D) = \mathbf{c}(D)(a_0 + a_1 D + \cdots + a_v D^v) + \mathbf{v}(D)(q_1 D + \cdots + q_v D^v)$$

which is the same as (3.44). In this implementation the delay elements, in general, do not form a shift register, as they are separated

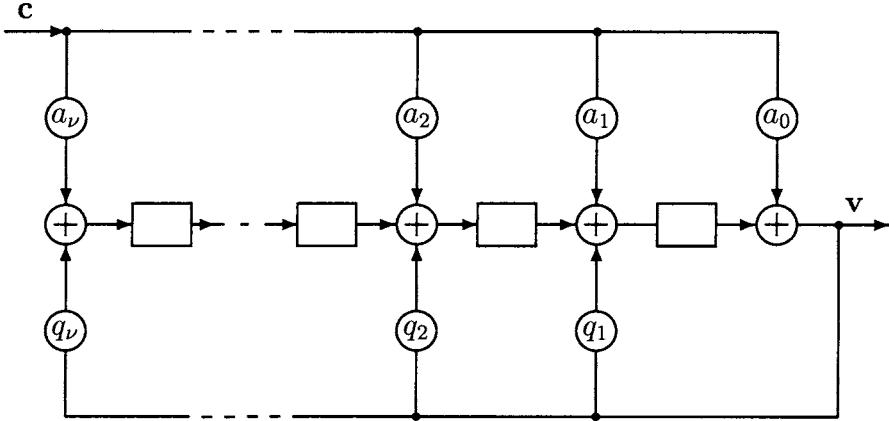


Fig. 3.5: The observer canonical form of a rational transfer function $\mathbf{a}(D)/\mathbf{q}(D)$

by adders.

Systematic encoders in the controller and observer canonical form, based on the generator matrix $\mathbf{G}_1(D)$, are shown in Figs. 3.6 and 3.7, respectively.

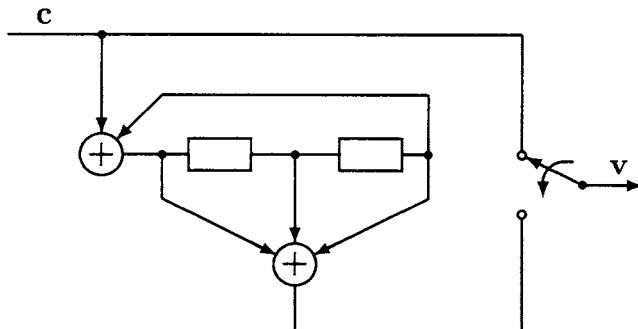


Fig. 3.6: The controller canonical form of the systematic $(2, 1)$ encoder with the generator matrix $\mathbf{G}_1(D)$

In some cases, the elements of the systematic code generator matrix are rational fraction of the unit delay operator D . For such generator matrices it is not possible to construct a systematic encoder with feedforward registers. It is, however, always possible to

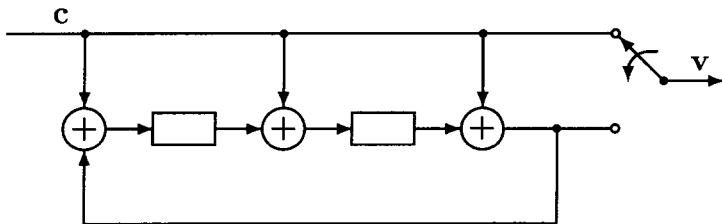


Fig. 3.7: The observer canonical form of the systematic $(2, 1)$ encoder with the generator matrix $\mathbf{G}_1(D)$

construct a systematic encoder with a feedback register.

3.5 Parity Check Matrix

We can describe a convolutional code by a parity check matrix in a similar way as for a block code. A parity check matrix $\mathbf{H}(D)$ is an $(n - k) \times n$ matrix of rational functions, that satisfies

$$\mathbf{G}(D)\mathbf{H}^T(D) = \mathbf{0} \quad (3.47)$$

The encoding operation can be written as

$$\mathbf{v}(D)\mathbf{H}^T(D) = \mathbf{0} \quad (3.48)$$

for all code sequences $\mathbf{v}(D)$.

If $\mathbf{G}(D)$ is a systematic generator matrix, the parity check matrix can be as

$$\mathbf{H}(D) = [\mathbf{P}^T(D) \quad \mathbf{I}] \quad (3.49)$$

where \mathbf{I} is an $(n - k) \times (n - k)$ identity matrix and $\mathbf{P}^T(D)$ is the transpose of matrix $\mathbf{P}(D)$ in (3.37).

We can always find a parity check matrix corresponding to a generator matrix $\mathbf{G}(D)$ if $\mathbf{G}(D)$ is transformed into a systematic form, by referring to (3.49) and (3.37).

For the $1/2$ rate encoder shown in Fig. 3.1 the parity check matrix obtained from the systematic generator matrix $\mathbf{G}_1(D)$ is

given by

$$\mathbf{H}(D) = \begin{bmatrix} 1 + D + D^2 & 1 \\ 1 + D^2 & 1 \end{bmatrix} \quad (3.50)$$

3.6 Catastrophic Codes

Systematic encoders have a significant advantage because the message is displayed in the encoded sequence and can be read out directly from the received sequence. On the other hand, with nonsystematic encoders the message is not visible in the coded sequence. Therefore, an inverter is required to recover the information sequence from the decoded sequence. The information sequence can be recovered by passing the decoded sequence, $\mathbf{v}(D)$, through an inverter, specified by the generator matrix $\mathbf{G}^{-1}(D)$

$$\mathbf{v}(D)\mathbf{G}^{-1}(D) = \mathbf{c}(D)\mathbf{G}(D)\mathbf{G}^{-1}(D) = \mathbf{c}(D)D^l \quad (3.51)$$

In order to obtain the original input sequence $\mathbf{c}(D)$ or its delayed version $\mathbf{c}(D)D^l$, where l is the time delay, the following relationship must be satisfied

$$\mathbf{G}(D)\mathbf{G}^{-1}(D) = \mathbf{I}D^l \quad (3.52)$$

where \mathbf{I} is a $k \times k$ identity matrix. That is, the generator matrix must be invertible.

To see under what condition $\mathbf{G}^{-1}(D)$ exists, let us consider an (n, k) convolutional code, defined by the generator matrix $\mathbf{G}(D)$.

Let $\Delta_i(D)$, $i = 1, 2, \dots, \binom{n}{k}$ be the determinants of the $\binom{n}{k}$ distinct $k \times k$ submatrices of the generator matrix $\mathbf{G}(D)$. Massey and Sain [13] have shown that the inverse polynomial matrix $\mathbf{G}^{-1}(D)$ exists if and only if

$$\text{GCD} \left[\Delta_i(D), i = 1, 2, \dots, \binom{n}{k} \right] = D^l, \quad l \geq 0 \quad (3.53)$$

where GCD denotes the greatest common divisor.

Example 3.5 For the encoder in Fig. 3.1

$$\text{GCD} \begin{bmatrix} 1 + D^2 & 1 + D + D^2 \end{bmatrix} = 1 \quad (3.54)$$

and the generator matrix inverse is

$$\mathbf{G}^{-1}(D) = \begin{bmatrix} 1 + D \\ D \end{bmatrix} \quad (3.55)$$

If the generator matrix inverse $\mathbf{G}^{-1}(D)$, does not exist, the code is called *catastrophic*. For catastrophic codes a finite number of channel errors causes an infinite number of decoding errors.

Example 3.6 The generator matrix

$$\mathbf{G}(D) = \begin{bmatrix} 1 + D & 1 + D^2 \end{bmatrix} \quad (3.56)$$

generates a catastrophic code.

As

$$\text{GCD} \begin{bmatrix} 1 + D & 1 + D^2 \end{bmatrix} = 1 + D \quad (3.57)$$

$\mathbf{G}^{-1}(D)$ does not exist. If the information sequence is

$$\mathbf{c}(D) = 1 + D + D^2 + \dots = \frac{1}{1 + D} \quad (3.58)$$

the code sequences are

$$\begin{aligned} \mathbf{v}^{(1)}(D) &= 1 \\ \mathbf{v}^{(2)}(D) &= 1 + D \end{aligned} \quad (3.59)$$

The weight of the code sequence is 3, while the weight of the input sequence is infinite. If this code sequence is transmitted over a binary symmetric channel (BSC) in which three errors occur, changing the three non zero symbols to zero, the received sequence will contain all zeros. Since this is a valid code sequence, the decoder will deliver it to the user. Thus, the decoded sequence will have an infinite number of errors, though there were only three errors in the channel.

Catastrophic codes should be avoided for obvious reasons.

Example 3.7 Consider the generator matrix for a $(3, 2)$ convolutional code

$$\mathbf{G}(D) = \begin{bmatrix} 1 & D^2 & 1 \\ 0 & 1 & D + D^2 \end{bmatrix} \quad (3.60)$$

The determinants of the 2×2 submatrices are

$$\Delta_1(D) = 1 \quad \Delta_2(D) = D + D^2 \quad \Delta_3(D) = 1 + D^3 + D^4$$

The GCD of these three polynomials is 1. Therefore, there exists the inverse polynomial matrix $\mathbf{G}^{-1}(D)$. It is given by

$$\mathbf{G}^{-1}(D) = \begin{bmatrix} D^3 + D^4 & 1 + D^2 + D^3 + D^4 \\ D + D^2 & 1 + D + D^2 \\ 1 & 1 \end{bmatrix} \quad (3.61)$$

Hence, $\mathbf{G}(D)$ generates a noncatastrophic code.

It can be seen that systematic convolutional codes always satisfy (3.53). That is due to the fact that one of the submatrices is the $k \times k$ identity matrix with unity determinant. Therefore all systematic convolutional encoders possess a polynomial right inverse matrix $\mathbf{G}^{-1}(D)$, which makes them noncatastrophic.

3.7 Systematic Encoders

Systematic encoders are simpler to implement, do not require an inverter and are never catastrophic.

However, unless feedback is allowed in the encoder, systematic generator matrices generate codes with worse error performance than nonsystematic codes of the same code rate and overall constraint length [15].

It has been shown [4] that every convolutional encoder generator matrix is equivalent to a systematic rational encoding matrix.

Let us assume that a given generator matrix $\mathbf{G}(D)$ is polynomial and that it has the polynomial right inverse $\mathbf{G}^{-1}(D)$. A polynomial generator matrix with the right inverse has the GCD of all $k \times k$ subdeterminants equal to D^l . Premultiplication by the inverse of

such a submatrix gives an equivalent systematic generator matrix, possibly rational. If the generator matrix is represented as

$$\mathbf{G}(D) = [\mathbf{T}(D) \quad \mathbf{S}(D)] \quad (3.62)$$

the equivalent systematic generator matrix $\mathbf{G}_1(D)$ is given by

$$\mathbf{G}_1(D) = \mathbf{T}^{-1}(D) \cdot \mathbf{G}(D) \quad (3.63)$$

Example 3.8 Consider the generator matrix

$$\mathbf{G}(D) = \begin{bmatrix} 1 & D & 1 \\ 0 & 1 & D \end{bmatrix} \quad (3.64)$$

Let $\mathbf{T}(D)$ be the 2×2 submatrix consisting of the first two columns of $\mathbf{G}(D)$

$$\mathbf{T}(D) = \begin{bmatrix} 1 & D \\ 0 & 1 \end{bmatrix} \quad (3.65)$$

Its determinant is given by

$$\det(\mathbf{T}(D)) = 1$$

and its inverse is

$$\mathbf{T}^{-1}(D) = \begin{bmatrix} 1 & D \\ 0 & 1 \end{bmatrix} \quad (3.66)$$

Multiplying $\mathbf{G}(D)$ by $\mathbf{T}^{-1}(D)$ yields a systematic generator matrix $\mathbf{G}_1(D)$ equivalent to $\mathbf{G}(D)$.

$$\begin{aligned} \mathbf{G}_1(D) &= \mathbf{T}^{-1}(D) \cdot \mathbf{G}(D) \\ &= \begin{bmatrix} 1 & 0 & 1+D^2 \\ 0 & 1 & D \end{bmatrix} \end{aligned} \quad (3.67)$$

The elements of $\mathbf{G}_1(D)$ are polynomials over $GF(2)$. Encoders with generator matrices $\mathbf{G}(D)$ and $\mathbf{G}_1(D)$ generates the same code. The obtained generator matrix is polynomial and can be implemented as a feedforward register.

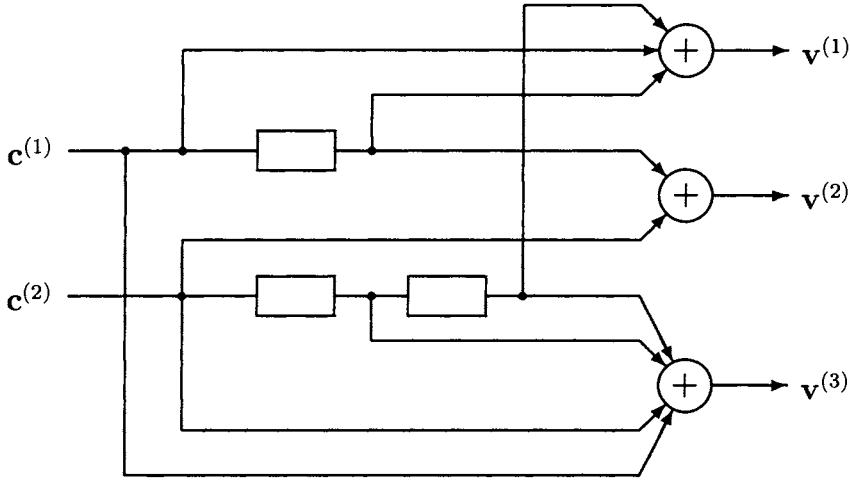


Fig. 3.8: Nonsystematic encoder in Example 3.9

Example 3.9 A rate 2/3 nonsystematic encoder with the basic generator matrix

$$\mathbf{G}(D) = \begin{bmatrix} 1+D & D & 1 \\ D^2 & 1 & 1+D+D^2 \end{bmatrix} \quad (3.68)$$

is shown in Fig. 3.8.

Let $\mathbf{T}(D)$ be the matrix formed of the first two columns of $\mathbf{G}(D)$

$$\mathbf{T}(D) = \begin{bmatrix} 1+D & D \\ D^2 & 1 \end{bmatrix} \quad (3.69)$$

Its determinant $\det(\mathbf{T}(D))$ is

$$\det(\mathbf{T}(D)) = 1 + D + D^3 \quad (3.70)$$

and its inverse $\mathbf{T}^{-1}(D)$ is given by

$$\mathbf{T}^{-1}(D) = \frac{1}{1+D+D^3} \begin{bmatrix} 1 & D \\ D^2 & 1+D \end{bmatrix} \quad (3.71)$$

The equivalent systematic generator matrix is obtained as

$$\begin{aligned} \mathbf{G}_1(D) &= \mathbf{T}^{-1}(D) \cdot \mathbf{G}(D) \\ \mathbf{G}_1(D) &= \begin{bmatrix} 1 & 0 & \frac{1+D+D^2+D^3}{1+D+D^3} \\ 0 & 1 & \frac{1+D^2+D^3}{1+D+D^3} \end{bmatrix} \end{aligned} \quad (3.72)$$

A systematic encoder in its observer canonical form for this generator matrix is shown in Fig. 3.9.

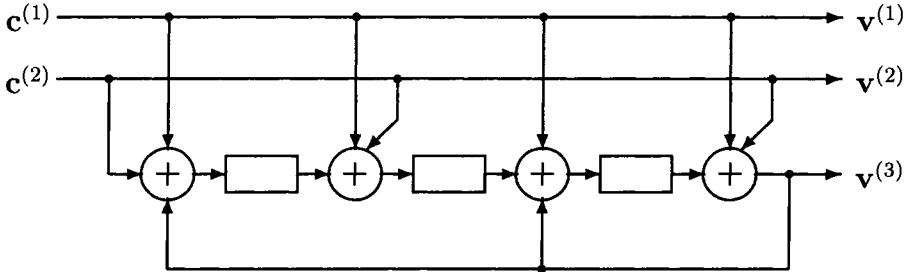


Fig. 3.9: Systematic encoder in Example 3.9

When the systematic encoder contains feedback, the entries of the generator matrix are rational functions of the form $\mathbf{a}_i^{(j)}(D)/\mathbf{q}(D)$, where

$$\mathbf{a}_i^{(j)}(D) = a_{i,0}^{(j)} + a_{i,1}^{(j)}D + \cdots + a_{i,\nu}^{(j)}D^\nu, \quad 1 \leq i \leq k, \quad (k+1) \leq j \leq n$$

and

$$\mathbf{q}(D) = 1 + q_1D + \cdots + q_\nu D^\nu$$

A rational generator matrix is realizable if $q_0 = 1$ and $\mathbf{q}(D)$ is delay free. A rational generator matrix can be written as

$$\mathbf{G}(D) = \begin{bmatrix} 1 & 0 & \cdots & 0 & \frac{\mathbf{a}_1^{(k+1)}(D)}{\mathbf{q}(D)} & \cdots & \frac{\mathbf{a}_1^{(n)}(D)}{\mathbf{q}(D)} \\ 0 & 1 & \cdots & 0 & \frac{\mathbf{a}_2^{(k+1)}(D)}{\mathbf{q}(D)} & \cdots & \frac{\mathbf{a}_2^{(n)}(D)}{\mathbf{q}(D)} \\ \vdots & & & & & & \vdots \\ 0 & 0 & \cdots & 1 & \frac{\mathbf{a}_k^{(k+1)}(D)}{\mathbf{q}(D)} & \cdots & \frac{\mathbf{a}_k^{(n)}(D)}{\mathbf{q}(D)} \end{bmatrix} \quad (3.73)$$

The encoder circuit in its canonical observer form for this systematic generator matrix is depicted in Fig. 3.10.

An $(n, n - 1)$ systematic rational generator matrix is given by

$$\mathbf{G}(D) = \begin{bmatrix} 1 & 0 & \cdots & 0 & \mathbf{a}_1^n(D)/\mathbf{q}(D) \\ 0 & 1 & \cdots & 0 & \mathbf{a}_2^n(D)/\mathbf{q}(D) \\ \vdots & & & & \vdots \\ 0 & 0 & \cdots & 1 & \mathbf{a}_{n-1}^n(D)/\mathbf{q}(D) \end{bmatrix} \quad (3.74)$$

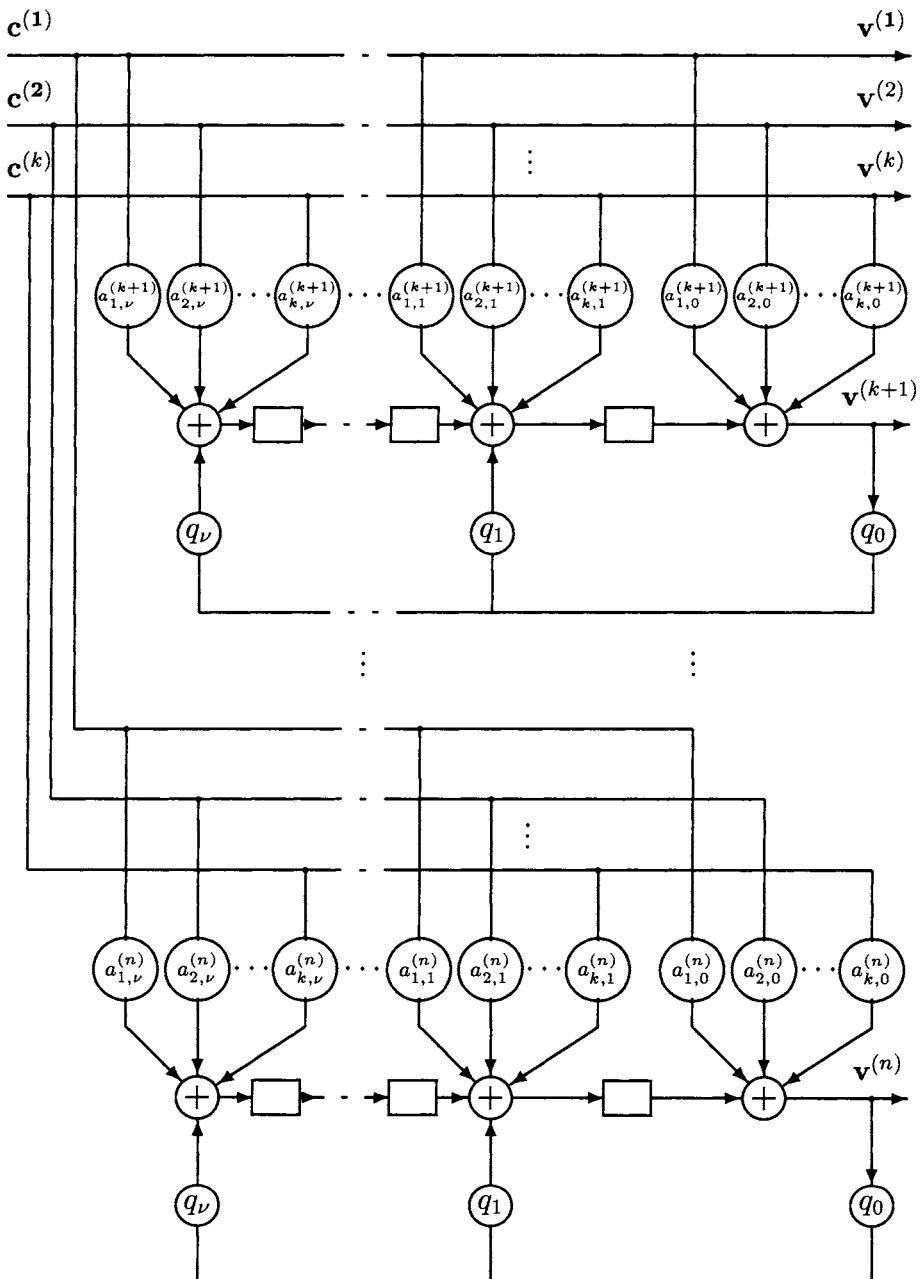


Fig. 3.10: A systematic encoder with the generator matrix in (3.73)

where

$$\mathbf{a}_i^{(n)}(D) = a_{i,0}^{(n)} + a_{i,1}^{(n)}D + \cdots + a_{i,\nu}^{(n)}D^\nu \quad (3.75)$$

for $1 \leq i \leq n - 1$. The parity check matrix for this encoder is given by

$$\mathbf{H}(D) = \left[\mathbf{a}_1^{(n)}(D)/\mathbf{q}(D) \quad \mathbf{a}_2^{(n)}(D)/\mathbf{q}(D) \quad \cdots \quad \mathbf{a}_{n-1}^{(n)}(D)/\mathbf{q}(D) \right]$$

or

$$\mathbf{H}(D) = \left[\mathbf{a}_1^{(n)}(D) \quad \mathbf{a}_2^{(n)}(D) \quad \cdots \quad \mathbf{a}_{n-1}^{(n)}(D) \right]$$

The encoder is shown in Fig. 3.11.

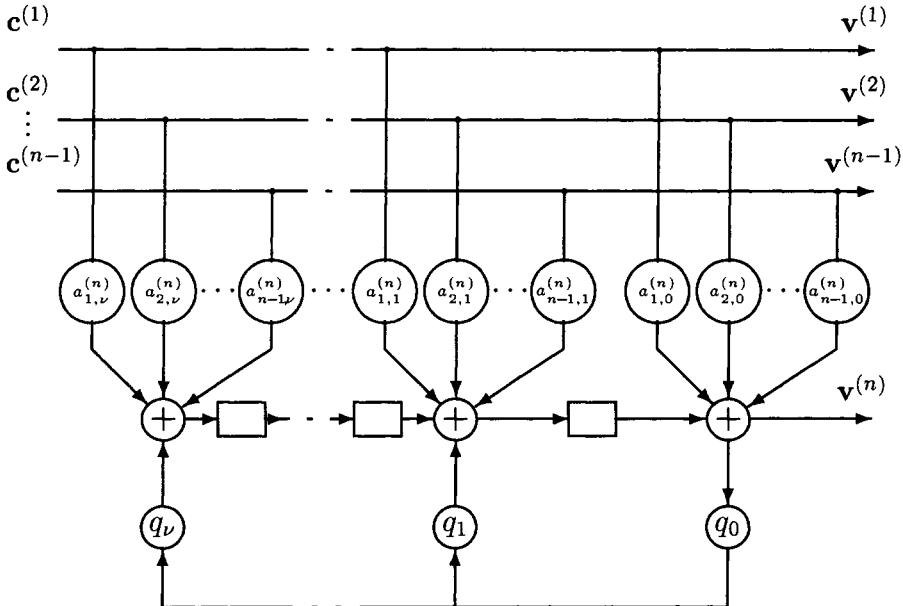


Fig. 3.11: Observer canonical form of an $(n, n - 1)$ systematic encoder

3.8 State Diagram

As a finite state linear circuit, a convolutional encoder can be described by a *state diagram*. The encoder state is defined as its

memory content. If K denotes the overall encoder memory, the total number of states is 2^K . The current state and the output of the encoder are uniquely determined by the previous state and current input. The encoder undergoes a state transition when a message block is shifted into the encoder. The state diagram is a graph that consists of nodes, representing the encoder states, and directed lines, representing state transitions. Each directed line is labelled with the input/output pair. Given a current encoder state, the information sequence at the input determines the path through the state diagram and the output sequence. For example, consider the nonsystematic encoder shown in Fig. 3.1. The state diagram for this encoder is shown in Fig. 3.12. The encoder has four states, labelled as S_0 , S_1 , S_2 and S_3 , where the subscript corresponds to the integer representations of the contents of the shift register elements. There are two paths leaving each state, corresponding to two possible values of the input message bit.

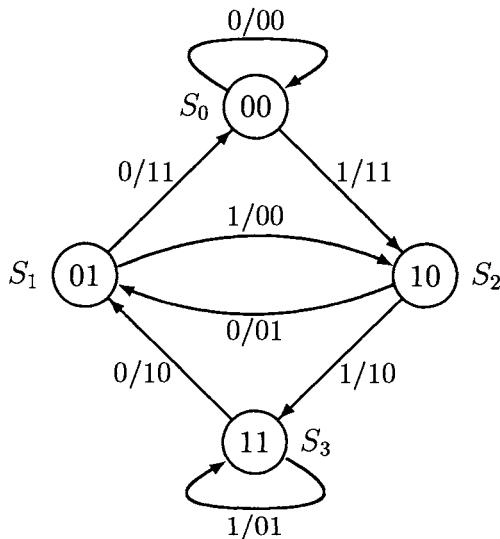


Fig. 3.12: State diagram for the $(2, 1)$ nonsystematic convolutional encoder from Fig. 3.1

The equivalent systematic encoder for this code in the observer canonical form is shown in Fig. 3.7. The state diagram for this

encoder is illustrated in Fig. 3.13.

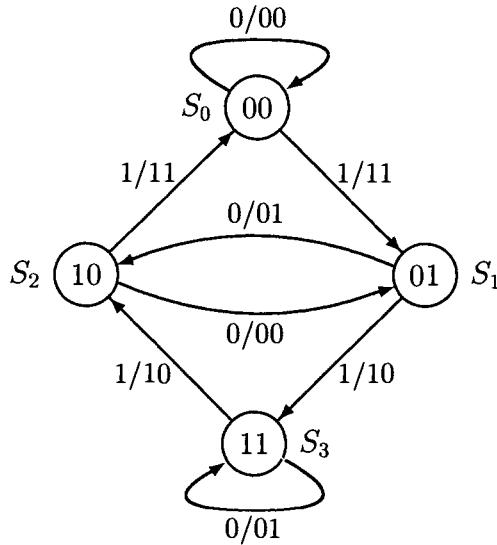


Fig. 3.13: State diagram for the (2,1) systematic encoder in Fig. 3.7

Note that the output sequences in both state diagrams in Figs. 3.12 and 3.13 are identical, as the encoders are equivalent. However, the mapping between the input message bits and the output code block is different for the two encoders.

3.9 Trellis Diagram

A *trellis diagram* is derived from the state diagrams by tracing all possible input/output sequences and state transitions. As an example, consider the trellis diagram for the (2,1) encoder described by the state diagram in Fig. 3.12. It is obtained by expanding in time every state transition and every path starting from all zero state (00). The resulting trellis is illustrated in Fig. 3.14.

When the first message is shifted into the register, the encoder can move either to state (10) if the input symbol is 1 or to state (00) if the input symbol is 0. When the next symbol enters the register

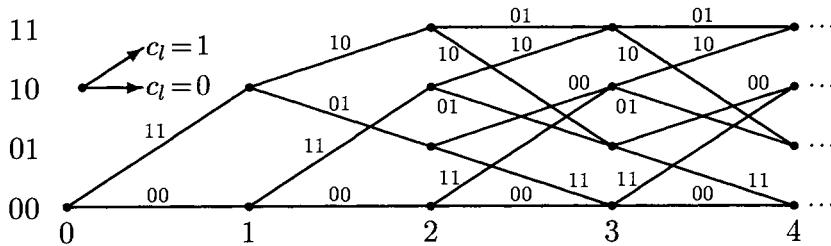


Fig. 3.14: Trellis diagram for the $(2, 1)$ nonsystematic encoder in Fig. 3.1

the encoder can be in one of four possible states (00) , (01) , (10) and (11) . The number of states is doubled upon every new shift. The encoder reaches the maximum possible number of states, 2^K after K time units. For $t > K$, the structure of trellis becomes repetitive. In the example, the encoder reaches the maximum number of states after two time units.

There are two branches emanating from each state corresponding to two different symbols. There are also two branches merging into each state. The transitions caused by an input symbol 0 are indicated by lower branches while the transitions caused by an input symbol 1 are indicated by upper branches. Each branch is labelled by the output block.

The output code sequence can be obtained by tracing a path in the trellis specified by the input information sequence. For example, if the input sequence is $\mathbf{c} = (11001)$, the output code sequence can be read out from the trellis

$$\mathbf{v} = (1110101111) \quad (3.76)$$

In general, the trellis of an (n, k) code has 2^k branches leaving each state and 2^k branches entering each state.

3.10 Distance Properties of Convolutional Codes

The error probability performance of convolutional codes depends on their distance properties. We consider here two types of distance. For hard decision decoding, the decoder operates with binary symbols and the code performance is measured by Hamming distance. A soft decision decoder receives quantized or analog signals from the demodulator and the decoding operation is based on Euclidean distance.

The *minimum free distance*, denoted by d_{free} , of a convolutional code is defined as the minimum Hamming distance between any two code sequences in the code. Since convolutional codes are linear, the Hamming distance between two code sequences is equal to the weight of their modulo-2 sum, which is another code sequence. Therefore, the minimum free distance is the minimum weight of all non-zero code sequences. In other words, the all-zero code sequence can be used as the reference sequence in determining the minimum free distance.

For example, for the code $(2, 1)$, represented by the trellis in Fig. 3.14, it is clear that the path $\mathbf{v} = (110111)$ is at the minimum Hamming distance from the all-zero path $\mathbf{0}$. Thus, the minimum free distance is $d_{\text{free}} = 5$.

The *minimum free Euclidean distance*, denoted by $d_{E,\text{free}}$, is defined as the minimum Euclidean distance between any two code sequences. The minimum free Euclidean distance depends both on the convolutional code trellis structure and the modulation type. For example if the BPSK modulation is used to map each binary symbol into a signal from the $\{-1, 1\}$ modulation signal set, the minimum free Euclidean distance for the $(2, 1)$ code with the trellis in Fig. 3.14 is $d_{E,\text{free}} = 2\sqrt{5}$.

3.11 Weight Distribution of Convolutional Codes

The weight distribution of a convolutional code is important for computing its error performance. We define A_i as the number of code sequences of weight i in the trellis that diverge from the all-zero path at a node and then remerge for the first time at a later node. The set

$$\{A_{d_{\text{free}}}, A_{d_{\text{free}}+1}, \dots, A_i, \dots\}$$

is called the weight distribution of a convolutional code.

The weight distribution can be computed by modifying the code state diagram. The modified state diagram is obtained by splitting the all-zero state S_0 into an initial state, S_{in} , and a final state, S_{out} , with the self-loop around S_0 being removed. Each path in the state diagram connecting the initial state S_{in} and the final state S_{out} represents a code sequence which diverges from and remerges with the all-zero state exactly once. The code sequence weight A_i represents the number of code sequences which diverge from the all-zero path at the same node and remerge for the first time at later nodes. In fact, A_i is equal to the number of paths of weight i in the modified state diagram which connect the initial and final state.

Let X be the indeterminate associated with the Hamming weight of the encoded output sequence i , Y the indeterminate associated with the Hamming weight of the information sequence j , and Z the indeterminate associated with every branch. Each branch in the modified state diagram is labelled with a branch gain $X^i Y^j Z$.

The modified state diagram with gain labels is called an *augmented state diagram*.

The weight distribution can be obtained from the generating function of the augmented state diagram. The augmented state diagram can be regarded as a signal flow graph and the generating function can be computed by Mason's rule [12]. Alternatively, the generating function can be obtained from a set of equations describing the state transitions in the augmented state diagram.

Example 3.10 Consider the $(2, 1)$ convolutional code with the state diagram shown in Fig. 3.12. The augmented state diagram is illustrated in Fig. 3.15.

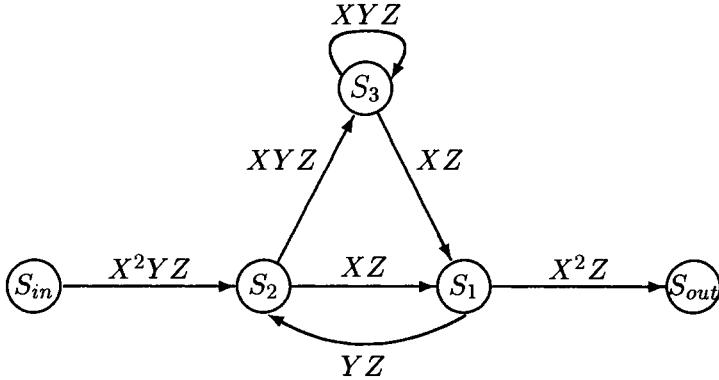


Fig. 3.15: Augmented state diagram of Fig. 3.12

The set of equations describing the state transitions in the augmented state diagram are

$$S_2 = YZS_1 + X^2YZS_{in} \quad (3.77)$$

$$S_1 = XZS_2 + XZS_3 \quad (3.78)$$

$$S_3 = XYZS_2 + XYZS_3 \quad (3.79)$$

$$S_{out} = X^2ZS_1 \quad (3.80)$$

Solving for S_3 from (3.79) we get

$$S_3 = \frac{XYZS_2}{1 - XYZ} \quad (3.81)$$

Substituting (3.81) into (3.78) we get

$$S_1 = \frac{XZS_2}{1 - XYZ} \quad (3.82)$$

Substituting (3.82) into (3.77) we get

$$S_2 = \frac{(1 - XYZ)X^2YZS_{in}}{1 - XYZ - XYZ^2} \quad (3.83)$$

Furthermore, substituting (3.82) into (3.80) we get for S_{out}

$$S_{out} = \frac{X^3 Z^2 S_2}{1 - XYZ} \quad (3.84)$$

Now, substituting (3.83) into (3.84) we get for the generating function $\mathbf{T}(X, Y, Z)$

$$\mathbf{T}(X, Y, Z) = \frac{S_{out}}{S_{in}} = \frac{X^5 Y Z^3}{1 - XYZ(1 + Z)} \quad (3.85)$$

we can write

$$\begin{aligned} \mathbf{T}(X, Y, Z) = & X^5 Y Z^3 + X^6 Y^2 (Z^4 + Z^5) \\ & + X^7 Y^3 (Z^5 + 2Z^6 + Z^7) \\ & + X^8 Y^4 (Z^6 + 3Z^7 + 3Z^8 + Z^9) + \dots \end{aligned}$$

Expanding $\mathbf{T}(X, Y, Z)$ in series, we can write

$$\begin{aligned} \mathbf{T}(X, Y, Z) = & X^5 Y Z^3 + X^6 Y^2 Z^4 + (X^6 Y^2 + X^7 Y^3) Z^5 \\ & + (2X^7 Y^3 + X^8 Y^4) Z^6 \\ & + (X^7 Y^3 + 3X^8 Y^4 + X^9 Y^5) Z^7 \\ & + (3X^8 Y^4 + 4X^9 Y^5 + X^{10} Y^6) Z^8 + \dots \end{aligned}$$

Note that the minimum free distance for the code is 5 and that the number of code sequences at this distance is $A_5 = 1$. The information sequence generating this code sequence has the Hamming weight of 1 and the code sequence contains three branches. Another information sequence of weight 2 produces a code sequence of weight 6 with four branches and so on.

If we ignore the branch lengths, by setting Z to 1, the generator function becomes

$$\mathbf{T}(X, Y) = X^5 Y + 2X^6 Y^2 + 4X^7 Y^3 + 8X^8 Y^4 + \dots \quad (3.86)$$

Furthermore, the weight distribution A_i can be obtained by setting $Y = 1$.

$$\mathbf{T}(X) = X^5 + 2X^6 + 4X^7 + 8X^8 + \dots \quad (3.87)$$

with $A_5 = 1, A_6 = 2, A_7 = 4$ and so on.

The total number of nonzero information bits on all paths of Hamming weight j is given by the partial derivative of $\mathbf{T}(X, Y, Z)$ with respect to Y

$$\left. \frac{\partial \mathbf{T}(X, Y, Z)}{\partial Y} \right|_{Y=1, Z=1} = X^5 + 4X^6 + 12X^7 + \dots \quad (3.88)$$

That is, on the path with a weight of 5 there is one information bit, on the path with a weight of 6, there are 4 information bits etc.

3.12 Punctured Convolutional Codes

In certain situations it is required to vary the code rate of a convolutional code and keep its trellis structure unchanged. Examples are unequal error protection (UEP) schemes in which one encoder and decoder with variable code rates are used. The code rate is varied by not transmitting certain code symbols, or *puncturing* the original code. Another reason for constructing punctured codes is that their trellises have simpler structure than those for the corresponding rate non-punctured codes.

Punctured codes are (n, k) convolutional codes derived from an $(n, 1)$ mother convolutional code. We will demonstrate the construction method on an example.

Let us consider the $(2, 1)$ encoder shown in Fig. 3.1 with the generator matrix $\mathbf{G}(D) = [1 + D^2 \ 1 + D + D^2]$ and the trellis diagram illustrated in Fig. 3.14. If one out of four encoder output symbols is punctured, the encoder produces three coded symbols for every two information bits. That is, every leading symbol of the rate $1/2$ code is punctured from every other branch. The new code is time varying and has a code rate of $2/3$.

The trellis diagram of the punctured $(3, 2)$ code is shown in Fig. 3.16. Note that X indicates the deleted symbols.

The set of code sequences in this punctured code is identical to the set of code sequences generated by the $2/3$ rate code with the

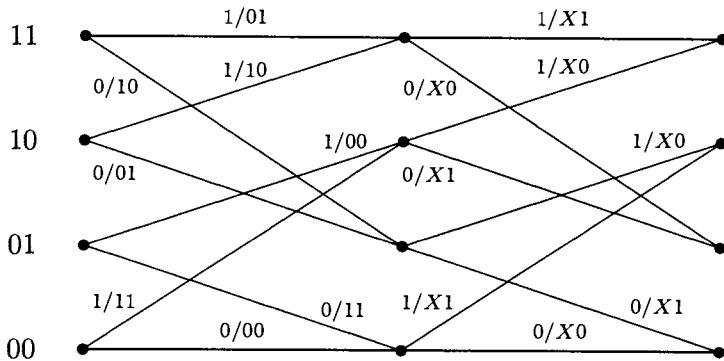


Fig. 3.16: Trellis diagram of a rate $2/3$ punctured code produced by periodically deleting symbols from a rate $1/2$ code

generator matrix

$$\mathbf{G}(D) = \begin{bmatrix} 1+D & 1+D & 1 \\ 0 & D & 1+D \end{bmatrix} \quad (3.89)$$

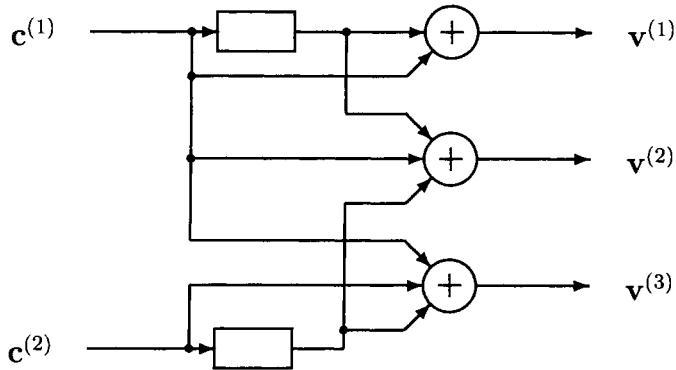
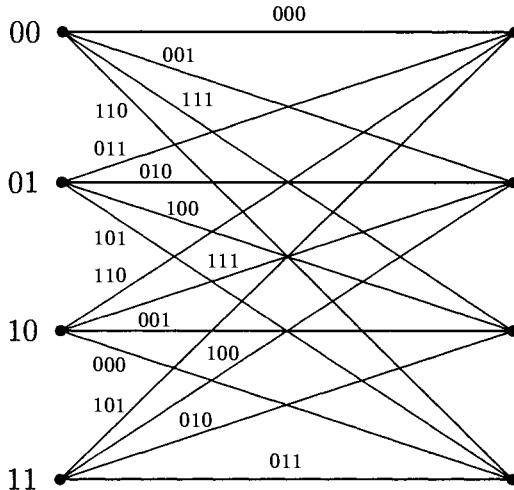
The encoder and the trellis diagram for this code are given in Figs 3.17 and 3.18, respectively. The trellis is more complex than the trellis for the punctured rate $2/3$ code shown in Fig. 3.16 since there are four paths entering each state rather than two. This leads to more complex encoding and decoding operation. Thus, punctured codes have an advantage, particularly for high rate applications.

The erasure of code symbols in the trellis in Fig. 3.16 can be described by a puncturing table

$$\mathbf{P} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \quad (3.90)$$

A zero in the puncturing table means that the code symbol is not transmitted. In the above example the first symbol in the second branch is not transmitted.

In general, a rate p/q punctured convolutional code can be constructed from an $(n, 1)$ convolutional code by deleting $np - q$ code

Fig. 3.17: Encoder for a rate $2/3$ codeFig. 3.18: Trellis diagram of a rate $2/3$ code

symbols from every np code symbols corresponding to p input information symbols. The $(n, 1)$ code is called the *mother code*. It is specified by the generator matrix

$$\mathbf{G}(D) = [\mathbf{g}^{(1)}(D) \ \mathbf{g}^{(2)}(D) \ \cdots \ \mathbf{g}^{(n)}(D)] \quad (3.91)$$

where

$$\mathbf{g}^{(i)}(D) = g_0^{(i)} + g_1^{(i)}D + \cdots + g_\nu^{(i)}D^\nu \quad (3.92)$$

where $1 \leq i \leq n$ and $g_l^{(i)} \in \{0, 1\}$, $l = 0 \dots \nu$.

The code rate of the punctured code, R_p , is defined by

$$R_p = \frac{p}{np - np + q}$$

The code symbol erasures are represented by the puncturing table

$$\mathbf{P} = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1p} \\ p_{21} & p_{22} & \cdots & p_{2p} \\ \vdots & & & \vdots \\ p_{n1} & p_{n2} & \cdots & p_{np} \end{bmatrix} \quad (3.93)$$

with $p_{ij} \in \{0, 1\}$, $1 \leq i \leq n$ and $1 \leq j \leq p$.

As the puncturing is performed periodically every np code symbols, p is called the *puncturing period*.

The puncturing tables and corresponding free distances for the punctured codes generated from the (2,1) mother code with the encoder memory of 6 are shown in Table 3.1. The generator matrix is

$$\mathbf{G}(D) = \begin{bmatrix} 1 + D^2 + D^3 + D^5 + D^6 & 1 + D + D^2 + D^3 + D^6 \end{bmatrix}$$

Table 3.1: Punctured convolutional codes

punctured code rate	puncturing table \mathbf{P}	d_{free}
2/3	11 10	6
3/4	110 101	5
4/5	1111 1000	4
5/6	11010 10101	4
6/7	111010 100101	3
7/8	1111010 1000101	3

Bibliography

- [1] G. Begin and D. Haccoun, “High Rate Punctured Convolutional Code Structure Properties and Construction Techniques”, *IEEE Trans. Commun.*, vol. COM-37, no. 12, Dec. 1989, pp 1813-1825.
- [2] Y. Bian, A. Popplewell and J. J. O'Reilly, “New Very High Rate Punctured Convolutional Codes”, *Elect. Lett.*, vol. 30, no. 14, July 1994, pp. 1119-1120.
- [3] J. B. Cain, G. C. Clark and J. M. Geist, “Punctured Convolutional Codes of Rate $(n - 1)/n$ and Simplified Maximum Likelihood Decoding”, *IEEE Trans. Inform. Theory*, vol. IT-25, no. 1, Jan. 1979, pp. 97-100.
- [4] D. J. Costello, Jr., “Constructions of Convolutional Codes for Sequential Decoding ”, Rep. EE-962, Univ. of Norte Dame, Norte Dame, IN, Aug 1969.
- [5] P. Elias, “Error-free Coding”, *IRE Trans. Inform. Theory*, vol. IT-4, 1954, pp.29-37.
- [6] G. D. Forney, Jr., “Convolutional Codes I: Algebraic Structure”, *IEEE Trans. Inform. Theory*, vol. IT-16, no. 6, Nov. 1970, pp. 720-738, and vol. IT-17, no. 3, May 1971, p. 360.
- [7] G. D. Forney, Jr., “Structural Analysis of Convolutional Codes via Dual Codes”, *IEEE Trans. Inform. Theory*, vol. IT-19, no. 4, July 1973, pp. 512-518.

- [8] J. Hagenauer, “Rate-compatible Punctured Convolutional Codes (RCPC codes) and Their Applications”, *IEEE Trans. Commun.*, vol. COM-36, no. 4, April 1988, pp.389-400.
- [9] R. Johannesson, and Z. X. Wan, “A Linear Algebraic Approach to Minimal Convolutional Encoder”, *IEEE Trans. Inform. Theory*, vol. IT-39, no. 4, July 1993, pp. 1219-1233.
- [10] L. H. C. Lee, “Convolutional Coding: Fundamentals and Applications”, Artech House, 1997.
- [11] S. Lin, and D.J. Costello, Jr., “Error Control Coding: Fundamentals and Applications”, Prentice-Hall, 1983.
- [12] S. Mason and Zimmermann H., Electronic Circuits, Signals and Systems, New York, Wiley, 1960.
- [13] J. L. Massey, and M. K. Sain, “Inverses of Linear Sequential Circuits”, *IEEE Trans. Comp.*, vol. 17, no. 4, Apr. 1968, pp. 330-337.
- [14] J. E. Porath, “Algorithms for Converting Convolutional Codes from Feedback to Feedforward and Vice Versa”, *Elect. Lett.*, vol. 25, no. 15, July 1989, pp. 1008-1009.
- [15] A. J. Viterbi and J. K. Omura, Principles of Digital Communications and Coding, McGraw Hill, 1979.
- [16] Y. Yasuda, Y. Kashiki and Y. Hirata, “High Rate Punctured Convolutional Codes for Soft-Decision Viterbi Decoding”, *IEEE Trans. Commun.*, vol. COM-32, no. 3, March 1984, pp. 315-319.

Chapter 4

Turbo Coding Performance Analysis and Code Design

4.1 Introduction

It is well known that a good trade-off between coding gain and complexity can be achieved by serial concatenated codes proposed by Forney [1]. A serial concatenated code is one that applies two levels of coding, an inner and an outer code linked by an interleaver. This approach has been used in space communications, with convolutional codes as the inner code and low redundancy Reed-Solomon codes as the outer code. The primary reason for using a concatenated code is to achieve a low error rate with an overall decoding complexity lower than that required for a single code of the corresponding performance. The low complexity is attained by decoding each component code separately. As the inner decoder generates burst errors an interleaver is typically incorporated between the two codes to decorrelate the received symbols affected by burst errors. Another application of concatenation is using a bandwidth efficient trellis code as an inner code [2] or concatenating two convolutional codes [3]. In decoding these concatenated codes, the inner decoder may use a soft-input/soft-output decoding algorithm to produce soft decisions for the outer decoder.

Turbo codes exploit a similar idea of connecting two codes and separating them by an interleaver [4]. The difference between

turbo and serial concatenated codes is that in turbo codes two identical systematic component codes are connected in parallel. The information bits for the second code are not transmitted thus increasing the code rate relative to a corresponding serial concatenated code. The primary reason for using a long interleaver in turbo coding is to generate a concatenated code with large block length which leads to a large coding gain. The turbo decoder consists of two concatenated decoders of the component codes separated by the same interleaver. The component decoders are based on a *maximum a posteriori* (MAP) probability algorithm or a *soft output Viterbi algorithm* (SOVA) generating a weighted soft estimate of the input sequence. The iterative process performs information exchange between the two component decoders. By increasing the number of iterations in turbo decoding, a bit error probability as low as 10^{-5} - 10^{-7} can be achieved at a signal-to-noise ratio close to the Shannon capacity limit.

In this chapter, we consider turbo and *serial concatenated convolutional codes* (SCCC). We present analytical upper bounds of code performance and propose design criteria for turbo and serial concatenated convolutional codes.

4.2 Turbo Coding

4.2.1 A Turbo Encoder

A turbo encoder is formed by parallel concatenation of two *recursive systematic convolutional* (RSC) encoders separated by a random interleaver [4].

The encoder structure is called parallel concatenation because the two encoders operate on the same set of input bits, rather than one encoding the output of the other. Thus turbo codes are also referred to as *parallel concatenated convolutional codes* (PCCC) [13]. A block diagram of a rate 1/3 turbo encoder is shown in Fig. 4.1.

The generator matrix of a rate 1/2 component RSC code can

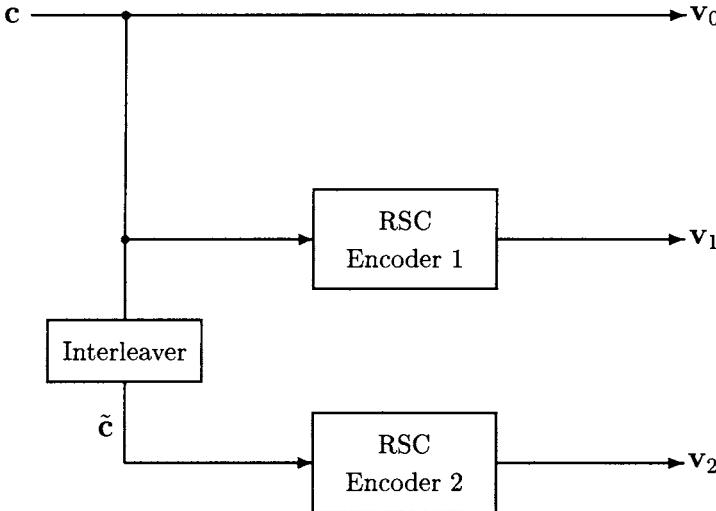


Fig. 4.1: A turbo encoder

be represented as

$$\mathbf{G}(D) = \begin{bmatrix} 1 & \frac{\mathbf{g}_1(D)}{\mathbf{g}_0(D)} \end{bmatrix} \quad (4.1)$$

where $\mathbf{g}_0(D)$ and $\mathbf{g}_1(D)$ are a feedback and feedforward polynomial with degree ν , respectively. In the encoder, the same information sequence is encoded twice but in a different order. The first encoder operates directly on the input sequence, denoted by \mathbf{c} , of length N . The first component encoder has two outputs. The first output, denoted by \mathbf{v}_0 , is equal to the input sequence since the encoder is systematic. The other output is the parity check sequence, denoted by \mathbf{v}_1 . The interleaved information sequence at the input of the second encoder is denoted by $\tilde{\mathbf{c}}$. Only the parity check sequence of the second encoder, denoted by \mathbf{v}_2 , is transmitted. The information sequence \mathbf{v}_0 and the parity check sequences of the two encoders, \mathbf{v}_1 and \mathbf{v}_2 , are multiplexed to generate the turbo code sequence. The overall code rate is $1/3$.

Example 4.1 A rate 1/3 turbo code encoder

A rate 1/3 turbo code encoder, based on a $(2, 1, 4)$ RSC code, is shown in Fig. 4.2. The component codes are identical $(2, 1, 4)$ RSC

codes with code rate 1/2 and memory order $\nu = 4$ (the number of states is $M_s=16$). The generator matrix of the RSC code is given by

$$\mathbf{G}(D) = \left[1, \quad \frac{1+D^4}{1+D+D^2+D^3+D^4} \right] \quad (4.2)$$

Let us assume that the input sequence is

$$\mathbf{c} = (1011001) \quad (4.3)$$

Then the two output sequences of the first component encoder are

$$\begin{aligned} \mathbf{v}_0 &= (1011001) \\ \mathbf{v}_1 &= (1110001) \end{aligned} \quad (4.4)$$

We assume that the interleaver permutes the information sequence to

$$\tilde{\mathbf{c}} = (1101010) \quad (4.5)$$

The parity check sequence of the second component encoder is

$$\mathbf{v}_2 = (1000000) \quad (4.6)$$

The turbo code sequence is given by

$$\mathbf{v} = (111, 010, 110, 100, 000, 000, 110) \quad (4.7)$$

4.2.2 Interleaving

The interleaver in turbo coding is a pseudo-random block scrambler defined by a permutation of N elements with no repetitions.

The first role of the interleaver is to generate a long block code from small memory convolutional codes. Secondly, it decorrelates the inputs to the two decoders so that an iterative suboptimum decoding algorithm based on information exchange between the two component decoders can be applied. If the input sequences to the two component decoders are decorrelated there is a high probability that after correction of some of the errors in one decoder some of the remaining errors should become correctable in the second decoder.

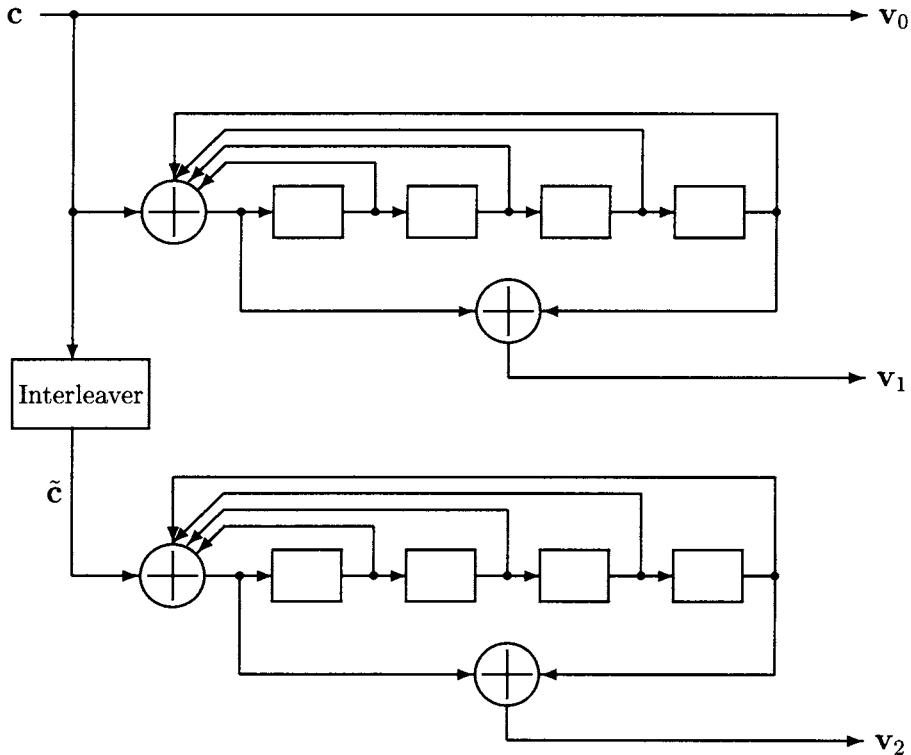


Fig. 4.2: A rate 1/3 turbo encoder

In a pseudo-random interleaver a block of N input bits is read into the interleaver and read out pseudo-randomly. The pseudo-random interleaving pattern must be available at the decoder as well.

4.2.3 Trellis Termination

A turbo encoder operating on each block of information bits can be regarded as a block encoder. Since the component codes are recursive, it is not possible to terminate the trellis by transmitting ν zero tail bits. Trellis termination means driving the encoder to the all-zero state. This is required at the end of each block to make sure that the initial state for the next block is the all-zero state.

The tail bits depend on the state of the component encoder after N information bits. A simple solution to this problem is shown in Fig. 4.3 [6]. A switch in each parallel component encoder is in position “A” for the first N clock cycles and in position “B” for ν additional cycles. This will drive the encoder to the all-zero state. Trellis termination is based on setting the input “ a_t ” to the shift register to zero. This will flush the register with zeros after ν shifts.

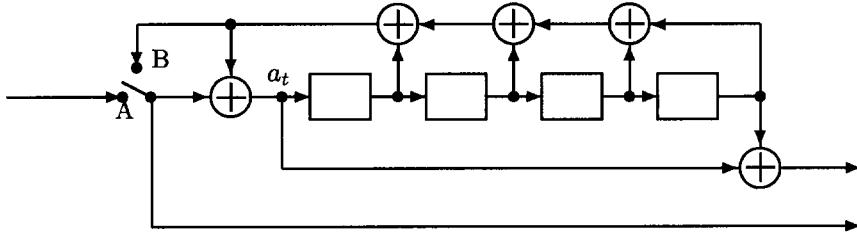


Fig. 4.3: Trellis termination

With a pseudo-random interleaver it is highly unlikely that this method will terminate both encoders simultaneously with only ν tail bits since the codes are recursive. Usually, only the first component encoder is forced to return to the all-zero state. The performance degradation produced by an unknown final state of the second encoder is negligible for a large interleaver size N [7]. A rate 1/3 turbo code with trellis termination is equivalent to a $(3(N + \nu), N)$ linear systematic block code.

In the remainder of the book we will assume that the first encoder is forced to the all-zero state. Special interleaver structures that can drive both encoders to the all-zero state are discussed in Chapter 7 and in [8]- [11].

4.2.4 High Rate Turbo Codes

In general, the two component codes and their rates are not necessarily the same. If the code rates of two component codes are denoted by R_1 and R_2 , respectively, the overall turbo code rate R will satisfy

$$\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2} - 1 \quad (4.8)$$

Various overall code rates such as $1/2$, $2/3$, $3/4$, $5/6$ and so on, can be obtained by puncturing the rate $1/3$ turbo encoder shown in Fig. 4.1.

When puncturing is considered, some output bits of \mathbf{v}_0 , \mathbf{v}_1 and \mathbf{v}_2 are deleted according to a chosen pattern defined by a puncturing matrix \mathbf{P} . For instance, a rate $1/2$ turbo code can be obtained by puncturing a rate $1/3$ turbo code. The commonly used puncturing matrix is given by

$$\mathbf{P} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (4.9)$$

where the puncturing period is 2. According to the puncturing matrix, the parity check digits from the two component encoders are alternately deleted. The punctured turbo code symbol at a given time consists of an information digit followed by a parity check digit which is alternately obtained from the first and the second component encoders.

An alternative approach in high rate turbo code design is to employ high rate component codes [12] [15].

Example 4.2 A rate $1/2$ turbo code encoder

A rate $1/2$ turbo code based on a $(3, 2, 4)$ RSC code is shown in Fig. 4.4. The generator matrix of the RSC code is given by

$$\mathbf{G}(D) = \begin{bmatrix} 1 & 0 & \frac{1+D^2+D^3+D^4}{1+D+D^4} \\ 0 & 1 & \frac{1+D+D^3+D^4}{1+D+D^4} \end{bmatrix} \quad (4.10)$$

The component codes are two identical $(3, 2, 4)$ RSC codes with code rate $2/3$ and memory order $\nu = 4$. The information sequences \mathbf{c}_0 and \mathbf{c}_1 are encoded by the first encoder to generate the first parity check sequence \mathbf{v}_2 . The interleaved sequences $\tilde{\mathbf{c}}_0$ and $\tilde{\mathbf{c}}_1$ are encoded by the second encoder to produce the second parity check sequence \mathbf{v}_3 . Then the information sequences \mathbf{v}_0 and \mathbf{v}_1 , and the parity check sequences of the two component encoders \mathbf{v}_2 and \mathbf{v}_3 , are multiplexed to generate the code sequence for a rate $1/2$ turbo encoder.

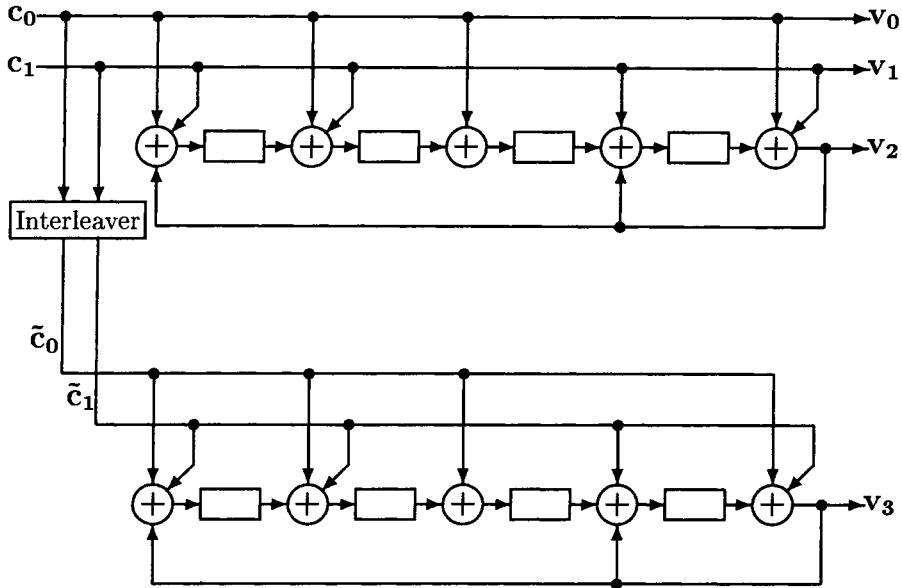


Fig. 4.4: A rate 1/2 turbo encoder

4.3 Performance Upper Bounds of Turbo Codes

The weight distribution of a turbo code can be used to compute error performance bounds for maximum likelihood decoding. As a turbo code can be represented by an equivalent block code if the component encoders are forced to the all-zero state at the end of each block, we first consider the weight distribution of its equivalent block code in this section. Subsequently, we will apply it in the calculation of bit error probability bounds on an additive white Gaussian noise channel.

4.3.1 Conditional WEF's of Average Turbo Codes

In order to evaluate turbo code error performance upper bounds, we need to calculate the conditional WEF's of the equivalent block

code.

A turbo codeword consists of the input information sequence and the parity check sequences from the first and second component encoders. Let w be the weight of an input information sequence, and z_1 and z_2 be the weights of the first and second parity check sequences, respectively. The weight of the corresponding codeword will be $d = w + z_1 + z_2$. If the conditional WEF's of the component code are known, it can be used to determine the weight of the first parity check sequence. However the weight of the second parity check sequence will not only depend on the weight of the input information sequence, but also on how the information bits have been permuted by the interleaver. The turbo code conditional WEF's cannot be uniquely determined because of various interleaver structures.

A simple solution to this problem is obtained by using a *uniform interleaver*, which is based on a probabilistic analysis of the ensemble of all interleavers [13].

A uniform interleaver is a probabilistic device which maps a given input sequence of length N and weight w into its all distinct $\binom{N}{w}$ permutations with equal probability of $1/\binom{N}{w}$.

For a uniform interleaver of size N , there are $N!$ possible permutations of binary sequences with length N , each of which has a probability $1/N!$. If the weight of an input sequence is w , there will be $w!(N-w)!$ permutations which generate the same output sequence. Thus the probability for each distinct permutation is

$$\frac{w! (N-w)!}{N!} = \frac{1}{\binom{N}{w}} \quad (4.11)$$

The uniform interleaver makes independent weight distributions of the parity check sequences generated by the first and second encoders. It enables the computation of the weight distribution of an average turbo code from the weight distribution of the component codes.

The conditional WEF's of the equivalent block code of a turbo

code with a uniform interleaver is given by

$$A_w(Z) = \frac{A_w^{c_1}(Z) \cdot A_w^{c_2}(Z)}{\binom{N}{w}} \quad (4.12)$$

where $A_w^{c_1}(Z)$ and $A_w^{c_2}(Z)$ are the conditional WEF's of the equivalent block codes of the first and second component codes, respectively. Since the conditional WEF's of the turbo code are averaged over the ensemble of interleavers of length N , they are also called the conditional WEF's of the “average” turbo code. If two identical component encoders are employed in a turbo encoder, it is appropriate to let

$$A_w^{c_1}(Z) = A_w^{c_2}(Z) = A_w^c(Z) \quad (4.13)$$

4.3.2 Conditional WEF's of Component Codes

The conditional WEF's of the equivalent block code of a turbo component code can be obtained from the code transfer function. For a component recursive systematic convolutional code, the code transfer function generated from the code augmented state diagram is given by [13]

$$T(W, Z, L) = \sum_{w,z,l} T_{w,z,l} W^w Z^z L^l \quad (4.14)$$

where $T_{w,z,l}$ is the number of error paths in the code trellis with an input sequence of weight w , parity check sequence of weight z and error path length l . A single error path is a path that diverges from the all-zero path in the code trellis and then remerges with the all-zero path within a finite number of branches. For the equivalent block code, an error path associated with an error event consists of all possible combinations of all single error paths. We call such an error path a compound error path.

Let us consider a compound error path as shown in Fig. 4.5. The compound error path concatenates n single error paths with a

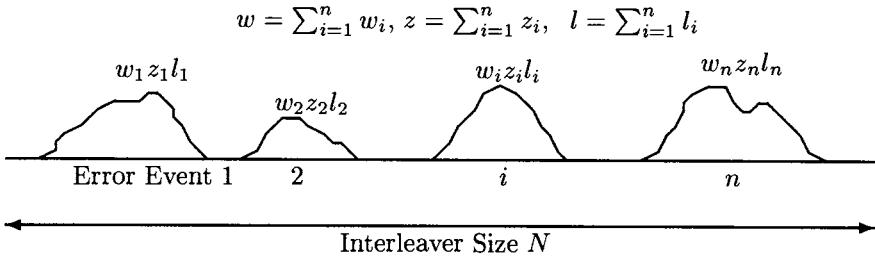


Fig. 4.5: A compound error path

total input weight of w , parity check weight of z , and error path length l within the block length N , equal to the interleaver size. It is clear that suitable sequences of zeros will connect these single error events so that the total length of the path equals N . Since the positions of the zero sequences do not affect the information and parity weight profile of the compound error path, for a combination of n error paths with total length l , there are

$$\binom{N - l + n}{n} \quad (4.15)$$

ways to obtain the same conditional WEF's.

Let $T(W, Z, L, E)$ be the modified transfer function of the equivalent block code of the component code which enumerates all possible combinations of single error paths in the trellis. The modified transfer function can be represented as

$$T(W, Z, L, E) = \sum_{w,z,l,n} T_{w,z,l,n} W^w Z^z L^l E^n \quad (4.16)$$

where $T_{w,z,l,n}$ is the number of error paths in the trellis produced by an information sequence of weight w , with parity check weight z , path length l and consisting of n concatenated single error paths.

The conditional WEF $A_w^c(Z)$ of the equivalent block code of the component convolutional code is given by

$$A_w^c(Z) = \sum_z A_{w,z} Z^z \quad (4.17)$$

where

$$A_{w,z}^c = \sum_{l,n} \binom{N - l + n}{n} T_{w,z,l,n} \quad (4.18)$$

For N much larger than the memory of the convolutional code, the conditional WEF of the equivalent block code can be approximated by

$$\begin{aligned} A_w^c(Z) &\approx \sum_{n=1}^{n_{\max}} \binom{N}{n} \sum_z Z^z \sum_l T_{w,z,l,n} \\ &= \sum_{n=1}^{n_{\max}} \binom{N}{n} \sum_z T_{w,z,n} Z^z \end{aligned} \quad (4.19)$$

where n_{\max} is the largest number of single error paths concatenated to produce a compound error path with an input sequence of weight w . $T_{w,z,n}$ is the number of error paths with information weight w , parity check weight z and number of concatenated error paths n . It is given by

$$T_{w,z,n} = \sum_l T_{w,z,n,l}. \quad (4.20)$$

Let

$$A(w, Z, n) = \sum_z T_{w,z,n} Z^z \quad (4.21)$$

be the redundancy weight enumerating function of the component code generated by concatenating n single error paths with total information weight w . The approximate conditional WEF of the equivalent block code is thus

$$A_w^c(Z) \approx \sum_{n=1}^{n_{\max}} \binom{N}{n} A(w, Z, n) \quad (4.22)$$

4.3.3 Average Upper Bounds on Bit Error Probability

Inserting (4.22) into (4.12) we get for the conditional WEF of the turbo code

$$A_w(Z) = \frac{A_w^c(Z) \cdot A_w^c(Z)}{\binom{N}{w}}$$

$$\approx \sum_{n_1=1}^{n_{\max}} \sum_{n_2=1}^{n_{\max}} \frac{\binom{N}{n_1} \binom{N}{n_2}}{\binom{N}{w}} A(w, Z, n_1) A(w, Z, n_2)$$

where

$$\frac{\binom{N}{n_1} \binom{N}{n_2}}{\binom{N}{w}} \quad (4.23)$$

is the contribution of each compound error path to the conditional WEF. Using the approximation of the binomial coefficient

$$\binom{N}{n} \approx \frac{N^n}{n!} \quad (4.24)$$

the conditional WEF of the turbo code can be rewritten as

$$A_w(Z) = \sum_{n_1=1}^{n_{\max}} \sum_{n_2=1}^{n_{\max}} \frac{w!}{n_1! \cdot n_2!} \cdot N^{n_1+n_2-w} \cdot A(w, Z, n_1) A(w, Z, n_2) \quad (4.25)$$

It is clear that, for large N , the conditional WEF is dominated by the terms which have the highest power of N . This can be achieved if $n_1 = n_2 = n_{\max}$. Thus (4.25) is further approximated by

$$A_w(Z) \approx \frac{w!}{n_{\max}!^2} N^{2n_{\max}-w} [A(w, Z, n_{\max})]^2 \quad (4.26)$$

Considering the bit error probability upper bounds (2.29) and (2.30), we get for the union upper bound

$$P_b(e) \leq \sum_{w=w_{\min}}^N \frac{w \cdot w!}{2 \cdot n_{\max}!^2} N^{2n_{\max}-w-1} \cdot W^w [A(w, Z, n_{\max})]^2 \Big|_{W=Z=e^{-R \frac{E_b}{N_0}}} \quad (4.27)$$

and the tighter union upper bound

$$P_b(e) \leq \sum_{w=w_{\min}}^N Q \left(\sqrt{2d_{\min} R \frac{E_b}{N_0}} \right) e^{d_{\min} R \frac{E_b}{N_0}} \frac{w \cdot w!}{n_{\max}!^2} N^{2n_{\max}-w-1} \cdot W^w [A(w, Z, n_{\max})]^2 \Big|_{W=Z=e^{-R \frac{E_b}{N_0}}} \quad (4.28)$$

where w_{\min} is the minimum information weight in error paths of the component codes. Since the uniform interleaver is used in the performance analysis, these upper bounds are also called average upper bounds.

Example 4.3 Turbo code bit error probability average upper bound.

Consider a rate 1/3 turbo code of memory order 2. The generator matrix of the turbo component code is given by

$$\mathbf{G}(D) = \left[1, \quad \frac{1+D^2}{1+D+D^2} \right] \quad (4.29)$$

The average upper bound on the bit error probability of the turbo code with interleaver size 500 is evaluated. The result is shown in Fig. 4.6.

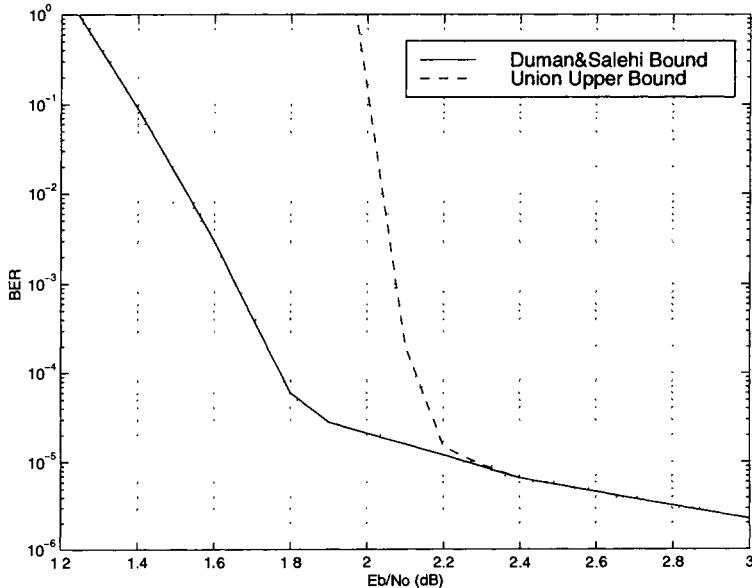


Fig. 4.6: Bit error probability upper bounds for a turbo code with interleaver size 500

Note that the average upper bound is based on union bound techniques. It diverges when the ratio of information bit energy

to noise power spectral density E_b/N_0 drops below the threshold determined by the computation cutoff rate R_0 , especially for large interleaver sizes. For a turbo code with rate $R = 1/3$, this threshold for E_b/N_0 is given by [17]

$$\begin{aligned}\frac{E_b}{N_0} &= -\frac{1}{R} \ln(2^{1-R} - 1) \\ &= 2.03 dB\end{aligned}\quad (4.30)$$

For E_b/N_0 above the computation cutoff rate threshold, the average upper bound can be used to estimate the code performance. In this region, evaluation of the bound requires only the first several terms of the distance spectrum in the summation. For E_b/N_0 below the threshold, the upper bounds will be useless for accurate performance evaluations for turbo codes with large interleaver sizes. To overcome the divergence problem of the average upper bound, several improved bounding techniques have been proposed. Tighter bounds for a larger range of E_b/N_0 extending below the cutoff rate threshold have been derived by Duman and Salehi in [18] and by Viterbi in [19] based on variants of Gallager's exponential bound. Sason and Shamai proposed ensemble upper bounds based on the tangential-sphere bound, where the ensemble is generated by a uniform choice of both the interleaver and the component codes [20]. In Fig. 4.6 Duman and Salehi's upper bound is also shown.

4.3.4 Interleaving Performance Gain

Using the analysis tools developed in the previous sections, we now investigate the role played by recursive systematic convolutional codes as component codes in turbo coding.

We begin with comparing error performance of recursive systematic convolutional codes and *nonrecursive convolutional* (NRC) or feedforward convolutional codes. Then, through an analytical upper-bounding technique, we will show that only a turbo code with RSC component codes can produce an interleaving performance gain, which is crucial for overall performance gain.

In order to compare the performance of RSC and NRC codes, we define two sets of error coefficients A_d and w_d for convolutional codes which can be obtained from the code transfer function $T(X, Y)$ [21]. Let

$$\begin{aligned} T(X, Y) \Big|_{Y=1} &= \sum_{d=d_{\text{free}}}^{\infty} A_d X^d \\ \frac{\partial T(X, Y)}{\partial Y} \Big|_{Y=1} &= \sum_{d=d_{\text{free}}}^{\infty} w_d X^d \end{aligned} \quad (4.31)$$

where d_{free} is the code free distance, A_d is the number of code sequence with weight d and w_d is the total weight of information sequences which generate code sequences of weight d . The two sets of error coefficients A_d and w_d can be used to derive the code word error probability and bit error probability upper bounds, respectively.

In [22], various RSC and NRC codes have been investigated and their error coefficients compared. It is shown that the set of error coefficients A_d is the same for RSC and NRC codes, since both RSC and NRC encoders generate the same set of code sequences. This results in the same word error probability for both codes. However, the set of w_d for RSC codes is different from that of NRC codes. This is due to a different input-output weight correspondence between RSC and NRC encoders, which leads to different bit error probability performance. In general, for code rates $R \leq 2/3$, the first two coefficients $w_{d_{\text{free}}}$ and $w_{d_{\text{free}}+1}$ of RSC codes are larger than those of NRC codes. Therefore, at high SNR's, the bit error probability performance of NRC codes is a little better than that of RSC codes. On the other hand, with increasing weight d , the value of w_d for RSC codes grows more slowly than that for NRC codes. Thus, at low SNR's, the bit error probability performance of RSC codes is superior to that of NRC codes.

The difference in the input-output weight correspondence between RSC and NRC encoders can also be explained by the difference in the minimum weight of information sequences generating a finite weight code sequence.

For NRC codes any information sequence of weight one will generate a finite weight code sequence. On the other hand RSC

codes may map finite weight information sequences into infinite weight code sequences. The generator matrix of RSC codes contains at least one entry which is a rational fraction of the unit delay operator D , such as $\mathbf{g}_1(D)/\mathbf{g}_0(D)$. The encoder thus generates a sequence of infinite weight in response to a finite weight information sequence, except if the polynomial representation of the information sequence is a multiple of the rational function denominator $\mathbf{g}_0(D)$. Since the rational function denominator is of the form $1 + \dots + D^\nu$, where ν is the encoder memory, it cannot divide a polynomial of the form D^j for any j . Therefore there are no finite weight code sequences generated by weight one information sequences in these codes. The minimum weight information sequence generating a finite weight code sequence is two.

If NRC encoders are employed in turbo coding, the minimum information weight in error paths is $w_{\min} = 1$. The largest number of single error paths in a compound error path with an information weight w is $n_{\max} = w$. In this case the conditional WEF is given by

$$A(w, Z, w) = [A(1, Z, 1)]^w \quad (4.32)$$

From (4.27), the code bit error probability is upper-bounded by

$$P_b(e) \leq \sum_{w=1}^N \frac{N^{w-1}}{2(w-1)!} W^w [A(1, Z, 1)]^{2w} \Big|_{W=Z=e^{-R \frac{E_b}{N_0}}} \quad (4.33)$$

The above upper bound shows that, for NRC component codes, error paths with information weight $w = 1$ and their compound error paths have a dominant effect on the turbo code bit error probability. At high SNR's the single error paths with $w = 1$ have the dominant contributions to the error performance. In this case, the bit error probability is independent of N , as the factor $N^{w-1} = 1$. Thus no performance gain can be achieved by interleaving for these codes. The same conclusion applies to turbo codes with component nonsystematic block codes [13].

In an RSC encoder, the minimum information weight for error paths is $w_{\min} = 2$. The largest number of single error paths in a compound error path with an information weight w is $n_{\max} = \lfloor w/2 \rfloor$, where $\lfloor x \rfloor$ means “integer part of x ”. We now separate the

analysis of terms in the sum in (4.27) into odd and even weight w . For odd values of w , i.e., $w = 2i + 1$, a term in the sum in (4.27) can be expressed as

$$\frac{(2i+1)(i+1)}{2} \binom{2i+1}{i} N^{-2} W^{2i+1} [A(2i+1, Z, i)]^2 \quad (4.34)$$

whereas, for even values of $w = 2i$, this term becomes

$$\begin{aligned} & i \binom{2i}{i} N^{-1} W^{2i} [A(2i, Z, i)]^2 \\ &= i \binom{2i}{i} N^{-1} W^{2i} [A(2, Z, 1)]^{2i} \end{aligned} \quad (4.35)$$

Comparing (4.34) and (4.35), it is clear that, for large N , the terms with odd w are negligible, since they depend on N^{-2} while the terms with even w depend on N^{-1} . Hence, for turbo codes with RSC component codes, the bit error probability is upper-bounded by

$$P_b(e) \leq \sum_{i=1}^{\lfloor \frac{N}{2} \rfloor} i \binom{2i}{i} N^{-1} W^{2i} [A(2, Z, 1)]^{2i} \Big|_{W=Z=e^{-R \frac{E_b}{N_0}}} \quad (4.36)$$

From the upper bound it can be seen that the performance of a turbo code with RSC component codes is affected by the interleaver size N . By increasing the interleaver size N times, the bit error probability is reduced by a factor N . This is called the *interleaving performance gain* [13].

To illustrate the interleaving performance gain, consider the turbo code in Example 4.3. The bit error probability average upper bounds for the turbo code with various interleaver sizes of 128, 256 and 512 are shown in Fig. 4.7. It is clear that the asymptotic bit error probability goes down with increasing the interleaver size.

4.3.5 Effective Free Distance

For turbo codes with RSC component encoders, the bit error probability upper bound shows that the error paths with minimum information weight $w_{\min} = 2$ and their compound error paths dominate

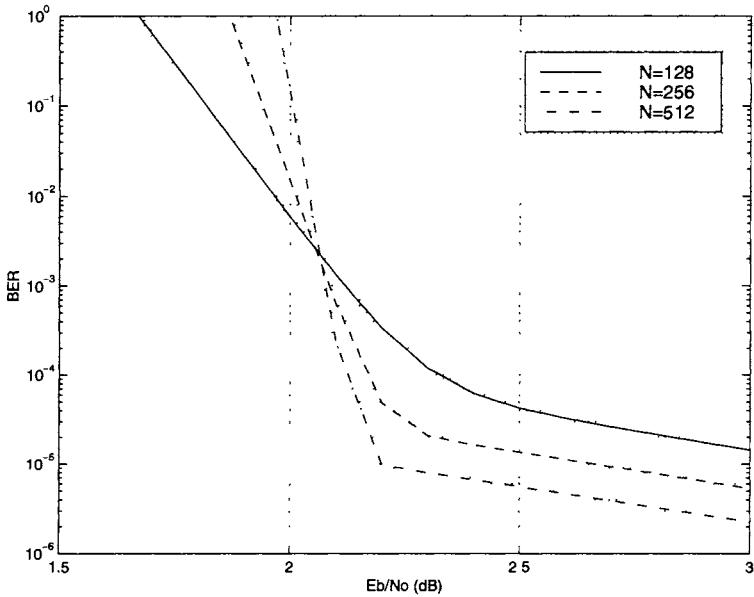


Fig. 4.7: Bit error probability upper bounds for a turbo code with various interleaver sizes

the turbo code bit error probability performance. Let z_{\min} denote the lowest weight of the parity check sequence in error paths of RSC component encoders generated by an information sequence with weight 2. The weight enumerating function of single error paths with information weight 2 for RSC component codes is given by [13]

$$\begin{aligned} A(2, Z, 1) &= Z^{z_{\min}} + Z^{2z_{\min}-2} + Z^{3z_{\min}-4} + \dots \\ &= \frac{Z^{z_{\min}}}{1 - Z^{z_{\min}-2}} \end{aligned} \quad (4.37)$$

Substituting (4.37) into (4.36) and setting $W = Z = H$, the turbo code bit error probability upper bound can be rewritten as

$$P_b(e) \leq \sum_{i=1}^{\lfloor \frac{N}{2} \rfloor} i \binom{2i}{i} N^{-1} \cdot \left. \frac{(H^{2+2z_{\min}})^i}{(1 - H^{z_{\min}-2})^{2i}} \right|_{H=e^{-R \frac{E_b}{N_0}}} \quad (4.38)$$

The above bound clearly shows that the most important parameter which has a significant influence upon turbo code performance

is z_{\min} . The *effective free distance* of a turbo code is defined as [23] [13]

$$d_{\text{free,eff}} = 2 + 2z_{\min} \quad (4.39)$$

It is the lowest weight of the turbo code sequence generated by an information sequence of weight 2 with a uniform interleaver. The effective free distance plays a role similar to that of free distance for convolutional codes.

For a turbo code with a uniform interleaver, the free distance dominates the error performance at high SNR's. The free distance may be smaller than the effective free distance $d_{\text{free,eff}}$. In this case the code sequence associated with the free distance is generated by an information sequence of weight $w \geq 3$. However, since a random interleaver is more likely to break information sequences with a high weight than a low weight, it can make the error coefficient shown in (4.41) of the free distance negligibly small. Therefore, for a turbo code with a random interleaver, the bit error probability at high SNR's is mainly determined by the code effective free distance. Effectively, this means that a random interleaver will break error paths corresponding to the free distance of uniform interleavers, so that the free distance becomes equal to the effective free distance.

From the above discussion we can conclude that the component encoders for turbo codes must be recursive, and that they should be chosen to maximize z_{\min} and thus the effective free distance $d_{\text{free,eff}}$, especially when the turbo code works in the region of high SNR's.

4.4 Turbo Code Performance Evaluation

In previous sections, turbo code performance is discussed based on the asymptotic bounds of the bit error probability. We now evaluate turbo code performance based on distance spectrum and its average upper bounds.

As in (2.26) the turbo code bit error probability upper bound

can be expressed based on the code distance spectrum by

$$P_b(e) \leq \sum_{d=d_{\text{free}}} B_d Q\left(\sqrt{2dR \frac{E_b}{N_0}}\right) \quad (4.40)$$

where the set of all pairs of (d, B_d) is the code distance spectrum. For turbo codes, the error coefficients B_d can be represented by

$$\begin{aligned} B_d &= \sum_{d=w+z} \frac{w}{N} \cdot A_{w,z} \\ &= \sum_{d=w+z_1+z_2} \frac{w}{N} \cdot \frac{A_{w,z_1}^c \cdot A_{w,z_2}^c}{\binom{N}{w}} \end{aligned} \quad (4.41)$$

where $A_{w,z}^c$ is the IRWEF coefficient for the component code and is given by (4.18).

For performance evaluation and comparison, we consider two turbo codes, denoted by TC1 and TC2, with generator matrices $\mathbf{G}_1(D)$ and $\mathbf{G}_2(D)$, respectively. The generator matrices are given by

$$\begin{aligned} \mathbf{G}_1(D) &= \left[1, \quad \frac{1+D^2}{1+D+D^2} \right] \\ \mathbf{G}_2(D) &= \left[1, \quad \frac{1+D+D^2}{1+D^2} \right] \end{aligned} \quad (4.42)$$

The turbo encoders are shown in Figs. 4.8 and 4.9, respectively.

In the performance evaluation, we consider interleavers with small sizes. This is due to the performance upper bound diverging severely from the actual bit error probability obtained by simulation in the region of small SNR's for turbo codes with large interleaver sizes. Using the transfer function method, the distance spectra of the component code of TC1 and the turbo code TC1 are calculated. The results for interleaver sizes of 20 and 50 are shown in Fig. 4.10. The bit error probability upper bounds obtained from (4.40) are plotted in Fig. 4.11. It is apparent from Fig. 4.10 that the shape of the distance spectrum of the turbo code is very similar to the component code. However, for small distance values, the

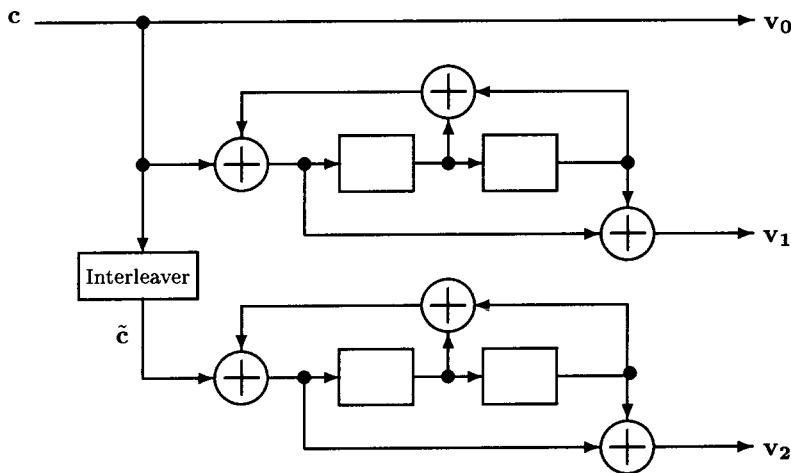


Fig. 4.8: Turbo encoder TC1

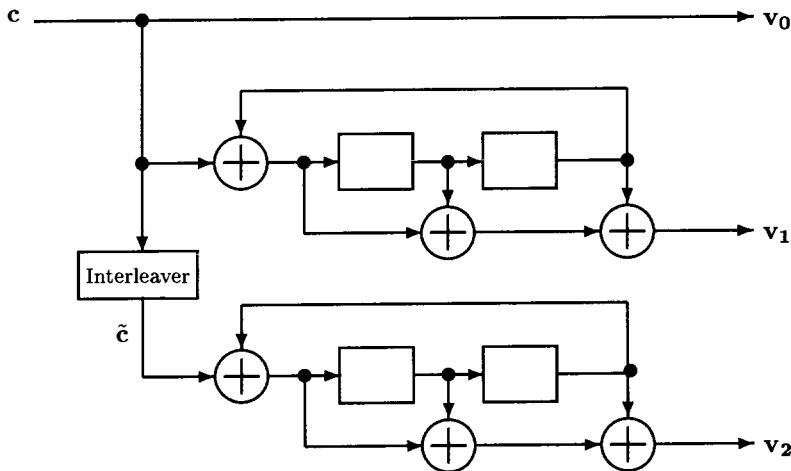


Fig. 4.9: Turbo encoder TC2

error coefficients of the turbo code are reduced by a factor between 10^{-3} to 10^{-5} for interleaver size 50 and between 10^{-2} to 10^{-3} for interleaver size 20, relative to the component code. This results in significant performance improvements, as shown in Fig. 4.11. This is consistent with the observation in the previous section that the

interleaver gain is proportional to the $1/N$.

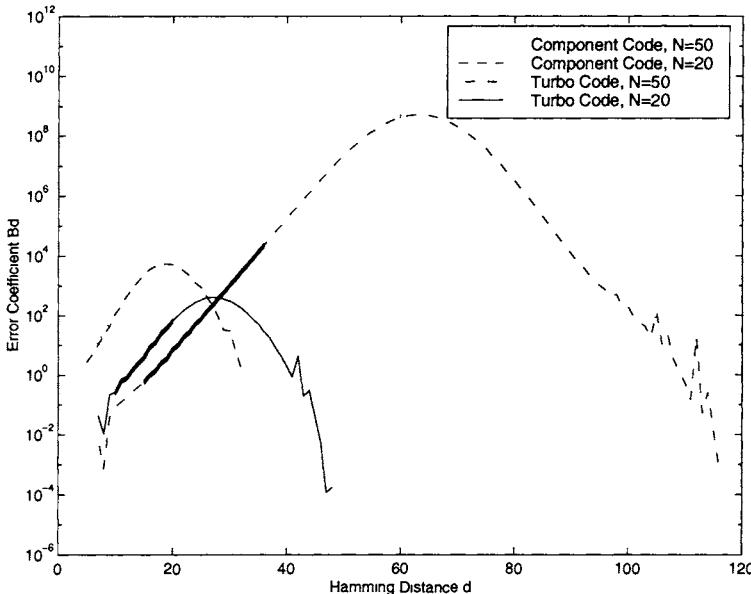


Fig. 4.10: Distance spectra for component code of TC1 and turbo code TC1 with interleaver sizes of 20 and 50

One may notice also that the distance spectrum is approximately symmetrical with respect to the middle distance spectral line. Since the error function decreases with increasing Hamming distance, only the distance spectral lines on the left side are important for the error performance.

In computing the union upper bound, each term in the sum on the right-hand side of (4.40) depends on the corresponding distance spectral line. The contribution of the i th term to the overall bit error probability, denoted by $P_i(e)$, can be written as

$$P_i(e) = B_{d_i} Q \left(\sqrt{2d_i R \frac{E_b}{N_0}} \right) \quad (4.43)$$

Its relative contribution to the total bit error probability can be represented as

$$\frac{P_i(e)}{\sum_i P_i(e)} \cdot 100 \quad (4.44)$$

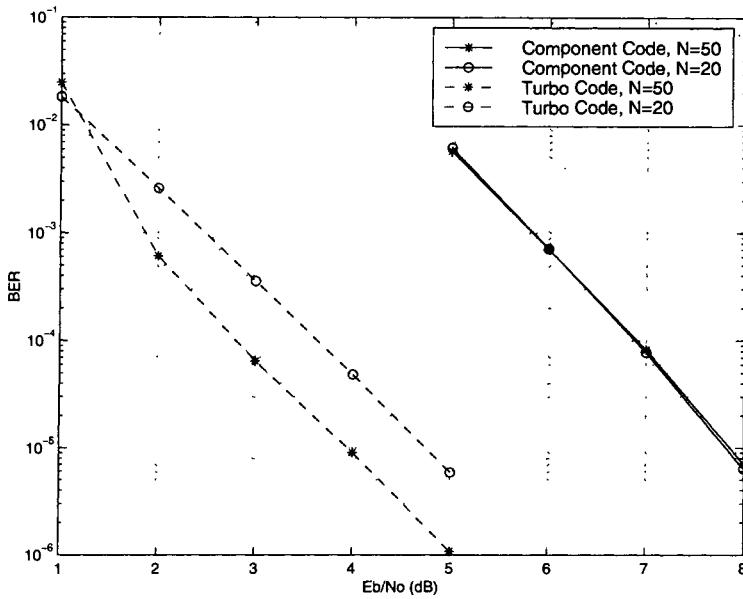


Fig. 4.11: Bit error probability upper bounds for component code of TC1 and turbo code TC1 with interleaver sizes of 20 and 50

For turbo code TC1, the relative contribution of each distance spectral line to the overall bit error probability is shown in Fig. 4.12 for interleaver size 20 and in Fig. 4.13 for interleaver size 50. From Figs. 4.12 and 4.13, it is clear that the bit error probability at low SNR's is dominated by a number of medium weight spectral line error coefficients B_d . The Hamming distances of these spectral lines are between 10 to 20 for interleaver size 20 and between 15 to 35 for interleaver size 50. At medium to high SNR's, the first several spectral lines have a significant effect on bit error probability. At very high SNR's, the free distance d_{free} determines the error performance. So the turbo code error performance is determined by a number of the medium distance spectral lines for low SNR's and the first several spectral lines for high SNR's.

The effect of distance spectrum on error probability can be explained by comparing the distance spectra of TC1 with interleaver sizes of 20 and 50, shown in Fig. 4.10. In the low to medium region of distances, up to 30, error coefficients for the code with $N = 50$

are lower than those for the code with $N = 20$. However, for large distances, the error coefficients for the code with $N = 50$ are higher. In spite of this, the code with $N = 50$ has better performance than the code with $N = 20$ in the medium to high SNR region as shown in Fig. 4.11. This property remains invariant for various codes. Consequently, we can conclude that only error coefficients at low to medium distances are significant for error performance. We call the part of the spectrum that gives a considerable contribution to the error probability the *significant spectral lines*. Significant spectral lines are shown by bold thick lines in Fig. 4.10.

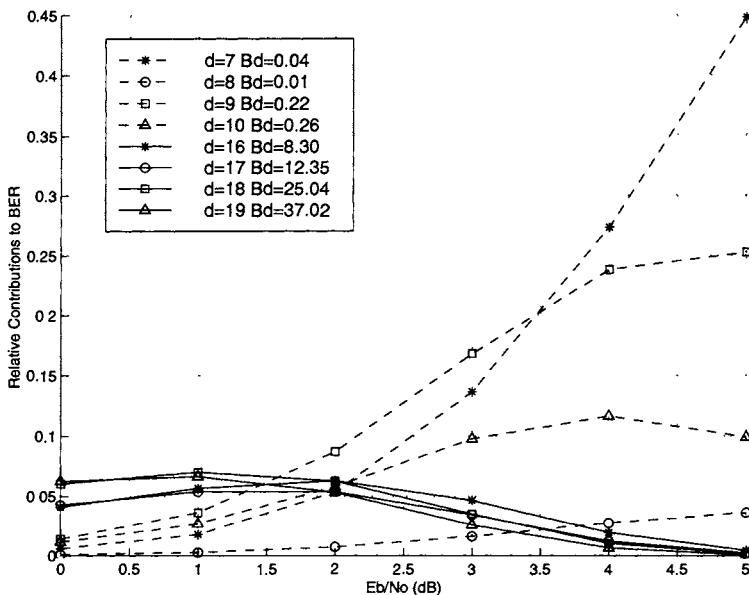


Fig. 4.12: Relative contributions of various distance spectral lines to overall bit error probability for turbo code TC1 with interleaver size 20

For performance comparison, we show the distance spectra and the bit error probability upper bounds of turbo codes TC1 and TC2 in Figs. 4.14 and 4.15. From these figures, it can be seen that the error coefficients of TC1 at the low to medium Hamming distances are much smaller than those of TC2. Therefore, TC1 will have a superior error performance. With the increase of the

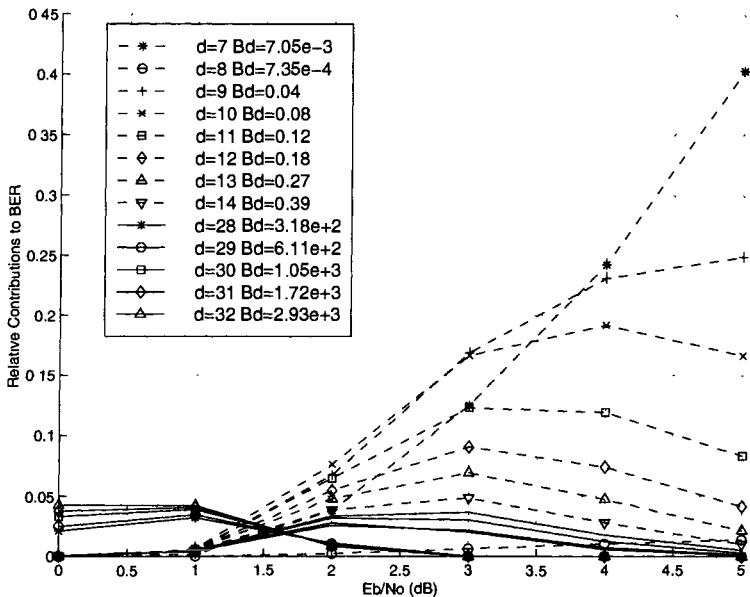


Fig. 4.13: Relative contributions of various distance spectral lines to overall bit error probability for turbo code TC1 with interleaver size 50

interleaver size, the error performance improvement achieved by TC1 over TC2 is increased. Fig. 4.14 also shows that the relative distance spectrum properties of various codes of a given memory order remain invariant with increasing interleaver size.

From these figures it can be observed that the impact of error coefficient B_d on the code error performance is much higher than that of free distance d_{free} at low and medium SNR's. If a code works at medium and low SNR's, it is desirable to have B_d minimized rather than d_{free} maximized.

4.5 Turbo Code Design

In previous sections we have shown that the distance spectrum and the effective free distance affect the turbo code error performance considerably. In particular, the average bit error probability

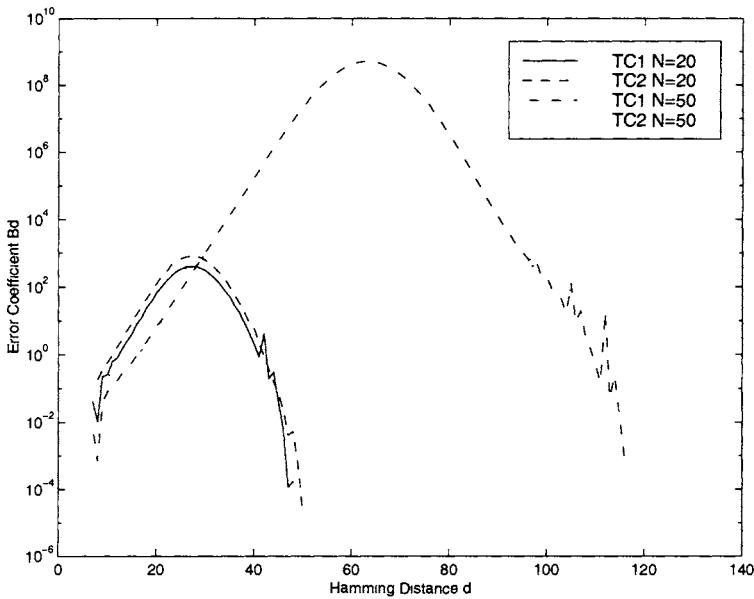


Fig. 4.14: Distance spectra for turbo codes TC1 and TC2 with interleaver sizes of 20 and 50

of turbo codes at low SNR's is dominated by error coefficients of medium weight distance spectral lines. On the other hand, turbo code performance at high SNR's is determined by the distances of the first several spectral lines. When turbo code interleaver size N is much larger than the code memory ν , the code effective free distance mainly dominates the code performance at high SNR's.

In this section we consider design of turbo codes with good error performance. We use the code distance spectrum and the code effective free distance as the design criteria to find the optimum component codes for a given memory. Obviously the code design criteria depend on the region of SNR's where the code is used. A turbo code performing well in the region of high SNR's may not perform well in the region of low SNR's, or vice versa. Thus, good turbo codes should be designed according to the range of SNR's where the code is used. The proper threshold values for SNR's depend on the code rate, the memory order and the interleaver size. The bit error probability corresponding to high SNR's is below

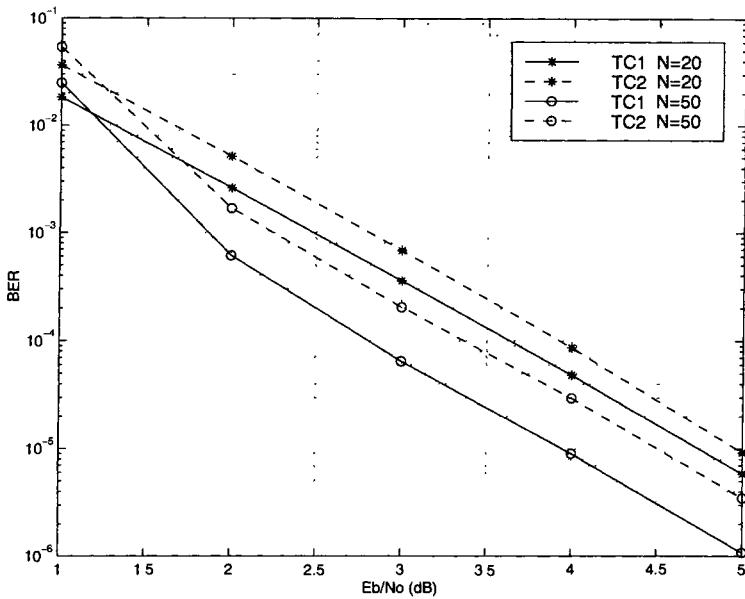


Fig. 4.15: Bit error probability upper bounds for turbo codes TC1 and TC2 with interleaver sizes of 20 and 50

10^{-5} , and the bit error probability corresponding to low SNR's is in the range of $10^{-1} \sim 10^{-5}$.

4.5.1 Turbo Code Design at High SNR's

Maximizing the code effective free distance can be used as the design criterion for constructing good turbo codes at high SNR's when the interleaver size N becomes large [14]. The design objective is equivalent to maximizing z_{\min} of the component recursive convolutional codes.

It is shown by Benedetto and Montorsi [14] and Divsalar and Pollara [12] that the design objective can be achieved if a primitive feedback polynomial is used in the RSC component encoders.

For a rate $1/n$ RSC code with memory ν and generator matrix

$$\mathbf{G}(D) = \left[1, \frac{\mathbf{g}_1(D)}{\mathbf{g}_0(D)}, \frac{\mathbf{g}_2(D)}{\mathbf{g}_0(D)}, \dots, \frac{\mathbf{g}_{n-1}(D)}{\mathbf{g}_0(D)} \right] \quad (4.45)$$

the following upper bound for z_{\min} holds

$$z_{\min} \leq (n - 1) (2^{\nu-1} + 2) \quad (4.46)$$

When $\mathbf{g}_0(D)$ is a primitive polynomial of degree ν and $\mathbf{g}_i(D)$, $1 \leq i \leq n - 1$, is any monic polynomial of degree ν , $\mathbf{g}_i(D) \neq \mathbf{g}_0(D)$, z_{\min} can achieve its upper bound [14] [12].

To illustrate why we choose $\mathbf{g}_0(D)$ as a primitive polynomial, for simplicity, we limit our discussion to the case of a component RSC code of rate 1/2 with generator matrix

$$\mathbf{G}(D) = \begin{bmatrix} 1, & \frac{\mathbf{g}_1(D)}{\mathbf{g}_0(D)} \end{bmatrix} \quad (4.47)$$

The extension to general codes is straightforward.

Let $1 + D^I$ be the shortest input sequence of weight 2 that generates a finite-length code sequence. The code parity check sequence is given by

$$(1 + D^I) \cdot \frac{\mathbf{g}_1(D)}{\mathbf{g}_0(D)} \quad (4.48)$$

Since $\mathbf{g}_1(D)$ and $\mathbf{g}_0(D)$ are relatively prime, the input sequence $1 + D^I$ must be a multiple of $\mathbf{g}_0(D)$ and periodic with a period of I . Increasing the period I will increase the length of the shortest code sequence with input weight 2. Intuitively, this will result in increasing weight of the code sequence. For polynomial $\mathbf{g}_0(D)$ with degree ν , any polynomial divisible by $\mathbf{g}_0(D)$ is periodic with period $I \leq 2^\nu - 1$. The maximum period is $2^\nu - 1$, which is obtained when $\mathbf{g}_0(D)$ is a primitive polynomial. The corresponding encoder is generated by a maximal length linear feedback shift register with degree ν [24]. In this case, the parity check sequence weight depends only on the primitive feedback polynomial and is independent of the polynomial $\mathbf{g}_1(D)$.

According to this design criterion, Benedetto and Montorsi found the best RSC component codes with rate 1/2 and memory order from one to five by computer search. The code search procedure can be summarized as follows [14]

1. Choose $\mathbf{g}_0(D)$ to be a primitive polynomial of degree ν .

2. Choose $\mathbf{g}_1(D)$ to be a polynomial of degree ν , where $\mathbf{g}_0(D)$ and $\mathbf{g}_1(D)$ are relatively prime.
3. Evaluate the average bit error probability bound of the candidate turbo code for a given interleaver size.
4. From all of the candidate codes, choose the one with the lowest bit error probability in the desired range of SNR's.

In the design procedure, Steps 1 and 2 make sure the candidate code has a maximum $z_{\min} = 2^{\nu-1} + 2$, and thus the maximum $d_{\text{free,eff}} = 2^{\nu} + 6$. Then the best code is chosen from all the candidate codes. Since the code performance at high SNR's is determined by the code effective free distance, the bit error probability can be expressed by the dominant term

$$P_b(e) \approx B_{\text{free,eff}} Q \left(\sqrt{2d_{\text{free,eff}} \cdot R \frac{E_b}{N_0}} \right) \quad (4.49)$$

where $B_{\text{free,eff}}$ is the error coefficient related to the code effective free distance. For a fixed effective free distance, optimizing the bit error probability implies minimization of the error coefficient. Thus, Steps 3 and 4 in the procedure can be replaced by

3. Evaluate the error coefficient $B_{\text{free,eff}}$ of the candidate code.
4. From all the candidate codes, choose the one with the minimum $B_{\text{free,eff}}$.

The above procedure was applied to find good rate 1/3 turbo codes using rate 1/2 RSC component codes [14]. A uniform interleaver with a size of 100 is employed in performance evaluation. The results are reported in Table 4.1, where generator polynomials $\mathbf{g}_0(D)$ and $\mathbf{g}_1(D)$, the effective free distance $d_{\text{free,eff}}$, the free distance d_{free} and its corresponding input weight w_{free} are shown. In the table, the generator polynomials are given in octal form. For example, the generator polynomials of the RSC code with memory $\nu = 4$, $\mathbf{g}_0(D) = 1 + D + D^4$ and $\mathbf{g}_1(D) = 1 + D + D^3 + D^4$, are represented by $\mathbf{g}_0(D) = (31)$ and $\mathbf{g}_1(D) = (33)$, respectively.

Table 4.1: Best rate 1/3 turbo codes at high SNR's [14]

v	$\mathbf{g}_0(D)$	$\mathbf{g}_1(D)$	$d_{\text{free,eff}}$	d_{free}	w_{free}
1	3	2	4	4	2
2	7	5	10	7	3
3	15	17	14	8	4
4	31	33	22	9	5
	31	27	22	9	5
5	51	77	38	10	6
	51	67	38	12	4

Turbo codes using the best rate 1/2 RSC component codes can achieve near optimum bit error probability performance at high SNR's. These codes have applications in many real systems operating at relatively low BER's, such as data transmission in satellite and space communications.

4.5.2 Turbo Code Design at Low SNR's

At low SNR's, the error probability of a turbo code is determined by its distance spectrum. So the distance spectrum can be used as a criterion to design good turbo codes in this region. In this section, we will present design criteria and construct *optimal distance spectrum* (ODS) turbo codes. These codes have minimum bit error probability at low SNR's [25].

A turbo code is said to be an optimal distance spectrum turbo code when its significant distance spectral lines are below those of any other code with the same rate, memory order and interleaver size. Assuming that the uniform interleaver is used, code design can be formulated as: (1) determine the distance spectrum of the turbo code, and (2) find good component codes by computer search which produce an optimal distance spectrum. The search procedure can be summarized as follows

1. Choose $\mathbf{g}_0(D)$ and $\mathbf{g}_1(D)$ to be relatively prime polynomials of degree ν .

2. Evaluate the distance spectrum of the candidate turbo code for a given interleaver size.
3. From all of the candidate codes, choose the one with the smallest error coefficients for low to medium Hamming distances.

Using the above method good component RSC codes with rate $1/2$ and memory order ν from two to five were found [25]. A uniform interleaver with size 40 is used to evaluate the distance spectrum. In Table 4.2, the code parameters of the ODS turbo codes are shown.

Table 4.2: Rate 1/3 ODS turbo codes at Low SNR's

ν	$\mathbf{g}_0(D)$	$\mathbf{g}_1(D)$	$d_{\text{free,eff}}$	d_{free}
2	7	5	10	7
3	15(13)	17	14	8
4	37	21	10	9
5	43	55	30	11

Comparing the ODS turbo code with the Benedetto-Montorsi (BM) codes shown in Table 4.1, we can observe that for memory orders 2 and 3, the BM codes are the same as the ODS turbo codes. However, the BM codes do not have the optimal distance spectrum for memory orders larger than 3. For memory orders 4 and 5, a turbo code with a nonmaximum effective free distance can have an optimal distance spectrum and better performance at low SNR's.

The distance spectra and bit error probability upper bounds for the ODS turbo codes are given in Figs. 4.16 and 4.17, respectively.

The ODS turbo codes can achieve optimum bit error probability performance at low SNR's. The codes can have applications in many real systems operating at relatively high BER's such as voice transmission in mobile communications, personal mobile communications, satellite mobile communications, military communications, etc.

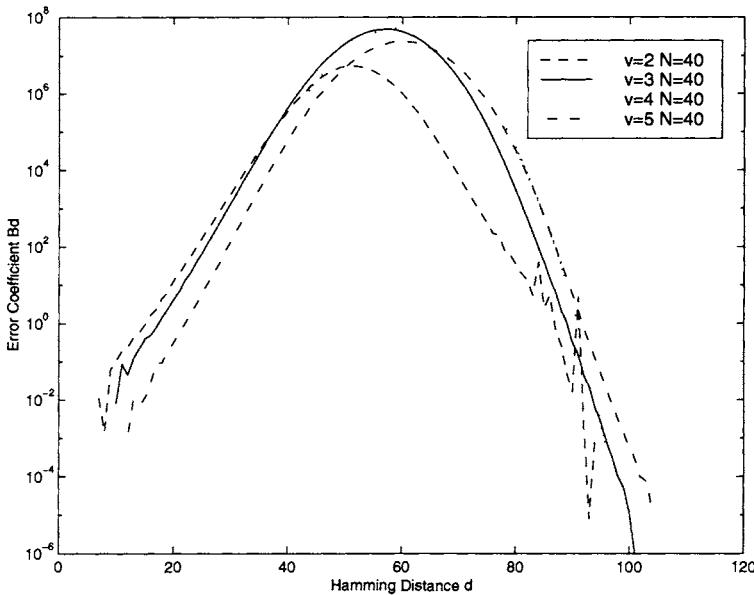


Fig. 4.16: Distance spectra for ODS turbo codes with interleaver size 40

4.5.3 Simulation Results

In order to compare the error performance of the ODS and BM turbo codes with rate 1/3, simulation results for both codes with memory order 4 are shown in Fig. 4.18. A pseudo-random interleaver with size 4096 is employed for both codes.

The turbo codes are decoded with an iterative maximum a posteriori probability algorithm. The number of iterations is 18. It is obvious that the ODS turbo code performs better than the BM turbo code at low SNR's corresponding to BER's from 10^{-1} to 10^{-5} . However, the BM code outperforms the ODS turbo code at high SNR's corresponding to BER's below 10^{-5} . This figure shows that code design based on performance analysis can also be verified by simulation.

The code design discussed in this chapter is based on uniform interleaving. We did not take into consideration the influence of interleaving structure on code performance. Since the interleaving structure affects turbo code error performance significantly at high

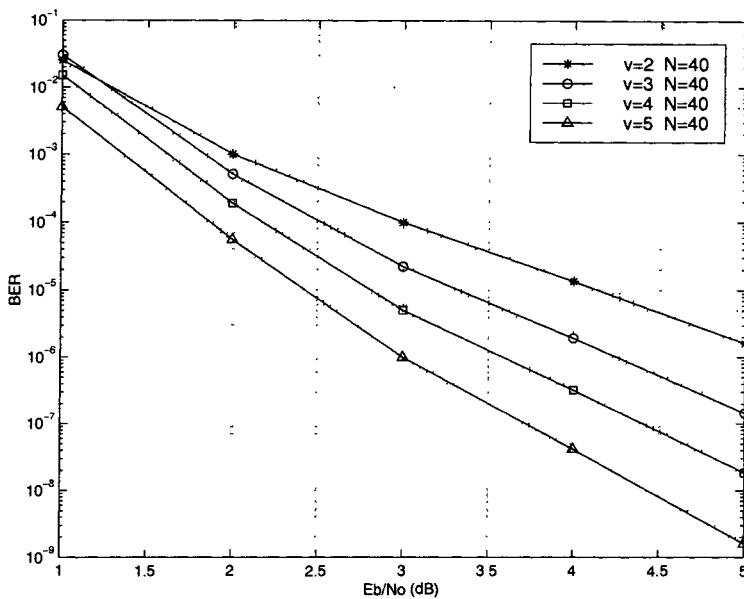


Fig. 4.17: Bit error probability upper bounds for ODS turbo codes with interleaver size 40

SNR's, interleaver design can be combined with code design to further improve turbo code error performance at high SNR's [25]. This is achieved by devising an interleaver which is matched to the optimum code found in the previous phase. The interleaver eliminates the first several spectral lines in the original distance spectrum. The bit error probability at high SNR's is then reduced and the asymptotic floor in the bit error probability curve is lowered. Interleaver design will be discussed in Chapter 7.

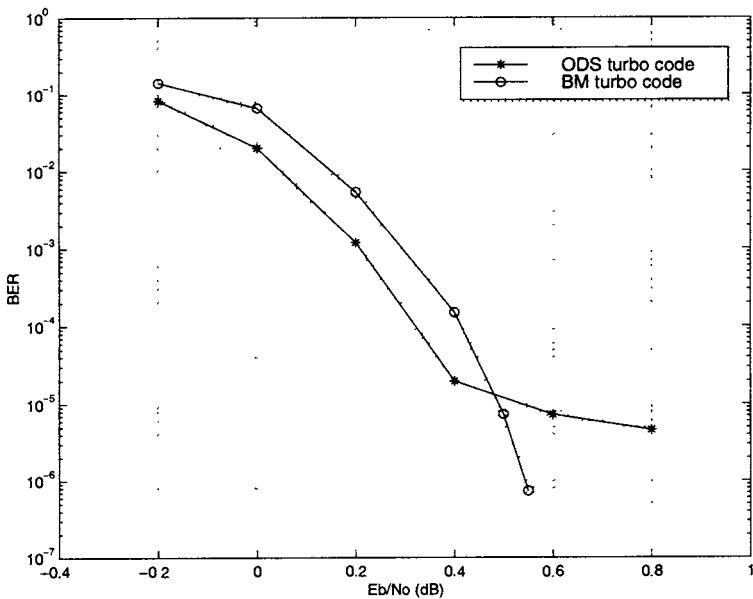


Fig. 4.18: Performance of ODS and BM turbo codes with rate 1/3 and memory order 4 on AWGN channels

4.6 Serial Concatenated Convolutional Codes

4.6.1 A Serial Concatenated Encoder

Let us consider a serial concatenated coding system consisting of an outer code C_o with rate $R_o = k/p$ and an inner code C_i with rate $R_i = p/n$. The two component codes are connected by an interleaver of size N . For simplicity we will assume that N is divisible by p . The overall concatenated code rate is $R = k/n$. A block diagram of a serial concatenated encoder is shown in Fig. 4.19.

For example, a rate 1/3 serial concatenated convolutional code is formed by an outer (2,1) convolutional code and an inner (3,2) convolutional code, joined by an interleaver. In this section we assume that the component codes are linear so that the serial concatenated code is linear as well. In addition, the interleaver is assumed to



Fig. 4.19: A serial concatenated encoder

be uniform which enables the weight distribution of a serial concatenated code to be computed from the weight distribution of the component codes.

4.6.2 Performance Analysis and Code Design

As a serial concatenated convolutional code is connected by an interleaver of size N , for N much larger than the memory of the convolutional codes, the overall concatenated code can be considered to be equivalent to an $(N/R_i, NR_o)$ block code. The codewords are sequences of the serial concatenated code that start and end at the zero states of both convolutional codes.

The upper bound of the bit error probability $P_b(e)$ for the serial concatenated convolutional code is given by [26]

$$P_b(e) \leq \sum_{d=d_{\min}}^{N/R_i} \sum_{w=1}^{NR_o} \frac{w}{NR_o} A_{w,d} Q\left(\sqrt{2dR \cdot \frac{E_b}{N_0}}\right) \quad (4.50)$$

where R is the overall turbo code rate, E_b is the received bit energy, N_0 is the one sided Gaussian noise spectral density, d is the Hamming distance between the codewords, N is the interleaver size, and d_{\min} is the concatenated code minimum distance. $A_{w,d}$ is the average number of concatenated codewords of the equivalent block code whose Hamming weight is d produced by a weight w input sequence. Using the uniform interleaver the coefficients $A_{w,d}$ can be expressed as

$$A_{w,d} = \sum_{l=0}^N \frac{A_{w,l}^{C_o} \cdot A_{l,d}^{C_i}}{\binom{N}{l}} \quad (4.51)$$

where $A_{w,l}^{C_o}$ and $A_{l,d}^{C_i}$ are the corresponding input-output weight distribution coefficients for equivalent block codes for the outer and

inner code, respectively. These coefficients for the equivalent block codes can be computed from the corresponding convolutional codes. By definition, the codewords of the equivalent block code are concatenation of error events of the convolutional codes.

Let

$$A(w, X, n) = \sum_d A_{w,d,n} X^d \quad (4.52)$$

be the weight enumerating function of sequences of the convolutional code that concatenate n error events with total input weight w , where $A_{w,d,n}$ is the number of sequences of weight d , input weight w and number of concatenated error events n in a codeword. For interleaver size N much larger than the memory of the convolutional codes, the input-output weight distribution coefficients of the equivalent block code $A_{w,d}^C$ can be approximated by

$$A_{w,d}^C \simeq \sum_{n=1}^{n_M} \binom{N/p}{n} A_{w,d,n} \quad (4.53)$$

where n_M is the largest number of error events concatenated in a codeword of weight d and generated by a weight w input sequence.

To compute these coefficients for the outer and inner equivalent block code, we will denote by n^o and n^i the number of concatenated error events in the outer and inner code, respectively. Also, we let A_{w,d,n^o} and A_{w,d,n^i} be the number of sequences of weight d , input weight w and the number of concatenated error events n^o or n^i in a codeword for the outer and inner code, respectively. By substituting the coefficients $A_{w,d}^{Co}$ and $A_{w,d}^{Ci}$ computed from (4.53) into (4.51), we can approximate the coefficient $A_{w,d}$ for the equivalent block code of the serially concatenated code as

$$A_{w,d} \simeq \sum_{l=d_f^o}^N \sum_{n^o=1}^{n_M^o} \sum_{n^i=1}^{n_M^i} \frac{\binom{N/p}{n^o} \binom{N/p}{n^i}}{\binom{N}{l}} A_{w,l,n^o}^o A_{l,d,n^i}^i \quad (4.54)$$

where d_f^o is the free distance of the outer code.

For large N the binomial coefficients can be approximated by

$$\binom{N}{n} \simeq \frac{N^n}{n!}$$

By substituting this approximation in (4.54) and in the expression for the bit error probability bound (4.50) we obtain [26]

$$\begin{aligned} P_b(e) \leq & \sum_{d=d_{\min}}^{N/R_i} \sum_{w=1}^{NR_o} Q\left(\sqrt{2dR \cdot \frac{E_b}{N_0}}\right) \sum_{l=d_f^o}^N \sum_{n^o=1}^{n_M^o} \sum_{n^i=1}^{n_M^i} N^{n^o+n^i-l-1} \\ & \cdot \frac{l! l!}{p^{n^o+n^i-1} n^o! n^i!} \frac{w}{k} A_{w,l,n^o}^o A_{l,d,n^i}^i \end{aligned} \quad (4.55)$$

The bound (4.55) can be represented in the form

$$P_b(e) \leq \sum_d C_d N^{\alpha(d)} Q\left(\sqrt{2dR \cdot \frac{E_b}{N_o}}\right) \quad (4.56)$$

where

$$\alpha(d) = \max_{w,l} \{n^o(d) + n^i(d) - l - 1\} \quad (4.57)$$

and C_d is a coefficient which does not depend on N . At high SNR's the bit error probability is dominated by the first term in the summation of (4.56). First we are going to compute $\alpha(d_{\min})$, where d_{\min} corresponds to the first term. For the first term it is possible to prove that [26]

$$\alpha(d_{\min}) \leq 1 - d_f^o \quad (4.58)$$

This result shows that the exponent of N corresponding to the minimum weight codewords of the serial concatenated code is always negative for $d_f^o \geq 2$. That is, at high SNR's there is always an interleaver gain if $d_f^o \geq 2$.

Next we are going to compute the largest exponent of N , defined as

$$\alpha_M = \max_d \alpha(d)$$

The term with the largest exponent of N in the summation of (4.56) dominates the contribution to the bit error probability for $N \rightarrow \infty$.

Let n_M^i and n_M^o be the largest number of concatenated error events in the codewords of the inner and outer code of weights d and l , respectively. For block and nonrecursive convolutional inner codes we have $n_M^i = l$. Thus

$$\alpha_M = n_M^o - 1 \geq 0 \quad (4.59)$$

This result indicates that for serial concatenated codes with block or nonrecursive convolutional inner codes there are always some terms of Hamming distance d , whose error coefficients in (4.55) increase with interleaver size of N . That means no interleaver gain is achieved for these terms.

For recursive convolutional inner codes the largest exponent of N is given by [26]

$$\alpha_M = - \left\lfloor \frac{d_f^o + 1}{2} \right\rfloor < 0 \quad (4.60)$$

As this result shows, if the inner code is a recursive convolutional code there is always an interleaver gain. The interleaver gain is $N^{-d_f^o/2}$ for even values of the outer code free distance and $N^{-(d_f^o+1)/2}$ for odd values of the outer code free distance.

The codeword weight associated with the highest exponent N , denoted by $d(\alpha_M)$, is given by

$$d(\alpha_M) = \begin{cases} \frac{d_f^o d_{f,eff}^i}{2} & \text{for } d_f^o \text{ even} \\ \frac{(d_f^o - 3)d_{f,eff}^i}{2} + d_{\min}^{(3)} & \text{for } d_f^o \text{ odd} \end{cases} \quad (4.61)$$

where $d_{f,eff}^i$ is the *effective free distance* of the inner code. This is the minimum weight of sequences of the inner code generated by a weight-2 input sequence. Also, $d_{\min}^{(3)}$ is the minimum weight of sequences of the inner code generated by a weight-3 input sequence.

The asymptotic expressions, for N very large, for the bit error probability are given by

$$P_b(e) \leq C_{even} N^{-(d_f^o/2)} Q \left(\sqrt{d_f^o d_{f,eff}^i R \frac{E_b}{N_0}} \right) \quad (4.62)$$

for even values of d_f^o , and

$$\begin{aligned} P_b(e) &\leq C_{odd} N^{-[(d_f^o+1)/2]} \\ &\cdot Q \left(\sqrt{[(d_f^o - 3)d_{f,eff}^i + 2d_{\min}^{(3)}] R \frac{E_b}{N_0}} \right) \end{aligned} \quad (4.63)$$

for odd values of d_f^o , where C_{even} and C_{odd} are the coefficients for d_f^o even and odd, respectively, which do not depend on N .

Equations (4.62) and (4.63) show that the bit error probability for large interleaver size N is dominated by the inner code effective free distance and the outer code free distance. On the basis of these results it is possible to formulate serial concatenated code design rules. They are summarized as follows.

Design Rules for Serial Concatenated Codes

1. The inner code should be chosen to be a recursive convolutional code. The outer code could be either nonrecursive or recursive.
2. The effective free distance of the inner code $d_{f,eff}^i$ should be maximized.
3. The free distance of the outer code d_f^o should be maximized.
4. The number of input sequences generating the free distance error events of the outer code and their input weights should be minimized.

As nonrecursive codes have error events with $w=1$ and, in general, less input errors associated with error events at free distance, it is appropriate to choose nonrecursive convolutional codes as outer codes.

Bibliography

- [1] G. D. Forney, Jr., *Concatenated Codes*. Cambridge, MA: MIT Press, 1966.
- [2] R. H. Deng and D. J. Costello, “High rate concatenated coding systems using bandwidth efficient trellis inner codes,” *IEEE Trans. Commun.*, vol. 37, no. 5, May 1989, pp. 420-427.
- [3] J. Hagenauer and P. Hoeher, “Concatenated Viterbi decoding,” in *Proc. 4th Joint Swedish-Soviet Int. Workshop on Inform. Theory*, Gotland, Sweden, Studenlitteratur, Lund, Aug. 1989, pp. 29-33.
- [4] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: Turbo-codes(1),” in *Proc. ICC’93*, Geneva, Switzerland, May 1993, pp. 1064-1070.
- [5] C. Berrou and A. Glavieux, “Near optimum error correcting coding and decoding: Turbo-codes,” *IEEE Trans. Commun.*, vol. 44, no. 10, Oct. 1996, pp. 1261-1271.
- [6] D. Divsalar and F. Pollara, “Turbo codes for PCS applications,” in *Proc. ICC’95*, Seattle, WA, June 1995, pp. 54-59.
- [7] P. Robertson, “Illuminating the structure of parallel concatenated recursive systematic (TURBO) codes,” in *Proc. GLOBECOM’94*, San Francisco, CA, Nov. 1994, pp. 1298-1303.

- [8] A. S. Barbulescu and S. S. Pietrobon, "Interleaver design for turbo codes," *Electron. Lett.*, vol. 30, no. 25, Dec. 1994, p. 2107.
- [9] A. S. Barbulescu and S. S. Pietrobon, "Terminating the trellis of turbo-codes in the same state," *Electron. Lett.*, vol. 31, no. 1, Jan. 1995, pp. 22-23.
- [10] A. S. Barbulescu and S. S. Pietrobon, "Interleaver design for three dimensional turbo codes," in *Proc. 1995 IEEE ISIT*, Whistler, BC, Canada, Sep. 1995, p. 37.
- [11] O. Joerssen and H. Meyr, "Terminating the trellis of turbo-codes," *Electron. Lett.*, vol. 30, no. 16, Aug. 1994, pp. 1285-1286.
- [12] D. Divsalar and F. Pollara, "On the design of turbo codes," TDA Progress Report 42-123, Jet Propulsion Lab., Nov. 1995, pp. 99-121.
- [13] S. Benedetto and G. Montorsi, "Unveiling turbo-codes: Some results on parallel concatenated coding schemes," *IEEE Trans. Inform. Theory*, vol. 42, no. 2, Mar. 1996, pp. 409-428.
- [14] S. Benedetto and G. Montorsi, "Design of parallel concatenated convolutional codes," *IEEE Trans. Commun.*, vol. 44, no. 5, May 1996, pp. 591-600.
- [15] S. Benedetto, R. Garello, and Guido Montorsi, "A search for good convolutional codes to be used in the construction of turbo codes," *IEEE Trans. Commun.*, vol. 46, no. 9, Sep. 1998, pp. 1101-1105.
- [16] J. G. Proakis, *Digital Communications*, 2nd Ed., McGraw-Hill, New York, 1989.
- [17] D. Divsalar, S. Dolinar, R. J. McEliece, and F. Pollara, "Transfer function bounds on the performance of turbo codes," TDA Progress Report 42-123, Jet Propulsion Lab., Aug. 1995, pp. 44-55.

- [18] T. M. Duman and M. Salehi, "New performance bounds for turbo codes," *IEEE Trans. Commun.*, vol. 46, no. 6, June 1998, pp. 717-723.
- [19] A. M. Viterbi and A. J. Viterbi, "Improved union bound on linear codes for the input-binary AWGN channel with applications to turbo-codes," in *Proc. IEEE 1998 ISIT*, MIT, MA, Aug. 1998, p. 29.
- [20] I. Sason and S. Shamai(shitz), "Improved upper bounds on the performance of parallel and serial concatenated turbo codes," in *Proc. IEEE 1998 ISIT*, MIT, MA, Aug. 1998, p. 30.
- [21] A. J. Viterbi, "Convolutional codes and their performance in communications systems," *IEEE Trans. Commun.*, vol. 19, no. 5, Oct. 1971, pp. 751-772.
- [22] P. Thitimajshima, "Les codes convolutifs recursifs systématiques et leur application à la concaténation parallèle," (in French), Ph.D. no. 284, University de Bretagne Occidentale, Brest, France, Dec. 1993.
- [23] D. Divsalar and R. J. McEliece, "The effective free distance of turbo codes," *Electron. lett.*, vol. 32, no. 5, Feb. 1996, pp. 445-446.
- [24] L. C. Perez, J. Seghers, and D. J. Costello, Jr., "A distance spectrum interpretation of turbo codes," *IEEE Trans. Inform. Theory*, vol. 42, no. 6, Nov. 1996, pp. 1698-1709.
- [25] J. Yuan, B. Vucetic, and W. Feng, "Combined turbo codes and interleaver design," *IEEE Trans. Commun.*, vol. 47, no. 4, Apr. 1999, pp. 484-487.
- [26] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial concatenation of interleaved codes: Performance analysis, design, and iterative decoding," *IEEE Trans. Inform. Theory*, vol. 44, no. 3, May 1998, pp. 909-926.

Chapter 5

Trellis Based Decoding of Linear Codes

5.1 Introduction

In Chapters 2 and 3 we presented methods of representing linear block and convolutional codes by trellises. In this chapter we first examine various trellis based strategies for decoding of linear codes. Each of these can be used as a basic building block for decoding of concatenated and turbo codes.

Trellis based decoding algorithms are recursive methods for estimation of the state sequence of a discrete-time finite-state Markov process observed in memoryless noise. The Viterbi algorithm (VA) minimizes the sequence error probability. Its output is in the form of hard-quantized estimation of symbols in the most likely transmitted code sequence. In concatenated systems, with multiple signal processing stages, the overall receiver performance is improved if the inner stages produce soft output estimation. We show how the VA can be modified to generate soft-output information. The related algorithm is known as the soft output Viterbi algorithm (SOVA). The SOVA produces in addition to the maximum likelihood hard estimates of a code sequence, a reliability measure for each received symbol. However, these reliability estimates are suboptimum.

In another class of decoders, the decoding criterion is minimization of the symbol or bit error probability. The decoder generates

optimum reliability estimates in the form of a posteriori symbol probabilities. The algorithm is known as the maximum a posteriori probability (MAP) decoding. We further examine various versions of the MAP algorithm, such as Max-Log-MAP and Log-MAP, in search for a good performance-complexity trade-off.

5.2 System Model

In this section we introduce the underlying system model for the decoding methods to be considered. The system block diagram is shown in Fig. 5.1.

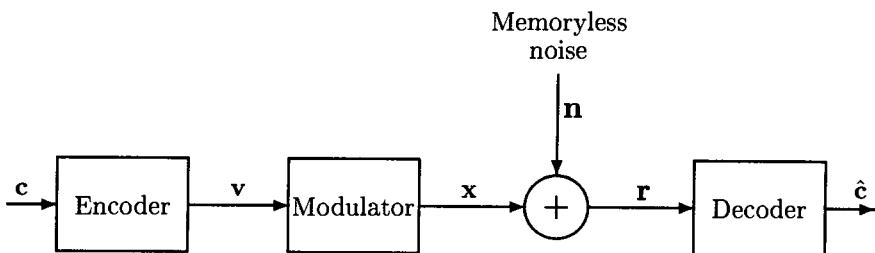


Fig. 5.1: System model

In order to simplify the analysis, the following description of the VA is specific to $(n, 1, \nu)$ binary convolutional codes, though it could easily be generalized to include k/n rate convolutional codes, as well as decoding of block codes. A binary message sequence, denoted by c and given by

$$c = (c_1, c_2, \dots, c_t, \dots, c_N), \quad (5.1)$$

where c_t is the message symbol at time t and N is the sequence length, is encoded by a linear code. In general the message symbols c_t can be nonbinary but for simplicity we assume that they are independently generated binary symbols and have equal a priori probabilities. The encoding operation is modelled as a discrete time finite-state Markov process. This process can be graphically

represented by state and trellis diagrams. In respect to the input c_t , the finite-state Markov process generates an output \mathbf{v}_t and changes its state from S_t to S_{t+1} , where $t + 1$ is the next time instant. The process can be completely specified by the following two relationships

$$\begin{aligned}\mathbf{v}_t &= f(S_t, c_t, t) \\ S_{t+1} &= g(S_t, c_t, t)\end{aligned}\quad (5.2)$$

The functions $f(\cdot)$ and $g(\cdot)$ are generally time varying.

The state sequence from time 0 to t is denoted by \mathbf{S}_0^t and is written as

$$\mathbf{S}_0^t = (S_0, S_1, \dots, S_t) \quad (5.3)$$

The state sequence is a Markov process, so that the probability $P(S_{t+1} | S_0, S_1, \dots, S_t)$ of being in state S_{t+1} , at time $(t + 1)$, given all states up to time t , depends only on the state S_t , at time t ,

$$P(S_{t+1} | S_0, S_1, \dots, S_t) = P(S_{t+1} | S_t) \quad (5.4)$$

The encoder output sequence from time 1 to t is represented as

$$\mathbf{v}_1^t = (\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_t) \quad (5.5)$$

where

$$\mathbf{v}_t = (v_{t,0}, v_{t,1}, \dots, v_{t,n-1}) \quad (5.6)$$

is the code block of length n .

The code sequence \mathbf{v}_1^t is modulated by a BPSK modulator. The modulated sequence is denoted by \mathbf{x}_1^t and is given by

$$\mathbf{x}_1^t = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_t) \quad (5.7)$$

where

$$\mathbf{x}_t = (x_{t,0}, x_{t,1}, \dots, x_{t,n-1}) \quad (5.8)$$

and

$$x_{t,i} = 2v_{t,i} - 1, \quad i = 0, 1, \dots, n - 1 \quad (5.9)$$

As there is one-to-one correspondence between the code and modulated sequence, the encoder/modulator pair can be represented by a discrete-time finite-state Markov process and can be graphically described by state or trellis diagrams.

The modulated sequence \mathbf{x}_1^t is corrupted by additive white Gaussian noise, resulting in the received sequence

$$\mathbf{r}_1^t = (\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_t) \quad (5.10)$$

where

$$\mathbf{r}_t = (r_{t,0}, r_{t,1}, \dots, r_{t,n-1}) \quad (5.11)$$

and

$$r_{t,i} = x_{t,i} + n_{t,i}, \quad i = 0, 1, \dots, n-1 \quad (5.12)$$

where $n_{t,i}$ is a zero-mean Gaussian noise random variable with variance σ^2 . Each noise sample is assumed to be independent from each other.

The decoder gives an estimate of the input to the discrete finite-state Markov source, by examining the received sequence \mathbf{r}_1^t . The decoding problem can be alternatively formulated as finding the modulated sequence \mathbf{x}_1^t or the coded sequence \mathbf{v}_1^t . As there is one-to-one correspondence between the sequences \mathbf{v}_1^t and \mathbf{x}_1^t , if one of them has been estimated, the other can be obtained by simple mapping.

The discrete-time finite-state Markov source model is applicable to a number of systems in communications, such as linear convolutional and block coding, continuous phase modulation and channels with intersymbol interference.

5.3 Optimization Criteria

In the system model shown in Fig. 5.1 we assumed BPSK modulation. In general, we can consider M -ary modulation where the code sequence \mathbf{v} is divided into groups of $\log_2 M$ binary symbols and each of them is mapped into an M -ary symbol in the modulator.

Assume that the modulated sequence \mathbf{x} contains N' M -ary symbols

$$\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{N'}) \quad (5.13)$$

The corresponding code sequence \mathbf{v} is $N' \cdot \log_2 M$ binary symbols long

$$\mathbf{v} = (v_{1,1}, \dots, v_{1,\log_2 M}, v_{2,1}, \dots, v_{2,\log_2 M}, \dots, v_{N',1}, \dots, v_{N',\log_2 M})$$

The message sequence is given by

$$\mathbf{c} = (c_1, c_2, \dots, c_N) \quad (5.14)$$

where $N = RN' \log_2 M$ and R is the code rate.

The receiver estimates the unknown message sequence \mathbf{c} , which can be written as

$$\hat{\mathbf{c}} = (\hat{c}_1, \hat{c}_2, \dots, \hat{c}_N) \quad (5.15)$$

Due to one-to-one correspondence between \mathbf{c} , \mathbf{v} and \mathbf{x} , the estimates of the code and modulated sequence can be obtained by simple mapping operations from $\hat{\mathbf{c}}$. They can be written as

$$\begin{aligned}\hat{\mathbf{v}} &= (\hat{v}_{1,1}, \dots, \hat{v}_{1,\log_2 M}, \hat{v}_{2,1}, \dots, \hat{v}_{2,\log_2 M}, \dots, \hat{v}_{N',1}, \dots, \hat{v}_{N',\log_2 M}) \\ \hat{\mathbf{x}} &= (\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \dots, \hat{\mathbf{x}}_{N'})\end{aligned}$$

An optimum receiver is designed to minimize one of the following error probabilities:

- Word error rate (WER) which is defined as the probability that $\hat{\mathbf{v}} \neq \mathbf{v}$, where \mathbf{v} and $\hat{\mathbf{v}}$ are the transmitted and estimated code sequences, respectively.
- Symbol error rate (SER) which is defined as the probability of $\hat{\mathbf{x}}_t \neq \mathbf{x}_t$, $t = 1, 2, \dots, N'$, where x_t and \hat{x}_t are the transmitted and estimated modulated symbols at time t , respectively.
- Bit error rate (BER) which is defined as the probability of $\hat{c}_t \neq c_t$, where c_t and \hat{c}_t are the transmitted and corresponding estimated bits at time t , respectively.

If a word error occurs, there must be at least one symbol error. If a symbol is in error, there is at least one bit in error. The above probabilities are defined for hard output estimates only.

It is possible to design three classes of optimum receivers, which correspond to minimization of WER, SER and BER. While these receivers have different performance at low signal-to-noise ratios (SNR), their performances are almost identical at high SNR's. That is, minimization of WER approximately produces minimum SER and BER.

In a binary transmission system, the SER is identical to BER. Even in non-binary systems, receivers minimizing SER have identical structure as those minimizing BER.

5.4 The Viterbi Algorithm

The Viterbi algorithm (VA) was originally proposed for decoding of convolutional codes [1]. Essentially, the algorithm performs estimation of the input sequence of a discrete time finite-state Markov process observed in memoryless noise. Hence it is applicable as a solution to various communication estimation problems, as long as the system can be modelled as a finite state machine and represented by a time invariant or time varying trellis diagram.

We consider the system model shown in Fig. 5.1. The pair encoder/decoder represents a discrete time finite-state Markov source described by a trellis diagram with M_s states.

Initially we assume that the process runs from state 0 at time instant 0 to state 0 at time instant τ , where τ is the terminal time instant.

The assumption of known start and final states is trivial and is not an essential requirement for the VA to work. The state sequence can be represented as

$$\mathbf{S} = (0, S_1, S_2, \dots, S_{\tau-1}, 0) \quad (5.16)$$

The VA finds an information sequence $\hat{\mathbf{c}} = \hat{\mathbf{c}}_1^\tau$ that corresponds to the modulated sequence $\hat{\mathbf{x}} = \hat{\mathbf{x}}_1^\tau$ in the trellis diagram such that the word error probability, denoted by P_w , is minimized. The word error probability can be expressed as

$$P_w = 1 - \int_{\mathbf{r}} P_r(\mathbf{c} | \mathbf{r}) P_r(\mathbf{r}) d\mathbf{r} \quad (5.17)$$

where \mathbf{r} is the received sequence $\mathbf{r} = \mathbf{r}_1^\tau$.

Minimizing the word error probability is equivalent to maximizing the second term in Eq. (5.17), which is the probability that a word is correct.

As $P_r(\mathbf{r})$ is positive and independent of \mathbf{c} , this is further equivalent to maximizing the a posteriori probability $P_r(\mathbf{c} | \mathbf{r})$. The receiver which maximizes this probability is known as a *maximum a posteriori probability* (MAP) receiver.

Using Bayes' rule we get

$$P_r(\mathbf{c} | \mathbf{r}) = \frac{P_r(\mathbf{c}) \cdot P_r(\mathbf{r} | \mathbf{c})}{P_r(\mathbf{r})} \quad (5.18)$$

Assuming that the signals are equally likely, it suffices for the receiver to maximize the likelihood function $P_r(\mathbf{r} | \mathbf{c})$. A decoder that selects its estimate by maximizing $P_r(\mathbf{r} | \mathbf{c})$ is called *maximum likelihood* (ML) decoder. The above expression shows that if the sequences are equally likely, MAP and ML decoders are equivalent in terms of word error probability.

The probability $P_r(\mathbf{r} | \mathbf{c})$ for the received sequence of length τ , can be expressed as

$$\begin{aligned} P_r(\mathbf{r} | \mathbf{c}) &= P_r(\mathbf{r}_1^\tau | \mathbf{c}_1^\tau) \\ &= P_r(\mathbf{r}_1^\tau | \mathbf{x}_1^\tau) \\ &= \prod_{t=1}^{\tau} P_r(\mathbf{r}_t | \mathbf{x}_t) \\ &= \prod_{t=1}^{\tau} \prod_{i=0}^{n-1} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(r_{t,i} - x_{t,i})^2}{2\sigma^2}} \end{aligned} \quad (5.19)$$

In order to simplify the operations we introduce the log function $\log P_r(\mathbf{r} | \mathbf{c})$, given by

$$\log P_r(\mathbf{r} | \mathbf{c}) = \sum_{t=1}^{\tau} \log P_r(\mathbf{r}_t | \mathbf{x}_t) \quad (5.20)$$

For the Gaussian channel $\log P_r(\mathbf{r} | \mathbf{c})$ becomes

$$\begin{aligned} \log P_r(\mathbf{r} | \mathbf{c}) &= \sum_{t=1}^{\tau} \log \prod_{i=0}^{n-1} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(r_{t,i} - x_{t,i})^2}{2\sigma^2}} \\ &= -\frac{n\tau}{2} \log(2\pi) - n\tau \log \sigma \\ &\quad - \sum_{t=1}^{\tau} \sum_{i=0}^{n-1} \frac{(r_{t,i} - x_{t,i})^2}{2\sigma^2} \end{aligned} \quad (5.21)$$

Expression (5.21) shows that maximizing $P(\mathbf{r} | \mathbf{c})$ is equivalent to minimizing the Euclidean difference

$$\sum_{t=1}^{\tau} \sum_{i=0}^{n-1} (r_{t,i} - x_{t,i})^2$$

between the received sequence \mathbf{r}_1^τ and the modulated sequence \mathbf{x}_1^τ in the trellis diagram.

We assign to each branch at time t on the path \mathbf{x} in the trellis, the Euclidean distance, called *branch metric*, denoted by $\nu_t^{\mathbf{x}}$, as follows

$$\nu_t^{\mathbf{x}} = \sum_{i=0}^{n-1} (r_{t,i} - x_{t,i})^2 \quad (5.22)$$

Then the *path metric* corresponding to the path \mathbf{x} , denoted by $\mu_t^{(\mathbf{x})}$ is given by

$$\begin{aligned} \mu_t^{(\mathbf{x})} &= \sum_{t'=1}^t \nu_{t'}^{(\mathbf{x})} \\ &= \mu_{t-1}^{(\mathbf{x})} + \nu_t^{(\mathbf{x})} \end{aligned} \quad (5.23)$$

Note that minimization of the Euclidean distance is equivalent to maximization of the expression $\sum_{t=1}^{\tau} \sum_{i=0}^{n-1} r_{t,i} x_{t,i}$, which is another definition of the path metric.

The VA is an efficient way of finding a path in the trellis with the minimum path metric. It is based on the idea that among the paths merging into a state in the code trellis only the most probable paths need to be saved for future processing, while all other paths can be discarded with no consequence to decoding optimality.

The computation is based on keeping only one path per node with the minimum path metric at each time instant. This path is called the *survivor*.

The decision on the message estimate \mathbf{c} is made at the final time instant τ . The maximum likelihood path is chosen as the survivor in the final node. If the minimum path metric corresponds to a path $\hat{\mathbf{x}}$, the decoder will select the binary sequence $\hat{\mathbf{c}}$ on this path as the hard estimate of the transmitted sequence \mathbf{c} .

The VA can be summarized as follows.

Summary of the VA

1. Set initial values

$$t = 0; \quad S_0 = 0; \quad \mu_0^{(x)}(S_0 = 0) = 0; \quad \mu_0^{(x)}(S_0 \neq 0) = \infty$$

2. Increase time t by 1

- Compute the branch metrics for all branches entering a node at time t
- Compute the path metrics for all paths entering a node at time t , by adding the branch metric of the branch entering the node to the path metric of the connecting survivor at a previous node at time $t - 1$, as in Eq. (5.23)
- Compare the path metrics for all the paths entering a node and find the survivor for each node. For each node store the survivor path and its metric.
- Repeat 2 until $t = \tau$

3. The survivor at node S_τ is the maximum likelihood path.

When the sequence length is long or infinite, it is necessary to truncate the survivors to some manageable length, called the *decision depth*, denoted by D .

The decoder makes the decision on the transmitted symbol c_t at time $t' = D + t$. In general, if D is large enough, there is a high probability that all survivors will stem from the same branch at time t . The symbol on the maximum likelihood path at time t can be delivered as the algorithm's firm decision. The decision on the maximum likelihood path at time $t' = t + D$ can be to choose either any survivor or the survivor with the minimum path metric. When D is large enough (about six times the encoder memory order), the effect of choosing any survivor on the performance is negligible.

If time t becomes large it is necessary to normalize the survivor path metrics from time to time by subtracting a constant from all of them. For quantized branch metrics, the normalization of the survivor path metrics can be avoided by using modulo-sum algorithms.

The VA may be required to start with no knowledge of the initial state. In this case, it might be initialized by any reasonable value such as $\mu_0^x(l) = 0$, $l = 1, 2, \dots, M_s$, for all states l . Usually, after an initial transient there is a high probability that all survivors will merge with the correct path. That is, the VA synchronizes itself with no special procedure.

The VA requires $M_s \cdot D$ memory units, to store the survivors and their path metrics. For an (n, k) convolutional code, there are 2^k branches entering a node. Thus, the algorithm computes $M_s \cdot 2^k$ branch metrics and performs $M_s \cdot 2^k$ additions and $M_s \cdot 2^k$ comparisons at each time.

Therefore, the memory requirement is proportional to the number of states, and the computational complexity is proportional to the number of branches and the number of states.

Example 5.1 Consider a $1/2$ rate RSC code with the encoder shown in Fig. 5.2 (a). Its state and trellis diagrams are shown in Figs. 5.2 (b) and (c), respectively. If the received sequence is

$$\mathbf{r} = [(1, -1), (0.8, 1), (-1, 1), (-1, 1), (1, -1)]$$

the branch metrics are shown in Fig. 5.3. As there are four message symbols, the transmitted sequence is extended by adding one input symbol equal to the feedback symbol, to terminate the trellis to the zero state.

The survivors and their path metrics at times $t = 1$ to $t = 5$ are shown in Figs. 5.4, (a) to (e). The transmitted information sequence estimate is $\mathbf{c} = (0, 1, 0, 0, 1)$.

5.5 The Bidirectional Soft Output Viterbi Algorithm

The drawback of the VA is that it generates hard symbol estimates resulting in performance loss in multistage decoding. It can be modified to produce soft outputs [6], [14], [12], [2]. Here we will present a bidirectional SOVA algorithm which does not require knowledge

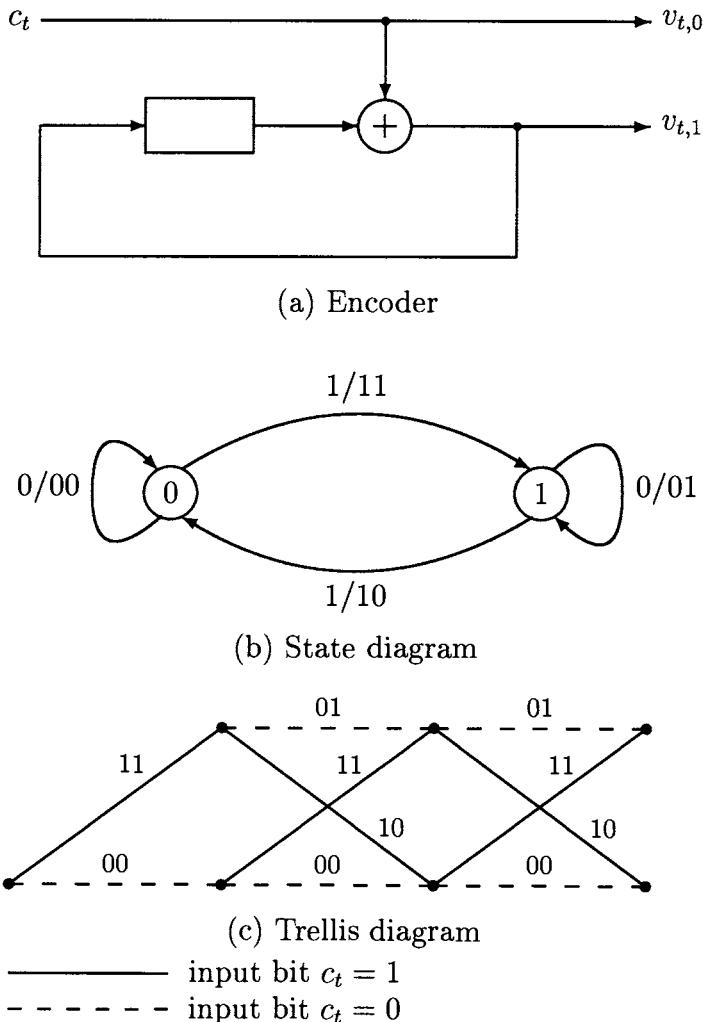


Fig. 5.2: A convolutional encoder and its graphical representation

of the noise variance and is simple to implement particularly for systems with block structured data [2] [12] [13].

The SOVA estimates the soft output information for each transmitted binary symbol in the form of the log-likelihood function

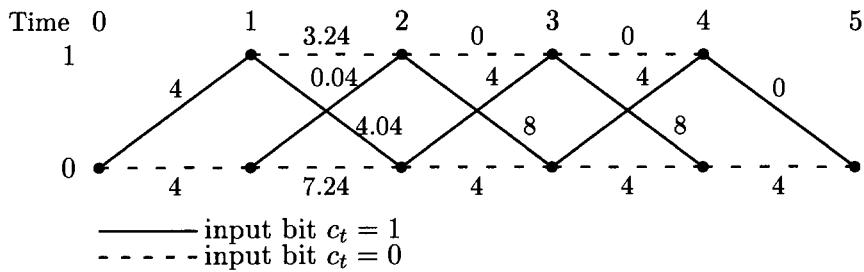


Fig. 5.3: The branch metrics in Example 5.1

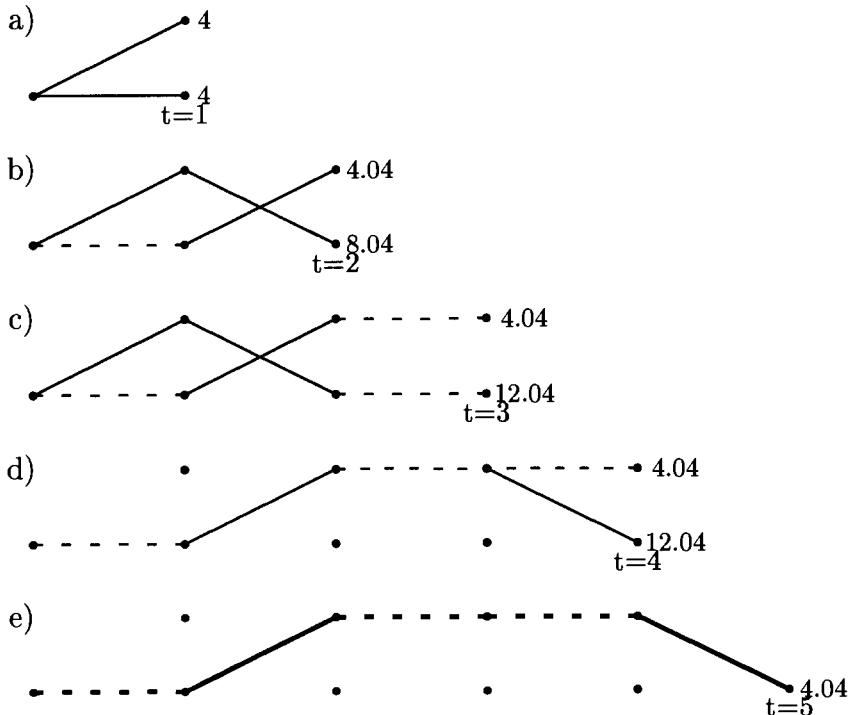


Fig. 5.4: The survivors and their path metrics in Example 5.1

$\Lambda(c_t)$, as follows

$$\Lambda(c_t) = \log \frac{P_r\{c_t = 1 \mid \mathbf{r}_1^*\}}{P_r\{c_t = 0 \mid \mathbf{r}_1^*\}} \quad (5.24)$$

where \mathbf{r}_1^τ is the received sequence and $P_r\{c_t = i | \mathbf{r}_1^\tau\}$, $i = 0, 1$, is the a posteriori probability (APP) of the transmitted symbol.

The SOVA decoder makes a hard decision by comparing $\Lambda(c_t)$ to a threshold equal to zero

$$c_t = \begin{cases} 1 & \text{if } \Lambda(c_t) \geq 0 \\ 0 & \text{otherwise} \end{cases} \quad (5.25)$$

The decoder selects the path $\hat{\mathbf{x}}$ with the minimum path metric $\mu_{\tau,\min}$ as the maximum likelihood (ML) path in the same way as the standard VA. The probability of selecting this path, from Eqs. (5.18), (5.19), (5.21) and (5.23), is proportional to

$$\begin{aligned} P_r(\mathbf{c} | \mathbf{r}_1^\tau) &= P_r(\mathbf{x} | \mathbf{r}_1^\tau) \\ &\sim e^{-\mu_{\tau,\min}} \end{aligned} \quad (5.26)$$

Let us denote by $\mu_{t,c}$ the minimum path metric of the paths with the complementary symbol to the ML symbol at time t . If the ML symbol at time t is 1, then its complementary symbol is 0. Therefore we can write

$$\begin{aligned} P_r(c_t = 1 | \mathbf{r}_1^\tau) &\sim e^{-\mu_{\tau,\min}} \\ P_r(c_t = 0 | \mathbf{r}_1^\tau) &\sim e^{-\mu_{t,c}} \end{aligned} \quad (5.27)$$

The logarithm of the ratio of the above two probabilities is given by

$$\begin{aligned} \log \frac{P_r\{c_t = 1 | \mathbf{r}_1^\tau\}}{P_r\{c_t = 0 | \mathbf{r}_1^\tau\}} &\sim \log \frac{e^{-\mu_{\tau,\min}}}{e^{-\mu_{t,c}}} \\ &= \mu_{t,c} - \mu_{\tau,\min} \end{aligned} \quad (5.28)$$

Let us denote by μ_t^1 the minimum path metric for all paths for which c_t is 1 and μ_t^0 the minimum path metric for all paths for which c_t is 0. If the ML estimate at time t is 1, its complementary symbol at time t is 0. Then $\mu_t^1 = \mu_{\tau,\min}$ and $\mu_t^0 = \mu_{t,c}$ and the log-likelihood in Eq. (5.28) becomes

$$\log \frac{P_r\{c_t = 1 | \mathbf{r}_1^\tau\}}{P_r\{c_t = 0 | \mathbf{r}_1^\tau\}} \sim \mu_t^0 - \mu_t^1 \quad (5.29)$$

On the other hand if the ML hard estimates at time t is 0, its complementary symbol is 1, giving $\mu_t^0 = \mu_{\tau,\min}$ and $\mu_t^1 = \mu_{t,c}$. The log-likelihood ratio for this case is given by

$$\begin{aligned} \log \frac{P_r\{c_t = 1 | \mathbf{r}_1^\tau\}}{P_r\{c_t = 0 | \mathbf{r}_1^\tau\}} &\sim \log \frac{e^{-\mu_{t,c}}}{e^{-\mu_{\tau,\min}}} \\ &= \mu_{\tau,\min} - \mu_{t,c} \\ &= \mu_t^0 - \mu_t^1 \end{aligned} \quad (5.30)$$

As Eqs. (5.29) and (5.30) indicate, regardless of the value of the ML hard estimate, the log-likelihood ratio can be expressed as

$$\begin{aligned} \Lambda(c_t) &= \log \frac{P_r\{c_t = 1 | \mathbf{r}_1^\tau\}}{P_r\{c_t = 0 | \mathbf{r}_1^\tau\}} \\ &\sim \mu_t^0 - \mu_t^1 \end{aligned} \quad (5.31)$$

That is, the soft output of the decoder can be obtained as the difference of the minimum path metric among all the paths with symbol 0 at time t and the minimum path metric among all the paths with symbol 1 at time t . The sign of $\Lambda(c_t)$ determines the hard estimate at time t and its absolute value represents the soft output information that can be used for decoding in the next stage.

If the decision is made on a finite length block, as in block codes, turbo codes or convolutional codes in TDMA systems, the SOVA can be implemented as a bidirectional recursive method with forward and backward recursions.

The SOVA can be summarized as follows.

Summary of the SOVA

I Forward recursion

1. Set initial values

$$t = 0; \quad S_0 = 0; \quad \mu_0^{(\mathbf{x})}(S_0 = 0) = 0; \quad \mu_0^{(\mathbf{x})}(S_0 \neq 0) = \infty; \quad S_\tau = 0$$

2. Increase time t by 1

- Compute the branch metrics for all branches entering a node at time t

- Compute the path metrics for all paths entering a node at time t
 - Compare the path metrics and find the survivor for each node
 - Store the survivor and its path metric for each node
 - Repeat 2 until $t = \tau$
3. The survivor at the node S_τ is the maximum likelihood path and its metric is $\mu_{\tau,\min}$.

II Backward recursion

1. Set the initial values

$$t = \tau; \quad S_\tau = 0; \quad \mu_\tau^{(\mathbf{x})}(S_\tau = 0) = 0; \quad \mu_\tau^{(\mathbf{x})}(S_\tau \neq 0) = \infty; \quad S_0 = 0$$

2. Decrease time t by 1

- Compute the branch metrics for all branches entering a node at time t
- Compute the path metrics for all paths entering a node at time t
- Compare the path metrics and find the survivor for each node
- Store the survivor path metric for each node
- Repeat 2 until $t = 0$

III Soft decision

1. Set $t = 0$

2. Increase time t by 1

- At time t , identify the maximum likelihood estimate $c_t = i$, $i = 0, 1$
- Determine μ_t^i as

$$\mu_t^i = \mu_{\tau,\min}$$

- Find the path metric of its best competitor μ_t^c , $c = i \oplus 1$, where \oplus is modulo 2 sum, as

$$\mu_t^c = \min_{l,c} \{\mu_{t-1}^f(l') + \nu_t^c(l', l) + \mu_t^b(l)\} \quad (5.32)$$

where $l', l = 0, 1, \dots, M_s - 1$, $\mu_{t-1}^f(l')$ is the path metric of the forward survivor at time $t - 1$ and node l' , $\nu_t^c(l', l)$ is the branch metric at time t for a complement symbols c from node l' to l , $\mu_t^b(l)$ is the backward survivor path metric at time t and node l .

- Compute $\Lambda(c_t)$ as

$$\Lambda(c_t) = \mu_t^0 - \mu_t^1 \quad (5.33)$$

Note that in order to reduce the decoding complexity, the backward recursion and soft decision steps can be performed simultaneously.

Example 5.2 Consider the encoder shown in Fig. (5.2). The received sequence is given by

$$r_1^4 = [(1\ 1), (-1\ 1), (0.8\ -1), (-1\ -1)] \quad (5.34)$$

and the encoder starts and ends at state 0. The branch metrics for this received sequence are shown in Fig. 5.5. They are computed as in the standard VA, Eq. (5.22). The result of applying the VA in the forward recursion is shown in Fig. 5.6. The survivor path metrics are shown above each node and the ML path is represented by the thick line. The final survivor is the ML path with the metric $\mu_{4,\min} = 0.04$.

The results of the backward recursion are shown in Fig. 5.7. The first number above each node shows the forward survivor path metric, the second number shows the backward survivor path metric.

At time $t = 1$, the ML hard estimate is $c_1 = 1$ and thus $\mu_1^1 = \mu_{4,\min} = 0.04$. μ_1^0 is the minimum path metric of the paths that have 0 at time 1. There is only one path with zero at time 1, and

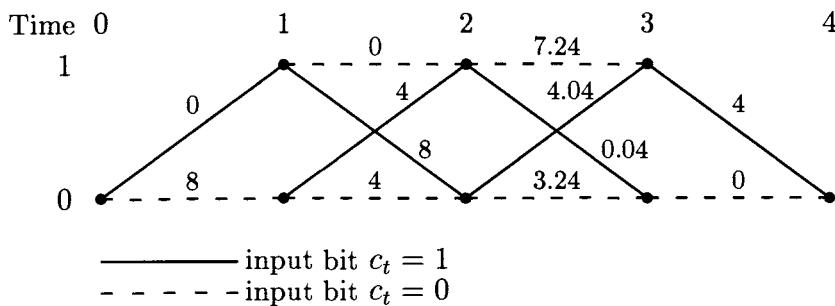


Fig. 5.5: The branch metrics in Example 5.2

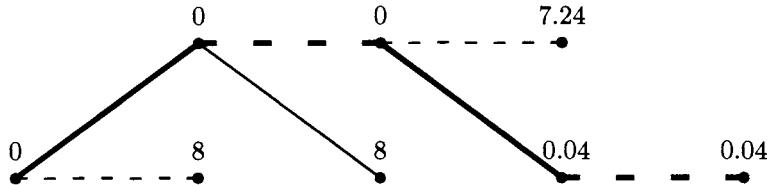


Fig. 5.6: The forward recursion in Example 5.2, the ML path is shown by the thick line

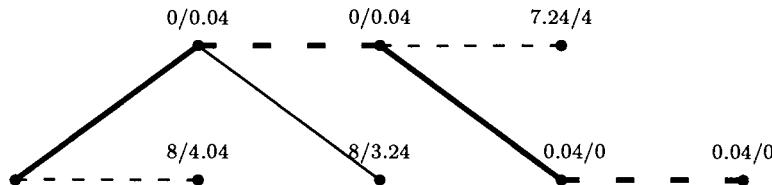


Fig. 5.7: The backward recursion in Example 5.2, the ML path is shown by the thick line

its path metric is calculated from Fig. 5.7 as the sum of the forward and backward survivor path metrics at node 0,

$$\mu_1^0 = 8 + 4.04 = 12.04 \quad (5.35)$$

The log-likelihood ratio at time 1 is given by

$$\begin{aligned}\Lambda(c_1) &= \mu_1^0 - \mu_1^1 \\ &= 12.04 - 0.04 \\ &= 12\end{aligned}\tag{5.36}$$

The ML symbol at time 2 is $c_2 = 0$, and the minimum path metric is $\mu_2^0 = \mu_{4,\min} = 0.04$. The best competitor path at time 2, denoted by μ_2^1 is obtained as

$$\mu_2^1 = \min_{l',1} \{\mu_1^f(l') + \nu_2^1(l', l) + \mu_2^b(l)\}\tag{5.37}$$

where $\mu_1^f(l')$ is the forward survivor path metric at time 1 and node l' , $l' = 1, 0$, $\nu_2^1(l', l)$ is the branch metric at time 2 for the input symbol 1 and $\mu_2^b(l)$ is the backward survivor path metric at time 2 and node l , $l = 1, 0$.

$$\begin{aligned}\mu_2^1 &= \min\{(0 + 8 + 3.24), (8 + 4 + 0.04)\} \\ &= 11.24\end{aligned}\tag{5.38}$$

The log-likelihood ratio at time 2 is given by

$$\Lambda(c_2) = \mu_2^0 - \mu_2^1 = 0.04 - 11.24 = -11.2\tag{5.39}$$

By applying this procedure to other time instants we obtain for other soft outputs

$$\Lambda(c_3) = 11.2 \quad \Lambda(c_4) = -11.2\tag{5.40}$$

The computational complexity of the forward recursion is equivalent to that of the Viterbi algorithm. The computational complexity of the backward recursion is usually less than that of the Viterbi algorithm, since there is no need to store backward survivors. Therefore, the computational complexity of the SOVA is upper-bounded by two times that of the VA. With binary inputs, the computational complexity is about 1.5 times of the VA.

The algorithm can be directly extended to the case $k > 1$, by considering all 2^{k-1} complement symbols to the ML symbol for each node in (5.32).

It can be similarly generalized to handle non-binary modulation schemes.

5.6 Sliding Window SOVA

The standard Viterbi decoder selects a path in the trellis with the minimum path metric. The decision making process is based on keeping only one path per node with the minimum metric at each time instant. The decisions are made with a delay, which depends on the code memory order. The minimum path metric is computed as the minimum among M_s survivor paths, where M_s is the number of states in the trellis.

In the standard SOVA algorithm the decision delay is equal to the received sequence length. For large sequence lengths, the memory required for decoder implementation is excessive. In this algorithm, the decisions are based on forward and backward recursions. The results of both recursions must be stored. The forward and backward recursions are performed on the whole sequence length, so that the decoder needs to store $\tau \times M_s$ survivors and survivor path metrics in each direction.

The SOVA algorithm can be simplified by reducing the required memory.

The decoder memory can be reduced as reliable decoding decisions for a convolutional code decoded by VA can be made after the decision depth which is about six times the encoder memory.

Another important point is that the Viterbi decoder can start at any time from any state. In the beginning, the path metrics that are generated are not reliable, but after several encoder memory lengths the results approach those achieved as if the decoder started from the initial node.

In the SOVA algorithm in both forward and backward recursions the decision on a symbol can be made with a delay at about six times the memory length. Let us assume that the required decision depth for a single component convolutional code is denoted by D . The decoder does not need to start from the initial node. It can start at any node and after the decision depth the decision can be made as reliably as starting from the initial node.

The simplified decoding operations can be summarized as follows.

Let the received sequence be divided into a few subblocks, with

subblock length of D . For the decoder output of each subblock, the forward and backward recursions are extended to the next subblock to generate the reliable metrics.

The forward algorithm processing starts at the initial time instant 0, computing path metrics for each node at each time instant, and storing these metrics in memory. At the $2D$ th time instant, it selects the path with the minimum path metric as the ML path, and makes the hard decisions for the first subblock. The backward recursion starts from the $2D$ th time instant and goes backward, setting each initial path metric to the same value and not storing anything until time instant D . By this time it has built up reliable path metrics. The decoding process is shown graphically in Fig. 5.8. (In this Figure, unreliable path metrics are shown by dashed lines.) The backward recursion continues from the D th time instant until it reaches the initial time instant 0. As soon as the backward recursion reaches time instant D , the decoder starts making soft decisions for the first subblock, based on both forward and backward recursions.

Note that since the soft decisions are computed at the same time as the backward recursions, there is no need to store all D backward path metrics, but only one for each node.

When the soft outputs are generated, all path metrics for the first subblock will be discarded and the emptied storage can be filled with the second subblock's metrics computed by the last forward recursion. The forward recursion computes the path metrics from the $(2D + 1)$ th to the $3D$ th time instant and selects the minimum path metric for the second subblock. The backward recursion begins from $3D$ th time instant and builds up reliable state metrics at $2D$ th time instant. Then simultaneously with the backward recursions, soft decisions from the $2D$ th to the D th time instant will be generated from the results of the forward and backward recursions.

The forward recursion only needs to store $2D \times M_s$ path metrics because this memory can be used repeatedly. The backward recursion just needs to store $1 \times M_s$ path metrics.

Note that for this algorithm the computation time is increased since the backward recursions are computed twice. However, the comparison of the standard SOVA and simplified SOVA shows that

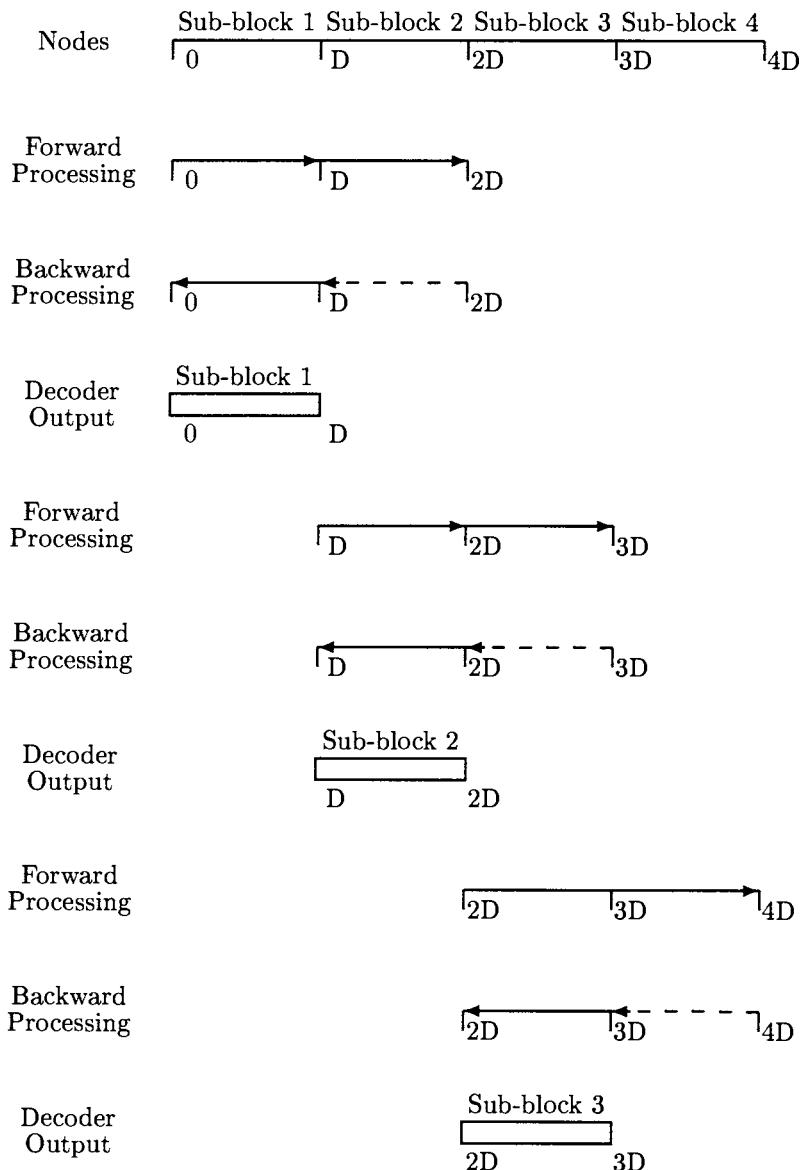


Fig. 5.8: Forward and Backward processing for the simplified SOVA

the increase in running time is marginal.

5.7 The MAP Algorithm

The MAP algorithm minimizes the symbol (or bit) error probability. For each transmitted symbol it generates its hard estimate and soft output in the form of the a posteriori probability on the basis of the received sequence \mathbf{r} . It computes the log-likelihood ratio

$$\Lambda(c_t) = \log \frac{P_r\{c_t = 1 | \mathbf{r}\}}{P_r\{c_t = 0 | \mathbf{r}\}} \quad (5.41)$$

for $1 \leq t \leq \tau$, where τ is the received sequence length, and compares this value to a zero threshold to determine the hard estimate c_t as

$$c_t = \begin{cases} 1 & \text{if } \Lambda(c_t) \geq 0 \\ 0 & \text{otherwise} \end{cases} \quad (5.42)$$

The value $\Lambda(c_t)$ represents the soft information associated with the hard estimate c_t . It might be used in a next decoding stage.

Consider again a system model shown in Fig. 5.1. For simplicity, we assume that a binary sequence \mathbf{c} of length N is encoded by a systematic convolutional code of rate $1/n$. The encoding process is modelled by a discrete-time finite-state Markov process described by a state and a trellis diagram with the number of states M_s . We assume that the initial state $S_0 = 0$ and the final state $S_\tau = 0$. The received sequence \mathbf{r} is corrupted by a zero-mean Gaussian noise with variance σ^2 .

As an example a rate 1/2 memory order 2 RSC encoder is shown in Fig. 5.9, and its state and trellis diagrams are illustrated in Figs. 5.10 and 5.11, respectively.

The content of the shift register in the encoder at time t represents S_t and it transits into S_{t+1} in response to the input c_{t+1} giving as output the coded block \mathbf{v}_{t+1} . The state transition of the encoder is shown in the state diagram.

The state transitions of the encoder are governed by the transition probabilities

$$p_t(l | l') = Pr \{S_t = l | S_{t-1} = l'\}; \quad 0 \leq l, l' \leq M_s - 1$$

The encoder output is determined by the probabilities

$$q_t(\mathbf{x}_t | l', l) = Pr \{\mathbf{x}_t | S_{t-1} = l', S_t = l\}; \quad 0 \leq l, l' \leq M_s - 1$$

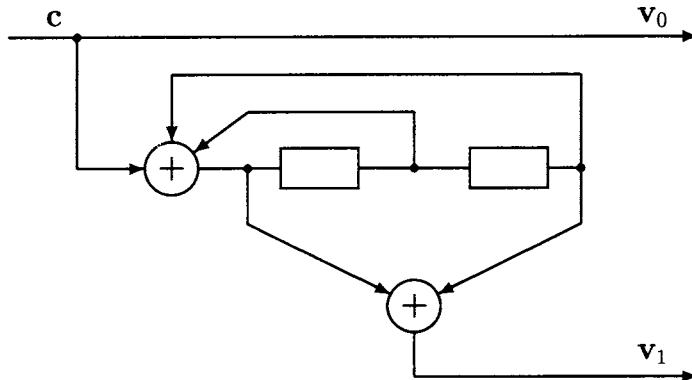


Fig. 5.9: A rate 1/2 memory order 2 RSC encoder

Because of one-to-one correspondence between \mathbf{x}_t and \mathbf{v}_t we have

$$q_t(\mathbf{x}_t | l', l) = \Pr\{\mathbf{v}_t | S_{t-1} = l', S_t = l\}; \quad 0 \leq l, \quad l' \leq M_s - 1$$

For the encoder in Fig. 5.9, $p_t(l | l')$ is either 0.5, when there is a connection from $S_{t-1} = l'$ to $S_t = l$ or 0 when there is no connection. $q_t(\mathbf{x} | l', l)$ is either 1 or 0. For example, from Figs. 5.10 and 5.11 we have

$$\begin{aligned} p_t(2|0) &= p_t(1) = 0.5; & p_t(1|2) &= p_t(1) = 0.5 \\ p_t(3|0) &= 0; & p_t(1|3) &= p_t(0) = 0.5 \end{aligned} \quad (5.43)$$

and

$$\begin{aligned} q_t(-1, -1|0, 0) &= 1 & q_t(-1, +1|0, 0) &= 0 \\ q_t(+1, -1|0, 1) &= 0 & q_t(+1, +1|0, 2) &= 1 \end{aligned} \quad (5.44)$$

For a given input sequence

$$\mathbf{c} = (c_1, c_2, \dots, c_N)$$

the encoding process starts at the initial state $S_0 = 0$ and produces an output sequence \mathbf{x}_1^τ ending in the terminal state $S_\tau = 0$, where $\tau = N + \nu$. The input to the channel is \mathbf{x}_1^τ and the output is $\mathbf{r}_1^\tau = (\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_\tau)$.

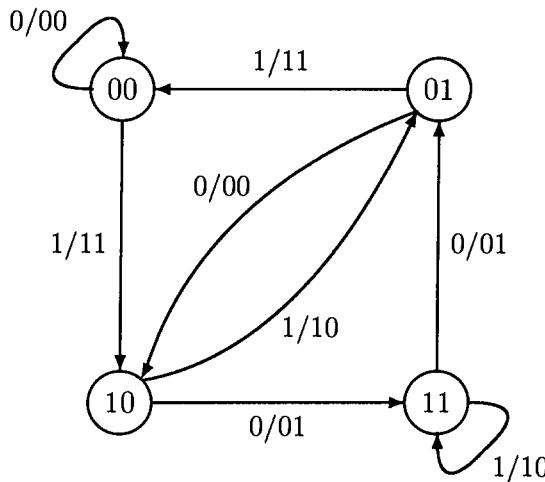


Fig. 5.10: State transition diagram for the (2,1,2) RSC code

The transition probabilities of the Gaussian channel are defined by

$$Pr \{ \mathbf{r}_1^\tau | \mathbf{x}_1^\tau \} = \prod_{j=1}^{\tau} R(\mathbf{r}_j | \mathbf{x}_j) \quad (5.45)$$

where

$$R(\mathbf{r}_j | \mathbf{x}_j) = \prod_{i=0}^{n-1} Pr(r_{j,i} | x_{j,i}) \quad (5.46)$$

and

$$Pr \{ r_{j,i} | x_{j,i} = -1 \} = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(r_{j,i}+1)^2}{2\sigma^2}} \quad (5.47)$$

$$Pr \{ r_{j,i} | x_{j,i} = +1 \} = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(r_{j,i}-1)^2}{2\sigma^2}} \quad (5.48)$$

where σ^2 is the noise variance.

Let c_t be the information bit associated with the transition S_{t-1} to S_t , producing as output \mathbf{v}_t . The decoder gives an estimate of

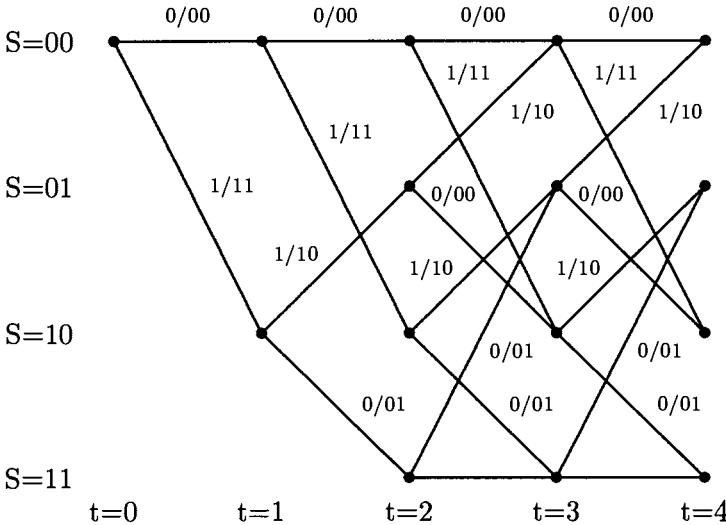


Fig. 5.11: Trellis diagram for the (2,1,2) RSC code

the input to the Markov source, by examining \mathbf{r}_t^τ . The MAP algorithm provides the log likelihood ratio, denoted by $\Lambda(c_t)$, given the received sequence \mathbf{r}_1^τ , as indicated in Eq. (5.41). The decoder makes a decision by comparing $\Lambda(c_t)$ to a threshold equal to zero.

We can compute the APP's in (5.41) as

$$\begin{aligned} Pr \{c_t = 0 | \mathbf{r}_1^\tau\} &= \sum_{(l', l) \in B_t^0} Pr \{S_{t-1} = l', S_t = l | \mathbf{r}_1^\tau\} \\ &= \sum_{(l', l) \in B_t^0} \frac{Pr \{S_{t-1} = l', S_t = l, \mathbf{r}_1^\tau\}}{Pr \{\mathbf{r}_1^\tau\}} \quad (5.49) \end{aligned}$$

where B_t^0 is the set of transitions $S_{t-1} = l' \rightarrow S_t = l$ that are caused by the input bit $c_t = 0$. For example, B_t^0 for the diagram in Fig. 5.11 are (3,1), (0,0), (1,2) and (2,3).

Also

$$\begin{aligned} Pr \{c_t = 1 | \mathbf{r}_1^\tau\} &= \sum_{(l', l) \in B_t^1} Pr \{S_{t-1} = l', S_t = l | \mathbf{r}_1^\tau\} \\ &= \sum_{(l', l) \in B_t^1} \frac{Pr \{S_{t-1} = l', S_t = l, \mathbf{r}_1^\tau\}}{Pr \{\mathbf{r}_1^\tau\}} \quad (5.50) \end{aligned}$$

where B_t^1 is the set of transitions $S_{t-1} = l' \rightarrow S_t = l$ that are caused by the input bit $c_t = 1$. For the diagram in Fig. 5.11, B_t^1 consists of (0,2), (2,1), (3,3) and (1,0).

The APP's of the decoded data bit c_t can be derived from the joint probability defined as

$$\sigma_t(l', l) = Pr \{S_{t-1} = l', S_t = l, \mathbf{r}_1^\tau\}, \quad l', l = 0, 1, \dots, M_s - 1$$

Eqs. (5.49) and (5.50) can be written as

$$Pr \{c_t = 0 | \mathbf{r}_1^\tau\} = \sum_{(l', l) \in B_t^0} \frac{\sigma_t(l', l)}{Pr \{\mathbf{r}_1^\tau\}} \quad (5.51)$$

$$Pr \{c_t = 1 | \mathbf{r}_1^\tau\} = \sum_{(l', l) \in B_t^1} \frac{\sigma_t(l', l)}{Pr \{\mathbf{r}_1^\tau\}} \quad (5.52)$$

The log-likelihood ratio $\Lambda(c_t)$ is then

$$\Lambda(c_t) = \log \frac{\sum_{(l', l) \in B_t^1} \sigma_t(l', l)}{\sum_{(l', l) \in B_t^0} \sigma_t(l', l)} \quad (5.53)$$

The log-likelihood $\Lambda(c_t)$ represents the soft output of the MAP decoder. It can be used as an input to another decoder in a concatenated scheme or in the next iteration in an iterative decoder. In the final operation, the decoder makes a hard decision by comparing $\Lambda(c_t)$ to a threshold equal to zero.

Definitions of α , β and γ

In order to compute the joint probability $\sigma_t(l', l)$ required for calculation of $\Lambda(c_t)$ in (5.53), we define the following probabilities

$$\alpha_t(l) = Pr \{S_t = l, \mathbf{r}_1^\tau\} \quad (5.54)$$

$$\beta_t(l) = Pr \{\mathbf{r}_{t+1}^\tau | S_t = l\} \quad (5.55)$$

$$\gamma_t^i(l', l) = Pr \{c_t = i, S_t = l, \mathbf{r}_t | S_{t-1} = l'\}; \quad i = 0, 1 \quad (5.56)$$

Now we can express $\sigma_t(l', l)$ as

$$\begin{aligned}
 \sigma_t(l', l) &= Pr\{S_{t-1} = l', S_t = l, \mathbf{r}_1^t\} \\
 &= Pr\{\mathbf{r}_{t+1}^t, \mathbf{r}_t, \mathbf{r}_1^{t-1}, S_t = l, S_{t-1} = l'\} \\
 &= Pr\{\mathbf{r}_{t+1}^t | \mathbf{r}_t, \mathbf{r}_1^{t-1}, S_t = l, S_{t-1} = l'\} \\
 &\quad \cdot Pr\{\mathbf{r}_t, \mathbf{r}_1^{t-1}, S_t = l, S_{t-1} = l'\} \\
 &= Pr\{\mathbf{r}_{t+1}^t | S_t = l\} Pr\{\mathbf{r}_t, \mathbf{r}_1^{t-1}, S_t = l, S_{t-1} = l'\} \\
 &= \beta_t(l) \cdot Pr\{\mathbf{r}_t, \mathbf{r}_1^{t-1}, S_t = l, S_{t-1} = l'\} \\
 &= \beta_t(l) \cdot Pr\{S_t = l, \mathbf{r}_t | S_{t-1} = l', \mathbf{r}_1^{t-1}\} \\
 &\quad \cdot Pr\{S_{t-1} = l', \mathbf{r}_1^{t-1}\} \\
 &= Pr\{S_{t-1} = l', \mathbf{r}_1^{t-1}\} \cdot \beta_t(l) \cdot Pr\{S_t = l, \mathbf{r}_t | S_{t-1} = l'\} \\
 &= \alpha_{t-1}(l') \cdot \beta_t(l) \cdot \sum_{i \in (0,1)} \gamma_t^i(l', l)
 \end{aligned} \tag{5.57}$$

The log-likelihood ratio $\Lambda(c_t)$ can be written as

$$\Lambda(c_t) = \log \frac{\sum_{(l', l) \in B_t^1} \alpha_{t-1}(l') \gamma_t^1(l', l) \beta_t(l)}{\sum_{(l', l) \in B_t^0} \alpha_{t-1}(l') \gamma_t^0(l', l) \beta_t(l)} \tag{5.58}$$

Derivation of α

We can obtain α defined in (5.54) as

$$\begin{aligned}
 \alpha_t(l) &= Pr\{S_t = l, \mathbf{r}_1^t\} \\
 &= \sum_{l'=0}^{M_s-1} Pr\{S_{t-1} = l', S_t = l, \mathbf{r}_1^t\} \\
 &= \sum_{l'=0}^{M_s-1} Pr\{S_{t-1} = l', S_t = l, \mathbf{r}_1^{t-1}, \mathbf{r}_t\} \\
 &= \sum_{l'=0}^{M_s-1} Pr\{S_{t-1} = l', \mathbf{r}_1^{t-1}\} \\
 &\quad \cdot Pr\{S_t = l, \mathbf{r}_t | S_{t-1} = l', \mathbf{r}_1^{t-1}\} \\
 &= \sum_{l'=0}^{M_s-1} \alpha_{t-1}(l') \cdot Pr\{S_t = l, \mathbf{r}_t | S_{t-1} = l'\} \\
 &= \sum_{l'=0}^{M_s-1} \alpha_{t-1}(l') \cdot \sum_{i \in (0,1)} Pr\{S_t = l, c_t = i, \mathbf{r}_t | S_{t-1} = l'\} \\
 &= \sum_{l'=0}^{M_s-1} \alpha_{t-1}(l') \cdot \sum_{i \in (0,1)} \gamma_t^i(l', l)
 \end{aligned} \tag{5.59}$$

for $t = 1, 2, \dots, \tau$.

For $t = 0$ we have the boundary conditions $\alpha_0(0) = 1$ and $\alpha_0(l) = 0$ for $l \neq 0$.

Derivation of β

We can express $\beta_t(l)$ defined in (5.55) as

$$\begin{aligned}
 \beta_t(l) &= \Pr \left\{ \mathbf{r}_{t+1}^{\tau} | S_t = l \right\} \\
 &= \sum_{l'=0}^{M_s-1} \Pr \left\{ S_{t+1} = l', \mathbf{r}_{t+1}^{\tau} | S_t = l \right\} \\
 &= \sum_{l'=0}^{M_s-1} \frac{\Pr \left\{ S_{t+1} = l', \mathbf{r}_{t+1}^{\tau}, S_t = l \right\}}{\Pr \left\{ S_t = l \right\}} \\
 &= \sum_{l'=0}^{M_s-1} \frac{\Pr \left\{ \mathbf{r}_{t+2}^{\tau}, \mathbf{r}_{t+1}, S_{t+1} = l', S_t = l \right\}}{\Pr \left\{ S_t = l \right\}} \\
 &= \sum_{l'=0}^{M_s-1} \frac{\Pr \left\{ \mathbf{r}_{t+2}^{\tau} | \mathbf{r}_{t+1}, S_{t+1} = l', S_t = l \right\} \Pr \left\{ \mathbf{r}_{t+1}, S_{t+1} = l', S_t = l \right\}}{\Pr \left\{ S_t = l \right\}} \\
 &= \sum_{l'=0}^{M_s-1} \frac{\Pr \left\{ \mathbf{r}_{t+2}^{\tau} | S_{t+1} = l' \right\} \cdot \Pr \left\{ \mathbf{r}_{t+1}, S_{t+1} = l', S_t = l \right\}}{\Pr \left\{ S_t = l \right\}} \\
 &= \sum_{l'=0}^{M_s-1} \frac{\beta_{t+1}(l') \Pr \left\{ S_{t+1} = l', \mathbf{r}_{t+1} | S_t = l \right\}}{\Pr \left\{ S_t = l \right\}} \\
 &= \sum_{l'=0}^{M_s-1} \beta_{t+1}(l') \Pr \left\{ S_{t+1} = l', \mathbf{r}_{t+1} | S_t = l \right\} \\
 &\quad \cdot \sum_{i \in \{0,1\}} \Pr \left\{ c_{t+1} = i, S_{t+1} = l', \mathbf{r}_{t+1} | S_t = l \right\} \\
 &= \sum_{l'=0}^{M_s-1} \beta_{t+1}(l') \sum_{i \in \{0,1\}} \gamma_{t+1}^i(l, l')
 \end{aligned} \tag{5.60}$$

for $t = \tau - 1, \dots, 1, 0$.

The boundary conditions are $\beta_{\tau}(0) = 1$ and $\beta_{\tau}(l) = 0$ for $l \neq 0$.

Derivation of γ

We can write for $\gamma_t^i(l', l)$ defined in (5.56)

$$\begin{aligned}
 \gamma_t^i(l', l) &= \Pr \left\{ c_t = i, S_t = l, \mathbf{r}_t | S_{t-1} = l' \right\} \\
 &= \frac{\Pr \left\{ \mathbf{r}_t, c_t = i, S_t = l, S_{t-1} = l' \right\}}{\Pr \left\{ S_{t-1} = l' \right\}} \\
 &= \frac{\Pr \left\{ \mathbf{r}_t | c_t = i, S_t = l, S_{t-1} = l' \right\} \cdot \Pr \left\{ c_t = i, S_t = l, S_{t-1} = l' \right\}}{\Pr \left\{ S_{t-1} = l' \right\}} \\
 &= \frac{\Pr \left\{ \mathbf{r}_t | \mathbf{x}_t \right\} \cdot \Pr \left\{ c_t = i, S_t = l, S_{t-1} = l' \right\}}{\Pr \left\{ S_{t-1} = l' \right\}} \\
 &= \frac{\Pr \left\{ \mathbf{r}_t | \mathbf{x}_t \right\} \cdot \Pr \left\{ c_t = i | S_t = l, S_{t-1} = l' \right\} \cdot \Pr \left\{ S_t = l, S_{t-1} = l' \right\}}{\Pr \left\{ S_{t-1} = l' \right\}} \\
 &= \Pr \left\{ \mathbf{r}_t | \mathbf{x}_t \right\} \cdot \Pr \left\{ \mathbf{x}_t | S_t = l, S_{t-1} = l' \right\} \\
 &\quad \cdot \Pr \left\{ S_t = l | S_{t-1} = l' \right\} \\
 &= p_t(l | l') \cdot q_t(\mathbf{x}_t | l', l) \cdot R(\mathbf{r}_t | \mathbf{x}_t)
 \end{aligned}$$

We can further express $\gamma_t^i(l', l)$ as

$$\gamma_t^i(l', l) = \begin{cases} p_t(i) \exp\left(-\frac{\sum_{j=0}^{n-1} (r_{t,j}^i - x_{t,j}^i(l))^2}{2\sigma^2}\right) & \text{for } (l, l') \in B_t^i \\ 0 & \text{otherwise} \end{cases} \quad (5.61)$$

where $p_t(i)$ is the a priori probability of $c_t = i$ and $x_{t,j}^i(l)$ is the encoder output associated with the transition $S_{t-1} = l'$ to $S_t = l$ and input $c_t = i$. Note that the expression for $R(\mathbf{r}_t | \mathbf{x}_t)$ is normalized by multiplying (5.46) $(\sqrt{2\pi}\sigma)^n$.

Summary of the MAP Algorithm

1. Forward recursion

- Initialize $\alpha_0(l)$, $l = 0, 1, \dots, M_s - 1$
 $\alpha_0(0) = 1$ and $\alpha_0(l) = 0$ for $l \neq 0$
- For $t = 1, 2, \dots, \tau$, $l = 0, 1, \dots, M_s - 1$ and all branches in the trellis calculate

$$\gamma_t^i(l', l) = p_t(i) \exp\left(-\frac{d^2(\mathbf{r}_t, \mathbf{x}_t)}{2\sigma^2}\right) \quad \text{for } i = 0, 1 \quad (5.62)$$

where $p_t(i)$ is the a priori probability of each information bit, $d^2(\mathbf{r}_t, \mathbf{x}_t)$ is the squared Euclidean distance between \mathbf{r}_t and the modulated symbol in the trellis \mathbf{x}_t .

- For $i = 0, 1$ store $\gamma_t^i(l', l)$.
- For $t = 1, 2, \dots, \tau$, and $l = 0, 1, \dots, M_s - 1$ calculate and store $\alpha_t(l)$

$$\alpha_t(l) = \sum_{l'=0}^{M_s-1} \sum_{i \in \{0,1\}} \alpha_{t-1}(l') \gamma_t^i(l', l) \quad (5.63)$$

The graphical representation of the forward recursion is given in Fig. 5.12.

2. Backward recursion

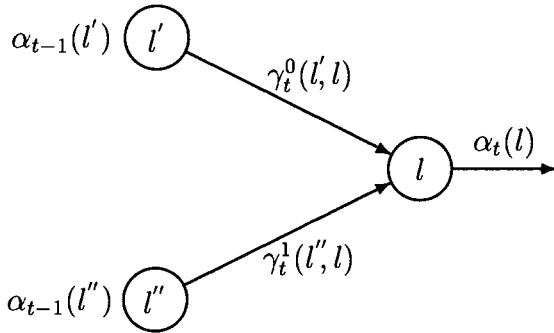


Fig. 5.12: Graphical representation of the forward recursion

- Initialize $\beta_\tau(l)$, $l = 0, 1, \dots, M_s - 1$
 $\beta_\tau(0) = 1$ and $\beta_\tau(l) = 0$ for $l \neq 0$
- For $t = \tau - 1, \dots, 1, 0$ and $l = 0, 1, \dots, M_s - 1$ calculate $\beta_t(l)$ as

$$\beta_t(l) = \sum_{l'=0}^{M_s-1} \sum_{i \in \{0,1\}} \beta_{t+1}(l') \gamma_{t+1}^i(l, l') \quad (5.64)$$

where $\gamma_{t+1}^i(l, l')$ was computed in the forward recursion.

- For $t < \tau$ calculate the log-likelihood $\Lambda(c_t)$ as

$$\Lambda(c_t) = \log \frac{\sum_{l=0}^{M_s-1} \alpha_{t-1}(l') \gamma_t^1(l', l) \beta_t(l)}{\sum_{l=0}^{M_s-1} \alpha_{t-1}(l') \gamma_t^0(l', l) \beta_t(l)} \quad (5.65)$$

The graphical representation of the backward recursion is shown in Fig. 5.13.

Note that because Eq. (5.65) is a ratio, the values for $\alpha_t(l')$ and $\beta_t(l)$ can be normalized at any node which keeps them from overflowing.

The MAP algorithm can be applied only if the sequence length τ is finite, while the VA can be used for any sequence length, by truncating the survivors.

However, it is possible to apply the sliding window MAP decoding as for the SOVA algorithm, in the same way as shown in Section 5.6 to achieve memory savings.

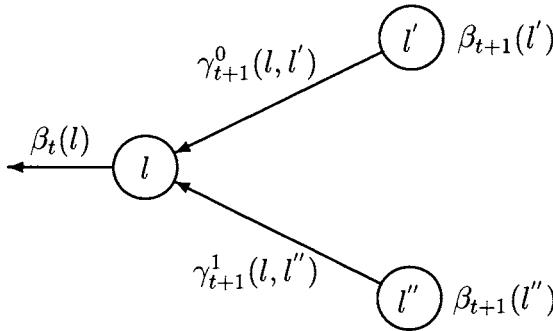


Fig. 5.13: Graphical representation of the backward recursion

The MAP requires the storage of $M_s \times \tau$ memory units to store the results of the forward recursion. The memory site grows linearly with the sequence length τ . If the number of branches leaving and entering each node is $N_b = 2^k$, where k is the number of information bits at the encoder input, then for each time unit $M_s \cdot N_b$ multiplications and $M_s(N_b - 1)$ additions are needed for both the forward and backward recursions. The computational requirement per time unit is constant and independent of the sequence length τ . The branch metric for the MAP is given by Eq. (5.62).

Comparing this expression with Eq. (5.22), we can see that the branch metric for the MAP is much more complex to calculate than for the VA. For this reason the branch metric calculation can be implemented as a look-up table.

It is worth noting that Chang and Hancock [11] developed a MAP algorithm similar to the one in [9] for removal of intersymbol interference (ISI). For the same application, Abend and Fritchman [8] proposed a different MAP algorithm, which requires only a forward recursion and can be used in continuous mode decoding. Raviv [7] used this MAP algorithm for character recognition. However, the memory and computational complexity of this algorithm grow exponentially with the decoding delay. A MAP symbol-by-symbol decoding algorithm with a fixed decoding delay and linear memory and complexity growth with decoding delay was proposed in [14]. A soft-input soft-output (SISO) MAP algorithm for decoding of turbo and serial concatenated codes has been reported in

[16].

It is also worth mentioning that the branch metric is dependent on the noise variance σ^2 , while the branch metric for the VA and SOVA is not. However, it has been shown that, at high SNR's, the accurate estimation of the noise variance is not very important [17].

If the final state of the trellis is not known, the probability $\beta_\tau(l)$, can be initialized as

$$\beta_\tau(l) = \frac{1}{M_s}, \quad \forall l \quad (5.66)$$

Example 5.3 In a system encoded by the $(2, 1, 1)$ recursive convolutional code with the generator matrix $(1, \frac{1}{1+D})$ with BPSK modulation and $\frac{E_b}{N_0} = 2dB$, the transmitted sequence is given by

$$\mathbf{c} = (11001) \quad (5.67)$$

After time $t = 5$ the encoder was driven to the zero state. The received sequence is given by

$$\mathbf{r} = (r_{1,0}, r_{1,1}, r_{2,0}, r_{2,1}, \dots, r_{6,0}, r_{6,1}) \quad (5.68)$$

and the values for $r_{t,j}$, $t = 1, 2, \dots, 6$, $j = 0, 1$, are shown in Table 5.1. Determine the soft and hard outputs of a MAP decoder.

Solution

The trellis diagram for this code is shown in Fig. 5.14.

The branch metrics $\gamma_t^{(i)}(l, l')$, $l, l' = 0, 1$, $t = 1, 2, \dots, 6$ are computed by Eq. (5.61).

Starting from the initial condition $\alpha_0(0) = 1$, $\alpha_0(1) = 0$ and using the forward recursion formula in Eq. (5.59) $\alpha_t(l)$, $l = 0, 1$, $t = 1, \dots, 6$, are computed.

Similarly, using the initial conditions $\beta_6(0) = 1$ and $\beta_6(1) = 0$, $\beta_t(l)$, $l = 0, 1$, $t = 1, 2, \dots, 6$, are computed by the backward recursion formula in Eq. (5.60).

The soft-outputs are calculated by Eq. (5.65).

The hard estimates \hat{c}_t are obtained by comparing the soft output $\Lambda_t(c_t)$ to the threshold of 0.

The values of c_t , $\gamma_t^{(i)}(l, l')$, $\alpha_t(l)$, $\beta_t(l)$, $\Lambda_t(c_t)$ and \hat{c}_t are shown in Table 5.1.

Table 5.1

t	1	2	3	4	5	6
c_t	1	1	0	0	1	
r_t^1	0.030041	-0.570849	-0.38405	-0.744790	0.525812	0.507154
r_t^2	0.726249	-0.753015	-1.107597	-0.495092	1.904994	-1.591323
$\gamma_t^0(0, 0)$	0.034678	0.425261	0.386226	0.404741	0.000408	0.088558
$\gamma_t^0(1, 1)$	0.236152	0.058185	0.020712	0.109450	0.062573	0.001323
$\gamma_t^1(0, 1)$	0.255655	0.012881	0.007510	0.015309	0.250953	0.005052
$\gamma_t^1(1, 0)$	0.037542	0.094143	0.140044	0.056593	0.001638	0.338085
$\alpha_t(0)$	0.034678	0.038815	0.017137	0.006971	0.000003	0.000599
$\alpha_t(1)$	0.255655	0.015322	0.000609	0.000329	0.001770	0.000002
$\beta_t(0)$	0.005783	0.013448	0.034680	0.089880	0.088558	1
$\beta_t(1)$	0.001557	0.005005	0.007135	0.021300	0.338085	0
$\Lambda_t(c_t)$	0.685647	0.177998	-1.920772	-4.239018	4.407100	7.598074
\hat{c}_t	1	1	0	0	1	

5.8 The Max-Log-MAP Algorithm

The MAP decoding requires large memory and a large number of operations involving exponentiations and multiplications. The algorithm is likely to be considered too complex for implementation in many communication systems.

One way of simplifying computations is to work with the logarithms of $\gamma_t^i(l', l)$, $\alpha_t(l)$ and $\beta_t(l)$, denoted by $\bar{\gamma}_t^i(l', l)$, $\bar{\alpha}_t(l)$ and $\bar{\beta}_t(l)$, respectively. $\bar{\gamma}_t^i(l', l)$ is given by

$$\bar{\gamma}_t^i(l', l) = \log \gamma_t^i(l', l) \quad (5.69)$$

Referring to Eq. (5.59), $\bar{\alpha}_t(l)$ can be expressed as

$$\begin{aligned} \bar{\alpha}_t(l) &= \log \alpha_t(l) \\ &= \log \sum_{l'=0}^{M_s-1} \sum_{i \in (0,1)} e^{\bar{\alpha}_{t-1}(l') + \bar{\gamma}_t^i(l', l)} \end{aligned} \quad (5.70)$$

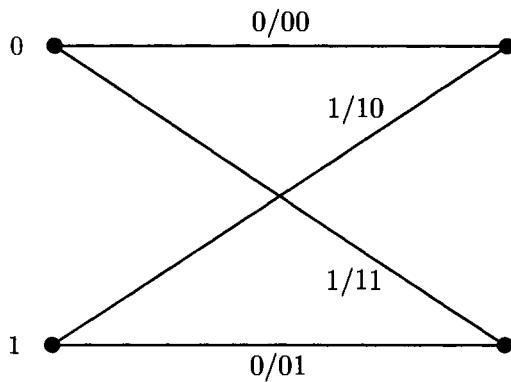


Fig. 5.14: Trellis diagram for the encoder in Example 5.3

with the initial conditions

$$\bar{\alpha}_0(0) = 0 \text{ and } \bar{\alpha}_0(l) = -\infty \text{ for } l \neq 0$$

Similarly, $\bar{\beta}_t(l)$ can be represented as (Eq. (5.60))

$$\begin{aligned} \bar{\beta}_t(l) &= \log \beta_t(l) \\ &= \log \sum_{l'=0}^{M_s-1} \sum_{i \in \{0,1\}} e^{\bar{\beta}_{t+1}(l') + \bar{\gamma}_{t+1}(l, l')} \end{aligned}$$

with the initial conditions

$$\bar{\beta}_\tau = 0 \text{ and } \bar{\beta}_\tau(l) = -\infty \text{ and } l \neq 0$$

By substituting values for $\bar{\alpha}_t(\alpha)$, $\bar{\beta}_t(l)$ and $\bar{\gamma}(l)$ in Eq. (5.65), the log-likelihood $\Lambda(c_t)$ can be expressed as

$$\Lambda(c_t) = \log \frac{\sum_{l=0}^{M_s-1} e^{\bar{\alpha}_{t-1}(l') + \bar{\gamma}_t^1(l', l) + \bar{\beta}_t(l)}}{\sum_{l=0}^{M_s-1} e^{\bar{\alpha}_{t-1}(l') + \bar{\gamma}_t'(l', l) + \bar{\beta}_t(l)}} \quad (5.71)$$

This expression can be simplified by using the approximation

$$\log(e^{\delta_1} + e^{\delta_2} + \dots + e^{\delta_n}) \approx \max_{i \in \{1, 2, \dots, n\}} \delta_i \quad (5.72)$$

where $\max_{i \in \{1, 2, \dots, n\}} \delta_i$ can be computed by successively calculating $(n-1)$ maximum function over only two values.

The log-likelihood $\Lambda(c_t)$ can now be approximated by

$$\begin{aligned}\Lambda(c_t) \approx & \max_l \left[\bar{\gamma}_t^1(l', l) + \bar{\alpha}_{t-1}(l') + \bar{\beta}_t(l) \right] \\ & - \max_l \left[\bar{\gamma}_t^0(l', l) + \bar{\alpha}_{t-1}(l') + \bar{\beta}_t(l) \right]\end{aligned}\quad (5.73)$$

The computations of $\bar{\alpha}_t(l')$ and $\bar{\beta}_t(l)$ in Eq. (5.73) is equivalent to the computation of path metrics in the forward and backward recursions, respectively, in the VA, with the branch metric $\bar{\gamma}_t^i(l', l)$, since $\bar{\alpha}_t(l)$ and $\bar{\beta}_t(l)$ can be written as

$$\begin{aligned}\bar{\alpha}_t(l) &= \max\{\bar{\alpha}_{t-1}(l') + \bar{\gamma}_t^i(l', l)\} \quad \text{for } 0 \leq l' \leq M_s - 1, i = 0, 1 \\ \bar{\beta}_t(l) &= \max\{\bar{\beta}_{t+1}(l') + \bar{\gamma}_{t+1}^i(l, l')\} \quad \text{for } 0 \leq l' \leq M_s - 1, i = 0, 1\end{aligned}$$

The operations involved in computing $\bar{\alpha}_t(l)$ and $\bar{\beta}_t(l)$ are the same as the add-compare-select operations in the VA. Thus multiplications in the MAP are replaced by additions in the Max-Log-MAP.

For each bit, the Max-Log-MAP algorithm calculates two Viterbi type metrics and takes the largest one.

5.9 The Log-MAP Algorithm

As the approximation in Eq. (5.72) is used for the computation of the log-likelihood function $\Lambda(c_t)$ the performance of the Max-Log-MAP is suboptimal. It could be improved by using the Jacobian algorithm [5]

$$\begin{aligned}\log(e^{\delta_1} + e^{\delta_2}) &= \max(\delta_1, \delta_2) + \log(1 + e^{-|\delta_2 - \delta_1|}) \\ &= \max(\delta_1, \delta_2) + f_c(|\delta_2 - \delta_1|)\end{aligned}\quad (5.74)$$

where $f_c(\cdot)$ is a correction function, which can be implemented using a look-up table.

The expression $\log(e^{\delta_1} + \dots + e^{\delta_n})$ can be computed exactly by a recursive algorithm in Eq. (5.74), as follows

$$\begin{aligned}\log(e^{\delta_1} + \dots + e^{\delta_n}) &= \log(\Delta + e^{\delta_n}), \quad \Delta = e^{\delta_1} + \dots + e^{\delta_{n-1}} = e^\delta \\ &= \max(\log \Delta, \delta_n) + f_c(|\log \Delta - \delta_n|) \\ &= \max(\delta, \delta_n) + f_c(|\delta - \delta_n|)\end{aligned}\quad (5.75)$$

The recursive procedure in (5.75) can be applied to evaluating $\Lambda(c_t)$ in (5.71), where

$$\delta_n = \bar{\alpha}_{t-1}(n') + \bar{\gamma}_t^i(n', n) + \bar{\beta}_t(n) \quad (5.76)$$

for $n = 0, 1, \dots, M_s - 1$, and $i = 0, 1$.

The performance of the Log-MAP is identical to the performance of the MAP algorithm. However, by computing $f_c(\cdot)$ at each step, there is a sacrifice in the low complexity of the Max-Log-MAP. To simplify the computations, $f_c(\cdot)$ is stored in a pre-computed table. Since the correction depends only on $|\delta - \delta_n|$, this table is one-dimensional.

5.10 Comparison of Decoding Algorithms

In computation of the log-likelihood for each received symbol, the MAP algorithm considers all paths in the trellis, but divides them into two sets, one that has a bit one at time t and the other that has a bit zero. It then calculates the log-likelihood function for all of these two sets.

The Max-Log-MAP considers only two paths per step: the best path with bit zero and the best path with bit one at time t . It computes the log-likelihood for each of the paths and returns its difference. From step to step one of these paths might change but one of them will always be the ML path.

The SOVA also takes two paths. One is the ML path and the other is the best path with the complementary symbol at time t to the ML path. Therefore, these two paths are identical to the two paths considered by the Max-Log-MAP algorithm.

A complexity comparison between MAP, SOVA, Log-MAP and Max-Log-MAP, per one time unit is illustrated in Table 5.1. Each algorithm is represented by the number of computation operations for an (n, k) convolutional code of memory order ν . The Log-MAP can be implemented by a look-up table. It has been shown that it is enough to store 8 values of $|\delta_2 - \delta_1|$, ranging between 0 and 5

[10], with no improvement with a finer representation. Assuming that one look-up operation costs as much as one addition, we can conclude that the Log-MAP is 3 times more complex than SOVA. On the other hand Max-Log-MAP is about twice as complex as SOVA.

Table 5.1: Decoder complexity comparison

	MAP	Log-MAP	Max-Log-MAP	SOVA
add.	$2 \cdot 2^k \cdot 2^\nu + 6$	$6 \cdot 2^k \cdot 2^\nu + 6$	$4 \cdot 2^k \cdot 2^\nu + 8$	$2 \cdot 2^k \cdot 2^\nu + 9$
multipl.	$5 \cdot 2^k \cdot 2^\nu + 8$	$2^k \cdot 2^\nu$	$2 \cdot 2^k \cdot 2^\nu$	$2^k \cdot 2^\nu$
max ops		$4 \cdot 2^\nu - 2$	$4 \cdot 2^\nu - 2$	
look-ups		$4 \cdot 2^\nu - 2$		$2 \cdot 2^\nu - 1$
exp.	$2 \cdot 2^k \cdot 2^\nu$			

The performance of the soft output algorithms cannot be adequately compared in only one stage of decoding by the BER results, because the BER is defined only for hard decision estimates. Thus the comparison on the basis of BER refers only to hard decision estimates. Fig. 5.15 shows the BER results obtained by simulations for a 4-state convolutional code with the MAP and SOVA decoding algorithms. The MAP and the Log-MAP are identical, as well as the Max-Log-MAP and the SOVA. The difference between the MAP and the SOVA becomes noticeable only at very high BER's, such as BER is above 10^{-2} .

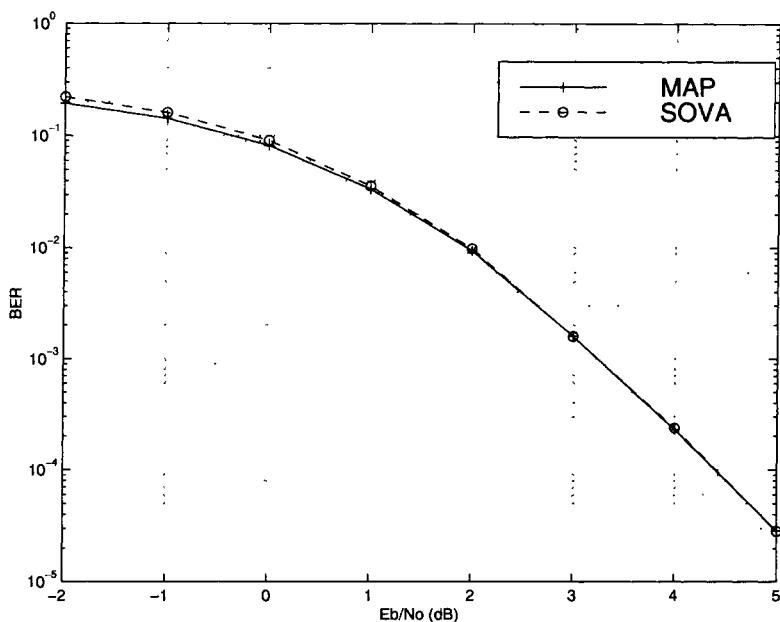


Fig. 5.15: Performance comparison of MAP and SOVA

Bibliography

- [1] A.J. Viterbi, “Error bounds for convolutional codes and an asymptotically optimum decoding algorithm”, IEEE Trans. Inform Theory, Vol.IT-13, No.2, pp.260-269, April 1967
- [2] B. Vucetic, “Iterative Decoding Algorithms”, Invited paper, The International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC’97, Sept. 1-4, Helsinki, Finland.
- [3] B. Vucetic, Iterative Decoding Algorithms, Wireless Communications, S. Glisic and P. Leppnen, Edt., Kluwer, 1997.
- [4] G.D. Forney, Jr. “The Viterbi algorithm”, Proc. IEEE, Vol.61, No. 3, pp.268-278, March 1973
- [5] J.A. Erfanian, S.Pasupathy and G.Gulot, “Reduced complexity symbol detectors with parallel structures for ISI channels”, IEEE Trans. Commun., Vol. 42, pp. 1661-1671, Feb./March/April 1994.
- [6] J. Hagenauer and P. Hoeher, “A Viterbi algorithm with soft-decision outputs and its applications”, Proc. Globecom’89, pp.1680-1686, Nov. 1989.
- [7] J. Raviv, “Decision making in Markov chains applied to the problem of pattern recognition”, IEEE Trans. Inform Theory, Vol.IT-13, pp. 536-551, Oct. 1967.
- [8] K. Abend and B.D. Fritchman, “Statistical detection for communication channels with intersymbol interference”, Proc. IEEE, Vol. 58, No. 5, pp.779-785, May 1970.

- [9] L. Bahl, J.Cocke, F. Jelinek and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate", IEEE Trans. Inform Theory, Vol.IT-20, pp.284-287, March 1979
- [10] P. Robertson, E. Villebrun and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain", Proc. ICC'95, Seattle, June 1995.
- [11] R.W. Chang and J.C. Hancock, "On receiver structures for channels having memory", IEEE Trans. Inform Theory, Vol. IT-12, pp. 463-468, Oct. 1966.
- [12] Y. Li and B. Vucetic, "A Generalized MLSE Algorithm", INNSP '95, China, December 10-13, 1995.
- [13] Y. Li and B. Vucetic, "A Low-complexity Soft-output TDMA Receiver", TELFOR '95, Belgrade, 3-8 Dec., Yugoslavia.
- [14] Y. Li, B. Vucetic and Y. Sato, "Optimum soft-output detection for channels with inter-symbol inference", IEEE Trans. Inform. Theory, Vol-41, No. 3, May 1995, pp. 704-713.
- [15] Y. Li and B. Vucetic, "A low complexity soft output receiver for cellular mobile applications", Technique Report,
- [16] S. Benedetto, G. Montorsi, D. Divsalar and F. Pollara, "A soft-input soft-output maximum a posteriori (MAP) module to decode parallel and serial concatenated codes," JPL TDA Progress Report 42-127, Nov. 15, 1996. Sep. 1995.
- [17] M. Reed and J. Asenstorfer, "A novel variance estimator for turbo code decoding," in Proc. Int. Conf. on Telecomm., Melbourne, Australia, Apr. 1997.

Chapter 6

Iterative Decoding

Turbo and serial concatenated codes can be decoded by MAP or ML decoding methods based on the overall code trellis presented in the previous chapter. These decoders could be implemented for small interleavers only as they are too complex for medium and large interleaver sizes. The practical importance of turbo and serial concatenated codes lies in the availability of a simple suboptimum decoding algorithm [1]. In this chapter we present a heuristic explanation of iterative decoding algorithms for turbo and serial concatenated codes. There are no proofs of convergence of the iterative decoding methods to the optimum MAP or ML decoding. We present the performance results of the iterative decoding relative to the ML performance bounds and some heuristic evidence that the suboptimum iterative methods can come very close to the optimum algorithms.

6.1 Optimum Decoding of Turbo Codes

Consider a turbo encoder consisting of two component RSC encoders of code rate 1/2, so that the overall code rate is 1/3 (Fig. 6.1). As before, we assume that the information sequence \mathbf{c} is a binary vector with components being independent identically distributed random variables with common distribution

$$P_r\{c_t = 1\} = P_r\{c_t = 0\} = 1/2 \quad (6.1)$$

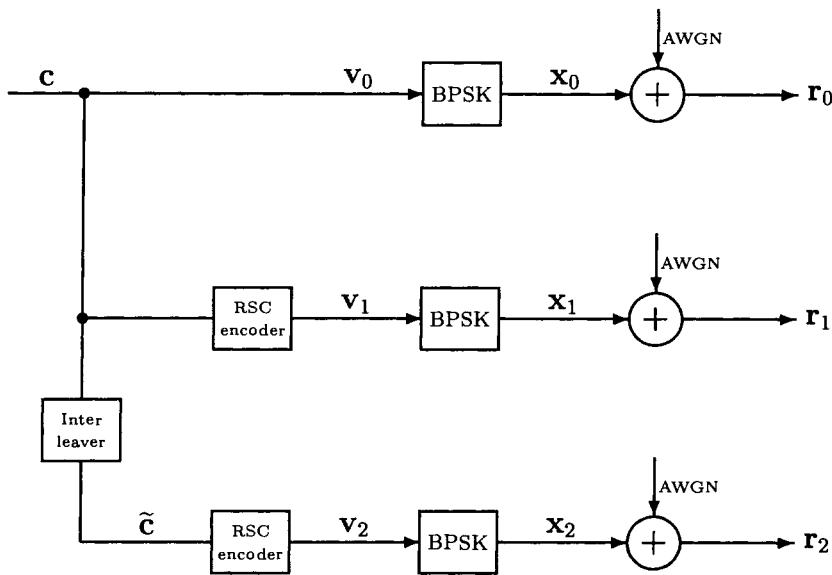


Fig. 6.1: Basic turbo encoder

The received sequences \mathbf{r}_0 and \mathbf{r}_1 form the input to the first decoder, denoted by \mathbf{r}'

$$\mathbf{r}' = \{\dots(r_{t,0}, r_{t,1}), (r_{t+1,0}, r_{t+1,1}), \dots\} \quad (6.2)$$

The deinterleaved received information sequence $\tilde{\mathbf{r}}_0$ and sequence \mathbf{r}_2 form the input to the second decoder, denoted by \mathbf{r}''

$$\mathbf{r}'' = \{\dots(\tilde{r}_{t,0}, r_{t,2}), (\tilde{r}_{t+1,0}, r_{t+1,2}), \dots\} \quad (6.3)$$

The optimum bit decision algorithm based on the observations \mathbf{r}' and \mathbf{r}'' computes the log-likelihood ratio $\Lambda(c_t)$ from the overall turbo code trellis as

$$\begin{aligned} \Lambda(c_t) &= \log \frac{P_r(c_t = 1 | \mathbf{r}', \mathbf{r}'')}{P_r(c_t = 0 | \mathbf{r}', \mathbf{r}'')} \\ &= \log \frac{\sum_{\mathbf{c}: c_t=1} P_r(\mathbf{r}', \mathbf{r}'' | \mathbf{c}) P_r(\mathbf{c})}{\sum_{\mathbf{c}: c_t=0} P_r(\mathbf{r}', \mathbf{r}'' | \mathbf{c}) P_r(\mathbf{c})} \end{aligned} \quad (6.4)$$

The decision rule is then

$$c_t = \begin{cases} 1 & \Lambda(c_t) \geq 0 \\ 0 & \Lambda(c_t) < 0 \end{cases} \quad (6.5)$$

If the random variables \mathbf{r}' and \mathbf{r}'' are uncorrelated we can assume

$$P_r(\mathbf{r}', \mathbf{r}'' | \mathbf{c}) = P_r(\mathbf{r}' | \mathbf{c}) P_r(\mathbf{r}'' | \mathbf{c}) \quad (6.6)$$

The log-likelihood ratio $\Lambda(c_t)$ then becomes

$$\Lambda(c_t) = \log \frac{\sum_{\mathbf{c}: c_t=1} P_r(\mathbf{r}' | \mathbf{c}) P_r(\mathbf{r}'' | \mathbf{c}) P_r(\mathbf{c})}{\sum_{\mathbf{c}: c_t=0} P_r(\mathbf{r}' | \mathbf{c}) P_r(\mathbf{r}'' | \mathbf{c}) P_r(\mathbf{c})} \quad (6.7)$$

Calculating the log-likelihood ratio $\Lambda(c_t)$ from (6.7) requires to evaluate the sums over all possible transmitted sequences. Therefore, if we apply the MAP algorithm presented in the previous chapter, its complexity will grow linearly with the number of states of the code trellis and the information sequence length. However, the overall trellis for a turbo code is time-varying and has a number of states that grows exponentially with the interleaver length. MAP decoding is thus possible for very short interleavers only.

6.2 Iterative Decoding of Turbo Codes Based on the MAP Algorithm

The iterative turbo decoding [1] consists of two component decoders serially concatenated via an interleaver, identical to the one in the encoder, as shown in Fig. 6.2.

The first MAP decoder takes as input the received information sequence \mathbf{r}_0 and the received parity sequence generated by the first encoder \mathbf{r}_1 . The decoder then produces a soft output, which is interleaved and used to produce an improved estimate of the a priori probabilities of the information sequence for the second decoder.

The other two inputs to the second MAP decoder are the interleaved received information sequence $\tilde{\mathbf{r}}_0$ and the received parity sequence produced by the second encoder \mathbf{r}_2 . The second MAP decoder also produces a soft output which is used to improve the estimate of the a priori probabilities for the information sequence at the input of the first MAP decoder. The decoder performance can be improved by this iterative operation relative to a single operation

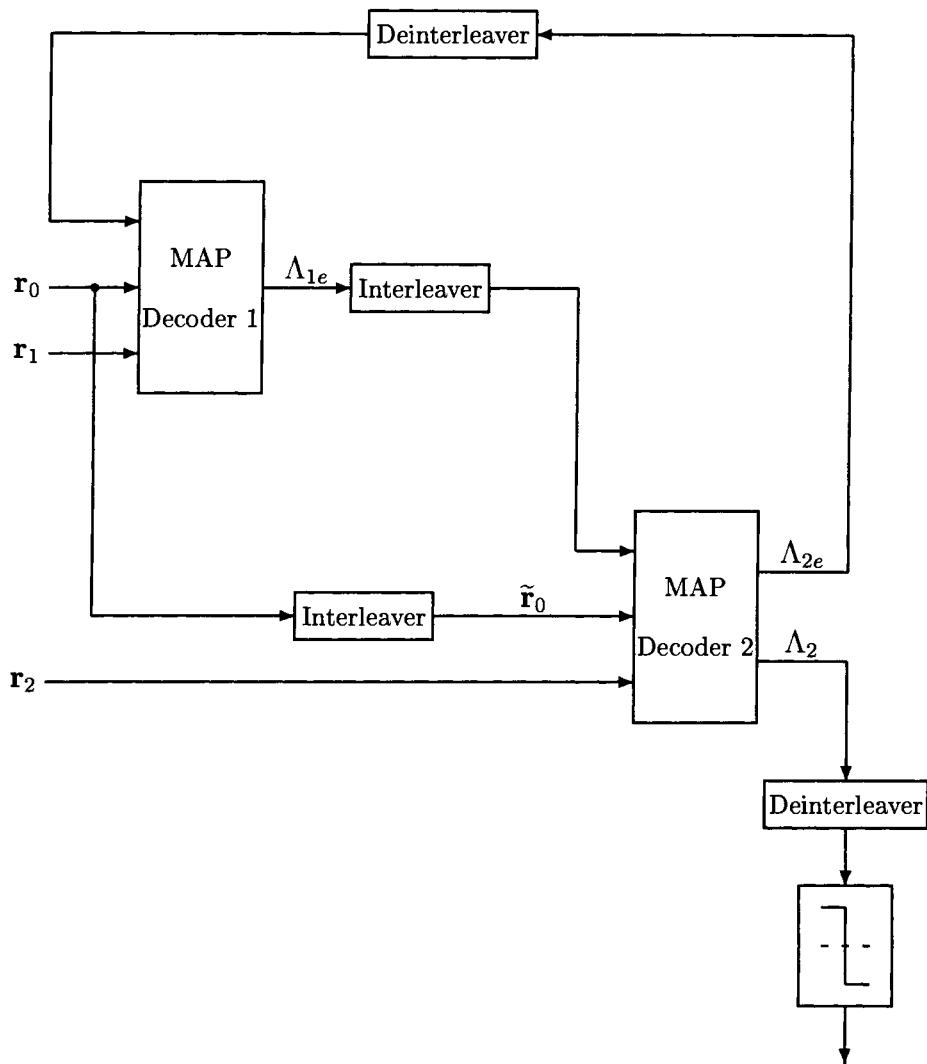


Fig. 6.2: An iterative turbo code decoder based on the MAP algorithm

serial concatenated decoder. The feedback loop is a distinguishing feature of this decoder and the name turbo code is given with reference to the principle of the turbo engine.

After a certain number of iterations the soft outputs of both

MAP decoders stop to produce further performance improvements. Then the last stage of decoding makes a hard decision after deinterleaving.

Note that for simplicity in Fig. 6.2 the delays introduced by two component decoders are not taken into account.

Let us examine the first MAP decoder in the first iteration for rate $\frac{1}{n}$ component codes. The log-likelihood ratio of the first MAP decoder is given by

$$\Lambda_1(c_t) = \log \frac{\sum_{l',l=0}^{M_s-1} \alpha_{t-1}(l') p_t^1(1) \exp\left(-\frac{\sum_{j=0}^{n-1} (r_{t,j} - x_{t,j}^1(l))^2}{2\sigma^2}\right) \cdot \beta_t(l)}{\sum_{l',l=0}^{M_s-1} \alpha_{t-1}(l') p_t^1(0) \exp\left(-\frac{\sum_{j=0}^{n-1} (r_{t,j} - x_{t,j}^0(l))^2}{2\sigma^2}\right) \cdot \beta_t(l)} \quad (6.8)$$

where we introduce the notation $p_t^1(1)$ and $p_t^1(0)$ for the a priori probabilities for 1 and 0 at the input of the first decoder, respectively, since they will differ from the corresponding a priori probabilities at the input of the second decoder denoted by $p_t^2(1)$ and $p_t^2(0)$.

In the initial decoding operation in the first decoder we assume that

$$p_t^1(1) = p_t^1(0) = 1/2 \quad (6.9)$$

Rewrite $\Lambda(c_t)$ as

$$\begin{aligned} \Lambda_1(c_t) &= \log \frac{p_t^1(1)}{p_t^1(0)} \\ &+ \log \frac{\sum_{l',l=0}^{M_s-1} \alpha_{t-1}(l') \exp\left(-\frac{(r_{t,0} - x_{t,0}^1)^2 + \sum_{j=1}^{n-1} (r_{t,j} - x_{t,j}^1(l))^2}{2\sigma^2}\right) \cdot \beta_t(l)}{\sum_{l',l=0}^{M_s-1} \alpha_{t-1}(l') \exp\left(-\frac{(r_{t,0} - x_{t,0}^0)^2 + \sum_{j=1}^{n-1} (r_{t,j} - x_{t,j}^0(l))^2}{2\sigma^2}\right) \cdot \beta_t(l)} \end{aligned}$$

where $(n - 1)$ is the number of parity digits in the encoded block.

Since the code is systematic $x_{t,0}^i$, $i = 0, 1$, are independent of the trellis and state l . That is $x_{t,0}^1 = 1$ and $x_{t,0}^0 = -1$. $\Lambda_1(c_t)$ could be further decomposed into

$$\Lambda_1(c_t) = \log \frac{p_t^1(1)}{p_t^1(0)} + \frac{2}{\sigma^2} r_{t,0} + \Lambda_{1e}(c_t) \quad (6.10)$$

where

$$\Lambda_{1e}(c_t) = \log \frac{\sum_{l'=0}^{M_s-1} \alpha_{t-1}(l') \exp \left(-\frac{\sum_{j=1}^{n-1} (r_{t,j} - x_{t,j}^1(l))^2}{2\sigma^2} \right) \cdot \beta_t(l)}{\sum_{l'=1}^{M_s-1} \alpha_{t-1}(l') \exp \left(-\frac{\sum_{j=0}^{n-1} (r_{t,j} - x_{t,j}^0(l))^2}{2\sigma^2} \right) \cdot \beta_t(l)} \quad (6.11)$$

$\Lambda_{1e}(c_t)$ is called the *extrinsic information*. It is a function of the redundant information introduced by the encoder. It does not contain the information decoder input $r_{t,0}$. This quantity may be used to improve the a priori probability estimate for the next decoding stage.

Let us observe the input to the second MAP decoder. Since the input to the second decoder includes the interleaved version of \mathbf{r}_0 , the received information signal $\tilde{r}_{t,0}$ correlates with the interleaved soft output from the first decoder, $\tilde{\Lambda}_1(c_t)$. Therefore, the contribution due to $r_{t,0}$ must be taken out from $\Lambda_1(c_t)$, to eliminate this correlation.

However, $\Lambda_{1e}(c_t)$ does not contain $r_{t,0}$ and it can be used as the a priori probability for decoding in the second stage. That is, the interleaved extrinsic information of the first decoder $\tilde{\Lambda}_{1e}$ is the a priori probability estimate for the second decoder

$$\tilde{\Lambda}_{1e}(c_t) = \log \frac{p_t^2(1)}{p_t^2(0)} \quad (6.12)$$

From Eq. (6.12) and the relationship

$$p_t^2(1) = 1 - p_t^2(0) \quad (6.13)$$

we can write for the a priori probabilities in the second decoder

$$p_t^2(1) = \frac{e^{\tilde{\Lambda}_{1e}(c_t)}}{1 + e^{\tilde{\Lambda}_{1e}(c_t)}} \quad (6.14)$$

and

$$p_t^2(0) = \frac{1}{1 + e^{\tilde{\Lambda}_{1e}(c_t)}} \quad (6.15)$$

In the second decoding stage the MAP decoder estimates the log-likelihood ratio $\Lambda_2(c_t)$. Similarly to (6.10) the log-likelihood ratio for the second MAP decoder can be decomposed into

$$\Lambda_2(c_t) = \log \frac{p_t^2(1)}{p_t^2(0)} + \frac{2}{\sigma^2} \tilde{r}_{t,0} + \Lambda_{2e}(c_t) \quad (6.16)$$

By substituting the a priori probabilities from Eq. (6.12) in Eq. (6.16), we get

$$\Lambda_2(c_t) = \tilde{\Lambda}_{1e}(c_t) + \frac{2}{\sigma^2} \tilde{r}_{t,0} + \Lambda_{2e}(c_t) \quad (6.17)$$

$\Lambda_{2e}(c_t)$ is the extrinsic information for the second decoder, which depends on the redundant information supplied by the second encoder, as in Eq. (6.11). The second decoder extrinsic information can be used as the estimates of the a priori probabilities for the first decoder as in (6.12). The log-likelihood ratio for the first decoder can be written as

$$\Lambda_1(c_t) = \tilde{\Lambda}_{2e}(c_t) + \frac{2}{\sigma^2} r_{t,0} + \Lambda_{1e}(c_t) \quad (6.18)$$

Summary of the Iterative MAP Decoding Method

1. Initialize $\Lambda_{2e}^{(0)}(c_t) = 0$.
2. For iterations $r = 1, 2, \dots, I$, where I is the total number of iterations
 - Compute $\Lambda_1^{(r)}(c_t)$ and $\Lambda_2^{(r)}(c_t)$ by using Eq. (6.8).
 - Compute $\Lambda_{1e}^{(r)}(c_t)$ as

$$\Lambda_{1e}^{(r)}(c_t) = \Lambda_1^{(r)}(c_t) - \frac{2}{\sigma^2} r_{t,0} - \tilde{\Lambda}_{2e}^{(r-1)}(c_t) \quad (6.19)$$

- Compute $\Lambda_{2e}^{(r)}(c_t)$ as

$$\Lambda_{2e}^{(r)}(c_t) = \Lambda_2^{(r)}(c_t) - \frac{2}{\sigma^2} \tilde{r}_{t,0} - \tilde{\Lambda}_{1e}^{(r)}(c_t) \quad (6.20)$$

3. After I iterations make a hard decision on c_t based on $\tilde{\Lambda}_2^{(I)}(c_t)$.

6.3 The Effect of the Number of Iterations on Turbo Code Performance

Let us consider the performance of turbo codes with random interleavers when the number of iterations in the iterative decoder is variable. As an example, a rate 1/3, 16 state turbo code with the generator polynomials $\mathbf{g}_0 = (37)$ and $\mathbf{g}_1 = (21)$ and the interleaver sizes of 4096 and 16384 is simulated. The simulation results based on the iterative MAP decoding algorithm are shown in Figs. 6.3 and 6.4 for the interleaver sizes of 4096 and 16384, respectively.

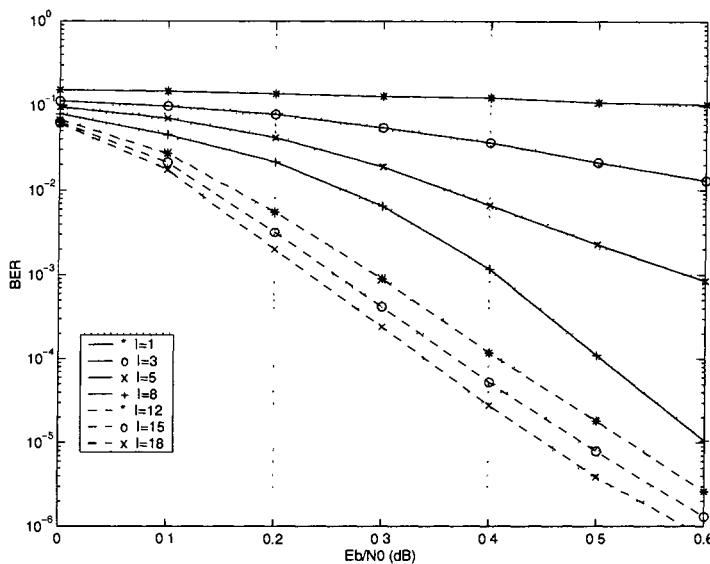


Fig. 6.3: BER performance of a 16 state, rate 1/3 turbo code with MAP algorithm on an AWGN channel, interleaver size 4096 bits, variable number of iterations.

Initially, increasing the number of iterations has a considerable effect on the performance. However, there are diminishing returns when the number of iterations becomes high. For interleaver size 4096, there is no significant improvement if the number of iterations increases above 12, while for interleaver size 16384 this threshold is about 15.

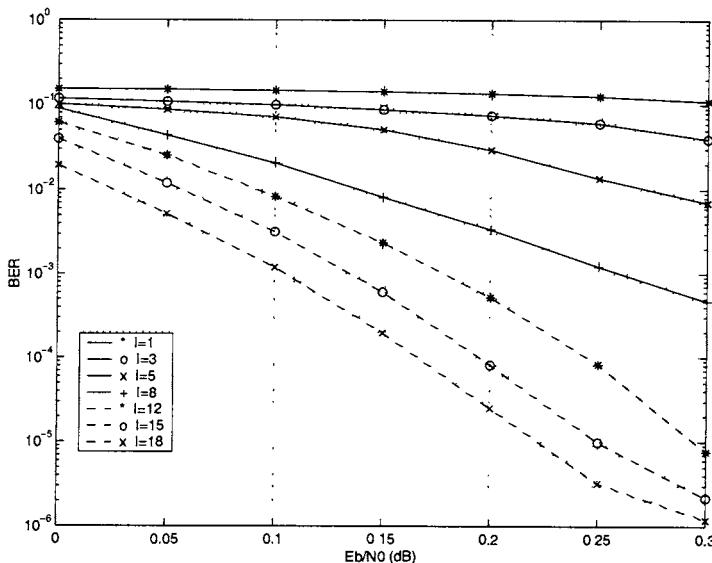


Fig. 6.4: BER performance of a 16 state, rate 1/3 turbo code with MAP algorithm on an AWGN channel, interleaver size 16384 bits, variable number of iterations

As the decoding approaches the performance limit of a given turbo code, any further iteration results in very little improvement. Therefore, it is important to devise an efficient criterion to stop the iteration process and prevent unnecessary computations and decoding delay. One such stopping criterion has been devised based on the cross entropy between the distributions of the estimates at the outputs of the decoders at each iteration [5] [15] [16]. This criterion is known as cross entropy criterion. It effectively stops the iteration process with very little performance degradation.

Another method of terminating the iterative decoding process is to use systematic cyclic redundancy checks (CRC). In the encoder, the input information is encoded by a CRC code, and the output of the CRC encoder is passed to the turbo encoder. In the receiver, the turbo decoder output at each iteration is fed to the CRC error detector. If the CRC code detects no errors in the decoder output, the iterative decoding is terminated [17].

6.4 The Effect of Interleaver Size on Turbo Code Performance

In the original turbo code a pseudo-random block interleaver is used where information is written row by row and read out following a non-uniform rule, based on randomly generated numbers. The interleaver length is critical for the code performance, particularly at low SNR's. The interleaver structure is important for the performance at high SNR's, as it affects the distance properties of the overall turbo code. It determines the code free distance which has a dominant effect on the asymptotic performance.

As the interleaver enables the information exchange between the two component decoders, increasing the interleaver size has the effect of randomizing the information sequence at the input of the second decoder. Consequently, the inputs to the two component decoders become less correlated, with respect to noise, improving the decoding performance.

For feedback component codes interleaving modifies the code weight distribution relative to the uninterleaved input. This is most significant for weight 2 information sequence which has the dominant effect on the performance. If an input weight 2 sequence to the first encoder generates a low weight path in the first code trellis, the interleaved sequence at the input of the second encoder will produce a higher weight path, thus improving the code performance.

The simulation results for a rate 1/3, 16-state turbo code with generator polynomials $g_0 = (37)$ and $g_1 = (21)$ and a pseudo-random interleaver of various lengths on an AWGN channel are shown in Fig. 6.5. The number of iterations for the interleaver sizes above 2048 was 18. For the interleaver sizes 420 to 2048 it was 8, as there is no improvement in performance for these interleavers if the number of iterations is increased above 8.

Clearly, the performance is consistently improved by increasing the interleaver size. This is consistent with the expression for the bit error probability, shown in Chapter 4, Eq. (4.38), where the bit error probability is inversely proportional to the interleaver size.

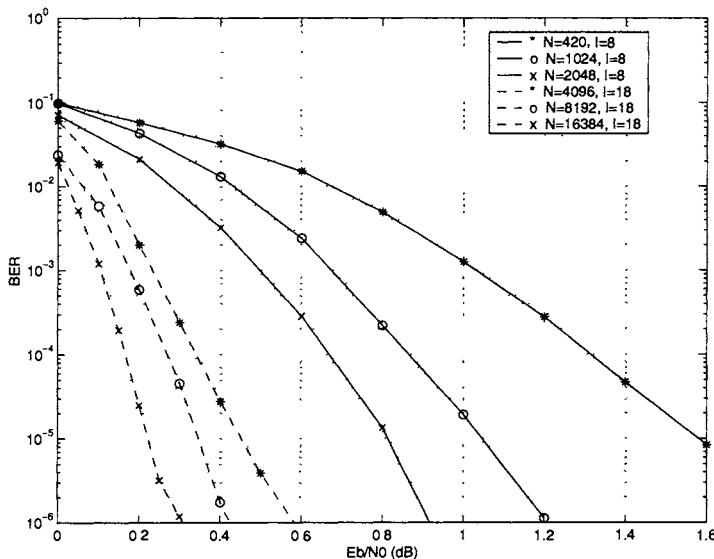


Fig. 6.5: BER performance of a 16 state, rate 1/3 turbo code with MAP algorithm on an AWGN channel, interleaver size N , the number of iterations 18

6.5 The Effect of Puncturing Component Codes on Turbo Code Performance

One way of increasing the rate of a turbo code is by puncturing the encoder outputs of the basic 1/3 rate turbo code. In this way it is possible to design rate 1/2, 2/3, 3/4, 5/6 etc. codes.

The performance results for the 16-state, rate 1/2 punctured turbo code, for the number of iterations of 18 and various interleaver sizes are shown in Fig. 6.6. They are obtained by simulation with an iterative MAP decoding method on an AWGN channel. The puncturing matrix for this code is

$$\mathbf{P} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

The code is obtained from the 1/3 rate turbo code with the generator polynomials $\mathbf{g}_0 = (37)$ and $\mathbf{g}_1 = (21)$. The performance loss relative to the 1/3 rate code for interleaver size 4096 is about 0.70 dB at the bit error rate of 10^{-4} .

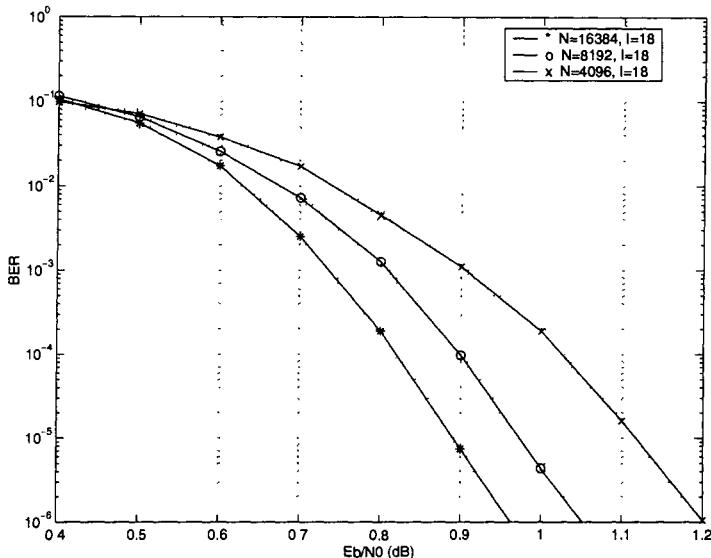


Fig. 6.6: BER performance of a 16 state, rate 1/2 turbo code with MAP algorithm on an AWGN channel, interleaver size N , the number of iterations 18

The simulation results for a rate 2/3, 16-state turbo code, obtained by puncturing the rate 1/2 turbo code in the previous example are shown in Fig. 6.7. The puncturing matrix is

$$\mathbf{P} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

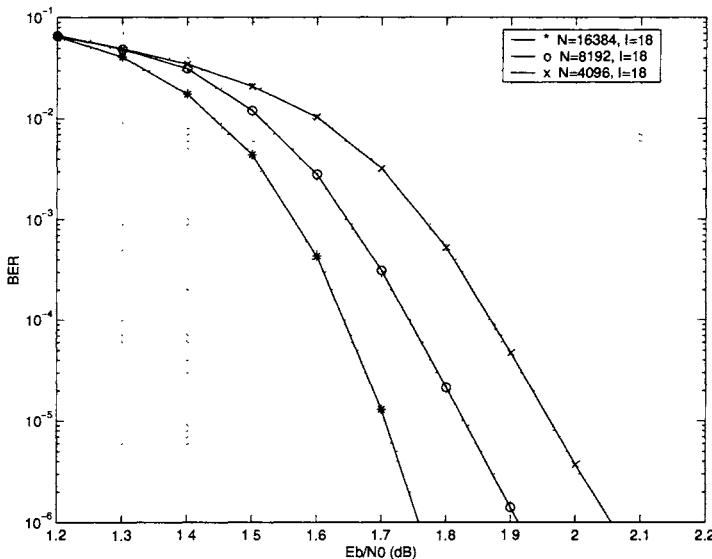


Fig. 6.7: BER performance of a 16 state, rate $2/3$ turbo code with MAP algorithm on an AWGN channel, interleaver size N , the number of iterations 18

6.6 Comparison Between Analytical Upper Bounds and Iterative Decoding Simulation Results

The upper bound (4.38) derived in Chapter 4 is tight for the bit error probabilities lower than 10^{-4} . This bound is a good estimate of the turbo codes optimum decoding and as such can be used to determine how far are the results of the iterative decoding method from the optimum one. The comparison is shown in Fig. 6.8. They are obtained for interleaver size 1024.

For the iterative decoding a random interleaver with no constraints is chosen and the system is simulated on an AWGN channel, with MAP component decoders and various numbers of iterations.

Clearly, the results show the convergence of the iterative decoding curves to the optimum decoding bound for increasing numbers

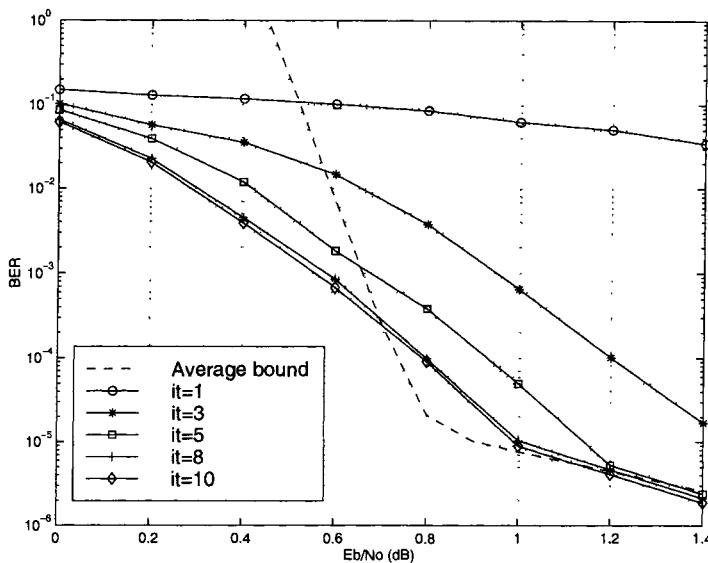


Fig. 6.8: Simulation result of a 16 state, rate 1/3 turbo code with MAP, interleaver size 1024 bits, variable number of iterations I and the theoretical bound on an AWGN channel.

of iterations.

Approaching the optimum performance requires a higher number of iterations for low SNR's. For example, 5 iterations are needed at the BER of 10^{-5} , while at BER 10^{-3} at least 8 iterations are required.

It is worth noting that, for particular interleaver designs, it is possible to achieve better results than for random and uniform interleavers, as discussed in Chapter 7.

6.7 Asymptotic Behavior of Turbo Codes

At high SNR's the bit error probability of turbo codes is dominated by the code free distance and it can be expressed as

$$P_b \cong B_{\text{free}} Q \left(\sqrt{2d_{\text{free}} \cdot R \frac{E_b}{N_0}} \right) \quad (6.21)$$

where B_{free} is the average number of ones on the minimum free distance path in the overall turbo code trellis, R is the code rate, E_b is the received bit energy, N_0 is the Gaussian noise one sided power spectral density, and d_{free} is the code minimum free distance. Eq. (6.21) defines the asymptotic behavior of turbo codes.

The value of d_{free} depends on the generator polynomials and the interleaver structure. For a uniform interleaver and the 16-state rate 1/3 code with the generator polynomials $\mathbf{g}_0 = (37)$ and $\mathbf{g}_1 = (21)$ it is 9. As this value of d_{free} is relatively low, the curve defined by Eq. (6.21) has a very low slope at high SNR's, which is often referred to as the "error floor".

The simulation results for this code with iterative MAP decoding on an AWGN channel, interleaver size 1024 and 10 iterations along the theoretical bound in Eq. (6.21) are shown in Fig. 6.9. A random interleaver is chosen in the simulation. It is clear that the bound in (6.21) closely determines the code performance at high SNR's.

6.8 Iterative SOVA Decoding of Turbo Codes

Let us consider a turbo encoder with rate 1/2 component RSC binary codes shown in Fig. 6.1. For iterative decoding of turbo codes it is possible to use the SOVA algorithm for component decoders [13] [14]. The block diagram of the iterative SOVA decoder is shown in Fig. 6.10.

The first SOVA decoder generates the soft output and subsequently the extrinsic information as in the iterative MAP decoder. The extrinsic information is interleaved and used by the second SOVA decoder as the estimate of the a priori probability. The second SOVA decoder also produces the extrinsic information and

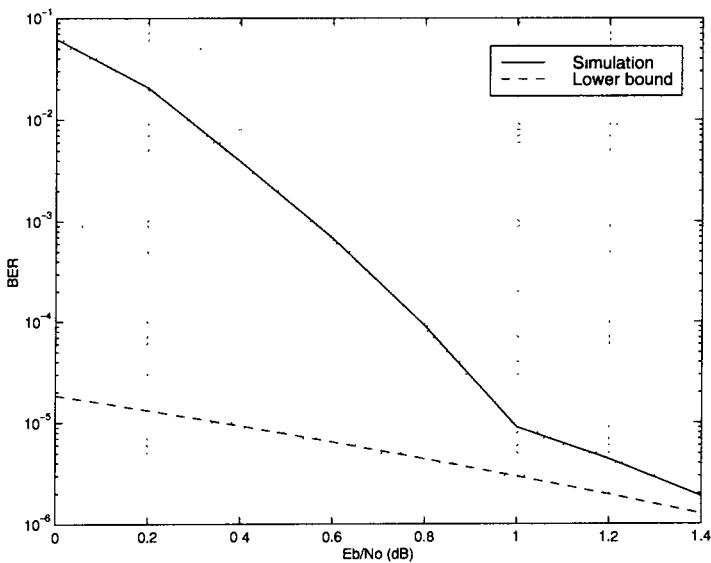


Fig. 6.9: Simulation result of a 16 state, rate 1/3 turbo code with MAP, interleaver size 1024 bits, the number of iterations 10 and the theoretical bound on an AWGN channel.

passes it after deinterleaving to the first SOVA decoder to be used in the next decoding operation.

The expression for the path metric of the SOVA was derived by maximizing the a posteriori probability, $P_r\{\mathbf{c}, \mathbf{r}\}$

$$P_r\{\mathbf{c}, \mathbf{r}\} = P_r\{\mathbf{c}\} \cdot P_r\{\mathbf{r} | \mathbf{c}\} \quad (6.22)$$

or its logarithm

$$\log P_r\{\mathbf{c}, \mathbf{r}\} = \log P_r\{\mathbf{c}\} + \log P_r\{\mathbf{r} | \mathbf{c}\} \quad (6.23)$$

By assuming that the channel is subject to independent Gaussian noise, we get for the logarithm of $P_r\{\mathbf{c}, \mathbf{r}\}$

$$\begin{aligned} \log P_r\{\mathbf{c}, \mathbf{r}\} &= \sum_{t=1}^{\tau} \log p_t(c_t) - \frac{n\tau}{2} \log(2\pi) \\ &\quad - n\tau \log \sigma - \sum_{t=1}^{\tau} \sum_{i=0}^{n-1} \frac{(r_{t,i} - x_{t,i})^2}{2\sigma^2} \end{aligned} \quad (6.24)$$

where the notation is the same as in the previous chapter. Assuming that the a priori probabilities $p_t(c_t)$ might change with the iteration number in an iterative decoding procedure, maximizing $\log P_r(\mathbf{c}, \mathbf{r})$ is equivalent to maximizing

$$\sum_{t=1}^{\tau} \log p_t(c_t) - \sum_{t=1}^{\tau} \sum_{i=0}^{n-1} (r_{t,i} - x_{t,i})^2 \quad (6.25)$$

Consequently, we define the branch metric, assigned to a trellis branch at time t , as

$$\nu_t^{c_t} = \sum_{i=0}^{n-1} (r_{t,i} - x_{t,i})^2 - \log p_t(c_t) \quad (6.26)$$

Similarly, we define the path metric, for a path \mathbf{x} in the trellis, as

$$\mu_t^{\mathbf{x}} = \sum_{t'=1}^t \nu_{t'}^{c_{t'}} = \mu_{t-1}^{\mathbf{x}} + \nu_t^{c_t} \quad (6.27)$$

The SOVA decoder selects a path with the minimum path metric. A SOVA decoder in the iterative scheme shown in Fig. 6.10 computes the soft output as

$$\Lambda(c_t) = \mu_t^0 - \mu_t^1 = (-1)^{c_t} (\mu_{\tau,\min} - \mu_{t,c}) \quad (6.28)$$

where μ_t^0 and μ_t^1 are the minimum path metrics with the symbol 0 and symbol 1, respectively, at time t , $\mu_{\tau,\min}$ is the maximum likelihood path metric and $\mu_{t,c}$ is the best competitor path metric, as described in Section 5.5. They can be computed by the following procedure.

At time t , the maximum likelihood estimate c_t is obtained from the maximum likelihood path μ_{t,c_t} , computed by using the Viterbi algorithm, as

$$\begin{aligned} \mu_{t,c_t} &= \mu_{\tau,\min} \\ &= \sum_{t'=0}^{t-1} \nu_{t'} + \nu_t^{c_t} + \sum_{t'=t+1}^{\tau} \nu_{t'} \end{aligned} \quad (6.29)$$

$$= \mu_t' + \nu_t^{c_t} \quad (6.30)$$

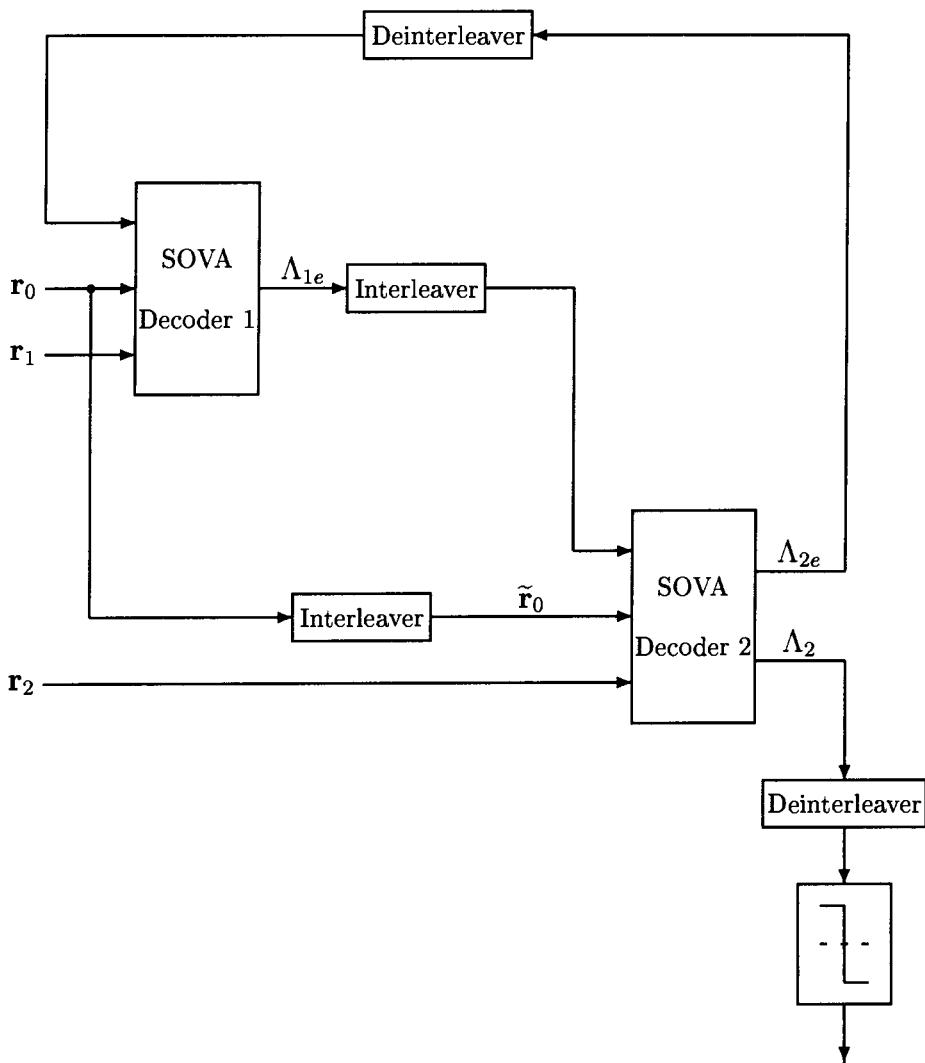


Fig. 6.10: An iterative turbo code decoder based on the SOVA algorithm

where

$$\mu'_t = \sum_{t'=0}^{t-1} \nu_{t'} + \sum_{t'=t+1}^{\tau} \nu_{t'} \quad (6.31)$$

and $\nu_t^{c_t}$ is given by

$$\nu_t^{c_t} = \sum_{i=0}^{n-1} (r_{t,i} - x_{t,i}^{c_t})^2 - \log p_t(c_t) \quad (6.32)$$

where $x_{t,i}^{c_t}$ is the i th modulated coded signal in the trellis, generated by input c_t .

Next, the path metric of the best competitor $\mu_{t,c}$, where $c = c_t \oplus 1$, can be computed as

$$\mu_{t,c} = \min \left\{ \mu_{t-1}^f(l') + \nu_t^c(l', l) + \mu_t^b(l) \right\} \quad (6.33)$$

where $l', l = 0, 1, \dots, M_s - 1$, $\mu_{t-1}^f(l')$ is the path metric of the forward survivor at time $t - 1$ and node l' , $\mu_t^b(l)$ is the path metric of the backward survivor at time t , M_s is the number of nodes in the trellis and $\nu_t^c(l', l)$ is the branch metric of the complement branch from node l' to l . $\mu_{t,c}$ can be represented as

$$\mu_{t,c} = \mu_t'' + \nu_t^c \quad (6.34)$$

where μ_t'' is a positive number, similar to (6.31), and ν_t^c is given by

$$\nu_t^c = \sum_{i=0}^{n-1} (r_{t,i} - x_{t,i}^c)^2 - \log p_t(c) \quad (6.35)$$

where $x_{t,i}^c$ is the i th modulated coded signal in the trellis, generated by input c .

By substituting the path metrics from Eqs. (6.30) and (6.34) into Eq. (6.28), we can compute the soft output for the first decoder, $\Lambda_1(c_t)$, as

$$\Lambda_1(c_t) = (-1)^{c_t} \left\{ (\mu_{t,1}' + \nu_{t,1}^{c_t}) - (\mu_{t,1}'' + \nu_{t,1}^c) \right\} \quad (6.36)$$

where $\mu_{t,1}', \nu_{t,1}^{c_t}, \mu_{t,1}''$ and $\nu_{t,1}^c$ refer to the quantities $\mu_t', \nu_t^{c_t}, \mu_t''$ and ν_t^c , for the first decoder, respectively.

By substituting $\nu_t^{c_t}$ and ν_t^c from Eqs. (6.32) and (6.35), respectively, into (6.36) and observing that $x_{t,0}^0 = -1$ and $x_{t,0}^1 = 1$, we obtain for $\Lambda_1(c_t)$

$$\Lambda_1(c_t) = \log \frac{p_t^1(1)}{p_t^1(0)} + 4r_{t,0} + \Lambda_{1e}(c_t) \quad (6.37)$$

where $\Lambda_{1e}(c_t)$ is the extrinsic information for the first decoder given by

$$\Lambda_{1e}(c_t) = (-1)^{c_t} \left\{ \mu'_{t,1} - \mu''_{t,1} - 2 \sum_{i=1}^{n-1} (x_{t,i}^{c_t} - x_{t,i}^c) \right\} \quad (6.38)$$

and $p_t^1(1)$ and $p_t^1(0)$ are the a priori probabilities for the binary 1 and 0, respectively, at the input of the first decoder.

The interleaved extrinsic information of the first decoder improves the a priori probabilities at the input of the second SOVA decoder, as in the MAP iterative decoder

$$\tilde{\Lambda}_{1e}(c_t) = \log \frac{p_t^2(1)}{p_t^2(0)} \quad (6.39)$$

where $p_t^2(1)$ and $p_t^2(0)$ are a priori probabilities for the binary 1 and 0 at the input of the second decoder, at time t , respectively, which can be estimated by Eqs. (6.14) and (6.15).

The second decoder computes the soft output $\Lambda_2(c_t)$ which can be written as

$$\Lambda_2(c_t) = \log \frac{p_t^2(1)}{p_t^2(0)} + 4\tilde{r}_{t,0} + \Lambda_{2e}(c_t) \quad (6.40)$$

where $\Lambda_{2e}(c_t)$ is the extrinsic information from the first decoder, given

$$\Lambda_{2e}(c_t) = (-1)^{c_t} \left\{ \mu'_{t,2} - \mu''_{t,2} - 2 \sum_{i=1}^{n-1} (\tilde{x}_{t,i}^{c_t} - \tilde{x}_{t,i}^c) \right\} \quad (6.41)$$

where $\mu'_{t,2}$ and $\mu''_{t,2}$ refer to the quantities μ'_t and μ''_t , for the second decoder, respectively.

Substituting the extrinsic information from the first decoder, $\tilde{\Lambda}_{1e}(c_t)$ from Eq. (6.39) into Eq. (6.40) we get for $\Lambda_2(c_t)$

$$\Lambda_2(c_t) = \tilde{\Lambda}_{1e}(c_t) + 4\tilde{r}_{t,0} + \Lambda_{2e}(c_t) \quad (6.42)$$

The interleaved extrinsic information from the second decoder, $\tilde{\Lambda}_{2e}(c_t)$, is used as the estimate of the logarithm of the encoder input probabilities in the next iteration

$$\tilde{\Lambda}_{2e}(c_t) = \log \frac{p_t^1(1)}{p_t^1(0)} \quad (6.43)$$

so that the log-likelihood ratio $\Lambda_1(c_t)$ in the next iteration can be written as

$$\Lambda_1(c_t) = \tilde{\Lambda}_{2e}(c_t) + 4r_{t,0} + \Lambda_{1e}(c_t) \quad (6.44)$$

Summary of the Iterative SOVA Decoding Method

1. Initialize $\Lambda_{2e}^{(0)}(c_t) = 0$.
2. For iterations $r = 1, 2, \dots, I$,
 - Compute $\Lambda_1^{(r)}(c_t)$ and $\Lambda_2^{(r)}(c_t)$ by using Eq. (6.28).
 - Compute $\Lambda_{1e}^{(r)}(c_t)$ as

$$\Lambda_{1e}^{(r)}(c_t) = \Lambda_1^{(r)}(c_t) - 4r_{t,0} - \tilde{\Lambda}_{2e}^{(r-1)}(c_t) \quad (6.45)$$

- Compute $\Lambda_{2e}^{(r)}(c_t)$ as

$$\Lambda_{2e}^{(r)}(c_t) = \Lambda_2^{(r)}(c_t) - 4\tilde{r}_{t,0} - \tilde{\Lambda}_{1e}^{(r)}(c_t) \quad (6.46)$$

3. After I iterations, make a hard decision on c_t based on $\Lambda_2(c_t)$.

6.9 Comparison of MAP and SOVA Iterative Decoding Algorithms

The bit error probabilities for the rate 1/3, 16-state turbo code decoded by the iterative MAP, Log-MAP and SOVA methods are shown in Fig. 6.11, for interleaver size 4096 and 18 iterations. The SOVA has a worse performance of 0.55 dB at the BER of 10^{-5} .

The complexity of the iterative SOVA is lower by the same factor compared to the iterative MAP as the complexity of the SOVA relative to the MAP, and these two algorithms are compared in Section 5.10.

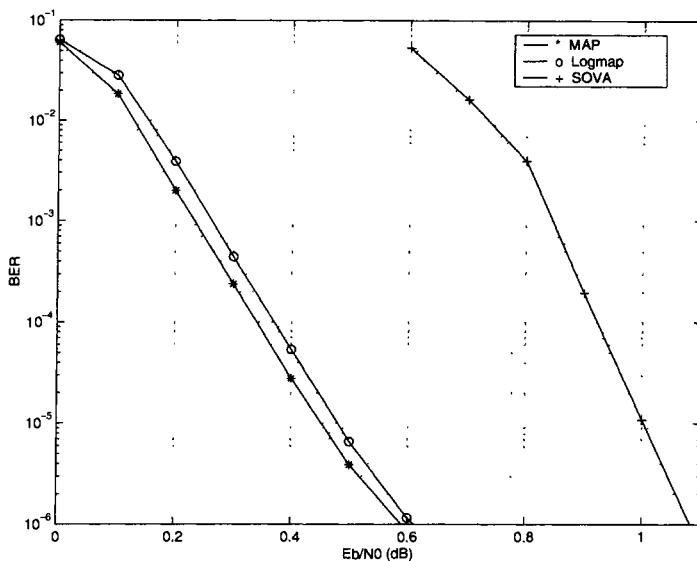


Fig. 6.11: BER performance of a 16 state, rate 1/3 turbo code with MAP, Log-MAP and SOVA algorithm on an AWGN channel, interleaver size 4096 bits, the number of iterations 18

Component decoders can be implemented also as Log-MAP or Max-Log-MAP decoders, with the iterative procedure similar to the iterative MAP.

As discussed in Section 5.9, the performance of the Log-MAP is almost identical to the MAP, while the performance of Max-Log-MAP is identical to the performance of the SOVA.

6.10 Iterative MAP Decoding of Serial Concatenated Convolutional Codes

The block diagram of an iterative MAP decoder for serial concatenated codes is shown in Fig. 6.12.

Following the design rules, as described in Chapter 4, we assume that the inner code is an (n_i, k_i) RSC code and that the outer code

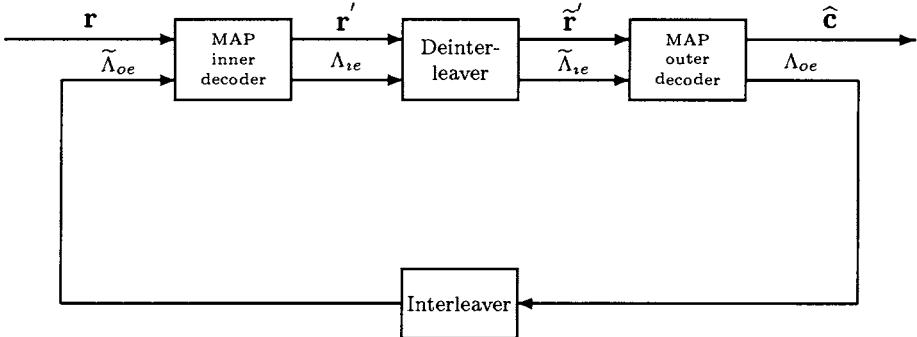


Fig. 6.12: Iterative MAP decoder for serial concatenated codes

is in general an (n_o, k_o) nonsystematic feedforward convolutional code.

In the first iteration the MAP inner decoder computes the log-likelihood ratio $\Lambda_i(c_{t,j})$, assuming that the binary message components are equally likely, in the form

$$\Lambda_i(c_{t,j}) = \tilde{\Lambda}_{oe}(c_{t,j}) + \frac{2}{\sigma^2} r_{t,j} + \Lambda_{ie}(c_{t,j}) \quad (6.47)$$

where $j = 0, 1, \dots, k_i - 1$, $\tilde{\Lambda}_{oe}(c_{t,j})$ is the interleaved extrinsic information of output bits conditioned on the constraint of the outer decoder and $\Lambda_{ie}(c_{t,j})$ is the extrinsic information for the inner decoder. $\tilde{\Lambda}_{oe}(c_{t,j})$ is assumed to be zero in the first iteration.

The extrinsic information and decoded soft output information sequence from the first decoder are deinterleaved and passed to the second decoder.

The deinterleaved information sequence of the inner code denoted by r' , is the received coded sequence for the outer code. The extrinsic information sequence from the inner decoder is the deinterleaved logarithm of the a priori probability ratio of the input to the outer decoder. The log-likelihood ratio of the outer decoder can be written as

$$\Lambda_o(c_{t,j}) = \tilde{\Lambda}_{ie}(c_{t,j}) + \Lambda_{oe}(c_{t,j}) \quad (6.48)$$

where $j = 0, 1, \dots, n_o - 1$. Note that as the outer code is non-systematic, it is not possible to extract the information bit at the

output of the outer decoder, so the soft output does not contain the contribution of the information bit.

The extrinsic information $\Lambda_{oe}(c_{t,j})$ is interleaved and passed to the first encoder as the a priori probability for the next iteration.

Summary of the Iterative MAP Algorithm for Serial Concatenated Codes

1. Initialize $\Lambda_{oe}(c_{t,j}) = 0$.
2. For iterations $r = 1, 2, \dots, I$,
 - Compute $\Lambda_i^{(r)}(c_{t,j})$ and $\Lambda_o^{(r)}(c_{t,j})$ by using Eqs. (6.47) and (6.48).
 - Compute $\Lambda_{ie}^{(r)}(c_{t,j})$ as

$$\Lambda_{ie}^{(r)} = \Lambda_i^{(r)}(c_{t,j}) - \frac{2}{\sigma^2} r_{t,j} - \tilde{\Lambda}_{oe}^{(r-1)}(c_{t,j}) \quad (6.49)$$

where $j = 0, 1, \dots, k_i - 1$.

- Compute $\Lambda_{oe}^{(r)}(c_{t,j})$ as

$$\Lambda_{oe}^{(r)}(c_{t,j}) = \Lambda_o^{(r)}(c_{t,j}) - \tilde{\Lambda}_{ie}^{(r)}(c_{t,j}) \quad (6.50)$$

where $j = 0, 1, \dots, n_o - 1$.

3. After I iterations, make a hard decision on $c_{t,j}$ based on $\Lambda_o^I(c_{t,j})$.

6.11 Iterative SOVA Decoding of Serial Concatenated Convolutional Codes

The block diagram of a SOVA iterative decoder for serial concatenated convolutional codes is shown in Fig. 6.13.

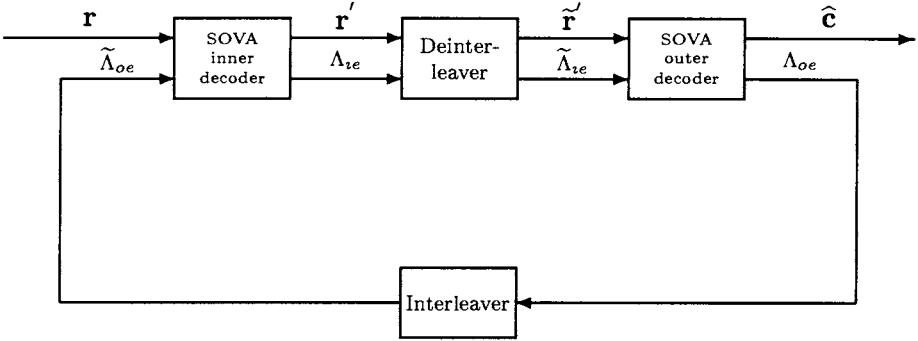


Fig. 6.13: Iterative SOVA decoder for serial concatenated codes

The soft output of the inner decoder can be expressed as

$$\Lambda_i(c_{t,j}) = \tilde{\Lambda}_{oe}(c_{t,j}) + 4r_{t,j} + \Lambda_{ie}(c_{t,j}), \quad j = 0, 1, \dots, k_i - 1 \quad (6.51)$$

where $\tilde{\Lambda}_{oe}(c_{t,j})$ is the interleaved extrinsic information from the outer decoder, which is assumed to be zero in the first iteration.

The received coded sequence for the outer code, r' , and the extrinsic information $\Lambda_{ie}(c_{t,j})$ are passed to the outer decoder. The soft output of the outer decoder can be written by

$$\Lambda_o(c_{t,j}) = \tilde{\Lambda}_{ie}(c_{t,j}) + \Lambda_{oe}(c_{t,j}), \quad j = 0, 1, \dots, n_o - 1 \quad (6.52)$$

Summary of the Iterative SOVA for Serial Concatenated Codes

1. Initialize $\tilde{\Lambda}_{oe}^{(0)}(c_{t,j}) = 0$.
2. For iterations $r = 1, 2, \dots, I$,
 - Compute $\Lambda_i^{(r)}(c_{t,j})$ and $\Lambda_o^{(r)}(c_{t,j})$ by using Eqs. (6.51) and (6.52).
 - Compute $\Lambda_{ie}^{(r)}(c_{t,j})$ as

$$\Lambda_{ie}^{(r)}(c_{t,j}) = \Lambda_i^{(r)}(c_{t,j}) - 4r_{t,j} - \tilde{\Lambda}_{oe}^{(r-1)}(c_{t,j}) \quad (6.53)$$

where $j = 0, 1, \dots, k_i - 1$.

- Compute $\Lambda_{oe}^{(r)}(c_{t,j})$ as

$$\Lambda_{oe}^{(r)}(c_{t,j}) = \Lambda_o^{(r)}(c_{t,j}) - \tilde{\Lambda}_{ie}^{(r)}(c_{t,j}) \quad (6.54)$$

where $j = 0, 1, \dots, n_o - 1$.

3. After I iterations, make a hard decision on $c_{t,j}$ based on $\Lambda_o^I(c_{t,j})$.

6.12 Serial Concatenated Convolutional Codes with Iterative Decoding

In this section we present the performance of serial concatenated codes with variable interleaver sizes, memory order and number of iterations for MAP and SOVA decoding.

We consider serial concatenated codes with two component codes, an inner and an outer code.

The inner code was selected to be a recursive convolutional code, which is for a given memory order designed to have a maximum effective free distance. The choice of a recursive code enables the benefit of the interleaving gain, and the effective free distance criterion gives the optimum performance in the BER range of interest in practical applications. The outer code was chosen to have a maximum free distance for a given memory order, resulting in the optimum performance at very large SNR's.

6.12.1 The Effect of Interleaver Size and the Number of Iterations on AWGN Channels

If the inner code in a serial concatenated scheme is a recursive convolutional code the bit error probability is proportional to $N^{-d_f^o/2}$ for even values of the outer code minimum free distance d_f^o and

to $N^{-(d_f^o+1)/2}$ for odd values of d_f^o . That is, by increasing the interleaver size, the bit error probability is reduced for this class of codes. For turbo codes there is a virtual error floor at high SNR's and by increasing the interleaver size it is possible to reduce the error floor. Serial concatenated codes due to much higher interleaver gain and minimum free distance do not exhibit an error floor in the BER range of interest in data communications. From the expressions for the BER in Eq. (6.21) it can be concluded that the asymptotic behavior of the error probability is determined by the overall code free distance and the interleaver size.

The impact of the interleaver size on serial concatenated codes on AWGN channels is shown in Fig. 6.14. The serial concatenated code consists of a rate 2/3, 4-state recursive systematic convolutional code as the inner code and a rate 1/2, 4-state feedforward convolutional code as the outer code, so the overall code rate is 1/3. The generator polynomial for the outer code, denoted by $\mathbf{G}_o(D)$, is given by

$$\mathbf{G}_o(D) = [1 + D + D^2, 1 + D^2] \quad (6.55)$$

The generator polynomial for the inner code, denoted by $\mathbf{G}_i(D)$, is given by

$$\mathbf{G}_i(D) = \left[\begin{array}{ccc} 1 & 0 & \frac{1+D^2}{1+D+D^2} \\ 0 & 1 & \frac{1+D}{1+D+D^2} \end{array} \right] \quad (6.56)$$

As the simulation results indicate, the bit error probability can be reduced by increasing the interleaver size.

Comparison of turbo and serial concatenated codes is illustrated in Fig. 6.15. This figure shows the turbo code outperforms the corresponding serial concatenated code for the BER above 10^{-5} . At lower BER's, the serial concatenated code is superior, since the BER of turbo code levels out in this region due to the small free distance, while the BER of serial concatenated code has no error floor even at the BER of 10^{-10} .

The effect of the variable number of iterations on the BER performance on AWGN channels is shown in Fig. 6.16. As with the turbo codes, increasing the number of iterations reduces the BER, up to a certain threshold, which increases with the interleaver size.

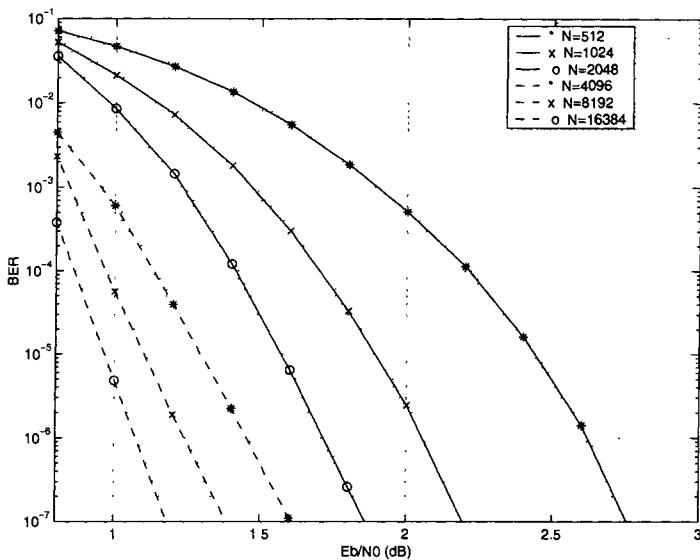


Fig. 6.14: Performance of a rate 1/3 serial concatenated code, with a rate 1/2, 4 state nonrecursive convolutional code as the outer code, a rate 2/3, 4 state recursive convolutional code as the inner code, AWGN channel, SOVA decoding algorithm, various interleaver size N , and the number of iterations 20

For interleaver size 1024, there is no further improvement above the number of iterations of 12, while for interleaver size 4096 and higher the threshold is 18.

6.12.2 The Effect of Memory Order on AWGN channels

Increasing the memory order of component codes generally results in increased free distance and effective free distance. As these two parameters determine the turbo code performance at high SNR's increasing the memory order will give the improved asymptotic error performance.

At low SNR's, the performance of turbo codes is dominated by error coefficients, which are generally higher for codes with higher

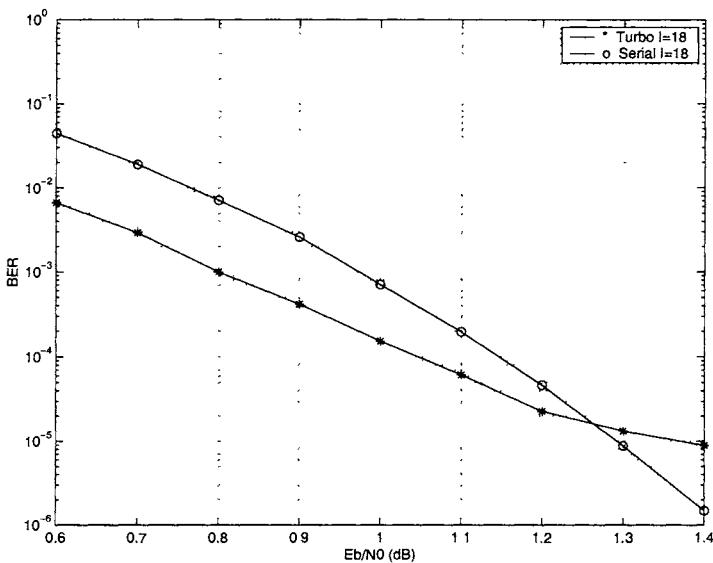


Fig. 6.15: Comparison of a rate 1/3, memory order 2 turbo code with interleaver size 4096 bits and a rate 1/3 serial concatenated code with memory order 2 outer code, interleaver size 4096 bits on an AWGN channel, SOVA decoding algorithm, the number of iterations 18.

memory orders. The effect of memory order on the turbo code performance is illustrated in Figs. 6.17 and 6.18, for two different interleaver sizes. For interleaver size 1024 bits, the memory order 4 code outperforms the memory order 3 and memory order 2 codes for E_b/N_o values above 1.5 dB. However increasing the interleaver size from 1024 to 4096, reduces the error coefficients in the same proportion, and thus the effective free distance becomes the dominant parameter at lower E_b/N_o , than in the previous case, so that the memory order 4 code has the best performance above E_b/N_o of 1 dB.

In iterative decoding of serial concatenated codes there is an additional effect of error propagation between the inner and outer decoder, which in general is more prominent with higher memory order codes. The error performance of rate 1/3 serial concatenated codes with variable memory order of the outer code is shown in Fig. 6.19, for interleaver size 1024. Clearly, in the E_b/N_o region up to

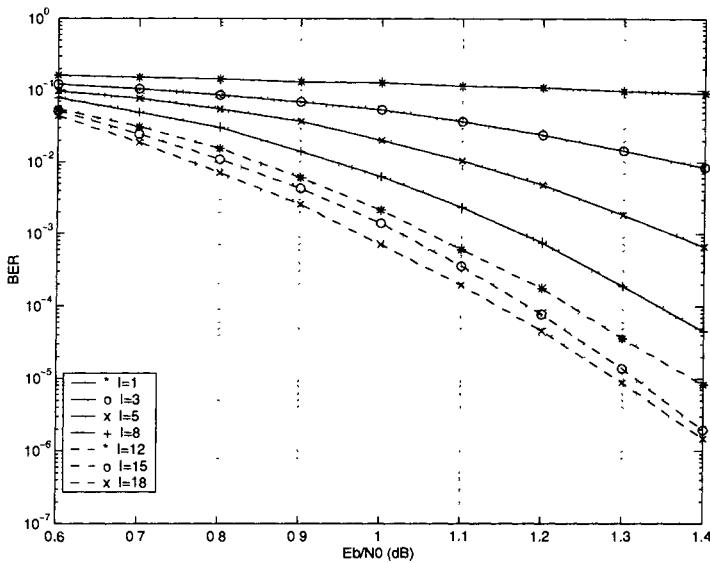


Fig. 6.16: BER performance of a rate 1/3 serial concatenated code with rate 1/2, 4 state outer code and rate 2/3, 4 state inner code with SOVA algorithm on an AWGN channel, interleaver size 4096 bits, variable number of iterations

1.8 dB, increasing the memory order results in worse performance. Increasing the interleaver size to 4096, diminishes the effect of error coefficients, so that for E_b/N_o above 1.5 dB the code with the highest memory order has the best performance, as shown in Fig. 6.20.

6.12.3 Comparison of MAP and SOVA Decoding Algorithms on AWGN Channels

The description of the MAP and SOVA iterative decoding algorithms for serial concatenated codes is presented in Sections 6.10 and 6.11.

The comparison of the MAP and SOVA decoding algorithms for a rate 1/3 serial concatenated code on an AWGN channel, obtained

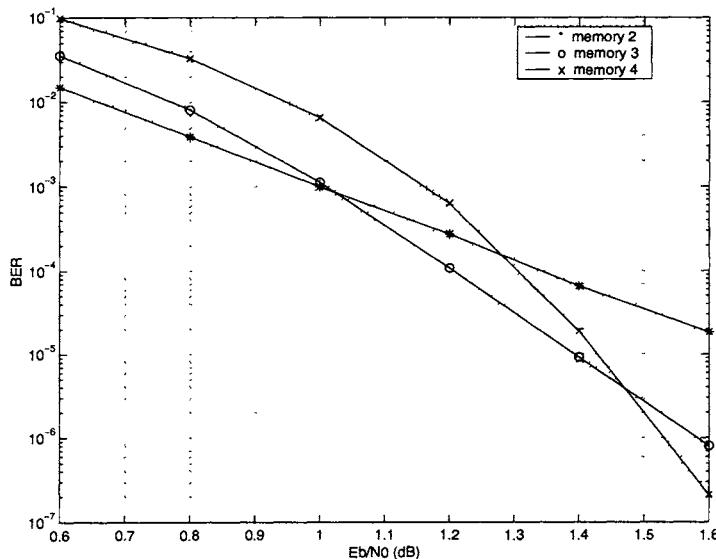


Fig. 6.17: Comparison of a rate 1/3 turbo code for different memory order with SOVA algorithm on an AWGN channel, interleaver size 1024 bits, the number of iterations 12

by simulation, is illustrated in Fig. 6.21. The serial concatenated code consists of a rate 2/3, 4 state RSC code as the inner code and a rate 1/2 16 state feedforward convolutional code as the outer code. The interleaver size is 1024. The figure shows that the MAP algorithm outperforms SOVA at the bit error probability of 10^{-4} by about 0.9 dB. Since ideal estimation of the Gaussian noise variance is assumed, this difference in reality might be smaller, as the MAP algorithm requires knowledge of the noise variance, while the SOVA does not.

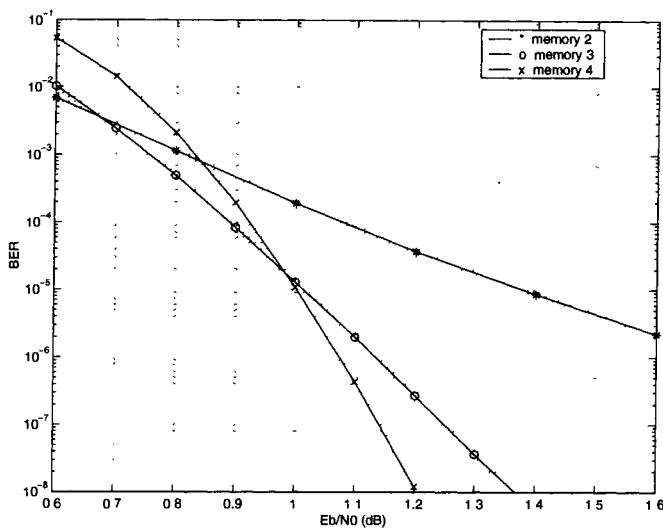


Fig. 6.18: Comparison of a rate 1/3 turbo code for different memory order with SOVA algorithm on an AWGN channel, interleaver size 4096 bits, the number of iterations 18

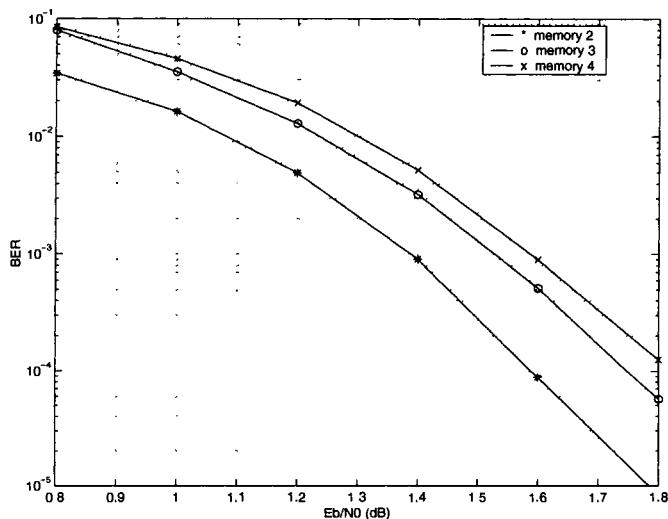


Fig. 6.19: Comparison of a rate 1/3 serial concatenated code for different outer code memory order with SOVA algorithm on an AWGN channel, interleaver size 1024 bits, the number of iterations 12.

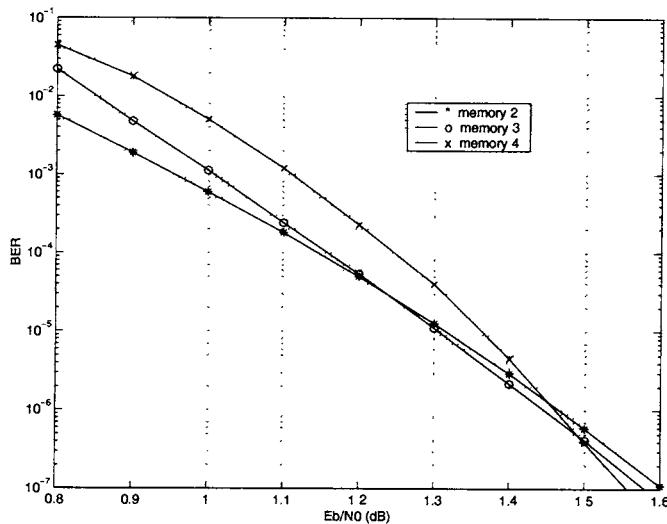


Fig. 6.20: Comparison of a rate 1/3 serial concatenated code for different outer code memory order with SOVA algorithm on an AWGN channel, interleaver size 4096 bits, the number of iterations 18

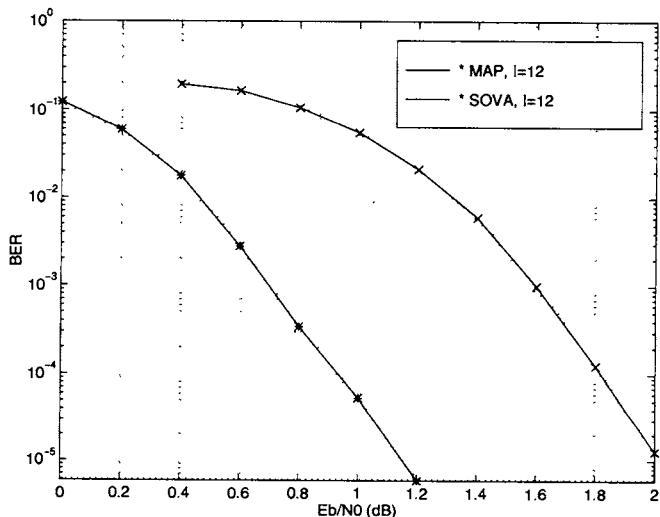


Fig. 6.21: Performance comparison of MAP and SOVA for a rate 1/3 serial concatenated convolutional code

Bibliography

- [1] C. Berrou, A. Glavieux and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: turbo-codes (1)," in Proc. ICC'93, May 1993.
- [2] G. Ungerboeck, "Channel coding with multilevel/phase signaling", IEEE Trans. Information Theory, Vol. 25, pp. 55-67, Jan. 1982.
- [3] J. Hagenauer, P. Robertson and L. Papke, "Iterative ('Turbo') decoding of systematic convolutional codes with the MAP and SOVA algorithms", in Proc. of the 1994 ITG Conference on Source and Channel Coding, Munich, October 1994.
- [4] B. Vucetic, "Iterative decoding algorithms", PIMRC'97, pp 99-120, Sept. 1997, Helsinki, Finland.
- [5] J. Hagenauer, E. Offer and L. Papke, "Iterative decoding of binary block and convolutional codes", IEEE Trans. Inform. Theory, Vol. 42, No. 2, March 1996, pp. 429-445.
- [6] G. Battail, M. C. Decouvelaere, and P. Godlewski, "Replication decoding", IEEE Trans. Inform. Theory, Vol. IT-25, pp. 332-345, May 1979.
- [7] J. Lodge, R. Young, P. Hoeher and J. Hagenauer, "Separable MAP "filters" for the decoding of product and concatenated codes", Proc. IEEE ICC'93, Geneva, Switzerland, may 1993, pp. 1740-1745.
- [8] P. Robertson, "Illuminating the structure of decoders of parallel concatenated recursive systematic (turbo) codes, Proc.

- IEEE Globecom Conf, San Francisco, CA, Dec. 1994, pp. 1298-1303.
- [9] P. Robertson, E. Villebrun and P. Hoeher, " A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain", Proc. IEEE ICC'95, Seattle, WA, June 1995, pp. 1009-1013.
 - [10] S. Benedetto, G. Montorsi, D. Divsalar 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.
 - [11] P. Jung, "Novel low complexity decoder for turbo codes", Electronics Letters, 19th Jan. 1995, Vol. 31, No. 2, pp. 86-87.
 - [12] A. Ushirokawa, T. Okamura, N. Kamiya and B. Vucetic, "Principles of turbo codes and their application to mobile communications," IEICE, Trans. Fundamentals, Vol. E81-A, No. 7, July 1998, pp. 1320-1329.
 - [13] B. Vucetic, Coding for fading channels, Lecture Notes, The University of Sydney, Sydney.
 - [14] W. Feng and B. Vucetic, "A list bidirectional soft output decoder of turbo codes," in Proc. Int. Symp. on Turbo Codes and Related Topics, Brest, France, Sep. 1997, pp. 288-292.
 - [15] M. Moher, "Decoding via cross entropy minimization," in Proc. IEEE Globecom. Conf., Houston, TX, Dec. 1993, pp. 809-813.
 - [16] R. Shao, S. Lin, and M. Fossorier, "Two simple stopping criteria for turbo decoding," IEEE Trans. Commun., vol. 47, no. 8, Aug. 1999, pp. 1117-1120.
 - [17] B. Vucetic, W. Feng, and J. Yuan, "Performance of turbo and serial concatenated convolutional codes," Technical Report, The University of Sydney, Aug. 1997.

Chapter 7

Interleavers

In the previous chapters we have presented the design and performance analysis of turbo codes. Interleaving plays an important role in achieving good performance of these schemes.

In this chapter, we consider interleaving techniques for turbo coding, the effects of interleaving size and structure on code performance and interleaver design for turbo codes. In particular, we discuss four types of interleaving structures: block interleavers, convolutional interleavers, random interleavers and code matched interleavers. A code matched interleaver design for turbo codes is proposed. We show that, with code matched interleavers, the turbo code error performance at moderate to high SNR's is improved and the "error floor" is reduced significantly compared to random interleavers. A new class of convolutional interleavers, called cyclic shift interleavers, is discussed. Their distinguishing features are low design complexity and memory, while the performance is comparable to random interleavers.

7.1 Interleaving

Interleaving is a process of rearranging the ordering of a data sequence in a one-to-one deterministic format. The inverse of this process is deinterleaving which restores the received sequence to its original order.

An interleaver device with size N is shown in Fig. 7.1. For simplicity, we assume that the data sequence at the input of the interleaver \mathbf{I} is binary, given by

$$\mathbf{c} = (c_1, c_2, c_3, \dots, c_N) \quad (7.1)$$

where $c_i \in \{0, 1\}$, $1 \leq i \leq N$. The interleaver permutes the sequence \mathbf{c} to a binary sequence

$$\tilde{\mathbf{c}} = (\tilde{c}_1, \tilde{c}_2, \tilde{c}_3, \dots, \tilde{c}_N) \quad (7.2)$$

where $\tilde{c}_j \in \{0, 1\}$, $1 \leq j \leq N$. The sequence $\tilde{\mathbf{c}}$ has all the elements of \mathbf{c} but in a different order. If we consider the input sequence \mathbf{c} and the output sequence $\tilde{\mathbf{c}}$ as a pair of sets with N elements, there is one-to-one correspondence $c_i \rightarrow \tilde{c}_j$ between each element of \mathbf{c} and each element of $\tilde{\mathbf{c}}$.

Let us define a set A as

$$A = \{1, 2, \dots, N\} \quad (7.3)$$

The interleaver can then be defined by a one-to-one index mapping function

$$\pi(A \rightarrow A) : j = \pi(i), i, j \in A \quad (7.4)$$

where i and j are the index of an element in the original sequence \mathbf{c} and the interleaved sequence $\tilde{\mathbf{c}}$, respectively. The mapping function can be represented by an interleaving vector

$$\pi_N = (\pi(1), \pi(2), \pi(3), \dots, \pi(N)) \quad (7.5)$$

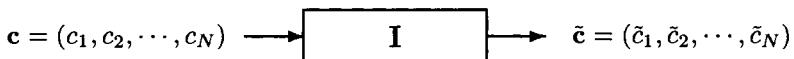


Fig. 7.1: An interleaver device

For example, we consider a pseudo-random interleaver with size $N = 8$. The input sequence is represented by

$$\mathbf{c} = (c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8)$$

The interleaved sequence is given by

$$\begin{aligned}\tilde{\mathbf{c}} &= (\tilde{c}_1, \tilde{c}_2, \tilde{c}_3, \tilde{c}_4, \tilde{c}_5, \tilde{c}_6, \tilde{c}_7, \tilde{c}_8) \\ &= (c_2, c_4, c_1, c_6, c_3, c_8, c_5, c_7)\end{aligned}$$

The mapping function is illustrated in Fig. 7.2. The interleaving vector can be expressed as

$$\begin{aligned}\pi_8 &= (\pi(1), \pi(2), \pi(3), \pi(4), \pi(5), \pi(5), \pi(7), \pi(8)) \\ &= (3, 1, 5, 2, 7, 4, 8, 6)\end{aligned}$$

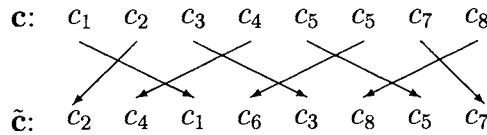


Fig. 7.2: An interleaver mapping

An interleaver or a deinterleaver is characterized by its delay and storage capacity. The interleaving or deinterleaving delay is the maximum delay encountered by any symbol before it is inserted into the output sequence. The storage capacity is the number of symbols stored by the interleaver or deinterleaver.

7.2 Interleaving with Error Control Coding

Interleaving is a practical technique to enhance the error correcting capability of coding. It has been widely used in conjunction with error control coding for channels that exhibit bursty error characteristics [1]. One example is a multipath fading channel in which signal variations due to multipath propagation often causes the signal to fall below the noise level, thus resulting in a large number of errors. A second example is a magnetic recording channel (tape or

disk) in which defects in the recording media result in clusters of errors.

An effective method to cope with burst errors is to insert an interleaver between the channel encoder and the channel. The coded data are reordered by the interleaver and then transmitted over the channel. At the receiver, the deinterleaver performs the reverse operation to restore the data to its original order. As a result of the interleaving/deinterleaving operation, bursty errors are spread out in time so that errors within a codeword appear independent. Thus, the burst error channel is transformed into a random error channel at the input of the decoder and a code designed for independent error channels could be used for burst error channels.

Another application of interleaving is for generation of product codes. Interleavers/deinterleavers are placed between successive encoders/decoders with the objective of decorrelating the component decoder input signals. Long product codes can be easily decoded in multiple stages. They provide a substantial improvement in performance when compared with one-stage codes.

7.3 Interleaving in Turbo Coding

In turbo coding, interleaving is employed before the information data is encoded by the second component encoder. In general, the interleaver size N is significantly larger than the code memory ν and the interleaver vector elements are chosen randomly.

The basic role of the interleaver is to construct a long block code from small memory convolutional codes, as long codes can approach the Shannon capacity limit. Secondly, it spreads out burst errors. The interleaver provides “scrambled” information data to the second component encoder and decorrelates the inputs to the two component decoders so that an iterative suboptimum decoding algorithm based on “uncorrelated” information exchange between the two component decoders can be applied. For example, after correction of some of the errors in the first component decoder, some of the remaining errors can be spread by the interleaver such that they become correctable in the other decoder. By increasing the number of iterations in the decoding process the bit error probability ap-

proaches the channel capacity. The final role of the interleaver is to break low weight input sequences, and hence increase the code free Hamming distance or reduce the number of codewords with small distances in the code distance spectrum.

7.3.1 The Effect of Interleaver Size on Code Performance

The turbo code error performance is determined by the code distance spectrum. The interleaver in a turbo encoder can reduce the error coefficients of low weight codewords through a process called “spectral thinning” [2]. This distance spectrum results in a reduced bit error probability by a factor $1/N$, which is called the interleaving gain [8]. The turbo code error performance at low SNR’s is dominated by the interleaver size.

To illustrate the effect of the interleaver size on the code error performance, we consider a memory order 2 turbo code with generator matrix $(1, 5/7)_{(oct)}$.

As shown in Chapter 4, the bit error probability of a turbo code over an additive white Gaussian noise channel is upper-bounded by

$$P_b(e) \leq \sum_{d=d_{\min}} B_d Q \left(\sqrt{2dR \frac{E_b}{N_0}} \right) \quad (7.6)$$

where B_d is the error coefficient and the set of all pairs of (d, B_d) represents the code distance spectrum.

The distance spectra of the turbo code with various interleaver sizes are calculated and substituted in (7.6) to calculate the bit error probability. The results are shown in Figs. 7.3 and Fig. 7.4 for the interleave sizes of 128, 256 and 512.

It is apparent from Fig. 7.3 that the shapes of the distance spectra for the turbo code with various interleaver sizes are quite similar. However, for significant spectral lines which dominate the turbo code performance, as discussed in Section 4.4, the error coefficients decrease with increasing interleaver size. The spectral thinning manifests in significant performance improvements proportional to the interleaver gain. As Fig. 7.4 shows the interleaver gain grows linearly with the interleaver size.

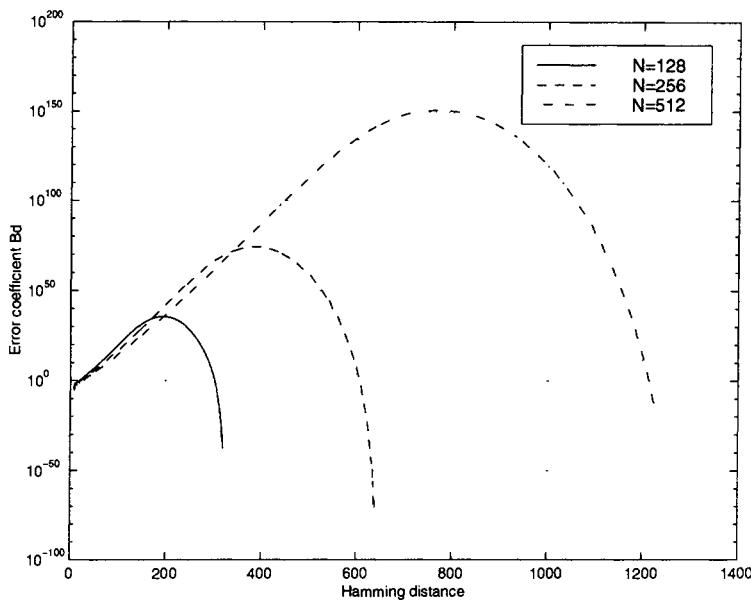


Fig. 7.3: Distance spectra for a turbo code with various interleaver sizes

7.3.2 The Effect of Interleaver Structure on Code Performance

As we discussed in the previous chapter, the turbo code performance at high SNR's is dominated by the code first several distance spectral lines which are produced by low weight input sequences. The interleaver structure affects the mapping of low weight input sequences to the interleaver output, and hence the first several distance spectral lines of the turbo code distance spectrum. It plays an important role in determining the code performance at high SNR's.

For example, we consider an input sequence to the first component encoder generating a low weight parity check sequence. It is desirable that the interleaver is capable of breaking this input pattern. That means that the interleaver does not produce the same input pattern to the second encoder, or an input sequence which generates a finite weight code sequence. In such a case, the input sequence to the second component encoder will most likely produce a high weight parity check sequence. This will result in an increase

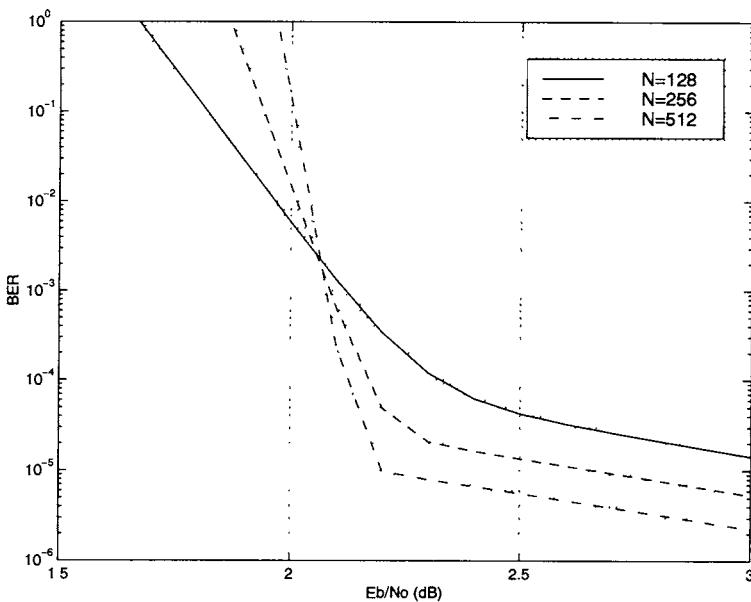


Fig. 7.4: Bit error probability upper bounds for a turbo code with various interleaver sizes

in the turbo codeword weight. If an interleaver is designed to break the low weight input sequences so that the resulting turbo code has a large minimum free distance, the error performance at high SNR's can be improved.

The previous analysis and discussion clearly show that the interleaver size and structure affect the turbo code error performance considerably. At low SNR's, the interleaver size is the only important factor, as the code performance is dominated by the interleaver gain. The effects induced by changing the interleaver structure at low SNR region are not significant. However, both the interleaver size and structure affect the turbo code minimum free distance and first several distance spectral lines. They play an important role in determining the code performance at high SNR's, and consequently, the asymptotic performance of the turbo code. It is possible to design particular interleavers which can result in good code performance at high SNR's. This is achieved by breaking several

low weight input patterns that produce low weight codewords in the overall code.

In the interleaver design, we first consider weight-2 input sequences as the most likely to generate low weight codewords. For example, the weight-2 input patterns for a turbo code with generator matrix $(1, 5/7)_{(oct)}$ are 1001, 100001, etc. A good interleaver ought to break as many of these input patterns as possible.

7.3.3 Interleaving Techniques

The use of pseudo-random interleavers does play a fundamental role in turbo coding schemes. As the pseudo-random structure is a major obstacle in the interleaver performance analysis and design, the use of the uniform interleavers, which are based on the probabilistic analysis of the ensemble of all interleavers, has been effective. However, in practice, one has to choose a particular interleaver structure which might perform better than the uniform interleaver. In the following sections, we will consider four types of interleaving techniques. They are block interleavers, convolutional interleavers, random interleavers, and code matched interleavers. The design of code matched interleavers will be described and turbo code performance with various interleavers will be presented.

7.4 Block Type Interleavers

7.4.1 Block Interleavers

A block interleaver formats the input sequence in a matrix of m rows and n columns, such that $N = m \times n$. The input sequence is written into the matrix row-wise and read out column-wise as illustrated in Fig. 7.5.

The deinterleaver stores the data into a matrix with the same format as the interleaver. The data are read out and delivered to the decoder row-wise.

The interleaving function of the block interleavers can be represented by

$$\pi(i) = [(i-1) \bmod n] \times m + \lfloor (i-1)/n \rfloor + 1, \quad i \in A \quad (7.7)$$

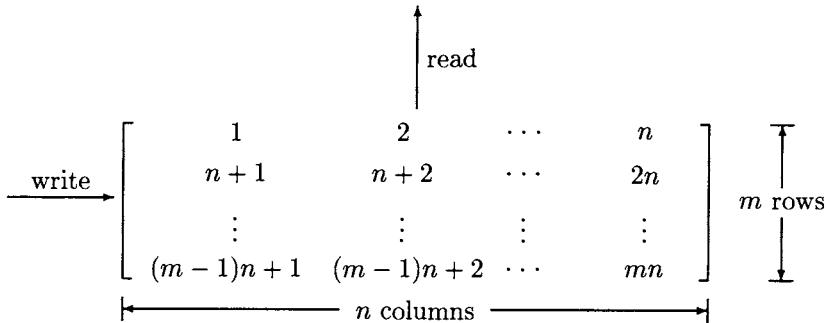


Fig. 7.5: A block interleaver

where $\lfloor x \rfloor$ means the integer of x . The end-to-end interleaver-deinterleaver delay is $2mn$. The memory requirement is mn for both the interleaver and deinterleaver.

In error control coding, the number of rows in the interleaver matrix is also called the interleaver degree (or depth) and the number of columns is called the interleaver span. The block interleavers and deinterleavers are easy to implement. However, they may fail to break certain low weight input patterns, such as weight 4, 6, 9, etc, square or rectangular input patterns [3]. For example, we consider a weight-2 input pattern 1001 generating a low weight parity check sequence. A weight-4 square input pattern shown in Fig. 7.6 cannot be broken by block interleavers. The weight-4 square input pattern concatenates two weight-2 input patterns. When the input sequence is written into the interleaving matrix, the two weight-2 input patterns are located in two rows with the four “1”s at the four corners of a square. When the input sequence is read out from the matrix, the two weight-2 input patterns remain in the interleaved sequence. That means block interleavers cannot break the weight-4 square input pattern.

Block interleavers are effective if the error patterns to be broken are confined to one row [12]. If the error patterns are confined to several consecutive rows, as for example concatenated error patterns, which are often long and spread over a number of rows, then the interleaving technique should be modified, such that the n

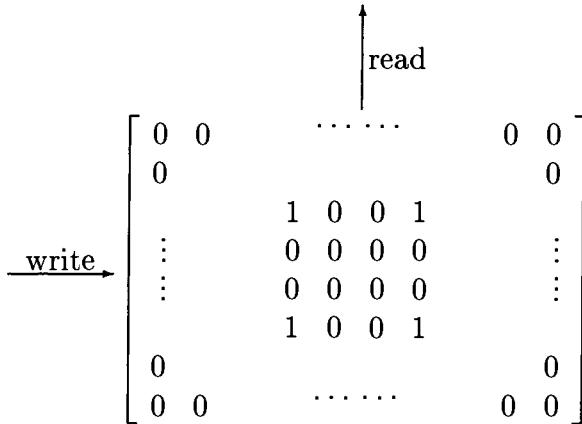


Fig. 7.6: A weight-4 square input pattern of block interleavers

columns of the interleaving matrix should be read out in a specified order to spread as many error patterns as possible. A method to reorder the columns is given in [11].

7.4.2 Odd-Even Block Interleavers

An odd-even interleaver is a particular type of interleaver which maps even position elements to even positions and odd position elements to odd positions. It can be expressed as

$$\pi(A \rightarrow A) : (\pi(i) + i) \bmod 2 = 0, \quad i \in A \quad (7.8)$$

For a rate 1/2 punctured turbo code, an odd-even interleaver can provide uniform error protection across the information sequence [4], as explained below.

For example, let us consider a binary data sequence

$$\mathbf{c} = (c_1, c_2, \dots, c_{15})$$

The data sequence is encoded by a rate 1/3 turbo encoder. A rate 1/2 turbo code can be obtained by puncturing the even parity check digits of the first component encoder and the odd parity check digits

of the second component encoder. The odd information bits $c_i, i = 1, 3, 5, \dots$, generate the encoded parity digits $v_{i,1}$ at the output of the first component encoder, as shown in Table 7.1.

Table 7.1 Odd coded parity digits of the first encoder

c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}
$v_{1,1}$	—	$v_{3,1}$	—	$v_{5,1}$	—	$v_{7,1}$	—	$v_{9,1}$	—	$v_{11,1}$	—	$v_{13,1}$	—	$v_{15,1}$

Let us first assume that the data sequence is interleaved by a pseudo-random interleaver. The interleaved data sequence is given by

$$\tilde{\mathbf{c}} = (c_3, c_5, c_{11}, c_8, c_{14}, c_{12}, c_9, c_6, c_2, c_{15}, c_1, c_4, c_7, c_{10}, c_{13}). \quad (7.9)$$

The second component encoder generates the even coded parity digits $v_{i,2}$, as shown in Table 7.2. The overall code parity sequence is obtained by multiplexing both the odd and even coded digits and it is shown in Table 7.3.

Table 7.2 Even coded parity digits of the second encoder

c_3	c_5	c_{11}	c_8	c_{14}	c_{12}	c_9	c_6	c_2	c_{15}	c_1	c_4	c_7	c_{10}	c_{13}
—	$v_{5,2}$	—	$v_{8,2}$	—	$v_{12,2}$	—	$v_{6,2}$	—	$v_{15,2}$	—	$v_{4,2}$	—	$v_{10,2}$	—

Table 7.3 Multiplexed coded sequence

c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}
$v_{1,1}$	$v_{5,2}$	$v_{3,1}$	$v_{8,2}$	$v_{5,1}$	$v_{12,2}$	$v_{7,1}$	$v_{6,2}$	$v_{9,1}$	$v_{15,2}$	$v_{11,1}$	$v_{4,2}$	$v_{13,1}$	$v_{10,2}$	$v_{15,1}$

It is obvious from Table 7.1 that each odd information bit has a coded parity digit associated with it. However, the even coded

digits, shown in Table 7.2, can be generated by either odd or even information bits. This means that some of the odd information bits have two, such as $v_{5,1}$ and $v_{5,2}$, but some of the even information bits have no parity digits associated with them, such as c_2 . Thus, error protection is not uniformly distributed across the information sequence. If some information bits with no associated parity digits are in error, they cannot be corrected by either of the two decoders.

An odd-even interleaver can overcome this problem. Here we present a block type odd-even interleaver. In this case the numbers of rows and columns for the interleaver matrix must be odd [4]. For the previous example, the interleaver matrix can be represented as

$$\begin{bmatrix} c_1 & c_2 & c_3 & c_4 & c_5 \\ c_6 & c_7 & c_8 & c_9 & c_{10} \\ c_{11} & c_{12} & c_{13} & c_{14} & c_{15} \end{bmatrix}$$

The interleaved data sequence can be expressed as

$$\tilde{\mathbf{c}} = (c_1, c_6, c_{11}, c_2, c_7, c_{12}, c_3, c_8, c_{13}, c_4, c_9, c_{14}, c_5, c_{10}, c_{15}) \quad (7.10)$$

The interleaved data sequence is encoded by the second component encoder, which generates the even parity digits shown in Table 7.4.

Table 7.4 Even coded parity digits after interleaving

c_1	c_6	c_{11}	c_2	c_7	c_{12}	c_3	c_8	c_{13}	c_4	c_9	c_{14}	c_5	c_{10}	c_{15}
—	$v_{6,2}$	—	$v_{2,2}$	—	$v_{12,2}$	—	$v_{8,2}$	—	$v_{4,2}$	—	$v_{14,2}$	—	$v_{10,2}$	—

Now each odd and even information bit has a coded digit associated with it. The coded sequence is obtained by multiplexing the coded sequences from Tables 7.1 and 7.4, as shown in Table 7.5. With this sequence the error protection is uniformly distributed, resulting in a better decoder performance, compared to the scheme with a non odd-even interleaver.

7.4.3 Block Helical Simile Interleavers

Block helical simile interleavers are based on the conventional block interleaver. The basic principle of the block helical simile interleaver

Table 7.5 Multiplexed coded sequence for an odd-even interleaver

c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}
$v_{1,1}$	$v_{6,2}$	$v_{3,1}$	$v_{2,2}$	$v_{5,1}$	$v_{12,2}$	$v_{7,1}$	$v_{8,2}$	$v_{9,1}$	$v_{4,2}$	$v_{11,1}$	$v_{14,2}$	$v_{13,1}$	$v_{10,2}$	$v_{15,1}$

is to read data diagonally instead of column-wise, with certain limitations on the number of columns in the interleaver matrix. The resulting interleaver terminates both component encoders in a turbo encoder to the all-zero state [5]. This is achieved by appending ν tail bits at the end of the information sequence, where ν is the memory of the component convolutional code. The interleaver is thus more efficient than the pseudo-random interleavers in which only one encoder is forced to go to the all-zero state.

To generate a block helical simile interleaver, the number of the columns for the interleaver matrix must be a multiple of $(\nu + 1)$ [5]. The information sequence is written into the matrix row-wise and read out diagonally from the left to the right and from the bottom to the top to prevent two adjacent data bits written in the same column or row from remaining neighbours in the interleaved sequence. This interleaver structure can avoid consecutive bits being output from the same column or row. An example of a block helical interleaver with size 15 for a memory-2 turbo code is given by the matrix

$$\begin{bmatrix} c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 \\ c_7 & c_8 & c_9 \\ c_{10} & c_{11} & c_{12} \\ c_{13} & c_{14} & c_{15} \end{bmatrix} \quad (7.11)$$

The interleaved sequence generated from this matrix is given by

$$\tilde{\mathbf{c}} = (c_{13}, c_{11}, c_9, c_4, c_2, c_{15}, c_{10}, c_8, c_6, c_1, c_{14}, c_{12}, c_7, c_5, c_3) \quad (7.12)$$

As we can see this scheme does not produce an odd-even interleaver. A necessary condition to generate a simile odd-even block

helical interleaver is to choose the number of columns to be an even number and a multiple of $(\nu + 1)$. Also, the number of columns and the number of rows have to be relatively prime [5].

7.5 Convolutional Type Interleavers

7.5.1 Convolutional Interleavers

Convolutional interleavers have been proposed by Ramsey [14] and Forney [15]. The structure proposed by Forney is illustrated in Fig. 7.7.

Both the interleaver and deinterleaver consist of an input and output commutator and a bank of L shift registers. The information sequence to be interleaved is arranged in blocks of L bits. The input commutator cyclically inserts each block of L bits into the bank of L registers. The i th bit in each block is delayed by the i th shift register and the delay of the shift register is $(i - 1)B$. The output commutator cyclically samples the bank of L registers in the same order as the input one.

The deinterleaver performs the inverse operation. That is, the i th bit in each block is delayed by the i th shift register, where the delay of the shift register is $(L - i)B$.

The convolutional interleaving function, after the initial states of the shift registers have been cleared out, can be expressed as

$$\pi(i) = i + [(i - 1) \bmod L] \cdot LB, \quad i \in A \quad (7.13)$$

This interleaver is also called an $LB \times L$ interleaver. The end-to-end interleaver-deinterleaver delay is $(L - 1)LB$. The memory requirement is $(L - 1)LB/2$ for either the interleaver or the deinterleaver. Therefore, there is a reduction of one-half in the end-to-end delay and the memory of a convolutional interleaver compared to the block interleaver to achieve a similar level of scrambling.

An example of convolutional interleaver with $L = 3$ and $B = 2$ is shown in Fig. 7.8. The interleaver size is 21 and the interleaved sequence is given by

$$\tilde{\mathbf{c}} = (c_1, 0, 0, c_4, 0, 0, c_7, c_2, 0, c_{10}, c_5, 0, c_{13}, c_8, c_3, c_{16}, c_{11}, c_6,$$

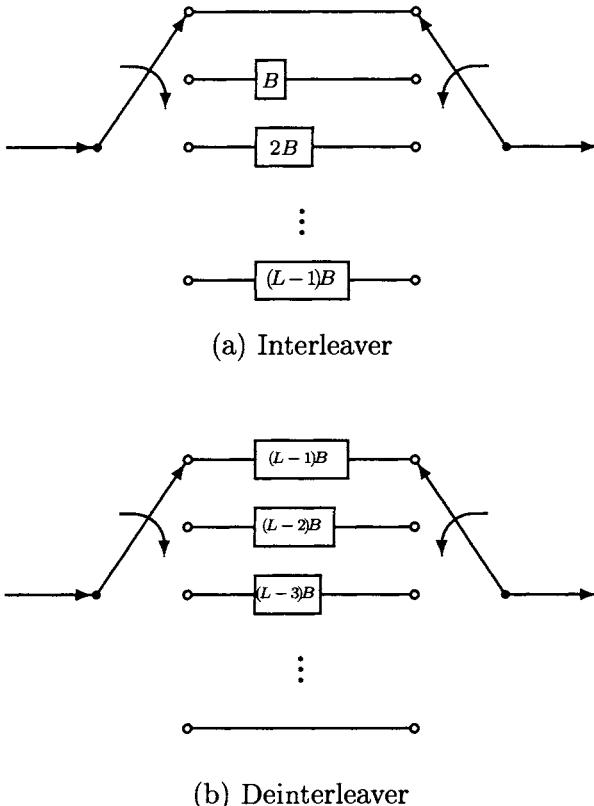


Fig. 7.7: A convolutional interleaver and deinterleaver

$$c_{19}, c_{14}, c_9, 0, c_{17}, c_{12}, 0, c_{20}, c_{15}, 0, 0, c_{18}, 0, 0, c_{21})$$

where zeros are produced by the initial zero states of the shift registers.

Ramsey's Type III (n_1, n_2) interleaver can be generated by using the same function as for the convolutional interleaver, given by (7.13), with $n_1 = LB - 1$ and $n_2 = L$. The interleaver has the following property [14]:

$$|i - j| \geq n_1 \quad (7.14)$$

19	16	13	10	7	4	1
20	17	14	11	8	5	2
21	18	15	12	9	6	3

Fig. 7.8: A convolutional interleaver with $L = 3$ and $B = 2$

whenever

$$|\pi(i) - \pi(j)| \leq n_2 - 1, \quad i, j \in A \quad (7.15)$$

If the parameters B and L of the convolutional interleaver are chosen properly, it can break some rectangular low weight input patterns which appear in the block interleavers, and it can give very good performance [16]. However, from Eq. (7.13), we can see that for any value of i , the corresponding value of $\pi(i)$ is always larger than or equal to i . That is, the output sequence is expanded by $(L - 1)LB$ symbols relative to the input sequence. One way to overcome the sequence expansion is to not include the zeros within the interleaved sequence. However, this will change the interleaving function and worsen the interleaver performance.

7.5.2 Cyclic Shift Interleavers

An alternative way of generating convolutional type interleavers is based on cyclic shifts. In the interleaver, the information sequence is written into a matrix column-wise. The matrix has m rows and n columns such that $N = m \times n$ and $m \leq n$. Then m row sequences of size n are applied to the bank of m registers, where the i th register shifts the i th row sequence in the matrix cyclically to the left by $(i - 1)B$, where B is a small integer whose value is chosen such that $B \leq n/m$. The shifted sequences form a new matrix and the data are read out from the matrix column-wise.

The deinterleaver performs a similar operation, whereby, the i th row sequence in the matrix is cyclically shifted to the right by $(i - 1)B$.

For example, we consider a cyclic shift interleaver with size 21

as shown in Fig. 7.9. The parameters m , n and B are set to 3, 7 and 2, respectively. The interleaved sequence is given by

$$\tilde{\mathbf{c}} = (c_1, c_{17}, c_{12}, c_4, c_{20}, c_{15}, c_7, c_2, c_{18}, c_{10}, c_5, c_{21}, \\ c_{13}, c_8, c_3, c_{16}, c_{11}, c_6, c_{19}, c_{14}, c_9)$$

$$\left[\begin{array}{ccccccc} 19 & 16 & 13 & 10 & 7 & 4 & 1 \\ 20 & 17 & 14 & 11 & 8 & 5 & 2 \\ 21 & 18 & 15 & 12 & 9 & 6 & 3 \end{array} \right] \xrightarrow{\text{cyclic shift}} \left[\begin{array}{ccccccc} 19 & 16 & 13 & 10 & 7 & 4 & 1 \\ 14 & 11 & 8 & 5 & 2 & 20 & 17 \\ 9 & 6 & 3 & 21 & 18 & 15 & 12 \end{array} \right]$$

Fig. 7.9: A cyclic shift interleaver

Comparing the convolutional interleaver in Fig. 7.8 and the cyclic shift interleaver in Fig. 7.9, it is obvious that the cyclic shift interleaver retains the property of the convolutional interleaver, expressed by (7.14) and (7.15) with $n_1 = mB - 1$ and $n_2 = m$, but it overcomes its disadvantage of expanding the output sequence.

7.6 Random Type Interleavers

7.6.1 Random Interleavers

In the random interleaver a block of N input bits is read into the interleaver and read out randomly. The interleaver vector $\pi(i)$, $i \in 1, 2, \dots, N$, can be generated according to the following algorithm, which requires N steps:

Step 1. Choose randomly an integer i_1 from the set $A = \{1, 2, \dots, N\}$, according to a uniform distribution between 1 and N , with the probability of $p(i_1) = \frac{1}{N}$. The chosen integer i_1 is set to be $\pi(1)$.

Step k . ($k > 1$) Choose randomly an integer i_k from the set $A_k = \{i \in A, i \neq i_1, i_2, \dots, i_{k-1}\}$, according to a uniform distribution, with the probability of $p(i_k) = \frac{1}{N-k+1}$. The chosen integer i_k is set to be $\pi(k)$.

When $k = N$, the last integer i_N is set to be $\pi(N)$.

When the interleaver size is $N = 2^m - 1$, a pseudo-random interleaver can be generated by an m -stage shift register with linear feedback as illustrated in Fig. 7.10. The feedback polynomial represented by

$$1 + a_1D + a_2D^2 + \cdots + a_mD^m \quad (7.16)$$

must be primitive with degree m . If the initial state of the shift register is not the all-zero state, the shift register will go through all $2^m - 1$ states cyclically. Therefore, the state of the m -stage shift register can represent the interleaving function.

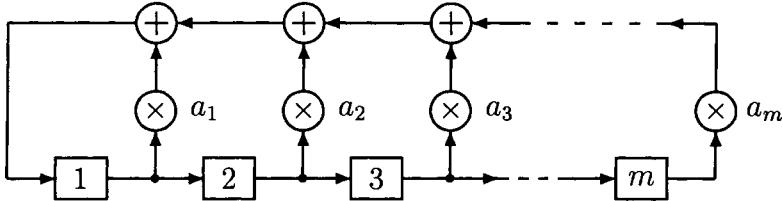


Fig. 7.10: A general m -stage shift register with linear feedback

For example, for interleaver size $N = 7$, a pseudo-random interleaver can be generated by a 3-stage shift register with the polynomial of $1 + D + D^3$. Assume the initial state is 010 and input sequence is

$$\mathbf{c} = (c_1, c_2, c_3, c_4, c_5, c_6, c_7). \quad (7.17)$$

The interleaved sequence generated by the pseudo-random interleaver is

$$\tilde{\mathbf{c}} = (c_2, c_5, c_6, c_7, c_3, c_1, c_4). \quad (7.18)$$

where the interleaver outputs are the octal form representative of state numbers.

7.6.2 Non-uniform Interleavers

The interleaver used in the original turbo codes is a non-uniform interleaver [3]. It is based on the ordinary square block interleaver,

but the data are read out diagonally with certain row and column jumps between each reading.

Let i and j be the addresses of the row and column for writing, and i_r and j_r the addresses of the row and column for reading. For $N = M \times M$, where M is a power of 2, the non-uniform interleaving may be described by [3]

$$\begin{aligned} i_r &= \left(\frac{M}{2} + 1\right) \cdot (i + j) \bmod M \\ k &= (i + j) \bmod L \\ j_r &= \{[P(k) \cdot (j + 1)] - 1\} \bmod M \end{aligned}$$

where L is a small integer whose value is chosen empirically as a function of M , $P(k)$, $k = 0, 1, 2, \dots, L - 1$, are functions of the row address i_r , and the values of $P(k)$ should be relatively prime with respect to M . Note that, for reading, the column index j_r is a function of the row index i_r . A multiplying factor $(\frac{M}{2} + 1)$ is used to prevent two input data written in two consecutive rows from remaining neighbours in the interleaved sequence. In addition, reading is performed diagonally with respect to writing to break rectangular low weight input patterns which appear in the conventional block interleavers.

Typically, for interleaver size $N = 256 \times 256$, L is chosen to be 8. The numbers of $P(k)$, $k = 0, 1, 2, \dots, 7$, are given by [3]

$$\begin{aligned} P(0) &= 17, & P(1) &= 37, & P(2) &= 19, & P(3) &= 29 \\ P(4) &= 41, & P(5) &= 23, & P(6) &= 13, & P(7) &= 7 \end{aligned} \quad (7.19)$$

7.6.3 *S*-random Interleavers

S-random interleavers proposed by Divsalar and Pollara are pseudo-random interleavers [13]. They are based on the random generation of N integers from 1 to N with an *S*-constraint, where *S* is defined as the minimum interleaving distance. *S*-random interleavers are also called spread interleavers.

An *S*-random interleaver is defined as follows. Each randomly selected integer is compared to the S_1 previously selected integers. If the absolute value of the difference between the current selected

integer and any of the S_1 previous selected integers is smaller than S_2 , then the current selected integer is rejected. This process is repeated until all N integers are selected.

In general an S -random interleaver can be described as

$$|i_1 - i_2| \geq S_2 \quad (7.20)$$

whenever

$$|\pi(i_1) - \pi(i_2)| \leq S_1, \quad i_1, i_2 \in A \quad (7.21)$$

where S_1 and S_2 are two integers smaller than N . In a turbo encoder, these two parameters should, in general, be chosen to correspond to the maximum input pattern lengths to be broken by the interleaver. Thus, they should be chosen as large as possible. The length of an input pattern is defined as the length of the input sequence, starting from the first binary “1” to the last binary “1”.

However, as the search time for this algorithm becomes prohibitively large for large values of S_1 and S_2 , a good trade-off between interleaver performance and search time is obtained for S_1 , $S_2 < \sqrt{N/2}$.

When two identical component encoders are employed in a turbo encoder, it is appropriate to set $S_1 = S_2$. In the following analysis, we assume the two component codes are identical and $S_1 = S_2 = S$. Note that, for $S = 1$, the S -random interleaver becomes a random interleaver.

For a turbo encoder, an S -random interleaver can break the input patterns with lengths up to $S + 1$ and generate high weight parity check sequences, as explained below.

Let $\{\mathbf{c}\}$ be the set of all the input patterns generating an error event of the component code. The length of an input pattern \mathbf{c} is denoted by $l(\mathbf{c})$, and the weight of the input pattern is denoted by $w(\mathbf{c})$. If the length of an input pattern is small, it will likely produce a low weight codeword. Therefore, the interleaver should break this kind of input patterns. With an S -random interleaver, the input pattern will be mapped to another sequence $\tilde{\mathbf{c}}$. If $\tilde{\mathbf{c}}$ is not an error pattern, we say that the input pattern for the component encoder is broken. The second encoder will produce a parity check sequence of infinite weight (if no termination is performed). Otherwise, if

$l(\mathbf{c}) \leq S + 1$, because of the S -constraint, $l(\tilde{\mathbf{c}}) > (w(\mathbf{c}) - 1)(S + 1)$. As the path length increases, $\tilde{\mathbf{c}}$ will likely produce a high weight parity check sequence. Thus, in both cases, the overall codeword weight will be high.

Based on the previous discussion, we can conclude that an S -random interleaver can either break the input patterns with length up to $S + 1$ or expand these input patterns to longer error patterns with length more than $(w - 1)(S + 1)$, where w is the input sequence weight, no matter what the component code is. Thus S -random interleavers can achieve better performance compared to pseudo-random interleavers.

It is worth noting that the S -random interleaver functions (7.20) and (7.21) agree with the property of the convolutional interleaver as shown in (7.14) and (7.15), if the parameters n_1 and n_2 of the convolutional interleaver are chosen in a particular way, such as $n_1 = S_2$, $n_2 = S_1 + 1$. That is to say these two types of interleavers are equivalent in the sense of breaking low weight input patterns.

In addition, dithered golden interleavers have been reported in [23]. It is shown that they perform quite similar to S -random interleavers for low rate turbo codes, but better for high rate punctured turbo codes.

7.7 Code Matched Interleavers

A code matched interleaver is a particular pseudo-random interleaver which can break several low weight input sequences in such a manner that the first several spectral lines of the turbo code distance spectrum are eliminated. It is obvious that these low weight input patterns to be broken depend on the component codes, and therefore, the interleaver should match the distance spectrum of the component codes. We call such an interleaver a *code matched interleaver* [9].

To break the low weight input patterns, we must compute the distance spectrum of the low weight codewords of the component codes first. Then on the basis of the performance analysis, we determine the most significant input patterns which give large contrib-

butions to the error probability for the overall code at high SNR's. In the interleaver design, we make sure that these significant input patterns are broken so that they do not appear after interleaving. This interleaver will eliminate first several spectral lines of the original distance spectrum and increase the overall turbo code Hamming distance. Consequently, the code performance at high SNR's is improved and the error floor is lowered.

7.8 Design of Code Matched Interleavers

For a given component code, several attempts have been taken to optimize the interleaver structure of a given length [17]-[22]. At present, there is no systematic method of designing code matched interleavers for turbo codes. This is partly due to a high complexity of the problem. An effective way is to derive interleaver design guidelines based on the code performance analysis. This is followed by construction of interleaver search algorithms and actual search for good interleaver structures. The guidelines proposed in [9] [10] are summarized as follows:

- 1) Keep the interleaver random.

Note that in iterative soft output decoding algorithms, the information exchange between the two component decoders is possible because of the interleaving/deinterleaving operations. The input and output sequences of the interleaver should be uncorrelated. The more "scrambled" the interleaver is, the more "uncorrelated" the information exchange is.

- 2) Break as many of the short length input patterns as possible. The input patterns with short lengths will likely produce low weight codewords. The interleaver should break these input patterns or expand these input patterns to longer paths so that the resulting overall codeword weight will increase. Obviously, there are quite a large number of these input patterns and we need to include as many of these patterns as possible in the interleaver design.

- 3) Determine the most significant low weight input patterns to be broken.

The most significant input patterns are those giving large contributions to the code error probability at high SNR's. These input patterns produce low weight codewords corresponding to the first several distance spectral lines in turbo code distance spectrum. The most significant input patterns are determined on the basis of the performance analysis. The interleaver design makes sure that these input patterns are broken so that the first several distance spectral lines of the original distance spectrum are eliminated.

Recall that the S -random interleaver functions (7.20) and (7.21) agree with the first two interleaver design guidelines. Because of the S -constraint, a large number of short input patterns with length up to $S + 1$ are broken. The main remaining task is to determine the most significant low weight input patterns to be broken.

Let us denote by w the input pattern weight, by $z_1(w)$ the parity check weight of the first component encoder and by $z_2(w)$ the parity check weight of the second component encoder. The overall weight of the error pattern is given by

$$d(w) = w + z_1(w) + z_2(w) \quad (7.22)$$

Note that, for recursive systematic convolutional component codes, the minimum input weight resulting in a finite weight error pattern is two. Throughout the section we use an example of a turbo code with generator matrix $(1, 21/37)_{(oct)}$ for both component codes.

(a) Input patterns with weight $w = 2$

The weight-2 input pattern generating the lowest weight of the parity check sequence, denoted by \mathbf{c}_2 , can be represented by

$$\mathbf{c}_2 = (00 \cdots 0010000100 \cdots 00) \quad (7.23)$$

where the distance between the two "1"s is 5. The parity check sequence of the component code generated by this weight-2 input pattern is given by

$$\mathbf{v} = (00 \cdots 0011001100 \cdots 00) \quad (7.24)$$

The weight of this parity check sequence, denoted by z_{\min} , is 4. If an interleaver maps the input sequence to a sequence with the same pattern, the resulting overall codeword weight will be equal to the code effective free distance $d_{\text{free,eff}}$ given by

$$d_{\text{free,eff}} = 2 + 2z_{\min} \quad (7.25)$$

In general, a weight-2 input sequence generating a finite weight codeword can be represented by a polynomial

$$\mathbf{c}_2(D) = (1 + D^{5t})D^\tau \quad (7.26)$$

where 5 is the fundamental period of the recursive systematic convolutional encoder, t and τ are integers such that $t > 0$, $5t$ is the distance between the two “1”s in the weight-2 input sequence and τ is a time delay. The corresponding parity check weight of the component code is

$$z(2) = t \cdot (z_{\min} - 2) + 2 \quad (7.27)$$

Let us denote by i and j the positions of “1”s in a weight-2 input sequence. If an interleaver function meets the following conditions

$$\begin{aligned} |i - j| \bmod 5 &= 0 \\ |\pi(i) - \pi(j)| \bmod 5 &= 0 \end{aligned} \quad (7.28)$$

this interleaver will map the input sequence \mathbf{c}_2 to another same pattern sequence $\tilde{\mathbf{c}}_2$. Thus the weight-2 input sequence with this pattern will produce a finite weight codeword as illustrated in Fig. 7.11, where $t_1, t_2 = 1, 2, 3, \dots$, and $5t_1$ and $5t_2$ are distances between the two “1”s in the input sequence \mathbf{c}_2 and its interleaved sequence $\tilde{\mathbf{c}}_2$, respectively. The overall codeword weight is

$$d(2) = 6 + (t_1 + t_2) \cdot (z_{\min} - 2) \quad (7.29)$$

A code matched interleaver should break this kind of weight-2 input patterns such that

$$|\pi(i) - \pi(j)| \bmod 5 \neq 0 \quad (7.30)$$

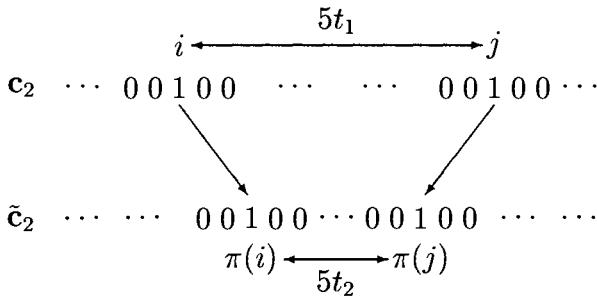


Fig. 7.11: A weight-2 input sequence pattern

whenever

$$|i - j| \bmod 5 = 0 \quad (7.31)$$

In fact, it is not possible to break all these patterns. For an unbroken input pattern, in order to maximize the overall codeword weight, we require maximizing the value of

$$\min(t_1 + t_2) = \min(|i - j| + |\pi(i) - \pi(j)|) / 5 \quad (7.32)$$

(b) Input patterns with weight $w = 3$

The procedure for breaking weight-3 input patterns is similar to the one described for weight-2 input patterns. The mapping from one weight-3 input pattern to another which satisfies the two distances between the three “1”s is not easy to make. Nevertheless, an interleaver with the S -constraint can either break a short weight-3 input pattern with length up to $S + 1$ or expand it to a longer one with length more than $2(S + 1)$. Thus, weight-3 input patterns will likely produce high weight codewords. We do not typically consider them in the interleaver design.

(c) Input patterns with weight $w = 4$

There are two types of weight-4 input patterns. The first type is a single error pattern with input weight-4. Similar to the case of input weight-3, an interleaver with S -constraint can either break a short pattern or expand it to length $3(S + 1)$. Thus we will not consider this case in the interleaver design. The other type is a

combination of two single weight-2 input patterns. This type of input sequences has the form

$$\mathbf{c}_4(D) = (1 + D^{5t_1})D^{\tau_1} + (1 + D^{5t_2})D^{\tau_2} \quad (7.33)$$

where τ_1 and τ_2 are integers with $\tau_2 > \tau_1 + 5t_1$.

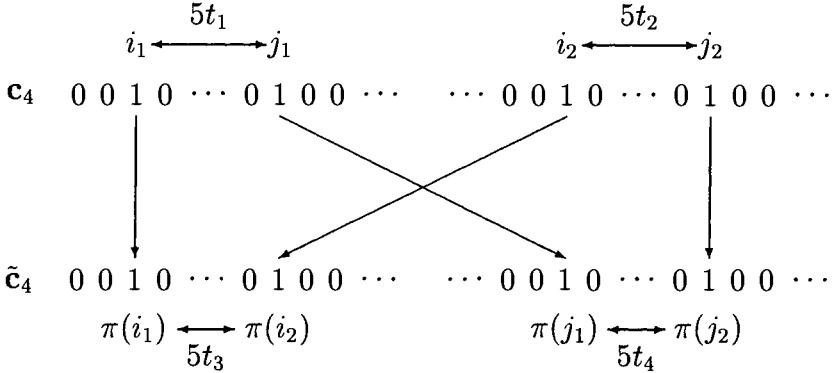


Fig. 7.12: A weight-4 input sequence pattern

Let us denote by i_1, j_1, i_2 , and j_2 the positions of data “1”s in a weight-4 input sequence, where $i_1 < j_1 < i_2 < j_2$. A weight-4 input pattern is shown in Fig. 7.12. If an interleaver mapping function satisfies the following conditions

$$\begin{aligned} |i_1 - j_1| \bmod 5 &= 0 \\ |i_2 - j_2| \bmod 5 &= 0 \\ |\pi(i_1) - \pi(i_2)| \bmod 5 &= 0 \\ |\pi(j_1) - \pi(j_2)| \bmod 5 &= 0 \end{aligned} \quad (7.34)$$

the input sequence will generate a low weight codeword. The overall codeword weight is

$$d(4) = 12 + (t_1 + t_2 + t_3 + t_4) \cdot (z_{\min} - 2) \quad (7.35)$$

where

$$t_1 = |i_1 - j_1| / 5$$

$$\begin{aligned} t_2 &= |i_2 - j_2| / 5 \\ t_3 &= |\pi(i_1) - \pi(i_2)| / 5 \\ t_4 &= |\pi(j_1) - \pi(j_2)| / 5 \end{aligned}$$

A code matched interleaver should avoid the crossing mapping of two single weight-2 input patterns as illustrated in Fig. 7.12 or maximizing the value of $\min(t_1 + t_2 + t_3 + t_4)$ for the unbroken input patterns..

(d) Input patterns with weight $w > 4$

The effect of error patterns with input weight $w > 4$ on code performance is small. This is partly due to the fact that they usually produce high weight codewords as a result of the S -constraint. Therefore, we will not consider them in the interleaver design.

From the above discussion we can conclude that the most significant low weight input patterns are those of weight-2 and combinations of weight-2. In the interleaver design, we only need to break these particular input patterns. This makes the interleaver design computationally efficient.

The code matched interleaver proposed in [9] is a modified S -random interleaver. The modification consists in breaking the most significant low weight error patterns. The weight of the input sequence to be broken is denoted by w . The design procedure is summarized as follows:

1. Select an integer randomly from the set $A = \{1, 2, 3, \dots, N\}$. The selected integer represents the position of the interleaver output.
2. Each randomly selected integer is compared to the S previous selected integers. If the absolute value of the difference between the current selected integer and any of the S previous selected is smaller than S , then reject the current selected integer and go to Step 1.
3. Check whether the current interleaver output at time t and $w - 1$ previous interleaver outputs within time $(1, 2, 3, \dots, t - 1)$ form an error pattern (or a compound error pattern) of

weight w , which we want to break. If these w selected integers form this error pattern, reject the current selected integer and go to Step 1.

4. If no integer from the set A can satisfy both Step 2 and Step 3 simultaneously, for a specified number of iterations, then reduce S by 1 and start the search again.
5. Save the current integer as the interleaver output. The process is repeated until all N integers are selected.

In addition to the interleaver design guidelines, it is highly desirable to develop a good search strategy such that the interleaver design algorithms are flexible and computationally efficient. One example is to use a “bidirectional process, boundary reflection” method [10]. It is composed of recursive forward and backward processing of selecting suitable integers to satisfy the design guidelines. The search method is faster and more efficient than other random number generation methods.

7.9 Performance of Turbo Codes with Code Matched Interleavers

The code matched interleavers are designed for a number of turbo codes following the proposed design method. The interleaver size was 1024 bits and S was chosen to be 10. The code bit error rate performance on an AWGN channel is estimated by simulation. In the simulation, iterative decoding with the SOVA algorithm is employed. The maximum number of iterations is 8.

The code performance is compared to random interleavers and S -random interleavers for the same interleaver size. S was chosen to be 15 for S -random interleavers. The code rate is 1/3. The simulation results are shown in Figs. 7.13-7.15.

From the figures it is clear that the code matched interleavers improve the code performance at moderate to high SNR's and lower the bit error rate “error floor” significantly.

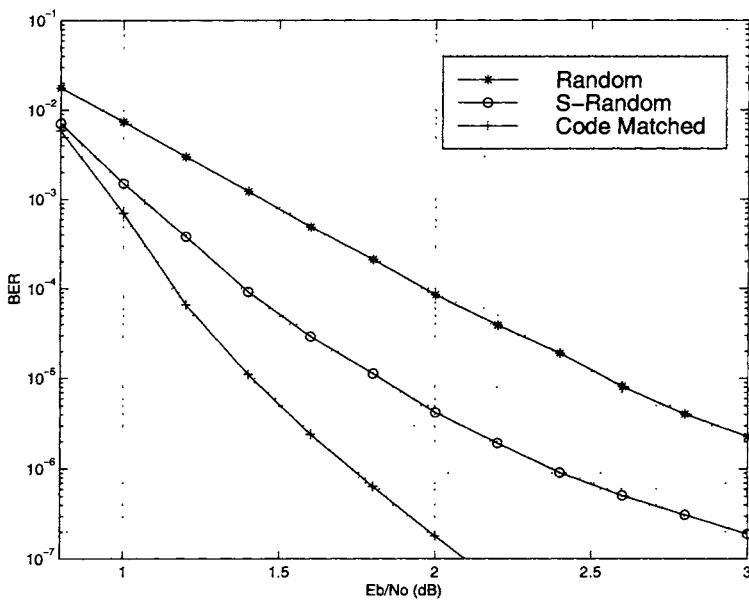


Fig. 7.13: BER performance of the 4-state, rate 1/3, (1, 5/7) turbo code with random, *S*-random and code matched interleavers on an AWGN channel

Comparing the simulation results of the 4-state code in Fig. 7.13, we observe that the code matched interleaver achieves an improvement of 0.4 dB relative to the *S*-random interleaver and 1.1 dB relative to the random interleaver at a BER of 10^{-5} . For the 8-state turbo code in Fig. 7.14, the performance gains achieved by the code matched interleaver are 0.1 dB and 0.6 dB relative to the *S*-random interleaver and random interleaver, respectively, at a BER of 10^{-5} . By comparing the simulation results of the 16-state turbo code in Fig. 7.15, the code matched interleaver outperforms the *S*-random interleaver by about 0.1 dB and the random interleaver by about 0.25 dB at the same BER. An interesting observation in Figs. 7.13-7.15 is that the BER for turbo codes with code matched interleavers decrease rapidly with the increasing E_b/N_0 . There is no “error-floor” observed in the simulations, especially for the turbo codes with large numbers of states.

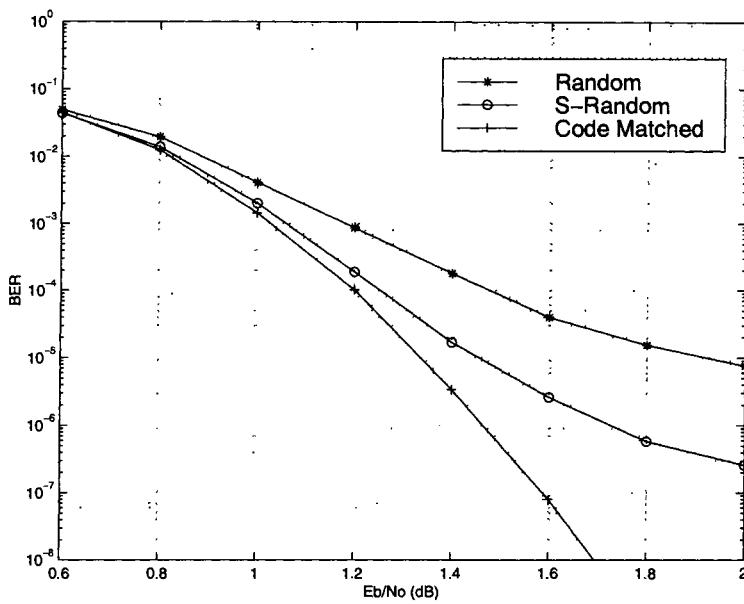


Fig. 7.14: BER performance of the 8-state, rate 1/3, (1, 17/15) turbo code with random, S -random and code matched interleavers on an AWGN channel

7.10 Performance of Turbo Codes with Cyclic Shift Interleavers

In Section 7.6.3, it is pointed out that the convolutional interleavers are equivalent to the S -random interleavers in the sense of their ability to break low weight input patterns. In this section, we consider cyclic shift type convolutional interleavers. Turbo code performance with cyclic shift interleavers is estimated by simulation and compared to the S -random interleavers.

Let a cyclic shift interleaver size be $N = m \times n$, where m and n are the number of rows and columns of the interleaving matrix, respectively, such that $m \leq n$. To generate a cyclic shift interleaver based on (7.14) and (7.15) comparable to an S -random interleaver based on (7.20) and (7.21), we should choose

$$m - 1 = S_1, \quad mB - 1 = S_2 \quad (7.36)$$

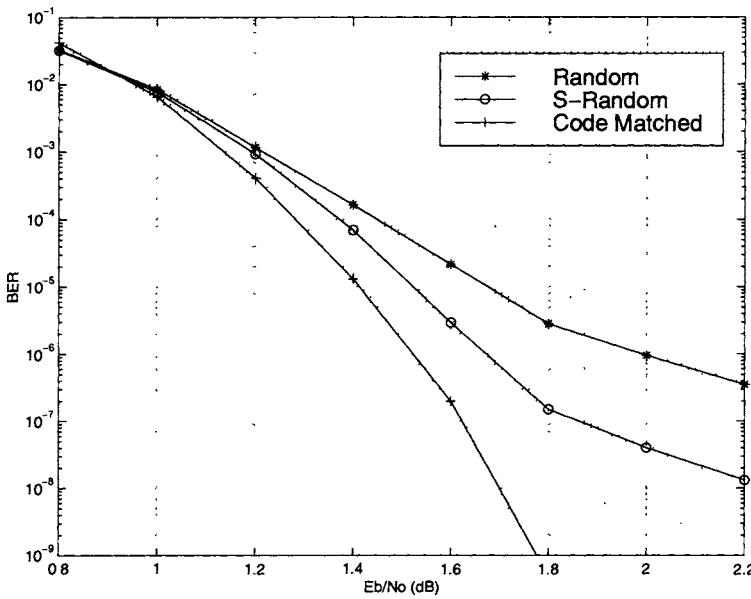


Fig. 7.15: BER performance of the 16-state, rate 1/3, (1, 33/31) turbo code with random, S -random and code matched interleavers on an AWGN channel

where S_1 and S_2 are the parameters of the S -random interleaver. For turbo codes with identical component codes, it is appropriate to set $S_1 = S_2 = S$. Therefore the parameters m and B from (7.36) become

$$m = S + 1, \quad B = 1 \quad (7.37)$$

In a cyclic shift interleaver design, B is an integer such that

$$B \leq n/m = N/m^2 \quad (7.38)$$

Increasing the value of B will result in breaking more low weight input patterns, which on the other hand improves the turbo code performance.

Cyclic shift interleavers are designed for interleaver size 1024. Turbo code performance with the cyclic shift interleavers is evaluated by simulations on AWGN channels. In the simulation, iterative decoding with the SOVA algorithm is employed. The number of iterations is 8.

The performance of a cyclic shift interleaver is compared to an S -random interleaver for the same interleaver size. S was chosen to be 15 for interleaver size 1024. For the cyclic shift interleaver, m was set to $S + 1$ and B was chosen from the set $(1, 2, \dots, \lfloor N/m^2 \rfloor)$.

The simulation results are shown in Fig. 7.16. It can be observed that the turbo code with an S -random interleaver can achieve a better performance relative to a cyclic shift interleaver with $B = 1$. However, increasing the value of B for the cyclic shift interleaver improves the code performance. For interleaver size 1024, the cyclic shift interleaver with $m = 16$ and $B = 2$ outperforms the S -random interleaver with $S = 15$.

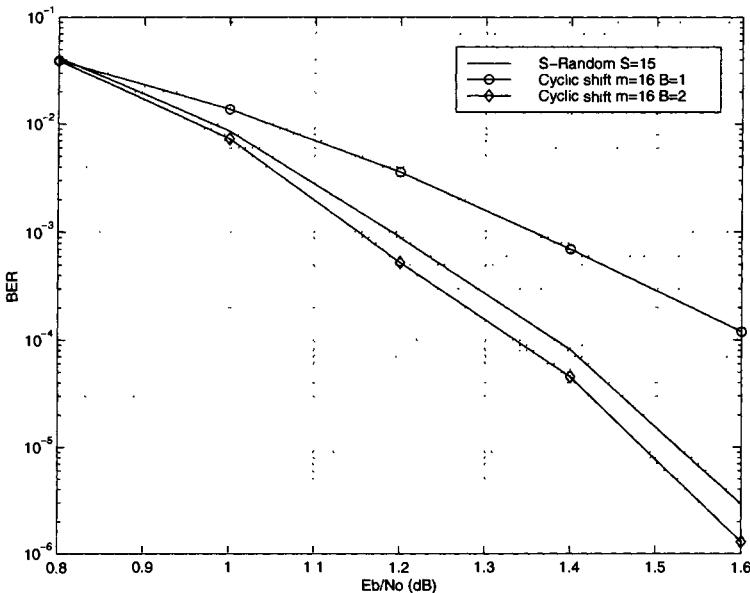


Fig. 7.16: BER performance of the 16-state, rate 1/3, (1, 33/31) turbo code with S -random and cyclic shift interleavers on an AWGN channel

In addition to the performance improvement, the cyclic shift interleavers have the advantages of low design complexity and memory requirement. First, for given parameters, it is easier to construct a cyclic shift interleaver than to search for an S -random interleaver. Secondly, the interleaving vector of an S -random inter-

leaver must be stored in memory for both the turbo encoder and decoder. However, for a cyclic shift interleaver, the interleaved or deinterleaved sequence can be generated from the interleaving matrix based on cyclic shifts. There is no need to store the interleaving vector. Therefore, the cyclic shift interleavers reduce the memory requirement and are easy to implement.

It is worth noting that the cyclic shift interleavers examined in Fig. 7.16 have not been designed for a particular turbo code. For a given turbo code, we could design a code matched interleaver based on the cyclic shifts to further improve the turbo code performance at high SNR's.

Bibliography

- [1] J. G. Proakis, *Digital Communications*, 2nd Ed., McGraw-Hill, New York, 1989.
- [2] L. C. Perez, J. Seghers, and D. J. Costello, Jr., "A distance spectrum interpretation of turbo codes," *IEEE Trans. Inform. Theory*, vol. 42, no. 6, Nov. 1996, pp. 1698-1709.
- [3] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: Turbo-codes," *IEEE Trans. Commun.*, vol. 44, no. 10, Oct. 1996, pp. 1261-1271.
- [4] A. S. Barbulescu and S. S. Pietrobon, "Interleaver design for turbo codes," *Electron. Lett.*, vol. 30, no. 25, Dec. 1994, p. 2107.
- [5] A. S. Barbulescu and S. S. Pietrobon, "Terminating the trellis of turbo-codes in the same state," *Electron. Lett.*, vol. 31, no. 1, Jan. 1995, pp. 22-23.
- [6] A. S. Barbulescu and S. S. Pietrobon, "Interleaver design for three dimensional turbo codes," in *Proc. 1995 IEEE ISIT*, Whistler, BC, Canada, Sep. 1995, p. 37.
- [7] O. Joerssen and H. Meyr, "Terminating the trellis of turbo-codes," *Electron. Lett.*, vol. 30, no. 16, Aug. 1994, pp. 1285-1286.
- [8] S. Benedetto and G. Montorsi, "Unveiling turbo-codes: Some results on parallel concatenated coding schemes," *IEEE Trans. Inform. Theory*, vol. 42, no. 2, Mar. 1996, pp. 409-428.

- [9] J. Yuan, B. Vucetic, and W. Feng, "Combined turbo codes and interleaver design," *IEEE Trans. Commun.*, vol. 47, no. 4, Apr. 1999, pp. 484-487.
- [10] W. Feng, J. Yuan, and B. Vucetic, "A code matched interleaver design for turbo codes," in *Proc. Int. Symposium on Personal, Indoor and Mobile radio Communications (PIMRC'99)*, Osaka, Japan, Sep. 1999, pp. 578-582.
- [11] E. Dunscombe and F. C. Piper, "Optimal interleaving scheme for convolutional coding," *Electron. Lett.*, vol. 25, no. 22, Oct. 1989, pp. 1517-1518.
- [12] S. Dolinar and D. Divsalar, "Weight distributions for turbo codes using random and nonrandom permutations," *TDA Progress Report 42-122*, Jet Propulsion Lab., Aug. 1995, pp. 56-65.
- [13] D. Divsalar and F. Pollara, "Turbo codes for PCS applications," in *Proc. ICC'95*, Seattle, WA, June 1995, pp. 54-59.
- [14] J. L. Ramsey, "Realization of optimum interleavers," *IEEE Trans. Inform. Theory*, vol. 16, no. 3, May 1970, pp. 338-345.
- [15] G. D. Forney, Jr., "Burst-correcting codes for the classic bursty channel," *IEEE Trans. Commun.*, vol. 19, no. 5, Oct. 1971, pp. 772-781.
- [16] E. K. Hall and S. G. Wilson, "Convolutional interleavers for stream-oriented parallel concatenated convolutional codes," in *Proc. 1998 IEEE Int. Symposium on Inform. Theory*, MIT, Cambridge, MA USA, Aug. 1998, p. 33.
- [17] J. D. Andersen, "Interleaver design for turbo coding," in *Proc. Int. Symposium on Turbo Codes and Related Topics*, Brest, France, Sep. 1997, pp. 154-156.
- [18] M. Oberg and P. H. Siegel, "Lowering the error floor for turbo codes," in *Proc. Int. Symposium on Turbo Codes and Related Topics*, Brest, France, Sep. 1997, pp. 204-207.

- [19] J. Hokfelt and T. Maseng, "Methodical interleaver design for turbo codes," in *Proc. Int. Symposium on Turbo Codes and Related Topics*, Brest, France, Sep. 1997, pp. 212-215.
- [20] F. Daneshgaran and M. Mondin, "Design of interleavers for turbo codes based on a cost function," in *Proc. Int. Symposium on Turbo Codes and Related Topics*, Brest, France, Sep. 1997, pp. 255-258.
- [21] O. Y. Takeshita and D. J. Costello, Jr., "New classes of algebraic interleavers for turbo codes," in *Proc. 1998 IEEE Int. Symposium on Inform. Theory*, MIT, Cambridge, MA USA, Aug. 1998, p. 419.
- [22] K. S. Andrews, C. Heegard, and D. Kozen, "Interleaver design methods for turbo codes," in *Proc. 1998 IEEE Int. Symposium on Inform. Theory*, MIT, Cambridge, MA USA, Aug. 1998, p. 420.
- [23] S. Crozier, J. Lodge, P. Guinand, and A. Hunt, "Performance of turbo-codes with relative prime and golden interleaving strategies," in *Proc. Sixth Int. Mobile Satellite Conf.*, Ottawa, June 1999, pp. 268-274.

Chapter 8

Turbo Coding for Fading Channels

8.1 Introduction

In the previous chapters we have discussed the design, performance analysis and iterative decoding of turbo codes for Gaussian channels. It is observed that turbo codes can achieve a remarkably low bit error rate with iterative decoding at a signal-to-noise ratio close to the Shannon capacity limit on AWGN channels.

However, on many real links such as radio, satellite and mobile channels, transmission errors are mainly caused by variations in received signal strength referred as fading. This severely degrades the digital transmission performance and powerful error control coding techniques are needed to reduce the penalty in signal-to-noise ratio.

In this chapter, we consider turbo and serial concatenated code performance on fading channels. Analytical average upper bounds for turbo and serial concatenated code performance based on the union bound and code weight distributions are derived for independent fading channels. For decoding, we consider the iterative MAP and SOVA decoding methods with channel state information. The turbo and serial concatenated code performance over independent and correlated fading channels is discussed.

8.2 Fading Channels

8.2.1 Multipath Propagation

In a cellular mobile radio environment, the surrounding objects, such as houses, building, and trees, act as reflectors of radio waves. These obstacles produce reflected waves with attenuated amplitude and phase. If a modulated signal is transmitted, multiple reflected waves of the transmitted signal will arrive at the receiving antenna from different directions with different propagation delays. These reflected waves are called *multipath waves* [1]. Due to the different arrival angles and times, the multipath waves at the receiver site will have different phases. When they are collected by the receiver antenna at any point in space, they may combine either in a constructive or a destructive way, depending on the random phases. The sum of these multipath components forms a spatially varying standing wave field. The mobile unit moving through the multipath field will receive a signal which can vary widely in amplitude and phase. When the mobile unit is stationary, the amplitude variations in the received signal are due to the movement of surrounding objects in the radio channel. The amplitude fluctuation of the received signal is called *signal fading*. It is caused by the time-variant multipath characteristics of the channel.

8.2.2 Doppler Shift

Due to the relative motion between the transmitter and the receiver, each multipath wave is subject to a shift in frequency. The frequency shift of the received signal caused by the relative motion is called the *Doppler shift*. It is proportional to the speed of the mobile unit. Consider a situation when only a single tone of frequency f_c is transmitted and a received signal consists of only one wave coming at an incident angle θ with respect to the direction of the vehicle motion. The Doppler shift of the received signal, denoted by f_d , is given by

$$f_d = \frac{v f_c}{c} \cos \theta \quad (8.1)$$

where v is the vehicle speed and c is the speed of the light. The Doppler shift effect in multipath propagation environment spreads the bandwidth of the multipath waves into the range of $f_c \pm f_{d_{\max}}$, where $f_{d_{\max}}$ is the maximum Doppler shift and it is given by

$$f_{d_{\max}} = \frac{vf_c}{c} \quad (8.2)$$

The maximum Doppler shift is also referred as the maximum *fade rate*. As a result, a single tone transmitted gives rise to a received signal with a spectrum of nonzero width. This phenomenon is called *frequency dispersion* of the channel.

8.3 Statistical Models for Fading Channels

Because of multiplicity of factors involved in propagation in a cellular mobile environment, it is necessary to apply statistical techniques to describe signal variations. In this section we will introduce Rayleigh and Rician fading models to describe signal variation in a multipath environment.

8.3.1 Rayleigh Fading

In a common land mobile radio channel, we may assume the direct wave is obstructed and the mobile unit receives only reflected waves. When the number of reflected waves is large, according to the central limit theorem, two quadrature components of the channel impulse response will be uncorrelated Gaussian random processes with a zero mean and variance σ_s^2 . As a result, the envelope of the channel impulse response at any time instant undergoes a Rayleigh probability distribution and the phase of the channel response obeys a uniform distribution between $-\pi$ and π . The probability density function (pdf) of the Rayleigh distribution is given by

$$p(a) = \begin{cases} \frac{a}{\sigma_s^2} \cdot e^{-a^2/\sigma_s^2} & a \geq 0 \\ 0 & a < 0 \end{cases} \quad (8.3)$$

The mean value, denoted by m_a , and variance, denoted by σ_a^2 , of the Rayleigh distributed random variable are given by

$$\begin{aligned} m_a &= \sqrt{\frac{\pi}{2}} \cdot \sigma_s = 1.2533\sigma_s \\ \sigma_a^2 &= \left(2 - \frac{\pi}{2}\right) \sigma_s^2 = 0.4292\sigma_s^2 \end{aligned} \quad (8.4)$$

If the probability density function in (8.3) is normalized so that the average signal power ($E[a^2]$) is unity, then the Rayleigh distribution will become

$$p(a) = \begin{cases} 2ae^{-a^2} & a \geq 0 \\ 0 & a < 0 \end{cases} \quad (8.5)$$

The mean value and the variance will be

$$\begin{aligned} m_a &= 0.8862 \\ \sigma_a^2 &= 0.2146 \end{aligned} \quad (8.6)$$

The pdf for Rayleigh distribution is shown in Fig. 8.1.

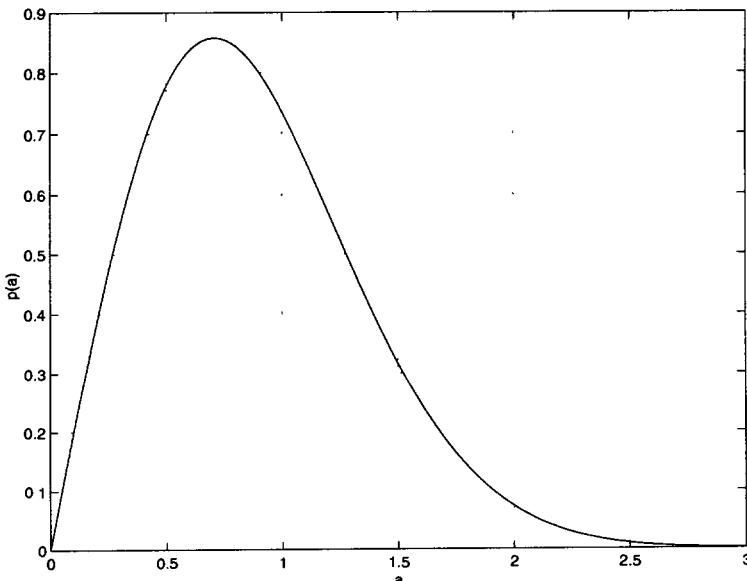


Fig. 8.1: The pdf of Rayleigh distribution

In fading channels, the received signal experiences a form of frequency spreading and is band-limited between $f_c \pm f_d$. Assuming an omnidirectional antenna with waves arriving in the horizontal plane, a large number of reflected waves and uniform power received for their incident angles, the power spectral density of the faded amplitude due to the Doppler shift, denoted by $|P(f)|$, is given by

$$|P(f)| = \begin{cases} \frac{1}{2\pi\sqrt{f_d^2 - f^2}} & \text{if } |f| \leq |f_d| \\ 0 & \text{otherwise} \end{cases} \quad (8.7)$$

where f is the frequency and f_d is the fade rate. The value of $f_d T_s$ is the fade rate normalized by the symbol rate. It serves as a measure of the channel memory. For correlated fading channels this parameter is in the range $0 < f_d T_s < 1$, indicating finite channel memory. The autocorrelation function of the fading process is given by

$$R(\tau) = J_0(2\pi f_d \tau) \quad (8.8)$$

where $J_0(\cdot)$ is the zero-order Bessel function of the first kind.

8.3.2 Rician Fading

In some propagation scenarios, such as satellite or microcellular mobile radio channels, there are essentially no obstacles on the line-of-sight path. The received signal consists of a direct wave and a number of reflected waves. The direct wave is a stationary nonfading signal with a constant amplitude. The reflected waves are independent random multipath signals. Their sum is called the *scattered component* of the received signal. When the number of reflected waves is large, the scattered component can be characterized as a Gaussian random process with a zero mean and variance σ_s^2 . The envelope of the scattered component will have a Rayleigh probability distribution.

The sum of a constant direct signal and a Rayleigh distributed scattered signal results in a signal with Rician envelope distribution. The pdf of Rician distribution is given by

$$p(a) = \begin{cases} \frac{a}{\sigma_s^2} e^{-\frac{(a^2 + D^2)}{2\sigma_s^2}} I_0\left(\frac{aD}{\sigma_s^2}\right) & a \geq 0 \\ 0 & a < 0 \end{cases} \quad (8.9)$$

where D^2 is the direct signal power and $I_0(\cdot)$ is the modified Bessel function of the first kind and zero-order.

By assuming that the total average signal power is normalized to unity, the pdf in (8.9) becomes

$$p(a) = \begin{cases} 2a(1+K)e^{-K-(1+K)a^2} I_0\left(2a\sqrt{K(K+1)}\right) & a \geq 0 \\ 0 & a < 0 \end{cases}$$

where K is the Rician factor denoting the power ratio of the direct and the scattered signal components. The Rician factor is given by

$$K = \frac{D^2}{2\sigma_s^2} \quad (8.10)$$

The mean and variance of the Rician distributed random variable are given by

$$\begin{aligned} m_a &= \frac{1}{2}\sqrt{\frac{\pi}{1+K}}e^{-\frac{K}{2}\left[(1+K)I_0\left(\frac{K}{2}\right)+KI_1\left(\frac{K}{2}\right)\right]} \\ \sigma_a^2 &= 1 - m_r^2 \end{aligned} \quad (8.11)$$

where $I_1(\cdot)$ is the first order modified Bessel function of the first kind. Small values of K indicate a severely faded channel. For $K = 0$, there is no direct signal component and the Rician pdf becomes a Rayleigh pdf. On the other hand, large values of K indicate a slightly faded channel. For K approaching infinity, there is no fading at all resulting in an AWGN channel. The Rician distributions with various K are shown in Fig. 8.2.

8.4 Capacity of Fading Channels

The channel capacity of AWGN channels was considered in Chapter 1. In this section, we consider the channel capacity of the fading channels as derived in [6]. The channel capacity can be used to measure the turbo code performance on fading channels.

In the analysis, we will use the Rayleigh fading model, as the Rayleigh model represents a worst fading case while offering analytical convenience. Let us assume that the channel is modelled by

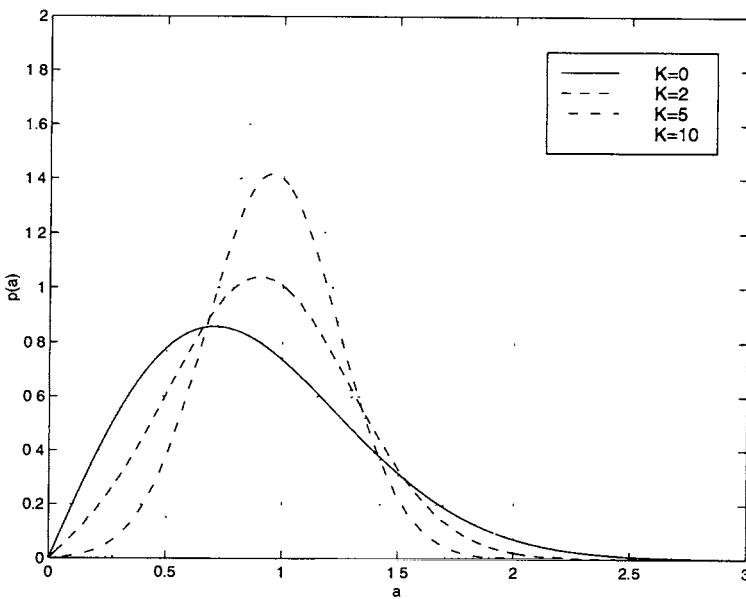


Fig. 8.2: The pdf of Rician distributions with various K

independent slow fading. Slow means that the fading bandwidth is small compared to the signal bandwidth so that the receiver will be able to track the phase variations, and hence compensate for it by using a reference pilot tone or a phase-locked loop technique. This will enable coherent detection. A slow fading process can be modelled as a constant random variable during each symbol interval. Furthermore, we assume that the channel interleaving depth is sufficiently large, resulting in independent and identically distributed (i.i.d.) fading amplitudes.

For BPSK modulation, let x_t be the transmitted signal at time t . The received signal at time t is r_t . The discrete-time channel model is given by

$$r_t = a_t x_t + n_t \quad (8.12)$$

where a_t is a random variable which represents the channel fading variation, and n_t is an additive white Gaussian noise with single sided power spectral density N_0 . The variance of the Gaussian noise is $\sigma^2 = N_0/2$. In this model, the envelope amplitude of the

fading attenuation a_t is a Rayleigh random variable.

Channel capacity is defined as the largest average mutual information between the channel input and output over the set of input probability assignments. For fading channels, it is obvious that the channel capacity depends on the knowledge of the values of the fading attenuation, called channel state information (CSI), available at the receiver. Here we consider the two extreme cases of perfect CSI and no CSI.

By assuming that the fading attenuation can be perfectly recovered at the receiver, the channel capacity is given by [6]

$$\begin{aligned} C = & - \int_a \int_r p(a) p(r | x = 1, a) \\ & \cdot \log_2 \left[\frac{1}{2} (1 + \Psi(r, a)) \right] dr da \end{aligned} \quad (8.13)$$

where $p(a)$ is the normalized Rayleigh pdf and $p(r | x = 1, a)$ is a Gaussian pdf with the mean of a and variance of σ^2 . The term $\Psi(r, a)$ is given by

$$\Psi(r, a) = \exp \left(-\frac{2}{\sigma^2} \cdot a \cdot r \right) \quad (8.14)$$

If CSI is absent at the receiver, the channel capacity is give by [6]

$$\begin{aligned} C = & - \int_a \int_r p(a) p(r | x = 1, a) \\ & \cdot \log_2 \left[\frac{1}{2} (1 + \Phi(r)) \right] dr da \end{aligned} \quad (8.15)$$

where

$$\Phi(r) = \frac{\int_a p(a) \cdot p(r | x = -1, a) da}{\int_a p(a) \cdot p(r | x = +1, a) da} \quad (8.16)$$

The channel capacity of (8.13) and (8.15) can be calculated by using numerical integration. The results from [6] are plotted in Fig. 8.3. From the figure we can see that the penalty in E_s/N_0 is in excess of $0.8 \sim 1$ dB when no CSI is utilized relative to the case with perfect CSI on Rayleigh fading channels for codes of rate $1/4$ to $1/2$.

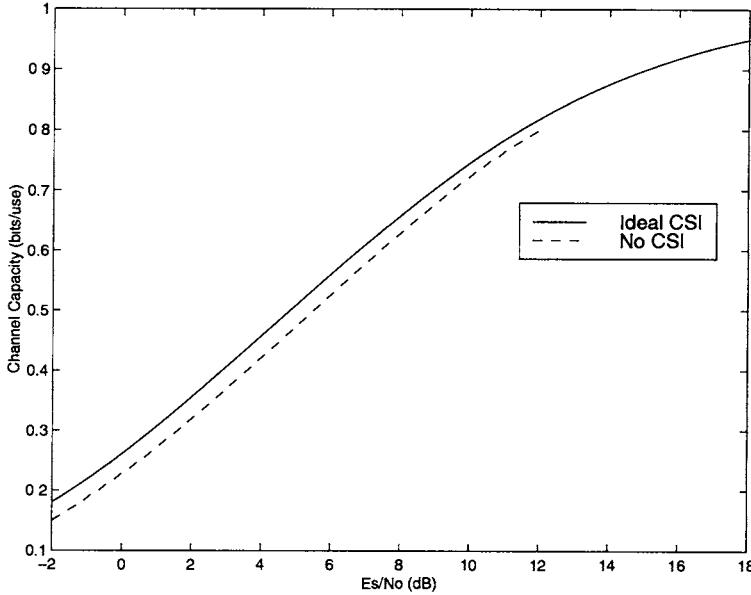


Fig. 8.3: Capacity of independent Rayleigh fading channels with coherent BPSK signalling

From the channel coding theorem, we can derive the inequality

$$R \leq \frac{C}{1 - H_b(e)} \quad (8.17)$$

where R is the code rate and $H_b(e)$ is the entropy of a binary source with probability $P_b(e)$. It is given by

$$H_b(e) = -P_b(e) \log P_b(e) - (1 - P_b(e)) \log (1 - P_b(e)) \quad (8.18)$$

Eqs. (8.17), (8.13) and (8.15) determine the channel capacity bounds for fading channels. Due to the complexity of the channel capacity expressions in (8.13) and (8.15), it is not possible to compute analytically the pairs $(P_b(e), E_b/N_0)$ satisfying (8.17) with equality. However, by letting the code rate equal to the channel capacity, we can obtain the smallest E_b/N_0 such that arbitrarily small error performance is achievable. For codes with rates of $1/2$ and $1/3$, the capacity limits for BPSK signalling on independent Rayleigh

fading channels are given in Table 8.1. These values can be used to indicate the channel capacity boundary of the Rayleigh fading channels.

Table 8.1: Channel capacity limits for coherent BPSK

Code Rate	Independent Rayleigh Fading		White Gaussian
	CSI	no CSI	
$R=1/2$	$E_b/N_0=1.8 \text{ dB}$	$E_b/N_0=2.6 \text{ dB}$	$E_b/N_0=0.2 \text{ dB}$
$R=1/3$	$E_b/N_0=0.5 \text{ dB}$	$E_b/N_0=1.4 \text{ dB}$	$E_b/N_0=-0.5 \text{ dB}$

8.5 Performance Upper Bounds on Fading Channels

The coded system structure is shown in Fig. 8.4. The channel encoder may be a turbo encoder or a serial concatenated convolutional encoder. The bit error probability of the coded system can be upper-bounded by using a union bound that sums contributions from all the codewords with various Hamming weights. As a turbo or serial concatenated code can be represented by an equivalent linear block code if the trellis termination is used to drive the component encoders to the all-zero state, the union upper bound for a block code can be applied to a concatenated code.

For example, a rate $1/3$ turbo code with interleaver size N and memory order ν is equivalent to an (n, k) block code, where $n = 3(N + \nu)$, $k = N$. In this section, we derive the upper bounds on the pairwise error probability for block codes on an independent Rician fading channel. Subsequently, we come up with the average upper bounds on the bit error probability for turbo and serial concatenated codes on a fading channel. In the analysis, we assume that the channel interleaving depth is sufficiently large, resulting in i.i.d. fading amplitudes.

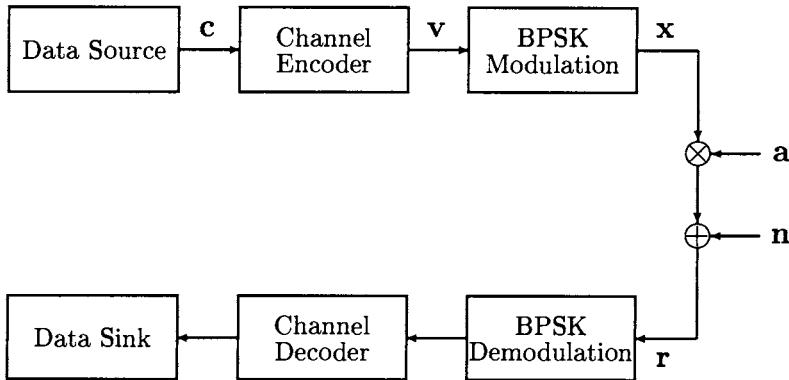


Fig. 8.4: Coded system block diagram

8.5.1 Upper Bounds on the Pairwise Error Probability

A. Decoding with Ideal CSI

Let a coded sequence be $\mathbf{v}_n = (v_1, v_2, \dots, v_n)$, where n is the code length. This sequence is modulated by a BPSK modulator. The modulated sequence $\mathbf{x}_n = (x_1, x_2, \dots, x_n)$ is transmitted over the fading channel. At the receiver, the received sequence $\mathbf{r}_n = (r_1, r_2, \dots, r_n)$ is observed. Furthermore, we assume that the fading attenuation $\mathbf{a}_n = (a_1, a_2, \dots, a_n)$ can be perfectly recovered at the receiver. The decoder then performs soft decoding using the modified Euclidean distance metric

$$m(\mathbf{r}_n, \mathbf{x}_n; \mathbf{a}_n) = \sum_{i=1}^n |r_i - a_i x_i|^2 \quad (8.19)$$

The pairwise error probability $P(\mathbf{x}_n, \hat{\mathbf{x}}_n)$ is the probability that the decoder chooses as its estimate the sequence $\hat{\mathbf{x}}_n = (\hat{x}_1, \hat{x}_2, \dots, \hat{x}_n)$ when the transmitted sequence was in fact $\mathbf{x}_n = (x_1, x_2, \dots, x_n)$. This occurs if

$$m(\mathbf{r}_n, \mathbf{x}_n; \mathbf{a}_n) \geq m(\mathbf{r}_n, \hat{\mathbf{x}}_n; \mathbf{a}_n) \quad (8.20)$$

or equivalently

$$\sum_{i=1}^n |r_i - a_i x_i|^2 \geq \sum_{i=1}^n |r_i - a_i \hat{x}_i|^2 \quad (8.21)$$

For a given realization of the fading variable \mathbf{a}_n , the conditional pairwise error probability is given by

$$\begin{aligned} P(\mathbf{x}_n, \hat{\mathbf{x}}_n | \mathbf{a}_n) &= Pr \left\{ \sum_{i=1}^n |r_i - a_i x_i|^2 \geq \sum_{i=1}^n |r_i - a_i \hat{x}_i|^2 \right\} \\ &= Pr \left\{ \sum_{i=1}^n 2a_i n_i (\hat{x}_i - x_i) \geq \sum_{i=1}^n |a_i (\hat{x}_i - x_i)|^2 \right\} \quad (8.22) \end{aligned}$$

Let us assume that phase recovery is ideal. Then a_i , $i = 1, 2, \dots, n$, after the phase recovery is real. The term on the right hand side of the inequality in (8.22) can be simplified as

$$\sum_{i=1}^n |a_i (\hat{x}_i - x_i)|^2 = 4 \sum_{k=1}^d a_{i_k}^2 \quad (8.23)$$

where a_{i_k} , $k = 1, 2, \dots, d$, represents the value of the fading amplitude in the i_k th bit interval in which the two sequences \mathbf{x}_n and $\hat{\mathbf{x}}_n$ are different, and d is the symbol Hamming distance between the two sequences \mathbf{x}_n and $\hat{\mathbf{x}}_n$. The symbol Hamming distance between two non-binary sequences is defined as the number of positions in which they differ. For a given realization of the fading variable \mathbf{a}_n , the term on the right hand side of the inequality in (8.22) is a constant which depends on \mathbf{a}_n .

We introduce the random variable

$$X = \sum_{i=1}^n 2a_i n_i (\hat{x}_i - x_i) \quad (8.24)$$

Since n_i is a zero mean Gaussian noise with variance σ^2 , the random variable X is a Gaussian variable with a zero mean and variance $\sigma_X^2 = 16\sigma^2 \sum_{i=k}^d a_{i_k}^2$. Therefore, the pairwise probability conditioned on \mathbf{a}_n is

$$P(\mathbf{x}_n, \hat{\mathbf{x}}_n | \mathbf{a}_n) = Q \left(\frac{4 \sum_{k=1}^d a_{i_k}^2}{\sigma_X} \right)$$

$$= Q \left(\sqrt{2R \frac{E_b}{N_0} \sum_{k=1}^d a_{i_k}^2} \right) \quad (8.25)$$

where $R = k/n$ is the code rate, $\frac{E_b}{N_0}$ is the signal-to-noise ratio per bit, and $Q(x)$ is the complementary error function defined by

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-\frac{t^2}{2}} dt \quad (8.26)$$

Introducing the inequality

$$Q(x) \leq \frac{1}{2} e^{-\frac{x^2}{2}}, x \gg 1 \quad (8.27)$$

in (8.25) we obtain

$$P(\mathbf{x}_n, \hat{\mathbf{x}}_n | \mathbf{a}_n) \leq \frac{1}{2} \exp \left(-R \frac{E_b}{N_0} \sum_{k=1}^d a_{i_k}^2 \right) \quad (8.28)$$

The pairwise error probability $P(\mathbf{x}_n, \hat{\mathbf{x}}_n)$ can then be determined by averaging over the random fading amplitudes $a_{i_k}, k = 1, 2, \dots, d$

$$\begin{aligned} P(\mathbf{x}_n, \hat{\mathbf{x}}_n) &= E[P(\mathbf{x}_n, \hat{\mathbf{x}}_n | \mathbf{a}_n)] \\ &\leq \frac{1}{2} E \left[\exp \left(-R \frac{E_b}{N_0} \sum_{k=1}^d a_{i_k}^2 \right) \right] \end{aligned} \quad (8.29)$$

where the expectation operator $E[\cdot]$ represents the joint expectation with respect to the components $a_{i_k}, k = 1, 2, \dots, d$. For the fully interleaved fading channels, the fading amplitudes $a_{i_k}, k = 1, 2, \dots, d$, are independent from interval-to-interval. Furthermore, for the Rician distributed random variable a , making use of the result

$$E \left[\exp(-wa^2) \right] = \frac{1+K}{1+K+w} \exp \left(-\frac{wK}{1+K+w} \right), \quad w > 0 \quad (8.30)$$

the pairwise error probability $P(\mathbf{x}_n, \hat{\mathbf{x}}_n)$ can be upper bounded by

$$P(\mathbf{x}_n, \hat{\mathbf{x}}_n) \leq \frac{1}{2} \left(\frac{1+K}{1+K+R \frac{E_b}{N_0}} \right)^d \exp \left(-\frac{KdR \frac{E_b}{N_0}}{1+K+R \frac{E_b}{N_0}} \right) \quad (8.31)$$

B. Decoding without CSI

In deriving the above result we assumed that the value of fading amplitude $\mathbf{a}_n = (a_1, a_2, \dots, a_n)$ can be recovered perfectly. However, in some systems it may not be practical to attempt CSI recovery, so the pairwise error bound in the absence of CSI becomes relevant.

When no channel state information is available at the receiver, the decoder performs soft decoding using the standard Euclidean distance metric

$$m(\mathbf{r}_n, \mathbf{x}_n) = \sum_{i=1}^n |r_i - x_i|^2 \quad (8.32)$$

The pairwise error probability $P(\mathbf{x}_n, \hat{\mathbf{x}}_n)$ is given by

$$\begin{aligned} P(\mathbf{x}_n, \hat{\mathbf{x}}_n) &= Pr \left\{ \sum_{i=1}^n |r_i - x_i|^2 \geq \sum_{i=1}^n |r_i - \hat{x}_i|^2 \right\} \\ &= Pr \left\{ \sum_{i=1}^n 2n_i(\hat{x}_i - x_i) \geq 4 \sum_{k=1}^d a_{ik} \right\} \end{aligned} \quad (8.33)$$

As before we will introduce a random variable X equal to the term on the left hand side of the inequality in (8.33). For a given realization of the fading variable \mathbf{a}_n , X is a Gaussian variable with a zero mean and variance $\sigma_X^2 = 16d\sigma^2$. The term on the right hand side is a constant which depends on \mathbf{a}_n . Therefore, the conditional pairwise error probability can be expressed as

$$\begin{aligned} P(\mathbf{x}_n, \hat{\mathbf{x}}_n | \mathbf{a}_n) &= Q \left(\frac{4 \sum_{k=1}^d a_{ik}}{\sigma_X} \right) \\ &= Q \left(\sqrt{\frac{2R}{d} \frac{E_b}{N_0}} \sum_{k=1}^d a_{ik} \right) \end{aligned} \quad (8.34)$$

Using the bound of (8.27), the conditional pairwise error probability can be upper-bounded by

$$P(\mathbf{x}_n, \hat{\mathbf{x}}_n | \mathbf{a}_n) \leq \frac{1}{2} \exp \left(-\frac{R}{d} \frac{E_b}{N_0} \left(\sum_{k=1}^d a_{ik} \right)^2 \right) \quad (8.35)$$

The pairwise error probability $P(\mathbf{x}_n, \hat{\mathbf{x}}_n)$ can then be determined by averaging over the random fading amplitudes a_{ik} , $k = 1, 2, \dots, d$,

and is given by

$$\begin{aligned} P(\mathbf{x}_n, \hat{\mathbf{x}}_n) &= E[P(\mathbf{x}_n, \hat{\mathbf{x}}_n | \mathbf{a}_n)] \\ &\leq \frac{1}{2} E \left[\exp \left(-\frac{R}{d} \frac{E_b}{N_0} \left(\sum_{k=1}^d a_{i_k} \right)^2 \right) \right] \end{aligned} \quad (8.36)$$

In this analysis, we will introduce the random variable

$$A = \sum_{k=1}^d a_{i_k} \quad (8.37)$$

As the fading amplitudes $\mathbf{a}_n = (a_1, a_2, \dots, a_n)$ are independent identically distributed random variables, for large Hamming distance d (larger than 5), according to the central limit theorem, the random variable A is approximately Gaussian with the mean

$$m_A = dm_a \quad (8.38)$$

and the variance

$$\sigma_A^2 = d\sigma_a^2 \quad (8.39)$$

where m_a and σ_a^2 are respectively the mean and variance of the Rician random variable a and they are given by (8.11). Noting that

$$E \left[\exp(wA^2) \right] = \frac{1}{\sqrt{1 - 2w\sigma_A^2}} \exp \left(\frac{wm_A^2}{1 - 2w\sigma_A^2} \right), \quad \text{if } \operatorname{Re}\{w\} < \frac{1}{2\sigma_A^2}$$

we get for the pairwise error probability upper-bound from (8.36)

$$P(\mathbf{x}_n, \hat{\mathbf{x}}_n) \leq \frac{1}{2} \frac{1}{\sqrt{1 + 2R \frac{E_b}{N_0} \sigma_a^2}} \exp \left(-\frac{dR \frac{E_b}{N_0} m_a^2}{1 + 2R \frac{E_b}{N_0} \sigma_a^2} \right) \quad (8.40)$$

For the turbo codes d is typically very large, so the above bound holds in most cases.

8.5.2 Average Upper Bounds on the Bit Error Probability

For an (n, k) linear systematic block code, the input-redundancy weight enumerating function can be represented as

$$A(W, Z) = \sum_{w,z} A_{w,z} W^w Z^z \quad (8.41)$$

where $A_{w,z}$ is the number of the codewords of the block code with the input information weight w and the parity check weight z . The overall Hamming weight of the codeword is $d = w + z$.

The error coefficient which determines the contribution of the codewords with the same weight d to the bit error probability, denoted by B_d , is given by

$$B_d = \sum_{d=w+z} \frac{w}{k} A_{w,z} \quad (8.42)$$

The error coefficient B_d is the average number of bit errors caused by the transitions between the all-zero codeword and codewords of weight d ($d \geq d_{\min}$), where d_{\min} is the code minimum distance. The bit error probability of the code decoded by a maximum-likelihood algorithm over a fading channel can be upper-bounded by

$$P_b(e) \leq \sum_{d=d_{\min}} B_d P_d(\mathbf{x}_n, \hat{\mathbf{x}}_n) \quad (8.43)$$

where $P_d(\mathbf{x}_n, \hat{\mathbf{x}}_n)$ is the pairwise error probability when the Hamming distance between the two sequences \mathbf{x}_n and $\hat{\mathbf{x}}_n$ is d .

Therefore, the bit error probability upper-bound on an independent Rician fading channel is given by

$$P_b(e) \leq \sum_{d=d_{\min}} \frac{1}{2} B_d \left(\frac{1+K}{1+K+R \frac{E_b}{N_0}} \right)^d \exp \left(-\frac{KdR \frac{E_b}{N_0}}{1+K+R \frac{E_b}{N_0}} \right)$$

when ideal channel state information is available, and equal to

$$P_b(e) \leq \sum_{d=d_{\min}} \frac{1}{2} B_d \frac{1}{\sqrt{1+2R \frac{E_b}{N_0} \sigma_a^2}} \exp \left(-\frac{dR \frac{E_b}{N_0} m_a^2}{1+2R \frac{E_b}{N_0} \sigma_a^2} \right)$$

when there is no channel state information.

The above bounds can be evaluated as functions of $\frac{E_b}{N_0}$ and K if the code distance spectrum is known. Here, we assume the uniform interleavers are used in the concatenated encoders so that the error coefficient can be obtained by averaging over all possible input sequence permutations [7] [8]. For a turbo code, the error coefficient can be written as [7]

$$B_d = \sum_w \sum_{z_1} \sum_{z_2} \frac{w \cdot w! \cdot (N-w)!}{N \cdot N!} A_{w,z_1}^c \cdot A_{w,z_2}^c \quad (8.44)$$

where $A_{w,z}^c$ is the input-redundancy weight enumerating coefficient of the equivalent block code for the component code, $d = w+z_1+z_2$, w is the information weight, z_1 and z_2 are the parity check weights of the first and the second component codes, respectively. For a serial code, the error coefficient is given by [12]

$$B_d = \sum_w \sum_l \frac{w \cdot l! \cdot (2N-l)!}{N \cdot (2N)!} A_{w,l}^{co} \cdot A_{l,d}^{ci} \quad (8.45)$$

where $A_{w,l}^{co}$ and $A_{l,d}^{ci}$ are the corresponding input-output weight enumerating coefficients for the equivalent block codes of the outer and inner code, respectively, w is input information weight of the outer code, l is the weight of the output sequence for the outer code and d is the weight of the output sequence for the inner code.

In the evaluation, we consider a turbo and a serial code with the same code rate $R = 1/3$ and the memory order $\nu = 2$. For fair comparison of the turbo and the serial code, the interleaver size should be chosen in such a way that the decoding delay due to the interleaver is the same. In other words, the input information block sizes for both codes should be the same. We chose an information size of $N = 100$ in the performance evaluation. For the turbo code, the generator matrix of the component codes is

$$G(D) = \begin{bmatrix} 1 & \frac{1+D^2}{1+D+D^2} \end{bmatrix}$$

The bit error probability upper-bounds for the turbo code with variable K are illustrated in Fig. 8.5 for the case of ideal channel

state information and in Fig. 8.6 for the case of no channel state information. For the serial code, the generator matrix of the outer code is

$$G_o(D) = \begin{bmatrix} 1 + D + D^2 & 1 + D^2 \end{bmatrix}$$

The generator matrix of the inner code is

$$G_i(D) = \begin{bmatrix} 1 & 0 & \frac{1+D^2}{1+D+D^2} \\ 0 & 1 & \frac{1+D}{1+D+D^2} \end{bmatrix}$$

The bit error probability upper-bounds for the serial code with variable K are illustrated in Fig. 8.7 for the case of ideal channel state information and in Fig. 8.8 for the case of no channel state information. From the figures, we can see that for the Rayleigh fading channel ($K = 0$), the error performance can be improved by approximately 1.5 dB at the bit error rate of 10^{-5} when ideal channel state information is utilized relative to the case with no channel state information. This improvement decreases as K is getting larger. When $K \rightarrow \infty$, the gain of channel state information disappears.

In order to compare the performance of the turbo and serial codes on Rayleigh fading channels, we show the distance spectra and the bit error probability upper bounds for both codes in Figs. 8.9 and 8.10, respectively.

From Fig. 8.9 we can see that the serial code has smaller error coefficients than the turbo code at low to medium Hamming distances. Since these distance spectral lines dominate the code error performance in the region of high SNR's [9], the serial code can achieve better performance than the turbo code on fading channels at high SNR's as illustrated in Fig. 8.10. It may also be observed from Fig. 8.9 that for medium distance values, the error coefficients of both codes are almost the same, which will result in almost the same error performance at low SNR's, as the code performance in this region is determined by the medium distance spectral lines [9].

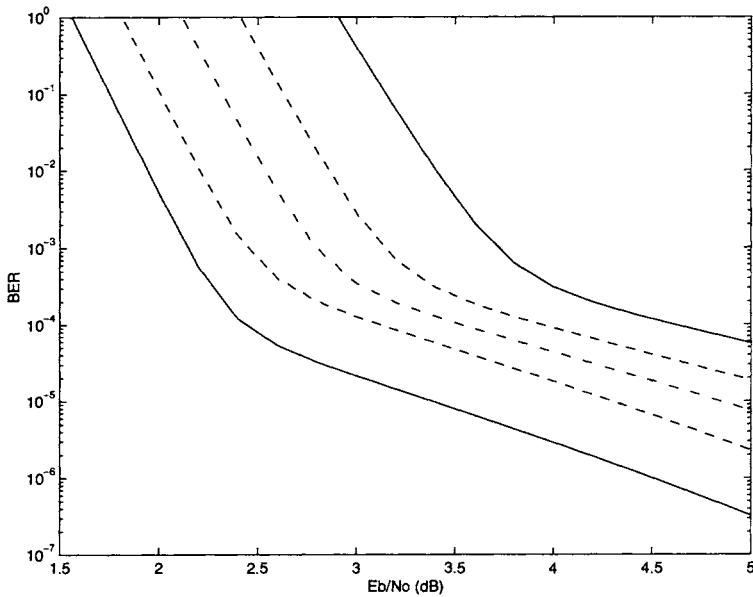


Fig. 8.5: Bit error probability upper bound for the 4 state, rate 1/3 turbo code with interleaver size 100 on independent Rician fading channels with ideal channel state information. The curves are for Rician channels with $K=0, 2, 5, 50$, starting from the top, with the bottom one referring to an AWGN channel.

8.6 Iterative Decoding on Fading Channels

On AWGN channels, the turbo decoding is performed by a sub-optimal iterative algorithm. The decoder consists of two identical decoders of the component codes separated by an interleaver. The component decoders are based on a maximum a posteriori algorithm or a soft output Viterbi algorithm generating a weighted soft estimate of the input sequence [2]-[5].

In the turbo decoding for AWGN channels, a branch metric based on the standard Euclidean distance between the received and transmitted signal sequences is used. This metric is optimum for Gaussian channels. On fading channels, the decoding metric should optimize the conditional probability $p(\mathbf{r}_n|\mathbf{x}_n)$. Since this condition

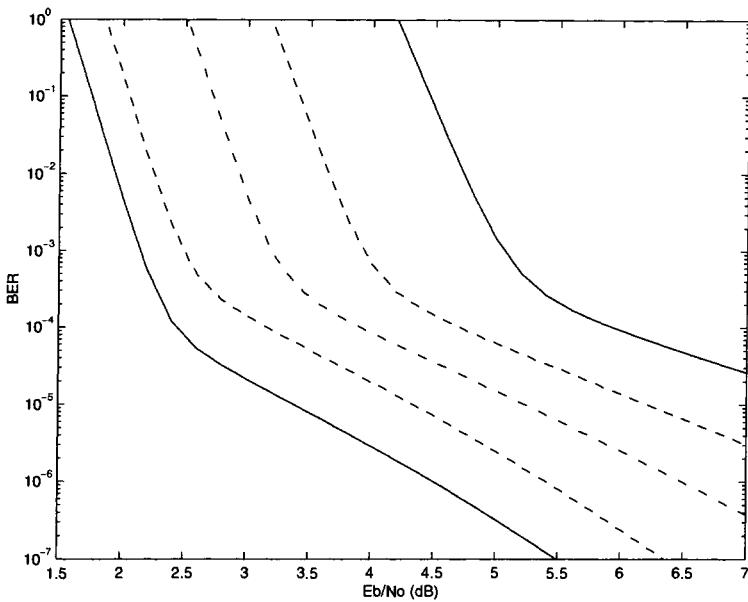


Fig. 8.6: Bit error probability upper bound for the 4 state, rate 1/3 turbo code with interleaver size 100 on independent Rician fading channels without channel state information. The curves are for Rician channels with $K=0, 2, 5, 50$, starting from the top, with the bottom one referring to an AWGN channel.

gives a fairly complicated metric computation, for practical reasons, the AWGN metric is used as the decoding metric when no channel state information is available at the receiver. So the MAP and SOVA iterative decoding algorithms derived for AWGN channels will be applied to decode turbo codes for fading channels without channel state information.

If ideal channel state information is available at the decoder, the branch metric is modified by replacing the transmitted signal x_i by $a_i x_i$. This metric is not the optimum one, but it is a reasonable trade-off between the complexity and performance, as the optimum metric is very complex.

Based on this branch metric, the log-likelihood ratio and extrinsic information for MAP and SOVA decoding methods have been modified. Note that the decoding methods described here are for turbo codes. However, the algorithms can be directly extended to

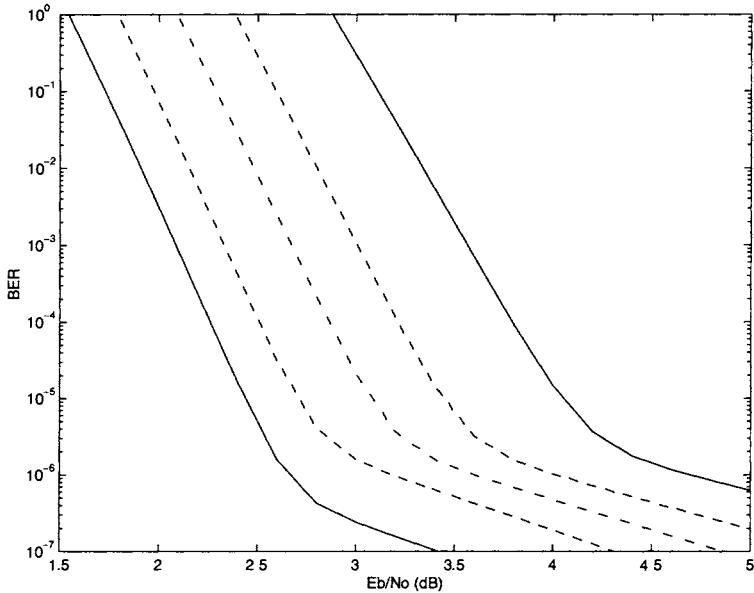


Fig. 8.7: Bit error probability upper bound for the 4 state, rate 1/3 serial code with information size 100 on independent Rician fading channels with ideal channel state information. The curves are for Rician channels with $K=0, 2, 5, 50$, starting from the top, with the bottom one referring to an AWGN channel.

serial concatenated codes.

Without loss of generality, we assume that the bit sequence $\mathbf{c} = (c_1, c_2, \dots, c_N)$ is encoded by an $(n, 1, \nu)$ RSC encoder. The code trellis has a total number of $M_s = 2^\nu$ distinct states, indexed by the integer l , $l = 0, 1, \dots, M_s - 1$. The RSC encoder output sequence is $\mathbf{v} = (\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_N)$, where $\mathbf{v}_i = (v_{i,0}, v_{i,1}, \dots, v_{i,n-1})$ are defined by n encoder generators. The corresponding modulated sequence \mathbf{x} and received sequence \mathbf{r} are given by

$$\begin{aligned}\mathbf{x} &= (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N) \\ &= (x_{1,0}, x_{1,1}, \dots, x_{1,n-1}, x_{2,0}, x_{2,1}, \dots, x_{N,0}, x_{N,1}, \dots, x_{N,n-1})\end{aligned}$$

$$\begin{aligned}\mathbf{r} &= (\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_N) \\ &= (r_{1,0}, r_{1,1}, \dots, r_{1,n-1}, r_{2,0}, r_{2,1}, \dots, r_{N,0}, r_{N,1}, \dots, r_{N,n-1})\end{aligned}$$

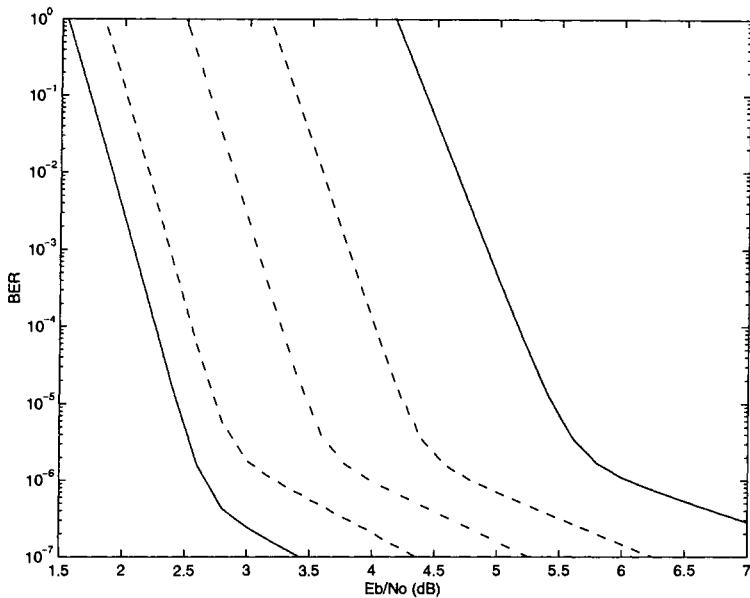


Fig. 8.8: Bit error probability upper bound for the 4 state, rate 1/3 serial code with information size 100 on independent Rician fading channels without channel state information. The curves are for Rician channels with $K=0, 2, 5, 50$, starting from the top, with the bottom one referring to an AWGN channel.

8.6.1 Modified MAP Decoding with CSI

In the MAP decoding method, the log-likelihood ratio for the input symbol at time t , denoted by $\Lambda(c_t)$, is given by [3] [4]

$$\Lambda(c_t) = \log \frac{\sum_{l=0}^{M_s-1} \alpha_{t-1}(l') \gamma_t^1(\mathbf{r}_t, l', l) \beta_t(l)}{\sum_{l=0}^{M_s-1} \alpha_{t-1}(l') \gamma_t^0(\mathbf{r}_t, l', l) \beta_t(l)} \quad (8.46)$$

where $\gamma_t^i(\mathbf{r}_t, l', l)$ is the branch transition probability which can be determined from the transition probabilities of the channel and the encoder trellis. $\alpha_{t-1}(l')$ and $\beta_t(l)$ are the forward and backward recursive probability functions obtained from $\gamma_t^i(\mathbf{r}_t, l', l)$. For an independent fading channel with ideal channel state information, the branch transition probability $\gamma_t^i(\mathbf{r}_t, l', l)$ based on the modified

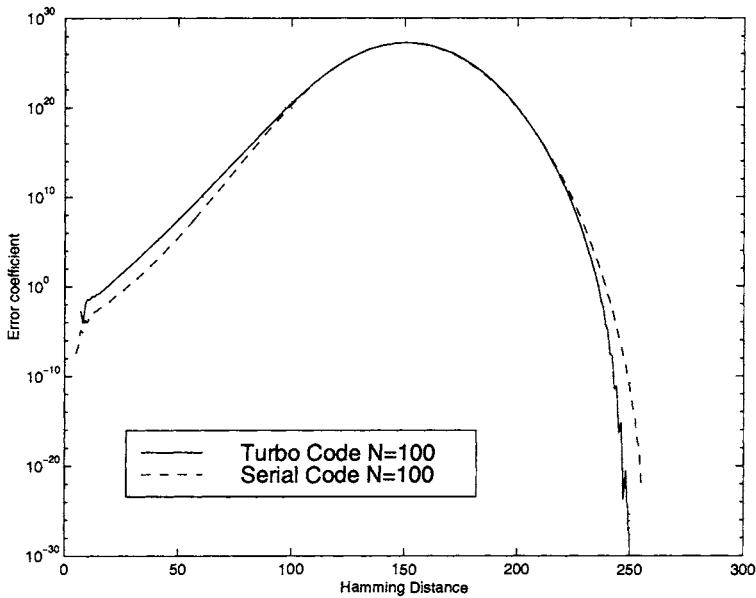


Fig. 8.9: Distance spectrum comparison of the 4 state, rate 1/3 turbo and serial concatenated codes with information size 100

branch metric can be computed as

$$\gamma_t^i(\mathbf{r}_t, l', l) = p_t(i) \cdot \exp\left(-\frac{\sum_{j=0}^{n-1} (r_{t,j} - a_{t,j}x_{t,j}^i)^2}{2\sigma^2}\right), \quad i = 0, 1$$

where $p_t(i)$ is the a priori probability of $c_t = i$ and σ^2 is the variance of the Gaussian noise. In a derivation similar to [3] [4] the log-likelihood ratio for the input symbol at time t , $\Lambda(c_t)$, can be decomposed into

$$\Lambda(c_t) = \log \frac{p_t(1)}{p_t(0)} + \frac{2}{\sigma^2} a_{t,0} r_{t,0} + \Lambda_e(u_t) \quad (8.47)$$

where the first term is the a priori information obtained from the other decoder. The second term is the systematic information generated by the code information bit. $\Lambda_e(c_t)$ is the extrinsic informa-

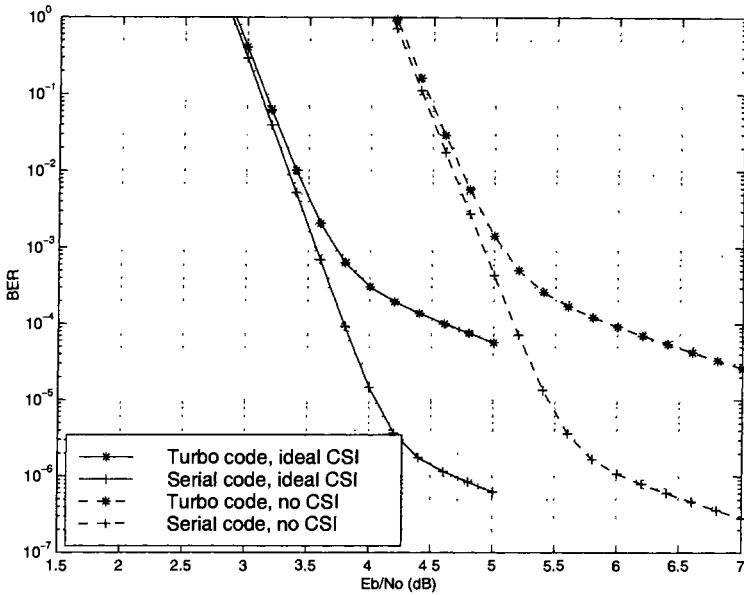


Fig. 8.10: Bit error probability upper bound comparison of the 4 state, rate 1/3 turbo and serial concatenated codes with information size 100 on independent Rayleigh fading channels

tion generated by the code parity check bit and is given by

$$\Lambda_e(c_t) = \log \frac{\sum_{l=0}^{M_s-1} \alpha_{t-1}(l') \exp\left(-\frac{\sum_{j=1}^{n-1} (r_{t,j} - a_{t,j} x_{t,j}^1)^2}{2\sigma^2}\right) \beta_t(l)}{\sum_{l=0}^{M_s-1} \alpha_{t-1}(l') \exp\left(-\frac{\sum_{j=1}^{n-1} (r_{t,j} - a_{t,j} x_{t,j}^0)^2}{2\sigma^2}\right) \beta_t(l)}$$

This extrinsic information $\Lambda_e(c_t)$ is interleaved (or deinterleaved) and then used as the a-priori information for the other decoder.

8.6.2 Modified SOVA Decoding with CSI

In the SOVA method, the path metric corresponding to path \mathbf{x} at time t , denoted by $\mu_t^{\mathbf{x}}$, is calculated by [5]

$$\mu_t^{\mathbf{x}} = \mu_{t-1}^{\mathbf{x}} + \sum_{j=0}^{n-1} \left[(r_{t,j} - a_{t,j} x_{t,j}^{\mathbf{x}})^2 \right] - \log p_t(i) \quad (8.48)$$

where $x_{t,j}^{\mathbf{x}}$ is the j th modulated symbol on path \mathbf{x} at time t , $\mu_{t-1}^{\mathbf{x}}$ is the smallest metric of the path connected to path \mathbf{x} at time $t-1$ and $\log p_t(i)$ is the logarithm of the a priori probability at time t obtained from the previous decoder for the information bit $c_t = i$.

The SOVA decoder provides a soft output in the form of an approximate log-likelihood ratio of the posteriori probabilities of the information bits 1 and 0. The soft output of the SOVA decoder for the input symbol at time t can be approximately expressed as the metric difference of the maximum likelihood path and its strongest competitor at time t . The strongest competitor of the maximum likelihood path is the path which has the minimum path metric among all paths obtained by replacing the trellis symbol on the maximum likelihood path at time t by its complementary symbol. The SOVA output, denoted by $\Lambda(c_t)$ at time t , can be expressed as [5]

$$\begin{aligned}\Lambda(c_t) &= \log \left[\frac{P\{c_t = 1 \mid \mathbf{r}\}}{P\{c_t = 0 \mid \mathbf{r}\}} \right] \\ &= \log \left[\frac{P\{x_{t,0} = +1 \mid \mathbf{r}\}}{P\{x_{t,0} = -1 \mid \mathbf{r}\}} \right] \\ &= \mu_t^{-1} - \mu_t^1\end{aligned}\quad (8.49)$$

where μ_t^{-1} is the minimum path metric corresponding to $x_{t,0} = -1$ and μ_t^1 is the minimum path metric corresponding to $x_{t,0} = 1$. The path metric and the SOVA output have the additive property. Following the derivation in MAP, we can split the soft output into two parts, the intrinsic information $\Lambda_i(c_t)$ and the extrinsic information $\Lambda_e(c_t)$, as follows

$$\Lambda(c_t) = \Lambda_i(c_t) + \Lambda_e(c_t) \quad (8.50)$$

The intrinsic information, $\Lambda_i(c_t)$, is given by

$$\Lambda_i(c_t) = \log \left[\frac{p_t(1)}{p_t(0)} \right] + 4a_{t,0}r_{t,0} \quad (8.51)$$

Therefore, the extrinsic information $\Lambda_e(c_t)$ can be obtained from (8.50) and (8.51) as

$$\Lambda_e(c_t) = \Lambda(c_t) - \log \left[\frac{p_t(1)}{p_t(0)} \right] - 4a_{t,0}r_{t,0} \quad (8.52)$$

and it will be used to calculate the a priori probabilities ratio in the next step of decoding after interleaving or deinterleaving.

8.7 Performance Simulation Results on Fading Channels

8.7.1 Performance Comparison Between MAP and SOVA Algorithms on Independent Fading Channels

The bit error rate performance of the turbo codes on fading channels is estimated by simulation. In the simulation, we considered the rate 1/3 turbo code with memory order $\nu = 4$ and information size $N = 1024$. The generator polynomials of the turbo code in the octal form are $\mathbf{g}_0 = (37)$ and $\mathbf{g}_1 = (21)$. In the receiver, iterative decoding with symbol-by-symbol MAP and SOVA methods with and without channel state information is employed. The number of iterations is 8 for interleaver size 1024.

For the independent Rayleigh fading channels, the performance simulation results are shown in Fig. 8.11. At a bit error rate of 10^{-5} , both MAP and SOVA decoders with CSI give an improvement of about 1dB relative to the respective decoders without CSI. It can also be observed that MAP decoding is superior to SOVA decoding by 0.5dB at the bit error rate of 10^{-5} , for both decoders with and without channel state information. A similar result has been obtained for AWGN channels [5].

8.7.2 Performance Comparison Between Turbo and Serial Concatenated Codes on Independent Fading Channels

The bit error rate performance comparison of the turbo and serial concatenated coding schemes on fading channels was also carried out by simulation. We simulated rate 1/3 turbo and serial codes with memory order $\nu = 2$ and information sizes of $N = 1024$ and

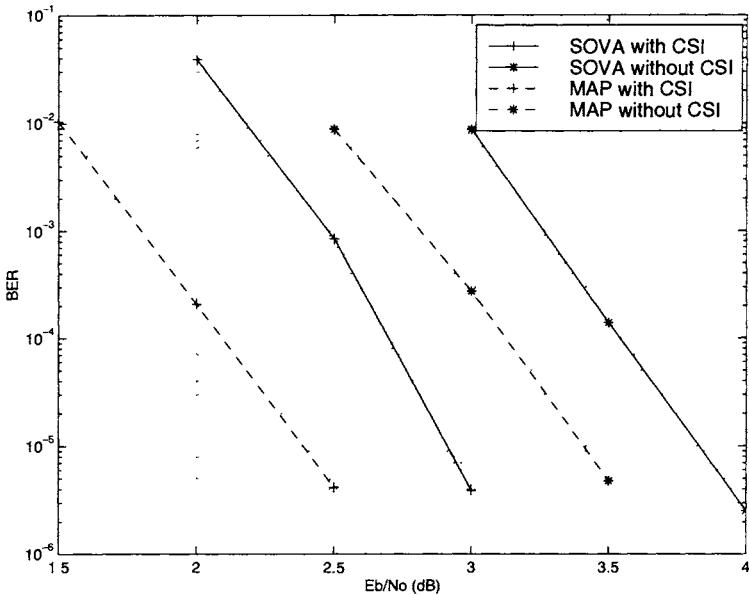


Fig. 8.11: Performance comparison of MAP and SOVA, with and without CSI, for the 16 state, rate 1/3 turbo code on an independent Rayleigh fading channel, information size 1024, the number of iterations 8

4096. In the receiver, iterative SOVA decoding is employed. The number of iterations was 8 for information size 1024 and 18 for information size 4096.

For the independent Rayleigh fading channels, the performance simulation results for the turbo and serial codes are shown in Fig. 8.12. It can be observed that the turbo code has a better performance at low SNR's than the serial code. However, the serial code outperforms the turbo code at high SNR's. For the turbo code, a BER error floor appears at 10^{-6} . A similar result has been obtained for AWGN channels [5].

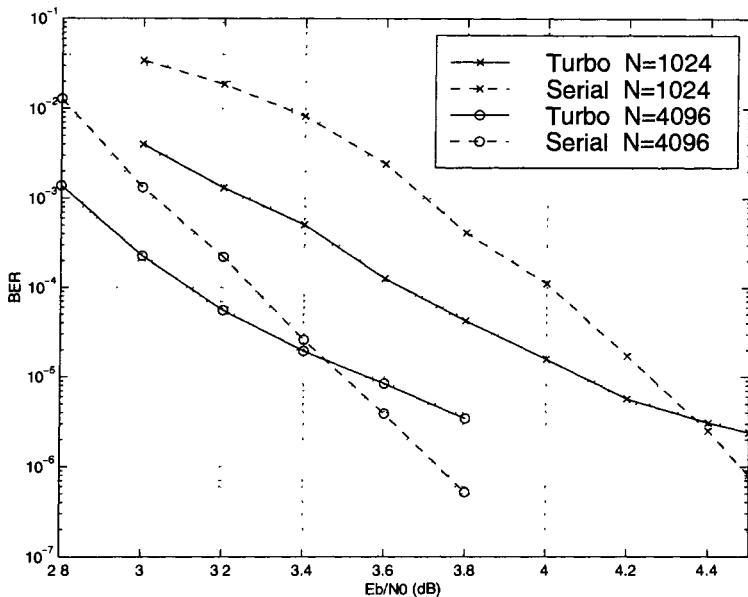


Fig. 8.12: Performance comparison for the 4 state, rate 1/3 turbo and serial codes on an independent Rayleigh fading channel

8.7.3 Performance Comparison Between MAP and SOVA Algorithms on Correlated Fading Channels

In some applications, such as mobile and satellite communications, the channel interleaving depth is severely constrained by the maximum delay requirements. This results in a correlated fading channel. However, the performance of the communication systems with correlated fades is not amenable to analytical evaluation. In this section, we investigate the bit error rate performance of the coding schemes on correlated Rayleigh fading channels by simulation.

In the simulations, the fade rate normalized by the symbol rate was 0.01. The performance comparison of MAP and SOVA, with and without channel state information for the rate 1/3 turbo code with memory order $\nu = 4$ and information size $N = 1024$ on a correlated Rayleigh fading channel is shown in Fig. 8.13. We can see from the figure that the MAP decoder with CSI achieves an im-

provement of about 1.2dB at a BER of 10^{-5} relative to the MAP decoder without CSI. The corresponding improvement for the SOVA decoder with ideal CSI is about 1.8dB. It can be also observed that without CSI the performance degradation of the SOVA decoding relative to the MAP decoding is approximately 0.7dB. However, the SOVA CSI decoder has almost the same performance as the MAP CSI decoder at high E_b/N_0 . By comparing Fig. 8.11 with Fig. 8.13, we can see that the turbo code performance degrades significantly due to the correlated fades in the channel. For the specified normalized fade rate, the turbo code performance is about 1~1.8dB worse relative to an independent fading channel.

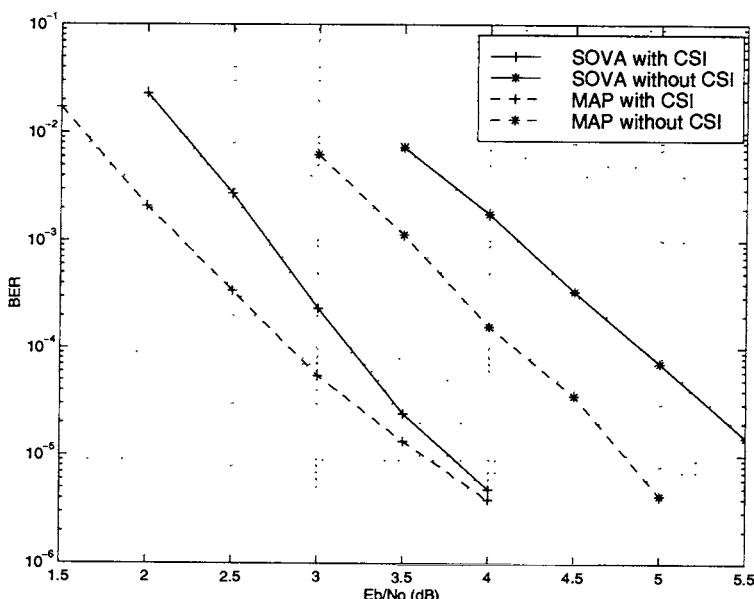


Fig. 8.13: Performance comparison of MAP and SOVA, with and without CSI, for the 16 state, rate 1/3 turbo code on a correlated Rayleigh fading channel, the fading rate normalized by the symbol rate is 10^{-2} , information size 1024, the number of iterations 8

8.7.4 Performance Comparison Between Turbo and Serial Concatenated Codes on Correlated Fading Channels

Performance comparison of the rate 1/3 turbo and serial codes with memory order $\nu = 2$ and various information sizes on a correlated Rayleigh fading channel is shown in Fig. 8.14. In the simulation, iterative SOVA decoding is employed and the information sizes are 1024 and 8192. The number of iterations is 8 for information size 1024 and 18 for information size 8192.

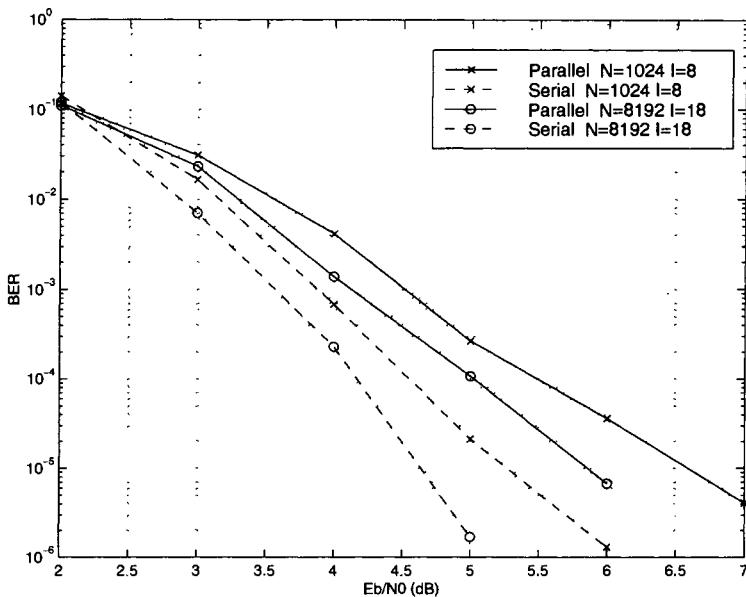


Fig. 8.14: Performance comparison for the turbo and serial codes on a correlated Rayleigh fading channel, the fading rate normalized by the symbol rate is 10^{-2} , information size N , the number of iterations I

We can see from the figure that the serial concatenated code consistently outperforms the turbo code on correlated fading channels. At a BER of 10^{-5} , the corresponding improvement achieved by the serial concatenated code relative to the turbo code is 1.25 dB and 1.2 dB for the information size of 1024 and 8192, respectively.

Bibliography

- [1] J. G. Proakis, *Digital Communications*, 2nd Ed., McGraw-Hill, New York, 1989.
- [2] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: Turbo-codes(1),” in *Proc. ICC’93*, Geneva, Switzerland, May 1993, pp. 1064-1070.
- [3] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, “Optimal decoding of linear code for minimizing symbol error rate,” *IEEE Trans. Inform. Theory*, vol. 20, no. 2, Mar. 1974, pp. 284-187.
- [4] J. Hagenauer, P. Robertson, and L. Papke, “Iterative (‘Turbo’) decoding of systematic convolutional codes with the MAP and SOVA algorithms,” in *Proc. of the 1994 ITG Conference on Source and Channel Coding*, Munich, Oct. 1994, pp. 1-9.
- [5] B. Vucetic, “Iterative decoding algorithms,” in *Proc. of IEEE Int. Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC’97)*, Helsinki, Finland, Sep. 1997, pp. 99-120.
- [6] E. K. Hall and S. G. Wilson, “Design and analysis of turbo codes on Rayleigh fading channels,” *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 2, Feb. 1998, pp. 160-174.
- [7] S. Benedetto and G. Montorsi, “Unveiling turbo-codes: Some results on parallel concatenated coding schemes,” *IEEE Trans. Inform. Theory*, vol. 42, no. 2, Mar. 1996, pp. 409-428.

- [8] D. Divsalar, S. Dolinar, R. J. McEliece, and F. Pollara, "Transfer function bounds on the performance of turbo codes," TDA Progress Report 42-123, Jet Propulsion Lab., Aug. 1995, pp. 44-55.
- [9] J. Yuan, B. Vucetic, and W. Feng, "Combined turbo codes and interleaver design," *IEEE Trans. Commun.*, vol. 47, no. 4, Apr. 1999, pp. 484-487.
- [10] J. Yuan and B. Vucetic, "Turbo code performance on Rician fading channels," in *Proc. ICC'99*, Vancouver, Bc, Canada, June 1999, pp. 83-87.
- [11] J. Yuan, W. Feng and B. Vucetic, "Parallel and serial concatenated codes for fading channels," in *Proc. PIMRC'99*, Osaka, Japan, Sep. 1999, pp. 862-866.
- [12] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial concatenation of interleaved codes: Performance analysis, design, and iterative decoding," *IEEE Trans. Inform. Theory*, vol. 44, no. 3, May 1998, pp. 909-926.

Chapter 9

Turbo Trellis Coded Modulation Schemes

9.1 Introduction

The effectiveness of trellis coded modulation (TCM) techniques, proposed by Ungerboeck in 1982 [1], for communication over bandwidth limited channels is well established. TCM schemes have been applied to telephone, satellite and microwave digital radio channels, where coding gains of the order of 3-6dB are obtained with no loss of bandwidth or data rate.

Turbo codes can achieve remarkable error performance at a low signal-to-noise ratio close to the Shannon capacity limit. However, the powerful binary coding schemes are not suitable for bandwidth limited communication systems.

In order to achieve simultaneously large coding gains and high bandwidth efficiency, a general method is to combine turbo codes with trellis coded modulations. In this chapter we discuss several bandwidth efficient turbo coding schemes, their encoding/decoding algorithms and performance on Gaussian and flat fading channels.

For Gaussian channels, turbo coded modulation techniques can be broadly classified into binary schemes and turbo trellis coded modulation. The first group can further be divided into “pragmatic” schemes with a single component binary turbo code and multilevel binary turbo codes. Turbo trellis coded modulation can

be classified into turbo coded modulation with parity symbol puncturing and turbo coded modulation with information symbol puncturing. For flat fading channels an I-Q turbo coded modulation technique is considered.

9.2 Binary Turbo Coded Modulation

9.2.1 Pragmatic Binary Turbo Coded Modulation

In pragmatic turbo coded modulation design [4] a single binary turbo code of rate $1/n$ is used as the component code. Its encoder outputs are suitably multiplexed and punctured to obtain \tilde{m} parity symbol and $m - \tilde{m}$ information symbols, as shown in Fig. 9.1. These encoded symbols are mapped into an M -PSK or M -QAM signal set consisting of 2^m points. If an M -PSK modulation is used, m encoded symbols select one M -PSK signal according to Gray code mapping. If an M -QAM modulation is used, we map $m/2$ encoded symbols to an in phase (I) level of an M -QAM signal and map the other $m/2$ encoded symbols to a quadrature (Q) level. Separate Gray mapping for the I and Q channels is applied. The spectral efficiency of the scheme is $(m - \tilde{m})$ bits/s/Hz.

Though an interleaver is included between the multiplexer and puncturer modules, with the aim of symbol decorrelation in slow fading channels, it is not needed on Gaussian channels.

The receiver, shown in Fig. 9.2, calculates the log-likelihood function for each encoded binary digit based on the received noisy symbol and the signal subsets in the signal constellation specified by each binary digit. The stream of the bit likelihood values is then deinterleaved and demultiplexed before passing to the binary turbo decoder which can be based either on MAP or SOVA.

The pragmatic approach is simple, as only one turbo encoder and one turbo decoder is used. By modifying the puncturing function and modulation signal constellation, it is possible to obtain a large family of turbo coded modulation schemes.

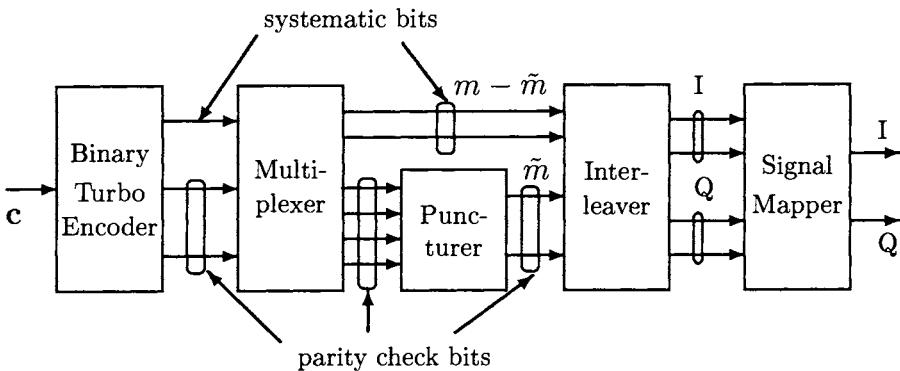


Fig. 9.1: Pragmatic turbo TCM encoder

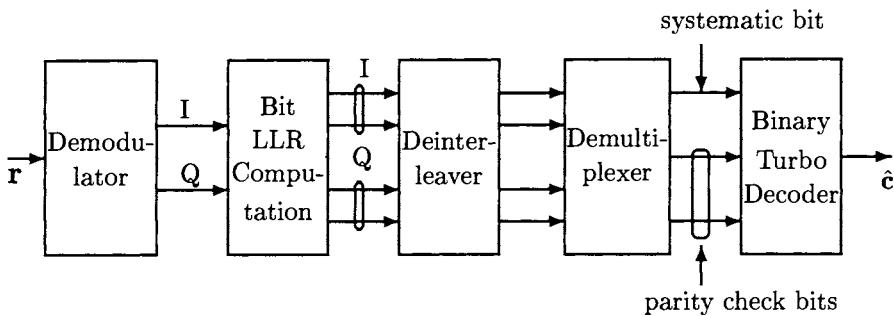


Fig. 9.2: Pragmatic turbo TCM decoder

To illustrate the pragmatic binary turbo coded modulations, we consider an example of a rate 1/2 turbo coded 16-QAM, where the parameters n , m and \tilde{m} in Fig. 9.1 are equal to 3, 4 and 2, respectively. A rate 1/3 binary turbo code is employed in the example. Every two encoding intervals, the encoded bits are multiplexed and punctured according to a puncturing matrix as

$$\mathbf{P} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (9.1)$$

This leads to a rate 1/2 turbo code. Let us denote four encoded bits at time t by $v_{t,i}$, $i = 1, 2, 3$ and 4. They are mapped to a 16-QAM

signal constellation to select a transmitted signal x_t , which is given by

$$x_t = x_{t,I} + jx_{t,Q} \quad (9.2)$$

where $x_{t,I}$ is the in-phase and $x_{t,Q}$ is the quadrature components of the transmitted signal x_t . A 16-QAM signal constellation with Gray mapping is shown in Fig. 9.3.

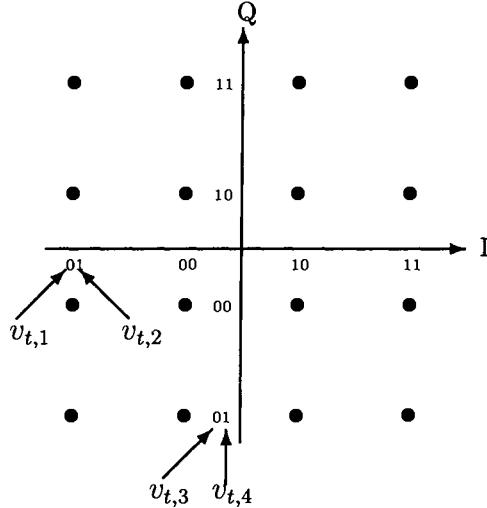


Fig. 9.3: 16-QAM with Gray mapping

Assuming coherent detection, the received signal at time t , denoted by r_t , can be represented by

$$r_t = r_{t,I} + jr_{t,Q} \quad (9.3)$$

where the I and Q component signals are given by

$$\begin{aligned} r_{t,I} &= x_{t,I} + n_{t,I} \\ r_{t,Q} &= x_{t,Q} + n_{t,Q} \end{aligned} \quad (9.4)$$

and where $n_{t,I}$ and $n_{t,Q}$ are two independent Gaussian noise signals with zero mean and variance σ^2 .

The log-likelihood ratio (LLR) for each encoded bit $v_{t,i}$, denoted by $\Lambda(v_{t,i})$, can be calculated by [4] [10]

$$\Lambda(v_{t,i}) = K_c \cdot \log \frac{Pr\{v_{t,i} = 1|r_t\}}{Pr\{v_{t,i} = 0|r_t\}} \quad (9.5)$$

where K_c is a constant. Since the in-phase and quadrature component signals are affected by independent noise signals, the application of Bayes' rule shows that the LLR's of $v_{t,1}$ and $v_{t,2}$ depend only on $r_{t,I}$, while the LLR's of $v_{t,3}$ and $v_{t,4}$ depend only on $r_{t,Q}$. Thus the LLR's of the encoded bits can be further written as

$$\begin{aligned}\Lambda(v_{t,i}) &= K_c \cdot \log \frac{\sum_{x_I:v_{t,i}=1} \exp\left(-\frac{1}{2\sigma^2} (r_{t,I} - x_I)^2\right)}{\sum_{x_I:v_{t,i}=0} \exp\left(-\frac{1}{2\sigma^2} (r_{t,I} - x_I)^2\right)}, \quad i = 1, 2 \\ \Lambda(v_{t,i}) &= K_c \cdot \log \frac{\sum_{x_Q:v_{t,i}=1} \exp\left(-\frac{1}{2\sigma^2} (r_{t,Q} - x_Q)^2\right)}{\sum_{x_Q:v_{t,i}=0} \exp\left(-\frac{1}{2\sigma^2} (r_{t,Q} - x_Q)^2\right)}, \quad i = 3, 4\end{aligned}$$

where x_I is the in-phase and x_Q is the quadrature components of a modulation signal of $v_{t,i} = j$, $j = 0, 1$. The log-likelihood values are deinterleaved and demultiplexed and then fed to the binary turbo iterative decoder.

It is worth noting that the LLR's of bits $v_{t,1}$ and $v_{t,2}$ affected by the same noise $n_{t,I}$ are not independent. The same applies to bits $v_{t,3}$ and $v_{t,4}$ affected by $n_{t,Q}$. In addition, for a given received signal r_t , the LLR's $\Lambda(v_{t,i})$ are not strictly Gaussian for all encoded bits $i = 1, 2, 3$, and 4. Thus, the binary turbo decoder is no longer optimal for the coded modulation scheme. Nevertheless, simulation results indicate that, for large SNR values, the LLR's $\Lambda(v_{t,i})$ are close to a Gaussian random variable with variance σ^2 if the constant K_c is set to $\sigma^2/2$. Therefore, the binary turbo decoder performs quite well in the pragmatic approach [4] [10].

9.2.2 Multilevel Turbo Coding

The multilevel coding scheme, devised by Imai and Hirakawa [5], is a combined coding and modulation method based on a number of binary component codes. Multilevel turbo codes are constructed by using binary turbo codes as the component codes. The transmitter for an M -ary signal constellation consists of $l = \log_2 M$ parallel binary encoders as shown in Fig. 9.4.

Let a message sequence of length k be

$$\mathbf{c} = (c_1, c_2, \dots, c_k). \quad (9.6)$$

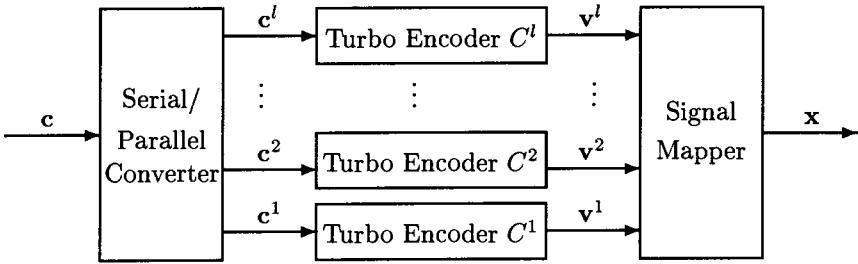


Fig. 9.4: Multilevel turbo encoder

The sequence is split into l blocks. The i th block sequence is given by

$$\mathbf{c}^i = (c_1^i, c_2^i, \dots, c_{k_i}^i) \quad (9.7)$$

where k_i is the length of the block and

$$k = \sum_{i=1}^l k_i \quad (9.8)$$

Each message block \mathbf{c}^i is encoded by an individual binary turbo encoder C^i generating a codeword

$$\mathbf{v}^i = (v_1^i, v_2^i, \dots, v_{n_i}^i) \quad (9.9)$$

where n_i is the code length of the component turbo code C^i . The code rate of C^i is $R_i = k_i/n_i$. For simplicity, we assume that all the component turbo codes have the same code length of n , that is, $n_1 = n_2 = \dots = n_l = n$. The output digits of the encoders at each instant time t , denoted by v_t^i , $i = 1, 2, \dots, l$, $t = 1, 2, \dots, n$, form a binary label $(v_t^1, v_t^2, \dots, v_t^l)$, which is mapped into a symbol x_t from the M -ary signal constellation. The transmitted signal sequence is represented by

$$\mathbf{x} = (x_1, x_2, \dots, x_t, \dots, x_n). \quad (9.10)$$

The overall code rate R of the multilevel turbo code is $R = k/nl$. The bandwidth efficiency of the coding scheme is

$$\eta = R \log_2 M = \frac{k}{n} \text{ bits/s/Hz.} \quad (9.11)$$

The maximum likelihood decoder operates on the overall code trellis. This decoder, in general, is too complex to implement. Alternatively, a suboptimum technique, called multistage decoding, can be used, resulting in the same asymptotic error performance as the maximum likelihood decoding. A multistage decoder for a multilevel turbo scheme is shown in Fig. 9.5.

The multistage decoder employs l component turbo decoders. Each component code C^i is successively decoded by the corresponding decoder D^i . The decisions at a lower level decoder will be used in higher level decoders. At stage i , the decoder D^i processes not only the received signal sequence, but also the decisions of the previous decoding stages j , denoted by \hat{x}^j , $j = 1, 2, \dots, i - 1$. As long as error free decisions $\hat{x}^j = x^j$ are produced by the decoder D^j , the multistage decoder can achieve the maximum likelihood decoding performance.

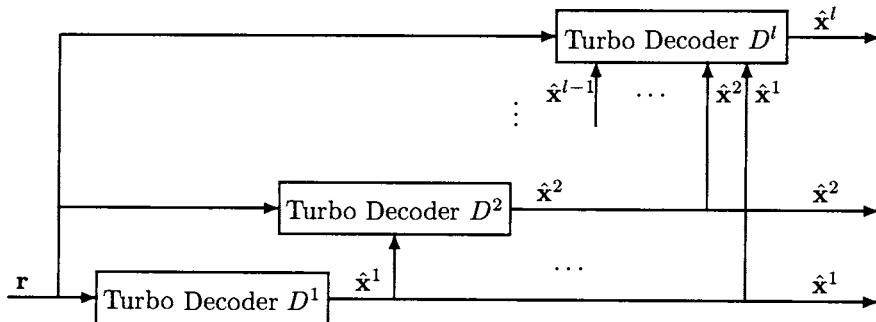


Fig. 9.5: Multilevel turbo decoder

An important issue in the code design is the choice of component codes and their code rates. Wachsmann and Huber [8] proposed a technique for selecting the component code rates. In this design, the component code rate at a particular modulation level, is chosen to be equal to the capacity of the equivalent binary input channel associated with that level. As the overall channel capacity is equal to the sum of the channel capacities for all levels, this design results, for infinite code lengths, in theory, in error free coding. Turbo codes come close to the Shannon capacity limit and provide almost error free coding. Therefore, they are suitable candidates

for component codes in a multilevel scheme. Another good property is that due to their good performance it can be assumed that there is negligible error propagation between the modulation levels. This is an important conclusion which enables the use of multistage decoding, as it asymptotically leads to the optimum result. However, for small block sizes, there can be a significant loss in terms of the signal-to-noise ratio needed to achieve a certain error rate. For example, for block sizes of the order of several hundred, the loss of a multilevel coding with multistage decoding is about 1dB relative to random coding [8]. In this example of multistage decoding there are no iterations between levels and only hard decisions are passed from one stage to the next one.

9.3 Turbo Trellis Coded Modulation

9.3.1 Schemes with Alternate Puncturing of Parity Digits

A method of combining turbo codes with multilevel modulation [6] [7] involves parallel concatenation of two recursive Ungerboeck type trellis codes with rate $k/(k+1)$. Fig. 9.6 shows the encoder structure comprising of two recursive convolutional encoders linked by a symbol interleaver and followed by a signal mapper. The encoder operates on a message block of N groups of k information bits, where N is the interleaver size. The message sequence, denoted by \mathbf{c} , is given by

$$\mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_t, \dots, \mathbf{c}_N)$$

where \mathbf{c}_t is a group of k information bits at time t

$$\mathbf{c}_t = (c_{t,0}, c_{t,1}, \dots, c_{t,k-1})$$

The encoder maps the input sequence into a block consisting of N encoded binary symbols,

$$\mathbf{v} = (\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_t, \dots, \mathbf{v}_N)$$

where \mathbf{v}_t is an encoded symbol at time t , of length $n = (k + 1)$ binary digits, given by

$$\mathbf{v}_t = (v_{t,0}, v_{t,1}, \dots, v_{t,k})$$

A symbol interleaver permutes the information bits in a symbol-wise way. That is, the ordering of k information bits arriving at the interleaver at a particular instant, remains unchanged.

For the component trellis code, some of the input bits may not be encoded. In practical implementations these inputs do not need to be interleaved, but are directly used to select the final point in a signal subset. At the receiver the values of these bits are estimated by subset decoding [1].

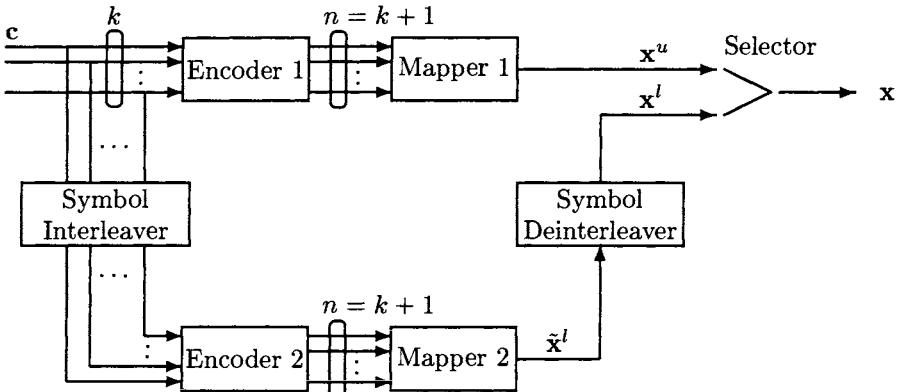


Fig. 9.6: Turbo TCM encoder with parity symbol puncturing

The encoded symbol sequence \mathbf{v} is mapped into the modulated sequence \mathbf{x}^u , given by

$$\mathbf{x}^u = (\mathbf{x}_1^u, \mathbf{x}_2^u, \dots, \mathbf{x}_t^u, \dots, \mathbf{x}_N^u)$$

where \mathbf{x}_t^u is a symbol from the modulation signal set, consisting of 2^{k+1} points, given by

$$\mathbf{x}_t^u = x_{t,I}^u + jx_{t,Q}^u$$

where $x_{t,I}^u$ is the in-phase and $x_{t,Q}^u$ is the quadrature component of the modulation signal \mathbf{x}_t^u . The mapping is based on the Ungerboeck set partitioning method [1].

The output of the second encoder is deinterleaved. This ensures that the k information bits which determine the encoded $(k + 1)$ binary digits of both the upper and lower encoder at a given time instant are identical.

Note that the k input information bits are contained in both modulated signals coming from the two parallel encoders. This is in contrast to binary turbo codes where the information bits are used only once, as they can be reconstructed from the other parallel received sequence. To prevent the transmission of each information bit twice, a selector is switched so that the modulated symbols are transmitted alternately from the upper and the lower mapper. In this way, for a signal set with 2^{k+1} points, a throughput of k bits/sec/Hz is achieved.

An example for the 8-state encoder and 8-PSK modulation is shown in Fig. 9.7. The system parameters are $k = 2$ and interleaver size $N = 6$ symbols.

We will illustrate the encoding/modulation operation by an example. Let us assume that a sequence of six information groups of 2 bits each ($k = 2$) enters the encoder

$$\mathbf{c} = (00, 01, 11, 10, 00, 11)$$

After encoding by a recursive systematic encoder and signal mapping based on the set partitioning rule, an 8-PSK sequence is generated by the upper encoder, denoted by \mathbf{x}^u ,

$$\mathbf{x}^u = (0, 2, 7, 5, 1, 6)$$

The information sequence is interleaved by a symbol interleaver and encoded into the modulated sequence by the lower decoder, denoted by $\tilde{\mathbf{x}}^l$,

$$\tilde{\mathbf{x}}^l = (6, 7, 0, 3, 0, 4)$$

After deinterleaving this sequence becomes

$$\mathbf{x}^l = (0, 3, 6, 4, 0, 7)$$

The selector alternately connects the upper and lower encoder to the channel, transmitting the first symbol from the upper encoder,

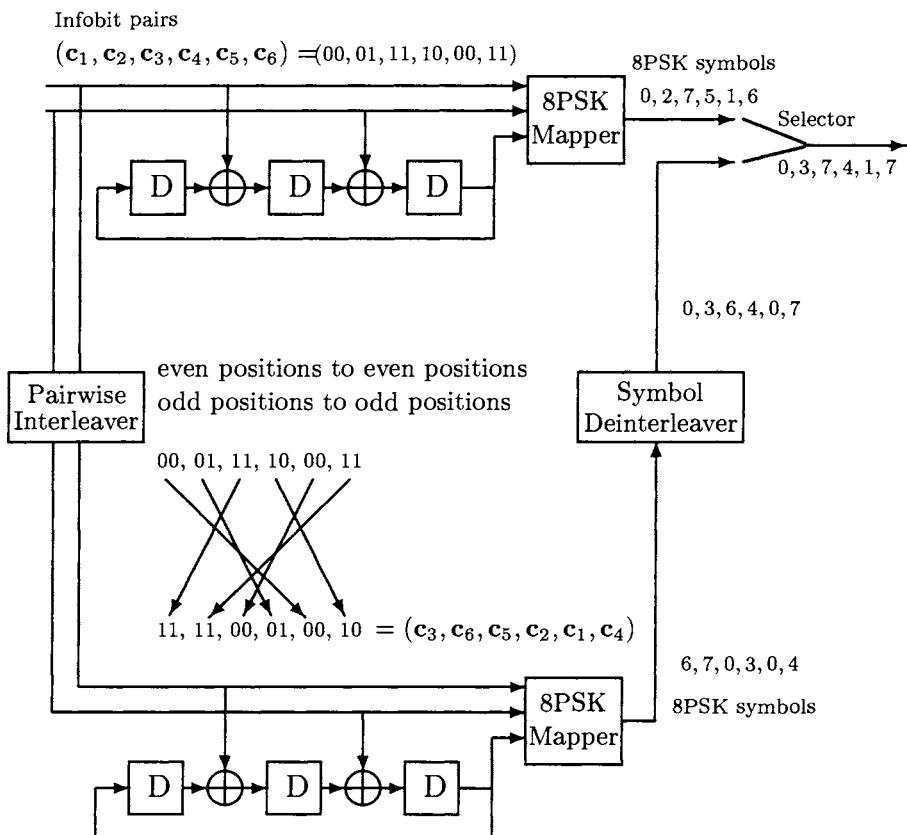


Fig. 9.7: Example of a turbo trellis coded 8-PSK with parity symbol puncturing

the second symbol from the lower encoder, the third symbol from the upper encoder, the fourth symbol from the lower encoder, etc. The transmitted sequence for this example is

$$\mathbf{x} = (0, 3, 7, 4, 1, 7)$$

Thus the parity symbol is alternately chosen from the upper and lower encoder. Each information group appears in the transmitted sequence only once.

For this coded modulation approach, the component code should be chosen with no parallel transitions so that each information bit

can benefit from interleaving. However, this condition can be relaxed, if the interleaver does not keep a group of k information bits unchanged [13] or if we desire a very high bandwidth efficiency [7]. If the interleaver swaps the two information bits in the previous example, the code is allowed to have parallel transitions. This operation ensures interleaving of information bits within a symbol.

To find good component codes, an exhaustive computer search can be employed. The search goal is maximizing the minimum Euclidean distance of each component code while randomly selecting the parity check digits of each second symbol [6].

9.3.2 Log-MAP Decoding Algorithm for Turbo Trellis Coded Modulation with Punctured Parity Digits

The decoder structure is shown in Fig. 9.8. In this scheme it is impractical to use the MAP algorithm, as the extrinsic information becomes either too large or too small, causing computational overflows. A Log-MAP algorithm is used instead in which the logarithm of probabilities is computed and passed to the next stage. In all other respects this algorithm follows the MAP algorithm, described in [2] [3].

The decoding process is very similar to the binary turbo code except that the symbol probability is used as the extrinsic information rather than the bit probability. The MAP decoding algorithm for nonbinary trellises is called symbol-by-symbol MAP algorithm.

The MAP decoder computes the LLR of each group of information bits $\mathbf{c}_t = i$. The soft output $\Lambda(\mathbf{c}_t = i)$ is given by [3]

$$\begin{aligned} \Lambda(\mathbf{c}_t = i) &= \log \frac{Pr\{\mathbf{c}_t = i | \mathbf{r}_1^\tau\}}{Pr\{\mathbf{c}_t = 0 | \mathbf{r}_1^\tau\}} \\ &= \log \frac{\sum_{(l', l) \in B_t^i} \alpha_{t-1}(l') \gamma_t^i(l', l) \beta_t(l)}{\sum_{(l', l) \in B_t^0} \alpha_{t-1}(l') \gamma_t^0(l', l) \beta_t(l)} \quad (9.12) \end{aligned}$$

where i denotes an information group from the set, $\{0, 1, 2, \dots, 2^k - 1\}$, and the probabilities $\alpha_t(l)$, $\beta_t(l)$ and $\gamma_t(l', l)$ can be computed recursively [3]. The symbol i with the largest log-likelihood ratio

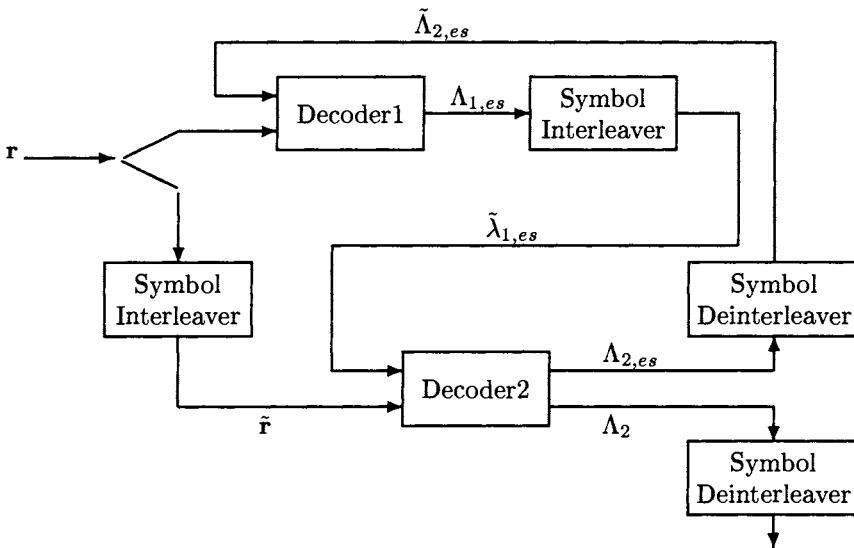


Fig. 9.8: Turbo TCM decoder with parity symbol puncturing

in Eq. (9.12), $i \in \{0, 1, 2, \dots, 2^k - 1\}$, is chosen as the hard decision output.

The MAP algorithm requires exponentiation and multiplication operations to compute the log-likelihood ratio $\Lambda(\mathbf{c}_t)$. One way of simplifying it is to work with the logarithms of probabilities of $\alpha_t(l)$, $\beta_t(l)$ and $\gamma_t(l', l)$, denoted by $\bar{\alpha}_t(l)$, $\bar{\beta}_t(l)$ and $\bar{\gamma}_t(l', l)$, respectively. The forward recursive variables can be computed as follows

$$\bar{\alpha}_t(l) = \log \sum_{l'=0}^{M_s-1} e^{\bar{\alpha}_{t-1}(l') + \bar{\gamma}_t(l', l)} \quad (9.13)$$

with the initial condition

$$\bar{\alpha}_0(0) = 0$$

$$\bar{\alpha}_0(l) = -\infty, \quad l \neq 0$$

and the backward recursive variables can be computed as

$$\bar{\beta}_t(l) = \log \sum_{l'=0}^{M_s-1} e^{\bar{\beta}_{t+1}(l') + \bar{\gamma}_{t+1}(l, l')} \quad (9.14)$$

with the initial condition

$$\bar{\beta}_\tau(0) = 0$$

$$\bar{\beta}_\tau(l) = -\infty, \quad l \neq 0$$

where

$$\bar{\gamma}_t(l', l) = \log \sum_{i=0}^{2^k-1} \gamma_t^i(l', l) \quad (9.15)$$

The branch transition probability at time t is calculated as

$$\gamma_t^i(l', l) = \begin{cases} \frac{p_t(i)}{p_t(0)} \exp \left(-\frac{(r_{t,I} - x_{t,I}^i(l))^2 + (r_{t,Q} - x_{t,Q}^i(l))^2}{2\sigma^2} \right), & \text{for } (l', l) \in B_t^i \\ 0, & \text{otherwise} \end{cases}$$

where $r_{t,I}$ and $r_{t,Q}$ are the channel output in-phase and quadrature components at time t respectively, $x_{t,I}^i(l)$ and $x_{t,Q}^i(l)$ are the modulated in-phase and quadrature components at time t , associated with the transition $S_{t-1} = l'$ to $S_t = l$ and input $\mathbf{c}_t = i$, respectively, and $p_t(i)$ is the a priori probability of $\mathbf{c}_t = i$.

The $\bar{\alpha}_t(l)$ and $\bar{\beta}_t(l)$ in Eqs. (9.13) and (9.14) can be calculated using the Jacobian algorithm

$$\begin{aligned} \log(e^{\delta_1} + e^{\delta_2}) &= \max(\delta_1, \delta_2) + \log(1 + e^{-|\delta_2 - \delta_1|}) \\ &= \max(\delta_1, \delta_2) + f_c(|\delta_2 - \delta_1|) \end{aligned} \quad (9.16)$$

where $f_c(\cdot)$ is a correction function, which can be implemented using a look-up table.

The expression $\log(e^{\delta_1} + e^{\delta_2} + \dots + e^{\delta_n})$ can be computed exactly by a recursive algorithm using Eg. (9.16) as follows

$$\begin{aligned} \log \sum_{i=1}^n e^{\delta_i} &= \log(\Delta + e^{\delta_n}), \quad \Delta = e^{\delta_1} + e^{\delta_2} + \dots + e^{\delta_{n-1}} = e^\delta \\ &= \max(\log \Delta, \delta_n) + f_c(|\log \Delta - \delta_n|) \\ &= \max(\delta, \delta_n) + f_c(|\delta - \delta_n|) \end{aligned}$$

The iterative process of the symbol-by-symbol MAP algorithm is similar to that of binary turbo decoders except that there is a difference in the nature of information exchange between the two component decoders [6]. For binary turbo decoders, a soft output

can be split into three terms. They are the a priori information generated by the other decoder, the systematic information generated by the code information bit and the extrinsic information generated by the code parity check digit. The extrinsic information is independent of the a priori and systematic information. The extrinsic information is exchanged between the two component decoders. In contrast to binary turbo codes, in turbo TCM we cannot separate the influence of the information and the parity-check components within one symbol. The systematic information and the extrinsic information are not independent. Thus both systematic and extrinsic information will be exchanged between the two component decoders. The joint extrinsic and systematic information of the first Log-Map decoder, denoted by $\Lambda_{1,es}(\mathbf{c}_t = i)$, can be obtained as

$$\Lambda_{1,es}(\mathbf{c}_t = i) = \Lambda_1(\mathbf{c}_t = i) - \log \frac{p_t(i)}{p_t(0)} \quad (9.17)$$

The joint extrinsic and systematic information $\Lambda_{1,es}(\mathbf{c}_t = i)$ is used as the estimate of the a priori probability ratio at the next decoding stage. After interleaving, it is denoted by $\tilde{\Lambda}_{1,es}(\mathbf{c}_t = i)$. The joint extrinsic and systematic information of the second decoder is given by

$$\Lambda_{2,es}(\mathbf{c}_t = i) = \Lambda_2(\mathbf{c}_t = i) - \tilde{\Lambda}_{1,es}(\mathbf{c}_t = i) \quad (9.18)$$

In the next iteration the a priori probability ratio in Eq. (9.17) is replaced by the deinterleaved joint extrinsic and systematic information from the second decoding stage, denoted by $\tilde{\Lambda}_{2,es}(\mathbf{c}_t = i)$.

It is worth noting that for symbol-by-symbol MAP decoding, each component decoder should avoid to use the same systematic information twice in every iterative decoding step [6]. In turbo TCM, each decoder alternately receives the noisy output of its own encoder and that of the other encoder. That is, the parity symbols in every second received signal belong to the other encoder and need to be treated as punctured.

For example, we consider the first decoder. For every odd received signal, the decoding operation proceeds as for the binary turbo codes when the decoder receives the symbol generated by its own encoder. The only difference is that the extrinsic information is replaced by the joint extrinsic and systematic information.

However, for every even received signal, the decoder receives the punctured symbol in which the parity digit is generated by the other encoder. The decoder in this case ignores this symbol by setting the branch transition metric to zero. The only input at this step in the trellis is the a priori component obtained from the other decoder. This component contains the systematic information.

9.3.3 SOVA Decoding Algorithm for Turbo Trellis Coded Modulation with Punctured Parity Digits

The SOVA decoder structure is similar to that shown in Fig. 9.8. As in decoding of binary turbo codes, we define the decoder soft output $\Lambda(\mathbf{c}_t = i)$, where $i \in \{0, 1, 2, \dots, 2^k - 1\}$, as

$$\begin{aligned}\Lambda(\mathbf{c}_t = i) &= \log \frac{Pr\{\mathbf{c}_t = i | \mathbf{r}_1^T\}}{Pr\{\mathbf{c}_t = 0 | \mathbf{r}_1^T\}} \\ &= \mu_t^0 - \mu_t^i\end{aligned}\quad (9.19)$$

where μ_t^0 is the minimum path metric for $\mathbf{c}_t = 0$, and μ_t^i is the minimum path metric for $\mathbf{c}_t = i$.

We define the branch metric, assigned to a trellis branch at time t , as

$$\nu_t = (r_{t,I} - x_{t,I})^2 + (r_{t,Q} - x_{t,Q})^2 - \log p_t(i) \quad (9.20)$$

where $r_{t,I}$ and $r_{t,Q}$ are the channel outputs of the in-phase and quadrature components at time t , respectively, $x_{t,I}$ and $x_{t,Q}$ are the modulated in-phase and quadrature components of the branch, generated by input $\mathbf{c}_t = i$, respectively, and $p_t(i)$ is the a priori probability of $\mathbf{c}_t = i$. The path metric, for a path \mathbf{x} in the trellis, can be computed as

$$\mu^{\mathbf{x}} = \sum_{t=1}^{\tau} \nu_t^{\mathbf{x}} \quad (9.21)$$

As in symbol-by-symbol MAP algorithm, for this scheme, we cannot separate the information bit and the parity-check binary digit in one coded symbol. Thus a decoder output can only be split into the a priori component and joint extrinsic and systematic

components. The joint extrinsic and systematic information of the first SOVA decoder, denoted by $\Lambda_{1,es}(\mathbf{c}_t = i)$, is given by

$$\Lambda_{1,es}(\mathbf{c}_t = i) = \Lambda_1(\mathbf{c}_t = i) - \log \frac{p_t(i)}{p_t(0)} \quad (9.22)$$

where $\Lambda_1(\mathbf{c}_t = i)$ is the soft output of the first decoder and $\log \frac{p_t(i)}{p_t(0)}$ is the a priori information for $\mathbf{c}_t = i$. The joint extrinsic and systematic information is used as the a priori probability at the next decoding stage as

$$\Lambda_{2,es}(\mathbf{c}_t = i) = \Lambda_2(\mathbf{c}_t = i) - \tilde{\Lambda}_{1,es}(\mathbf{c}_t = i). \quad (9.23)$$

9.3.4 Performance of Turbo Trellis Coded Modulation with Punctured Parity Digits

Let us consider the performance of rate 2/3 turbo trellis coded 8-PSK schemes with bandwidth efficiency 2 bits/s/Hz on AWGN channels. The generator polynomials of the 4, 8 and 16-state component codes are shown in Table 9.1. The bit error probabilities of the codes with various interleaver sizes of 1024 and 4096 are simulated. The number of iterations is 8 for the interleaver size of 1024 and 18 for the size of 4096. The simulation results based on the iterative SOVA decoding algorithm are shown in Figs. 9.9 - 9.11 [16].

Table 9.1: Rate 2/3 turbo trellis coded 8-PSK schemes

Trellis States	Generator Polynomials		
	\mathbf{g}_0	\mathbf{g}_1	\mathbf{g}_2
4	5	6	7
8	17	15	13
16	35	27	23

Performance comparison of the Log-MAP and SOVA for the 8-state turbo trellis code with interleaver size 1024 is shown in Fig.

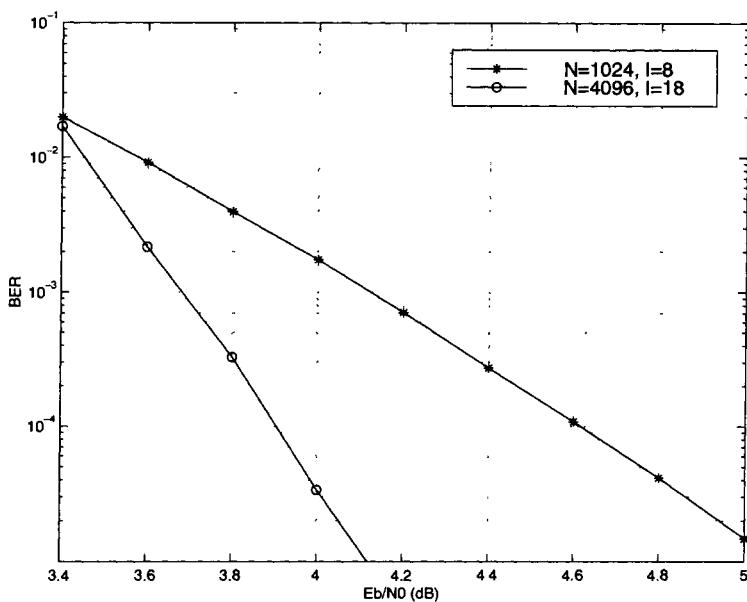


Fig. 9.9: Performance of the rate 2/3, 4-state turbo trellis coded 8-PSK with various interleaver sizes on an AWGN channel, SOVA decoding algorithm, the number of iterations I , bandwidth efficiency 2 bits/s/Hz

9.12. The SOVA is worse by 0.4 dB at the BER of 10^{-3} and by 0.2dB at the BER of 10^{-5} .

The performance of rate 3/4 turbo trellis coded 16-QAM with bandwidth efficiency 3bits/s/Hz on AWGN channels is also considered. The generator polynomials of the component codes with various numbers of trellis states are given in Table 9.2. The effect of the variable interleaver size on the performance of the codes with various numbers of trellis states on AWGN channels is shown in Figs. 9.13 - 9.15.

The comparative results of Log-MAP and SOVA decoded turbo trellis coded 16-QAM with bandwidth efficiency 3 bits/s/Hz, with various interleaver sizes and number of trellis states on AWGN channels are also shown in Figs. 9.13 - 9.15. It is interesting to note that the difference between the two algorithms is negligible

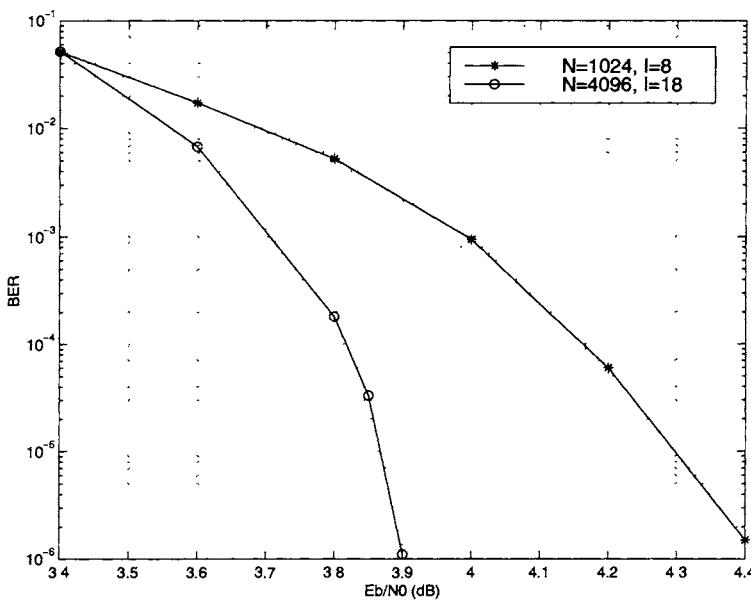


Fig. 9.10: Performance of the rate 2/3, 8-state turbo trellis coded 8-PSK with various interleaver sizes on an AWGN channel, SOVA decoding algorithm, the number of iterations I , bandwidth efficiency 2 bits/s/Hz

Table 9.2: Rate 3/4 turbo trellis coded 16-QAM schemes

Trellis States	Generator Polynomials			
	g_0	g_1	g_2	g_3
4	5	6	4	7
8	17	15	5	13
16	23	35	33	37

for a low number of states and increases with the growing number of states, reaching a value of about 0.3dB at the BER of 10^{-5} for the 16-state code, as shown in Fig. 9.15.

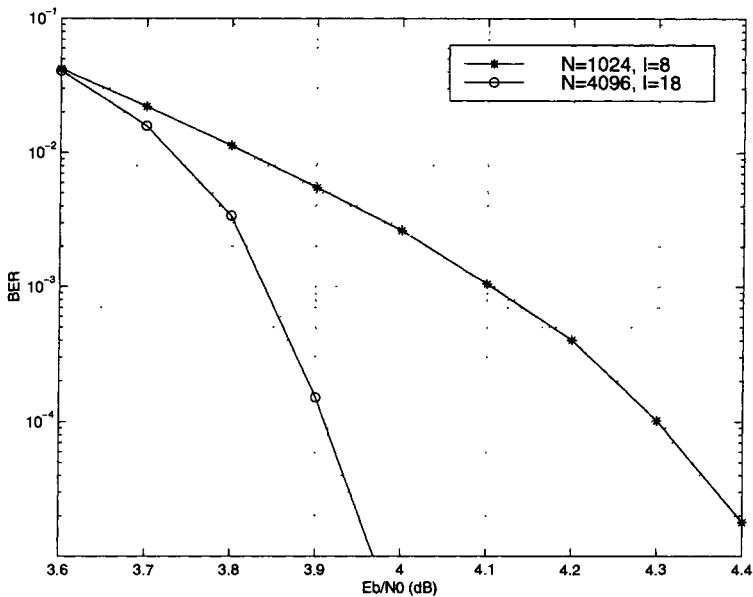


Fig. 9.11: Performance of the rate $2/3$, 16-state turbo trellis coded 8-PSK with various interleaver sizes on an AWGN channel, SOVA decoding algorithm, the number of iterations I , bandwidth efficiency 2 bits/s/Hz

9.3.5 Schemes with Puncturing of Systematic Bits

Parallel concatenation of two recursive trellis codes with puncturing of systematic bits was proposed by Benedetto, Divsalar, Montorsi and Pollara [9]. The basic idea of the scheme is to puncture the output symbols of each trellis encoder and select the puncturing pattern such that the output symbols of the parallel concatenated code contain the input information only once. In contrast to the scheme with symbol interleaving and puncturing of parity digits, this scheme uses multiple bit interleaving and punctures systematic bits of both component trellis codes.

In the code structure, two high rate $k/(k+1)$ component encoders are linked in parallel. We assume that k is even. Each encoder generates one parity and k information binary digits. Both

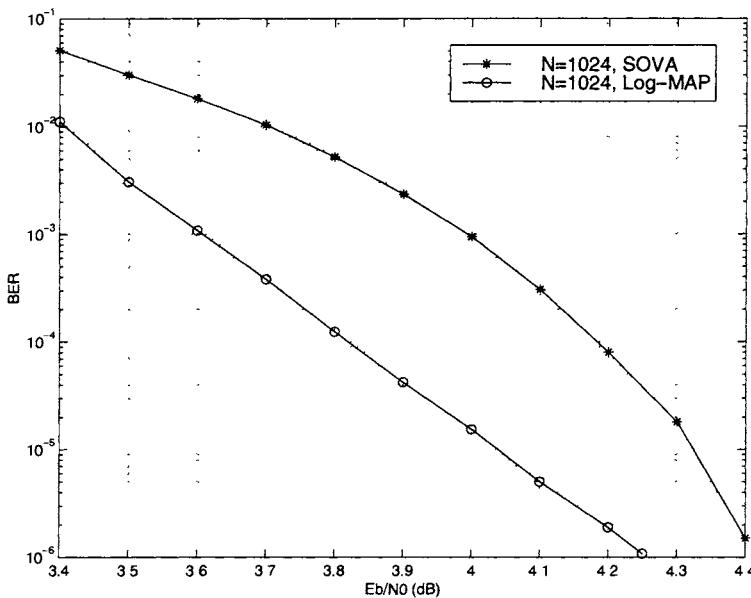


Fig. 9.12: Performance comparison of the Log-MAP and SOVA for the rate 2/3, 8-state turbo trellis coded 8-PSK with interleaver size 1024 on an AWGN channel, bandwidth efficiency 2 bits/s/Hz

of these two parity digits are transmitted in their respective modulated signals. In order to avoid transmission of each information bit twice, the first $k/2$ information bits are used at the upper mapper and the second $k/2$ at the lower mapper. That is, $k/2$ information bits are punctured at each encoder. However, all k information bits affect the states of both encoders. The total number of encoded bits is $k + 2$. These bits can be mapped to either M -PSK or M -QAM signal constellations. If we map $1 + k/2$ encoded bits of each component trellis code to an M -PSK or M -QAM signal, where $M = 2^{1+k/2}$, we can achieve a bandwidth efficiency of $k/2$ bits/s/Hz. Since two modulated signals are generated in every encoding interval, this scheme is equivalent to a multi-dimensional M -PSK or M -QAM trellis code. For M -QAM we can also map the $1 + k/2$ encoded bits of the first trellis code to I-channel and the $1 + k/2$ encoded bits of the second trellis code to Q-channel, where

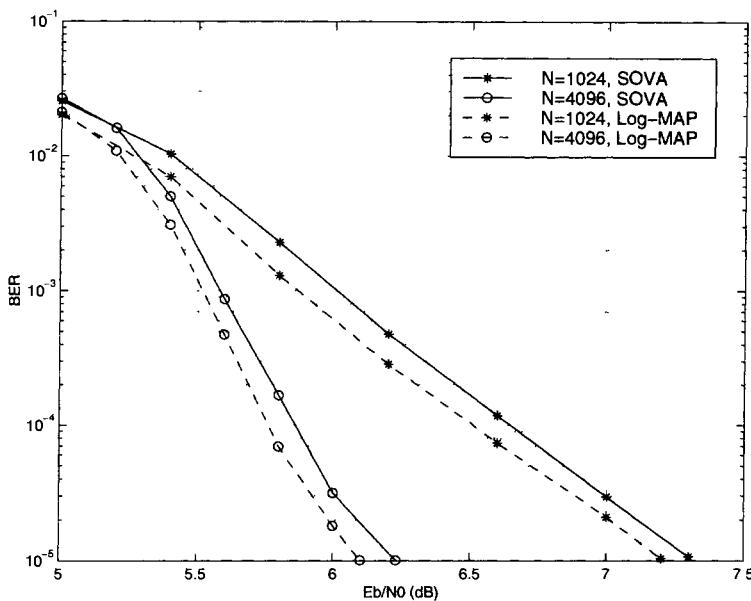


Fig. 9.13: Performance comparison of the Log-MAP and SOVA for the rate 3/4, 4-state turbo trellis coded 16-QAM with various interleaver sizes on an AWGN channel, bandwidth efficiency 3 bits/s/Hz

$M = 2^{k+2}$. In this case, a bandwidth efficiency of k bits/s/Hz can be achieved.

To optimize the code performance, the component trellis code should be designed to maximize the code *effective free Euclidean distance* for a given modulation and signal mapping. The effective free Euclidean distance of a turbo trellis coded modulation scheme is defined by the minimum Euclidean distance between all pairs of coded sequences, whose input sequences have a Hamming distance of 2 [9].

Decoding is a straightforward application of the iterative symbol-by-symbol MAP algorithm for the binary turbo codes. The only difference is that 1) the extrinsic information computed for a symbol needs to be converted to a bit level since they are carried out on a bit level, and 2) after the interleaving/deinterleaving operations, the bit a priori probabilities need to be converted to a symbol level

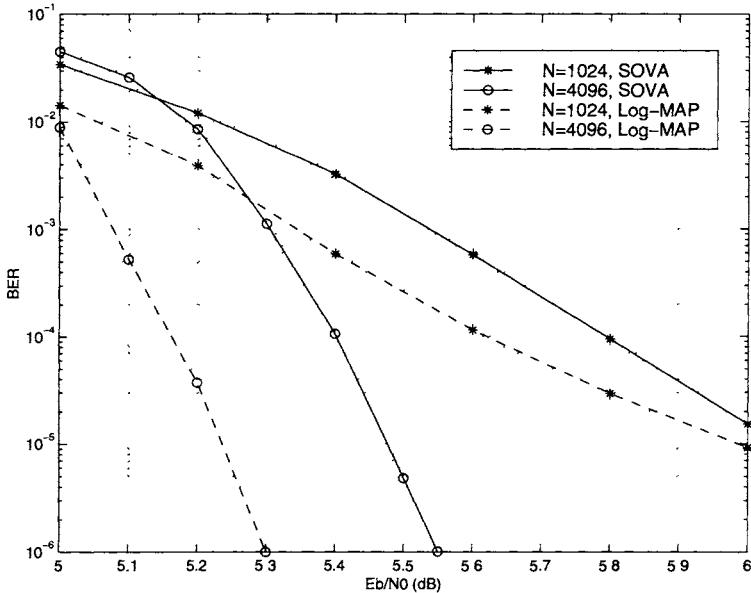


Fig. 9.14: Performance comparison of the Log-MAP and SOVA for the rate 3/4, 8-state turbo trellis coded 16-QAM with various interleaver sizes on an AWGN channel, bandwidth efficiency 3 bits/s/Hz

since they will be used in the branch transition probability calculation in the symbol MAP algorithm.

For a symbol of a group of k information bits given by

$$\mathbf{c} = (c_1, c_2, \dots, c_k), \quad (9.24)$$

where $c_j = 0, 1$, $j = 1, 2, \dots, k$. If we denote the extrinsic information of the symbol \mathbf{c} by $\Lambda_e(\mathbf{c})$, the extrinsic information of the j th bit can be represented by [9]

$$\Lambda_e(c_j) = \log \frac{\sum_{\mathbf{c}:c_j=1} e^{\Lambda_e(\mathbf{c})}}{\sum_{\mathbf{c}:c_j=0} e^{\Lambda_e(\mathbf{c})}} \quad (9.25)$$

After the interleaving/deinterleaving operations, the a priori probability of any symbol can be given by [9]

$$P(\mathbf{c} = (c_1, c_2, \dots, c_k)) = \prod_{j=1}^k \frac{e^{c_j \cdot \tilde{\Lambda}_e(c_j)}}{1 + e^{\tilde{\Lambda}_e(c_j)}} \quad (9.26)$$

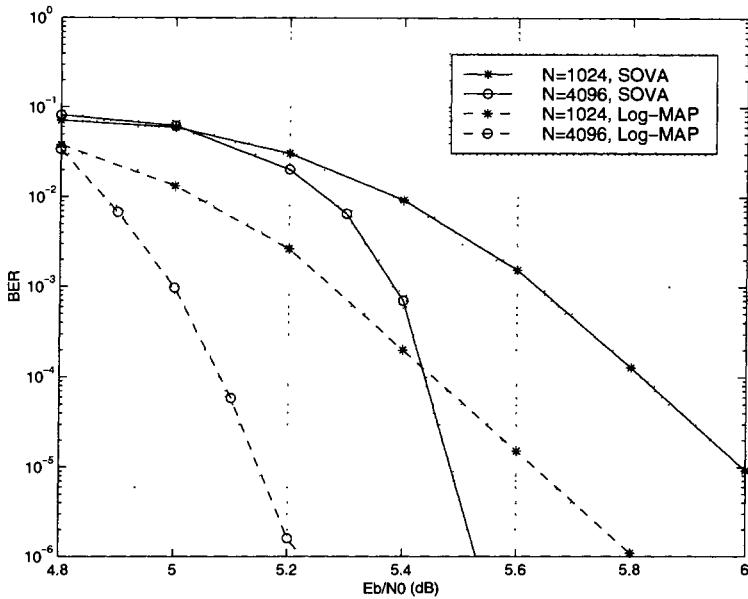


Fig. 9.15: Performance comparison of the Log-MAP and SOVA for the rate 3/4, 16-state turbo trellis coded 16-QAM with various interleaver sizes on an AWGN channel, bandwidth efficiency 3 bits/s/Hz

We will illustrate the turbo trellis coded modulation schemes by examples. First we consider a coded 16-QAM scheme with 2 bits/s/Hz as shown in Fig. 9.16. The number of information bits in a block, k , is 2. Two 16-state, rate 2/3 trellis codes are linked by two bit interleavers in parallel. The generator polynomials are

$$\mathbf{g}_0 = (23), \quad \mathbf{g}_1 = (16), \quad \mathbf{g}_2 = (27).$$

The encoded bits of the two component trellis codes are punctured and then mapped to the I and Q branches of a 16-QAM signal according to natural mapping, respectively. The squared effective free Euclidean distance of the code is 7.2. The BER performance of the coding scheme with two interleavers of size 16384 each and 8 iterations is shown in Fig. 9.17. The performance of the pragmatic turbo coded 16-QAM with the same bandwidth efficiency and interleaver size 32768 [14] is also shown in the figure. It is clear that the turbo trellis coded 16-QAM with systematic symbol puncturing

outperforms the pragmatic turbo coded 16-QAM by about 0.1 dB at a BER of 10^{-5} .

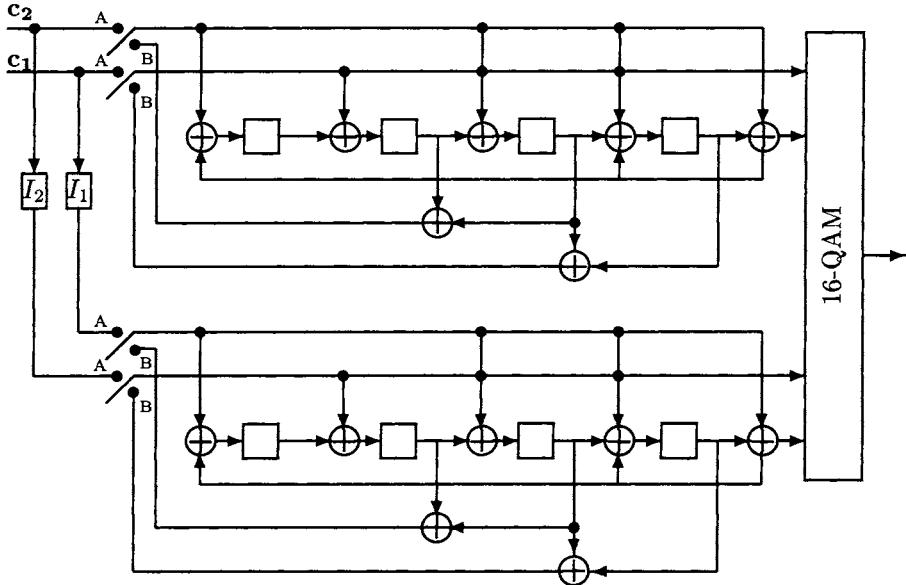


Fig. 9.16: Turbo trellis coded 16-QAM with systematic symbol puncturing

Another example is a turbo trellis coded 8-PSK modulation scheme as shown in Fig. 9.18. Each parallel information sequence is interleaved before passing to the lower encoder. For the given example, the 16-state component codes have the same code rate of $4/5$ and the encoder contains four interleavers. The generator polynomials are

$$\mathbf{g}_0 = (23), \quad \mathbf{g}_1 = (14), \quad \mathbf{g}_2 = (16), \quad \mathbf{g}_3 = (21), \quad \mathbf{g}_4 = (31).$$

The encoded bits of each component trellis code are punctured and then mapped to an 8-PSK signal constellation with reordered mapping [9]. The spectral efficiency is 2 bits/sec/Hz. The squared effective free Euclidean distance of the code is 6.34. The BER performance of the code with four interleavers of size 4096 each and variable number of iterations is shown in Fig. 9.19 [9].

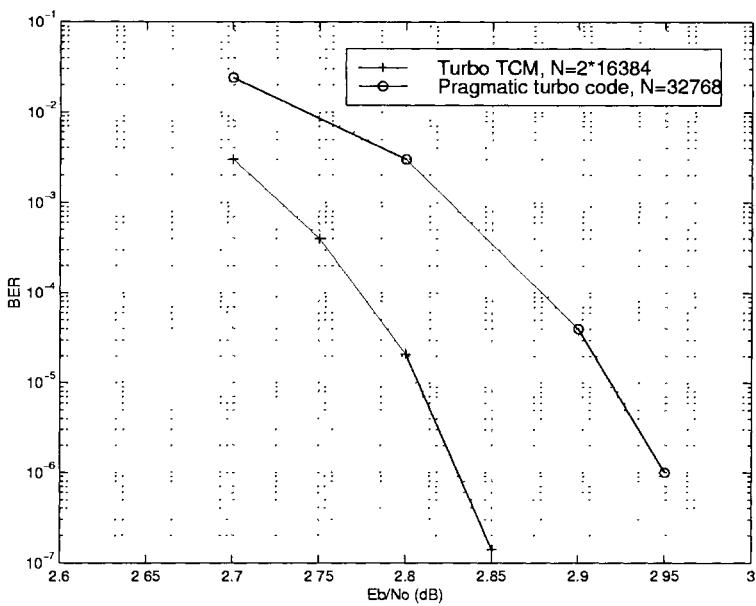


Fig. 9.17: Performance comparison of the turbo trellis coded 16-QAM with systematic symbol puncturing and the pragmatic turbo coded 16-QAM with bandwidth efficiency 2 bits/s/Hz and interleaver size 32768 on an AWGN channel, the number of iterations 8

9.4 I-Q Turbo Coded Modulation for Fading Channels

The performance of a coded modulation scheme over a Rayleigh distributed fading channel depends strongly on the code minimum symbol-wise Hamming distance. In [15], Al-Semari and Fuja proposed I-Q TCM schemes that could have substantially higher minimum symbol-wise Hamming distances than the conventional TCM schemes. They showed that significant performance improvement could be achieved by using simple convolutional codes optimized in terms of minimum Hamming distance combined with I-Q parallel encoding and mapping. In this section, we use the turbo codes optimized for binary modulation combined with the I-Q approach to construct good bandwidth efficient coding schemes [12]. In an I-Q

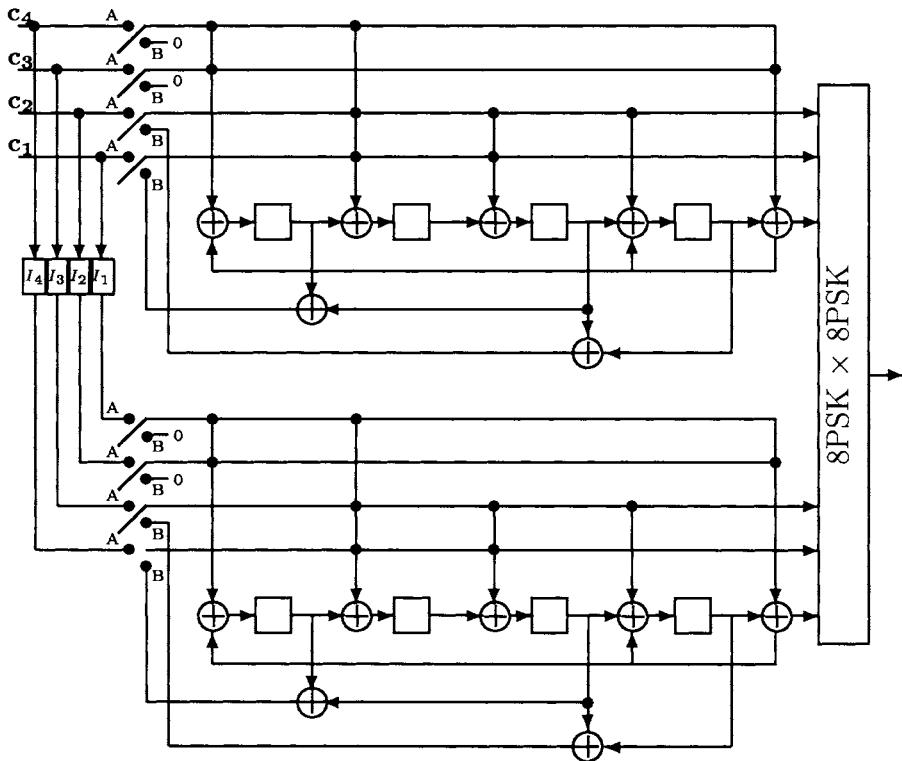


Fig. 9.18: Turbo trellis coded 8-PSK with systematic symbol puncturing

turbo encoder, each group of information bits is associated with its parity check digit which is alternately obtained from the first and second turbo component encoders. The coded digits from the two component encoders are mapped to the in-phase and quadrature components of a modulation signal constellation, separately, according to Gray code mapping. The coded modulation scheme has the underlying I-Q pragmatic structure which can achieve a larger minimum symbol-wise Hamming distance. In addition, the less complex symbol-by-symbol MAP iterative decoding algorithm can be employed in the decoder.

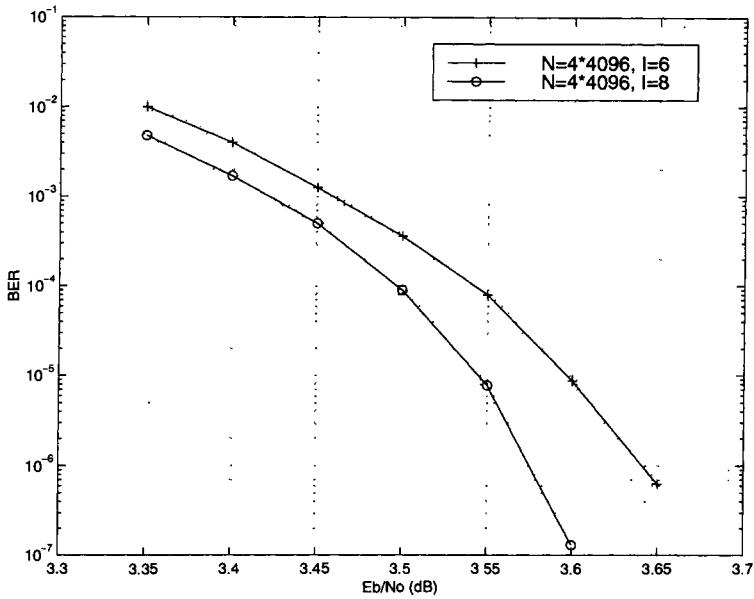


Fig. 9.19: Performance of the turbo trellis coded 8-PSK with systematic symbol puncturing with bandwidth efficiency 2 bits/s/Hz and interleaver size 16384 on an AWGN channel, the number of iterations I

9.4.1 I-Q Coded Modulation Structure

We consider a rate 1/2 turbo coded 16-QAM scheme with bandwidth efficiency 2 bits/s/Hz. The turbo encoder structure is illustrated in Fig. 9.20. Two identical component systematic recursive convolutional encoders with the rate 1/2 and the memory order ν are separated by a random interleaver with size N . The first encoder operates directly on the input sequence given by

$$\mathbf{c} = (c_1, c_2, \dots, c_N). \quad (9.27)$$

The output contains the information sequence

$$\mathbf{v}^{1s} = (v_1^{1s}, v_2^{1s}, \dots, v_N^{1s}) \quad (9.28)$$

and the parity check sequence

$$\mathbf{v}^{1p} = (v_1^{1p}, v_2^{1p}, \dots, v_N^{1p}). \quad (9.29)$$

The input sequence to the second encoder is permuted by the interleaver \mathbf{I} of size N . The parity check sequence of the second component encoder is deinterleaved to ensure that its output sequence

$$\mathbf{v}^{2p} = (v_1^{2p}, v_2^{2p}, \dots, v_N^{2p}) \quad (9.30)$$

has the same order as the input information sequence \mathbf{c} . By using an odd-even puncturing technique, the global turbo code rate is $1/2$ and the turbo code sequence, denoted by \mathbf{v} , consists of each information bit followed by its parity check digit which is alternately obtained from the first and the second component encoder. If we assume the interleaver size N is even, the code sequence can be represented as

$$\mathbf{v} = (v_1^{1s}, v_1^{1p}, v_2^{1s}, v_2^{2p}, \dots, v_{N-1}^{1s}, v_{N-1}^{1p}, v_N^{1s}, v_N^{2p}) \quad (9.31)$$

The code sequence is mapped to a 16-QAM constellation. In the given example, every two encoded digits select the in-phase or quadrature component of the modulated signal according to Gray code mapping. The output of the first turbo component encoder is mapped to the in-phase component of the modulated signal while the output of the second encoder is mapped to the quadrature component.

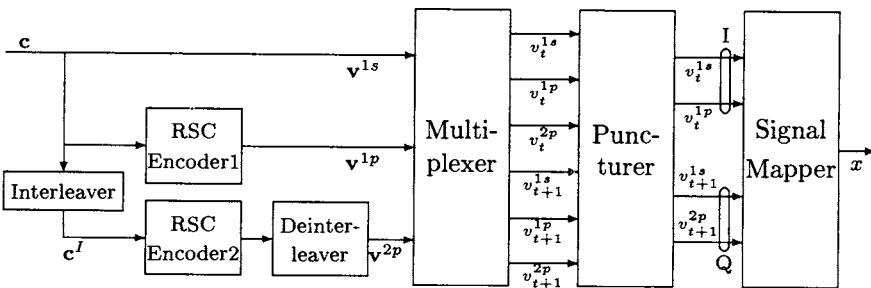


Fig. 9.20: I-Q turbo trellis coded 16-QAM

This I-Q coded modulation approach can be applied to produce coding schemes with high bandwidth efficiency. For example, to

transmit 4 bits/s/Hz, a rate 2/3 turbo coded 64-QAM is used. The three output digits from the first turbo component encoder specify the in-phase component of a 64-QAM signal according to the Gray code mapping, while the three output digits from the second component encoder specify the quadrature component of a 64-QAM signal. With an even-odd puncturing, one 64-QAM signal is produced every two turbo coding cycles.

9.4.2 The Decoder

In the proposed system, each group of information bits associated with its parity check digit is mapped to the signal subsets in the signal constellation. An iterative decoder with the symbol-by-symbol MAP algorithm, which has been derived for a non-binary trellis can be applied. The algorithm has lower complexity than the bit-by-bit MAP algorithm, as the receiver does not need to calculate the log-likelihood ratio (LLR) for each encoded binary digit based on the received noisy symbol before passing it to the binary turbo decoder [4] or convert the extrinsic information between a bit level and a symbol level [9].

Furthermore, we assume the channel fading attenuation can be perfectly recovered at the receiver, and hence channel state information may be used to decode turbo codes for fading channels. In this case, the branch metric should be modified by multiplying the transmitted signals in the trellis by the channel attenuations. This metric is not the optimum one, but the use of channel state information gives better performance than decoding with the standard Euclidean distance metric.

9.4.3 Performance of I-Q Turbo Coded Modulation on Fading Channels

The bit error rate performance of a coding scheme with bandwidth efficiency 2 bits/s/Hz on an independent Rayleigh fading channel is estimated by simulations. The generator polynomials are $\mathbf{g}_0 = (23)$ and $\mathbf{g}_1 = (35)$.

In the simulation, iterative decoding with symbol-by-symbol

Log-MAP algorithm is employed. The interleaver size is chosen to be 1024 and 4096. The number of iterations is 8. The simulation results are shown in Fig. 9.21.

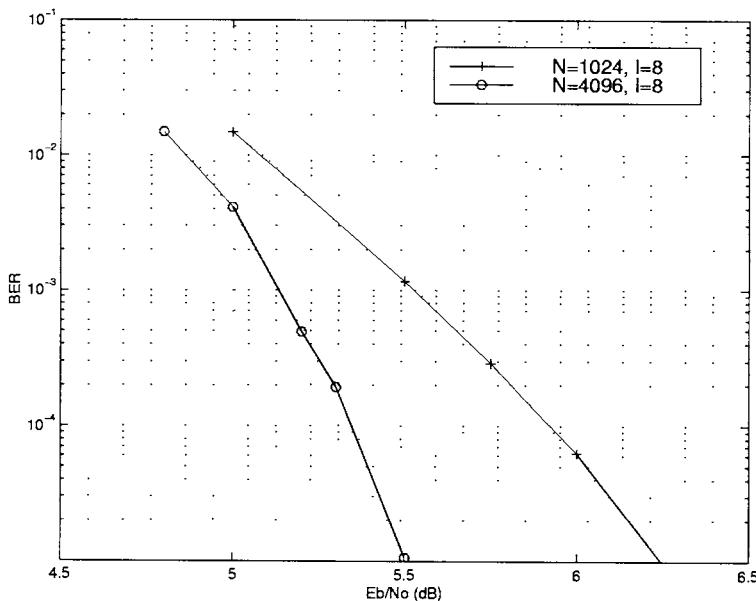


Fig. 9.21: Performance of the I-Q turbo coded 16-QAM with bandwidth efficiency 2 bits/s/Hz and various interleaver sizes on a Rayleigh fading channel

This scheme is also compared with the pragmatic binary turbo coded modulation of the same spectral efficiency reported in [4]. We simulated a Log-MAP decoding with 8 iterations for the pragmatic binary turbo coded modulation of interleaver size 4096. The result is shown in Fig. 9.22. It is clear that the I-Q scheme outperforms the pragmatic scheme by about 0.5 dB at a BER of 10^{-5} for the same bandwidth efficiency and interleaver size.

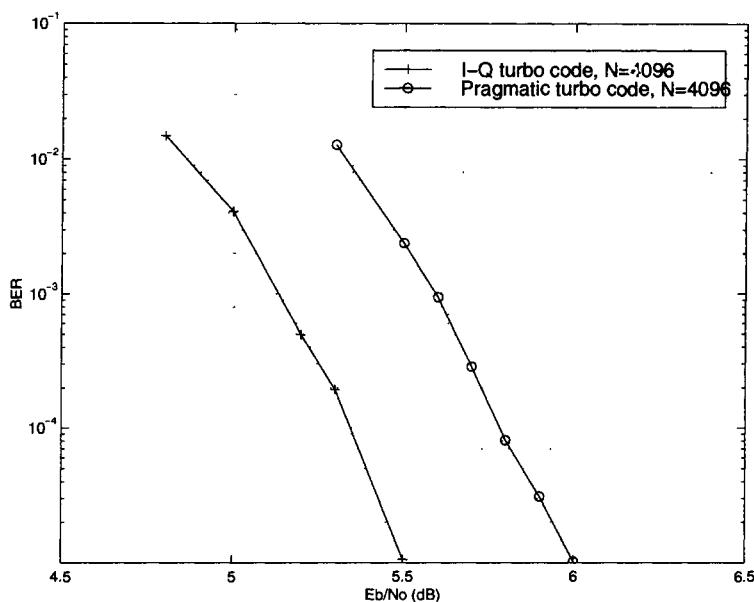


Fig. 9.22: Performance comparison of the I-Q turbo coded 16-QAM and the pragmatic turbo coded 16-QAM with bandwidth efficiency 2 bits/s/Hz and interleaver size 4096 on a Rayleigh fading channel

Bibliography

- [1] G. Ungerboeck, “Channel coding with multilevel/phase signalling,” *IEEE Trans. Inform Theory*, vol. 25, no. 1, Jan. 1982, pp. 55-67.
- [2] J. Hagenauer, P. Robertson, and L. Papke, “Iterative (‘Turbo’) decoding of systematic convolutional codes with the MAP and SOVA algorithms,” in *Proc. of the 1994 ITG Conference on Source and Channel Coding*, Munich, Oct. 1994, pp. 1-9.
- [3] B. Vucetic, “Iterative decoding algorithm,” in *PIMRC’97*, Helsinki, Finland, Sep. 1997, pp. 99-120.
- [4] S. LeGoff, A. Glavieux, and C. Berrou, “Turbo codes and high efficiency modulation,” in *Proc. of IEEE ICC’94*, New Orleans, LA, May 1994, pp. 645-649.
- [5] H. Imai and S. Hirakawa, “A new multilevel coding method using error correcting codes,” *IEEE Trans. Inform. Theory*, vol. 23, no. 3, May 1977, pp. 371-377.
- [6] P. Robertson, and T. Woerz, “Novel coded modulation scheme employing turbo codes,” *Electronics Letters*, vol. 31, no. 18, Aug. 1995, pp. 1546-1547.
- [7] P. Robertson and T. Woerz, “Bandwidth efficient turbo trellis-coded modulation using punctured component codes,” *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 2, Feb. 1998, pp. 206-218.
- [8] L. U. Wachsmann and J. Huber, “Power and bandwidth efficient digital communication using turbo codes in multilevel

- codes," *European Trans. Telecommun.*, vol. 6, no. 5, Sep./Oct. 1995, pp. 557-567.
- [9] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Parallel concatenated trellis coded modulation," in *Proc. of IEEE ICC'96*, 1996, pp. 974-978.
- [10] T. M. Duman, "Turbo codes and turbo-coded modulation: Analysis and performance bounds," Ph.D. dissertation, Elect. Comput. Eng. Dep., Northeastern Univ., Boston, MA, May 1998.
- [11] D. Divsalar and F. Pollara, "On the design of turbo codes," *TDA Progress Report 42-123*, JPL, Nov. 1995, pp. 99-121.
- [12] J. Yuan, B. Vucetic, and W. Feng, "I-Q turbo coded modulation," in *Proc. of IEEE GLOBECOM'98, Communications Theory Mini Conference*, Sydney, Australia, Nov. 1998, pp. 191-195.
- [13] W. Blackert and S. Wilson, "Turbo trellis coded modulation," in *Proc. CISS'96*, 1996.
- [14] S. A. Barbulescu, W. Farrell, P. Gray, and M. Rice, "Bandwidth efficient turbo coding for high speed mobile satellite communications," in *Proc. International Symposium on Turbo Codes*, Brest, France, Sep. 1997, pp. 119-126.
- [15] S. A. Al-Semari and T. E. Fuja, "I-Q TCM: Reliable communication over the Rayleigh fading channel close to the cutoff rate", *IEEE Trans. Inform Theory*, vol. 43, no. 1, Jan. 1997, pp. 250-262.
- [16] B. Vucetic, W. Feng, and J. Yuan, "Performance of turbo and serial concatenated convolutional codes," Technical Report, The University of Sydney, Aug. 1997.

Chapter 10

Applications of Turbo Codes

Turbo and serial concatenated convolutional codes have been proposed for various communication systems, such as deep space, cellular mobile and satellite communication networks. The standard turbo codes are presented in this chapter.

10.1 Turbo Codes for Deep Space Communications

NASA's next generation deep space transponder will contain a turbo code [1]. The Consultative Committee for Space Data Systems (CCSDS) has adopted 16-state rate 1/2, 1/3, 1/4 and 1/6 turbo codes as a new standard for telemetry channel coding [2]-[5]. The turbo encoder block diagram is shown in Fig. 10.1 [3].

The rate 1/3 turbo code is obtained by parallel concatenation of two identical 16-state rate 1/2 RSC encoders, where the systematic information bit of the second component encoder is eliminated. The generator matrix of the RSC code is given by

$$\mathbf{G}(D) = \left[1 \quad \frac{1+D+D^3+D^4}{1+D^3+D^4} \right]$$

The rate 1/2 turbo code is obtained by puncturing every other parity symbol of each component code in the rate 1/3 turbo encoder.

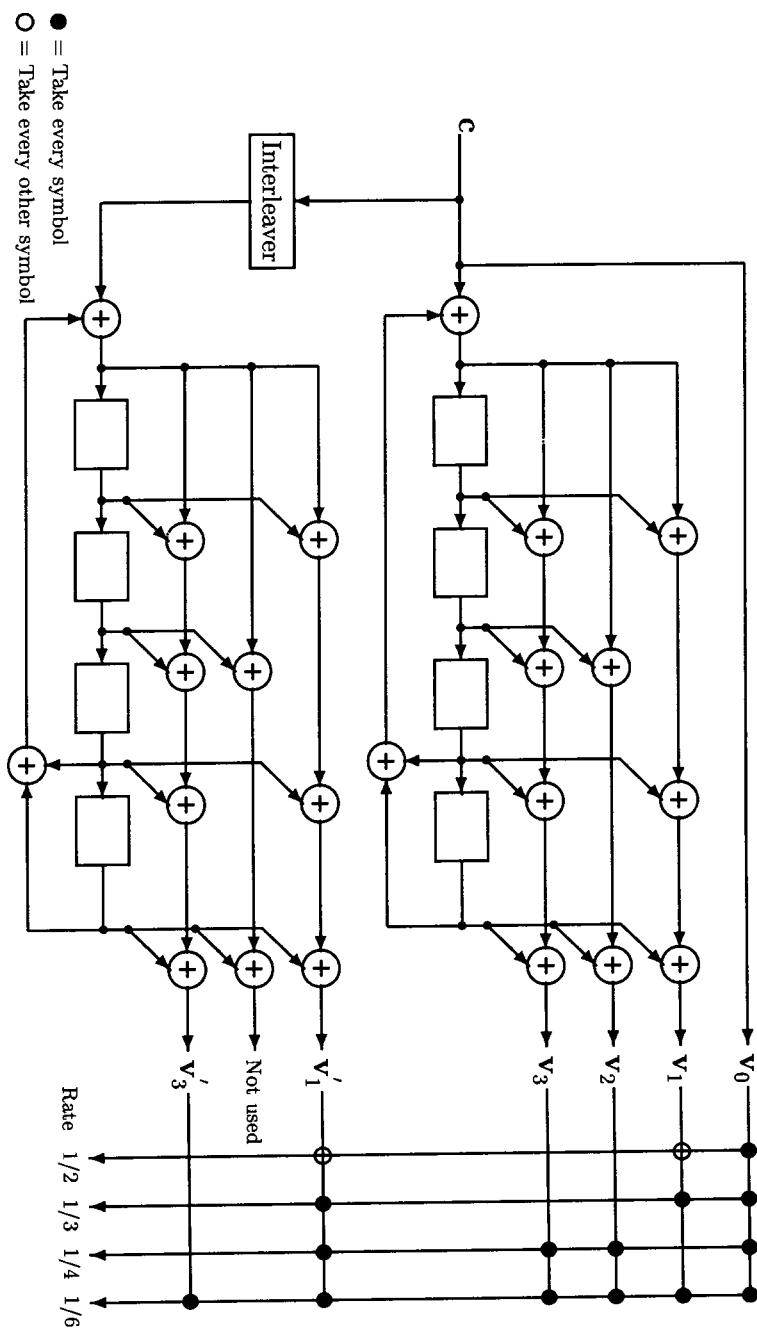


Fig. 10.1: CCSDS turbo encoder block diagram

The rate 1/4 turbo code is obtained by parallel concatenation of a 16-state rate 1/3 RSC encoder and a 16-state rate 1/2 RSC encoder whose systematic information bit is eliminated. The generator matrix of the rate 1/3 RSC code is given by

$$\mathbf{G}(D) = \left[1 \quad \frac{1+D^2+D^4}{1+D^3+D^4} \quad \frac{1+D+D^2+D^3+D^4}{1+D^3+D^4} \right]$$

The rate 1/2 RSC code is the same as the component code in the rate 1/3 turbo encoder.

The rate 1/6 turbo code is obtained by parallel concatenation of a 16-state rate 1/4 RSC encoder and a 16-state rate 1/3 RSC encoder whose systematic information bit is eliminated. The generator matrices of the rate 1/4 and 1/3 RSC codes are given by

$$\mathbf{G}(D) = \left[1 \quad \frac{1+D+D^3+D^4}{1+D^3+D^4} \quad \frac{1+D^2+D^4}{1+D^3+D^4} \quad \frac{1+D+D^2+D^3+D^4}{1+D^3+D^4} \right]$$

and

$$\mathbf{G}(D) = \left[1 \quad \frac{1+D+D^3+D^4}{1+D^3+D^4} \quad \frac{1+D+D^2+D^3+D^4}{1+D^3+D^4} \right]$$

respectively.

10.2 Turbo Codes for CDMA2000

In CDMA2000 proposal, turbo codes are recommended for both forward and reverse supplemental channels in the 3rd generation of wideband code division multiple access (CDMA) cellular mobile systems [6]. The block diagram for the reverse link turbo encoder is shown in Fig. 10.2. The component codes are identical (3, 1, 3) RSC codes. The generator matrix of the RSC code is given by

$$\mathbf{G}(D) = \left[1 \quad \frac{1+D^2+D^3}{1+D+D^3} \quad \frac{1+D+D^2+D^3}{1+D+D^3} \right]$$

By puncturing the component encoder outputs, variable code rates of 1/4, 1/2 and 1/3 can be achieved. The rate 1/4 turbo code is obtained by puncturing the parity digits \mathbf{v}_2 and \mathbf{v}'_2 of the two component encoders alternately. Puncturing the parity digits \mathbf{v}_1 and \mathbf{v}'_1 results in a rate 1/3 turbo code. For the rate 1/2 turbo

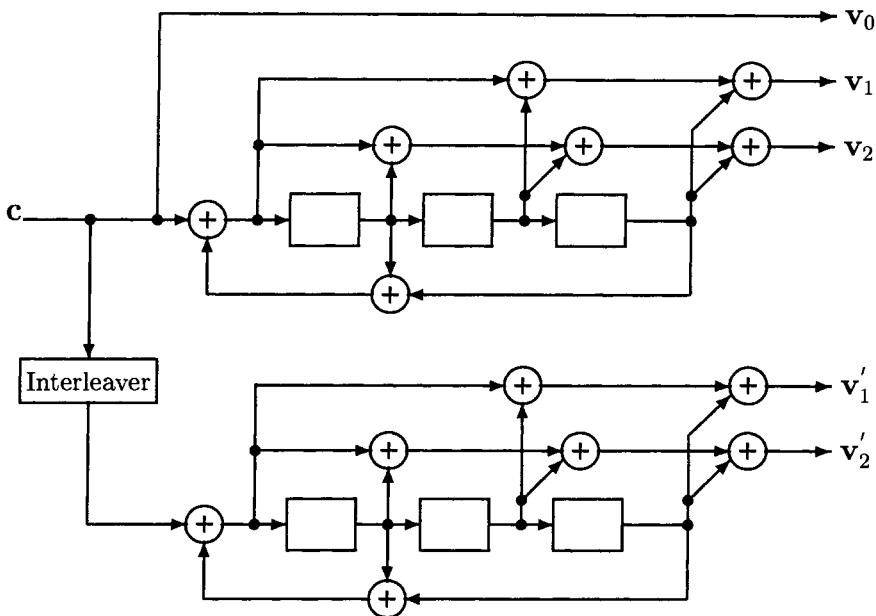


Fig. 10.2: The reverse link turbo encoder for CDMA2000

code, the parity digits v_2 and v'_2 , and every other parity digits v_1 and v'_1 for both encoders are punctured.

The forward link turbo encoder is based on two identical (3, 1, 3) RSC codes with generator matrix

$$\mathbf{G}(D) = \left[1 \quad \frac{1+D+D^3}{1+D^2+D^3} \quad \frac{1+D+D^2+D^3}{1+D^2+D^3} \right]$$

For the rate 1/4 turbo code, the parity digits v_1 from the first component encoder and v'_2 from the second component encoder are alternately punctured. For the rate 1/3 turbo code, the parity digits v_2 and v'_2 for both encoders are punctured. The rate 1/2 turbo code is generated in a similar way as the punctured rate 1/2 code for the reverse link.

10.3 Turbo Codes for 3GPP

There are two candidates for the 3rd generation cellular mobile communications based on the 3rd Generation Partnership Project

(3GPP) [7]. They include an 8-state rate 1/3 and 1/2 turbo code and a 4-state rate 1/3 serial concatenated convolutional code.

The 8-state turbo code is generated by parallel concatenation of two identical rate 1/2 RSC encoders with generator matrix

$$\mathbf{G}(D) = \left[1 \quad \frac{1+D+D^3}{1+D^2+D^3} \right]$$

The output of the turbo encoders is punctured to produce coded digits corresponding to the desired code rate of 1/3 or 1/2. For rate 1/3, the systematic information bit of the second component encoder is punctured. For rate 1/2, the parity digits produced by the two component encoders are alternately punctured. The encoder structure of the 8-state rate 1/3 turbo code is shown in Fig. 10.3.

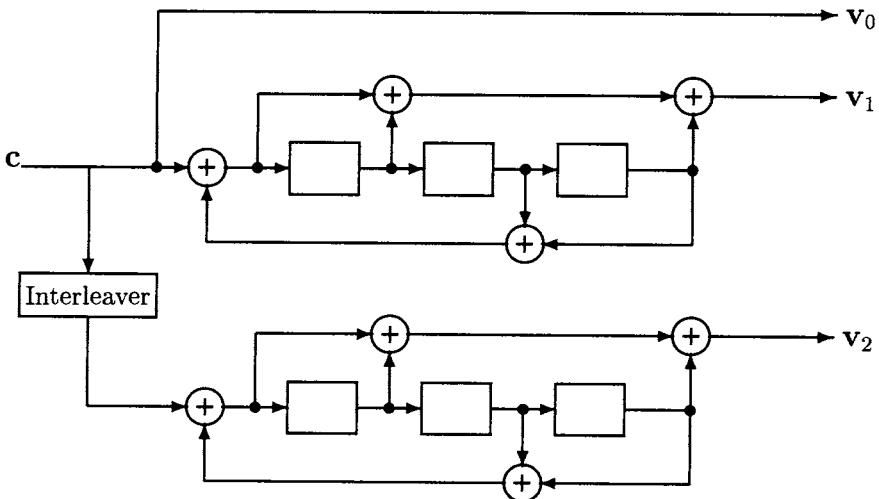


Fig. 10.3: The turbo encoder for 3GPP

The encoder structure of the 4-state rate 1/3 serial concatenated convolutional code is illustrated in Fig. 10.4. The inner code is a rate 1/2 RSC code with generator matrix

$$\mathbf{G}(D) = \left[1 \quad \frac{1+D^2}{1+D+D^2} \right]$$

The outer code is of rate 2/3 obtained by puncturing a rate 1/2 RSC code with the same generator matrix as for the inner code.

The rate 2/3 is obtained by puncturing every other parity-check digit. The puncturing matrix is given by

$$\mathbf{P} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

Note that trellis termination is performed in both the turbo and serial concatenated encoders. In the serial encoder, the tailing bits of the outer encoder are included in the interleaver.

10.4 Turbo Codes for Satellite Communications

Turbo codes are also recommended for INMARSAT's new mobile multimedia service [8] [9]. A data rate of 64 kbits/s is provided based on the combined turbo coding and 16-QAM modulation. In addition, INTELSAT is currently investigating the application of turbo codes for digital services.

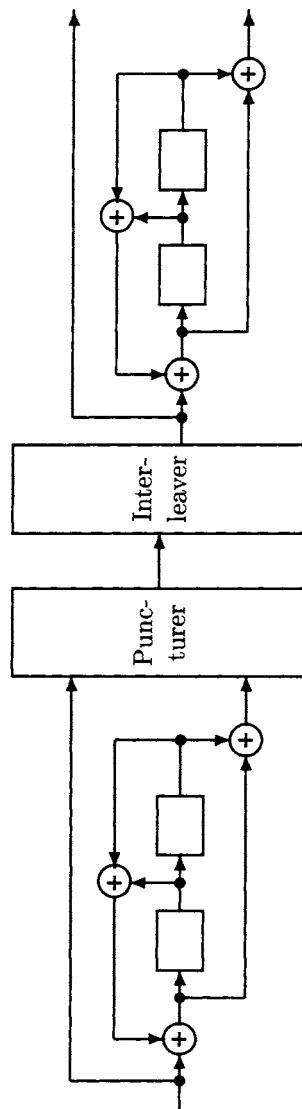


Fig. 10.4: The serial concatenated convolutional encoder for 3GPP

Bibliography

- [1] C. D. Edwards, C. T. Steilzvied, L. J. Deutsch and L. Swanson, “NASA’s deep-space telecommunications road MAP,” TMO Progress Report 42-126, Feb. 1999.
- [2] E. Vassallo, G. P. Calzolari, S. Benedetto, and G. Montorsi, “More performant codes for deep-space missions: the turbo codes option,”
- [3] CCSDS recommendation for telemetry channel coding, Jun. 1998.
- [4] D. Divsalar and F. Pollara, “Turbo codes for deep-space communications,” TDA Progress Report, 42-120, Feb. 1995.
- [5] D. Divsalar and F. Pollara, “Multiple turbo codes for deep-space communications,” TDA Progress Report, 42-121, May. 1995.
- [6] The cdma2000 ITU-R RTT Candidate Submission (0.18), Jul. 1998.
- [7] 3rd Generation Partnership Project (3GPP) Technical Specification Group: Radio Access Network Working Group 1, Multiplexing and Channel Coding, TS 25. 212 v1.1.0, June 1999.
- [8] H. Feldman and D. V. Ramana, “An introduction to Inmarsat’s new mobile multimedia service,” in *Proc. Sixth Int. Mobile Satellite Conf.*, Ottawa, June 1999, pp. 226-229.
- [9] E. Trachman and T. Hart, “Research elements leading to the development of Inmarsat’s new mobile multimedia services,”

- in *Proc. Sixth Int. Mobile Satellite Conf.*, Ottawa, June 1999, pp. 209-212.
- [10] S. A. Barbulescu, W. Farrell, P. Gray, and M. Rice, "Bandwidth efficient turbo coding for high speed mobile satellite communications," in *Proc. International Symposium on Turbo Codes*, Brest, France, Sep. 1997, pp. 119-126.

Index

a posteriori probability, 118, 123, 138
a priori information, 277
a priori probability, 145, 159
additive white Gaussian noise, 6
Al-Semari S. A., 288
asymptotic behavior, 170
asymptotic coding gain, 28
augmented state diagram, 63
average upper bound, 86

backward recursion, 131, 145
Bahl L. R., 30
bandwidth efficiency, 5, 263, 268, 283, 290, 292
Benedetto S., 100, 282
Berrou C., 1, 73
bidirectional SOVA, 127
binary symmetric channel, 4
bit error probability, 6, 20, 26, 80
bit error rate, 121
block code, 5, 13
block interleaver, 200
 helical simile, 204
 odd-even, 202
branch, 31, 61
branch metric, 124, 173
burst error, 73

carrier, 3
catastrophic code, 52
CCSDS, 297
cellular mobile radio, 232
channel, 3
channel capacity, 1, 6
channel decoder, 4
channel encoder, 3
channel model, 4
channel noise, 4
channel state information, 238
Cocke J., 30
code division multiple access, 299
code matched interleaver, 213, 219, 220
code performance, 164
code rate, 3
coded modulation, 5
codeword, 3
coding gain, 28, 263
communication system, 2
complementary error function, 25
component code, 73, 107
compound error path, 82
concatenated code, 8, 73
conditional weight enumerating function, 22, 81
constellation, 266

- controller canonical form, 48
convolutional code, 5, 37
convolutional interleaver, 206, 213, 222
cross entropy, 165
cutoff rate, 87
cyclic redundancy check, 165
cyclic shift interleaver, 208, 222
data rate, 1, 2
data transmission, 103
decision depth, 125
deinterleaver, 193
demodulator, 4
digital communication, 1
discrete channel, 4
distance spectral line, 95
distance spectrum, 26, 93, 103, 213
dithered golden interleaver, 213
Divsalar D., 100, 211, 282
Doppler shift, 232
Duman T., 87
effective free distance, 92, 100, 111
effective free Euclidean distance, 284
Elias P., 37
encoder memory, 44
equivalent generator matrix, 46
error, 3
error coefficient, 25, 26, 93, 246
error control coding, 1
error event probability, 24
error floor, 171
error path, 82
error pattern, 196
concatenated, 201
error performance bound, 80, 240
Euclidean distance, 20
extrinsic information, 162, 176, 277, 285
fading channel, 254, 255
fade rate, 233
fading channel, 233, 292
correlated, 258
independent, 256
power spectral density, 235
fading channel capacity, 236
feedback shift register, 47
feedforward convolutional code, 87
feedforward shift register, 47
finite-state Markov process, 117, 118
Forney G. D., Jr., 30, 37, 73, 206
forward recursion, 130, 145
free distance, 88
frequency dispersion, 233
Fuja T. E., 288
Gallager, 87
Gallieo, 8
Gaussian random variable, 20
generating function, 63, 65
generator matrix, 14, 40, 42, 74, 100
generator polynomial, 41
Glavieux A., 1, 73
Gray mapping, 264, 291
greatest common divisor, 51
Hamming code, 22

Hamming distance, 17, 25
Hamming weight, 17, 240
hard decision decoding, 4
Hirakawa S., 267
Huber J., 269

I-Q trellis coded modulation, 288
I-Q turbo coded modulation, 290
Imai H., 267
information source, 2
inner code, 73, 107
input pattern, 198
length, 212
low weight, 199, 213
square or rectangular, 201
input-output weight enumerating function, 21
input-redundancy weight enumerating function, 21
interleaver, 73, 76, 194
delay, 195
device, 194
mapping function, 194
size, 166, 197
storage capacity, 195
structure, 198
vector, 194
interleaving performance gain, 90, 197
intrinsic information
fading channel, 255
iterative decoding, 9, 157, 196
fading channel, 250
iterative MAP decoding, 159, 163, 178

iterative SOVA decoding, 171, 177, 180
Jelinek F., 30
joint extrinsic and systematic information, 277, 279

linear block code, 13
linear feedforward shift register, 38
log-likelihood ratio, 128, 138, 158, 266, 274
fading channel, 252
Log-MAP algorithm, 151, 274

Mason's rule, 63
Massey-Berlekamp decoding, 8
Max-Log-MAP algorithm, 149
maximum a posteriori probability, 74, 118, 138
fading channel, 252
maximum likelihood, 123
maximum likelihood decoding, 18, 30
maximum likelihood path, 124
maximum spectral efficiency, 6
message, 2
military communication, 104
minimum distance decoding, 19
minimum Euclidean distance, 29
minimum free distance, 62
minimum free Euclidean distance, 62
minimum Hamming distance, 17, 25
minimum path metric, 129
mobile communication, 104, 258, 297

- modulation, 3
 M-PSK, 264, 283
 M-QAM, 264, 283
 16-QAM, 280, 286, 290
 64-QAM, 292
 8-PSK, 272, 279, 287
 BPSK, 23
- modulator, 3
- Montorsi G., 100, 282
- multilevel coding, 267
- multilevel turbo code, 267
- multipath fading, 232
- multipath propagation, 232
- multipath waves, 232
- multistage decoding, 269
- NASA, 297
- non-uniform interleaver, 210
- nonrecursive convolutional code, 87
- normalized fade rate, 235
- number of iterations, 160
- observer canonical form, 48
- odd-even interleaver, 202
- Odenwalder J. P., 7
- optimal distance spectrum, 103
- optimum decoding, 157
- outer code, 73, 107
- pairwise error probability, 24
- parallel concatenated convolutional code, 74
- parity check, 15
- parity check matrix, 16, 50
- path, 31, 61
- path metric, 124, 173
- perfect codes, 21
- personal mobile communication, 104
- Pollara F., 100, 211, 282
- pragmatic turbo coded modulation, 264, 293
- primitive polynomial, 100
- pseudo-random interleaver, 77, 210
- punctured convolutional code, 67
- puncturing, 79, 167, 265
 matrix, 79
 period, 69
- Q*-function, 25
- Q*-function bound, 25
- Ramsey J. L., 206
- random code, 196
- random interleaver, 209, 220
- rational transfer function, 48
- Raviv J., 30
- Rayleigh fading, 233
- receiver, 4
- recursive systematic convolutional code, 74, 87
- Reed-Solomon code, 8, 73
- Rician factor, 236
- Rician fading, 235
- S*-constraint, 211
- S*-random interleaver, 211, 220, 222
- Salehi M., 87
- Sason I., 87
- satellite communication, 7, 103, 258, 297
- satellite mobile communication, 104

- serial concatenated code, 73, 107
serial concatenated convolutional code, 74, 107, 182
Shamai S., 87
Shannon C. E., 1
Shannon capacity limit, 7, 74, 196
signal flow graph, 63
signal-to-noise ratio, 6
signalling interval, 3
significant spectral line, 97
single error path, 82
sliding window SOVA, 135
soft decision, 73, 131
soft decision decoding, 5
soft output, 117, 126
soft output Viterbi algorithm, 74, 117, 278
 fading channel, 254
soft-input soft-output, 147
soft-input/soft-output decoder, 73
source decoder, 4
source encoder, 2
space communication, 73, 103, 297
spectral efficiency, 5
spread interleaver, 211
state, 31, 61
state diagram, 58
survivor, 124
symbol error rate, 121
symbol rate, 3
symbol-wise Hamming distance, 288
system bandwidth, 5
systematic block code, 15
systematic convolutional code, 45
systematic encoder, 49, 53
systematic information, 277
thermal noise, 4
Thitimajshima P., 1, 73
transfer function, 82, 88
 modified, 83
transmitter, 2
trellis based decoding algorithm, 117
trellis coded modulation, 9, 73, 263
trellis construction, 30
trellis diagram, 60
trellis termination, 77
turbo code, 1, 9
turbo coding, 73
turbo decoding, 74
turbo encoder, 74
turbo trellis coded modulation, 263, 270
 puncturing of parity digits, 270
 puncturing of systematic bits, 282
unequal error protection, 66
Ungerboeck G., 263
uniform interleaver, 81
upper bound
 bit error probability, 26, 85, 108, 246
 pairwise error probability, 241
word error probability, 25

Viterbi A. M., 87
Viterbi algorithm, 30, 117, 122
voice transmission, 104
Voyager, 7

Wachsmann L. U., 269
weight distribution, 20, 63, 80
weight enumerating function,
 21
Wolf J. K., 30
word error probability, 20, 25
word error rate, 121