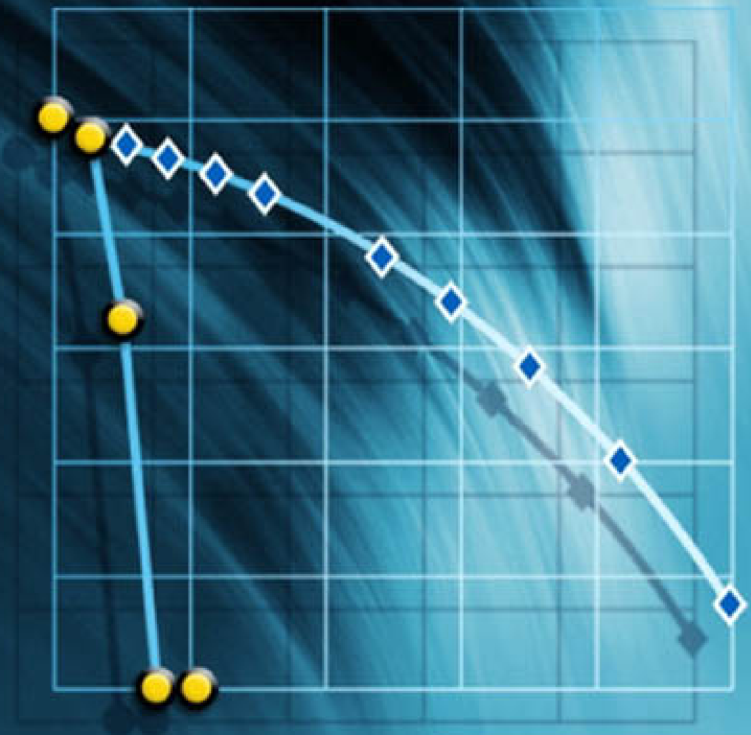
Jorge Castiñeira Moreira
Patrick Guy Farrell

Essentials of Error-Control Coding



Companion Website

WILEY

ESSENTIALS OF ERROR-CONTROL CODING

Jorge Castiñeira Moreira

University of Mar del Plata, Argentina

Patrick Guy Farrell

Lancaster University, UK

John Wiley & Sons, Ltd

ESSENTIALS OF ERROR-CONTROL CODING

ESSENTIALS OF ERROR-CONTROL CODING

Jorge Castiñeira Moreira

University of Mar del Plata, Argentina

Patrick Guy Farrell

Lancaster University, UK

John Wiley & Sons, Ltd

Telephone (+44) 1243 779777

Email (for orders and customer service enquiries): cs-books@wiley.co.uk
Visit our Home Page on www.wiley.com

Other Wiley Editorial Offices

John Wiley & Sons Inc., 111 River Street, Hoboken, NJ 07030, USA

Jossey-Bass, 989 Market Street, San Francisco, CA 94103-1741, USA

Wiley-VCH Verlag GmbH, Boschstr. 12, D-69469 Weinheim, Germany

John Wiley & Sons Australia Ltd, 42 McDougall Street, Milton, Queensland 4064, Australia

John Wiley & Sons (Asia) Pte Ltd, 2 Clementi Loop #02-01, Jin Xing Distripark, Singapore 129809

John Wiley & Sons Canada Ltd, 6045 Freemont Blvd, Mississauga, ONT, L5R 4J3, Canada

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library

ISBN-13 978-0-470-02920-6 (HB)
ISBN-10 0-470-02920-X (HB)

Typeset in 10/12pt Times by TechBooks, New Delhi, India.
Printed and bound in Great Britain by Antony Rowe Ltd, Chippenham, England.
This book is printed on acid-free paper responsibly manufactured from sustainable forestry in which at least two trees are planted for each one used for paper production.

We dedicate this book to

my son Santiago José,
Melisa and Belén,
Maria, Isabel, Alejandra and Daniel,
and the memory of my Father.

J.C.M.

and to all my families and friends.

P.G.F.

Contents

Preface

The subject of this book is the detection and correction of errors in digital information. Such errors almost inevitably occur after the transmission, storage or processing of information in digital (mainly binary) form, because of noise and interference in communication channels, or imperfections in storage media, for example. Protecting digital information with a suitable error-control code enables the efficient detection and correction of any errors that may have occurred.

Error-control codes are now used in almost the entire range of information communication, storage and processing systems. Rapid advances in electronic and optical devices and systems have enabled the implementation of very powerful codes with close to optimum error-control performance. In addition, new types of code, and new decoding methods, have recently been developed and are starting to be applied. However, error-control coding is complex, novel and unfamiliar, not yet widely understood and appreciated. This book sets out to provide a clear description of the essentials of the topic, with comprehensive and up-to-date coverage of the most useful codes and their decoding algorithms. The book has a practical engineering and information technology emphasis, but includes relevant background material and fundamental theoretical aspects. Several system applications of error-control codes are described, and there are many worked examples and problems for the reader to solve.

The book is an advanced text aimed at postgraduate and third/final year undergraduate students of courses on telecommunications engineering, communication networks, electronic engineering, computer science, information systems and technology, digital signal processing, and applied mathematics, and for engineers and researchers working in any of these areas. The book is designed to be virtually self-contained for a reader with any of these backgrounds. Enough information and signal theory, and coding mathematics, is included to enable a full understanding of any of the error-control topics described in the book.

Chapter 1 provides an introduction to information theory and how it relates to error-control coding. The theory defines what we mean by information, determines limits on the capacity of an information channel and tells us how efficient a code is at detecting and correcting errors. Chapter 2 describes the basic concepts of error detection and correction, in the context of the parameters, encoding and decoding of some simple binary block error-control codes. Block codes were the first type of error-control code to be discovered, in the decade from about 1940 to 1950. The two basic ways in which error coding is applied to an information system are also described: forward error correction and retransmission error control. A particularly useful kind of block code, the cyclic code, is introduced in Chapter 3, together with an example of a practical application, the cyclic redundancy check (CRC) code for the Ethernet standard. In Chapters 4 and 5 two very effective and widely used classes of cyclic codes are described, the Bose–Chaudhuri–Hocquenghem (BCH) and Reed–Solomon (RS) codes, named after their inventors. BCH codes can be binary or non-binary, but the RS codes are non-binary and are particularly effective in a large number of error-control scenarios. One of the best known of these, also described in Chapter 5, is the application of RS codes to error correction in the compact disk (CD).

Not long after the discovery of block codes, a second type of error-control codes emerged, initially called recurrent and later convolutional codes. Encoding and decoding even a quite powerful convolutional code involves rather simple, repetitive, quasi-continuous processes, applied on a very convenient trellis representation of the code, instead of the more complex block processing that seems to be required in the case of a powerful block code. This makes it relatively easy to use maximum likelihood (soft-decision) decoding with convolutional codes, in the form of the optimum Viterbi algorithm (VA). Convolutional codes, their trellis and state diagrams, soft-decision detection, the Viterbi decoding algorithm, and practical punctured and rate-compatible coding schemes are all presented in Chapter 6. Disappointingly, however, even very powerful convolutional codes were found to be incapable of achieving performances close to the limits first published by Shannon, the father of information theory, in 1948. This was still true even when very powerful combinations of block and convolutional codes, called concatenated codes, were devised. The breakthrough, by Berrou, Glavieux and Thitimajshima in 1993, was to use a special kind of interleaved concatenation, in conjunction with iterative soft-decision decoding. All aspects of these very effective coding schemes, called turbo codes because of the supercharging effect of the iterative decoding algorithm, are fully described in Chapter 7.

The final chapter returns to the topic of block codes, in the form of low-density parity check (LDPC) codes. Block codes had been found to have trellis representations, so that they could be soft-decision decoded with performances almost as good as those of convolutional codes. Also, they could be used in effective turbo coding schemes. Complexity remained a problem, however, until it was quite recently realized that a particularly simple class of codes, the LDPC codes discovered by Gallager in 1962, was capable of delivering performances as good or better than those of turbo codes when decoded by an appropriate iterative algorithm. All aspects of the construction, encoding, decoding and performance of LDPC codes are fully described in Chapter 8, together with various forms of LDPC codes which are particularly effective for use in communication networks.

Appendix A shows how to calculate the error probability of digital signals transmitted over additive white Gaussian noise (AWGN) channels, and Appendix B introduces various topics in discrete mathematics. These are followed by a list of the answers to the problems located at the end of each chapter. Detailed solutions are available on the website associated with this book, which can be found at the following address:

http://elaf1.fi.mdp.edu.ar/Error_Control

The website also contains additional material, which will be regularly updated in response to comments and questions from readers.

Acknowledgements

We are very grateful for all the help, support and encouragement we have had during the writing of this book, from our colleagues past and present, from many generations of research assistants and students, from the reviewers and from our families and friends. We particularly thank Damian Levin and Leonardo Arnone for their contributions to Chapters 7 and 8, respectively; Mario Blaum, Rolando Carrasco, Evan Ciner, Bahram Honary, Garik Markarian and Robert McEliece for stimulating discussions and very welcome support; and Sarah Hinton at John Wiley & Sons, Ltd who patiently waited for her initial suggestion to bear fruit.

List of Symbols

Chapter 1

α	probability of occurrence of a source symbol (Chapter 1)
δ, ε	arbitrary small numbers
σ	standard deviation
$\Omega(\alpha)$	entropy of the binary source evaluated using logs to base 2
B	bandwidth of a channel
C	capacity of a channel, bits per second
$\boldsymbol{c}$	code vector, codeword
C_s	capacity of a channel, bits per symbol
d, i, j, k, l, m, n	integer numbers
E_b	average bit energy
E_b/N_0	average bit energy-to-noise power spectral density ratio
$H(X)$	entropy in bits per second
$H(X^n)$	entropy of an extended source
$H(X/y_j)$	*a posteriori* entropy
$H(X/Y)$	equivocation
$H(Y/X)$	noise entropy
$H_b(X)$	entropy of a discrete source calculated in logs to base b
$I(x_i, y_j)$	mutual information of x_i, y_j
$I(X, Y)$	average mutual information
I_i	information of the symbol x_i
M	number of symbols of a discrete source
n	length of a block of information, block code length
$N_0/2$	noise power spectral density
n_f	large number of emitted symbols
p	error probability of the BSC or BEC
P	power of a signal
$P(x_i) = P_i$	probability of occurrence of the symbol x_i
$P(x_i/y_j)$	backward transition probability
$P(x_i, y_j)$	joint probability of x_i, y_j
$P(\boldsymbol{X}/\boldsymbol{Y})$	conditional probability of vector $\boldsymbol{X}$ given vector $\boldsymbol{Y}$
$P_{ij} = P(y_j/x_i)$	conditional probability of symbol y_j given x_i, also transition probability of a channel; forward transition probability
P_{ke}	error probability, in general k identifies a particular index

P_{N}	noise power
$\boldsymbol{P}_{\mathbf{ch}}$	transition probability matrix
Q_i	a probability
R	information rate
r_{b}	bit rate
s, r	symbol rate
S/N	signal-to-noise ratio
T	signal time duration
T_{s}	sampling period
W	bandwidth of a signal
x	variable in general, also a particular value of random variable X
X	random variable (Chapters 1, 7 and 8), and variable of a polynomial expression (Chapters 3, 4 and 5)
$x(t), s(t)$	signals in the time domain
x_i	value of a source symbol, also a symbol input to a channel
$x_k = x(kT_{\mathrm{s}})$	sample of signal $x(t)$
$\vert\vert\boldsymbol{X}\vert\vert$	norm of vector $\boldsymbol{X}$
y_j	value of a symbol, generally a channel output

Chapter 2

A	amplitude of a signal or symbol
A_i	number of codewords of weight i
D	stopping time (Chapter 2); D-transform domain variable
$d(\boldsymbol{c}_i, \boldsymbol{c}_j)$	Hamming distance between two code vectors
D_i	set of codewords
d_{min}	minimum distance of a code
$\boldsymbol{e}$	error pattern vector
F	a field
$f(\boldsymbol{m})$	redundancy obtained, code C_0, hybrid ARQ
$\boldsymbol{G}$	generator matrix
$\boldsymbol{g}_i$	row vector of generator matrix $\boldsymbol{G}$
g_{ij}	element of generator matrix
$\mathrm{GF}(q)$	Galois or finite field
$\boldsymbol{H}$	parity check matrix
$\boldsymbol{h}_j$	row vector of parity check matrix $\boldsymbol{H}$
k, n	message and code lengths in a block code
l	number of detectable errors in a codeword
m	random number of transmissions (Chapter 2)
$\boldsymbol{m}$	message vector
N	integer number
$P(i, n)$	probability of i erroneous symbols in a block of n symbols
$\boldsymbol{P}$	parity check submatrix
p_{ij}	element of the parity check submatrix
p_{prime}	prime number

P_{be}	bit error rate (BER)
P_{ret}	probability of a retransmission in ARQ schemes
$P_{\mathrm{U}}(E)$	probability of undetected errors
P_{we}	word or code vector error probability
q	power of a prime number p_{prime}
$q(\boldsymbol{m})$	redundancy obtained, code C_1, hybrid ARQ
$\boldsymbol{r}$	received vector
R_{c}	code rate
S	subspace of a vector space V (Chapter 2)
$\boldsymbol{S}$	syndrome vector (Chapters 2–5, 8)
s_i	component of a syndrome vector (Chapters 2–5, 8)
S_{d}	dual subspace of the subspace S
t	number of correctable errors in a codeword
t_{d}	transmission delay
T_{w}	duration of a word
$\boldsymbol{u} = (u_1, u_2, \ldots u_{n-1})$	vector of n components
V	a vector space
V_n	vector space of dimension n
$w(\boldsymbol{c})$	Hamming weight of code vector $\boldsymbol{c}$

Chapter 3

α^i	primitive element of Galois field GF(q) (Chapters 4 and 5, Appendix B)
β^i	root of minimal polynomial (Chapters 4 and 5, Appendix B)
$c(X)$	code polynomial
$c^{(i)}(X)$	i-position right-shift rotated version of the polynomial $c(X)$
$e(X)$	error polynomial
$g(X)$	generator polynomial
$m(X)$	message polynomial
$p(X)$	remainder polynomial (redundancy polynomial in systematic form) (Chapter 3),
$p_i(X)$	primitive polynomial
r	level of redundancy and degree of the generator polynomial (Chapters 3 and 4 only)
$r(X)$	received polynomial
$S(X)$	syndrome polynomial

Chapter 4

β_l, α^{jl}	error-location numbers
$\Phi_i(X)$	minimal polynomial
$\mu(X)$	auxiliary polynomial in the key equation
$\sigma(X)$	error-location polynomial (Euclidean algorithm)
τ	number of errors in a received vector

e_{jh}	value of an error
j_l	position of an error in a received vector
q_i, r_i, s_i, t_i	auxiliary numbers in the Euclidean algorithm (Chapters 4 and 5)
$r_i(X), s_i(X), t_i(X)$	auxiliary polynomials in the Euclidean algorithm (Chapters 4 and 5)
$W(X)$	error-evaluation polynomial

Chapter 5

ρ	a previous step with respect to μ in the Berlekamp–Massey (B–M) algorithm
$\sigma_{\mathrm{BM}}^{(\mu)}(X)$	error-location polynomial, B–M algorithm, μth iteration
d_μ	μth discrepancy, B–M algorithm
l_μ	degree of the polynomial $\sigma_{\mathrm{BM}}^{(\mu)}(X)$, B–M algorithm
$\hat{\boldsymbol{m}}$	estimate of a message vector
s_{RS}	number of shortened symbols in a shortened RS code
$Z(X)$	polynomial for determining error values in the B–M algorithm

Chapter 6

A_i	number of sequences of weight i (Chapter 6)
$A_{i,j,l}$	number of paths of weight i, of length j, which result from an input of weight l
$b_i(T)$	sampled value of $b_i(t)$, the noise-free signal, at time instant T
$C(D)$	code polynomial expressions in the D domain
c_i	ith branch of code sequence $\boldsymbol{c}$
$\boldsymbol{c_i}$	n-tuple of coded elements
$\boldsymbol{C}_m(D)$	multiplexed output of a convolutional encoder in the D domain
c_{ji}	jth code symbol of c_i
$C^{(j)}(D)$	output sequence of the jth branch of a convolutional encoder, in the D domain
$\boldsymbol{c_i^{(j)}} = (c_0^{(j)}, c_1^{(j)}, c_2^{(j)}, \ldots)$	output sequence of the jth branch of a convolutional encoder
d_{f}	minimum free distance of a convolutional code
d_{H}	Hamming distance
$G(D)$	rational transfer function of polynomial expressions in the D domain
$\boldsymbol{G}(D)$	rational transfer function matrix in the D domain
$G_i^{(j)}(D)$	impulse response of the jth branch of a convolutional encoder, in the D domain
$\boldsymbol{g_i^{(j)}} = (g_{i0}^{(j)}, g_{i1}^{(j)}, g_{i2}^{(j)}, \ldots)$	impulse response of the jth branch of a convolutional encoder
$[\mathrm{GF}(q)]^n$	extended vector space

H_0	hypothesis of the transmission of symbol '0'
H_1	hypothesis of the transmission of symbol '1'
J	decoding length
K	number of memory units of a convolutional encoder
$K+1$	constraint length of a convolutional code
K_i	length of the ith register of a convolutional encoder
L	length of a sequence
$M(D)$	message polynomial expressions in the D domain
$\boldsymbol{m_i}$	k-tuple of message elements
n_A	constraint length of a convolutional code, measured in bits
$\boldsymbol{P}_{\mathbf{p}}$	puncturing matrix
	$\boldsymbol{S}(D)$ state transfer function
$s_i(k)$	state sequences in the time domain
$s_i(t)$	a signal in the time domain
$S_i(D)$	state sequences in the D domain
$\boldsymbol{S_j} = (s_{0j}, s_{1j}, s_{2j}, \ldots)$	state vectors of a convolutional encoder
$\boldsymbol{s}_{\mathbf{r}}$	received sequence
s_{r_i}	ith branch of received sequence $\boldsymbol{s}_{\mathbf{r}}$
$s_{\mathrm{r},ji}$	jth symbol of s_{r_i}
$T(X)$	generating function of a convolutional code
$T(X, Y, Z)$	modified generating function
t_i	time instant
T_{p}	puncturing period

Chapter 7

$\alpha_i(u)$	forward recursion coefficients of the BCJR algorithm
$\beta_i(u)$	backward recursion coefficients of the BCJR algorithm
$\lambda_i(u)$, $\sigma_i(u, u')$, $\gamma_i(u', u)$	quantities involved in the BCJR algorithm
$\mu(x)$	measure or metric of the event x
$\mu(x, y)$	joint measure for a pair of random variables X and Y
$\mu_{\mathrm{MAP}}(x)$	maximum *a posteriori* measure or metric of the event x
$\mu_{\mathrm{ML}}(x)$	maximum likelihood measure or metric of the event x
μ_Y	mean value of random variable Y
$\pi(i)$	permutation
σ_Y^2	variance of a random variable Y
A	random variable of *a priori* estimates
D	random variable of extrinsic estimates of bits
E	random variable of extrinsic estimates
$E^{(i)}$	extrinsic estimates for bit i
$\mathrm{hist}_E(\xi/X = x)$	histogram that represents the probability density function
$I\{.\}$ interleaver permutation	$p_E(\xi/X = x)$
I_A, $I(X; A)$	mutual information between the random variables A and X
I_E, $I(X; E)$	mutual information between the random variables E and X

$I_E = T_{\mathrm{r}}(I_A, E_{\mathrm{b}}/N_0)$	extrinsic information transfer function
$J(\sigma)$	mutual information function
J_{MTC}	number of encoders in a multiple turbo code
$J^{-1}(I_A)$	inverse of the mutual information function
$L(x)$	metric of a given event x
$L(b_i)$	log likelihood ratio for bit b_i
$L(b_i/\boldsymbol{Y}), L(b_i/Y_1^n)$	conditioned log likelihood ratio given the received sequence $\boldsymbol{Y}$, for bit b_i
L_{c}	measure of the channel signal-to-noise ratio
$L_{\mathrm{c}}\boldsymbol{Y}^{(j)}$	channel information for a turbo decoder, jth iteration
$L_{\mathrm{e}}(b_i)$	extrinsic log likelihood ratio for bit b_i
$L_{\mathrm{e}}^{(j)}(b_i)$	extrinsic log likelihood ratio for bit b_i, j-th iteration
$L^{(j)}(b_i/\boldsymbol{Y})$	conditioned log likelihood ratio given the received sequence $\boldsymbol{Y}$, for bit b_i, jth iteration
$M_{\mathrm{I}} \times N_{\mathrm{I}}$	size of a block interleaver
n_Y	random variable with zero mean value and variance σ_Y^2
$p(x)$	probability distribution of a discrete random variable
$p(X_j)$	source marginal distribution function
$p_A(\xi/X = x)$	probability density function of *a priori* estimates A for $X = x$
$p_E(\xi/X = x)$	probability density function of extrinsic estimates E for $X = x$
p_{MTC}	the angular coefficient of a linear interleaver
$R_j(Y_j/X_j)$	channel transition probability
s_{MTC}	linear shift of a linear interleaver
$S_i^j = \{S_i, S_{i+1}, \ldots, S_j\}$	generic vector or sequence of states of a Hidden Markov source
u	current state value
u'	previous state value
$\boldsymbol{X} = X_1^n = \{X_1, X_2, \ldots, X_n\}$	vector or sequence of n random variables
$X_i^j = \{X_i, X_{i+1}, \ldots, X_j\}$	generic vector or sequence of random variables

Chapter 8

δQ_{ij}	difference of coefficients Q_{ij}^x
δR_{ij}	difference of coefficients R_{ij}^x
$\boldsymbol{A}$ and $\boldsymbol{B}$	sparse submatrices of the parity check matrix $\boldsymbol{H}$ (Chapter 8)
$A_{ij}^{(it)}$	*a posteriori* estimate in iteration number *it*
$\boldsymbol{d}$	decoded vector
$\hat{\boldsymbol{d}}$	estimated decoded vector
d_j	symbol nodes
$d_{\mathrm{c}}^{(i)}$	number of symbol nodes or bits related to parity check node h_i
$d_{\mathrm{v}}^{(j)}$	number of parity check equations in which the bit or symbol d_j participates

$\boldsymbol{dp}$	message packet code vector
dp_n	message packet in a fountain or linear random code
E_x	number of excess packets of a fountain or linear random code
$f_+(\lvert z_1\rvert, \lvert z_2\rvert)$, $f_-(\lvert z_1\rvert, \lvert z_2\rvert)$	look-up tables for an LDPC decoder implementation with entries $\lvert z_1\rvert$, $\lvert z_2\rvert$
f_j^x	*a priori* estimates of the received symbols
$\boldsymbol{G}_{\mathbf{fr}}$	fragment generator matrix
$\{G_{kn}\}$	generator matrix of a fountain or linear random code
h_i	parity check nodes
$I_{\mathrm{E,SND}}(I_\mathrm{A}, d_\mathrm{v}, E_\mathrm{b}/N_0, R_\mathrm{c})$	EXIT chart for the symbol node decoder
$I_{\mathrm{E,PCND}}(I_\mathrm{A}, d_\mathrm{c})$	EXIT chart for the parity check node decoder
$L(b_1 \oplus b_2)$, $L(b_1)[\oplus]L(b_2)$	LLR of an exclusive-OR sum of two bits
$L_\mathrm{ch} = L_\mathrm{ch}^{(0)}$	channel LLR
$L_{ij}^{(it)}$	LLR that each parity check node h_i sends to each symbol node d_j in iteration number *it*
$\lvert LQ_{ij}^x\rvert$, $\lvert Lf_j^x\rvert$, $\lvert LR_{ij}^x\rvert$, $\lvert LQ_j^x\rvert$	L values for Q_{ij}^x, f_j^x, R_{ij}^x, Q_j^x, respectively
$\lvert Lz\rvert$	an L value, that is, the absolute value of the natural log of z
$M(j)$	set of indexes of all the children parity check nodes connected to the symbol node d_j
$M(j)\backslash i$	set of indexes of all the children parity check nodes connected to the symbol node d_j with the exclusion of the child parity check node h_i
$N(i)$	set of indexes of all the parent symbol nodes connected to the parity check node h_i
$N(i)\backslash j$	set of indexes of all the parent symbol nodes connected to the parity check node h_i with the exclusion of the parent symbol node d_j
N_t	number of entries of a look-up table for the logarithmic LDPC decoder
Q_j^x	*a posteriori* probabilities
Q_{ij}^x	estimate that each symbol node d_j sends to each of its children parity check nodes h_i in the sum–product algorithm
R_{ij}^x	estimate that each parity check node h_i sends to each of its parent symbol nodes d_j in the sum–product algorithm
s	number of ‘1’s per column of parity check matrix $\boldsymbol{H}$ (Chapter 8)
$\boldsymbol{tp}$	transmitted packet code vector
tp_n	transmitted packet in a fountain or linear random code
v	number of ‘1’s per row of parity check matrix $\boldsymbol{H}$ (Chapter 8)
z	positive real number such that $z \leq 1$
$Z_{ij}^{(it)}$	LLR that each symbol node d_j sends to each parity check node h_i in iteration number *it*

Appendix A

τ	time duration of a given pulse (Appendix A)
a_k	amplitude of the symbol k in a digital amplitude modulated signal
N_{R}	received noise power
$p(t)$	signal in a digital amplitude modulated transmission
$Q(k)$	normalized Gaussian probability density function
T	duration of the transmitted symbol in a digital amplitude-modulated signal
S_{R}	received signal power
U	threshold voltage (Appendix A)
$x(t)$, $y(t)$, $n(t)$	transmitted, received and noise signals, respectively

Appendix B

$\phi(X)$	minimum-degree polynomial
F	field
$f(X)$	polynomial defined over GF(2)
G_{r}	group

Abbreviations

ACK	positive acknowledgement
APP	*a posteriori* probability
ARQ	automatic repeat request
AWGN	additive white Gaussian noise
BCH	Bose, Chaudhuri, Hocquenghem (code)
BCJR	Bahl, Cocke, Jelinek, Raviv (algorithm)
BEC	binary erasure channel
BER	bit error rate
BM/B–M	Berlekamp–Massey (algorithm)
BPS/bps	bits per second
BSC	binary symmetric channel
ch	channel
CD	compact disk
CIRC	cross-interleaved Reed–Solomon code
conv	convolutional (code)
CRC	cyclic redundancy check
dec	decoder
deg	degree
DMC	discrete memoryless channel
DMS	discrete memoryless source
DRP	dithered relatively prime (interleaver)
enc	encoder
EFM	eight-to-fourteen modulation
EXIT	extrinsic information transfer
FCS	frame check sequence
FEC	forward error correction
FIR	finite impulse response
FSSM	finite state sequential machine
GF	Galois field
HCF/hcf	highest common factor
IIR	infinite impulse response
ISI	inter-symbol interference
lim	limit
LCM/lcm	lowest common multiple
LDPC	low-density parity check (code)

LLR	log likelihood ratio
LT	Luby transform
MAP	maximum *a posteriori* probability
ML	maximum likelihood
MLD	maximum likelihood detection
mod	modulo
MTC	multiple turbo code
NAK	negative acknowledgement
NRZ	non-return to zero
ns	non-systematic
opt	optimum
PCND	parity check node decoder
RCPC	rate-compatible punctured code(s)
RLL	run length limited
RS	Reed–Solomon (code)
RSC	recursive systematic convolutional (code/encoder)
RZ	return to zero
SND	symbol node decoder
SOVA	soft-output Viterbi algorithm
SPA	sum–product algorithm
VA	Viterbi algorithm

1

Information and Coding Theory

In his classic paper ‘A Mathematical Theory of Communication’, Claude Shannon [1] introduced the main concepts and theorems of what is known as information theory. Definitions and models for two important elements are presented in this theory. These elements are the binary source (BS) and the binary symmetric channel (BSC). A binary source is a device that generates one of the two possible symbols ‘0’ and ‘1’ at a given rate r, measured in symbols per second. These symbols are called *bits* (**b**inary dig**its**) and are generated randomly.

The BSC is a medium through which it is possible to transmit one symbol per time unit. However, this channel is not reliable, and is characterized by the error probability p $(0 \leq p \leq 1/2)$ that an output bit can be different from the corresponding input. The symmetry of this channel comes from the fact that the error probability p is the same for both of the symbols involved.

Information theory attempts to analyse communication between a transmitter and a receiver through an unreliable channel, and in this approach performs, on the one hand, an analysis of information sources, especially the amount of information produced by a given source, and, on the other hand, states the conditions for performing reliable transmission through an unreliable channel.

There are three main concepts in this theory:

1. The first one is the definition of a quantity that can be a valid measurement of information, which should be consistent with a physical understanding of its properties.
2. The second concept deals with the relationship between the information and the source that generates it. This concept will be referred to as source information. Well-known information theory techniques like compression and encryption are related to this concept.
3. The third concept deals with the relationship between the information and the unreliable channel through which it is going to be transmitted. This concept leads to the definition of a very important parameter called the channel capacity. A well-known information theory technique called error-correction coding is closely related to this concept. This type of coding forms the main subject of this book.

One of the most used techniques in information theory is a procedure called coding, which is intended to optimize transmission and to make efficient use of the capacity of a given channel.

Essentials of Error-Control Coding Jorge Castiñeira Moreira and Patrick Guy Farrell

Table 1.1 Coding: a codeword for each message

Messages	Codewords
s_1	101
s_2	01
s_3	110
s_4	000

In general terms, coding is a bijective assignment between a set of messages to be transmitted, and a set of codewords that are used for transmitting these messages. Usually this procedure adopts the form of a table in which each message of the transmission is in correspondence with the codeword that represents it (see an example in Table 1.1).

Table 1.1 shows four codewords used for representing four different messages. As seen in this simple example, the length of the codeword is not constant. One important property of a coding table is that it is constructed in such a way that every codeword is uniquely decodable. This means that in the transmission of a sequence composed of these codewords there should be only one possible way of interpreting that sequence. This is necessary when variable-length coding is used.

If the code shown in Table 1.1 is compared with a constant-length code for the same case, constituted from four codewords of two bits, 00, 01, 10, 11, it is seen that the code in Table 1.1 adds redundancy. Assuming equally likely messages, the average number of transmitted bits per symbol is equal to 2.75. However, if for instance symbol s_2 were characterized by a probability of being transmitted of 0.76, and all other symbols in this code were characterized by a probability of being transmitted equal to 0.08, then this source would transmit an average number of bits per symbol of 2.24 bits. As seen in this simple example, a level of compression is possible when the information source is not uniform, that is, when a source generates messages that are not equally likely.

The source information measure, the channel capacity measure and coding are all related by one of the Shannon theorems, the channel coding theorem, which is stated as follows:

If the information rate of a given source does not exceed the capacity of a given channel, then there exists a coding technique that makes possible transmission through this unreliable channel with an arbitrarily low error rate.

This important theorem predicts the possibility of error-free transmission through a noisy or unreliable channel. This is obtained by using coding. The above theorem is due to Claude Shannon [1, 2], and states the restrictions on the transmission of information through a noisy channel, stating also that the solution for overcoming those restrictions is the application of a rather sophisticated coding technique. What is not formally stated is how to implement this coding technique.

A block diagram of a communication system as related to information theory is shown in Figure 1.1.

The block diagram seen in Figure 1.1 shows two types of encoders. The channel encoder is designed to perform error correction with the aim of converting an unreliable channel into

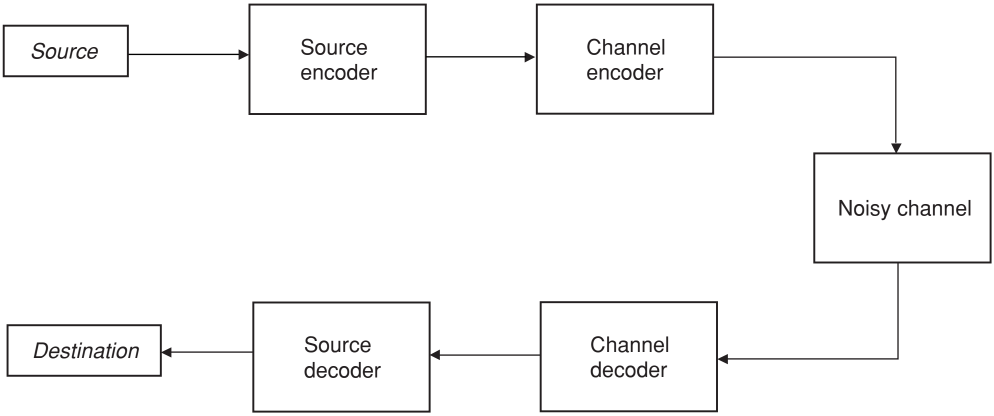


Figure 1.1 A communication system: source and channel coding

a reliable one. On the other hand, there also exists a source encoder that is designed to make the source information rate approach the channel capacity. The destination is also called the information sink.

Some concepts relating to the transmission of discrete information are introduced in the following sections.

1.1 Information

1.1.1 A Measure of Information

From the point of view of information theory, information is not knowledge, as commonly understood, but instead relates to the probabilities of the symbols used to send messages between a source and a destination over an unreliable channel. A quantitative measure of symbol information is related to its probability of occurrence, either as it emerges from a source or when it arrives at its destination. The less likely the event of a symbol occurrence, the higher is the information provided by this event. This suggests that a quantitative measure of symbol information will be inversely proportional to the probability of occurrence.

Assuming an arbitrary message x_i which is one of the possible messages from a set a given discrete source can emit, and $P(x_i) = P_i$ is the probability that this message is emitted, the output of this information source can be modelled as a random variable X that can adopt any of the possible values x_i, so that $P(X = x_i) = P_i$. Shannon defined a measure of the information for the event x_i by using a logarithmic measure operating over the base b:

$$I_i \equiv -\log_b P_i = \log_b \left(\frac{1}{P_i} \right) \tag{1}$$

The information of the event depends only on its probability of occurrence, and is not dependent on its content.

The base of the logarithmic measure can be converted by using

$$\log_a(x) = \log_b(x)\frac{1}{\log_b(a)} \tag{2}$$

If this measure is calculated to base 2, the information is said to be measured in *bits*. If the measure is calculated using natural logarithms, the information is said to be measured in *nat*s. As an example, if the event is characterized by a probability of $P_i = 1/2$, the corresponding information is $I_i = 1$ bit. From this point of view, a bit is the amount of information obtained from one of two possible, and equally likely, events. This use of the term *bit* is essentially different from what has been described as the binary digit. In this sense the bit acts as the unit of the measure of information.

Some properties of information are derived from its definition:

$$\begin{aligned} I_i &\geq 0 \quad 0 \leq P_i \leq 1 \\ I_i &\to 0 \quad \text{if } P_i \to 1 \\ I_i &> I_j \quad \text{if } P_i < P_j \end{aligned}$$

For any two independent source messages x_i and x_j with probabilities P_i and P_j respectively, and with joint probability $P(x_i, x_j) = P_i P_j$, the information of the two messages is the addition of the information in each message:

$$I_{ij} = \log_b \frac{1}{P_i P_j} = \log_b \frac{1}{P_i} + \log_b \frac{1}{P_j} = I_i + I_j$$

1.2 Entropy and Information Rate

In general, an information source generates any of a set of M different symbols, which are considered as representatives of a discrete random variable X that adopts any value in the range $A = \{x_1, x_2, \ldots, x_M\}$. Each symbol x_i has the probability P_i of being emitted and contains information I_i. The symbol probabilities must be in agreement with the fact that at least one of them will be emitted, so

$$\sum_{i=1}^{M} P_i = 1 \tag{3}$$

The source symbol probability distribution is stationary, and the symbols are independent and transmitted at a rate of r symbols per second. This description corresponds to a discrete memoryless source (DMS), as shown in Figure 1.2.

Each symbol contains the information I_i so that the set $\{I_1, I_2, \ldots, I_M\}$ can be seen as a discrete random variable with average information

$$H_b(X) = \sum_{i=1}^{M} P_i I_i = \sum_{i=1}^{M} P_i \log_b\left(\frac{1}{P_i}\right) \tag{4}$$

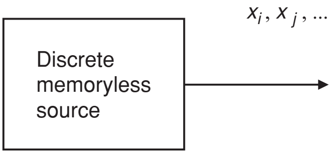


Figure 1.2 A discrete memoryless source

The function so defined is called the entropy of the source. When base 2 is used, the entropy is measured in bits per symbol:

$$H(X) = \sum_{i=1}^{M} P_i I_i = \sum_{i=1}^{M} P_i \log_2\left(\frac{1}{P_i}\right) \text{ bits per symbol} \tag{5}$$

The symbol information value when $P_i = 0$ is mathematically undefined. To solve this situation, the following condition is imposed: $I_i = \infty$ if $P_i = 0$. Therefore $P_i \log_2\left(1/P_i\right) = 0$ (L'Hopital's rule) if $P_i = 0$. On the other hand, $P_i \log\left(1/P_i\right) = 0$ if $P_i = 1$.

Example 1.1: Suppose that a DMS is defined over the range of X, $A = \{x_1, x_2, x_3, x_4\}$, and the corresponding probability values for each symbol are $P(X = x_1) = 1/2$, $P(X = x_2) = P(X = x_3) = 1/8$ and $P(X = x_4) = 1/4$.

Entropy for this DMS is evaluated as

$$H(X) = \sum_{i=1}^{M} P_i \log_2\left(\frac{1}{P_i}\right) = \frac{1}{2}\log_2(2) + \frac{1}{8}\log_2(8) + \frac{1}{8}\log_2(8) + \frac{1}{4}\log_2(4)$$
$$= 1.75 \text{ bits per symbol}$$

Example 1.2: A source characterized in the frequency domain with a bandwidth of $W = 4000$ Hz is sampled at the Nyquist rate, generating a sequence of values taken from the range $A = \{-2, -1, 0, 1, 2\}$ with the following corresponding set of probabilities $\left\{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{16}\right\}$. Calculate the source rate in bits per second.

Entropy is first evaluated as

$$H(X) = \sum_{i=1}^{M} P_i \log_2\left(\frac{1}{P_i}\right) = \frac{1}{2}\log_2(2) + \frac{1}{4}\log_2(4) + \frac{1}{8}\log_2(8)$$
$$+2 \times \frac{1}{16}\log_2(16) = \frac{15}{8} \text{ bits per sample}$$

The minimum sampling frequency is equal to 8000 samples per second, so that the information rate is equal to 15 kbps.

Entropy can be evaluated to a different base by using

$$H_b(X) = \frac{H(X)}{\log_2(b)} \tag{6}$$

Entropy $H(X)$ can be understood as the mean value of the information per symbol provided by the source being measured, or, equivalently, as the mean value experienced by an observer before knowing the source output. In another sense, entropy is a measure of the randomness of the source being analysed. The entropy function provides an adequate quantitative measure of the parameters of a given source and is in agreement with physical understanding of the information emitted by a source.

Another interpretation of the entropy function [5] is seen by assuming that if $n \gg 1$ symbols are emitted, $nH(X)$ bits is the total amount of information emitted. As the source generates r symbols per second, the whole emitted sequence takes n/r seconds. Thus, information will be transmitted at a rate of

$$\frac{nH(X)}{(n/r)} \text{ bps} \tag{7}$$

The information rate is then equal to

$$R = rH(X) \text{ bps} \tag{8}$$

The Shannon theorem states that information provided by a given DMS can be coded using binary digits and transmitted over an equivalent noise-free channel at a rate of

$$r_b \geq R \text{ symbols or binary digits per second}$$

It is again noted here that the bit is the unit of information, whereas the symbol or binary digit is one of the two possible symbols or signals ‘0’ or ‘1’, usually also called bits.

Theorem 1.1: Let X be a random variable that adopts values in the range $A = \{x_1, x_2, \ldots, x_M\}$ and represents the output of a given source. Then it is possible to show that

$$0 \leq H(X) \leq \log_2(M) \tag{9}$$

Additionally,

$$\begin{aligned} &H(X) = 0 \quad \text{if and only if } P_i = 1 \text{ for some } i \\ &H(X) = \log_2(M) \quad \text{if and only if } P_i = 1\big/M \text{ for every } i \end{aligned} \tag{10}$$

The condition $0 \leq H(X)$ can be verified by applying the following:

$$P_i \log_2(1/P_i) \to 0 \quad \text{if } P_i \to 0$$

The condition $H(X) \leq \log_2(M)$ can be verified in the following manner:

Let $Q_1, Q_2, \ldots, Q_M$ be arbitrary probability values that are used to replace terms $1/P_i$ by the terms Q_i/P_i in the expression of the entropy [equation (5)]. Then the following inequality is used:

$$\ln(x) \leq x - 1$$

where equality occurs if $x = 1$ (see Figure 1.3).

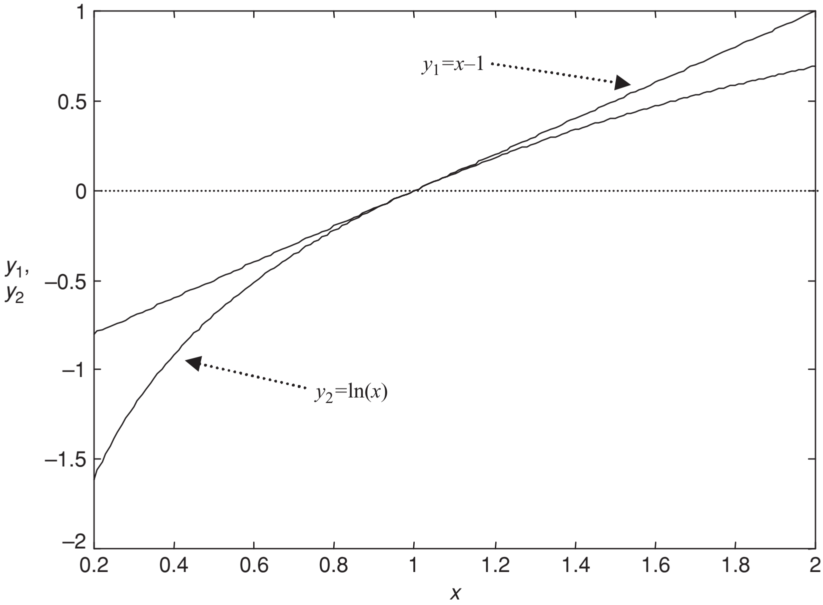


Figure 1.3 Inequality $\ln(x) \leq x - 1$

After converting entropy to its natural logarithmic form, we obtain

$$\sum_{i=1}^{M} P_i \log_2\left(\frac{Q_i}{P_i}\right) = \frac{1}{\ln(2)} \sum_{i=1}^{M} P_i \ \ln\left(\frac{Q_i}{P_i}\right)$$

and if $x = Q_i \big/ P_i$,

$$\sum_{i=1}^{M} P_i \ \ln\left(\frac{Q_i}{P_i}\right) \leq \sum_{i=1}^{M} P_i \left(\frac{Q_i}{P_i} - 1\right) = \sum_{i=1}^{M} Q_i - \sum_{i=1}^{M} P_i \tag{11}$$

As the coefficients Q_i are probability values, they fit the normalizing condition $\sum_{i=1}^{M} Q_i \leq 1$, and it is also true that $\sum_{i=1}^{M} P_i = 1$.
Then

$$\sum_{i=1}^{M} P_i \log_2\left(\frac{Q_i}{P_i}\right) \leq 0 \tag{12}$$

If now the probabilities Q_i adopt equally likely values $Q_i = 1\big/M$,

$$\sum_{i=1}^{M} P_i \log_2\left(\frac{1}{P_i M}\right) = \sum_{i=1}^{M} P_i \log_2\left(\frac{1}{P_i}\right) - \sum_{i=1}^{M} P_i \log_2(M) = H(X) - \log_2(M) \leq 0$$

$$H(X) \leq \log_2(M) \tag{13}$$

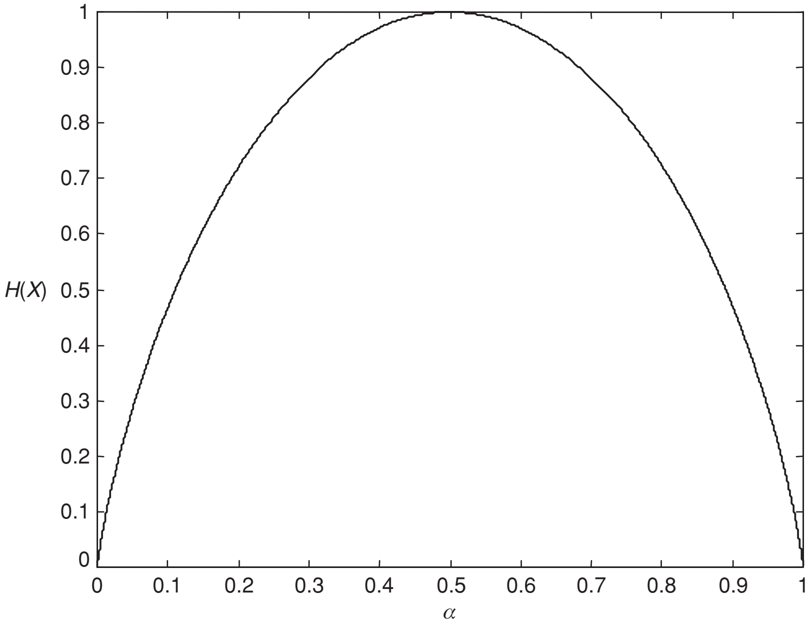


Figure 1.4 Entropy function for the binary source

In the above inequality, equality occurs when $\log_2\left(1/P_i\right) = \log_2(M)$, which means that $P_i = 1/M$.

The maximum value of the entropy is then $\log_2(M)$, and occurs when all the symbols transmitted by a given source are equally likely. Uniform distribution corresponds to maximum entropy.

In the case of a binary source ($M = 2$) and assuming that the probabilities of the symbols are the values

$$P_0 = \alpha \qquad P_1 = 1 - \alpha \tag{14}$$

the entropy is equal to

$$H(X) = \Omega(\alpha) = \alpha \log_2\left(\frac{1}{\alpha}\right) + (1-\alpha)\log_2\left(\frac{1}{1-\alpha}\right) \tag{15}$$

This expression is depicted in Figure 1.4.

The maximum value of this function is given when $\alpha = 1 - \alpha$, that is, $\alpha = 1/2$, so that the entropy is equal to $H(X) = \log_2 2 = 1$ bps. (This is the same as saying one bit per binary digit or binary symbol.)

When $\alpha \to 1$, entropy tends to zero. The function $\Omega(\alpha)$ will be used to represent the entropy of the binary source, evaluated using logarithms to base 2.

Example 1.3: A given source emits $r = 3000$ symbols per second from a range of four symbols, with the probabilities given in Table 1.2.

Table 1.2 Example 1.3

x_i	P_i	I_i
A	1/3	1.5849
B	1/3	1.5849
C	1/6	2.5849
D	1/6	2.5849

The entropy is evaluated as

$$H(X) = 2 \times \frac{1}{3} \times \log_2(3) + 2 \times \frac{1}{6} \times \log_2(6) = 1.9183 \text{ bits per symbol}$$

And this value is close to the maximum possible value, which is $\log_2(4) = 2$ bits per symbol. The information rate is equal to

$$R = rH(X) = (3000)1.9183 = 5754.9 \text{ bps}$$

1.3 Extended DMSs

In certain circumstances it is useful to consider information as grouped into blocks of symbols. This is generally done in binary format. For a memoryless source that takes values in the range $\{x_1, x_2, \ldots, x_M\}$, and where P_i is the probability that the symbol x_i is emitted, the order n extension of the range of a source has M^n symbols $\{y_1, y_2, \ldots, y_{Mn}\}$. The symbol y_i is constituted from a sequence of n symbols x_{ij}. The probability $P(Y = y_i)$ is the probability of the corresponding sequence $x_{i1}, x_{i2}, \ldots, x_{in}$:

$$P(Y = y_i) = P_{i1}, P_{i2}, \ldots, P_{in} \tag{16}$$

where y_i is the symbol of the extended source that corresponds to the sequence $x_{i1}, x_{i2}, \ldots, x_{in}$. Then

$$H(X^n) = \sum_{y=x^n} P(y_i) \log_2 \frac{1}{P(y_i)} \tag{17}$$

Example 1.4: Construct the order 2 extension of the source of Example 1.1, and calculate its entropy.

Symbols of the original source are characterized by the probabilities $P(X = x_1) = 1/2$, $P(X = x_2) = P(X = x_3) = 1/8$ and $P(X = x_4) = 1/4$.

Symbol probabilities for the desired order 2 extended source are given in Table 1.3.

The entropy of this extended source is equal to

$$\begin{aligned} H(X^2) &= \sum_{i=1}^{M^2} P_i \log_2 \left(\frac{1}{P_i}\right) \\ &= 0.25 \log_2(4) + 2 \times 0.125 \log_2(8) + 5 \times 0.0625 \log_2(16) \\ &\quad + 4 \times 0.03125 \log_2(32) + 4 \times 0.015625 \log_2(64) = 3.5 \text{ bits per symbol} \end{aligned}$$

Table 1.3 Symbols of the order 2 extended source and their probabilities for Example 1.4

Symbol	Probability	Symbol	Probability	Symbol	Probability	Symbol	Probability
x_1x_1	0.25	x_2x_1	0.0625	x_3x_1	0.0625	x_4x_1	0.125
x_1x_2	0.0625	x_2x_2	0.015625	x_3x_2	0.015625	x_4x_2	0.03125
x_1x_3	0.0625	x_2x_3	0.015625	x_3x_3	0.015625	x_4x_3	0.03125
x_1x_4	0.125	x_2x_4	0.03125	x_3x_4	0.03125	x_4x_4	0.0625

As seen in this example, the order 2 extended source has an entropy which is twice that of the entropy of the original, non-extended source. It can be shown that the order n extension of a DMS fits the condition $H(X^n) = nH(X)$.

1.4 Channels and Mutual Information

1.4.1 Information Transmission over Discrete Channels

A quantitative measure of source information has been introduced in the above sections. Now the transmission of that information through a given channel will be considered. This will provide a quantitative measure of the information received after its transmission through that channel. Here attention is on the transmission of the information, rather than on its generation.

A channel is always a medium through which the information being transmitted can suffer from the effect of noise, which produces errors, that is, changes of the values initially transmitted. In this sense there will be a probability that a given transmitted symbol is converted into another symbol. From this point of view the channel is considered as unreliable. The Shannon channel coding theorem gives the conditions for achieving reliable transmission through an unreliable channel, as stated previously.

1.4.2 Information Channels

Definition 1.1: An information channel is characterized by an input range of symbols $\{x_1, x_2, \ldots, x_U\}$, an output range $\{y_1, y_2, \ldots, y_V\}$ and a set of conditional probabilities $P(y_j/x_i)$ that determines the relationship between the input x_i and the output y_j. This conditional probability corresponds to that of receiving symbol y_j if symbol x_i was previously transmitted, as shown in Figure 1.5.

The set of probabilities $P(y_j/x_i)$ is arranged into a matrix $\boldsymbol{P}_{\text{ch}}$ that characterizes completely the corresponding discrete channel:

$$P_{ij} = P(y_j/x_i)$$

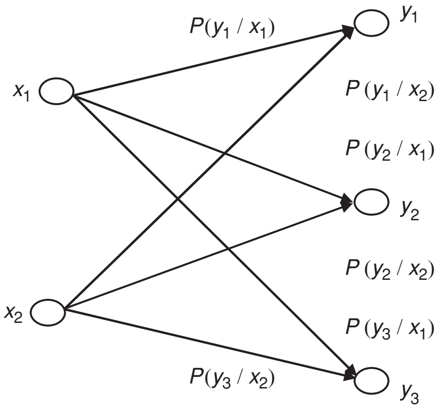


Figure 1.5 A discrete transmission channel

$$\boldsymbol{P}_{\text{ch}} = \begin{bmatrix} P(y_1/x_1) & P(y_2/x_1) & \cdots & P(y_V/x_1) \\ P(y_1/x_2) & P(y_2/x_2) & \cdots & P(y_V/x_2) \\ \vdots & \vdots & & \vdots \\ P(y_1/x_U) & P(y_2/x_U) & \cdots & P(y_V/x_U) \end{bmatrix} \tag{18}$$

$$\boldsymbol{P}_{\text{ch}} = \begin{bmatrix} P_{11} & P_{12} & \cdots & P_{1V} \\ P_{21} & P_{22} & \cdots & P_{2V} \\ \vdots & \vdots & & \vdots \\ P_{U1} & P_{U2} & \cdots & P_{UV} \end{bmatrix} \tag{19}$$

Each row in this matrix corresponds to an input, and each column corresponds to an output. Addition of all the values of a row is equal to one. This is because after transmitting a symbol x_i, there must be a received symbol y_j at the channel output.

Therefore,

$$\sum_{j=1}^{V} P_{ij} = 1, \quad i = 1, 2, \ldots, U \tag{20}$$

Example 1.5: The binary symmetric channel (BSC).

The BSC is characterized by a probability p that one of the binary symbols converts into the other one (see Figure 1.6). Each binary symbol has, on the other hand, a probability of being transmitted. The probabilities of a 0 or a 1 being transmitted are α and $1 - \alpha$ respectively.

According to the notation used,

$$x_1 = 0, \quad x_2 = 1 \quad \text{and} \quad y_1 = 0, \quad y_2 = 1$$

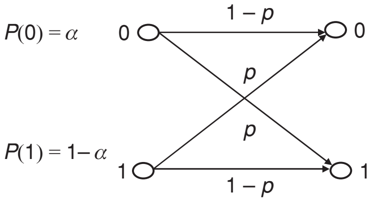


Figure 1.6 Binary symmetric channel

The probability matrix for the BSC is equal to

$$\boldsymbol{P}_{\mathrm{ch}} = \begin{bmatrix} 1-p & p \\ p & 1-p \end{bmatrix} \tag{21}$$

Example 1.6: The binary erasure channel (BEC).

In its most basic form, the transmission of binary information involves sending two different waveforms to identify the symbols ‘0’ and ‘1’. At the receiver, normally an optimum detection operation is used to decide whether the waveform received, affected by filtering and noise in the channel, corresponds to a ‘0’ or a ‘1’. This operation, often called matched filter detection, can sometimes give an indecisive result. If confidence in the received symbol is not high, it may be preferable to indicate a doubtful result by means of an erasure symbol. Correction of the erasure symbols is then normally carried out by other means in another part of the system.

In other scenarios the transmitted information is coded, which makes it possible to detect if there are errors in a bit or packet of information. In these cases it is also possible to apply the concept of data erasures. This is used, for example, in the concatenated coding system of the compact disc, where on receipt of the information the first decoder detects errors and marks or erases a group of symbols, thus enabling the correction of these symbols in the second decoder. Another example of the erasure channel arises during the transmission of packets over the Internet. If errors are detected in a received packet, then they can be erased, and the erasures corrected by means of retransmission protocols (normally involving the use of a parallel feedback channel).

The use of erasures modifies the BSC model, giving rise to the BEC, as shown in Figure 1.7. For this channel, $0 \leq p \leq 1/2$, where p is the erasure probability, and the channel model has two inputs and three outputs. When the received values are unreliable, or if blocks are detected

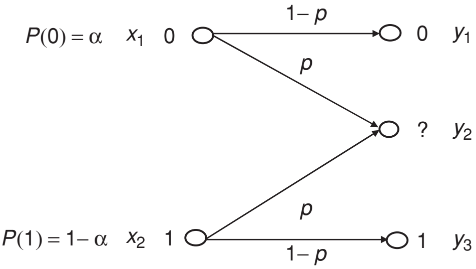


Figure 1.7 Binary erasure channel

to contain errors, then erasures are declared, indicated by the symbol '?'. The probability matrix of the BEC is the following:

$$\boldsymbol{P}_{\text{ch}} = \begin{bmatrix} 1-p & p & 0 \\ 0 & p & 1-p \end{bmatrix} \tag{22}$$

1.5 Channel Probability Relationships

As stated above, the probability matrix $\boldsymbol{P}_{\text{ch}}$ characterizes a channel. This matrix is of order $U \times V$ for a channel with U input symbols and V output symbols. Input symbols are characterized by the set of probabilities $\{P(x_1), P(x_2), \ldots, P(x_U)\}$, whereas output symbols are characterized by the set of probabilities $\{P(y_1), P(y_2), \ldots, P(y_V)\}$.

$$\boldsymbol{P}_{\text{ch}} = \begin{bmatrix} P_{11} & P_{12} & \cdots & P_{1V} \\ P_{21} & P_{22} & \cdots & P_{2V} \\ \vdots & \vdots & & \vdots \\ P_{U1} & P_{U2} & & P_{UV} \end{bmatrix}$$

The relationships between input and output probabilities are the following: The symbol y_1 can be received in U different ways. In fact this symbol can be received with probability P_{11} if symbol x_1 was actually transmitted, with probability P_{21} if symbol x_2 was actually transmitted, and so on.

Any of the U input symbols can be converted by the channel into the output symbol y_1. The probability of the reception of symbol y_1, $P(y_1)$, is calculated as $P(y_1) = P_{11}P(x_1) + P_{21}P(x_2) + \cdots + P_{U1}P(x_U)$. Calculation of the probabilities of the output symbols leads to the following system of equations:

$$\begin{aligned} P_{11}P(x_1) + P_{21}P(x_2) + \cdots + P_{U1}P(x_U) &= P(y_1) \\ P_{12}P(x_1) + P_{22}P(x_2) + \cdots + P_{U2}P(x_U) &= P(y_2) \\ &\vdots \\ P_{1V}P(x_1) + P_{2V}P(x_2) + \cdots + P_{UV}P(x_U) &= P(y_V) \end{aligned} \tag{23}$$

Output symbol probabilities are calculated as a function of the input symbol probabilities $P(x_i)$ and the conditional probabilities $P(y_j/x_i)$. It is however to be noted that knowledge of the output probabilities $P(y_j)$ and the conditional probabilities $P(y_j/x_i)$ provides solutions for values of $P(x_i)$ that are not unique. This is because there are many input probability distributions that give the same output distribution.

Application of the Bayes rule to the conditional probabilities $P(y_j/x_i)$ allows us to determine the conditional probability of a given input x_i after receiving a given output y_j:

$$P(x_i/y_j) = \frac{P(y_j/x_i)P(x_i)}{P(y_j)} \tag{24}$$

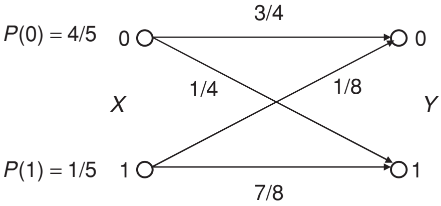


Figure 1.8 Example 1.7

By combining this expression with expression (23), equation (24) can be written as

$$P(x_i/y_j) = \frac{P(y_j/x_i)P(x_i)}{\sum_{i=1}^{U} P(y_j/x_i)P(x_i)} \tag{25}$$

Conditional probabilities $P(y_j/x_i)$ are usually called forward probabilities, and conditional probabilities $P(x_i/y_j)$ are known as backward probabilities. The numerator in the above expression describes the probability of the joint event:

$$P(x_i, y_j) = P(y_j/x_i)P(x_i) = P(x_i/y_j)P(y_j) \tag{26}$$

Example 1.7: Consider the binary channel for which the input range and output range are in both cases equal to {0, 1}. The corresponding transition probability matrix is in this case equal to

$$\boldsymbol{P}_{\text{ch}} = \begin{bmatrix} 3/4 & 1/4 \\ 1/8 & 7/8 \end{bmatrix}$$

Figure 1.8 represents this binary channel.

Source probabilities provide the statistical information about the input symbols. In this case it happens that $P(X = 0) = 4/5$ and $P(X = 1) = 1/5$. According to the transition probability matrix for this case,

$$\begin{array}{ll} P(Y = 0/X = 0) = 3/4 & P(Y = 1/X = 0) = 1/4 \\ P(Y = 0/X = 1) = 1/8 & P(Y = 1/X = 1) = 7/8 \end{array}$$

These values can be used to calculate the output symbol probabilities:

$$\begin{aligned} P(Y = 0) &= P(Y = 0/X = 0)P(X = 0) + P(Y = 0/X = 1)P(X = 1) \\ &= \frac{3}{4} \times \frac{4}{5} + \frac{1}{8} \times \frac{1}{5} = \frac{25}{40} \end{aligned}$$

$$\begin{aligned} P(Y = 1) &= P(Y = 1/X = 0)P(X = 0) + P(Y = 1/X = 1)P(X = 1) \\ &= \frac{1}{4} \times \frac{4}{5} + \frac{7}{8} \times \frac{1}{5} = \frac{15}{40} \end{aligned}$$

which confirms that $P(Y = 0) + P(Y = 1) = 1$ is true.

These values can be used to evaluate the backward conditional probabilities:

$$P(X=0/Y=0) = \frac{P(Y=0/X=0)P(X=0)}{P(Y=0)} = \frac{(3/4)(4/5)}{(25/40)} = \frac{24}{25}$$

$$P(X=0/Y=1) = \frac{P(Y=1/X=0)P(X=0)}{P(Y=1)} = \frac{(1/4)(4/5)}{(15/40)} = \frac{8}{15}$$

$$P(X=1/Y=1) = \frac{P(Y=1/X=1)P(X=1)}{P(Y=1)} = \frac{(7/8)(1/5)}{(15/40)} = \frac{7}{15}$$

$$P(X=1/Y=0) = \frac{P(Y=0/X=1)P(X=1)}{P(Y=0)} = \frac{(1/8)(1/5)}{(25/40)} = \frac{1}{25}$$

1.6 The *A Priori* and *A Posteriori* Entropies

The probability of occurrence of a given output symbol y_j is $P(y_j)$, calculated using expression (23). However, if the actual transmitted symbol x_i is known, then the related conditional probability of the output symbol becomes $P(y_j/x_i)$. In the same way, the probability of a given input symbol, initially $P(x_i)$, can also be refined if the actual output is known. Thus, if the received symbol y_j appears at the output of the channel, then the related input symbol conditional probability becomes $P(x_i/y_j)$.

The probability $P(x_i)$ is known as the *a priori* probability; that is, it is the probability that characterizes the input symbol before the presence of any output symbol is known. Normally, this probability is equal to the probability that the input symbol has of being emitted by the source (the source symbol probability). The probability $P(x_i/y_j)$ is an estimate of the symbol x_i after knowing that a given symbol y_j appeared at the channel output, and is called the *a posteriori* probability.

As has been defined, the source entropy is an average calculated over the information of a set of symbols for a given source:

$$H(X) = \sum_i P(x_i)\log_2\left[\frac{1}{P(x_i)}\right]$$

This definition corresponds to the *a priori* entropy. The *a posteriori* entropy is given by the following expression:

$$H(X/y_j) = \sum_i P(x_i/y_j)\log_2\left[\frac{1}{P(x_i/y_j)}\right] \quad i = 1, 2, \ldots, U \tag{27}$$

Example 1.8: Determine the *a priori* and *a posteriori* entropies for the channel of Example 1.7.

The *a priori* entropy is equal to

$$H(X) = \frac{4}{5}\log_2\left(\frac{5}{4}\right) + \frac{1}{5}\log_2(5) = 0.7219 \text{ bits}$$

Assuming that a '0' is present at the channel output,

$$H(X/0) = \frac{24}{25}\log_2\left(\frac{25}{24}\right) + \frac{1}{25}\log_2(25) = 0.2423 \text{ bits}$$

and in the case of a '1' present at the channel output,

$$H(X/1) = \frac{8}{15}\log_2\left(\frac{15}{8}\right) + \frac{7}{15}\log_2\left(\frac{15}{7}\right) = 0.9968 \text{ bits}$$

Thus, entropy decreases after receiving a '0' and increases after receiving a '1'.

1.7 Mutual Information

According to the description of a channel depicted in Figure 1.5, $P(x_i)$ is the probability that a given input symbol is emitted by the source, $P(y_j)$ determines the probability that a given output symbol y_j is present at the channel output, $P(x_i, y_j)$ is the joint probability of having symbol x_i at the input and symbol y_j at the output, $P(y_j/x_i)$ is the probability that the channel converts the input symbol x_i into the output symbol y_j and $P(x_i/y_j)$ is the probability that x_i has been transmitted if y_j is received.

1.7.1 Mutual Information: Definition

Mutual information measures the information transferred when x_i is sent and y_j is received, and is defined as

$$I(x_i, y_j) = \log_2 \frac{P(x_i/y_j)}{P(x_i)} \text{ bits} \tag{28}$$

In a noise-free channel, each y_j is uniquely connected to the corresponding x_i, and so they constitute an input–output pair (x_i, y_j) for which $P(x_i/y_j) = 1$ and $I(x_i, y_j) = \log_2 \frac{1}{P(x_i)}$ bits; that is, the transferred information is equal to the self-information that corresponds to the input x_i.

In a very noisy channel, the output y_j and the input x_i would be completely uncorrelated, and so $P(x_i/y_j) = P(x_i)$ and also $I(x_i, y_j) = 0$; that is, there is no transference of information. In general, a given channel will operate between these two extremes.

The mutual information is defined between the input and the output of a given channel. An average of the calculation of the mutual information for all input–output pairs of a given channel is the average mutual information:

$$I(X, Y) = \sum_{i,j} P(x_i, y_j) I(x_i, y_j) = \sum_{i,j} P(x_i, y_j) \log_2 \left[\frac{P(x_i/y_j)}{P(x_i)}\right] \text{ bits per symbol} \tag{29}$$

This calculation is done over the input and output alphabets. The average mutual information measures the average amount of source information obtained from each output symbol.

The following expressions are useful for modifying the mutual information expression:

$$P(x_i, y_j) = P(x_i/y_j)P(y_j) = P(y_j/x_i)P(x_i)$$
$$P(y_j) = \sum_i P(y_j/x_i)P(x_i)$$
$$P(x_i) = \sum_j P(x_i/y_j)P(y_j)$$

Then

$$\begin{aligned} I(X, Y) &= \sum_{i,j} P(x_i, y_j) I(x_i, y_j) \\ &= \sum_{i,j} P(x_i, y_j) \log_2 \left[\frac{1}{P(x_i)}\right] - \sum_{i,j} P(x_i, y_j) \log_2 \left[\frac{1}{P(x_i/y_j)}\right] \end{aligned} \tag{30}$$

$$\begin{aligned} &\sum_{i,j} P(x_i, y_j) \log_2 \left[\frac{1}{P(x_i)}\right] = \sum_i \left[\sum_j P(x_i/y_j)P(y_j)\right] \log_2 \frac{1}{P(x_i)} \\ &\sum_i P(x_i) \log_2 \frac{1}{P(x_i)} = H(X) \\ &\quad I(X, Y) = H(X) - H(X/Y) \end{aligned} \tag{31}$$

where $H(X\big/Y) = \sum_{i,j} P(x_i, y_j) \log_2 \frac{1}{P(x_i/y_j)}$ is usually called the *equivocation*.

In a sense, the equivocation can be seen as the information lost in the noisy channel, and is a function of the backward conditional probability. The observation of an output symbol y_j provides $H(X) - H(X/Y)$ bits of information. This difference is the mutual information of the channel.

1.7.2 Mutual Information: Properties

Since

$$P(x_i/y_j)P(y_j) = P(y_j/x_i)P(x_i)$$

the mutual information fits the condition

$$I(X, Y) = I(Y, X)$$

And by interchanging input and output it is also true that

$$I(X, Y) = H(Y) - H(Y/X) \tag{32}$$

where

$$H(Y) = \sum_j P(y_j) \log_2 \frac{1}{P(y_j)}$$

which is the destination entropy or output channel entropy:

$$H(Y/X) = \sum_{i,j} P(x_i, y_j) \log_2 \frac{1}{P(y_j/x_i)} \tag{33}$$

This last entropy is usually called the noise entropy.

Thus, the information transferred through the channel is the difference between the output entropy and the noise entropy. Alternatively, it can be said that the channel mutual information is the difference between the number of bits needed for determining a given input symbol before knowing the corresponding output symbol, and the number of bits needed for determining a given input symbol after knowing the corresponding output symbol, $I(X, Y) = H(X) - H(X/Y)$.

As the channel mutual information expression is a difference between two quantities, it seems that this parameter can adopt negative values. However, and in spite of the fact that for some y_j, $H(X/y_j)$ can be larger than $H(X)$, this is not possible for the average value calculated over all the outputs:

$$\sum_{i,j} P(x_i, y_j) \log_2 \frac{P(x_i/y_j)}{P(x_i)} = \sum_{i,j} P(x_i, y_j) \log_2 \frac{P(x_i, y_j)}{P(x_i)P(y_j)}$$

then

$$-I(X, Y) = \sum_{i,j} P(x_i, y_j) \log_2 \frac{P(x_i)P(y_j)}{P(x_i, y_j)} \leq 0$$

because this expression is of the form

$$\sum_{i=1}^{M} P_i \log_2 \left(\frac{Q_i}{P_i} \right) \leq 0 \tag{34}$$

which is the expression (12) used for demonstrating Theorem 1.1. The above expression can be applied due to the factor $P(x_i)P(y_j)$, which is the product of two probabilities, so that it behaves as the quantity Q_i, which in this expression is a dummy variable that fits the condition $\sum_i Q_i \leq 1$.

It can be concluded that the average mutual information is a non-negative number. It can also be equal to zero, when the input and the output are independent of each other.

A related entropy called the joint entropy is defined as

$$\begin{aligned} H(X, Y) &= \sum_{i,j} P(x_i, y_j) \log_2 \frac{1}{P(x_i, y_j)} \\ &= \sum_{i,j} P(x_i, y_j) \log_2 \frac{P(x_i)P(y_j)}{P(x_i, y_j)} + \sum_{i,j} P(x_i, y_j) \log_2 \frac{1}{P(x_i)P(y_j)} \end{aligned} \tag{35}$$

Then the set of all the entropies defined so far can be represented in Figure 1.9. The circles define regions for entropies $H(X)$ and $H(Y)$, the intersection between these two entropies is the mutual information $I(X, Y)$, while the differences between the input and output entropies are $H(X/Y)$ and $H(Y/X)$ respectively (Figure 1.9). The union of these entropies constitutes the joint entropy $H(X, Y)$.

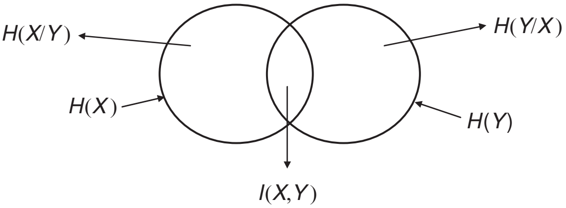


Figure 1.9 Relationships among the different entropies

Example 1.9: Entropies of the binary symmetric channel (BSC).

The BSC is constructed with two inputs (x_1, x_2) and two outputs (y_1, y_2), with alphabets over the range $A = \{0, 1\}$. The symbol probabilities are $P(x_1) = \alpha$ and $P(x_2) = 1 - \alpha$, and the transition probabilities are $P(y_1/x_2) = P(y_2/x_1) = p$ and $P(y_1/x_1) = P(y_2/x_2) = 1 - p$ (see Figure 1.10). This means that the error probability p is equal for the two possible symbols. The average error probability is equal to

$$P = P(x_1)P(y_2/x_1) + P(x_2)P(y_1/x_2) = \alpha p + (1 - \alpha)p = p$$

The mutual information can be calculated as

$$I(X, Y) = H(Y) - H(Y/X)$$

The output Y has two symbols y_1 and y_2, such that $P(y_2) = 1 - P(y_1)$. Since

$$\begin{aligned} P(y_1) &= P(y_1/x_1)P(x_1) + P(y_1/x_2)P(x_2) \\ &= (1 - p)\alpha + p(1 - \alpha) \\ &= \alpha - p\alpha + p - p\alpha = \alpha + p - 2\alpha p \end{aligned} \tag{36}$$

the destination or sink entropy is equal to

$$\begin{aligned} H(Y) &= P(y_1)\log_2 \frac{1}{P(y_1)} + [1 - P(y_1)]\log_2 \frac{1}{[1 - P(y_1)]} = \Omega\,[P(y_1)] \\ &= \Omega\,(\alpha + p - 2\alpha p) \end{aligned} \tag{37}$$

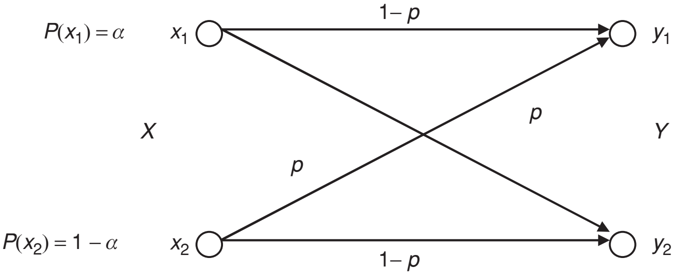


Figure 1.10 BSC of Example 1.9

The noise entropy $H(Y/X)$ can be calculated as

$$\begin{aligned}
H(Y/X) &= \sum_{i,j} P(x_i, y_j) \log_2 \frac{1}{P(y_j/x_i)} \\
&= \sum_{i,j} P(y_j/x_i) P(x_i) \log_2 \frac{1}{P(y_j/x_i)} \\
&= \sum_{i} P(x_i) \left[\sum_{j} P(y_j/x_i) \log_2 \frac{1}{P(y_j/x_i)} \right] \\
&= P(x_1) \left[P(y_2/x_1) \log_2 \frac{1}{P(y_2/x_1)} + P(y_1/x_1) \log_2 \frac{1}{P(y_1/x_1)} \right] \\
&\quad + P(x_2) \left[P(y_2/x_2) \log_2 \frac{1}{P(y_2/x_2)} + P(y_1/x_2) \log_2 \frac{1}{P(y_1/x_2)} \right] \\
&= \alpha \left[p \log_2 \frac{1}{p} + (1-p) \log_2 \frac{1}{(1-p)} \right] + (1-\alpha) \left[(1-p) \log_2 \frac{1}{(1-p)} + p \log_2 \frac{1}{p} \right] \\
&= p \log_2 \frac{1}{p} + (1-p) \log_2 \frac{1}{(1-p)} = \Omega(p) \qquad (38)
\end{aligned}$$

Note that the noise entropy of the BSC is determined only by the forward conditional probabilities of the channel, being independent of the source probabilities. This facilitates the calculation of the channel capacity for this channel, as explained in the following section.

Finally,

$$I(X, Y) = H(Y) - H(Y/X) = \Omega(\alpha + p - 2\alpha p) - \Omega(p) \qquad (39)$$

The average mutual information of the BSC depends on the source probability α and on the channel error probability p.

When the channel error probability p is very small, then

$$I(X, Y) \approx \Omega(\alpha) - \Omega(0) \approx \Omega(\alpha) = H(X)$$

This means that the average mutual information, which represents the amount of information transferred through the channel, is equal to the source entropy. On the other hand, when the channel error probability approaches its maximum value $p \approx 1/2$, then

$$I(X, Y) = \Omega(\alpha + 1/2 - \alpha) - \Omega(1/2) = 0$$

and the average mutual information tends to zero, showing that there is no transference of information between the input and the output.

Example 1.10: Entropies of the binary erasure channel (BEC).

The BEC is defined with an alphabet of two inputs and three outputs, with symbol probabilities $P(x_1) = \alpha$ and $P(x_2) = 1 - \alpha$, and transition probabilities $P(y_1/x_1) = 1 - p$ and $P(y_2/x_1) = p$, $P(y_3/x_1) = 0$ and $P(y_1/x_2) = 0$, and $P(y_2/x_2) = p$ and $P(y_3/x_2) = 1 - p$.

Now to calculate the mutual information as $I(X, Y) = H(Y) - H(Y/X)$, the following values are determined:

$$P(y_1) = P(y_1/x_1)P(x_1) + P(y_1/x_2)P(x_2) = \alpha(1 - p)$$
$$P(y_2) = P(y_2/x_1)P(x_1) + P(y_2/x_2)P(x_2) = p$$
$$P(y_3) = P(y_3/x_1)P(x_1) + P(y_3/x_2)P(x_2) = (1 - \alpha)(1 - p)$$

In this way the output or sink entropy is equal to

$$\begin{aligned} H(Y) &= P(y_1)\log_2 \frac{1}{P(y_1)} + P(y_2)\log_2 \frac{1}{P(y_2)} + P(y_3)\log_2 \frac{1}{P(y_3)} \\ &= \alpha(1 - p)\log_2 \frac{1}{\alpha(1 - p)} + p\log_2 \frac{1}{p} + (1 - \alpha)(1 - p)\log_2 \frac{1}{(1 - \alpha)(1 - p)} \\ &= (1 - p)\Omega(\alpha) + \Omega(p) \end{aligned}$$

The noise entropy $H(Y/X)$ remains to be calculated:

$$H(Y/X) = \sum_{i,j} P(y_j/x_i)P(x_i)\log_2 \frac{1}{P(y_j/x_i)} = p\log_2 \frac{1}{p} + (1 - p)\log_2 \frac{1}{(1 - p)} = \Omega(p)$$

after which the mutual information is finally given by

$$I(X, Y) = H(Y) - H(Y/X) = (1 - p)\Omega(\alpha)$$

1.8 Capacity of a Discrete Channel

The definition of the average mutual information allows us to introduce the concept of channel capacity. This parameter characterizes the channel and is basically defined as the maximum possible value that the average mutual information can adopt for a given channel:

$$C_s = \max_{P(x_i)} I(X, Y) \text{ bits per symbol} \tag{40}$$

It is noted that the definition of the channel capacity involves not only the channel itself but also the source and its statistical properties. However the channel capacity depends only on the conditional probabilities of the channel, and not on the probabilities of the source symbols, since the capacity is a value of the average mutual information given for particular values of the source symbols.

Channel capacity represents the maximum amount of information per symbol transferred through that channel.

In the case of the BSC, maximization of the average mutual information is obtained by maximizing the expression

$$\begin{aligned} C_s &= \max_{P(x_i)} I(X, Y) = \max_{P(x_i)} \{H(Y) - H(Y/X)\} \\ &= \max_{P(x_i)} \{\Omega(\alpha + p - 2\alpha p) - \Omega(p)\} = 1 - \Omega(p) = 1 - H(p) \end{aligned} \tag{41}$$

which is obtained when $\alpha = 1 - \alpha = 1/2$.

If the maximum rate of symbols per second, s, allowed in the channel is known, then the capacity of the channel per time unit is equal to

$$C = sC_{\mathrm{s}} \text{ bps} \tag{42}$$

which, as will be seen, represents the maximum rate of information transference in the channel.

1.9 The Shannon Theorems

1.9.1 Source Coding Theorem

The source coding theorem and the channel coding (channel capacity) theorem are the two main theorems stated by Shannon [1, 2]. The source coding theorem determines a bound on the level of compression of a given information source. The definitions for the different classes of entropies presented in previous sections, and particularly the definition of the source entropy, are applied to the analysis of this theorem.

Information entropy has an intuitive interpretation [1, 6]. If the DMS emits a large number of symbols n_{f} taken from an alphabet $A = \{x_1, x_2, \ldots, x_M\}$ in the form of a sequence of n_{f} symbols, symbol x_1 will appear $n_{\mathrm{f}}P(x_1)$ times, symbol $x_2, n_{\mathrm{f}}P(x_2)$ times, and symbol $x_M, n_{\mathrm{f}}P(x_M)$ times. These sequences are known as typical sequences and are characterized by the probability

$$P \approx \prod_{i=1}^{M} [P(x_i)]^{n_{\mathrm{f}}P(x_i)} \tag{43}$$

since

$$P(x_i) = 2^{\log_2[P(x_i)]}$$

$$\begin{aligned} P \approx \prod_{i=1}^{M} [P(x_i)]^{n_{\mathrm{f}}P(x_i)} &= \prod_{i=1}^{M} 2^{\log_2[P(x_i)]n_{\mathrm{f}}P(x_i)} = \prod_{i=1}^{M} 2^{n_{\mathrm{f}}\log_2[P(x_i)]P(x_i)} \\ &= 2^{n_{\mathrm{f}}\sum_{i=1}^{M} p(x_i)\log_2[P(x_i)]} \\ &= 2^{-n_{\mathrm{f}}H(X)} \end{aligned} \tag{44}$$

Typical sequences are those with the maximum probability of being emitted by the information source. Non-typical sequences are those with very low probability of occurrence. This means that of the total of $M^{n_{\mathrm{f}}}$ possible sequences that can be emitted from the information source with alphabet $A = \{x_1, x_2, \ldots, x_M\}$, only $2^{n_{\mathrm{f}}H(X)}$ sequences have a significant probability of occurring. An error of magnitude ε is made by assuming that only $2^{n_{\mathrm{f}}H(X)}$ sequences are transmitted instead of the total possible number of them. This error can be arbitrarily small if $n_{\mathrm{f}} \to \infty$. This is the essence of the data compression theorem.

This means that the source information can be transmitted using a significantly lower number of sequences than the total possible number of them.

If only $2^{n_{\mathrm{f}}H(X)}$ sequences are to be transmitted, and using a binary format of representing information, there will be $n_{\mathrm{f}}H(X)$ bits needed for representing this information. Since sequences are constituted of symbols, there will be $H(X)$ bits per symbol needed for a suitable representation of this information. This means that the source entropy is the amount of information per symbol of the source.

For a DMS with independent symbols, it can be said that compression of the information provided by this source is possible only if the probability density function of this source is not uniform, that is, if the symbols of this source are not equally likely. As seen in previous sections, a source with M equally likely symbols fits the following conditions:

$$H(X) = \log_2 M, \qquad 2^{n_{\mathrm{f}}(X)} = 2^{n_{\mathrm{f}} \log_2 M} = M^{n_{\mathrm{f}}} \tag{45}$$

The number of typical sequences for a DMS with equally likely symbols is equal to the maximum possible number of sequences that this source can emit.

This has been a short introduction to the concept of data and information compression. However, the aim of this chapter is to introduce the main concepts of a technique called error-control coding, closely related to the Shannon channel coding (capacity) theorem.

1.9.2 Channel Capacity and Coding

Communication between a source and a destination happens by the sending of information from the former to the latter, through a medium called the communication channel. Communication channels are properly modelled by using the conditional probability matrix defined between the input and the output, which allows us to determine the reliability of the information arriving at the receiver. The important result provided by the Shannon capacity theorem is that it is possible to have an error-free (reliable) transmission through a noisy (unreliable) channel, by means of the use of a rather sophisticated coding technique, as long as the transmission rate is kept to a value less than or equal to the channel capacity. The bound imposed by this theorem is over the transmission rate of the communication, but not over the reliability of the communication.

In the following, transmission of sequences or blocks of n bits over a BSC is considered. In this case the input and the output are n-tuples or vectors defined over the extensions X^n and Y^n respectively. The conditional probabilities will be used:

$$P(\boldsymbol{X}/\boldsymbol{Y}) = \prod_{i=1}^{n} P(x_i/y_j)$$

Input and output vectors $\boldsymbol{X}$ and $\boldsymbol{Y}$ are words of n bits. By transmitting a given input vector $\boldsymbol{X}$, and making the assumption that the number of bits n is relatively large, the error probability p of the BSC determines that the output vector $\boldsymbol{Y}$ of this channel will differ in np positions with respect to the input vector $\boldsymbol{X}$.

On the other hand, the number of sequences of n bits with differences in np positions is equal to

$$\binom{n}{np} \tag{46}$$

By using the Stirling approximation [6]

$$n! \approx n^n e^{-n} \sqrt{2\pi n} \tag{47}$$

it can be shown that

$$\binom{n}{np} \approx 2^{n\Omega(p)} \tag{48}$$

This result indicates that for each input block of n bits, there exists $2^{n\Omega(p)}$ possible output sequences as a result of the errors introduced by the channel.

On the other hand, the output of the channel can be considered as a discrete source from which $2^{nH(Y)}$ typical sequences can be emitted. Then the amount

$$M = \frac{2^{nH(Y)}}{2^{n\Omega(p)}} = 2^{n[H(Y)-\Omega(p)]} \tag{49}$$

represents the maximum number of possible inputs able to be transmitted and to be converted by the distortion of the channel into non-overlapping sequences.

The smaller the error probability of the channel, the larger is the number of non-overlapping sequences. By applying the base 2 logarithmic function,

$$\log_2 M = n\,[H(Y) - \Omega(p)]$$

and then

$$R_{\mathrm{s}} = \frac{\log_2 M}{n} = H(Y) - \Omega(p) \tag{50}$$

The probability density function of the random variable Y depends on the probability density function of the message and on the statistical properties of the channel. There is in general terms a probability density function of the message X that can maximize the entropy $H(Y)$. If the input is characterized by a uniform probability density function and the channel is a BSC, the output has a maximum entropy, $H(Y) = 1$. This makes the expression (50) adopt its maximum value

$$R_{\mathrm{s}} = 1 - \Omega(p) \tag{51}$$

which is valid for the BSC.

This is indeed the parameter that has been defined as the channel capacity. This will therefore be the maximum possible transmission rate for the BSC if error-free transmission is desired over that channel. This could be obtained by the use of a rather sophisticated error coding technique.

Equation (51) for the BSC is depicted in Figure 1.11.

The channel capacity is the maximum transmission rate over that channel for reliable transmission. The worst case for the BSC is given when $p = 1/2$ because the extreme value $p = 1$ corresponds after all to a transmission where the roles of the transmitted symbols are interchanged (binary transmission).

So far, a description of the channel coding theorem has been developed by analysing the communication channel as a medium that distorts the sequences being transmitted.

The channel coding theorem is stated in the following section.

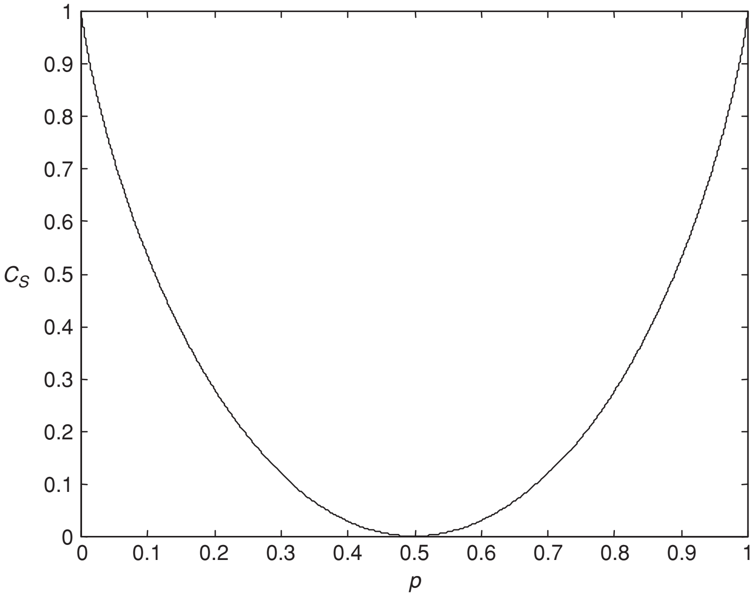


Figure 1.11 Channel capacity for the BSC

1.9.3 Channel Coding Theorem

The channel capacity of a discrete memoryless channel is equal to

$$C_s = \max_{P(x_i)} I(X, Y) \text{ bits per symbol} \tag{52}$$

The channel capacity per unit time C is related to the channel capacity C_s by the expression $C = sC_s$. If the transmission rate R fits the condition $R < C$, then for an arbitrary value $\varepsilon > 0$, there exists a code with block length n that makes the error probability of the transmission be less than ε. If $R > C$ then there is no guarantee of reliable transmission; that is, there is no guarantee that the arbitrary value of ε is a bound for the error probability, as it may be exceeded. The limiting value of this arbitrary constant ε is zero.

Example 1.11: Determine the channel capacity of the channel of Figure 1.12 if all the input symbols are equally likely, and

$$P(y_1/x_1) = P(y_2/x_2) = P(y_3/x_3) = 0.5$$
$$P(y_1/x_2) = P(y_1/x_3) = 0.25$$
$$P(y_2/x_1) = P(y_2/x_3) = 0.25$$
$$P(y_3/x_1) = P(y_3/x_2) = 0.25$$

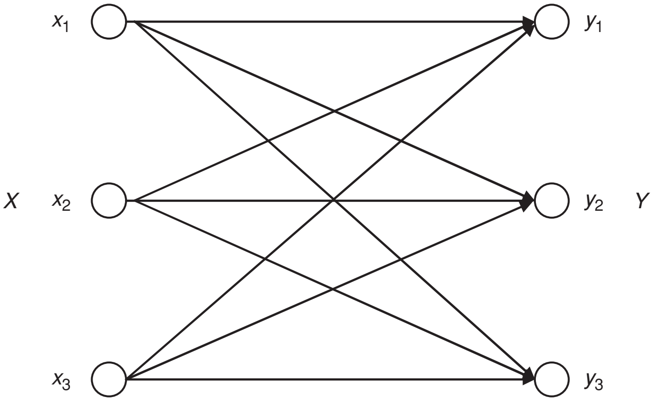


Figure 1.12 Example 1.11

The channel capacity can be calculated by first determining the mutual information and then maximizing this parameter. This maximization consists of looking for the input probability density function that makes the output entropy be maximal.

In this case the input probability density function is uniform and this makes the output probability density function be maximum. However this is not always the case. In a general case, the probability density function should be selected to maximize the mutual information. For this example,

$$H(Y/X) = P(x_1)H(Y/X = x_1) + P(x_2)H(Y/X = x_2) + P(x_3)H(Y/X = x_3)$$

and

$$\begin{aligned} H(Y/X = x_1) = H(Y/X = x_2) = H(Y/X = x_3) &= \frac{1}{4}\log_2(4) + \frac{1}{4}\log_2(4) + \frac{1}{2}\log_2(2) \\ &= 0.5 + 0.5 + 0.5 = 1.5 \\ H(Y/X) &= 1.5 \end{aligned}$$

Therefore,

$$I(X, Y) = H(Y) - 1.5$$

The output entropy is maximal for an output alphabet with equally likely symbols, so that

$$H(Y) = \frac{1}{3}\log_2(3) + \frac{1}{3}\log_2(3) + \frac{1}{3}\log_2(3) = \log_2(3) = 1.585$$

Then

$$C_{\text{s}} = 1.585 - 1.5 = 0.085 \text{ bits per symbol}$$

This rather small channel capacity is a consequence of the fact that each input symbol has a probability of 1/2 of emerging from the channel in error.

1.10 Signal Spaces and the Channel Coding Theorem

The theory of vector spaces can be applied to the field of the communication signals and is a very useful tool for understanding the Shannon channel coding theorem [2, 5, 6].

For a given signal $x(t)$ that is transmitted through a continuous channel with a bandwidth B, there is an equivalent representation that is based on the sampling theorem:

$$x(t) = \sum_k x_k \ \mathrm{sinc}(2Bt - k) \tag{53}$$

where

$$x_k = x(kT_\mathrm{s}) \quad \text{and} \quad T_\mathrm{s} = 1\big/2B \tag{54}$$

with $x_k = x(kT_\mathrm{s})$ being the samples of the signal obtained at a sampling rate $1\big/T_\mathrm{s}$.

Signals are in general power limited, and this power limit can be expressed as a function of the samples x_k as

$$P = \overline{x^2} = \overline{x_k^2} \tag{55}$$

Assuming that the signal has duration T, this signal can be represented by a discrete number of samples $n = T\big/T_\mathrm{s} = 2BT$. This means that the n numbers $x_1, x_2, \ldots, x_n$ represent this signal. This is true because of the sampling theorem, which states that the signal can be perfectly reconstructed if this set of n samples is known. This set of numbers can be thought of as a vector, which becomes a vectorial representation of the signal, with the property of allowing us a perfect reconstruction of this signal by calculating

$$x(t) = \sum_k x(kT_\mathrm{s}) \ \mathrm{sinc}(f_\mathrm{s}t - k)$$

$$f_\mathrm{s} = \frac{1}{T_\mathrm{s}} \geq 2W \tag{56}$$

where W is the bandwidth of the signal that has to fit the condition

$$W \leq B \leq f_\mathrm{s} - W \tag{57}$$

This vector represents the signal $x(t)$ and is denoted as $\boldsymbol{X} = (x_1, x_2, \ldots, x_n)$ with $n = 2BT = 2WT$. The reconstruction of the signal $x(t)$, based on this vector representation [expression (53)], is given in terms of a signal representation over a set of orthogonal functions like the sinc functions. The representative vector is n dimensional. Its norm can be calculated as

$$||\boldsymbol{X}||^2 = x_1^2 + x_2^2 + \cdots + x_n^2 = \sum_{i=1}^{n} x_i^2 \tag{58}$$

If the number of samples is large, $n \gg 1$, and

$$\frac{1}{n}||\boldsymbol{X}||^2 = \frac{1}{n}\sum_{i=1}^{n} x_i^2 = \overline{x_k^2} = P \tag{59}$$

$$||\boldsymbol{X}|| = \sqrt{nP} = \sqrt{2BTP} \tag{60}$$

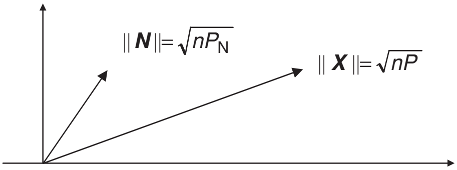

$\|\boldsymbol{N}\| = \sqrt{nP_{\mathrm{N}}}$
$\|\boldsymbol{X}\| = \sqrt{nP}$

Figure 1.13 Vector representation of signals

so the norm of the vector is proportional to its power. By allowing the components of the vector $\boldsymbol{X}$ vary through all their possible values, a hypersphere will appear in the corresponding vector space. This hypersphere is of radius $||\boldsymbol{X}||$, and all the possible vectors will be enclosed by this sphere. The volume of this hypersphere is equal to $V_{\mathrm{ol,n}} = K_n||\boldsymbol{X}||^n$.

As noise is also a signal, it can adopt a vector representation. This signal is usually passed through a filter of bandwidth B and then sampled, and so this set of samples constitutes a vector that represents the filtered noise. This vector will be denoted as $\boldsymbol{N} = (N_1, N_2, \ldots, N_n)$, and if P_{N} is the noise power, then this vector has a norm equal to $||\boldsymbol{N}|| = \sqrt{nP_{\mathrm{N}}}$. Thus, signals and noise have vector representation as shown in Figure 1.13.

Noise in this model is additive and independent of (uncorrelated with) the transmitted signals. During transmission, channel distortion transforms the input vector $\boldsymbol{X}$ into an output vector $\boldsymbol{Y}$ whose norm will be equal to $||\boldsymbol{Y}|| = ||\boldsymbol{X} + \boldsymbol{N}|| = \sqrt{n(P + P_{\mathrm{N}})}$ [2] (see Figure 1.14). (Signal and noise powers are added as they are uncorrelated.)

1.10.1 Capacity of the Gaussian Channel

The Gaussian channel resulting from the sampling of the signals is a discrete channel, which is described in Figure 1.15.

The variable N represents the samples of a Gaussian variable and is in turn a Gaussian random variable with squared variance P_{N}. The signal has a power P. If all the variables are represented by vectors of length n, they are related by

$$\boldsymbol{Y} = \boldsymbol{X} + \boldsymbol{N} \tag{61}$$

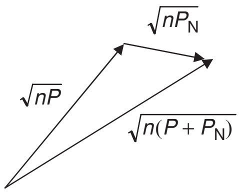


Figure 1.14 Vector addition of signals and noise

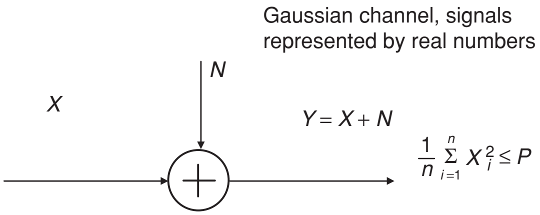

Gaussian channel, signals represented by real numbers
N
X
Y = X + N
$\frac{1}{n}\sum_{i=1}^{n} X_i^2 \le P$

Figure 1.15 Gaussian channel

If the number of samples, n, is large, the noise power can be calculated as the average of the noise samples:

$$\frac{1}{n}\sum_{i=1}^{n} N_i^2 = \frac{1}{n}\sum_{i=1}^{n} |\boldsymbol{Y}-\boldsymbol{X}|^2 \le P_{\mathrm{N}} \tag{62}$$

which means that

$$|\boldsymbol{Y}-\boldsymbol{X}|^2 \le nP_{\mathrm{N}} \tag{63}$$

This can be seen as the noise sphere representing the tip of the output vector $\boldsymbol{Y}$ around the transmitted (or true) vector $\boldsymbol{X}$, whose radius is $\sqrt{nP_{\mathrm{N}}}$, which is proportional to the noise power at the input. Since noise and transmitted signals are uncorrelated,

$$\frac{1}{n}\sum_{i=1}^{n} y_i^2 = \frac{1}{n}\sum_{i=1}^{n} x_i^2 + \frac{1}{n}\sum_{i=1}^{n} N_i^2 \le P + P_{\mathrm{N}} \tag{64}$$

Then

$$|\boldsymbol{Y}|^2 \le n(P + P_{\mathrm{N}}) \tag{65}$$

and the output sequences are inside n-dimensional spheres of radius $\sqrt{n(P+P_{\mathrm{N}})}$ centred at the origin, as shown in Figure 1.16.

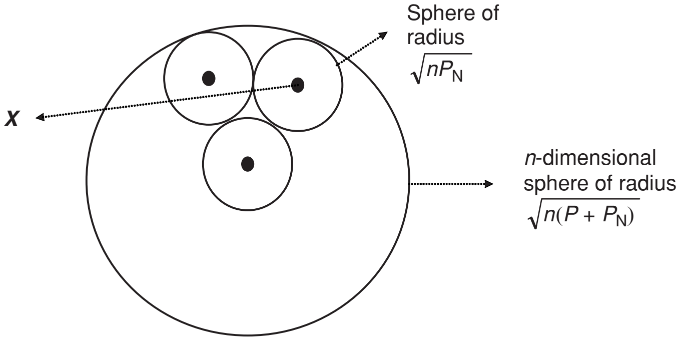

Sphere of radius $\sqrt{nP_{\mathrm{N}}}$
$\boldsymbol{X}$
n-dimensional sphere of radius $\sqrt{n(P+P_{\mathrm{N}})}$

Figure 1.16 A representation of the output vector space

Figure 1.16 can be understood as follows. The transmission of the input vector $\boldsymbol{X}$ generates an associated sphere whose radius is proportional to the noise power of the channel, $\sqrt{nP_{\mathrm{N}}}$. In addition, output vectors generate the output vector space, a hypersphere with radius $\sqrt{n(P+P_{\mathrm{N}})}$. The question is how many spheres of radius $\sqrt{nP_{\mathrm{N}}}$ can be placed, avoiding overlapping, inside a hypersphere of radius $\sqrt{n(P+P_{\mathrm{N}})}$?

For a given n-dimensional hypersphere of radius R_e, the volume is equal to

$$V_{\mathrm{ol,n}} = K_n R_{\mathrm{e}}^n \tag{66}$$

where K_n is a constant and R_{e} is the radius of the sphere. The number of non-overlapped messages that are able to be transmitted reliably in this channel is [2, 6]

$$M = \frac{K_n\,[n(P+P_{\mathrm{N}})]^{n/2}}{K_n(nP_{\mathrm{N}})^{n/2}} = \left(\frac{P+P_{\mathrm{N}}}{P_{\mathrm{N}}}\right)^{n/2} \tag{67}$$

The channel capacity, the number of possible signals that can be transmitted reliably, and the length of the transmitted vectors are related as follows:

$$C_{\mathrm{s}} = \frac{1}{n}\log_2(M) = \frac{1}{n}\frac{n}{2}\log_2\left(1+\frac{P}{P_{\mathrm{N}}}\right) = \frac{1}{2}\log_2\left(1+\frac{P}{P_{\mathrm{N}}}\right) \tag{68}$$

A continuous channel with power spectral density $N_0/2$, bandwidth B and signal power P can be converted into a discrete channel by sampling it at the Nyquist rate. The noise sample power is equal to

$$P_{\mathrm{N}} = \int_{-B}^{B}(N_0/2)\,\mathrm{d}f = N_0 B \tag{69}$$

Then

$$C_{\mathrm{s}} = \frac{1}{2}\log_2\left(1+\frac{P}{N_0 B}\right) \tag{70}$$

A given signal with bandwidth W transmitted through this channel and sampled at the Nyquist rate will fulfil the condition $W = B$, and will be represented by $2W = 2B$ samples per second.

The channel capacity per second is calculated by multiplying C_{s}, the capacity per symbol or sample, by the number of samples per second of the signal:

$$C = 2BC_{\mathrm{s}} = B\log_2\left(1+\frac{P}{N_0 B}\right)\ \text{bps} \tag{71}$$

This Shannon equation states that in order to reliably transmit signals through a given channel they should be selected by taking into account that, after being affected by noise, at the channel output the noise spheres must remain non-overlapped, so that each signal can be properly distinguished.

There will be therefore a number M of coded messages of length T, that is, of M coded vectors of n components each, resulting from evaluating how many spheres of radius $\sqrt{nP_{\mathrm{N}}}$

can be placed in a hypersphere (output vector space) of radius $\sqrt{n(P+P_{\mathrm{N}})}$, where

$$M = \left[\frac{\sqrt{n(P+P_{\mathrm{N}})}}{\sqrt{nP_{\mathrm{N}}}}\right]^n = \left(1+\frac{P}{P_{\mathrm{N}}}\right)^{n/2}$$

Assuming now that during time T, one of μ possible symbols is transmitted at r symbols per second, the number of different signals that can be constructed is

$$M = \mu^{rT} \tag{72}$$

In the particular case of binary signals, that is for $\mu = 2$, the number of possible non-overlapped signals is $M = 2^{rT}$.

The channel capacity, as defined by Shannon, is the maximum amount of information that can be transmitted per unit time. The Shannon theorem determines the amount of information that can be reliably transmitted through a given channel. The number of possible messages of length T that can be reliably transmitted is M. From this point of view, the combination of source and channel can be seen as a discrete output source Y with an alphabet of size M. The maximum entropy of this destination source (or information sink) is achieved when all the output symbols are equally likely, and this entropy is equal to $\log_2 M$, which is in turn the amount of information provided at this sink. The maximum rate measured in symbols per second is then equal to $(1/T)\log_2 M$. The channel capacity is measured as the limiting value of this maximum rate when the length of the message tends to infinity:

$$C = \lim_{T\to\infty}\frac{1}{T}\log_2 M \text{ bps} \tag{73}$$

Then, taking into account previous expressions,

$$C = \lim_{T\to\infty}\frac{1}{T}\log_2 M = \lim_{T\to\infty}\frac{1}{T}\log_2(\mu^{rT}) = \lim_{T\to\infty}\frac{rT}{T}\log_2\mu = r\log_2\mu \text{ bps} \tag{74}$$

This calculation considers the channel to be noise free. For instance, in the case of the binary alphabet, $\mu = 2$ and $C = r$. The number of distinguishable signals for reliable transmission is equal to

$$M \le \left(1+\frac{P}{P_{\mathrm{N}}}\right)^{n/2} \quad \text{with } n = 2BT \tag{75}$$

$$\begin{aligned} C &= \lim_{T\to\infty}\frac{1}{T}\log_2 M = \lim_{T\to\infty}\frac{1}{T}\log_2\left(1+\frac{P}{P_{\mathrm{N}}}\right)^{n/2} = \frac{2BT}{2T}\log_2\left(1+\frac{P}{P_{\mathrm{N}}}\right) \\ &= B\log_2\left(1+\frac{P}{P_{\mathrm{N}}}\right) \text{ bps} \end{aligned} \tag{76}$$

Example 1.12: Find the channel capacity of the telephony channel, assuming that the minimum signal-to-noise ratio of this system is $(P/P_{\mathrm{N}})_{\mathrm{dB}} = 30$ dB and the signal and channel transmission bandwidths are both equal to $W = 3$ kHz.

$$\text{As } (P/P_{\mathrm{N}})_{\mathrm{dB}} = 30 \text{ dB and } (P/P_{\mathrm{N}}) = 1000$$

$$C = 3000\log_2(1+1000) \approx 30 \text{ kbps}$$

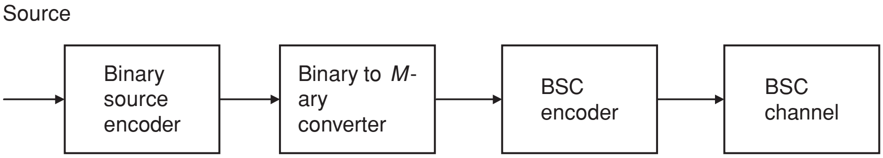


Figure 1.17 An encoder for the BSC

1.11 Error-Control Coding

Figure 1.17 shows the block diagram of an encoding scheme. The source generates information symbols at a rate R. The channel is a BSC, characterized by a capacity $C_s = 1 - \Omega(p)$ and a symbol rate s. Mutual information is maximized by previously coding the source in order to make it have equally likely binary symbols or bits. Then, an encoder takes blocks of bits and converts them into M-ary equally likely symbols [1, 2] that carry $\log_2 M$ bits per symbol each. In this way, information adopts a format suitable for its transmission through the channel. The BSC encoder represents each symbol by a randomly selected vector or word of n binary symbols.

Each binary symbol of the vector of length n carries an amount of information equal to $\log_2 M/n$. Since s symbols per second are transmitted, the encoded source information rate is

$$R = \frac{s \log_2 M}{n} \tag{77}$$

The Shannon theorem requires that

$$R \le C = sC_s$$

which in this case means that

$$\frac{\log_2 M}{n} \le C_s$$

$$\log_2 M \le nC_s \tag{78}$$

$$M = 2^{n(C_s - \delta)} \tag{79}$$

$$0 \le \delta < C_s \tag{80}$$

δ can be arbitrarily small, and in this case $R \to C$.

Assume now that the coded vectors of length n bits are in an n-dimensional vector space. If the vector components are taken from the binary field, the coordinates of this vector representation adopt one of the two possible values, one or zero. In this case the distance between any two vectors can be evaluated using the number of different components they have. Thus, if $\boldsymbol{c}$ is a given vector of the code, or codeword, and $\boldsymbol{c}'$ is a vector that differs in l positions with respect to $\boldsymbol{c}$, the distance between $\boldsymbol{c}$ and $\boldsymbol{c}'$ is l, a random variable with values between 0 and n. If the value of n is very large, vector $\boldsymbol{c}'$ is always within a sphere of radius $d < n$. The decoder will decide that $\boldsymbol{c}$ has been the transmitted vector when receiving $\boldsymbol{c}'$ if this received vector is inside the sphere of radius d and none of the remaining $M - 1$ codewords are inside that sphere. Incorrect decoding happens when the number of errors produced during transmission is such that the received vector is outside the sphere of radius d and lies in the sphere of another codeword different from $\boldsymbol{c}$. Incorrect decoding also occurs even if the received vector lies in the sphere of radius d, but another codeword is also inside that sphere. Then, the total error probability is equal to [5]

$$P_{\mathrm{e}} = P_{\mathrm{le}} + P_{\mathrm{ce}} \tag{81}$$

where P_{le} is the probability of the fact that the received vector is outside the sphere of radius d and P_{ce} is the probability that two or more codewords are inside the same sphere. It is noted that this event is possible because of the random encoding process, and so two or more codewords can be within the same sphere of radius d.

P_{le} is the probability of the error event $l \geq d$. Transmission errors are statistically independent and happen with a probability $p < 1/2$, and the number of errors l is a random variable governed by the binomial distribution

$$\bar{l} = np, \qquad \sigma^2 = n(1-p)p \tag{82}$$

If the sphere radius is adopted as

$$d = n\beta, \quad p < \beta < 1/2 \tag{83}$$

it is taken as slightly larger than the number of expected errors per word. The error probability P_{le} is then equal to

$$P_{\mathrm{le}} = P\left(l \geq d\right) \leq \left(\frac{\sigma}{d-\bar{l}}\right)^2 = \frac{p(1-p)}{n(\beta-p)^2} \tag{84}$$

and if the conditions expressed in (83) are true, then $P_{\mathrm{le}} \to 0$ as $n \to \infty$.

On the other hand, in order to estimate the error probability, a number m is defined to describe the number of words or vectors contained within the sphere of radius d surrounding a particular one of the M codewords. As before, Shannon assumed a random encoding technique for solving this problem. From this point of view, the remaining $M-1$ codewords are inside the n-dimensional vector space, and the probability that a randomly encoded vector or codeword is inside the sphere containing m vectors is

$$\frac{m}{2^n} \tag{85}$$

Apart from the particular codeword selected, there exist $M-1$ other code vectors, so that using equation (79)

$$\begin{aligned} P_{\mathrm{ce}} &= (M-1)\,m\,2^{-n} < M\,m\,2^{-n} < m\,2^{-n}\,2^{n(C_{\mathrm{s}}-\delta)} \\ &= m\,2^{-n}\,2^{n[1-\Omega(p)-\delta]} = m\,2^{-n[\Omega(p)+\delta]} \end{aligned} \tag{86}$$

All the m vectors that are inside the sphere of radius d, defined around the codeword $\boldsymbol{c}$, have d different positions with respect to the codeword, or less. The number of possible codewords with d different positions with respect to the codeword is equal to $\binom{n}{d}$. In general, the number

m of codewords inside the sphere of radius d is

$$m = \sum_{i=0}^{d} \binom{n}{i} = \binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{d}, \quad d = n\beta \tag{87}$$

Among all the terms in the above expression, the term $\binom{n}{d}$ is the largest, and it can be considered as a bound on the sum of the $d + 1$ other terms as follows:

$$m \leq (d+1)\binom{n}{d} = \frac{n!}{(n-d)!d!}(d+1) \tag{88}$$

Since $d = n\beta$, and by using the Stirling approximation for the factorial number $n!$ ($n! \approx n^n\, e^{-n}\sqrt{2\pi n}$ if $n \gg 1$),

$$m \leq (d+1)\binom{n}{d} = 2^{n\Omega(\beta)}\frac{n\beta + 1}{\sqrt{2\pi n\beta(1-\beta)}}$$

and by combining it with expression (86),

$$P_{\mathrm{ce}} \leq \frac{n\beta + 1}{\sqrt{2\pi n\beta(1-\beta)}} 2^{-n[\delta + \Omega(p) - \Omega(\beta)]} = \frac{n\beta + 1}{\sqrt{2\pi n\beta(1-\beta)}} 2^{-n\{\delta - [\Omega(\beta) - \Omega(p)]\}} \tag{89}$$

The above expression says that the random coding error probability P_{ce} tends to zero if $\delta > \Omega(\beta) - \Omega(p)$, which is a function of the parameter p, the error probability of the BSC, and if $n \to \infty$. Once again the error probability tends to zero if the length of the codeword tends to infinity. The value of the parameter δ is the degree of sacrifice of the channel capacity, and it should fit the condition $0 \leq \delta < C_{\mathrm{s}}$.

Finally, replacing the two terms of the error probability corresponding, respectively, to the effect of the noise and to the random coding [5], we obtain

$$P_{\mathrm{e}} = P_{\mathrm{le}} + P_{\mathrm{ce}} \leq \frac{p(1-p)}{n(\beta - p)^2} + \frac{n\beta + 1}{\sqrt{2\pi n\beta(1-\beta)}} 2^{-n\{\delta - [\Omega(\beta) - \Omega(p)]\}} \tag{90}$$

For a given value of p, if β is taken according to expression (83), and fitting also the condition $\delta > \Omega(\beta) - \Omega(p)$, then $\delta' = \delta - [\Omega(\beta) - \Omega(p)] > 0$ and the error probability is

$$P_{\mathrm{e}} = P_{\mathrm{le}} + P_{\mathrm{ce}} \leq \frac{K_1}{n} + \sqrt{n}K_2\, 2^{-nK_3} + \frac{K_4}{\sqrt{n}}\, 2^{-nK_3} \tag{91}$$

where K_1, K_2, K_3 and K_4 are positive constants. The first and third terms clearly tend to zero as $n \to \infty$, and the same happens with the term $\sqrt{n}K_2\, 2^{-nK_3} = \frac{\sqrt{n}K_2}{2^{nK_3}}$ if it is analysed using the L'Hopital rule. Hence, $P_e \to 0$ as long as $n \to \infty$, and so error-free transmission is possible when $R < C$.

1.12 Limits to Communication and their Consequences

In a communication system operating over the additive white Gaussian noise (AWGN) channel for which there exists a restriction on the bandwidth, the Nyquist and Shannon theorems are enough to provide a design framework for such a system [5, 7].

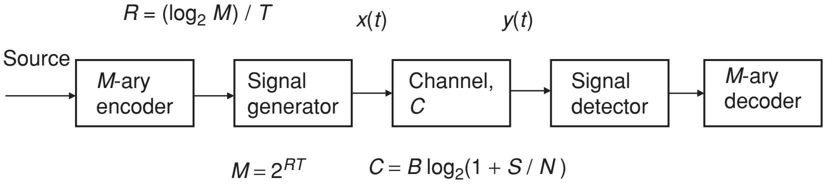


Figure 1.18 An ideal communication system

An ideal communication system characterized by a given signal-to-noise ratio $\frac{P}{P_N} = \frac{S}{N}$ and a given bandwidth B is able to perform error-free transmission at a rate $R = B \log_2(1 + S/N)$. The ideal system as defined by Shannon is one as seen in Figure 1.18 [5].

The source information is provided in blocks of duration T and encoded as one of the M possible signals such that $R = \log_2 M\big/T$. There is a set of $M = 2^{RT}$ possible signals. The signal $y(t) = x(t) + n(t)$ is the noisy version of the transmitted signal $x(t)$, which is obtained after passing through the band-limited AWGN channel. The Shannon theorem states that

$$\lim_{P_e \to 0} \lim_{T \to \infty} \frac{\log_2 M}{T} = B \log_2\left(1 + \frac{S}{N}\right) \tag{92}$$

The transmission rate of the communication system tends to the channel capacity, $R \to C$, if the coding block length, and hence the decoding delay, tends to infinity, $T \to \infty$. Then, from this point of view, this is a non-practical system.

An inspection of the expression $C = B \log_2 (1 + S/N)$ leads to the conclusion that both the bandwidth and the signal-to-noise ratio contribute to the performance of the system, as their increase provides a higher capacity, and their product is constant for a given capacity, and so they can be interchanged to improve the system performance. This expression is depicted in Figure 1.19.

For a band-limited communication system of bandwidth B and in the presence of white noise, the noise power is equal to $N = N_0 B$, where N_0 is the power spectral density of the noise in that channel. Then

$$\frac{C}{B} = \log_2\left(1 + \frac{S}{N_0 B}\right)$$

There is an equivalent expression for the signal-to-noise ratio described in terms of the average bit energy E_b and the transmission rate R.

If $R = C$ then

$$\frac{E_b}{N_0} = \frac{S}{N_0 R} = \frac{S}{N_0 C} \tag{93}$$

$$\frac{C}{B} = \log_2\left(1 + \frac{E_b}{N_0}\frac{C}{B}\right), \qquad 2^{C/B} = 1 + \frac{E_b}{N_0}\left(\frac{C}{B}\right) \tag{94}$$

$$\frac{E_b}{N_0} = \frac{B}{C}\left(2^{C/B} - 1\right) \tag{95}$$

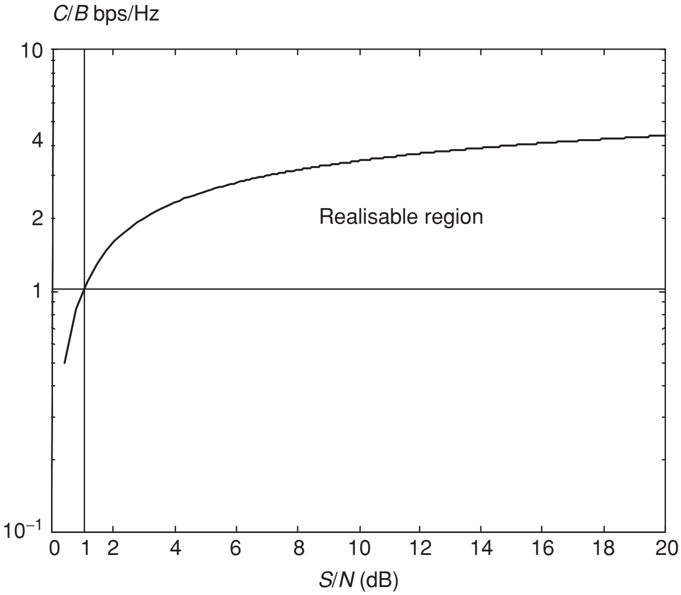


Figure 1.19 Practical and non-practical operation regions

The Shannon limit can be now analysed from equation

$$\frac{C}{B} = \log_2\left(1 + \frac{E_\mathrm{b}}{N_0}\frac{C}{B}\right) \tag{96}$$

making use of the expression

$$\lim_{x\to 0}(1+x)^{1/x} = e$$

where $x = \frac{E_\mathrm{b}}{N_0}\left(\frac{C}{B}\right)$.

Since $\log_2(1+x) = x\frac{1}{x}\log_2(1+x) = x\log_2\left[(1+x)^{1/x}\right]$ [7],

$$\begin{aligned}\frac{C}{B} &= \frac{C}{B}\frac{E_\mathrm{b}}{N_0}\log_2\left(1+\frac{E_\mathrm{b}}{N_0}\frac{C}{B}\right)^{N_0 B/CE_\mathrm{b}}\\ \Rightarrow\ 1 &= \frac{E_\mathrm{b}}{N_0}\log_2\left(1+\frac{E_\mathrm{b}}{N_0}\frac{C}{B}\right)^{N_0 B/CE_\mathrm{b}}\end{aligned} \tag{97}$$

If $\frac{C}{B} \to 0$, obtained by letting $B \to \infty$,

$$\frac{E_\mathrm{b}}{N_0} = \frac{1}{\log_2(e)} = 0.693$$

or

$$\left(\frac{E_\text{b}}{N_0}\right)_\text{dB} = -1.59 \text{ dB} \tag{98}$$

This value is usually called the Shannon limit. This is a performance bound on the value of the ratio $E_\text{b}\big/N_0$, using a rather sophisticated coding technique, and for which the channel bandwidth and the code length n are very large. This means that if the ratio $E_\text{b}\big/N_0$ is kept slightly higher than this value, it is possible to have error-free transmission by means of the use of such a sophisticated coding technique.

From the equation

$$2^{C/B} = 1 + \frac{E_\text{b}}{N_0}\left(\frac{C}{B}\right) \tag{99}$$

a curve can be obtained relating the normalized bandwidth B/C (Hz/bps) and the ratio $E_\text{b}\big/N_0$. For a particular transmission rate R,

$$2^{R/B} \le 1 + \frac{E_\text{b}}{N_0}\left(\frac{R}{B}\right) \tag{100}$$

$$\frac{E_\text{b}}{N_0} \ge \left(\frac{B}{R}\right)\left(2^{R/B} - 1\right) \tag{101}$$

Expression (100) can be also depicted, and it defines two operating regions, one of practical use and another one of impractical use [1, 2, 5, 7]. This curve is seen in Figure 1.20, which represents the quotient R/B as a function of the ratio E_b/N_0. The two regions are separated by the curve that corresponds to the case $R = C$ [equation (99)]. This curve shows the Shannon limit when $R/B \to 0$. However, for each value of R/B, there exists a different bound, which can be obtained by using this curve.

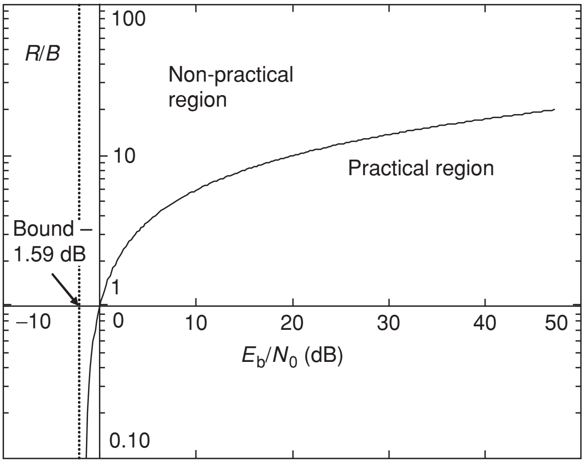


Figure 1.20 Practical and non-practical operation regions. The Shannon limit

Bibliography and References

[1] Shannon, C. E., “A mathematical theory of communication,” *Bell Syst. Tech. J.*, vol. 27, pp. 379–423, 623–656, July and October 1948.

[2] Shannon, C. E., “Communications in the presence of noise,” *Proc. IEEE*, vol. 86, no. 2, pp. 447–458, February 1998.

[3] McEliece, R. J., *The Theory of Information and Coding*, Addison-Wesley, Massachusetts, 1977.

[4] Abramson, N., *Information Theory and Coding*, McGraw-Hill, New York, 1963.

[5] Carlson, B., *Communication Systems: An Introduction to Signals and Noise in Electrical Communication*, 3rd Edition, McGraw-Hill, New York, 1986.

[6] Proakis, J. G. and Salehi, M., *Communication Systems Engineering*, Prentice Hall, New Jersey, 1993.

[7] Sklar, B., *Digital Communications, Fundamentals and Applications*, Prentice Hall, New York, 1993.

[8] Proakis, J. G., *Digital Communications*, 2nd Edition, McGraw-Hill, New York, 1989.

[9] Adámek, J., *Foundations of Coding: Theory and Applications of Error-Correcting Codes with an Introduction to Cryptography and Information Theory*, Wiley Interscience, New York, 1991.

Problems

1.1 A DMS produces symbols with the probabilities as given in Table P.1.1.

Table P.1.1 Probabilities of the symbols of a discrete source

A	0.4
B	0.2
C	0.2
D	0.1
E	0.05
F	0.05

(a) Find the self-information associated with each symbol, and the entropy of the source.

(b) Calculate the maximum possible source entropy, and hence determine the source efficiency.

1.2 (a) Calculate the entropy of a DMS that generates five symbols $\{A, B, C, D, E\}$ with probabilities $P_A = 1/2$, $P_B = 1/4$, $P_C = 1/8$, $P_D = 1/16$ and $P_E = 1/16$.

(b) Determine the information contained in the emitted sequence DADED.

1.3 Calculate the source entropy, the transinformation $I(X, Y)$ and the capacity of the BSC defined in Figure P.1.1.

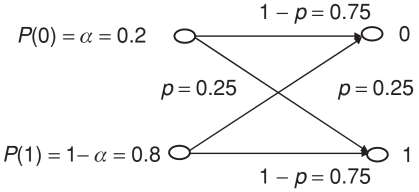


Figure P.1.1 A binary symmetric channel

1.4 Show that for the BSC, the entropy is maximum when all the symbols of the discrete source are equally likely.

1.5 An independent-symbol binary source with probabilities 0.25 and 0.75 is transmitted over a BSC with transition (error) probability $p = 0.01$. Calculate the equivocation $H(X/Y)$ and the transinformation $I(X, Y)$.

1.6 What is the capacity of the cascade of BSCs as given in Figure P.1.2?

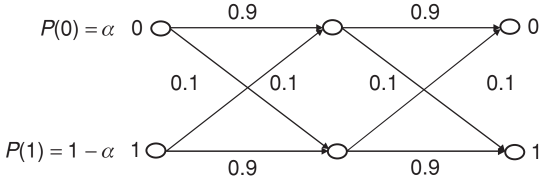


Figure P.1.2 A cascade of BSCs

1.7 Consider a binary channel with input and output alphabets {0, 1} and the transition probability matrix:

$$\boldsymbol{P}_{\text{ch}} = \begin{bmatrix} 3/5 & 2/5 \\ 1/5 & 4/5 \end{bmatrix}$$

Determine the *a priori* and the two *a posteriori* entropies of this channel.

1.8 Find the conditional probabilities $P(x_i/y_j)$ of the BEC with an erasure probability of 0.469, when the source probabilities are 0.25 and 0.75. Hence find the equivocation, transinformation and capacity of the channel.

1.9 Calculate the transinformation and estimate the capacity of the non-symmetric erasure channel given in Figure P.1.3.

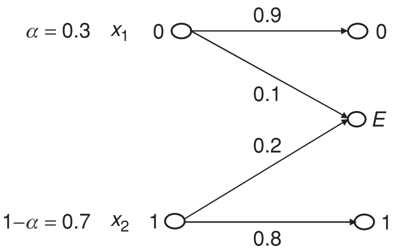


Figure P.1.3 A non-symmetric erasure channel

1.10 Figure P.1.4 shows a non-symmetric binary channel. Show that in this case $I(X, Y) = \Omega\,[q + (1 - p - q)\alpha] - \alpha\Omega(p) - (1 - \alpha)\Omega(q)$.

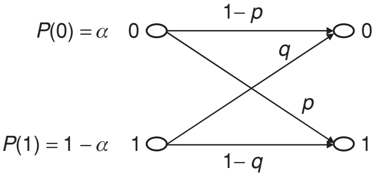


Figure P.1.4 A non-symmetric binary channel

1.11 Find the transinformation, the capacity and the channel efficiency of the symmetric erasure and error channel given in Figure P.1.5.

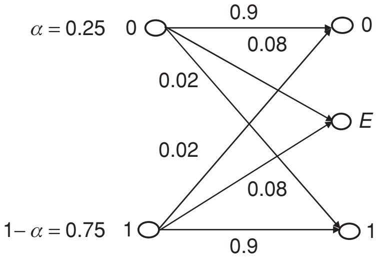


Figure P.1.5 A symmetric erasure and error channel

1.12 Consider transmission over a telephone line with a bandwidth $B = 3$ kHz. This is an analogue channel which can be considered as perturbed by AWGN, and for which the power signal-to-noise ratio is at least 30 dB.

(a) What is the capacity of this channel, in the above conditions?

(b) What is the required signal-to-noise ratio to transmit an M-ary signal able to carry 19,200 bps?

1.13 An analogue channel perturbed by AWGN has a bandwidth $B = 25$ kHz and a power signal-to-noise ratio SNR of 18 dB. What is the capacity of this channel in bits per second?

2

Block Codes

2.1 Error-Control Coding

One of the predictions made in the Shannon channel coding theorem is that a rather sophisticated coding technique can convert a noisy channel (unreliable transmission) into an error-free channel (reliable transmission).

Demonstration of the theorem about the possibility of having error-free transmission is done by using a coding technique of a random nature [1, 3]. In this technique, message words are arranged as blocks of k bits, which are randomly assigned codewords of n bits, $n > k$, in an assignment that is basically a bijective function characterized by the addition of redundancy. This bijective assignment allows us to uniquely decode each message. This coding technique is essentially a block coding method. However, what is not completely defined in the theorem is a constructive method for designing such a sophisticated coding technique.

There are basically two mechanisms for adding redundancy, in relation to error-control coding techniques [4]. These two basic mechanisms are block coding and convolutional coding. This chapter is devoted to block coding.

Errors can be detected or corrected. In general, for a given code, more errors can be detected than corrected, because correction requires knowledge of both the position and the magnitude of the error.

2.2 Error Detection and Correction

For a given practical requirement, detection of errors is simpler than the correction of errors. The decision for applying detection or correction in a given code design depends on the characteristics of the application. When the communication system is able to provide a full-duplex transmission (that is, a transmission for which the source and the destination can communicate at the same time, and in a two way mode, as it is in the case of telephone connection, for instance), codes can be designed for detecting errors, because the correction is performed by requiring a repetition of the transmission. These schemes are known as automatic repeat reQuest (ARQ) schemes.

Essentials of Error-Control Coding Jorge Castiñeira Moreira and Patrick Guy Farrell

In any ARQ system there is the possibility of requiring a retransmission of a given message. There are on the other hand communication systems for which the full-duplex mode is not allowed. An example of one of them is the communication system called paging, a sending of alphanumerical characters as text messages for a mobile user. In this type of communication system, there is no possibility of requiring retransmission in the case of a detected error, and so the receiver has to implement some error-correction algorithm to properly decode the message. This transmission mode is known as forward error correction (FEC).

2.2.1 Simple Codes: The Repetition Code

One of the simplest ways of performing coding is to repeat a transmitted symbol n times. If this transmission uses a binary alphabet then the bit '1' is usually represented by a sequence of n '1's, while the bit '0' is represented by a sequence of n '0's.

If errors happen randomly and with an error probability $P_\mathrm{e} = p$ [as happens in the case of the binary symmetric channel (BSC)], the binomial distribution describes the probability of having i errors in a word of n bits:

$$\begin{aligned} &P(i,n) = \binom{n}{i} p^i (1-p)^{n-i} \cong \binom{n}{i} p^i \quad p \ll 1 \\ &\binom{n}{i} = \frac{n!}{i!(n-i)!} \end{aligned} \tag{1}$$

Usually the value of p is small enough to validate the approximation made in equation (1). On the other hand and for the same reason, it will be also true that if $p \ll 1$, then the probability of having i errors is higher than that of having $i+1$ errors; that is, $P(i+1,n) \ll P(i,n)$.

For the particular case of a repetition code with $n = 3$ for instance, the codewords are (111) and (000). Usually, the former represents the bit '1' and the latter represents the bit '0'. An error-detection rule will be that the reception of any of the possible three-bit words different from the codewords (six in total) will be considered as an error event. Thus, error detection is possible, and for instance the reception of the word (110) can be considered as an error event. Since codewords and, in general, binary words can be represented as vectors in a vector space, coding can be understood as a procedure in which the messages are represented by codewords selected from an expanded vector space. Thus, coding basically means an expansion of the dimension of the message vector space from which some vectors are selected as codewords, while other vector are not. In this example there are eight vectors in the expanded vector space, from which only two are selected as codewords. The remaining six possible received patterns are considered error patterns. It can be said that this code can detect error patterns of one or two errors. By considering that the probability of having one error is higher than that of having two errors, the patterns

$$(110), \quad (101) \quad \text{and} \quad (011)$$

will be considered as transmitted sequences of three '1's that, affected by noise, suffered from one error. According to this rule, the decoder will decide that by receiving these words, the transmitted codeword was (111), and the transmitted message was '1'. In the same way, the patterns

$$(001), \quad (010) \quad \text{and} \quad (100)$$

will be considered as sequences of three '0's that, affected by noise, suffered from one error. According to this rule, the decoder will decide that on receiving these words, the transmitted codeword was (000), and the transmitted message was '0'. This decoding rule cannot correct two-error patterns.

If this code is used in an error-detection scheme, a pattern of three errors cannot be detected, because such an error event means that, for instance, a sequence of three '0's is converted into a sequence of three '1's, which is a valid codeword, and vice versa. When receiving this pattern, the decoder can only assume that it is a valid codeword, thus being unable to detect this error event. The word error probability (P_{we}) for this case can be evaluated as

$$P_{\mathrm{we}} = P(3,3) = p^3$$

If this code is used in an error-correction mode, a two-error pattern event makes the decoder fail, and so the word error probability is given by the following expression:

$$P_{\mathrm{we}} = P(2,3) + P(3,3) = 3p^2(1-p) + p^3 = 3p^2 - 2p^3$$

In all the cases, $P_{\mathrm{e}} = p$ is the error probability the communication system has without the use of coding. For the BSC, this probability is that of a given bit being converted into its complement. After the application of coding, P_{we} measures the error probability per word. In general, this error probability will be smaller than the error probability of the uncoded system. However, the use of coding involves the transmission of redundancy, which means that the transmission rate has deteriorated. At the end of this chapter, consideration will be given to making a fair comparison between the coded and uncoded cases (see Section 2.11.1).

The code rate is defined as the ratio between the number of information bits and the number of coded bits:

$$R_{\mathrm{c}} = \frac{k}{n} \tag{2}$$

Repetition codes have a large error-detection/correction capability, but a very small code rate. In this book some other coding techniques will be analysed that provide an error-correction capability similar to the repetition code, but with a better code rate. They will be thus considered as better coding techniques than repetition coding. The repetition code is a nice example that shows the difference between error detection and correction.

2.3 Block Codes: Introduction and Parameters

Block coding of information is organized so that the message to be transmitted, basically presented in binary format, is grouped into blocks of k bits, which are called the message bits, constituting a set of 2^k possible messages. The encoder takes each block of k bits, and converts it into a longer block of $n > k$ bits, called the coded bits or the bits of the codeword. In this procedure there are $(n-k)$ bits that the encoder adds to the message word, which are usually called redundant bits or parity check bits. As explained in the previous chapter, error-control coding requires the use of a mechanism for adding redundancy to the message word. This redundancy addition (encoding operation) can be performed in different ways, but always in a way that by applying the inverse operation (decoding) at the decoder the message information can be successfully recovered.

The final step in the decoding process involves the application of the decoding procedure, and then the discarding of the redundancy bits, since they do not contain any message information. These types of codes are called block codes and are denoted by $C_{\mathrm{b}}(n, k)$. The rate of the code, $R_{\mathrm{c}} = k/n$, is a measure of the level of redundancy applied in a given code, being the ratio of the coded bits that represent information or message bits. This is closely related to the increased bandwidth needed in the transmission when coding is used.

If for example a block code with code rate $R_{\mathrm{c}} = 2/3$ is utilized, then it should be taken into account that in the same time T during which in the uncoded scheme the two signals representing the two message bits are transmitted, there now will be in the coded case three signals transmitted, so that each signal has a duration that changes from being $T/2$ for the uncoded case to $T/3$ for the coded case. This means that the spectral occupancy is higher in the coded case. Equivalently, storing coded digital information will require more physical space than that of the uncoded information. Thus an important practical consideration is to keep the code rate at a reasonable level, even though in general it leads to a trade-off with respect to the error-correction capability of the code.

Since the 2^k messages are converted into codewords of n bits, this encoding procedure can be understood as an expansion of the message vector space of size 2^k to a coded vector space of larger size 2^n, from which a set of 2^k codewords is conveniently selected. Block codes can be properly analysed by using vector space theory.

2.4 The Vector Space over the Binary Field

A vector space is essentially a set of vectors ruled by certain conditions, which are verified by performing operations over these vectors, operations that are usually defined over a given field F.

The vector space V consists of a set of elements over which a binary operation called addition, denoted by the symbol $\oplus$, is defined. If F is a field, the binary operation called product, denoted by the symbol $\bullet$, is defined between an element of the field F and the vectors of the space V. Thus, V is a vector space that is said to be defined over the field F [4–9].

The following conditions are verified for a given vector space:

- V is a commutative group for the binary operation of addition.
- For any $a \in F$ and any $\boldsymbol{u} \in V$, $a \bullet \boldsymbol{u} \in V$.
- For any $\boldsymbol{u}, \boldsymbol{v} \in V$ and any $a, b \in F$, $a \bullet (\boldsymbol{u} + \boldsymbol{v}) = a \bullet u + a \bullet v$

  and also $(a + b) \bullet (u) = a \bullet \boldsymbol{u} + b \bullet \boldsymbol{u}$.

- For any $\boldsymbol{u} \in V$ and any $a, b \in F$, $(a \bullet b) \bullet \boldsymbol{u} = a \bullet (b \bullet \boldsymbol{u})$.
- If 1 is the unit element in F then $1 \bullet \boldsymbol{u} = \boldsymbol{u}$ for any $\boldsymbol{u} \in V$.

It is also true that for this type of vector space

- For any $\boldsymbol{u} \in V$, if 0 is the zero element in F, $0 \bullet \boldsymbol{u} = \mathbf{0}$.
- For any scalar $c \in F$, $c \bullet \mathbf{0} = \mathbf{0}$.
- For any scalar $c \in F$ and any vector $\boldsymbol{u} \in V$, $(-c) \bullet \boldsymbol{u} = c \bullet (-\boldsymbol{u}) = -(c \bullet \boldsymbol{u})$ where $(-c) \bullet \boldsymbol{u} = c \bullet (-\boldsymbol{u})$ is the additive inverse of $c \bullet \boldsymbol{u}$.

A very useful vector space for the description of block codes is the vector space defined over the binary field, or Galois field GF(2). Galois fields GF(q) are defined for all the prime numbers q and their powers. The binary field GF(2) is a particular case of a Galois field for which $q = 2$. Consider an ordered sequence of n components $(a_0, a_1, \ldots, a_{n-1})$ where each component a_i is an element of the field GF(2), that is, an element adopting one of the two possible values 0 or 1. This sequence will be called an n-component vector. There will be total of 2^n vectors. The corresponding vector space for this set of vectors will be denoted as V_n.

The binary addition operation $\oplus$ is defined for this vector space as follows: if $\boldsymbol{u} = (u_1, u_2, \ldots, u_{n-1})$ and $\boldsymbol{v} = (v_1, v_2, \ldots, v_{n-1})$ are vectors in V_n, then

$$\boldsymbol{u} \oplus \boldsymbol{v} = (u_1 \oplus v_1, u_2 \oplus v_2, \ldots, u_n \oplus v_n) \tag{3}$$

where $\oplus$ is the classic modulo-2 addition. Since the sum vector is also an n-component vector, this vector also belongs to the vector space V_n, and so the vector space is said to be closed under the addition operation $\oplus$. The addition of any two vectors of a given vector space is also another vector of the same vector space.

Operations defined over the binary field are modulo-2 addition and multiplication. They are described as follows:

Modulo-2 addition

$0 \oplus 0 = 0$
$0 \oplus 1 = 1$
$1 \oplus 0 = 1$
$1 \oplus 1 = 0$

Modulo-2 multiplication

$0 \bullet 0 = 0$
$0 \bullet 1 = 0$
$1 \bullet 0 = 0$
$1 \bullet 1 = 1$

V_n is a commutative group under the addition operation. The all-zero vector $\boldsymbol{0} = (0, 0, \ldots, 0)$ is also in the vector space and is the identity for the addition operation:

$$\boldsymbol{u} \oplus \boldsymbol{0} = (u_1 \oplus 0, u_2 \oplus 0, \ldots, u_n \oplus 0) = \boldsymbol{u} \tag{4}$$

and

$$\boldsymbol{u} \oplus \boldsymbol{u} = (u_1 \oplus u_1, u_2 \oplus u_2, \ldots, u_n \oplus u_n) = \boldsymbol{0} \tag{5}$$

Each vector of a vector space defined over the binary field is its own additive inverse. It can be shown that the vector space defined over GF(2) is a commutative group, so that associative and commutative laws are verified. The product between a vector of the vectorial space $\boldsymbol{u} \in V$ and a scalar of the binary field $a \in$ GF(2) can be defined as

$$\mathrm{a} \bullet \boldsymbol{u} = (a \bullet u_1, a \bullet u_2, \ldots, a \bullet u_{n-1}) \tag{6}$$

where $a \bullet u_i$ is a modulo-2 multiplication. It can be shown that the addition and scalar multiplication fit the associative, commutative and distributive laws, so that the set of vectors V_n is a vector space defined over the binary field GF(2).

Example 2.1: The vector space of vectors with four components consists of $2^4 = 16$ vectors:

$$\begin{aligned} V_4 = \{&(0000), (0001), (0010), (0011), (0100), (0101), (0110), (0111),\\ &(1000), (1001), (1010), (1011), (1100), (1101), (1110), (1111)\} \end{aligned}$$

The addition of any two of these vectors is another vector in the same vector space:

$$(1011) \oplus (0010) = (1001)$$

For each vector of this vector space, there are only two scalar multiplications:

$$\begin{gathered} 0 \bullet (1011) = (0000) \\ 1 \bullet (1011) = (1011) \end{gathered}$$

2.4.1 Vector Subspaces

For a given set of vectors forming a vector space V defined over a field F, it is possible to find a subset of vectors inside the vector space V, which can obey all the conditions for also being a vector space. This subset S is called a subspace of the vector space V. This non-empty subset S of the vector space V is a subspace if the following conditions are obeyed:

- For any two vectors in S, $\boldsymbol{u}, \boldsymbol{v} \in S$, the sum vector $(\boldsymbol{u} + \boldsymbol{v}) \in S$.
- For any element of the field $a \in F$ and any vector $\boldsymbol{u} \in S$, the scalar multiplication $\boldsymbol{a} \bullet \boldsymbol{u} \in S$.

Example 2.2: The following subset is a subspace of the vector space V_4:

$$S = \{(0000), (1001), (0100), (1101)\}$$

On the other hand, if $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k\}$ is a set of vectors of the vector space V defined over F and $a_1, a_2, \ldots, a_k$ are scalar numbers of the field F, the sum

$$a_1 \bullet \boldsymbol{v}_1 \oplus a_2 \bullet \boldsymbol{v}_2 \oplus \cdots \oplus a_k \bullet \boldsymbol{v}_k \tag{7}$$

is called a linear combination of the vectors $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k\}$. Addition of linear combinations and multiplication of a linear combination by an element of the field F are also linear combinations of the vectors $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k\}$.

Theorem 2.1: If $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k\}$ are k vectors in V defined over F, the set of all the linear combinations of $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k\}$ is a subspace S of V.

Example 2.3: By considering two vectors (1001) and (0100) of the vector space V_4, their linear combinations form the same subspace S as shown in the above example:

$$\begin{gathered}0 \bullet (1001) \oplus 0 \bullet (0100) = (0000)\\ 0 \bullet (1001) \oplus 1 \bullet (0100) = (0100)\\ 1 \bullet (1001) \oplus 0 \bullet (0100) = (1001)\\ 1 \bullet (1001) \oplus 1 \bullet (0100) = (1101)\end{gathered}$$

A set of k vectors $\{\boldsymbol{v_1}, \boldsymbol{v_2}, \ldots, \boldsymbol{v_k}\}$ is said to be linearly dependent if and only if there exist k scalars of the field F, not all equal to zero, such that a linear combination is equal to the all-zero vector:

$$a_1 \bullet \boldsymbol{v_1} \oplus a_2 \bullet \boldsymbol{v_2} \oplus \cdots \oplus a_k \bullet \boldsymbol{v_k} = \boldsymbol{0} \tag{8}$$

If the set of vectors is not linearly dependent, then this set is said to be linearly independent.

Example 2.4: Vectors (1001), (0100) and (1101) are linearly dependent because

$$1 \bullet (1001) \oplus 1 \bullet (0100) \oplus 1 \bullet (1101) = (0000)$$

A set of vectors is said to generate a vector space V if each vector in that vector space is a linear combination of the vectors of the set.

In any vector space or subspace there exists a set of at least B_{li} linearly independent vectors that generate such a vectorial space or subspace.

For a given vector space V_n defined over GF(2), the following set of vectors

$$\begin{aligned}\boldsymbol{e_0} &= (1, 0, \ldots 0)\\ \boldsymbol{e_1} &= (0, 1, \ldots, 0)\\ &\vdots\\ \boldsymbol{e_{n-1}} &= (0, 0, \ldots, 1)\end{aligned} \tag{9}$$

is the set of vectors $\boldsymbol{e_i}$ that have a non-zero component only at position i. This set of vectors is linearly independent. Any vector of the vector space can be described as a function of this set:

$$(a_0, a_1, \ldots, a_{n-1}) = a_0 \bullet \boldsymbol{e_0} + a_1 \bullet \boldsymbol{e_1} + \cdots + a_{n-1} \bullet \boldsymbol{e_{n-1}} \tag{10}$$

This set of linearly independent vectors $\left\{\boldsymbol{e_0}, \boldsymbol{e_1}, \ldots, \boldsymbol{e_{n-1}}\right\}$ generates the vector space V_n, whose dimension is n. If $k < n$, the set of linearly independent vectors $\{\boldsymbol{v_1}, \boldsymbol{v_2}, \ldots, \boldsymbol{v_k}\}$ generates the vector space S of V_n through all their possible linear combinations:

$$\boldsymbol{c} = m_1 \bullet \boldsymbol{v_1} \oplus m_2 \bullet \boldsymbol{v_2} \oplus \cdots \oplus m_k \bullet \boldsymbol{v_k} \tag{11}$$

The subspace formed is of dimension k and it consists of 2^k vectors. The number of combinations is 2^k because the coefficients $m_i \in$ GF(2) adopt only one of the two possible values 0 or 1.

Another interesting operation to be considered is the inner product between two vectors. Given the vectors $\boldsymbol{u} = (u_1, u_2, \ldots, u_{n-1})$ and $\boldsymbol{v} = (v_1, v_2, \ldots, v_{n-1})$, the inner product is defined as

$$\boldsymbol{u} \circ \boldsymbol{v} = u_0 \bullet v_0 \oplus u_1 \bullet v_1 \oplus \cdots \oplus u_{n-1} \bullet v_{n-1} \tag{12}$$

where additions and multiplications are done modulo 2.

This product obeys the commutative, associative and distributive laws. If $\boldsymbol{u} \circ \boldsymbol{v} = 0$ then it is said that vectors $\boldsymbol{u} = (u_1, u_2, \ldots, u_{n-1})$ and $\boldsymbol{v} = (v_1, v_2, \ldots, v_{n-1})$ are orthogonal.

2.4.2 Dual Subspace

If S is a k-dimensional subspace of the n-dimensional vector space V_n, the set S_d of vectors $\boldsymbol{v}$ for which for any $\boldsymbol{u} \in S$ and $\boldsymbol{v} \in S_\mathrm{d}$, $\boldsymbol{u} \circ \boldsymbol{v} = 0$ is called the dual subspace of S. It is possible to demonstrate that this set is also a subspace of V_n. Moreover, it can also be demonstrated that if the subspace S is of dimension k, the dual subspace S_d is of dimension $(n - k)$. In other words,

$$\dim(S) + \dim(S_\mathrm{d}) = n \tag{13}$$

Example 2.5: For the vector space V_4 over GF(2), the following set of vectors $S = \{(0000), (0011), (0110), (0100), (0101), (0111), (0010), (0001)\}$ is a three-dimensional subspace of V_4 for which the one-dimensional subspace $S_\mathrm{d} = \{(0000), (1000)\}$ is the dual subspace S_d of S.

2.4.3 Matrix Form

The linearly independent vectors that generate a given space or subspace can be organized as row vectors of a matrix. Such a matrix of size $k \times n$ is defined over GF(2), and is a rectangular array of k rows and n columns:

$$\boldsymbol{G} = \begin{bmatrix} g_{00} & g_{01} & \cdots & g_{0,n-1} \\ g_{10} & g_{11} & \cdots & g_{1,n-1} \\ \vdots & \vdots & & \vdots \\ g_{k-1,0} & g_{k-1,1} & \cdots & g_{k-1,n-1} \end{bmatrix} \tag{14}$$

Each entry of this matrix belongs to the binary field GF(2), $g_{ij} \in \mathrm{GF}(2)$. Each of its rows can be understood as a vector of dimension $1 \times n$. If the k rows of this matrix are linearly independent, they can be considered as a basis that generates 2^k possible linear combinations that, as a set, becomes a k-dimensional vector subspace of the vector space V_n. This subspace is called the row space of $\boldsymbol{G}$. It is possible to perform linear operations and permutations between rows, or sum of rows, over the matrix $\boldsymbol{G}$ but the subspace generated by the modified matrix ***G'*** is the same as that generated by the matrix $\boldsymbol{G}$.

Example 2.6: In the following matrix $\boldsymbol{G}$, the third row is replaced by the addition of the second and third rows, and the first and second rows are permuted, generating the matrix $\boldsymbol{G}'$:

$$\boldsymbol{G} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{bmatrix} \qquad \boldsymbol{G}' = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Both matrices generate the same subspace, which is the following three-dimensional subspace of the vector space V_5:

$$\begin{aligned}
0 \bullet (10110) \oplus 0 \bullet (01001) \oplus 0 \bullet (11011) &= (00000)\\
0 \bullet (10110) \oplus 0 \bullet (01001) \oplus 1 \bullet (11011) &= (11011)\\
0 \bullet (10110) \oplus 1 \bullet (01001) \oplus 0 \bullet (11011) &= (01001)\\
1 \bullet (10110) \oplus 0 \bullet (01001) \oplus 0 \bullet (11011) &= (10110)\\
0 \bullet (10110) \oplus 1 \bullet (01001) \oplus 1 \bullet (11011) &= (10010)\\
1 \bullet (10110) \oplus 1 \bullet (01001) \oplus 0 \bullet (11011) &= (11111)\\
1 \bullet (10110) \oplus 0 \bullet (01001) \oplus 1 \bullet (11011) &= (01101)\\
1 \bullet (10110) \oplus 1 \bullet (01001) \oplus 1 \bullet (11011) &= (00100)
\end{aligned}$$

2.4.4 Dual Subspace Matrix

For the vector subspace S, which is the row space of the matrix $\boldsymbol{G}$ that has k linearly independent rows, if S_{d} is the dual subspace, the dimension of this dual subspace is $n - k$. Matrix $\boldsymbol{G}$ can be described more compactly if its rows are denoted as row vectors of dimension $1 \times n$:

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{g}_0 \\ \boldsymbol{g}_1 \\ \vdots \\ \boldsymbol{g}_{k-1} \end{bmatrix} \tag{15}$$

Let $\boldsymbol{h}_0, \boldsymbol{h}_1, \ldots, \boldsymbol{h}_{n-k-1}$ be the linearly independent row vectors of a matrix for which S_{d} is the row subspace. These vectors generate S_{d}. Therefore, a matrix $\boldsymbol{H}$ of dimension $(n - k) \times n$ can be constructed using the vectors $\boldsymbol{h}_0, \boldsymbol{h}_1, \ldots, \boldsymbol{h}_{n-k-1}$ as row vectors:

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{h}_0 \\ \boldsymbol{h}_1 \\ \vdots \\ \boldsymbol{h}_{n-k-1} \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & \cdots & h_{0,n-1} \\ h_{10} & h_{11} & \cdots & h_{1,n-1} \\ \vdots & \vdots & & \vdots \\ h_{n-k-1,0} & h_{n-k-1,1} & \cdots & h_{n-k-1,n-1} \end{bmatrix} \tag{16}$$

The row space of $\boldsymbol{H}$ is S_{d}, the dual subspace of S, which in turn is the row space of $\boldsymbol{G}$. Since each row vector $\boldsymbol{g}_i$ of $\boldsymbol{G}$ is a vector in S, and each row vector $\boldsymbol{h}_j$ of $\boldsymbol{H}$ is a vector in S_{d}, the inner product between them is zero, $\boldsymbol{g}_i \circ \boldsymbol{h}_j = 0$. The row space of $\boldsymbol{G}$ is the dual space of the row space of $\boldsymbol{H}$. Thus for each matrix $\boldsymbol{G}$ of dimension $k \times n$ with k linearly independent vectors, there exists a matrix $\boldsymbol{H}$ of dimension $(n - k) \times n$ with $n - k$ linearly independent vectors, so that for each row vector $\boldsymbol{g}_i$ of $\boldsymbol{G}$ and each row vector $\boldsymbol{h}_j$ of $\boldsymbol{H}$ it is true that $\boldsymbol{g}_i \circ \boldsymbol{h}_j = 0$ [4].

Example 2.7: Given the matrix

$$\boldsymbol{G} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{bmatrix}$$

its rows generate a subspace consisting of the following vectors:

$$S = \{(00000), (11011), (10110), (01001), (10010), (11111), (01101), (00100)\}$$

The matrix

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

has row vectors that generate the row space S_d consisting of the following vectors:

$$S_\mathrm{d} = \{(00000), (01001), (10010), (11011)\}$$

which is the dual space of S. It is indeed verified that $\boldsymbol{g}_i \circ \boldsymbol{h}_j = 0$:

$$\begin{aligned} (10110) \circ (01001) &= 0 \\ (10110) \circ (10010) &= 0 \\ (01001) \circ (01001) &= 0 \\ (01001) \circ (10010) &= 0 \\ (11011) \circ (01001) &= 0 \\ (11011) \circ (10010) &= 0 \end{aligned}$$

2.5 Linear Block Codes

The above considerations regarding vector space theory will be useful for the description of a block code. Message information to be encoded is grouped into a k-bit block constituting a generic message $\boldsymbol{m} = (m_0, m_1, \ldots, m_{k-1})$ that is one of 2^k possible messages. The encoder takes this message and generates a codeword or code vector $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1})$ of n components, where normally $n > k$; that is, redundancy is added. This procedure is basically a bijective assignment between the 2^k vectors of the message vector space and 2^k of the 2^n possible vectors of the encoded vector space.

When k and n are small numbers, this assignment can be done by means of a table, but when these numbers are large, there is a need to find a generating mechanism for the encoding process. Given this need, linearity of the operations in this mechanism greatly simplifies the encoding procedure.

Definition 2.1: A block code of length n and 2^k message words is said to be a linear block code $C_\mathrm{b}(n, k)$ if the 2^k codewords form a vector subspace, of dimension k, of the vector space V_n of all the vectors of length n with components in the field GF(2) [3, 4, 6].

Encoding basically means to take the 2^k binary message words of k bits each, and assign to them some of the 2^n vectors of n bits. This is a bijective function. Since usually $k < n$, there are more vectors of n bits than those of k bits, and so the selection of the vectors of n bits has to be done using the lowest level of redundancy while maximizing the distance among the codewords.

The set of 2^k codewords constitute a vector subspace of the set of words of n bits. As a consequence of its definition, a linear block code is characterized by the fact that the sum of any of two codewords is also a codeword.

2.5.1 Generator Matrix $\boldsymbol{G}$

Since a linear block code $C_b(n, k)$ is a vector subspace of the vector space V_n, there will be k linearly independent vectors that in turn are codewords $\boldsymbol{g_0}, \boldsymbol{g_1}, \ldots, \boldsymbol{g_{k-1}}$, such that each possible codeword is a linear combination of them:

$$\boldsymbol{c} = m_0 \bullet \boldsymbol{g_0} \oplus m_1 \bullet \boldsymbol{g_1} \oplus \cdots \oplus m_{k-1} \bullet \boldsymbol{g_{k-1}} \tag{17}$$

These linearly independent vectors can be arranged in a matrix called the generator matrix $\boldsymbol{G}$:

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{g_0} \\ \boldsymbol{g_1} \\ \vdots \\ \boldsymbol{g_{k-1}} \end{bmatrix} = \begin{bmatrix} g_{00} & g_{01} & \cdots & g_{0,n-1} \\ g_{10} & g_{11} & \cdots & g_{1,n-1} \\ \vdots & \vdots & & \vdots \\ g_{k-1,0} & g_{k-1,1} & \cdots & g_{k-1,n-1} \end{bmatrix} \tag{18}$$

This is a matrix mechanism for generating any codeword. For a given message vector $\boldsymbol{m} = (m_0, m_1, \ldots, m_{k-1})$, the corresponding codeword is obtained by matrix multiplication:

$$\begin{aligned} \boldsymbol{c} = \boldsymbol{m} \circ \boldsymbol{G} &= (m_0, m_1, \ldots, m_{k-1}) \circ \begin{bmatrix} g_{00} & g_{01} & \cdots & g_{0,n-1} \\ g_{10} & g_{11} & \cdots & g_{1,n-1} \\ \vdots & \vdots & & \vdots \\ g_{k-1,0} & g_{k-1,1} & \cdots & g_{k-1,n-1} \end{bmatrix} \\ &= (m_0, m_1, \ldots, m_{k-1}) \circ \begin{bmatrix} \boldsymbol{g_0} \\ \boldsymbol{g_1} \\ \vdots \\ \boldsymbol{g_{k-1}} \end{bmatrix} = m_0 \bullet \boldsymbol{g_0} \oplus m_1 \bullet \boldsymbol{g_1} \oplus \cdots \oplus m_{k-1} \bullet \boldsymbol{g_{k-1}} \end{aligned} \tag{19}$$

Note that the symbol '∘' represents the inner product between vectors or matrices, whereas the symbol '•' represents the multiplication by a scalar in the field GF(2) of a vector of the vector space or subspace used.

The rows of the generator matrix $\boldsymbol{G}$ generate the linear block code $C_b(n, k)$, or, equivalently, the k linearly independent rows of $\boldsymbol{G}$ completely define the code.

Table 2.1 Codewords of a linear block code $C_b(7, 4)$

Message				Codewords						
0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	1	0	0	0	1
0	0	1	0	1	1	1	0	0	1	0
0	0	1	1	0	1	0	0	0	1	1
0	1	0	0	0	1	1	0	1	0	0
0	1	0	1	1	1	0	0	1	0	1
0	1	1	0	1	0	0	0	1	1	0
0	1	1	1	0	0	1	0	1	1	1
1	0	0	0	1	1	0	1	0	0	0
1	0	0	1	0	1	1	1	0	0	1
1	0	1	0	0	0	1	1	0	1	0
1	0	1	1	1	0	0	1	0	1	1
1	1	0	0	1	0	1	1	1	0	0
1	1	0	1	0	0	0	1	1	0	1
1	1	1	0	0	1	0	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1

Example 2.8: Consider the following generator matrix of size 4 × 7 and obtain the codeword corresponding to the vector message $\boldsymbol{m} = (1001)$:

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{g}_0 \\ \boldsymbol{g}_1 \\ \boldsymbol{g}_2 \\ \boldsymbol{g}_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The corresponding codeword is

$$\boldsymbol{c} = \boldsymbol{m} \circ \boldsymbol{G} = 1 \bullet \boldsymbol{g}_0 \oplus 0 \bullet \boldsymbol{g}_1 \oplus 0 \bullet \boldsymbol{g}_2 \oplus 1 \bullet \boldsymbol{g}_3 = (1101000) \oplus (1010001) = (0111001)$$

Table 2.1 shows the code generated by the generator matrix $\boldsymbol{G}$ of this example.

2.5.2 Block Codes in Systematic Form

In Table 2.1 it can be seen that the last four bits of each codeword are the same as the message bits; that is, the message appears as it is, inside the codeword. In this case, the first three bits are the so-called parity check or redundancy bits. This particular form of the codeword is called systematic form. In this form, the codewords consist of the $(n - k)$ parity check bits followed by the k bits of the message. The structure of a codeword in systematic form is shown in Figure 2.1.

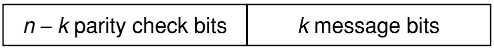


Figure 2.1 Systematic form of a codeword of a block code

In the convention selected in this book, the message bits are placed at the end of the codeword, while the redundancy bits are placed at the beginning of the codeword, but this can be done the other way round. However, the choice of convention does not modify the properties of a given block code, although some mathematic expressions related to the code will of course adopt a different form in each case. In the current bibliography on error-correcting codes, the systematic form can be found adopting both of the two conventions described above.

A systematic linear block code $C_{\mathrm{b}}(n, k)$ is uniquely specified by a generator matrix of the form

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{g_0} \\ \boldsymbol{g_1} \\ \vdots \\ \boldsymbol{g_{k-1}} \end{bmatrix} = \left[\underbrace{\begin{matrix} p_{00} & p_{01} & \cdots & p_{0,n-k-1} \\ p_{10} & p_{11} & \cdots & p_{1,n-k-1} \\ \vdots & \vdots & & \vdots \\ p_{k-1,0} & p_{k-1,1} & \cdots & p_{k-1,n-k-1} \end{matrix}}_{\text{submatrix } \boldsymbol{P} \text{ k} \times \text{(n} - \text{k)}} \quad \underbrace{\begin{matrix} 1 & 0 & 0 & \cdots \\ 0 & 1 & 0 & \cdots \\ \vdots & \vdots & \vdots & \\ 0 & 0 & 0 & \cdots \end{matrix}}_{\text{submatrix } \boldsymbol{I} \text{ k} \times \text{k}} \begin{matrix} 0 \\ 0 \\ \vdots \\ 1 \end{matrix} \right] \tag{20}$$

which, in a compact notation, is

$$\boldsymbol{G} = [\boldsymbol{P} \quad \boldsymbol{I_k}] \tag{21}$$

In systematic block coding, it is possible to establish an analytical expression between the parity check and the message bits. If $\boldsymbol{m} = (m_0, m_1, \ldots, m_{k-1})$ is the message vector and $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1})$ the coded vector, the parity check bits can be obtained as a function of the message bits using the following expression:

$$\begin{aligned} &c_{n-k+i} = m_i \\ &c_j = m_0 \bullet p_{0j} + m_1 \bullet p_{1j} + \cdots + m_{k-1} \bullet p_{k-1,j} \quad 0 \leq j < n - k \end{aligned} \tag{22}$$

These $n - k$ equations are called the parity check equations.

Example 2.9: Consider the generator matrix of the linear block code $C_{\mathrm{b}}(7, 4)$, presented in the previous example, and state the parity check equations for the following case:

$$\boldsymbol{c} = \boldsymbol{m} \circ \boldsymbol{G} = (m_0, m_1, m_2, m_3) \circ \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Then the parity check equations adopt the form

$$\begin{aligned} c_0 &= m_0 \oplus m_2 \oplus m_3 \\ c_1 &= m_0 \oplus m_1 \oplus m_2 \\ c_2 &= m_1 \oplus m_2 \oplus m_3 \\ c_3 &= m_0 \\ c_4 &= m_1 \\ c_5 &= m_2 \\ c_6 &= m_3 \end{aligned}$$

2.5.3 Parity Check Matrix $\boldsymbol{H}$

As explained in previous sections, the generator matrix $\boldsymbol{G}$ contains k linearly independent vectors that generate the vector subspace S of the vector space V_n, which is in turn associated with a dual vector subspace S_d of the same vector space V_n that is generated by the rows of a matrix $\boldsymbol{H}$. Each vector of the row space of the matrix $\boldsymbol{G}$ is orthogonal to the rows of the matrix $\boldsymbol{H}$ and vice versa.

The 2^{n-k} linear combinations of the matrix $\boldsymbol{H}$ generate the dual code $C_\mathrm{bd}(n, n-k)$, which is the dual subspace of the code C_b generated by the matrix $\boldsymbol{G}$. The systematic form of the parity check matrix $\boldsymbol{H}$ of the code C_b generated by the generator matrix $\boldsymbol{G}$ is

$$\boldsymbol{H} = \left[\underbrace{\begin{matrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \end{matrix}}_{\text{submatrix } \boldsymbol{I}\,(\mathrm{n}-\mathrm{k}) \times (\mathrm{n}-\mathrm{k})} \quad \underbrace{\begin{matrix} p_{00} & p_{10} & \cdots & p_{k-1,0} \\ p_{01} & p_{11} & \cdots & p_{k-1,1} \\ \vdots & \vdots & & \vdots \\ p_{0,n-k-1} & p_{1,n-k-1} & \cdots & p_{k-1,n-k-1} \end{matrix}}_{\text{submatrix } \boldsymbol{P}^T\,(\mathrm{n}-\mathrm{k}) \times \mathrm{k}}\right] = \left[\boldsymbol{I}_{n-k} \quad \boldsymbol{P}^{\boldsymbol{T}}\right] \tag{23}$$

where $\boldsymbol{P}^{\boldsymbol{T}}$ is the transpose of the parity check submatrix $\boldsymbol{P}$. The matrix $\boldsymbol{H}$ is constructed so that the inner product between any row vector $\boldsymbol{g}_i$ of $\boldsymbol{G}$ and any row vector $\boldsymbol{h}_j$ of $\boldsymbol{H}$ is zero:

$$\boldsymbol{g}_i \circ \boldsymbol{h}_j = p_{ij} \oplus p_{ij} = 0 \tag{24}$$

This condition can be summarized in the matrix expression

$$\boldsymbol{G} \circ \boldsymbol{H}^{\boldsymbol{T}} = \boldsymbol{0} \tag{25}$$

It can also be verified that the parity check equations can be obtained from the parity check matrix $\boldsymbol{H}$, so that this matrix also specifies completely a given block code. Since a codeword in systematic form is expressed as

$$\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-k-1}, m_0, m_1, \ldots, m_{k-1}) \tag{26}$$

then since

$$\boldsymbol{c} \circ \boldsymbol{H}^{\boldsymbol{T}} = \boldsymbol{m} \circ \boldsymbol{G} \circ \boldsymbol{H}^{T} = \boldsymbol{0} \tag{27}$$

for the row j of $\boldsymbol{H}$,

$$c_j \oplus p_{0j} \bullet m_0 \oplus p_{1j} \bullet m_1 \oplus \cdots \oplus p_{k-1,j} \bullet m_{k-1} = 0 \tag{28}$$

or, equivalently,

$$c_j = p_{0j} \bullet m_0 \oplus p_{1j} \bullet m_1 \oplus \cdots \oplus p_{k-1,j} \bullet m_{k-1} \quad 0 \le j < n-k \tag{29}$$

Example 2.10: Determine the parity check matrix $\boldsymbol{H}$ for the linear block code $C_\mathrm{b}(7, 4)$ generated by the generator matrix

$$\boldsymbol{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Since

$$\boldsymbol{G} = \left[\begin{array}{ccc|cccc} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right]$$

$$\underbrace{\qquad\qquad}_{\text{submatrix } \boldsymbol{P}} \qquad \underbrace{\qquad\qquad}_{\text{submatrix } \boldsymbol{I}}$$

the parity check matrix $\boldsymbol{H}$ is constructed using these submatrices:

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

A practical implementation of these codes could be done using combinational logic for the parity check equations.

2.6 Syndrome Error Detection

So far the definitions for the generator and the parity check matrices of a given block code have been presented. The codeword $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1})$ is such that its components are taken from the binary field GF(2), $c_i \in$ GF(2). As a consequence of its transmission through a noisy channel, this codeword could be received containing some possible errors. The received vector can therefore be different from the corresponding transmitted codeword, and it will be denoted as $\boldsymbol{r} = (r_0, r_1, \ldots, r_{n-1})$, where it is also true that $r_i \in$ GF(2).

An error event can be modelled as an error vector or error pattern $\boldsymbol{e} = (e_0, e_1, \ldots, e_{n-1})$, whose components are also defined over the binary field, $e_i \in$ GF(2), and which is related to the codeword and received vectors as follows:

$$\boldsymbol{e} = \boldsymbol{r} \oplus \boldsymbol{c} \tag{30}$$

The error vector has non-zero components in the positions where an error has occurred. Once the error vector is determined, a task to be performed by the decoder, it is possible to do a correction of the received vector in order to determine an estimate of the valid codeword, and this can be done by using the expression

$$\boldsymbol{c} = \boldsymbol{r} \oplus \boldsymbol{e} \tag{31}$$

Since any codeword should obey the condition

$$\boldsymbol{c} \circ \boldsymbol{H}^T = \boldsymbol{0}$$

an error-detection mechanism can be implemented based on the above expression, which adopts the following form:

$$\boldsymbol{S} = \boldsymbol{r} \circ \boldsymbol{H}^T = (s_0, s_1, \ldots, s_{n-k-1}) \tag{32}$$

This vector is called the syndrome vector. The detection operation is performed over the received vector, so that if this operation results in the all-zero vector, then the received vector is considered to be a valid codeword; if otherwise, the decoder has detected errors.

Since $\boldsymbol{r} = \boldsymbol{c} \oplus \boldsymbol{e}$,

$$\boldsymbol{S} = \boldsymbol{r} \circ \boldsymbol{H}^T = (\boldsymbol{c} \oplus \boldsymbol{e}) \circ \boldsymbol{H}^T = \boldsymbol{c} \circ \boldsymbol{H}^T \oplus \boldsymbol{e} \circ \boldsymbol{H}^T = \boldsymbol{e} \circ \boldsymbol{H}^T \tag{33}$$

If the error pattern is the all-zero vector, then the syndrome vector will also be an all-zero vector, and thus the received vector is a valid codeword. When the syndrome vector contains at least one non-zero component, it will be detecting the presence of errors in the received vector. There is however a possibility that the syndrome vector can be the all-zero vector in spite of the presence of errors in the received vector. This is in fact possible if the error pattern is equal to a codeword; that is, if the number and positions of the errors are such that the transmitted codeword is converted into another codeword. This error pattern will not be detected by the syndrome operation. This is what is called an undetected error pattern, and as such is not within the error-correction capability of the code.

As said above, the undetected error patterns are characterized by satisfying the condition $\boldsymbol{S} = \boldsymbol{e} \circ \boldsymbol{H}^T = \mathbf{0}$; that is, these are the error patterns that are equal to one of the codewords ($\boldsymbol{e} = \boldsymbol{c}$). There will be therefore $2^k - 1$ undetectable non-zero error patterns.

According to the expression for calculating the syndrome vector, each of its bits can be evaluated as follows:

$$\begin{aligned}
s_0 &= r_0 \oplus r_{n-k} \bullet p_{00} \oplus r_{n-k+1} \bullet p_{10} \oplus \cdots \oplus r_{n-1} \bullet p_{k-1,0} \\
s_1 &= r_1 \oplus r_{n-k} \bullet p_{01} \oplus r_{n-k+1} \bullet p_{11} \oplus \cdots \oplus r_{n-1} \bullet p_{k-1,1} \\
&\vdots \\
s_{n-k-1} &= r_{n-k-1} \oplus r_{n-k} \bullet p_{0,n-k-1} \oplus r_{n-k+1} \bullet p_{1,n-k-1} \oplus \cdots \oplus r_{n-1} \bullet p_{k-1,n-k-1}
\end{aligned} \tag{34}$$

The dimension of the syndrome vector is 1 × $(n - k)$.

Example 2.11: For the same linear block code $C_b(7, 4)$, as seen in previous examples, obtain the analytical expressions for the syndrome vector's bits.

If $\boldsymbol{r} = (r_0, r_1, r_2, r_3, r_4, r_5, r_6)$
then

$$\boldsymbol{S} = (s_0, s_1, s_2) = (r_0, r_1, r_2, r_3, r_4, r_5, r_6) \circ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

$$s_0 = r_0 \oplus r_3 \oplus r_5 \oplus r_6$$

$$s_1 = r_1 \oplus r_3 \oplus r_4 \oplus r_5$$

$$s_2 = r_2 \oplus r_4 \oplus r_5 \oplus r_6$$

The syndrome vector does not depend on the received vector, but on the error vector. Thus, solving the following system of equations

$$\begin{aligned}
s_0 &= e_0 \oplus e_{n-k} \bullet p_{00} \oplus e_{n-k+1} \bullet p_{10} \oplus \cdots \oplus e_{n-1} \bullet p_{k-1,0} \\
s_1 &= e_1 \oplus e_{n-k} \bullet p_{01} \oplus e_{n-k+1} \bullet p_{11} \oplus \cdots \oplus e_{n-1} \bullet p_{k-1,1} \\
&\vdots \\
s_{n-k-1} &= e_{n-k-1} \oplus e_{n-k} \bullet p_{0,n-k-1} \oplus e_{n-k+1} \bullet p_{1,n-k-1} \oplus \cdots \oplus e_{n-1} \bullet p_{k-1,n-k-1}
\end{aligned} \tag{35}$$

will allow us to evaluate the error vector, which in turn will allow us to do an estimation of a valid codeword. However, this set of $(n-k)$ equations does not have a unique solution, but exhibits 2^k solutions. This is due to the fact that there are 2^k error patterns that produce the same syndrome vector. In spite of this, and because the noise power normally acts with minimum effect, the error pattern with the least number of errors will be considered to be the true solution of this system of equations.

Example 2.12: For the linear block code $C_{\mathrm{b}}(7,4)$ of the previous example, a transmitted codeword $\boldsymbol{c} = (0011010)$ is affected by the channel noise and received as the vector $\boldsymbol{r} = (0001010)$. The calculation of the syndrome vector results in the vector $\boldsymbol{S} = (001)$, which in terms of the system of equations (35) becomes

$$\begin{aligned} 0 &= e_0 \oplus e_3 \oplus e_5 \oplus e_6 \\ 0 &= e_1 \oplus e_3 \oplus e_4 \oplus e_5 \\ 1 &= e_2 \oplus e_4 \oplus e_5 \oplus e_6 \end{aligned}$$

There are $2^4 = 16$ different error patterns that satisfy the above equations (see Table 2.2).

Since errors in a codeword of n bits are governed by the binomial distribution, the error pattern with i errors is more likely than the error pattern of $i+1$ errors, which means that for channels like the BSC, the error pattern with the smallest number of non-zero components will be considered as the true error pattern. In this case the error pattern $\boldsymbol{e} = (0010000)$ is considered, among the 16 possibilities, to be the true error pattern, and so

$$\boldsymbol{c} = \boldsymbol{r} \oplus \boldsymbol{e} = (0011010) = (0001010) \oplus (0010000)$$

Table 2.2 Error patterns that satisfy the equations of Example 2.12

e_0	e_1	e_2	e_3	e_4	e_5	e_6
0	0	1	0	0	0	0
1	1	1	1	0	0	0
1	0	0	0	0	0	1
0	1	0	1	0	0	1
0	0	0	1	0	1	0
1	1	0	0	0	1	0
0	1	1	0	0	1	1
1	0	1	1	0	1	1
0	1	0	0	1	0	0
1	0	0	1	1	0	0
1	1	1	0	1	0	1
0	0	1	1	1	0	1
1	0	1	0	1	1	0
0	1	1	1	1	1	0
1	1	0	1	1	1	1
0	0	0	0	1	1	1

2.7 Minimum Distance of a Block Code

The minimum distance $d_{\min}$ is an important parameter of a code, especially for a block code. Before defining this parameter, other useful definitions related to the minimum distance are first provided [3, 4].

Definition 2.2: The number of non-zero components $c_i \neq 0$ of a given vector $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1})$ of size $(1 \times n)$ is called the weight, or Hamming weight, $w(\boldsymbol{c})$, of that vector. In the case of a vector defined over the binary field GF(2), the weight is the number of '1's in the vector.

Definition 2.3: The Hamming distance between any two vectors $\boldsymbol{c_1} = \left(c_{01}, c_{11}, \ldots, c_{n-1,1}\right)$ and $\boldsymbol{c_2} = \left(c_{02}, c_{12}, \ldots, c_{n-1,2}\right)$, $d(\boldsymbol{c_1}, \boldsymbol{c_2})$, is the number of component positions in which the two vectors differ.

For instance, if $\boldsymbol{c_1} = (0011010)$ and $\boldsymbol{c_2} = (1011100)$, then $d(\boldsymbol{c_1}, \boldsymbol{c_2}) = 3$.

According to the above definitions, it can be verified that

$$d(\boldsymbol{c_i}, \boldsymbol{c_j}) = w(\boldsymbol{c_i} \oplus \boldsymbol{c_j}) \tag{36}$$

For a given code, the minimum value of the distance between all possible pairs of codewords can be calculated. This minimum value of the distance evaluated over all the codewords of the code is called the minimum distance of the code, $d_{\min}$:

$$d_{\min} = \min\left\{d(\boldsymbol{c_i}, \boldsymbol{c_j}); \boldsymbol{c_i}, \boldsymbol{c_j} \in C_{\mathrm{b}}; \boldsymbol{c_i} \neq \boldsymbol{c_j}\right\} \tag{37}$$

Since, in general, block codes are designed to be linear, the addition of any two code vectors is another code vector. From this point of view, any codeword can be seen as the addition of at least two other codewords. Since the Hamming distance is the number of positions in which two vectors differ, and on the other hand the weight of the sum of two vectors is the Hamming distance between these two vectors, then the weight of a codeword is at the same time the distance between two other vectors of that code. Thus, the minimum value of the weight evaluated over all the codewords of a code, excepting the all-zero vector, is the minimum distance of the code:

$$d_{\min} = \min\left\{w(\boldsymbol{c_i} \oplus \boldsymbol{c_j}); \boldsymbol{c_i}, \boldsymbol{c_j} \in C_{\mathrm{b}}; \boldsymbol{c_i} \neq \boldsymbol{c_j}\right\} = \min\left\{w(\boldsymbol{c_m}); \boldsymbol{c_m} \in C_{\mathrm{b}}; c_m \neq 0\right\} \tag{38}$$

Therefore, the minimum distance of a linear block code $C_{\mathrm{b}}(n, k)$ is the minimum value of the weight of the non-zero codewords of that code.

As an example, the linear block code analysed in previous examples has minimum distance $d_{\min} = 3$, because this is the minimum value of the weight evaluated over all the non-zero codewords of this code (see Table 2.1).

*2.7.1 Minimum Distance and the Structure of the **H** Matrix*

There is an interesting relationship between the minimum distance $d_{\min}$ of a code and its parity check matrix $\boldsymbol{H}$.

Theorem 2.2: Consider a linear block code $C_{\mathrm{b}}(n, k)$ completely determined by its parity check matrix $\boldsymbol{H}$. For each codeword of Hamming weight p_{H}, there exist p_{H} columns of the parity check matrix $\boldsymbol{H}$ that when added together result in the all-zero vector. In the same way, it can be said that if the parity check matrix $\boldsymbol{H}$ contains p_{H} columns that when added give the all-zero vector, then there is in the code a vector of weight p_{H} [4].

In order to see this, parity check matrix $\boldsymbol{H}$ is described in the following form:

$$\boldsymbol{H} = \left[\boldsymbol{h_0}, \boldsymbol{h_1}, \ldots, \boldsymbol{h_{n-1}}\right] \tag{39}$$

where $\boldsymbol{h_i}$ is the ith column of this matrix. If a codeword $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1})$ has a weight p_{H}, then there exist p_{H} non-zero components in that vector $c_{i1} = c_{i2} = \cdots = c_{ip_{\mathrm{H}}} = 1$, for which $0 \leq i_1 < i_2 < \cdots < i_{p_{\mathrm{H}}} \leq n - 1$.

Since

$$\boldsymbol{c} \circ \boldsymbol{H}^T = \boldsymbol{0}$$

then

$$\begin{aligned} &c_0 \bullet \boldsymbol{h_0} \oplus c_1 \bullet \boldsymbol{h_1} \oplus \cdots \oplus c_{n-1} \bullet \boldsymbol{h_{n-1}} \\ &\quad = c_{i1} \bullet \boldsymbol{h_{i1}} \oplus c_{i2} \bullet \boldsymbol{h_{i2}} \oplus \cdots \oplus c_{ip_{\mathrm{H}}} \bullet \boldsymbol{h_{ip_{\mathrm{H}}}} \\ &\quad = \boldsymbol{h_{i1}} \oplus \boldsymbol{h_{i2}} \oplus \cdots \oplus \boldsymbol{h_{ip_{\mathrm{H}}}} = \boldsymbol{0} \end{aligned} \tag{40}$$

Similarly, the second part of the theorem can be demonstrated.

The following corollary is then derived:

Corollary 2.7.1: For a linear block code $C_{\mathrm{b}}(n, k)$ completely determined by its parity check matrix $\boldsymbol{H}$, the minimum weight or minimum distance of this code is equal to the minimum number of columns of that matrix which when added together result in the all-zero vector $\boldsymbol{0}$.

Example 2.13: For the linear block code $C_{\mathrm{b}}(7, 4)$, as seen in previous examples, whose parity check matrix is of the form

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

determine the minimum distance of this code.

It can be seen that the addition of the first, third and seventh column results in the all-zero vector $\boldsymbol{0}$. Hence, and because the same result cannot be obtained by the addition of only two columns, the minimum distance of this code is $d_{\min} = 3$.

2.8 Error-Correction Capability of a Block Code

The minimum distance of a code is the minimum number of components changed by the effect of the noise that converts a code vector into another vector of the same code. If having transmitted the codeword $\boldsymbol{c}$ the noise transforms this vector in the received vector $\boldsymbol{r}$, the distance between $\boldsymbol{c}$ and $\boldsymbol{r}$ is the weight of the error pattern $d(\boldsymbol{c}, \boldsymbol{r}) = w(\boldsymbol{e}) = l$, that is, the number of positions that change their value in the original vector $\boldsymbol{c}$ due to the effects of noise. If the noise modifies $d_{\min}$ positions, then it is possible in the worst case that a code vector is transformed into another vector of the same code, so that the error event is undetectable. If the number of positions the noise alters is $d_{\min} - 1$, it is guaranteed that the codeword cannot be converted into another codeword. Thus, the error-detection capability of a linear block code $C_{\mathrm{b}}(n, k)$ of minimum distance $d_{\min}$ is $d_{\min} - 1$. This is evaluated for the worst case, that is, for the case in which the error event of $d_{\min}$ bits happens over a codeword that has a Hamming weight $d_{\min}$. However, there could be other detectable error patterns of the weight $d_{\min}$. Based on this analysis, the error-detection capability of a code can be measured by means of the probability that the code fails to determine an estimate of the codeword from the received vector, which is evaluated using the weight distribution function of the code. Since a detection failure happens when the error pattern is equal to a non-zero codeword,

$$P_{\mathrm{U}}(E) = \sum_{i=1}^{n} A_i p^i (1-p)^{n-i} \tag{41}$$

where A_i is the number of codewords of weight i and p is the error probability for the BSC, for which this analysis is valid. When the minimum distance is $d_{\min}$, the values of A_1 to $A_{d_{\min}-1}$ are all zero.

Example 2.14: For the linear block code $C_{\mathrm{b}}(7, 4)$, previously analysed values of the weight distribution function are equal to

$$A_0 = 1, \qquad A_1 = A_2 = 0, \qquad A_3 = 7, \qquad A_4 = 7, \qquad A_5 = A_6 = 0, \qquad A_7 = 1$$

The probability of an undetected error is therefore

$$P_{\mathrm{U}}(E) = \sum_{i=1}^{n} A_i p^i (1-p)^{n-i} = 7p^3(1-p)^4 + 7p^4(1-p)^3 + p^7 \approx 7p^3$$

where the approximation is based on the error probability for the BSC being a small number $p \ll 1$.

In order to determine the error-correction capability of a linear block code $C_{\mathrm{b}}(n, k)$, an integer number t that fits the condition

$$2t + 1 \leq d_{\min} \leq 2t + 2 \tag{42}$$

will represent the number of bits that can be corrected.

If having transmitted a codeword $\boldsymbol{c}_1$ the noise effects transform this vector into the received vector $\boldsymbol{r}$, then with respect to another codeword $\boldsymbol{c}_2$ the following inequality will be true:

$$d(\boldsymbol{c}_1, \boldsymbol{r}) + d(\boldsymbol{c}_2, \boldsymbol{r}) \geq d(\boldsymbol{c}_1, \boldsymbol{c}_2) \tag{43}$$

By assuming an error pattern of t' errors, $d(\boldsymbol{c}_1, \boldsymbol{r}) = t'$. As $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ are codewords, $d(\boldsymbol{c}_1, \boldsymbol{c}_2) \geq d_{\min} \geq 2t + 1$ and

$$d(\boldsymbol{c}_2, \boldsymbol{r}) \geq 2t + 1 - t' \tag{44}$$

By adopting $t' \leq t$,

$$d(\boldsymbol{c}_2, \boldsymbol{r}) > t \tag{45}$$

This means that for an error pattern of weight t or less, the distance between any other codeword $\boldsymbol{c}_2$ and the received vector $\boldsymbol{r}$ is higher than the distance between the codeword $\boldsymbol{c}_1$ and the received vector $\boldsymbol{r}$, which is $t' \leq t$. This also means that the probability $P(\boldsymbol{r}/\boldsymbol{c}_1)$ is higher than the probability $P(\boldsymbol{r}/\boldsymbol{c}_2)$ for any other codeword $\boldsymbol{c}_2$. This is the process of maximum likelihood decoding, and in this operation the received vector $\boldsymbol{r}$ is decoded as the codeword $\boldsymbol{c}_1$. This way the code is able to successfully decode any error pattern of weight $t = \left\lfloor \frac{d_{\min}-1}{2} \right\rfloor$, where $\lfloor\rfloor$ means the largest integer number no greater than $\frac{d_{\min}-1}{2}$.

As happened in the detection of errors, for the correction of errors there are more possible correctable patterns than those determined by the number t. For a linear block code $C_\mathrm{b}(n, k)$ able to correct up to t errors, there are 2^{n-k} correctable error patterns including the error patterns of t or fewer errors.

If a linear block code $C_\mathrm{b}(n, k)$, able to correct all the error patterns of weight t or less, is used in a transmission over the BSC with error probability p, the error probability of the coded system is given by

$$P_{\mathrm{we}} = \sum_{i=t+1}^{n} \binom{n}{i} p^i (1-p)^{n-i} \tag{46}$$

In hybrid systems, where errors are in part corrected, and in part detected, these codes are utilized in such a way that error patterns of weight λ are corrected and error patterns of weight $l > \lambda$ are detected. If the error pattern is of weight λ or less, the system corrects it, and if the error pattern is of weight larger than λ, but less than $l + 1$, the system detects it. This is possible if $d_{\min} \geq l + \lambda + 1$.

If for example the minimum distance of a linear block code $C_\mathrm{b}(n, k)$ is $d_{\min} = 7$, this code can be used for correcting error patterns of weight $\lambda = 2$ or less and detecting error patterns of weight $l = 4$ or less.

2.9 Syndrome Detection and the Standard Array

A linear block code $C_\mathrm{b}(n, k)$ is constructed as a bijective assignment between the 2^k message vectors and the set $\{\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_{2^k}\}$. Each of these vectors is transmitted through the channel and converted into a received vector $\boldsymbol{r}$ that can be any vector of the 2^n vectors of the vector space V_n defined over the binary field GF(2). Any decoding technique is essentially a decision rule, based on the vector space V_n being partitioned into 2^k possible disjoint sets $D_1, D_2, \ldots, D_{2^k}$ such that the vector $\boldsymbol{c}_i$ is in the set D_i. There is a unique correspondence between the set D_i and the vector $\boldsymbol{c}_i$. If the received vector is in D_i, it will be decoded as $\boldsymbol{c}_i$.

The standard array is a method for doing this operation [4, 6]. The array is constructed in the following way:

A row containing the codewords, including and starting from the all-zero vector $(0, 0, \ldots, 0)$, is constructed. This row contains 2^k vectors taken from the whole set of 2^n possible vectors.

$$\boldsymbol{c}_1 = (0, 0, \ldots, 0) \;\; \boldsymbol{c}_2 \;\; \boldsymbol{c}_3 \;\; \ldots \;\; \boldsymbol{c}_{2^k} \tag{47}$$

Then an error pattern $\boldsymbol{e}_2$ is selected and placed below $\boldsymbol{c}_1$ (the all-zero vector), and then the sum vector $\boldsymbol{c}_i \oplus \boldsymbol{e}_2$ is placed below $\boldsymbol{c}_i$. This is done with all the error patterns taken from the vector space that have to be allocated, a total of 2^{n-k} vectors.

$$\begin{array}{lllllll} \boldsymbol{c}_1 = (0, 0, \ldots, 0) & \boldsymbol{c}_2 & \ldots & \boldsymbol{c}_i & \ldots & \boldsymbol{c}_{2^k} \\ \boldsymbol{e}_2 & \boldsymbol{c}_2 \oplus \boldsymbol{e}_2 & \ldots & \boldsymbol{c}_i \oplus \boldsymbol{e}_2 & \ldots & \boldsymbol{c}_{2^k} \oplus \boldsymbol{e}_2 \\ \vdots & & & & & \\ \boldsymbol{e}_{2^{n-k}} & \boldsymbol{c}_2 \oplus \boldsymbol{e}_{2^{n-k}} & \ldots & \boldsymbol{c}_i \oplus \boldsymbol{e}_{2^{n-k}} & \ldots & \boldsymbol{c}_{2^k} \oplus \boldsymbol{e}_{2^{n-k}} \end{array} \tag{48}$$

In this array the sum of any two vectors in the same row is a code vector. There are only $2^n/2^k$ disjoint rows in this array. These rows are the so-called cosets of the linear block code $C_\text{b}(n, k)$. The vector that starts each coset is called the leader of that coset, and it can be any vector of that row.

Example 2.15: For the linear block code $C_\text{b}(5, 3)$ generated by the matrix given below, determine the standard array.

$$\boldsymbol{G} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

There are in this case $2^k = 2^3 = 8$ columns and $2^{n-k} = 2^2 = 4$ rows in the standard array. Table 2.3 shows the standard array for the code of Example 2.15.

The standard array can also be seen as constituted of 2^k disjoint columns, and in each of these columns are 2^{n-k} vectors having as the first vector a code vector. These 2^k disjoint columns $D_1, D_2, \ldots, D_{2^k}$ can be used in a decoding procedure. If having transmitted a codeword $\boldsymbol{c}_i$, the received vector is $\boldsymbol{r}$, this received vector will be in D_i if the error pattern that occurred is its coset leader. In this case the received vector will be successfully decoded. If the error pattern is not a coset leader, a decoding error happens. Due to this, the 2^{n-k} coset leaders including the all-zero pattern are the correctable error patterns. It can be deduced that a linear block code $C_\text{b}(n, k)$ can correct 2^{n-k} error patterns.

In order to minimize the error probability, all the correctable error patterns, which are the coset leaders, will have to be the most likely patterns. In the case of a transmission over the BSC, the most likely patterns are those of the lowest possible weight. Thus, each coset leader will be of the lowest possible weight among the vectors of that row. The decoding in this case will be maximum likelihood decoding, that is, minimum distance decoding, so that the decoded code vector is at minimum distance with respect to the received vector.

Table 2.3 Standard array for the code of Example 2.15

0 0 0 0 0	1 1 0 1 1	1 0 1 1 0	0 1 0 0 1	1 0 0 1 0	1 1 1 1 1	0 1 1 0 1	0 0 1 0 0
1 0 0 0 0	0 1 0 1 1	0 0 1 1 0	1 1 0 0 1	0 0 0 1 0	0 1 1 1 1	1 1 1 0 1	1 0 1 0 0
0 0 0 1 0	1 1 0 0 1	1 0 1 0 0	0 1 0 1 1	1 0 0 0 0	1 1 1 0 1	0 1 1 1 1	0 0 1 1 0
0 0 0 0 1	1 1 0 1 0	1 0 1 1 1	0 1 0 0 0	1 0 0 1 1	1 1 1 1 0	0 1 1 0 0	0 0 1 0 1

In conclusion it can be said that a linear block code $C_{\mathrm{b}}(n, k)$ is able to detect $2^n - 2^k$ error patterns and correct 2^{n-k} error patterns.

It can also be said that

For a linear block code $C_{\mathrm{b}}(n, k)$ with minimum distance $d_{\min}$, all the vectors of weight $t = \left\lfloor \frac{d_{\min}-1}{2} \right\rfloor$ or less can be used as coset leaders. This is in agreement with the fact that not all the weight $t + 1$ error patterns can be corrected, even when some of them can be. On the other hand, all the vectors of the same coset have the same syndrome, whereas syndromes for different cosets are different.

By taking a coset leader as the vector $\boldsymbol{e}_i$, any other vector of that coset is the sum of the leader vector and the code vector $\boldsymbol{c}_i$. For this case, the syndrome is calculated as

$$(\boldsymbol{c}_i \oplus \boldsymbol{e}_i) \circ \boldsymbol{H}^T = \boldsymbol{c}_i \circ \boldsymbol{H}^T \oplus \boldsymbol{e}_i \circ \boldsymbol{H}^T = \boldsymbol{e}_i \circ \boldsymbol{H}^T \tag{49}$$

The syndrome of any vector of the coset is equal to the syndrome of the leader of that coset. Syndromes are vectors with $(n - k)$ components that have a bijective assignment with the cosets. For each correctable error pattern, there is a different syndrome vector. This allows us to implement simpler decoding by constructing a table where correctable error patterns and their corresponding syndrome vectors are arranged, so that when the decoder makes the syndrome calculation and knows the syndrome vector, it can recognize the corresponding error pattern. Thus, the decoder is able to correct the received vector by adding the error pattern to that received vector. Thus syndrome decoding consists of the following steps: With the information provided by the table $S \to e$, the syndrome vector is calculated as a function of the received vector using $\boldsymbol{S} = \boldsymbol{r} \circ \boldsymbol{H}^T$; then the decoder resorts to the table to identify which error pattern $\boldsymbol{e}_i$ corresponds to the calculated syndrome vector and finally corrects the received vector by doing $\boldsymbol{c}_i = \boldsymbol{r} \oplus \boldsymbol{e}_i$. This procedure can be used when the table $S \to e$ is of a reasonable size to be implemented in practice.

Example 2.16: For the linear block code $C_{\mathrm{b}}(7, 4)$ with parity check matrix

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

there are $2^4 = 16$ code vectors and $2^{7-4} = 8$ cosets, or correctable error patterns. The minimum distance has also been calculated and is equal to $d_{\min} = 3$ and so this code is able to correct any pattern of one error. In this case the total number of correctable error patterns is equal to the number of error patterns the code can correct, and there are seven correctable error patterns and the all-zero error pattern, as the possible patterns. The table $S \to e$ for this code is seen in Table 2.4.

As an example, assume that the transmitted code vector was $\boldsymbol{c} = (1010001)$ and after the transmission of this vector the received vector is $\boldsymbol{r} = (1010011)$, then the syndrome vector in this case is $\boldsymbol{r} \circ \boldsymbol{H}^T = (111)$ and so the leader of the corresponding coset is (0000010) and $\boldsymbol{c} = (1010001) = (1010011) \oplus (0000010)$.

Table 2.4 Error patterns and their corresponding syndrome vectors, Example 2.16

Error patterns							Syndromes		
1	0	0	0	0	0	0	1	0	0
0	1	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	1
0	0	0	1	0	0	0	1	1	0
0	0	0	0	1	0	0	0	1	1
0	0	0	0	0	1	0	1	1	1
0	0	0	0	0	0	1	1	0	1

2.10 Hamming Codes

A widely used class of linear block codes is the Hamming code family [11]. For any positive integer $m \geq 3$, there exists a Hamming code with the following characteristics:

Length	$n = 2^m - 1$
Number of message bits	$k = 2^m - m - 1$
Number of parity check bits	$n - k = m$
Error-correction capability	$t = 1$, $(d_{\min} = 3)$

The parity check matrix $\boldsymbol{H}$ of these codes is formed of the non-zero columns of m bits, and can be implemented in systematic form:

$$\boldsymbol{H} = [\boldsymbol{I_m}\ \boldsymbol{Q}]$$

where the identity submatrix $\boldsymbol{I_m}$ is a square matrix of size $m \times m$ and the submatrix $\boldsymbol{Q}$ consists of the $2^m - m - 1$ columns formed with vectors of weight 2 or more.

For the simplest case, for which $m = 3$,

$$n = 2^3 - 1 = 7$$

$$k = 2^3 - 3 - 1 = 4$$

$$n - k = m = 3$$

$$t = 1(d_{\min} = 3)$$

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

which is the linear block code $C_{\mathrm{b}}(7, 4)$ that has been analysed previously in this chapter. The generator matrix can be constructed using the following expression for linear block codes of systematic form:

$$\boldsymbol{G} = \left[\boldsymbol{Q}^{\boldsymbol{T}}\, I_{2^m - m - 1}\right]$$

In the parity check matrix $\boldsymbol{H}$, the sum of three columns can result in the all-zero vector, and it is not possible for the sum of two columns to give the same result, and so the minimum distance of the code is $d_{\min} = 3$. This means that they can be used for correcting any error pattern of one error, or detecting any error pattern of up to two errors. In this case there are also $2^m - 1$ correctable error patterns and on the other hand there exist 2^m cosets, so that the number of possible correctable error patterns is the same as the number of different cosets (syndrome vectors). The codes with this characteristic are called perfect codes. The code of Example 2.10 is a Hamming code.

2.11 Forward Error Correction and Automatic Repeat ReQuest

2.11.1 Forward Error Correction

Communication systems that use the FEC approach are not able to request a repetition of the transmission of coded information. Due to this, all the capability of the code is used for error correction. The source information generates a binary signal representing equally likely symbols at a rate r_b. The encoder takes a group of k message bits, and adds to it $n - k$ parity check bits. This is the encoding procedure for a linear block code $C_\mathrm{b}(n, k)$ whose code rate is $R_\mathrm{c} = k/n$, with $R_\mathrm{c} < 1$. Figure 2.2 shows a block diagram of an FEC communication system.

The transmission rate r over the channel has to be higher than the source information rate r_b:

$$r = \left(\frac{n}{k}\right) r_\mathrm{b} = \frac{r_\mathrm{b}}{R_\mathrm{c}} \tag{50}$$

The code used in the FEC system is characterized by having a minimum distance $d_{\min} = 2t + 1$. The performance is evaluated of a communication system perturbed by additive white Gaussian noise (AWGN) in the channel, leading to an error probability $p \ll 1$. The source (uncoded) information has an average bit energy E_b, so that the average bit energy for a coded bit is reduced to $R_\mathrm{c} E_\mathrm{b}$. The ratio E_b/N_0 is equal to

$$\left(\frac{E_\mathrm{b}}{N_0}\right)_\mathrm{C} = \frac{R_\mathrm{c} E_\mathrm{b}}{N_0} = R_\mathrm{c}\left(\frac{E_\mathrm{b}}{N_0}\right) \tag{51}$$

As seen in Chapter 1, the quotient between the average bit energy E_b and the power spectral density N_0 plays an important role in the characterization of communication systems, and it will be used a basic parameter for comparison proposes.

In this section the difference between the coded and uncoded cases is analysed.

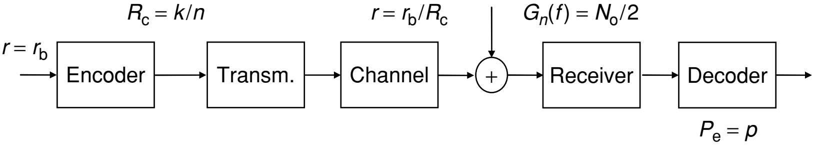


Figure 2.2 Block diagram of an FEC system

There are two error probability definitions: the message bit error rate (BER) or bit error probability, denoted here as P_{be}, and the word error probability, denoted as P_{we}. A given linear block code can correct error patterns of weight t, or less, in a block or codeword of n bits. The word error probability is bounded by

$$P_{\mathrm{we}} \leq \sum_{i=t+1}^{n} P(i,n) \tag{52}$$

Assuming that the error probability of the channel is small, $p \ll 1$, the following approximation can be made:

$$P_{\mathrm{we}} \cong P(t+1,n) \cong \binom{n}{t+1} p^{t+1} \tag{53}$$

This basically means that the words that the code cannot correct have $t+1$ errors. Since the codeword is truncated after decoding, in each uncorrected word there are $\left(\frac{k}{n}\right)(t+1)$ message bit errors, on average. If a large number $N \gg 1$ of words are transmitted, then Nk information source or message bits are transmitted, and the bit error probability is equal to

$$P_{\mathrm{be}} = \frac{\left(\frac{k}{n}\right)(t+1)NP_{\mathrm{we}}}{kN} = \frac{1}{n}(t+1)\,P_{\mathrm{we}} \tag{54}$$

$$P_{\mathrm{be}} \cong \binom{n-1}{t} p^{t+1} \tag{55}$$

For a communication system designed to operate over the AWGN channel, for which the power spectral density is $G_n(f) = N_0/2$, and for which binary polar format and matched filtering are applied, the error probability P_{e} is given by (see Appendix 1)

$$P_{\mathrm{e}} = Q\left(\sqrt{2\frac{E_{\mathrm{b}}}{N_0}}\right) \tag{56}$$

and then

$$p = Q\left(\sqrt{2\left(\frac{E_{\mathrm{b}}}{N_0}\right)_{\mathrm{C}}}\right) = \mathrm{Q}\left(\sqrt{2R_{\mathrm{c}}\frac{E_{\mathrm{b}}}{N_0}}\right) \tag{57}$$

The bit error probability for an FEC system is then

$$P_{\mathrm{be}} \cong \binom{n-1}{t}\left[Q\left(\sqrt{2R_{\mathrm{c}}\frac{E_{\mathrm{b}}}{N_0}}\right)\right]^{t+1} \tag{58}$$

An uncoded communication system has an error probability

$$P_{\mathrm{be}} \cong Q\left(\sqrt{2\frac{E_{\mathrm{b}}}{N_0}}\right) \tag{59}$$

Expressions (58) and (59) allow us to do a comparison between the coded and uncoded cases. This comparison will indicate if there is an improvement or not when using FEC error-control coding, with respect to the uncoded case. This comparison depends on the values of t and R_c, characteristic parameters of the code being used. Even for a good code, if the amount of noise power present in the channel is very large, the coded case usually performs worse than the uncoded case. However, for a reasonable level of noise power, the coded case is better than the uncoded one if a good code is used. Now it is clear why the comparison between the coded and uncoded case is not fair if it is done as in Section 2.2.1, in which the repetition code was introduced, because in that comparison the rate of the code, 1/3, had not been taken into account. The triple repetition of a bit means that the energy per information (message) bit is three times higher, and this should be taken into account if a fair comparison is intended.

Example 2.17: The triple repetition code

By making use of expressions (58) and (59) determine if the triple repetition code has a good performance in comparison with the uncoded case, by plotting the bit error probability as a function of the parameter E_b/N_0 for both cases.

In Figure 2.3 the bit error probability curve (dotted) describes the performance of the repetition code with $n = 3$ [theoretical estimation using (58)], and a simulation curve (dashed), with respect to uncoded binary transmission (solid curve), in both cases using polar format ($A = \pm 1$). This shows that the repetition code with $n = 3$ is even worse than uncoded transmission. This is because the code rate of the repetition code is very small with respect to its error-correction capability.

In the chapters which follow, more efficient codes than the repetition code will be introduced.

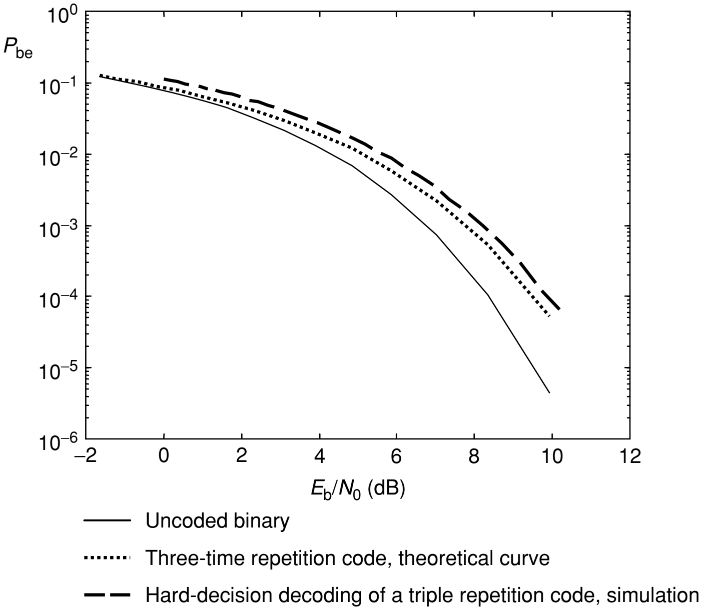


Figure 2.3 Bit error probability for the three-time repetition code

2.11.2 Automatic Repeat ReQuest

ARQ communication systems are based on the detection of errors in a coded block or frame and on the retransmission of the block or frame when errors have been detected. In this case a two-way channel is needed in order to request retransmissions. For a given code, the error-detection capability possible in an ARQ system is higher than the error-correction capability of its FEC counterpart, because the error-control capability of the code is spent only on detection, while the correction requires not only the detection but also the localization of the errors. On the other hand, there is an additional cost in an ARQ system, which is the need for a retransmission link. There are also additional operations for the acknowledgement and repetition processes that reduce the transmission rate of the communication system. A block diagram of an ARQ system is seen in Figure 2.4.

Each codeword is stored in the transmitter buffer and then transmitted. This codeword can be affected by noise, so that at the receiving end the decoder analyses if the received vector belongs or not to the code. A positive acknowledgement (ACK) is transmitted by the receiver if the received vector or word is a codeword; that is, the decoder did not detect any error in that vector. Otherwise the decoder transmits a negative acknowledgement (NAK) if the decoder identifies that the received vector has some errors, and is not a code vector. Upon reception of a NAK message at the transmitter, the corresponding block in the transmitter buffer is retransmitted.

An ARQ system has a reduced transmission rate with respect to an FEC system as a result of the retransmission process. In all of this, it is considered that the error probability over the retransmission channel is negligible. This means that the ACK and NAK messages do not suffer from errors in their transmission. In this way, every word found to have transmission errors is successfully corrected by means of one or more retransmissions. In this case all the error-control capability is spent on error detection, so that $d_{\min} = l + 1$, and the system is able to detect any error pattern of up to l errors. The error probability is determined by the event of an undetected error pattern, that is, an error pattern with $d_{\min} = l + 1$ or more errors:

$$P_{\mathrm{we}} = \sum_{i=l+1}^{n} P(i,n) \cong P(l+1,n) \cong \binom{n}{l+1} p^{l+1} \tag{60}$$

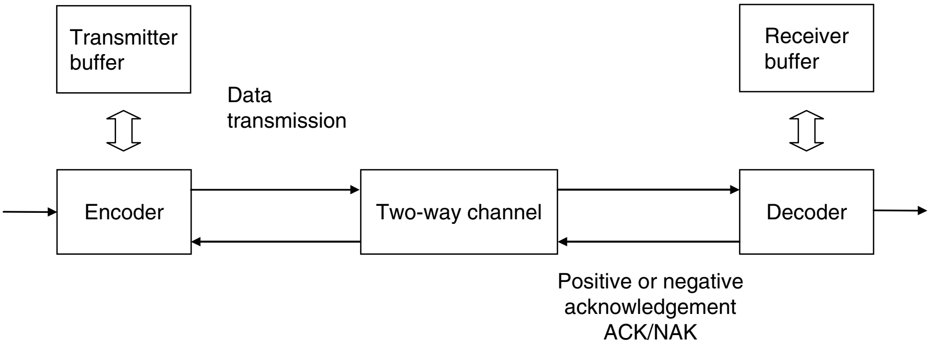


Figure 2.4 Block diagram of an ARQ system

Then the bit error probability is

$$P_{\text{be}} = \left(\frac{l+1}{n}\right) P_{\text{we}} \cong \binom{n-1}{l} p^{l+1} \tag{61}$$

These expressions are similar to those given for the FEC system, and are obtained by replacing t by l.

Retransmissions happen with a certain probability. Retransmission is not required when the receiver has received a valid codeword, an event with probability $P(0, n)$, or when the number of errors in the received vector produces an undetectable error pattern, an event with probability P_{we}. Therefore, the retransmission probability P_{ret} is given by [3]

$$P_{\text{ret}} = 1 - (P(0, n) + P_{\text{we}}) \tag{62}$$

Since usually $P_{\text{we}} \ll P(0, n)$,

$$P_{\text{ret}} \cong 1 - P(0, n) = 1 - (1 - p)^n \cong np \tag{63}$$

2.11.3 ARQ Schemes

2.11.3.1 Stop and wait

The stop-and-wait scheme is such that the transmission of a block or word requires the reception of an acknowledgement (ACK or NAK) of the previous word. The transmitter does not send the following word until the present word has arrived. Figure 2.5 shows the operation of a stop-and-wait ARQ scheme. This scheme requires storage of only one word at the transmitter, which means that the transmitter buffer is of minimum size, but the stopping time D could be very long, and it is related to the transmission delay of the system, t_{d}, where $D \geq 2t_{\text{d}}$.

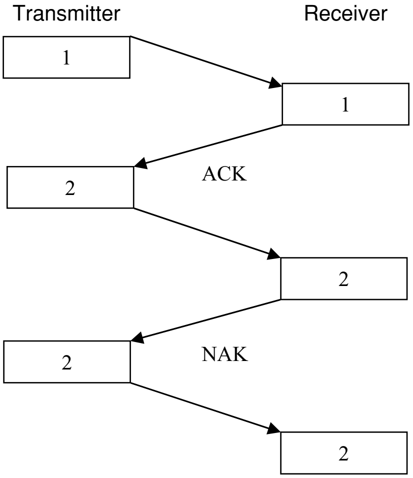


Figure 2.5 Stop-and-wait ARQ scheme

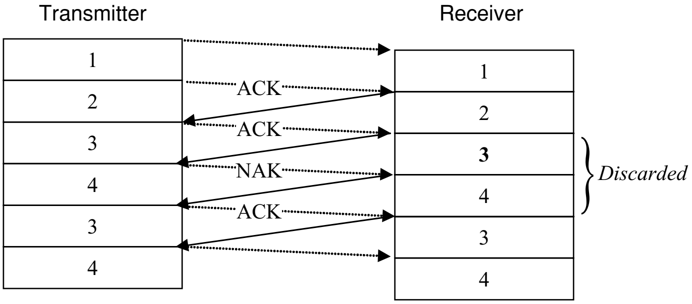


Figure 2.6 A go-back-N ARQ scheme

2.11.3.2 Go-back N

The go-back-N scheme involves a continuous transmission of words. When the receiver sends a NAK, the transmitter goes back to the corresponding word, stored in the transmitter buffer, and restarts the transmission from that word. This requires the storage of N words in the transmitter buffer, where N is determined by the round-trip delay of the system. The receiver discards the $N-1$ words received after detecting one with errors in spite of the possibility of their being correctly received, in order to preserve the order of the word sequence. Thus the receiver needs to store only one word. Figure 2.6 describes the process.

Figure 2.6 shows a go-back-N ARQ scheme with $N = 2$. Both the received word in which errors have been detected and the one which follows it are discarded.

2.11.3.3 Selective repeat

The selective-repeat scheme is the most efficient ARQ scheme in terms of transmission rate, but has the largest memory requirement. When a NAK arrives, the transmitter resends only the corresponding word, and the order of correctly received words is then re-established at the receiver. Figure 2.7 illustrates the selective-repeat ARQ scheme.

In order to properly analyse the overall code rate or efficiency, R_c, of these ARQ schemes, it is necessary to take into account their statistics and delay times.

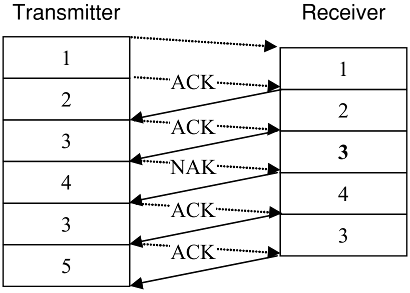


Figure 2.7 Selective-repeat ARQ scheme

2.11.4 ARQ Scheme Efficiencies

To determine the transmission efficiency of the ARQ transmission schemes, the effect of the repetitions or retransmissions in each type of scheme needs to be evaluated.

In schemes based on the retransmission of single words (or blocks, packets or frames), the total number of retransmissions is a random variable m determined by the retransmission probability

$$P(m = 1) = 1 - P_{\text{ret}}, \qquad P(m = 2) = P_{\text{ret}}(1 - P_{\text{ret}}), \text{ etc.}$$

The average number of transmissions of a word in order for it to be accepted as correct is

$$\begin{aligned}\bar{m} &= 1(1 - P_{\text{ret}}) + 2P_{\text{ret}}(1 - P_{\text{ret}}) + 3P_{\text{ret}}^2(1 - P_{\text{ret}}) + \cdots \\ &= (1 - P_{\text{ret}})(1 + 2P_{\text{ret}} + 3P_{\text{ret}}^2 + \cdots) \\ &= \frac{1}{1 - P_{\text{ret}}}\end{aligned} \tag{64}$$

On average the system must transmit $n\bar{m}$ bits to send k message bits. The efficiency therefore is

$$R_c' = \frac{k}{n\bar{m}} = \frac{k}{n}(1 - P_{\text{ret}}) = \frac{k}{n}(1 - p)^n \tag{65}$$

The transmission bit rate r_b and r are related by

$$r = \frac{r_b}{R_c'} \tag{66}$$

The probability of error p is calculated using the appropriate expression [as in equation (57), for example] with R_c' instead of R_c.

The previous expressions apply to systems where single words are retransmitted. In the form described by equations (65) and (66), they characterize the selective-repeat ARQ scheme. However, in the case of the stop-and-wait scheme, the transmission rate is reduced by the factor $\frac{T_w}{T_w+D}$, where $D \geq 2t_d$. The duration t_d is the channel delay, and T_w is the duration of the word or packet, and so

$$T_w = \frac{n}{r} \leq \frac{k}{r_b} \tag{67}$$

Therefore the efficiency of the stop-and-wait ARQ scheme is

$$R_c' = \frac{k}{n}\frac{(1 - P_{\text{ret}})}{(1 + D/T_w)} \leq \left(\frac{k}{n}\right)\frac{(1 - P_{\text{ret}})}{\left(1 + \frac{2t_d r_b}{k}\right)} \tag{68}$$

The ratio D/T_w can be expressed as a function of fixed parameters of the system:

$$\frac{D}{T_w} \geq \frac{2t_d r_b}{k} \tag{69}$$

A go-back-N ARQ scheme does not suffer from wasted transmitter stop times, but has to restart transmission when there are errors. For this case the average number of transmissions of a word is

$$\begin{aligned}\bar{m} &= 1(1-P_{\mathrm{ret}}) + (1+N)P_{\mathrm{ret}}(1-P_{\mathrm{ret}}) + (1+2N)P_{\mathrm{ret}}^2(1-P_{\mathrm{ret}}) + \cdots \\ &= (1-P_{\mathrm{ret}})[1 + P_{\mathrm{ret}} + P_{\mathrm{ret}}^2 + P_{\mathrm{ret}}^3 + \cdots + NP_{\mathrm{ret}}(1 + 2P_{\mathrm{ret}} + 3P_{\mathrm{ret}}^2 + \cdots)] \\ &= (1-P_{\mathrm{ret}})\left[\frac{1}{1-P_{\mathrm{ret}}} + \frac{NP_{\mathrm{ret}}}{(1-P_{\mathrm{ret}})^2}\right] \\ &= 1 + \frac{NP_{\mathrm{ret}}}{1-P_{\mathrm{ret}}}\end{aligned} \tag{70}$$

and so the transmission rate is modified by the factor

$$R_{\mathrm{c}}' = \frac{k}{n}\frac{(1-P_{\mathrm{ret}})}{(1-P_{\mathrm{ret}}+NP_{\mathrm{ret}})} \leq \left(\frac{k}{n}\right)\frac{(1-P_{\mathrm{ret}})}{\left(1-P_{\mathrm{ret}}+\left(\frac{2t_d r_b}{k}\right)P_{\mathrm{ret}}\right)} \tag{71}$$

which involves use of the expression

$$N \geq \frac{2t_{\mathrm{d}}}{T_{\mathrm{w}}} \tag{72}$$

Unlike in the case of a selective-repeat scheme, stop-and-wait and go-back-N schemes exhibit transmission efficiencies that depend on the channel delay. Thus the efficiency R_{c}' in these latter cases is considered to be acceptable if $2t_{\mathrm{d}}r_{\mathrm{b}} \ll k$. Even if $2t_{\mathrm{d}}r_{\mathrm{b}} > k$, go-back-$N$ scheme operates better than stop-and-wait scheme if p is small.

Figure 2.8 shows the efficiencies relative to k/n of the three ARQ schemes. Selective repeat is the most efficient, assuming infinite transmitter memory. Go-back N also has good efficiency if the channel delay is not too large. Channel delay is seen to have a significant effect on the efficiency of the stop-and-wait scheme, becoming unacceptably low if the delay is too long. Selective repeat is the best option in scenarios where high-transmission rates over channels with large delays is required [4].

2.11.5 Hybrid-ARQ Schemes

Up to now the characteristics of FEC and ARQ coding schemes have been described. Their essential difference resides in the possibility of having or not a return link for the transmission of ACK or NAK and to request a retransmission. From the operational point of view, FEC schemes having a constant coding rate, but not a return link over which to request a retransmission, must decode using only the received vector. As previously remarked, given a particular code with minimum distance $d_{\min}$, the number of errors it is capable of correcting, $t = \left\lfloor \frac{d_{\min}-1}{2} \right\rfloor$, is less than the number it is capable of detecting, $l = d_{\min} - 1$. In terms of the most recent and efficient coding techniques, like turbo and low-density parity check codes, it is necessary to use relatively long block lengths in order to obtain efficient and powerful error correction, and the complexity of the decoding operation is normally high.

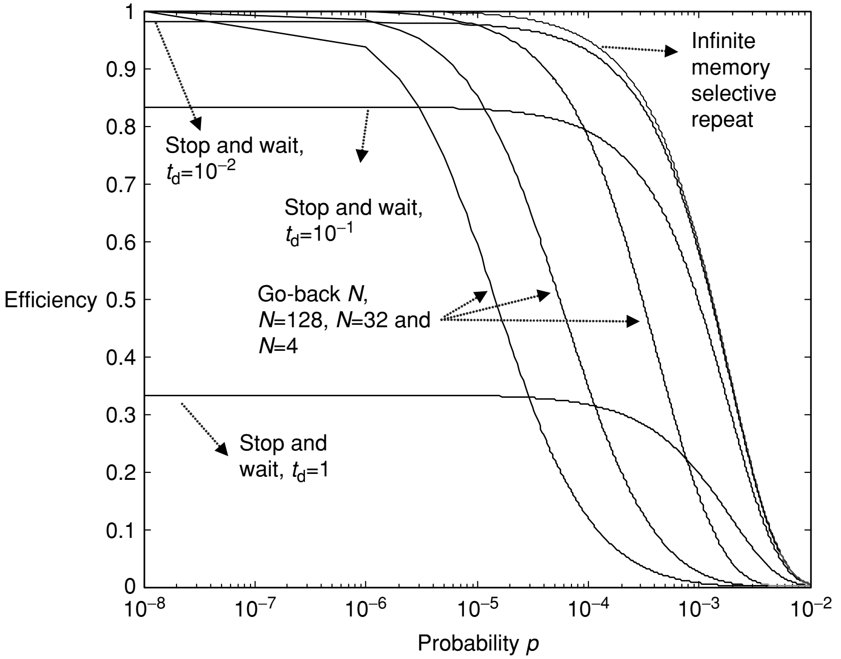


Figure 2.8 Efficiency relative to k/n for stop-and-wait, go-back-N and selective-repeat ARQ schemes, with $r_b = 262$, $k = 262$, $n = 524$; $N = 4$, 32 and 128 (go-back N); and for $t_d = 10^{-2}$, 10^{-1} and 1 (stop and wait)

On the other hand, ARQ systems have a coding rate that varies with the channel noise conditions. In addition, even the detection of very few errors leads to a retransmission and hence a loss of efficiency. In this sense, FEC systems maintain their code rate while correcting small numbers of errors without needing retransmissions. However, there exists the possibility of erroneous decoding if the number of errors exceeds the correction capability of the code, and so the possibility of retransmission can lead to ARQ schemes being more reliable.

Depending therefore on the state of the channel, one or other of the systems will be more efficient. This suggests that where possible a combination of the two techniques might operate more efficiently than either on its own. This combination of ARQ and FEC schemes is called hybrid ARQ. Here the FEC is used to correct small numbers of errors, which also will normally occur most often. Given that correction of only a small number of errors per word or packet is intended, the corresponding code can be quite simple. The FEC attempts correction of a word, and if the syndrome evaluated over the corrected word indicates that it is a valid codeword, then it is accepted. If the syndrome indicates that errors are detected in the corrected word, then it is highly likely that there were more errors in the received word than the FEC could correct, and ARQ is used to ask for a retransmission.

There are two forms of hybrid ARQ, called type 1 and type 2. Hybrid-ARQ type 1 is typically based on a block code that is designed partly to correct a small number of errors and partly to detect a large number of errors, as determined by the expression $d_{\min} \geq l + \lambda + 1$. So for example a code with minimum distance $d_{\min} = 11$ could be used to correct patterns of up to $\lambda = 3$ errors, and also to detect any pattern of up to $l = 7$ errors. It should be remembered that $l > \lambda$ always applies in the above expression.

As will be seen in the following chapters, block codes can often be decoded by solving a system of equations. Appropriate construction of the system of equations allows the number of errors in correctable error patterns to be controlled, permitting correction of all patterns of up to λ errors. The code can also have additional capability devoted to error detection. The decoder therefore attempts to correct up to λ errors per block or packet, and recalculates the syndrome vector of the decoded word. If the syndrome is equal to zero, the word is accepted, and if not, a retransmission is requested. In general, compared to a purely ARQ scheme (non-hybrid), the code rate R_c in a hybrid scheme will be less than that for a non-hybrid scheme. This reduces the efficiency at low-error probabilities. However, at high-error probabilities the efficiency of a non-hybrid scheme decreases more rapidly than that of a hybrid scheme as the error rate increases.

Hybrid-ARQ-type 2 schemes generate an improvement in transmission efficiency by sending parity bits for error correction only when they are needed. This technique involves two kinds of codes, one a code $C_0(n, k)$ which has high rate and is used for error detection only, and the other a 1/2-rate code $C_1(2k, k)$ which has the property that the information bits can be obtained from the parity bits by a process of inversion, and for that reason it is called an invertible code. In what follows, a codeword will be represented in its systematic form as a vector where the parity bits are denoted as a function of the message (information) bits:

$$\boldsymbol{c} = (f(m_0, m_1, \ldots, m_{k-1}), m_0, m_1, \ldots, m_{k-1}) = (f(\boldsymbol{m}), \boldsymbol{m})$$

The operation of the hybrid-ARQ-type 2 scheme can be seen in Figures 2.9 and 2.10. In these figures the notation used is, for example, that $\tilde{\boldsymbol{c}} = \left(\tilde{f}(\boldsymbol{m}), \tilde{\boldsymbol{m}}\right)$ is the received version (affected by the channel and so possibly containing errors) of $\boldsymbol{c} = (f(\boldsymbol{m}), \boldsymbol{m})$. Also, $f(\boldsymbol{m})$ is the redundancy obtained using code C_0, while $q(\boldsymbol{m})$ is that using C_1. The hybrid-ARQ-type 2 scheme operates by repeatedly iterating between the two alternatives described in the flow charts of Figures 2.9 and 2.10, respectively. In this iteration the transmission or retransmission is affected by sending the codeword produced by code C_0 from the message itself, $\boldsymbol{m}$, or by sending the redundancy (parity bits) of the codeword produced by code C_1, $q(\boldsymbol{m})$, respectively. The receiver stores the received versions of the message (information bits) or the redundancy (parity bits), respectively, depending on which alternative is being carried out. Having the message bits and the parity bits generated by code C_1, the decoder in the receiver can apply error correction. As the redundancy and the message have the same number of bits, the retransmission does not lead to a loss of efficiency over a system that always retransmits the message $\boldsymbol{m}$ while also permitting an additional decoding process for more effective overall error control. The process continues until the message bits are correctly decoded. For more details of hybrid-ARQ schemes, and particularly of the invertible codes used in type 2, the reader is referred to the book by Lin and Costello [4].

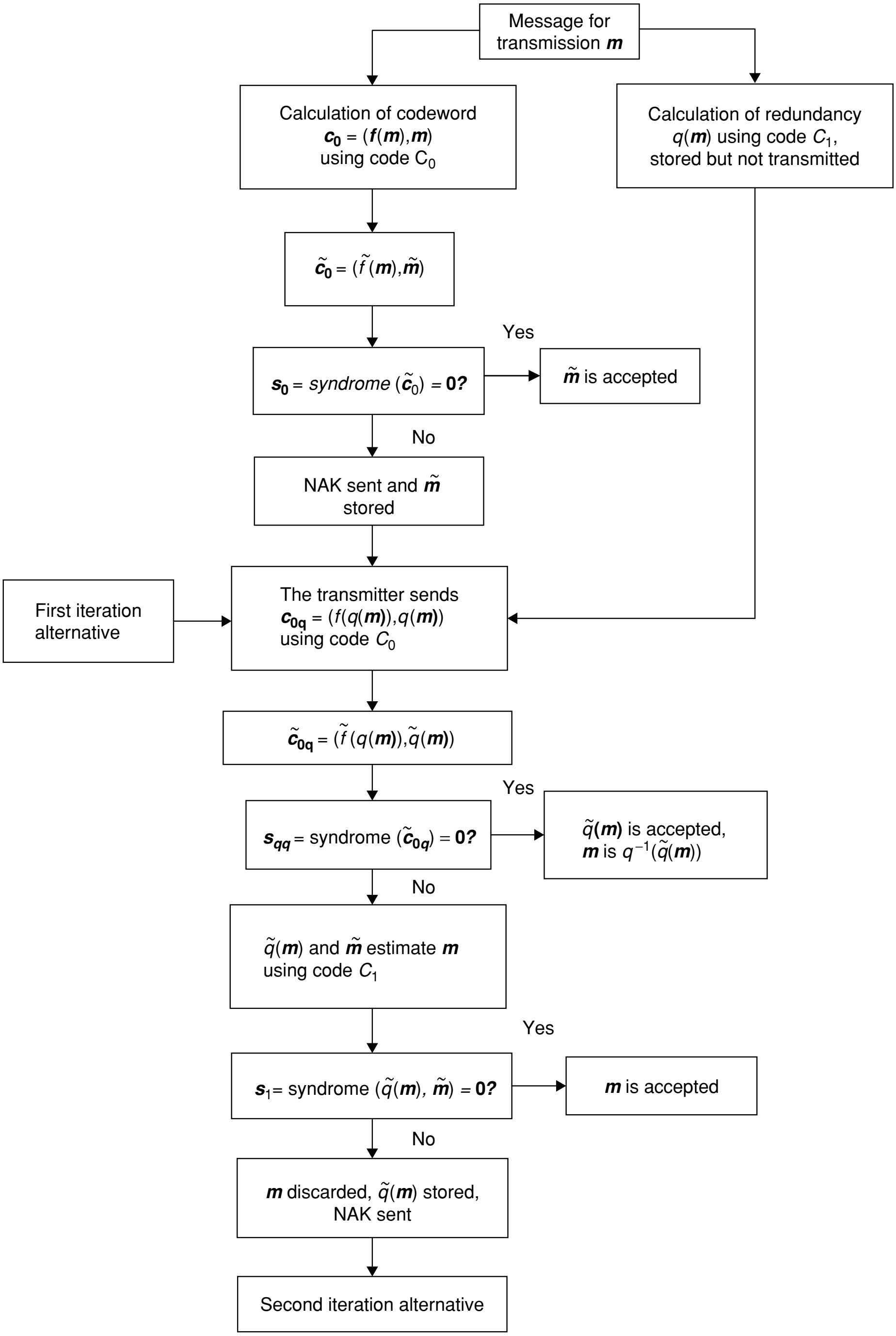

Message for transmission $\boldsymbol{m}$
Calculation of codeword $\boldsymbol{c}_0 = (\boldsymbol{f}(\boldsymbol{m}),\boldsymbol{m})$ using code C_0
Calculation of redundancy $q(\boldsymbol{m})$ using code C_1, stored but not transmitted
$\tilde{\boldsymbol{c}}_0 = (\tilde{f}(\boldsymbol{m}),\tilde{\boldsymbol{m}})$
Yes
$\boldsymbol{s}_0 = syndrome\ (\tilde{\boldsymbol{c}}_0) = \boldsymbol{0}?$
$\tilde{\boldsymbol{m}}$ is accepted
No
NAK sent and $\tilde{\boldsymbol{m}}$ stored
First iteration alternative
The transmitter sends $\boldsymbol{c}_{0q} = (f(q(\boldsymbol{m})),q(\boldsymbol{m}))$ using code C_0
$\tilde{\boldsymbol{c}}_{0q} = (\tilde{f}(q(\boldsymbol{m})),\tilde{q}(\boldsymbol{m}))$
Yes
$\boldsymbol{s}_{qq} =$ syndrome $(\tilde{\boldsymbol{c}}_{0q}) = \boldsymbol{0}?$
$\tilde{q}(\boldsymbol{m})$ is accepted, $\boldsymbol{m}$ is $q^{-1}(\tilde{q}(\boldsymbol{m}))$
No
$\tilde{q}(\boldsymbol{m})$ and $\tilde{\boldsymbol{m}}$ estimate $\boldsymbol{m}$ using code C_1
Yes
$\boldsymbol{s}_1 =$ syndrome $(\tilde{q}(\boldsymbol{m}), \tilde{\boldsymbol{m}}) = \boldsymbol{0}?$
$\boldsymbol{m}$ is accepted
No
$\boldsymbol{m}$ discarded, $\tilde{q}(\boldsymbol{m})$ stored, NAK sent
Second iteration alternative

Figure 2.9 First alternative of the hybrid-ARQ-type 2 iterative scheme

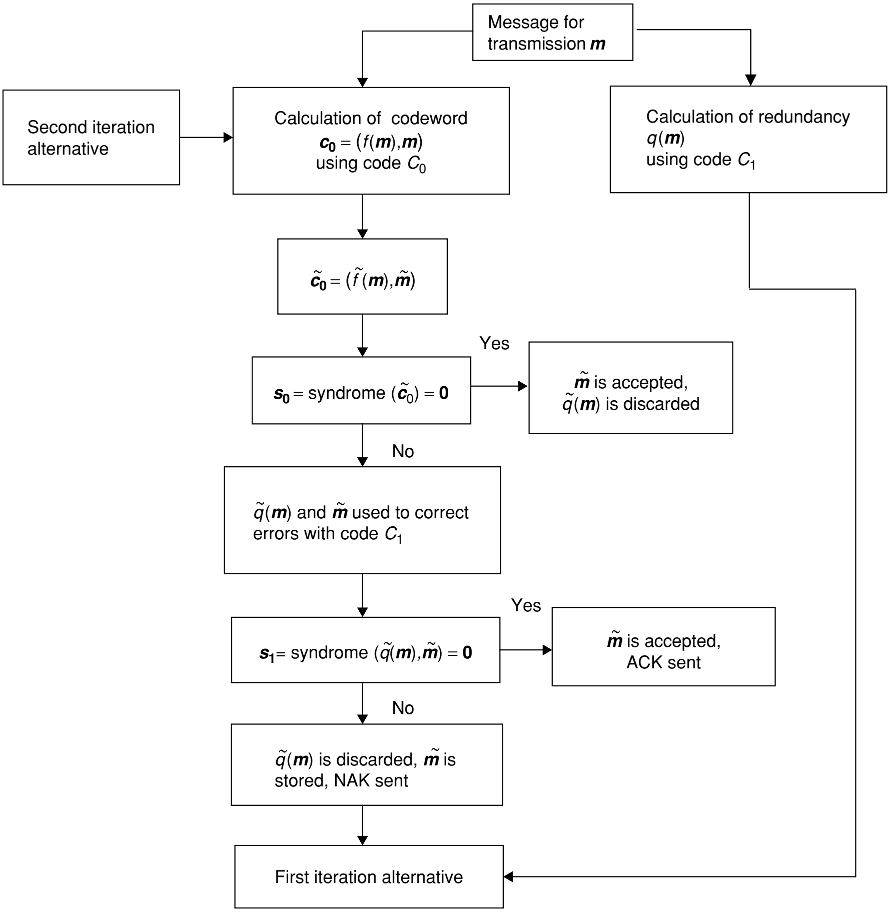

Message for transmission $\boldsymbol{m}$
Second iteration alternative
Calculation of codeword $\boldsymbol{c}_0 = (f(\boldsymbol{m}),\boldsymbol{m})$ using code C_0
Calculation of redundancy $q(\boldsymbol{m})$ using code C_1
$\tilde{\boldsymbol{c}}_0 = (\tilde{f}(\boldsymbol{m}),\tilde{\boldsymbol{m}})$
Yes
$\boldsymbol{s}_0$ = syndrome $(\tilde{\boldsymbol{c}}_0) = \mathbf{0}$
$\tilde{\boldsymbol{m}}$ is accepted, $\tilde{q}(\boldsymbol{m})$ is discarded
No
$\tilde{q}(\boldsymbol{m})$ and $\tilde{\boldsymbol{m}}$ used to correct errors with code C_1
Yes
$\boldsymbol{s}_1$= syndrome $(\tilde{q}(\boldsymbol{m}),\tilde{\boldsymbol{m}}) = \mathbf{0}$
$\tilde{\boldsymbol{m}}$ is accepted, ACK sent
No
$\tilde{q}(\boldsymbol{m})$ is discarded, $\tilde{\boldsymbol{m}}$ is stored, NAK sent
First iteration alternative

Figure 2.10 Second alternative of the hybrid-ARQ-type 2 iterative scheme

Bibliography and References

[1] Shannon, C. E., “A mathematical theory of communication,” *Bell Syst. Tech. J.*, vol. 27, pp. 379–423, 623–656, July and October 1948.

[2] Shannon, C. E., “Communications in the presence of noise,”*Proc. IEEE*, vol. 86, no. 2, pp. 447–458, February 1998.

[3] Carlson, B., *Communication Systems. An Introduction to Signals and Noise in Electrical Communication*, 3rd Edition, McGraw-Hill, New York, 1986.

[4] Lin, S. and Costello, D. J., Jr., *Error Control Coding: Fundamentals and Applications*, Prentice Hall, Englewood Cliffs, New Jersey, Ed. 1983 and 2004.

[5] MacWilliams, F. J. and Sloane, N. J. A., *The Theory of Error-Correcting Codes*, North-Holland, Amsterdam, 1977.

[6] Sklar, B., *Digital Communications, Fundamentals and Applications,* Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[7] Berlekamp, E. R., *Algebraic Coding Theory,* McGraw-Hill, New York, 1968.

[8] Peterson, W. W. and Weldon, E. J., Jr., *Error-Correcting Codes*, 2nd Edition, MIY Press, Cambridge, Massachusetts, 1972.

[9] Hillma, A. P. and Alexanderson, G. L., *A First Undergraduate Course in Abstract Algebra,* 2nd Edition, Wadsworth, Belmont, California, 1978.

[10] Allenby, R. B. J., *Rings, Fields and Groups: An Introduction to Abstract Algebra*, Edward Arnold, London, 1983.

[11] Hamming, R. W., "Error detecting and error correcting codes," *Bell Syst. Tech. J.*, vol. 29, pp. 147–160, April 1950.

[12] Proakis, J. G. and Salehi, M., *Communication Systems Engineering*, Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[13] Proakis, J. G., *Digital Communications*, 2nd Edition, McGraw-Hill, New York, 1989.

[14] McEliece, R. J., *The Theory of Information and Coding*, Addison-Wesley, Massachusetts, 1977.

[15] Adámek, J., *Foundations of Coding: Theory and Applications of Error-Correcting Codes with an Introduction to Cryptography and Information Theory*, Wiley Interscience, New York, 1991.

[16] Slepian, D., "Group codes for the Gaussian channel," *Bell Syst. Tech. J.,* vol. 47, pp. 575–602, 1968.

[17] Caire, G. and Biglieri, E., "Linear block codes over cyclic groups," *IEEE Trans. Inf. Theory*, vol. 41, no. 5, pp. 1246–1256, September 1995.

[18] Forney, G. D., Jr., "Coset codes–part I: Introduction and geometrical classification,"*IEEE Trans. Inf. Theory,* vol. 34, no. 5, pp. 1123–1151, September 1988.

Problems

2.1 For the triple repetition block code $C_b(3, 1)$ generated by using the parity check submatrix $\boldsymbol{P} = [1\ 1]$, construct the table of all the possible received vectors and calculate the corresponding syndrome vectors $\boldsymbol{S} = \boldsymbol{r} \bullet \boldsymbol{H}^T$ to determine correctable and detectable error patterns, according to the error-correction capability of the code.

2.2 The minimum Hamming distance of a block code is $d_{\min} = 11$. Determine the error-correction and error-detection capability of this code.

2.3 (a) Determine the minimum Hamming distance of a code of code length n that should detect up to six errors and correct up to four errors per code vector, in a transmission over a BSC. (b) If the same block code is used on a binary erasure channel, how many erasures in a block can it detect, and how many can it correct? (c) What is the minimum block length that a code with this minimum Hamming distance can have?

2.4 A binary block code has code vectors in systematic form as given in Table P.2.1.

Table P.2.1 A block code table

0	0	0	0	0	0
0	1	1	1	0	0
1	0	1	0	1	0
1	1	0	1	1	0
1	1	0	0	0	1
1	0	1	1	0	1
0	1	1	0	1	1
0	0	0	1	1	1

(a) What is the rate of the code?
(b) Write down the generator and parity check matrices of this code in systematic form.
(c) What is the minimum Hamming distance of the code?
(d) How many errors can it correct, and how many can it detect?
(e) Compute the syndrome vector for the received vector $\boldsymbol{r} = (101011)$ and hence find the location of any error.

2.5 (a) Construct a linear block code $C_{\mathrm{b}}(5, 2)$, maximizing its minimum Hamming distance.
(b) Determine the generator and parity check matrices of this code.

2.6 A binary linear block code has the following generator matrix in systematic form:

$$\boldsymbol{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

(a) Find the parity check matrix $\boldsymbol{H}$ and hence write down the parity check equations.
(b) Find the minimum Hamming distance of the code.

2.7 The generator matrix of a binary linear block code is given below:

$$\boldsymbol{G} = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}$$

(a) Write down the parity check equations of the code.
(b) Determine the code rate and minimum Hamming distance.
(c) If the error rate at the input of the decoder is 10^{-3}, estimate the error rate at the output of the decoder.

2.8 The Hamming block code $C_b(15, 11)$ has the following parity check submatrix:

$$\boldsymbol{P} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

(a) Construct the parity check matrix of the code.
(b) Construct the error pattern syndrome table.
(c) Apply syndrome decoding to the received vector $\boldsymbol{r} = (011111001011011)$.

2.9 A binary Hamming single-error-correcting code has length $n = 10$. What is
(a) the number of information digits k;
(b) the number of parity check digits $n - k$;
(c) the rate of the code;
(d) the parity check equation;
(e) the code vector if all the information digits are '1's and
(f) the syndrome if an error occurs at the seventh digit of a code vector?

2.10 Random errors with probability $p = 10^{-3}$ on a BSC are to be corrected by a random-error-control block code with $n = 15$, $k = 11$ and $d_{\min} = 3$. What is the overall throughput rate and block error probability after decoding, when the code is used
(a) in FEC mode and
(b) in retransmission error correction (ARQ) mode?

2.11 An FEC scheme operates on an AWGN channel and it should perform with a bit error rate $P_{be} < 10^{-4}$, using the minimum transmitted power. Options for this scheme are the block codes as given in Table P.2.2.

Table P.2.2 Options for Problem 2.11

n	k	$d_{\min}$
31	26	3
31	21	5
31	16	7

(a) Determine the best option if minimum transmitted power is required.
(b) Calculate the coding gain at the desired P_{be} with respect to uncoded transmission.

2.12 An ARQ scheme operates on an AWGN channel and it should perform with a bit error rate $P_{be} < 10^{-5}$. Options for this scheme are the block codes as given in Table P.2.3.

(a) Determine for each case the required value of r_b/r.

(b) Determine for each case the required value of E_b/N_0.

Table P.2.3 Options for Problem 2.12

n	k	d_{min}
12	11	2
15	11	3
16	11	4

2.13 Show that the generator matrix of a linear block error-correcting code can be derived from the parity check equations of the code.

3

Cyclic Codes

Cyclic codes are an important class of linear block codes, characterized by the fact of being easily implemented using sequential logic or shift registers.

3.1 Description

For a given vector of n components, $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1})$, a right-shift rotation of its components generates a different vector. If this right-shift rotation is done i times, a cyclically rotated version of the original vector is obtained as follows:

$$\boldsymbol{c}^{(i)} = (c_{n-i},\ c_{n-i+1}, \ldots,\ c_{n-1},\ c_0,\ c_1, \ldots,\ c_{n-i-1})$$

A given linear block code is said to be cyclic if for each of its code vectors the ith cyclic rotation is also a code vector of the same code [1–3]. Also, remember that being a linear block code, the sum of any two code vectors of a cyclic code is also a code vector. As an example, the linear block code $C_\mathrm{b}(7, 4)$ described in Chapter 2 is also a cyclic linear block code.

3.2 Polynomial Representation of Codewords

Codewords of a given cyclic code $C_\mathrm{cyc}(n,\ k)$ can be represented by polynomials. These polynomials are defined over a Galois field GF(2^m), being of particular interest those defined over the binary field GF(2).

A polynomial is an expression in a variable X, constituted of terms of the form $c_i X^i$, where the coefficients c_i belong to the field GF(2^m) over which the polynomial is defined, and the exponent i is an integer number that corresponds to the position of the coefficient or element in a given code vector. A polynomial representation $c(X)$ of a code vector $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1})$ is then of the form

$$c(X) = c_0 + c_1 X + \cdots + c_{n-1} X^{n-1} \qquad c_i \in \mathrm{GF}(2^m) \tag{1}$$

Operations with polynomials defined over a given field are the same as usual, but following the operation rules defined over that field. If $c_{n-1} = 1$, the polynomial is called monic. In

Essentials of Error-Control Coding Jorge Castiñeira Moreira and Patrick Guy Farrell

the case of polynomials defined over the binary field GF(2), the operations are addition and multiplication modulo 2, described in the previous chapter. The addition of two polynomials defined over GF(2), $c_1(X)$ and $c_2(X)$, is therefore the modulo-2 addition of the coefficients corresponding to the same term X^i, that is, those with the same exponent (or at the same position in the code vector). If

$$c_1(X) = c_{01} + c_{11}X + \cdots + c_{n-1,1}X^{n-1}$$

and

$$c_2(X) = c_{02} + c_{12}X + \cdots + c_{n-1,2}X^{n-1} \tag{2}$$

then the addition of these two polynomials is equal to

$$c_1(X) \oplus c_2(X) = c_{01} \oplus c_{02} + (c_{11} \oplus c_{12})\, X + \cdots + \left(c_{n-1,1} \oplus c_{n-1,2}\right) X^{n-1} \tag{3}$$

In this case, the degree of the polynomial, which is the highest value of the exponent over the variable X, is $n - 1$. However the addition of any two polynomials can also be calculated using expression (3) even when the degrees of the polynomials are not the same.

The multiplication of two polynomials is done using multiplication and addition modulo 2. The multiplication of polynomials $c_1(X)$ and $c_2(X)$, $c_1(X) \bullet c_2(X)$, is calculated as

$$c_1(X) \bullet c_2(X) = c_{01} \bullet c_{02} + (c_{01} \bullet c_{12} \oplus c_{02} \bullet c_{11})\, X + \cdots + \left(c_{n-1,1} \bullet c_{n-1,2}\right) X^{2(n-1)} \tag{4}$$

Operations in expressions (3) and (4) are all additions and multiplications modulo 2. Addition and multiplication of polynomials obey the commutative, associative and distributive laws.

It is also possible to define division of polynomials. Let $c_1(X)$ and $c_2(X)$ be two polynomials defined over the binary field, both of the form of equation (2), and where $c_2(X) \neq 0$, a non-zero polynomial. The division operation between these two polynomials is defined by the existence of two unique polynomials of the same binary field $q(X)$ and $r(X)$, called the quotient and the remainder, respectively, which fit the following equation:

$$c_1(X) = q(X) \bullet c_2(X) \oplus r(X) \tag{5}$$

Additional properties of polynomials defined over the Galois field GF(2^m) can be found in Appendix B.

In the following, and for notational simplicity, addition and multiplication will be denoted by '+' and '.', instead of being denoted by the modulo-2 operation symbols '$\oplus$' and '$\bullet$'.

The polynomial description defined over a given Galois field, and, in particular, when it is the binary field, allows us a more suitable analysis of a cyclic code $C_{\mathrm{cyc}}(n, k)$. In this polynomial representation, the variable X is used in its exponential form X^j, where j identifies the position of '1's in the code vector equivalent to the polynomial. This is the same as saying that if the term X^j exists in the polynomial corresponding to a given code vector, there is a one '1' in position j of that code vector or codeword, while if it does not exist, this position is occupied by a zero '0'. Therefore, the polynomial expression $c(X)$ of a code vector $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1})$ is of the form of equation (1)

$$c(X) = c_0 + c_1X + \cdots + c_{n-1}X^{n-1}$$

Thus, a polynomial for a code vector of n components is a polynomial of degree $n - 1$ or less. Codewords of a given cyclic code $C_{\mathrm{cyc}}(n, k)$ will be equivalently referred to as code vectors or code polynomials.

3.3 Generator Polynomial of a Cyclic Code

The i-position right-shift rotation of a code vector $\boldsymbol{c}$ has the following polynomial expression:

$$c^{(i)}(X) = c_{n-i} + c_{n-i+1}X + \cdots + c_{n-i-1}X^{n-1} \tag{6}$$

The relationship between the i-position right-shift rotated polynomial $c^{(i)}(X)$ and the original code polynomial $c(X)$ is of the form

$$X^i c(X) = q(X)(X^n + 1) + c^{(i)}(X) \tag{7}$$

The polynomial expression for the i-position right-shift rotated polynomial $c^{(i)}(X)$ of the original code polynomial $c(X)$ is then equal to

$$c^{(i)}(X) = X^i c(X) \bmod (X^n + 1) \tag{8}$$

Mod is the modulo operation defined now over polynomials; that is, it is calculated by taking the remainder of the division of $X^i c(X)$ and $X^n + 1$.

Among all the code polynomials of a given cyclic code $C_{\mathrm{cyc}}(n, k)$, there will be a certain polynomial of minimum degree [1]. This polynomial will have minimum degree r, so that in its polynomial expression the term of the form X^r will exist; that is, the coefficient c_r will be equal to '1'. Therefore, this polynomial will be of the form $g(X) = g_0 + g_1X + \cdots + X^r$. If there is another polynomial with minimum degree, this polynomial would be of the form $g_1(X) = g_{10} + g_{11}X + \cdots + X^r$. However, because the cyclic code $C_{\mathrm{cyc}}(n, k)$ is a linear block code, the sum of these two code polynomials should belong to the code, and this sum will end up being a polynomial of degree $(r - 1)$, which contradicts the initial assumption that the minimum possible degree is r. Therefore, the non-zero minimum-degree code polynomial of a given cyclic code $C_{\mathrm{cyc}}(n, k)$ is unique. It is possible to demonstrate that in the non-zero minimum-degree polynomial of a given cyclic code $C_{\mathrm{cyc}}(n, k)$, $g_0 = 1$. Then, the expression for such a non-zero minimum-degree polynomial of a given cyclic code $C_{\mathrm{cyc}}(n, k)$ is

$$g(X) = 1 + g_1X + \cdots + g_{r-1}X^{r-1} + X^r \tag{9}$$

On the other hand, polynomials of the form $Xg(X),\ X^2g(X), \ldots, X^{n-r-1}g(X)$ are all polynomials of degree less than n, and so by using expression (7) to express $X^i g(X)$, it can be seen that division of each of these polynomials by the polynomial $X^n + 1$ will result in a quotient equal to zero, $q(X) = 0$, which means that

$$\begin{aligned} &Xg(X) = g^{(1)}(X) \\ &X^2g(X) = g^{(2)}(X) \\ &\vdots \\ &X^{(n-r-1)}g(X) = g^{(n-r-1)}(X) \end{aligned} \tag{10}$$

So these polynomials are right-shift rotations of the minimum-degree polynomial, and are also code polynomials. Since a cyclic code $C_{\text{cyc}}(n, k)$ is also a linear block code, linear combinations of code polynomials are also code polynomials, and therefore

$$c(X) = m_0 g(X) + m_1 X g(X) + \cdots + m_{n-r-1} X^{n-r-1} g(X)$$

$$c(X) = \left(m_0 + m_1 X + \cdots + m_{n-r-1} X^{n-r-1}\right) g(X) \tag{11}$$

In expression (11), $g(X)$ is the non-zero minimum-degree polynomial of the cyclic code $C_{\text{cyc}}(n, k)$, described in equation (9). Expression (11) determines that a code polynomial $c(X)$ is a multiple of the non-zero minimum-degree polynomial $g(X)$. This property is very useful for the encoding and decoding of a cyclic code $C_{\text{cyc}}(n, k)$. Coefficients $m_i,\ i = 0,\ 1,\ 2, \ldots,\ n - r - 1$, in expression (11) are elements of GF(2); that is, they are equal to zero or one. Then there will be 2^{n-r} polynomials of degree $n - 1$ or less that are multiples of $g(X)$. These are all the possible linear combinations of the initial set of code polynomials so that they form a cyclic code $C_{\text{cyc}}(n, k)$. For a bijective assignment between the message and the coded vector spaces, there should be 2^k possible linear combinations. Therefore, $2^{n-r} = 2^k$ or $r = n - k$. In other words, r, the degree of the non-zero minimum-degree polynomial, is also the number of redundancy bits the code adds to the message vector.

The non-zero minimum-degree polynomial is then of the form

$$g(X) = 1 + g_1 X + \cdots + g_{n-k-1} X^{n-k-1} + X^{n-k} \tag{12}$$

Summarizing, in a linear cyclic code $C_{\text{cyc}}(n, k)$, there is a unique non-zero minimum-degree code polynomial, and any other code polynomial is a multiple of this polynomial.

The non-zero minimum-degree polynomial is of degree r, and any other code polynomial of the linear cyclic code $C_{\text{cyc}}(n, k)$ is of degree $n - 1$ or less, and so

$$c(X) = m(X) g(X) = \left(m_0 + m_1 X + \cdots + m_{k-1} X^{k-1}\right) g(X) \tag{13}$$

where $m_i,\ i = 0,\ 1,\ 2, \ldots,\ k - 1$, are the bits of the message vector to be encoded. Since the non-zero minimum-degree code polynomial completely determines and generates the linear cyclic code $C_{\text{cyc}}(n, k)$, it is called the generator polynomial.

Example 3.1: Determine the code vectors corresponding to the message vectors $\boldsymbol{m}_0 = (0000)$, $\boldsymbol{m}_1 = (1000)$, $\boldsymbol{m}_2 = (0100)$ and $\boldsymbol{m}_3 = (1100)$ of the linear cyclic code $C_{\text{cyc}}(7, 4)$ generated by the generator polynomial $g(X) = 1 + X + X^3$.

The corresponding code vectors and code polynomials are listed in Table 3.1.

Table 3.1 Code polynomials of a linear cyclic code $C_{\text{cyc}}(7, 4)$

Message $\boldsymbol{m}$	Code vectors $\boldsymbol{c}$	Code polynomials $c(X)$
0000	0000000	$0 = 0g(X)$
1000	1101000	$1 + X + X^3 = 1g(X)$
0100	0110100	$X + X^2 + X^4 = Xg(X)$
1100	1011100	$1 + X^2 + X^3 + X^4 = (1 + X)g(X)$

There are two important relationships between the generator polynomial $g(X)$ and the polynomial $X^n + 1$. The first one is that if $g(X)$ is a generator polynomial of a given linear cyclic code $C_{\text{cyc}}(n, k)$, then $g(X)$ is a factor of $X^n + 1$.

The demonstration of this is as follows: If the generator polynomial of degree r is multiplied by X^k, it converts into a polynomial of degree n, because $n = k + r$. By applying equation (8), and dividing $X^k g(X)$ by $X^n + 1$, the quotient $q(X)$ is equal to one, because these are two monic polynomials of the same degree. Thus

$$X^k g(X) = (X^n + 1) + g^{(k)}(X) = (X^n + 1) + a(X)g(X)$$

$$\left(X^k + a(X)\right) g(X) = (X^n + 1) \tag{14}$$

since $g^{(k)}(X)$ is a code polynomial as it is a kth right-shift rotation of $g(X)$. Thus, $g(X)$ is a factor of $X^n + 1$.

The same can be stated in reverse; that is, if a polynomial of degree $r = n - k$ is a factor of $X^n + 1$, then this polynomial generates a linear cyclic code $C_{\text{cyc}}(n, k)$.

Any polynomial factor of $X^n + 1$ can generate a linear cyclic code $C_{\text{cyc}}(n, k)$.

3.4 Cyclic Codes in Systematic Form

So far, the encoding procedure for a linear cyclic code $C_{\text{cyc}}(n, k)$ has been introduced as a multiplication between the message polynomial $m(X)$ and the generator polynomial $g(X)$, and this operation is sufficient to generate any code polynomial of the code. However, this encoding procedure is essentially non-systematic. Given a polynomial that fits the conditions for being the generator polynomial of a linear cyclic code $C_{\text{cyc}}(n, k)$, and if the message polynomial is of the form

$$m(X) = m_0 + m_1 X + \cdots + m_{k-1} X^{k-1} \tag{15}$$

then the systematic version of the linear cyclic code $C_{\text{cyc}}(n, k)$ can be obtained by performing the following operations [1–3]:

The polynomial $X^{n-k} m(X) = m_0 X^{n-k} + m_1 X^{n-k+1} + \cdots + m_{k-1} X^{n-1}$ is first formed, and then divided by the generator polynomial $g(X)$:

$$X^{n-k} m(X) = q(X)g(X) + p(X) \tag{16}$$

Here $p(X)$ is the remainder polynomial of the division of equation (16), which has degree $n - k - 1$ or less, since the degree of $g(X)$ is $n - k$. By reordering equation (16), we obtain

$$X^{n-k} m(X) + p(X) = q(X)g(X)$$

where it is seen that the polynomial $X^{n-k} m(X) + p(X)$ is a code polynomial because it is a factor of $g(X)$. In this polynomial, the term $X^{n-k} m(X)$ represents the message polynomial right-shifted $n - k$ positions, whereas $p(X)$ is the remainder polynomial of this division and acts as the redundancy polynomial, occupying the lower degree terms of the polynomial expression in X. This procedure allows the code polynomial to adopt the systematic form

$$\begin{aligned} c(X) &= X^{n-k} m(X) + p(X) \\ &= p_0 + p_1 X + \cdots + p_{n-k-1} X^{n-k-1} + m_0 X^{n-k} + m_1 X^{n-k+1} + \cdots + m_{k-1} X^{n-1} \end{aligned} \tag{17}$$

that when expressed as a code vector is equal to

$$\boldsymbol{c} = (p_0,\ p_1,\ \ldots,\ p_{n-k-1},\ m_0,\ m_1,\ \ldots,\ m_{k-1}) \tag{18}$$

Example 3.2: For the linear cyclic code $C_{\text{cyc}}(7,4)$ generated by the generator polynomial $g(X) = 1 + X + X^3$, determine the systematic form of the codeword corresponding to the message vector $\boldsymbol{m} = (1010)$.

The message polynomial is $m(X) = 1 + X^2$, and as $n - k = 7 - 4 = 3$, the polynomial $X^3 m(X) = X^3 + X^5$ is calculated. The polynomial division is done as follows:

$$\begin{array}{lcl} X^5 + X^3 & | & X^3 + X + 1 \\ X^5 + X^3 + X^2 & & X^2 \\ -\,-\,-\,-\,-\,-\,-\,- & & \\ X^2 = p(X) & & \end{array}$$

Then

$$c(X) = p(X) + X^3 m(X) = X^2 + X^3 + X^5$$

and so

$$\boldsymbol{c} = (0011010)$$

Table 3.2 shows the linear cyclic code $C_{\text{cyc}}(7,\ 4)$ generated by the polynomial $g(X) = 1 + X + X^3$, which is the same as that introduced in Chapter 2 as the linear block code $C_{\text{b}}(7,4)$.

Table 3.2 Linear cyclic code $C_{\text{cyc}}(7,4)$ generated by the polynomial $g(X) = 1 + X + X^3$

Message $\boldsymbol{m}$	Code vector $\boldsymbol{c}$
0 0 0 0	0 0 0 0 0 0 0
0 0 0 1	1 0 1 0 0 0 1
0 0 1 0	1 1 1 0 0 1 0
0 0 1 1	0 1 0 0 0 1 1
0 1 0 0	0 1 1 0 1 0 0
0 1 0 1	1 1 0 0 1 0 1
0 1 1 0	1 0 0 0 1 1 0
0 1 1 1	0 0 1 0 1 1 1
1 0 0 0	1 1 0 1 0 0 0
1 0 0 1	0 1 1 1 0 0 1
1 0 1 0	0 0 1 1 0 1 0
1 0 1 1	1 0 0 1 0 1 1
1 1 0 0	1 0 1 1 1 0 0
1 1 0 1	0 0 0 1 1 0 1
1 1 1 0	0 1 0 1 1 1 0
1 1 1 1	1 1 1 1 1 1 1

3.5 Generator Matrix of a Cyclic Code

As seen in previous sections, a linear cyclic code $C_{\text{cyc}}(n, k)$ generated by the generator polynomial $g(X) = 1 + g_1 X + \cdots + g_{n-k-1} X^{n-k-1} + X^{n-k}$ is spanned by the k code polynomials, $g(X),\ Xg(X), \ldots, X^{n-k} g(X)$, which can be represented as row vectors of a generator matrix of dimension $k \times n$:

$$\boldsymbol{G} = \begin{bmatrix} g_0 & g_1 & g_2 & \cdots & g_{n-k} & 0 & 0 & \cdots & 0 \\ 0 & g_0 & g_1 & \cdots & g_{n-k-1} & g_{n-k} & 0 & \cdots & 0 \\ \vdots & \vdots & & & \vdots & \vdots & & & \vdots \\ 0 & 0 & & \cdots & g_0 & g_1 & g_2 & \cdots & g_{n-k} \end{bmatrix} \tag{19}$$

where $g_0 = g_{n-k} = 1$.

This generator matrix is not of systematic form. In general, and by operating over the rows of this matrix, a systematic form generator matrix can be obtained.

Example 3.3: For the linear cyclic code $C_{\text{cyc}}(7, 4)$ generated by the polynomial $g(X) = 1 + X + X^3$, determine the corresponding generator matrix and then convert it into a systematic generator matrix.

In this case the matrix is of the form

$$\boldsymbol{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Linear row operations over this matrix can be carried out to obtain a systematic form of the generator matrix. These row operations are additions and multiplications in the binary field GF(2). Thus, and by replacing the third row by the addition of the first and third rows, the matrix becomes

$$\boldsymbol{G}' = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

And replacing the fourth row by the addition of the first, second and fourth rows, the matrix becomes

$$G'' = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

This last modified matrix $\boldsymbol{G}''$ generates the same code as that of the generator matrix $\boldsymbol{G}$, but the assignment between the message and the code vector spaces is different in each case. Observe that the modified and systematic generator matrix $\boldsymbol{G}''$ is the same as that of the linear block code $C_{\text{b}}(7, 4)$ introduced in Chapter 2.

Summarizing, the systematic form of a linear cyclic code $C_{\text{cyc}}(n, k)$ and its corresponding expressions are obtained by calculating the following:

The polynomial X^{n-k+i}, $i = 0, 1, 2, \ldots, k-1$, is divided by the generator polynomial $g(X)$:

$$X^{n-k+i} = q_i(X)g(X) + p_i(X) \tag{20}$$

where $p_i(X)$ is the remainder of the division and

$$p_i(X) = p_{i0} + p_{i1}X + \cdots + p_{i,n-k-1}X^{n-k-1} \tag{21}$$

Since $X^{n-k+i} + p_i(X) = q_i(X)g(X)$ is multiple of $g(X)$, it is a code polynomial. Its coefficients can be ordered into a matrix

$$\boldsymbol{G} = \begin{bmatrix} p_{00} & p_{01} & \cdots & p_{0,n-k-1} & 1 & 0 & 0 & \cdots & 0 \\ p_{10} & p_{11} & \cdots & p_{1,n-k-1} & 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ p_{k-1,0} & p_{k-1,1} & \cdots & p_{k-1,n-k-1} & 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \tag{22}$$

This matrix is a generator matrix for the linear cyclic code $C_{\text{cyc}}(n, k)$ in systematic form. Similarly, the corresponding parity check matrix $\boldsymbol{H}$ can also be found as follows:

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & p_{00} & p_{10} & \cdots & p_{k-1,0} \\ 0 & 1 & \cdots & 0 & 0 & p_{01} & p_{11} & \cdots & p_{k-1,1} \\ \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 & 1 & p_{0,n-k-1} & p_{1,n-k-1} & \cdots & p_{k-1,n-k-1} \end{bmatrix} \tag{23}$$

Example 3.4: The polynomial $X^7 + 1$ can be factorized as follows:

$$X^7 + 1 = (1 + X)(1 + X + X^3)(1 + X^2 + X^3)$$

Since $n = 7$, both the polynomials $g_2(X) = 1 + X + X^3$ and $g_3(X) = 1 + X^2 + X^3$, which are of degree $r = n - k = 3$, generate cyclic codes $C_{\text{cyc}}(7, 4)$. In the same way, the polynomial $g_4(X) = (1 + X)(1 + X + X^3) = 1 + X^2 + X^3 + X^4$ generates a cyclic code $C_{\text{cyc}}(7, 3)$.

The decomposition into factors of $X^7 + 1$ is unique, and is of the form seen above, so that the different cyclic codes of length $n = 7$ can be identified and generated.

The cyclic code generated by the polynomial $g_1(X) = 1 + X$ is a linear block code of even parity. This is because its generator matrix is of the form

$$G_1 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

which can be converted by Gaussian elimination (the general form of the process introduced in Example 3.3) into a matrix of systematic form

$$G_1' = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

where it is seen that the sum of any number of rows results in a codeword with an even number of ‘1’s.

The generator polynomial $g_2(X) = 1 + X + X^3$ corresponds to a cyclic Hamming code, as introduced in Example 3.2. The generator polynomial $g_3(X) = 1 + X^2 + X^3$ is also the generator polynomial of a cyclic Hamming code whose generator matrix in systematic form is equal to

$$G_3' = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

In this generator matrix, the submatrix $\boldsymbol{P}$ has at least two ‘1’s in each row, which makes it a generator matrix of a Hamming code $C_{\mathrm{b}}(7, 4)$.

Both the generator polynomials $g_4(X) = (1 + X)(1 + X + X^3) = 1 + X^2 + X^3 + X^4$ and $g_5(X) = (1 + X)(1 + X^2 + X^3) = 1 + X + X^2 + X^4$ generate cyclic codes $C_{\mathrm{cyc}}(7,\ 3)$.

Finally, the generator polynomial $g_6(X) = (1 + X + X^3)(1 + X^2 + X^3) = 1 + X + X^2 + X^3 + X^4 + X^5 + X^6$ corresponds to a repetition code $C_{\mathrm{cyc}}(7,\ 1)$ with a generator matrix of the form

$$G_7 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

3.6 Syndrome Calculation and Error Detection

As defined in Chapter 2 for block codes, the received vector, which is the transmitted vector containing possible errors, is $\boldsymbol{r} = (r_0,\ r_1, \ldots,\ r_{n-1})$. This is a vector with elements of the Galois field GF(2), which can also have a polynomial representation

$$r(X) = r_0 + r_1X + r_2X^2 + \cdots + r_{n-1}X^{n-1} \tag{24}$$

Dividing this polynomial by $g(X)$ gives

$$r(X) = q(X)g(X) + S(X) \tag{25}$$

where the remainder of this division is a polynomial of degree $n - k - 1$ or less. Since a code polynomial is a multiple of $g(X)$, then if the remainder of the division (25) is zero, the received polynomial is a code polynomial. If the division (25) has a non-zero polynomial as the remainder, then the procedure detects a polynomial that does not belong to the code. The syndrome vector is again a vector of $n - k$ components that are at the same time the coefficients of the polynomial $S(X) = s_0 + s_1X + \cdots + s_{n-k-1}X_{n-k-1}$.

The following theorem can be stated for the syndrome polynomial:

Theorem 3.1: If the received polynomial $r(X) = r_0 + r_1X + r_2X^2 + \cdots + r_{n-1}X^{n-1}$ generates the syndrome polynomial $S(X)$, then a cyclic shift (rotation) of the received polynomial $r^{(1)}(X)$ generates the syndrome polynomial $S^{(1)}(X)$.

From equation (7),

$$Xr(X) = r_{n-1}(X^n + 1) + r^{(1)}(X) \tag{26}$$

or

$$r^{(1)}(X) = r_{n-1}(X^n + 1) + Xr(X) \tag{27}$$

If this expression is divided by $g(X)$,

$$f(X)g(X) + t(X) = r_{n-1}g(X)h(X) + X\,[q(X)g(X) + S(X)] \tag{28}$$

where $t(X)$ is the syndrome polynomial of $r^{(1)}(X)$. Reordering this equation, we get

$$XS(X) = [f(X) + r_{n-1}h(X) + Xq(X)]\,g(X) + t(X) \tag{29}$$

which means that $t(X)$ is the remainder of the division of $XS(X)$ by $g(X)$, and so $t(X) = S^{(1)}(X)$. This procedure can be extended by induction in order to determine that $S^{(i)}(X)$ is the syndrome polynomial of $r^{(i)}(X)$.

As in the case of any block code, if $c(X) = c_0 + c_1X + \cdots + c_{n-1}X^{n-1}$ is a code polynomial and $e(X) = e_0 + e_1X + \cdots + e_{n-1}X^{n-1}$ is the error pattern in polynomial form, then

$$r(X) = c(X) + e(X) = q(X)g(X) + S(X) \tag{30}$$

and

$$\begin{aligned} e(X) &= c(X) + q(X)g(X) + S(X) \\ &= f(X)g(X) + q(X)g(X) + S(X) \\ &= [f(X) + q(X)]\,g(X) + S(X) \end{aligned} \tag{31}$$

If the received polynomial $r(X)$ is divided by the generator polynomial $g(X)$, the remainder obtained by this operation is the corresponding syndrome polynomial $S(X)$.

3.7 Decoding of Cyclic Codes

The decoding of cyclic codes can be implemented in the same way as for block codes. A table $S \to e$ identifying the relationship between the syndromes and the error patterns can be constructed first, and then the syndrome polynomial is evaluated for the received polynomial $r(X)$ by dividing this polynomial by the generator polynomial $g(X)$ to obtain the syndrome polynomial. The constructed table allows us to identify the error pattern that corresponds to

Table 3.3 Single error patterns and their syndromes for the cyclic linear block code $C_{\text{cyc}}(7,4)$

Error patterns (polynomial form)	Syndrome polynomials	Vector Form
$e_6(X) = X^6$	$S(X) = 1 + X^2$	101
$e_5(X) = X^5$	$S(X) = 1 + X + X^2$	111
$e_4(X) = X^4$	$S(X) = X + X^2$	011
$e_3(X) = X^3$	$S(X) = 1 + X$	110
$e_2(X) = X^2$	$S(X) = X^2$	001
$e_1(X) = X^1$	$S(X) = X$	010
$e_6(X) = X^0$	$S(X) = 1$	100

the calculated syndrome, according to equation (31). The following is an example based on the cyclic code $C_{\text{cyc}}(7,4)$.

Example 3.5: The cyclic linear block code $C_{\text{cyc}}(7,4)$ generated by the polynomial $g(X) = 1 + X + X^3$, which has a minimum Hamming distance $d_{\min} = 3$, can correct any single error pattern. There are seven patterns of this kind. Table 3.3 shows single error patterns and their corresponding syndromes for this cyclic linear block code. The all-zero pattern corresponds to the decoding of a code polynomial (i.e., the no-error case).

As in the block code case, once the syndrome has been calculated and the corresponding error pattern found from the table, then the error-correction procedure is to add the error pattern to the received vector. Of course, if an error-pattern polynomial has the form of a code polynomial, then it will be undetectable and therefore uncorrectable, as both the syndrome and the error pattern will be null vectors.

Theorem 3.1 is the basis of several other decoding algorithms for cyclic codes [1]. One of them is the Meggitt algorithm [5]. This decoder is based on the relationship between the cyclically shifted versions of a given codeword and their corresponding shifts in the received vector. It operates in a bit-by-bit mode, so that it first determines if the most significant bit contains an error or not, and then cyclically shifts the received vector and the corresponding syndrome in order to analyse the following bits of the received vector. The algorithm decides whether the most significant bit contains an error, or not, by determining if the syndrome evaluated over the received polynomial corresponds to a syndrome for the error pattern with an error in the most significant bit. If this is the case, the algorithm changes that bit, and then accordingly modifies the corresponding syndrome. By continuing to cyclically shift the modified received vector and syndrome, errors will be corrected as they are shifted into the most significant position. The algorithm accepts the received vector or polynomial as a code vector or polynomial if at the end of all the modifications the final version of the syndrome polynomial is the zero polynomial. Since the decoder operates only on the most significant bit, it is necessary to store only the syndromes corresponding to errors in that bit (as well as in other bits if correcting multiple errors), thus reducing the size of the table $S \to e$.

As a result of this interesting property, which allows the cyclic decoding to be performed in a bit-by-bit mode, and with cyclic shifts of the received vector, cyclic codes are particularly useful for the correction of burst errors, either clustered within a given code vector or affecting the first and the final parts of a code vector. In this case the error pattern is of the form $e(X) = X^j B(X)$, where $0 \leq j \leq n-1$, and $B(X)$ is a polynomial of degree equal to or less than $n-k-1$, which will not contain $g(X)$ as a factor.

A modification of the Meggitt decoder is the error-trapping decoder. This algorithm is particularly efficient in the correction of single error patterns, double error patterns in some codes and burst errors. Its efficiency falls for other error patterns. The error-trapping decoder can be applied to error patterns with certain properties, such as a sequence of $n-k$ bits or less placed the most significant positions of the received vector. In this case it is possible to show that the error polynomial $e(X)$ is equal to $X^k S^{(n-k)}(X)$, where $S^{(n-k)}(X)$ is the syndrome polynomial of the received polynomial $r(X)$ shifted by $n-k$ positions. This property makes easier the correction of such an error pattern, because it is necessary to calculate only $S^{(n-k)}(X)$ and then add $X^k S^{(n-k)}(X)$ to $r(X)$. In the same way, it is possible to perform the decoding of error patterns of size $n-k$ or less even when they do not occur in the most significant positions of the code vector, hence the name of the error-trapping algorithm. It can be shown that, for a given cyclic code, capable of correcting error patterns of t random errors or less, error-trapping happens if the weight of the syndrome vector is less than or equal to t. Details of this algorithm can be found in [1].

3.8 An Application Example: Cyclic Redundancy Check Code for the Ethernet Standard

One of the most interesting applications of cyclic codes is the cyclic redundancy check (CRC) code utilized in the Ethernet protocol. The redundancy calculated by the systematic procedure of cyclic coding is placed in the so-called frame check sequence (FCS) field, which follows the data block in the data frame of that protocol. The cyclic code used in this case is a cyclic code that adds 32 redundancy bits, and its corresponding polynomial generator is

$$g(X) = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$$

One interesting property of cyclic block codes is that the number of parity check bits the code adds to the message vector is equal to the degree of the generator polynomial. Thus, and independently of the size of the data packet to be transmitted, the CRC for the standard Ethernet, for instance, always adds 32 bits of redundancy.

In the case of the standard protocol Ethernet, the data packet size varies from $k = 512$ to $k = 12{,}144$ bits. The k information bits are the first part of the whole message, and are considered as a polynomial $m(X)$, which is followed by the $n-k = 32$ redundancy bits that result from the division of the shifted message polynomial $X^{n-k}m(X)$ by $g(X)$. The receiver does the same operation on the message bits, so that if the redundancy calculated at the receiver is equal to the redundancy sent in the FCS, then the packet is accepted as a valid one. A retransmission of the packet is required if the recalculated redundancy is different from the contents of the FCS.

Consider a simple example of this procedure, using the cyclic block code $C_{\text{cyc}}(7,4)$ introduced in Example 3.3. Assume that the message $\boldsymbol{m} = (1010)$ has to be transmitted. After encoding this message, the resulting code vector is $\boldsymbol{c} = (\mathbf{001}1010)$, where the redundancy is in bold. One way of performing the decoding of this code vector is to recalculate at the receiver the redundancy obtained by encoding the message vector $\boldsymbol{m} = (1010)$, and by verifying that

Table 3.4 Minimum Hamming distance for different packet configurations in the standard protocol Ethernet

Code length n	Minimum Hamming distance $d_{\min}$
3007–12,144	4
301–3006	5
204–300	6
124–203	7
90–123	8

this redundancy is equal to (**001**). An equivalent decoding method consists of evaluating the syndrome vector over the whole code vector $\boldsymbol{c} = (\mathbf{001}1010)$, in order to verify if this syndrome vector is the all- zero vector. In the case of the Ethernet protocol, the decoding is performed as in the former case.

The code length can be variable in the Ethernet protocol, as we have seen. This makes the error-detection capability also variable with the code length. In [6] the values of the minimum Hamming distance as a function of the code length are presented, in the case of the standard Ethernet protocol. Table 3.4 determines the minimum Hamming distance as a function of the code length.

This means that, depending on the packet length, three to seven random errors will always be detectable, as well as certain patterns of much larger numbers of random errors and many burst error patterns (in fact, any error pattern which is not a codeword).

Bibliography and References

[1] Lin, S. and Costello, D. J., Jr., *Error Control Coding: Fundamentals and Applications*, Prentice Hall, Englewood Cliffs, New Jersey, 1983.

[2] Carlson, B., *Communication Systems: An Introduction to Signals and Noise in Electrical Communication*, 3rd Edition, McGraw-Hill, New York, 1986.

[3] Sklar, B., *Digital Communications, Fundamentals and Applications*, Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[4] Berlekamp, E. R., *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.

[5] Meggitt, J. E., "Error correcting codes and their implementation," *IRE Trans. Inf. Theory*, vol. IT-7, pp. 232–244, October 1961.

[6] Adámek, J., *Foundations of Coding: Theory and Applications of Error-Correcting Codes with an Introduction to Cryptography and Information Theory*, Wiley Interscience, New York, 1991.

[7] Peterson, W. W. and Wledon, E. J., Jr., *Error-Correcting Codes*, 2nd Edition, MIY Press, Cambridge, Massachusetts, 1972.

[8] McEliece, R. J., *The Theory of Information and Coding*, Addison-Wesley, Massachusetts, 1977.

[9] MacWilliams, F. J. and Sloane, N. J. A., *The Theory of Error-Correcting Codes*, North-Holland, Amsterdam, The Netherlands, 1977.

[10] Baldini, R., *Coded Modulation Based on Ring of Integers*, PhD Thesis, University of Manchester, Manchester, 1992.

[11] Baldini, R. and Farrell, P. G., "Coded modulation based on rings of integers modulo-q. Part 1: Block codes," *IEE Proc. Commun.*, vol. 141, no. 3, pp.129–136, June 1994.

[12] Piret, P., "Algebraic construction of cyclic codes over Z with a good Euclidean minimum distance," *IEEE Trans. Inf. Theory*, vol. 41, no. 3, May 1995.

[13] Hillma, A. P. and Alexanderson, G. L., *A First Undergraduate Course in Abstract Algebra*, 2nd Edition, Wadsworth, Belmont, California, 1978.

[14] Allenby, R. B. J., *Rings, Fields and Groups: An Introduction to Abstract Algebra*, Edward Arnold, London, 1983.

Problems

3.1 Determine if the polynomial $1 + X + X^3 + X^4$ is a generator polynomial of a binary linear cyclic block code with code length $n \leq 7$.

3.2 Verify that the generator polynomial $g(X) = 1 + X + X^2 + X^3$ generates a binary cyclic code $C_{\text{cyc}}(8, 5)$ and determine the code polynomial for the message vector $\boldsymbol{m} = (10101)$ in systematic form.

3.3 A binary linear cyclic code $C_{\text{cyc}}(n, k)$ has code length $n = 7$ and generator polynomial $g(X) = 1 + X^2 + X^3 + X^4$.
(a) Find the code rate, the generator and parity check matrices of the code in systematic form, and its Hamming distance.
(b) If all the information symbols are '1's, what is the corresponding code vector?
(c) Find the syndrome corresponding to an error in the first information symbol, and show that the code is capable of correcting this error.

3.4 Define what is meant by a cyclic error-control code.

3.5 A binary linear cyclic block code $C_{\text{cyc}}(n, k)$ has code length $n = 14$ and generator polynomial $g(X) = 1 + X^3 + X^4 + X^5$.
(a) If all the information symbols are '1's, what is the corresponding code vector?
(b) Find the syndrome corresponding to an error in the last information symbol. Is this code capable of correcting this error?
(c) Can cyclic codes be non-linear?

3.6 (a) Determine the table of code vectors of the binary linear cyclic block code $C_{\text{cyc}}(6, 2)$ generated by the polynomial $g(X) = 1 + X + X^3 + X^4$.
(b) Calculate the minimum Hamming distance of the code, and its error-correction capability.

3.7 A binary linear cyclic block code with a code length of $n = 14$ has the generator polynomial $g(X) = 1 + X^2 + X^6$.

(a) Determine the number of information and parity check bits in each code vector.

(b) Determine the number of code vectors in the code.

(c) Determine the generator and parity check matrices of the code.

(d) Determine the minimum Hamming distance of the code.

(e) Determine the burst error-correction capability of the code.

(f) Describe briefly how to encode and decode this code.

3.8 For a given binary linear cyclic block code $C_{\text{cyc}}(15,\ 11)$ generated by the polynomial $g(X) = 1 + X + X^4$,

(a) determine the code vector in systematic form of the message vector $m = (11001101011)$ and

(b) decode the received vector $\boldsymbol{r} = (000010001101011)$.

■

4

BCH Codes

BCH (Bose, Chaudhuri [1], Hocquenghem [2]) codes are a class of linear and cyclic block codes that can be considered as a generalization of the Hamming codes, as they can be designed for any value of the error-correction capability t. These codes are defined in the binary field GF(2), and also in their non-binary version, over the Galois field GF(q). Included in this latter case is the most relevant family of non-binary codes, which is the family of Reed–Solomon codes [3], to be presented in Chapter 5.

4.1 Introduction: The Minimal Polynomial

As seen in the previous chapter, cyclic codes are linear block codes with the special property that cyclically shifted versions of code vectors are also code vectors. As a special case introduced as an example of a linear block code, the Hamming code $C_{\mathrm{cyc}}(7,\ 4)$ has also been shown to be a cyclic code generated by the generator polynomial $g(X) = 1 + X + X^3$.

Cyclic codes have the property that the code vectors are generated by multiplying the message polynomial $m(X)$ by the generator polynomial $g(X)$. In fact, in the non-systematic form the message polynomial $m(X)$ is multiplied by the generator polynomial $g(X)$, and in the systematic form there is an equivalent polynomial $q(X)$ that is multiplied by the generator polynomial $g(X)$. Being a polynomial, $g(X)$ has to have roots that are in number equal to its degree. A generator polynomial with coefficients over GF(2) need not have all of its roots belonging to this field, as some of the roots can be elements of the extended field GF(2^m). This concept is described and clarified in Appendix B, in which an example of the extended field GF(2^3) = GF(8) is introduced.

As an example, the Hamming code $C_{\mathrm{cyc}}(7,\ 4)$ introduced in Chapters 2 and 3 is analysed here, in order to see which are the roots of its generator polynomial $g(X) = g_1(X)$. Since any code vector is generated by multiplying a given message vector by this generator polynomial, any code vector, seen as a code polynomial, will have at least the same roots as this generator polynomial, $g(X) = g_1(X)$. This allows us to construct a system of syndrome equations, as the roots of the code polynomials are known. Taking into account the example in Table B.3 of Appendix B, for the case of the extended Galois field GF(8), it is seen that α, which is a primitive element of that field, is also a root of the primitive polynomial $p(X) = 1 + X + X^3$,

Essentials of Error-Control Coding Jorge Castiñeira Moreira and Patrick Guy Farrell

which should not be confused with the generator polynomial of the code under analysis, $g_1(X)$. However, in this particular case, they are the same. Therefore it is possible to say that α is a root of $g_1(X)$ because, from Table B.3,

$$g_1(\alpha) = 1 + \alpha + \alpha^3 = 1 + \alpha + 1 + \alpha = 0$$

If α is a root of $g_1(X)$, then it is also a root of any code polynomial $g_1(X)$ of the Hamming code $C_{\text{cyc}}(7,\ 4)$, and this allows us to state a first syndrome equation $s_1 = r(\alpha)$. As the degree of the generator polynomial is 3, there are still two other roots to be found for this polynomial. By substituting the element α^2 of the extended field GF(8), it is found that

$$g_1(\alpha^2) = 1 + \alpha^2 + \alpha^6 = 1 + \alpha^2 + 1 + \alpha^2 = 0$$

This verifies that α^2 is a root of the generator polynomial $g_1(X)$ of the Hamming code $C_{\text{cyc}}(7,\ 4)$, and so it is also a root of any code polynomial of this code, allowing us to form a second syndrome equation $s_2 = r(\alpha^2)$. By substituting all the elements of the extended field, it is possible to identify that α^4 is the third root of the generator polynomial $g_1(X)$ of the Hamming code $C_{\text{cyc}}(7,\ 4)$, and also to state that $1,\ \alpha^3,\ \alpha^5$ and α^6 are not roots of that polynomial. Since $\alpha,\ \alpha^2$ and α^4 are roots of the polynomial $g_1(X)$,

$$\begin{aligned} g_1(X) &= (X + \alpha)(X + \alpha^2)(X + \alpha^4) \\ &= X^3 + (\alpha + \alpha^2 + \alpha^4)X^2 + (\alpha^3 + \alpha^5 + \alpha^6)X + 1 \\ &= X^3 + X + 1 \end{aligned}$$

By substituting in the expression for a received polynomial the roots α and α^2, for instance, it is possible to find a system of two equations that allows the solution of two unknowns, which are the position and the value of a single error in that polynomial.

In order to correct an error pattern containing two errors, for instance, it would be necessary to have a system of at least four equations, to solve for the positions and values of the two errors. This would be the case of a linear block code able to correct error patterns of size $t = 2$. In the case of the Hamming code $C_{\text{cyc}}(7,\ 4)$, there is just one additional root, α^4, associated with the additional equation $s_4 = r(\alpha^4)$, making only three in all. This is not enough to allow the construction of a set of four equations, the minimal requirement for correcting error patterns of size $t = 2$. This is in agreement with the fact that this Hamming code $C_{\text{cyc}}(7,\ 4)$, characterized by a minimum Hamming distance $d_{\min} = 3$, can correct only error patterns of size $t = 1$.

The other elements of the extended field GF(8), $\alpha^3,\ \alpha^5$ and α^6, which were found not to be roots of the generator polynomial $g_1(X)$ of the Hamming code $C_{\text{cyc}}(7,\ 4)$, can however be the roots of another polynomial, and thus

$$\begin{aligned} g_2(X) &= (X + \alpha^3)(X + \alpha^5)(X + \alpha^6) \\ &= X^3 + (\alpha^3 + \alpha^5 + \alpha^6)X^2 + (\alpha + \alpha^2 + \alpha^4)X + 1 \\ &= X^3 + X^2 + 1 \end{aligned}$$

This polynomial also generates a linear cyclic block code $C_{\text{cyc}}(7,\ 4)$, as seen in Example 3.4 in Chapter 3, because this polynomial is a factor of $X^7 + 1$. In fact both polynomials $g_1(X)$ and $g_2(X)$ are factors of $X^7 + 1$, and the remaining polynomial that gives the whole factorization of $X^7 + 1$ is $g_3(X) = X + 1$, whose unique root is the element 1 of the extended field GF(8). In this way, all the non-zero elements of the extended field GF(8) are roots of $X^7 + 1$.

The polynomial $g_1(X) = \Phi_1(X)$ is called the minimal polynomial of the elements α, α^2 and α^4, and it is essentially a polynomial for which these elements are roots [4]. In the same way, $g_2(X) = \Phi_2(X)$ is called the minimal polynomial of the elements α^3, α^5 and α^6, and $g_3(X) = \Phi_3(X)$ is the minimal polynomial of the element 1.

Since, for instance, the Hamming code $C_{\mathrm{cyc}}(7,\ 4)$ has a generator polynomial with just three roots, and is not able to guarantee the correction of any error pattern of size $t = 2$, it would be possible to add the missing root, α^3, by forming a generator polynomial as the multiplication of $g_1(X) = \Phi_1(X)$ and $g_2(X) = \Phi_2(X)$. However, it could be more appropriate to take the lowest common multiple (LCM) of these two polynomials, in order to avoid multiple roots, which will only add redundancy without improving the error-correction capability of the code. Note that the degree of the generator polynomial is the level of redundancy added by the coding technique. Therefore, in this particular case, the minimum common multiple of $g_1(X) = \Phi_1(X)$ and $g_2(X) = \Phi_2(X)$ will form a generator polynomial with roots $\alpha,\ \alpha^2,\ \alpha^3,\ \alpha^4,\ \alpha^5$ and α^6, so that α^5 and α^6 are also added automatically. Then the resulting generator polynomial is of the form

$$\begin{aligned} g_4(X) &= \Phi_1(X)\Phi_2(X) \\ &= (X^3 + X + 1)(X^3 + X^2 + 1) \\ &= X^6 + X^5 + X^4 + X^3 + X^2 + X + 1 \end{aligned}$$

which is, as has been seen in Chapter 3, Example 3.4, the generator polynomial of a cyclic repetition code with $n = 7$, $C_{\mathrm{cyc}}(7,\ 1)$, whose minimum Hamming distance is $d_{\min} = 7$, able to correct any error pattern of size $t = 3$ or less. This is in agreement with the fact that, in this case, there is a system of six equations that allows us to determine the positions and values of up to three errors in a given codeword, as a consequence of its generator polynomial $g_4(X)$ having as roots the elements $\alpha,\ \alpha^2,\ \alpha^3,\ \alpha^4,\ \alpha^5$ and α^6.

This introduction leads to a more formal definition of a BCH code.

4.2 Description of BCH Cyclic Codes

As said in the above sections, BCH codes are a generalization of Hamming codes, and they can be designed to be able to correct any error pattern of size t or less [1, 2, 4]. In this sense the generalization of the Hamming codes extends the design of codes for $t = 1$ (Hamming codes) to codes for any desired higher value of t (BCH codes). The design method is based on taking an LCM of appropriate minimal polynomials, as described in the previous section for a particular example.

For any positive integer $m \geq 3$ and $t < 2^{m-1}$, there exists a binary BCH code $C_{\mathrm{BCH}}(n, k)$ with the following properties:

Code length	$n = 2^m - 1$
Number of parity bits	$n - k \leq mt$
Minimum Hamming distance	$d_{\min} \geq 2t + 1$
Error-correction capability	t errors in a code vector

These codes are able to correct any error pattern of size t or less, in a code vector of length $n, n = 2^m - 1$.

The generator polynomial of a BCH code is described in terms of its roots, taken from the Galois field GF(2^m). If α is a primitive element in GF(2^m), the generator polynomial $g(X)$ of a BCH code for correcting t errors in a code vector of length $n = 2^m - 1$ is the minimum-degree polynomial over GF(2) that has $\alpha, \alpha^2, \dots, \alpha^{2t}$ as its roots:

$$g(\alpha^i) = 0, \quad i = 1, 2, \dots, 2t \tag{1}$$

It also true that $g(X)$ has α^i and its conjugate as its roots (see Appendix B). On the other hand, if $\Phi_i(X)$ is the minimal polynomial of α^i then the LCM of $\Phi_1(X), \Phi_2(X), \dots, \Phi_{2t}(X)$ is the generator polynomial $g(X)$:

$$g(X) = \text{LCM}\,\{\Phi_1(X), \Phi_2(X), \dots, \Phi_{2t}(X)\} \tag{2}$$

However, and due to repetition of conjugate roots, it can be shown that the generator polynomial $g(X)$ can be formed with only the odd index minimal polynomials [4]:

$$g(X) = \text{LCM}\,\{\Phi_1(X), \Phi_3(X), \dots, \Phi_{2t-1}(X)\} \tag{3}$$

Since the degree of each minimal polynomial is m or less, the degree of $g(X)$ is at most mt. As BCH codes are cyclic codes, this means that the value of $n - k$ can be at most mt.

The Hamming codes are a particular class of BCH codes, for which the generator polynomial is $g(X) = \Phi_1(X)$. A BCH code for $t = 1$ is then a Hamming code. Since α is a primitive element of GF(2^m), then $\Phi_1(X)$ is a polynomial of degree m.

Example 4.1: Let α be a primitive element of GF(2^4), as seen in Table B.4 of the Appendix B, then $1 + \alpha + \alpha^4 = 0$. From Table B.5 (Appendix B), the minimal polynomials of α, α^3 and α^5 are, respectively,

$$\Phi_1(X) = 1 + X + X^4$$
$$\Phi_3(X) = 1 + X + X^2 + X^3 + X^4$$
$$\Phi_5(X) = 1 + X + X^2$$

A BCH code for correcting error patterns of size $t = 2$ or less, and with block length $n = 2^4 - 1 = 15$, will have the generator polynomial

$$g(X) = \text{LCM}\,\{\Phi_1(X), \Phi_3(X)\}$$

Since $\Phi_1(X)$ and $\Phi_3(X)$ are two irreducible and distinct polynomials,

$$\begin{aligned} g(X) &= \Phi_1(X)\Phi_3(X) \\ &= (1 + X + X^4)(1 + X + X^2 + X^3 + X^4) \\ &= 1 + X^4 + X^6 + X^7 + X^8 \end{aligned}$$

This is the BCH code $C_{\text{BCH}}(15, 7)$ with minimum Hamming distance $d_{\min} \geq 5$. Since the generator polynomial is of weight 5, the minimum Hamming distance of the BCH code which this polynomial generates is $d_{\min} = 5$.

In order to increase the error-correction capability to any error pattern of size $t = 3$ or less, the corresponding binary BCH code is $C_{\mathrm{BCH}}(15,\ 5)$ with minimum distance $d_{\min} \geq 7$, which can be constructed using the generator polynomial

$$\begin{aligned} g(X) &= \Phi_1(X)\Phi_3(X)\Phi_5(X) \\ &= (1 + X + X^4)(1 + X + X^2 + X^3 + X^4)(1 + X + X^2) \\ &= 1 + X + X^2 + X^4 + X^5 + X^8 + X^{10} \end{aligned}$$

This generator polynomial is of weight 7, and so it generates a BCH code of minimum Hamming distance $d_{\min} = 7$.

As a result of the definition of a linear binary block BCH code $C_{\mathrm{BCH}}(n,\ k)$ for correcting error patterns of size t or less, and with code length $n = 2^m - 1$, it is possible to affirm that any code polynomial of such a code will have $\alpha, \alpha^2, \ldots, \alpha^{2t}$ and their conjugates as its roots. This is so because any code polynomial is a multiple of the corresponding generator polynomial $g(X)$, and also of all the minimal polynomials $\Phi_1(X),\ \Phi_2(X), \ldots,\ \Phi_{2t}(X)$. Any code polynomial $c(X) = c_0 + c_1X + \ldots + c_{n-1}X^{n-1}$ of $C_{\mathrm{BCH}}(n,\ k)$ has primitive element α^i as a root:

$$c(\alpha^i) = c_0 + c_1\alpha^i + \cdots + c_{n-1}\alpha^{i(n-1)} = 0 \tag{4}$$

In matrix form,

$$(c_0,\ c_1, \ldots,\ c_{n-1}) \circ \begin{bmatrix} 1 \\ \alpha^i \\ \alpha^{2i} \\ \vdots \\ \alpha^{(n-1)i} \end{bmatrix} = \mathbf{0} \tag{5}$$

The inner product of the code vector $(c_0,\ c_1, \ldots,\ c_{n-1})$ and the vector of roots $\left(1, \alpha^i, \alpha^{2i}, \ldots,\ \alpha^{(n-1)i}\right)$ is equal to zero. The following matrix can then be formed:

$$\boldsymbol{H} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \cdots & \alpha^{n-1} \\ 1 & \alpha^2 & (\alpha^2)^2 & (\alpha^2)^3 & \cdots & (\alpha^2)^{n-1} \\ 1 & \alpha^3 & (\alpha^3)^2 & (\alpha^3)^3 & \cdots & (\alpha^3)^{n-1} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 1 & \alpha^{2t} & (\alpha^{2t})^2 & (\alpha^{2t})^3 & \cdots & (\alpha^{2t})^{n-1} \end{bmatrix} \tag{6}$$

If $\boldsymbol{c}$ is a code vector, it should be true that

$$\boldsymbol{c} \circ H^T = \mathbf{0} \tag{7}$$

From this point of view, the linear binary block BCH code $C_{\mathrm{BCH}}(n,\ k)$ is the dual row space of the matrix $\boldsymbol{H}$, and this matrix is in turn its parity check matrix. If for some i and some j, α^j is the conjugate of α^i, then $c(\alpha^j) = 0$. This means that the inner product of $\boldsymbol{c} = (c_0,\ c_1, \ldots,\ c_{n-1})$

with the ith row of $\boldsymbol{H}$ is zero, so that these rows can be omitted in the construction of the matrix $\boldsymbol{H}$, which then adopts the form

$$\boldsymbol{H} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \cdots & \alpha^{n-1} \\ 1 & \alpha^3 & (\alpha^3)^2 & (\alpha^3)^3 & \cdots & (\alpha^3)^{n-1} \\ 1 & \alpha^5 & (\alpha^5)^2 & (\alpha^5)^3 & \cdots & (\alpha^5)^{n-1} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 1 & \alpha^{2t-1} & (\alpha^{2t-1})^2 & (\alpha^{2t-1})^3 & \cdots & (\alpha^{2t-1})^{n-1} \end{bmatrix} \tag{8}$$

Each element of the matrix $\boldsymbol{H}$ is an element of GF(2^m), which can be represented as an m-component vector taken over GF(2), arranged as a column, which allows us to construct the same matrix in binary form.

Example 4.2: For the binary BCH code $C_{\mathrm{BCH}}(15,\ 7)$ of length $n = 2^4 - 1 = 15$, able to correct any error pattern of size $t = 2$ or less, and α being a primitive element of GF(2^4), the parity check matrix $\boldsymbol{H}$ is of the form

$$\boldsymbol{H} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & \alpha^7 & \alpha^8 & \alpha^9 & \alpha^{10} & \alpha^{11} & \alpha^{12} & \alpha^{13} & \alpha^{14} \\ 1 & \alpha^3 & \alpha^6 & \alpha^9 & \alpha^{12} & \alpha^0 & \alpha^3 & \alpha^6 & \alpha^9 & \alpha^{12} & \alpha^0 & \alpha^3 & \alpha^6 & \alpha^9 & \alpha^{12} \end{bmatrix}$$

which can be described in binary form by making use of Table B.4 of Appendix B:

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

4.2.1 Bounds on the Error-Correction Capability of a BCH Code: The Vandermonde Determinant

It can be shown that a given BCH code must have minimum distance $d_{\min} \geq 2t + 1$, so that its corresponding parity check matrix $\boldsymbol{H}$ has $2t + 1$ columns that sum to the zero vector. BCH codes are linear block codes, and so the minimum distance is defined by the non-zero code vector of minimum weight. Should there exist a non-zero code vector of weight $p_{\mathrm{H}} \leq 2t$, with non-zero elements $c_{j1},\ c_{j2}, \ldots,\ c_{jp_{\mathrm{H}}}$, then

$$(c_{j1},\ c_{j2}, \ldots,\ c_{jp_{\mathrm{H}}}) \circ \begin{bmatrix} \alpha^{j1} & (\alpha^2)^{j1} & \cdots & (\alpha^{2t})^{j1} \\ \alpha^{j2} & (\alpha^2)^{j2} & & (\alpha^{2t})^{j2} \\ \vdots & \vdots & & \vdots \\ \alpha^{jp_{\mathrm{H}}} & (\alpha^2)^{jp_{\mathrm{H}}} & \cdots & (\alpha^{2t})^{jp_{\mathrm{H}}} \end{bmatrix} = \mathbf{0} \tag{9}$$

By making use of $\left(\alpha^{2t}\right)^{ji} = \left(\alpha^{ji}\right)^{2t}$ and as $p_{\mathrm{H}} \leq 2t$, we obtain

$$(c_{j1},\ c_{j2}, \ldots,\ c_{jp_{\mathrm{H}}}) \circ \begin{bmatrix} \alpha^{j1} & (\alpha^{j1})^2 & \cdots & (\alpha^{j1})^{p_{\mathrm{H}}} \\ \alpha^{j2} & (\alpha^{j2})^2 & & (\alpha^{j2})^{p_{\mathrm{H}}} \\ \vdots & \vdots & & \vdots \\ \alpha^{jp_{\mathrm{H}}} & (\alpha^{jp_{\mathrm{H}}})^2 & \cdots & (\alpha^{jp_{\mathrm{H}}})^{p_{\mathrm{H}}} \end{bmatrix} = \mathbf{0} \tag{10}$$

which becomes a $p_{\mathrm{H}} \times p_{\mathrm{H}}$ matrix that fits the result indicated in equation (10) only if its determinant is zero:

$$\begin{vmatrix} \alpha^{j1} & (\alpha^{j1})^2 & \cdots & (\alpha^{j1})^{p_{\mathrm{H}}} \\ \alpha^{j2} & (\alpha^{j2})^2 & & (\alpha^{j2})^{p_{\mathrm{H}}} \\ \vdots & \vdots & & \vdots \\ \alpha^{jp_{\mathrm{H}}} & (\alpha^{jp_{\mathrm{H}}})^2 & \cdots & (\alpha^{jp_{\mathrm{H}}})^{p_{\mathrm{H}}} \end{vmatrix} = 0 \tag{11}$$

Extracting a common factor from each row, we get

$$\alpha^{(j1+j2+\cdots+jp_{\mathrm{H}})} \begin{vmatrix} 1 & \alpha^{j1} & \cdots & (\alpha^{j1})^{(p_{\mathrm{H}}-1)} \\ 1 & \alpha^{j2} & & (\alpha^{j2})^{(p_{\mathrm{H}}-1)} \\ \vdots & \vdots & & \vdots \\ 1 & \alpha^{jp_{\mathrm{H}}} & \cdots & (\alpha^{jp_{\mathrm{H}}})^{(p_{\mathrm{H}}-1)} \end{vmatrix} = 0 \tag{12}$$

This determinant is called the Vandermonde determinant, and is a non-zero determinant [4, 5]. Thus, the initial assumption that $p_{\mathrm{H}} \leq 2t$ is not valid, and the minimum Hamming distance of a binary BCH code is then equal to $2t + 1$ or more. Demonstration of the non-zero property of the Vandermonde determinant is delayed until the next chapter, on Reed–Solomon codes. The parameter $2t + 1$ is called the designed distance of a BCH code, but the actual minimum distance can be higher.

Binary BCH codes can also be designed with block lengths less than $2^m - 1$, in a similar way to that described for BCH codes of length equal to $2^m - 1$. If β is an element of order n in the GF(2^m), then n is a factor of $2^m - 1$. Let $g(X)$ be a minimum-degree binary polynomial that has $\beta,\ \beta^2, \ldots,\ \beta^{2t}$ as its roots. Let $\Phi_1(X),\ \Phi_2(X), \ldots,\ \Phi_{2t}(X)$ be minimal polynomials of $\beta,\ \beta^2, \ldots,\ \beta^{2t}$, respectively. Then

$$g(X) = \mathrm{LCM}\,\{\Phi_1(X),\ \Phi_2(X), \ldots,\ \Phi_{2t}(X)\} \tag{13}$$

Since $\beta^n = 1, \beta,\ \beta^2, \ldots,\ \beta^{2t}$ are roots of $X^n + 1$. Therefore the cyclic code generated by $g(X)$ is a code of code length n. It can be shown, in the same way as for binary BCH codes of code length $n = 2^m - 1$, that the number of parity check bits is not greater than mt, and that the minimum Hamming distance is at least $d_{\min} \geq 2t + 1$.

The above analysis provides us a more general definition of a binary BCH code [4]. If β is an element of GF(2^m) and u_0 a positive integer, then the binary BCH code with a designed minimum distance d_0 is generated by the minimum-degree generator polynomial $g(X)$ that has as its roots powers of the element β, β, $\beta^{n_0+1}, \ldots,\ \beta^{n_0+d_0-2}$, with $0 \leq i < d_0 - 1$:

$$g(X) = \mathrm{LCM}\left\{\Phi_1(X),\ \Phi_2(X), \ldots,\ \Phi_{d_0-2}(X)\right\} \tag{14}$$

Here, $\Phi_i(X)$ is the minimal polynomial of β^{u_0+i} and n_i is its order. The constructed binary BCH code has a code length equal to n:

$$n = \mathrm{LCM}\left\{n_1,\ n_2, \ldots,\ n_{d_0-2}\right\} \tag{15}$$

The designed binary BCH code has minimum distance d_0, a maximum number $m\ (d_0 - 1)$ of parity check bits, and is able to correct any error pattern of size $\left\lfloor \frac{d_0-1}{2} \right\rfloor$.

When $u_0 = 1$ then $d_0 = 2t + 1$, and if β is a primitive element of GF(2^m) then the code length of the binary BCH code is $n = 2^m - 1$. In this case the binary BCH code is said to be primitive. When $u_0 = 1$ and with $d_0 = 2t + 1$, if β is not a primitive element of GF(2^m) then the code length of the binary BCH code is not $2^m - 1$, but is equal to the order of β. In this case the binary BCH code is said to be non-primitive. The requirement that the $(d_0 - 1)$ powers of β have to be roots of the generator polynomial $g(X)$ ensures that the binary BCH code has a minimum distance of at least d_0.

4.3 Decoding of BCH Codes

The code polynomial, the error polynomial and the received polynomial are related by the following expression:

$$r(X) = c(X) + e(X) \tag{16}$$

Syndrome decoding can be used with BCH codes, as they are linear cyclic block codes. Recall that, for a given code polynomial $c(X)$, $c(\alpha^i) = 0$, and that this is equivalent to

$$\boldsymbol{c} \circ H^T = \boldsymbol{0} \tag{17}$$

Combining this expression with that used to calculate the syndrome vector,

$$\boldsymbol{S} = (s_0,\ s_1, \ldots,\ s_{2t}) = \boldsymbol{r} \circ H^T$$

and the following set of equations is obtained:

$$s_i = r(\alpha^i) = e(\alpha^i) = r_0 + r_1(\alpha^i) + \cdots + r_{n-1}(\alpha^i)^{n-1} \tag{18}$$

with $1 \leq i \leq 2t$.

These equations allow us to calculate the ith component of the syndrome vector by replacing the variable X with the root α^i in the received polynomial $r(X)$. The syndrome vector consists of elements of the Galois field GF(2^m). Another method of evaluating these elements proceeds in the following way: The received polynomial is first divided by $\Phi_i(X)$, which is the minimal polynomial corresponding to the root α^i, so that $\Phi_i(\alpha^i) = 0$, and then

$$r(X) = a_i(X)\Phi_i(X) + b_i(X) \tag{19}$$

which gives, when substituting α^i,

$$r(\alpha^i) = b_i(\alpha^i) \tag{20}$$

Example 4.3: For the binary linear cyclic block code BCH $C_{\text{BCH}}(15,\ 7)$ able to correct any error pattern of size 2 or less, and if the received vector is of the form $\boldsymbol{r} = (100000001000000)$, which in polynomial form is equal to $r(X) = 1 + X^8$, determine the syndrome vector.

The above method leads to

$$\begin{aligned}
s_1 &= r(\alpha) = 1 + \alpha^8 = \alpha^2 \\
s_2 &= r(\alpha^2) = 1 + \alpha = \alpha^4 \\
s_3 &= r(\alpha^3) = 1 + \alpha^9 = 1 + \alpha + \alpha^3 = \alpha^7 \\
s_4 &= r(\alpha^4) = 1 + \alpha^2 = \alpha^8
\end{aligned}$$

4.4 Error-Location and Error-Evaluation Polynomials

Rearranging equation (16), the code polynomial $c(X)$ is related to the received polynomial $r(X)$ and the error polynomial $e(X)$ as follows:

$$c(X) = r(X) + e(X) \tag{21}$$

All these polynomials are defined with coefficients over GF(2). Let us assume that the error vector contains τ non-zero elements, representing an error pattern of τ errors placed at positions $X^{j1},\ X^{j2}, \ldots,\ X^{j\tau}$, where $0 \leq j_1 < j_2 < \cdots < j_\tau \leq n - 1$.

The error-location number is then defined as

$$\beta_l = \alpha^{jl} \tag{22}$$

where $l = 1, 2, 3, \ldots, \tau$.

The syndrome vector components are calculated, as was stated in previous sections, by replacing the variable X in the received polynomial $r(X)$ with the roots α^i, $i = 1,\ 2, \ldots,\ 2t$. It is then true that

$$s_i = r(\alpha^i) = c(\alpha^i) + e(\alpha^i) = e(\alpha^i) \tag{23}$$

Thus, a system of $2t$ equations can be formed as follows:

$$\begin{aligned}
s_1 &= r(\alpha) = e(\alpha) = e_{j1}\beta_1 + e_{j2}\beta_2 + \cdots + e_{j\tau}\beta_\tau \\
s_2 &= r(\alpha^2) = e(\alpha^2) = e_{j1}\beta_1^2 + e_{j2}\beta_2^2 + \cdots + e_{j\tau}\beta_\tau^2 \\
&\vdots \\
s_{2t} &= r(\alpha^{2t}) = e(\alpha^{2t}) = e_{j1}\beta_1^{2t} + e_{j2}\beta_2^{2t} + \cdots + e_{j\tau}\beta_\tau^{2t}
\end{aligned} \tag{24}$$

Variables $\beta_1,\ \beta_2, \ldots,\ \beta_\tau$ are unknown. An algorithm that solves this set of equations is a decoding algorithm for a binary BCH code. The unknown variables are the positions of the errors. There are 2^k different solutions for these equations, but in the random error case only the solution with the minimum weight will be the true solution.

In order to decode a binary BCH code, it is convenient to define the following polynomials:

The error-location polynomial is defined as

$$\sigma(X) = (X - \alpha^{-j1})(X - \alpha^{-j2}) \cdots (X - \alpha^{-j\tau}) = \prod_{l=1}^{\tau} (X - \alpha^{-jl}) \tag{25}$$

and the error-evaluation polynomial is

$$W(X) = \sum_{l=1}^{\tau} e_{jl} \prod_{\substack{i=1 \\ i\neq l}}^{\tau} (X - \alpha^{-ji}) \tag{26}$$

This last polynomial is needed only for non-binary ($q \geq 3$) BCH codes, because in the binary case the error values are always 1. The error values can be calculated from

$$e_{jl} = \frac{W(\alpha^{-jl})}{\sigma'(\alpha^{-jl})} \tag{27}$$

where $\sigma'(X)$ is the derivative of the polynomial $\sigma(X)$ with respect to X. Polynomials $\sigma(X)$ and $W(X)$ are relative prime, since from the way they are defined, they do not have roots in common. Certainly, if α^{-jh} is a root of $\sigma(X)$, then

$$W(\alpha^{-jh}) = \sum_{l=1}^{\tau} e_{jl} \prod_{\substack{i=1 \\ i\neq l}}^{\tau} (\alpha^{-jh} - \alpha^{-ji}) = e_{jh} \prod_{\substack{i=1 \\ i\neq l}}^{\tau} (\alpha^{-jh} - \alpha^{-ji}) \neq 0$$

On the other hand, the derivative of the error-location polynomial is equal to

$$\sigma^{'}(X) = \sum_{l=1}^{\tau} \prod_{\substack{i=1 \\ i\neq l}}^{\tau} (X - \alpha^{-ji}) \tag{28}$$

By replacing X with the root value α^{-jh},

$$\sigma^{'}(\alpha^{-jh}) = \prod_{\substack{i=1 \\ i\neq l}}^{\tau} (\alpha^{-jh} - \alpha^{-ji}) \tag{29}$$

So from the above equations,

$$e_{jh} = \frac{W(\alpha^{-jh})}{\sigma'(\alpha^{-jh})}$$

Polynomials $\sigma(X)$ and $W(X)$ allow us to calculate the positions of the errors and to determine their values by using expression (27). Additionally, the syndrome polynomial of degree $\deg\{S(X)\} \leq 2t-1$ is defined as

$$S(X) = s_1 + s_2X + s_3X^2 + \cdots + s_{2t}X^{2t-1} = \sum_{j=0}^{2t-1} s_{j+1}X^j \tag{30}$$

If $S(X) = 0$ then the received polynomial is a code polynomial or contains an uncorrectable error pattern.

4.5 The Key Equation

There exists a relationship between the polynomials $\sigma(X)$, $S(X)$ and $W(X)$ which is called the key equation, whose solution is a decoding algorithm for a BCH code. The following theorem states this relationship:

Theorem 4.1: There exists a polynomial $\mu(X)$ such that the polynomials $\sigma(X)$, $S(X)$ and $W(X)$ fit the key equation

$$\sigma(X)S(X) = -W(X) + \mu(X)X^{2t} \tag{31}$$

This is the same as saying that

$$\{\sigma(X)S(X) + W(X)\} \bmod (X^{2t}) = 0 \tag{32}$$

In the expression that defines the syndrome polynomial $S(X)$,

$$S(X) = \sum_{j=0}^{2t-1} s_{j+1}X^j = \sum_{j=0}^{2t-1}\left(\sum_{i=1}^{\tau} e_{ji}\alpha^{ji(j+1)}\right)X^j = \sum_{i=1}^{\tau} e_{ji}\alpha^{ji}\sum_{j=0}^{2t-1}(\alpha^{ji}X)^j$$

$$S(X) = \sum_{i=1}^{\tau} e_{ji}\alpha^{ji}\frac{\left(\alpha^{ji}X\right)^{2t}-1}{\left(\alpha^{ji}X\right)-1} = \sum_{i=1}^{\tau} e_{ji}\frac{\left(\alpha^{ji}X\right)^{2t}-1}{X-\alpha^{-ji}} \tag{33}$$

and then

$$\begin{aligned}\sigma(X)S(X) &= \sum_{i=1}^{\tau} e_{ji}\frac{\left(\alpha^{ji}X\right)^{2t}-1}{X-\alpha^{-ji}}\prod_{l=1}^{\tau}(X-\alpha^{-jl}) = \sum_{i=1}^{\tau} e_{ji}\left[(\alpha^{ji}X)^{2t}-1\right]\prod_{\substack{i=1\\ i\neq l}}^{\tau}(X-\alpha^{-jl})\\ &= -\sum_{i=1}^{\tau} e_{ji}\prod_{\substack{i=1\\ i\neq l}}^{\tau}(X-\alpha^{-jl}) + \left[\sum_{i=1}^{\tau} e_{ji}\alpha^{ji^{(2t)}}\prod_{\substack{i=1\\ i\neq l}}^{\tau}(X-\alpha^{-jl})\right]X^{2t}\\ &= -W(X) + \mu(X)X^{2t}\end{aligned} \tag{34}$$

The key equation offers a decoding method for BCH codes. In order to solve this equation, the Euclidean algorithm, which applies not only to numbers but also to polynomials, is utilized

[5]. A more detailed explanation of the key equation will be developed in Chapter 5 on Reed–Solomon codes. This equation is also involved in other decoding algorithms for binary BCH codes, and for Reed–Solomon codes. One of them is the Berlekamp–Massey algorithm [8, 9], which will be described in Chapter 5.

4.6 Decoding of Binary BCH Codes Using the Euclidean Algorithm

For two given numbers A and B, the Euclidean algorithm determines the highest common factor (HCF) of these two numbers, $C = \text{HCF}(A,\ B)$. It also finds two integer numbers, or for our purpose, two polynomials S and T, such that

$$C = SA + TB \tag{35}$$

This algorithm is useful for solving the key equation that involves the polynomials

$$-\mu(X)X^{2t} + \sigma(X)S(X) = -W(X) \tag{36}$$

where X^{2t} plays the role of A and the syndrome polynomial $S(X)$ plays the role of B.

4.6.1 The Euclidean Algorithm

Let A and B be two integer numbers such that $A \geq B$, or equivalently let A and B be two polynomials such that $\deg(A) \geq \deg(B)$. The initial conditions are $r_{-1} = A$ and $r_0 = B$. In a recursive calculation, and in the ith recursion, the value r_i is obtained as the remainder of the division of r_{i-2} by r_{i-1}; that is,

$$r_{i-2} = q_i r_{i-1} + r_i \tag{37}$$

where $r_i < r_{i-1}$ or, for polynomials, $\deg(r_i) < \deg(r_{i-1})$.

The recursive equation is then

$$r_i = r_{i-2} - q_i r_{i-1} \tag{38}$$

Expressions for s_i and t_i can also be obtained as

$$r_i = s_i A + t_i B \tag{39}$$

The recursion (38) is also valid for these coefficients:

$$\begin{aligned} s_i &= s_{i-2} - q_i s_{i-1} \\ t_i &= t_{i-2} - q_i t_{i-1} \end{aligned} \tag{40}$$

Then

$$\begin{aligned} r_{-1} = A &= (1)A + (0)B \\ r_0 = B &= (0)A + (1)B \end{aligned} \tag{41}$$

The initial conditions are

$$s_{-1} = 1, \qquad t_{-1} = 0 \tag{42}$$

Example 4.4: Apply the Euclidean algorithm to the numbers $A = 112$ and $B = 54$.

$112/54 = 2$, remainder 4

$$\begin{aligned} 4 &= 112 + (-2) \times 54 \\ r_1 &= r_{-1} - q_1 r_0 \\ r_{-1} &= 112, \qquad r_0 = 54, \qquad r_1 = 4 \end{aligned}$$

$54/4 = 13$, remainder 2

$$\begin{aligned} 2 &= 54 + (-13) \times 4 \\ r_2 &= r_0 - q_2 r_1 \\ r_2 &= 2 \end{aligned}$$

$4/2 = 2$, remainder 0

Therefore, 2 is the HCF of 112 and 54.

A more suitable way of implementing this algorithm is by constructing a table like Table 4.1.

The HCF is 2 because the remainder in the next step in the table is 0. In each step of the recursion, it happens that

$$\begin{aligned} 112 &= (1) \times 112 + (0) \times 54 \\ 54 &= (0) \times 112 + (1) \times 54 \\ 4 &= (1) \times 112 + (-2) \times 54 \\ 2 &= (-13) \times 112 + (27) \times 54 \end{aligned}$$

Now Euclidean algorithm is applied to the key equation

$$-\mu(X)X^{2t} + \sigma(X)S(X) = -W(X)$$

The polynomials involved are X^{2t} and $S(X)$, and the ith recursion is of the form

$$r_i(X) = s_i(X)X^{2t} + t_i(X)S(X) \tag{43}$$

Table 4.1 Euclidean algorithm for evaluating the HCF of two integer numbers

i	$r_i = r_{i-2} - q_i r_{i-1}$	q_i	$s_i = s_{i-2} - q_i s_{i-1}$	$t_i = t_{i-2} - q_i t_{i-1}$
−1	112	–	1	0
0	54	–	0	1
1	4	2	1	−2
2	2	13	−13	27

Multiplying equation (43) by λ, we obtain

$$\lambda r_i(X) = \lambda s_i(X)X^{2t} + \lambda t_i(X)S(X) = -W(X) = -\mu(X)X^{2t} + \sigma(X)S(X) \tag{44}$$

where

$$\deg(r_i(X)) \leq t + 1. \tag{45}$$

Thus,

$$\begin{aligned} W(X) &= -\lambda r_i(X) \\ \sigma(X) &= \lambda t_i(X) \end{aligned} \tag{46}$$

where λ is a constant that makes the resulting polynomial be a monic polynomial.

Example 4.5: For the binary linear cyclic block BCH code $C_{\mathrm{BCH}}(15, 7)$ with $t = 2$ and for the received vector $\boldsymbol{r} = (100000001000000)$, which in polynomial form is equal to $r(X) = 1 + X^8$ (from Example 4.3), determine, by using the Euclidean algorithm, the decoded code polynomial.

The syndrome vector components were calculated in Example 4.3:

$$\begin{aligned} s_1 &= r(\alpha) = \alpha^2 \\ s_2 &= r(\alpha^2) = \alpha^4 \\ s_3 &= r(\alpha^3) = \alpha^7 \\ s_4 &= r(\alpha^4) = \alpha^8 \end{aligned}$$

Therefore the syndrome polynomial is

$$S(X) = \alpha^8 X^3 + \alpha^7 X^2 + \alpha^4 X + \alpha^2$$

Note that while operating over the extended Galois field GF(q), where $q = 2^m$, the additive inverse of any element of that field is that same element (see Appendix B), and the minus signs in equations (36), (44) and (46) convert into plus signs. The Euclidean algorithm is applied by constructing Table 4.2.

When the degree of the polynomial in column $r_i(X)$ is lower than the degree of the polynomial in column $t_i(X)$, the recursion is halted. In this case,

$$\begin{aligned} r_i(X) &= \alpha^5 \\ t_i(X) &= \alpha^{11}X^2 + \alpha^5 X + \alpha^3 \end{aligned}$$

Table 4.2 Euclidean algorithm for the key equation, Example 4.5

i	$r_i = r_{i-2} - q_i r_{i-1}$	q_i	$t_i = t_{i-2} - q_i t_{i-1}$
-1	$X^{2t} = X^4$	–	0
0	$S(X) = \alpha^8 X^3 + \alpha^7 X^2 + \alpha^4 X + \alpha^2$	–	1
1	$\alpha^4 X^2 + \alpha^{13} X + \alpha^8$	$\alpha^7 X + \alpha^6$	$\alpha^7 X + \alpha^6$
2	α^5	$\alpha^4 X + \alpha^8$	$\alpha^{11} X^2 + \alpha^5 X + \alpha^3$

The polynomial $t_i(X)$ is multiplied by an element $\lambda \in \mathrm{GF}(2^4)$, which is conveniently selected to convert this polynomial into a monic polynomial. This value of λ is $\lambda = \alpha^4$. Therefore,

$$W(X) = -\lambda r_i(X) = \lambda r_i(X) = \alpha^4\alpha^5 = \alpha^9$$

and

$$\sigma(X) = \lambda t_i(X) = \alpha^4\left(\alpha^{11}X^2 + \alpha^5 X + \alpha^3\right) = X^2 + \alpha^9 X + \alpha^7$$

The following step consists of substituting in turn all the elements of the corresponding Galois field in the error-location polynomial, in order to determine its roots. This procedure is known as the Chien search [6]. Thus, variable X in the error-location polynomial $\sigma(X)$ is replaced with $1, \alpha, \alpha^2, \ldots, \alpha^{n-1}$, where $n = 2^m - 1$. Since $\alpha^n = 1$ and $\alpha^{-h} = \alpha^{n-h}$, then if α^h is a root of the error-location polynomial $\sigma(X)$, α^{n-h} is the corresponding error-location number, which determines that bit r_{n-h} is in error. In the binary case this information is enough to do the error correction, because if the bit r_{n-h} is in error, then its value is inverted to correct it.

Performing the above Chien search, the roots of the error-location polynomials are found to be $\alpha^{-j1} = 1$ and $\alpha^{-j2} = \alpha^7$. Then, and since

$$\alpha^0 = \alpha^{-j1} = \alpha^{-0}$$
$$j_1 = 0$$

and

$$\alpha^7 = \alpha^{-j2} = \alpha^{-8}$$
$$j_2 = 8$$

Errors are located in positions $j_1 = 0$ and $j_2 = 8$. Values of the errors are determined by evaluating the derivative of the error-location polynomial

$$\sigma^{'}(X) = \alpha^9$$

Then, and by applying expression (27),

$$e_{j1} = \frac{W(\alpha^{-j1})}{\sigma'(\alpha^{-j1})} = \frac{W(\alpha^0)}{\sigma'(\alpha^0)} = \frac{\alpha^9}{\alpha^9} = 1$$
$$e_{j2} = \frac{W(\alpha^{-j2})}{\sigma'(\alpha^{-j2})} = \frac{W(\alpha^7)}{\sigma'(\alpha^7)} = \frac{\alpha^9}{\alpha^9} = 1$$

The result is obvious as this BCH code is a binary code, and so errors are always of value 1. The error polynomial is therefore

$$e(X) = X^0 + X^8 = 1 + X^8$$

After the correction of the received polynomial, done by using equation (21), the decoded polynomial is the zero polynomial, that is, the all-zero vector.

Bibliography and References

[1] Bose, R. C. and Ray-Chaudhuri, D. K., “On a class of error correcting binary group codes,” *Inf. Control.*, vol. 3, pp. 68–79, March 1960.

[2] Hocquenghem, A., “Codes correcteurs d’erreurs,” *Chiffres*, vol. 2, pp. 147–156, 1959.

[3] Reed, I. S. and Solomon, G., “Polynomial codes over certain finite fields,” *J. Soc. Ind. Appl. Math.*, vol. 8, pp. 300–304, 1960.

[4] Lin, S. and Costello, D. J., Jr., *Error Control Coding: Fundamentals and Applications*, Prentice Hall, Englewood Cliffs, New Jersey, 1983.

[5] Blaum, M., *A Course on Error Correcting Codes*, 2001.

[6] Chien, R. T., “Cyclic decoding procedure for the Bose–Chaudhuri–Hocquenghem codes,” *IEEE Tans. Inf. Theory*, vol. IT-10, pp. 357–363, October 1964.

[7] Forney, G. D., Jr., “On decoding BCH codes,” *IEEE Trans. Inf. Theory*, vol. IT-11, pp. 59–557, October 1965.

[8] Berlekamp, E. R., “On decoding binary Bose–Chaudhuri–Hocq uenghem codes,” *IEEE Trans. Inf. Theory*, vol. IT-11, pp. 577–580, October 1965.

[9] Massey, J. L., “Step-by-step decoding of the Bose–Chaudhuri–Hocq uenghem codes,” *IEEE Trans. Inf. Theory*, vol. IT-11, pp. 580–585, October 1965.

[10] Sloane, N. J. A. and Peterson, W. W., *The Theory of Error-Correcting Codes*, North-Holland, Amsterdam, The Netherlands, 1998.

[11] Berlekamp, E. R., *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.

[12] Perterson, W. W. and Wledon, E. J., Jr., *Error-Correcting Codes*, 2nd Edition, MIY Press, Cambridge, Massachusetts, 1972.

[13] Wicker, S. B. and Bhargava, V. K., *Reed–Solomon Codes and Their Applications*, IEEE Press, New York, 1994.

[14] Shankar, P., “On BCH codes over arbitrary integer rings,” *IEEE Trans. Inf. Theory*, vol. IT-25, pp. 480–483, 965–975, July 1979.

Problems

4.1 Verify that the polynomial $p(X) = 1 + X^2 + X^5$ is an irreducible polynomial. It can also be a primitive polynomial. What are the conditions for this to happen?

4.2 Construct the Galois field GF(2^5) generated by $p(X) = 1 + X^2 + X^5$, showing a table with the polynomial and binary representations of its elements.

4.3 Determine the minimal polynomials of the elements of the Galois field GF(2^5) constructed in Problem 4.2.

4.4 Determine the generator polynomial of the binary BCH code $C_{\mathrm{BCH}}(31, 16)$ able to correct error patterns of size $t = 3$ or less.

4.5 Determine the generator polynomial of a binary BCH code of code length $n = 31$ able to correct error patterns of size $t = 2$ or less. Also, determine the value of k and the minimum Hamming distance of the code.

4.6 A binary cyclic BCH code $C_{\text{BCH}}(n, k)$ has code length $n = 15$ and generator polynomial $g(X) = (X + 1)(1 + X + X^4)(1 + X + X^2 + X^3 + X^4)$.
(a) What is the minimum Hamming distance of the code?
(b) Describe a decoding algorithm for this code, and demonstrate its operation by means of an example.
Note: Use the Galois field GF(2^4) shown in Appendix B, Table B.4.

4.7 (a) Show that the shortest binary cyclic BCH code with the generator polynomial $g(X) = (1 + X + X^4)(1 + X + X^2 + X^3 + X^4)$ has code length $n = 15$ and minimum Hamming distance $d_{\text{min}} = 5$.
(b) Describe the Meggitt or an algebraic decoding method for the above code.
(c) Use the decoding method you have described to show how errors in the first two positions of a received vector would be corrected.

4.8 Use the BCH bound to show that the minimum Hamming distance of the cyclic code with block length $n = 7$ and $g(X) = (X + 1)(1 + X + X^3)$ is 4.
(a) What is the minimum Hamming distance if $n = 14$ and why?

4.9 The binary cyclic BCH code $C_{\text{BCH}}(15, 7)$ is able to correct error patterns of size $t = 2$ or less, and has a generator polynomial of the form $g(X) = (1 + X + X^4)(1 + X + X^2 + X^3 + X^4) = 1 + X^4 + X^6 + X^7 + X^8$, which operates over the Galois field GF(2^4) (Appendix B, Table B.4).

Assume that the received vector is $\boldsymbol{r} = (100000001000000)$ and decode it using the Euclidean algorithm.

4.10 Show that for a double-error-correcting cyclic code,

$$s_3 = s_1^2 + s_1^2\beta_l + s_1\beta_l^2 \quad \text{where } \beta_l = \alpha^{j_l}$$

and hence find the errors in the received vector $\boldsymbol{r} = (000100111111011)$, given that the transmitted vector is from the cyclic BCH code $C_{\text{BCH}}(15, 7)$ generated by $g(X) = (1 + X + X^4)(1 + X + X^2 + X^3 + X^4)$.

4.11 For the binary BCH code $C_{\text{BCH}}(31, 21)$, obtained in Problem 4.5, perform the decoding of the received polynomials:
(a) $r_1(X) = X^7 + X^{30}$;
(b) $r_2(X) = 1 + X^{17} + X^{28}$.

■

5

Reed–Solomon Codes

Reed–Solomon (RS) codes are a class of linear, non-binary, cyclic block codes [1]. This class is a subfamily of the family of the linear, non-binary, cyclic BCH codes which form a generalization over the Galois field GF(q) of the binary BCH codes introduced in Chapter 4 [2, 3]. Here q is a power of a prime number p_{prime}, $q = p_{\mathrm{prime}}^m$, where m is a positive integer. These non-binary BCH codes are usually called q-ary codes since they operate over the alphabet of q elements of the Galois field GF(q), with $q > 2$. In this sense these codes are different from binary codes, which have elements taken from the binary field GF(2). This is why q-ary codes are also called non-binary codes. All the concepts and properties verified for binary BCH codes are also valid for these non-binary codes.

5.1 Introduction

A block code $C_{\mathrm{b}}(n, k)$ defined over GF(q) is a subspace of dimension k of the vector space V_n of vectors of n components, where the components are elements of GF(q). A cyclic q-ary code over GF(q) is generated by a generator polynomial of degree $n - k$ whose coefficients are elements of GF(q). This generator polynomial is a factor of $X^n - 1$. The encoding of q-ary codes is similar to the encoding of binary BCH codes.

Binary BCH codes can be generalized to operate over the Galois field GF(q):
For any two positive integer numbers v and t, there exists a q-ary code of code length $n = q^v - 1$, able to correct any error pattern of size t or less, which is constructed with at least $2vt$ parity check elements. If α is a primitive element of the Galois field GF(q^v), the generator polynomial of a q-ary BCH code able to correct any error pattern of size t or less is the minimum-degree polynomial with coefficients from GF(q) that has $\alpha, \alpha^2, \ldots, \alpha^{2t}$ as its roots. If $\phi_i(X)$ is the minimal polynomial of α^i [2], then

$$g(X) = \mathrm{LCM}\,\{\phi_1(X), \phi_2(X), \ldots, \phi_{2t}(X)\} \tag{1}$$

The degree of each minimal polynomial is v or less. The degree of the generator polynomial $g(X)$ is at most $2vt$, which means that the maximum number of parity check elements is $2vt$. In the case of $q = 2$, the definition corresponds to the binary BCH codes presented in Chapter 4.

Essentials of Error-Control Coding Jorge Castiñeira Moreira and Patrick Guy Farrell

A special subfamily of q-ary BCH codes is obtained when $v = 1$, and they are the so-called Reed–Solomon codes, named in honour of their discoverers.

An RS code $C_{\mathrm{RS}}(n, k)$ able to correct any error pattern of size t or less is defined over the Galois field GF(q), and it has as parameters

Code length	$n = q - 1$
Number of parity check elements	$n - k = 2t$
Minimum distance	$d_{\min} = 2t + 1$
Error-correction capability	t element errors per code vector

If α is a primitive element of GF(q) then $\alpha^{q-1} = 1$. An RS code $C_{\mathrm{RS}}(n, k)$ of length $n = q - 1$ and dimension k is the linear, cyclic, block RS code generated by the polynomial

$$\begin{aligned} g(X) &= (X - \alpha)\left(X - \alpha^2\right)\cdots\left(X - \alpha^{n-k}\right) \\ &= (X - \alpha)\left(X - \alpha^2\right)\cdots\left(X - \alpha^{2t}\right) \\ &= g_0 + g_1 X + g_2 X^2 + \cdots + g_{2t-1}X^{2t-1} + g_{2t}X^{2t} \end{aligned} \tag{2}$$

The difference with respect to the definition given for a binary BCH code is that coefficients g_i of this generator polynomial belong to the extended Galois field GF(q). On the other hand, the minimal polynomials are of the simplest form, $\Phi_i(X) = X - \alpha^i$.

The most useful codes, in practice, of this class are defined over Galois fields of the form GF(2^m), that is, finite fields with elements that have a binary representation in the form of a vector with elements over GF(2). Each element α^i is root of the minimal polynomial $X - \alpha^i$ so that $X - \alpha^i$ is a factor of $X^n - 1$. As a consequence of this, $g(X)$ is also a factor of $X^n - 1$ and hence it is the generator polynomial of a cyclic code with elements taken from GF(2^m). Since operations are defined over GF(2^m), where the additive inverse of a given element is that same element, addition and subtraction are the same operation, and so to say that α^i is a root of $X - \alpha^i$ is the same as to say that it is a root of $X + \alpha^i$.

An RS code can be equivalently defined as the set of code polynomials $c(X)$ over GF(q) of degree $\deg\{c(X)\} \leq n - 1$ that have $\alpha, \alpha^2, \ldots, \alpha^{n-k}$ as their roots [3]. Therefore $c(X) \in C_{\mathrm{RS}}$ if and only if

$$c(\alpha) = c(\alpha^2) = c(\alpha^3) = \cdots = c(\alpha^{2t}) = 0 \quad \text{where} \deg\{c(X)\} \leq n - 1 \tag{3}$$

This definition is in agreement with the fact that the corresponding generator polynomial has $\alpha, \alpha^2, \ldots, \alpha^{n-k}$ as its roots, and that any code polynomial is generated by multiplying a given message polynomial by the generator polynomial. Then if

$$c(X) = c_0 + c_1 X + \cdots + c_{n-1}X^{n-1} \in C_{\mathrm{RS}},$$

it is true that

$$c(\alpha^i) = c_0 + c_1\alpha^i + \cdots + c_{n-1}(\alpha^i)^{n-1} = c_0 + c_1\alpha^i + \cdots + c_{n-1}\alpha^{(n-1)i} = 0, \tag{4}$$
$$1 \leq i \leq n - k$$

5.2 Error-Correction Capability of RS Codes: The Vandermonde Determinant

Equation (4) states that the inner product between the code vector $(c_0, c_1, \dots, c_{n-1})$ and the root vector $(1, \alpha^i, \alpha^{2i}, \dots, \alpha^{(n-1)i})$ is zero, for $1 \le i \le n-k$. This condition can be summarized in the form of a matrix, which is the parity check matrix $\boldsymbol{H}$:

$$\boldsymbol{H} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \cdots & \alpha^{n-1} \\ 1 & \alpha^2 & (\alpha^2)^2 & (\alpha^2)^3 & \cdots & (\alpha^2)^{n-1} \\ 1 & \alpha^3 & (\alpha^3)^2 & (\alpha^3)^3 & \cdots & (\alpha^3)^{n-1} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 1 & \alpha^{n-k} & (\alpha^{n-k})^2 & (\alpha^{n-k})^3 & \cdots & (\alpha^{n-k})^{n-1} \end{bmatrix} \tag{5}$$

In this matrix any set of $n-k$ or fewer columns is linearly independent. In order to show this, consider a set of $n-k$ columns $i_1, i_2, \dots, i_{n-k}$ with $0 \le i_1 < i_2 < \cdots < i_{n-k} \le n-1$. It will be convenient to use the modified notation $\alpha_j = \alpha^{i_j}$, where $1 \le j \le n-k$. The set of columns $i_1, i_2, \dots, i_{n-k}$ is linearly independent if and only if the following determinant is zero [2]:

$$\begin{vmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_{n-k} \\ (\alpha_1)^2 & (\alpha_2)^2 & \cdots & (\alpha_{n-k})^2 \\ \vdots & \vdots & & \vdots \\ (\alpha_1)^{n-k} & (\alpha_2)^{n-k} & \cdots & (\alpha_{n-k})^{n-k} \end{vmatrix} \neq 0 \tag{6}$$

This determinant can be converted into

$$\alpha_1\alpha_2 \dots \alpha_{n-k} \begin{vmatrix} 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_{n-k} \\ \vdots & \vdots & & \vdots \\ (\alpha_1)^{n-k-1} & (\alpha_2)^{n-k-1} & \cdots & (\alpha_{n-k})^{n-k-1} \end{vmatrix} \tag{7}$$

$$= \alpha_1\alpha_2 \dots \alpha_{n-k} V(\alpha_1, \alpha_2, \dots, \alpha_{n-k}) \neq 0$$

where $V(\alpha_1, \alpha_2, \dots, \alpha_{n-k})$ is the so-called Vandermonde determinant. Since $\alpha_i \neq 0$, it will be sufficient to prove that if $V(\alpha_1, \alpha_2, \dots, \alpha_{n-k}) \neq 0$, then the $n-k$ columns of the matrix $\boldsymbol{H}$ are linearly independent. For $n-k=2$ [3],

$$\begin{vmatrix} 1 & 1 \\ \alpha_1 & \alpha_2 \end{vmatrix} = \alpha_2 - \alpha_1 \neq 0$$

If the polynomial

$$f(X) = \begin{vmatrix} 1 & 1 & \cdots & 1 \\ X & \alpha_2 & \cdots & \alpha_{n-k} \\ \vdots & \vdots & & \vdots \\ X^{n-k-1} & (\alpha_2)^{n-k-1} & \cdots & (\alpha_{n-k})^{n-k-1} \end{vmatrix} \tag{8}$$

is constructed, it can be seen that α_i is root of that polynomial for $2 \leq i \leq n-k$, since two columns of this determinant will be equal when X is replaced by α_i, resulting in $f(\alpha_i) = 0$. Therefore,

$$f(X) = c(X - \alpha_2)(X - \alpha_3) \cdots (X - \alpha_{n-k}) = (-1)^{n-k-1}\, c \prod_{j=2}^{n-k} (\alpha_j - X) \tag{9}$$

where c is a constant. Now

$$f(\alpha_1) = V(\alpha_1, \alpha_2, \ldots, \alpha_{n-k}) \tag{10}$$

and, by using determinant properties, the constant c can be obtained:

$$\begin{aligned} c &= (-1)^{n-k-1} \begin{vmatrix} 1 & 1 & \cdots & 1 \\ \alpha_2 & \alpha_3 & \cdots & \alpha_{n-k} \\ \vdots & \vdots & & \vdots \\ (\alpha_2)^{n-k-2} & (\alpha_3)^{n-k-2} & \cdots & (\alpha_{n-k})^{n-k-2} \end{vmatrix} \\ &= (-1)^{n-k-1}\, V(\alpha_2, \alpha_3, \ldots, \alpha_{n-k}) \end{aligned} \tag{11}$$

Then, by induction, the value of c can be finally determined as

$$c = (-1)^{n-k-1} \prod_{2 \leq i < j \leq n-k}^{n-k} (\alpha_j - \alpha_i) \tag{12}$$

By replacing X with α_1 and including the value of the constant c from the above equation, we obtain

$$V(\alpha_1, \alpha_2, \ldots, \alpha_{n-k}) = \prod_{1 \leq i < j \leq n-k} (\alpha_j - \alpha_i) \tag{13}$$

Since $\alpha_i = \alpha^{j_i}$ and as α is a primitive element of GF(q), then it is also true that $\alpha_i \neq \alpha_j$ if $i \neq j$ so that $V(\alpha_1, \alpha_2, \ldots, \alpha_{n-k}) \neq 0$. As a consequence of this, there are $n-k+1$ columns that are linearly dependent; that is, there are $n-k+1$ columns that added result in the all-zero vector. Therefore the minimum distance of an RS code $C_{\mathrm{RS}}(n, k)$ is equal to $d_{\min} = n - k + 1 = 2t + 1$.

Another conclusion derived from the above analysis is that RS codes are maximum separable distance codes, since their minimum distance meets the Singleton bound $d_{\min} \leq n - k + 1$ with equality.

Example 5.1: Construct the Galois field GF(2^3) generated by the irreducible polynomial $p_i(X) = 1 + X^2 + X^3$.

If α is a primitive element of GF(2^3), then $p_i(\alpha) = 1 + \alpha^2 + \alpha^3 = 0$, or $\alpha^3 = 1 + \alpha^2$. The Galois field GF(2^3) can be constructed, utilizing the above expression. Thus, for instance, $\alpha^4 = \alpha\alpha^3 = \alpha(1 + \alpha^2) = \alpha + \alpha^3 = 1 + \alpha + \alpha^2$. Table 5.1 shows the polynomial and binary forms of the elements of the Galois field GF(2^3) generated by the irreducible polynomial $p_i(X) = 1 + X^2 + X^3$.

Table 5.1 Galois field GF(2^3) generated by $p_i(X) = 1 + X^2 + X^3$

Exponential form	Polynomial form	Binary form
0	0	0 0 0
1	1	1 0 0
α	α	0 1 0
α^2	α^2	0 0 1
α^3	$1+\alpha^2$	1 0 1
α^4	$1+\alpha+\alpha^2$	1 1 1
α^5	$1+\alpha$	1 1 0
α^6	$\alpha+\alpha^2$	0 1 1

If this Galois field is compared with that introduced in Appendix B, generated by $p_i(X) = 1 + X + X^3$, then it is seen that the differences are in the polynomial and binary representations for each element.

Example 5.2: Construct the generator polynomial of an RS code $C_{\text{RS}}(7, 5)$ that operates over the Galois field GF(2^3), which has $p_i(X) = 1 + X^2 + X^3$ as its primitive polynomial.

Since this RS code has $n - k = 2$, it is able to correct any error pattern of size $t = 1$.

The corresponding generator polynomial is

$$g(X) = (X + \alpha)(X + \alpha^2) = X^2 + (\alpha + \alpha^2)X + \alpha^3 = X^2 + \alpha^6 X + \alpha^3$$

5.3 RS Codes in Systematic Form

An RS code generated by a given generator polynomial of the form of (2) is a linear and cyclic block RS code $C_{\text{RS}}(n, n - 2t)$ consisting of code polynomials $c(X)$ of degree $n - 1$ or less. All these polynomials are formed with coefficients that are elements of the Galois field GF(2^m). Code polynomials are multiples of the generator polynomial $g(X)$, thus containing all its roots.

A message polynomial is of the form

$$m(X) = m_0 + m_1 X + \cdots + m_{k-1} X^{k-1} \tag{14}$$

This message polynomial is also formed with coefficients that are elements of GF(2^m). The systematic form for these codes is obtained in the same way as for binary BCH codes, that is, by obtaining the remainder $p(x)$ of the division of $X^{n-k}m(X)$ by $g(X)$:

$$X^{n-k}m(X) = q(X)g(X) + p(X) \tag{15}$$

Example 5.3: Determine the code vector in systematic form for the RS code of Example 5.2 when the message to be transmitted is (001 101 111 010 011).

The message is converted into its polynomial form:

$$m(X) = \alpha^2 + \alpha^3 X + \alpha^4 X^2 + \alpha X^3 + \alpha^6 X^4$$

Then

$$\begin{aligned}X^{n-k}m(X) &= X^{7-5}m(X)\\ &= X^2\left[\alpha^2+\alpha^3X+\alpha^4X^2+\alpha X^3+\alpha^6X^4\right]\\ &= \alpha^2X^2+\alpha^3X^3+\alpha^4X^4+\alpha X^5+\alpha^6X^6\end{aligned}$$

In systematic form,

$$\begin{array}{rl}
\alpha^6X^6+\alpha X^5+\alpha^4X^4+\alpha^3X^3+\alpha^2X^2 & |X^2+\alpha^6X+\alpha^3 \\ \cline{2-2}
\underline{\alpha^6X^6+\alpha^5X^5+\alpha^2X^4} & \alpha^6X^4+X^3+\alpha^3X^2+\alpha^2X \\
X^5+\alpha^5X^4+\alpha^3X^3+\alpha^2X^2 & \\
\underline{X^5+\alpha^6X^4+\alpha^3X^3} & \\
\alpha^3X^4+0X^3+\alpha^2X^2 & \\
\underline{\alpha^3X^4+\alpha^2X^3+\alpha^6X^2} & \\
\alpha^2X^3+\alpha X^2 & \\
\underline{\alpha^2X^3+\alpha X^2+\alpha^5X} & \\
p(X)=\alpha^5X &
\end{array}$$

The code polynomial is then

$$c(X)=\alpha^5X+\alpha^2X^2+\alpha^3X^3+\alpha^4X^4+\alpha X^5+\alpha^6X^6$$

which in vector form is equal to

$$(000\ 110\ 001\ 101\ 111\ 010\ 011)$$

It can be verified that this is a code vector or code polynomial by replacing the variable X in $c(X)$ with the roots α and α^2, to see that the result is zero:

$$\begin{aligned}c(\alpha) &= \alpha^6+\alpha^4+\alpha^6+\alpha+\alpha^6+\alpha^5=\alpha+\alpha^5+\alpha^4+\alpha^6=1+1=0\\ c(\alpha^2) &= 1+\alpha^6+\alpha^2+\alpha^5+\alpha^4+\alpha^4=1+\alpha^6+\alpha^2+\alpha^5=\alpha^4+\alpha^4=0\end{aligned}$$

5.4 Syndrome Decoding of RS Codes

After encoding a given message, the code polynomial

$$c(X)=c_0+c_1X+\cdots+c_{n-1}X^{n-1}$$

is transmitted and affected by noise, and converted into a received polynomial of the form

$$r(X)=r_0+r_1X+\cdots+r_{n-1}X^{n-1}$$

which is related to the error polynomial $e(X) = e_0 + e_1X + \cdots + e_{n-1}X^{n-1}$ and the code polynomial $c(X)$ as follows:

$$r(X) = c(X) + e(X) \tag{16}$$

Let us assume that the error polynomial contains τ non-zero elements, which means that during transmission τ errors occurred, placed at positions $X^{j_1}, X^{j_2}, \ldots, X^{j_\tau}$, where $0 \leq j_1 < j_2 < \cdots < j_\tau \leq n - 1$. In the case of a non-binary code defined over GF(2^m), all the vectors involved, the code vector, the received vector and the error vector, are formed from elements of that field. In this case the error-correction procedure requires not only knowledge of the position of the errors but also their values in GF(2^m).

Let us again define the error-location number as

$$\beta_i = \alpha^{j_i} \quad \text{where } i = 1, 2, 3, \ldots, \tau \tag{17}$$

The syndrome calculation is performed in the same way as for the binary BCH codes, and it implies the evaluation of the received polynomial $r(X)$, replacing the variable X by the roots α^i, $i = 1, 2, \ldots, 2t$. Once again

$$r(\alpha^i) = c(\alpha^i) + e(\alpha^i) = e(\alpha^i) \tag{18}$$

A system of equations is then formed with these expressions:

$$\begin{aligned}
s_1 &= r(\alpha) = e(\alpha) = e_{j_1}\beta_1 + e_{j_2}\beta_2 + \cdots + e_{j_\tau}\beta_\tau \\
s_2 &= r(\alpha^2) = e(\alpha^2) = e_{j_1}\beta_1^2 + e_{j_2}\beta_2^2 + \cdots + e_{j_\tau}\beta_\tau^2 \\
&\vdots \\
s_{2t} &= r(\alpha^{2t}) = e(\alpha^{2t}) = e_{j_1}\beta_1^{2t} + e_{j_2}\beta_2^{2t} + \cdots + e_{j_\tau}\beta_\tau^{2t}
\end{aligned} \tag{19}$$

In the particular case of RS codes $C_{\mathrm{RS}}(n, n-2)$ that are able to correct any error pattern of size $t = 1$, syndrome calculation involves only two equations, which leads to a rather simple solution

$$\begin{aligned}
s_1 &= r(\alpha) = e(\alpha) = e_{j_1}\beta_1 = e_{j_1}\alpha^{j_1} \\
s_2 &= r(\alpha^2) = e(\alpha^2) = e_{j_1}\beta_1^2 = e_{j_1}\alpha^{2j_1}
\end{aligned} \tag{20}$$

Then

$$\begin{aligned}
\frac{e_{j1}\alpha^{2j_1}}{e_{j1}\alpha^{j1}} &= \alpha^{j_1} = \frac{s_2}{s_1} \\
s_1 &= e_{j_1}\alpha^{j_1} = e_{j_1}\frac{s_2}{s_1} \\
e_{j_1} &= \frac{s_1^2}{s_2}
\end{aligned} \tag{21}$$

This system has two equations and is able to determine two unknown variables, which are the error location and the error value.

Example 5.4: For the case of the RS code of Example 5.3, assume that the received vector is $\boldsymbol{r} = (000\ 101\ 111\ 111\ 011) = (0\ \alpha^5\ \alpha^2\ \alpha^3\ \alpha^4\ \alpha^4\ \alpha^6)$. Determine the position and value of the single error that happened during transmission, and the corrected code polynomial.

$$s_1 = r(\alpha) = \alpha^6 + \alpha^4 + \alpha^6 + \alpha + \alpha^2 + \alpha^5 = \alpha^2 + \alpha^6 = \alpha$$
$$s_2 = r(\alpha^2) = 1 + \alpha^6 + \alpha^2 + \alpha^5 + 1 + \alpha^4 = 1 + \alpha^4 = \alpha^6$$

$$\alpha^{j_1} = \frac{s_2}{s_1} = \frac{\alpha^6}{\alpha} = \alpha^5$$

$$e_{j_1} = \frac{s_1^2}{s_2} = \frac{\alpha^2}{\alpha^6} = \alpha^{-4} = \alpha^3$$

Hence the error-location polynomial is

$$e(X) = \alpha^3 X^5$$

$$\begin{aligned} c(X) &= e(X) + r(X) \\ &= \alpha^5 X + \alpha^2 X^2 + \alpha^3 X^3 + \alpha^4 X^4 + (\alpha^4 + \alpha^3) X^5 + \alpha^6 X^6 \\ &= \alpha^5 X + \alpha^2 X^2 + \alpha^3 X^3 + \alpha^4 X^4 + \alpha X^5 + \alpha^6 X^6 \end{aligned}$$

5.5 The Euclidean Algorithm: Error-Location and Error-Evaluation Polynomials

As a result of the non-binary nature of these codes, there is a need to determine both the locations and the values of the errors, so that the following two polynomials are defined [2, 3].

The error-location polynomial is

$$\sigma(X) = (X - \alpha^{-j_1})(X - \alpha^{-j_2}) \cdots (X - \alpha^{-j_\tau}) = \prod_{l=1}^{\tau} (X - \alpha^{-j_l}) \tag{22}$$

and the error-evaluation polynomial is

$$W(X) = \sum_{l=1}^{\tau} e_{j_l} \prod_{\substack{i=1 \\ i \neq l}}^{\tau} (X - \alpha^{-j_i}) \tag{23}$$

It can be shown that the error value is equal to

$$e_{j_l} = \frac{W(\alpha^{-j_l})}{\sigma'(\alpha^{-j_l})} \tag{24}$$

where $\sigma'(X)$ is the derivative of the error-location polynomial $\sigma(X)$.

Polynomials $\sigma(X)$ and $W(X)$ are relative prime, since, in the way they are defined, they do not have any root in common. Therefore, if α^{-j_h} is a root of $\sigma(X)$, then

$$W(\alpha^{-j_h}) = \sum_{l=1}^{\tau} e_{j_l} \prod_{\substack{i=1\\ i\neq l}}^{\tau} (\alpha^{-j_h} - \alpha^{-j_i}) = e_{jh} \prod_{\substack{i=1\\ i\neq l}}^{\tau} (\alpha^{-j_h} - \alpha^{-j_i}) \neq 0$$

On the other hand, the derivative with respect to X of the polynomial $\sigma(X)$, $\sigma'(X)$, is equal to

$$\sigma'(X) = \sum_{l=1}^{\tau} \prod_{\substack{i=1\\ i\neq l}}^{\tau} (X - \alpha^{-j_i}) \tag{25}$$

By substituting for the variable X the value of the root α^{-jh}, we get

$$\sigma'(\alpha^{-j_h}) = \prod_{\substack{i=1\\ i\neq h}}^{\tau} (\alpha^{-j_h} - \alpha^{-j_i}) \tag{26}$$

Thus, from the above equations,

$$e_{j_h} = \frac{W(\alpha^{-j_h})}{\sigma'(\alpha^{-j_h})}$$

Roots of the polynomial $\sigma(X)$ determine the positions of the errors, and then polynomial $W(X)$ and equation (24) can be used to determine the values of the errors.

An additional polynomial, $S(X)$, called the syndrome polynomial, whose degree is $\deg\{S(X)\} \leq n-k-1$, is also defined as

$$S(X) = s_1 + s_2X + s_3X^2 + \cdots + s_{n-k}X^{n-k-1} = \sum_{j=0}^{n-k-1} s_{j+1}X^j \tag{27}$$

If $S(X) = 0$ then the corresponding received polynomial is a code polynomial belonging to the RS code, i.e., $r(X) \in C_{\mathrm{RS}}(n, k)$, unless an uncorrectable error pattern has occurred.

As in the case of binary BCH codes, there is a relationship among the polynomials $\sigma(X)$, $S(X)$ and $W(X)$, which is called the key equation, whose solution is a decoding method for an RS code. The following theorem states this relationship:

Theorem 5.1: There exists a polynomial $\mu(X)$, such that polynomials $\sigma(X)$, $S(X)$ and $W(X)$ satisfy the key equation

$$\sigma(X)S(X) = -W(X) + \mu(X)X^{n-k} \tag{28}$$

This theorem has already been demonstrated in Chapter 4, for binary BCH codes.

As an example, consider the family of RS codes $C_{\mathrm{RS}}(n, n-4)$ that are able to correct any error pattern of size $t = 2$ or less. Errors are in positions j_1 and j_2, such that

$$0 \leq j_1 < j_2 \leq n-1$$

The received polynomial is evaluated for the roots, leading to a system of four equations, with four unknown variables, which are the positions and the values of the two errors:

$$\begin{aligned}
s_1 &= r(\alpha) = e(\alpha) = e_{j_1}\alpha^{j_1} + e_{j_2}\alpha^{j_2} \\
s_2 &= r(\alpha^2) = e(\alpha^2) = e_{j_1}\alpha^{2j_1} + e_{j_2}\alpha^{2j_2} \\
s_3 &= r(\alpha^3) = e(\alpha^3) = e_{j_1}\alpha^{3j_1} + e_{j_2}\alpha^{3j_2} \\
s_4 &= r(\alpha^4) = e(\alpha^4) = e_{j_1}\alpha^{4j_1} + e_{j_2}\alpha^{4j_2}
\end{aligned}$$

This system of four equations is non-linear, but solvable. The polynomials $\sigma(X)$ and $W(X)$ are calculated as follows:

$$\sigma(X) = (X - \alpha^{-j_1})(X - \alpha^{-j_2})$$

and

$$W(X) = e_{j_1}(X - \alpha^{-j_2}) + e_{j_2}(X - \alpha^{-j_1})$$

These are co-prime polynomials for which 1 is the highest common factor. Roots of $\sigma(X)$ are different from roots of $W(X)$. The values of the errors are

$$e_{j_1} = \frac{W(\alpha^{-j_1})}{\sigma'(\alpha^{-j_1})} \quad \text{and} \quad e_{j_2} = \frac{W(\alpha^{-j_2})}{\sigma'(\alpha^{-j_2})}$$

$$\sigma^{'}(X) = (X - \alpha^{-j_1}) + (X - \alpha^{-j_2})$$

Then

$$\begin{aligned}
\sigma^{'}(\alpha^{-j_1}) &= \alpha^{-j_1} - \alpha^{-j_2} \\
\sigma^{'}(\alpha^{-j_2}) &= \alpha^{-j_2} - \alpha^{-j_1}
\end{aligned}$$

and

$$\begin{aligned}
W(\alpha^{-j_1}) &= e_{j_1}(\alpha^{-j_1} - \alpha^{-j_2}) \\
W(\alpha^{-j_2}) &= e_{j_2}(\alpha^{-j_2} - \alpha^{-j_1})
\end{aligned}$$

The syndrome polynomial is of the form

$$\begin{aligned}
S(X) &= s_1 + s_2X + s_3X^2 + s_4X^3 \\
&= \sum_{j=0}^{3} s_{j+1}X^j \\
&= \sum_{j=0}^{3} \left[e_{j_1}\alpha^{(j+1)j_1} + e_{j_2}\alpha^{(j+1)j_2}\right] X^j \\
&= e_{j_1}\alpha^{j_1} \sum_{j=0}^{3}(\alpha^{j_1}X)^j + e_{j_2}\alpha^{j_2} \sum_{j=0}^{3}\left(\alpha^{j_2}X\right)^j
\end{aligned}$$

and by using the expression $\sum_{j=0}^{\tau} X^j = \frac{X^{\tau+1}-1}{X-1}$,

$$S(X) = e_{j_1}\alpha^{j_1}\frac{(\alpha^{j_1}X)^4 - 1}{\alpha^{j_1}X - 1} + e_{j_2}\alpha^{j_2}\frac{(\alpha^{j_2}X)^4 - 1}{\alpha^{j_2}X - 1} = e_{j_1}\frac{\alpha^{4j_1}X^4 - 1}{X - \alpha^{-j_1}} + e_{j_2}\frac{\alpha^{4j_2}X^4 - 1}{X - \alpha^{-j_2}}$$

$$S(X) = -\frac{e_{j_1}}{X - \alpha^{-j_1}} - \frac{e_{j_2}}{X - \alpha^{-j_2}} + X^4\left[e_{j_1}\frac{\alpha^{4j_1}}{X - \alpha^{-j_1}} + e_{j_2}\frac{\alpha^{4j_2}}{X - \alpha^{-j_2}}\right]$$

Multiplying $S(X)$ by $\sigma(X)$,

$$\sigma(X)S(X) = -e_{j_1}(X - \alpha^{-j_2}) - e_{j_2}(X - \alpha^{-j_1}) + X^4\mu(X) = -W(X) + X^4\mu(X)$$

which is the key equation corresponding to this example.

5.6 Decoding of RS Codes Using the Euclidean Algorithm

A simple example of the use of the Euclidean algorithm for factorizing two integer numbers was introduced in the previous chapter. This algorithm is now utilized for solving the key equation in the decoding of RS codes [3]. The key equation is of the form

$$\sigma(X)S(X) - \mu(X)X^{n-k} = -W(X)$$

The algorithm is applied to the polynomials X^{n-k} and $S(X)$ so that the ith recursion is

$$r_i(X) = s_i(X)X^{n-k} + t_i(X)S(X) \tag{29}$$

and

$$\deg\{r_i(X)\} \le \left\lfloor \frac{n-k}{2} \right\rfloor + 1 \tag{30}$$

On the other hand,

$$W(X) = -\lambda r_i(X) \tag{31}$$

$$\sigma(X) = \lambda t_i(X)$$

where λ is a constant that converts the polynomial into a monic polynomial.

Example 5.5: For the RS code $C_{\mathrm{RS}}(7, 3)$ defined over GF(2^3), generated by the primitive polynomial $p_i(X) = 1 + X^2 + X^3$, and for the received vector $\boldsymbol{r}$ = (000 000 011 000 111 000 000), determine using the Euclidean algorithm the error polynomial and the decoded vector.

According to the received vector, the received polynomial is equal to

$$r(X) = \alpha^6 X^2 + \alpha^4 X^4$$

Table 5.2 Euclidean algorithm for solving the key equation, Example 5.5

i	$r_i = r_{i-2} - q_i r_{i-1}$	q_i	$t_i = t_{i-2} - q_i t_{i-1}$
-1	$X^{n-k} = X^4$		0
0	$S(X) = \alpha^4 X^3 + \alpha^4 X^2 + \alpha^6 X$		1
1	$\alpha^3 X^2 + \alpha^2 X$	$\alpha^3 X + \alpha^3$	$\alpha^3 X + \alpha^3$
2	$\alpha^4 X$	$\alpha X + \alpha^5$	$\alpha^4 X^2 + \alpha^3 X + \alpha^5$

The syndrome vector components are

$$s_1 = r(\alpha) = \alpha + \alpha = 0$$
$$s_2 = r(\alpha^2) = \alpha^3 + \alpha^5 = \alpha^6$$
$$s_3 = r(\alpha^3) = \alpha^5 + \alpha^2 = \alpha^4$$
$$s_4 = r(\alpha^4) = 1 + \alpha^6 = \alpha^4$$

The syndrome polynomial is then

$$S(X) = \alpha^6 X + \alpha^4 X^2 + \alpha^4 X^3$$

The Euclidean algorithm is applied using Table 5.2.

If the degree of the polynomial in the column $r_i(X)$ is less than the degree of the polynomial in the column $t_i(X)$, then the recursion is halted. It also happens that $\alpha^4 X$ is a factor of $\alpha^3 X^2 + \alpha^2 X$. Then

$$W_1(X) = \alpha^4 X$$
$$\sigma_1(X) = \alpha^4 X^2 + \alpha^3 X + \alpha^5$$

The polynomial so obtained, $\sigma_1(X)$, is multiplied by an element of the Galois field $\lambda \in \mathrm{GF}(2^3)$ in order to convert it into a monic polynomial. This value of λ is $\lambda = \alpha^3$. Then

$$W(X) = \lambda W_1(X) = \alpha^3 \alpha^4 X = X$$

and

$$\sigma(X) = \lambda \sigma_1(X) = \alpha^3 \left(\alpha^4 X^2 + \alpha^3 X + \alpha^5\right) = X^2 + \alpha^6 X + \alpha$$

The Chien search [5] (as described in Chapter 4) is then used to determine the roots of this error-location polynomial.

These roots are found to be α^3 and α^5. Therefore,

$$\alpha^3 = \alpha^{-j_1} = \alpha^{-4}$$
$$j_1 = 4$$

and

$$\alpha^5 = \alpha^{-j_2} = \alpha^{-2}$$
$$j_2 = 2$$

Errors are thus located at positions $j_1 = 4$ and $j_2 = 2$. The derivative of the error-location polynomial is

$$\sigma'(X) = \alpha^6$$

and so the error values are

$$e_{j_1} = \frac{W(\alpha^{-j_1})}{\sigma'(\alpha^{-j_1})} = \frac{W(\alpha^3)}{\sigma'(\alpha^3)} = \frac{\alpha^3}{\alpha^6} = \alpha^{-3} = \alpha^4$$

$$e_{j_2} = \frac{W(\alpha^{-j_2})}{\sigma'(\alpha^{-j_2})} = \frac{W(\alpha^5)}{\sigma'(\alpha^5)} = \frac{\alpha^5}{\alpha^6} = \alpha^{-1} = \alpha^6$$

The error polynomial is then

$$e(X) = \alpha^6 X^2 + \alpha^4 X^4$$

so that the received vector is corrected by adding to it the error pattern, and thus the decoded vector is finally the all-zero vector.

5.6.1 Steps of the Euclidean Algorithm

For an RS code $C_{\mathrm{RS}}(n, k)$ with error-correction capability t, where $2t \leq n - k$, and for a received polynomial $r(X)$, application of the Euclidean algorithm involves the following steps [3]:

1. Calculate the syndrome vector components $s_j = r(\alpha^j)$, $1 \leq j \leq n - k$, and then construct the syndrome polynomial

$$S(X) = \sum_{j=1}^{n-k} s_j X^{j+1} \tag{32}$$

2. If $S(X) = 0$ then the corresponding received vector is considered to be a code vector.
3. If $S(X) \neq 0$ then the algorithm is initialized as

$$\begin{aligned} r_{-1}(X) &= X^{n-k} \\ r_0 &= S(X) \\ t_{-1}(X) &= 0 \\ t_0(X) &= 1 \\ i &= -1 \end{aligned} \tag{33}$$

4. Recursion parameters are determined as

$$r_i(X) = r_{i-2}(X) - q(X) r_{i-1}(X) \tag{34}$$

and

$$t_i(X) = t_{i-2}(X) - q(X)t_{i-1}(X) \tag{35}$$

This recursion proceeds as long as $\deg(r_i(X)) \geq t$.

5. When $\deg(r_i(X)) < t$, the recursion halts and a number $\lambda \in \mathrm{GF}(2^m)$ is obtained and so $\lambda t_i(X)$ becomes a monic polynomial. Then

$$\sigma(X) = \lambda t_i(X) \tag{36}$$

and

$$W(X) = -\lambda r_i(X) \tag{37}$$

6. Roots of the polynomial $\sigma(X)$ are obtained by using the Chien search.
7. Error values are calculated using

$$e_{jh} = \frac{W(\alpha^{-jh})}{\sigma'(\alpha^{-jh})}$$

8. The error polynomial is constructed using

$$e(X) = e_{j_1}X^{j_1} + e_{j_2}X^{j_2} + \cdots + e_{j_\tau}X^{j_\tau} \tag{38}$$

9. Error correction is verified, such that if

$$e(\alpha^i) \neq r(\alpha^i) \quad \text{for any } i \tag{39}$$

then error correction is discarded since (39) means that the number of errors was over the error-correction capability of the code.

If

$$e(\alpha^i) = r(\alpha^i) \quad \text{for all } i \tag{40}$$

then

$$c(X) = r(X) + e(X)$$

5.7 Decoding of RS and BCH Codes Using the Berlekamp–Massey Algorithm

The Berlekamp–Massey (B–M) algorithm [2, 6, 7] is an alternative algebraic decoding algorithm for RS and BCH codes. In the case of binary BCH codes, there is no need to calculate the error values, as it is enough to determine the positions of the errors to perform error correction in GF(2). This is different from the case of non-binary BCH codes and RS codes, where both error location and error values have to be determined to perform error correction.

In the case of BCH codes, equations (16) and (18) of Chapter 4 are still valid; that is, syndrome decoding involves the calculation of the $2t$ components of the syndrome vector

$$\boldsymbol{S} = (s_1, s_2, \ldots, s_{2\mathrm{t}})$$

Syndrome vector components can be calculated by replacing the variable X with the different roots α^i, $i = 1, 2, \ldots, 2t$, in the expression of the received polynomial $r(X)$. As both RS and BCH codes are linear, the syndrome depends only on the error event. Assume that the error pattern contains τ errors in positions $X^{j_1}, X^{j_2}, \ldots, X^{j_\tau}$, so that the error pattern in polynomial form is

$$e(X) = X^{j_1} + X^{j_2} + \cdots + X^{j_\tau}$$

Since syndrome components can be evaluated as a function of the error pattern,

$$\begin{aligned} s_1 &= \alpha^{j_1} + \alpha^{j_2} + \cdots + \alpha^{j_\tau} \\ s_2 &= (\alpha^{j_1})^2 + (\alpha^{j_2})^2 + \cdots + (\alpha^{j_\tau})^2 \\ &\vdots \\ s_{2t} &= (\alpha^{j_1})^{2t} + (\alpha^{j_2})^{2t} + \cdots + (\alpha^{j_\tau})^{2t} \end{aligned} \tag{41}$$

The α^{j_i} values are unknown. Any algorithm able to find a solution to this system of equations can be considered as a decoding algorithm for BCH codes. Equation (19) is the equivalent system of equations for RS codes.

A procedure to calculate values $\alpha^{j_1}, \alpha^{j_2}, \ldots, \alpha^{j_\tau}$ allows us to determine the error positions $j_1, j_2, \ldots, j_\tau$ which in the case of binary BCH codes constitute enough information to perform error correction. The so-called error-location numbers are usually defined as

$$\beta_i = \alpha^{j_i}$$

By using this definition the system of equations (41) looks like the system of equations (24) of Chapter 4. Indeed, in the case of binary BCH codes, coefficients e_{ji} are all equal to unity, $e_{ji} = 1$, giving

$$\begin{aligned} s_1 &= \beta_1 + \beta_2 + \cdots + \beta_\tau \\ s_2 &= (\beta_1)^2 + (\beta_2)^2 + \cdots + (\beta_\tau)^2 \\ &\vdots \\ s_2t &= (\beta_1)^{2t} + (\beta_2)^{2t} + \cdots + (\beta_\tau)^{2t} \end{aligned} \tag{42}$$

The error-location polynomial can have a slightly different definition with respect to expression (22), which is the following:

$$\begin{aligned} \sigma_{\mathrm{BM}}(X) &= (1 + \beta_1 X)(1 + \beta_2 X)\ldots(1 + \beta_\tau X) \\ &= \sigma + \sigma_1 X + \cdots + \sigma_\tau X^\tau \end{aligned} \tag{43}$$

Roots of this polynomial are $\beta_1^{-1}, \beta_2^{-1}, \ldots, \beta_\tau^{-1}$, the inverses of the error-location numbers. This is a modified definition of the error-location polynomial with respect to the definition given for the decoding of RS codes using the Euclidean algorithm, as a tool for solving the key equation. This modified definition is however more suitable for the description of the B–M algorithm.

Coefficients of this modified polynomial can be expressed in the following manner:

$$\begin{aligned}
\sigma_0 &= 1\\
\sigma_1 &= \beta_1 + \beta_2 + \cdots + \beta_\tau\\
\sigma_2 &= \beta_1\beta_2 + \beta_2\beta_3 + \cdots + \beta_{\tau-1}\beta_\tau\\
&\vdots\\
\sigma_\tau &= \beta_1\beta_2\ldots\beta_\tau
\end{aligned} \tag{44}$$

This set of equations is known as the elementary symmetric functions and is related to the system of equations (42) as follows:

$$\begin{aligned}
&s_1 + \sigma_1 = 0\\
&s_2 + \sigma_1 s_1 = 0\\
&s_3 + \sigma_1 s_2 + \sigma_2 s_1 + \sigma_3 = 0\\
&\vdots\\
&s_{\tau+1} + \sigma_1 s_\tau + \cdots + \sigma_{\tau-1} s_2 + \sigma_\tau s_1 = 0
\end{aligned} \tag{45}$$

These equations are called Newton identities [2]. Thus, for instance,

$$s_2 + \sigma_1 s_1 = (\beta_1)^2 + (\beta_2)^2 + \cdots + (\beta_\tau)^2 + (\beta_1 + \beta_2 + \cdots + \beta_\tau)(\beta_1 + \beta_2 + \cdots + \beta_\tau) = 0$$

since in GF(2^m) the products $\beta_i\beta_j + \beta_j\beta_i = 0$. The remaining Newton identities can be derived in the same way.

5.7.1 B–M Iterative Algorithm for Finding the Error-Location Polynomial

The decoding procedure based on the B–M algorithm is now introduced [6, 7]. Demonstration of its main properties can be found in [4]. The algorithm basically consists of finding the coefficients of the error-location polynomial, $\sigma_1, \sigma_2, \ldots, \sigma_\tau$, whose roots then determine the error positions.

The B–M algorithm starts with the evaluation of the $2t$ syndrome vector components $\boldsymbol{S} = (s_1, s_2, \ldots, s_{2t})$, which then allow us to find the coefficients $\sigma_1, \sigma_2, \ldots, \sigma_\tau$ of the error-location polynomial, whose roots are the inverses of the error-location numbers $\beta_1, \beta_2, \ldots, \beta_\tau$. In the case of a binary BCH code, the final step is to perform error correction at the determined error positions, and in the case of an RS code or a non-binary BCH code, the error values at those positions have to be calculated, in order to finally perform error correction. As indicated above, the core of the B–M algorithm is an iterative method for determining the error-location polynomial coefficients $\sigma_1, \sigma_2, \ldots, \sigma_\tau$.

The algorithm proceeds as follows [2]: The first step is to determine a minimum-degree polynomial $\sigma_{\mathrm{BM}}^{(1)}(X)$ that satisfies the first Newton identity described in (45). Then the second Newton identity is tested. If the polynomial $\sigma_{\mathrm{BM}}^{(1)}(X)$ satisfies the second Newton identity in (45), then $\sigma_{\mathrm{BM}}^{(2)}(X) = \sigma_{\mathrm{BM}}^{(1)}(X)$. Otherwise the decoding procedure adds a correction term to

$\sigma_{\mathrm{BM}}^{(1)}(X)$ in order to form the polynomial $\sigma_{\mathrm{BM}}^{(2)}(X)$, able to satisfy the first two Newton identities. This procedure is subsequently applied to find $\sigma_{\mathrm{BM}}^{(3)}(X)$, and the following polynomials, until determination of the polynomial $\sigma_{\mathrm{BM}}^{(2t)}(X)$ is complete. Once the algorithm reaches this step, the polynomial $\sigma_{\mathrm{BM}}^{(2t)}(X)$ is adopted as the error-location polynomial $\sigma_{\mathrm{BM}}(X)$, $\sigma_{\mathrm{BM}}(X) = \sigma_{\mathrm{BM}}^{(2t)}(X)$, since this last polynomial satisfies the whole set of Newton identities described in (45).

This algorithm can be implemented in iterative form. Let the minimum-degree polynomial obtained in the μth iteration and able to satisfy the first μ Newton identities, denoted by $\sigma_{\mathrm{BM}}^{(\mu)}(X)$, be of the form

$$\sigma_{\mathrm{BM}}^{(\mu)}(X) = 1 + \sigma_1^{(\mu)} X + \sigma_2^{(\mu)} X^2 + \cdots + \sigma_{l_\mu}^{(\mu)} X^{l_\mu} \tag{46}$$

where l_μ is the degree of the polynomial $\sigma_{\mathrm{BM}}^{(\mu)}(X)$. Then, a quantity d_μ, called the μth discrepancy, is obtained by using the following expression:

$$d_\mu = s_{\mu+1} + \sigma_1^{(\mu)} s_\mu + \sigma_2^{(\mu)} s_{\mu-1} + \cdots + \sigma_{l_\mu}^{(\mu)} s_{\mu+1-l_\mu} \tag{47}$$

If the discrepancy d_μ is equal to zero, $d_\mu = 0$, then the minimum-degree polynomial $\sigma_{\mathrm{BM}}^{(\mu)}(X)$ satisfies $(\mu + 1)$th Newton identity, and it becomes $\sigma_{\mathrm{BM}}^{(\mu+1)}(X)$:

$$\sigma_{\mathrm{BM}}^{(\mu+1)}(X) = \sigma_{\mathrm{BM}}^{(\mu)}(X) \tag{48}$$

If the discrepancy d_μ is not equal to zero, $d_\mu \neq 0$, then the minimum-degree polynomial $\sigma_{\mathrm{BM}}^{(\mu)}(X)$ does not satisfy the $(\mu + 1)$th Newton identity, and a correction term is calculated to be added to $\sigma_{\mathrm{BM}}^{(\mu)}(X)$, in order to form $\sigma_{\mathrm{BM}}^{(\mu+1)}(X)$.

In the calculation of the correction term, the algorithm resorts to a previous step ρ of the iteration, with respect to μ, such that the discrepancy $d_\rho \neq 0$ and $\rho - l_\rho$ is a maximum. The number l_ρ is the degree of the polynomial $\sigma_{\mathrm{BM}}^{(\rho)}(X)$.

Then

$$\sigma^{(\mu+1)}(X) = \sigma^{(\mu)}(X) + d_\mu d_\rho^{-1} X^{(\mu-\rho)} \sigma^{(\rho)}(X) \tag{49}$$

and this polynomial of minimum degree satisfies the $(\mu + 1)$th Newton identity.

The B–M algorithm can be implemented in the form of a table, as given in Table 5.3.

Table 5.3 B–M algorithm table for determining the error-location polynomial

μ	$\sigma_{\mathrm{BM}}^{(\mu)}(X)$	d_μ	l_μ	$\mu - l_\mu$
-1	1	1	0	-1
0	1	s_1	0	0
1				
.				
.				
$2t$				

The minimum-degree polynomial $\sigma_{\mathrm{BM}}^{(\mu+1)}(X)$ in iteration $\mu + 1$ can be calculated as a function of the minimum-degree polynomial $\sigma_{\mathrm{BM}}^{(\mu)}(X)$ of the μth iteration as follows:

$$\text{If } d_\mu = 0 \qquad \text{then } \sigma_{\mathrm{BM}}^{(\mu+1)}(X) = \sigma_{\mathrm{BM}}^{(\mu)}(X),\, l_{\mu+1} = l_\mu.$$

If $d_\mu \neq 0$, the algorithm resorts to a previous row ρ, such that $d_\rho \neq 0$ and $\rho - l_\rho$ is maximum. Then

$$\begin{aligned}
\sigma_{\mathrm{BM}}^{(\mu+1)}(X) &= \sigma_{\mathrm{BM}}^{\mu}(X) + d_\mu d_\rho^{-1} X^{(\mu-\rho)} \sigma^{(\rho)}(X),\\
l_{\mu+1} &= \max(l_\mu, l_\rho + \mu - \rho),\\
d_{\mu+1} &= s_{\mu+2} + \sigma_1^{(\mu+1)} s_{\mu+1} + \cdots + \sigma_{l_{\mu+1}}^{(\mu+1)} s_{\mu+2-l_{\mu+1}}
\end{aligned} \tag{50}$$

The above procedure has to be applied for the $2t$ rows of the table, until the minimum-degree polynomial $\sigma_{\mathrm{BM}}^{(2t)}(X)$ is obtained, and finally this last polynomial is adopted as the error-location polynomial $\sigma_{\mathrm{BM}}^{(2t)}(X) = \sigma_{\mathrm{BM}}(X)$. In general, if the degree of $\sigma_{\mathrm{BM}}^{(2t)}(X)$ is larger than t, then it is likely that its roots do not correspond to the inverses of the real error-location numbers, since it is also likely that the error event was severe, and affected more than t elements, being over the error-correction capability of the code.

After the determination of the error-location polynomial, the roots of this polynomial are calculated by applying the Chien search, as used in Euclidean algorithm decoding, by replacing the variable X with all the elements of the Galois field GF(q), $1, \alpha, \alpha^2, \ldots, \alpha^{q-2}$, in the expression of the obtained error-location polynomial, looking for the condition $\sigma_{\mathrm{BM}}(\alpha^i) = 0$. This procedure leads to an estimate of the error pattern of minimum weight, which solves the system of syndrome equations. This will be the true error pattern that occurred on the channel if the number of errors τ in this pattern is $\tau \leq t$.

Example 5.6: Apply the B–M algorithm to Example 4.3, which concerns the binary BCH code $C_{\mathrm{BCH}}(15, 7)$ with $t = 2$, and a received polynomial of the form $r(X) = 1 + X^8$.

The syndrome components were calculated in Example 4.3 as

$$\begin{aligned}
s_1 &= r(\alpha) = \alpha^2\\
s_2 &= r(\alpha^2) = \alpha^4\\
s_3 &= r(\alpha^3) = \alpha^7\\
s_4 &= r(\alpha^4) = \alpha^8
\end{aligned}$$

Table 5.4 is used to apply the B–M algorithm in order to determine the error-location polynomial for the case of Example 5.6.

As an example, row $\mu + 1 = 1$ is obtained from information at row $\mu = 0$ by evaluating

$$\begin{aligned}
l_1 &= \max(l_0, l_{-1} + 0 - (-1)) = 1\\
\sigma_{\mathrm{BM}}^{(1)}(X) &= \sigma_{\mathrm{BM}}^{(0)}(X) + d_0 d_{-1}^{-1} X^{(0-(-1))} \sigma^{(-1)}(X) = 1 + \alpha^2 1^{-1} X^1 1 = 1 + \alpha^2 X
\end{aligned}$$

and

$$d_1 = s_2 + \sigma_1^{(1)} s_1 = \alpha^4 + \alpha^2 \alpha^2 = 0$$

Table 5.4 B–M algorithm table, Example 5.6

μ	$\sigma_{\mathrm{BM}}^{(\mu)}(X)$	d_μ	l_μ	$\mu - l_\mu$	ρ
-1	1	1	0	-1	
0	1	α^2	0	0	
1	$1+\alpha^2 X$	0	1	0	-1
2	$1+\alpha^2 X$	α^{10}	1	1	
3	$1+\alpha^2 X+\alpha^8 X^2$	0	2	1	0
4	$1+\alpha^2 X+\alpha^8 X^2$				

The next step is performed using row $\mu = 1$, whose information is useful to determine the values in row $\mu + 1 = 2$, and since the discrepancy is zero, $d_1 = 0$, then

$$\sigma_{\mathrm{BM}}^{(2)}(X) = \sigma_{\mathrm{BM}}^{(1)}(X), \qquad l_2 = l_1 = 1 \qquad d_2 = s_3 + \sigma_1^{(2)} s_2 = \alpha^7 + \alpha^2\alpha^4 = \alpha^{10}$$

The procedure ends at row $\mu = 4$, where the error-location polynomial is finally

$$\sigma_{\mathrm{BM}}(X) = \sigma_{\mathrm{BM}}^{(4)}(X) = 1 + \alpha^2 X + \alpha^8 X^2$$

whose roots are $\beta_1^{-1} = 1 = \alpha^0 = \alpha^{-j_1}$ and $\beta_2^{-1} = \alpha^7 = \alpha^{-8} = \alpha^{-j_2}$, and so error positions are at $j_1 = 0$ and $j_2 = 8$.

The error-location polynomial $\sigma_{\mathrm{BM}}(X) = 1 + \alpha^2 X + \alpha^8 X^2$ (B–M algorithm, Example 5.6) is obtained as a function of the error-location polynomial $\sigma(X) = \alpha^7 + \alpha^9 X + X^2$ (Euclidean algorithm, Example 4.5) by multiplying $\sigma(X) = \alpha^7 + \alpha^9 X + X^2$ by α^8. Therefore, both polynomials have the same roots. After performing error correction, the decoded polynomial is the all-zero polynomial, as in the case of Example 4.5.

5.7.2 B–M Decoding of RS Codes

RS codes are non-binary codes, and this means that a given error, located at a given position, can adopt any value among the elements of the Galois field GF(q) used for the design of that code. As explained above, this brings an additional step into the decoding of RS codes, which is the need to determine not only the position but also the value of an error, to perform error correction. The following examples can be used to illustrate that the error-location polynomial, calculated for a given error pattern, does not depend on the error value. Therefore the B–M algorithm, which is in essence useful for determining this error-location polynomial, can be applied in the same way as for binary BCH codes in order to determine the error-location numbers in the decoding of an RS code. Indeed, the B–M algorithm solves part of a system of $2t$ equations, with $2t$ unknowns, by forming a system of t equations, to determine the t unknowns which are the positions of the errors. Therefore the system of equations (42) is to be solved for determining error-location numbers in an RS code, whose complete system of equations is of the form of (19). The difference between these two systems of equations is that the system of equations (19) leaves open the actual error values, while the system of equations (42) considers the existence of an error to be an error event of value 1.

Once the B–M algorithm determines the error-location polynomial, and by using the Chien Search, its roots can be calculated, and the error positions are then determined. At this point, error values can be obtained by using expressions (22)–(24), as done in the Euclidean algorithm. There is however an equivalent method, described by Berlekamp [4] for evaluating error values. This equivalent method requires the definition of the following polynomial [2]:

$$\begin{aligned} Z(X) = 1 + (s_1 + \sigma_1)\,X + (s_2 + \sigma_1 s_1 + \sigma_2)\,X^2 + \cdots + \\ (s_\tau + \sigma_1 s_{\tau-1} + \sigma_2 s_{\tau-2} + \cdots + \sigma_\tau)\,X^\tau \end{aligned} \tag{51}$$

Error values at position $\beta_i = \alpha_{j_i}$ are calculated as

$$e_{j_i} = \frac{Z\left(\beta_i^{-1}\right)}{\prod_{\substack{k=1\\k\neq i}}^{\tau}\left(1 + \beta_k \beta_i^{-1}\right)} \tag{52}$$

Example 5.7: Decode the received vector $\boldsymbol{r}$ = (000 000 011 000 111 000 000) for the RS code $C_{\mathrm{RS}}(7, 3)$, over GF(8) (prime polynomial $p_i(X) = 1 + X^2 + X^3$), using the B–M algorithm (see Example 5.5).

According to the received vector, converted into its polynomial form

$$r(X) = \alpha^6 X^2 + \alpha^4 X^4$$

the syndrome vector components are

$$\begin{aligned} s_1 &= r(\alpha) = 0 \\ s_2 &= r(\alpha^2) = \alpha^6 \\ s_3 &= r(\alpha^3) = \alpha^4 \\ s_4 &= r(\alpha^4) = \alpha^4 \end{aligned}$$

The B–M algorithm is applied by means of Table 5.5.

The error-location polynomial is then

$$\sigma_{\mathrm{BM}}(X) = \sigma_{\mathrm{BM}}^{(4)}(X) = 1 + \alpha^5 X + \alpha^6 X^2$$

whose roots are $\beta_1^{-1} = \alpha^3 = \alpha^{-4} = \alpha^{-j_1}$ and $\beta_2^{-1} = \alpha^5 = \alpha^{-2} = \alpha^{-j_2}$, and so error positions are $j_1 = 4$ and $j_2 = 2$.

Table 5.5 B–M algorithm table, Example 5.7

μ	$\sigma_{\mathrm{BM}}^{(\mu)}(X)$	d_μ	l_μ	$\mu - l_\mu$	ρ
−1	1	1	0	−1	
0	1	0	0	0	
1	1	α^6	1	0	−1
2	$1 + \alpha^6 X^2$	α^4	2	0	1
3	$1 + \alpha^5 X + \alpha^6 X^2$	0	2	1	
4	$1 + \alpha^5 X + \alpha^6 X^2$				

The error-location polynomial $\sigma_{\mathrm{BM}}(X) = 1 + \alpha^5 X + \alpha^6 X^2$ is obtained as a function of the error-location polynomial $\sigma(X) = \alpha + \alpha^6 X + X^2$ obtained for the same case by applying the Euclidean algorithm in Example 5.5, Table 5.2, by multiplying $\sigma(X) = \alpha + \alpha^6 X + X^2$ by α^6. Therefore both polynomials have the same roots.

Calculation of the error values requires to form the polynomial

$$Z(X) = 1 + (s_1 + \sigma_1)X + (s_2 + \sigma_1 s_1 + \sigma_2)X^2 = 1 + \alpha^5 X + (\alpha^6 + \alpha^5 0 + \alpha^6)X^2 = 1 + \alpha^5 X$$

Then

$$e_{j_1} = \frac{Z\left(\beta_1^{-1}\right)}{\prod_{\substack{k=1\\k\neq 1}}^{2}\left(1 + \beta_k \beta_1^{-1}\right)} = \frac{1 + \alpha^5\alpha^3}{(1 + \alpha^{-5}\alpha^3)} = \frac{\alpha^5}{\alpha} = \alpha^4$$

$$e_{j_2} = \frac{Z\left(\beta_2^{-1}\right)}{\prod_{\substack{k=1\\k\neq 2}}^{2}\left(1 + \beta_k \beta_2^{-1}\right)} = \frac{1 + \alpha^5\alpha^5}{(1 + \alpha^{-3}\alpha^5)} = \frac{\alpha^2}{\alpha^3} = \alpha^6$$

Example 5.8: Decode the received vector $\boldsymbol{r} = (000\ 000\ 100\ 000\ 100\ 000\ 000)$ for an RS code $C_{\mathrm{RS}}(7, 3)$ over GF(2^3) (prime polynomial $p_i(X) = 1 + X^2 + X^3$). In this example, the error pattern presents the same error positions as in Example 5.7, but all errors are of value 1.

The received error vector represented in its polynomial form is

$$r(X) = X^2 + X^4$$

The syndrome vector components are

$$s_1 = r(\alpha) = \alpha^5$$
$$s_2 = r(\alpha^2) = \alpha^3$$
$$s_3 = r(\alpha^3) = \alpha^3$$
$$s_4 = r(\alpha^4) = \alpha^6$$

The B–M algorithm is applied by means of Table 5.6.

The error-location polynomial is then

$$\sigma_{\mathrm{BM}}(X) = \sigma_{\mathrm{BM}}^{(4)}(X) = 1 + \alpha^5 X + \alpha^6 X^2$$

Table 5.6 B–M algorithm table, Example 5.8

μ	$\sigma_{\mathrm{BM}}^{(\mu)}(X)$	d_μ	l_μ	$\mu - l_\mu$	ρ
-1	1	1	0	-1	
0	1	α^5	0	0	-1
1	$1 + \alpha^5 X$	0	1	0	
2	$1 + \alpha^5 X$	α^4	1	1	1
3	$1 + \alpha^5 X + \alpha^6 X^2$	0	2	1	0
4	$1 + \alpha^5 X + \alpha^6 X^2$				

which is the same error-location polynomial calculated in Example 5.7, where the error pattern has errors at the same positions, but of different magnitude in comparison with the error pattern of this example.

Roots of this polynomial are $\beta_1^{-1} = \alpha^3 = \alpha^{-4} = \alpha^{-j_1}$ and $\beta_2^{-1} = \alpha^5 = \alpha^{-2} = \alpha^{-j_2}$, and errors are at positions $j_1 = 4$ and $j_2 = 2$.

Error values are calculated by using the following polynomial:

$$\begin{aligned} Z(X) &= 1 + (s_1 + \sigma_1)X + (s_2 + \sigma_1 s_1 + \sigma_2)X^2 \\ &= 1 + (\alpha^5 + \alpha^5)X + (\alpha^3 + \alpha^5\alpha^5 + \alpha^6)X^2 \\ &= 1 + \alpha^6 X^2 \end{aligned}$$

So

$$e_{j_1} = \frac{Z\left(\beta_1^{-1}\right)}{\prod_{\substack{k=1\\k\neq 1}}^{2}\left(1 + \beta_k\beta_1^{-1}\right)} = \frac{1 + \alpha^6\alpha^6}{(1 + \alpha^{-5}\alpha^3)} = \frac{\alpha}{\alpha} = 1$$

$$e_{j_2} = \frac{Z\left(\beta_2^{-1}\right)}{\prod_{\substack{k=1\\k\neq 2}}^{2}\left(1 + \beta_k\beta_2^{-1}\right)} = \frac{1 + \alpha^6\alpha^{10}}{(1 + \alpha^{-3}\alpha^5)} = \frac{\alpha^3}{\alpha^3} = 1$$

5.7.3 Relationship Between the Error-Location Polynomials of the Euclidean and B–M Algorithms

Error-location polynomials defined in both of these algorithms are practically the same. As an example, for the case of RS codes able to correct error patterns of size $t = 2$ or less, and for the Euclidean algorithm, the error-location polynomial is equal to

$$\begin{aligned} \sigma(X) &= (X - \alpha^{-j_1})(X - \alpha^{-j_2}) \\ &= (X + \alpha^{-j_1})(X + \alpha^{-j_2}) \\ &= \left(X + \beta_1^{-1}\right)\left(X + \beta_2^{-1}\right) \\ &= (1 + \beta_1 X)(1 + \beta_2 X)/(\beta_1\beta_2) \\ &= \sigma_{\mathrm{BM}}(X)/(\beta_1\beta_2) \end{aligned}$$

So, for the same error event, both the Euclidean and the B–M error-location polynomials have the same roots, since they differ only by a constant factor. In general,

$$\sigma(X) = \frac{\sigma_{\mathrm{BM}}(X)}{\prod_{i=1}^{\tau}\beta_i}$$

5.8 A Practical Application: Error-Control Coding for the Compact Disk

5.8.1 Compact Disk Characteristics

The error-control coding system for the compact disk (CD) is perhaps one of the most interesting applications of RS codes [8–15]. This system has been designed from the point of view of a communication system, where the transmitter is the recording process, the channel is the disk and the receiver is the CD reader. This storage technique is used for both digital and analog information, like music for instance. In the case of dealing with analog signals, an analog-to-digital conversion of that signal is required. This is done over the right and left channels of a stereo audio signal. The sampling frequency is 44.1 kHz, allowing the conversion of high-quality audio signals with spectral content up to 20 kHz into a digital format. The quantization used is of 16 bits, so that the signal-to-quantization noise ratio is about 90 dB. Distortion is therefore around 0.005%. The sampling frequency has been selected as a function of parameters of the television signal,

$$\{(625 - 37)/625\} \times 3 \times 15{,}625 = 44.1 \text{ kHz}$$

where 625 is the number of lines in PAL system, 37 is the number of unused lines, 3 is the number of audio samples recorded per line and 15,625 Hz is the line frequency [8–14].

The digitized information is then protected against noise by using RS codes, which in the case of the error-correction coding for the CD are two shortened versions of the RS code $C_{\text{RS}}(255, 251)$, which operate over the Galois field GF(2^8), able to correct error patterns of size $t = 2$ or less each, implemented in concatenated form.

The channel is a practical example of channels with burst errors, whose effect is diminished in this coding technique by using concatenated RS codes and interleaving.

The two shortened versions of the RS code are concatenated by using an interleaver, giving form to the so-called cross-interleaved Reed–Solomon code. The analog signal is sampled by taking six samples from each of the audio channels, the right and left channels, thus forming a group of 12 vectors of 16 bits each, that is, a vector of 24 bytes. This information is first processed by an interleaver, and then input to the first encoder, which is a shortened version $C_{\text{RS}}(28, 24)$ of the RS code $C_{\text{RS}}(255, 251)$. This encoder generates an output of 28 bytes that is input to another interleaver, which in turn passes these 28-byte interleaved vectors to a second encoder, another shortened version $C_{\text{RS}}(32, 28)$ of the RS code $C_{\text{RS}}(255, 251)$. The shortening procedure is described in Section 5.9.

The coded information is added to another byte, containing synchronization information, and is then modified by a process known as eight-to-fourteen modulation (EFM) before being printed on the disk surface in digital format. This modulation is applied on the one hand to remove low-frequency components in the spectrum of the signal so as to avoid interference with the tracking control system needed for the reading of the disk, and on the other hand because the CD reader uses self-synchronization, that is, obtains a synchronization signal from the received signal itself, it is important to avoid long chains of ‘0’s, to keep the synchronizer locked. In this particular case the EFM avoids the existence of chains of ‘0’s longer than 10. EFM is essentially a conversion of vectors of 8 bits into vectors of 14 bits, such that the run length limited (RLL) conditions are obeyed. These RLL conditions are such that there should be at least two ‘0’s between any two ‘1’s, and at most ten ‘0’s between any two ‘1’s. Since the vectors have to be concatenated, there is a need to add interconnecting vectors of 3 bits, in order to maintain the RLL conditions over the concatenation of vectors. So finally each vector of 8 bits is converted into a vector of 17 bits. At the end of the whole process, which involves error-control coding, additional information and EFM, the six samples of audio stereo convert into a sequence of 588 bits. This information is transferred to the disk surface at a speed of 4.332 Mbps.

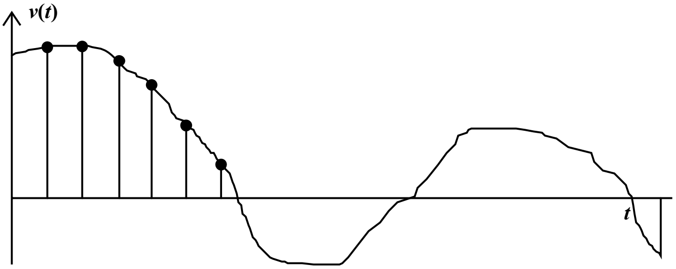


Figure 5.1 Sampled audio signal

5.8.2 *Channel Characteristics*

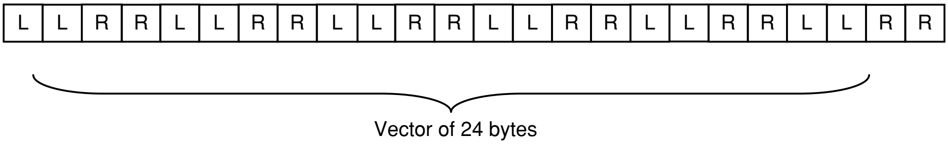
The disk on which information is printed is a plastic disk covered by an aluminium–copper alloy, with a diameter of 120 mm, a thickness of 1.2 mm and a printed pit thickness of 1.6 μm. The CD reader has an AlGaAs laser of wavelength 0.8 μm that reads over the pits to detect the printed value, at a constant speed of 1.25 m/s, and so the angular speed has to vary between 8 and 3.5 revolutions per second. This is done to keep the data rate at a constant value. The error-control coding applied for CDs has a strong error-correction capability, and this makes possible simple manufacture of disks because errors expected due to imperfections in the printing process are automatically corrected by the CD reader [10–14].

5.8.3 *Coding Procedure*

As described in the previous section, the coding procedure for the CD is a combination of RS codes and interleavers. The basic coding block is an array of six samples of 16 bits each taken over the right and left audio channels, resulting in an uncoded vector of 24 bytes. Figure 5.1 represents the left or right audio signal that is sampled to form the uncoded vector.

These six samples are arranged in a vector as shown in Figure 5.2, where samples coming from the right and left channels are identified.

Interleaver I1 takes the even-numbered samples for each stereo channel and separates them from the odd-numbered samples, moving two time windows and filling the empty windows with previous data. Encoder C1 is a shortened version $C_{\mathrm{RS}}(28, 24)$ of the RS code $C_{\mathrm{RS}}(255, 251)$ that adds 4 bytes to the uncoded information, generating a vector of 28 bytes. This is the so-called outer code.



L	L	R	R	L	L	R	R	L	L	R	R	L	L	R	R	L	L	R	R	L	L	R	R

Figure 5.2 Uncoded message format

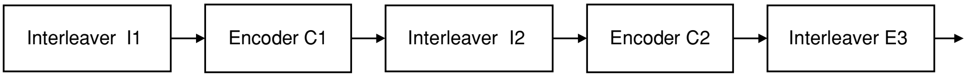


Figure 5.3 Coding procedure for the CD

Interleaver I2 generates a delay of 4 bytes over each processed vector with respect to the previous vector, performing a variable-delay interleave for each position, but a constant-delay interleave between consecutive positions. Encoder C2 is also a shortened version $C_{\mathrm{RS}}(32, 28)$ of the RS code $C_{\mathrm{RS}}(255, 251)$ that adds 4 bytes to the interleaved and coded vector received from interleaver I2, which is a vector of 28 bytes. This is the so-called inner code. After encoding, the resulting interleaved and coded vector is of 32 bytes. Interleaver I3 performs delays and some element inversions to facilitate the operation of an interpolation procedure that takes place after decoding, and that makes imperceptible in the recovered audio signal any errors produced by the whole decoder [8–10]. Figure 5.3 shows this interleaving and encoding procedure.

The decoder performs the inverse of each of the operations described, in order to recover the information.

5.9 Encoding for RS codes $C_{\mathrm{RS}}(28, 24)$, $C_{\mathrm{RS}}(32, 28)$ and $C_{\mathrm{RS}}(255, 251)$

In its original design an RS code is designed for operation over the Galois field GF(2^m), and its code length is $n = 2^m - 1$. One interesting application of these codes is the design of RS codes over the extended Galois field GF(2^8) = GF(256), because any element of a code vector in these codes is itself a vector of 8 bits, or a byte. An RS code designed over this field, able to correct any error pattern of size $t = 2$ or less, is the RS code $C_{\mathrm{RS}}(255, 251)$. A table of the codewords in this code would be enormous. However, the first page of the table (or set of codewords at the top of the table) would contain codewords with ‘0’s only in the most significant message positions. Therefore a shortened RS code can be formed by taking this page of codewords and deleting the positions that are ‘0’s in all the codewords on this page. More specifically, a shortened RS code can be constructed by setting s_{RS} message symbols to zero, where $1 \leq s_{\mathrm{RS}} < k$. Then the length of the code is $n - s_{\mathrm{RS}}$ symbols, the number of message symbols is $k - s_{\mathrm{RS}}$ and the number of parity symbols is $n - k$ as before, where all the symbols remain in GF(2^m). The generator polynomial and the error-correcting capability of the shortened code is the same as that of the unshortened code, but the code is no longer cyclic since not all cyclic shifts of codewords in the shortened code are also in the shortened code. For this reason the shortened code is said to be quasi-cyclic.

In this way, an RS code can be designed without the restriction of having a fixed code length $n = 2^m - 1$, so that the code length can be significantly reduced, without losing any good property of the main code. Thus, shortened RS codes $C_{\mathrm{RS}}(28, 24)$ and $C_{\mathrm{RS}}(32, 28)$ are obtained from the original RS code $C_{\mathrm{RS}}(255, 251)$ by setting $s_{\mathrm{RS}} = 227$ and $s_{\mathrm{RS}} = 223$, respectively, and both the shortened codes have the same error-correction capability as that of the main code $C_{\mathrm{RS}}(255, 251)$, which is $t = 2$. The main code and the shortened versions of this code all have a minimum distance $d_{\min} = 5$. These two codes are the constituent codes of the CD error-control coding system.

For the main code,

$$q = 2^m = 2^8$$

$$n = 2^m - 1 = 255$$

$$n - k = 2t = 255 - 251 = 4$$

$$d_{\min} = 2t + 1 = 5$$

The shortened versions of the main code $C_{\mathrm{RS}}(255, 251)$, the codes $C_{\mathrm{RS}}(28, 24)$ and $C_{\mathrm{RS}}(32, 28)$, all have the same parameters, except that the shortened codes have $n = 28$ and $k = 24$ for the former, and $n = 32$ and $k = 28$ for the latter.

The coding procedure for the CD utilizes an interleaver between these two shortened RS codes, which essentially creates a delayed distribution of the bytes in a given code vector. In this way, the uncoded message of 24 bytes is first encoded by the shortened RS code $C_{\mathrm{RS}}(28, 24)$ that generates a code vector of 28 bytes, and then the interleaver forms a vector of 28 bytes containing bytes from previous code vectors generated by the first encoder. In this interleaving process, each element of the vector of 28 bytes generated by the first encoder is placed in different vectors of 28 bytes position by position. The resulting vector of 28 bytes is input to the encoder of the shortened RS code $C_{\mathrm{RS}}(32, 28)$ that adds 4 bytes to the vector and generates at the end a code vector of 32 bytes.

An example of the operation of these shortened RS codes is introduced here. The encoder operates in systematic form. A message vector, expressed in polynomial form $m(X)$, is multiplied by $X^{2t} = X^4$, as usually done in the systematic encoding of an RS code, generating the polynomial $X^4 m(X)$, which is then divided by the generator polynomial $g(X)$ of the code. As said above, this generator polynomial is the same as that of the main code, and is also the same for both shortened versions of RS codes $C_{\mathrm{RS}}(28, 24)$ and $C_{\mathrm{RS}}(32, 28)$. Therefore, if $t = 2$, then

$$\begin{aligned}
g(X) &= (X + \alpha)(X + \alpha^2)(X + \alpha^3)(X + \alpha^4) \\
g(X) &= (X^2 + \alpha^{26} X + \alpha^3)(X^2 + \alpha^{28} X + \alpha^7) \\
g(X) &= X^4 + (\alpha^{26} + \alpha^{28})X^3 + (\alpha^7 + \alpha^{54})X^2 + (\alpha^{31} + \alpha^{33})X + \alpha^{10} \\
g(X) &= X^4 + \alpha^{76} X^3 + \alpha^{251} X^2 + \alpha^{81} X + \alpha^{10} \qquad (53)
\end{aligned}$$

All operations performed in the calculation of this generator polynomial are done in GF(2^8). This is the generator polynomial of RS codes $C_{\mathrm{RS}}(255, 251)$, $C_{\mathrm{RS}}(28, 24)$ and $C_{\mathrm{RS}}(32, 28)$. As explained above, systematic encoding of a given message polynomial $m(X)$ consists in taking the remainder polynomial of the division of $X^4 m(X)$ by $g(X)$. This remainder polynomial is of degree $2t - 1$ or less, and represents the 4 bytes added by this encoding.

As a result of both shortened RS codes having the same generator polynomial, in this concatenated scheme, the code vector generated by the outer code, the shortened RS code $C_{\mathrm{RS}}(28, 24)$, has to be altered, in order to conveniently enable operation of the second encoder. This is so because, as said above, the generator polynomial for both shortened RS codes in this concatenation is the same. Indeed, after the encoding of a vector of 24 bytes into a code vector of 28 bytes, the resulting code vector belongs to the code, and so it is a multiple of $g(X)$. Even after shifting by 4 bytes the positions of this code polynomial before the second encoding, it is very likely that the shifted code vector of the inner code is still a code vector of the shortened

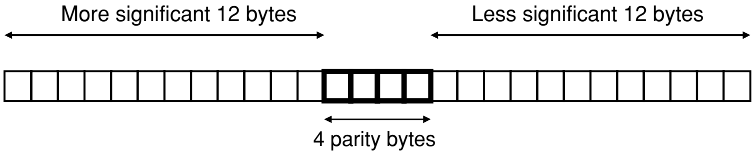


Figure 5.4 Code vector generated by the shortened RS code $C_{\mathrm{RS}}(28, 24)$

RS code $C_{\mathrm{RS}}(32, 28)$. If this is so, then in the systematic encoding procedure for the second (inner) encoder, it is very likely that the remainder, that is, the redundancy, will be a zero polynomial, because the vector to be encoded already belongs to this code. For this reason, the parity bytes generated by the outer code, the shortened RS code $C_{\mathrm{RS}}(28, 24)$, are placed in the middle of the code vector before the inner encoding process, as shown in Figure 5.4 [9, 15].

Example 5.9: An arbitrary message vector $\boldsymbol{m}$ of 24 bytes will be encoded by the shortened RS code $C_{\mathrm{RS}}(28, 24)$ [15]:

$$\begin{aligned}\boldsymbol{m} = \big(&\alpha^{100} \quad \alpha^{90} \quad \alpha^{80} \quad \alpha^{70} \quad \alpha^{0} \quad 0 \quad 0 \quad \alpha^{70} \quad \alpha^{60} \quad \alpha^{50} \quad \alpha^{200} \quad \alpha^{100} \quad \alpha^{2} \quad \alpha^{1} \quad \alpha^{0} \quad 0\\ &\alpha^{4} \quad \alpha^{3} \quad \alpha^{2} \quad \alpha^{1} \quad \alpha^{40} \quad \alpha^{30} \quad \alpha^{20} \quad \alpha^{10}\big)\end{aligned}$$

The convention used here is that the most significant element of the Galois field GF(2^8) = GF(256) is on the left, and the least significant element is on the right. The message vector is composed of elements of the field; for example, element α^2 has a binary representation of the form (00100000). The resulting encoded vector generated by the shortened RS code $C_{\mathrm{RS}}(28, 24)$, expressed in a way that the 4 bytes of the redundancy are in the middle of the code vector, is

$$\begin{aligned}\boldsymbol{c}_1 = \big(&\alpha^{100} \quad \alpha^{90} \quad \alpha^{80} \quad \alpha^{70} \quad \alpha^{0} \quad 0 \quad 0 \quad \alpha^{70} \quad \alpha^{60} \quad \alpha^{50} \quad \alpha^{200} \quad \alpha^{100} \quad \alpha^{\boldsymbol{89}} \quad \alpha^{\boldsymbol{139}} \quad \alpha^{\boldsymbol{249}}\\ &\alpha^{\boldsymbol{228}} \quad \alpha^{2} \quad \alpha^{1} \quad \alpha^{0} \quad 0 \quad \alpha^{4} \quad \alpha^{3} \quad \alpha^{2} \quad \alpha^{1} \quad \alpha^{40} \quad \alpha^{30} \quad \alpha^{20} \quad \alpha^{10}\big)\end{aligned}$$

Part of the polynomial division involved in the calculation of this code vector is as follows:

$$\begin{array}{l|l}
X^4 + \alpha^{76}X^3 + \alpha^{251}X^2 + \alpha^{81}X + \alpha^{10} & \alpha^{100}X^{27} + \alpha^{90}X^{26} + \alpha^{80}X^{25} + \alpha^{70}X^{24} + \alpha^{0}X^{23} \\
 & \quad + 0.X^{22} + 0.X^{21} + \alpha^{70}X^{20} \ldots \\
 & \quad \alpha^{100}X^{27} + \alpha^{176}X^{26} + \alpha^{96}X^{25} + \alpha^{181}X^{24} + \alpha^{110}X^{23} \\ \cline{2-2}
 & + \alpha^{77}X^{26} + \alpha^{225}X^{25} + \alpha^{61}X^{24} + \alpha^{126}X^{23} \\
 & + \alpha^{77}X^{26} + \alpha^{153}X^{25} + \alpha^{73}X^{24} + \alpha^{158}X^{23} + \alpha^{87}X^{22} \\ \cline{2-2}
 & \quad + \alpha^{93}X^{25} + \alpha^{188}X^{24} + \alpha^{161}X^{23} + \alpha^{87}X^{22} \\
 & \qquad \vdots
\end{array}$$

The division stops after 23 operations. The last part of this division is

$$\begin{array}{c} \alpha^{218}X^4 + \alpha^{41}X^3 + \alpha^{83}X^2 + \alpha^{251}X \\ \underline{\alpha^{218}X^4 + \alpha^{39}X^3 + \alpha^{284}X^2 + \alpha^{44}X + \alpha^{228}} \\ +\alpha^{89}X^3 + \alpha^{139}X^2 + \alpha^{249}X + \alpha^{228} \end{array}$$

The remainder polynomial is the calculated redundancy, which is, as said above, placed in the middle of the message vector.

The second encoder in this concatenated scheme, for the shortened RS code $C_{\text{RS}}(32, 28)$, takes the code vector of 28 bytes generated by the first encoder, and operates in the same way as that described above, in order to calculate the additional 4 bytes of redundancy. The number of steps in the division is now 27, instead of 23. The encoded vector is finally of the form

$$\begin{aligned} \mathbf{c_2} = (&\alpha^{100}\ \ \alpha^{90}\ \ \alpha^{80}\ \ \alpha^{70}\ \ \alpha^{0}\ \ 0\ \ 0\ \ \alpha^{70}\ \ \alpha^{60}\ \ \alpha^{50}\ \ \alpha^{200}\ \ \alpha^{100}\ \ \alpha^{\boldsymbol{89}}\ \ \alpha^{\boldsymbol{139}}\ \ \alpha^{\boldsymbol{249}}\ \ \alpha^{\boldsymbol{228}} \\ &\alpha^{2}\ \ \alpha^{1}\ \ \alpha^{0}\ \ 0\ \ \alpha^{4}\ \ \alpha^{3}\ \ \alpha^{2}\ \ \alpha^{1}\ \ \alpha^{40}\ \ \alpha^{30}\ \ \alpha^{20}\ \ \alpha^{10}\ \ \alpha^{\boldsymbol{144}}\ \ \alpha^{\boldsymbol{68}}\ \ \alpha^{\boldsymbol{240}}\ \ \alpha^{\boldsymbol{5}}) \end{aligned}$$

Now the redundant bytes are placed as traditionally done at the end of the resulting code vector, that is, in the least significant positions. It is noted here that in the real CD coding procedure, there is an interleaver between these two concatenated encoders.

It is practically impossible to enumerate all the code vectors that form these codes, even in the case of shortened RS codes. Any of the 256 elements of the field being used have the possibility of being in each position of the message vector of 24 bytes. This means that the total number of code vectors in the shortened RS code $C_{\text{RS}}(28, 24)$ is

$$2^{km} = 2^{24\times 8} = 6.27710 \times 10^{57}$$

This is the size of the input of the table of code vectors, and also the number of message vectors that can be encoded. The shortened RS code $C_{\text{RS}}(32, 28)$ expands this vector space into a space of the following number of code vectors:

$$2^{nm} = 2^{28\times 8} = 2.69599 \times 10^{67}$$

The relationship between these quantities is

$$2^{nm}/2^{km} = 2^{28\times 8}/2^{24\times 8} = 2^{32} = 4.29 \times 10^{9}$$

which gives an idea of the expansion capability of the encoding procedure.

5.10 Decoding of RS Codes $C_{\text{RS}}(28, 24)$ and $C_{\text{RS}}(32, 28)$

5.10.1 B–M Decoding

For both shortened versions of RS codes under analysis, the error-correction capability is $t = 2$, and so the error pattern polynomial is of the form

$$e(X) = e_{j1}X^{j_1} + e_{j2}X^{j_2} \tag{54}$$

where e_{j_1}, e_{j_2}, j_1 and j_2 are unknown variables. There are four unknown variables that can be determined by finding the solution of a linearly independent system of four equations. After determining the positions and values of these two errors, error-correction can be performed.

As explained in the previous section, the B–M algorithm [6, 7] can determine the error-location polynomial and then, together with expressions (51) and (52), it is possible to determine the positions and values of the errors. In this case, both shortened RS codes have the same error-correction capability; that is, they are able to correct any error pattern of size $t = 2$ or less. The B–M table in this case contains rows from $\mu = -1$ to $\mu = 4$. The error-location polynomial for the shortened RS codes C_{RS} (28, 24) and C_{RS} (32, 28) is of the form

$$\sigma(X) = 1 + \sigma_1 X + \sigma_2 X^2 \tag{55}$$

The following polynomial is also necessary for evaluating the error values:

$$Z(X) = 1 + (S_1 + \sigma)X + (S_2 + \sigma_1 S_1 + \sigma_2)X^2 \tag{56}$$

Then

$$\begin{aligned} e_{j_1} &= \frac{Z(\beta_1^{-1})}{\prod_{\substack{k=1\\k\neq 1}}^{2} \left(1 + \beta_k \beta_1^{-1}\right)} \\ e_{j_2} &= \frac{Z(\beta_2^{-1})}{\prod_{\substack{k=1\\k\neq 2}}^{2} \left(1 + \beta_k \beta_2^{-1}\right)} \end{aligned} \tag{57}$$

The error polynomial is thus obtained, and is then added to the received polynomial $r(X)$ to perform error correction.

Example 5.10: Decoding for the concatenation of the RS codes $C_{\text{RS}}(28, 24)$ and $C_{\text{RS}}(32, 28)$ of a code vector of 32 bytes. The received vector contains two errors at positions 8 and 16. In the particular example, errors are erasures of the elements at those positions, represented in the received vector with the symbol 0^{**}. The received vector $\boldsymbol{r}$ is the code vector calculated in Example 5.9 that when affected by the above error pattern becomes

$$\begin{aligned} \boldsymbol{r} = (&\alpha^{100}\ \ \alpha^{90}\ \ \alpha^{80}\ \ \alpha^{70}\ \ \alpha^{0}\ \ 0\ \ 0\ \ \alpha^{70}\ \ \alpha^{60}\ \ \alpha^{50}\ \ \alpha^{200}\ \ \alpha^{100}\ \ \boldsymbol{\alpha^{89}}\ \ \boldsymbol{\alpha^{139}}\ \ \boldsymbol{\alpha^{249}}\ \ \boldsymbol{0}^{**} \\ &\alpha^{2}\ \ \alpha^{1}\ \ \alpha^{0}\ \ 0\ \ \alpha^{4}\ \ \alpha^{3}\ \ \alpha^{2}\ \ 0^{**}\ \ \alpha^{40}\ \ \alpha^{30}\ \ \alpha^{20}\ \ \alpha^{10}\ \ \boldsymbol{\alpha^{144}}\ \ \boldsymbol{\alpha^{68}}\ \ \boldsymbol{\alpha^{240}}\ \ \boldsymbol{\alpha^{5}}) \end{aligned}$$

The syndrome vector components are

$$s_1 = \alpha^{31}, \qquad s_2 = \alpha^{132}, \qquad s_3 = \alpha^{121}, \qquad s_4 = \alpha^{133}$$

These values are necessary for applying the B–M algorithm (see Table 5.7).

Therefore the error-location polynomial is equal to

$$\sigma_{\text{BM}}(X) = 1 + \sigma_1 X + \sigma_2 X^2 = 1 + \alpha^{208} X + \alpha^{24} X^2$$

whose roots are

$$\beta_1^{-1} = \alpha^{239} \qquad \beta_2^{-1} = \alpha^{247}$$

Table 5.7 B–M algorithm table, Example 5.10

μ	$\sigma_{\mathrm{BM}}^{(\mu)}(X)$	d_μ	l_μ	$\mu - l_\mu$
-1	1	1	0	-1
0	1	α^{31}	0	0
1	$1+\alpha^{31}X$	α^{126}	1	0
2	$1+\alpha^{101}X$	α^{128}	1	1
3	$1+\alpha^{101}X+\alpha^{97}X^2$	α^{32}	2	1
4	$1+\alpha^{208}X+\alpha^{24}X^2$	α^{168}	2	2

Then the error-location numbers are

$$\beta_1 = \alpha^{-239} = \alpha^{255-239} = \alpha^{16} = \alpha^{j_1}$$
$$\beta_2 = \alpha^{-247} = \alpha^{255-247} = \alpha^{8} = \alpha^{j_2}$$

and so error positions are

$$j_1 = 16 \qquad j_2 = 8$$

The polynomial $Z(X)$ is, for this case,

$$Z(X) = 1 + (s_1 + \sigma)X + (s_2 + \sigma_1 s_1 + \sigma_2)X^2 = 1 + \alpha^{165}X + \alpha^{138}X^2$$

and the error values at the error positions are

$$e_{j_1} = \frac{Z\left(\beta_1^{-1}\right)}{\prod_{\substack{k=1\\k\neq 1}}^{2}\left(1+\beta_k\beta_1^{-1}\right)} = \alpha^{128}$$
$$e_{j_2} = \frac{Z\left(\beta_2^{-1}\right)}{\prod_{\substack{k=1\\k\neq 2}}^{2}\left(1+\beta_k\beta_2^{-1}\right)} = \alpha^{1} = \alpha$$

The first decoder finds these two errors and adds the error polynomial

$$e(X) = e_{j_1}X^{j_1} + e_{j_2}X^{j_2} = \alpha^{128}X^{16} + \alpha X^{8}$$

to the received polynomial, converting the received vector

$$\begin{aligned}\boldsymbol{r} = (&\alpha^{100}\ \ \alpha^{90}\ \ \alpha^{80}\ \ \alpha^{70}\ \ \alpha^{0}\ \ 0\ \ 0\ \ \alpha^{70}\ \ \alpha^{60}\ \ \alpha^{50}\ \ \alpha^{200}\ \ \alpha^{100}\ \ \boldsymbol{\alpha^{89}}\ \ \boldsymbol{\alpha^{139}}\ \ \boldsymbol{\alpha^{249}}\ \ \mathbf{0}^{**}\\ &\alpha^{2}\ \ \alpha^{1}\ \ \alpha^{0}\ \ 0\ \ \alpha^{4}\ \ \alpha^{3}\ \ \alpha^{2}\ \ 0^{**}\ \ \alpha^{40}\ \ \alpha^{30}\ \ \alpha^{20}\ \ \alpha^{10}\ \ \boldsymbol{\alpha^{144}}\ \ \boldsymbol{\alpha^{68}}\ \ \boldsymbol{\alpha^{240}}\ \ \boldsymbol{\alpha^{5}})\end{aligned}$$

into the decoded vector

$$\begin{aligned}\boldsymbol{c} = (&\alpha^{100}\ \ \alpha^{90}\ \ \alpha^{80}\ \ \alpha^{70}\ \ \alpha^{0}\ \ 0\ \ 0\ \ \alpha^{70}\ \ \alpha^{60}\ \ \alpha^{50}\ \ \alpha^{200}\ \ \alpha^{100}\ \ \boldsymbol{\alpha^{89}}\ \ \boldsymbol{\alpha^{139}}\ \ \boldsymbol{\alpha^{249}}\ \ \boldsymbol{\alpha^{128}}\\ &\alpha^{2}\ \ \alpha^{1}\ \ \alpha^{0}\ \ 0\ \ \alpha^{4}\ \ \alpha^{3}\ \ \alpha^{2}\ \ \alpha^{1}\ \ \alpha^{40}\ \ \alpha^{30}\ \ \alpha^{20}\ \ \alpha^{10}\ \ \boldsymbol{\alpha^{144}}\ \ \boldsymbol{\alpha^{68}}\ \ \boldsymbol{\alpha^{240}}\ \ \boldsymbol{\alpha^{5}})\end{aligned}$$

The first decoder ends its operation by truncating the corresponding redundancy and by passing to the second decoder the following vector:

$$\begin{aligned}\boldsymbol{c}' = (&\alpha^{100}\quad \alpha^{90}\quad \alpha^{80}\quad \alpha^{70}\quad \alpha^{0}\quad 0\quad 0\quad \alpha^{70}\quad \alpha^{60}\quad \alpha^{50}\quad \alpha^{200}\quad \alpha^{100}\quad \boldsymbol{\alpha}^{\mathbf{89}}\quad \boldsymbol{\alpha}^{\mathbf{139}}\quad \boldsymbol{\alpha}^{\mathbf{249}}\\ &\boldsymbol{\alpha}^{\mathbf{128}}\quad \alpha^{2}\quad \alpha^{1}\quad \alpha^{0}\quad 0\quad \alpha^{4}\quad \alpha^{3}\quad \alpha^{2}\quad \alpha^{1}\quad \alpha^{40}\quad \alpha^{30}\quad \alpha^{20}\quad \alpha^{10})\end{aligned}$$

The second decoder calculates the syndrome of this vector, which turns out to be the all – zero vector in this case, so that the vector is already a code vector. This example indicates that the concatenation, in this way, of two RS codes does not make much sense if both are of similar characteristics, since the error-correction capability of the scheme is almost equal to that of one of the two RS codes involved. In only a few cases is the second decoder able to correct errors effectively, as explained in more detail in Section 5.11 below. This emphasizes the importance of the interleaver actually used in the coding procedure for the CD. Finally, the decoded vector is

$$\begin{aligned}\hat{\boldsymbol{m}} = (&\alpha^{100}\quad \alpha^{90}\quad \alpha^{80}\quad \alpha^{70}\quad \alpha^{0}\quad 0\quad 0\quad \alpha^{70}\quad \alpha^{60}\quad \alpha^{50}\quad \alpha^{200}\quad \alpha^{100}\quad \alpha^{2}\quad \alpha^{1}\quad \alpha^{0}\quad 0\\ &\alpha^{4}\quad \alpha^{3}\quad \alpha^{2}\quad \alpha^{1}\quad \alpha^{40}\quad \alpha^{30}\quad \alpha^{20}\quad \alpha^{10})\end{aligned}$$

which is the true message vector transmitted in this example.

5.10.2 Alternative Decoding Methods

As pointed out in previous sections, any algorithm able to solve the system of equations (19) can be used in a decoding algorithm for RS codes. Among these algorithms are the Euclidean algorithm and the B–M algorithm, both introduced in previous sections in this chapter. In the particular case of the concatenation of the RS codes $C_{\text{RS}}(28, 24)$ and $C_{\text{RS}}(32, 28)$, the error-correction capability of each of these codes is $t = 2$, so that the error pattern polynomial is of the form of (54), in which e_{j_1}, e_{j_2}, X^{j_1} and X^{j_2} are the unknown variables that are the positions and values of the two possible errors. Since the number of unknown variables is 4, there must be a system of at least four equations to uniquely determine these unknown variables. This system of equations comes from the calculation of the four syndrome vector components obtained by replacing the variable X in the expression of the received polynomial $r(X)$ by the roots of the corresponding Galois field $\alpha, \alpha^2, \alpha^3$ and α^4. The received vector is considered to be a valid code vector if the four components of the syndrome vector are all equal to zero. Otherwise there is at least one error in the received vector. If the number of errors is equal to the minimum distance of the code, in this case $d_{\min} = 5$, then the received vector may convert into another code vector. In this case the error pattern is beyond the error-correction capability of the code.

The most well-known algorithms for decoding RS codes are those already introduced in previous sections, the Euclidean and the B–M algorithms [2, 4, 6, 7], which can be implemented in either the time domain (as described in this chapter) or the finite field transform (spectral) domain [16]. Other algorithms can also be implemented. Among them, there exists a decoding algorithm based on the direct solution of the corresponding system of equations, which, in this particular case, is of low complexity because a maximum of only two errors ($t = 2$) have to be determined.

5.10.3 Direct Solution of Syndrome Equations

If the maximum number of errors to be corrected is relatively small, as it is in this case, then the direct solution of the corresponding system of syndrome equations can be a simpler alternative to the well-known decoding algorithms. In this case, the system of syndrome equations is of the form

$$\begin{aligned} s_1 &= e(\alpha) = e_{j_1}\beta_1 + e_{j_2}\beta_2 \\ s_2 &= e(\alpha^2) = e_{j_1}\beta_1^2 + e_{j_2}\beta_2^2 \\ s_3 &= e(\alpha^3) = e_{j_1}\beta_1^3 + e_{j_2}\beta_2^3 \\ s_4 &= e(\alpha^4) = e_{j_1}\beta_1^4 + e_{j_2}\beta_2^4 \end{aligned} \tag{58}$$

where

$$\begin{aligned} \beta_1 &= \alpha^{j_1} \\ \beta_2 &= \alpha^{j_2} \end{aligned} \tag{59}$$

From equation (58), we can form the terms

$$\begin{aligned} &s_1 s_3 + s_2^2 \\ &s_1 s_4 + s_2 s_3 \\ &s_2 s_4 + s_3^2 \end{aligned} \tag{60}$$

and by multiplying these terms by β_1^2, β_1 and 1, respectively, the following equation is obtained:

$$\left(s_1 s_3 + s_2^2\right)\beta_1^2 + (s_1 s_4 + s_2 s_3)\beta_1 + s_2 s_4 + s_3^2 = 0$$

In the same way, but multiplying now by β_2^2, β_2 and 1, respectively, the following equation is also obtained:

$$\left(s_1 s_3 + s_2^2\right)\beta_2^2 + (s_1 s_4 + s_2 s_3)\beta_2 + s_2 s_4 + s_3^2 = 0$$

These last two equations are almost the same, the difference being that the former is expressed in β_1 and the latter is expressed in β_2. They can however be combined into one equation as

$$\left(s_1 s_3 + s_2^2\right)\beta^2 + (s_1 s_4 + s_2 s_3)\beta + s_2 s_4 + s_3^2 = 0 \tag{61}$$

where $\beta = \beta_1, \beta_2$ are the two roots of equation (61). Roots for this equation can be found by using the Chien search over the possible values of β, which are the positions numbered from 0 to 31 (α^0 and α^{31}) for the case of the RS code $C_{\mathrm{RS}}(32, 28)$, and from 0 to 27 (α^0 and α^{27}) for the case of the RS code $C_{\mathrm{RS}}(28, 24)$, leading to the solution of roots β_1 and β_2. Another way is that once one of the roots has been determined, then the other one can be obtained by using

$$\beta_2 = \beta_1 + \frac{s_1 s_4 + s_2 s_3}{s_1 s_3 + s_2^2} \tag{62}$$

an expression that comes from the well-known relationship of the roots of a quadratic equation. Once the values of the roots are determined, equation (58) can be solved to calculate the error values. On the one hand,

$$s_2 + s_1\beta_2 = e_{j_1}\left(\beta_1^2 + \beta_1\beta_2\right)$$

and thus

$$e_{j_1} = (s_2 + s_1\beta_2)/\left(\beta_1^2 + \beta_1\beta_2\right) \tag{63}$$

On the other hand,

$$s_2 + s_1\beta_1 = e_{j_2}\left(\beta_2^2 + \beta_1\beta_2\right)$$

and thus

$$e_{j_2} = (s_2 + s_1\beta_1)/\left(\beta_2^2 + \beta_1\beta_2\right) \tag{64}$$

If the number of errors in the received vector is 2, then the solution is unique. Error correction is then performed by adding the received vector to the error vector.

The complexity of this proposed direct-solution algorithm is less than the Euclidean algorithm complexity, since equation (61) is directly obtained with the syndrome vector components, and its roots are the error-location numbers β_1 and β_2, whose calculation leads to the whole solution of the system. There is no need to look for the error-location polynomial $\sigma(X)$. It is just that the left-hand term in equation (61) is somehow an error-location polynomial for this method, since if β is replaced by $1/X$, and if the resulting polynomial is normalized to be monic, then this expression becomes the error-location polynomial of the Euclidean algorithm (see Examples 5.5 and 5.7).

If the number of errors is different from 2, then it is quite likely that the system of syndrome equations will be incorrectly solved. Therefore a suitable decoding process consists of three steps to sequentially evaluate three different situations. First, the four syndrome vector components have to be calculated. If all these components are equal to zero, the decoding procedure adopts the received vector as a code vector. Otherwise the decoder assumes that the received vector has only one error, and performs error correction according to expressions (21), which are valid for determining the position and the value of one error. This procedure is very simple. Expressions (21) can be used to calculate the magnitude and position of the error, and then the error correction is performed by using (16). After performing this single-error correction, the syndrome of the corrected vector is calculated. If the syndrome vector is the all-zero vector, then the decoder considers that this correction was successful, and that the corrected vector is a code vector. The decoding procedure halts at this step and proceeds to the next received vector. Otherwise the decoder assumes that the received vector contains two errors, and proceeds to calculate the roots of equation (61) by means of the Chien search. After determining the values and positions of the two errors, error correction is again performed, another corrected vector is evaluated and the syndrome vector components are determined for this corrected vector. If after the assumption of a two-error pattern the correction is successful; that is, the syndrome calculated over the corrected vector has all its components equal to zero, then the corrected vector is considered as a code vector. Otherwise the received vector is left as it has been received, and the decoder proceeds to decode the next received vector.

Example 5.11: Solve Example 5.5, using expressions (61)–(64).

According to the received polynomial

$$r(X) = \alpha^6 X^2 + \alpha^4 X^4$$

the syndrome vector components are

$$\begin{aligned} s_1 &= r(\alpha) = \alpha^8 + \alpha^8 = 0 \\ s_2 &= r(\alpha^2) = \alpha^3 + \alpha^5 = \alpha^6 \\ s_3 &= r(\alpha^3) = \alpha^5 + \alpha^2 = \alpha^4 \\ s_4 &= r(\alpha^4) = 1 + \alpha^6 = \alpha^4 \end{aligned}$$

Equation (61) is of the form

$$\begin{aligned} &\left(s_1 s_3 + s_2^2\right)\beta^2 + (s_1 s_4 + s_2 s_3)\beta + s_2 s_4 + s_3^2 \\ &= [0.\alpha^4 + (\alpha^6)^2]\beta^2 + [0.\alpha^4 + \alpha^6\alpha^4]\beta + \alpha^6\alpha^4 + (\alpha^4)^2 \\ &= \alpha^5\beta^2 + \alpha^3\beta + \alpha^4 \\ &= 0 \end{aligned}$$

This equation has two roots, $\beta_1 = \alpha^2$ and $\beta_2 = \alpha^4$. This means that the first error is at position $j_1 = 2$, and the other error is at position $j_2 = 4$.

Values of the errors are calculated by using (63) and (64):

$$e_{j_1} = \frac{(s_2 + s_1\beta_2)}{\left(\beta_1^2 + \beta_1\beta_2\right)} = \frac{\alpha^6 + 0.\alpha^4}{\alpha^4 + \alpha^2\alpha^4} = \alpha^6$$

$$e_{j_2} = \frac{(s_2 + s_1\beta_1)}{\left(\beta_2^2 + \beta_1\beta_2\right)} = \frac{\alpha^6 + 0.\alpha^2}{\alpha^8 + \alpha^2\alpha^4} = \alpha^4$$

The error polynomial is then

$$e(X) = \alpha^6 X^2 + \alpha^4 X^4$$

and the transmitted vector was the all-zero vector.

5.11 Importance of Interleaving

In general terms, concatenation of RS codes is not efficient without interleaving [8, 9, 15]. Burst errors are very common in the CD system, since information is printed in digital format over the disk surface as a long and continuous spiral line, and so any scratch, rip or mark over the surface can produce a serious damage of the printed information, that is, essentially a chain or burst of errors in the digital sequence. Interleaving plays an important role in reducing the effect of this sort of error event.

At first sight, it seems that the concatenation of the two shortened RS codes of the CD system, with error-correction capability $t = 2$ each, could have an error-correction capability of $t = 4$ errors in a code vector of 32 bytes, since the first encoder adds four redundancy elements to correct two errors, and the second encoder does the same to correct another two errors. From this point of view, it would be necessary to solve a system of eight syndrome equations, in order to determine four error positions and their four error values.

However, the most common way of decoding a serial concatenation of codes is to first perform the operation of decoding the inner code, and then to pass the resulting vector to the outer decoder. In this case, most of the error patterns of size larger than 2, $t > 2$, make the decoder of the RS code $C_{\mathrm{RS}}(32, 28)$ collapse, because its corresponding system of syndrome equations cannot be solved properly, and so this decoder passes the received vector to the outer decoder, which in turn cannot correct that error pattern either. Thus the concatenated system is in the end incapable of correcting more than a few error patterns of size $t = 3$ or $t = 4$. The above explanation is true for the concatenation of two RS codes without using interleaving in between the two codes. Here is noted the importance of the interleaving/de-interleaving procedure, which causes a given burst error pattern, which is essentially a long chain of consecutive errors, be distributed over many different consecutive code vectors, each one containing a small number of bytes in error. Thus, for instance, a burst error pattern of three errors, serially decoded without interleaving between the codes, generally cannot be corrected. However, this error pattern is converted by the interleaving procedure into single error patterns in three different received vectors, making the three-error event correctable, since the serial decoders can manage error events of size $t = 1$. This is the essence of interleaving; that is, the interleaving somehow randomizes and distributes a burst over many received vectors, reducing the size of the error event per received vector.

By making use of the interleaver, the most common way of decoding in the CD system is to use the first decoder in erasure mode; that is, in a mode where it first detects errors and then erases them, before passing the resulting vector to the second decoder. This means that the first decoder performs only error detection.

If the first decoder detects errors in the received vector, then it erases all its positions, de-interleaves the erased received vector and passes it to the second decoder. The second decoder knows which positions are erased, and therefore knows the positions of the possible errors. The system of four syndrome equations of the second decoder is then able to determine the error values in up to four of these error positions, thus performing the correction of error patterns of size $t = 4$ or less. The relationship between the positions of the code vectors of the RS codes $C_{\mathrm{RS}}(32, 28)$ and $C_{\mathrm{RS}}(28, 24)$ is illustrated in Figure 5.5.

The vector of 32 bytes is input to the first decoder. After determining the syndrome vector and detecting errors, it erases and reorders the received vector, taking out the four parity bytes at positions 0, 1, 2 and 3 of the vector of 32 bytes, and passes to the second decoder a vector of 28 bytes. This second decoder performs error correction and takes out bytes at positions 0, 1, 2 and 3 of the vector of 28 bytes, to obtain the decoded vector of 24 bytes. This procedure would be implemented taking into account the interleaver between the two codes in the CD system. Since the first decoder erases those received vectors found in error, and passes the erased vector via the de-interleaver to the second decoder, the whole error-control coding scheme for the CD is capable of correcting any error pattern of size $t = 4$ or less. The second decoder, which corresponds to the RS code $C_{\mathrm{RS}}(28, 24)$, corrects these error patterns by solving a system of

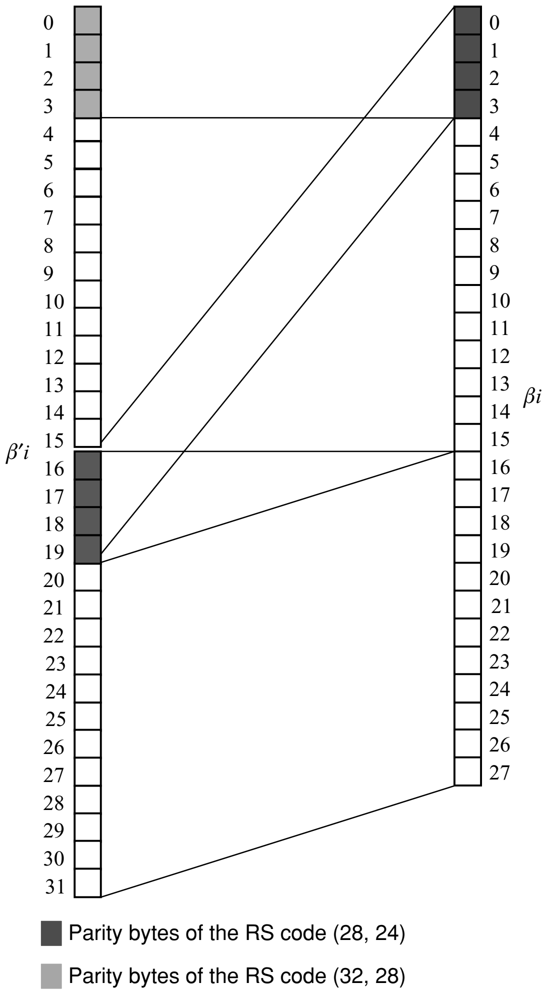


Figure 5.5 Relationship between positions of code vectors of the RS codes $C_{\mathrm{RS}}(32, 28)$ and $C_{\mathrm{RS}}(28, 24)$

equations of the form

$$\begin{aligned}
s_1 &= r(\alpha) = e_{j_1}\beta_1 + e_{j_2}\beta_2 + e_{j_3}\beta_3 + e_{j_4}\beta_4 \\
s_2 &= r(\alpha^2) = e_{j_1}(\beta_1)^2 + e_{j_2}(\beta_2)^2 + e_{j_3}(\beta_3)^2 + e_{j_4}(\beta_4)^2 \\
s_3 &= r(\alpha^3) = e_{j_1}(\beta_1)^3 + e_{j_2}(\beta_2)^3 + e_{j_3}(\beta_3)^3 + e_{j_4}(\beta_4)^3 \\
s_4 &= r(\alpha^4) = e_{j_1}(\beta_1)^4 + e_{j_2}(\beta_2)^4 + e_{j_3}(\beta_3)^4 + e_{j_4}(\beta_4)^4
\end{aligned} \tag{65}$$

where positions $\beta_i = \alpha^{j_i}$, $0 \leq j_i < 28$, are known, as they are passed from the first decoder to the second one. It is necessary to determine only the values of errors e_{j_i}. Interleaving provides

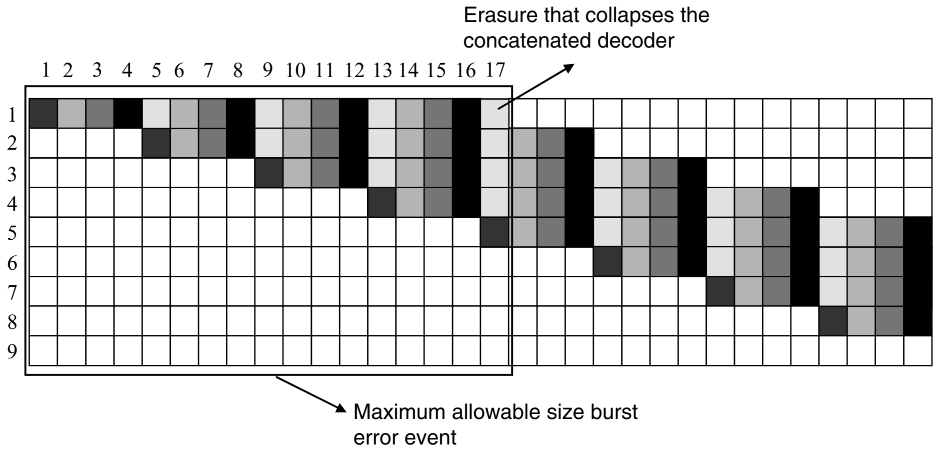

Erasure that collapses the concatenated decoder
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
1
2
3
4
5
6
7
8
9
Maximum allowable size burst error event

Figure 5.6 Limiting burst error pattern of error-correction coding for the CD

the whole coding system with a higher error-correction capability. This is briefly described in Figure 5.6.

In Figure 5.6 the horizontal axis is the time sequential order in which bytes are transmitted, whereas in the vertical axis the vector of 28 bytes received from the decoder of the RS code $C_{\mathrm{RS}}(32, 28)$ can be observed. This figure shows only the first 17 steps of the interleaving, and only 9 bytes of the vector of 28 bytes. The whole array would have 112 columns and 28 rows, and it would be an array of vectors that the first decoder passes to the second decoder. Each vector is distributed by the de-interleaving process, placing its bytes with consecutive delays of size 4D, D being the size of a byte. The first vector coming from the decoder of the RS code $C_{\mathrm{RS}}(32, 28)$ is for instance placed at positions numbered (1, 1), (2, 5), (3, 9), (4, 13), etc., as depicted in Figure 5.6 with oblique lines. The 4 bytes provided by the first encoder are discarded here. If for instance two vectors of 32 bytes were found to have errors and erased, the second erased vector would be placed in positions numbered (1, 2), (2, 6), (3, 10), (4, 14), and so on. When the decoder of the RS code $C_{\mathrm{RS}}(32, 28)$ receives a vector of 32 bytes, it calculates the syndrome vector components, and if this vector is different from the all-zero vector, then instead of performing error correction, the decoder erases all the positions of the received vector. After this, the decoder takes out the 4 bytes of redundancy and places as a column the 28 resulting bytes, in 28 different columns, so that the first byte is the first in column 1, the second byte is the second in column 5, the third byte is the third in column 9, and so on. Thus, the 28 received bytes of the received vector of 32 bytes that result from the removal of the first four redundant bytes become single bytes in 1 of 28 different columns or vectors. If for instance a burst error pattern of 32 bytes occurs, which could not be corrected if the concatenated system does not use interleaving, it is converted by the interleaving procedure into a series of 28 vectors with only one error in each, now able to be corrected by the second decoder that corresponds to the RS code $C_{\mathrm{RS}}(28, 24)$.

The erasure technique is not very efficient if the received vector has only few errors, because the first decoder erases valid bytes, but if as happens usually in the CD channel a burst of errors affects the received vector, the efficiency is very high. When a burst of 17 vectors of

32 bytes each affects the transmission, which is an error event that can be seen as affecting the first 17 columns in Figure 5.6, an erasure of 5 bytes happens in the column 17 of that array, which is a received vector for the second decoder. This column contains bytes of received vectors numbered 1, 5, 9, 13 and 17, which were received and erased by the first decoder after truncating 4 bytes of redundancy. The second decoder [RS code $C_{\mathrm{RS}}(28, 24)$] cannot correct this error pattern. Therefore the whole system is able to correct a burst error event of 16 vectors of 32 bytes, and so the total number of bits that can be corrected in a burst error pattern is

$$16 \times 24 \times 8 = 3072 \text{ bits}$$

As seen in the above description of the interleaving procedure, there is a big difference in the error-correction capability of the concatenated RS codes when interleaving is used. The difference with respect to the concatenation of the same RS codes without interleaving is now evident, as in this latter case, error events of more than 2 bytes in a given vector normally cannot be corrected. As pointed out previously, the concatenation of the two RS codes without interleaving has an error-correction capability similar to that of only one of the RS codes used.

RS codes demonstrate a very strong error-correction capability, especially against the burst errors that happen in mobile communications and in the reading process of a CD. One of the reasons for such a high correction capability is that RS codes are non-binary, and their error correction is performed over an element of a Galois field, that is, over a group of bits, no matter whether one or all of them are in error. This error-correction capability increases enormously if these codes are implemented in serial concatenation with interleaving, which has the ability to spread burst errors into several successive outer code vectors, converting burst errors into random-like errors.

Bibliography and References

[1] Reed, I. S. and Solomon, G., “Polynomial codes over certain finite fields,” *J. Soc. Ind. Appl. Math.*, vol. 8, pp. 300–304, 1960.

[2] Lin, S. and Costello, D. J., Jr., *Error Control Coding: Fundamentals and Applications*, Prentice Hall, Englewood Cliffs, New Jersey, 1983.

[3] Blaum, M., *A Course on Error Correcting Codes*, 2001.

[4] Berlekamp, E. R., *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.

[5] Chien, R. T., “Cyclic decoding procedure for the Bose–Chaudhuri–Hocquenghem codes,” *IEEE Tans. Inf. Theory*, vol. IT-10, pp. 357–363, October 1964.

[6] Massey, J. L., “Step-by-step decoding of the Bose–Chaudhuri–Hocquenghem codes,” *IEEE Trans. Inf. Theory*, vol. IT-11, pp. 580–585, October 1965.

[7] Berlekamp, E. R., “On decoding binary Bose–Chaudhuri–Hocquenghem codes,” *IEEE Trans. Inf. Theory*, vol. IT-11, pp. 577–580, October 1965.

[8] Sklar, B., *Digital Communications, Fundamentals and Applications,* Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[9] Wicker, S. B. and Bhargava, V. K., *Reed–Solomon Codes and Their Applications*, IEEE Press, New York, 1994.

[10] Peek, J. B. H., “Communications aspects of the compact disc audio system,” *IEEE Commun. Mag.*, vol. 23, no. 2, pp. 7–15, February 1985.

[11] Hoeve, H., Timmermans, J. and Vries, L. B., "Error correction and concealment in the compact disc system," *Philips Tech. Rev.*, vol. 40, no. 6, pp. 166–172, 1982.

[12] Immink, K. A. S., *Coding Techniques for Digital Recorders*, Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[13] Heemskerk, J. P. J. and Immink, K. A. S., "Compact disc: System aspects and modulation," *Philips Tech. Rev.*, vol. 40, pp. 157–164, 1982.

[14] Immink, K. A. S, Nijboer, J. G., Ogawa, H. and Odaka, K., "Method of coding binary data," *United States Patent 4,501,000*, February 1985.

[15] Castiñeira Moreira, J., Markarian, G. and Honary, B., *An Improvement of Write/Read Characteristics in Optical Storage Systems (E. G. Compact Discs and CD-Roms)*, MSc Project Report, Lancaster University, Lancaster, United Kingdom, 1996.

[16] Blahut, R. E., "Transform techniques for error control codes," *IBM J. Res. Dev.*, vol. 23, no. 3, May 1979.

[17] Sloane, N. J. A. and Peterson, W. W., *The Theory of Error-Correcting Codes*, North-Holland, Amsterdam, The Netherlands, 1998.

[18] Adámek, J., *Foundations of Coding: Theory and Applications of Error-Correcting Codes with an Introduction to Cryptography and Information Theory*, Wiley Interscience, New York, 1991.

[19] Massey, J. L. and Blahut, R. E., *Communications and Cryptography: Two Sides of One Tapestry,* Kluwer, Massachusetts, 1994.

Problems

5.1 A cyclic code over GF(4) has the generator polynomial $g(X) = X + 1$, and is to have a code length $n = 3$. The field elements are generated by the polynomial $\alpha^2 + \alpha + 1 = 0$. Find the generator matrix of the code in systematic form, the minimum Hamming distance of the code and the syndrome vector if the received vector is $\boldsymbol{r} = (\alpha\ \alpha\ \alpha)$.

5.2 (a) Determine the generator polynomial of an RS code $C_{\mathrm{RS}}(n, k)$ that operates over the field GF(2^4) and is able to correct any error pattern of size $t = 2$ or less.

(b) For the RS code of item (a), decode the received polynomial $r(X) = \alpha X^3 + \alpha^{11} X^7$ by using the Euclidean algorithm.

(c) For the RS code of item (a), decode the received polynomial $r(X) = \alpha^8 X^5$.

5.3 (a) Determine the generator polynomial of an RS code that operates over the field GF(2^4) and is able to correct any error pattern of size $t = 3$ or less.

(b) Determine the values of n and k.

5.4 For the RS code of Problem 5.3, decode the received vector $\boldsymbol{r} = (000\alpha^7\,00\alpha^3\ 00000\alpha^4\,00)$ by using the Euclidean and the B–M algorithms.

5.5 Consider the RS code with symbols in GF(2^3), with three information symbols, $k = 3$, code length $n = 7$, and the generator polynomial $g(X) = (X + \alpha^2)(X + \alpha^3)(X + \alpha^4)(X + \alpha^5)$ where α is a root of the primitive polynomial $p_i(X) = 1 + X + X^3$ used to represent the elements of GF(2^3).

(a) How many symbol errors can this code correct?

(b) Decode the received vector $\boldsymbol{r} = (0110111)$ to determine the transmitted code vector.

5.6 The redundant symbols of a ½-rate RS code over GF(5) with code length 4 are given by

$$c_1 = 4k_1 + 2k_2$$

$$c_2 = 3k_1 + 3k_2$$

(a) Find the number of code vectors, the generator matrix and the Hamming distance of the code.

(b) Show that the vector (1134) is not a code vector, and find the closest code vector.

5.7 An extended RS code over GF(2^2) has the following generator matrix:

$$\boldsymbol{G} = \begin{bmatrix} 1 & \alpha & 1 & 0 & 0 \\ 1 & \alpha^2 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

(a) What is the rate of the code, and its minimum Hamming distance?

(b) Is $\boldsymbol{r} = (\alpha^2 \quad \alpha \quad \alpha \quad 0 \quad 1)$ a code vector of this code?

(c) The received vector $\boldsymbol{r} = (0 \quad \alpha \quad 1 \quad \alpha^2 \quad 0)$ contains a single error: find its position and value.

5.8 Consider a shortened RS code $C_{\mathrm{RS}}(8, 4)$ operating over the Galois field GF(2^4) with error-correction capability $t = 2$.

(a) Obtain its generator polynomial, and then the code vector for the message vector $\boldsymbol{m} = (\alpha^4 \quad \alpha^7 \quad 0 \quad \alpha^5)$.

(b) Consider now that this code vector is input to a second shortened RS code $C_{\mathrm{RS}}(12, 8)$, also operating over the same field, and with the same generator polynomial and error-correction capability as the shortened RS code $C_{\mathrm{RS}}(8, 4)$. Determine the resulting concatenated code vector.

(c) Use either the Euclidean or the B–M decoding algorithm to decode the resulting concatenated code vector of item (b) when it is affected by either the error pattern $e(X) = X^3 + X^{10} + X^{11}$ or the error pattern $e(X) = X + X^6 + X^9$.

5.9 The concatenated scheme of Problem 5.8 is now constructed using between the two concatenated codes a convolutional interleaver like that seen in Figure P.5.1. After the first encoding, a word of eight elements is generated. The first element of this word is placed in the first position of the first column of the interleaver, the

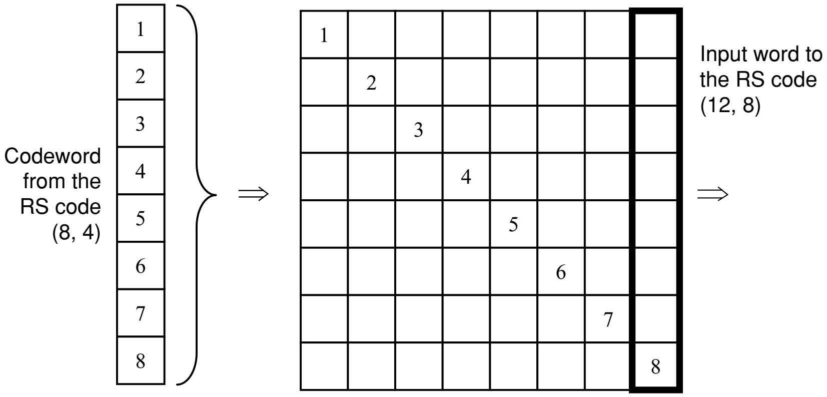

1
2
3
4
5
6
7
8
Codeword from the RS code (8, 4)
⇒
1
2
3
4
5
6
7
8
Input word to the RS code (12, 8)
⇒

Figure P.5.1 An interleaver for a concatenation of two RS codes

second element in the second position of the second column of the interleaver, and so on. Then the resulting word is input to the second encoder to finally generate a codeword of 12 elements. Determine the increased error-correction capability of this scheme with respect to the direct concatenation performed in Problem 5.8.

■

6

Convolutional Codes

A second important technique in error-control coding is that of convolutional coding [1–6]. In this type of coding the encoder output is not in block form, but is in the form of an encoded sequence generated from an input information sequence. The encoded output sequence is generated from present and previous message input elements, in a continuous encoding process that creates redundancy relationships in the encoded sequence of elements. A given message sequence generates a particular encoded sequence. The redundancy in the encoded sequence is used by the corresponding decoder to infer the message sequence by performing error correction. The whole set of encoded sequences form a convolutional code C_{conv}, where there exists a bijective (one-to-one) relationship between message sequences and encoded sequences.

From this point of view, a sequence can also be considered as a vector. Then, message sequences belong to a message vector space, and encoded sequences belong to a code vector space. Message sequence vectors are shorter than code sequence vectors, and so there are potentially many more possible code sequences than message sequences, which permits the selection of code sequences containing redundancy, thus allowing errors to be corrected. The set of selected sequences in the code vector space is the convolutional code. A suitable decoding algorithm can allow us to determine the message sequence as a function of the received sequence, which is the code sequence affected by the errors on the channel.

In general terms, convolutional encoding is designed so that its decoding can be performed in some structured and simplified way. One of the design assumptions that simplifies decoding is linearity of the code. For this reason, linear convolutional codes are preferred. The source alphabet is taken from a finite field or Galois field GF(q). The message sequence is a sequence of segments of k elements that are simultaneously input to the encoder. For each segment of k elements that belongs to the extended vector space $[\mathrm{GF}(q)]^k$, the encoder generates a segment of n elements, $n > k$, which belongs to the extended vector space $[\mathrm{GF}(q)]^n$. Unlike in block coding, the n elements that form the encoded segment do not depend only on the segment of k elements that are input at a given instant i, but also on the previous segments input at instants $i-1, i-2, \ldots, i-K$, where K is the memory of the encoder. The higher the level of memory, the higher the complexity of the convolutional decoder, and the stronger the error-correction capability of the convolutional code. Linear convolutional codes are a subspace of dimension k of the vector space $[\mathrm{GF}(q)]^n$ defined over GF(q). Linear convolutional codes exist with elements from GF(q), but in most practical applications, however, message and code sequences are composed of elements of the binary field GF(2), and the most common structure of the corresponding convolutional code utilizes $k = 1, n = 2$. A convolutional code with parameters n, k and K will be denoted as $C_{\text{conv}}(n, k, K)$.

6.1 Linear Sequential Circuits

Linear sequential circuits are an important part of convolutional encoders. They are constructed by using basic memory units, or delays, combined with adders and scalar multipliers that operate over GF(q). These linear sequential circuits are also known as finite state sequential machines (FSSMs) [7]. The number of memory units, or delays, defines the level of memory of a given convolutional code $C_{\text{conv}}(n, k, K)$, determining also its error-correction capability. Each memory unit is assigned to a corresponding state of the FSSM. Variables in these machines or circuits can be bits, or a vector of bits understood as an element of a field, group or ring over which the FSSM is defined [9–15]. In these algebraic structures there is usually a binary representation of the elements that adopt the form of a vector of components taken from GF(2).

FSSM analysis is usually performed by means of a rational transfer function $G(D) = P(D)/Q(D)$ of polynomial expressions in the D domain, called the delay domain, where message and code sequences adopt the polynomial form $M(D)$ and $C(D)$, respectively. For multiple input–multiple output FSSMs, the relationship between the message sequences and the code sequences is described by a rational transfer function matrix $\boldsymbol{G}(D)$.

A convolutional encoder is basically a structure created using FSSMs that for a given input sequence generates a given output sequence. The set of all the code sequences constitutes the convolutional code C_{conv}.

6.2 Convolutional Codes and Encoders

A convolutional encoder takes a k-tuple $\boldsymbol{m}_i$ of message elements as the input, and generates the n-tuple $\boldsymbol{c}_i$ of coded elements as the output at a given instant i, which depends not only on the input k-tuple $\boldsymbol{m}_i$ of the message at instant i but also on previous k-tuples, $\boldsymbol{m}_j$ present at instants $j < i$.

As an example, Figure 6.1 shows a convolutional encoder that at each instant i takes two input elements and generates at the same instant three output elements. Rectangular blocks identify memory units or delays of duration D, which is defined as the time unit of the FSSM,

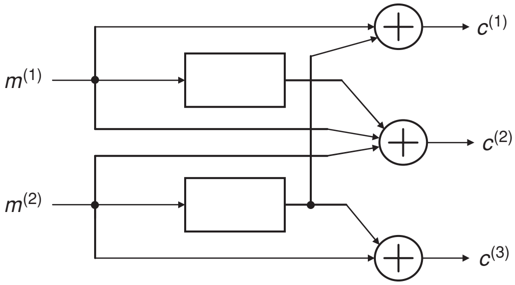


Figure 6.1 A convolutional encoder

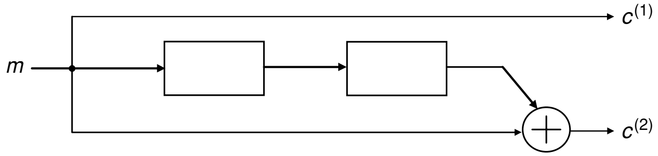


Figure 6.2 Structure of a systematic convolutional encoder of rate $R_c = 1/2$

and is normally equal to the duration of an element of the Galois field GF(q) over which the FSSM operates. The circles containing plus signs represent GF(q) adders, and it is also possible to have GF(q) multipliers. In general terms, equivalent convolutional encoders can exist, that is, convolutional encoders of different structures that generate the same convolutional code C_{conv}.

The quotient between the number of input elements k and the number of output elements n defines what is called the rate of the convolutional code, $R_c = k/n$. In the case of the convolutional encoder of Figure 6.1, for instance, this parameter is $R_c = 2/3$.

A differently structured convolutional encoder is shown in Figure 6.2, which is called a systematic encoder, since the message elements appear explicitly in the output sequence together with the redundant elements. In this case the rate of the convolutional code is $R_c = 1/2$.

Properties of convolutional coding will be illustrated by means of an example based on the convolutional encoder seen in Figure 6.3. This convolutional encoder is an FSSM that operates over the binary field GF(2) where the input message k-tuple is simply one bit, $\boldsymbol{m}$, and at each instant i the encoder generates an output n-tuple of two bits $c_i^{(1)}$ and $c_i^{(2)}$. The input sequence $\boldsymbol{m}_i = (m_0, m_1, m_2, \ldots)$ generates two output sequences $\boldsymbol{c}^{(1)} = (c_0^{(1)}, c_1^{(1)}, c_2^{(1)}, \ldots)$ and $\boldsymbol{c}^{(2)} = (c_0^{(2)}, c_1^{(2)}, c_2^{(2)}, \ldots)$. These two output sequences can be obtained as the convolution between the input sequence and the two impulse responses of the encoder defined for each of its outputs. Impulse responses can be obtained by applying the unit impulse input sequence $\boldsymbol{m} = (1, 0, 0, \ldots)$ and observing the resulting outputs $c_i^{(1)}$ and $c_i^{(2)}$. In general, a convolutional encoder has K memory units (2 in this example), counting units in parallel (which occur when $k > 1$) as a single unit, so that impulse responses extend for no more than $K + 1$ time units, and are sequences of the form

$$\boldsymbol{g}^{(1)} = \left(g_0^{(1)}, g_1^{(1)}, g_2^{(1)}, \ldots, g_K^{(1)}\right)$$
$$\boldsymbol{g}^{(2)} = \left(g_0^{(2)}, g_1^{(2)}, g_2^{(2)}, \ldots, g_K^{(2)}\right) \tag{1}$$

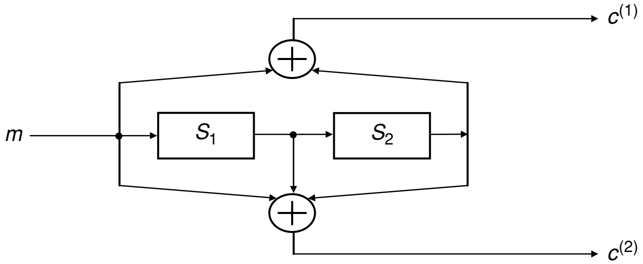


Figure 6.3 Convolutional encoder of rate $R_c = 1/2$

Table 6.1 Generator sequences for the FSSM of Figure 6.3

i	m	$\boldsymbol{S}_1$	$\boldsymbol{S}_2$	$\boldsymbol{c}^{(1)}$	$\boldsymbol{c}^{(2)}$
0	1	0	0	1	1
1	0	1	0	0	1
2	0	0	1	1	1
3	0	0	0	0	0

Impulse response analysis can be applied to the convolutional encoder, as seen in Figure 6.3, assuming that the FSSM starts in the all-zero state (00). Any state of the FSSM is described by the state vectors $\boldsymbol{S}_1 = (s_{01}, s_{11}, s_{21}, \ldots)$ and $\boldsymbol{S}_2 = (s_{02}, s_{12}, s_{22}, \ldots)$. For a given input sequence, the evolution of the FSSM can be easily observed by means of a table that describes all the parameters of that FSSM. Table 6.1 shows the evolution of the FSSM of Figure 6.3, for the unit impulse input.

The value $K + 1$ (3 for this example) is called the constraint length of the convolutional code C_{conv}. It is measured in time units, and is the maximum number of time units that a given bit of the input sequence can influence the output sequence values.

If the input is the unit impulse, then $\boldsymbol{c}^{(1)} = g^{(1)}$ and $\boldsymbol{c}^{(2)} = \boldsymbol{g}^{(2)}$. For this example,

$$\boldsymbol{g}^{(1)} = (101)$$
$$\boldsymbol{g}^{(2)} = (111)$$

These vectors describe the impulse responses of the FSSM and they are also a description of the connections of the structure of the FSSM, so that when a given memory unit is connected to an output, the corresponding bit in the impulse response vector is ‘1’, whereas for an absent connection this bit is ‘0’.

The impulse responses are also known as the generator sequences of the convolutional code C_{conv}. From this point of view, it is possible to express the encoded sequences as

$$\boldsymbol{c}^{(1)} = \boldsymbol{u} * \boldsymbol{g}^{(1)}$$
$$\boldsymbol{c}^{(2)} = \boldsymbol{u} * \boldsymbol{g}^{(2)} \tag{2}$$

where the operator ‘$*$’ denotes discrete convolution modulo 2. This means that for an integer number $l \geq 0$,

$$c_l^{(j)} = \sum_{i=0}^{K} m_{l-i} g_i^{(j)} = m_l g_0^{(j)} + m_{l-1} g_1^{(j)} + \cdots + m_{l-K} g_K^{(j)} \tag{3}$$

In the particular case of the example under analysis, the FSSM of Figure 6.3,

$$c_l^{(1)} = \sum_{i=0}^{2} m_{l-i} g_i^{(1)} = m_l + m_{l-2}$$

$$c_l^{(2)} = \sum_{i=0}^{2} m_{l-i} g_i^{(2)} = m_l + m_{l-1} + m_{l-2}$$

Both output sequences are concatenated to form a unique output or code sequence:

$$\boldsymbol{c} = \left(c_0^{(1)} c_0^{(2)}, c_1^{(1)} c_1^{(2)}, c_2^{(1)} c_2^{(2)}, \ldots\right)$$

6.3 Description in the D-Transform Domain

As is well known in the field of signals and their spectra, convolution in the time domain becomes multiplication in the spectral domain. This suggests a better description of convolutional codes based on expressions given in the D-transform domain. In this domain, also called the delay domain D, convolution ($*$) becomes multiplication, so that sequences adopt a polynomial form expressed in the variable D, where the exponent of this variable determines the position of the element in the sequence represented.

The message sequence $\boldsymbol{m}^{(l)} = (m_0^{(l)}, m_1^{(l)}, m_2^{(l)}, \ldots)$ can be represented in polynomial form as

$$M^{(l)}(D) = m_0^{(l)} + m_1^{(l)} D + m_2^{(l)} D^2 + \cdots \tag{4}$$

Delay D can be interpreted as a shift parameter, and it plays the same role as the term Z^{-1} in the Z transform.

Impulse responses can also adopt a polynomial form

$$\boldsymbol{g}_{\boldsymbol{i}}^{(j)} = (g_{i0}^{(j)}, g_{i1}^{(j)}, g_{i2}^{(j)}, \ldots) \tag{5}$$

$$G_{\boldsymbol{i}}^{(j)}(D) = g_{i0}^{(j)} + g_{i1}^{(j)} D + g_{i2}^{(j)} D^2 + \cdots \tag{6}$$

Polynomial expressions of the output sequences can be then obtained as a function of the above expressions. Thus, and for the example of the FSSM in Figure 6.3,

$$\begin{aligned} C^{(1)}(D) &= M(D)G^{(1)}(D) = M(D)(1 + D^2) = c_0^{(1)} + c_1^{(1)} D + c_2^{(1)} D^2 + \cdots \\ C^{(2)}(D) &= M(D)G^{(2)}(D) = M(D)(1 + D + D^2) = c_0^{(2)} + c_1^{(2)} D + c_2^{(2)} D^2 + \cdots \end{aligned} \tag{7}$$

By multiplexing output polynomials $C^{(1)}(D)$ and $C^{(2)}(D)$, the code sequence in polynomial form is finally obtained as

$$\boldsymbol{C}_{\boldsymbol{m}}(D) = C^{(1)}(D^2) + DC^{(2)}(D^2) \tag{8}$$

Polynomial expressions of the impulse responses also indicate that the presence of a connection is described in polynomial form by the existence of the corresponding power of D (this term is multiplied by a '1'), while the absence of connection is seen as the absence of such a term (this term is multiplied by a '0'). Polynomial expressions of the impulse responses can also be considered as generator polynomials for each output sequence of the FSSM.

For more general structures of FSSMs or convolutional encoders where there is more than one input and more than one output, the relationship between input i and output j is given by the corresponding transfer function $G_i^{(j)}(D)$. In this input-to-output path the number of delays or memory units D is called the length of the register. This number is equal to the degree of the corresponding generator polynomial for such a path. To make sense, the last delay or register stage should be connected to at least one output, so that the length K_i for the ith register is defined as [1]

$$K_i = \max_{1 \le j \le n} \left\{ \deg\left(g_i^{(j)}(D) \right) \right\} \quad 1 \le i \le k \tag{9}$$

The memory order of the encoder K is obtained as a function of the above definition as

$$K = \max_{1 \le i \le k} K_i = \max_{\substack{1 \le j \le n \\ 1 \le i \le k}} \left\{ \deg\left(g_i^{(j)}(D) \right) \right\} \tag{10}$$

If $M^{(i)}(D)$ is the polynomial expression of the input sequence at input i and $C^{(j)}(D)$ is the polynomial expression of the output j generated by this input, then the polynomial $G_i^{(j)}(D) = C^{(j)}(D)/M^{(i)}(D)$ is the transfer function that relates input i and output j. In a more general FSSM structure for which there are k inputs and n outputs, there will be kn transfer functions that can be arranged in matrix form as

$$\boldsymbol{G}(D) = \begin{bmatrix} G_1^{(1)}(D) & G_1^{(2)}(D) & \cdots & G_1^{(n)}(D) \\ G_2^{(1)}(D) & G_2^{(2)}(D) & \cdots & G_2^{(n)}(D) \\ \vdots & \vdots & & \vdots \\ G_k^{(1)}(D) & G_k^{(2)}(D) & \cdots & G_k^{(n)}(D) \end{bmatrix} \tag{11}$$

A convolutional code $C_{\text{conv}}(n, k, K)$ produces an output sequence expressed in polynomial form as

$$\boldsymbol{C}(D) = \boldsymbol{M}(D)\boldsymbol{G}(D) \tag{12}$$

where

$$\boldsymbol{M}(D) = \left(M^{(1)}(D), M^{(2)}(D), \ldots, M^{(k)}(D) \right) \tag{13}$$

and

$$\boldsymbol{C}(D) = \left(C^{(1)}(D), C^{(2)}(D), \ldots, C^{(n)}(D) \right) \tag{14}$$

so that after multiplexing,

$$\boldsymbol{C_m}(D) = C^{(1)}(D^n) + DC^{(2)}(D^n) + \cdots + D^{n-1}C^{(\boldsymbol{n})}(D^n) \tag{15}$$

Example 6.1: For the convolutional code $C_{\text{conv}}(2, 1, 2)$ whose encoder is seen in Figure 6.3, determine the polynomial expression of the output for the input sequence (100011).

The input sequence in polynomial form is

$$M(D) = 1 + D^4 + D^5$$

The generator matrix is of the form

$$\boldsymbol{G}(D) = \left[\, 1 + D^2 \;\; 1 + D + D^2 \,\right]$$

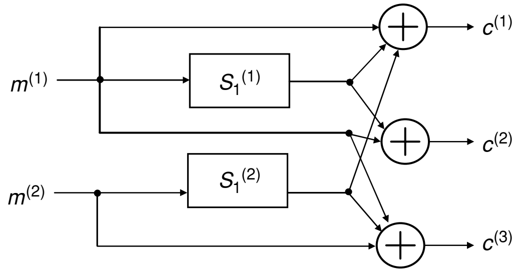


Figure 6.4 Encoder of convolutional code C_{conv} (3, 2, 1) of code rate $R_{\text{c}} = 2/3$

Then

$$\begin{aligned} \boldsymbol{C}(D) &= \left(C^{(1)}(D)\ \ C^{(2)}(D)\right) \\ &= \left[1 + D^4 + D^5\right]\left[\,1 + D^2\ \ 1 + D + D^2\,\right] \\ &= \left[1 + D^2 + D^4 + D^5 + D^6 + D^7\ \ 1 + D + D^2 + D^4 + D^7\right] \end{aligned}$$

$$\boldsymbol{c}^{(1)} = (10101111)$$

$$\boldsymbol{c}^{(2)} = (11101001)$$

Finally, the output sequence is

$$\boldsymbol{c} = (11, 01, 11, 00, 11, 10, 10, 11)\,.$$

Note that each output sequence of the encoder has K bits more than the corresponding input sequence, making the code sequence $2K$ bits longer. The reason is that the output sequences are determined by the generator sequences of the code, which in turn represent the impulse responses of the encoder. It is as if K zeros were added to the input sequence to finally determine the output sequence.

For the encoder of the convolutional code $C_{\text{conv}}(3, 2, 1)$ seen in Figure 6.4, the input is now a vector of 2 bits that generates an output vector of 3 bits at each instant i. Note that the input bits are simultaneously applied, and output bits are consequently simultaneously generated at each time instant. This code is of rate $R_{\text{c}} = 2/3$. The memory of this FSSM is defined by one delay or register stage in each branch of the structure.

The input vector is

$$\boldsymbol{m} = \left(m_0^{(1)}m_0^{(2)}, m_1^{(1)}m_1^{(2)}, m_2^{(1)}m_2^{(2)}, \ldots\right) \tag{16}$$

and it is constructed using the input sequences

$$\boldsymbol{m}^{(1)} = \left(m_0^{(1)}, m_1^{(1)}, m_2^{(1)}, \ldots\right)$$

$$\boldsymbol{m}^{(2)} = \left(m_0^{(2)}, m_1^{(2)}, m_2^{(2)}, \ldots\right) \tag{17}$$

Impulse responses are described as

$$\boldsymbol{g}_i^{(j)} = \left(g_{i,0}^{(j)}, g_{i,1}^{(j)}, \ldots, g_{i,K}^{(j)}\right) \tag{18}$$

Table 6.2 Response to the unit impulse input $m^{(1)}$

i	$m^{(1)}$	$s_1^{(1)}$	$s_1^{(2)}$	$c^{(1)}$	$c^{(2)}$	$c^{(3)}$
0	1	0	0	1	1	1
1	0	1	0	1	1	0
2	0	0	0	0	0	0

which relate input i to output j. The impulse responses for the first input are given in Table 6.2.

In this case the other input is set to zero $m_i^{(2)} = 0, \forall i$. Impulse responses for the second input are given in Table 6.3.

In this case the other input is set to zero, $m_i^{(1)} = 0, \forall i$.

Then

$$g_1^{(1)} = (1 \quad 1) \qquad g_1^{(2)} = (1 \quad 1) \qquad g_1^{(3)} = (1 \quad 0)$$
$$g_2^{(1)} = (0 \quad 1) \qquad g_2^{(2)} = (0 \quad 0) \qquad g_2^{(3)} = (1 \quad 1)$$

and the encoding equations can be expressed as

$$c_l^{(1)} = m_l^{(1)} + m_{l-1}^{(1)} + m_{l-1}^{(2)}$$
$$c_l^{(2)} = m_l^{(1)} + m_{l-1}^{(1)}$$
$$c_l^{(3)} = m_l^{(1)} + m_l^{(2)} + m_{l-1}^{(2)}$$

Thus, the code sequence is of the form

$$c = \left(c_0^{(1)}c_0^{(2)}c_0^{(3)}, c_1^{(1)}c_1^{(2)}c_1^{(3)}, c_2^{(1)}c_2^{(2)}c_2^{(3)}, \ldots\right)$$

Expressions for the generator polynomials in the D domain are

$$G_1^{(1)}(D) = 1 + D \qquad G_1^{(2)}(D) = 1 + D \qquad G_1^{(3)}(D) = 1$$
$$G_2^{(1)}(D) = D \qquad G_2^{(2)}(D) = 0 \qquad G_2^{(3)}(D) = 1 + D$$

Thus, and if the input vector is for instance equal to

$$\boldsymbol{m}^{(1)} = (101) \qquad \boldsymbol{m}^{(2)} = (011)$$

Table 6.3 Response to the unit impulse input $m^{(2)}$

i	$m^{(2)}$	$s_1^{(1)}$	$s_1^{(2)}$	$c^{(1)}$	$c^{(2)}$	$c^{(3)}$
0	1	0	0	0	0	1
1	0	1	0	1	0	1
2	0	0	0	0	0	0

then the corresponding polynomial expression is

$$M^{(1)}(D) = 1 + D^2 \qquad M^{(2)}(D) = D + D^2$$

$$\begin{aligned}
C^{(1)}(D) &= M^{(1)}(D)G_1^{(1)}(D) + M^{(2)}(D)G_2^{(1)}(D)\\
&= (1 + D^2)(1 + D) + (D + D^2)D\\
&= 1 + D
\end{aligned}$$

$$\begin{aligned}
C^{(2)}(D) &= M^{(1)}(D)G_1^{(2)}(D) + M^{(2)}(D)G_2^{(2)}(D)\\
&= (1 + D^2)(1 + D) + (D + D^2)0\\
&= 1 + D + D^2 + D^3
\end{aligned}$$

$$\begin{aligned}
C^{(3)}(D) &= M^{(1)}(D)G_1^{(3)}(D) + M^{(2)}(D)G_2^{(3)}(D)\\
&= (1 + D^2)(1) + (D + D^2)(1 + D)\\
&= 1 + D + D^2 + D^3
\end{aligned}$$

and the code sequence is

$$c = (111, 111, 011, 011)$$

The general structure of the encoder can be designed to have different memory levels K_i in each of its branches. In this case, as noted previously, the memory of the encoder is defined as the maximum register length. If K_i is the register length of the ith register, then the memory order is defined as [1]

$$K = \max_{1 \le i \le k} (K_i) \tag{19}$$

For a given convolutional code $C_{\text{conv}}(n, k, K)$, the input vector is a sequence of kL information bits and the code sequence contains $N = nL + nK = n(L + K)$ bits. The nK additional bits are related to the memory of the FSSM or encoder. The amount

$$n_{\text{A}} = n(K + 1) \tag{20}$$

is the maximum number of output bits that a given input bit can influence. Then n_{A} is the constraint length of the code, measured in bits.

In general terms, an input of k bits generates an output of n bits, and it is said that the rate of a convolutional code $C_{\text{conv}}(n, k, K)$ is k/n. However, and for a given finite input sequence of length L, the corresponding output sequence will contain $n(L + K)$ bits, as after the input of the L vectors of k bits each, a sequence of '0's is input to empty all the registers of the FSSM. From this point of view, the operation of the convolutional code is similar to that of a block code, and the code rate would be

$$\frac{kL}{n(L + K)} \tag{21}$$

This number tends to k/n for a sufficiently large input sequence of length $L \gg K$.

6.4 Convolutional Encoder Representations

6.4.1 Representation of Connections

As seen for instance in Figure 6.3, which is the encoder of a convolutional code $C_{\mathrm{conv}}(2, 1, 2)$, in each clock cycle the bits contained in each register stage are right shifted to the following stage, and, on the other hand, the two outputs are sampled to generate the two output bits for each input bit. Output values depend on the way the registers are connected to the outputs. A different output sequence would be obtained if these connections were made in a different manner.

One form of describing a convolutional code $C_{\mathrm{conv}}(n, k, K)$ is by means of a vector description of the connections that the FSSM has, which are directly described by the vector representation of the generator polynomials $\boldsymbol{g}^{(1)}$ and $\boldsymbol{g}^{(2)}$ that correspond to the upper and lower branches of the FSSM of Figure 6.3, respectively. In this description, a '1' means that there is connection, and a '0' means that the corresponding register is not connected:

$$\boldsymbol{g}^{(1)} = (1\ 0\ 1)$$

$$\boldsymbol{g}^{(2)} = (1\ 1\ 1)$$

For a given input sequence, this code description can provide the corresponding output sequence. This can be seen by implementing a table. For example, Table 6.4 describes the register contents, the present state, the future state and the output values $\boldsymbol{c}^{(1)}$ and $\boldsymbol{c}^{(2)}$ when the input sequence is $\boldsymbol{m} = (100011)$, for the FSSM of Figure 6.3.

The resulting output sequence is $\boldsymbol{c} = (11011100111010 11)$. Table 6.5 is a useful tool for constructing the corresponding state diagram of the encoder of Figure 6.3.

Note that in Table 6.5 there is no relationship between a row and the next or previous rows, and so in this sense it is different from Table 6.4, where this relationship indeed exists.

6.4.2 State Diagram Representation

The state of the FSSM that forms the encoder of a $1/n$-rate convolutional code is defined as the contents of its K register stages. The future state is obtained by shifting one delay D to the right the contents of the present state so that the empty stage generated in the left-most position

Table 6.4 Output sequence for a given input sequence to the encoder of Figure 6.3

Input m_i	State at t_i	State at t_{i+1}	$c^{(1)}$	$c^{(2)}$
–	0 0	0 0	–	–
1	0 0	1 0	1	1
0	1 0	0 1	0	1
0	0 1	0 0	1	1
0	0 0	0 0	0	0
1	0 0	1 0	1	1
1	1 0	1 1	1	0
0	1 1	0 1	1	0
0	0 1	0 0	1	1
0	0 0	0 0	0	0

Table 6.5 Table of all the possible transitions for constructing a state diagram of a convolutional encoder

Input m_i	State at t_i	State at t_{i+1}	$c^{(1)}$	$c^{(2)}$
–	0 0	0 0	–	–
0	0 0	0 0	0	0
1	0 0	1 0	1	1
0	0 1	0 0	1	1
1	0 1	1 0	0	0
0	1 0	0 1	0	1
1	1 0	1 1	1	0
0	1 1	0 1	1	0
1	1 1	1 1	0	1

is filled with the value of the input bit at that time instant. The state diagram is a pictorial representation of the evolution of the state sequences for these codes. The FSSM encoder of Figure 6.3 has, for instance, the state diagram shown in Figure 6.5 [2, 4].

In this particular case there are four states, labelled $S_\mathrm{a} = 00$, $S_\mathrm{b} = 10$, $S_\mathrm{c} = 01$ and $S_\mathrm{d} = 11$. There are only two transitions emerging from and arriving at any of these states, because there are only two possible input values; that is, the input is either ‘1’ or ‘’0’. Transitions in Figure 6.5 by convention are shown in input–output form.

The state diagram of a convolutional encoder shows an interesting characteristic of these codes. As described above, there are only two transitions emerging from a given state, but there are, in this case, four states. Therefore there is some level of memory related to this fact, since if the FSSM is in a given state, it is not possible to go to any other state in an arbitrary manner, but only to two specific states as shown in the diagram. This sort of memory will be useful in determining that some transitions are not allowed in the decoded sequence, thus assisting the decisions required for error correction.

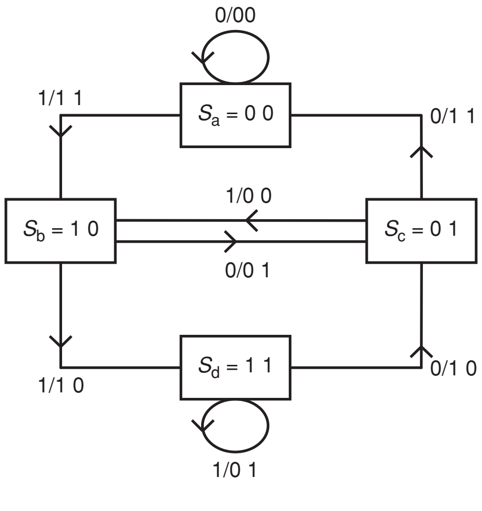


Figure 6.5 State diagram for the convolutional encoder of Figure 6.3

6.4.3 Trellis Representation

A way of representing systems based on FSSMs is the tree diagram [1, 2, 4]. This representation is useful to indicate the start of the state sequence, but however the repetitive structure of state evolution is not clearly presented in this case. One of the interesting characteristics of the state evolution of a convolutional code is precisely that after $K + 1$ initial transitions, the state structure becomes repetitive, where $K + 1$ is the constraint length of the code.

On the one hand, the state diagram clearly shows the repetitive structure of the state evolution of a convolutional code, but it is not clear for describing the initial evolution. On the other hand, the tree clearly shows the initial sequence, but not the repetitive structure of the state evolution. A representation that clearly describes these two different parts of the state structure of a given convolutional code is the so-called trellis diagram. Figure 6.6 depicts the trellis diagram of the convolutional code $C_{\mathrm{conv}}(2, 1, 2)$ being used as an example.

The same convention used in the state diagram for denoting transitions is also adopted in this trellis representation. This trellis diagram is a state versus time instant representation. There are 2^K possible states in this diagram. As seen in Figure 6.6, the state structure becomes repetitive after time instant t_4. There are two branches emerging from and arriving at a given state, which correspond to transitions produced by the two possible inputs to the FSSM.

6.5 Convolutional Codes in Systematic Form

In a systematic code, message information can be seen and directly extracted from the encoded information. In the case of a convolutional code, this means that

$$c^{(i)} = m^{(i)}, \quad i = 1, 2, \ldots, k \tag{22}$$

$$g_i^{(j)} = \begin{cases} 1 & j = i \\ 0 & j \neq i \end{cases} \tag{23}$$

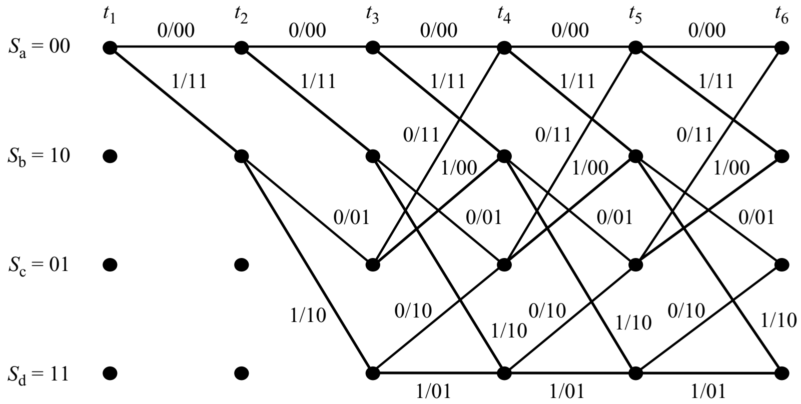


Figure 6.6 Trellis representation of the convolutional code of Figure 6.3

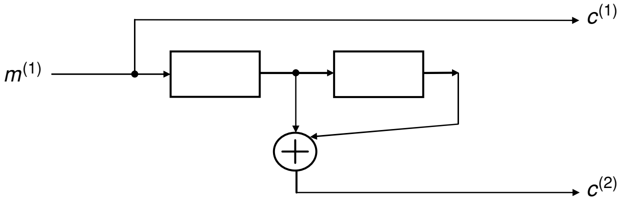


Figure 6.7 A systematic convolutional encoder

The transfer function for a systematic convolutional code is of the form

$$\boldsymbol{G}(D) = \begin{bmatrix} 1 & 0 & \cdots & 0 & G_1^{(k+1)}(D) & G_1^{(k+2)}(D) & \cdots & G_1^{(n)}(D) \\ 0 & 1 & \cdots & 0 & G_2^{(k+1)}(D) & G_2^{(k+2)}(D) & \cdots & G_2^{(n)}(D) \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 & G_k^{(k+1)}(D) & G_k^{(k+2)}(D) & \cdots & G_k^{(n)}(D) \end{bmatrix} \tag{24}$$

Example 6.2: Determine the transfer function of the systematic convolutional code as shown in Figure 6.7, and then obtain the code sequence for the input sequence $\boldsymbol{m} = (1101)$.

The transfer function in this case is

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

and so the code sequence for the given input sequence, which in polynomial form is $m(D) = 1 + D + D^3$, is obtained from

$$C^{(1)}(D) = M(D)G^{(1)}(D) = 1 + D + D^3$$
$$C^{(2)}(D) = M(D)G^{(2)}(D) = \left(1 + D + D^3\right)\left(D + D^2\right) = D + D^3 + D^4 + D^5$$

Then

$$c = (10, 11, 00, 11, 01, 01)$$

In the case of systematic convolutional codes, there is no need to have an inverse transfer function decoder to obtain the input sequence, because this is directly read from the code sequence. However, for non-systematic convolutional codes, there needs to be an $n \times k$ matrix $\boldsymbol{G}^{-1}(D)$, such that

$$\boldsymbol{G}(D) \circ \boldsymbol{G}^{-1}(D) = \boldsymbol{I}_k D^l \quad \text{for some } l \geq 0 \tag{25}$$

where $\boldsymbol{I}_k$ is the identity matrix of size $k \times k$. For a given convolutional code $C_{\text{conv}}(n, 1, K)$, it can be shown that its matrix $\boldsymbol{G}(D)$ has an inverse $\boldsymbol{G}^{-1}(D)$ if and only if [16]

$$\text{HCF}\left\{G^{(1)}(D), G^{(2)}(D), \ldots, G^{(n)}(D)\right\} = D^l, \quad l \geq 0 \tag{26}$$

A convolutional code characterized by its transfer function matrix $\boldsymbol{G}(D)$, for which an inverse matrix $\boldsymbol{G}^{-1}(D)$ exists, also has the property of being non-catastrophic [16].

Example 6.3: Verify that the following convolutional code $C_{\text{conv}}(2, 1, 2)$ is catastrophic:

$$g^{(1)}(D) = 1 + D$$
$$g^{(2)}(D) = 1 + D^2$$

Since

$$\text{HCF}\left\{G^{(1)}(D), G^{(2)}(D)\right\} = 1 + D \neq D^l, \quad l \geq 0$$

and applying the infinite input sequence

$$1 + D + D^2 + \cdots = \frac{1}{1 + D}$$

the outputs for this encoder will be

$$c^{(1)}(D) = 1$$
$$c^{(2)}(D) = 1 + D$$

which give the finite output sequence $\boldsymbol{c} = (11, 01)$, followed by an infinite sequence of ‘0’s.

Let us assume that the above infinite input sequence is transmitted, and that the encoder generates the corresponding finite sequence $\boldsymbol{c} = (11, 01)$ followed by ‘0’s. Let us also assume that, in the channel, the transmitted sequence is affected by errors in such a way that it is converted into the sequence $\boldsymbol{c} = (00, 00)$ followed by ‘0’s. The decoder will receive the all-zero sequence, and in a linear code, this corresponds to the all-zero input sequence. Therefore, the decoder will decode the infinite input sequence $1 + D + D^2 + \cdots = 1/(1 + D)$ as the all-zero input, thus producing an infinite number of errors, a catastrophic result.

Another characteristic of catastrophic convolutional codes is that their state diagrams show loops of zero weight at states that are different from the self-loop at state S_{a}. A very interesting characteristic of systematic linear convolutional codes is that they are inherently non-catastrophic [16].

6.6 General Structure of Finite Impulse Response and Infinite Impulse Response FSSMs

6.6.1 Finite Impulse Response FSSMs

Convolutional encoders are usually constructed using FSSMs. Figure 6.8 shows the structure of a finite impulse response (FIR) FSSM that can be used as part of a convolutional encoder.

The coefficients of these structures are taken from the Galois field over which they are defined, $a_i \in \text{GF}(q)$. In the particular case of the binary field GF(2), they can be equal to one

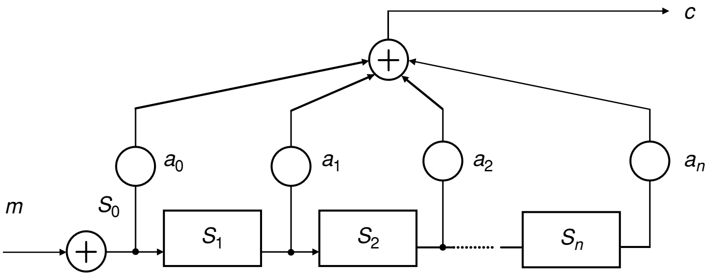


Figure 6.8 An FIR FSSM

or zero. The transfer function for the FIR FSSM, as shown in Figure 6.8, is

$$G(D) = \frac{C(D)}{M(D)} = a_0 + a_1 D + a_2 D^2 + \cdots + a_n D^n \tag{27}$$

A transfer function for a particular case is obtained in the following section, and this procedure for obtaining transfer functions can be easily generalized to other cases.

6.6.2 *Infinite Impulse Response FSSM*

An infinite impulse response (IIR) structure contains feedback coefficients that connect the outputs of the registers to an adder, placed at the input. A general structure for IIR FSSMs is shown in Figure 6.9.

The transfer function for this structure is shown to be

$$G(D) = \frac{C(D)}{M(D)} = \frac{a_0 + a_1 D + a_2 D^2 + \cdots + a_n D^n}{1 + f_1 D + f_2 D^2 + \cdots + f_n D^n} \tag{28}$$

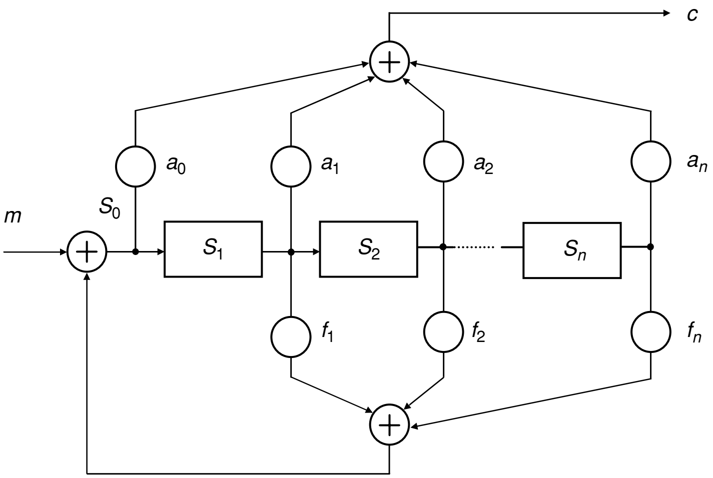


Figure 6.9 An IIR FSSM

In general, convolutional encoders can be constructed by using either FIR or IIR FSSMs, and can generate systematic or non-systematic convolutional codes. There is a relationship between the systematic and the non-systematic form of a given convolutional encoder.

6.7 State Transfer Function Matrix: Calculation of the Transfer Function

6.7.1 State Transfer Function for FIR FSSMs

The state transfer function for a convolutional encoder or FSSM can be defined in the same way as for the input–output transfer function, a definition that is more conveniently done in the D domain. In order to introduce the state transfer function, the following FSSM, which is part of the encoder shown in Figure 6.3, is now presented in Figure 6.10, with a slightly different notation, which in this case shows the variables involved in the discrete time domain.

In this FSSM,

$$\begin{aligned} m(k) &= s_0(k) \\ c^{(1)}(k) &= s_0(k) + s_2(k) = s_0(k) + s_0(k-2) \end{aligned}$$

In the D domain,

$$\begin{aligned} C^{(1)}(D) &= S_0(D) + D^2 S_0(D) = (1 + D^2) S_0(D) = (1 + D^2) M(D) \\ S_0(D) &= M(D) \end{aligned}$$

where

$$\begin{aligned} s_1(k) &= s_0(k-1) \\ s_2(k) &= s_0(k-2) \\ S_1(D) &= D S_0(D) = D M(D) \\ S_2(D) &= D^2 S_0(D) = D^2 M(D) \end{aligned}$$

The transfer function is

$$G(D) = \frac{C^{(1)}(D)}{M(D)} = 1 + D^2$$

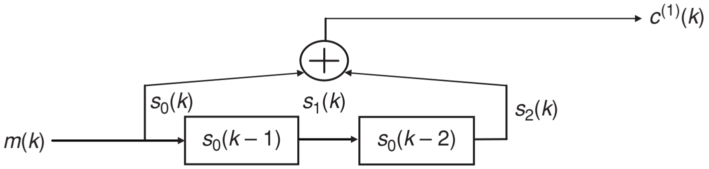


Figure 6.10 FIR FSSM in the discrete time domain

and the state transfer function is

$$S(D) = \left[S_0(D)/M(D)\ S_1(D)/M(D)\ S_2(D)/M(D) \right] = \left[1\ D\ D^2 \right]$$

The state transfer function can be used to determine the evolution of the states of the corresponding FSSM with respect to a particular input sequence. In the case of the unit impulse input sequence, $M(D) = 1$, this state transfer function describes the state sequence as sequences in the D domain. When the FSSM is an FIR FSSM, the impulse response additionally describes the shortest state sequence.

In this particular example, the state of the FSSM is defined by the pair $(S_1(D)\ \ S_2(D))$. Now, and for this example, the state impulse response is

$$\left[S_0(D)\ \ S_1(D)\ \ S_2(D) \right] = \left[1\ D\ D^2 \right] \bullet M(D)$$
$$\left[S_0(D)\ \ S_1(D)\ \ S_2(D) \right] = \left[1\ D\ D^2 \right] \bullet 1 = \left[1\ D\ D^2 \right]$$

$$S_0 = (1, 0, 0, 0, 0, 0, \ldots)$$
$$S_1 = (0, 1, 0, 0, 0, 0, \ldots)$$
$$S_2 = (0, 0, 1, 0, 0, 0, \ldots)$$

Therefore the state transitions vector for the impulse response is $(S_1\ \ S_2) = (00, 10, 01, 00, 00, \ldots)$, which describes the shortest state transition of the FSSM, and is also the shortest sequence as seen in the corresponding trellis in Figure 6.6, described by the state sequence $S_\mathrm{a} S_\mathrm{b} S_\mathrm{c} S_\mathrm{a}$. The output sequence of the FSSM is

$$C^{(1)}(D) = (1 + D^2)m(D) = 1 + D^2$$

This output is of weight 2.

The state transfer function is useful for determining analytically the state sequence of an FSSM or convolutional encoder. For FIR FSSMs, the state S_0 completely describes the state evolution of the FSSM, and since $S_0(D) = M(D)$, the input sequence completely determines the state sequence.

6.7.2 State Transfer Function for IIR FSSMs

Let us consider the FSSM seen in Figure 6.11, which, as will be seen in the following section, is part of the systematic convolutional encoder that is equivalent to the encoder shown in Figure 6.3. In this figure variables are described in the discrete time domain.

A similar analysis to that presented for FIR FSSMs is the following:

$$s_0(k) = m(k) + s_2(k)$$

where

$$s_1(k) = s_0(k-1)$$
$$s_2(k) = s_0(k-2)$$
$$c(k) = s_0(k) + s_0(k-1) + s_0(k-2)$$

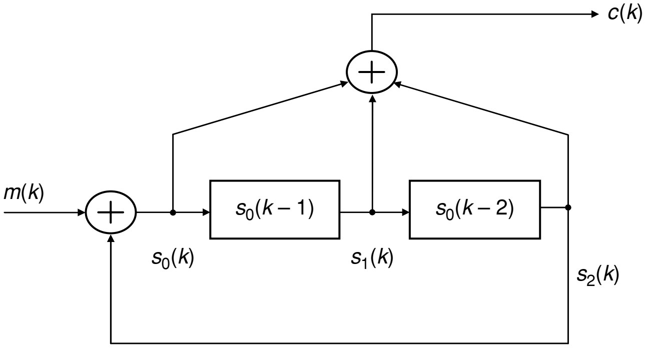


Figure 6.11 An IIR FSSM

In the D domain,

$$S_1(D) = DS_0(D)$$

$$S_2(D) = D^2 S_0(D)$$

$$S_0(D) = M(D) + S_2(D) = M(D) + D^2 S_0(D)$$

$$S_0(D) + D^2 S_0(D) = M(D)$$

$$S_0(D) = \frac{M(D)}{1 + D^2}$$

$$C(D) = S_0(D) + DS_0(D) + D^2 S_0(D) = (1 + D + D^2)S_0(D) = (1 + D + D^2)\frac{M(D)}{1 + D^2}$$

The transfer function is

$$G(D) = \frac{C(D)}{M(D)} = \frac{1 + D + D^2}{1 + D^2}$$

and the state transfer function is [15]

$$S(D) = \left[\frac{S_0(D)}{M(D)} \; \frac{S_1(D)}{M(D)} \; \frac{S_2(D)}{M(D)} \right] = \left[\frac{1}{1 + D^2} \; \frac{D}{1 + D^2} \; \frac{D^2}{1 + D^2} \right]$$

The impulse response is now infinite, and it does not correspond to the shortest sequence of the FSSM. The state transfer function can be used to identify which is the input for generating the shortest sequence. In this particular case, if the input is $M(D) = 1 + D^2$, the corresponding state sequence is

$$\begin{aligned}
[S_0(D) \quad S_1(D) \quad S_2(D)] &= \left[\frac{1}{1 + D^2} \quad \frac{D}{1 + D^2} \quad \frac{D^2}{1 + D^2} \right] \bullet M(D) \\
&= \left[\frac{1}{1 + D^2} \quad \frac{D}{1 + D^2} \quad \frac{D^2}{1 + D^2} \right] \bullet (1 + D^2) \\
&= \left[1 \quad D \quad D^2 \right]
\end{aligned}$$

which is the same as the shortest sequence of the IIR FSSM, as shown in the previous example. For a one input–one output FSSM with K registers, the state transfer function is defined as

$$\boldsymbol{S}(D) = \begin{bmatrix} \frac{S_0(D)}{M(D)} & \frac{S_1(D)}{M(D)} & \cdots & \frac{S_K(D)}{M(D)} \end{bmatrix} \tag{29}$$

6.8 Relationship Between the Systematic and the Non-Systematic Forms

The transfer function description of an FSSM encoder can be used to obtain the equivalent systematic form of a given non-systematic encoder [7]. This conversion method consists of converting the transfer function of a non-systematic form, as given in expression (11), into an expression of systematic form, like that of expression (24), by means of matrix operations.

Example 6.4: Determine the equivalent systematic version of the convolutional encoder generated by the transfer function

$$\boldsymbol{G}(D) = \boldsymbol{G}_{\mathrm{ns}}(D) = \begin{bmatrix} 1 + D^2 & 1 + D + D^2 \end{bmatrix}$$

The non-systematic convolutional encoder of the code described by this transfer function is shown in Figure 6.3. The transfer function should adopt the form of equation (24) to correspond to a systematic convolutional encoder. In this case the procedure is quite simple, because it only consists of dividing both polynomials of the transfer function by the polynomial $1 + D^2$. This procedure converts the transfer functions of the original convolutional code, which are in this case of FIR type, into transfer functions of IIR type. This division is done in order to make the matrix transfer function contain the identity submatrix, which in this example is $G_{11}(D) = 1$. The resulting transfer function is

$$\boldsymbol{G}_s(D) = \begin{bmatrix} 1 & \frac{1 + D + D^2}{1 + D^2} \end{bmatrix}$$

According to this expression, a non-systematic convolutional code encoded with FIR transfer functions has an equivalent systematic convolutional code encoded with IIR transfer functions, like the corresponding FSSM as shown in Figure 6.11. Indeed the transfer function $\frac{1+D+D^2}{1+D^2}$ is of the form of equation (28) with $a_0 = a_1 = a_2 = f_2 = 1$, $f_1 = 0, a_i = f_i = 0$ for $i > 2$, and is implemented by an FSSM like that seen in Figure 6.11. Then equivalent systematic convolutional encoder for the convolutional code of Example 6.4 is as seen in Figure 6.12.

Table 6.6 is a suitable tool for analysis of the convolutional encoder of Figure 6.12.

The corresponding trellis, obtained from Table 6.6, is given in Figure 6.13.

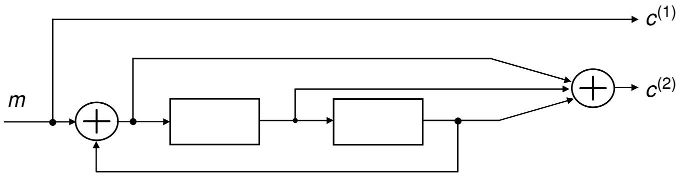


Figure 6.12 Equivalent systematic convolutional encoder of the encoder of Figure 6.3

Table 6.6 Table for constructing the state diagram of the convolutional encoder of Figure 6.12

Input m_i	State t_i	State at t_{i+1}	$c^{(1)}$	$c^{(2)}$
–	0 0	0 0	–	–
0	0 0	0 0	0	0
1	0 0	1 0	1	1
0	0 1	1 0	0	0
1	0 1	0 0	1	1
0	1 0	0 1	0	1
1	1 0	1 1	1	0
0	1 1	1 1	0	1
1	1 1	0 1	1	0

It can be verified that transitions in the trellis of Figure 6.6, which corresponds to the convolutional encoder of Figure 6.3, have the same output assignments as the trellis of Figure 6.13, which corresponds to the equivalent convolutional encoder in systematic form.

The difference between the systematic and the non-systematic forms of the same convolutional code is in the way the input is assigned a given output. As in the case of block codes, the systematic convolutional encoder (or systematic generator matrix) generates the same code as its corresponding non-systematic encoder (or the corresponding non-systematic generator matrix), but with different input–output assignments.

For the convolutional encoder in systematic form, as seen in Figure 6.12, the transfer function matrix and the state transfer function matrix are

$$\boldsymbol{G}(D) = \begin{bmatrix} 1 & \dfrac{1 + D + D^2}{1 + D^2} \end{bmatrix}$$

and

$$\boldsymbol{S}(D) = \begin{bmatrix} \dfrac{1}{1 + D^2} & \dfrac{D}{1 + D^2} & \dfrac{D^2}{1 + D^2} \end{bmatrix}$$

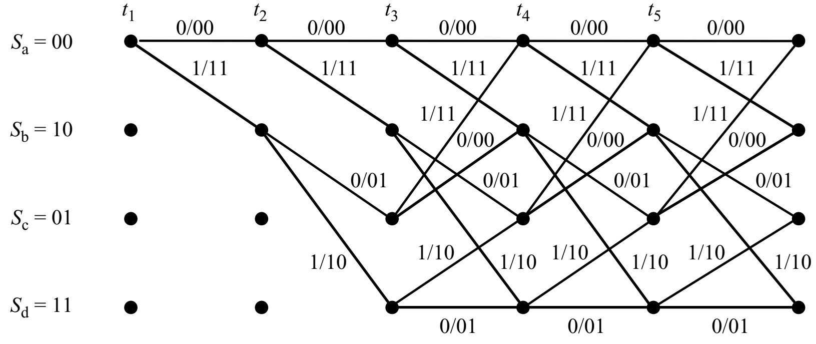


Figure 6.13 Trellis for the convolutional encoder of Figure 6.12

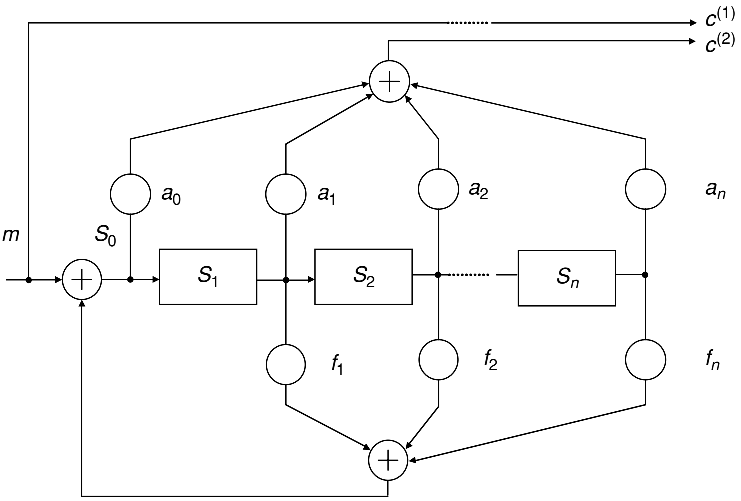


Figure 6.14 General structure of systematic IIR convolutional encoders of rate $R_c = 1/2$

The shortest state sequence for the systematic convolutional encoder, characterized by being implemented using IIR transfer functions, does not correspond to the unit impulse input, which in this case generates an infinite output or state sequence. The input that produces the shortest state sequence can be obtained by inspection of the corresponding state transfer function, so that if this input sequence in polynomial form is equal to $M(D) = 1 + D^2$, the system generates the state sequence $(S_1 \ S_2) = (00, 10, 01, 00, 00, \ldots)$, that is, the shortest sequence in the trellis as seen in Figure 6.13. The corresponding output sequence, for this case, is

$$C^{(1)}(D) = 1M(D) = 1 + D^2$$

$$C^{(2)}(D) = \frac{1 + D + D^2}{1 + D^2} M(D) = 1 + D + D^2$$

which is an output of weight 5.

In the general case, IIR convolutional encoders of rate $R_c = 1/2$ are of the form as given in Figure 6.14.

The coefficients in this structure belong to the field over which the IIR FSSM is defined, $a_i \in \mathrm{GF}(q)$ and $f_j \in \mathrm{GF}(q)$. In the particular case of encoders operating over GF(2), these coefficients will be 1 or 0. The transfer function matrix of the general structure of Figure 6.14 for an IIR systematic convolutional encoder of rate $R_c = 1/2$ is shown to be [15]

$$\boldsymbol{G}(D) = \left[1 \quad \frac{a_0 + a_1 D + a_2 D^2 + \cdots + a_n D^n}{1 + f_1 D + f_2 D^2 + \cdots + f_n D^n} \right] \tag{30}$$

6.9 Distance Properties of Convolutional Codes

One of the most significant parameters of an error-correcting or error-detecting code is the minimum distance of the code, normally evaluated as the minimum value of the distance that exists between any two code vectors of the code. When the code is linear, it is sufficient to determine the distance between any code vector and the all-zero vector, which is, in the end, the weight of that code vector [1, 2, 4, 5].

As seen for block codes, the minimum distance can be interpreted as the minimum-weight error pattern that converts a given code vector into another code vector in the same code. In the case of convolutional codes, this becomes the number of errors that convert a given code sequence into another valid code sequence.

Since almost all convolutional codes of practical use are linear, the minimum distance of the code can be determined by finding the code sequence of minimum weight. From this point of view, the above analysis implies a search for the minimum number of errors that convert the all-zero sequence into another code sequence. This can be seen in the corresponding trellis of the convolutional code as a sequence that emerges from the all-zero state, normally labelled S_a, and arrives back at the same state after a certain number of transitions. Then the Hamming or minimum distance of the code can be determined by calculating the minimum weight among all the sequences that emerge from and arrive back at the all-zero state S_a after a finite number of transitions.

A tool for analysing the distance properties of a convolutional code is obtained by modifying the traditional state diagram, in such a way that the modified diagram starts and ends in the all-zero state S_a. In this modified diagram [1, 4], the self-loop that represents the transition from state S_a to itself is omitted. In this modified state diagram branches emerging and arriving at the states are denoted by the term X^i, where i is the weight of the code sequence that corresponds to that branch.

For the example of the convolutional code $C_{\text{conv}}(2, 1, 2)$,introduced in previous sections, the modified state diagram is shown in Figure 6.15.

Paths starting and arriving at the all-zero state S_a have a weight that can be calculated by adding the exponents i of the corresponding terms of the form X^i. In this case, for instance, the path $S_aS_bS_cS_a$ has a total weight 5, and path $S_aS_bS_dS_cS_a$ is of weight 6. The remaining paths include the loops S_dS_d and $S_bS_cS_b$ that only add weight to the paths described above. Therefore the minimum distance in this code is equal to 5, and it is usually called the minimum free distance, $d_f = 5$. The adjective *free* comes from the fact that there are no restrictions on the length of the paths of the corresponding trellis or state diagram involved in its calculation.

For convolutional codes of more complex structure, the above calculation of the modified state diagram, and its solution to determine the minimum free distance, is not that simple.

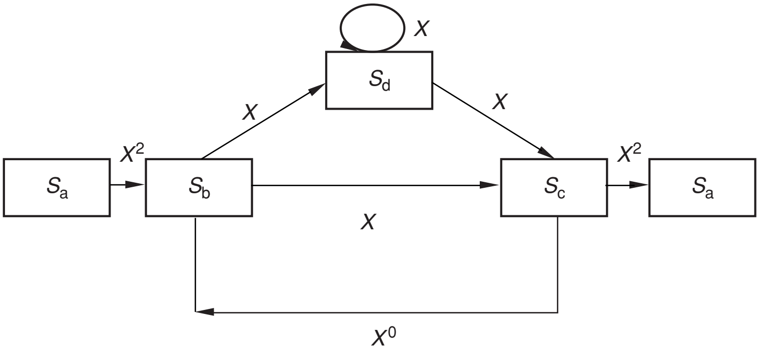


Figure 6.15 Modified state diagram

More complex modified diagrams are solved by means of the Mason rule over what is called the generating function $T(X)$ (see details in [1]). This generating function is defined as

$$T(X) = \sum_i A_i X^i \tag{31}$$

where A_i is the number of sequences of weight i. A simplified approach to finding the generating function can be applied to the example under analysis, the convolutional code $C_{\mathrm{conv}}(2, 1, 2)$. Let us assume that the input to the modified state diagram is 1, and so the output of this diagram is then the generating function $T(X)$. In the following, the names of the states are used as phantom variables in order to estimate the generating function, obtained from the modified state diagram. This is a slight abuse of notation in which S_{b}, S_{c} and S_{d}, for instance, are utilized as variables over the modified state diagram to determine the desired generating function $T(X)$. Thus,

$$\begin{aligned}
S_{\mathrm{b}} &= X^2 + S_{\mathrm{c}} \\
S_{\mathrm{c}} &= XS_{\mathrm{b}} + XS_{\mathrm{d}} = S_{\mathrm{d}} = XS_{\mathrm{b}} + XS_{\mathrm{d}} \\
T(X) &= X^2 S_{\mathrm{c}}
\end{aligned}$$

Then

$$\begin{aligned}
XS_{\mathrm{b}} &= S_{\mathrm{c}} - XS_{\mathrm{d}} = S_{\mathrm{c}}(1 - X) \\
S_{\mathrm{b}} &= \frac{S_{\mathrm{c}}(1 - X)}{X}
\end{aligned}$$

and

$$\frac{S_{\mathrm{c}}(1 - X)}{X} = X^2 + S_{\mathrm{c}}$$

or

$$S_{\mathrm{c}} = \frac{X^3}{1 - 2X}$$

Hence,

$$T(X) = \frac{X^5}{1 - 2X} \tag{32}$$

$$\begin{array}{ll}
X^5 & |1 - 2X \\
-X^5 + 2X^6 & X^5 + 2X^6 + 4X^7 + \cdots \\
\cline{1-1}
\qquad 2X^6 & \\
\quad - 2X^6 + 4X^7 & \\
\cline{1-1}
\qquad\quad 4X^7 &
\end{array}$$

The resulting expression $T(X) = X^5 + 2X^6 + 4X^7 + \cdots$ can be interpreted as follows: There is one path of weight 5, two paths of weight 6 and four paths of weight 7, and so on. The minimum free distance of this code is therefore $d_{\mathrm{f}} = 5$. The state diagram will be analysed further in Section 6.13.

6.10 Minimum Free Distance of a Convolutional Code

The minimum free distance determines the properties of a convolutional code, and it is defined as

$$d_{\mathrm{f}} = \min \left\{ d(\boldsymbol{c}_i, \boldsymbol{c}_j) : \boldsymbol{m}_i \neq \boldsymbol{m}_j \right\} \tag{33}$$

where $\boldsymbol{c}_i, \boldsymbol{c}_j$ are two code sequences that correspond to the message sequences $\boldsymbol{m}_i \neq \boldsymbol{m}_j$. The minimum free distance is defined as the minimum distance between any two code sequences of the convolutional code. Assuming that the convolutional code is linear and that transmission over the channel makes use of a geometrically uniform signal set [17–26], the calculation of the minimum distance between any two code sequences is the same as determining the weight of the sum of these two code sequences as

$$d_{\mathrm{f}} = \min \left\{ w(\boldsymbol{c}_i \oplus \boldsymbol{c}_j) : \boldsymbol{m}_i \neq \boldsymbol{m}_j \right\} = \min \left\{ w(\boldsymbol{c}) : \boldsymbol{m}_i \neq \boldsymbol{0} \right\} \tag{34}$$

That is, the all-zero sequence is representative of the code in terms of the minimum distance evaluation of that code. This also implies that the minimum free distance of a convolutional code is the minimum weight calculated among all the code sequences that emerge from and return to the all-zero state, and that are not the all-zero sequences, $\boldsymbol{c} \neq \boldsymbol{0}$.

Example 6.5: Determine the minimum free distance of the convolutional code $C_{\mathrm{conv}}(2, 1, 2)$ used in previous examples, by employing the above procedure, implemented over the corresponding trellis.

The sequence corresponding to the path described by the sequence of states $S_{\mathrm{a}}S_{\mathrm{b}}S_{\mathrm{c}}S_{a}$, seen in bold in Figure 6.16, is the sequence of minimum weight, which is equal to 5. There are other sequences like those described by the state sequences $S_{\mathrm{a}}S_{\mathrm{b}}S_{\mathrm{c}}S_{\mathrm{b}}S_{\mathrm{c}}S_{\mathrm{a}}$ and $S_{\mathrm{a}}S_{\mathrm{b}}S_{\mathrm{d}}S_{\mathrm{c}}S_{\mathrm{a}}$ that both are of weight 6. The remaining paths are all of larger weight, and so the minimum free distance of this convolutional code is $d_{\mathrm{f}} = 5$.

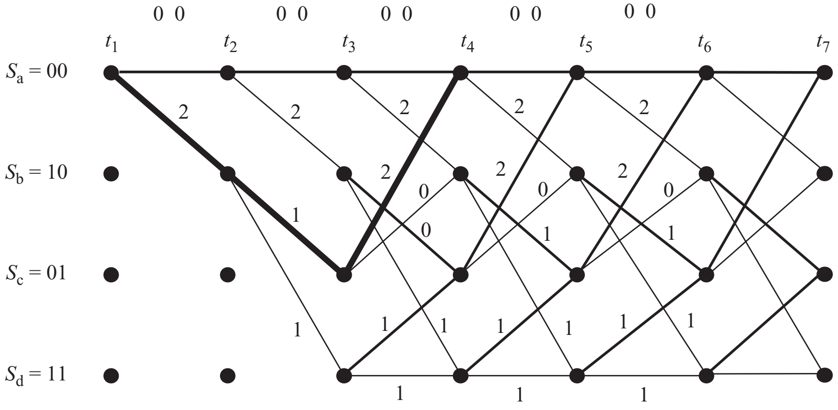


Figure 6.16 Minimum free distance sequence evaluated on the trellis

In the case of convolutional codes the distance between any two code sequences is not clearly determined, since the transmission is done not in blocks of information but as a continuous sequence of bits with a degree of memory. However, when the all-zero sequence is transmitted, and this sequence is converted by the effect of the channel errors into another code sequence, it is clear that an undetectable error pattern has occurred. This is the same as in the case of block codes, where the minimum Hamming distance can be considered as the minimum number of errors produced by the channel that have to occur in a transmitted code vector to convert it in another code vector; these errors in fact then corresponding to an undetectable error pattern. In the case of convolutional codes, the weight of the minimum-weight undetectable error pattern is also the same as the minimum free distance d_f of the code. The error-correction capability of the code is then defined as the number t of errors that can be corrected, which is equal to

$$t = \left\lfloor \frac{d_\mathrm{f} - 1}{2} \right\rfloor \tag{35}$$

This error-correction capability is obtained when error events are separated by at least the constraint length of the code, measured in bits.

6.11 Maximum Likelihood Detection

For a given code sequence $\boldsymbol{c}$ generated by the encoder of a convolutional code, the channel noise converts this sequence into the received sequence $\boldsymbol{s}_\mathrm{r}$, which is essentially the code sequence $\boldsymbol{c}$ with errors produced in the transmission. An optimal decoder is one that is able to compare the conditional probabilities $P(\boldsymbol{s}_\mathrm{r}/\boldsymbol{c}')$ that the received sequence $\boldsymbol{s}_\mathrm{r}$ corresponds to a possible code sequence $\boldsymbol{c}'$, and then decide upon the code sequence with the highest conditional probability:

$$P(\boldsymbol{s}_\mathrm{r}/\boldsymbol{c}') = \max_{\text{all } c} P(\boldsymbol{s}_\mathrm{r}/\boldsymbol{c}) \tag{36}$$

This is the maximum likelihood criterion. It is in agreement with the intuitive idea of decoding by selecting the code sequence that is most alike the received sequence. The application of this criterion in the case of convolutional decoding faces the fact that there are so many possible code sequences to be considered in the decoding procedure. For a code sequence of length L bits, there are $2^{R_\mathrm{c}L}$ possible sequences, where R_c is the rate of the code. The maximum likelihood decoder selects a sequence $\boldsymbol{c}'$, from the set of all these possible sequences, which has the maximum similarity to the received sequence.

If the channel is memoryless, and the noise is additive, white and Gaussian (AWGN), each symbol is independently affected by this noise. For a convolutional code of rate $1/n$, the probability of being alike to the received sequence is measured as

$$P(\boldsymbol{s}_\mathrm{r}/\boldsymbol{c}) = \prod_{i=1}^{\infty} P(s_{ri} \,/\, c_i) = \prod_{i=1}^{\infty} \prod_{j=1}^{n} P(s_{\mathrm{r},ji} \,/\, c_{ji}) \tag{37}$$

where on the trellis of the code $s_{\mathrm{r}_\mathrm{i}}$ is the ith branch of the received sequence $\boldsymbol{s}_\mathrm{r}$, c_i is the ith branch of the code sequence $\boldsymbol{c}$, $s_{\mathrm{r},ji}$ is the jth symbol of s_{r_i}, and c_{ji} is the jth code symbol of c_i, and where each branch is constituted of n code symbols. The decoding procedure consists of selecting a sequence that maximizes the probability function (37). One algorithm that performs this procedure for convolutional codes is the Viterbi decoding algorithm.

6.12 Decoding of Convolutional Codes: The Viterbi Algorithm

The Viterbi algorithm (VA) performs maximum likelihood decoding. It is applied to the trellis of a convolutional code whose properties are conveniently used to implement this algorithm. As explained above, one of the main problems that faces maximum likelihood decoding is the number of calculations that have to be done over all the possible code sequences. The VA reduces this complexity of calculation by avoiding having to take into account all the possible sequences. The decoding procedure consists of calculating the cumulative distance between the received sequence at an instant t_i at a given state of the trellis, and each of all the code sequences that arrive at that state at that instant t_i. This calculation is done for all states of the trellis, and for successive time instants, in order to look for the sequence with the minimum cumulative distance. The sequence with the minimum cumulative distance is the same as the sequence with the highest probability of being alike to the received sequence if transmission is done over the AWGN channel.

The following example illustrates the application of the Viterbi decoding algorithm. The distance used as a measure of the decoding procedure is the Hamming distance; that is, the distance between any two sequences is defined as the number of differences between these two sequences.

Example 6.6: Apply the Viterbi decoding algorithm to the convolutional code of Figure 6.12, whose trellis is seen in Figure 6.13, if the received sequence is $\boldsymbol{s}_{\mathrm{r}} = 11\,01\,01\,00\,11\ldots$.

The first step in the application of this algorithm is to determine the Hamming distance between the received sequence and the outputs at the different states and time instants, over the corresponding trellis. This is shown in Figure 6.17.

Message sequence	1 0 1 0 1
Code sequence	11 01 11 00 11
Received sequence	11 01 01 00 11

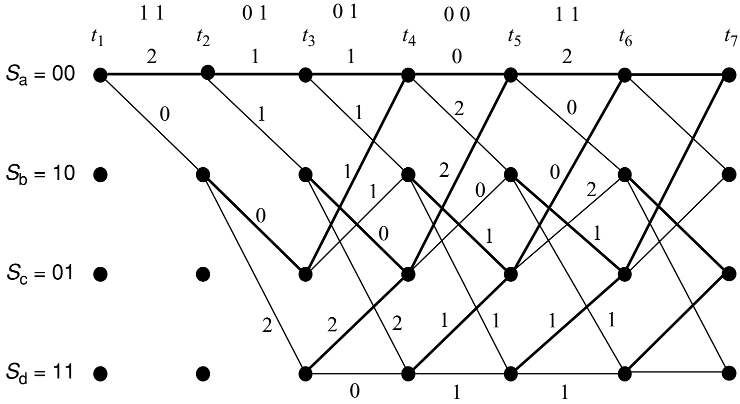


Figure 6.17 Hamming distance calculations for the VA

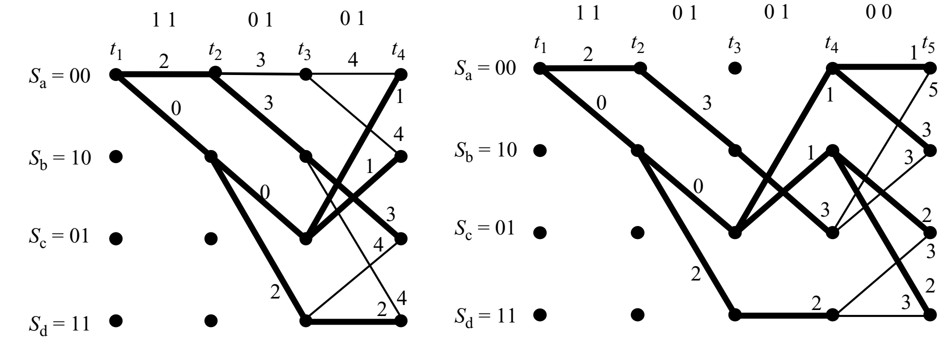


Figure 6.18 Survivor paths in the VA

The essence of the VA is that when two or more paths arrive at a given time instant and state of the trellis, only one of them would have the minimum cumulative distance, and should be selected from among the others as the survivor. In fact, this decision procedure significantly reduces the number of distance calculations required. Decisions of this kind start to be performed as soon as two or more paths arrive at the same state and time instant of the trellis. This happens after time instant t_4 in this example, as seen in Figure 6.18.

In this particular example, at time instant t_4, and at successive time instants, decisions can be taken at all the states of the trellis. The reason is that there are two paths arriving at each state, but only one of them has the minimum Hamming distance in each case. The other path has a higher cumulative Hamming distance, and it is discarded. However, it is seen that at time instant t_5, in state S_b, there are two arriving paths that both have the same cumulative Hamming distance. In this case the decision is taken by randomly selecting one of these two possible paths as the survivor; the upper path in this example. On average, these random selections do not prevent effective operation of the VA: If the error pattern is within the error-correction capability of the code, then the decoded sequence does not pass through the state node concerned; if the code's correction capability has been exceeded, then the decoder fails anyway, and normally outputs a burst of errors in the decoded sequence.

The discarding procedure is successively applied to the following time instants over each state, now taking into account previous decisions already taken. After a given number of time instants, the procedure is truncated, and the sequence with the minimum cumulative Hamming distance is selected as the decoded sequence. This is shown for this example in Figure 6.19.

As seen in Figure 6.20, the decision taken at time instant t_6 has produced the correct decoded sequence. The sequence selected as the final survivor is that of minimum cumulative distance. Then, by looking at the information provided by the trellis of Figure 6.6, the transmitted message is finally determined by identifying along the decoded sequence which is the input message bit that generated each of its transitions. In this example the decoder determines that the transmitted message was the sequence 10101 ..., and at this point it can also determine that the decoder could correct one error present at the received sequence s_r. As is seen in this example, the Viterbi decoding algorithm leads directly to the estimated message sequence, and performs error correction without the need of decoding table look-up, or of algebraic equation solution, as in the case of traditional syndrome decoding of block codes. This fact

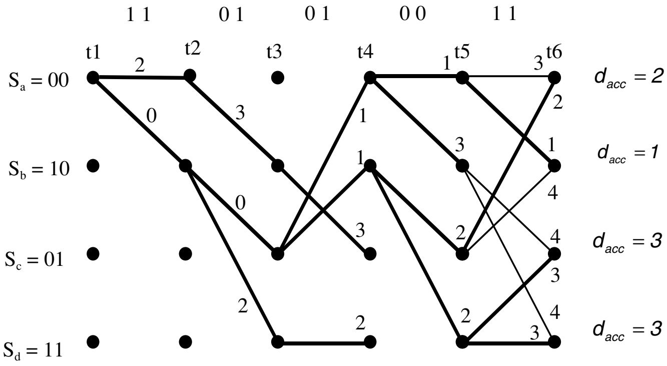


Figure 6.19 Viterbi decoding algorithm, time instant t_6

makes convolutional codes particularly efficient in FEC systems, or in general, for those coding schemes where error correction is preferred over error detection.

In the above example, the decision determining the maximum likelihood path at time instant t_6 was easily taken because the value of the minimum cumulative distance was unique. However, it is possible that at the moment of truncating the decoding sequence, to decide the final survivor sequence, more than one sequence could have the minimum value of the cumulative distance. To mitigate any problems this might cause, the algorithm operates as follows. As seen in Figure 6.19, the survivor paths at time instant t_6 are the same in the first stage of the trellis (from t_1 to t_2). This indicates that, with high probability, the first two bits of the transmitted sequence were 11, which corresponds to the message bit 1. The survivors are also the same in the second stage, but disagree in subsequent stages. If the message sequence, the code sequence and the received sequence are long enough, it can be shown that survivor paths agree with high probability at the time instant t_i, and that the decoding decision taken at time t_i is correct, if the survivor sequences are extended to time instant t_{i+J}, where J is called the decoding length, measured in time instants. It can be heuristically shown that the error-correction capability of the convolutional code is maximized if J is approximately equal to five times the constraint length of the code; that is, $J \approx 5(K + 1)$.

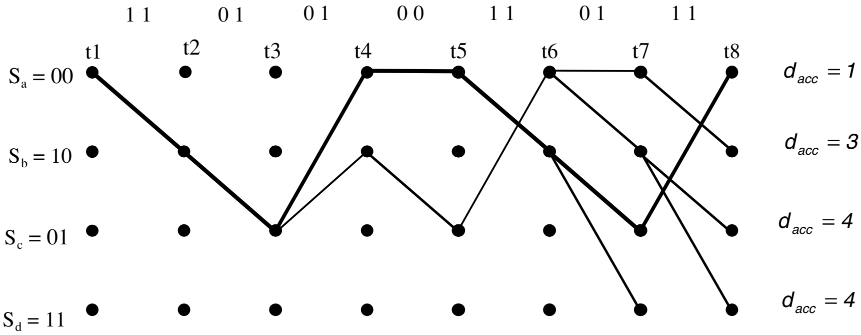


Figure 6.20 Viterbi decoding algorithm, decoded sequence at time instant t_8

This implies that, on one hand, convolutional codes are more powerful for longer messages, and, on the other hand, it is necessary to add kK zeros to the end of the message sequence in order to maintain the error-correction capability of the code at the end of a given message. This makes the trellis terminate in the all-zero state S_a, enabling the decoding of a unique survivor, as seen in Figure 6.20.

6.13 Extended and Modified State Diagram

The extended and modified state diagram is a useful tool for the analysis of convolutional codes. The exponent of the variable X^i is again used to indicate the weight of the corresponding branch, the exponent of the variable Y^j is used to determine the length of the branch j and variable Z is present only in the description of a given branch when this branch has been generated by an input '1'. The modified generating function $T(X, Y, Z)$ is then defined as

$$T(X, Y, Z) = \sum_{i,j,l} A_{i,j,l} X^i Y^j Z^l \tag{38}$$

where $A_{i,j,l}$ is the number of paths of weight i, which are of length j, and which result from an input of weight l. The extended and modified state diagram of the convolutional code $C_\mathrm{conv}(2, 1, 2)$ of the example under analysis, whose trellis is seen in Figure 6.6, is seen in Figure 6.21.

If again the labels of the states are taken as phantom variables, and letting $S_\mathrm{a} = 1$,

$$S_\mathrm{b} = X^2YZS_\mathrm{a} + YZS_\mathrm{c}, \quad \text{but } S_\mathrm{a} = 1$$

and so

$$\begin{aligned} S_\mathrm{b} &= X^2YZ + YZS_\mathrm{c} \\ S_\mathrm{c} &= XYS_\mathrm{b} + XYS_\mathrm{d} \\ S_\mathrm{d} &= XYZS_\mathrm{d} + XYZS_\mathrm{b} \\ T(X, Y, Z) &= X^2YS_\mathrm{c} \end{aligned}$$

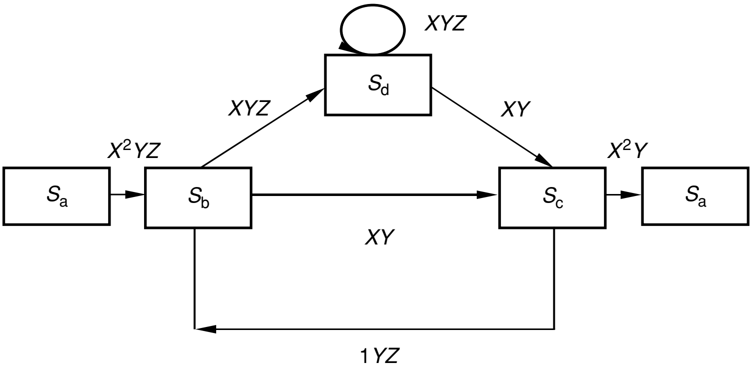


Figure 6.21 Extended and modified state diagram

By operating over these phantom variables, it follows that

$$T(X, Y, Z) = \frac{X^5Y^3Z}{1 - XYZ(1 + Y)}$$

$$\begin{array}{ll}
X^5Y^3Z & |1 - XY(1+Y)Z \\
-X^5Y^3Z + X^6Y^4Z^2(1+Y) & X^5Y^3Z + X^6Y^4Z^2(1+Y) + \cdots \\
\hline
\qquad X^6Y^4Z^2(1+Y) & \\
\quad -X^6Y^4Z^2(1+Y) + X^7Y^5Z^3(1+Y)^2 & \\
\hline
\qquad\qquad X^7Y^5Z^3(1+Y)^2 &
\end{array}$$

Thus, the modified generating function for this example is equal to

$$T(X, Y, Z) = X^5Y^3Z + X^6Y^4Z^2(1 + Y) + X^7Y^5Z^3(1 + Y)^2 + \cdots \tag{39}$$

There is a path of weight 5, with length 3 (three transitions), generated by an input '1'. There are also two paths of weight 6, which are $X^6Y^4Z^2$ and $X^6Y^4Z^2Y$, and both are generated by an input sequence of two '1's, the former being of length 4 transitions and the latter of length 5 transitions. The modified generating function is useful for assessing the performance of a convolutional code.

6.14 Error Probability Analysis for Convolutional Codes

Bit error rate (BER) performance is a straightforward measure of the error-correction capability of any coding technique, and of course of convolutional codes. The most useful error probability measure is the bit error probability, but first it is simpler to analyse the so-called node error probability [1, 2, 4].

As described in previous sections, the all-zero sequence of a convolutional code can be used to represent any code sequence in terms of the behaviour of the code in the presence of errors, and all sequences that emerge from and return to the all-zero state S_a can be seen as modified versions of the all-zero sequence that channel errors convert into other valid code sequences. The node error probability is the probability that an erroneous sequence which emerges from that node on the all-zero sequence (where a node is a state at a given time instant), and later returns to it, is selected as a valid code sequence. The erroneous sequence is selected as valid, instead of the correct sequence, if the number of coincidences the received sequence $\boldsymbol{s}_\mathrm{r}$ has with respect to the erroneous code sequence is higher than the number of coincidences it has with respect to the correct sequence (in this analysis, the all-zero sequence).

The erroneous paths are those defined by the modified generating function $T(X, Y, Z)$. In the example under analysis in the previous section [equation (39)], there will be a node error if three or more bits in the received sequence $\boldsymbol{s}_\mathrm{r}$, of the five positions in which the erroneous and the correct paths differ, are closer to the erroneous sequence than to the correct one. This indicates that the weight of the received sequence is 3 or larger. In the case of the binary symmetric channel (BSC) with error or transition probability p, the probability P_e of this occurring is

$$P_{\mathrm{e}} = P(3 \text{ or more '1's in the received sequence}) = \sum_{e=3}^{5} \binom{5}{e} p^e (1-p)^{5-e}$$

For paths of length 6, an error event of length 3 generates a sequence that can be adopted as correct or not, with equal probability. Paths with weights equal to or larger than 4 will be erroneously decoded as

$$P_{\mathrm{e}} = \binom{6}{3} p^3 (1-p)^3 + \sum_{e=1}^{6} \binom{6}{e} p^e (1-p)^{6-e}$$

In general, for an erroneous path of weight d,

$$P_d = \begin{cases} \sum_{e=(d+1)/2}^{d} \binom{d}{e} p^e (1-p)^{d-e} & d \text{ odd} \\ \frac{1}{2} \binom{d}{d/2} p^{d/2} (1-p)^{d/2} + \sum_{e=d/2+1}^{d} \binom{d}{e} p^e (1-p)^{d-e} & d \text{ even} \end{cases} \tag{40}$$

The node error probability is bounded by the union bound of all the possible events of this kind, including all the possible erroneous paths at node j. This bound is given by

$$P_{\mathrm{e}}(j) \le \sum_{d=d_{\mathrm{f}}}^{\infty} A_d P_d \tag{41}$$

where A_d is the number of code sequences of weight d, since there are A_d paths of that Hamming weight with respect to the all-zero sequence. This bound is indeed independent of j, and so it is equal to the desired node error probability, $P_{\mathrm{e}}(j) = P_{\mathrm{e}}$.

This error probability can be upper bounded. For d odd,

$$\begin{aligned} P_d &= \sum_{e=(d+1)/2}^{d} \binom{d}{e} p^e (1-p)^{d-e} < \sum_{e=(d+1)/2}^{d} \binom{d}{e} p^{d/2} (1-p)^{d/2} \\ &= p^{d/2} (1-p)^{d/2} \sum_{e=(d+1)/2}^{d} \binom{d}{e} < p^{d/2} (1-p)^{d/2} \sum_{e=0}^{d} \binom{d}{e} \\ &= 2^d p^{d/2} (1-p)^{d/2} \end{aligned} \tag{42}$$

This expression is also valid for even values of d. The above bound can be used to provide an upper bound on the node error probability [1, 2, 4]:

$$P_{\mathrm{e}} < \sum_{d=d_{\mathrm{f}}}^{\infty} A_d \left[2\sqrt{(1-p)p} \right]^d \tag{43}$$

This expression is related to the generating function $T(X) = \sum_{d=d_\mathrm{f}}^{\infty} A_d X^d$, so that

$$P_\mathrm{e} < T(X)\Big|_{X = 2\sqrt{(1-p)p}} \tag{44}$$

Since the error probability on a BSC is, in general, a small number $p \ll 1$, the above summation can be approximately calculated from its first term as follows:

$$P_\mathrm{e} < A_{d_\mathrm{f}}\left[2\sqrt{p(1-p)}\right]^{d_\mathrm{f}} = A_{d_\mathrm{f}} 2^{d_\mathrm{f}}\,[p(1-p)]^{d_\mathrm{f}/2} \approx A_{d_\mathrm{f}} 2^{d_\mathrm{f}} p^{d_\mathrm{f}/2} \tag{45}$$

Example 6.7: For the convolutional code C_conv (2, 1, 2), $d_\mathrm{f} = 5$ and $A_{d_\mathrm{f}} = 1$, and if for instance $p = 10^{-2}$, then

$$P_\mathrm{e} < A_{d_\mathrm{f}} 2^{d_\mathrm{f}} p^{d_\mathrm{f}/2} = 1.2^5\left(10^{-2}\right)^{2,5} \approx 3.10^{-4}$$

The above analysis corresponds to the evaluation of the node error probability. This analysis can then be used to determine the bit error probability, or bit error rate.

In each error event, there are erroneous bits that are in number equal to the weight of the erroneous sequence. An estimate of the number of erroneous bits in a given time unit can be made if the error probability P_d is modified by the total number of '1's that all the sequences of weight d have. This number can be divided by k, which is the number of message bits transmitted in that time unit, in order to obtain

$$P_\mathrm{be} < \frac{1}{k}\sum_{d=d_\mathrm{f}}^{\infty} B_d P_d \tag{46}$$

where B_d is the total number of '1's that all the sequences of weight d have.

The extended and modified generating function is expressed as

$$T(X, Y, Z) = \sum_{i,j,l} A_{i,j,l} X^i Y^j Z^l$$

where $A_{i,j,l}$ is the number of paths of weight i, of length$j$, and generated by an input of weight l. This expression can be arranged to adopt the form

$$T(X, Z) = \sum_{d=d_\mathrm{f}}^{\infty}\sum_{b=1}^{\infty} A_{d,b} X^d Z^b \tag{47}$$

and, by calculating its derivative with respect to Z,

$$\frac{\partial T(X, Z)}{\partial Z}\Bigg|_{Z=1} = \sum_{d=d_\mathrm{f}}^{\infty}\sum_{b=1}^{\infty} b A_{d,b} X^d = \sum_{d=d_\mathrm{f}}^{\infty} B_d X^d \tag{48}$$

is obtained, where

$$B_d = \sum_{b=1}^{\infty} b A_{d,b} \tag{49}$$

By using the calculated bound for P_d,

$$P_d = 2^d p^{d/2}(1-p)^{d/2} \tag{50}$$

$$P_{\text{be}} < \frac{1}{k}\sum_{d=d_{\text{f}}}^{\infty} B_d P_d = \frac{1}{k}\sum_{d=d_{\text{f}}}^{\infty} B_d \left[2\sqrt{p(1-p)}\right]^d = \frac{1}{k}\frac{\partial T(X,Z)}{\partial Z}\Bigg|_{\substack{Z=1 \\ X=2\sqrt{p(1-p)}}} \tag{51}$$

Expression (51) can be simplified by assuming that the most significant term is the first of the summation, so that [1]

$$P_{\text{be}} \approx \frac{1}{k} B_{d_{\text{f}}} 2^{d_{\text{f}}} p^{d_{\text{f}}/2} \tag{52}$$

Example 6.8: For the case of the convolutional code $C_{\text{conv}}(2,1,2)$, with $d_{\text{f}} = 5$ and $B_{d_{\text{f}}} = 1$, and since $T(X,Y,Z) = X^5Y^3Z^1 + \cdots$, then if for instance $p = 10^{-3}$, the BER is equal to $P_{\text{be}} < B_{d_{\text{f}}} 2^{d_{\text{f}}} p^{d_{\text{f}}/2} = 1.2^5 \times (10^{-3})^{2,5} \approx 1.01 \times 10^{-6}$.

6.15 Hard and Soft Decisions

So far most of the coding techniques introduced have made use of decoding algorithms based on what is called hard-decision detection. A hard-decision detector takes samples of the received signal, determines whether each sample is over or under a given threshold and thus decides whether the incoming signal represents a ‘1’ or a ‘0’. Samples are taken using a synchronizing signal that is generated by other parts of the communication system, so that, from the decoding point of view, perfect synchronization is assumed.

This procedure is usually called hard-decision detection because the input signal, essentially in the form of a continuous waveform, is translated into a discrete alphabet in which there are only two possible values for binary signalling, or q possible values for non-binary signalling. Therefore, continuous channel information is converted into discrete detected information. Based on this hard decision, in most of the codes already studied, the decoder was implemented by evaluating the Hamming distance, d_{H}, between a code vector or sequence and a received vector or sequence.

However, when considering the concept of distance, a more intuitive idea is to think about the so-called Euclidean distance, which is the real-number distance normally measured in geometrical space. This concept known as the Euclidean distance d can be generalized for a vector space of dimension n. In this case and given the vectors $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$, the Euclidean distance is the norm of the difference between these two vectors $d = ||\boldsymbol{c}_1 - \boldsymbol{c}_2||$. The reason for being interested in Euclidean distance is that it enables the use of soft-decision detection, which overcomes the loss of information inherent in hard-decision detection. In hard-decision detection, sampled values are converted into the same hard value no matter how close to, or far away from, the decision threshold they are. In soft-decision detection, the distance of a sample value from the decision threshold is measured, and then used to enhance the decoding process.

The vector representation of code vectors of the triple repetition code and the even parity code with $n = 3$, already described in Chapter 2, can help to introduce the concept and importance of soft-decision detection and decoding. A given bit is usually transmitted or represented by a

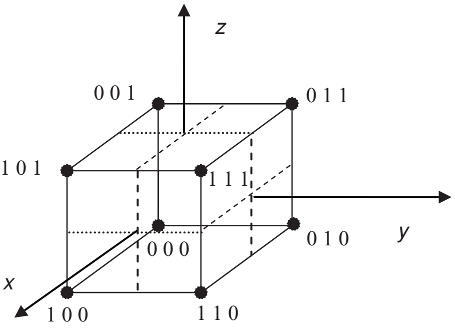


Figure 6.22 Vector representation (polar format) of code vectors in a vector space of dimension $n = 3$

signal, such as a rectangular pulse of normalized amplitude ±1 in polar format. Code vectors can be represented as vectors of a vector space, so that the Euclidean distance between any two of them can also be determined. Thus, for instance, both the triple repetition code and the even parity code, defined for a code length of $n = 3$, have a vector representation in a vector space of dimension $n = 3$, as can be seen in Figure 6.22. In this figure the binary format is replaced by the polar format ($1 \rightarrow +1, 0 \rightarrow -1$). Code vector projections over x, y, z are all equal to ± 1.

Figure 6.22 shows a vector space of dimension $n = 3$. Thus, for instance, the Euclidean distance between the two code vectors (000) and (111) of the triple repetition code, represented in polar format, is equal to $2\sqrt{3}$. Synchronized samples taken of the received signal are the n real numbers that are the coordinates of the received vector, which can also be represented in the same vector space of Figure 6.22. Then it is possible to evaluate the Euclidean distance between this received vector, and any code vector of a given code.

Example 6.9: Determine the Euclidean distance between the received vector and each of the code vectors of the even parity code with $n = 3$, when the transmitted code vector is (101). The transmission is done in polar format. Assume that after this transmission over a Gaussian channel, the received signal looks like that seen in Figure 6.23.

Figure 6.23 shows the code vector 101 $(+1\ -1\ +1)$ of the even parity code with $n = 3$, which was transmitted in polar format through a Gaussian channel, and altered by the effect of this noise. This signal $x(t)$ is sampled to obtain the following sampled values:

$$x(15) = -0.1072$$

$$x(45) = -1.0504$$

$$x(75) = +1.6875$$

The received vector is then $r = (-0.1072\ -1.0504\ +1.6875\,)$. The noise-free vectors of this code are $\boldsymbol{c}_1 = (\,-1\ -1\ -1\,)$, $\boldsymbol{c}_2 = (\,-1\ +1\ +1\,)$, $\boldsymbol{c}_3 = (\,+1\ -1\ +1\,)$ and $\boldsymbol{c}_4 = (\,+1\ +1\ -1\,)$. If hard decisions are taken, the decoded vector would be equal to (001), and so the even parity code would detect an error and would require a retransmission for a correct reception of this information. However, if soft decisions are performed by calculating the Euclidean distance

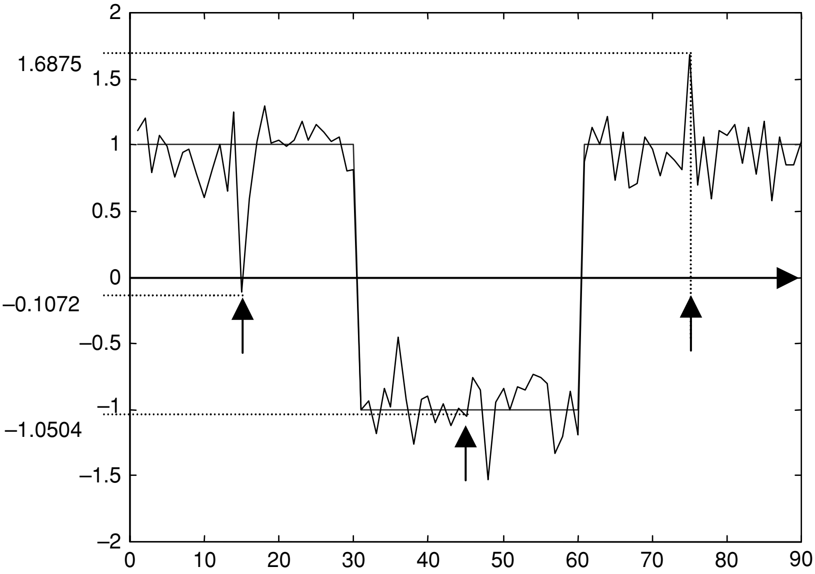


Figure 6.23 Signal resulting from the transmission of the code vector (101) in polar format over a Gaussian channel

between the received vector and each of the code vectors, it would happen that

$$d(\boldsymbol{r}, \boldsymbol{c}_1) = \sqrt{(-0.1072 + 1)^2 + (-1.0504 + 1)^2 + (1.6875 + 1)^2} = 2.8324$$

and

$$d(\boldsymbol{r}, \boldsymbol{c}_2) = 2.3396$$

$$d(\boldsymbol{r}, \boldsymbol{c}_3) = 1.3043$$

$$d(\boldsymbol{r}, \boldsymbol{c}_4) = 3.5571$$

A soft-decision decoder would then decide that the decoded vector is the code vector that has the minimum Euclidean distance with respect to the received vector, and, in this case, this decoder will decide in favour of the correct code vector, which is $\boldsymbol{c}_3$. This is a simple example to show the essential difference between hard- and soft-decision detection, and the decoding advantage that the latter provides.

Essentially, hard decision is closely related to the BSC, as introduced in Chapter 1, where symbols 0 and 1 have a probability p of being in error, and the output alphabet is also binary. This approach leads to the design of decoding algorithms that operate over the binary field, such as syndrome error detection and correction.

The BSC, characterized by being symmetric, binary and memoryless, can be extended in such a way that each source symbol, either 0 or 1, is transmitted as a signal whose samples belong to a continuous alphabet with values in the range $(-\infty, +\infty)$. This is the so-called Gaussian channel, and it is also a memoryless channel. A soft decision is performed if the detector produces a value that belongs to a continuous alphabet (i.e., a real number), but the same is also true, with little loss in performance advantage, even when the decision is performed by producing a discrete quantized value of the continuous alphabet. In general terms, the BER performance of a soft-decision decoder will be better than that of a hard-decision decoder, for the same code, but this advantage is obtained in a trade-off with decoding complexity. At first sight, soft decision can be applied without restrictions to the decoding of both block and convolutional codes. However, and particularly in the case of long block codes, the calculations involved are of such a complexity that optimal soft decision of these block codes is difficult to implement in practice. But it is possible to represent block codes, as well as convolutional codes, by means of a trellis [31]. In addition, the VA is easily modified to make use of soft-decision detection, so that optimal soft-decision trellis decoding is feasible for block codes using the VA, with a consequent reduction in complexity. Powerful convolutional codes can also be of considerable size, and therefore complex to decode. Again, however, the use of the soft-decision VA will considerably reduce the complexity of optimal decoding of a powerful convolutional code.

6.15.1 Maximum Likelihood Criterion for the Gaussian Channel

As seen in Section 6.12, optimal decoding of convolutional codes is related to the maximum likelihood algorithm, which essentially looks for the most likely code sequence among all possible code sequences, based on the similarity of these sequences with respect to the received sequence.

The maximum likelihood criterion assumes that information bits or message bits are equally likely. A given signal $s_i(t)$ taken from the set of signals $\{s_i(t)\}$, with $i = 0, 1, \ldots, M-1$, is transmitted, and this signal is affected by noise to become the received signal $r(t) = s_i(t) + n(t)$. At the time instant T, the signal is sampled, and the value of the sampled signal is

$$y(T) = b_i(T) + n(T) \tag{53}$$

where $b_i(T)$ represents the value of the noise-free signal that represents the transmitted symbol, and $n(t)$ is a random Gaussian variable of zero mean value.

Since the decoder attempts to determine the value of a transmitted message bit based on the received signal, and considering the binary case for instance, where there are just two transmitted signals, the maximum likelihood criterion can be stated as follows:

$$\begin{array}{ll} P(s_1/y) > P(s_0/y) & \text{the decoder decides for hypothesis } H_1 \\ P(s_0/y) > P(s_1/y) & \text{the decoder decides for hypothesis } H_0 \end{array} \tag{54}$$

Hypothesis H_1 corresponds to the transmission of symbol '1', and hypothesis H_0 corresponds to the transmission of symbol '0'.

By using the Bayes rule, these conditional probabilities can be expressed in the following manner:

$$\begin{array}{ll} p(y/s_1)P(s_1) > p(y/s_0)P(s_0) & \text{the decoder decides for hypothesis } H_1 \\ p(y/s_1)P(s_1) < p(y/s_0)P(s_0) & \text{the decoder decides for hypothesis } H_0 \end{array} \tag{55}$$

or, equivalently,

$$\begin{aligned}
&\frac{p(y/s_1)}{p(y/s_0)} > \frac{P(s_0)}{P(s_1)} \quad \text{the decoder decides for hypothesis } H_1\\
&\frac{p(y/s_1)}{p(y/s_0)} < \frac{P(s_0)}{P(s_1)} \quad \text{the decoder decides for hypothesis } H_0
\end{aligned} \tag{56}$$

which is known as the likelihood ratio. If the transmitted symbols are equally likely, then

$$\begin{aligned}
&\frac{p(y/s_1)}{p(y/s_0)} > 1 \quad \text{the decoder decides for hypothesis } H_1\\
&\frac{p(y/s_1)}{p(y/s_0)} < 1 \quad \text{the decoder decides for hypothesis } H_0
\end{aligned} \tag{57}$$

Let us assume that the two possible transmitted signals were s_1 and s_0, such that the random variables obtained after sampling the corresponding signals are $y(T) = b_1 + n$ and $y(T) = b_0 + n$. If the transmission is over an AWGN channel, the probability density function that characterizes these random variables is of the form

$$p(n) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{n}{\sigma}\right)^2}$$

This expression describes the previously presented sampled random variables when $b_0 = b_1 = 0$. For the general case, where transmitted symbols are different from zero, the probability density function is also a Gaussian probability density function, but with a mean value that is equal to the noise-free value of that symbol.

The likelihood ratio can also be expressed in terms of these probability density functions for each of the transmitted symbols, so that if

$$\frac{\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{y-b_1}{\sigma}\right)^2}}{\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{y-b_0}{\sigma}\right)^2}} > \frac{P(s_0)}{P(s_1)}, \quad \text{the decoder decides for hypothesis } H_1 \tag{58}$$

or, simplifying, if

$$e^{-\frac{1}{2}\left[\left(\frac{y-b_1}{\sigma}\right)^2 - \left(\frac{y-b_0}{\sigma}\right)^2\right]} > \frac{P(s_0)}{P(s_1)}, \quad \text{the decoder decides for hypothesis } H_1 \tag{59}$$

By applying natural logarithms to both sides of the inequality, if

$$\frac{1}{2\sigma^2}\left[(y-b_1)^2 - (y-b_0)^2\right] > \ln\left(\frac{P(s_1)}{P(s_0)}\right), \quad \text{the decoder decides for hypothesis } H_1$$

And, for equally likely symbols, if

$$(y-b_1)^2 > (y-b_0)^2, \quad \text{the decoder decides for hypothesis } H_1 \tag{60}$$

For the Gaussian channel, the maximum likelihood decision is equivalent to the soft minimum-distance decision. This Euclidean distance is calculated between the sampled values of the received signal and the values that should correspond to the elements of a given code sequence. The decision is taken over the whole code vector or sequence. Thus, the maximum likelihood criterion is applied by considering which code vector or sequence is closest to the received vector or sequence, that is, which code vector or sequence minimizes the distance with respect to the received vector or sequence.

In the case of block codes, the maximum likelihood criterion decides that the code vector $\boldsymbol{c}_i$ is the decoded vector if

$$d(\boldsymbol{r}, \boldsymbol{c}_i) = \min \left\{ d(\boldsymbol{r}, \boldsymbol{c}_j) \right\} \quad \text{for all } \boldsymbol{c}_j \tag{61}$$

In the case of convolutional codes, the definition of block length is lost, since it becomes semi-infinite. As we saw above, however, for the current purpose it can be replaced by the decoding length. A given code sequence $\boldsymbol{c}$ generated by a convolutional code converts into a received sequence $\boldsymbol{s}_\mathrm{r}$, which is essentially the code sequence $\boldsymbol{c}$ containing some errors, generated by the effect of the channel noise. An optimal decoder will compare the conditional probability $P(\boldsymbol{s}_\mathrm{r}/\boldsymbol{c}')$ with the received sequence $\boldsymbol{s}_\mathrm{r}$ corresponding to one of the possible code sequences $\boldsymbol{c}'$, and will take the decision in favour of that code sequence that has the maximum probability of being $\boldsymbol{s}_\mathrm{r}$:

$$P(\boldsymbol{s}_\mathrm{r}/\boldsymbol{c}) = \max_{\text{all } \boldsymbol{c}} \boldsymbol{s}_\mathrm{r}/\boldsymbol{c} \tag{62}$$

6.15.2 Bounds for Soft-Decision Detection

A symmetric channel that models a soft-decision-detection scheme is, for example, that shown in Figure 6.24.

The number of symbols of the output alphabet of this soft-decision channel can be increased until it is virtually converted into a continuous output channel. If, for instance, the soft-decision channel of Figure 6.24 would have an output alphabet of eight outputs, usually labelled outputs 0–7, then when the detector provides output 7, for example, it gives decoding information that means that the decoded bit is highly likely to be a ‘1’. If however the detected value was

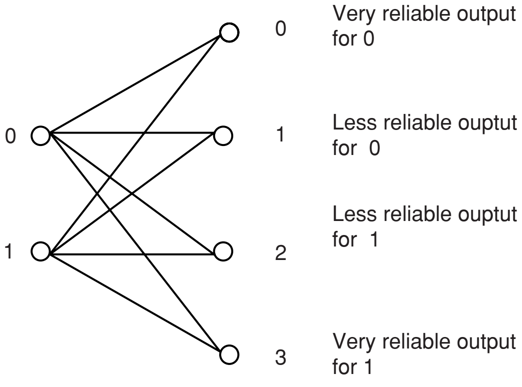

0
1
0
1
2
3
Very reliable output for 0
Less reliable ouptut for 0
Less reliable ouptut for 1
Very reliable output for 1

Figure 6.24 A soft-decision channel

a 4, it would be giving decoding information that means that the decoded bit is still a ‘1’, but with relatively low probability. This decision contains more information than that of a hard decision, which would just consist of deciding that the detected bit is a ‘1’ without any additional information about the reliability of that decision. This additional information would not be useful if the bit was to be decoded alone. However, when bits are related to other bits in a given code vector or code sequence, the additional information provided by the soft decisions helps to improve the estimates of all the bits.

It can be shown that there is an additional coding gain of around 2.2 dB, if soft decision is applied instead of hard decision. This is the case when the soft-decision detector operates over a continuous output range. If instead the output is quantized using eight values or levels, then this coding gain is about 2 dB, and so there is not a significant improvement when using more than eight quantization levels. Figure 6.25 shows the BER performance of the triple repetition code ($n = 3$) described in Chapter 2, with curves for both hard and unquantized soft-decision decoding, as well as for no coding. It can be seen that the coding gain of the soft-decision decoder with respect to the hard-decision decoder approaches asymptotically the value of 2.2 dB.

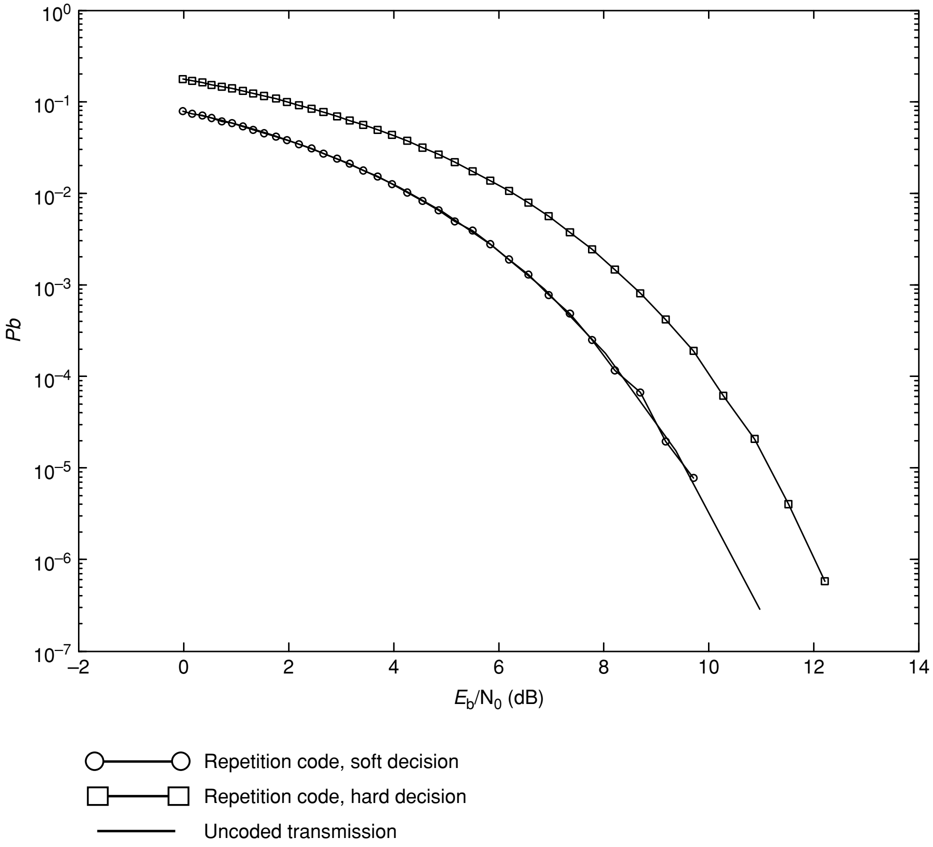


Figure 6.25 A comparison between hard- and soft-decision decoding of the triple repetition code ($n = 3$), and uncoded transmission

As pointed out in Chapter 2, the triple repetition code has a low code rate $R_{\mathrm{c}} = 1/3$ that produces a poor BER performance, which is even worse than uncoded transmission, if a hard-decision decoder is utilized. A curious phenomenon is seen in Figure 6.25, where the BER performance of the triple repetition code using a soft-decision decoder is practically coincident with that of uncoded transmission. This is so because, at the end of the day, the triple repetition of a given bit is the same as to transmit it with a signal of three times more energy than in the case of uncoded transmission. The soft-decision decoder takes advantage of this additional energy, so that repetition is in a way equivalent to an increase in the number of output levels of the detector to more than 2. As pointed out above, the improvement of applying soft decision is not very significant for more than eight levels at the output of the soft-decision channel.

From this explanation it can be concluded that the triple repetition code is equivalent to uncoded transmission if the redundancy added is understood as an increase of the bit energy, a fact that should be taken into account in order to make a fair comparison of BER performances. This increase in the energy per bit, which is determined by the rate R_{c} of the code, is a penalty that must be offset against the gain in performance offered by the error-correcting power of the code. Seen in a rather different way, the triple repetition code is equivalent to uncoded transmission where each bit is sampled three times at the detector. In the case of soft-decision decoding, the decoder uses the values of these three samples to evaluate the Euclidean distance over an AWGN channel.

When its use is possible, soft-decision decoding provides a better BER performance than that of hard-decision decoding of the same code, for all the coding techniques presented in previous chapters. Sometimes, however, it is impractical to use soft-decision decoding, as discussed previously. In the case of very long block length RS codes, for example, even soft-decision trellis decoding has an implementation complexity which makes it impractical for most applications.

6.15.3 An Example of Soft-Decision Decoding of Convolutional Codes

A soft-decision decoder for the convolutional code described in Figure 6.12, whose trellis is also seen in Figure 6.13, is presented in order to show differences between hard- and soft-decision decoding. In this example the message sequence is $\boldsymbol{m} = (10101)$, which after being encoded and transmitted in polar format, produces the code sequence $\boldsymbol{c} = (+1\,+1\ -1\,+1\ +1\ +1\ -1\,-1\ +1\,+1)$. This sequence is transmitted over an AWGN channel.

Let $\boldsymbol{s}_{\mathrm{r}} = (+1.35\ -0.15\ -1.25\ +1.40\ -0.85\ -0.10\ -0.95\ -1.75\ +0.5\ +1.30)$ be the received sequence. The soft-decision VA [1–6] will be applied to decode this sequence. Then a comparison with a hard-decision decoder is also presented.

The first step in the application of the soft-decision decoding algorithm is to calculate the Euclidean distance between the samples of the received signal and the corresponding outputs for each transition of the trellis:

Message sequence	$\boldsymbol{m} = 10101$
Code sequence	$\boldsymbol{c} = +1 + 1\ -1\,+1\ +1 + 1\ -1\,-1\ +1 + 1$
Received sequence	$\boldsymbol{s}_{\mathbf{r}} = +1.35\ -0.15\ -1.25\ +1.40\ -0.85\ -0.10\ -0.95$ $-1.75\ +0.5\ +1.30$

If a hard-decision decoder operated on the received sequence, the sampled values of this received sequence would be converted by hard-decision detection into a set of values normalized

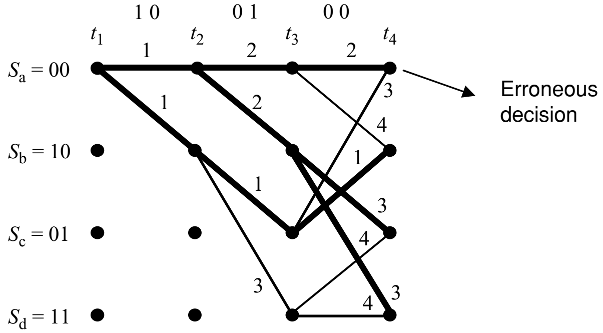


Figure 6.26 Hard-decision decoding of example of Section 16.15.3

to be equal to ±1; that is, the output alphabet is the binary alphabet in polar format, and the detection threshold would be set to zero. Therefore, and after converting the polar format into the classic binary format, a trellis decoder would use the sequence 10 01 00 00 11 as the received sequence, which has three errors with respect to the true transmitted sequence 11 01 11 00 11. The error event in this case is such that the hard-decision Viterbi convolutional decoder fails at time instant t_4, because at that stage of the algorithm, the decoder discards the true sequence, thus providing an incorrect decoded sequence. This can be seen in Figure 6.26.

Even when the algorithm still continues, to finally determine all the estimated bits of the sequence, the decoder has already produced a decoding mistake in the first part of the sequence, so that it will erroneously decode the whole received sequence.

Figure 6.27 is the trellis from Figure 6.13 in which now the transitions are denoted with the corresponding output values in polar format, and input values are omitted. This trellis representation will be useful for implementation of the soft-decision VA.

The squared Euclidean distance is the metric utilized to determine the minimum cumulative distance of a given path. This cumulative distance is simply the sum of all the squared distances involved in that path. The different paths arriving at a given node of the trellis are characterized by the cumulative sum of squared distances for increasing time instants. In general, in a convolutional code of rate $1/n$, there will be 2^n possible outputs for each transition of the trellis,

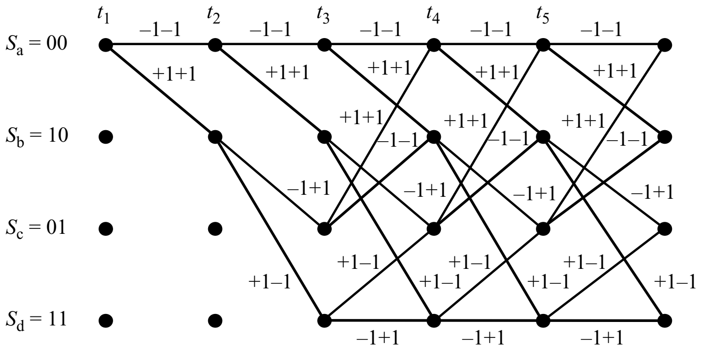


Figure 6.27 Trellis of the convolutional encoder of Figure 6.12 with output values in polar format

which can be described as $\boldsymbol{c}_0, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_{2^n-1}$. These vectors per transition have n components, adopting the form $\boldsymbol{c}_k = \left(c_k^{(0)}, c_k^{(1)}, \ldots, c_k^{(n-1)}\right)$. For the particular example being studied, these vectors are $\boldsymbol{c}_0 = (-1 - 1), \boldsymbol{c}_1 = (-1 + 1), \boldsymbol{c}_2 = (+ 1 - 1), \boldsymbol{c}_3 = (+ 1 + 1)$. The received sequence is also arranged as vectors of n components that are the samples of the received signal, which adopts the form $\boldsymbol{s}_{r(i-1)} = \left(s_{\mathrm{r(i-1)}}^{(0)}, s_{\mathrm{r(i-1)}}^{(1)}, \ldots, s_{\mathrm{r(i-1)}}^{(n-1)}\right)$. The squared values of the distance between the received samples $\boldsymbol{s}_{\mathrm{r}(i-1)} = \left(s_{\mathrm{r(i-1)}}^{(0)}, s_{\mathrm{r(i-1)}}^{(1)}, \ldots, s_{\mathrm{r(i-1)}}^{(n-1)}\right)$ at a given time instant t_i that defines the value for the transition $i - 1$ and some of the k possible output values for that transition are calculated as

$$d_{(i-1)}^2(\boldsymbol{s}_{\mathrm{r}(i-1)}, \boldsymbol{c}_k) = \sum_{j=1}^{n-1} (s_{\mathrm{r(i-1)}}^{(j)} - c_k^{(j)})^2 \tag{63}$$

For a path of U transitions of the trellis, the cumulative squared distance is calculated as

$$d_U^2 = \sum_{v=2}^{U+1} d_{v-1}^2(\boldsymbol{s}_{\mathrm{r}(v-1)}, \boldsymbol{c}_k) \tag{64}$$

where k varies according to the transition for which the cumulative squared distance is calculated.

The following is the soft-decision decoding of the received sequence for the example presented in this section.

As an example of the squared distance calculation, at time instant t_2, the squared distance of the transition from state S_a to the same state S_a is

$$d_1^2[(+1.35, -0.15), (-1, -1)] = (1.35 + 1)^2 + (-0.15 + 1)^2 = 6.245$$

All the remaining squared distances associated with the different transitions of the trellis can be calculated as above, in order to determine, by addition of the corresponding values, the cumulative squared distances at each time instant and state (node of the trellis), as shown in Figures 6.28–6.30.

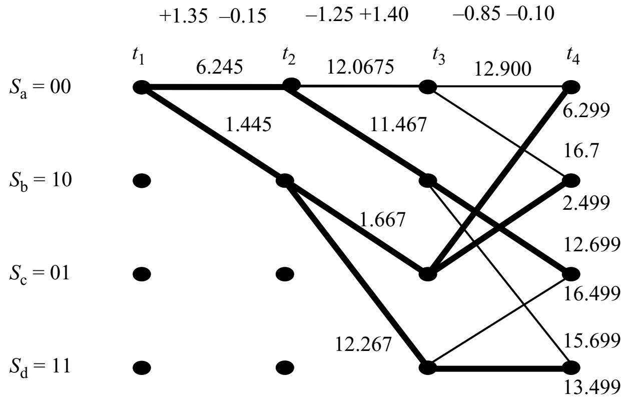


Figure 6.28 Soft-decision decoding to determine the survivor at time instant t_4 on the corresponding trellis

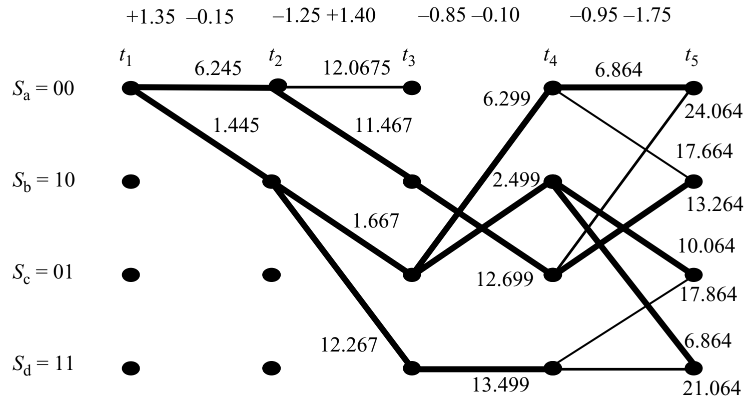


Figure 6.29 Soft-decision decoding to determine the survivor at time instant t_5 on the corresponding trellis

If a decision is taken at time instant t_6, the received sequence is correctly decoded, since the minimum cumulative squared Euclidean distance is seen to be equal to $d^2_{\text{acum\_min}} = 7.204$, for the path that ends at the trellis node defined by the state $S_b = 10$, time instant t_6, as seen in Figure 6.31.

The soft-decision Viterbi decoding of the received sequence of this example shows the advantages of using this type of decoding. Once again, soft decision leads to a better decoding result than that of hard decision. In the former case, the decoder is able to correctly estimate the sequence at instant t_4, and then successfully decode the whole sequence correctly. In comparison, the hard-decision decoder was shown to fail, owing to a wrong decision made at time instant t_4. In addition, the use of the squared Euclidean distance as a measure makes it quite unlikely that two paths arriving at the same node of the trellis would have the same value of cumulative squared distance.

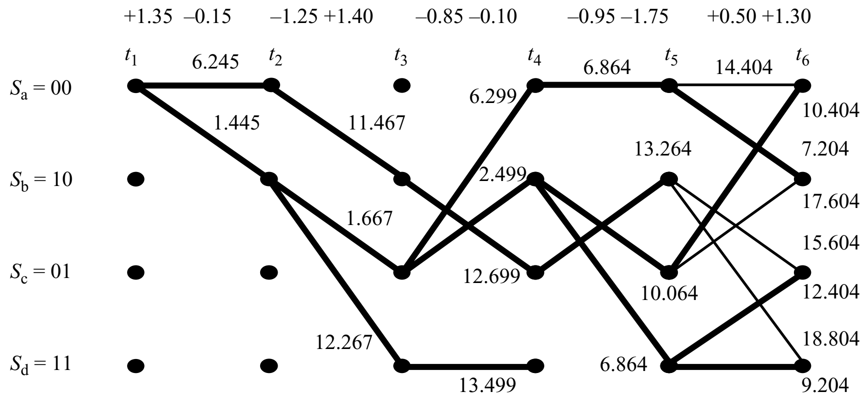


Figure 6.30 Soft-decision decoding to determine the survivor at time instant t_6 on the corresponding trellis

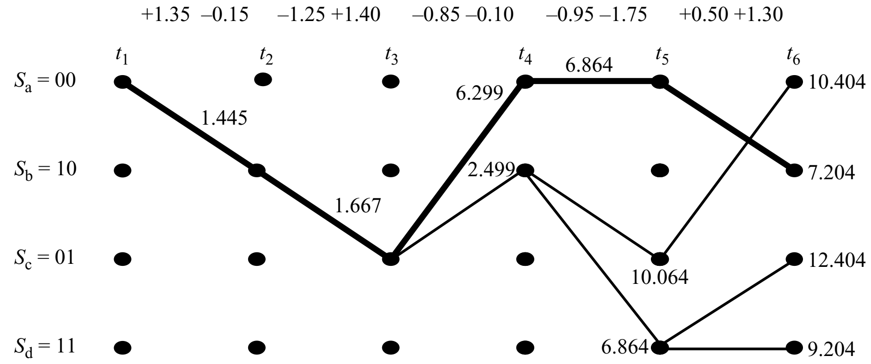


Figure 6.31 Soft-decision decoding to determine the final survivor on the corresponding trellis

In this example hard-decision decoding was seen to produce a burst of decoding errors. This is a characteristic of convolutional decoding in the presence of an error event that is over the error-correction capability of the convolutional code. For this reason, convolutional codes are widely used as the inner code of serially concatenated coding schemes where the outer code is normally a code with high capability of correcting burst errors, like a Reed–Solomon (RS) code. Interleaving is also very commonly used between these two codes, as has been shown in Chapter 5 to be particularly efficient in the case of the cross-interleaved Reed–Solomon code coding scheme for the compact disk. In this way, with the help of the outer code and the interleaver, burst decoding errors produced by a collapsed convolutional decoder can be properly eliminated. This combination of convolutional (inner) codes and RS (outer) codes is found in many practical applications, including digital satellite transmission and deep space communications.

6.16 Punctured Convolutional Codes and Rate-Compatible Schemes

Punctured convolutional codes are convolutional codes obtained by puncturing of some of the outputs of the convolutional encoder. The puncturing rule selects the outputs that are eliminated and not transmitted to the channel. Puncturing increases the rate of a convolutional code, and is a useful design tool because it makes it easy to achieve convolutional codes with relatively high rates. This enhancement of code rate is desirable because low-rate codes are associated with a higher loss of bit rate, or a larger increase in transmission bandwidth, in comparison with uncoded transmission.

This procedure is applied to a base or mother code whose code rate is always smaller than the desired code rate, so that for a given block of k message bits, only a selection of the total number n of coded bits that the base encoder produces are transmitted. This technique was first introduced by Cain, Clark and Geist [30]. In general, the base code is a convolutional code of rate $R_c = 1/2$ which is used to construct punctured convolutional codes of rate $(n - 1)/n$, with $n \geq 3$. The main concept to keep in mind in the construction of a punctured convolutional code is that its trellis should maintain the same state and transition structure of the base code of rate $R_c = 1/2$; that is, in this case, a trellis that looks like that of the Figure 6.6, where there are two transitions emerging from and arriving at each state. This differentiates the trellis of a punctured convolutional code of rate $(n - 1)/n$ from that of a convolutional code of the same rate constructed in the traditional way. In the latter case, the rate of the code determines the encoder structure, by requiring $n - 1$ parallel inputs, so that the resulting trellis will have 2^{n-1} transitions emerging from and arriving at each of its nodes. The punctured code trellis has therefore fewer branches per state than that of the traditional code trellis. Thus the complexity of the trellis is reduced, so that it also reduces the decoding complexity, whether using VA, or the so-called BCJR algorithm described in Chapter 7. Another interesting application of punctured convolutional codes will also be introduced in Chapter 7, involving turbo codes.

The construction of a given punctured convolutional code requires the definition of the puncturing rule; that is, the rule that determines which of the coded outputs are not transmitted to the channel. This puncturing rule can be properly described by means of a matrix, $\boldsymbol{P}_\mathrm{p}$, where a ‘1’ indicates that the corresponding output is transmitted, whereas a ‘0’ indicates that the corresponding output is not transmitted. The puncturing procedure has a period known as the puncturing period T_p, so that the puncturing matrix is of size $2 \times T_\mathrm{p}$ when applied to a ½-rate base code such as the code generated by the encoder of Figure 6.3. In the puncturing matrix $\boldsymbol{P}_\mathrm{p}$, the first row determines the output bits that can be transmitted (‘1’s) or not (‘0’s) generated by the convolutional base encoder output $c^{(1)}$, and the second row determines the output bits that can be transmitted (‘1’s) or not (‘0’s) generated by the convolutional base encoder output $c^{(2)}$. Read in column format, the puncturing matrix $\boldsymbol{P}_\mathrm{p}$ defines for each transition or time instant, if the outputs $c^{(1)}$ and/or $c^{(2)}$ are transmitted or not.

Example 6.10: Construct a punctured convolutional code of rate $R_\mathrm{c} = 2/3$, using as the base code the convolutional code of rate $R_\mathrm{c} = 1/2$ whose encoder is shown in Figure 6.3. Figure 6.32 shows this punctured convolutional code of rate $R_\mathrm{c} = 2/3$.

The construction of a punctured convolutional code of rate $R_\mathrm{c} = 2/3$ is done by taking into account the fact that the base encoder generates four output bits for every pair of input bits, and so the punctured encoder should transmit only three of these four outputs to obtain the desired code rate $R_\mathrm{c} = 2/3$. This obviously requires the elimination of one output every four outputs, making the corresponding puncturing period be equal to $T_\mathrm{p} = 2$. A puncturing matrix for this case would be of the form

$$\boldsymbol{P}_\mathrm{p} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

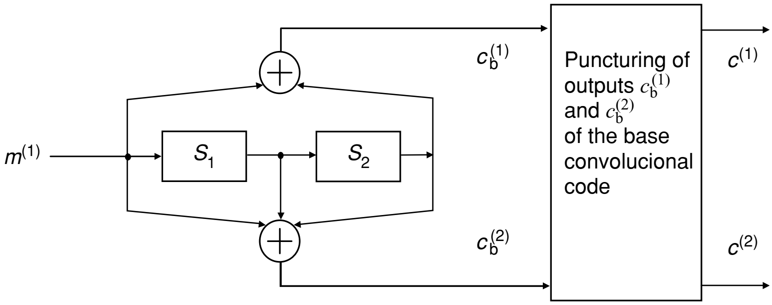

$m^{(1)}$
S_1
S_2
$c_\mathrm{b}^{(1)}$
$c_\mathrm{b}^{(2)}$
Puncturing of outputs $c_\mathrm{b}^{(1)}$ and $c_\mathrm{b}^{(2)}$ of the base convolucional code
$c^{(1)}$
$c^{(2)}$

Figure 6.32 Punctured convolutional encoder of rate $R_\mathrm{c} = 2/3$ based on a convolutional code of rate $R_\mathrm{c} = 1/2$

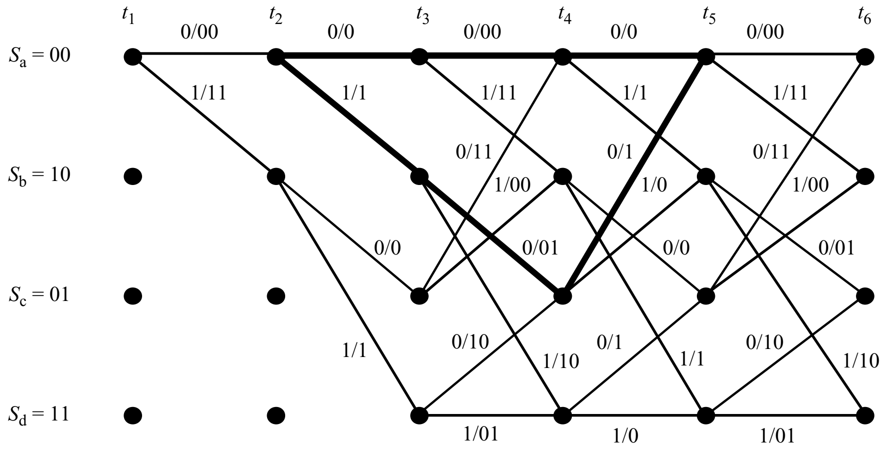


Figure 6.33 Trellis for a punctured convolutional code of rate $R_\mathrm{c} = 2/3$ based on a convolutional code of rate $R_\mathrm{c} = 1/2$

As read in column format, the above matrix indicates that in the first transition or time instant both outputs $c^{(1)}$ and $c^{(2)}$ of the base convolutional encoder are transmitted, and in the following time instant only $c^{(1)}$ is transmitted. In this way two input bits generate three output bits, and the code rate is the desired rate $R_\mathrm{c} = 2/3$. The trellis of this punctured convolutional encoder, constructed from the base encoder as seen in Figure 6.3, has the trellis as shown in Figure 6.33, where in even time index transitions output $c^{(2)}$ is not transmitted.

After the puncturing procedure, the base code of rate $R_\mathrm{c} = 1/2$ and minimum Hamming free distance $d_\mathrm{f} = 5$ converts into a punctured code of rate $R_\mathrm{c} = 2/3$ and minimum Hamming free distance $d_\mathrm{f} = 3$. One path with the minimum Hamming weight of 3 is shown in bold in Figure 6.33. This minimum free distance reduction is a logical consequence of the puncturing procedure, but in this case, however, the punctured convolutional code has the maximum available value of that parameter for convolutional codes of that rate. This is, in general, not true for punctured convolutional codes of rate $(n-1)/n$. From this point of view, the punctured convolutional code of rate $R_\mathrm{c} = 2/3$ has the same properties as the traditionally constructed convolutional code of the same rate, but with the advantage of a lower decoding complexity.

Figure 6.34 shows the first two stages of the trellis of a convolutional code of rate $R_\mathrm{c} = 2/3$ constructed in the traditional way, which corresponds to the convolutional encoder as seen in Figure 6.4. It can be seen that the structural complexity of this trellis is higher than that of the trellis in Figure 6.33. This complexity arises not only due to the four branches emerging from and arriving at each node of the trellis, but also due to the larger number of bits per transition, so that both the number and the length of decoding distance calculations are increased.

Puncturing of convolutional codes emerges as a nice tool for the design of convolutional codes of a desired code rate, since this procedure only requires the use of a fixed base convolutional code of rate $R_\mathrm{c} = 1/2$, together with a suitable puncturing matrix $\boldsymbol{P}_\mathrm{p}$. For a given base convolutional code of rate $R_\mathrm{c} = 1/2$, and a given puncturing matrix of size $2 \times T_\mathrm{p}$, changes in the element values in the matrix $\boldsymbol{P}_\mathrm{p}$ generate a family of punctured codes with different code rates. These code rates are in general larger than the initial code rate of the base code, and

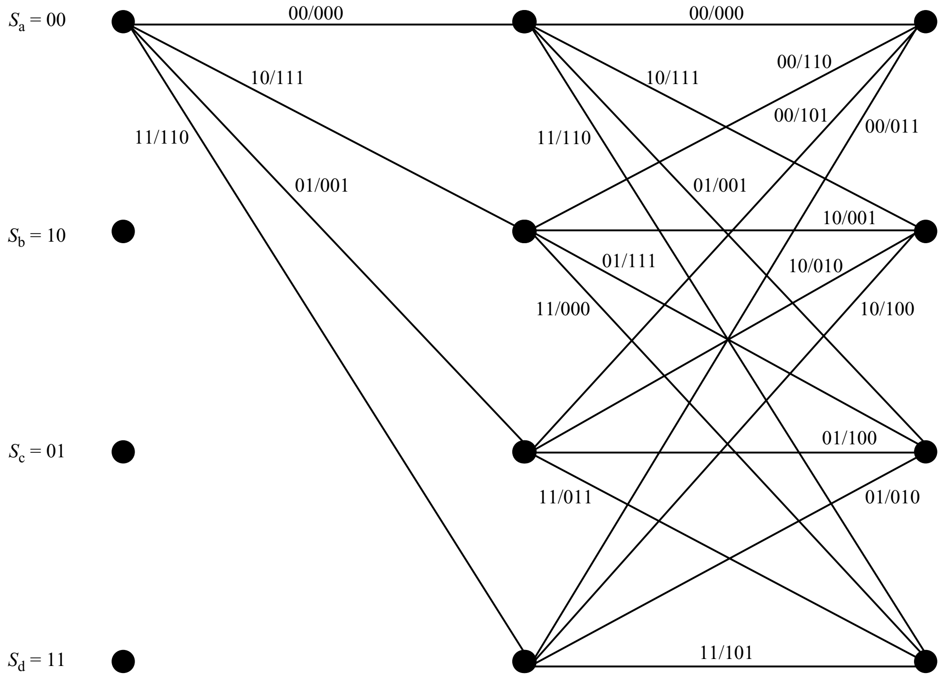


Figure 6.34 Trellis for a convolutional code of rate $R_c = 2/3$ constructed in the traditional way

these different punctured convolutional codes are obtained by simply modifying the puncturing matrix $\boldsymbol{P}_p$. On the other hand, both the punctured encoder and decoder are based on the trellis structure of the base convolutional code, so that they can adaptively operate in the family of punctured convolutional codes by just knowing the changes in the puncturing matrix $\boldsymbol{P}_p$. This is called a set of rate-compatible punctured convolutional (RCPC) codes.

RCPC codes are very useful in automatic repeat request (ARQ) and hybrid-ARQ schemes in order to optimize the data rate and BER performance as a function of the signal-to-noise ratio of the channel. Thus, when the signal-to-noise ratio is high, high-rate codes (i.e., medium error-control capability codes) are used, and when the signal-to-noise ratio in the channel is low, relatively small-rate codes (i.e., high error-control capability codes) are then utilized. The relative absence or presence of repeat transmission requests can be used to raise or lower the rate of the codes in the RCPC set, usually starting from a high rate. Other system requirements or status information can also be used to control the RCPC scheme, such as the quality-of-service requirements of a sender, or channel state information in a multiuser wireless system.

Bibliography and References

[1] Lin, S. and Costello, D. J., Jr., *Error Control Coding: Fundamentals and Applications*, Prentice Hall, Englewood Cliffs, New Jersey, 1983.

[2] Sklar, B., *Digital Communications, Fundamentals and Applications*, Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[3] Viterbi, A. J., "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inf. Theory*, vol. IT-13, pp. 260–269, April 1967.

[4] Viterbi, A. J. and Omura, J. K., *Principles of Digital Communication and Coding*, McGraw-Hill, New York, 1979.

[5] Carlson, B., *Communication Systems: An Introduction to Signals and Noise in Electrical Communication*, 3rd Edition, McGraw-Hill, New York, 1986.

[6] Proakis, J. G. and Salehi, M., *Communication Systems Engineering*, Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[7] Heegard, C. and Wicker, S., *Turbo Coding*, Kluwer, Massachusetts, 1999.

[8] Massey, J. L. and Mittelholzer, T., "Codes over rings – practical necessity," *AAECC Symposium*, Toulouse, France, June 1989.

[9] Massey, J. L. and Mittelholzer, T., "Convolutional codes over rings," *Proc. Fourth Joint Swedish–Soviet International Workshop Inf. Theory*, Gottland, Sweden, August 27–September 1, 1989.

[10] Baldini, R., *Coded Modulation Based on Ring of Integers*, PhD Thesis, University of Manchester, Manchester, 1992.

[11] Baldini, R. and Farrell, P. G., "Coded modulation based on ring of integers modulo-q. Part 2: Convolutional codes," *IEE Proc. Commun.*, vol. 141, no. 3, pp. 137–142, June 1994.

[12] Ahmadian-Attari, M., *Efficient Ring-TCM Coding Schemes for Fading Channels,* PhD Thesis, University of Manchester, 1997.

[13] Lopez, F. J., *Optimal Design and Application of Trellis Coded Modulation Techniques Defined over the Ring of Integers*, PhD Thesis, Staffordshire University, Stafford, 1994.

[14] Ahmadian-Attari, M. and Farrell, P. G., "Multidimensional ring-TCM codes for fading channels," *IMA Conf. Cryptography & Coding*, Cirencester, vol. 18–20, pp. 158–168, December 1995.

[15] Castiñeira Moreira, J., *Signal Space Coding over Rings*, PhD Thesis, Lancaster University, Lancaster, 2000.

[16] Massey, J. L. and Sain, M. K., "Inverse of linear sequential circuits," *IEEE Trans. Comput.,* vol. C17, pp. 330–337, April 1988.

[17] Forney, G. D., Jr., "Geometrically uniform codes," *IEEE Trans. Inf. Theory*, vol. 37, no. 5, pp. 1241–1260, September 1991.

[18] Forney, G. D., Jr., "Coset codes. Part I: Introduction and geometrical classification," *IEEE Trans. Inf. Theory,* vol. 34, no. 5, pp. 1123–1151, September 1988.

[19] Forney, G. D., Jr. and Wei, L.-F., "Multidimensional constellations. Part I: Introduction, figures of merit, and generalised cross constellations," *IEEE Select. Areas Commun.,* vol. 7, no. 6, pp. 877–892, August 1989.

[20] Forney, G. D., Jr. and Wei, L.-F., "Multidimensional constellations. Part II: Voronoi constellations," *IEEE Select. Areas Commun.,* vol. 7, no. 6, pp. 941–956, August 1989.

[21] Ungerboeck, G., "Channel coding with multilevel/phase signals," *IEEE Trans. Inf. Theory,* vol. IT-28, pp. 56–67, January 1982.

[22] Divsalar, D. and Yuen, J. H., "Asymmetric MPSK for trellis codes," *GLOBECOM'84*, Atlanta, Georgia, pp. 20.6.1–20.6.8, November 26–29, 1984.

[23] Benedetto, S., Garello, R., Mondin, M. and Montorsi, G., "Geometrically uniform partitions of LxMPSK constellations and related binary trellis codes," *IEEE Trans. Inf. Theory,* vol. 42, no. 2–4, pp. 1995–1607, April 1994.

[24] Forney, G. D., Jr., "Coset codes. Part II: Binary lattices and related codes," *IEEE Trans. Inf. Theory,* vol. 34, no. 5, pp. 1152–1187, September 1988.

[25] Castiñeira Moreira, J., Edwards, R., Honary, B. and Farrell, P. G., "Design of ring-TCM schemes of rate *m*/*n* over *N*-dimensional constellations," *IEE Proc. Commun.*, vol. 146, pp. 283–290, October 1999.

[26] Benedetto, S., Garello, R. and Mondin, M., "Geometrically uniform TCM codes over groups based on LxMPSK constellations," *IEEE Trans. Inf. Theory*, vol. 40, no. 1, pp. 137–152, January 1994.

[27] Biglieri, E., Divsalar, D., McLane, P. J. and Simon, M. K., *Introduction to Trellis-Coded Modulation with Applications*, McMillan, New York, 1991.

[28] Viterbi, A., "Convolutional codes and their performance in communication systems," *IEEE Trans. Commun. Technol.*, vol. COM-19, no. 5, pp. 751–772, October 1971.

[29] Omura, J. K., "On the Viterbi decoding algorithm," *IEEE Trans. Inf. Theory*, vol. IT-15, pp. 177–179, January 1969.

[30] Cain, J. B., Clark, G. C. and Geist, J. M., "Punctured convolutional codes of rate (*n*-1)/*n* and simplified maximum likelihood decoding," *IEEE Trans. Inf. Theory*, vol. IT-25, pp. 97–100, January 1979.

[31] Honary B. and Markarian G., *Trellis Decoding of Block Codes: A Practical Approach*, Kluwer, Massachusetts, 1997.

Problems

6.1 (a) Determine the state and trellis diagram for a convolutional code with $K = 2$, code rate $R_c = 1/3$ and generator sequences given by the following polynomials:

$$g^{(1)}(D) = D + D^2, \quad g^{(2)}(D) = 1 + D \; and \; g^{(3)}(D) = 1 + D + D^2.$$

(b) What is the minimum free distance of the code?
(c) Give an example to show that this code can correct double errors.
(d) Is this code catastrophic?

6.2 A binary convolutional error-correcting code has $k = 1$, $n = 3$, $K = 2$, $g^{(1)}(D) = 1 + D^2$, $g^{(2)}(D) = D$ and $g^{(3)}(D) = D + D^2$.
(a) Draw the encoder circuit and its trellis diagram, and calculate the free distance of the code.
(b) Is the code systematic or non-systematic?

6.3 (a) For the convolutional encoder of the Figure P.6.1,
determine the generator polynomials of the encoder.
(b) Is this a catastrophic code? Justify the answer.
(c) Determine the coded output for the input message $\boldsymbol{m} = (101)$.

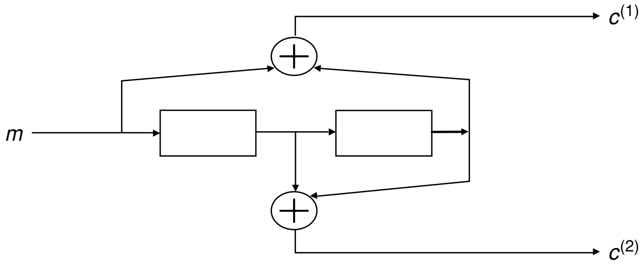


Figure P.6.1 Convolutional encoder, Problem 6.3

6.4 Draw the trellis diagram of the binary convolutional encoder given in Figure P.6.2, for which $R_c = 1/3$.

(a) What is the constraint length and the minimum free distance of the code generated by this encoder?

(b) Draw the path through the extended trellis diagram corresponding to the input sequence $\boldsymbol{m} = (1110100)$, starting from the all-zero state, and thus determine the output sequence.

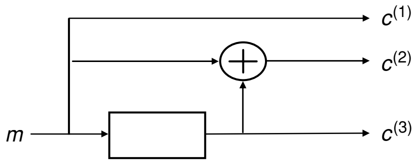


Figure P.6.2 Convolutional encoder, Problem 6.4

6.5 (a) Draw the trellis diagram of the binary convolutional code generated by the encoder of Figure P.6.3, and determine its minimum free distance.

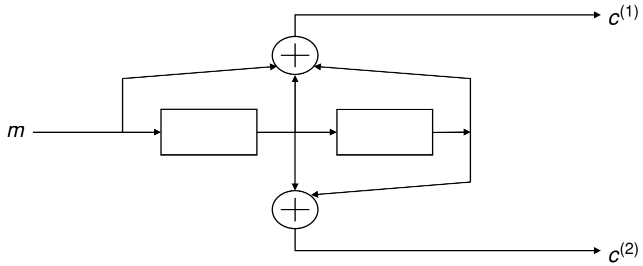


Figure P.6.3 Convolutional encoder, Problem 6.5

(b) Obtain the impulse response of the encoder, and its relationship with item (a).

(c) Confirm the minimum free distance of this code using the generating function approach.
(d) What is the node error probability for this code on a BSC with $p = 10^{-3}$?

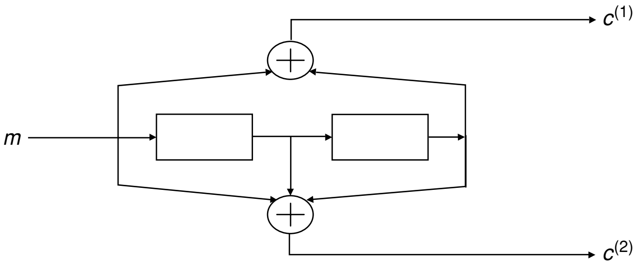


Figure P.6.4 Convolutional encoder, Problem 6.6

6.6 The received sequence $\boldsymbol{s}_\mathrm{r} = (01\ 11\ 00\ 01\ 11\ 00\ 00\ldots)$ is applied to the input of a decoder for the binary convolutional code generated by the encoder of Figure P.6.4.
(a) Determine the corresponding input sequence by using the Viterbi decoding algorithm, assuming that the encoder starts in the all-zero state.

6.7 The trellis diagram of a binary error-correcting convolutional encoder is shown in Figure P.6.5.

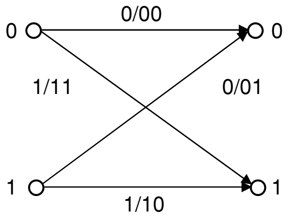


Figure P.6.5 Trellis diagram, Problem 6.7

A sequence output by the encoder is transmitted over a channel subject to random errors, and is received as the sequence

$$\boldsymbol{s}_\mathrm{r} = (11\quad 10\quad 11\quad 00\quad 11\ldots)$$

(a) Using the Viterbi decoding algorithm, find the sequence most likely to have been transmitted, and hence determine the positions of any errors which may have occurred during transmission.

(b) A sequence from the encoder of item (a) is transmitted over an AWGN channel and is received as the sequence

$$\mathbf{s}_{\mathrm{r}} = (33 \quad 10 \quad 23 \quad 00 \quad 33\ldots)$$

after soft-decision detection with four levels. Find the most likely error pattern in the received sequence.

6.8 The convolutional encoder of IIR type and code rate $R_{\mathrm{c}} = 1/2$, as seen in Figure P.6.6, operates with coefficients over the binary field GF(2). Determine the transfer function and the state transfer function matrices of this code.

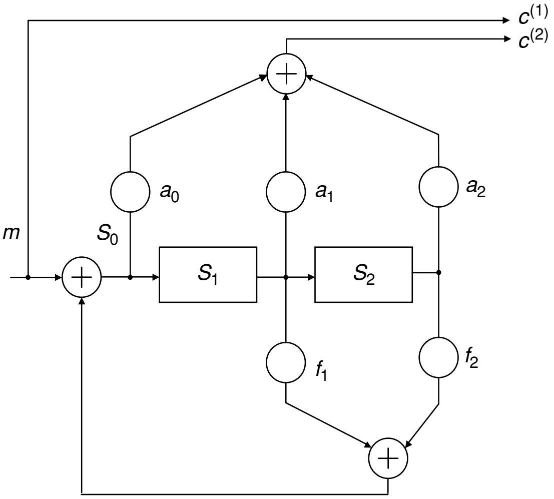


Figure P.6.6 Convolutional encoder, Problem 6.8

6.9 A binary convolutional error-correcting code has $k = 1$, $n = 2$, $K = 2$, $g^{(1)}(D) = 1 + D + D^2$ and $g^{(2)}(D) = D + D^2$.

(a) Draw the encoder circuit and its trellis diagram, and determine the rate and the free distance of the code.

(b) Is the code systematic or non-systematic?

6.10 An information sequence encoded using the encoder of Problem 6.9 is transmitted through a channel subject to random errors and received as the sequence.

$$\mathbf{s}_{\mathrm{r}} = (10 \quad 01 \quad 00 \quad 01 \quad 11 \quad 11 \quad 10)$$

What is the information sequence?

■

7

Turbo Codes

Berrou, Glavieux and Thitimajshima [1] introduced in 1993 a novel and apparently revolutionary error-control coding technique, which they called turbo coding. This coding technique consists essentially of a parallel concatenation of two binary convolutional codes, decoded by an iterative decoding algorithm. These codes obtain an excellent bit error rate (BER) performance by making use of three main components. They are constructed using two systematic convolutional encoders that are IIR FSSMs, usually known as recursive systematic convolutional (RSC) encoders, which are concatenated in parallel. In this parallel concatenation, a random interleaver plays a very important role as the randomizing constituent part of the coding technique. This coding scheme is decoded by means of an iterative decoder that makes the resulting BER performance be close to the Shannon limit.

In the original structure of a turbo code, two recursive convolutional encoders are arranged in parallel concatenation, so that each input element is encoded twice, but the input to the second encoder passes first through a random interleaver [2, 3]. This interleaving procedure is designed to make the encoder output sequences be statistically independent from each other. The systematic encoders are binary FSSMs of IIR type, as introduced in Chapter 6, and usually have code rate $R_\mathrm{c} = 1/2$. As a result of the systematic form of the coding scheme, and the double encoding of each input bit, the resulting code rate is $R_\mathrm{c} = 1/3$. In order to improve the rate, another useful technique normally included in a turbo coding scheme is puncturing of the convolutional encoder outputs, as introduced in Chapter 6.

The decoding algorithm for the turbo coding scheme involves the corresponding decoders of the two convolutional codes iteratively exchanging soft-decision information, so that the information can be passed from one decoder to the other. The decoders operate in a soft-input–soft-output mode; that is, both the input applied to each decoder, and the resulting output generated by the decoder, should be soft decisions or estimates [3]. Both decoders operate by utilizing what is called *a priori* information, and together with the channel information provided by the samples of the received sequence, and information about the structure of the code, they produce an estimate of the message bits. They are also able to produce an estimate called the extrinsic information, which is passed to the other decoder, information that in the following iteration will be used as the *a priori* information of the other decoder. Thus the first decoder generates extrinsic information that is taken by the second decoder as its *a priori*

Essentials of Error-Control Coding Jorge Castiñeira Moreira and Patrick Guy Farrell

information. This procedure is repeated in the second decoder, which by using the *a priori* information, the channel information and the code information generates again an estimation of the message information, and also an extrinsic information that is now passed to the first decoder. The first decoder then takes the received information as its *a priori* information for the new iteration, and operates in the same way as described above, and so on.

The iterative passing of information between the first and the second decoders continues until a given number of iterations is reached. With each iteration the estimates of the message bits improve, and they usually converge to a correct estimate of the message. The number of errors corrected increases as the number of iterations increases. However, the improvement of the estimates does not increase linearly, and so, in practice, it is enough to utilize a reasonable small number of iterations to achieve acceptable performance.

One of the most suitable decoding algorithms that performs soft-input–soft-output decisions is a maximum *a posteriori* (MAP) algorithm known as the BCJR (Bahl, Cocke, Jelinek, Raviv, 1974) algorithm [4]. Further optimizations of this algorithm lead to lower complexity algorithms, like SOVA (soft-output Viterbi algorithm), and the LOG MAP algorithm, which is basically the BCJR algorithm with logarithmic computation [2].

7.1 A Turbo Encoder

A turbo encoder constructed using two RSC encoders arranged in parallel, and combined with a random interleaver, together with a multiplexing and puncturing block, is seen in Figure 7.1.

In the traditional structure of a turbo encoder, the encoders E_1 and E_2 are usually RSC encoders of rate $R_c = 1/2$, such that $c'_1 = c_1$, $c'_2 = c_2$ and the lengths of the sequences $\boldsymbol{m}$, $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$, and $\boldsymbol{c}'_1$ and $\boldsymbol{c}'_2$ are all the same. Then the overall turbo code rate is $R_c = 1/3$. Puncturing [2, 6] is a technique very commonly used to improve the overall rate of the code. The puncturing selection process is performed by periodically eliminating one or more of the outputs generated by the constituent RSC encoders. Thus, for instance, the parity bits generated by these two encoders can be alternately eliminated so that the redundant bit of the first encoder is first transmitted, eliminating that of the second decoder, and in the following time instant the redundant bit of the second encoder is transmitted, eliminating that of the first. In this way,

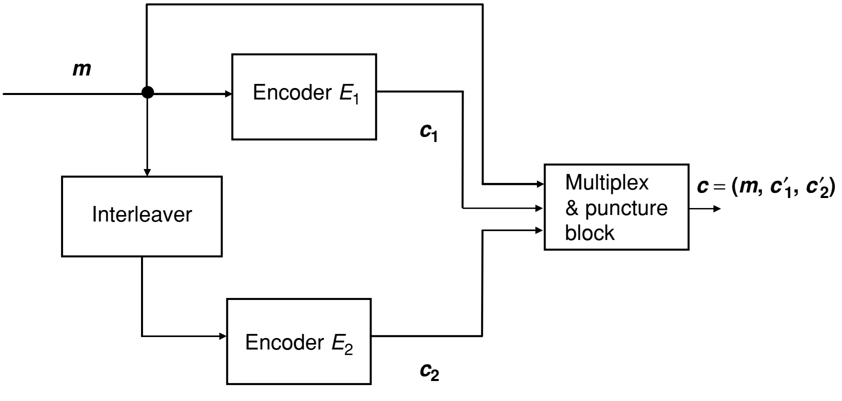

m
Encoder E_1
c_1
Interleaver
Multiplex & puncture block
$\boldsymbol{c} = (\boldsymbol{m}, \boldsymbol{c}'_1, \boldsymbol{c}'_2)$
Encoder E_2
c_2

Figure 7.1 A turbo encoder

the lengths of $\boldsymbol{c}_1'$ and $\boldsymbol{c}_2'$ are half the lengths of $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$, respectively, and the resulting overall rate becomes $R_c = 1/2$. Puncturing is not usually applied to the message (systematic) bits, because this causes a BER performance loss.

There are two important components of a turbo encoder whose parameters have a major influence on the BER performance of a turbo code: the first is the interleaver, especially its length and structure, and the second is the use of RSC IIR FSSMs as constituent encoders [2, 3]. The excellent BER performance of these codes is enhanced when the length of the interleaver is significantly large, but also important is its pseudo-random nature. The interleaving block, and its corresponding de-interleaver in the decoder, does not much increase the complexity of a turbo scheme, but it does introduce a significant delay in the system, which in some cases can be a strong drawback, depending on the application. The RSC-generated convolutional codes are comparatively simple, but offer excellent performance when iteratively decoded using soft-input–soft-output algorithms.

7.2 Decoding of Turbo Codes

7.2.1 The Turbo Decoder

Turbo codes are so named because of their iterative soft-decision decoding process, which enables the combination of relatively simple RSC codes to achieve near-optimum performance. Turbo decoding involves iterative exchange between the constituent decoders of progressively better estimates of the message bits, in a decoding procedure that is helped by the statistical independence of the two code sequences generated by each input bit. The turbo decoder is shown in Figure 7.2.

In the decoding procedure, each decoder takes into account the information provided by the samples of the channel, which correspond to the systematic (message) and parity bits,

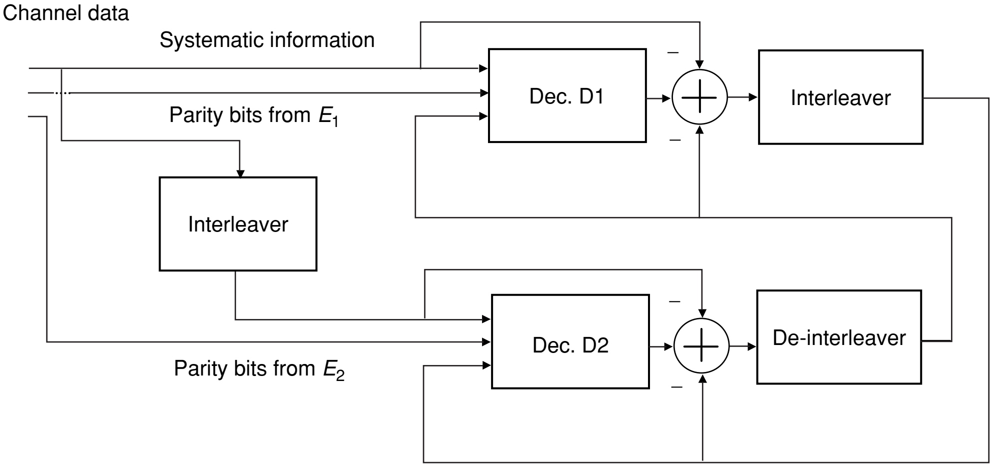


Figure 7.2 A turbo decoder

together with the *a priori* information that was provided by the other decoder, which was calculated as its extrinsic information in the previous iteration. However, instead of making a hard decision on the estimated message bits, as done for instance in the traditional decoding of convolutional codes using the Viterbi algorithm, the decoder produces a soft-decision estimate of each message bit. This soft-decision information is an estimate of the corresponding bit being a '1' or a '0'; that is, it is a measure of the probability that the decoded bit is a '1' or a '0'. This information is more conveniently evaluated in logarithmic form, by using what is known as a log likelihood ratio (LLR), to be defined below. This measure is very suitable because it is a signed number, and its sign directly indicates whether the bit being estimated is a '1' (positive sign) or a '0' (negative sign), whereas its magnitude gives a quantitative measure of the probability that the decoded bit is a '1' or a '0'. There are many algorithms that operate using LLRs, and perform decoding using soft-input–soft- output values. One of these algorithms is the BCJR algorithm [4]. Some background on the measures and probabilities involved in this algorithm is presented next, in order to then introduce the BCJR algorithm [3].

7.2.2 Probabilities and Estimates

The probability distribution is a useful description of a discrete random variable X whose values are taken from a discrete alphabet of symbols A_X. The distribution (or histogram when plotted graphically) is a function that assigns to each value of the variable the probability of occurrence of that value. In the case of continuous random variables, the histogram becomes the so-called probability density function. A probability distribution for a discrete random variable is of the form

$$P(X = x) = p(x) \geq 0 \quad \text{and} \quad \sum_{x \in A_X} p(x) = 1 \tag{1}$$

where a non-negative number $p(x)$ is assigned to each value of the random variable $x \in A_X$. An equivalent quantity often utilized in decoding algorithms is the probability measure or metric $\mu(x)$ of the event $x \in A_X$. A measure or estimate of an event $x \in A_X$ of a discrete random variable is a generalization of the probability distribution, where the restriction that the sum over all the probabilities in the distribution does not necessarily have to be equal to 1. Then the relationship between measures $\mu(x)$ and probabilities $p(x)$ is given by

$$p(x) = \frac{\mu(x)}{\sum_{x \in A} \mu(x)} \tag{2}$$

Measures have properties similar to those of probabilities. The marginal measure of an event $x \in A_X$, conditioned on a random variable Y, is obtained by summing over all the events of the associated random variable Y:

$$\mu(x) = \sum_{y \in A_Y} \mu(x, y) \tag{3}$$

where $\mu(x, y)$ is the joint measure for a pair of random variables X and Y. The Bayes rule is also applicable to joint measures:

$$\mu(x, y) = \mu(y/x)\mu(x) \tag{4}$$

It is usually more convenient to convert products into sums by using measures in logarithmic form. In this case the measure is called a metric of a given variable:

$$L(x) = -\ln(\mu(x)) \tag{5}$$

$$\mu(x) = e^{-L(x)} \tag{6}$$

It is also true that

$$L(\mu(x) + \mu(y)) = -\ln\left[e^{-L(x)} + e^{-L(y)}\right] \tag{7}$$

and

$$L(\mu(x)\mu(y)) = L(x) + L(y) \tag{8}$$

7.2.3 Symbol Detection

Symbol detection is performed in receivers and decoders, and it is applied in order to determine the value of a given discrete random variable X by observing events of another random variable Y, which is related to the variable X. Thus, for instance, a given discrete random variable X that takes values from the range of a discrete alphabet $A_X = \{0, 1\}$ can be the input of a binary symmetric channel with transition or error probability p, and the corresponding output of this channel can be the discrete random variable Y. If in this channel the output discrete random variable Y takes values from the discrete alphabet $A_Y = \{0, 1\}$, it is said that the channel is a hard-decision channel.

Another model is the Gaussian channel, where a given discrete random variable X with values taken from the discrete polar format alphabet $A_X = \{+1, -1\}$ generates a continuous random variable Y, which after being affected by white and Gaussian noise takes values from the set of real numbers $A_Y = \Re$. In general terms, a hard decision over X is said to occur when after observing the variable Y, the decoder or receiver makes a firm decision that selects one of the two possible values of X, $A_X = \{+1, -1\}$, a decision that will be denoted as $\hat{x}$. This is the case of a decoder or receiver that takes samples of the received signal and compares these samples with a voltage threshold in order to decide for a '1' or a '0' depending on this comparison.

A soft decision of X is said to happen when after observing the variable Y the decoder or receiver assigns a measure, metric or estimate $\mu(x)$ of X based on the observed value of Y. There are many ways of assigning a measure $\mu(x)$ of X, that is, to make a soft decision of X, but the more significant ones are those provided by the maximum likelihood (ML) and the maximum a posteriori (MAP) decoding methods.

In the ML decoding method, the soft decision of X based on the event of the variable $y \in A_Y$ is given by the following conditional distribution function [3]:

$$\mu_{\mathrm{ML}}(x) = p(y/x) = \frac{p(x, y)}{p(x)} \tag{9}$$

In the MAP decoding method, the soft decision of X based on the event of the variable $y \in A_Y$ is given by the following conditional probability distribution function:

$$\mu_{\mathrm{MAP}}(x) = p(x/y) = \frac{p(x, y)}{p(y)} \tag{10}$$

The ML measure is not a probability distribution function because the normalizing condition, the sum over the alphabet of X, A_X, should be equal to 1, is not obeyed in this case. However, it is obeyed in the case of the MAP estimation, which is indeed a probability density function.

The MAP measure is proportional to the joint probability $p(x, y)$. Since

$$\mu_{\mathrm{MAP}}(x) \propto p(x, y) \tag{11}$$

then

$$\mu_{\mathrm{MAP}}(x) \propto \mu_{\mathrm{ML}}(x) p(x) \tag{12}$$

7.2.4 The Log Likelihood Ratio

The LLR [2, 5] is the most common information measure or metric used in iterative decoding algorithms, like the LOG MAP BCJR algorithm to be described below, and it is usually the measure or the extrinsic estimate that each decoder communicates to the other in a turbo decoder.

The LLR for a bit b_i is denoted as $L(b_i)$, and it is defined as the natural logarithm of the quotient between the probabilities that the bit is equal to '1' or '0'. Since this is a signed number, this sign can be directly considered as representative of the symbol which is being estimated, and so it is more convenient to define it as the quotient of the probability that the bit is equal to $+1$ or -1, using the polar format. This is the same as saying that the decision is taken over the transmitted signal alphabet in the range $\{\pm 1\}$, rather than over the binary information alphabet $\{0, 1\}$. This estimate is then defined as

$$L(b_i) = \ln\left(\frac{P(b_i = +1)}{P(b_i = -1)}\right) \tag{13}$$

This definition will be found more convenient in the description of the decoding algorithms, where the sign of the LLR is directly used as the hard decision of the estimate, and its value is utilized as the magnitude of the reliability of the estimate. Thus, a soft decision can be understood as a weighted hard decision.

Figure 7.3 shows the LLR as a function of the bit probability of the symbol $+1$, which is positive if $P(b_i = +1) > 0.5$ (symbol '1' is more likely than symbol '0'), and it is negative if $P(b_i = +1) < 0.5$ (symbol '0' is more likely than symbol '1'). The magnitude of this amount is a measure of the probability that the estimated bit adopts one of these two values.

From (13), and as $P(b_i = +1) = 1 - P(b_i = -1)$, then [5]

$$e^{L(b_i)} = \frac{P(b_i = +1)}{1 - P(b_i = +1)} \tag{14}$$

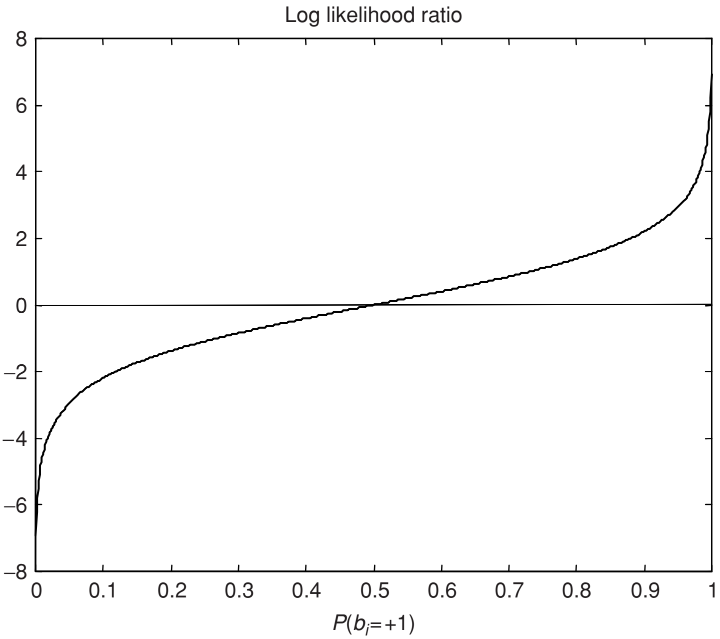


Figure 7.3 LLR as a function of the bit probability of the symbol +1

and also

$$P(b_i = +1) = \frac{e^{L(b_i)}}{1 + e^{L(b_i)}} = \frac{1}{1 + e^{-L(b_i)}} \tag{15}$$

$$P(b_i = -1) = \frac{e^{-L(b_i)}}{1 + e^{-L(b_i)}} = \frac{1}{1 + e^{+L(b_i)}} \tag{16}$$

Expressions (15) and (16) can be summarized as follows:

$$P(b_i = \pm 1) = \frac{e^{-L(b_i)/2}}{1 + e^{-L(b_i)}} e^{\pm(L(b_i)/2)} = \frac{e^{-L(b_i)/2}}{1 + e^{-L(b_i)}} e^{(b_i L(b_i)/2)} \tag{17}$$

since the bit $b_i = \pm 1$.

7.3 Markov Sources and Discrete Channels

As seen in Chapter 1, a discrete channel can be described by its corresponding transition probability matrix. The most commonly used model of a channel is the memoryless channel, where a given input sequence $X_1, X_2, X_3, \ldots$ of statistically independent values taken from a discrete alphabet generates an output sequence $Y_1, Y_2, Y_3, \ldots$ of values that are also taken from a statistically independent discrete alphabet. Under this assumption, the output conditional probability is obtained as the product of the transition probabilities of the channel:

$$p(\boldsymbol{Y}/\boldsymbol{X}) = \prod_{j=1}^{n} R_j(Y_j/X_j) \tag{18}$$

where $R_j(Y_j/X_j)$ is the channel transition probability $p(y_j/x_j)$ for the transmitted symbol x_j. In this notation, x_j represents the value of the signal at instant j, X_j is a random variable that represents x_j and takes values from the discrete alphabet A_x, and $\boldsymbol{X} = X_1^n = \{X_1, X_2, \ldots, X_n\}$ is a vector or sequence of random variables. The difference between X_j and x_j is that the former is the random variable, and the latter is a particular value of that random variable.

Since the channel is stationary, transition probabilities are time invariant. In general, the data source is characterized by a uniformly distributed probability distribution function, which is called the source marginal distribution function $p(X_j)$ of the system.

When the source output is a sequence of independent discrete data, then the source is considered to be a discrete memoryless source. However, this is not the most suitable model for the encoded output sequence generated by a trellis encoder, for instance, since the output symbols are related and the sequence contains some degree of memory. The behaviour of this sort of encoded sequences is more appropriately described by the model of the so-called discrete hidden Markov source.

A sequence input to a channel is considered to be a discrete hidden Markov source if its elements are selected from a discrete alphabet, and the joint probabilities of the symbol sequences are of the form

$$p(\boldsymbol{X}) = p(X_1) \prod_{j=2}^{n} Q_j(X_j/X_{j-1}) \tag{19}$$

where the source transition probabilities are

$$Q(X_j/X_{j-1}) = \frac{p(X_{j-1}, X_j)}{p(X_{j-1})} \tag{20}$$

These are the probabilities that describe the degree of dependence among the symbols generated by a given discrete hidden Markov source. One important property of a discrete hidden Markov source is that the corresponding probability distribution functions are such that

$$p(X_j) = \sum_{x_{j-1} \in A_{X_{j-1}}} p(X_{j-1}) Q_j(X_j/X_{j-1}) \tag{21}$$

The most relevant characteristic of a sequence generated by a discrete hidden Markov source is that, at any time instant j, knowledge of the symbol X_j makes the past and future sequences $X_1^{J-1} = \{X_1, X_2, \ldots, X_{j-1}\}$ and $X_{j+1}^n = \{X_{j+1}, X_{j+2}, \ldots, X_n\}$ be independent from each other. This allows us to do a decomposition of the sequence generated by a discrete hidden Markov source into three subsequences, and accordingly to write the corresponding probability distribution function as the product of three factors [1, 3]:

$$p(\boldsymbol{X}) = p\left(X_1^{j-1}, X_j, X_{j+1}^n\right) = p\left(X_1^{j-1}/X_j\right) p(X_j) p\left(X_{j+1}^n/X_j\right) \tag{22}$$

This expression identifies the past, present and future sequences, and their statistical independence, a fact resulting from the use of the discrete hidden Markov source model. This property will be used in following sections.

The discrete hidden Markov source model is very suitable for describing the behaviour of the output of a trellis encoder, which generates a sequence of related or dependent symbols that are input to a discrete memoryless channel. Let $B_1, B_2, B_3, \ldots$ be a sequence of branches from the trellis of a given encoder, which assigns values of a discrete alphabet A_B each transition or branch, and let $X_1, X_2, X_3, \ldots$ be the sequence of values taken from a discrete alphabet A_X, such that X_j is the output assigned a given transition or branch B_j. Then the sequence $X_1, X_2, X_3, \ldots$ can be considered as coming from a discrete hidden Markov source. This sequence is input to a discrete memoryless channel, and the resulting output sequence $Y_1, Y_2, Y_3, \ldots$ can also be considered as a discrete hidden Markov source.

The trellis of an encoder, which essentially represents an FSSM, can be designed for either block or convolutional codes, and is an example of the operation of a discrete hidden Markov source. In a trellis encoder the output assigned to each branch represents the input value that is transmitted by means of an output sequence of related or dependant symbols. In this case the event of the occurrence $Br_j = br_j$ of a transition or branch of the trellis is described by the input value m_j that produces such a transition, the output value X_j that is transmitted at that branch and the previous and present states S_{j-1} and S_j that define that branch or transition.

In general, data input to an FSSM are supposed to be independent so that the joint probability distribution function of a sequence of these input data is equal to the product $\prod_{j=1}^{n} p(m_j)$. Therefore the transition probabilities of the source are equal to

$$Q_j(br_j/br_{j-1}) = \begin{cases} p(m_j) & \text{transition } S_{j-1} \to S_j \text{ associated to } m_j, X_j \\ 0 & \text{transition } S_{j-1} \to S_j \text{ does not exist} \end{cases} \tag{23}$$

When dealing with a given discrete hidden Markov source, the intention is to determine the hidden variables as a function of the observable variables. In the case of a discrete memoryless channel through which a hidden Markov chain is transmitted, the intention is to determine input variables represented by a sequence $\boldsymbol{X}$, as a function of the observation of the output variables of the channel, represented by a sequence $\boldsymbol{Y}$.

An iterative solution for the above problem is the Baum and Welch algorithm, which was applied to the decoding of convolutional codes by Bahl, Cocke, Jelinek and Raviv (BCJR) [4] in the design of the so-called BCJR algorithm. The aim of this algorithm is to determine an estimate or soft decision for a given sequence element at position j, X_j, of the hidden Markov chain, by observing the output sequence of a discrete memoryless channel $\boldsymbol{Y}$ that corresponds to a given input sequence $\boldsymbol{X}$. Thus, and by observing the output of the channel $\boldsymbol{Y} = Y_1^n = \{Y_1, Y_2, \ldots, Y_n\}$ that corresponds to an input $\boldsymbol{X}$, the following MAP estimate can be calculated:

$$\mu_{\text{MAP}}(X_j) = p(X_j, \boldsymbol{Y}) \tag{24}$$

where $\boldsymbol{Y}$ is a vector that contains the observed output values. A maximum likelihood measure of the same event is

$$\mu_{\text{ML}}(X_j) = \frac{p(X_j, \boldsymbol{Y})}{p(X_j)} \tag{25}$$

The algorithm estimates the joint probability $p(X_j, \boldsymbol{Y})$ that defines either of these two measures, and that can be factorized as in (22).

The Bayes rule and the following properties for joint events will be useful for obtaining expressions for $p(X_j, \boldsymbol{Y})$:

For any two random variables X and Y, the joint probability of X and Y, $P(X, Y)$, can be expressed as a function of the conditional probability of X and Y, $P(X/Y)$, as

$$P(X, Y) = P(X/Y)P(Y) \tag{26}$$

For the joint events $V = \{X, Y\}$, $W = \{Y, Z\}$ considered as other random variables, the Bayes rule and expression (26) lead to the following expression:

$$P(\{X, Y\}/Z) = P(V/Z) = \frac{P(V, Z)}{P(Z)} = \frac{P(X, Y, Z)}{P(Z)} = \frac{P(X, W)}{P(Z)} = \frac{P(X/W)P(W)}{P(Z)} \tag{27}$$

$$P(\{X, Y\}/Z) = P(X/\{Y, Z\})\frac{P(Y, Z)}{P(Z)} = P(X/\{Y, Z\})P(Y/Z) \tag{28}$$

On the other hand, and by applying the statistical properties of a Markov chain and of a memoryless channel,

$$p\left(Y_j/\left\{X_j, Y_1^{j-1}\right\}\right) = p(Y_j/X_j) \tag{29}$$

and

$$p\left(Y_{j+1}^n/\left\{Y_j, X_j, Y_1^{j-1}\right\}\right) = p(Y_{j+1}^n/X_j) \tag{30}$$

7.4 The BCJR Algorithm: Trellis Coding and Discrete Memoryless Channels

So far, discrete hidden Markov sources and their relationship to soft-decision decoding have been described. The problem of the decoding of a turbo code is essentially to determine MAP estimates or soft decisions of states and transitions of a trellis encoder, seen as a discrete hidden Markov source whose output sequence is observed through a discrete memoryless channel. This is shown in Figure 7.4.

The discrete hidden Markov source, as given in Figure 7.4, represents a trellis encoder (for either block or convolutional codes), or in general, an FSSM seen as a discrete source of finite states. This discrete hidden Markov has U states $u = 0, 1, 2, \ldots, U - 1$. The state of the source

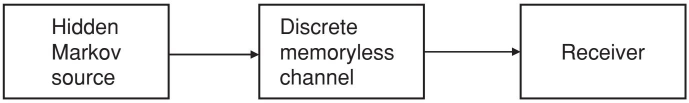


Figure 7.4 Scenario for the BCJR algorithm

in time instant i is denoted as S_i, and its output is X_i. A sequence of states from time instant i to time instant j will be denoted as $S_i^j = \{S_i, S_{i+1}, \ldots, S_j\}$, and will be described by the corresponding output sequence $X_i^j = \{X_i, X_{i+1}, \ldots, X_j\}$. X_i is the ith output symbol taken from a discrete alphabet. The state transitions are determined by the transition probabilities

$$p_i(u/u') = P(S_i = u/S_{i-1} = u') \tag{31}$$

and the corresponding outputs by the probabilities

$$q_i(X/\{u', u\}) = P(X_i = x/\{S_{i-1} = u', S_i = u\}) \tag{32}$$

where x is taken from the discrete output alphabet.

The discrete hidden Markov source generates a sequence X_1^n that starts at state $S_0 = 0$ and ends at the same state $S_0 = 0$. The output of the discrete hidden Markov source X_1^n is the input of a noisy discrete memoryless channel that generates the distorted sequence $Y_1^n = \{Y_1, Y_2, \ldots, Y_n\}$. Transition probabilities of the discrete memoryless channel are defined as $R(Y_j/X_j)$, such that for every time instant $1 \leq i \leq n$,

$$P\left(Y_1^i/X_1^i\right) = \prod_{j=1}^{i} R(Y_j/X_j) \tag{33}$$

The term $R(Y_j/X_j)$ determines the probability that at time instant j, the symbol Y_j is the output of the channel if the symbol X_j was input to that channel. This will happen with a transition probability $P(y_j/x_j)$ that the input symbol x_j converts into the output symbol y_j.

A decoder for this Markov process has to estimate the MAP probability of states and outputs of the discrete hidden Markov source by observing the output sequence $Y_1^n = \{Y_1, Y_2, \ldots, Y_n\}$. This means that it should calculate the probabilities

$$P\big(S_i = u/Y_1^n\big) = \frac{P\left(S_i = u, Y_1^n\right)}{P(Y_1^n)} \tag{34}$$

$$P\big(\{S_{i-1} = u', S_i = u\}/Y_1^n\big) = \frac{P\left(S_{i-1} = u', S_i = u, Y_1^n\right)}{P(Y_1^n)} \tag{35}$$

The notation here is that the state S_i defines a given state i in a trellis, whereas its particular value is obtained from an alphabet U of states of the trellis, with $u = 0, 1, 2, \ldots, U - 1$. Therefore, in a trellis, the sequence $Y_1^n = \{Y_1, Y_2, \ldots, Y_n\}$ is represented by a unique path.

The following MAP probability is associated with each node or state of a trellis:

$$P\left(S_i = u/Y_1^n\right) \tag{36}$$

and the following MAP probability is associated with each branch or transition of the trellis:

$$P\left(\{S_{i-1} = u', S_i = u\}/Y_1^n\right) \tag{37}$$

The decoder will calculate these probabilities by observing the output sequence $Y_1^n = \{Y_1, Y_2, \ldots, Y_n\}$. The decoder can also calculate the joint probabilities

$$\lambda_i(u) = P\left(S_i = u, Y_1^n\right) \tag{38}$$

and

$$\sigma_i(u', u) = P\left(S_{i-1} = u', S_i = u, Y_1^n\right) \tag{39}$$

Since for a given output sequence $Y_1^n = \{Y_1, Y_2, \ldots, Y_n\}$, the probability $P\left(Y_1^n\right)$ is a constant value, the quantities $\lambda_i(u)$ and $\sigma_i(u', u)$ can be divided by $P\left(Y_1^n\right)$ to determine the desired MAP probabilities. Thus, there is a method for calculating the probabilities $\lambda_i(u)$ and $\sigma_i(u', u)$. This is obtained by defining the probabilities

$$\alpha_i(u) = P\left(S_i = u, Y_1^i\right) \tag{40}$$

$$\beta_i(u) = P\left(Y_{i+1}^n / S_i = u\right) \tag{41}$$

$$\gamma_i(u', u) = P\left(\{S_i = u, Y_i\} / S_{i-1} = u'\right) \tag{42}$$

Then

$$\lambda_i(u) = P\left(S_i = u, Y_1^n\right) = P\left(S_i = u, Y_1^i\right) P\left(Y_{i+1}^n / \left\{S_i = u, Y_1^i\right\}\right) \tag{43}$$

where

$$P\left(Y_{i+1}^n / \left\{S_i = u, Y_1^i\right\}\right) = \frac{P\left(S_i = u, Y_{i+1}^n, Y_1^i\right)}{P\left(S_i = u, Y_1^i\right)} = \frac{P\left(S_i = u, Y_1^n\right)}{P\left(S_i = u, Y_1^i\right)} \tag{44}$$

But since $\alpha_i(u) = P\left(S_i = u, Y_1^i\right)$,

$$\lambda_i(u) = P\left(S_i = u, Y_1^i\right) P\left(Y_{i+1}^n / \left\{S_i = u, Y_1^i\right\}\right) = \alpha_i(u) P\left(Y_{i+1}^n / S_i = u\right) \tag{45}$$

The above simplification is obtained by applying the property of discrete hidden Markov sources that states that, for a given event characterized by the state S_i, past and future events do not depend on the value at this state, so that past, present and future events are all statistically independent.

Then, and since $\beta_i(u) = P\left(Y_{i+1}^n / S_i = u\right)$,

$$\lambda_i(u) = \alpha_i(u) P\left(Y_{i+1}^n / S_i = u\right) = \alpha_i(u)\beta_i(u) \tag{46}$$

An equivalent expression can be obtained for $\sigma_i(u', u)$ as

$$\begin{aligned}
\sigma_i(u', u) &= P\left(S_{i-1} = u', S_i = u, Y_1^n\right) \\
&= P\left(Y_{i+1}^n / \left\{S_{i-1} = u', S_i = u, Y_1^{i-1}, Y_i\right\}\right) P\left(S_{i-1} = u', S_i = u, Y_1^{i-1}, Y_i\right) \\
&= P\left(Y_{i+1}^n / S_i = u\right) P\left(S_{i-1} = u', S_i = u, Y_1^{i-1}, Y_i\right) \\
&= P\left(S_{i-1} = u', Y_1^{i-1}\right) P\left(\{S_i = u, Y_i\} / S_{i-1} = u'\right) P\left(Y_{i+1}^n / S_i = u\right) \\
&= P\left(Y_{i+1}^n / S_i = u\right) P\left(\{Y_i, S_i = u\} / \left\{S_{i-1} = u', Y_1^{i-1}\right\}\right) P\left(\left\{S_{i-1} = u', Y_1^{i-1}\right\}\right) \\
&= P\left(Y_{i+1}^n / S_1 = u\right) P\left(\{S_i = u, Y_i\} / S_{i-1} = u'\right) P\left(S_{i-1} = u', Y_1^{i-1}\right)
\end{aligned} \tag{47}$$

Thus

$$\begin{aligned}\sigma_i(u', u) &= P\left(S_{i-1} = u', Y_1^{i-1}\right) P\left(\{S_i = u, Y_i\} / S_{i-1} = u'\right) P\left(Y_{i+1}^n / S_i = u\right) \\ &= \alpha_{i-1}(u')\gamma_i(u', u)\beta_i(u)\end{aligned} \tag{48}$$

7.5 Iterative Coefficient Calculation

In the following, coefficients $\alpha_i(u)$ and $\beta_i(u)$ are calculated by iteration as a function of coefficients $\gamma_i(u', u)$.

For $i = 0, 1, 2, \ldots, n$, the definition of $\alpha_{i-1}(u')$ allows us to describe the term $\alpha_i(u)$ as

$$\alpha_i(u) = P\left(S_i = u, Y_1^i\right) = P\left(S_i = u, Y_1^{i-1}, Y_i\right) \tag{49}$$

$$\begin{aligned}\alpha_i(u) &= \sum_{u'=0}^{U-1} P\left(S_{i-1} = u', S_i = u, Y_1^{i-1}, Y_i\right) \\ &= \sum_{u'=0}^{U-1} P\left(S_{i-1} = u', Y_1^{i-1}\right) P\left(\{S_i = u, Y_i\} / \left\{S_{i-1} = u', Y_1^{i-1}\right\}\right)\end{aligned} \tag{50}$$

$$\alpha_i(u) = \sum_{u'=0}^{U-1} P\left(S_{i-1} = u', Y_1^{i-1}\right) P\left(\{S_i = u, Y_i\} / S_{i-1} = u'\right) = \sum_{u'=0}^{U-1} \alpha_{i-1}(u')\gamma_i(u', u) \tag{51}$$

For $i = 0$, the decoder utilizes the initial conditions $\alpha_0(0) = 1$ and $\alpha_0(u) = 0, u \neq 0$. In the same way, for $i = 1, 2, \ldots, n-1$,

$$\begin{aligned}\beta_i(u) &= \sum_{u'=0}^{U-1} P\left(\left\{S_{i+1} = u', Y_{i+1}^n\right\} / S_i = u\right) \\ &= \sum_{u'=0}^{U-1} P\left(\left\{S_{i+1} = u', Y_{i+1}\right\} / S_i = u\right) P\left(Y_{i+2}^n / S_{i+1} = u'\right) \\ \beta_i(u) &= \sum_{u'=0}^{U-1} = \beta_{i+1}(u')\gamma_{i+1}(u, u')\end{aligned} \tag{52}$$

For $i = n$, the decoder utilizes the contour conditions $\beta_n(0) = 1$ and $\beta_n(u) = 0, u \neq 0$. This is true for a terminated trellis, that is, for a trellis that starts and ends in the zero state S_0. If this is not the case, then $\beta_n(u) = 1 \forall u$.

On the other hand, values of $\gamma_i(u', u)$ are also calculated as

$$\begin{aligned}\gamma_i(u', u) &= \sum_{A_X} P(S_i = u / S_{i-1} = u') P(X_i = x / \{S_{i-1} = u', S_i = u\}) P\left(Y_i / X_i\right) \\ \gamma_i(u', u) &= \sum_{A_X} p_i(u/u') q_i(X / \{u', u\}) R(Y_i / X_i)\end{aligned} \tag{53}$$

The sum in the above expression is done over the entire input alphabet A_x.

The decoding procedure for calculating the values of $\lambda_i(u)$ and $\sigma_i(u', u)$ is applied as follows:

1. Set the initial conditions $\alpha_0(0) = 1$ and $\alpha_0(u) = 0, u \neq 0$, and the contour conditions $\beta_n(0) = 1$ and $\beta_n(u) = 0, u \neq 0$ for $u = 0, 1, 2, \ldots, U - 1$.
2. After receiving Y_i, the decoder calculates $\gamma_i(u', u)$ with equation (53) and determines $\alpha_i(u)$ with equation (51). The obtained values are stored for every i and every u.
3. After receiving the whole sequence Y_1^n, the decoder recursively calculates the values $\beta_i(u)$ by using expression (52). Once all the values $\beta_i(u)$ are determined, they can be multiplied by $\alpha_i(u)$ and $\gamma_i(u', u)$ in order to determine values of $\lambda_i(u)$ and $\sigma_i(u', u)$ according to expressions (46) and (48).

Any event dependent on the trellis states can be measured by adding the corresponding probabilities $\lambda_i(u)$, and any event dependent on the trellis transitions can be measured by adding the corresponding probabilities $\sigma_i(u', u)$.

The iterative calculation of the values $\alpha_i(u)$ is usually called the forward recursion, while the iterative calculation of values $\beta_i(u)$ is called the backward recursion. The input information probability is related to the values $\lambda_i(u)$, and the coded information probability is related to the values $\sigma_i(u', u)$.

Example 7.1: The BCJR algorithm is applied to the decoding of the block code $C_b(5, 3)$ with minimum Hamming distance $d_{\min} = 2$ and the generator matrix

$$\boldsymbol{G} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

In this code there are eight code vectors. The parity check matrix is conveniently obtained by converting the generator matrix into systematic form. This can be done by adding the three rows of the generator matrix and by replacing the third row by this sum of rows. The resulting matrix is of a systematic form

$$\boldsymbol{G}' = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Then

$$\boldsymbol{P}' = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$$

and

$$\boldsymbol{H}' = \left[\boldsymbol{I}_q \boldsymbol{P}'^T\right] = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix} = \boldsymbol{H}$$

Table 7.1 Code vectors of the block code $C_b(5, 3)$

0	0	0	0	0
1	1	0	0	1
0	1	0	1	0
1	0	0	1	1
1	0	1	0	0
0	1	1	0	1
1	1	1	1	0
0	0	1	1	1

It is verified that

$$\boldsymbol{G} \circ \boldsymbol{H}^T = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} = \mathbf{0}$$

Matrix $\boldsymbol{H}$ is the parity check matrix of the code. The eight code vectors of this code can be obtained by multiplying the message vectors by the generator matrix $\boldsymbol{G}$.

Table 7.1 allows us to determine the minimum weight, and therefore the minimum Hamming distance of the code, which is $p_{\mathrm{H}} = d_{\min} = 2$.

A trellis for this block code can be constructed, based on the information provided by the code table, as shown in Figure 7.5.

It is assumed that the message bits that are input to the encoder of the block code are equally likely. The operation of this code will be studied on the soft-decision, discrete, symmetric and memoryless channel that is shown in Figure 7.6, whose transition probabilities are described in Table 7.2. This channel has two inputs and four outputs, two of high reliability, and two of low reliability.

It is assumed, in this example, that the transmitted code vector is $\boldsymbol{c} = (00000)$ and the corresponding received vector is $\boldsymbol{r} = (10200)$. Note that elements of the transmitted vector are inputs of the channel of Figure 7.6, and elements of the received vector are outputs of that

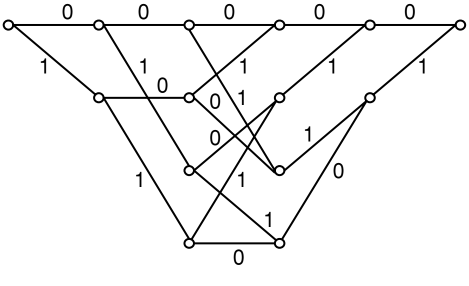


Figure 7.5 A trellis for the block code $C_b(5, 3)$

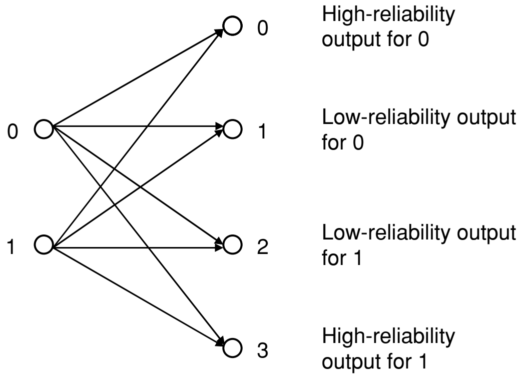

0
1
0
1
2
3
High-reliability output for 0
Low-reliability output for 0
Low-reliability output for 1
High-reliability output for 1

Figure 7.6 A soft-decision discrete symmetric memoryless channel

channel. Table 7.3 shows the transition probabilities for the input elements '1' and '0' for each received element.

The values $\gamma_i(u', u)$ are first calculated and then $\alpha_i(u)$ and $\beta_i(u)$ can also be determined. First, the calculations are described in detail as

$$\begin{aligned}\gamma_1(0,0) &= \sum_{x \in A_x} P(S_1 = 0/S_0 = 0)P(X_1 = x/\{S_0 = 0, S_1 = 0\})P(Y_1/X_1 = x)\\ &= P(S_1 = 0/S_0 = 0)P(X_1 = 0/\{S_0 = 0, S_1 = 0\})P(Y_1/X_1 = 0)\\ &\quad + P(S_1 = 0/S_0 = 0)P(X_1 = 1/\{S_0 = 0, S_1 = 0\})P(Y_1/X_1 = 1)\\ &= 0.5 \times 1 \times 0.3 + 0.5 \times 0 \times 0.15\\ &= 0.15\end{aligned}$$

$$\begin{aligned}\gamma_1(0,1) &= \sum_{X} P(S_1 = 1/S_0 = 0)P(X_1 = x/\{S_0 = 0, S_1 = 1\})P(Y_1/X_1 = x)\\ &= P(S_1 = 1/S_0 = 0)P(X_1 = 0/\{S_0 = 0, S_1 = 1\})P(Y_1/X_1 = 0)\\ &\quad + P(S_1 = 1/S_0 = 0)P(X_1 = 1/\{S_0 = 0, S_1 = 1\})P(Y_1/X_1 = 1)\\ &= 0.5 \times 0 \times 0.3 + 0.5 \times 1 \times 0.15\\ &= 0.075\end{aligned}$$

Table 7.2 Transition probabilities of the channel of Figure 7.6

	$P(y/x)$			
x, y	0	1	2	3
0	0.5	0.3	0.15	0.05
1	0.05	0.15	0.3	0.5

Table 7.3 Transition probabilities for the received vector in Example 7.1

j	1	2	3	4	5
$(P(y_j/0), P(y_j/1))$	(0.3, 0.15)	(0.5, 0.05)	(0.15, 0.3)	(0.5, 0.05)	(0.5, 0.05)

$$\begin{aligned}
\gamma_2(0,0) &= \sum_X P(S_2 = 0/S_1 = 0)P(X_2 = x/\{S_1 = 0, S_2 = 0\})P(Y_2/X_2 = x) \\
&= P(S_2 = 0/S_1 = 0)P(X_2 = 0/\{S_1 = 0, S_2 = 0\})P(Y_2/X_2 = 0) \\
&\quad + P(S_2 = 0/S_1 = 0)P(X_2 = 1/\{S_1 = 0, S_2 = 0\})P(Y_2/X_2 = 1) \\
&= 0.5 \times 1 \times 0.5 + 0.5 \times 0 \times 0.05 \\
&= 0.25
\end{aligned}$$

$$\begin{aligned}
\gamma_2(0,1) &= P(S_2 = 1/S_1 = 0)P(X_2 = 0/\{S_1 = 0, S_2 = 1\})P(Y_2/X_2 = 0) \\
&\quad + P(S_2 = 1/S_1 = 0)P(X_2 = 1/\{S_1 = 0, S_2 = 1\})P(Y_2/X_2 = 1) \\
&= 0 \times 0 \times 0.5 + 0 \times 0 \times 0.05 \\
&= 0
\end{aligned}$$

$$\begin{aligned}
\gamma_2(0,2) &= P(S_2 = 2/S_1 = 0)P(X_2 = 0/\{S_1 = 0, S_2 = 2\})P(Y_2/X_2 = 0) \\
&\quad + P(S_2 = 2/S_1 = 0)P(X_2 = 1/\{S_1 = 0, S_2 = 2\})P(Y_2/X_2 = 1) \\
&= 0.5 \times 0 \times 0.5 + 0.5 \times 1 \times 0.05 \\
&= 0.025
\end{aligned}$$

$$\begin{aligned}
\gamma_2(0,3) &= P(S_2 = 3/S_1 = 0)P(X_2 = 0/\{S_1 = 0, S_2 = 3\})P(Y_2/X_2 = 0) \\
&\quad + P(S_2 = 3/S_1 = 0)P(X_2 = 1/\{S_1 = 0, S_2 = 3\})P(Y_2/X_2 = 1) \\
&= 0 \times 0 \times 0.5 + 0 \times 0 \times 0.05 \\
&= 0
\end{aligned}$$

In the same way, the following values are also determined:

$$\begin{aligned}
\gamma_2(1,0) &= 0 \times 0 \times 0.5 + 0 \times 0 \times 0.05 = 0 \\
\gamma_2(1,1) &= 0.5 \times 1 \times 0.5 + 1 \times 0 \times 0.05 = 0.25 \\
\gamma_2(1,2) &= 0 \times 0 \times 0.5 + 0 \times 0 \times 0.05 = 0 \\
\gamma_2(1,3) &= 0.5 \times 0 \times 0.5 + 0.5 \times 1 \times 0.05 = 0.025
\end{aligned}$$

$$\begin{aligned}
\gamma_3(0,0) &= 0.5 \times 1 \times 0.15 + 0.5 \times 0 \times 0.3 = 0.075 \\
\gamma_3(0,1) &= 0 \times 0 \times 0.15 + 0 \times 0 \times 0.3 = 0
\end{aligned}$$

$$\gamma_3(0,2) = 0.5 \times 0 \times 0.15 + 0.5 \times 1 \times 0.3 = 0.15$$

$$\gamma_3(0,3) = 0 \times 0 \times 0.15 + 0 \times 0 \times 0.3 = 0$$

$$\gamma_3(1,0) = 0.5 \times 0 \times 0.15 + 0.5 \times 1 \times 0.3 = 0.15$$

$$\gamma_3(1,1) = 0 \times 0 \times 0.15 + 0 \times 0 \times 0.3 = 0$$

$$\gamma_3(1,2) = 0.5 \times 1 \times 0.15 + 0.5 \times 0 \times 0.3 = 0.075$$

$$\gamma_3(1,3) = 0 \times 0 \times 0.15 + 0 \times 0 \times 0.3 = 0$$

$$\gamma_3(2,0) = 0 \times 0 \times 0.15 + 0 \times 0 \times 0.3 = 0$$

$$\gamma_3(2,1) = 0.5 \times 1 \times 0.15 + 0.5 \times 0 \times 0.3 = 0.075$$

$$\gamma_3(2,2) = 0 \times 0 \times 0.15 + 0 \times 0 \times 0.3 = 0$$

$$\gamma_3(2,3) = 0.5 \times 0 \times 0.15 + 0.5 \times 1 \times 0.3 = 0.15$$

$$\gamma_3(3,0) = 0 \times 0 \times 0.15 + 0 \times 0 \times 0.3 = 0$$

$$\gamma_3(3,1) = 0.5 \times 0 \times 0.15 + 0.5 \times 1 \times 0.3 = 0.15$$

$$\gamma_3(3,2) = 0 \times 0 \times 0.15 + 0 \times 0 \times 0.3 = 0$$

$$\gamma_3(3,3) = 0.5 \times 1 \times 0.15 + 0.5 \times 0 \times 0.3 = 0.075$$

$$\gamma_4(0,0) = 1 \times 1 \times 0.5 + 1 \times 0 \times 0.05 = 0.5$$

$$\gamma_4(0,1) = 0 \times 0 \times 0.5 + 0 \times 0 \times 0.05 = 0$$

$$\gamma_4(1,0) = 1 \times 0 \times 0.5 + 1 \times 1 \times 0.05 = 0.05$$

$$\gamma_4(1,1) = 0 \times 0 \times 0.5 + 0 \times 0 \times 0.05 = 0$$

$$\gamma_4(2,0) = 0 \times 0 \times 0.5 + 0 \times 0 \times 0.05 = 0$$

$$\gamma_4(2,1) = 1 \times 0 \times 0.5 + 1 \times 1 \times 0.05 = 0.05$$

$$\gamma_4(3,0) = 0 \times 0 \times 0.5 + 0 \times 0 \times 0.05 = 0$$

$$\gamma_4(3,1) = 1 \times 1 \times 0.5 + 1 \times 0 \times 0.05 = 0.5$$

$$\gamma_5(0,0) = 1 \times 1 \times 0.5 + 1 \times 0 \times 0.05 = 0.5$$

$$\gamma_5(1,0) = 1 \times 0 \times 0.5 + 1 \times 1 \times 0.05 = 0.05$$

Forward recursive calculation of the values $\alpha_i(u)$ is started by setting the initial conditions $\alpha_0(m) = 0, m \neq 0$:

$$\begin{aligned}\alpha_1(0) &= \sum_{u'=0}^{U-1} \alpha_0(u')\gamma_1(u', u)\\ &= \sum_{u'=0}^{1} \alpha_0(u')\gamma_1(u', u)\end{aligned}$$

$$
\begin{aligned}
&= \alpha_0(0)\gamma_1(0,0) + \alpha_0(1)\gamma_1(1,0)\\
&= 1 \times 0.15 + 0 \times 0 = 0.15
\end{aligned}
$$

$$\alpha_1(1) = \alpha_0(0)\gamma_1(0,1) + \alpha_0(1)\gamma_1(1,1) = 1 \times 0.075 + 0 \times 0 = 0.075$$

$$
\begin{aligned}
\alpha_2(0) &= \alpha_1(0)\gamma_2(0,0) + \alpha_1(1)\gamma_2(1,0) = 0.15 \times 0.25 + 0.075 \times 0 = 0.0375\\
\alpha_2(1) &= \alpha_1(0)\gamma_2(0,\ 1) + \alpha_1(1)\gamma_2(1,\ 1) = 0.15 \times 0 + 0.075 \times 0.25 = 0.01875\\
\alpha_2(2) &= \alpha_1(0)\gamma_2(0,2) + \alpha_1(1)\gamma_2(1,2) = 0.15 \times 0.025 + 0.075 \times 0 = 0.00375\\
\alpha_2(3) &= \alpha_1(0)\gamma_2(0,3) + \alpha_1(1)\gamma_2(1,3) = 0.15 \times 0 + 0.075 \times 0.025 = 0.001875
\end{aligned}
$$

$$
\begin{aligned}
\alpha_3(0) &= \alpha_2(0)\gamma_3(0,0) + \alpha_2(1)\gamma_3(1,0) = 0.0375 \times 0.075 + 0.01875 \times 0.15 = 0.005625\\
\alpha_3(1) &= \alpha_2(0)\gamma_3(0,1) + \alpha_2(1)\gamma_3(1,1) + \alpha_2(2)\gamma_3(2,1) + \alpha_2(3)\gamma_3(3,1)\\
&= 0.0375 \times 0 + 0.01875 \times 0 + 0.00375 \times 0.075 + 0.001875 \times 0.15\\
&= 0.0005625\\
\alpha_3(2) &= \alpha_2(0)\gamma_3(0,2) + \alpha_2(1)\gamma_3(1,2) + \alpha_2(2)\gamma_3(2,2) + \alpha_2(3)\gamma_3(3,2)\\
&= 0.0375 \times 0.15 + 0.01875 \times 0.075 + 0.00375 \times 0 + 0.001875 \times 0\\
&= 0.00703125\\
\alpha_3(3) &= \alpha_2(2)\gamma_3(2,3) + \alpha_2(3)\gamma_3(3,3)\\
&= 0.00375 \times 0.15 + 0.001875 \times 0.075\\
&= 0.000703125
\end{aligned}
$$

$$
\begin{aligned}
\alpha_4(0) &= \alpha_3(0)\gamma_4(0,0) + \alpha_3(1)\gamma_4(1,0) + \alpha_3(2)\gamma_4(2,0) + \alpha_3(3)\gamma_4(3,0)\\
&= 0.005625 \times 0.5 + 0.0005625 \times 0.05 + 0.00703125 \times 0 + 0.000703125 \times 0\\
&= 0.002840625\\
\alpha_4(1) &= \alpha_3(0)\gamma_4(0,1) + \alpha_3(1)\gamma_4(1,1) + \alpha_3(2)\gamma_4(2,1) + \alpha_3(3)\gamma_4(3,1)\\
&= 0.005625 \times 0 + 0.0005625 \times 0 + 0.00703125 \times 0.05 + 0.000703125 \times 0.5\\
&= 0.000703125
\end{aligned}
$$

$$
\begin{aligned}
\alpha_5(0) &= \alpha_4(0)\gamma_5(0,0) + \alpha_4(1)\gamma_5(1,0)\\
&= 0.002840625 \times 0.5 + 0.000703125 \times 0.05\\
&= 0.00145546875
\end{aligned}
$$

Backward recursive calculation of the values $\beta_i(u)$ is done by setting the contour conditions $\beta_5(0) = 1$, $\beta_5(m) = 0$, $m \neq 0$:

$$\beta_4(0) = \beta_5(0)\gamma_5(0, 0) = 1 \times 0.5 = 0.5$$

$$\beta_4(1) = \beta_5(0)\gamma_5(1, 0) = 1 \times 0.05 = 0.05$$

$$\beta_3(0) = \beta_4(1)\gamma_4(0, 1) + \beta_4(0)\gamma_4(0, 0) = 0.05 \times 0 + 0.5 \times 0.5 = 0.25$$

$$\beta_3(1) = \beta_4(0)\gamma_4(1, 0) + \beta_4(1)\gamma_4(1, 1) = 0.5 \times 0.05 + 0.05 \times 0 = 0.025$$

$$\beta_3(2) = \beta_4(0)\gamma_4(2, 0) + \beta_4(1)\gamma_4(2, 1) = 0.5 \times 0 + 0.05 \times 0.05 = 0.0025$$

$$\beta_3(3) = \beta_4(0)\gamma_4(3, 0) + \beta_4(1)\gamma_4(3, 1) = 0.5 \times 0 + 0.05 \times 0.5 = 0.025$$

$$\begin{aligned}\beta_2(0) &= \beta_3(0)\gamma_3(0, 0) + \beta_3(1)\gamma_3(0, 1) + \beta_3(2)\gamma_3(0, 2) + \beta_3(3)\gamma_3(0, 3)\\ &= 0.25 \times 0.075 + 0.025 \times 0 + 0.0025 \times 0.15 + 0.025 \times 0\\ &= 0.019125\end{aligned}$$

$$\begin{aligned}\beta_2(1) &= \beta_3(0)\gamma_3(1, 0) + \beta_3(1)\gamma_3(1, 1) + \beta_3(2)\gamma_3(1, 2) + \beta_3(3)\gamma_3(1, 3)\\ &= 0.25 \times 0.15 + 0.025 \times 0 + 0.0025 \times 0.075 + 0.025 \times 0\\ &= 0.0376875\end{aligned}$$

$$\begin{aligned}\beta_2(2) &= \beta_3(0)\gamma_3(2, 0) + \beta_3(1)\gamma_3(2, 1) + \beta_3(2)\gamma_3(2, 2) + \beta_3(3)\gamma_3(2, 3)\\ &= 0.25 \times 0 + 0.025 \times 0.075 + 0.0025 \times 0 + 0.025 \times 0.15\\ &= 0.005625\end{aligned}$$

$$\begin{aligned}\beta_2(3) &= \beta_3(0)\gamma_3(3, 0) + \beta_3(1)\gamma_3(3, 1) + \beta_3(2)\gamma_3(3, 2) + \beta_3(3)\gamma_3(3, 3)\\ &= 0.25 \times 0 + 0.025 \times 0.15 + 0.0025 \times 0 + 0.025 \times 0.075\\ &= 0.005625\end{aligned}$$

$$\begin{aligned}\beta_1(0) &= \beta_2(0)\gamma_2(0, 0) + \beta_2(1)\gamma_2(0, 1) + \beta_2(2)\gamma_2(0, 2) + \beta_2(3)\gamma_2(0, 3)\\ &= 0.019125 \times 0.25 + 0.0376875 \times 0 + 0.005625 \times 0.025 + 0.005625 \times 0\\ &= 0.004921875\end{aligned}$$

$$\begin{aligned}\beta_1(1) &= \beta_2(0)\gamma_2(1, 0) + \beta_2(1)\gamma_2(1, 1) + \beta_2(2)\gamma_2(1, 2) + \beta_2(3)\gamma_2(1, 3)\\ &= 0.0019125 \times 0 + 0.0376875 \times 0.25 + 0.005625 \times 0 + 0.005625 \times 0.025\\ &= 0.0095625\end{aligned}$$

$$\begin{aligned}\beta_0(0) &= \beta_1(0)\gamma_1(0, 0) + \beta_1(1)\gamma_1(0, 1)\\ &= 0.004921875 \times 0.15 + 0.0095625 \times 0.075\\ &= 0.00145546875\end{aligned}$$

Once the values $\gamma_i(u', u)$, $\alpha_i(u)$ and $\beta_i(u)$ have been determined, then the values $\lambda_i(u)$ and $\sigma_i(u', u)$ can be calculated as

$$\lambda_1(0) = \alpha_1(0)\beta_1(0) = 0.15 \times 0.004921675 = 0.00073828125$$

$$\lambda_1(1) = \alpha_1(1)\beta_1(1) = 0.075 \times 0.0095625 = 0.0007171875$$

$$\lambda_2(0) = \alpha_2(0)\beta_2(0) = 0.00375 \times 0.019125 = 0.0007171875$$

$$\lambda_2(1) = \alpha_2(1)\beta_2(1) = 0.01875 \times 0.0376875 = 0.000706640625$$

$$\lambda_2(2) = \alpha_2(2)\beta_2(2) = 0.00375 \times 0.005625 = 0.00002109375$$

$$\lambda_2(3) = \alpha_2(3)\beta_2(3) = 0.001875 \times 0.005625 = 0.000010546875$$

With all the values already calculated, an estimate or soft decision can be made for each step i of the decoded sequence. The coefficients $\lambda_i(u)$ determine the estimates for input symbols '1' and '0' when there is only one branch or transition of the trellis arriving at a given node, which then defines the value of that node. This happens for instance in the trellis of Figure 7.5 at nodes $\lambda_1(0)$, $\lambda_1(1)$, $\lambda_2(0)$, $\lambda_2(1)$, $\lambda_2(2)$ and $\lambda_2(3)$:

$$\frac{\lambda_1(0)}{\lambda_1(0) + \lambda_1(1)} = 0.5072 \quad \text{Soft decision for '0' at position 1}$$

$$\frac{\lambda_1(1)}{\lambda_1(0) + \lambda_1(1)} = 0.4928 \quad \text{Soft decision for '1' at position 1}$$

$$\frac{\lambda_2(0) + \lambda_2(1)}{\lambda_2(0) + \lambda_2(1) + \lambda_2(2) + \lambda_2(3)} = 0.97826 \quad \text{Soft decision for '0' at position 2}$$

$$\frac{\lambda_2(2) + \lambda_2(3)}{\lambda_2(0) + \lambda_2(1) + \lambda_2(2) + \lambda_2(3)} = 0.0217 \quad \text{Soft decision for '1' at position 2}$$

Coefficients $\sigma_i(u', u)$ are then utilized for determining the soft decisions when there are two or more transitions or branches arriving at a given node of the trellis, and when these branches are assigned the different input symbols:

$$\sigma_3(0, 0) = \alpha_2(0)\gamma_3(0, 0)\beta_3(0) = 0.0375 \times 0.075 \times 0.25 = 0.000703125$$

$$\sigma_3(1, 0) = \alpha_2(1)\gamma_3(1, 0)\beta_3(0) = 0.01875 \times 0.15 \times 0.25 = 0.000703125$$

$$\sigma_3(0, 2) = \alpha_2(0)\gamma_3(0, 2)\beta_3(2) = 0.0375 \times 0.15 \times 0.0025 = 0.0000140625$$

$$\sigma_3(1, 2) = \alpha_2(1)\gamma_3(1, 2)\beta_3(2) = 0.01875 \times 0.075 \times 0.0025 = 0.000003515625$$

$$\sigma_3(2, 1) = \alpha_2(2)\gamma_3(2, 1)\beta_3(1) = 0.00375 \times 0.075 \times 0.025 = 0.00000703125$$

$$\sigma_3(3, 1) = \alpha_2(3)\gamma_3(3, 1)\beta_3(1) = 0.001875 \times 0.15 \times 0.025 = 0.00000703125$$

$$\sigma_3(2, 3) = \alpha_2(2)\gamma_3(2, 3)\beta_3(3) = 0.00375 \times 0.15 \times 0.025 = 0.0000140625$$

$$\sigma_3(3, 3) = \alpha_2(3)\gamma_3(3, 3)\beta_3(3) = 0.001875 \times 0.075 \times 0.025 = 0.000003515625$$

$$\sigma_4(0,0) = \alpha_3(0)\gamma_4(0,0)\beta_4(0) = 0.005625 \times 0.5 \times 0.5 = 0.00140625$$

$$\sigma_4(1,0) = \alpha_3(1)\gamma_4(1,0)\beta_4(0) = 0.0005625 \times 0.05 \times 0.5 = 0.0000140625$$

$$\sigma_4(2,1) = \alpha_3(2)\gamma_4(2,1)\beta_4(1) = 0.00703125 \times 0.05 \times 0.05 = 0.000017578125$$

$$\sigma_4(3,1) = \alpha_3(3)\gamma_4(3,1)\beta_4(1) = 0.000703125 \times 0.5 \times 0.05 = 0.000017578125$$

$$\sigma_5(0,0) = \alpha_4(0)\gamma_5(0,0)\beta_5(0) = 0.002840625 \times 0.5 \times 1 = 0.0014203125$$

$$\sigma_5(1,0) = \alpha_4(1)\gamma_5(1,0)\beta_5(0) = 0.000703125 \times 0.05 \times 1 = 0.00003515625$$

These values allow us to determine soft decisions for the corresponding nodes. For instance, for position $i = 3$, the trellis transition probabilities involved in the calculation of a soft decision for ‘0’ are

$$\frac{\sigma_3(0,0) + \sigma_3(1,2) + \sigma_3(2,1) + \sigma_3(3,3)}{\sigma_3(0,0) + \sigma_3(1,0) + \sigma_3(0,2) + \sigma_3(1,2) + \sigma_3(2,1) + \sigma_3(3,1) + \sigma_3(2,3) + \sigma_3(3,3)}$$
$$= 0.49275$$

which is a soft decision for ‘0’ at position 3. The soft decision for ‘1’ at that position is then $1 - 0.49275 = 0.5072$. For position $i = 4$, the trellis transition probabilities involved in the calculation of a soft decision for ‘0’ are

$$\frac{\sigma_4(0,0) + \sigma_4(3,1)}{\sigma_4(1,0) + \sigma_4(0,0) + \sigma_4(2,1) + \sigma_4(3,1)} = 0.97826$$

and the soft decision for ‘1’ at position $i = 4$ is

$$\frac{\sigma_4(1,0) + \sigma_4(2,1)}{\sigma_4(1,0) + \sigma_4(0,0) + \sigma_4(2,1) + \sigma_4(3,1)} = 0.021739$$

For position $i = 5$, the trellis transition probabilities involved in the calculation of a soft decision for ‘0’ are

$$\frac{\sigma_5(0,0)}{\sigma_5(0,0) + \sigma_5(1,0)} = 0.97584$$

and the soft decision for ‘1’ at position $i = 5$ is

$$\frac{\sigma_5(1,0)}{\sigma_5(0,0) + \sigma_5(1,0)} = 0.02415$$

Based on the above calculations, the decoder will decide that the decoded vector is $\boldsymbol{d} =$ (00100), which is not a code vector. The code vectors closest to the decoded vector are $\boldsymbol{c} =$ (00000) and $\boldsymbol{c} =$ (10100). Table 7.4 shows the distance between any code vector and the received vector $\boldsymbol{r} =$ (10200). This can help to understand why the decoder is not able to correctly decide on the true code vector in this case.

In the Table 7.4, distances are measured as soft distances calculated over the soft-decision channel of this example. In this table it is seen that the two code vectors $\boldsymbol{c} =$ (00000) and $\boldsymbol{c} =$ (10100) that are closest to the received vector $\boldsymbol{r} =$ (10200) have the same distance with

Table 7.4 Soft-decision distances from the code vectors to the received vector $\boldsymbol{r} = (10200)$

Code Vectors					Distance to $\boldsymbol{r} = (10200)$					
					1	**0**	**2**	**0**	**0**	d
0	0	0	0	0	0	0	0	0	0	3
1	0	0	1	1	3	0	0	3	3	10
0	1	0	1	0	0	3	0	3	0	9
1	1	0	0	1	3	3	0	0	3	10
0	0	1	1	1	0	0	3	3	3	8
1	0	1	0	0	3	0	3	0	0	3
0	1	1	0	1	0	3	3	0	3	8
1	1	1	1	0	3	3	3	3	0	9

respect to this vector, so that the decoder cannot correctly decode this error pattern. However soft decisions or estimates for the bits of the received vector might be helpful to the user, and could be iteratively updated to converge to the right solution if they were involved in a turbo decoding scheme with other component codes.

Example 7.2: Decode the received vector of Example 7.1 by using the ML Viterbi algorithm, where trellis transitions are assigned the conditional probability values of the channel utilized in that example.

The ML Viterbi algorithm can be used for decoding the received vector of the Example 7.1 if the trellis transitions in the corresponding trellis (Figure 7.5) are assigned the transition probabilities for the input elements '1' and '0' of the soft-decision channel used in that example (Figure 7.6). In this way, the decoding algorithm operates as shown in Figure 7.7, where, at time instant t_3, decisions can be already taken, in order to decide which is the survivor path among those that arrive at a given node of the trellis.

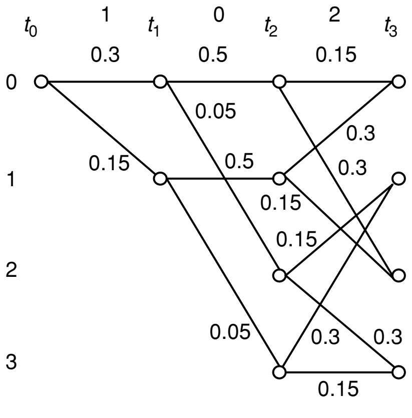


Figure 7.7 ML Viterbi decoding algorithm at time instant t_3, Example 7.2

At time instant t_3, and for state 0, there are two arriving branches with the cumulative probabilities

$$\text{time instant } t_3,\ \text{state } 0\ \Rightarrow \begin{cases} b_1 \rightarrow 0.3 \times 0.5 \times 0.15 = 0.0225(^*) \\ b_2 \rightarrow 0.15 \times 0.5 \times 0.3 = 0.0225 \end{cases}$$

At this trellis node the cumulative probability is equal for both arriving branches, and so an arbitrary decision is made in favour of the upper branch. This decision will influence the final decision that the decoding algorithm will take, which will fortuitously decide for the true vector. The decision taken at each node is denoted with an asterisk (*).

The same procedure is applied to determine the soft decision for the other values of this state, and thus

$$\text{time instant } t_3,\ \text{state } 1\ \Rightarrow \begin{cases} b_1 \rightarrow 0.3 \times 0.05 \times 0.15 = 0.00225(^*) \\ b_2 \rightarrow 0.15 \times 0.05 \times 0.3 = 0.00225 \end{cases}$$

Once again, the decision for state 1 is determined by making the arbitrary choice that the upper branch is the survivor path. This repeated occurrence of the need to make an arbitrary choice of survivor path is evidence that the received vector contains an error event that exceeds the correcting power of the code, and is such that two code vectors are at the same distance from the received vector.

$$\text{time instant } t_3,\ \text{state } 2\ \Rightarrow \begin{cases} b_1 \rightarrow 0.3 \times 0.5 \times 0.3 = 0.045(^*) \\ b_2 \rightarrow 0.15 \times 0.5 \times 0.15 = 0.01125 \end{cases}$$

$$\text{time instant } t_3,\ \text{state } 3\ \Rightarrow \begin{cases} b_1 \rightarrow 0.3 \times 0.05 \times 0.3 = 0.0045(^*) \\ b_2 \rightarrow 0.15 \times 0.05 \times 0.15 = 0.001125 \end{cases}$$

Figure 7.8 shows the resulting situation after the discarding of some paths at time instant t_3.

For a clearer description of the calculations involved, each arriving path is marked with the product of all the channel transition probabilities that contribute to the calculation of the soft metric of that path. Thus, for example, at time instant t_4 the following decisions are taken:

$$\text{time instant } t_4,\ \text{state } 0\ \Rightarrow \begin{cases} b_1 \rightarrow 0.3 \times 0.5 \times 0.15 \times 0.5 = 0.01125(^*) \\ b_2 \rightarrow 0.3 \times 0.05 \times 0.15 \times 0.05 = 0.0001125 \end{cases}$$

$$\text{time instant } t_4,\ \text{state } 1\ \Rightarrow \begin{cases} b_1 \rightarrow 0.3 \times 0.5 \times 0.3 \times 0.05 = 0.00225(^*) \\ b_2 \rightarrow 0.3 \times 0.05 \times 0.3 \times 0.5 = 0.00225 \end{cases}$$

Figure 7.9 shows the resulting situation after the discarding of some paths at time instant t_4.

The final decision adopted at time instant t_5 is then

$$\text{time instant } t_5,\ \text{state } 0\ \Rightarrow \begin{cases} b_1 \rightarrow 0.3 \times 0.5 \times 0.15 \times 0.5 \times 0.5 = 0.005625(^*) \\ b_2 \rightarrow 0.3 \times 0.5 \times 0.3 \times 0.05 \times 0.05 = 0.0001125 \end{cases}$$

The most likely path is the survivor path, as indicated in Figure 7.9 in bold, and the ML Viterbi decoding algorithm decides for the code vector $\boldsymbol{d} = \boldsymbol{c} = (00000)$. This decision is the correct decision, but it is being obtained by chance, because of the two arbitrary upper branch survivor path selections taken at time instant t_3. If however the decision rule at this critical time instant, where cumulative probabilities are equal for two arriving branches at the

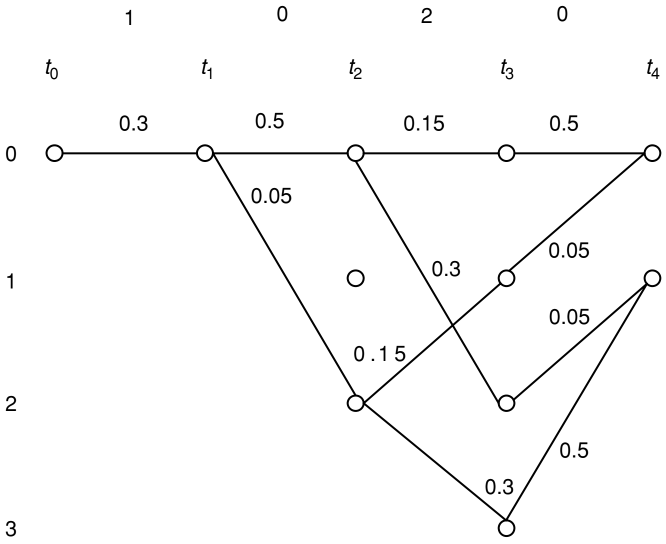


Figure 7.8 ML Viterbi decoding algorithm at time instant t_4, Example 7.2

nodes corresponding to state 0 and state 1, was such that the lower branch had been selected, then the final result would have been to decode the code vector $\boldsymbol{d} = \boldsymbol{c} = (10100)$, that is, the other code vector that is at the same distance as the code vector $\boldsymbol{c} = (00000)$ to the received vector $\boldsymbol{r} = (10200)$. This emergence of two equally possible codewords confirms the same result achieved by using the soft distances given in Table 7.3. Unless additional decoding information is available, if for instance the code is part of a turbo scheme, then there is no way to determine which codeword is the correct one. However, even in an iterative decoding algorithm this sort of ambiguous situation can arise after a given number of iterations, seen as a fluctuating and alternating decision between code vectors that are at the same distance with

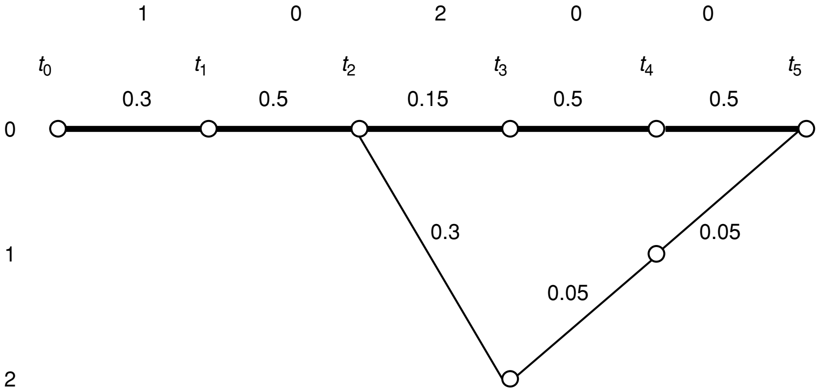


Figure 7.9 ML Viterbi algorithm at time instant t_5 and final decision, Example 7.2

respect to the received vector. Thus, the decision could fortuitously be in favour of the true code vector or not depending on when the iteration of the algorithm is truncated.

This simple example shows the difference in decoding complexity between the BCJR algorithm and the ML Viterbi algorithm. In Example 7.3 it will be however evident that in spite of this higher complexity, the MAP BCJR decoding algorithm has a very efficient error-correction capability when it is involved in iterative decoding algorithms. In this sense, it can be said that the use of the MAP BCJR algorithm in the constituent decoders of an iterative turbo decoding scheme is much less complex than the use of the Viterbi decoding algorithm applied over the trellis of the complete turbo code, which will be very complex, especially for large-size interleavers.

7.6 The BCJR MAP Algorithm and the LLR

The LLR can also be defined for conditional probabilities. Indeed MAP decoding algorithms perform a soft decision or estimation of a given bit conditioned or based on the reception of sampled values of a given received sequence $\boldsymbol{Y}$. In this case the LLR is denoted as $L(b_i/\boldsymbol{Y})$ and is defined as follows [2, 5]:

$$L(b_i/\boldsymbol{Y}) = \ln\left(\frac{P(b_i = +1/\boldsymbol{Y})}{P(b_i = -1/\boldsymbol{Y})}\right) \tag{54}$$

This estimation is based on the *a posteriori* probabilities of the bit b_i that are determined in iterative decoding algorithms as soft-input–soft-output decisions for each constituent decoder in the decoding of a turbo code.

Another useful conditional LLR is based on the ratio of the probabilities that the output of an optimal decoder is y_i if the corresponding transmitted bit x_i adopts one of its two possible values $+1$ or -1. In logarithmic form this conditional LLR is equal to

$$L(y_i/x_i) = \ln\left(\frac{P(y_i/x_i = +1)}{P(y_i/x_i = -1)}\right) \tag{55}$$

For the additive white Gaussian noise (AWGN) channel and for transmitted bits in polar format $x_i = \pm 1$, which after transmission are received using an optimal receiver, the conditional LLRs described in (55) take the form

$$P(y_i/x_i = +1) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-E_\mathrm{b}}{2\sigma^2}(y_i-1)^2} \tag{56}$$

$$P(y_i/x_i = -1) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-E_\mathrm{b}}{2\sigma^2}(y_i+1)^2} \tag{57}$$

$$L(y_i/x_i) = \ln\left(\frac{e^{\frac{-E_\mathrm{b}}{2\sigma^2}(y_i-1)^2}}{e^{\frac{-E_\mathrm{b}}{2\sigma^2}(y_i+1)^2}}\right) = \frac{-E_\mathrm{b}}{2\sigma^2}(y_i-1)^2 + \frac{E_\mathrm{b}}{2\sigma^2}(y_i+1)^2 = 2\frac{E_\mathrm{b}}{\sigma^2}y_i = L_\mathrm{c}y_i \tag{58}$$

Thus, the conditional LLR for an AWGN channel is proportional to the value of the sample of the optimally received signal y_i, and the constant of proportion is $L_\mathrm{c} = 2E_\mathrm{b}/\sigma^2$, a measure of the signal-to-noise ratio in the channel.

7.6.1 The BCJR MAP Algorithm: LLR Calculation

The decoding algorithm introduced by Bahl, Cocke, Jelinek and Raviv [4] was first implemented for the trellis decoding of both block and convolutional codes, and, in comparison with the well-known Viterbi decoding algorithm, the proposed BCJR algorithm did not provide any particular advantage, as its complexity was higher than that of the Viterbi decoder. However, this is a decoding algorithm that inherently utilizes soft-input–soft-output decisions, and this becomes a decisive factor for its application in the iterative decoding of turbo codes.

In the following, a description of the BCJR MAP decoding algorithm in terms of LLRs is developed. The definition of an LLR as a logarithm of a quotient allows some of the constant terms involved to be cancelled, so that there is no need to calculate them in this form of implementation of the BCJR algorithm.

The BCJR MAP decoding algorithm determines the probability that a given transmitted bit was a $+1$ or a -1, depending on the received sequence $\mathbf{Y} = Y_1^n$. The LLR $L(b_i/\mathbf{Y})$ summarizes these two possibilities by calculating a unique number

$$L(b_i/\mathbf{Y}) = \ln\left(\frac{P(b_i = +1/\mathbf{Y})}{P(b_i = -1/\mathbf{Y})}\right) \tag{59}$$

The Bayes rule is used to express (59) as

$$L(b_i/\mathbf{Y}) = \ln\left(\frac{P(b_i = +1, \mathbf{Y})}{P(b_i = -1, \mathbf{Y})}\right) \tag{60}$$

Figure 7.10 shows the middle part of the trellis that is seen in Figure 7.5, in polar format. Here, symbol ‘0’ is transmitted through the channel as -1, and symbol ‘1’ is transmitted through the channel as $+1$. As pointed out in previous sections, the use of the polar format is very convenient if decoders make use of LLRs, since this is a signed quantity whose sign is the hard-decision part of the decoded or estimated value, which was transmitted in normalized form as either a $+1$ or a -1.

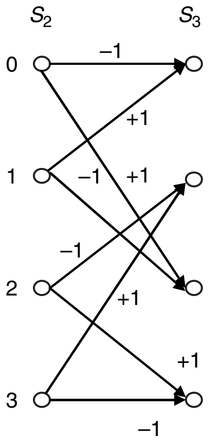


Figure 7.10 Trellis transitions of the block code $C_b(5, 3)$

In this case, for example in the transition from state S_2 to state S_3, the probability that $b_3 = -1$ is given by the probability that this state transition is one of the four possible trellis transitions for which $b_3 = -1$. Then the probability that $b_3 = -1$ is the addition of all the probabilities associated with the information bit -1.

In general the estimation of the information bit b_i is done over the trellis transition with which the bit is associated, defined from state S_{i-1} to state S_i. Equation (60) is then written as

$$\begin{aligned} L(b_i/\mathbf{Y}) &= \ln\left(\frac{\sum_{\{u',u\}\Rightarrow b_i=+1} P(S_{i-1}=u', S_i=u, \mathbf{Y})}{\sum_{\{u',u\}\Rightarrow b_i=-1} P(S_{i-1}=u', S_i=u, \mathbf{Y})}\right) \\ &= \ln\left(\frac{\sum_{\{u',u\}\Rightarrow b_i=+1} P(S_{i-1}=u', S_i=u, Y_1^n)}{\sum_{\{u',u\}\Rightarrow b_i=-1} P(S_{i-1}=u', S_i=u, Y_1^n)}\right) \end{aligned} \tag{61}$$

Here $\{u', u\} \Rightarrow b_i = +1$ represents the set of all the transitions that correspond to the message or information bit $b_i = +1$. The same is true for $\{u', u\} \Rightarrow b_i = -1$ with respect to the message bit $b_i = -1$. Terms of the form $\sigma_i(u', u) = P(S_{i-1} = u', S_i = u, Y_1^n)$ can be expressed as

$$\begin{aligned} \sigma_i(u', u) &= P\left(S_{i-1}=u', S_i=u, Y_1^n\right) \\ &= P\left(S_{i-1}=u', Y_1^{i-1}\right) P\left(\{S_i=u, Y_i\}/S_{i-1}=u'\right) P\left(Y_{i+1}^n/S_i=u\right) \\ &= \alpha_{i-1}(u')\gamma_i(u', u)\beta_i(u) \end{aligned} \tag{62}$$

Then

$$\begin{aligned} L(b_i/\mathbf{Y}) &= L(b_i/Y_1^n) \\ &= \ln\left(\frac{\sum_{\{u',u\}\Rightarrow b_i=+1} P(S_{i-1}=u', S_i=u, Y_1^n)}{\sum_{\{u',u\}\Rightarrow b_i=-1} P(S_{i-1}=u', S_i=u, Y_1^n)}\right) \\ &= \ln\left(\frac{\sum_{\{u',u\}\Rightarrow b_i=+1} \alpha_{i-1}(u')\gamma_i(u', u)\beta_i(u)}{\sum_{\{u',u\}\Rightarrow b_i=-1} \alpha_{i-1}(u')\gamma_i(u', u)\beta_i(u)}\right) \end{aligned} \tag{63}$$

7.6.2 Calculation of Coefficients $\gamma_i(u', u)$

Coefficients $\alpha_{i-1}(u')$ and $\beta_i(u)$ are recursively calculated as functions of coefficients $\gamma_i(u', u)$, so that coefficients $\gamma_i(u', u)$ have to be evaluated first to obtain all the quantities involved in this algorithm. Equation (42) describes the coefficients $\gamma_i(u', u)$ and can be expressed in a more convenient way by making use of the properties described by expressions (26)–(30):

$$\begin{aligned} \gamma_i(u', u) &= P\left(\{S_i=u, Y_i\}/S_{i-1}=u'\right) \\ &= P(Y_i/\{u', u\})P(u/u') \\ &= P(Y_i/\{u', u\})P(b_i) \end{aligned} \tag{64}$$

The bit probability of the ith transition can be calculated by using expression (17):

$$P(b_i) = P(b_i = \pm 1) = \frac{e^{-L(b_i)/2}}{1 + e^{-L(b_i)}} e^{b_i L(b_i)/2} = C_1\, e^{b_i L(b_i)/2} \tag{65}$$

On the other hand, the calculation of the term $P(Y_i/\{u', u\})$ is equivalent to calculating the probability $P(Y_i/X_i)$, where X_i is the vector associated with the transition from $S_{i-1} = u'$ to $S_i = u$, which in general is a vector of n bits. If the channel is a memoryless channel, then

$$P(Y_i/\{u', u\}) = P(Y_i/X_i) = \prod_{k=1}^{n} P(y_{ik}/x_{ik}) \tag{66}$$

where y_{ik} and x_{ik} are the bits of the received and transmitted vectors Y_i and X_i, respectively. If transmission is done in polar format over the AWGN channel, the transmitted bits x_{ik} take the normalized values $+1$ or -1, and [2]

$$L_{\mathrm{e}}(b_i) = \ln\left(\frac{\sum_{\{u',u\}\Rightarrow b_i=+1} \alpha_{i-1}(u')\gamma_{i\_\mathrm{extr}}(u', u)\beta_i(u)}{\sum_{\{u',u\}\Rightarrow b_i=-1} \alpha_{i-1}(u')\gamma_{i\_\mathrm{extr}}(u', u)\beta_i(u)}\right) \tag{67}$$

$$P(Y_i/\{u', u\}) = \prod_{k=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{E_{\mathrm{b}}}{2\sigma^2}\sum_{k=1}^{n}(y_{ik} - x_{ik})^2} \tag{68}$$

$$= \frac{1}{\left(\sqrt{2\pi}\sigma\right)^n} e^{-\frac{E_{\mathrm{b}}}{2\sigma^2}\sum_{k=1}^{n}\left(y_{ik}^2 + x_{ik}^2\right)} e^{\frac{E_{\mathrm{b}}}{\sigma^2}\sum_{k=1}^{n}(y_{ik}x_{ik})} = C_2\, e^{\frac{E_b}{\sigma^2}\sum_{k=1}^{n}(y_{ik}x_{ik})} \tag{69}$$

where only the term $e^{\frac{E_{\mathrm{b}}}{\sigma^2}\sum_{k=1}^{n}(y_{ik}x_{ik})}$ is significant because all the other terms in this expression are constants. The expression for calculating coefficients $\gamma_i(u', u)$ is finally

$$\gamma_i(u', u) = C\, e^{b_i L(b_i)/2}\, e^{\frac{E_{\mathrm{b}}}{\sigma^2}\sum_{k=1}^{n}(y_{ik}x_{ik})}, \quad C = C_1 C_2 \tag{70}$$

The above coefficients can be calculated for the BCJR MAP decoding algorithm by taking into account the channel information or sampled values $L_{\mathrm{c}} y_{ik}$, and the *a priori* information for each constituent decoder $L(b_i)$. This latter information is the extrinsic information provided by the other decoder, information that has been calculated in the previous iteration. With the channel and *a priori* estimations, the decoder is able to determine values of $\gamma_i(u', u)$ for all the transitions of the trellis.

Forward recursive calculations allow us to determine values of coefficients $\alpha_{i-1}(u')$ as a function of coefficients $\gamma_i(u', u)$, and this can be done while the signal is being received. After receiving the whole sequence Y_1^n, backward recursive calculation can be used to determine values of coefficients $\beta_i(u)$. Conditional LLRs $L(b_i/Y_1^n)$ can be finally calculated after determining the values of coefficients $\alpha_{i-1}(u')$, $\gamma_i(u', u)$ and $\beta_i(u)$.

In this particular example, and as seen in Figure 7.5 for the trellis of the block code $C_{\mathrm{b}}(5, 3)$, transitions are assigned only one bit. In general, however, and particularly in the case of trellises for convolutional codes, trellis transitions are usually assigned one or more input bits, and more than one output bits, in the format input bits/output bits. In the most common structures of turbo codes, constituent encoders are RSC encoders of code rate $R_\mathrm{c} = 1/2$, and so the corresponding assignment for inputs in the trellis transitions is done with just one bit. In this latter case it is possible to distinguish between the message information and the redundant information, in the calculation of the coefficients $\gamma_i(u', u)$:

$$\begin{aligned}\gamma_i(u', u) &= C\, e^{(b_i L(b_i)/2)}\, e^{\frac{L_\mathrm{c}}{2}\sum_{k=1}^{n}(y_{ik}x_{ik})}\\ &= C\, e^{(b_i L(b_i)/2)}\, e^{\frac{L_\mathrm{c}}{2}y_{i1}x_{i1}}\, e^{\frac{L_\mathrm{c}}{2}\sum_{k=2}^{n}(y_{ik}x_{ik})}\\ &= C\, e^{(b_i L(b_i)/2)}\, e^{\frac{L_\mathrm{c}}{2}y_{i1}b_i}\, e^{\frac{L_\mathrm{c}}{2}\sum_{k=2}^{n}(y_{ik}x_{ik})}\\ &= C\, e^{(b_i L(b_i)/2)}\, e^{\frac{L_\mathrm{c}}{2}y_{i1}b_i}\, \gamma_{i\_\mathrm{extr}(u',u)} \end{aligned} \tag{71}$$

The received bit y_{i1} in equation (71) corresponds to the transmitted bit $x_{i1} = b_i$, which is the message or source bit that appears first in the encoded sequence of each trellis transition, when systematic convolutional coding is used.

By taking into account the above considerations, and also that in the definition of the LLR $L(b_i/Y_1^n)$, the numerator is composed of the terms associated with $b_i = +1$, whereas the denominator is composed of the terms associated with $b_i = -1$, the LLR $L(b_i/Y_1^n)$ can be written as

$$\begin{aligned} L(b_i/Y_1^n) &= \ln\left(\frac{\sum_{\{u',u\}\Rightarrow b_i=+1}\alpha_{i-1}(u')\gamma_i(u', u)\beta_i(u)}{\sum_{\{u',u\}\Rightarrow b_i=-1}\alpha_{i-1}(u')\gamma_i(u', u)\beta_i(u)}\right)\\ &= \ln\left(\frac{\sum_{\{u',u\}\Rightarrow b_i=+1}\alpha_{i-1}(u')\, e^{+L(b_i)/2}\, e^{+L_\mathrm{c}y_{i1}/2}\gamma_{i\_\mathrm{extr}}(u', u)\beta_i(u)}{\sum_{\{u',u\}\Rightarrow b_i=-1}\alpha_{i-1}(u')\, e^{-L(b_i)/2}\, e^{-L_\mathrm{c}y_{i1}/2}\gamma_{i\_\mathrm{extr}}(u', u)\beta_i(u)}\right)\\ &= L(b_i) + L_\mathrm{c}y_{i1} + L_\mathrm{e}(b_i) \end{aligned} \tag{72}$$

where

$$L_\mathrm{e}(b_i) = \ln\left(\frac{\sum_{\{u',u\}\Rightarrow b_i=+1}\alpha_{i-1}(u')\gamma_{i\_\mathrm{extr}}(u', u)\beta_i(u)}{\sum_{\{u',u\}\Rightarrow b_i=-1}\alpha_{i-1}(u')\gamma_{i\_\mathrm{extr}}(u', u)\beta_i(u)}\right) \tag{73}$$

is the so-called extrinsic LLR that is the estimation or soft decision that each decoder communicates to the other with respect to the information or message bit b_i. This extrinsic LLR contains the information provided by other bits related to b_i that are different for each decoder, as a consequence of the fact that the same bit b_i was interleaved and encoded in a different manner by each constituent encoder. This means that the bit b_i has been encoded by encoder E_1 of the turbo encoder together with a group of bits that is different from the one utilized by encoder E_2.

In each iteration, each constituent decoder communicates to the other decoder the extrinsic LLR

$$L_\mathrm{e}(b_i) = L(b_i/Y_1^n) - L(b_i) - L_\mathrm{c}y_{i1} \tag{74}$$

which is determined with the estimation of the message bit b_i done by the decoder, from which the *a priori* information and the channel information used to calculate that estimation are subtracted. This extrinsic information contains channel information that comes from an error event that is different for each received sequence, which in turn depends on which decoder is determining this information.

7.7 Turbo Decoding

So far the characteristics of the BCJR MAP algorithm usually implemented for the decoding of turbo codes have been introduced. This decoding algorithm is essentially an iterative decoding algorithm where each constituent decoder generates soft decisions or estimates of the message bits that are calculated using the channel information obtained from the sampled values of the received sequence, and the *a priori* information that has been provided by the other decoder in the previous iteration. The information passed is the extrinsic LLR that each decoder has determined in the previous iteration, which becomes the *a priori* information of the other decoder in the present iteration. This iterative procedure of interchanging information is such that, under certain conditions, estimates of the message bits are closer to the true values as the number of iterations increases.

A priori information is information that is neither related to the channel information (sampled values of the received sequence) nor related to the coding information (information provided by the trellis structure of the code). The first decoder does not have *a priori* information for the message bits in the first iteration, and so all the message bits in that circumstance are equally likely. This means that the *a priori* LLR $L(b_i)$ for the first decoder in the first iteration is equal to zero, as seen in Figure 7.3, corresponding to the situation for which $P(b_i = +1) = P(b_i = -1) = 0.5$.

The first decoder takes into account this initial *a priori* information and the channel information, which is essentially provided by the sampled values of the received sequence affected by the channel factor L_{c}. This sequence $L_{\mathrm{c}}\boldsymbol{Y}_1^{(1)}$ consists of the systematic or message bits, and of the parity check or redundancy bits. In most practical turbo codes, there is only one message or systematic bit, and also only one parity check bit, since puncturing is applied to both outputs of the two constituent encoders to make the whole turbo code be of code rate $k/n = 1/2$. The example below shows that the use of puncturing is solved in the decoding process by filling with zeros those positions that were punctured in the received sequence for each constituent decoder.

The first decoder utilizes its *a priori* information and its channel information to determine the first estimate or LLR $L_1^{(1)}(b_i/\boldsymbol{Y})$. Here the subscript identifies the decoder that generated the LLR, and the superscript identifies the order or number of the iteration. In this calculation the decoder needs to first determine the values of the coefficients $\gamma_i(u', u)$ and then to calculate the values of the coefficients $\alpha_{i-1}(u')$ and $\beta_i(u)$, which in turn are necessary for finally determining the LLR $L_1^{(1)}(b_i/\boldsymbol{Y})$. After determining these estimates, the decoder has to communicate to the other decoder the extrinsic information. Extrinsic information is the information that includes neither the *a priori* information utilized in the present calculation of $L_1^{(1)}(b_i/\boldsymbol{Y})$ nor the channel information of the message bit for which the extrinsic information is calculated.

Extrinsic information $L_{\mathrm{e}1}^{(1)}(b_i)$ is calculated by using expression (74). The second decoder is then able to perform its estimations with the available information. This decoder makes use of the received sequence $L_{\mathrm{c}}\boldsymbol{Y}_2^{(1)}$ containing the samples of interleaved message bits, and the samples of the corresponding parity check bits generated over the interleaved message bits sequence by the encoder E_2. If puncturing was applied, the punctured positions of the encoded sequence are filled with zeros in this sequence. The second decoder takes as its *a priori* information the extrinsic information $L_{e1}^{(1)}(b_i)$ of each message bit b_i, generated in the current iteration by the first decoder. However, and since interleaving has also been applied to the message bits, this extrinsic information should be reordered according to the interleaving rule, before being processed by the second decoder. If the forward operation of the interleaver is described by a function $I\{.\}$, then $L_2^{(1)}(b_i) = I\{L_{e1}^{(1)}(b_i)\}$. The second decoder takes the information $L_2^{(1)}(b_i)$ as its *a priori* information, and together with the channel information $L_c\boldsymbol{Y}_2^{(1)}$, it is able to determine the LLR $L_2^{(1)}(b_i/\boldsymbol{Y})$, and by applying equation (74), it can be used to finally calculate the extrinsic information $L_{e2}^{(1)}(b_i)$ to be communicated to the first decoder.

Since the extrinsic information $L_{e2}^{(1)}(b_i)$ corresponding to the message bit b_i is affected by interleaving, as the second decoder received the interleaved version, de-interleaving takes place for reordering this information, before transmitting it to the first decoder. The de-interleaving operation is defined by the operator $I^{-1}\{.\}$. The extrinsic information provided by the second decoder is reordered to be converted into the *a priori* information of the first decoder, so that $L_1^{(2)}(b_i) = I^{-1}\{L_{e2}^{(1)}(b_i)\}$. In this second iteration the first decoder utilizes again the same available channel information $L_c\boldsymbol{Y}_1^{(1)}$, but now the *a priori* information is different from zero, because this information is updated by the extrinsic information provided by the second decoder in the first iteration. In this way the first decoder produces improved estimates or LLRs of the message bits $L_1^{(2)}(b_i/\boldsymbol{Y})$. Figure 7.11 describes this iterative decoding procedure for turbo codes.

Example 7.3: This example shows a turbo code with 1/2-rate RSC constituent encoders, like those introduced as IIR systematic convolutional encoders in Chapter 6. The block diagram of one of these RSC encoders is shown in Figure 7.12.

The RSC encoder shown in Figure 7.12 has the trellis section shown in Figure 7.13.

For each constituent encoder of the turbo code, called encoders E_1 and E_2, the transfer function is of the form

$$\boldsymbol{G}(D) = \begin{bmatrix} 1 & \dfrac{1 + D + D^2}{1 + D^2} \end{bmatrix}$$

Each RSC code has minimum free distance $d_f = 5$. Since, as mentioned earlier, the polar format is a more convenient way of describing variables in a turbo code, the trellis of Figure 7.13 is depicted in Figure 7.14 with the assignment of inputs and outputs (variables x_1 and x_2) in polar format.

Each constituent RSC encoder of the turbo scheme has a trellis of the form as given in Figure 7.14. For brevity, these encoders are usually described in octal notation, for describing the connexions of the corresponding convolutional encoder. Thus, for this example, the convolutional encoder is described as RSC code (111, 101) or (7, 5). Puncturing is also utilized in this example to make the turbo code be of code rate $R_c = 1/2$. If all the outputs of these two encoders and the corresponding message information were transmitted, the resulting code rate would be $R_c = 1/3$. The puncturing rule adopted in this example is such that, alternately, one of the two redundancy outputs is transmitted, while the other is not transmitted. Message bits

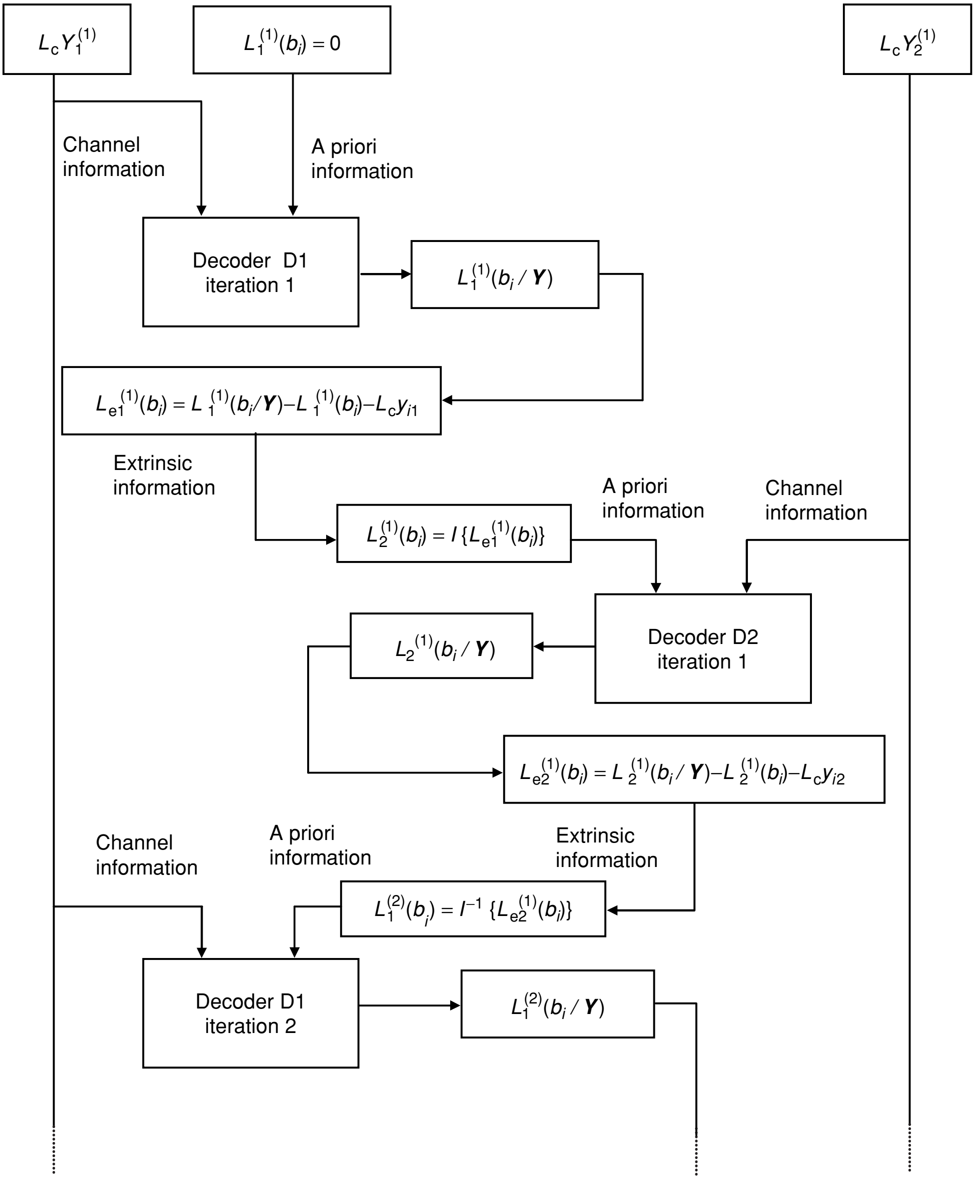

$L_cY_1^{(1)}$
$L_1^{(1)}(b_i) = 0$
$L_cY_2^{(1)}$
Channel information
A priori information
Decoder D1 iteration 1
$L_1^{(1)}(b_i/\mathbf{Y})$
$L_{e1}^{(1)}(b_i) = L_1^{(1)}(b_i/\mathbf{Y}) - L_1^{(1)}(b_i) - L_c y_{i1}$
Extrinsic information
A priori information
Channel information
$L_2^{(1)}(b_i) = I\{L_{e1}^{(1)}(b_i)\}$
Decoder D2 iteration 1
$L_2^{(1)}(b_i/\mathbf{Y})$
$L_{e2}^{(1)}(b_i) = L_2^{(1)}(b_i/\mathbf{Y}) - L_2^{(1)}(b_i) - L_c y_{i2}$
Channel information
A priori information
Extrinsic information
$L_1^{(2)}(b_i) = I^{-1}\{L_{e2}^{(1)}(b_i)\}$
Decoder D1 iteration 2
$L_1^{(2)}(b_i/\mathbf{Y})$

Figure 7.11 Iterative decoding of turbo codes

are not affected by the puncturing rule, as it is known that this can result in a reduction of the BER performance of the turbo code. The turbo code scheme is seen in Figure 7.15.

The puncturing rule adopted in this example of a turbo code is such that the systematic output $c_1^{(1)}$ is always transmitted, together with one of the two outputs $c_1^{(2)}$ or $c_2^{(2)}$ that are alternately transmitted through the channel. In this case the puncturing matrices for each encoder of the

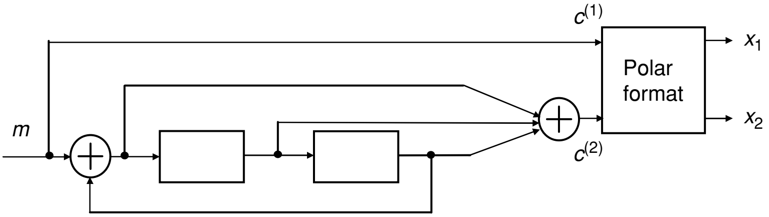


Figure 7.12 An RSC IIR systematic convolutional encoder of rate $k/n = 1/2$

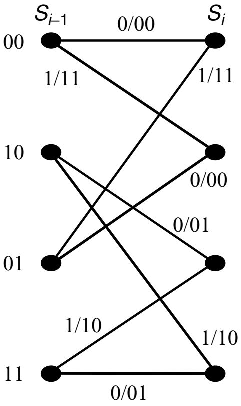


Figure 7.13 Trellis section of the RSC encoder

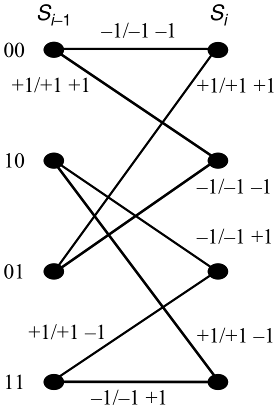


Figure 7.14 Trellis with transition information in polar format

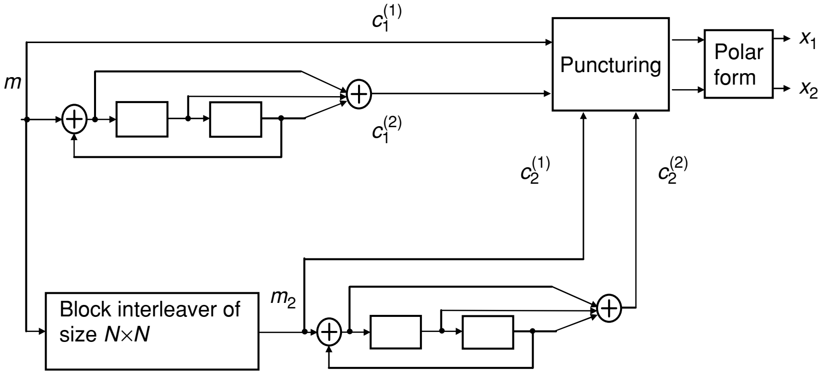


Figure 7.15 A 1/2 -rate turbo encoder

turbo code are respectively

$$\boldsymbol{P}_{p1} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

$$\boldsymbol{P}_{p2} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

The interleaver can be a random or block interleaver. There are of course other types of interleavers, but these two are the most common ones. In the case of Example 7.3, a block interleaver of size $N \times N = 4 \times 4$ is used. Input bit sequences consist of 16 message bits, and two of these 16 bits are determined so that the code sequence of the first encoder is terminated. A terminated code sequence in this example is one where the sequence starts at the all-zero state (00), and ends in the same state.

Generally, the termination of the sequence can be ensured for the first encoder, but not necessarily for the second encoder, since the interleaving procedure randomizes the input to this second encoder. The puncturing rule is such that the parity check bits at odd positions of the first encoder and the parity check bits at even positions of the second encoder are those bits that are being alternately transmitted together with the corresponding message or systematic bits.

This way there are 14 message bits, and two additional bits that are utilized to terminate the code sequence generated by the first encoder.

The interleaved sequence is obtained, in a block interleaver, by writing the input bits in row format, and reading them in column format. Thus, for instance, permutation of the block interleaver of Table 7.5 is the following:

$$I\{.\} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ 1 & 5 & 9 & 13 & 2 & 6 & 10 & 14 & 3 & 7 & 11 & 15 & 4 & 8 & 12 & 16 \end{pmatrix}$$

The inverse operation consists of arranging the interleaved bits in row format, and reading them in column format, as shown in Table 7.6.

Therefore both the interleaving permutation $I\{.\}$ and its corresponding inverse operation are the same. This is true for block interleavers, but it is in general not true for other types of

Table 7.5 The block interleaver of size 4 × 4

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

interleavers. Thus, if the operator $I\{.\}$ is applied to the interleaved sequence

$$\left(1\ 5\ 9\ 13\ 2\ 6\ 10\ 14\ 3\ 7\ 11\ 15\ 4\ 8\ 12\ 16\right)$$

then the de-interleaved sequence is

$$\left(1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15\ 16\right)$$

In the following, the operation of the encoder and the decoder of the turbo code of this example is described by using tables. Table 7.7 shows an example of a transmission where the input or message sequence, the parity check sequence generated by each encoder, the output sequence and the received sequences for each decoder are tabulated. The transmission is over an AWGN channel with $\sigma = 1.2$.

Table 7.8 shows the estimates or soft decisions $L_1^{(1)}(b_i/\boldsymbol{Y})$ for the 16 message bits, generated by the first decoder, which is denoted as Dec1, obtained by applying the BCJR MAP decoding algorithm.

As seen in Table 7.8, this first estimation of the first decoder contains four errors (marked in bold) with respect to the message bit sequence actually transmitted.

The extrinsic LLR $L_{\mathrm{e1}}^{(1)}(b_i)$ is also evaluated in this first iteration by the first decoder, and these values are listed in Table 7.9. Note that this information should be interleaved before being passed as *a priori* information to the second decoder.

The LLRs of Table 7.9 are the *a priori* values of the second decoder, which after interleaving and together with the corresponding channel information is able to determine the LLR $L_2^{(1)}(b_i/\boldsymbol{Y})$ for each bit. The second decoder determines the LLRs $L_2^{(1)}(b_i/\boldsymbol{Y})$ in the first iteration, and these values are listed in Table 7.10.

At this stage the output of the second decoder still contains six message bit estimates in error, as seen in Table 7.10. The extrinsic LLR $L_{\mathrm{e2}}^{(1)}(b_i)$ for each bit can be calculated by this second decoder as a function of the LLR $L_2^{(1)}(b_i/\boldsymbol{Y})$. These values are listed in Table 7.11.

Table 7.6 The block de-interleaver

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

Table 7.7 Input or message sequence, parity check sequence generated by each encoder, output sequence, and received sequences for each decoder, for the turbo code of Example 7.3

Input Sequence	Parity Cod1	Parity Cod2	Output Sequence	Received Sequence		Sequence for Dec1		Sequence for Dec2	
+1	+1	+1	+1 +1	+0.213	+0.364	+0.213	+0.364	+0.213	0.000
+1	−1	−1	+1 −1	−0.371	−0.351	−0.371	0.000	+0.539	−0.351
−1	+1	+1	−1 +1	+0.139	+1.818	+0.139	+1.818	+0.323	0.000
+1	−1	+1	+1 +1	+0.514	+1.646	+0.514	0.000	−1.701	+1.646
+1	+1	−1	+1 +1	+0.539	+0.388	+0.539	+0.388	−0.371	0.000
−1	−1	−1	−1 −1	−0.422	−2.587	−0.422	0.000	−0.422	−2.587
+1	+1	−1	+1 +1	+1.533	+ 0.267	+1.533	+0.267	+2.028	0.000
+1	−1	−1	+1 −1	+1.457	−1.678	+1.457	0.000	−0.175	−1.678
−1	+1	−1	−1 +1	+0.323	+1.103	+0.323	+1.103	+0.139	0.000
+1	−1	−1	+1 −1	+2.028	−0.170	+2.028	0.000	+1.533	−0.170
+1	+1	−1	+1 +1	−0.414	+3.560	−0.414	+3.560	−0.414	0.000
+1	+1	−1	+1 −1	+1.482	−1.003	+1.482	0.000	−0.862	−1.003
−1	+1	−1	−1 +1	−1.701	+0.893	−1.701	+0.893	+0.514	0.000
+1	+1	−1	+1 −1	−0.175	−1.306	−0.175	0.000	+1.457	−1.306
−1	−1	+1	−1 −1	−0.862	−2.049	−0.862	−2.049	+1.482	0.000
−1	−1	−1	−1 −1	−0.918	−0.492	−0.918	0.000	−0.918	−0.492

Table 7.8 LLR estimates performed by the first decoder, in the first iteration

Position of b_i	$L_1^{(1)}(b_i/\boldsymbol{Y})$	Estimated Bits	Input or Message Bits
1	0.786280	+1	+1
2	−0.547963	**−1**	**+1**
3	0.499806	**+1**	**−1**
4	0.689170	+1	+1
5	0.641047	+1	+1
6	−0.584477	−1	−1
7	2.082215	+1	+1
8	1.987796	+1	+1
9	0.295812	**+1**	**−1**
10	2.808172	+1	+1
11	−0.050543	**−1**	**+1**
12	1.831479	+1	+1
13	−1.987958	*−1*	−1
14	0.165691	+1	+1
15	−2.243063	*−1*	−1
16	−2.247578	−1	−1

Table 7.9 Extrinsic LLRs calculated by the first decoder, in the first iteration

$L_{e1}^{(1)}(b_i)$
0.490347
−0.032905
0.306964
−0.025372
−0.107293
0.001045
−0.046297
−0.036260
−0.152547
−0.008128
0.524750
−0.227359
0.374643
0.409279
−1.046018
−0.972807

This extrinsic information is de-interleaved to become the *a priori* information of the first decoder, in the second iteration. This decoder determines the LLRs $L_1^{(2)}(b_i/\boldsymbol{Y})$ based on this *a priori* information and the channel information, to give the estimates as listed in Table 7.12.

A comparison of these results with those of the first iteration in Table 7.8 shows that the estimates have improved, and the number of errors is reduced to 2.

Table 7.10 LLR estimations performed by the second decoder, in the first iteration

Position of b_i	$L_2^{(1)}(b_i/\boldsymbol{Y})$	Estimated Bits	Input Bits
1	0.768221	+1	+1
2	0.784769	+1	+1
3	−0.207484	−1	−1
4	−1.748824	**−1**	**+1**
5	−0.186654	**−1**	**+1**
6	−0.373027	−1	−1
7	2.820018	+1	+1
8	0.309994	+1	+1
9	0.473731	**+1**	**−1**
10	2.068479	+1	+1
11	0.012778	+1	+1
12	−2.510279	**−1**	**+1**
13	1.363218	**+1**	**−1**
14	2.492216	+1	+1
15	2.181300	**+1**	**−1**
16	−2.559998	−1	−1

Table 7.11 Extrinsic LLRs calculated by the second decoder, in the first iteration

$L_{e2}^{(1)}(b_i)$
−0.018059
0.143722
−0.503296
0.239134
0.361309
0.211449
0.011845
0.144303
−0.026075
−0.013736
0.063321
−0.267216
0.674047
0.504420
0.349821
−0.312420

The decoding procedure continues to alternate, with appropriate interleaving and de-interleaving, and so in each iteration the first decoder communicates to the second decoder extrinsic information that this second decoder uses as its *a priori* information, in order to produce extrinsic information that the first decoder takes as its *a priori* information in the next iteration. Extrinsic estimates calculated by the second decoder in the second iteration are listed in Table 7.13.

Table 7.12 LLR estimations performed by the first decoder, in the second iteration

Position of b_i	$L_1^{(2)}(b_i/Y)$	Estimated Bits	Input Bits
1	0.840464	+1	+1
2	−0.031009	**−1**	**+1**
3	0.044545	**+1**	**−1**
4	1.404301	+1	+1
5	0.846298	+1	+1
6	−0.406661	−1	−1
7	2.170691	+1	+1
8	2.562218	+1	+1
9	−0.343457	−1	−1
10	2.874132	+1	+1
11	0.431024	+1	+1
12	2.316240	+1	+1
13	−1.982660	−1	−1
14	0.550832	+1	+1
15	−2.811396	−1	−1
16	−2.825644	−1	−1

Table 7.13 Extrinsic LLRs calculated by the second decoder, in the second iteration

$L_{e1}^{(2)}(b_i)$
0.064426
0.173966
−0.588380
0.189984
0.529265
−0.034277
0.052512
0.233310
−0.130874
0.026234
0.111812
−0.308991
0.756415
0.561614
0.380584
−0.336884

This is again converted by properly de-interleaving of the values into the *a priori* information of the first decoder in the iteration number 3. The estimates at this stage of the decoding obtained by the first decoder are given in Table 7.14.

In this third iteration the first decoder has been able to correctly decode the message sequence, and further iterations will not change this result, although individual bit estimates may continue to improve. However, the estimates increase magnitude as the number of iterations increases, and this sometimes brings overflow or underflow problems in practical calculations of these quantities. This is the reason for the design of logarithmic versions of the iterative decoding algorithms for turbo codes and other iteratively decoded codes, since logarithmic operations greatly reduce these calculation difficulties.

7.7.1 Initial Conditions of Coefficients $\alpha_{i-1}(u')$ and $\beta_i(u)$

As pointed out in Section 7.5, coefficients $\alpha_{i-1}(u')$ and $\beta_i(u)$ are obtained by forward and backward recursions respectively, and so it is necessary to set the initial and contour conditions for these calculations.

If a code sequence generated by one of the constituent encoders is terminated, usually at the all-zero state, it is already known at the receiver that the initial state and the final state of the decoded sequence should be the same state, as was the case in the previous example, where it was known that the decoded sequence should start and end at the all-zero state. In the decoder, knowledge of the initial state can be taken into account by setting $\alpha_0(0) = 1$ and $\alpha_0(u) = 0, u \neq 0$. For $i = n$, and in the case of a terminated sequence, knowledge of the ending state can also be taken into account by setting the contour conditions $\beta_n(0) = 1$ and $\beta_n(u) = 0, u \neq 0$. The sequence encoded by the second encoder is not usually terminated. Therefore

Table 7.14 LLR estimations performed by the first decoder, in the third iteration

Position of b_i	$L_1^{(3)}(b_i/Y)$	Estimated Bits	Input Bits
1	0.981615	+1	+1
2	0.318157	+1	+1
3	−0.289548	−1	−1
4	1.543603	+1	+1
5	0.982070	+1	+1
6	−0.716361	−1	−1
7	2.273836	+1	+1
8	2.660691	+1	+1
9	−0.554936	−1	−1
10	2.969750	+1	+1
11	0.662567	+1	+1
12	2.407738	+1	+1
13	−2.161687	−1	−1
14	0.773773	+1	+1
15	−2.923458	−1	−1
16	−2.934754	−1	−1

this latter case can be taken into account by setting the contour conditions $\beta_n(u) = 1 \ \forall u$. This means that all the ending states in the corresponding trellis are equally likely.

7.8 Construction Methods for Turbo Codes

7.8.1 Interleavers

Interleaving is a widely used technique in digital communication and storage systems. An interleaver takes a given sequence of symbols and permutes their positions, arranging them in a different temporal order. The basic goal of an interleaver is to randomize the data sequence. When used against burst errors, interleavers are designed to convert error patterns that contain long sequences of serial erroneous data into a more random error pattern, thus distributing errors among many code vectors [3, 7]. Burst errors are characteristic of some channels, like the wireless channel, and they also occur in concatenated codes, where an inner decoder overloaded with errors can pass a burst of errors to the outer decoder.

In general, data interleavers can be classified into block, convolutional, random, and linear interleavers.

In a block interleaver, data are first written in row format in a permutation matrix, and then read in column format. A pseudo-random interleaver is a variation of a block interleaver where data are stored in a register at positions that are determined randomly. Convolutional interleavers are characterized by a shift of the data, usually applied in a fixed and cumulative way. Linear interleavers are block interleavers where the data positions are altered by following a linear law.

7.8.2 Block Interleavers

As explained previously, block interleavers consist of a matrix array of size $M_{\mathrm{I}} \times N_{\mathrm{I}}$ where data are usually written in row format and then read in column format. Filling of all the positions in the matrix is required, and this results in a delay of $M_{\mathrm{I}} \times N_{\mathrm{I}}$ intervals. The operation can be equivalently performed by first writing data in column format and then by reading data in row format. The block interleaver introduced in Example 7.3 as part of a turbo coding scheme is an example of a block interleaver.

A block interleaver of size $M_{\mathrm{I}} \times N_{\mathrm{I}}$ separates the symbols of any burst error pattern of length less than M_{I} by at least N_{I} symbols [3]. If, for example, a burst of three consecutive errors in the following sequence is written by columns into a 4 x 4 de-interleaver (1 5 9 13 **2 6 10** 14 3 7 11 15 4 8 12 16), then these errors will be separated by at least four intervals

1	**2**	3	4
5	**6**	7	8
9	**10**	11	12
13	14	15	16

The de-interleaved sequence in this case is

(1 **2** 3 4 5 **6** 7 8 9 **10** 11 12 13 14 15 16)

which confirms that the errors are separated by four positions.

In a given application of error-control coding, a block interleaver is selected to have a number of rows that should be ideally larger than the longest burst of errors expected, and in practice at least as large as the length of most expected bursts. The other parameter of a block interleaver is the number of columns of the permutation matrix, N_{I}, which is normally selected to be equal to or larger than the block or decoding length of the code that is used. In this way, a burst of N_{I} errors will produce only one error per code vector. For error-correcting codes able to correct any error pattern of size t or less, the value of N_{I} can be set to be larger than the expected burst length divided by t.

7.8.3 Convolutional Interleavers

A convolutional interleaver is formed with a set of N registers that are multiplexed in such a way that each register stores L symbols more than the previous register.

The order zero register does not contain delay and it consists of the direct transmission of the corresponding symbol. The multiplexers commute through the different register outputs and take out the 'oldest' symbol stored in each register, while another symbol is input to that register at the same time [7]. The operation of convolutional interleaving is shown in Figure 7.16.

Convolutional interleavers are also known as multiplexed interleavers. The interleaver operation can be properly described by a permutation rule defined over a set of N integer numbers

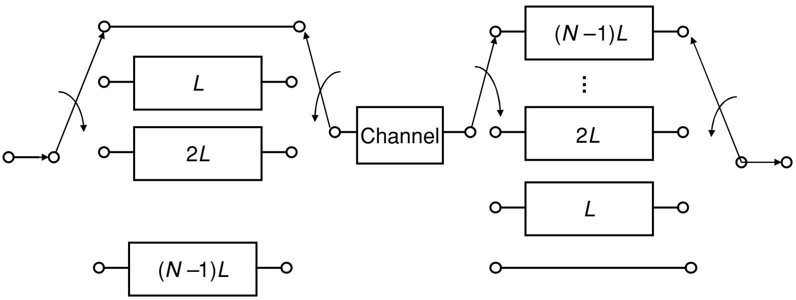


Figure 7.16 A convolutional interleaver

$Z_N = \{0, 1, \ldots, N-1\}$:

$$\begin{pmatrix} 0 & 1 & \cdots & N-1 \\ \pi\{0\} & \pi\{1\} & & \pi\{N-1\} \end{pmatrix} \tag{75}$$

Expression (75) corresponds to the permutation rule of a given interleaver if the following operation performed over the set $\{\pi\{0\}, \pi\{1\}, \ldots, \pi\{N-1\}\}$:

$$\{\pi\{0\}, \pi\{1\}, \ldots, \pi\{N-1\}\} \quad \text{modulo } N$$

results in the set of integer numbers $Z_N = \{0, 1, \ldots, N-1\}$.

7.8.4 Random Interleavers

Random interleavers are constructed as block interleavers where the data positions are determined randomly. A pseudo-random generator can be utilized for constructing these interleavers. The memory requirement of a random interleaver is of size $M_{\mathrm{I}} \times N_{\mathrm{I}}$ symbols, but and since there is the practical need of having two interleavers, one for being written (filled) and another one for being read (emptied), the actual memory requirement is then $2M_{\mathrm{I}} \times N_{\mathrm{I}}$ symbols.

In a turbo coding scheme, the interleaver plays a very important role. In general, the BER performance is improved if the length of the interleaver that is part of the scheme is increased. Either block or random interleavers can be used in a turbo code. In general, it is shown in [2] that block interleavers perform better than random interleavers if the size $M_{\mathrm{I}} \times N_{\mathrm{I}}$ of the interleaver is small, and random interleavers perform better than block interleavers when the size $M_{\mathrm{I}} \times N_{\mathrm{I}}$ of the interleaver is medium or large. The BER performance of a turbo code with large random interleavers is significantly better than that of a turbo code with block interleavers of the same size. This can be seen in Figure 7.17. However, the larger the interleaver, the larger is the delay in the system. Sometimes, and depending on the application, the delay occasioned by a turbo code, or more precisely, by its interleaver, can be unacceptable for a given practical application, and so in spite of their impressive BER performance, turbo codes with large random interleavers cannot be used. This is the case for instance in audio applications, where sometimes the delay of a turbo code cannot be tolerated. If the delay is acceptable in a particular application, large random interleavers allow the turbo coding BER performance to be close to the Shannon limit. It can be concluded that both families of turbo codes, those

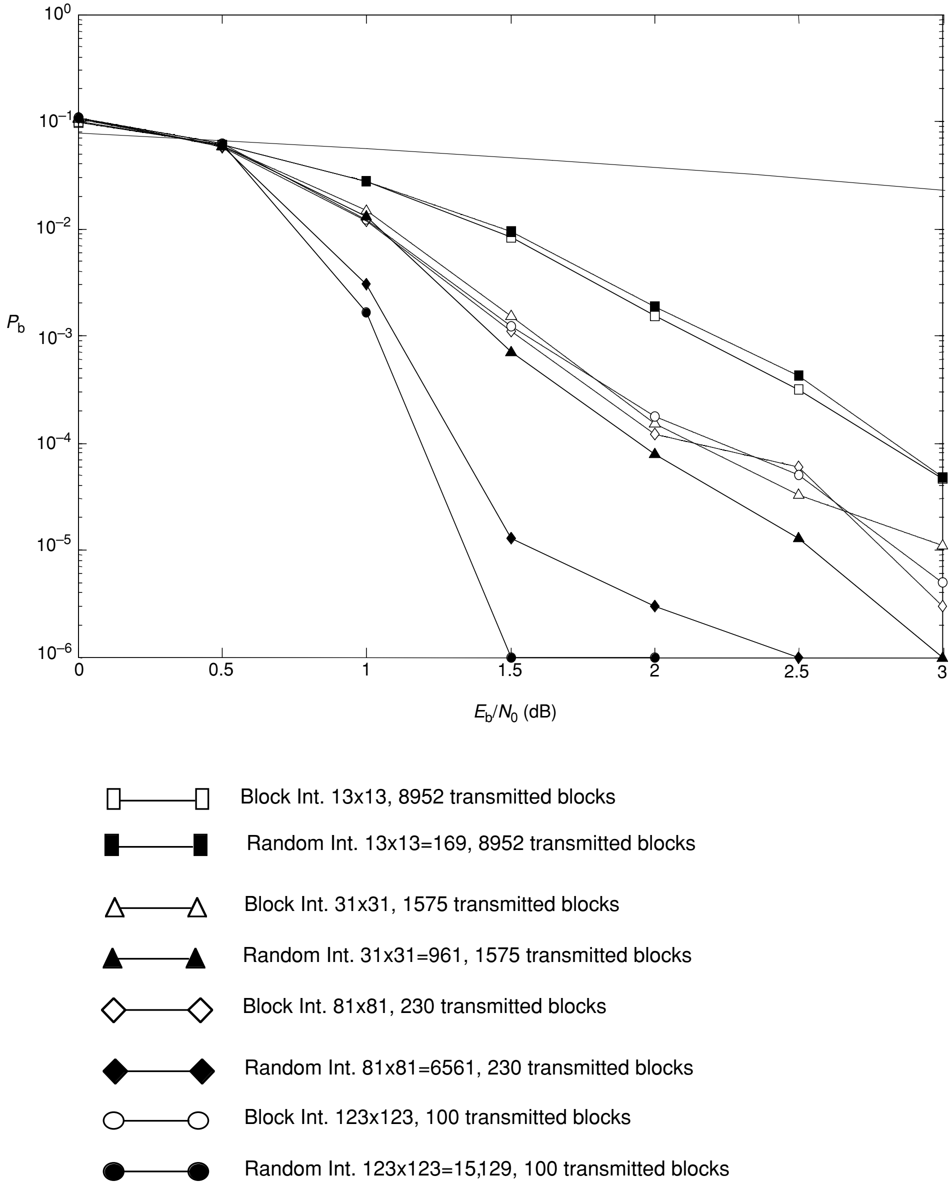


Figure 7.17 BER performance of a turbo code as a function of the type and size of the interleaver

constructed using small block interleavers, and those constructed with considerably larger random interleavers, can be used in practice, depending on the application. It has also been shown in [2] that square block interleavers are better than rectangular block interleavers, and that odd dimension interleavers are also better than even dimension interleavers. Therefore, the best selection of a block interleaver is obtained by using $M_I = N_I$, and by making M_I and N_I be odd numbers.

7.8.5 Linear Interleavers

Another kind of interleaver also utilized in turbo coding schemes is the linear interleaver. One interesting characteristic of this interleaver is that it has a mathematical expression for generating the interleaving permutation, which avoids the need to store all the structure of the interleaver, usually in the form of a big memory allocation, which is the case for random or block interleavers.

In general, turbo codes have an impressive BER performance in the so-called waterfall region, which is where the curve of P_{be} versus E_{b}/N_0 falls steeply. There is also another characteristic region of the turbo code BER performance curve, which is known as the error floor region. This floor region arises because of the degradation in the BER performance caused by the relatively small minimum distance of a turbo code. This floor region is also a consequence of the minimum distance of each of the constituent codes [2], so that the smaller the minimum distance of the constituent codes, the higher is the BER at which the floor effect starts appearing. In addition, the type and size of the interleaver plays an important role in determining the minimum distance of the turbo code.

One solution for reducing the floor effect is the use of multiple turbo codes (MTC). These codes consist of a modification of the classic structure of a turbo scheme, involving usually one interleaver and two constituent codes. In the general structure of an MTC, there are $J_{\mathrm{MTC}} > 2$ constituent convolutional codes and $J_{\mathrm{MTC}} - 1$ interleavers, and the use of linear interleavers in an MTC scheme can be very effective.

A linear interleaver of length L_{I} can be described by the following permutation rule:

$$\begin{pmatrix} 0 & 1 & \cdots & L_{\mathrm{I}}-1 \\ \pi\{0\} & \pi\{1\} & & \pi\{L_{\mathrm{I}}-1\} \end{pmatrix} \tag{76}$$

where

$$\pi(i) = (ip_{\mathrm{MTC}} + s_{\mathrm{MTC}}) \operatorname{mod} L_{\mathrm{I}} \tag{77}$$

In this expression p_{MTC}, $0 \leq p_{\mathrm{MTC}} \leq L_{\mathrm{I}} - 1$, is a parameter called the angular coefficient, and s_{MTC}, $0 \leq s_{\mathrm{MTC}} \leq L_{\mathrm{I}} - 1$, is a parameter called the linear shift. It is required that the highest common factor (HCF) between p_{MTC} and L_{I} be HCF $(p_{\mathrm{MTC}}, L_{\mathrm{I}}) = 1$. This definition is extracted from [10], where the authors introduce an analysis of linear interleavers with regard to the minimum distance of an MTC. A conclusion obtained in that paper is that whereas the minimum distance of a traditional turbo code, constructed with two constituent convolutional codes and one interleaver, increases logarithmically with the size of the interleaver, the minimum distance of an MTC exhibits a higher increase that is of order $L_{\mathrm{I}}^{(J_{\mathrm{MTC}}-2)/J_{\mathrm{MTC}}}$. Therefore, linear interleavers appear to be easily constructed interleavers which also provide the turbo code with enhanced minimum distance properties, comparable to and even better than those of turbo codes with other types of interleavers, such as dithered relative prime (DRP) interleavers [11] or S-random interleavers [12].

7.8.6 Code Concatenation Methods

Concatenation of codes [8] is a very useful technique that leads to the construction of very efficient codes by using two or more constituent codes of relatively small size and complexity. Thus, a big, powerful code with high BER performance, but of impractical complexity, can be constructed in an equivalent concatenated form by combining two or more constituent

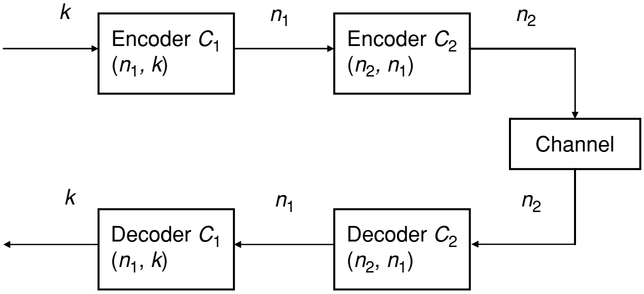


Figure 7.18 Serial concatenation of codes

codes to provide the same performance at a lower cost in terms of complexity. The reduced complexity is important especially for the decoding of these codes, which can take advantage of the combined structure of a concatenated code. Decoding is done by combining two or more relatively low complexity decoders, thus effectively decomposing the problem of the decoding of a big code. If these partial decoders properly share the decoded information, by using an iterative technique, for example, then there need be no loss in performance.

There are essentially two ways of concatenating codes: traditionally, by using the so-called serial concatenation, and more recently, by using the parallel concatenated structure of the first turbo coding schemes. Both concatenation techniques allow the use of iterative decoding.

7.8.6.1 Serial concatenation

Serial concatenation of codes was introduced by David Forney [8]. In a serial concatenated code a message block of k elements is first encoded by a code $C_1(n_1, k)$, normally called the outer code, which generates a code vector of n_1 elements that are then encoded by a second code $C_2(n_2, n_1)$, usually called the inner code, which generates a code vector of n_2 elements. A block diagram of a serial concatenated code is seen in Figure 7.18. The decoding of the concatenated code operates in two stages: first performing the decoding of the inner code C_2 and then the decoding of the outer code C_1. The decoding complexity decomposed into these two decoders is much lower than that of the direct decoding of the whole code equivalent to the concatenated code, and the error-control efficiency can be the same if the two decoders interactively share their decoded information, as in turbo decoding.

An example of serial concatenation of codes, where a convolutional interleaver would be implemented in between the two constituent encoders, is the coding scheme for the compact disk, already described in Chapter 5.

7.8.6.2 Parallel concatenation

Parallel concatenation of codes was introduced by Berrou, Glavieux and Thitimajshima [1] as an efficient technique suitable for turbo decoding. Iterative decoding and parallel concatenation of codes are two of the most relevant concepts introduced in the construction of a turbo code, which have a strong influence on the impressive BER performance of these codes.

A simple structure for a parallel concatenated encoder is seen in Figure 7.19, which is the encoder of a turbo code of code rate $R_c = 1/3$.

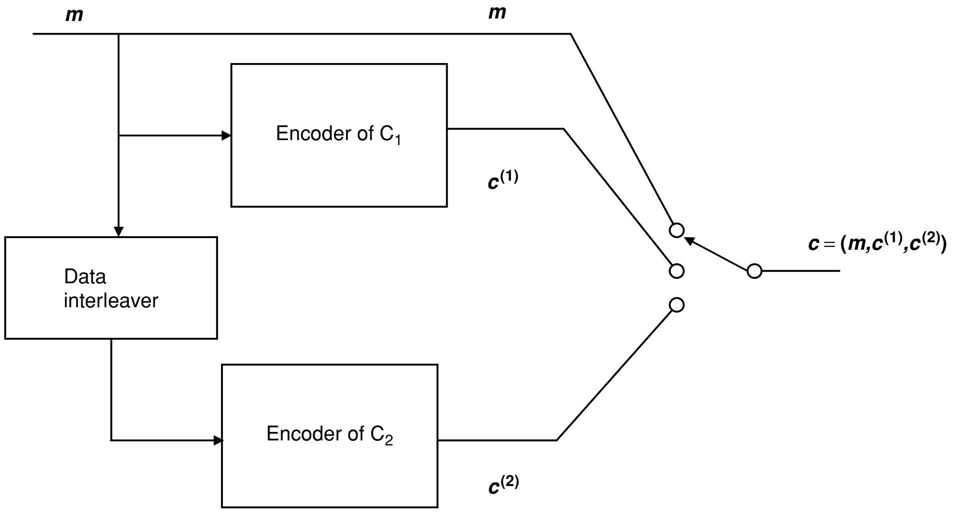


Figure 7.19 A parallel concatenated code

A block or sequence of message bits $\boldsymbol{m}$ is input to the encoder of one of the constituent codes C_1, which in the case of convolutional codes is an FSSM, generating an output sequence $\boldsymbol{c}_1$. In addition, the same input sequence is first interleaved, and then input to the encoder of the second code C_2, which in the case of convolutional codes is also an FSSM, generating an output sequence $\boldsymbol{c}_2$. The output of both encoders, $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$, are multiplexed with the input $\boldsymbol{m}$ to form the output or code sequence $\boldsymbol{c}$, so that the concatenated code is of code rate $R_c = 1/3$ if all the multiplexed sequences are of the same length.

Puncturing can be applied to the encoder outputs, as described in Chapter 6, to modify the code rate of the concatenated code. Both puncturing and parallel concatenation are suitable techniques for the construction of a turbo code. Under certain conditions, the use of more than two constituent codes, as in MTC schemes described previously, can lead to better BER performance.

Example 7.4: Determine the BER performance curve of P_{be} versus E_b/N_0, for a turbo code of rate $R_c = 1/2$ constructed using ½-rate RSC (5, 7) encoders, in a parallel concatenation like that shown in Figure 7.20. Make use of the puncturing procedure of Example 7.3, such that the systematic bit is always transmitted, together with one of the outputs $c_1^{(2)}$ or $c_2^{(2)}$ alternately. Use a random interleaver of size $L_I = N \times N = 100 \times 100 = 10,000$.

The simulation shown in Figure 7.21 was done by transmitting 400 blocks of 10,000 bits of information each. This curve shows the three main regions of the BER performance of a turbo code [13]. In the first region, the parameter E_b/N_0 is very low, and the code produces only a degradation of the average bit energy that results in a BER performance that is worse than that of uncoded transmission (also seen in Figure 7.21). The next region is the so-called waterfall region where the BER performance of the turbo code is impressively good, falling steeply over a small middle range of E_b/N_0. Finally there is the so-called floor region, characterized by the flattening of the curve, where the parameter E_b/N_0 is relatively high, but the error rate decreases only slowly.

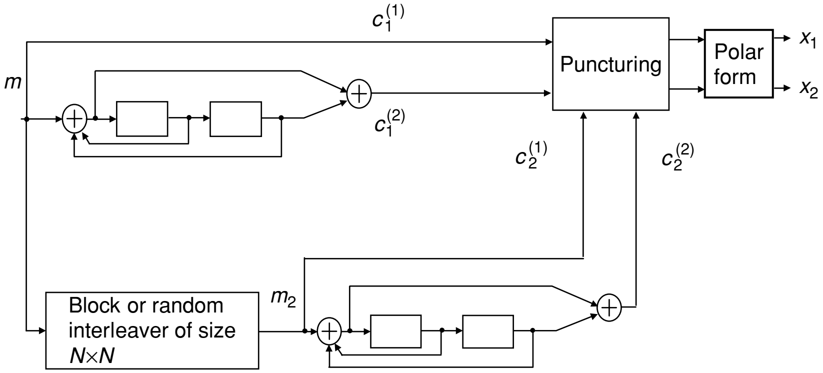


Figure 7.20 Turbo encoder with a block or random interleaver

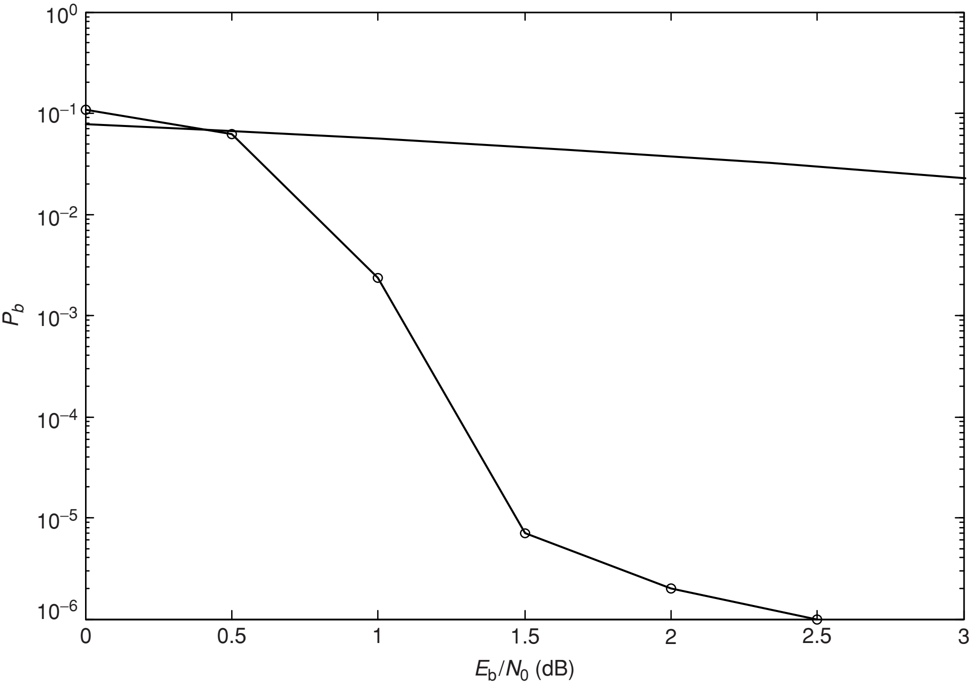


Figure 7.21 BER performance curve of P_{be} versus E_{b}/N_0, for a turbo code of rate $R_{\mathrm{c}} = 1/2$, ½-rate RSC (5, 7) encoders with puncturing, random interleaver of size $L_{\mathrm{I}} = N \times N = 10,000$, decoded by the LOG MAP BCJR algorithm with eight iterations, in comparison with uncoded transmission

7.8.7 Turbo Code Performance as a Function of Size and Type of Interleaver

The BER performance of binary turbo codes is evaluated as a function of the type and size of the interleaver. Simulations are done for the standard structure, involving the use of two $1/2$-rate RSC (5, 7) encoders in parallel concatenation with puncturing, as in Example 7.3, to control the code rate, and either a block or a random interleaver, as shown in Figure 7.20. This turbo scheme is decoded by using the LOG MAP BCJR algorithm with eight iterations. Simulations are such that, depending on the size of the interleaver, the total number of transmitted bits is approximately equal to 1.5 Mbits in each case.

7.9 Other Decoding Algorithms for Turbo Codes

The decoding algorithm already introduced in this chapter for decoding turbo codes is the MAP BCJR algorithm. This algorithm is in general of high complexity, and, on the other hand, sums and products involved in its calculation can lead to underflow and overflow problems in practical implementations. These calculations also require considerable amount of memory to store all the values, until a decoding decision is taken. A logarithmic version of this algorithm appears to be a solution for many of the above calculation problems that the original version of the BCJR algorithm faces [2, 9]. The basic idea is that by converting calculations into their logarithmic form, products convert into sums. The logarithm of a sum of two or more terms seems to be a new complication, but this operation is solved by using the following equation:

$$\ln(e^A + e^B) = \max(A, B) + \ln(1 + e^{-|A-B|}) = \max(A, B) + f_c(|A - B|) \tag{78}$$

where $f_c(|A - B|)$ is a correction factor that can be either exactly calculated or, in practical implementations of this algorithm, obtained from a look-up table.

The logarithmic version of the MAP BCJR algorithm greatly reduces the overflow and underflow effects in its application. This logarithmic version is known as the LOG MAP BCJR algorithm. As explained above, the correction term in equation (78) can be appropriately taken from a look-up table. Another and even simpler version of the LOG MAP BCJR algorithm is the so-called MAX LOG MAP BCJR algorithm, in which the correction term is omitted in the calculation, and the equation (78) is used by simply evaluating the MAX value of the involved quantities. A detailed analysis can be found in [2], where it is shown that the MAX LOG MAP BCJR algorithm and the SOVA (soft-output Viterbi algorithm) are those of minimal complexity, but with a level of degradation in BER performance with respect to the LOG MAP BCJR algorithm. Therefore, as usual, decoding complexity is in a trade-off with BER performance. The degradation in BER performance is around 0.6 dB between the best decoder, the LOG MAP BCJR decoder, and the worst, the SOVA decoder.

7.10 EXIT Charts for Turbo Codes

Stephan Ten Brink [13, 14] introduced a very useful tool for the analysis of iterative decoders, which is known as the extrinsic information transfer (EXIT) chart. This tool allows us to analyse the iteration process in decoders that utilize soft-input–soft-output estimates that are passed form one decoder to the other. This process of interchanging information is represented in a graphical chart that depicts the transfer of mutual information between the *a priori* information that is input to these decoders and the extrinsic information that is generated by these decoders.

The EXIT chart emerged as a development of another technique, known as density evolution, also applied to iteratively decoded coding schemes [28]. Both tools are suitable for the analysis of iterative decoding, but EXIT charts involve less-complex calculations and appear to be easier to use [13–20].

EXIT charts analyse the transfer between the *a priori* information, which in the case of the LOG MAP BCJR algorithm described in Section 7.6 is the LLR $L(b_i)$ and acts as the input of the decoder, and the extrinsic information generated by the decoder, which in the case of the LOG MAP BCJR algorithm is denoted as the LLR $L_e(b_i)$. Both the *a priori* and the extrinsic information are measured by using the mutual information between these quantities and the information in the systematic or message bits. These quantities are related by expression (74). Following the notation introduced by S. Ten Brink [13, 14], expression (74) can be written as

$$L_e(b_i) = L(b_i/Y_1^n) - L(b_i) - L_c y_{i1} = E_{i1} = D_{i1} - A_{i1} - Y_{i1} \qquad (79)$$

where A identifies the *a priori* information, E identifies the extrinsic information and Y represents the channel information. All these quantities are LLRs like those described by expression (59). The vector $\boldsymbol{E}$ will represent a set of values E_i and the same notation will be used for the rest of the quantities involved.

7.10.1 Introduction to EXIT Charts

As described in previous sections, the BER performance curve of P_{be} versus E_b/N_0 of a turbo code basically shows three main regions:

- A first region at low values of E_b/N_0, where iterative decoding performs worse than uncoded transmission, even for a large number of iterations.
- A second region at low to medium values of E_b/N_0, where iterative decoding is extremely effective. This is the waterfall region, where the performance increases, but not linearly, with an increase in the number of iterations.
- A third region at higher value of E_b/N_0, the error floor, where decoding converges in few iterations, but performance increases only slowly as E_b/N_0 increases.

The EXIT chart is an especially good tool for the analysis of the waterfall region, and also illustrates the behaviour of the code in the other two regions. The chart is constructed from the mutual information between the *a priori* information and the message bit information on the one hand, and the mutual information between the extrinsic information and the message bit information on the other. The *a priori* information and the extrinsic information are the input and output measures, respectively that a given LOG MAP decoder utilizes for iterative decoding.

As mentioned earlier, the performance of iterative decoding is enhanced by increasing the number of iterations, but the enhancement is not linear with this increase. It is found that the relative improvement in performance reaches a practical limit beyond which an increase in the number of iterations does not result in a significant increase in coding gain. This fact is also evident in the EXIT chart, a tool that can be used to determine the number of iterations considered sufficient to achieve a given BER performance with a particular turbo coding scheme.

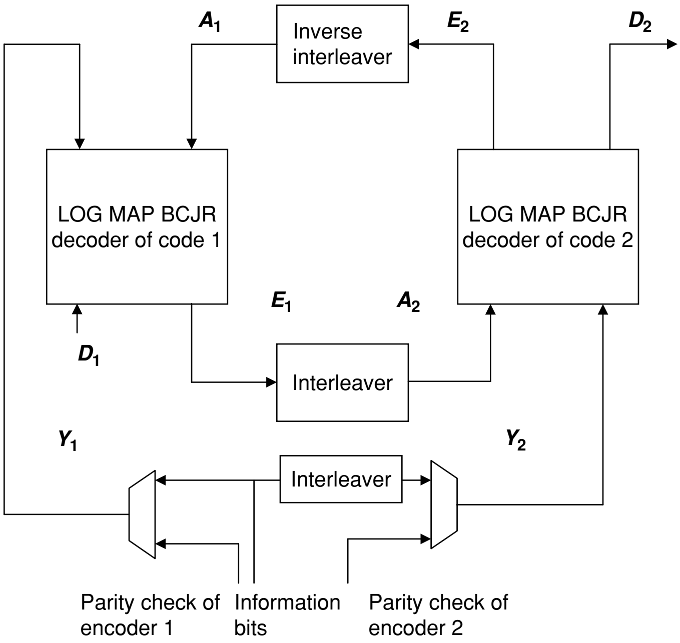

A_1
Inverse interleaver
E_2
D_2
LOG MAP BCJR decoder of code 1
LOG MAP BCJR decoder of code 2
E_1
A_2
D_1
Interleaver
Y_1
Y_2
Interleaver
Parity check of encoder 1
Information bits
Parity check of encoder 2

Figure 7.22 *A priori*, extrinsic and channel informations managed by a LOG MAP BCJR decoder

7.10.2 Construction of the EXIT Chart

In this section the EXIT chart of the turbo code of Example 7.4, when decoded using the LOG MAP BCJR decoder, will be constructed. This turbo code consists of two ½-rate RSC (5, 7) encoders, with a random interleaver of size $N \times N = 10,000$, and output puncturing applied to make the code rate be $R_c = 1/2$. As in Example 7.3, the first encoder generates sequences terminating in the all-zero state.

Figure 7.22 illustrates the operation of the LOG MAP BCJR decoder for this turbo code at an intermediate iteration.

The first decoder makes use of the *a priori* information, together with the channel information of the systematic and parity check bits generated by the first encoder, to generate a vector $\boldsymbol{D}_1$ of estimates or soft decisions with components $D_i = L(b_i/\boldsymbol{Y}_1)$. This vector of estimates is used by the first decoder to produce a vector of extrinsic information, which is properly interleaved to become the *a priori* information vector of the current iteration for the second decoder. This extrinsic information vector

$$\boldsymbol{E}_1 = \boldsymbol{D}_1 - \boldsymbol{A}_1 - \boldsymbol{Y}_1$$

with components

$$E_{i1} = L_{e1}(b_i) = L_1(b_i/\boldsymbol{Y}_1) - L_1(b_i) - L_c y_{i1} = D_{i1} - A_{i1} - Y_{i1}$$

is generated by the first decoder as shown in Figure 7.11.

The second decoder processes its input information, which consists of the channel information $\boldsymbol{Y}_2$ formed from the samples of the received message or systematic bits, properly interleaved, and the samples of the received parity check bits that have been generated by the second encoder, together with the *a priori* information $\boldsymbol{A}_2$ received from the first decoder, and generates an extrinsic information vector

$$\boldsymbol{E}_2 = \boldsymbol{D}_2 - \boldsymbol{A}_2 - \boldsymbol{Y}_2$$

whose components are

$$E_{i2} = L_{\mathrm{e}2}(b_i) = L_2(b_i/Y_2) - L_2(b_i) - L_{\mathrm{c}} y_{i2} = D_{i2} - A_{i2} - Y_{i2}$$

as indicated in Figure 7.11. This extrinsic information becomes now the *a priori* information vector $\boldsymbol{A}_1$ for the first decoder, in the next iteration.

As pointed out earlier, the EXIT chart is a representation of the relationship between two mutual informations: the mutual information between the *a priori* information and the message information, and the mutual information between the extrinsic information and the message information.

For the AWGN channel, the input and output variables are considered to be random variables that are related by

$$Y = X + n$$

or

$$y = x + n \tag{80}$$

where X denotes a random variable that represents the message bits x that can take one of the two possible values $x = +1$ or $x = -1$, Y denotes the random variable that results from the detected samples of the transmitted information and n denotes the noise random variable. The function that characterizes this channel is the Gaussian probability density function already introduced and described by equations (56) and (57). Then equation (58) determines the logarithmic value of the conditional probability [14]:

$$L(y/x) = 2\frac{E_{\mathrm{b}}}{\sigma^2} y = \frac{2}{\sigma^2} y \tag{81}$$

Here the average bit energy is equal to $E_{\mathrm{b}} = 1$ for the normalized polar format adopted in this case. This expression allows us to determine the random variable Y that represents the logarithmic values of the samples of the received signal that are channel observations of the form

$$Y = \frac{2}{\sigma^2} y = \frac{2}{\sigma^2}(x + n) \tag{82}$$

This can also be written as

$$Y = \frac{2}{\sigma^2} y = \mu_Y x + n_Y \tag{83}$$

where

$$\mu_{\mathrm{Y}} = 2/\sigma^2 \tag{84}$$

and n_Y is a random variable with zero mean value and variance

$$\sigma_Y^2 = 4/\sigma^2 \tag{85}$$

The mean value and the variance of this random variable are related by

$$\mu_Y = \sigma_Y^2/2 \tag{86}$$

This relationship will be useful in the construction of the EXIT chart. In a turbo code, both decoders utilize the same decoding algorithm, and often also the same code, and therefore the same trellis or code information. However, there is in general a slight difference between the first and the second constituent codes in a turbo scheme, which is that the first code usually generates terminated sequences, whereas this is not necessarily the case for the sequences generated by the second constituent code. This difference was the reason for the setting of slightly different initial or contour conditions in the decoding algorithm used, the LOG MAP BCJR algorithm, as pointed out in previous sections. However, if the code sequences are long enough, then the terminated sequence effect can be neglected, so that both decoders operate in virtually the same way, and it will be sufficient to perform the EXIT chart analysis for only one of the decoders, for the first decoder, for instance.

7.10.3 Extrinsic Transfer Characteristics of the Constituent Decoders

EXIT charts are generated for the decoding operation of one of the constituent decoders of a turbo code. Since the mathematical complexity of the LOG MAP BCJR decoding algorithm is very high, then the analysis for EXIT charts is derived by Monte Carlo simulation over the parameters of interest.

Monte Carlo simulations of the operation of a LOG MAP BCJR decoder allow us to affirm that the values of the *a priori* information A are independent and uncorrelated from the observations or sampled values of the channel Y. On the other hand, the probability density function, or more exactly, the histogram of the values of the extrinsic information E generated by the LOG MAP BCJR decoder, is a Gaussian histogram (equivalent for the continuous case to a Gaussian probability density function). It is also true that these values become the *a priori* values for the next iteration, so that a similar conclusion can be stated for the probability density function of the *a priori* information.

The following figures show non-normalized histograms of the extrinsic information values generated by Monte Carlo simulation of the operation of a LOG MAP BCJR decoder for the turbo code of Example 7.4. Figure 7.23 shows the non-normalized histogram of extrinsic estimates for the input ‘0’, which is proportional to the probability density function denoted as $p_E(\xi/x = -1)$, and Figure 7.24 shows the non-normalized histogram of extrinsic estimates for the input ‘1’, which is proportional to the probability density function denoted as $p_E(\xi/x = +1)$. Histograms are shown for different values of the parameter $E_{\mathrm{b}}/N_0 = 0.5$, 1.0, 1.5 and 2.0 dB.

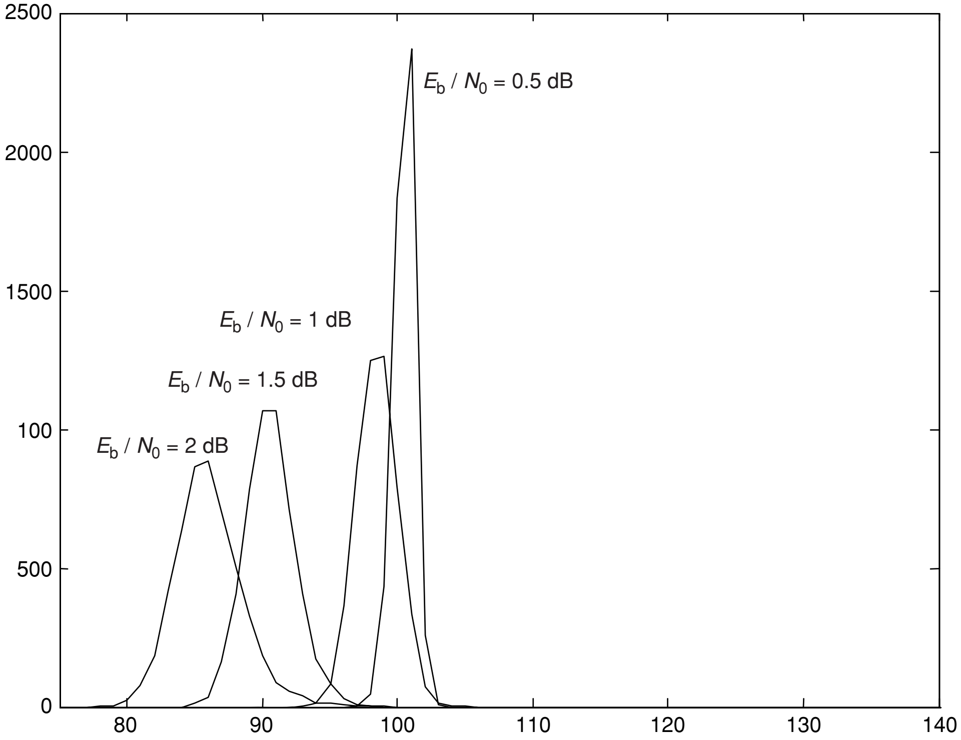


Figure 7.23 Non-normalized histogram for extrinsic estimates for the input symbol ‘0’ of a LOG MAP BCJR decoder of a turbo code, as a function of E_b/N_0

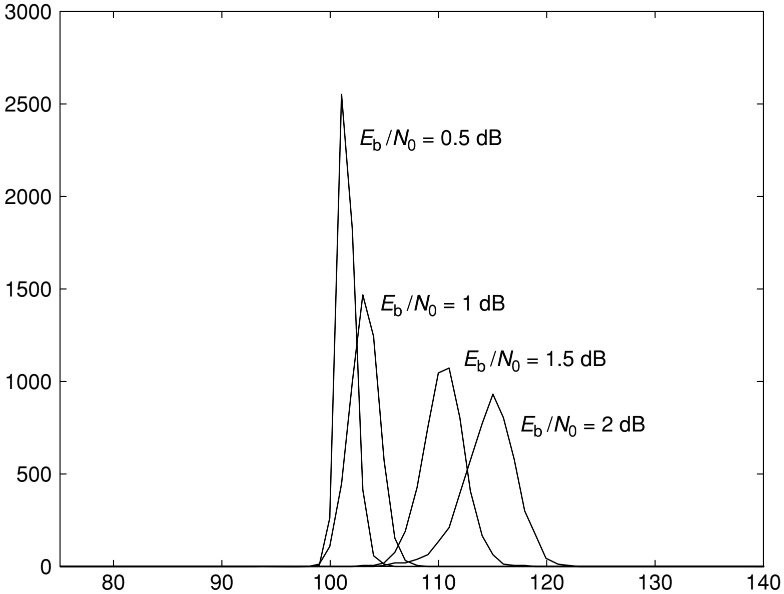


Figure 7.24 Non-normalized histogram for extrinsic estimations for the input symbol ‘1’ of a LOG MAP BCJR decoder of a turbo code, as a function of E_b/N_0

Figures 7.23 and 7.24 show that extrinsic estimates generated by a LOG MAP BCJR decoder of a turbo code are characterized by non-normalized Gaussian histograms, or extrapolating to the continuous case, Gaussian probability density functions. If these histograms are depicted over the same abscissa, it can be seen that an increase of the parameter E_b/N_0 produces a shift to the right of the extrinsic estimates for input '1', and a shift to the left of the extrinsic estimates for input '0'; that is, they start to be more separated from each other.

These Monte Carlo simulations confirm that the extrinsic information E generated by a LOG MAP BCJR decoder can be modelled as an independent and Gaussian random variable n_E of zero mean value and variance σ_E^2, which is added to the transmitted systematic bit x multiplied by μ_E. This is true for both the extrinsic estimates for '0', $E^{(0)}$ and for '1', $E^{(1)}$. Therefore

$$\begin{aligned} E^{(0)} &= \mu_{E0}x + n_{E0} \\ E^{(1)} &= \mu_{E1}x + n_{E1} \end{aligned} \tag{87}$$

where

$$\mu_{E0} = \sigma_{E0}^2/2$$

and

$$\mu_{E1} = \sigma_{E1}^2/2 \tag{88}$$

An interesting observation can be made about Figure 7.25, where a detailed view of the non-normalized histograms of the extrinsic informations for '0' and for '1' obtained by Monte Carlo simulations over the LOG MAP BCJR decoder of the turbo code of Example 7.4 is presented. Bold lines describe the extrinsic estimates for '1' and dotted lines describe the extrinsic estimates for '0'. The non-normalized histogram of the highest peak value corresponds to $E_b/N_0 = 0.2$ dB, and then lower peak value histograms correspond in decreasing order to $E_b/N_0 = 0.3, 0.4, 0.5$ and 0.6 dB.

The circled line highlights the non-normalized histograms for '0' and for '1' when $E_b/N_0 = 0.6$ dB, which is the first case for which the two peak values have different abscissa values. Now at this value of $E_b/N_0 = 0.6$ dB, the BER performance curve of Figure 7.21, which relates to the same turbo code, is in the waterfall region that starts at $E_b/N_0 = 0.5$ dB and ends at about 1.5 dB. For values of E_b/N_0 lower than 0.5 dB, the non-normalized histograms for '0' and '1' have their peaks at the same value of the abscissa, as seen in Figure 7.25.

Abscissa values of the peaks of the histograms in Figure 7.25 for low values of E_b/N_0 are coincident due to the quantization effects of the histogram representation. This means anyway that parameters μ_{E0} and μ_{E1} are in this case practically equal. The peak value of the histogram for '0' is at the abscissa value μ_{E0} that corresponds to the abscissa for the probability density function for '0', $p_E(\xi/X = -1)$, which is of the form $p_E(\xi/X = -1) = e^{-((\xi+\mu_{E0}x)^2)/2\sigma_{E0}^2}/\sqrt{2\pi}\sigma_{E0}$. The peak value of the histogram for '1' is at the abscissa value μ_{E1} that corresponds to the abscissa for the probability density function for '1', $p_E(\xi/X = +1)$, which is of the form $p_E(\xi/X = +1) = e^{-\left((\xi-\mu_{E1}x)^2\right)/2\sigma_{E1}^2}/\sqrt{2\pi}\sigma_{E1}$. It will be observed by means of these simulations that when the values of μ_{E0} and μ_{E1} are similar, the code is in the BER performance region where the coded scheme is worse than uncoded transmission. As the

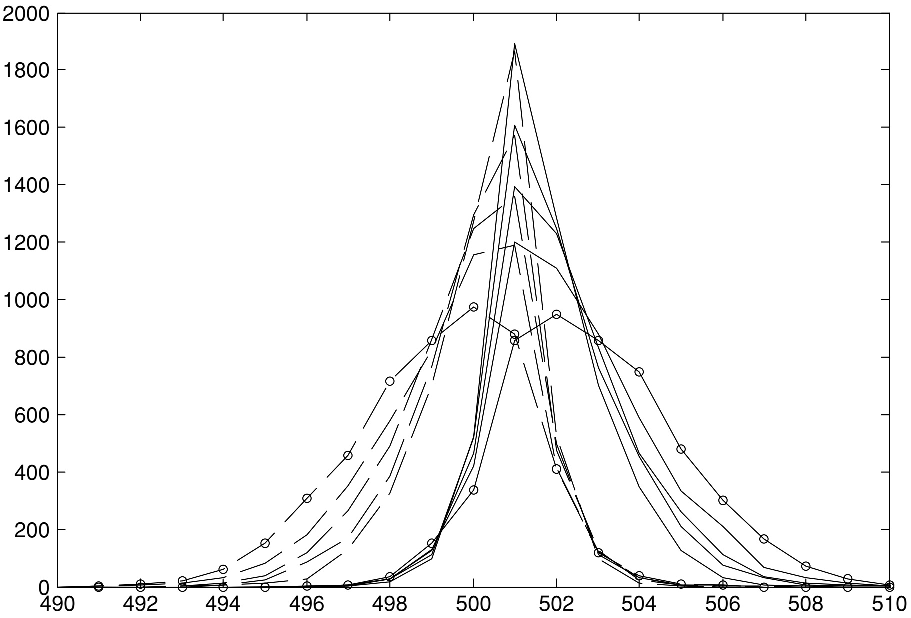


Figure 7.25 Non-normalized histogram for extrinsic estimations for the input symbol '0' of a LOG MAP BCJR decoder of the turbo code of Example 7.4, for $E_b/N_0 = 0.2, 0.3, 0.4, 0.5$ and 0.6 dB

values of μ_{E0} and μ_{E1} start to be significantly different, the turbo code performance moves into the waterfall region.

These simulations are also useful to show that since the extrinsic information of a given iteration becomes the *a priori* information of the following iteration, the *a priori* estimates A can also be modelled as a Gaussian random variable n_A of zero mean value and variance σ_A^2, which is added to the value of the transmitted systematic or message bit x multiplied by μ_A. Note that the interleaving or de-interleaving applied to convert the extrinsic information of the current iteration into *a priori* information of the following iteration changes only the position of the values of the estimations but not their statistical properties [13, 14]. Therefore,

$$A = \mu_A x + n_A \tag{89}$$

where

$$\mu_A = \sigma_A^2/2 \tag{90}$$

Thus, the probability density function for this random variable is equal to

$$p_A(\xi/X = x) = \frac{e^{-\left(\left(\xi-(\sigma_A^2/2)x\right)^2\right)/2\sigma_A^2}}{\sqrt{2\pi}\sigma_A} \tag{91}$$

The mutual information between the random variable A, which represents the *a priori* estimates, and the variable X, which represents the systematic or message bits, is utilized to determine a measure of the *a priori* information. This mutual information can be calculated as [22]

$$I_A = I(X;A) = \frac{1}{2}\sum_{x=-1,+1}\int_{-\infty}^{\infty} p_A(\xi/X=x)\,\log_2 \frac{2p_A(\xi/X=x)}{p_A(\xi/X=-1)+p_A(\xi/X=+1)}\,\mathrm{d}\xi \tag{92}$$

where $0 \le I_A \le 1$.

Expression (92) can be combined with expression (91) to give

$$I_A = 1 - \int_{-\infty}^{\infty} \frac{e^{-(\xi-\sigma_A^2/2)/2\sigma_A^2}}{\sqrt{2\pi}\sigma_A}\,\log_2(1+e^{-\xi})\,\mathrm{d}\xi \tag{93}$$

The mutual information between the random variable E, which represents the extrinsic estimates, and the variable X, which represents the systematic or message bits, is also utilized to determine a measure of the extrinsic information:

$$I_E = I(X;E) = \frac{1}{2}\sum_{x=-1,+1}\int_{-\infty}^{\infty} p_E(\xi/X=x)\,\log_2 \frac{2p_E(\xi/X=x)}{p_E(\xi/X=-1)+p_E(\xi/X=+1)}\,\mathrm{d}\xi \tag{94}$$

where $0 \le I_E \le 1$.

The EXIT chart describes, for each value of E_b/N_0, the relationship between the mutual information of the *a priori* information and the message bit information, I_A, and the mutual information of the extrinsic information and the message bit information, I_E. This extrinsic information transfer function is defined as

$$I_E = T_\mathrm{r}(I_A, E_\mathrm{b}/N_0) \tag{95}$$

The curve can be depicted by calculating, for a given value of I_A, and a given value of the parameter E_b/N_0, the corresponding value of I_E. This calculation assumes that the *a priori* information A given by expressions (89) and (90), with a probability density function described by (91), has a determined value of I_A, obtained for instance by using (93).

This *a priori* information is applied to the LOG MAP BCJR decoder together with a block of code vectors affected by noise according to the value of the parameter E_b/N_0 for which the EXIT chart is being calculated. The LOG MAP BCJR decoder generates a set or vector $\boldsymbol{E}$ of extrinsic estimates characterized by a given value of I_E. This value is obtained by Monte Carlo simulation via the extrinsic estimates E, by operating on the probability density function $p_E(\xi/X=x)$ in expression (94). All the estimates that belong to the systematic or message bits '0' will form the histogram hist $_E(\xi/X=-1)$ that represents the function $p_E(\xi/X=-1)$, and all the estimates that belong to the systematic or message bits '1' will form the histogram hist $_E(\xi/X=+1)$ that represents the function $p_E(\xi/X=+1)$. The mutual information $I_E = I(X;E)$ can be then calculated over these histograms.

This procedure is described as follows for the Example 7.4. A vector of values of *a priori* information $\boldsymbol{A}$ of size $N \times N = 10,000$ is generated by using expressions (89) and (90) for this example, for a given value of σ_A. At the same time, an array of code vectors is also generated

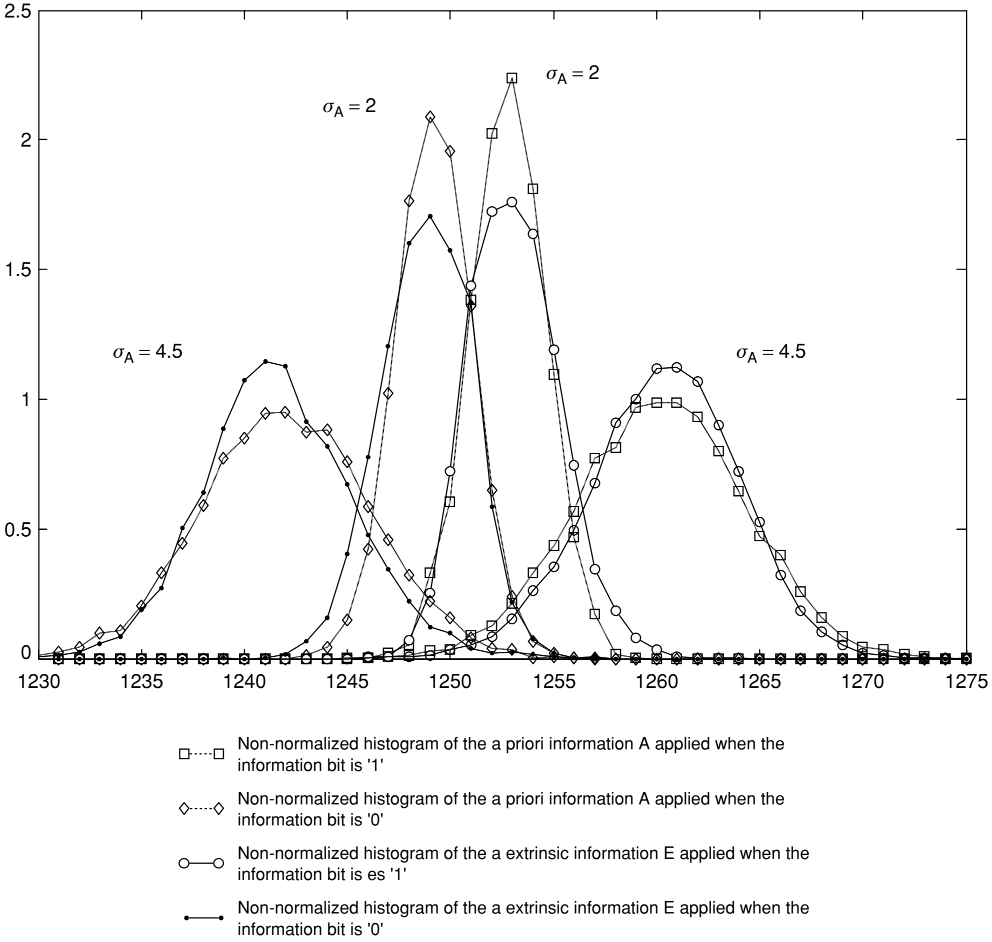


Figure 7.26 Non-normalized histograms of the *a priori* information A applied to a LOG MAP BCJR decoder and the resulting non-normalized histograms of the extrinsic information E, for Example 7.4

with a block of message bits of size $N \times N = 10,000$ that, after being encoded by the code of code rate $R_c = 1/2$, becomes a transmitted array of size $2N \times N = 20,000$ coded bits, which are affected by noise according to the value of the parameter E_b/N_0, in this case $E_b/N_0 = 1\text{dB}$. The LOG MAP BCJR decoder generates with these inputs a vector of extrinsic information $\boldsymbol{E}$ of size $N \times N = 10,000$. Figure 7.26 shows the results of this process, for $E_b/N_0 = 1\text{dB}$ and two values of σ_A, $\sigma_A = 2$ and $\sigma_A = 4.5$, which consist of the non-normalized histograms of the extrinsic information values and of the applied *a priori* values.

These histograms allow the calculation of the mutual informations of the EXIT chart. The mutual information I_A can be calculated by using expression (93), evaluated numerically, and the corresponding value of the mutual information I_E is also obtained by numerical integration over the resulting histograms like those seen in Figure 7.26 for example, and by using expression

(94) in the form

$$\begin{aligned} I_E &= I(X;E) \\ &= \frac{1}{2}\left[\int_{-\infty}^{\infty} \text{hist}_E(\xi/X=+1)\,\log_2 \frac{2\text{hist}_E(\xi/X=+1)}{\text{hist}_E(\xi/X=-1)+\text{hist}_E(\xi/X=+1)}\,\text{d}\xi \right. \\ &\left. + \int_{-\infty}^{\infty} \text{hist}_E(\xi/X=-1)\,\log_2 \frac{2\text{hist}_E(\xi/X=-1)}{\text{hist}_E(\xi/X=-1)+\text{hist}_E(\xi/X=+1)}\,\text{d}\xi \right] \end{aligned} \tag{96}$$

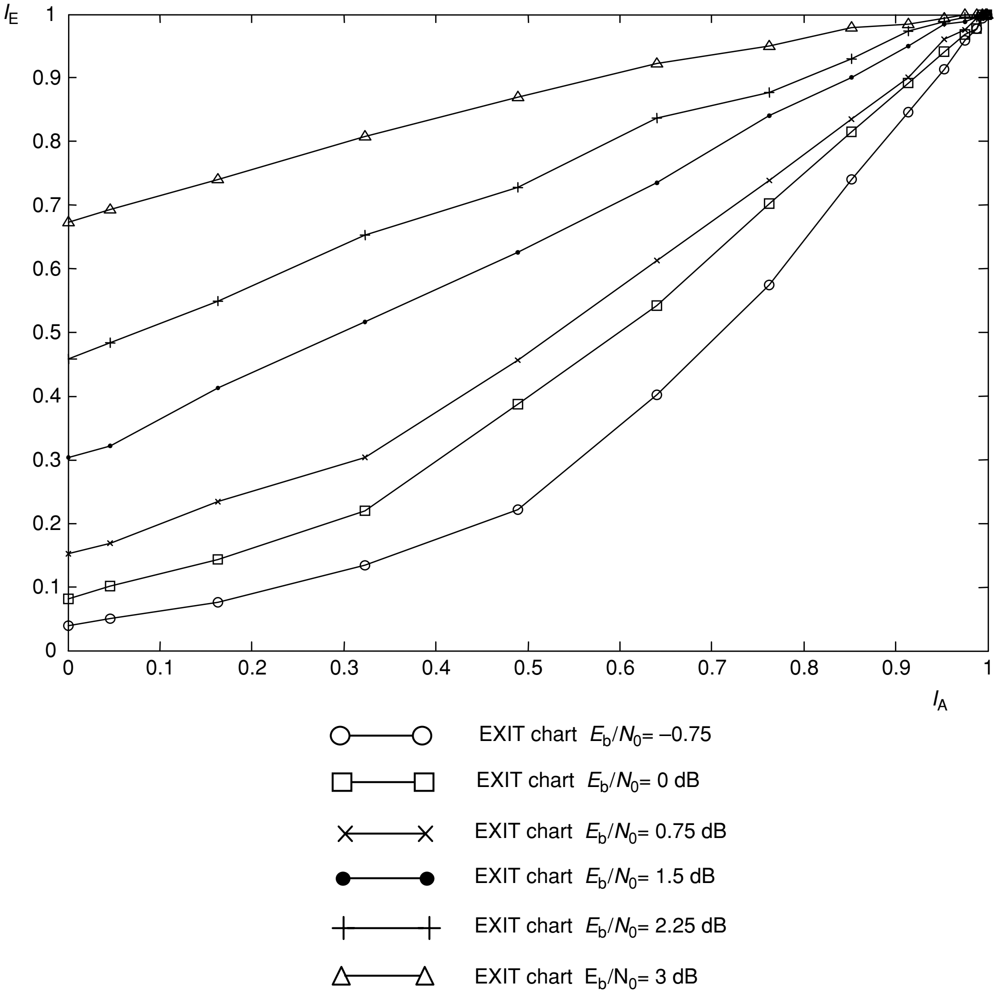


Figure 7.27 EXIT chart for the turbo code of Example 7.4, and different values of the parameter E_b/N_0

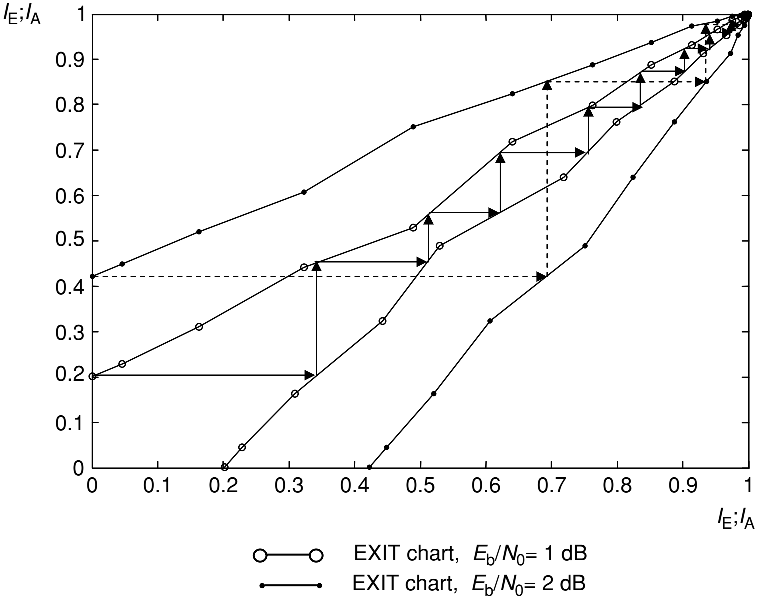


Figure 7.28 Analysis of iterative decoding of turbo codes using EXIT charts

where $\text{hist}_E(\xi/X = +1)$ and $\text{hist}_E(\xi/X = -1)$ play the role of $p_E(\xi/X = +1)$ and $p_E(\xi/X = -1)$, respectively. Integrals in (96) are solved as discrete sums over the corresponding histograms.

Values of I_A and I_E can vary slightly depending on the error event simulated. For a fixed error event and a value of E_b/N_0, and for different values of σ_A, the EXIT chart corresponding to that value of E_b/N_0 can be depicted. The dependence of this EXIT chart on the value of the parameter E_b/N_0 can also be determined. This is seen in Figure 7.27 where the EXIT chart for the turbo code of Example 7.4 describes the mutual information I_E as a function of the mutual information I_A, for different values of the parameter E_b/N_0.

Similarly, an EXIT chart can also describe the mutual information I_A as a function of the mutual information I_E, which can be useful for understanding the process in which the extrinsic information of the current iteration becomes the *a priori* information of the following iteration. This transfer function can be superposed over the transfer function of I_E as a function of I_A, to visualize the process of information interchange in an iterative decoder. This will give a clear picture of the transference of information.

This is seen in Figure 7.28, where the iterative decoding procedure is clearly represented. The procedure starts with the first decoder, and at the first iteration, when the *a priori* information is equal to zero, resulting in a given value of the extrinsic information and also of the mutual information I_E for a given value of E_b/N_0. This extrinsic information becomes the *a priori* information of the other decoder, assumed to be exactly the same as the first decoder, in the same iteration, which now operates with a non-zero *a priori* information. The symmetry of the EXIT chart analysis seen in Figure 7.28 is due to the two constituent decoders being the same, and operating under the same conditions. This process continues such that, for each

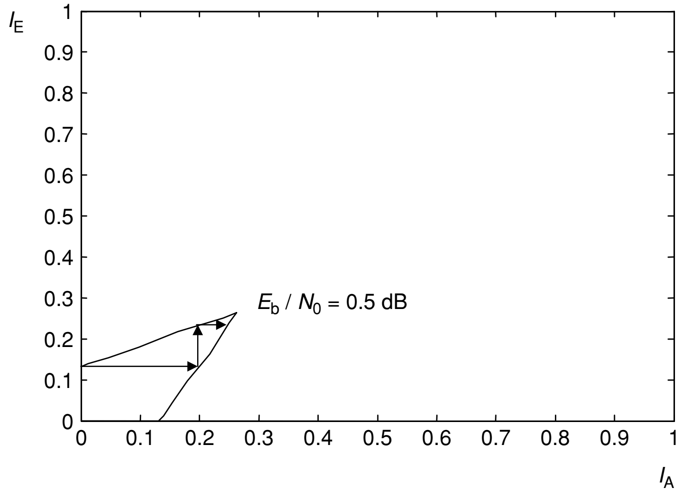


Figure 7.29 Analysis of iterative decoding of the turbo code of Example 7.4 using EXIT charts when $E_b/N_0 = 0.5$ dB

updated value of I_A, there is a corresponding value of I_E (vertical transitions between the two curves) and then this value of I_E converts into the following value of I_A for the other decoder (horizontal transitions between the two curves).

The curves in Figure 7.28 are for the iterative decoding of the turbo code of Example 7.4, using the LOG MAP BCJR algorithm, for two values of the parameter E_b/N_0, 1 dB and 2 dB. It is seen that the number of significant iterations is larger in the case of $E_b/N_0 = 1$dB. The behaviour described in Figure 7.28 for these two values of the parameter E_b/N_0 indicates that the decoder is passing from the waterfall region to the error floor region of the BER performance curve of this turbo code, that is, from a region where there are many significant iterations to another where there are few.

The iterative decoding procedure does not make sense (is ineffective) beyond the point in which the curve $I_E = T_r(I_A, E_b/N_0)$ and its reversed axes version $I_A = T_r(I_E, E_b/N_0)$ intersect. This is seen in Figure 7.29 for the turbo code of Example 7.4 and for $E_b/N_0 = 0.5$ dB.

As has been observed in Figures 7.21 and 7.25, the value of the parameter $E_b/N_0 = 0.5$ dB defines approximately the starting value for the waterfall region, but it is still a value at which error correction operates worse than uncoded transmission. This is clearly confirmed by the EXIT chart analysis as shown in Figure 7.29. This value of E_b/N_0 is known as the threshold of the coding scheme under iterative decoding [28].

Bibliography and References

[1] Berrou, C., Glavieux, A. and Thitimajshima, P., "Near Shannon limit error-correcting coding and decoding: Turbo codes," *Proc.* 1993 *IEEE International Conference on Communications*, Geneva, Switzerland, pp. 1064–1070, May 1993.

[2] Hanzo, L., Liew, T. H. and Yeap, B. L., *Turbo Coding, Turbo Equalisation and Space-Time Coding, for Transmission over Fading Channels*, IEEE Press/Wiley, New York, 2001.

[3] Heegard, C. and Wicker, S., *Turbo Coding*, Kluwer, Massachusetts, 1999.

[4] Bahl, L., Cocke, J., Jelinek, F. and Raviv, J., "Optimal decoding of linear codes for minimising symbol error rate," *IEEE Trans. Inf. Theory*, vol. IT-20, pp. 284–287, March 1974.

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

[6] Cain, J. B., Clark, G. C. and Geist, J. M., "Punctured convolutional codes of rate (n-1)/n and simplified maximum likelihood decoding," *IEEE Trans. Inf. Theory*, vol. IT-25, pp. 97–100, January 1979.

[7] Sklar, B., *Digital Communications, Fundamentals and Applications*, Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[8] Forney, G. D., Jr., *Concatenated Codes*, MIT press, Cambridge, Massachusetts, 1966.

[9] Woodard, J.P. and Hanzo, L., "Comparative study of turbo decoding techniques," *IEEE Trans. Veh. Technol.*, vol. 49, no. 6, November 2000.

[10] He, Ch., Lentmaier, M., Costello, D. J., Jr. and Zigangirov, K. Sh., "Designing linear interleavers for multiple turbo codes," *Proc. 8th International Symposium on Communications Theory and Applications*, St. Martin's College, Ambleside, United Kingdom, pp. 252–257, July 2005.

[11] Crozer, S. and Guinand, P., "Distance upper bounds and true minimum distance results for turbo-codes designed with DRP interleavers," *Proc. 3rd Internatioal Sympsium on Turbo Codes and Related Topics*, Brest, France, pp. 169–172, September 2003.

[12] Dolinar, S. and Divsalar, D., "Weight distributions for turbo codes using random and nonrandom permutations," *JPL TDA Progress Report*, pp. 56–65, August 1995.

[13] Ten Brink, S., "Convergence behaviour of iteratively decoded parallel concatenated codes," *IEEE Trans. Commun.*, vol. 49, pp. 1727–1737, October 2001.

[14] Ten Brink, S., "Convergence of iterative decoding," *Electron. Lett.*, vol. 35, no. 10, May 1999.

[15] Ten Brink, S., Speidel, J. and Yan, R., "Iterative demapping and decoding for multilevel modulation," *Proc. IEEE Globecom Conf. 98*, Sydney, NSW, Australia, pp. 579–584, November 1998.

[16] Ten Brink, S., "Exploiting the chain rule of mutual information for the design of iterative decoding schemes," *Proc. 39th Allerton Conf.*, Monticello, Illinois, October 2001.

[17] Tuchler, M., Ten Brink, S. and Hagenauer, J., "Measures for tracing convergence of iterative decoding algorithms," *Proc. 4th IEEE/ITG Conf. on Source and Channel Coding*, Berlin, Germany, pp. 53–60, January 2002.

[18] Ten Brink, S., Kramer, G. and Ashikhmin, A., "Design of low-density parity-check codes for modulation and detection," *IEEE Trans. Commun.*, vol. 52, no. 4, April 2004.

[19] Ashikhmin, A., Kramer, G. and Ten Brink, S., "Extrinsic information transfer functions: A model and two properties," *Proc. Conf. Information Sciences and Systems*, Princeton, New Jersey, March 20–22, 2002, pp. 742–747.

[20] Sharon, E., Ashikhmim, A. and Litsyn, S., "EXIT functions for the Gaussian channel," *Prov. 40th Annu. Allerton Conf. Communication*, *Control, Computers*, Allerton, Illinois, pp. 972–981, October 2003.

[21] Battail, G., "A conceptual framework for understanding turbo codes," *IEEE J. Select. Areas Commun.*, vol. 16, no. 2, pp. 245–254, February 1998.

[22] Hamming, R. W., *Coding and Information Theory,* Prentice Hall, New Jersey, 1986.

[23] Benedetto, S. and Montorsi, G., "Unveiling turbo codes: Some results on parallel concatenated coding schemes," *IEEE Trans. Inf. Theory*, vol. 42, no. 2, pp. 409–428, March 1996.

[24] Divsalar, D., Dolinar, S. and Pollara, F., "Iterative turbo decoder analysis based on gaussian density evolution," *Proc. MILCOM 2000*, vol. 1, pp. 202–208, Los Angeles, California, October 2000.

[25] Ashikhmin, A., Kramer, G. and Ten Brink, S., "Extrinsic information transfer functions: Model and erasure channel properties,“ *IEEE Trans. Inf. Theory*, vol. 50, no. 11, pp. 2657–2672, November 2004.

[26] Meyerhans, G., *Interleave and Code Design for Parallel and Serial Concatenated Convolutional Codes*, PhD Thesis, Swiss Federal Institute of Technology, University of Notre Dame, Notre Dame, Australia, 1996.

[27] Barbulescu, S. A. and Pietrobon, S. S., "Interleaver design for turbo codes," *Electron. Lett.*, vol. 30, no. 25, pp. 2107–2108, December 1994.

[28] Schlegel, Ch. and Perez, L., *Trellis and Turbo Coding*, Wiley, New Jersey, March 2004.

[29] Honary B. and Markarian G., *Trellis Decoding of Block Codes: A Practical Approach*, Kluwer, Massachusetts, 1997.

Problems

Concatenated Codes

7.1 The output codeword of a block code $C_b(6, 3)$ generated by the generator matrix $\boldsymbol{G}$ is then input to a convolutional encoder like that seen in Figure P.7.1, operating in pseudo-block form. This means that after inputting the 6 bits of the codeword of the block code, additional bits are also input to clear the registers of the convolutional encoder.

(a) Determine the transpose of the parity check matrix of the block code, the syndrome-error pattern table, the trellis diagram of the convolutional code and its minimum free distance.

$$\boldsymbol{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

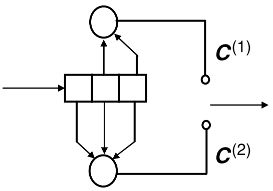


Figure P.7.1 Convolutional encoder in a serial concatenation with a block code

(b) Determine the minimum distance of the concatenated code.
(c) Decode the received sequence $r = (01\,10\,10\,00\,11\,11\,00\,00)$ to find possible errors, and the transmitted sequence.

7.2 The cyclic code $C_{\mathrm{cyc}}(3, 1)$ generated by the polynomial $g(X) = 1 + X + X^2$ is applied on a given bit ‘horizontally’ and then a second cyclic code $C_{\mathrm{cyc}}(7, 3)$ generated by the polynomial $g(X) = 1 + X + X^2 + X^4$ is applied ‘vertically’ over the codeword in an array code format, as seen in Figure P.7.2.
(a) Determine the rate and error-correction capability of this array code.
(b) What is the relationship between the error-correction capability of this array code and the individual error-correction capabilities of each cyclic code?

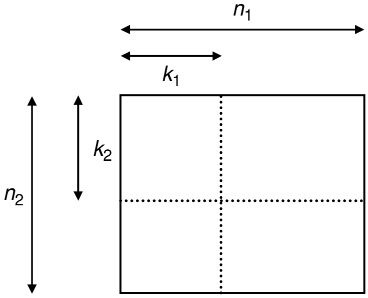


Figure 7.2 An array code

(c) Construct the array code by applying first the cyclic code $C_{\mathrm{cyc}}(7, 3)$ and then the cyclic code $C_{\mathrm{cyc}}(3, 1)$ in order to compare with the result of item (b).

Turbo Codes

7.3 The simple binary array code (or punctured product code) has codewords with block length $n = 8$ and $k = 4$ information bits in the format as given in Figure P.7.3.

1	2	5
3	4	6
7	8	

Figure 7.3 A punctured array code

Symbols 1, 2, 3 and 4 are the information bits, symbols 5 and 6 are row check bits and symbols 7 and 8 are column check bits. Thus bits 1, 2, 5 and 3, 4, 6, form two single-parity check (SPC) row component codewords, and bits 1, 3, 7 and 2, 4, 8 form two SPC column component codewords, where each component code has the parameters $(n, k) = (3, 2)$.

This array code can be regarded as a simple form of turbo code. In terms of the turbo code structure shown in Figure P.7.4, the parallel concatenated component encoders calculate the row and column parity checks of the array code, and the permuter alters the order in which the information bits enter the column encoder from $\{1, 2, 3, 4\}$ to $\{1, 3, 2, 4\}$. The multiplexer then collects the information and parity bits to form a complete codeword.

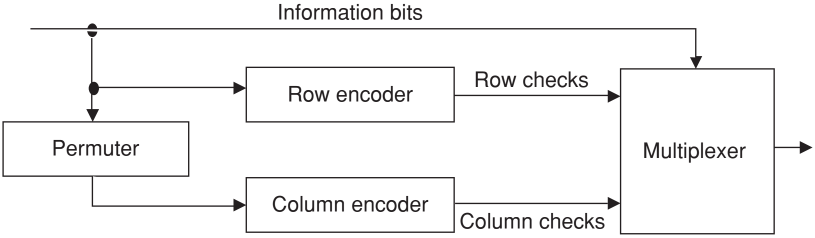


Figure 7.4 An array code viewed as a turbo code

(a) What is the rate and Hamming distance of this code?

(b) A codeword from the code is modulated, transmitted over a soft-decision discrete symmetric memoryless channel like that seen in Figure P.7.5, with the conditional probabilities described in Table P.7.1, and received as the vector $\boldsymbol{r} = (10300000)$. Using the turbo (iterative) MAP decoding algorithm, determine the information bits that were transmitted.

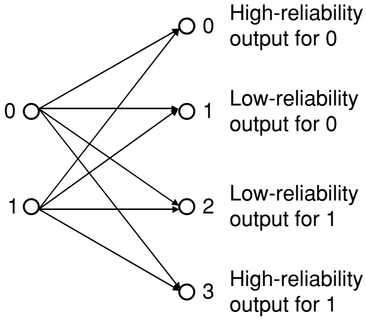

0
1
0
1
2
3
High-reliability output for 0
Low-reliability output for 0
Low-reliability output for 1
High-reliability output for 1

Figure 7.5 A soft-decision discrete symmetric memoryless channel

Table P.7.1 Transition probabilities of the channel of Figure P.7.5

		$P(y/x)$		
x, y	0	1	2	3
0	0.4	0.3	0.2	0.1
1	0.1	0.2	0.3	0.4

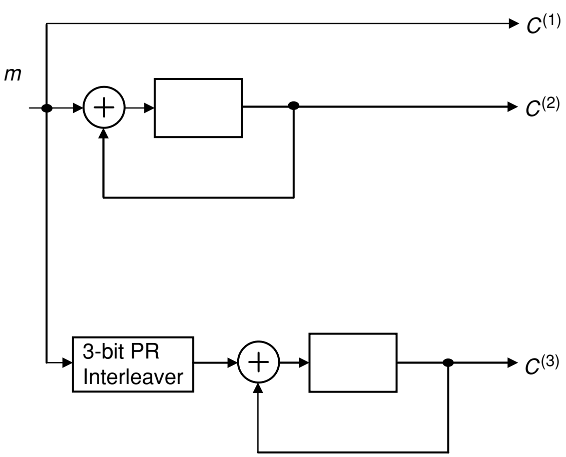


Figure 7.6 A 1/3-rate turbo code

7.4 Determine the minimum free distance of each of the constituent codes of the 1/3-rate turbo code encoded as shown in Figure P.7.6. Then, also determine the minimum free distance of this turbo code.

The 3-bit pseudo-random interleaver has the permutation rule $\begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix}$.

Table P.7.2 Input message vector and received sequence, Problem 7.5

Input Sequence	Received Sequence	
−1	−0.5290	− 0.3144
−1	−0.01479	− 0.1210
−1	−0.1959	+ 0.03498
+1	1.6356	− 2.0913
−1	−0.9556	+ 1.2332
+1	1.7448	− 0.7383
−1	−0.3742	− 0.1085
−1	−1.2812	− 1.8162
+1	+0.5848	+ 0.1905
+1	+0.6745	− 1.1447
−1	−2.6226	− 0.5711
+1	+0.7426	+ 1.0968
+1	1.1303	− 1.6990
−1	−0.6537	− 1.6155
+1	2.5879	− 0.5120
−1	−1.3861	− 2.0449

7.5 For a turbo code with the structure of Figure 7.20, with a block interleaver of size $N \times N = 4 \times 4$ like that used in Example 7.3, constructed using 1/2-rate SRC (5, 7) encoders and a puncturing rule like that utilized in Example 7.3, the input or message vector is $\boldsymbol{m} = \left(-1\ -1\ -1\ +1\ -1\ +1\ -1\ -1\ +1\ +1\ -1\ +1\ +1\ -1\ +1\ -1\right)$.This input vector makes the first encoder sequence be terminated.

(a) Determine the input for the second decoder, and the corresponding output of the turbo code.

(b) After being transmitted and corrupted by AWGN, the received sequence as tabulated in Table P.7.2 is then applied to the decoder.

Use the MAP BCJR decoding algorithm to determine the decoded sequence. Estimate the number of iterations needed to arrive at the correct solution in this particular case.

8

Low-Density Parity Check Codes

In his seminal 1949 paper [1], Shannon stated theoretical bounds on the performance of error-correction coding. Since that time, many practical error-correction schemes have been proposed, but none has achieved performances close to the ideal until the turbo coding scheme, described in Chapter 7, was discovered in 1993 by Berrou, Glavieux and Thitimajshima [3]. Following this stimulating discovery, 3 years later in 1996 MacKay and Neal [4, 5] rediscovered a class of codes first introduced by Gallager in 1962 [6], which are now recognized also to have near-ideal performance. These Gallager codes are the subject of this chapter.

Gallager codes, now widely known as low-density parity check (LDPC) codes, are linear block codes that are constructed by designing a sparse parity check matrix $\boldsymbol{H}$, that is, for the binary case, a matrix that contains relatively few '1's spread among many '0's. Gallager's original paper, apart from various LDPC code constructions, also presented an iterative method of decoding the codes, which was capable of achieving excellent performance. However, the complexity of the iterative decoding algorithm was beyond the capabilities of the electronic processors available then, which is why the codes were forgotten until 1996, even in spite of an attempt by Tanner in 1981 [7] to revive interest in them.

The first construction method proposed for the design of the sparse parity check matrix $\boldsymbol{H}$ associated with these codes involves the use of a fixed number of '1's per row and per column of that matrix. In this case LDPC codes are said to be regular. However, the number of '1's per row and column can be varied, leading to the so-called irregular LDPC codes.

The bit error rate (BER) performance of LDPC codes is close to that of the turbo codes. Modified versions of the original scheme, basically implemented as irregular LDPC codes, and also operating over GF(q), with $q = 4$, 8 and 16 [8], are shown to perform even better than the best-known turbo codes, being very close to the Shannon limits. A common factor between LDPC codes and turbo codes is that the best BER performance of these coding techniques is obtained when a pseudo-random process is applied to the design of parts of these coding schemes. Thus, this pseudo-random procedure is present in the design of the random interleaver of a turbo code, and in the construction of the sparse parity check matrix $\boldsymbol{H}$ of an LDPC code.

Essentials of Error-Control Coding Jorge Castiñeira Moreira and Patrick Guy Farrell

8.1 Different Systematic Forms of a Block Code

As seen in Chapter 2, a systematic linear block code $C_{\mathrm{b}}(n, k)$ is uniquely specified by its generator matrix, which in the case of systematic block codes is of the form

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{g}_0 \\ \boldsymbol{g}_1 \\ \vdots \\ \boldsymbol{g}_{k-1} \end{bmatrix} = \left[\underbrace{\begin{matrix} p_{00} & p_{01} & \cdots & p_{0,n-k-1} \\ p_{10} & p_{11} & \cdots & p_{1,n-k-1} \\ \vdots & \vdots & \vdots & \vdots \\ p_{k-1,0} & p_{k-1,1} & \cdots & p_{k,n-k-1} \end{matrix}}_{\text{Submatrix } \boldsymbol{P}\ k \times (n-k)} \quad \underbrace{\begin{matrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{matrix}}_{\text{Submatrix } \boldsymbol{I}\ k \times k} \right] \tag{1}$$

A shorter notation for this matrix is

$$\boldsymbol{G} = [\boldsymbol{P}\ \boldsymbol{I}_k] \tag{2}$$

where $\boldsymbol{P}$ is the parity submatrix and $\boldsymbol{I_k}$ is the identity submatrix of dimension $k \times k$. In this form of systematic encoding, the message bits appear at the end of the code vector.

The systematic form of the parity check matrix $\boldsymbol{H}$ of the code C_{b} generated by the generator matrix $\boldsymbol{G}$ is

$$\boldsymbol{H} = \left[\underbrace{\begin{matrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \end{matrix} \quad \begin{matrix} p_{00} \\ p_{01} \\ \vdots \\ p_{0,n-k-1} \end{matrix} \quad \begin{matrix} p_{10} \\ p_{11} \\ \vdots \\ p_{1,n-k-1} \end{matrix}}_{\text{Submatrix } \boldsymbol{I}(n-k) \times (n-k)} \quad \underbrace{\begin{matrix} \cdots & p_{k-1,0} \\ \cdots & p_{k-1,1} \\ & \vdots \\ \cdots & p_{k-1,n-k-1} \end{matrix} \right] = \left[\boldsymbol{I_{n-k}}\ \boldsymbol{P^T} \right]}_{\text{Submatrix } \boldsymbol{P}^T(n-k) \times k} \tag{3}$$

where $\boldsymbol{P}^T$ is the transpose of the parity submatrix $\boldsymbol{P}$. The parity check matrix $\boldsymbol{H}$ is such that the inner product between a row vector $\boldsymbol{g}_i$ of the generator matrix $\boldsymbol{G}$ and a row vector $\boldsymbol{h}_j$ of the parity check matrix $\boldsymbol{H}$ is zero; that is, $\boldsymbol{g}_i$ and $\boldsymbol{h}_j$ are orthogonal.

Therefore,

$$\boldsymbol{G} \circ \boldsymbol{H}^T = \boldsymbol{0} \tag{4}$$

and then

$$\boldsymbol{c} \circ \boldsymbol{H}^T = \boldsymbol{m} \circ \boldsymbol{G} \circ \boldsymbol{H}^T = \boldsymbol{0} \tag{5}$$

As will be seen in the following section, the design of an LDPC code starts with the construction of the corresponding parity check matrix $\boldsymbol{H}$, from which an equivalent systematic parity check matrix is obtained, leading to formulation of the generator matrix $\boldsymbol{G}$ of the code.

The syndrome equation for a block code can be described in terms of the parity check matrix $\boldsymbol{H}$, instead of in terms of the transpose of this matrix, $\boldsymbol{H}^T$, as it was done in Chapter 4. Then the code vector is generated from

$$\boldsymbol{c} = \boldsymbol{G}^T \circ \boldsymbol{m} \tag{6}$$

Since $\boldsymbol{G}^T$ is a generator matrix of dimension $n \times k$, and the message vector $\boldsymbol{m}$ is of dimension $k \times 1$, the result of the operation in (6) is a code vector $\boldsymbol{c}$ of dimension $n \times 1$, which is a column vector. If a code vector is generated by using expression (6), then the corresponding syndrome decoding is based on the calculation of a syndrome vector $\boldsymbol{S}$ of the form

$$\boldsymbol{S} = \boldsymbol{H} \circ \boldsymbol{c} \tag{7}$$

which means that every code vector satisfies the condition

$$\boldsymbol{H} \circ \boldsymbol{c} = \boldsymbol{0} \tag{8}$$

and that equation (4) is of the form

$$\boldsymbol{H} \circ \boldsymbol{G}^T = \boldsymbol{0} \tag{9}$$

Since the parity check matrix $\boldsymbol{H}$ is of dimension $(n - k) \times n$, and the generator matrix $\boldsymbol{G^T}$ is of dimension $n \times k$, the syndrome condition is represented by a matrix of dimension $(n - k) \times k$ with all its elements equal to zero. This alternative way of encoding and decoding a block code indicates that the parity check matrix $\boldsymbol{H}$ contains all the necessary information to completely describe the block code. On the other hand, expression (9) will be useful for designing an iterative decoder based on the sum–product algorithm, as will be shown in the following sections.

8.2 Description of LDPC Codes

LDPC codes are usually designed to be linear and binary block codes. In this case there is a generator matrix $\boldsymbol{G}$ that converts a message vector $\boldsymbol{m}$ into a code vector $\boldsymbol{c}$ by means of a matrix multiplication. The corresponding parity check matrix $\boldsymbol{H}$ has the property that it is constructed with linearly independent row vectors that form the dual subspace of the subspace generated by the linearly independent row vectors of $\boldsymbol{G}$. This means that every code vector satisfies the condition $\boldsymbol{H} \circ \boldsymbol{c} = \boldsymbol{0}$.

LDPC codes are designed by an appropriate construction of the corresponding parity check matrix $\boldsymbol{H}$, which is characterized by being sparse. According to the definition given by Gallager, an LDPC code is denoted as $C_{\mathrm{LDPC}}(n, s, v)$, where n is the code length, s is the number of '1's per column, with in general $s \geq 3$, and v is the number of '1's per row. If the rows of the parity check matrix are linearly independent then the code rate is equal to $(v - s)/v$ [4]. Otherwise the code rate is $\left(n - s'\right)/n$, where s' is the actual dimension of the row subspace generated by the parity check matrix $\boldsymbol{H}$. This relationship is obtained by counting the total number of '1's per row, and then per column, giving that $ns = (n - k)v$, which by algebraic manipulation leads to the expression for the code rate. If the LDPC code is irregular, the numbers of '1's per row v and/or per column s are not fixed, and so the rate of the code is obtained by using the average values of these parameters.

The code construction proposed by Gallager allowed him to demonstrate the main properties of these codes: the error probability decreases exponentially with increasing code block length, and the minimum distance of the code also increases with increasing code length. Tanner [7] generalized the Gallager construction, defining the so-called bipartite graphs, where equations related to the graph are generalized to be independent equations for each graph condition, instead of being simply stated as parity check conditions.

8.3 Construction of LDPC Codes

8.3.1 Regular LDPC Codes

The construction method proposed by Gallager consists of forming a sparse parity check matrix $\boldsymbol{H}$ by randomly determining the positions of '1's, with a fixed number of ones '1's per column and per row, thus creating a regular LDPC code. The condition on the number of '1's per column and per row can be relaxed, provided that the number of '1's per column s satisfies $s > 2$. In this case, the LDPC code is said to be irregular. The conditions to be satisfied in the construction of the parity check matrix $\boldsymbol{H}$ of a binary regular LDPC code are [9]

- The corresponding parity check matrix $\boldsymbol{H}$ should have a fixed number v of '1's per row.
- The corresponding parity check matrix $\boldsymbol{H}$ should have a fixed number s of '1's per column.
- The overlapping of '1's per column and per row should be at most equal to one. This is a necessary condition for avoiding the presence of cycles in the corresponding bipartite graph.
- The parameters s and v should be small numbers compared with the code length.

It is however very difficult to satisfy the third condition if the intention is to construct good LDPC codes, because cycles are unavoidable in the bipartite graph of an efficient LDPC code [18].

The above construction does not normally lead to the design of a sparse parity check matrix $\boldsymbol{H}$ of systematic form, and so it is usually necessary to utilize Gaussian elimination to convert this resulting matrix into a systematic parity check matrix $\boldsymbol{H}' = [\boldsymbol{I}_{n-k} \quad \boldsymbol{P}^T]$, where $\boldsymbol{I}_{n-k}$ is the identity submatrix of dimension $(n-k) \times (n-k)$. The initially designed sparse parity check matrix $\boldsymbol{H}$ is the parity check matrix of the LDPC code, whose generator matrix $\boldsymbol{G}$ is of the form $\boldsymbol{G} = [\,\boldsymbol{P} \ \boldsymbol{I}_k\,]$.

Summarizing the design method for an LDPC code, a sparse parity check matrix $\boldsymbol{H} = [\boldsymbol{A}\,\boldsymbol{B}]$ is constructed first, obeying the corresponding construction conditions. In general, this initial matrix is not in systematic form. Submatrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are sparse. Submatrix $\boldsymbol{A}$ is a square matrix of dimension $(n-k) \times (n-k)$ that is non-singular, and so it has an inverse matrix $\boldsymbol{A}^{-1}$. Submatrix $\boldsymbol{B}$ is of dimension $(n-k) \times k$.

The Gaussian elimination method, operating over the binary field, modifies the matrix $\boldsymbol{H} = [\boldsymbol{A} \ \boldsymbol{B}\,]$ into the form $\boldsymbol{H}' = [\,\boldsymbol{I}_k \ \boldsymbol{A}^{-1}\boldsymbol{B}\,] = [\,\boldsymbol{I}_k \ \boldsymbol{P}^T\,]$. This operation is equivalent to premultiplying $\boldsymbol{H} = [\,\boldsymbol{A} \ \boldsymbol{B}\,]$ by $\boldsymbol{A}^{-1}$. Once the equivalent parity check matrix $\boldsymbol{H}'$ has been formed, the corresponding generator matrix $\boldsymbol{G}$ can be constructed by using the submatrices obtained, to give $\boldsymbol{G} = [\,\boldsymbol{P} \ \boldsymbol{I}_k\,]$. In this way both the generator and the parity check matrices are defined, and the LDPC code is finally designed. Note that the matrices of interest are $\boldsymbol{H}$ and $\boldsymbol{G}$.

LDPC codes can be classified, according to the construction method used for generating the corresponding sparse parity check matrix $\boldsymbol{H}$, into [11]

- random LDPC codes and
- structured LDPC codes

In general, random LDPC codes show a slightly better BER performance than that of structured LDPC codes, but these latter codes are much less complex to encode than the former codes. The construction approach proposed by MacKay [4, 5] is random, while other approaches include those based on finite field geometries, balanced incomplete block designs and cyclic or quasi-cyclic structures [10, 11].

8.3.2 Irregular LDPC Codes

As described in previous sections, an irregular LDPC code is one with a sparse parity check matrix $\boldsymbol{H}$ that has a variable number of '1's per row or per column. In general, the BER performances of irregular LDPC codes are better than those of regular LDPC codes. There are several construction methods for irregular LDPC codes [9].

8.3.3 Decoding of LDPC Codes: The Tanner Graph

As described in Section 8.1, an alternative encoding method for block codes operates on the message vector $\boldsymbol{m}$ by means of the matrix operation $\boldsymbol{c} = \boldsymbol{G}^T \circ \boldsymbol{m}$, where

$$\boldsymbol{G}^T = \begin{bmatrix} \boldsymbol{P}^T \\ \boldsymbol{I}_k \end{bmatrix}$$

to generate the code vector $\boldsymbol{c}$. The transmitted vector is affected by the channel noise to be converted in the received vector $\boldsymbol{r} = \boldsymbol{c} + \boldsymbol{n}$, which is the input information for traditional decoders of block codes based on calculation of the syndrome vector $\boldsymbol{S} = \boldsymbol{H} \circ \boldsymbol{r} = \boldsymbol{H} \circ (\boldsymbol{G}^T \circ \boldsymbol{m} + \boldsymbol{n}) = \boldsymbol{H} \circ \boldsymbol{n}$. An alternative decoding algorithm is introduced in this section, also based on this syndrome calculation. The essence of this decoding algorithm is to determine an estimate of a vector $\boldsymbol{d}$ that satisfies the condition $\boldsymbol{H} \circ \boldsymbol{d} = \boldsymbol{0}$.

This algorithm is known as the sum–product algorithm, or belief propagation algorithm. This algorithm determines the *a posteriori* probability of each message symbol as a function of the received signal, the code information, expressed in the case of LDPC codes as parity equations, and the channel characteristics. This algorithm is conveniently described over a bipartite graph, called the Tanner graph [7], which is defined by the parity equations described in the corresponding parity check matrix $\boldsymbol{H}$. The bipartite graph depicts the relationship between two types of nodes, the symbol nodes d_j, which represent the transmitted symbols or bits, and the parity check nodes h_i, which represent the parity equations in which the bits or symbols are related. Rows of the parity check matrix $\boldsymbol{H}$ identify the symbols involved in each parity equation, so that a given row describes a parity check equation, and positions filled with '1's determine the positions of the symbols involved in that parity check equation. In this way, and for binary LDPC codes, if the entry $\{i, j\}$ of the sparse parity check matrix $\boldsymbol{H}$ is equal to one, $H_{ij} = 1$, then there exists in the corresponding bipartite graph a connection between the symbol node d_j and the check node h_i; otherwise, the connection is not present.

The state of a given parity check node depends on the values of the symbol nodes actually connected to it. In general, the parity check nodes connected to a given symbol node are said to be the children nodes of that symbol node, and the symbol nodes connected to a given parity check node are said to be the parent nodes of that parity check node.

In the sum–product algorithm, each symbol node d_j sends to each of its children parity check nodes h_i an estimate Q_{ij}^x of the probability that the parity check node is in state x, based

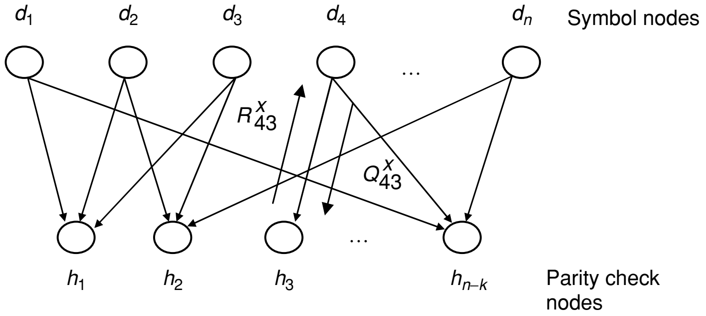


Figure 8.1 A Tanner graph, a bipartite graph linking symbol and parity check nodes

on the information provided by the other children nodes of that symbol node. On the other hand, each parity check node h_i sends to each of its parent symbol nodes d_j an estimate R^x_{ij} of the probability that the parity equation i related to the parity check node h_i is satisfied, if the symbol or parent node is in state x, by taking into account the information provided by all the other parent symbol nodes connected to this parity check node. This is seen in the Tanner graph of Figure 8.1.

This is an iterative process of interchanging information between the two types of nodes on the bipartite graph. The iterative process is halted if after calculating the syndrome condition over the estimated decoded vector $\boldsymbol{d}$, at a given iteration, the resulting syndrome vector is the all-zero vector. If after several successive iterations the syndrome does not become the all-zero vector, the decoder is halted when it reaches a given predetermined number of iterations. In both cases, the decoder generates optimally decoded symbols or bits, in the *a posteriori* probability sense, but these will not form a code vector if the syndrome is not an all-zero vector. In this sense the sum–product algorithm performs in the same way as the MAP BCJR algorithm, defining the best possible estimate of each symbol of the received vector, but not necessarily defining the best estimate of the whole code vector that was initially transmitted through the channel. This is a consequence of the sum–product algorithm being a maximum *a posteriori* (MAP) decoding algorithm, whereas other decoding algorithms like the Viterbi algorithm are maximum likelihood (ML) decoding algorithms, which optimize the decoding of the whole code vector or sequence.

In general, iterative decoding of an LDPC code converges to the true message information when the corresponding bipartite graph has a tree structure, that is, contains no cycles. However, the presence of cycles of relatively short lengths in the bipartite graph is virtually unavoidable when the corresponding LDPC code has good properties, but it is often possible to remove the shortest cycles (of length 4, 6, 8, etc.), or least reduce their number. The degrading effect of short-length cycles in the bipartite graph however diminishes as the code length increases and is strongly reduced if the code length is large ($>$1000 bits).

8.4 The Sum–Product Algorithm

In the following, the sum–product algorithm is described. The algorithm requires an initialization procedure that consists of determining the values Q^x_{ij}, which are set to the *a priori* estimates of the received symbols, denoted as f^x_j, the probability that the jth symbol is x. This information depends on the channel model utilized. In the case of the additive white Gaussian noise (AWGN) channel, these probabilities are determined by using a Gaussian probability density function.

After the initialization, the interchange of information between the symbol and the parity check nodes begins. The information R_{ij}^{x} that each parity check node h_i sends to its parent symbol node d_j is the probability that the parity check node h_i is satisfied (that is, the parity check equation related to this node is satisfied) when the parent symbol node being informed is in state x. The probability of the parity check equation being satisfied is given by

$$P(h_i/d_j = x) = \sum_{\boldsymbol{d}:d_j=x} \boldsymbol{P}(h_i/\boldsymbol{d})P(\boldsymbol{d}/d_j = x) \tag{10}$$

This probability is calculated over all the possible decoded vectors $\boldsymbol{d}$ for which the parity check equation is satisfied, when the informed parent symbol node is in state x.

For the parity check node h_i, the information to be sent to the parent symbol node d_j is calculated for each value of x and is given by

$$R_{ij}^{x} = \sum_{\boldsymbol{d}:d_j=x} P\left(\boldsymbol{h_i}/\boldsymbol{d}\right) \prod_{k \in N(i)\backslash j} Q_{ik}^{d_k} \tag{11}$$

In this expression, $N(i)$ represents the set of indexes of all the parent symbol nodes connected to the parity check node h_i, whereas $N(i)\backslash j$ represents the same set with the exclusion of the parent symbol node d_j. The probability $P\left(\boldsymbol{h_i}/\boldsymbol{d}\right)$ that the parity check equation is satisfied or not is equal to 1 or 0 respectively, for a given vector $\boldsymbol{d}$. The symbol node d_j sends to its children parity check nodes h_i the estimate Q_{ij}^{x}, which is the estimate that the node is in state x according to the information provided by the other children parity check nodes connected to it. Then, and by using the Bayes rule,

$$P\left(d_j = x/\left\{h_i\right\}_{i \in M(j)\backslash i}\right) = \frac{P\left(d_j = x\right) P\left(\left\{h_i\right\}_{i \in M(j)\backslash i}/d_j = x\right)}{P\left(\left\{h_i\right\}_{i \in M(j)\backslash i}\right)} \tag{12}$$

The information that the symbol node d_j sends to its children parity check nodes is then

$$Q_{ij}^{x} = \alpha_{ij} f_j^{x} \prod_{k \in M(j)\backslash i} R_{kj}^{x} \tag{13}$$

where $M(j)$ represents the set of indexes of all the children parity check nodes connected to the symbol node d_j, whereas $M(j)\backslash i$ represents the same set with the exclusion of the children parity check node h_i. The coefficient f_j^{x} is the *a priori* probability that d_j is in state x. The normalizing constant α_{ij} is set to satisfy the normalizing condition $\sum_x Q_{ij}^{x} = 1$.

In this way, the calculation of coefficients Q_{ij}^{x} allow us to determine the values of the coefficients R_{ij}^{x} that in turn can be used to perform an estimate for each value of the index j. This is in the end an estimate for each symbol of the received vector, represented in the binary case by the estimates for the two possible values of the variable x. This estimate is equal to

$$\hat{d}_j = \arg\max_{x} f_j^{x} \prod_{k \in M(j)} R_{kj}^{x} \tag{14}$$

which constitutes the estimate for the symbol at position j. If the estimated decoded vector $\hat{d}$ satisfies the syndrome condition $\boldsymbol{H} \circ \hat{\boldsymbol{d}} = \boldsymbol{S}$ (in general expressed in the form of $\boldsymbol{H} \circ \hat{\boldsymbol{d}} = \boldsymbol{0}$), then the estimated decoded vector $\hat{\boldsymbol{d}}$ is considered as a valid code vector $\boldsymbol{c} = \hat{\boldsymbol{d}}$. Otherwise, and if the decoder reaches the predetermined limiting number of iterations without finding a suitable code vector that satisfies the above syndrome condition, then each symbol has been optimally estimated, even though not all the symbols of the code vector actually transmitted have been correctly decoded.

8.5 Sum–Product Algorithm for LDPC Codes: An Example

In this example, the following fairly sparse parity check matrix $\boldsymbol{H}$ of dimension 8×12 corresponds to a linear block code $C_{\mathrm{b}}(12, 4)$, of code rate $R_{\mathrm{c}} = 1/3$, which is an irregular LDPC code whose systematic generator matrix $\boldsymbol{G}$ is shown below:

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

$$\boldsymbol{G} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The encoding procedure adopted here is the traditional one, where the code vector $\boldsymbol{c}$ is generated by multiplying the message vector $\boldsymbol{m}$ by the generator matrix $\boldsymbol{G}$, $\boldsymbol{c} = \boldsymbol{m} \circ \boldsymbol{G}$, so that the code vector satisfies the syndrome equation $\boldsymbol{c} \circ \boldsymbol{H}^T = \boldsymbol{0}$. As explained in Section 8.1, there is an equivalent encoding method in which these operations are performed using $\boldsymbol{c} = \boldsymbol{G}^T \circ \boldsymbol{m}$ and $\boldsymbol{H} \circ \boldsymbol{c} = \boldsymbol{0}$.

In this example, the message vector is $\boldsymbol{m} = (1\ 0\ 0\ 0)$, which generates the code vector $\boldsymbol{c} = (1\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0)$. This code vector is transmitted in polar format as the vector $\boldsymbol{t} = (+1\ +1\ +1\ +1\ +1\ -1\ -1\ -1\ +1\ -1\ -1\ -1)$. The transmission is done over an AWGN channel with a standard deviation of $\sigma = 0.8$, and as a result of the transmission and the sampling procedure, the following received vector is obtained:

$$\boldsymbol{r} = (+1.3129\ +2.6584\ +0.7413\ +2.1745\ +0.5981\ -0.8323\ -0.3962\ -1.7586$$
$$+1.4905\ +0.4084\ -0.9290\ +1.0765)$$

If a hard-decision decoder was utilized, then the decoded vector would be

$$(1\quad 1\quad 1\quad 1\quad 1\quad 0\quad 0\quad 0\quad 1\quad 1\quad 0\quad 1)$$

so that the channel produced two errors, at positions 10 and 12.

As the channel in this transmission is the AWGN channel, coefficients f_j^x corresponding to the received vector can be calculated using the Gaussian probability density function, and thus

$$f_j^0 = \frac{1}{\sqrt{2\pi}\sigma} e^{-(r_j+1)^2/(2\sigma^2)} \tag{15}$$

$$f_j^1 = \frac{1}{\sqrt{2\pi}\sigma} e^{-(r_j-1)^2/(2\sigma^2)} \tag{16}$$

These estimates require us to know the value of the standard deviation of the noise σ; that is, they require knowledge of the channel characteristics. Table 8.1 shows the values obtained in this particular case.

In Table 8.1 and the following tables, note that values are truncated to four decimal places, and so iterative calculation done with these truncated values could lead to slight differences in numerical results throughout the decoding. Thus, for example, the value of f_2^0 is small (1.43×10^{-5}), but appears in the table as zero. The actual calculations were done more accurately using MATLAB® Program 5.3.

Values of the coefficients f_j^x represent the estimates of the channel information. The coefficients involved in the iterative calculation take into account the code structure, described in the corresponding bipartite graph, which represent the parity check equations. Thus, the syndrome condition, expressed either as $\boldsymbol{H} \circ \boldsymbol{c} = \boldsymbol{0}$ or as $\boldsymbol{c} \circ \boldsymbol{H}^T = \boldsymbol{0}$, means that the multiplication of the code vector $\boldsymbol{c}$ (using addition and multiplication over GF(2)) should be equal to the all-zero vector. Therefore the parity check equations can be written as

$$\begin{aligned}
c_2 \oplus c_4 \oplus c_6 \oplus c_7 \oplus c_8 \oplus c_{12} &= 0 \\
c_1 \oplus c_3 \oplus c_4 \oplus c_9 &= 0 \\
c_2 \oplus c_5 \oplus c_7 \oplus c_{12} &= 0 \\
c_1 \oplus c_4 \oplus c_{10} \oplus c_{11} &= 0 \\
c_3 \oplus c_5 \oplus c_6 \oplus c_{10} &= 0 \\
c_1 \oplus c_3 \oplus c_7 \oplus c_8 \oplus c_{11} &= 0 \\
c_2 \oplus c_6 \oplus c_8 \oplus c_9 \oplus c_{10} &= 0 \\
c_5 \oplus c_9 \oplus c_{11} \oplus c_{12} &= 0
\end{aligned}$$

This information can be properly represented by means of the bipartite or Tanner graph, as is done in Figure 8.2 for the example under analysis.

Each row of the parity check matrix $\boldsymbol{H}$ corresponds to a parity check equation, and thus to a parity check node. Each bit of the code vector corresponds to a symbol node. Thus, for instance, the children parity check nodes of the symbol node 2 are the parity check nodes 1, 3 and 7, whereas the parent symbol nodes of the parity check node 1 are the symbol nodes 2, 4, 6, 7, 8 and 12.

The initialization of the sum–product algorithm is done by setting the two coefficients of the information to be sent from symbol nodes to parity check nodes in the first iteration,

Table 8.1 Values of the received vector and corresponding values of coefficients f_j^x

j	1	2	3	4	5	6	7	8	9	10	11	12
$\boldsymbol{r}$	+1.3129	+2.6584	+0.7413	+2.1745	+0.5981	−0.8323	−0.3962	−1.7586	+1.4905	+0.4084	−0.9290	+1.0765
$\boldsymbol{t}$	+1	+1	+1	+1	+1	−1	−1	−1	+1	−1	−1	−1
f_j^0	0.0076	0.0000	0.0467	0.0002	0.0678	0.4878	0.3751	0.3181	0.0039	0.1059	0.4967	0.0172
f_j^1	0.4619	0.0582	0.4733	0.1697	0.4396	0.0362	0.1088	0.0013	0.4132	0.3794	0.0272	0.4964

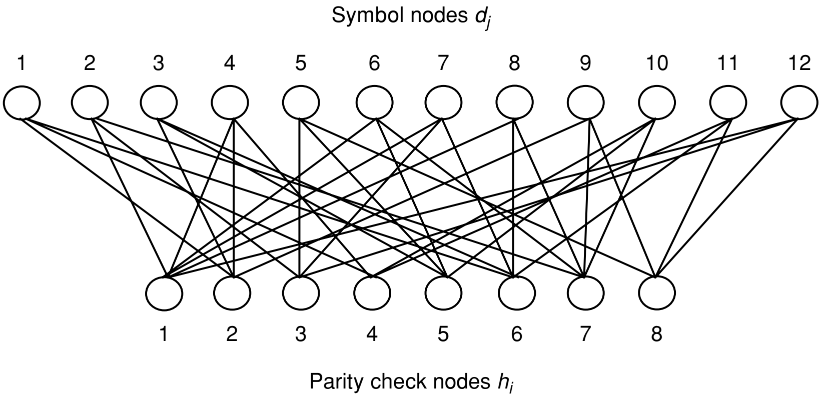


Figure 8.2 Bipartite graph for the example introduced in Section 8.5

Q_{ij}^0 and Q_{ij}^1, to be equal to the estimates that come from the channel information f_j^0 and f_j^1, respectively. Thus, for this example,

$$\begin{aligned}
Q_{12}^0 &= 0.0000 & Q_{12}^1 &= 0.0582\\
Q_{14}^0 &= 0.0002 & Q_{14}^1 &= 0.1697\\
Q_{16}^0 &= 0.4878 & Q_{16}^1 &= 0.0362\\
Q_{17}^0 &= 0.3751 & Q_{17}^1 &= 0.1088\\
Q_{18}^0 &= 0.3181 & Q_{18}^1 &= 0.0013\\
Q_{1,12}^0 &= 0.0172 & Q_{1,12}^1 &= 0.4964\\
\\
Q_{21}^0 &= 0.0076 & Q_{21}^1 &= 0.4619\\
Q_{23}^0 &= 0.0467 & Q_{23}^1 &= 0.4733\\
Q_{24}^0 &= 0.0002 & Q_{24}^1 &= 0.1697\\
Q_{29}^0 &= 0.0039 & Q_{29}^1 &= 0.4132\\
\\
Q_{32}^0 &= 0.0000 & Q_{32}^1 &= 0.0582\\
Q_{35}^0 &= 0.0678 & Q_{35}^1 &= 0.4396\\
Q_{37}^0 &= 0.3751 & Q_{37}^1 &= 0.1088\\
Q_{3,12}^0 &= 0.0172 & Q_{3,12}^1 &= 0.4964\\
\\
Q_{41}^0 &= 0.0076 & Q_{41}^1 &= 0.4619\\
Q_{44}^0 &= 0.0002 & Q_{44}^1 &= 0.1697\\
Q_{4,10}^0 &= 0.1059 & Q_{4,10}^1 &= 0.3794\\
Q_{4,11}^0 &= 0.4967 & Q_{4,11}^1 &= 0.0272
\end{aligned}$$

$$\begin{array}{llll}
Q_{53}^{0} & = 0.0467 & Q_{53}^{1} & = 0.4733 \\
Q_{55}^{0} & = 0.0678 & Q_{55}^{1} & = 0.4396 \\
Q_{56}^{0} & = 0.4878 & Q_{56}^{1} & = 0.0362 \\
Q_{5,10}^{0} & = 0.1059 & Q_{5,10}^{1} & = 0.3794
\end{array}$$

$$\begin{array}{llll}
Q_{61}^{0} & = 0.0076 & Q_{61}^{1} & = 0.4619 \\
Q_{63}^{0} & = 0.0467 & Q_{63}^{1} & = 0.4733 \\
Q_{67}^{0} & = 0.3751 & Q_{67}^{1} & = 0.1088 \\
Q_{68}^{0} & = 0.3181 & Q_{68}^{1} & = 0.0013 \\
Q_{6,11}^{0} & = 0.4967 & Q_{6,11}^{1} & = 0.0272
\end{array}$$

$$\begin{array}{llll}
Q_{72}^{0} & = 0.0000 & Q_{72}^{1} & = 0.0582 \\
Q_{76}^{0} & = 0.4878 & Q_{76}^{1} & = 0.0362 \\
Q_{78}^{0} & = 0.3181 & Q_{78}^{1} & = 0.0013 \\
Q_{79}^{0} & = 0.0039 & Q_{79}^{1} & = 0.4132 \\
Q_{7,10}^{0} & = 0.1059 & Q_{7,10}^{1} & = 0.3794
\end{array}$$

$$\begin{array}{llll}
Q_{85}^{0} & = 0.0678 & Q_{85}^{1} & = 0.4396 \\
Q_{89}^{0} & = 0.0039 & Q_{89}^{1} & = 0.4132 \\
Q_{8,11}^{0} & = 0.4967 & Q_{8,11}^{1} & = 0.0272 \\
Q_{8,12}^{0} & = 0.0172 & Q_{8,12}^{1} & = 0.4964
\end{array}$$

These initialization values allow the calculation of coefficients R_{ij}^{0} and R_{ij}^{1}, which will be iteratively updated during the decoding. These values are the estimates that go from the parity check nodes to the symbol nodes, in the bipartite graph. Thus, for example, the coefficient R_{12}^{0} is the estimate that the child parity check node 1 sends to its parent symbol node 2, and is calculated assuming that its corresponding parity check equation, which is $c_2 \oplus c_4 \oplus c_6 \oplus c_7 \oplus c_8 \oplus c_{12} = 0$, is satisfied, when the bit or symbol 2 is in state $c_2 = 0$. In this sense, there are 16 combinations (even number of '1's) of the bits c_4, c_6, c_7, c_8 and c_{12} that can satisfy such a condition. The probabilities associated to each combination are added to calculate the estimate R_{12}^{0} as

$$\begin{aligned}
R_{12}^{0} = {} & Q_{14}^{0}Q_{16}^{0}Q_{17}^{0}Q_{18}^{0}Q_{1,12}^{0} + Q_{14}^{0}Q_{16}^{0}Q_{17}^{0}Q_{18}^{1}Q_{1,12}^{1} + Q_{14}^{0}Q_{16}^{0}Q_{17}^{1}Q_{18}^{0}Q_{1,12}^{1} + Q_{14}^{0}Q_{16}^{0}Q_{17}^{1}Q_{18}^{1}Q_{1,12}^{0} \\
& + Q_{14}^{0}Q_{16}^{1}Q_{17}^{0}Q_{18}^{0}Q_{1,12}^{1} + Q_{14}^{0}Q_{16}^{1}Q_{17}^{0}Q_{18}^{1}Q_{1,12}^{0} + Q_{14}^{0}Q_{16}^{1}Q_{17}^{1}Q_{18}^{0}Q_{1,12}^{0} + Q_{14}^{0}Q_{16}^{1}Q_{17}^{1}Q_{18}^{1}Q_{1,12}^{1} \\
& + Q_{14}^{1}Q_{16}^{0}Q_{17}^{0}Q_{18}^{0}Q_{1,12}^{1} + Q_{14}^{1}Q_{16}^{0}Q_{17}^{0}Q_{18}^{1}Q_{1,12}^{0} + Q_{14}^{1}Q_{16}^{0}Q_{17}^{1}Q_{18}^{0}Q_{1,12}^{0} + Q_{14}^{1}Q_{16}^{0}Q_{17}^{1}Q_{18}^{1}Q_{1,12}^{1} \\
& + Q_{14}^{1}Q_{16}^{1}Q_{17}^{0}Q_{18}^{0}Q_{1,12}^{0} + Q_{14}^{1}Q_{16}^{1}Q_{17}^{0}Q_{18}^{1}Q_{1,12}^{1} + Q_{14}^{1}Q_{16}^{1}Q_{17}^{1}Q_{18}^{0}Q_{1,12}^{1} + Q_{14}^{1}Q_{16}^{1}Q_{17}^{1}Q_{18}^{1}Q_{1,12}^{0} \\
= {} & 0.0051
\end{aligned}$$

In the same way the coefficient R_{12}^{1} is the estimate that the child parity check node 1 sends to its parent symbol node 2, and is calculated assuming that its corresponding parity check equation, which is $c_2 \oplus c_4 \oplus c_6 \oplus c_7 \oplus c_8 \oplus c_{12} = \mathit{0}$, is satisfied, when the bit or symbol 2 is in state $c_2 = 1$. In this sense, there are again 16 combinations (odd number of '1's) of the bits c_4, c_6, c_7, c_8 and c_{12} that can satisfy such a condition. The probabilities associated with each

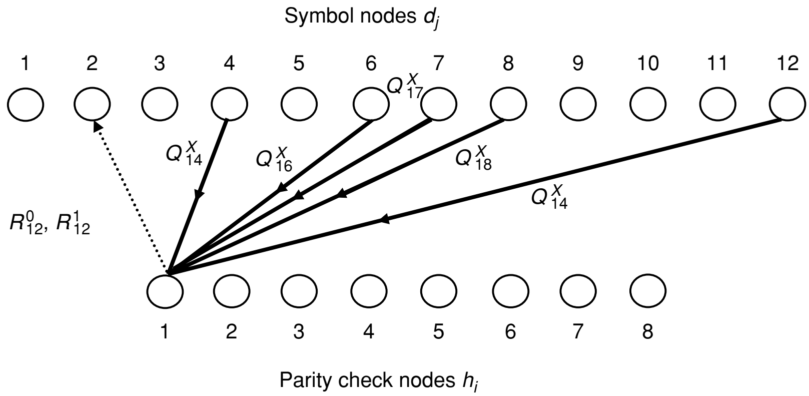


Figure 8.3 Calculation of coefficients R_{12}^{0} and R_{12}^{1}

combination are added to calculate the estimate R_{12}^{1} as

$$\begin{aligned} R_{12}^{1} = {} & Q_{14}^{0}Q_{16}^{0}Q_{17}^{0}Q_{18}^{0}Q_{1,12}^{1} + Q_{14}^{0}Q_{16}^{0}Q_{17}^{0}Q_{18}^{1}Q_{1,12}^{0} + Q_{14}^{0}Q_{16}^{0}Q_{17}^{1}Q_{18}^{0}Q_{1,12}^{0} + Q_{14}^{0}Q_{16}^{0}Q_{17}^{1}Q_{18}^{1}Q_{1,12}^{1} \\ & + Q_{14}^{0}Q_{16}^{1}Q_{17}^{0}Q_{18}^{0}Q_{1,12}^{0} + Q_{14}^{0}Q_{16}^{1}Q_{17}^{0}Q_{18}^{1}Q_{1,12}^{1} + Q_{14}^{0}Q_{16}^{1}Q_{17}^{1}Q_{18}^{0}Q_{1,12}^{1} + Q_{14}^{0}Q_{16}^{1}Q_{17}^{1}Q_{18}^{1}Q_{1,12}^{0} \\ & + Q_{14}^{1}Q_{16}^{0}Q_{17}^{0}Q_{18}^{0}Q_{1,12}^{0} + Q_{14}^{1}Q_{16}^{0}Q_{17}^{0}Q_{18}^{1}Q_{1,12}^{1} + Q_{14}^{1}Q_{16}^{0}Q_{17}^{1}Q_{18}^{0}Q_{1,12}^{1} + Q_{14}^{1}Q_{16}^{0}Q_{17}^{1}Q_{18}^{1}Q_{1,12}^{0} \\ & + Q_{14}^{1}Q_{16}^{1}Q_{17}^{0}Q_{18}^{0}Q_{1,12}^{1} + Q_{14}^{1}Q_{16}^{1}Q_{17}^{0}Q_{18}^{1}Q_{1,12}^{0} + Q_{14}^{1}Q_{16}^{1}Q_{17}^{1}Q_{18}^{0}Q_{1,12}^{0} + Q_{14}^{1}Q_{16}^{1}Q_{17}^{1}Q_{18}^{1}Q_{1,12}^{1} \\ = {} & 0.0020 \end{aligned}$$

The process of interchanging information can be seen in Figure 8.3.

The node that is updated or informed does not participate in the calculation of the corresponding estimate. This makes the iterative decoding converge to the right solution. The larger the number of '1's in each row of the parity check matrix, the larger is the number of combinations of the bits needed to calculate the coefficients R_{ij}^{0} and R_{ij}^{1}. Tables 8.2 and 8.3 show the values of these coefficients in the form of a matrix, where index i corresponds to a row, and index j corresponds to a column. Table 8.2 represents coefficients R_{ij}^{0} that are the estimates for the bit or symbol $x = 0$, and Table 8.3 shows the values of coefficients R_{ij}^{1} that are the estimates for the bit or symbol $x = 1$.

To clarify notation, the value of R_{ij}^{0} for indexes $i = 4$, $j = 10$ is equal to $R_{4,10}^{0} = 0.0390$. Values of coefficients R_{ij}^{0} and R_{ij}^{1} allow us to determine the first estimate of the decoded vector

Table 8.2 Values of coefficients R_{ij}^{0}, first iteration

	1	2	3	4	5	6	7	8	9	10	11	12
1		0.0051		0.0017		0.0002	0.0001	0.0004				0.0006
2	0.0036		0.0009	0.0113					0.0043			
3		0.0868			0.0109		0.0024					0.0100
4	0.0325			0.0889						0.0390	0.0088	
5			0.0875		0.0925	0.0423				0.1049		
6	0.0126		0.0100				0.0348	0.0431			0.0270	
7		0.0250				0.0008		0.0016	0.0035	0.0037		
8					0.1022				0.1101		0.0179	0.0912

Table 8.3 Values of coefficients R_{ij}^1, first iteration

	1	2	3	4	5	6	7	8	9	10	11	12
1		0.0020		0.0007		0.0006	0.0008	0.0009				0.0002
2	0.0332		0.0324	0.0906					0.0372			
3		0.0393			0.0035		0.0128					0.0043
4	0.0107			0.0305						0.0028	0.0299	
5			0.0416		0.0398	0.0857				0.0333		
6	0.0295		0.0280				0.0060	0.0188			0.0107	
7		0.0089				0.0029		0.0046	0.0012	0.0003		
8					0.0101				0.0264		0.0908	0.0197

$\hat{\boldsymbol{d}}$, using expression (14). Thus,

$$\hat{d}_1 = \left\{ \begin{array}{l} \hat{0} \rightarrow f_1^0 \times R_{21}^0 \times R_{41}^0 \times R_{61}^0 = 1.13 \times 10^{-8} \\ \hat{1} \rightarrow f_1^1 \times R_{21}^1 \times R_{41}^1 \times R_{61}^1 = 4.85 \times 10^{-6} \end{array} \right\} \Rightarrow \ '1'$$

$$\hat{d}_2 = \left\{ \begin{array}{l} \hat{0} \rightarrow f_2^0 \times R_{12}^0 \times R_{32}^0 \times R_{72}^0 = 1.58 \times 10^{-10} \\ \hat{1} \rightarrow f_2^1 \times R_{12}^1 \times R_{32}^1 \times R_{72}^1 = 4.06 \times 10^{-8} \end{array} \right\} \Rightarrow \ '1'$$

$$\hat{d}_3 = \left\{ \begin{array}{l} \hat{0} \rightarrow f_3^0 \times R_{23}^0 \times R_{53}^0 \times R_{63}^0 = 3.59 \times 10^{-8} \\ \hat{1} \rightarrow f_3^1 \times R_{23}^1 \times R_{53}^1 \times R_{63}^1 = 1.785 \times 10^{-5} \end{array} \right\} \Rightarrow \ '1'$$

$$\hat{d}_4 = \left\{ \begin{array}{l} \hat{0} \rightarrow f_4^0 \times R_{14}^0 \times R_{24}^0 \times R_{44}^0 = 3.31 \times 10^{-10} \\ \hat{1} \rightarrow f_4^1 \times R_{14}^1 \times R_{24}^1 \times R_{44}^1 = 3.19 \times 10^{-7} \end{array} \right\} \Rightarrow \ '1'$$

$$\hat{d}_5 = \left\{ \begin{array}{l} \hat{0} \rightarrow f_5^0 \times R_{35}^0 \times R_{55}^0 \times R_{85}^0 = 7.007 \times 10^{-6} \\ \hat{1} \rightarrow f_5^1 \times R_{35}^1 \times R_{55}^1 \times R_{85}^1 = 6.20 \times 10^{-7} \end{array} \right\} \Rightarrow \ '0'$$

$$\hat{d}_6 = \left\{ \begin{array}{l} \hat{0} \rightarrow f_6^0 \times R_{16}^0 \times R_{56}^0 \times R_{76}^0 = 3.39 \times 10^{-9} \\ \hat{1} \rightarrow f_6^1 \times R_{16}^1 \times R_{56}^1 \times R_{76}^1 = 5.34 \times 10^{-9} \end{array} \right\} \Rightarrow \ '1'$$

$$\hat{d}_7 = \left\{ \begin{array}{l} \hat{0} \rightarrow f_7^0 \times R_{17}^0 \times R_{37}^0 \times R_{67}^0 = 2.73 \times 10^{-9} \\ \hat{1} \rightarrow f_7^1 \times R_{17}^1 \times R_{37}^1 \times R_{67}^1 = 6.37 \times 10^{-9} \end{array} \right\} \Rightarrow \ '1'$$

$$\hat{d}_8 = \left\{ \begin{array}{l} \hat{0} \rightarrow f_8^0 \times R_{18}^0 \times R_{68}^0 \times R_{78}^0 = 7.96 \times 10^{-9} \\ \hat{1} \rightarrow f_8^1 \times R_{18}^1 \times R_{68}^1 \times R_{78}^1 = 1.03 \times 10^{-10} \end{array} \right\} \Rightarrow \ '0'$$

$$\hat{d}_9 = \left\{ \begin{array}{l} \hat{0} \rightarrow f_9^0 \times R_{29}^0 \times R_{79}^0 \times R_{89}^0 = 6.52 \times 10^{-9} \\ \hat{1} \rightarrow f_9^1 \times R_{29}^1 \times R_{79}^1 \times R_{89}^1 = 4.98 \times 10^{-7} \end{array} \right\} \Rightarrow \ '1'$$

$$\hat{d}_{10} = \left\{ \begin{array}{l} \hat{0} \rightarrow f_{10}^0 \times R_{4,10}^0 \times R_{5,10}^0 \times R_{7,10}^0 = 1.62 \times 10^{-6} \\ \hat{1} \rightarrow f_{10}^1 \times R_{4,10}^1 \times R_{5,10}^1 \times R_{7,10}^1 = \ 1.17 \times 10^{-8} \end{array} \right\} \Rightarrow \ '0'$$

$$\hat{d}_{11} = \left\{ \begin{array}{l} \hat{0} \rightarrow f_{11}^0 \times R_{4,11}^0 \times R_{6,11}^0 \times R_{8,11}^0 = 2.12 \times 10^{-6} \\ \hat{1} \rightarrow f_{11}^1 \times R_{4,11}^1 \times R_{6,11}^1 \times R_{8,11}^1 = \ 7.91 \times 10^{-7} \end{array} \right\} \Rightarrow \ '0'$$

$$\hat{d}_{12} = \left\{ \begin{array}{l} \hat{0} \rightarrow f_{12}^0 \times R_{1,12}^0 \times R_{3,12}^0 \times R_{8,12}^0 = 9.23 \times 10^{-9} \\ \hat{1} \rightarrow f_{12}^1 \times R_{1,12}^1 \times R_{3,12}^1 \times R_{8,12}^1 = \ 8.87 \times 10^{-9} \end{array} \right\} \Rightarrow \ '1'$$

The first estimate of the decoded vector is

$$\hat{\boldsymbol{d}} = \begin{pmatrix} 1\ 1\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 0 \end{pmatrix}$$

which contains three errors with respect to the transmitted code vector $\boldsymbol{c} =$ (1 1 1 1 1 0 0 0 1 0 0 0)

The decoding process continues since the syndrome for this estimated vector is not the all-zero vector.

The next iteration starts with the calculation of the values of coefficients Q_{ij}^0 and Q_{ij}^1. These values are determined using expression (13), where there are normalizing coefficients so that the condition $Q_{ij}^0 + Q_{ij}^1 = 1$ is satisfied. Thus, for example, the value of the coefficient Q_{12}^0 is the estimate that the parent symbol node 2 sends to child parity check node 1, calculated by forming the product of the estimates R_{k2}^0 of all its children parity check nodes, excepting that of child node 1, which is the node that is being updated. In the same way, the value of the coefficient Q_{12}^1 is the estimate that the parent symbol node 2 sends to child parity check node 1, calculated by forming the product of the estimates R_{k2}^1 of all its children parity check nodes, excepting that of child node 1, which is the node that is being updated. In this notation, k is the index of the child node, that is, the index of each of the parity check nodes that are connected in the bipartite graph to the parent node 2. The calculated coefficients are normalized by using the normalizing constants.

The values of coefficients Q_{12}^0 and Q_{12}^1, in this example, are calculated as follows:

$$Q_{12}^0 = \alpha_{12} f_2^0 R_{32}^0 R_{72}^0$$

$$Q_{12}^1 = \alpha_{12} f_2^1 R_{32}^1 R_{72}^1$$

where

$$\alpha_{12} = \frac{1}{f_2^0 R_{32}^0 R_{72}^0 + f_2^1 R_{32}^1 R_{72}^1}$$

Figure 8.4 shows the flow of information and nodes participating in the calculation of coefficients Q_{12}^0 and Q_{12}^1 for this example.

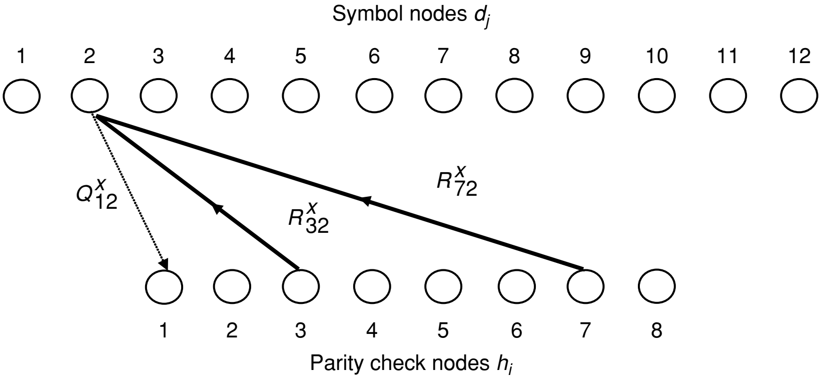


Figure 8.4 Calculation of the values of coefficients Q_{12}^0 and Q_{12}^1

Table 8.4 Values of the coefficients Q_{ij}^0, second iteration

	1	2	3	4	5	6	7	8	9	10	11	12
1		0.0015		0.0004		0.6601	0.7898	0.9950				0.2731
2	0.0209		0.0691	0.0083					0.1014			
3		0.0018			0.7843		0.6946					0.3068
4	0.0008			0.0004						0.9091	0.9008	
5			0.0010		0.8294	0.5620				0.9777		
6	0.0054		0.0056				0.0688	0.9710			0.5147	
7		0.0014				0.6847		0.9954	0.0046	0.9239		
8					0.5273				0.0031		0.9316	0.1838

Tables 8.4 and 8.5 show the calculated values of coefficients Q_{ij}^0 and Q_{ij}^1, respectively.

In the second iteration, the updated values of coefficients Q_{ij}^0 and Q_{ij}^1 allow us to determine the updated values of coefficients R_{ij}^0 and R_{ij}^1. This iteration is different from the first iteration in the sense that coefficients Q_{ij}^0 and Q_{ij}^1 now contain updated information, rather than simply channel information. In this second iteration the calculation of values of coefficients R_{ij}^0 and R_{ij}^1 leads to a new estimated decoded vector $\hat{\boldsymbol{d}}$, which again does not satisfy the syndrome condition, and so the decoding process continues.

In the example under analysis, the decoder is able to find the correct code vector after three iterations, and is also able to correct the two errors that the hard-decision received vector contained. In this particular case the errors are in the message part of the code vector, that is, in two of the four bits that are finally taken as the message bits, after truncating the redundancy. The iterative decoding algorithm is able to correct these two errors. Tables 8.6 and 8.7 illustrate the evolution of the decoding algorithm by presenting the values of the coefficients involved, until arriving at the final solution. These tables show values that are truncated to four decimal places, though actual values were determined in a more accurate way by using MATLAB® Program 5.3.

The updated values of coefficients R_{ij}^0 and R_{ij}^1 allow us to determine a new estimated decoded vector. This second estimate of the decoded vector is shown in Table 8.8.

Table 8.5 Values of the coefficients Q_{ij}^1, second iteration

	1	2	3	4	5	6	7	8	9	10	11	12
1		0.9985		0.9996		0.3399	0.2102	0.0050				0.7269
2	0.9791		0.9309	0.9917					0.8986			
3		0.9982			0.2157		0.3054					0.6932
4	0.9992			0.9996						0.0909	0.0992	
5			0.9990		0.1706	0.4380				0.0223		
6	0.9946		0.9944				0.9312	0.0290			0.4853	
7		0.9986				0.3153		0.0046	0.9954	0.0761		
8					0.4727				0.9969		0.0684	0.8162

Table 8.6 Values of coefficients R^0_{ij}, second iteration

	1	2	3	4	5	6	7	8	9	10	11	12
1		0.5416		0.5415		0.3704	0.4284	0.4581				0.5915
2	0.1622		0.1244	0.1709					0.0940			
3		0.4572			0.5750		0.6095					0.3897
4	0.1723			0.1726						0.8999	0.9081	
5			0.5390		0.4409	0.1859				0.4593		
6	0.5118		0.5118				0.5135	0.4876			0.1027	
7		0.3463				0.9150		0.6547	0.3453	0.6808		
8					0.7713				0.4851		0.5172	0.4766

According to the values shown in Table 8.8, the estimated decoded vector in the second iteration is $\hat{\boldsymbol{d}} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$, which contains only one error, in the last bit, with respect to the true code vector. This estimated decoded vector produces a non-zero syndrome so that the decoder proceeds to the third iteration. Again, values of coefficients Q^0_{ij} and Q^1_{ij} are updated, and they are shown in Tables 8.9 and 8.10.

The updated values of coefficients R^0_{ij} and R^1_{ij} of this third iteration are shown in Tables 8.11 and 8.12.

With these values, a new estimate of the decoded vector is formed, as given in Table 8.13.

According to the values shown in Table 8.8, the estimated decoded vector after the third iteration is $\hat{\boldsymbol{d}} = \boldsymbol{c} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$, whose syndrome vector is the all-zero vector, and so the decoder decides that this is a code vector and the decoded message vector is

$$\boldsymbol{m} = \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix}$$

Thus, the iterative decoding algorithm was able to correctly decode the received vector of 12 bits of this example. The minimum Hamming distance of the block code of this example is $d_{\min} = 4$, as determined by inspection of the minimum weight among all the non-zero code vectors, or equivalently, by noting that there are four columns (columns 3, 4, 8, and 10 for instance) in the corresponding parity check matrix $\boldsymbol{H}$ that when added result in the all-zero vector. This allows us to say that this code is able, using hard-decision decoding, to correct

Table 8.7 Values of coefficients R^1_{ij}, second iteration

	1	2	3	4	5	6	7	8	9	10	11	12
1		0.4584		0.4585		0.6296	0.5716	0.5419				0.4085
2	0.8378		0.8756	0.8291					0.9060			
3		0.5428			0.4250		0.3905					0.6103
4	0.8277			0.8274						0.1001	0.0919	
5			0.4610		0.5591	0.8141				0.5407		
6	0.4882		0.4882				0.4865	0.5124			0.8973	
7		0.6537				0.0850		0.3453	0.6547	0.3192		
8					0.2287				0.5149		0.4828	0.5234

Table 8.8 Estimate of the decoded vector after the second iteration

$\boldsymbol{r}$	+1.3129	+2.6584	+0.7413	+2.1745	+0.5981	−0.8323	−0.3962	−1.7586	+1.4905	+0.4084	−0.9290	+1.0765
$\boldsymbol{t}$	+1	+1	+1	+1	+1	−1	−1	−1	+1	−1	−1	−1
d_j^0	0.0001	0.0000	0.0016	0.0000	0.0133	0.0307	0.0503	0.0465	0.0001	0.0298	0.0240	0.0019
d_j^1	0.1564	0.0095	0.0933	0.0534	0.0239	0.0016	0.0118	0.0001	0.1262	0.0066	0.0011	0.0648

Table 8.9 Values of coefficients Q_{ij}^0, third iteration

	1	2	3	4	5	6	7	8	9	10	11	12
1		0.0001		0.0000		0.9707	0.8503	0.9977				0.0197
2	0.0036		0.1078	0.0003					0.0047			
3		0.0002			0.2909		0.7318					0.0436
4	0.0033			0.0003						0.3358	0.6910	
5			0.0145		0.4130	0.9884				0.8426		
6	0.0007		0.0161				0.8013	0.9974			0.9948	
7		0.0002				0.6442		0.9949	0.0009	0.6807		
8					0.1413				0.0005		0.9538	0.0310

Table 8.10 Values of coefficients Q_{ij}^1, third iteration

	1	2	3	4	5	6	7	8	9	10	11	12
1		0.9999		1.0000		0.0293	0.1497	0.0023				0.9803
2	0.9964		0.8922	0.9997					0.9953			
3		0.9998			0.7091		0.2682					0.9564
4	0.9967			0.9997						0.6642	0.3090	
5			0.9855		0.5870	0.0116				0.1574		
6	0.9993		0.9839				0.1987	0.0026			0.0052	
7		0.9998				0.3558		0.0051	0.9991	0.3193		
8					0.8587				0.9995		0.0462	0.9690

Table 8.11 Values of coefficients R_{ij}^0, third iteration

	1	2	3	4	5	6	7	8	9	10	11	12
1		0.8153		0.8153		0.1651	0.0501	0.1833				0.8282
2	0.1117		0.0085	0.1143					0.1109			
3		0.5885			0.7115		0.3092					0.5969
4	0.5627			0.5623						0.6896	0.3370	
5			0.4418		0.1750	0.5579				0.5825		
6	0.2129		0.2037				0.9758	0.7882			0.7897	
7		0.4485				0.6784		0.5520	0.4485	0.6424		
8					0.9252				0.8053		0.1639	0.8252

Table 8.12 Values of coefficients R_{ij}^1, third iteration

	1	2	3	4	5	6	7	8	9	10	11	12
1		0.1847		0.1847		0.8349	0.9499	0.8167				0.1718
2	0.8883		0.9915	0.8857					0.8891			
3		0.4115			0.2885		0.6908					0.4031
4	0.4373			0.4377						0.3104	0.6630	
5			0.5582		0.8250	0.4421				0.4175		
6	0.7871		0.7963				0.0242	0.2118			0.2103	
7		0.5515				0.3216		0.4480	0.5515	0.3576		
8					0.0748				0.1947		0.8361	0.1748

Table 8.13 Estimate of the decoded vector after the third iteration

$\boldsymbol{r}$	+1.3129	+2.6584	+0.7413	+2.1745	+0.5981	−0.8323	−0.3962	−1.7586	+1.4905	+0.4084	−0.9290	+1.0765
$\boldsymbol{t}$	+1	+1	+1	+1	+1	−1	−1	−1	+1	−1	−1	−1
d_j^0	0.0001	0.0000	0.0000	0.0000	0.0077	0.0305	0.0057	0.0254	0.0002	0.0273	0.0217	0.0070
d_j^1	0.1412	0.0024	0.2086	0.0122	0.0078	0.0043	0.0017	0.0001	0.0394	0.0176	0.0032	0.0060

any error pattern of size $t = 1$, with a residual error-correction capability that can be used to correct some error patterns of size larger than $t = 1$. Using a soft-decision decoding algorithm, the code can correct a double error pattern, as the above example confirms. In fact, using the sum–product decoding algorithm, the code is capable of correcting almost all possible double error patterns.

However, the code rate in this example is $R_c = 1/3$, and according to a rather simple estimate, the product $R_c(t + 1)$ gives an indication if the error-correction capability of a block code is efficient or not, when it is compared with uncoded transmission. A given block code performs better than uncoded transmission if the product $R_c(t + 1)$ satisfies the condition $R_c(t + 1) > 1$. In this example, the product is equal to $(1/3) \times 2 = 0.667$, so that the use of this simple code would not produce a particular advantage with respect to uncoded transmission. This is not surprising, because the example was introduced only for the purpose of explaining in some detail how the sum–product algorithm operates. In addition, it is rather difficult to design good, small, sparse parity check matrices with at least three '1's per column, and a rather small number of '1's per row, as required in the classic construction of an LDPC or Gallager code. Apart from this constraint, the predicted poor overall performance of this simple code is due to the fact that its Tanner graph (see Figure 8.2) contains several cycles of length 4 (the shortest possible cycle length). As has already been mentioned, this degrades the performance of an iterative soft-decision decoding algorithm like the sum–product algorithm. Cycles of length 4 are avoided if the corresponding parity check matrix $\boldsymbol{H}$ does not contain rectangular patterns of '1's, as seen in the following matrix, which is another example of a parity check matrix $\boldsymbol{H}$ with length 4 cycles in its corresponding bipartite graph [13]:

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

Similar rectangular patterns are of course also seen in the $\boldsymbol{H}$ matrix of the previously analysed code, corresponding to the bipartite graph of Figure 8.2.

Figure 8.5 shows the BER performance of the irregular LDPC code $C_\mathrm{b}(60, 30)$ with rate $R_\mathrm{c} = 1/2$, where the effect of varying the maximum or predetermined number of iterations of the sum–product decoding algorithm is clear.

LDPC codes of course behave in agreement with Shannon's predictions, and so they perform better for larger code lengths n. In the case of large-code length LDPC codes and for a sufficient number of iterations, the BER performance of LDPC codes is close to the Shannon limits.

8.6 Simplifications of the Sum–Product Algorithm

The aim of the sum–product decoding algorithm is to find a decoded vector $\boldsymbol{d}$, which is an estimate of the code vector actually transmitted, $\boldsymbol{c}$, able to satisfy the syndrome condition:

$$\boldsymbol{H} \circ \boldsymbol{d} = \boldsymbol{0}$$

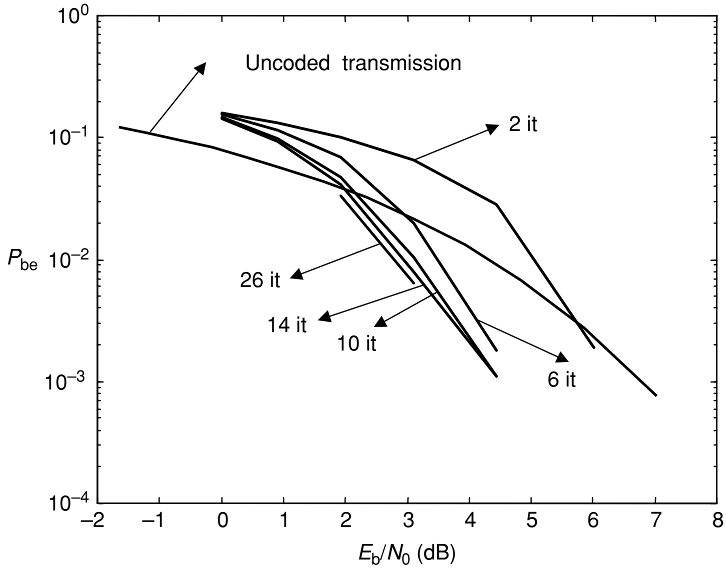


Figure 8.5 BER performance of the irregular LDPC code $C_b(60, 30)$ of rate $R_c = 1/2$, as a function of the number of iterations

As described in Section 8.3.3, in the sum–product algorithm, each symbol node d_j sends to each child parity check node h_i the estimate Q_{ij}^x, based on the information provided by the other children parity check nodes that the corresponding parity check node is in state x. On the other hand, each parity check node h_i sends to each parent symbol node d_j the estimate R_{ij}^x, calculated with the information provided by the other symbol nodes, indicating that the corresponding parity check equation i is satisfied if the symbol node is in state x.

The channel information can be determined by using the following expression:

$$f_j^1 = \frac{1}{1 + e^{-\frac{2Ay_j}{\sigma^2}}} \tag{17}$$

so that

$$f_j^0 = 1 - f_j^1 \tag{18}$$

where y_j is the channel output at time instant j, and bits are transmitted in polar format with amplitudes $\pm A$. In general, in this text, the normalized polar format ± 1 is utilized.

As has been seen in the example of Section 8.5, values of coefficients R_{ij}^0 and R_{ij}^1 are determined as a function of the values of coefficients Q_{ij}^0 and Q_{ij}^1 by taking into account all the combinations of code bits that satisfy the parity check equation related to that calculation. However, Mackay and Neal introduced in their paper [4] a calculation method that avoids having to take into account all these possibilities of the parity check equation in the calculation of values of coefficients R_{ij}^0 and R_{ij}^1.

The initialization of this modified version of the sum–product algorithm is the same as that used in the traditional form of the algorithm, as performed for the example shown in Section 8.5.

$$Q_{ij}^0 = f_j^0 \quad \text{and} \quad Q_{ij}^1 = f_j^1 \tag{19}$$

The modified version carries out the iterative calculation by implementing two steps, the horizontal and the vertical steps, according to the form in which the values are taken from the corresponding parity check matrix $\boldsymbol{H}$. The quantity δQ_{ij} is calculated as

$$\delta Q_{ij} = Q_{ij}^0 - Q_{ij}^1 \tag{20}$$

and the quantity δR_{ij} is also defined for a pair of values or nodes i and j as

$$\delta R_{ij} = \prod_{j' \in N(i) \backslash j} \delta Q_{ij'} \tag{21}$$

Remember that in these expressions $N(i)$ represents the set of indexes of all the parent symbol nodes connected to the parity check node h_i, whereas $N(i) \backslash j$ represents the same set with the exclusion of the parent symbol node d_j.

Coefficients R_{ij}^0 and R_{ij}^1 are then calculated by performing

$$R_{ij}^0 = 1/2 \left(1 + \delta R_{ij}\right) \tag{22}$$

and

$$R_{ij}^1 = 1/2 \left(1 - \delta R_{ij}\right) \tag{23}$$

The coefficients Q_{ij}^0 and Q_{ij}^1 are updated in the vertical step. They are determined for every pair of nodes i, j, and for every possible value of x, which in the binary case are $x = 0$ or $x = 1$, as follows:

$$Q_{ij}^x = \alpha_{ij} f_j^x \prod_{i' \in M(j) \backslash i} R_{i'j}^x \tag{24}$$

where $M(j)$ represents the set of indexes of all the children parity check nodes connected to the symbol node d_j, whereas $M(j) \backslash i$ represents the same set with the exclusion of the children parity check node h_i.

The coefficient f_j^x is the *a priori* probability that the symbol node d_j is in state x. Constant α_{ij} is selected so that $Q_{ij}^0 + Q_{ij}^1 = 1$.

The estimate of the decoded vector for the current iteration requires the calculation of the coefficients or *a posteriori probabilities* Q_j^0 and Q_j^1, which are equal to

$$Q_j^x = \alpha_j f_j^x \prod_{i \in M(j)} R_{ij}^x \tag{25}$$

where, once again, constant α_j is selected so that $Q_j^0 + Q_j^1 = 1$.

The estimate of the decoded vector $\hat{\boldsymbol{d}}$ can be finally obtained by calculating

$$\widehat{d}_j = \max\left(Q_j^x\right) \tag{26}$$

which means that

$$\text{if } Q_j^0 > Q_j^1 \text{ then } \widehat{d}_j = 0, \ \text{ else } \widehat{d}_j = 1 \tag{27}$$

Example 8.1: Apply the Mackay–Neal simplified sum–product decoding algorithm to the example of Section 8.5, in order to see the equivalence of this method with respect to the traditional sum–product decoding algorithm.

The Mackay–Neal simplified sum–product decoding algorithm avoids having to take into account all the combinations or possibilities that the corresponding parity check equation, associated with the calculation of the involved coefficient, is satisfied. The initialization procedure is the same as that of the traditional algorithm, and it is the same for this example as that used in Section 8.5. This means that, according to the values of Table 8.1, the initialization sets $Q_{ij}^0 = f_j^0$ and $Q_{ij}^1 = f_j^1$.

The modified method is applied by forming, with the corresponding values, the following calculation matrices:

$$\boldsymbol{H_{Q0}} = \begin{bmatrix} 0 & Q_{12}^0 & 0 & Q_{14}^0 & 0 & Q_{16}^0 & Q_{17}^0 & Q_{18}^0 & 0 & 0 & 0 & Q_{1,12}^0 \\ Q_{21}^0 & 0 & Q_{23}^0 & Q_{24}^0 & 0 & 0 & 0 & 0 & Q_{29}^0 & 0 & 0 & 0 \\ 0 & Q_{32}^0 & 0 & 0 & Q_{35}^0 & 0 & Q_{37}^0 & 0 & 0 & 0 & 0 & Q_{3,12}^0 \\ Q_{41}^0 & 0 & 0 & Q_{44}^0 & 0 & 0 & 0 & 0 & 0 & Q_{4,10}^0 & Q_{4,11}^0 & 0 \\ 0 & 0 & Q_{53}^0 & 0 & Q_{55}^0 & Q_{56}^0 & 0 & 0 & 0 & Q_{5,10}^0 & 0 & 0 \\ Q_{61}^0 & 0 & Q_{63}^0 & 0 & 0 & 0 & Q_{67}^0 & Q_{68}^0 & 0 & 0 & Q_{6,11}^0 & 0 \\ 0 & Q_{72}^0 & 0 & 0 & 0 & Q_{76}^0 & 0 & Q_{78}^0 & Q_{79}^0 & Q_{7,10}^0 & 0 & 0 \\ 0 & 0 & 0 & 0 & Q_{85}^0 & 0 & 0 & 0 & Q_{89}^0 & 0 & Q_{8,11}^0 & Q_{8,12}^0 \end{bmatrix}$$

$$\boldsymbol{H_{Q1}} = \begin{bmatrix} 0 & Q_{12}^1 & 0 & Q_{14}^1 & 0 & Q_{16}^1 & Q_{17}^1 & Q_{18}^1 & 0 & 0 & 0 & Q_{1,12}^1 \\ Q_{21}^1 & 0 & Q_{23}^1 & Q_{24}^1 & 0 & 0 & 0 & 0 & Q_{29}^1 & 0 & 0 & 0 \\ 0 & Q_{32}^1 & 0 & 0 & Q_{35}^1 & 0 & Q_{37}^1 & 0 & 0 & 0 & 0 & Q_{3,12}^1 \\ Q_{41}^1 & 0 & 0 & Q_{44}^1 & 0 & 0 & 0 & 0 & 0 & Q_{4,10}^1 & Q_{4,11}^1 & 0 \\ 0 & 0 & Q_{53}^1 & 0 & Q_{55}^1 & Q_{56}^1 & 0 & 0 & 0 & Q_{5,10}^1 & 0 & 0 \\ Q_{61}^1 & 0 & Q_{63}^1 & 0 & 0 & 0 & Q_{67}^1 & Q_{68}^1 & 0 & 0 & Q_{6,11}^1 & 0 \\ 0 & Q_{72}^1 & 0 & 0 & 0 & Q_{76}^1 & 0 & Q_{78}^1 & Q_{79}^1 & Q_{7,10}^1 & 0 & 0 \\ 0 & 0 & 0 & 0 & Q_{85}^1 & 0 & 0 & 0 & Q_{89}^1 & 0 & Q_{8,11}^1 & Q_{8,12}^1 \end{bmatrix}$$

These matrices can be subtracted to form the difference matrix

$$\boldsymbol{H_{\delta Q}} = \begin{bmatrix} 0 & \delta Q_{12} & 0 & \delta Q_{14} & 0 & \delta Q_{16} & \delta Q_{17} & \delta Q_{18} & 0 & 0 & 0 & \delta Q_{1,12} \\ \delta Q_{21} & 0 & \delta Q_{23} & \delta Q_{24} & 0 & 0 & 0 & 0 & \delta Q_{29} & 0 & 0 & 0 \\ 0 & \delta Q_{32} & 0 & 0 & \delta Q_{35} & 0 & \delta Q_{37} & 0 & 0 & 0 & 0 & \delta Q_{3,12} \\ \delta Q_{41} & 0 & 0 & \delta Q_{44} & 0 & 0 & 0 & 0 & 0 & \delta Q_{4,10} & \delta Q_{4,11} & 0 \\ 0 & 0 & \delta Q_{53} & 0 & \delta Q_{55} & \delta Q_{56} & 0 & 0 & 0 & \delta Q_{5,10} & 0 & 0 \\ \delta Q_{61} & 0 & \delta Q_{63} & 0 & 0 & 0 & \delta Q_{67} & \delta Q_{68} & 0 & 0 & \delta Q_{6,11} & 0 \\ 0 & \delta Q_{72} & 0 & 0 & 0 & \delta Q_{76} & 0 & \delta Q_{78} & \delta Q_{79} & \delta Q_{7,10} & 0 & 0 \\ 0 & 0 & 0 & 0 & \delta Q_{85} & 0 & 0 & 0 & \delta Q_{89} & 0 & \delta Q_{8,11} & \delta Q_{8,12} \end{bmatrix}$$

An example of the calculation of the parity check equation of the second row of the corresponding difference matrix $\boldsymbol{H}_{\delta Q}$ is done here to illustrate the equivalence of this method with respect to the traditional one. Thus, for instance, the value of the coefficient R_{21}^0 is calculated as

$$\begin{aligned}
\delta R_{21} &= \delta Q_{23}\,\delta Q_{24}\,\delta Q_{29} \\
&= \left(Q_{23}^0 - Q_{23}^1\right)\left(Q_{24}^0 - Q_{24}^1\right)\left(Q_{29}^0 - Q_{29}^1\right) \\
&= \left(Q_{23}^0 Q_{24}^0 - Q_{23}^0 Q_{24}^1 - Q_{23}^1 Q_{24}^0 + Q_{23}^1 Q_{24}^1\right)\left(Q_{29}^0 - Q_{29}^1\right) \\
&= Q_{23}^0 Q_{24}^0 Q_{29}^0 - Q_{23}^0 Q_{24}^1 Q_{29}^0 - Q_{23}^1 Q_{24}^0 Q_{29}^0 + Q_{23}^1 Q_{24}^1 Q_{29}^0 - Q_{23}^0 Q_{24}^0 Q_{29}^1 \\
&+ Q_{23}^0 Q_{24}^1 Q_{29}^1 + Q_{23}^1 Q_{24}^0 Q_{29}^1 - Q_{23}^1 Q_{24}^1 Q_{29}^1
\end{aligned}$$

By taking into account that

$$Q_{24}^0 = 1 - Q_{24}^1 \quad \text{and} \quad Q_{29}^0 = 1 - Q_{29}^1$$

then

$$\begin{aligned}
R_{21}^0 &= (1/2)\left(1 + \delta R_{21}\right) \\
&= (1/2)\left(1 + Q_{23}^0 Q_{24}^0 Q_{29}^0 - Q_{23}^0 Q_{24}^1 Q_{29}^0 - Q_{23}^1 Q_{24}^0 Q_{29}^0 + Q_{23}^1 Q_{24}^1 Q_{29}^0 - Q_{23}^0 Q_{24}^0 Q_{29}^1\right. \\
&\quad \left. + Q_{23}^0 Q_{24}^1 Q_{29}^1 + Q_{23}^1 Q_{24}^0 Q_{29}^1 - Q_{23}^1 Q_{24}^1 Q_{29}^1\right) \\
&= (1/2)\left[1 + Q_{23}^0 Q_{24}^0 Q_{29}^0 - Q_{23}^0\left(1 - Q_{24}^0\right) Q_{29}^0 - Q_{23}^1\left(1 - Q_{24}^1\right) Q_{29}^0 + Q_{23}^1 Q_{24}^1 Q_{29}^0\right. \\
&\quad \left. - Q_{23}^0\left(1 - Q_{24}^1\right) Q_{29}^1 + Q_{23}^0 Q_{24}^1 Q_{29}^1 + Q_{23}^1 Q_{24}^0 Q_{29}^1 - Q_{23}^1\left(1 - Q_{24}^0\right) Q_{29}^1\right] \\
&= (1/2)\left[1 + Q_{23}^0 Q_{24}^0 Q_{29}^0 - Q_{23}^0 Q_{29}^0 + Q_{23}^0 Q_{24}^0 Q_{29}^0 - Q_{23}^1 Q_{29}^0 + Q_{23}^1 Q_{24}^1 Q_{29}^0\right. \\
&\quad + Q_{23}^1 Q_{24}^1 Q_{29}^0 - Q_{23}^0 Q_{29}^1 + Q_{23}^0 Q_{24}^1 Q_{29}^1 + Q_{23}^0 Q_{24}^1 Q_{29}^1 + Q_{23}^1 Q_{24}^0 Q_{29}^1 - Q_{23}^1 Q_{29}^1 \\
&\quad \left. + Q_{23}^1 Q_{24}^0 Q_{29}^1\right] \\
&= (1/2)\left[2Q_{23}^0 Q_{24}^0 Q_{29}^0 + 2Q_{23}^1 Q_{24}^1 Q_{29}^0 + 2Q_{23}^0 Q_{24}^1 Q_{29}^1 + 2Q_{23}^1 Q_{24}^0 Q_{29}^1\right. \\
&\quad \left. + 1 - Q_{23}^0 Q_{29}^0 - Q_{23}^1 Q_{29}^0 - Q_{23}^0 Q_{29}^1 - Q_{23}^1 Q_{29}^1\right] \\
&= (1/2)\left[2Q_{23}^0 Q_{24}^0 Q_{29}^0 + 2Q_{23}^1 Q_{24}^1 Q_{29}^0 + 2Q_{23}^0 Q_{24}^1 Q_{29}^1 + 2Q_{23}^1 Q_{24}^0 Q_{29}^1\right. \\
&\quad \left. + 1 - Q_{23}^0\left(Q_{29}^0 + Q_{29}^1\right) - Q_{23}^1\left(Q_{29}^0 + Q_{29}^1\right)\right] \\
&= Q_{23}^0 Q_{24}^0 Q_{29}^0 + Q_{23}^1 Q_{24}^1 Q_{29}^0 + Q_{23}^0 Q_{24}^1 Q_{29}^1 + Q_{23}^1 Q_{24}^0 Q_{29}^1
\end{aligned}$$

Similarly, the calculation of the coefficient R_{21}^1 gives

$$R_{21}^1 = (1/2)\left(1 - \delta R_{21}\right) = Q_{23}^0 Q_{24}^1 Q_{29}^0 + Q_{23}^1 Q_{24}^0 Q_{29}^0 + Q_{23}^0 Q_{24}^0 Q_{29}^1 + Q_{23}^1 Q_{24}^1 Q_{29}^1$$

This example illustrates the equivalence of these two methods.

8.7 A Logarithmic LDPC Decoder

Another important simplification of the sum–product algorithm makes use of logarithmic calculation, in order to convert products or quotients into additions or subtractions. However, an additional complication arises, because there is a need to calculate the logarithm of a sum of terms in this algorithm. The logarithmic decoder is constructed on the basis of the MacKay–Neal simplified sum–product decoding algorithm, and it is essentially a logarithmic reformulation of the original algorithm. The decoding complexity is thereby drastically reduced. Here, we recognize the significant contribution of Leonardo Arnone to the development of the method presented in this section [12].

A useful expression for the development of this logarithmic algorithm is given for a number $z \leq 1$, which has an equivalent way of being expressed:

$$z = e^{-|Lz|} \rightarrow |Lz| = |\ln(z)| \tag{28}$$

The following expressions will also be useful in calculating the logarithm of a sum or a difference:

$$\ln(e^m + e^n) = \max(m, n) + \ln(1 + e^{-|m-n|}) \tag{29}$$

$$\ln(e^m - e^n) = \max(m, n) + \ln(1 - e^{-|m-n|}), m > n \tag{30}$$

The logarithmic decoder is basically implemented by performing the same steps as the MacKay–Neal simplified sum–product decoding algorithm, introduced in Section 8.6.

8.7.1 Initialization

In the initialization of the decoding algorithm, the values of coefficients Q_{ij}^x are set to be equal to the *a priori* probabilities f_j^x of the symbols. Coefficient f_j^x is the probability that the jth symbol is equal to x. Thus, coefficients Q_{ij}^0 and Q_{ij}^1 are set to be equal to f_j^0 and f_j^1, respectively. Since f_j^x is a probability, it is a number less than or equal to 1, and so

$$f_j^x = e^{-\left|Lf_j^x\right|} \rightarrow \left|Lf_j^x\right| = \left|\ln\left(f_j^x\right)\right| \tag{31}$$

and

$$Q_{ij}^x = e^{-\left|LQ_{ij}^x\right|} \rightarrow \left|LQ_{ij}^x\right| = \left|\ln\left(Q_{ij}^x\right)\right| \tag{32}$$

8.7.2 Horizontal Step

Equation (20) can be written as

$$e^{-|L\delta Q_{ij}|} = e^{-\left|LQ_{ij}^0\right|} - e^{-\left|LQ_{ij}^1\right|} \tag{33}$$

As this quantity is signed, it is better to rewrite it as

$$\delta Q_{ij} = (-1)^{s_{ij}} \left| e^{-\left|L\delta Q_{ij}\right|} \right| = (-1)^{s_{ij}} \left| e^{-\left|LQ_{ij}^{0}\right|} - e^{-\left|LQ_{ij}^{1}\right|} \right| \tag{34}$$

where

$$\text{if } \left|LQ_{ij}^{0}\right| \leq \left|LQ_{ij}^{1}\right| \text{ then } s_{ij} = 0$$

or

$$\text{if } \left|LQ_{ij}^{0}\right| > \left|LQ_{ij}^{1}\right| \text{ then } s_{ij} = 1 \tag{35}$$

Using expression (30),

$$\left|\ln\left(\left|e^{-|m|} - e^{-|n|}\right|\right)\right| = \min(|m|, |n|) + |\ln\left(1 - e^{-||m|-|n||}\right)|$$

then $|L\delta Q_{ij}|$ can be written as

$$\left|L\delta Q_{ij}\right| = \min\left(\left|LQ_{ij}^{0}\right|, \left|LQ_{ij}^{1}\right|\right) + \left|\ln\left(1 - e^{-\left|\left|LQ_{ij}^{0}\right| - \left|LQ_{ij}^{1}\right|\right|}\right)\right| \tag{36}$$

or

$$\left|L\delta Q_{ij}\right| = \min\left(\left|LQ_{ij}^{0}\right|, \left|LQ_{ij}^{1}\right|\right) + f_{-}\left(\left|\left|LQ_{ij}^{0}\right| - \left|LQ_{ij}^{1}\right|\right|\right) \tag{37}$$

where f_{-} is a look-up table with entries $|LQ_{ij}^{0}|$ and $|LQ_{ij}^{1}|$.

Expression (21) is written as

$$\delta R_{ij} = e^{-\left|L\delta R_{ij}\right|} = \prod_{j' \in N(i)\backslash j} (-1)^{s_{ij'}} \left| e^{-\left|L\delta Q_{ij'}\right|} \right| = (-1)^{\sum s_{ij'}} \prod_{j' \in N(i)\backslash j} \left| e^{-\left|L\delta Q_{ij'}\right|} \right| \tag{38}$$

and then

$$\left|L\delta R_{ij}\right| = \sum_{j' \in N(i)\backslash j} \left|L\delta Q_{ij'}\right| \tag{39}$$

$$s\delta R_{ij} = \sum_{j' \in N(i)\backslash j} s_{ij'} \tag{40}$$

or, equivalently,

$$\delta R_{ij} = (-1)^{s\delta R_{ij}} \left| e^{-\left|L\delta R_{ij}\right|} \right| \tag{41}$$

Coefficients R_{ij}^{0} and R_{ij}^{1} can be obtained by using expressions (22) and (23) with the values of coefficients δR_{ij}, and so in logarithmic form

$$\ln\left(R_{ij}^{0}\right) = -\left|LR_{ij}^{0}\right| = \ln\left(1 + (-1)^{s\delta R_{ij}} \left| e^{-\left|L\delta R_{ij}\right|} \right|\right) - \ln(2) \tag{42}$$

If $s\delta R_{ij}$ is even,

$$\left|LR_{ij}^{0}\right| = \ln(2) - \left|\ln\left(1 + \left|e^{-|L\delta R_{ij}|}\right|\right)\right| = \ln(2) - f_{+}\left(\left|L\delta R_{ij}\right|\right) \tag{43}$$

If $s\delta Rr_{ij}$ is odd,

$$\left|LR_{ij}^{0}\right| = \ln(2) + \left|\ln\left(1 - \left|e^{-|L\delta R_{ij}|}\right|\right)\right| = \ln(2) + f_{-}\left(\left|L\delta R_{ij}\right|\right) \tag{44}$$

where $f_{+}(|L\delta Rr_{ij}|)$ and $f_{-}(|L\delta R_{ij}|)$ are obtained from look-up tables. In a similar way, and if $s\delta R_{ij}$ is even, then

$$\left|LR_{ij}^{1}\right| = \ln(2) + \left|\ln\left(1 - \left|e^{-|L\delta R_{ij}|}\right|\right)\right| = \ln(2) + f_{-}\left(\left|L\delta R_{ij}\right|\right) \tag{45}$$

and if $s\delta R_{ij}$ is odd,

$$\left|LR_{ij}^{1}\right| = \ln(2) - \left|\ln\left(1 + \left|e^{-|L\delta R_{ij}|}\right|\right)\right| = \ln(2) - f_{+}\left(\left|L\delta R_{ij}\right|\right) \tag{46}$$

8.7.3 Vertical Step

In this step, and in order to solve equation (24), it is convenient to define the following constant, for $x = 0, 1$:

$$c_{ij}^{x} = f_{j}^{x} \prod_{i' \in M(j)\backslash i} R_{i'j}^{x} \tag{47}$$

which, in logarithmic form, is equal to

$$\ln\left(c_{ij}^{x}\right) = \ln\left(e^{-\left|Lc_{ij}^{x}\right|}\right) = \ln\left(e^{-\left|Lf_{j}^{x}\right|}\right) + \sum_{i' \in M(j)\backslash i} \ln\left(e^{-\left|LR_{i'j}^{x}\right|}\right) \tag{48}$$

or

$$|Lc_{ij}^{x}| = |Lf_{j}^{x}| + \sum_{i' \in M(j)\backslash i} |LR_{i'j}^{x}| \tag{49}$$

Then

$$\alpha_{ij} = 1/\left(c_{ij}^{0} + c_{ij}^{1}\right) \tag{50}$$

and hence

$$Q_{ij}^{0} = e^{-\left|LQ_{ij}^{0}\right|} = \frac{e^{-\left|Lc_{ij}^{0}\right|}}{e^{-\left|Lc_{ij}^{0}\right|} + e^{-\left|Lc_{ij}^{1}\right|}} \tag{51}$$

Expression (51) allows to determine $|LQ_{ij}^{0}|$ as

$$\left|LQ_{ij}^{0}\right| = \left|Lc_{ij}^{0}\right| - \min\left(\left|Lc_{ij}^{0}\right|, \left|Lc_{ij}^{1}\right|\right) + f_{+}\left(\left||Lc_{ij}^{0}| - |Lc_{ij}^{1}|\right|\right) \tag{52}$$

Similarly,

$$\left|LQ_{ij}^{1}\right| = \left|Lc_{ij}^{1}\right| - \min\left(\left|Lc_{ij}^{0}\right|, \left|Lc_{ij}^{1}\right|\right) + f_{+}\left(\left||Lc_{ij}^{0}| - |Lc_{ij}^{1}|\right|\right) \tag{53}$$

In each iteration, the decoder determines an estimate of the decoded vector by using expression (27). Two additional constants are defined to facilitate calculations, as done in the vertical step. Since

$$Q_j^x = \alpha_j f_j^x \prod_{i \in M(j)} R_{ij}^x = \alpha_j f_j^x R_{ij}^x \prod_{i' \in M(j)\backslash i} R_{i'j}^x$$

the following constant is defined for $x = 0, 1$:

$$c_j^x = e^{-|Lc_j^x|} = f_j^x \prod_{i \in M(j)} R_{ij}^x \tag{54}$$

such that

$$|Lc_j^x| = |Lc_{ij}^x| + |LR_{ij}^x| \tag{55}$$

The non-logarithmic value of the coefficient of interest is obtained as

$$Q_j^0 = e^{-\left|LQ_j^0\right|} = \frac{e^{-\left|Lc_j^0\right|}}{e^{-\left|Lc_j^0\right|} + e^{-\left|Lc_j^1\right|}} \tag{56}$$

which, in logarithmic form, is

$$\left|LQ_j^0\right| = \left|Lc_j^0\right| - \min\left(\left|Lc_j^0\right|, \left|Lc_j^1\right|\right) + f_{+}\left(\left|\left|Lc_j^0\right| - \left|Lc_j^1\right|\right|\right) \tag{57}$$

Similarly,

$$\left|LQ_j^1\right| = \left|Lc_j^1\right| - \min\left(\left|Lc_j^0\right|, \left|Lc_j^1\right|\right) + f_{+}\left(\left|\left|Lc_j^0\right| - \left|Lc_j^1\right|\right|\right) \tag{58}$$

An estimate of the decoded vector $\hat{\boldsymbol{d}}$ is finally obtained by estimating each of its bits d_j, such that

$$\text{if } Q_j^0 > Q_j^1 \text{ then } \widehat{d}_j = 0, \text{ else } \widehat{d}_j = 1 \tag{59}$$

since

$$Q_j^0 = e^{-\left|LQ_j^0\right|} \quad \text{and} \quad Q_j^1 = e^{-\left|LQ_j^1\right|}$$

Logarithmically,

$$\text{if } \left|LQ_j^0\right| < \left|LQ_j^1\right| \text{ then } \widehat{d}_j = 0, \text{ else } \widehat{d}_j = 1 \tag{60}$$

8.7.4 Summary of the Logarithmic Decoding Algorithm

Initialization: Logarithmic values of coefficients $|LQ_{ij}^x|$ are set equal to the logarithmic values of the *a priori* probabilities of the symbols $|Lf_j^x|$.

Horizontal step: Logarithmic values of the coefficients $|LR_{ij}^0|$ and $|LR_{ij}^1|$ are calculated for each pair i, jj, using (39)–(46).

Vertical step: Logarithmic values of the coefficients $|LQ_{ij}^0|$ and $|LQ_{ij}^1|$ are calculated for each pair i, j, using (49), (52) and (53).

Estimate of the decoded vector: An estimate of each symbol $\hat{d}_j$ of the received vector is obtained at the end of each iteration.

The values of coefficients $|LQ_j^0|$ and $|LQ_j^1|$ are calculated by using (55), (57) and (58), and the final estimate is determined by using (60). If $\boldsymbol{H} \circ \widehat{\boldsymbol{d}} = \boldsymbol{0}$ then the decoded vector is a code vector, and the decoding halts. Otherwise the decoder performs the next iteration.

8.7.5 Construction of the Look-up Tables

The effective BER Performance of an LDPC Code depends, as usual, on the decoding algorithm used to decode it. In the case of the logarithmic version of the sum–product decoding algorithm proposed in this section, there is a need to construct the two look-up tables that are called $f_+(|z_1|, |z_2|)$ and $f_-(|z_1|, |z_2|)$. The maximum number of bits for representing numbers in these tables is c, such that the maximum number of entries of these two tables is $N_t = 2^c$. The effect of the quantization of the values in these tables is seen in Figures 8.6 and 8.7 where the BER performances of two LDPC codes, obtained by simulation, are depicted. One LDPC code is of a relatively small size, with parity check matrix $\boldsymbol{H_1}$ of 30 rows and 60 columns, whereas the other code, extracted from MacKay's website [26], is a medium-size LDPC code, with parity check matrix $\boldsymbol{H_2}$ of 504 rows and 1008 columns. Look-up tables were constructed using numerical representations of $c = 16$ bits, so that the maximum number of entries in these tables is $N_t = 2^c = 65{,}536$.

Simulations show that small look-up tables of 256 entries can be used without producing a significant loss in the BER performance of these LDPC codes.

An analysis of decoding complexity determines that if n is the number of columns of the parity check matrix $\boldsymbol{H}$ corresponding to a given LDPC code, and s is the average number of '1's per column in that matrix, the traditional sum–product algorithm requires the calculation of $6ns$ products and $5ns$ sums. In the case of the logarithmic algorithm, the calculation of $14ns$ sums and $3ns$ subtractions is required. It is seen that the logarithmic decoder requires more sums than that of the traditional algorithm, but it should be taken into account that there is no need of performing products, which are essentially implemented as a considerable number of sums in most of the practical implementations of this operation. Overall, the complexity of the logarithmic decoding algorithm is much less than that of the traditional sum–product algorithm.

8.8 Extrinsic Information Transfer Charts for LDPC Codes

8.8.1 Introduction

So far the sum–product algorithm has been introduced as an efficient iterative decoding algorithm for decoding LDPC codes, and it has been presented in its traditional form, in the

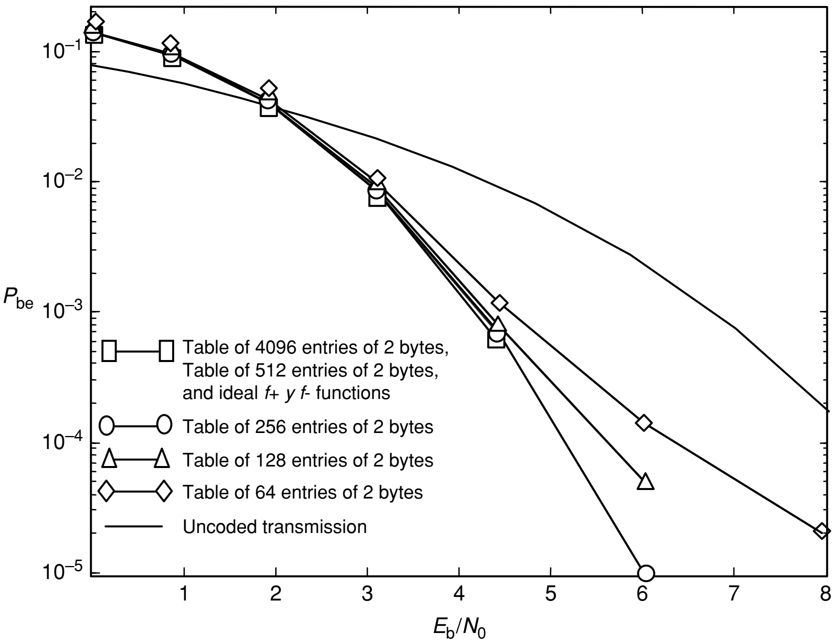


Figure 8.6 Logarithmic decoding of LDPC code $C_b(60, 30)$

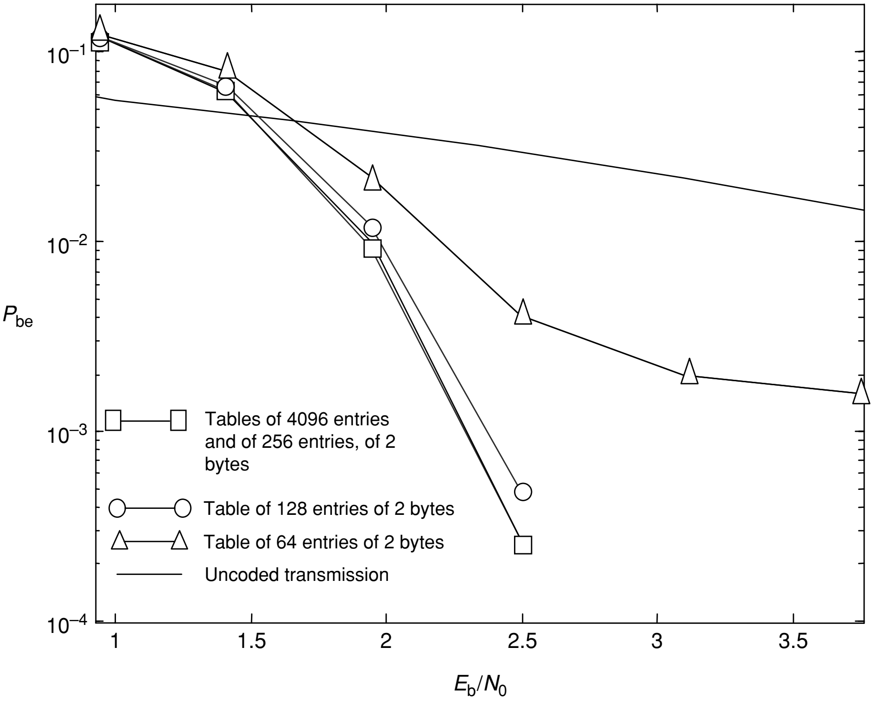


Figure 8.7 Logarithmic decoding of LDPC code $C_b(1008, 504)$ [26]

MacKay–Neal simplified version, and in a logarithmic version (Section 8.7). These algorithms operate on the basis of a convergent updating of interchange information communicated between the symbol nodes and the parity check nodes. In what follows, the convention

$$\begin{aligned} \text{‘0’} &\rightarrow +1 \\ \text{‘1’} &\rightarrow -1 \end{aligned}$$

is adopted to simplify the mathematical expressions involved in extrinsic information transfer (EXIT) chart analysis for LDPC codes.

A given LDPC code $C_{\mathrm{b}}(n, k)$, of code rate $R = k/n$ has n symbol nodes and $n - k$ parity check nodes, connected as described by the corresponding bipartite graph. The bit or symbol d_j participates in $d_{\mathrm{v}}^{(j)}$ parity check equations, which means that, in the corresponding bipartite graph, this symbol node is connected to $s^{(j)} = d_{\mathrm{v}}^{(j)}$ parity check nodes, where $s^{(j)}$ is the number of ‘1’s per column of the parity check matrix $\boldsymbol{H}$. In the same way, the parity check node h_i relates $d_{\mathrm{c}}^{(i)}$ symbol nodes or bits in its corresponding parity check equation, so that in the corresponding bipartite graph this parity check node is connected to $v^{(i)} = d_{\mathrm{c}}^{(i)}$ symbol nodes. In a regular LDPC code, the quantities $s^{(j)} = d_{\mathrm{v}}^{(j)}$ and $v^{(i)} = d_{\mathrm{c}}^{(i)}$ are the same for every row and column, respectively.

Example 8.2: Form the sparse parity check matrix $\boldsymbol{H}$ of a regular LDPC code $C_{\mathrm{b}}(14, 7)$ of code rate $R_{\mathrm{c}} = 1/2$ for which $d_{\mathrm{v}} = 3$ and $d_{\mathrm{c}} = 6$.

An example of a matrix of this kind is the following:

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

Example 8.2 is an illustrative example of a regular LDPC code that has a bipartite graph like that of Figure 8.8, where it is seen that there are cycles of length 4, one of which is noted in bold line in this figure.

As seen in Figure 8.8, there are three connections emerging from each symbol node, and there are six connections arriving at each parity check code. The bipartite graph can be interpreted as an interleaving of connections. This is seen in Figure 8.9.

The graphical representation as given in Figure 8.9 then allows us to see a given LDPC code as a code constructed using two encoders, each one with its corresponding decoder. There is a code for the symbol nodes and another code for the parity check nodes, which are related through a connection interleaver like that seen in Figure 8.9. This connection interleaver acts during the iterative decoding process, which consists of the interchange of soft-decision information (LLRs) between the symbol node decoder (SND) and the parity check node decoder (PCND), as shown in Figure 8.10 [14]. The MAP decoder converts *a priori* and channel LLRs into *a posteriori* LLRs. Both decoders, the SND and the PCND, perform this type of operation, generating LLRs as their outputs. If the *a priori* LLR is subtracted from the corresponding

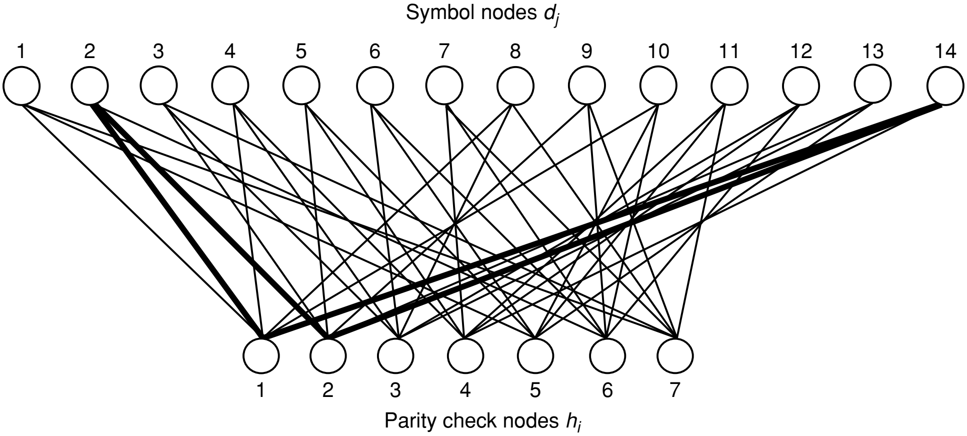


Figure 8.8 A bipartite graph for a regular LDPC code

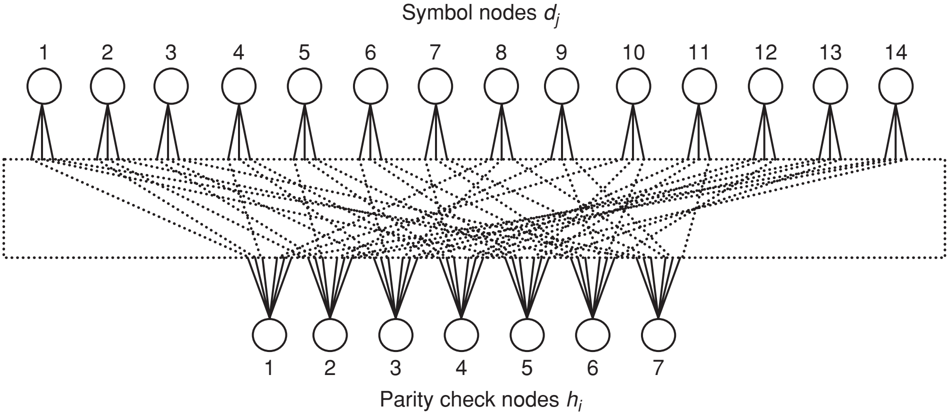


Figure 8.9 Connection interleaver for a regular LDPC code

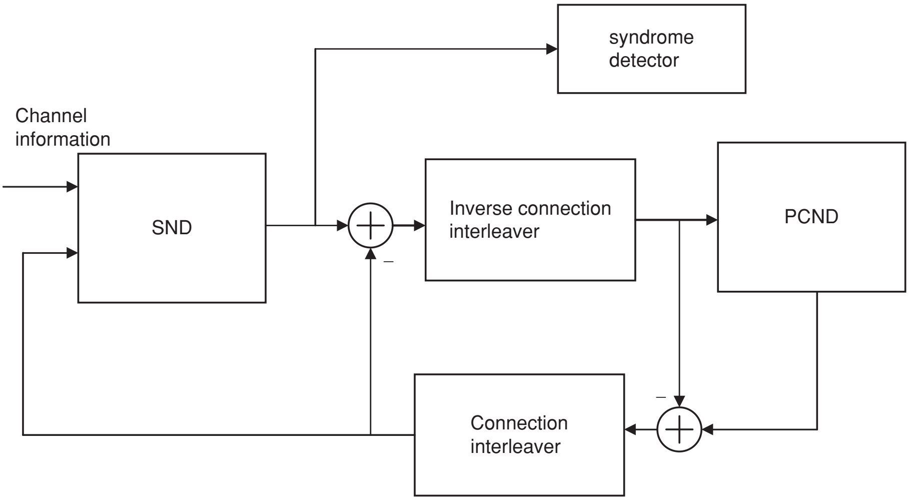


Figure 8.10 Interchange of LLRs between the SND and the PCND of an LDPC decoder

a posteriori LLR, then the extrinsic LLR is obtained. The extrinsic LLR of the current iteration then becomes the *a priori* LLR for the next iteration.

From this point of view, an LDPC code can also be understood as a mixture of inner repetition codes and a mixture of outer simple parity check codes, which operate like a serial concatenated code [14]. This allows us to comprehend the similarity between LDPC codes and other iteratively decoded codes, like the turbo codes introduced in Chapter 7.

8.8.2 Iterative Decoding of Block Codes

The expression of the LLR has been introduced in Chapter 7, equation (13), which is rewritten here for clarity:

$$L(b_i) = \ln\left(\frac{P(b_i = +1)}{P(b_i = -1)}\right)$$

Remember that the sign of this quantity is the hard decision of the estimated value, while its absolute value is the reliability of that decision. From this definition, the following expressions are obtained:

$$e^{L(b_i)} = \frac{P(b_i = +1)}{P(b_i = -1)} = \frac{P(b_i = +1)}{1 - P(b_i = +1)}$$

or

$$P(b_i = +1) = \frac{e^{L(b_i)}}{1 + e^{L(b_i)}} \tag{61}$$

and

$$P(b_i = -1) = \frac{1}{1 + e^{L(b_i)}} \tag{62}$$

When decisions are taken conditioned to another variable, like a received vector $\boldsymbol{Y}$, the LLR is of the form of equation (54) of Chapter 7, and considering equation (55) of Chapter 7, it can be written as

$$L(b_i/\boldsymbol{Y}) = \ln\left(\frac{P(b_i = +1/\boldsymbol{Y})}{P(b_i = -1/\boldsymbol{Y})}\right) = \ln\left(\frac{P(b_i = +1)}{P(b_i = -1)}\right) + \ln\left(\frac{P(y_i/b_i = +1)}{P(y_i/b_i = -1)}\right) \tag{63}$$

As described in previous sections, the sum–product algorithm operates over parity check equations, so that it will be useful to determine the LLR of an exclusive-OR summation of two or more bits. For this, and for the exclusive-OR sum of two bits,

$$P\left((b_1 \oplus b_2) = +1\right) = P(b_1 = +1)P(b_2 = +1) + (1 - P(b_1 = +1))(1 - P(b_2 = +1)) \tag{64}$$

$$P\left((b_1 \oplus b_2) = -1\right) = P(b_1 = +1)P(b_2 = -1) + (1 - P(b_1 = +1))(1 - P(b_2 = -1)) \tag{65}$$

where $P(b_i = +1)$ is given by equation (61). If bits b_1 and b_2 are generated by independent random sources, then

$$P\left((b_1 \oplus b_2) = +1\right) = \frac{1 + e^{L(b_1)}\, e^{L(b_2)}}{\left(1 + e^{L(b_1)}\right)\left(1 + e^{L(b_2)}\right)} \tag{66}$$

$$P\left((b_1 \oplus b_2) = -1\right) = \frac{e^{L(b_1)} + e^{L(b_2)}}{\left(1 + e^{L(b_1)}\right)\left(1 + e^{L(b_2)}\right)} \tag{67}$$

and so [15]

$$\begin{aligned} L(b_1 \oplus b_2) &= \ln\left[\frac{P((b_1 \oplus b_2) = +1)}{P((b_1 \oplus b_2) = -1)}\right] = \ln\left[\frac{1 + e^{L(b_1)} e^{L(b_2)}}{e^{L(b_1)} + e^{L(b_2)}}\right] \\ &\approx \operatorname{sign}(L(b_1))\operatorname{sign}(L(b_2))\min\left(|L(b_1)|, |L(b_2)|\right) \end{aligned} \tag{68}$$

This operation deserves a distinguishing notation that is defined in [15], and that in this text is described by the symbol[$\oplus$]:

$$L(b_1)\,[\oplus]\,L(b_2) = L(b_1 \oplus b_2) \tag{69}$$

The following rules apply to this operation:

$$L(b_1)\,[\oplus]\,\infty = L(b_1) \tag{70}$$

$$L(b_1)\,[\oplus] - \infty = -L(b_1) \tag{71}$$

$$L(b_1)\,[\oplus]\,0 = 0 \tag{72}$$

On the other hand, the expression can be extended to the operation over more than two bits by induction

$$\sum_{j=1}^{J} [\oplus] L(b_j) = L\left(\sum_{j=1}^{J} \oplus b_j\right) = \ln\left[\frac{\prod_{j=1}^{J}\left(e^{L(b_j)} + 1\right) + \prod_{j=1}^{J}\left(e^{L(b_j)} - 1\right)}{\prod_{j=1}^{J}\left(e^{L(b_j)} + 1\right) - \prod_{j=1}^{J}\left(e^{L(b_j)} - 1\right)}\right] \tag{73}$$

$$\sum_{j=1}^{J} [\oplus] L(b_j) = L\left(\sum_{j=1}^{J} \oplus b_j\right) \approx \left[\prod_{j=1}^{J} \operatorname{sign}(L(b_j))\right] \min_{j=1\ldots J} |L(b_j)| \tag{74}$$

and by using

$$\tan\, \mathrm{h}(b/2) = \frac{e^b - 1}{e^b + 1}$$

it becomes

$$\sum_{j=1}^{J}[\oplus]L(b_j) = L\left(\sum_{j=1}^{J}\oplus b_j\right) = \ln\left[\frac{1+\prod_{j=1}^{J}\tan\,\mathrm{h}\left(L(b_j)/2\right)}{1-\prod_{j=1}^{J}\tan\,\mathrm{h}\left(L(b_j)/2\right)}\right]$$
$$= 2\,\tan\,\mathrm{h}^{-1}\left(\prod_{j=1}^{J}\tan\,\mathrm{h}\left(L(b_j)/2\right)\right) \quad (75)$$

which can be approximately calculated as

$$\sum_{j=1}^{J}[\oplus]L(b_j) = L\left(\sum_{j=1}^{J}\oplus b_j\right) \approx \left(\prod_{j=1}^{J}\mathrm{sign}(L(b_j))\right)\min_{j=1\ldots J}|L(b_j)| \quad (76)$$

8.8.3 *EXIT Chart Construction for LDPC Codes*

The transfer of mutual information between the SND and the PCND determines the EXIT chart for an LDPC code. This analysis is simplified by applying it to a regular LDPC code; that is, an LDPC code where the number of '1's per column and per row is fixed. An example of a regular LDPC code is presented in Example 8.2, whose bipartite graph is seen in Figures 8.8 and 8.9. The notation used in this section is the same as that used in the case of the EXIT chart analysis for turbo codes, presented in Chapter 7 [19–22]. Thus, I_A is the mutual information between the information of symbols or bits that correspond to symbol nodes, those over which estimates are performed, and the *a priori* information, in both cases determined by LLRs. Similarly, I_E is the mutual information between the information of symbols or bits that correspond to symbol nodes, and the extrinsic information. The EXIT chart for the SND and the PCND of an LDPC code is described in terms of mutual information of the involved quantities, as described in Chapter 7, and developed in the next section.

8.8.4 *Mutual Information Function*

For the AWGN channel, the relation of the average bit energy E_b and the noise power spectral density N_0 is equal to $E_\mathrm{b}/N_0 = 1\big/\left[2R_\mathrm{c}\sigma_\mathrm{n}^2\right]$, where R_c is the code rate and $\sigma_\mathrm{n}^2 = N_0/2$ is the noise variance. Then the channel LLR $L_\mathrm{ch} = L_\mathrm{ch}^{(0)}$ is equal to

$$L_\mathrm{ch} = \ln\frac{p(y/x=+1)}{p(y/x=-1)} = \frac{2}{\sigma_\mathrm{n}^2}y = \frac{2}{\sigma_\mathrm{n}^2}(x+n) \quad (77)$$

where

$$p(y/X=x) = \frac{e^{-(y-x)^2/2\sigma_\mathrm{n}^2}}{\sqrt{2\pi}\sigma_\mathrm{n}^2}$$

The variance σ_ch^2 can be expressed as

$$\sigma_\mathrm{ch}^2 = \left(\frac{2}{\sigma_\mathrm{n}^2}\sigma_\mathrm{n}\right)^2 = \frac{4}{\sigma_\mathrm{n}^2} = 8R_\mathrm{c}\frac{E_\mathrm{b}}{N_0} \quad (78)$$

By taking into account the analytical expression for I_A, as given in Chapter 7,

$$I_\mathrm{A} = I_\mathrm{A}(\sigma_\mathrm{A}) = 1 - \int_{-\infty}^{\infty} \frac{e^{-(\xi-\sigma_\mathrm{A}^2/2)/2\sigma_\mathrm{A}^2}}{\sqrt{2\pi}\sigma_\mathrm{A}} \log_2(1+e^{-\xi})\,\mathrm{d}\xi$$

the following short notation is used:

$$J(\sigma) = I_\mathrm{A}(\sigma_\mathrm{A} = \sigma) \tag{79}$$

and thus

$$\lim_{\sigma\to 0} J(\sigma) = 0, \qquad \lim_{\sigma\to\infty} J(\sigma) = 0, \quad \sigma > 0$$

This is a monotonically decreasing and invertible function for which the value of σ_A can be obtained as

$$\sigma_\mathrm{A} = J^{-1}(I_\mathrm{A}) \tag{80}$$

A polynomial approximation of functions $J(\sigma)$ and $J^{-1}(I)$ is presented in [14] and used below. The input X and the output Y of the AWGN channel are related as $Y = X + n$, where n is a Gaussian random variable with zero mean value and variance σ_n^2. The LLR described in expression (77) is a function of y. On the other hand, $L_\mathrm{ch}(Y)$ is a variable conditioned on the variable $X = \pm 1$, and so it also has a Gaussian distribution of mean value $\mu_\mathrm{ch} = \pm 2\big/\sigma_\mathrm{n}^2$ and variance $\sigma_\mathrm{ch}^2 = 4\big/\sigma_\mathrm{n}^2$, such that $\mu_\mathrm{ch} = \pm\sigma_\mathrm{ch}^2\big/2$.

If the mutual information between the LLR $L_\mathrm{ch}(Y)$ and the input X is

$$J(\sigma_\mathrm{ch}) = I(X; L_\mathrm{ch}(Y)) \tag{81}$$

then

$$J(\sigma_\mathrm{ch}) = H(X) - H(X/L_\mathrm{ch}(Y)) = 1 - \int_{-\infty}^{\infty} \frac{e^{-(\xi-\sigma_\mathrm{ch}^2/2)/2\sigma_\mathrm{ch}^2}}{\sqrt{2\pi}\sigma_\mathrm{ch}} \log_2(1+e^{-\xi})\,\mathrm{d}\xi \tag{82}$$

where $H(X)$ is the entropy of the channel input X, and $H(X/L_\mathrm{ch}(Y))$ is the entropy of X conditioned to $L_\mathrm{ch}(Y)$. However, $J(\sigma_\mathrm{ch}) = I(X; L_\mathrm{ch}(Y))$ is equal to $I(X;Y)$ so that the capacity of the AWGN channel is equal to $J(\sigma_\mathrm{ch}) = J(2/\sigma_\mathrm{n})$ [14, 23]. Following [14], a polynomial approximation for $J(\sigma)$ is

$$J(\sigma) \approx \begin{cases} -(0.0421061)\,\sigma^3 + (0.209252)\,\sigma^2 - (0.00640081)\,\sigma & 0 \le \sigma \le 1.6363 \\ 1 - e^{(0.00181491)\sigma^3 - (0.142675)\sigma^2 - (0.0822054)\sigma + 0.0549608} & 1.6363 < \sigma < 10 \\ 1 \quad \sigma \ge 10 \end{cases} \tag{83}$$

and for the inverse function $J^{-1}(I)$,

$$J^{-1}(I) \approx \begin{cases} (1.09542)\,I^2 + (0.214217)\,I + (2.33727)\,\sqrt{I} & 0 \le I \le 0.3646 \\ -(0.706692)\ln\left[(0.386013)\,(1-I)\right] + (1.75017)\,I & 0.3646 < \mathrm{I} < 1 \end{cases} \tag{84}$$

8.8.5 EXIT Chart for the SND

Since this analysis is restricted to regular LDPC codes, the number of parity equations in which a given symbol is present is constant and equal to d_v. In the case of the Example 8.2, this parameter is $d_\mathrm{v} = 3$. The LLR that each symbol node d_j sends to each parity check node h_i in the iteration number *it* is denoted as $Z_{ij}^{(it)}$, and the LLR that each parity check node h_i sends to each symbol node d_j in the iteration number *it* is denoted as $L_{ij}^{(it)}$.

Each symbol node has as its input information a channel LLR $L_\mathrm{ch} = L_\mathrm{ch}^{(0)}$ coming from the channel, and an LLR $L_{i'j}^{(it)}$ that comes from each of its children parity check nodes. The symbol node uses this information to generate the LLR $Z_{ij}^{(it)}$ to be sent to its d_v children parity check nodes, according to the expression

$$Z_{ij}^{(it)} = L_\mathrm{ch} + \sum_{i' \in M(j) \backslash i} L_{i'j}^{(it)} \tag{85}$$

In expression (85) the LLR $L_{i'j}^{(it)}$ is the *a priori* LLR for the SND, $Z_{ij}^{(it)}$ is the extrinsic LLR generated by the SND and $L_\mathrm{ch} = L_\mathrm{ch}^{(0)}$ is the LLR from the channel. At the end of this current iteration, the SND determines an estimate, which is in turn an LLR, useful to determine an estimate of each of the bits of the received code vector. This *a posteriori* estimate is equal to

$$A_{ij}^{(it)} = L_\mathrm{ch} + \sum_{i \in M(j)} L_{ij}^{(it)} \tag{86}$$

The LLR is $Z_{ij}^{(it)} = |Lc_{ij}^{-1}| - |Lc_{ij}^{+1}|$, calculated using (49).

The EXIT chart for the SND is determined by the value of the mutual information function for the value σ of the standard deviation of the variable $Z_{ij}^{(it)}$ described by expression (85).

The logarithmic version of the sum–product algorithm defines a linear relationship between the quantities involved in describing the operation of the SND, as seen in expression (85). Since channel LLRs and *a priori* LLRs are independent random variables, the variance of the variable

$$Z_{ij}^{(it)} = L_\mathrm{ch} + \sum_{i' \in M(j) \backslash i} L_{i'j}^{(it)}$$

is equal to [14]

$$\sigma_{Z_{ij}}^2 = \sigma_\mathrm{ch}^2 + (d_\mathrm{v} - 1)\,\sigma_\mathrm{A}^2 = 8R_\mathrm{c}\frac{E_\mathrm{b}}{N_0} + (d_\mathrm{v} - 1)\left[J^{-1}(I_\mathrm{A})\right]^2 \tag{87}$$

This leads to an analytical determination of the EXIT charts for LDPC codes, in comparison with the heuristically implemented method described for the EXIT charts of turbo codes. Once the standard deviation of the extrinsic LLRs generated by the SND has been determined, then the mutual information between these LLRs and the bit or symbol information is directly obtained by using expressions (80) and (83) with $\sigma_\mathrm{A} = \sigma = \sigma_{Z_{ij}}$,

$$I_\mathrm{E,SND}(I_\mathrm{A}, d_\mathrm{v}, E_\mathrm{b}/N_0, R_\mathrm{c}) = J\left(\sqrt{\sigma_\mathrm{ch}^2 + (d_\mathrm{v} - 1)\,\sigma_\mathrm{A}^2}\right) \tag{88}$$

8.8.6 EXIT Chart for the PCND

Parity check nodes generate their LLRs taking into account the parity check equations that are defined for each of them. The estimate or LLR is calculated assuming that the parity check equation is satisfied. An LLR expression for a parity check equation has already been obtained, and it is given by the expression (73) in its exact version, or by (74) in its approximated version. They are conveniently rewritten for this case as

$$L_{ij}^{(it-1)} = \ln\left[\frac{\prod_{i'=1}^{I'}\left(e^{Z_{i'j}^{(it-1)}}+1\right)+\prod_{i'=1}^{I'}\left(e^{Z_{i'j}^{(it-1)}}-1\right)}{\prod_{i'=1}^{I'}\left(e^{Z_{i'j}^{(it-1)}}+1\right)-\prod_{i'=1}^{I'}\left(e^{Z_{i'j}^{(it-1)}}-1\right)}\right] \tag{89}$$

$$L_{ij}^{(it-1)} \approx \left[\prod_{i'=1}^{I'}\operatorname{sign}\left(Z_{i'j}^{(it-1)}\right)\right]\min_{i'=1\ldots I}\left|Z_{i'j}^{(it-1)}\right| \tag{90}$$

Here $Z_{i'j}^{(it-1)}$ is the estimate generated in the previous iteration, adopted as the *a priori* value for the current iteration.

Remember that the convention adopted in [15] is that the bit or symbol ‘0’ is represented by the signal $+1$, and the bit or symbol ‘1’ is represented by the signal -1. If the convention is taken the other way round, the argument in expressions (73) and (89) should be inverted. This expression is a summation of LLRs defined by the operator $[\oplus]$. In [16] and [17] it is shown that the EXIT chart for the PCND for the AWGN channel can be approximately, but with high accuracy, calculated as

$$I_{\mathrm{E,PCND}}(I_{\mathrm{A}}, d_{\mathrm{c}}) \approx 1 - J\left(\sqrt{d_{\mathrm{c}}-1}\,J^{-1}(1-I_{\mathrm{A}})\right) \tag{91}$$

It is more useful to determine the inverse function of (91), which defines the mutual information between the bits of the decoded vector and the *a priori* information, as a function of the mutual information between the bits of the decoded vector and the extrinsic information:

$$I_{\mathrm{A,PCND}}(I_{\mathrm{E}}, d_{\mathrm{c}}) \approx 1 - J\left(\frac{J^{-1}(1-I_{\mathrm{E}})}{\sqrt{d_{\mathrm{c}}-1}}\right) \tag{92}$$

Figure 8.11 shows the EXIT chart for the SND, for different values of the parameter d_{v}, and Figure 8.12 shows the EXIT chart for the PCND, for different values of the parameter d_{c}.

LDPC codes have a more analytical procedure for determining the EXIT charts than that of turbo codes. In [16] it is shown that, for the erasure channel, optimum design of an LDPC code is done by matching up the corresponding EXIT charts for the SND and the PCND. This conclusion is approximately valid for other channels including the AWGN channel. EXIT charts for LDPC codes are also useful for determining for instance a practical bound on the number of iterations for decoding LDPC codes. However, the EXIT chart analysis is also a good tool for the design of LDPC codes. An example of an LDPC code design is given in [14].

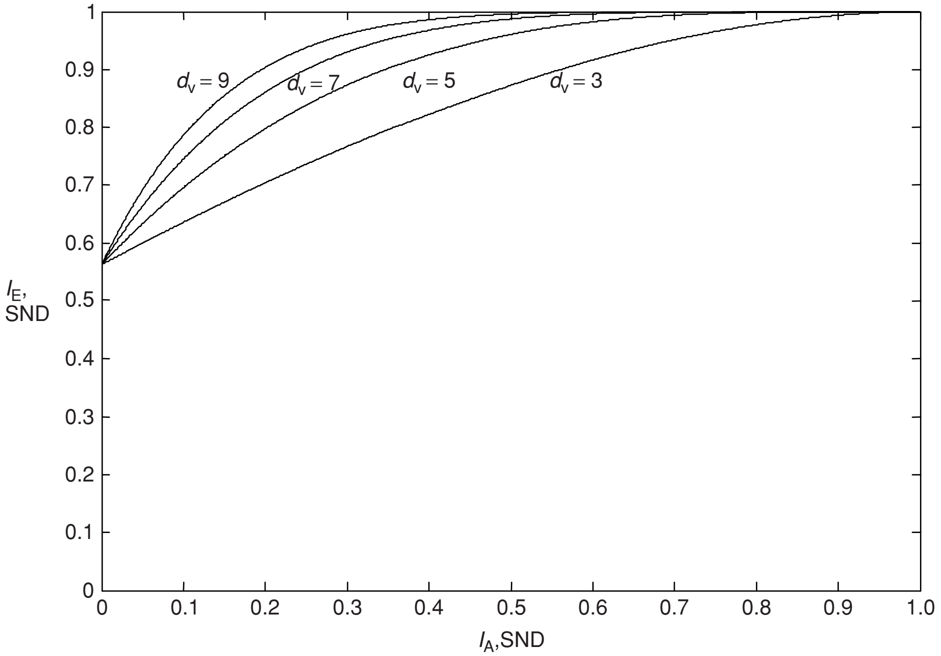


Figure 8.11 EXIT chart for the SND

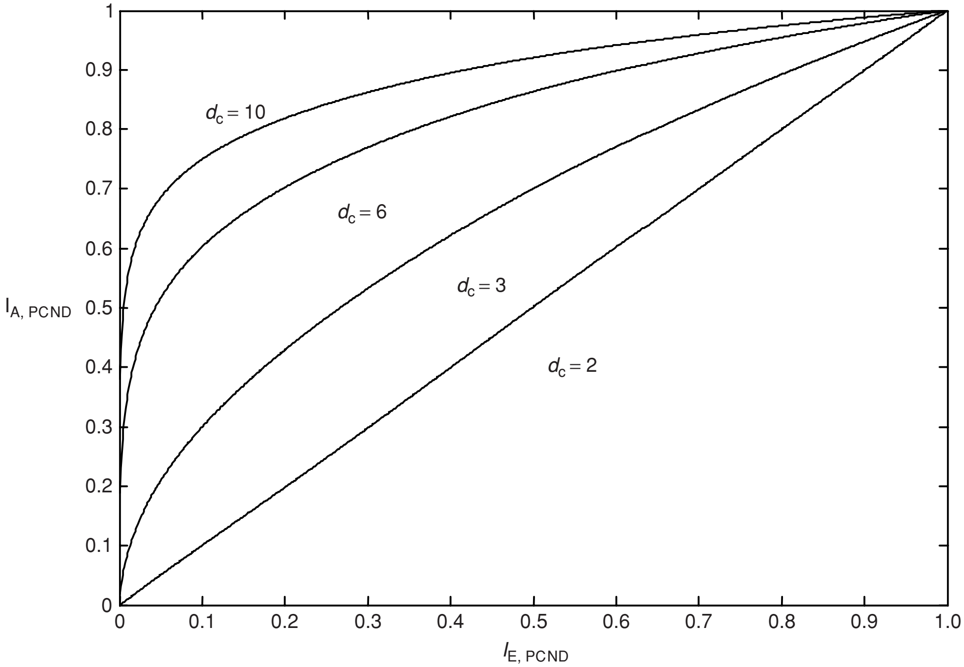


Figure 8.12 EXIT chart for the PCND

8.9 Fountain and LT Codes

8.9.1 Introduction

An interesting application field for LDPC codes is on the so-called erasure channel, introduced in Chapter 1, and its relationship with transmission in data networks. In data transmission over networks like the Internet, information is usually fragmented into data packets of fixed size before being transmitted through the network. For transmission, the information in each packet is usually encoded for detecting errors by using a cyclic redundancy check (CRC), and the receiver detects errors by syndrome decoding applied over each data packet in order to decide whether it accepts the data packet, or requires its retransmission. When the syndrome calculation determines that the data packet is not a valid packet, then the system resorts to the retransmission procedure called automatic repeat request (ARQ), described in Chapter 2. A second channel in this duplex system is used for transmitting retransmission requests. Packets found to contain errors are discarded, given the expectation that they will be retransmitted. This is the traditional approach to error control in data networks, based on ARQ schemes. However, a retransmission implies reuse of the transmission channel, and therefore throughput is reduced.

A different approach to network error control is provided by the use of so-called fountain codes [24, 27] in addition to the CRC. These codes generate data packets that are essentially random functions of the whole file to be transmitted. The transmitter floods the receiver with these data packets, without needing to know which of them are correctly received. The receiver discards any packet containing errors, as determined by the CRC syndrome. If the original size of the whole file to be transmitted is K data packets, and if the decoder receives N data packets, then it is possible to correctly recover at the receiver the original information (i.e., the whole file), provided N is sufficiently larger than K. How much larger N needs to be is determined by the random functions used to generate the packets and by the error rate on the channel.

The process of discarding of data packets can be suitably modelled by the erasure channel, introduced in Chapter 1, in which a packet that is discarded can be regarded as a packet erasure. The probability of an erasure or discard is p for the binary erasure channel (BEC), and its capacity is equal to $1 - p$, as calculated in Chapter 1. If this channel operates over a non-binary alphabet GF(q), for which $q = 2^m$, then the erasure channel has an increased capacity of $(1 - p)m$. In the transmission of data packets of a fixed length of m bits, the erasure channel with non-binary alphabet GF(q) is a suitable model for the discarding of such packets, so that each data packet is represented by one of the $q = 2^m$ elements of GF(q).

As demonstrated by Shannon, the capacity of a given channel does not depend on the existence or not of a physical means for requesting retransmissions. In the case of the q-ary erasure channel where $q = 2^m$, this capacity remains equal to $(1 - p)m$ independently of the existence or not of retransmissions.

In a typical ARQ scheme, retransmissions are required regardless of the value of the erasure or discard probability p, and this process could become enormously demanding if, for instance, transmission is taking place in the presence of a very noisy channel. ARQ schemes are also impractical in broadcast transmission scenarios, where the transmitter sends data packets to a multiplicity of users over independent or partially independent channels. Here the number of retransmissions required could seriously decrease the throughput of the transmitter. The need to maintain high throughput rates suggests the use of an FEC scheme for the transmission of packets in data networks, in order to reduce, and preferably eliminate, retransmissions. As pointed out earlier, the capacity of the q-ary erasure channel remains equal to $(1 - p)m$ independently of the existence or not of retransmissions, and suitable erasure-correction codes exist to take advantage of this property.

One of the most efficient error-control techniques was introduced in Chapter 5, and is the Reed–Solomon coding technique. An interesting property of a RS code $C_{\mathrm{RS}}(N, K)$ defined over GF(q), where $q = 2^m$, is that if any K of the N transmitted symbols are received, the K information symbols can be successfully recovered. However, RS codes are complex to decode and encode, particularly if m is large, and so they do not appear to be the most suitable codes for data networks, where packets are normally of a large size. In addition, the rate of an RS code needs to be determined before transmission, and is difficult to change during transmission. Of course the transmission can start with the code rate set to match the capacity of the erasure channel. However, the erasure probability p, which determines the capacity, can change during transmission over a network, as a function of the users locations, for instance. Therefore a dynamically variable code rate would be advantageous, and fountain codes appear able to provide it very effectively.

8.9.2 Fountain Codes

A fountain code [24, 27] can be seen as a code that generates a continuous flow of transmitted data packets, simulating the action of water falling from a spring into a collecting receptacle, the data packet receiver. In this coding scheme, the whole information file to be transmitted is of size Km, such that there are K data packets of m bits each. The receiver collects a set of K data packets or more, enough to successfully recover the original transmitted information. From this point of view, the rate of a fountain code tends to zero, because transmission is supposedly time unlimited. However, in a practical implementation, the number of data packets to be transmitted is dynamically determined to be finite, according to the necessities described earlier. This usually results in a variable, but relatively high code rate. The simplest fountain code is the linear random code.

8.9.3 Linear Random Codes

Let a whole file of information be fragmented into K data packets $\mathrm{dp}_1\ \mathrm{dp}_2 \ldots \mathrm{dp}_K$. Each transmitted data packet contains m bits, and is going to be correctly received or not, depending on the channel noise. Transmission is synchronous, and successive transmitted packets tp_n are ordered by a time index n. At time instant n, the encoder generates a random binary K-tuple $\{G_{kn}\}$, and then the transmitted packet tp_n is the exclusive-OR sum of all the information data packets for which the random bits in $\{G_{kn}\}$ are equal to '1':

$$\mathrm{tp}_n = \sum_{k=1}^{K} \mathrm{dp}_k G_{kn} \tag{93}$$

This encoding procedure can be seen as performed by an encoder whose generator matrix has an increasing number of columns, and so at each time instant it adds another random (in practice pseudo-random) column to its structure. An example of such a matrix could be the following:

$$\boldsymbol{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 & \cdots \\ 0 & 1 & 1 & 0 & 1 & 1 & \cdots \\ 1 & 0 & 0 & 0 & 1 & 1 & \cdots \\ 0 & 0 & 1 & 1 & 1 & 0 & \cdots \end{bmatrix} \tag{94}$$

The number of rows of this increasing matrix is K, and each of its semi-infinite number of columns are successively generated as random K-tuples $\{G_{kn}\}$, where $n = 1, 2, 3, \ldots$. For the generator matrix of (94), for example, the first transmitted data packet is the exclusive-OR sum of the original data packets dp_1 and dp_3, the second transmitted packet is the exclusive-OR sum of the original data packets dp_1 and dp_2, and so on.

The erasure channel will affect the transmission by erasing some of the transmitted data packets, but the receiver collects a set of N data packets in order to form a matrix of size $K \times N$, such that $N = K + E_x$, where E_x is the number of excess packets with respect to K, which can be understood as the redundancy in the transmission using the fountain code. The question that now arises is, can the original packets of information be successfully recovered from these N received data packets? The receiver is assumed to know the so-called fragment generator matrix $\boldsymbol{G}_{\mathrm{fr}}$, which is a matrix formed in the decoder from the N correctly received data packets, and each column of this fragment matrix has a known value of time index n. If the decoder also knows the pseudo-random rule that generated the K-tuple columns of the increasing matrix used by the encoder, then it is possible to recover the original information.

If $N < K$, then there is no way of successfully recovering the original information. For any non-zero value of the excess E_x, there is the possibility of a successful recovery. In the case of $N = K$, the original information can be recovered if an inverse matrix of the fragment generator matrix $\boldsymbol{G}_{\mathrm{fr}}$ exists. The probability of the existence of such an inverse matrix is determined in [24], and it is equal to

$$\left(1 - 2^{-K}\right)\left(1 - 2^{-(K-1)}\right) \cdots (1 - 1/8)(1 - 1/4)(1 - 1/2)$$

which turns out to be equal to 0.289 for any $K > 10$.

If $N > K$, then the probability δ of the existence of an invertible submatrix of size $K \times K$, in the fragment generator matrix $\boldsymbol{G}_{\mathrm{fr}}$, has to be determined. This probability, shown in [24], is to be bounded by a quantity that is a function of the excess E_x of data packets correctly received:

$$\delta \leq 2^{-E_x} \tag{95}$$

This means that the probability of successful recovery of data packets is $1 - \delta$, and that this happens if at least $K + \log_2(1/\delta)$ data packets are received.

Summarizing, the probability of successful recovery of the original information is equal to 0.289 if no redundant data packets are received, and this probability increases to $1 - \delta$ when E_x excess (redundant) packets are received.

However, the decoding complexity of these linear random codes is dominated by the inversion of the fragment generator matrix $\boldsymbol{G}_{\mathrm{fr}}$, which requires approximately K^3 binary operations, a limiting drawback for the transmission of a large number of large data packets, as is usually necessary in data networks.

8.9.4 Luby Transform Codes

Luby transform (LT) codes [25] appear to be more suitable for the erasure channel network application. They can be understood as fountain codes characterized by a linear sparse matrix similar to that used to define an LDPC code.

These codes are designed using a statistical analysis that models the problem of throwing balls in order to fill a set of empty baskets. A first question arising from this particular problem is to determine how many balls should be thrown to ensure that there is at least one ball in each basket. Another question is to determine the number of empty baskets that results from the throwing of a given number of balls at the set of baskets. Thus, for instance, if N balls are thrown at K baskets, $K\ e^{-N/K}$ is the expected number of empty baskets. This means that the expected number of empty baskets is a small number δ if $N > K\ \ln(K/\delta)$ [24, 27].

8.9.4.1 LT encoder

The encoder of an LT code takes a set of K data packets $\mathrm{dp}_1\ \mathrm{dp}_2 \ldots \mathrm{dp}_K$ to generate the coded data packet as follows:

A degree d_n is selected from a degree distribution function $\rho(d)$, conveniently designed as a function of the size K of the file to be encoded.

A set of d_n data packets is selected in a uniform random manner to form the coded packet tp_n as the exclusive-OR of these packets.

This encoding mechanism is associated with a bipartite graph which is similar to that for LDPC codes, and which is formed between the coded data packets tp_n and the information packets dp_k.

A sparse graph is obtained when the average value of the degree d_n is significantly smaller than the number of information packets K. This encoding mechanism can be interpreted as an LDPC code.

8.9.4.2 LT decoder

Since the encoding of an LT code is similar to that of an LDPC code, creating the transmitted or coded data packets from the information source or message packets, the decoding of an LT code consists of determining the vector $\boldsymbol{dp}$ as a function of the vector $\boldsymbol{tp}$, which are related by the expression $\boldsymbol{tp} = \boldsymbol{dp} \cdot \boldsymbol{G}$, where the matrix $\boldsymbol{G}$ corresponds to the bipartite graph of the encoding. Both sides of the transmission know this matrix, even when it is normally a pseudo-randomly generated matrix.

At first, this similarity with LDPC codes suggests the use of the sum–product algorithm for decoding LT codes, but in this case the entities involved are packets that are either completely reliable or completely unreliable (erased). This means that there are packets dp_k with unity probability of being true packets, or packets dp_k that have all the same probability of not being true packets. The decoding algorithm, in this case, operates in a very simple manner. In the decoder coded packets tp_n play the role of parity check nodes, and message packets dp_k play the role of symbol nodes. The decoder of an LT code then operates as follows:

1. It looks for a parity check node tp_n that is connected to only one symbol node dp_k. If this is not the case, the algorithm cannot proceed to decode the coded packets.

2. It sets

$$\mathrm{dp}_k = \mathrm{tp}_n$$

3. It sums dp_k to all the parity check nodes tp'_n that are connected to dp_k as

$$\mathrm{tp}_{n'} = \mathrm{tp}_{n'} + \mathrm{dp}_k, \quad \text{for all } n' \text{ for which the bits in } G_{n'k} = 1$$

4. It removes all the connections related to symbol node dp_k.
5. Steps (1)–(4) are repeated for all dp_k.

Example 8.3: For the LT code described by the following matrix $\boldsymbol{G}$, determine the coded data packets for the message packet set $\mathrm{dp}_1\mathrm{dp}_2\mathrm{dp}_3 = (11, 10, 01)$, where the message data packets consist of two bits. Then decode the coded data packets.

$$G = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

According to the above generator matrix $\boldsymbol{G}$, supposed to be randomly generated, the coded data packets are

$$\begin{aligned} \mathrm{tp}_1 &= \mathrm{dp}_1 = 11 \\ \mathrm{tp}_2 &= \mathrm{dp}_1 \oplus \mathrm{dp}_3 = 10 \\ \mathrm{tp}_3 &= \mathrm{dp}_2 \oplus \mathrm{dp}_3 = 11 \\ \mathrm{tp}_4 &= \mathrm{dp}_1 \oplus \mathrm{dp}_2 = 01 \end{aligned}$$

This is expressed as the coded vector $\boldsymbol{tp} = (\mathrm{tp}_1\ \mathrm{tp}_2\ \mathrm{tp}_3\ \mathrm{tp}_4) = (11, 10, 11, 01)$

The corresponding bipartite graph is of the form as given in Figure 8.13.

The decoding procedure is then applied. A parity check node connected to only one symbol node is found (11), and then that packet is assigned the decoded packet $\mathrm{dp}_1 = \mathrm{tp}_1 = 11$. This result is added to the other parity check nodes connected to this symbol node, and then the connections are removed. This is seen in Figure 8.14.

After removing connections, the algorithm searches for another parity check node that is connected to only one symbol node, and chooses $\mathrm{dp}_2 = 10$. After appropriately summing the result and removing connections, the final configuration seen in Figure 8.15 determines the end of the decoding by setting $\mathrm{dp}_3 = 01$.

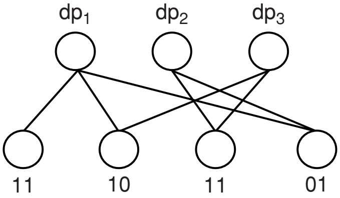


Figure 8.13 Bipartite graph for the LT code of Example 8.3

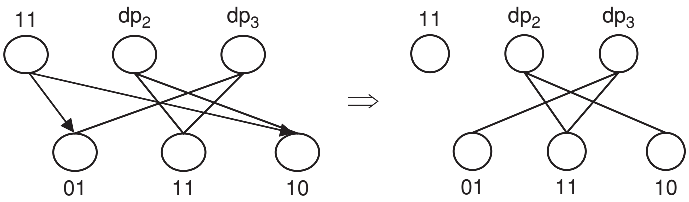


Figure 8.14 First steps in the decoding of the LT code of Example 8.3

A detailed description of the design of LT codes can be found in [24] and [25]. These codes can be used in many practical applications. One of them is as a coding technique for distributed multiuser information storage systems, where stored coded files that have been damaged could be recovered by discarding (erasing) them and then decoding suitable combinations of coded files stored elsewhere in the system.

8.10 LDPC and Turbo Codes

A common characteristic of these two coding techniques, which are the most efficient of all those described in this book, is that they can be iteratively decoded using alternating exchanges of soft-decision information. It is possible to demonstrate a certain degree of equivalence between the decoders for these two impressive error-control techniques, as seen in Section 8.8.1 for instance. There are, however, some differences between them.

LDPC codes are extremely good in terms of the BER performance if the code length is large enough. Thus, LDPC codes of length $n = 10{,}000$, for instance, have a BER performance curve that is less than 0.1 dB from the Shannon limit. But these long block lengths lead to significant decoding delay, and considerable encoding and decoding complexity.

On the other hand, turbo codes are constructed with relatively low complexity constituent codes, and they also show a very good BER performance, but the error floor effect is present at relatively high BERs. They are however more suitable for intermediate block or constraint length applications.

For both turbo and LDPC codes, the original iterative decoding methods are more complex than their logarithmic versions. Simplified variants of the logarithmic decoding algorithms lead to even lower complexity decoding algorithms, usually performed by applying max or min functions. As might be expected, however, the trade-off is some level of degradation in the corresponding BER performance.

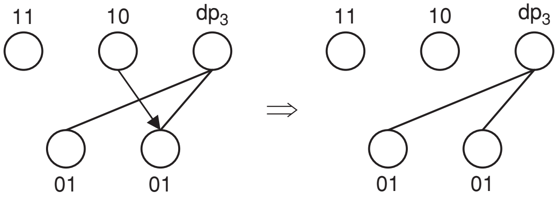


Figure 8.15 Final steps in the decoding of the LT code of Example 8.3

Bibliography and References

[1] Shannon, C. E., "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, pp. 379–423, 623–656, July and October 1948.

[2] Shannon, C. E., "Communications in the presence of noise," *Proc. IEEE*, vol. 86, no. 2, pp. 447–458, February 1998.

[3] Berrou, C., Glavieux, A. and Thitimajshima, P., "Near Shannon limit error-correcting coding and decoding: turbo codes," *Proc. 1993 IEEE International Conference on Communications*, Geneva, Switzerland, vol. 2, pp. 1064–1070, May 1993.

[4] MacKay, D. J. C. and Neal, R. M., "Near Shannon limit performance of low density parity check codes," *Electron. Lett.*, vol. 33, no. 6, March 13, 1997.

[5] MacKay, D. J. C. and Neal, R. M., "Good error-correcting codes based on very sparse matrices," available at http://www.inference.phy.cam.ac.uk/mackay/CodesGallager.html

[6] Gallager, R. G., "Low-density parity-check codes," *IRE Trans. Inf. Theory*, vol. IT-8, no. 1, pp. 21–28, January 1962.

[7] Tanner, L. M., "A recursive approach to low complexity codes," *IEEE Trans. Inf. Theory*, vol. 27, no. 5, pp. 533–547, 1981.

[8] Davey, M. C., *Error-Correction Using Low-Density Parity-Check Codes*, PhD Thesis, University of Cambridge, Cambridge, United Kingdom, 1999.

[9] Tang, H., Xu, J., Kou, Y., Lin, S. and Abdel-Ghaffar, K., "On algebraic construction of Gallager and circulant low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 50, no. 6, pp. 1269–1279, June 2004.

[10] Kou, Y., Lin, S. and Fossorier, M., "Low-density parity-check codes based on finite geometries: A rediscovery and new results," *IEEE Trans. Inf. Theory*, vol. 47, pp. 2711–2736, November 2001.

[11] Ammar, B., Honary B., Kou, Y., Xu J. and Lin, S., "Construction of low-density parity-check codes based on balanced incomplete block designs," *IEEE Trans. Inf. Theory*, vol. 50, no. 6, pp. 1257–1269, June 2004.

[12] Arnone, L., Gayoso, C., Gonzalez, C., and Castiñeira Moreira, J., "A LDPC logarithmic decoder implementation," *Proc. VIII International Symposium on Communications Theory and Applications*, St. Martin's College, Ambleside, United Kingdom, pp. 356–361, July 2005.

[13] LDPC toolkit for Matlab, available at http://arun-10.tripod.com/ldpc/generate.html

[14] Ten Brink, S., Kramer, G. and Ashikhmin, A., "Design of low-density parity-check codes for modulation and detection," *IEEE Trans. Commun.*, vol. 52, no. 4, pp. 670–678, April 2004.

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

[16] Ashikhmin, A., Kramer, G. and Ten Brink, S., "Extrinsic information transfer functions: A model and two properties," *Proc. Conf. Information Sciences and Systems*, Princeton, New Jersey, pp. 742–747, March 20–22, 2002.

[17] Sharon, E., Ashikhmim, A. and Litsyn, S., "EXIT functions for the Gaussian channel," *Prov. 40th Annu. Allerton Conf. Communication, Control, Computers*, Allerton, Illinois, pp. 972–981, October 2003.

[18] Etzion, T., Trachtenberg, A. and Vardy, A., "Which codes have cycle-free Tanner graphs?" *IEEE Trans. Inf. Theory*, vol. 45, no. 6, pp. 2173–2180, September 1999.

[19] Ten Brink, S., “Convergence behaviour of iteratively decoded parallel concatenated codes,” *IEEE Trans. Commun.*, vol. 49, pp. 1727–1737, October 2001.

[20] Ten Brink, S., Speidel, J. and Yan, R., “Iterative demapping and decoding for multilevel modulation,” *Proc. IEEE Globecom Conf. 98*, Sydney, NSW, Australia, vol. 1, pp. 579–584, November 1998.

[21] Ten Brink, S., “Exploiting the chain rule of mutual information for the design of iterative decoding schemes,” *Proc. 39th Allerton Conf.*, Monticello, Illinois, October 2001.

[22] Tuchler, M., Ten Brink, S. and Hagenauer, J., “Measures for tracing convergence of iterative decoding algorithms,” *Proc. 4th IEEE/ITG Conf. Source and Channel Coding*, Berlin, Germany, pp. 53–60, January 2002.

[23] McEliece, R. J., *The Theory of Information and Coding*, Addison-Wesley, Massachusetts, 1977.

[24] MacKay, D. J. C., “Digital fountain codes,” available at http://www.inference.phy.cam.ac.uk/mackay/DFountain.html

[25] Luby, M., “LT codes,” available at http://www.inference.phy.cam.ac.uk/mackay/dfountain/LT.pdf

[26] MacKay, D. J. C., Web site available at http://www.inference.phy.cam.ac.uk/mackay/

[27] MacKay, D. J. C., “Fountain codes,” *IEE Proc. Commun.*, vol. 152, no. 6, pp. 1062–1068, December 2005.

[28] MacKay, D. J. C., *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press, Cambridge, United Kingdom, 2003.

Problems

8.1 (a) Determine the number and size of the short cycles in the bipartite graph of the irregular LDPC code $C_b(12, 4)$ described in Section 8.5.

(b) Reduce the number of short cycles by changing the positions of ‘1’s in the parity check matrix of item (a), but keeping the number of ‘1’s per column the same, $s = 3$.

(c) Does the modified Tanner graph correspond to the same LDPC code or to a different one?

8.2 A simple binary cyclic LDPC code can be constructed from the following circulant matrix:

$$\boldsymbol{M} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

(a) Determine the rank of the above matrix, and use that to find the cyclic and systematic parity check matrices of the code, and its systematic generator matrix.

(b) Calculate the average number of 1s per row and column of the two parity check matrices, and hence calculate the rate of the code, confirming that it is the same in both cases, and also the same as the rate calculated from the dimensions of the parity check matrices. What is the Hamming distance of the code?

(c) Sketch the Tanner graphs of the two parity check matrices and of the circulant matrix, and determine the length of the shortest cycle in each case. Which graph might be best for decoding the code by means of the sum–product algorithm?

8.3 For the LDPC code of Problem 8.2,

(a) Use the systematic generator matrix found in part (a) of Problem 8.2 to determine the codeword corresponding to the message vector $\boldsymbol{m} = (100)$.

(b) The codeword of part (a) of this problem is transmitted over an AWGN channel in normalized polar format (±1), and is received as the vector $\boldsymbol{r} = \left(1.0187\ -0.6225\ 2.0720\ 1.6941\ -1.3798\ -0.7431\ -0.2565\right)$.

Using the sum–product algorithm, decode this received vector on each of the three Tanner graphs found in part (c) of Problem 8.2, and comment on the processes and the results obtained.

8.4 The block code $C_{\mathrm{b}}(5, 3)$ introduced in Example 7.1 in Chapter 7 has the following generator and parity check matrices:

$$\boldsymbol{G} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}, \qquad \boldsymbol{H} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

In that example the codeword $\boldsymbol{c} = (00000)$ is transmitted over a soft-decision channel, and the corresponding received vector is $\boldsymbol{r} = (10200)$. This soft-decision channel is described in Chapter 7, in Figure 7.6 and Table 7.2.

(a) Decode the received vector using the SPA over the parity check matrix $\boldsymbol{H}$ to show that, after enough iterations, the decision of the decoder fluctuates between the two code vectors $\boldsymbol{c} = (00000)$ and $\boldsymbol{c} = (10100)$, which are the closest to the received vector $\boldsymbol{r} = (10200)$.

(b) Describe the deficiencies of the bipartite graph associated with the parity check matrix $\boldsymbol{H}$ of this code, with respect to the iterative passing of information.

■

Appendix A: Error Probability in the Transmission of Digital Signals

The two main problems in the transmission of digital data signals are the effects of channel noise and inter-symbol interference (ISI) [2, 4]. In this appendix the effect of the channel noise, assumed to be additive white Gaussian noise (AWGN), is studied, in the absence of inter-symbol interference.

A.1 Digital Signalling

A.1.1 Pulse Amplitude Modulated Digital Signals

A digital signal can be described as a sequence of pulses that are amplitude modulated. The corresponding signal is of the form

$$x(t) = \sum_{k=-\infty}^{k=\infty} a_k p(t - kT) \tag{1}$$

where coefficient a_k is the kth symbol of the sequence, such that the coefficient a_k is one of the M possible values of the information to be transmitted, taken from a discrete alphabet of symbols. The pulse $p(t)$ is the basic signal to be transmitted, which is multiplied by a_k to identify the different signals that make up the transmission.

The signal $a_k p(t - kT)$ is the kth symbol that is transmitted at the kth time interval, where T is the duration of such a time interval. Thus, the transmission consists of a sequence of amplitude-modulated signals that are orthogonal in the time domain.

As seen in Figure A.1, the data sequence $a_k = A, 0, A, A, 0, A$, corresponding to digital information in binary format (101101), is a set of coefficients that multiply a normalized basic signal or pulse $p(t - kT)$. If these coefficients are selected from an alphabet $\{0, A\}$, the digital transmission is said to have a unipolar format. If coefficients are selected from an alphabet

Essentials of Error-Control Coding Jorge Castiñeira Moreira and Patrick Guy Farrell

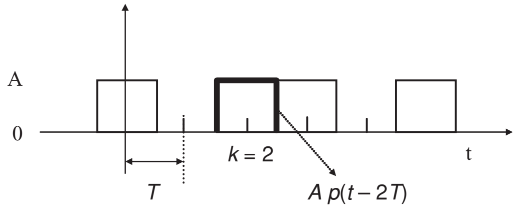


Figure A.1 A digital signal

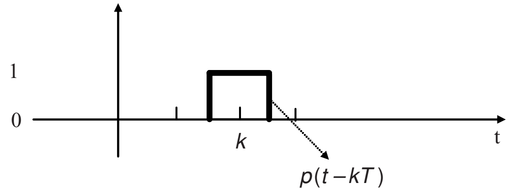


Figure A.2 A normalized pulse at time interval k multiplied by a given coefficient a_k

$\{-A/2, A/2\}$, the digital transmission is said to have a polar format. In this latter case, the sequence of this example would be given by $a_k = A/2, -A/2, A/2, A/2, -A/2, A/2$.

Index k adopts integer values from minus to plus infinity. As seen in Figure A.2, the basic signal $p(t)$ is normalized and of fixed shape, centred at the corresponding time interval k, and multiplied by a given coefficient that contains the information to be transmitted. This basic normalized pulse is such that

$$p(t) = \begin{cases} 1 & t = 0 \\ 0 & t = \pm T, \pm 2T, \ldots \end{cases} \tag{2}$$

The normalized pulse is centred at the corresponding time interval k and so its sample value at the centre of that time interval is equal to 1, whereas its samples obtained at time instants different from $t = kT$ are equal to 0. This condition does not necessarily imply that the pulse is time limited. Samples are taken synchronously at time instants $t = kT$, where $k = 0$, $\pm 1, \pm 2, \ldots$, such that for a particular time instant $t = k_1 T$,

$$x(k_1 T) = \sum_{\infty} a_{k_1} p(k_1 T - kT) = a_{k_1} \tag{3}$$

since $(k_1 T - kT) = 0$, for every k, except $k = k_1$.

Conditions (2) describe the transmission without ISI, and are satisfied by many signal pulse shapes. The classic rectangular pulse satisfies condition (2) if its duration τ is less than or equal to T. The pulse $\sin c(t)$ also satisfies the orthogonality condition described in the time domain by equation (2), but it is a pulse that is unlimited in time, however. Figure A.3 shows the transmission of the binary information sequence (11001), using $\sin c(t)$ pulses modulated in a polar format. At each sampling time instant $t = kT$, the pulse being sampled has amplitude different from 0, while the other pulses are all equal to 0.

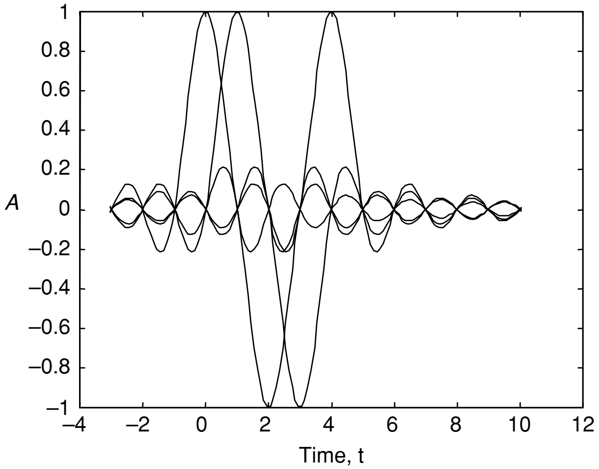


Figure A.3 A digital transmission using sinc (t) pulses modulated in polar format

Each pulse occurs in a time interval of duration T. The inverse of this duration is the symbol rate of the transmission, since it is the number of symbols that are transmitted in a unit of time (usually a second). The symbol rate r is then equal to

$$r = 1/T \text{ (symbols per second)} \tag{4}$$

which is measured in symbols per second. When the discrete alphabet used in the transmission contains only two symbols, $M = 2$, then it is binary transmission, and the corresponding symbol rate $r = r_\mathrm{b}$ is the binary signalling rate

$$r_\mathrm{b} = 1/T_\mathrm{b} \text{ (bit per second)} \tag{5}$$

where $T = T_\mathrm{b}$ is the in time duration of each bit. The binary signalling rate is measured in bits per second (bps).

A.2 Bit Error Rate

Figure A.4 shows the basic structure of a binary receiver.

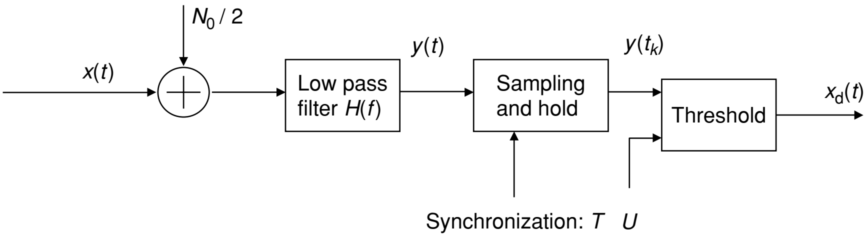


Figure A.4 A binary receiver

The signal $x(t)$ is a digital signal $\sum_k a_k p(t - kT)$, that is, an amplitude-modulated pulse signal. This signal is affected by AWGN noise in the channel and is then input to the receiver. The first block in this receiver is a low pass filter that eliminates part of the input noise without producing ISI, giving the signal $y(t)$. The receiver takes synchronized samples of this signal, and generates after the sample-and-hold operation a random variable of the form

$$y(t_k) = a_k + n(t_k) \tag{6}$$

Sampled values $y(t_k)$ constitute a continuous random variable Y, and noise samples $n(t_k)$ taken from a random signal $n(t)$ form a random variable n.

The lowest complexity decision rule for deciding the received binary value is the so-called hard decision, which consists only of comparing the sampled value $y(t_k)$ with a threshold U, such that if $y(t_k) > U$, then the receiver considers that the transmitted bit is a 1, and if $y(t_k) < U$ then the receiver considers that the transmitted bit is a 0. In this way the received sampled signal $y(t_k)$ is converted into a signal $x_{\mathrm{hd}}(t)$, basically of the same kind as that expressed in equation (1), an apparently noise-free signal but possibly containing some errors with respect to the original transmitted signal.

The probability density function of the random variable Y is related to the noise, and to conditional probability of the transmitted symbols. The following hypotheses are relevant:

H_0 is the hypothesis that a ‘0’ was transmitted $\quad a_k = 0, Y = n$

H_1 is the hypothesis that a ‘1’ was transmitted $\quad a_k = A, Y = A + n.$

The probability density function of the random variable Y conditioned on the event H_0 is given by

$$p_{\mathrm{Y}}(y/H_0) = p_{\mathrm{N}}(y) \tag{7}$$

where $p_{\mathrm{N}}(y)$ is the Gaussian probability density function.

For hypothesis H_1,

$$p_{\mathrm{Y}}(y/H_1) = p_{\mathrm{N}}(y - A) \tag{8}$$

The probability density function in this case is shifted to the value $n = y - A$. Thus, the probability density function for the noisy signal is the probability density function for the noise-free discrete signal 0 or A (unipolar format) added to the probability density function of the noise $p_{\mathrm{N}}(n)$. Figure A.5 shows the reception of a given digital signal performed using hard decision.

The probability density function for each signal is the Gaussian probability density function centred at the value of the amplitude that is transmitted.

Figure A.6 shows the shadowed areas under each probability density function that correspond to the probability of error associated with each hypothesis. Thus, the receiver assumes that if $Y < U$, hypothesis H_0 has occurred, and if $Y > U$, hypothesis H_1 has occurred. Error

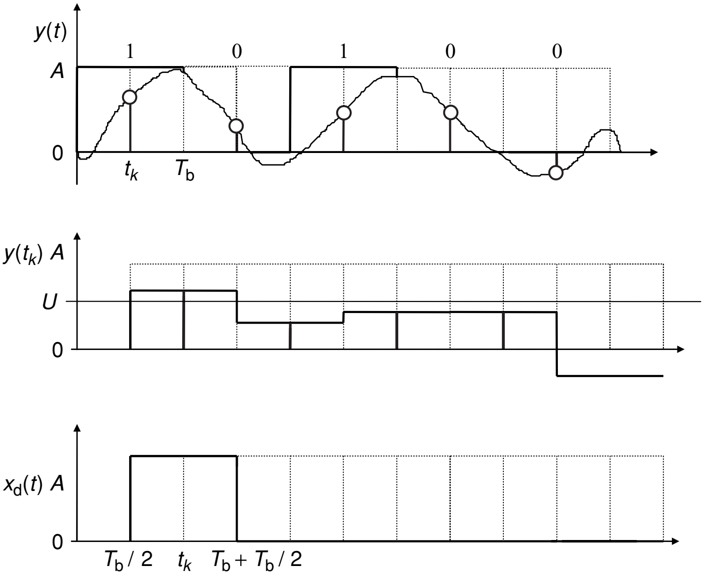


Figure A.5 Reception of a digital signal

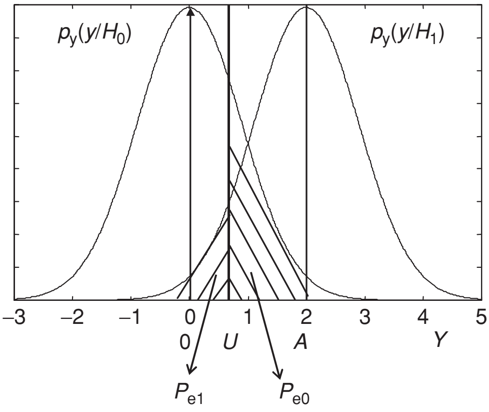


Figure A.6 Bit error rate calculation

probabilities associated with each hypothesis are described in Figure A.6, and are equal to

$$P_{e0} = P(Y > U/H_0) = \int_U^{\infty} p_{\mathrm{Y}}(y/H_0)\,\mathrm{d}y \tag{9}$$

$$P_{e1} = P(Y < U/H_1) = \int_{-\infty}^{U} p_Y(y/H_1)\,\mathrm{d}y \tag{10}$$

The threshold value U should be conveniently determined. A threshold value U close to the amplitude 0 reduces the error probability associated with the symbol '1', but strongly increases the error probability associate with the symbol '0', and vice versa. The error probability of the whole transmission is an average over these two error probabilities, and its calculation can lead to a proper determination of the value of the threshold U:

$$P_e = P_0 P_{e0} + P_1 P_{e1} \tag{11}$$

where $P_0 = P(H_0)$, $P_1 = P(H_1)$.

P_0 and P_1 are the source symbol probabilities; that is, the probabilities of the transmission of a symbol '0' and '1'. The average error probability is precisely the mean value of the errors in the transmission that takes into account the probability of occurrence of each symbol.

The derivative with respect to the threshold U of the average error probability is set to be equal to zero, to determine the optimal value of the threshold:

$$\mathrm{d}P_e/\mathrm{d}U = 0 \tag{12}$$

This operation leads to the following expression:

$$P_0 p_Y(U_{\mathrm{opt}}/H_0) = P_1 p_Y(U_{\mathrm{opt}}/H_1) \tag{13}$$

If the symbols '0' and '1' of the transmission are equally likely

$$P_0 = P_1 = \frac{1}{2} \tag{14}$$

then

$$P_e = \frac{1}{2}(P_{e0} + P_{e1}) \tag{15}$$

and the optimal value of the threshold is then

$$p_{\mathrm{Y}}(U_{\mathrm{opt}}/H_0) = p_{\mathrm{Y}}(U_{\mathrm{opt}}/H_1) \tag{16}$$

As seems reasonable, the optimal value of the threshold U is set to be in the middle of the two amplitudes, $U_{\mathrm{opt}} = A/2$, if the symbol source probabilities are equal; that is, if symbols are equally likely (see Figure A.6).

The Gaussian probability density function with zero mean value and variance σ^2 characterizes the error probability of the involved symbols if they are transmitted over the AWGN channel. This function is of the form

$$p_{\mathrm{N}}(y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{y^2}{2\sigma^2}} \tag{17}$$

In general, this probability density function is shifted to a mean value m and has a variance σ^2, such that

$$p_{\mathrm{N}}(y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-m)^2}{2\sigma^2}} \tag{18}$$

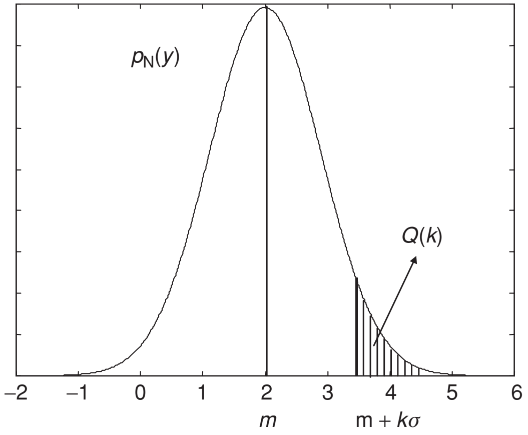


Figure A.7 Normalized Gaussian probability density function $Q(k)$

The probability that a given value of the random variable Y is larger than a value $m + k\sigma$ is a function of the number k, and it is given by

$$P(Y > m + k\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \int_{m+k\sigma}^{\infty} e^{-\frac{(y-m)^2}{2\sigma^2}}\,\mathrm{d}y \tag{19}$$

These calculations are simplified by using the normalized Gaussian probability density function, also known as the function $Q(k)$ (Figure A.7):

$$Q(k) = \frac{1}{\sqrt{2\pi}} \int_{k}^{\infty} e^{-\frac{(\lambda)^2}{2}}\,\mathrm{d}\lambda \tag{20}$$

obtained by putting

$$\lambda = \frac{y - m}{\sigma} \tag{21}$$

This normalized function can be used to calculate the error probabilities of the digital transmission described in equations (9) and (10).

$$P_{e0} = \int_{U}^{\infty} p_{\mathrm{N}}(y)\,\mathrm{d}y = \frac{1}{\sqrt{2\pi\sigma^2}} \int_{U}^{\infty} e^{-\frac{y^2}{2\sigma^2}}\,\mathrm{d}y = Q(U/\sigma) \tag{22}$$

and

$$P_{e1} = \int_{-\infty}^{U} p_{\mathrm{N}}(y - A)\,\mathrm{d}y = \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{U} e^{-\frac{(y-A)^2}{2\sigma^2}}\,\mathrm{d}y = Q((A - U)\,/\,\sigma) \tag{23}$$

If $U = U_{\text{opt}}$, the curves intersect in the middle point $U_{\text{opt}} = A/2$.

In this case these error probabilities are equal to

$$P_{e0} = P_{e1} = Q\left(\frac{A}{2\sigma}\right) \tag{24}$$

$$P_e = \frac{1}{2}(P_{e0} + P_{e1}) = Q\left(\frac{A}{2\sigma}\right)$$

This is the minimum value of the average error probability for the transmission of two equally likely symbols over the AWGN channel. As seen in the above expressions, the term $A/2\sigma$ (or equivalently its squared value) defines the magnitude of the number of errors in the transmission, that is, the error probability or bit error rate of the transmission.

The result is the same for transmission using the polar format ($a_k = \pm A/2$), if the symbol amplitudes remain the same distance A apart.

The above expressions for the error probability can be generalized for the transmission of M symbols taken from a discrete source, and they can also be described in terms of the signal-to-noise ratio. The power associated with the transmission of the signal described in equation (1) is useful for this latter purpose. Let us take a sufficiently long time interval T_0, such that $T_0 = NT$, and $N \gg 1$. The amplitude-modulated pulse signal uses the normalized pulse

$$p(t) = \begin{cases} 1 & |t| < \tau/2 \\ 0 & ||t| > \tau/2 \end{cases} \tag{25}$$

where $\tau \leq T$. Then the power associated with this signal is equal to

$$\begin{aligned}
S_{\mathrm{R}} &= \frac{1}{T_0}\int_{-T_0/2}^{T_0/2}\left(\sum_k a_k p(t-kT)^2\right)\mathrm{d}t = \frac{1}{T_0}\int_{-T_0/2}^{T_0/2}\sum_k a_k^2 p^2(t-kT)\,\mathrm{d}t \\
&= \frac{1}{T_0}\int_{-T_0/2}^{T_0/2}\sum_{k=-N/2}^{k=N/2} a_k^2 p^2(t-kT)\,\mathrm{d}t \\
S_{\mathrm{R}} &= \sum_k \frac{1}{NT}\int_{-T/2}^{T/2} a_k^2 p^2(t)\,\mathrm{d}t = \frac{N_0}{NT}\int_{-T/2}^{T/2} a_0^2 p^2(t)\,\mathrm{d}t + \frac{N_1}{NT}\int_{-T/2}^{T/2} a_1^2 p^2(t)\,\mathrm{d}t \\
S_{\mathrm{R}} &= P_0\frac{1}{T}\int_{-\tau/2}^{\tau/2} a_0^2 p^2(t)\,\mathrm{d}t + P_1\frac{1}{T}\int_{-\tau/2}^{\tau/2} a_1^2 p^2(t)\,\mathrm{d}t
\end{aligned} \tag{26}$$

The duration of the pulse can be equal to the whole time interval $T = \tau = T_{\mathrm{b}}$, in this case it is said that the format is non-return-to-zero (NRZ), or it can be shorter than the whole time interval $\tau < T_{\mathrm{b}}$, then the format is said to be return-to-zero (RZ). For the NRZ format,

$$S_{\mathrm{R}} = A^2/2 \quad \text{unipolar NRZ}$$

$$S_{\mathrm{R}} = A^2/4 \quad \text{polar NRZ}$$

$$A = \begin{cases} \sqrt{2S_{\mathrm{R}}} & \text{unipolar} \\ \sqrt{4S_{\mathrm{R}}} & \text{polar} \end{cases} \tag{27}$$

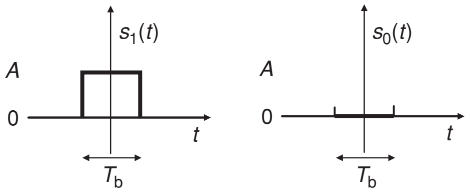


Figure A.8 Signalling in the NRZ unipolar format

If σ^2 is the noise power N_R at the output of the receiver filter, then

$$\left(\frac{A}{2\sigma}\right)^2 = \frac{A^2}{4N_R} = \begin{cases} (1/2)(S/N)_R & \text{unipolar} \\ (S/N)_R & \text{polar} \end{cases} \tag{28}$$

Thus, unipolar format needs twice the signal-to-noise ratio to have the same BER performance as that of the polar format.

The error probability was determined as a function of the parameter $A/2\sigma$. However, a more convenient way of describing this performance is by means of the so-called average bit energy-to-noise power spectral density ratio E_b/N_0. This new parameter requires the following definitions:

$$E_b = \frac{S_R}{r_b} \quad \text{average bit energy} \tag{29}$$

$$\frac{E_b}{N_0} = \frac{S_R}{N_0 r_b} \quad \text{average bit energy-to-noise power spectral density ratio} \tag{30}$$

The average bit energy of a sequence of symbols such as those described by the digital signal (1) is calculated as

$$E_b = E\left[a_k^2 \int_{-\infty}^{\infty} p^2(t - kD)\,dt\right] = E\left[a_k^2 \int_{-\infty}^{\infty} p^2(t)\,dt\right] = \overline{a_k^2} \int_{-\infty}^{\infty} p^2(t)\,dt \tag{31}$$

The above parameters are calculated for the unipolar NRZ format. In this format a ‘1’ is usually transmitted as a rectangular pulse of amplitude A, and a ‘0’ is transmitted with zero amplitude as in Figure A.8.

The average bit energy E_b is equal to

$$E_1 = \int_0^{T_b} s_1^2(t)\,dt = A^2 T_b$$

$$E_0 = \int_0^{T_b} s_0^2(t)\,dt = 0$$

$$E_b = P_0 E_0 + P_1 E_1 = \frac{1}{2}(E_0 + E_1) = \frac{A^2 T_b}{2} \tag{32}$$

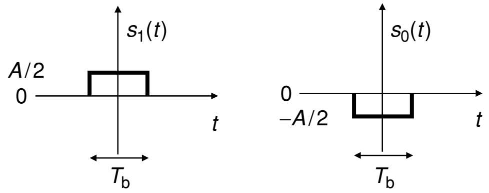


Figure A.9 Signalling in the NRZ polar format

Since the transmission is over the AWGN channel of bandwidth B and for the maximum possible value of the symbol or bit rate $r_b = 2B$, [1–4], the input noise is equal to

$$N_R = \sigma^2 = N_0 B = \frac{N_0 r_b}{2} \tag{33}$$

This is the minimum amount of noise that inputs the receiver if a matched filter is used [1–4]. The quotient $(A/2\sigma)^2$ can be now expressed as

$$\left(\frac{A}{2\sigma}\right)^2 = \frac{A^2}{4\sigma^2} = \frac{2E_b r_b}{4N_0 r_b/2} = \frac{E_b}{N_0} \tag{34}$$

In the case of the NRZ polar format, where a '1' is usually transmitted as a rectangular pulse of amplitude $A/2$ and a '0' is transmitted as a rectangular pulse of amplitude $-A/2$ (Figure A.9).

Then the average bit energy E_b is

$$E_1 = \int_0^{T_b} s_1^2(t)\,dt = \frac{A^2 T_b}{4}$$

$$E_0 = \int_0^{T_b} s_0^2(t)\,dt = \frac{A^2 T_b}{4}$$

$$E_b = P_0 E_0 + P_1 E_1 = \frac{1}{2}(E_0 + E_1) = \frac{A^2 T_b}{4} \tag{35}$$

and so

$$\left(\frac{A}{2\sigma}\right)^2 = \frac{A^2}{4\sigma^2} = \frac{4E_b r_b}{4N_0 r_b/2} = \frac{2E_b}{N_0} \tag{36}$$

It is again seen that the polar format has twice the value of $(A/2\sigma)^2$ for a given value of E_b/N_0 with respect to the unipolar format:

$$\left(\frac{A}{2\sigma}\right)^2 = \begin{cases} \frac{E_b}{N_0} & \text{unipolar} \\ \frac{2E_b}{N_0} & \text{polar} \end{cases} \tag{37}$$

Now expressing the error probabilities of the two formats in terms of the parameter E_b/N_0, we obtain

$$P_e = \begin{cases} Q\left(\sqrt{\frac{E_\mathrm{b}}{N_0}}\right) & \text{unipolar} \\ Q\left(\sqrt{\frac{2E_\mathrm{b}}{N_0}}\right) & \text{polar} \end{cases} \tag{38}$$

This is the minimum value of the error probability and is given when the receiver uses the matched filter. Any other filter will result in a higher bit error rate than that expressed in (38). The matched filter is optimum in terms of maximizing the signal-to-noise ratio for the reception of a given pulse shape, over a given channel transfer function, and affected by a given noise probability density function.

Bibliography

[1] Carlson, A. B., *Communication Systems: An Introduction to Signals and Noise in Electrical Communication*, 3rd Edition, McGraw-Hill, New York, 1986.

[2] Sklar, B., *Digital Communications: Fundamentals and Applications*, Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[3] Couch, L. W., *Digital and Analog Communications Systems*, MacMillan, New York, 1996.

[4] Proakis, J. G. and Salehi, M., *Communication Systems Engineering*, Prentice Hall, Englewood Cliffs, New Jersey, 1994.

Appendix B: Galois Fields GF(q)

This appendix is devoted to an introduction to finite fields, usually called Galois fields GF(q). A related algebraic structure called a group is first described. The aim of this appendix is to define polynomial operations over these algebraic structures. The main concept in terms of its utility for designing error-control codes is that a polynomial defined over a finite field GF(p_{prime}) has roots in that field, or in one of its extensions GF(q). In the same way, each element a of the extended finite field GF(q) is a root of some polynomials with coefficients in the finite field GF(p_{prime}). The polynomial of minimum degree that satisfies this condition is called a minimum polynomial of a.

B.1 Groups

A group G_{r} is defined as a set of elements that are related by some specific operations. For a given group G_{r} of elements, the binary operation $*$ is defined as an assignment rule for any two elements of this group, a and b. In this rule these two elements are assigned a unique element c of the same group, such that $c = a^*b$. This operation is said to be closed over the group G_{r} because its result is another element of the same group. This operation is said to be associative if it satisfies

$$a * (b * c) = (a * b) * c \tag{1}$$

B.1.1 Definition of a Group G_{r}

A set of elements G_{r} over which the binary operation $*$ is defined is said to be a group, if the following conditions are satisfied:

1. The binary operation $*$ is associative.
2. The set of elements G_{r} contains an element e, such that for every element of the set $a \in G_{\text{r}}$,

$$e * a = a * e = a \tag{2}$$

The element e is called the identity for the binary operation $*$.

Essentials of Error-Control Coding Jorge Castiñeira Moreira and Patrick Guy Farrell

3. For every element of the set $a \in G_\mathrm{r}$, there is another element of the same set $a' \in G_\mathrm{r}$, such that

$$a * a' = a' * a = e \tag{3}$$

The element a' is called the inverse element of a.

A group G_r is said to be commutative if, for every pair of its elements $a, b \in G_\mathrm{r}$, it is true that

$$a * b = b * a \tag{4}$$

It can be shown that both the inverse element a' of an element a and the identity e of the binary operation defined over the group G_r are unique.

B.2 Addition and Multiplication Modulo *m*

For a set of elements $G_\mathrm{r} = \{0, 1, 2, \ldots, i, j, \ldots, m-1\}$ that satisfies the conditions for being a group, the addition operation $\oplus$ between any two of its elements i and j is defined as

$$i \oplus j = r$$

$$r = (i + j)\mathrm{mod}(m) \tag{5}$$

that is, the addition of any two elements of the group i and j is the remainder of the division of the arithmetic addition $(i + j)$ by m. This operation is called modulo-m addition.

Modulo-2 addition, for instance, is defined over the group $G_\mathrm{r} = \{0, 1\}$:

$$0 \oplus 0 = 0,$$

$$1 \oplus 1 = 0,$$

$$0 \oplus 1 = 1,$$

$$1 \oplus 0 = 1,$$

As an example, the last result comes from the calculation of $1 + 0 = 1$, and $1/2 = 0$ with remainder 1, then $1 \oplus 0 = 1$. A group constituted of p_prime elements $G_\mathrm{r} = \{1, 2, 3, \ldots, p_\mathrm{prime} - 1\}$, where p_prime is a prime number. p_prime : 2, 3, 5, 7, 11, ... is a commutative group under modulo-p_prime addition.

Multiplication modulo-p_prime between any two elements i and j is defined as

$$i \otimes j = r$$

$$r = ij \bmod p_\mathrm{prime} \tag{6}$$

For the binary group $G_\mathrm{r} = \{0, 1\}$, this operation is determined by the following table:

$$0 \otimes 0 = 0$$

$$1 \otimes 1 = 1$$

$$0 \otimes 1 = 0$$

$$1 \otimes 0 = 0$$

Table B.1 Modulo-2 addition

$\oplus$	0	1
0	0	1
1	1	0

As an example, the last result of the above table comes from the calculation of $1 \times 0 = 0$, and $0/2 = 0$ with remainder 0, then $1 \otimes 0 = 0$.

B.3 Fields

The definition of groups is useful for introducing the definition of what is called a finite field. A field is a set of elements F for which addition, multiplication, subtraction and division performed with its elements result in another element of the same set. Once again, the definition of a field is based on the operations described over such a field. For addition and multiplication operations, the following conditions define a field:

1. F is a commutative group with respect to the addition operation. The identity element for the addition is called '0'.
2. F is a commutative group for the multiplication operation. The identity element for multiplication is called '1'.
3. Multiplication is distributive with respect to addition:

$$a(b + c) = ab + ac \tag{7}$$

The number of elements of a field is called the order of that field. A field with a finite number of elements is usually called a finite field, or Galois field GF.

The inverse for the addition operation of an element of the field $a \in F$ is denoted as $-a$, and inverse for the multiplication operation of an element of the field is denoted as a^{-1}. Subtraction and division operations are defined as a function of the inverse elements as

$$\begin{aligned} a - b &= a + (-b) \\ a/b &= a(b^{-1}) \end{aligned} \tag{8}$$

The set $G_{\mathrm{r}} = \{0, 1\}$ defined under addition and multiplication modulo 2 is such that $G_{\mathrm{r}} = \{0, 1\}$ is a commutative group with respect to the addition operation, and is also a commutative group with respect to the multiplication operation. This is the so-called binary field GF(2).

Operations in this binary field are defined by Tables B.1 and B.2.

Table B.2 Modulo-2 multiplication

•	0	1
0	0	0
1	0	1

For a given prime number p_{prime}, the set of integer numbers $\{0, 1, 2, 3, \ldots, p_{\text{prime}} - 1\}$ is a commutative group with respect to modulo-p_{prime} addition. The set of integer numbers $\{1, 2, 3, \ldots, p_{\text{prime}} - 1\}$ is a commutative group with respect to multiplication modulo p_{prime}. This set is therefore a field of order p_{prime}. They are also called prime fields GF (p_{prime}).

An extension of a prime field GF(p_{prime}) is called an extended finite field GF(q) = GF(p_{prime}^m), with m a positive integer number. This extended field is also a Galois field. Particular cases of practical interest are the finite fields of the form GF(2^m), with m a positive integer number.

For a given finite field GF(q), and for an element of this field $a \in$ GF(q), the powers of this element are also elements of the finite field, since the multiplication operation is a closed operation. Therefore,

$$a^1 = a, \qquad a^2 = a \bullet a, \qquad a^3 = a \bullet a \bullet a \ldots$$

are also elements of the same finite field GF(q). However, these powers will start to repeat because the field is a finite field, and its order is a finite number.

In other words, there should exist two integer numbers k and m, such that $m > k$ and $a^m = a^k$. Since a^{-k} is the multiplicative inverse of a^k, $a^{-k}a^m = a^{-k}a^k$, or $a^{m-k} = 1$. There is therefore a number n such that $a^n = 1$, and this number is called the order of the element a. Thus, powers $a^1, a^2, a^3, \ldots, a^{n-1}$ are all different and form a group under multiplication in GF(q).

It can be shown that if a is a non-zero element of the finite field GF(q), then $a^{q-1} = 1$. It is also true that if a is a non-zero element of the finite field GF(q), and if n is the order of that element, then n divides $q - 1$.

A non-zero element a of a finite field GF(q) is said to be a primitive element of that field if the order of that element is $q - 1$. All the powers of a primitive element $a \in$ GF(q) of a field generate all the non-zero elements of that field GF(q). Every finite field has at least one primitive element.

B.4 Polynomials over Binary Fields

The most commonly used fields are extensions of the binary field GF(2), and they are called Galois fields GF(2^m). Binary arithmetic uses addition and multiplication modulo 2. A polynomial $f(X)$ defined over GF(2) is of the form

$$f(X) = f_0 + f_1X + f_2X^2 + \ldots + f_nX^n \tag{9}$$

where the coefficients f_i are either 0 or 1. The highest exponent of the variable X is called the degree of the polynomial. There are 2^n polynomials of degree n. Some of them are

$$n = 1 \qquad X, X + 1$$

$$n = 2 \qquad X^2, 1 + X^2, X + X^2, 1 + X + X^2$$

Polynomial addition and multiplication are done using operations modulo 2, and satisfy the commutative, associative and distributive laws. An important operation is the division of two polynomials. As an example, the division of polynomial $X^3 + X + 1$ by the polynomial $X + 1$

is done as follows:

$$
\begin{array}{lll}
X^3 \qquad + X + 1 & | & X + 1 \\
 & & -\,-\,-\,-\,-\,-\,-\,-\,- \\
X^3 + X^2 & & X^2 + X \\
-\,-\,-\,-\,-\,-\,-\,- & & \\
\quad X^2 + X + 1 & & \\
\quad X^2 + X & & \\
\quad -\,-\,-\,-\,- & & \\
\qquad r(X) = 1 & &
\end{array}
$$

The division is of the form

$$f(X) = q(X)g(X) + r(X) \tag{10}$$

where, in this example,

$$r(X) = 1$$
$$q(X) = X + X^2$$

Definition B.1: An element of the field a is a zero or root of a polynomial $f(X)$ if $f(a) = 0$. In this case a is said to be a root of $f(X)$ and it also happens that $X - a$ is factor of this polynomial $f(X)$.

Thus, for example, $a = 1$ is a root of the polynomial$f(X) = 1 + X^2 + X^3 + X^4$, and so $X + 1$ is a factor of this polynomial $f(X)$. The division of $f(X)$ by $X + 1$ has the quotient polynomial $q(X) = 1 + X + X^3$. Remember that the additive inverse of a, $-a$, is equal to a, $a = -a$, for modulo-2 operations.

Definition B.2: A polynomial $p(X)$ defined over GF(2), of degree m, is said to be irreducible, if $p(X)$ has no factor polynomials of degree higher than zero and lower than m.

For example, the polynomial $1 + X + X^2$ is an irreducible polynomial, since neither X nor $X + 1$ are its factors. A polynomial of degree 2 is irreducible if it has no factor polynomials of degree 1. A property of irreducible polynomials over the binary field GF(2), of degree m, is that they are factors of the polynomial $X^{2^m-1} + 1$. For example, the polynomial $1 + X + X^3$ is a factor of $X^{2^3-1} + 1 = X^7 + 1$.

Furthermore, an irreducible polynomial $p_i(X)$ of degree m is a primitive polynomial if the smallest integer number n, for which $p_i(X)$ is a factor of $X^n + 1$, is $n = 2^m - 1$. For example, the polynomial $X^4 + X + 1$ is a factor of $X^{2^4-1}{+}1 = X^{15}{+}1$, and it is not a factor of any other polynomial of the form $X^n + 1$, where $1 \leq n < 15$. This means that the polynomial $X^4 + X + 1$ is a primitive polynomial.

Another interesting property of polynomials over GF(2) is that

$$(f(X))^{2^l} = f(X^{2^l}) \tag{11}$$

B.5 Construction of a Galois Field GF(2^m)

An extended Galois field contains not only the binary elements ‘0’ and ‘1’ but also the element α and its powers. For this new element,

$$0\alpha = \alpha 0 = 0$$
$$1\alpha = \alpha 1 = \alpha$$
$$\alpha^2 = \alpha\alpha, \qquad \alpha^3 = \alpha\alpha^2$$
$$\alpha^i \alpha^j = \alpha^{i+j} = \alpha^j \alpha^i$$

A set of these elements is

$$F = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^k, \ldots\} \tag{12}$$

which contains 2^m elements. Since a primitive polynomial $p_i(X)$, over GF(2) of degree m, is a factor of $X^{2^m-1}+1$, and taking into account that $p_i(\alpha) = 0$,

$$\begin{aligned} X^{2^m-1} + 1 &= p(X)q(X) \\ \alpha^{2^m-1} + 1 &= p(\alpha)q(\alpha) = 0 \\ \alpha^{2^m-1} &= 1 \end{aligned} \tag{13}$$

Therefore the set F is a finite set of 2^m elements:

$$F = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^{2^m-2}\} \tag{14}$$

The condition

$$i + j < 2^m - 1 \tag{15}$$

should be satisfied to make the set be closed with respect to the multiplication operation. This means that if any two elements of the set α^i and α^j are multiplied, the result $\alpha^k = \alpha^i \alpha^j$ should be an element of the same set; that is, $k < 2^m - 1$.

If

$$i + j = (2^m - 1) + r, \quad 0 \le r < 2^m - 1 \tag{16}$$

then

$$\alpha^i \alpha^j = \alpha^{(i+j)} = \alpha^{(2^m-1)+r} = \alpha^r$$

and this result shows that the set is closed with respect to the multiplication operation. On the other hand, for a given integer number i, such that $0 < i < 2^m - 1$,

$$\alpha^{2^m-1-i} \text{ is the multiplicative inverse of } \alpha^i \tag{17}$$

Thus, the set $F = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^{2^m-2}\}$ is a group of order $2^m - 1$ with respect to the multiplication operation. To ensure that the set F is a commutative group under addition, the operation of addition in the set must be defined. For $0 \leq i < 2^m - 1$, X^i is divided by $p(X)$, resulting in

$$X^i = q_i(X)p(X) + a_i(X) \tag{18}$$

$a_i(X)$ is of degree $m - 1$ or less, and $a(X) = a_{i0} + a_{i1}X + a_{i2}X^2 + \cdots + a_{i,m-1}X^{m-1}$. For $0 \leq i,\ j < 2^m - 1$,

$$a_i(X) \neq a_j(X) \tag{19}$$

If $i = 0, 1, 2, \ldots, 2^m-2$, there are $2^m - 1$ different polynomials $a_i(X)$:

$$\begin{aligned} \alpha^i &= q_i(\alpha)p(\alpha) + a_i(\alpha) = a_i(\alpha) \\ \alpha^i &= a_{i0}+a_{i1}X + a_{i2}X^2 + \cdots + a_{i,m-1}X^{m-1} \end{aligned} \tag{20}$$

These polynomials represent $2^m - 1$ non-zero elements $\alpha^0, \alpha^1, \alpha^2, \ldots, \alpha^{2^m-2}$.

There are $2^m - 1$ different polynomials in α over GF(2) which represent the $2^m - 1$ different non-zero elements of the set F. This leads to a binary representation for each element of the set.

The addition operation is defined as

$$0 \oplus 0 = 0$$

$$0 \oplus \alpha^i = \alpha^i \oplus 0 = \alpha^i$$

and

$$\alpha^i \oplus \alpha^j = (a_{i0} \oplus a_{j0}) + (a_{i1} \oplus a_{j1})X + (a_{i2} \oplus a_{j2})X^2 + \cdots + (a_{i,m-1} \oplus a_{j,m-1})X^{m-1} \tag{21}$$

where addition element by element is done modulo 2. This is the same as saying that the addition of any two elements of the set $F = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^{2^m-2}\}$ is the exclusive-OR bitwise operation between the binary representation of those two elements, which are equivalent to the corresponding polynomial expressions in α.

This set F of elements defined as above is commutative with respect to the addition operation, and the set of non-zero elements of F is commutative with respect to the multiplication operation. Therefore the set

$$F = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^{2^m-2}\}$$

is a Galois field or finite field of 2^m elements, GF(2^m).

Example B.1: Let $m = 3$, and $p_i(X) = 1 + X + X^3$ a primitive polynomial over GF(2). Since $p_i(\alpha) = 1 + \alpha + \alpha^3 = 0$, then $\alpha^3 = 1 + \alpha$. The field GF(2^3) can be constructed, making use of the above expression, in order to determine all the non-zero elements of that field. Thus, for example, $\alpha^4 = \alpha\alpha^3 = \alpha(1 + \alpha) = \alpha + \alpha^2$.

Table B.3 shows all the elements of the Galois field GF(2^3) generated by $p_i(X) = 1 + X + X^3$. Examples of the product and sum of two elements in this field are calculated as follows:

$$\alpha^4\alpha^6 = \alpha^{10} = \alpha^{10-7} = \alpha^3$$

$$\alpha^2 + \alpha^4 = \alpha^2 + \alpha + \alpha^2 = \alpha$$

Table B.3 The Galois field GF(2^3) generated by $p_i(X) = 1 + X + X^3$

Exp. representation	Polynomial representation	Vector representation
0	0	0 0 0
1	1	1 0 0
α	α	0 1 0
α^2	α^2	0 0 1
α^3	$1 \quad +\alpha$	1 1 0
α^4	$+\alpha \quad +\alpha^2$	0 1 1
α^5	$1 \quad +\alpha \quad +\alpha^2$	1 1 1
α^6	$1 \quad +\alpha^2$	1 0 1

The most commonly used way of determining the sum of two elements of a Galois field is by doing the bitwise exclusive-OR operation over the binary representations of these two elements.

Example B.2: Determine the table of the elements of the Galois field GF(2^4) generated by the primitive polynomial $p_i(X) = 1 + X + X^4$.

According to the expression for the primitive polynomial, $p_i(\alpha) = 1 + \alpha + \alpha^4 = 0$, or $\alpha^4 = 1 + \alpha$. The generated field GF(2^4) is shown in Table B.4.

B.6 Properties of Extended Galois Fields GF(2^m)

Polynomials defined over the binary field GF(2) can have roots that belong to an extended field GF(2^m). This is the same as what happens in the case of polynomials defined over the

Table B.4 The Galois field GF(2^4) generated by $p_i(X) = 1 + X + X^4$

Exp. representation	Polynomial representation	Vector representation
0	0	0 0 0 0
1	1	1 0 0 0
α	α	0 1 0 0
α^2	α^2	0 0 1 0
α^3	α^3	0 0 0 1
α^4	$1 \quad +\alpha$	1 1 0 0
α^5	$\alpha \quad +\alpha^2$	0 1 1 0
α^6	$+\alpha^2 \quad +\alpha^3$	0 0 1 1
α^7	$1 \quad +\alpha \quad +\alpha^3$	1 1 0 1
α^8	$1 \quad +\alpha^2$	1 0 1 0
α^9	$\alpha \quad +\alpha^3$	0 1 0 1
α^{10}	$1 \quad +\alpha \quad +\alpha^2$	1 1 1 0
α^{11}	$\alpha \quad +\alpha^2 \quad +\alpha^3$	0 1 1 1
α^{12}	$1 \quad +\alpha \quad +\alpha^2 \quad +\alpha^3$	1 1 1 1
α^{13}	$1 \quad +\alpha^2 \quad +\alpha^3$	1 0 1 1
α^{14}	$1 \quad +\alpha^3$	1 0 0 1

set of real numbers, which can have roots outside that set; that is, roots that are complex numbers.

As an example, the polynomial $p_i(X) = 1 + X^3 + X^4$ is irreducible over GF(2) since it has no roots in that field, but it has, however, its four roots in the extended Galois field GF(2^4). By simply replacing the variable X in the expression for the polynomial with the elements as given in Table B.4 of the Galois field GF(2^4), it can be verified that $\alpha^7, \alpha^{11}, \alpha^{13}$ and α^{14} are indeed the roots of that polynomial. As a consequence of this,

$$\begin{aligned}
p_i(X) &= 1 + X^3 + X^4 \\
&= (X + \alpha^7)(X + \alpha^{11})(X + \alpha^{13})(X + \alpha^{14}) \\
&= [X^2 + (\alpha^7 + \alpha^{11})X + \alpha^{18}][X^2 + (\alpha^{13} + \alpha^{14})X + \alpha^{27}] \\
&= [X^2 + (\alpha^8)X + \alpha^3][X^2 + (\alpha^2)X + \alpha^{12}] \\
&= X^4 + (\alpha^8 + \alpha^2)X^3 + (\alpha^{12} + \alpha^{10} + \alpha^3)X^2 + (\alpha^{20} + \alpha^5)X + \alpha^{15} \\
&= X^4 + X^3 + 1
\end{aligned}$$

The following theorem determines a condition to be satisfied by the roots of a polynomial taken from an extended field. This theorem allows determination of all the roots of a given polynomial as a function of one of these roots β.

Theorem B.1: Let $f(X)$ be a polynomial defined over GF(2). If an element β of the extended Galois field GF(2^m) is a root of the polynomial $f(X)$, then for any positive integer $l \geq 0$, β^{2^l} is also a root of that polynomial.

Demonstration of this theorem is based on equation (11), and is done by simply replacing the variable X in the polynomial expression of $f(X)$ with the corresponding root

$$(f(\beta))^{2^l} = (0)^{2^l} = f(\beta^{2^l}) = 0$$

The element β^{2^l} is called the conjugate of β.

This theorem states that if β is an element of the extended field GF(2^m) and also a root of the polynomial $f(X)$, its conjugates are also elements of the same field and roots of the same polynomial.

Example B.3: The polynomial $p_i(X) = 1 + X^3 + X^4$ defined over GF(2) has α^7 as one of its roots. This means that, by applying Theorem B.1, $(\alpha^7)^2 = \alpha^{14}$, $(\alpha^7)^4 = \alpha^{28} = \alpha^{13}$ and $(\alpha^7)^8 = \alpha^{56} = \alpha^{11}$ are also roots of that polynomial. This is the whole set of roots since the next operation $(\alpha^7)^{16} = \alpha^{112} = \alpha^7$ repeats the value of the original root.

In this example it is also verified that the root $\beta = \alpha^7$ satisfies the condition $\beta^{2^m-1} = \beta^{15} = (\alpha^7)^{15} = \alpha^{105} = \alpha^0 = 1$. In general, it is verified that $\beta^{2^m-1} = 1$, because for an element $a \in GF(q)$, it is true that $a^{q-1} = 1$. Equivalently,

$$\beta^{2^m-1} + 1 = 0$$

that is, β is a root of the polynomial $X^{2^m-1} + 1$. In general, every non-zero element of the Galois field GF(2^m) is a root of the polynomial $X^{2^m-1} + 1$. Since the degree of the polynomial

$X^{2^m-1}+1$ is 2^m-1, the 2^m-1 non-zero elements of GF(2^m) are all roots of $X^{2^m-1}+1$. Since the zero element 0 of the field GF(2^m) is the root of the polynomial X, it is possible to say that the elements of the field GF(2^m) are all the roots of the polynomial $X^{2^m}+X$.

B.7 Minimal Polynomials

Since every element β of the Galois field GF(2^m) is a root of the polynomial $X^{2^m}+X$, the same element could be a root of a polynomial defined over GF(2) whose degree is less than 2^m.

Definition B.3: The minimum-degree polynomial $\phi(X)$, defined over GF(2) that has β as its root, is called the minimal polynomial of β. This is the same as to say that $\phi(\beta)=0$.

Thus, the minimal polynomial of the zero element 0 is X, and the minimum polynomial of the element 1 is $1+X$.

B.7.1 Properties of Minimal Polynomials

Minimal polynomials have the following properties [1]:

Theorem B.2: The minimum polynomial of an element β of a Galois field GF(2^m) is an irreducible polynomial.

Demonstration of this property is based on the fact that if the minimal polynomial was not irreducible, it could be expressed as the product of at least two other polynomials $\phi(X)=\phi_1(X)\,\phi_2(X)$, but since $\phi(\beta)=\phi_1(\beta)\,\phi_2(\beta)=0$, it should be true that either $\phi_1(\beta)=0$ or $\phi_2(\beta)=0$, which is contradictory with the fact that $\phi(X)$ is of minimum degree.

Theorem B.3: For a given polynomial $f(X)$ defined over GF(2), and $\phi(X)$ being the minimal polynomial of β, if β is a root of $f(X)$, it follows that $\phi(X)$ is a factor of $f(X)$.

Theorem B.4: The minimal polynomial $\phi(X)$ of the element β of the Galois field GF(2^m) is a factor of $X^{2^m}+X$.

Theorem B.5: Let $f(X)$ be an irreducible polynomial defined over GF(2), and $\phi(X)$ be the minimal polynomial of an element β of the Galois field GF(2^m). If $f(\beta)=0$, then $f(X)=\phi(X)$.

This last theorem means that if an irreducible polynomial has the element β of the Galois field GF(2^m) as its root, then that polynomial is the minimal polynomial $\phi(X)$ of that element.

Theorem B.6: Let $\phi(X)$ be the minimal polynomial of the element β of the Galois field GF(2^m), and let e be the smallest integer number for which $\beta^{2^e}=\beta$, then the minimal polynomial of β is

$$\phi(X)=\prod_{i=0}^{e-1}(X+\beta^{2^l})$$

Table B.5 Minimal polynomials of all the elements of the Galois field GF(2^4) generated by $p_i(X) = 1 + X + X^4$

Conjugate roots	Minimal polynomials
0	X
1	$1 + X$
$\alpha, \alpha^2, \alpha^4, \alpha^8$	$1 + X + X^4$
$\alpha^3, \alpha^6, \alpha^9, \alpha^{12}$	$1 + X + X^2 + X^3 + X^4$
α^5, α^{10}	$1 + X + X^2$
$\alpha^7, \alpha^{11}, \alpha^{13}, \alpha^{14}$	$1 + X^3 + X^4$

Example B.4: Determine the minimal polynomial $\phi(X)$ of $\beta = \alpha^7$ in GF(2^4). As seen in Example B.3, the conjugates $\beta^2 = (\alpha^7)^2 = \alpha^{14}$, $\beta^{2^2} = (\alpha^7)^4 = \alpha^{28} = \alpha^{13}$ and $\beta^{2^3} = (\alpha^7)^8 = \alpha^{56} = \alpha^{11}$ are also roots of the polynomial for which $\beta = \alpha^7$ is a root. Since $\beta^{2^e} = \beta^{16} = (\alpha^7)^{16} = \alpha^{112} = \alpha^7 = \beta$, then $e = 4$ so that

$$\begin{aligned}
\phi(X) &= (X + \alpha^7)(X + \alpha^{11})(X + \alpha^{13})(X + \alpha^{14}) \\
&= [X^2 + (\alpha^7 + \alpha^{11})X + \alpha^{18}][X^2 + (\alpha^{13} + \alpha^{14})X + \alpha^{27}] \\
&= [X^2 + (\alpha^8)X + \alpha^3][X^2 + (\alpha^2)X + \alpha^{12}] \\
&= X^4 + (\alpha^8 + \alpha^2)X^3 + (\alpha^{12} + \alpha^{10} + \alpha^3)X^2 + (\alpha^{20} + \alpha^5)X + \alpha^{15} \\
&= X^4 + X^3 + 1
\end{aligned}$$

The construction of the Galois field GF(2^m) is done by considering that the primitive polynomial $p_i(X)$ of degree m has α as its root, $p_i(\alpha) = 0$. Since all the powers of α generate all the elements of the Galois field GF(2^m), α is said to be a primitive element.

All the conjugates of α are also primitive elements of the Galois field GF(2^m). In general, it can be said that if β is a primitive element of the Galois field GF(2^m), then all its conjugates β^{2^l} are also elements of the Galois field GF(2^m).

Table B.5 shows the minimal polynomials of all the elements of the Galois field GF(2^4) generated by $p_i(X) = 1 + X + X^4$, as seen in Example B.2.

Bibliography

[1] Lin, S. and Costello, D. J., Jr., *Error Control Coding: Fundamentals and Applications*, Prentice Hall, Englewood Cliffs, New Jersey, 1983.

[2] Allenby, R. B. J., *Rings, Fields and Groups: An Introduction to Abstract Algebra*, Edward Arnold, London, 1983.

[3] Hillma, A. P. and Alexanderson, G. L., *A First Undergraduate Course in Abstract Algebra*, 2nd Edition, Wadsworth, Belmont, California, 1978.

[4] McEliece, R. J., *Finite Fields for Computer Scientists and Engineers*, Kluwer, Massachusetts, 1987.

Answers to Problems

Chapter 1

1.1 (a) 1.32, 2.32, 2.32, 3.32, 4.32, 4.32, 2.22
(b) 2.58, 86%
1.2 (a) 1.875 bits/symbol
(b) 17 bits
1.3 0.722, 0.123, 0.189
1.5 0.0703, 0.741
1.6 0.3199
1.7 1, 0.8112, 0.9182
1.8 1, 0.25, 0.75, 1, 0.38, 0.431, 0.531
1.9 0.767, 0.862 when $\alpha = 0.48$
1.11 0.622, 0.781, 79.6%
1.12 (a) 29,902 bps,
(b) 19.21 dB
1.13 150,053 bps

Chapter 2

2.1 See Chapter 2, Section 2.2
2.2 5, 10
2.3 (a) 11 (b) n, 10 (c) 11
2.4 (a) 0.5

(b) $\boldsymbol{G} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \boldsymbol{H}^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$

(c) 3
(d) 1, 2
(e) (110), error in sixth position

Essentials of Error-Control Coding Jorge Castiñeira Moreira and Patrick Guy Farrell

2.5 A possible solution: $G = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix}, \quad H = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}$

2.6 (a)

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

(b) 7

2.7 (b) 0.25, 5 (c) 2.1×10^{-8}

2.8 (a) $\boldsymbol{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$

(c) (0110), error in the eighth position

2.9 (a) 6 (b) 4 (c) 0.6 (e) (0010111111) (f) (0110)
2.10 (a) 0.73, 1.04×10^{-4} (b) 0.722, 4.5×10^{-7}
2.11 (a) Option 3 (b) 1.5 dB
2.12 (a) $(12/11)\,r_\text{b}$, $(15/11)\,r_\text{b}$, $(16/11)\,r_\text{b}$ (b) 7.2 dB, 6.63 dB, 5.85 dB

Chapter 3

3.1 It is for $n = 6$
3.2 (01010101)
3.3 (a) 0.43, 4 (b) (0010111) (c) $r(X) = 1$
3.5 (a) (00010111111111) (b) (00111), yes (c) Yes

3.6 (a)

00	0	0	0	0	0	0
01	1	0	1	1	0	1
10	1	1	0	1	1	0
11	0	1	1	0	1	1

(b) 4, $l = 3$, $t = 1$

3.7 (a) 8, 6 (b) 256 (d) 3 (e) 6
3.8 (a) (000011001101011) (b) A correctable error in the fifth position

Chapter 4

4.2 Examples of two elements: $\alpha^9 \to \alpha + \alpha^3 + \alpha^4 \to 01011, \alpha^{20} \to \alpha^2 + \alpha^3 \to 00110$

4.3

α	X
1	$1 + X$
$\alpha, \alpha^2, \alpha^4, \alpha^8, \alpha^{16}$	$1 + X^2 + X^5$
$\alpha^3, \alpha^6, \alpha^{12}, \alpha^{24}, \alpha^{17}$	$1 + X^2 + X^3 + X^4 + X^5$
$\alpha^5, \alpha^{10}, \alpha^{20}, \alpha^9, \alpha^{18}$	$1 + X + X^2 + X^4 + X^5$
$\alpha^7, \alpha^{14}, \alpha^{28}, \alpha^{25}, \alpha^{19}$	$1 + X + X^2 + X^3 + X^5$
$\alpha^{11}, \alpha^{22}, \alpha^{13}, \alpha^{26}, \alpha^{21}$	$1 + X + X^3 + X^4 + X^5$
$\alpha^{15}, \alpha^{30}, \alpha^{29}, \alpha^{27}, \alpha^{23}$	$1 + X^3 + X^5$

4.4 $g(X) = 1 + X + X^2 + X^3 + X^5 + X^7 + X^8 + X^9 + X^{10} + X^{11} + X^{15}$

4.5 $g(X) = 1 + X^3 + X^5 + X^6 + X^8 + X^9 + X^{10}, 21, d_{\min} = 7$

4.6 (a) 6

4.7 $g(X) = 1 + X^4 + X^6 + X^7 + X^8, d_{\min} = 5$

4.8 (b) The consecutive roots are $1, \alpha, \alpha^2$; 2

4.9 $e(X) = 1 + X^8$

4.10 Errors at positions $j_1 = 11$ and $j_2 = 4$

4.11 (a) $e(X) = X^7 + X^{30}$ (b) It does not detect the error positions

Chapter 5

5.1 $\boldsymbol{G} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}, 2, \alpha$

5.2 (a) $g(X) = X^4 + \alpha^{13}X^3 + \alpha^6X^2 + \alpha^3X + \alpha^{10}$ (b) $e(X) = \alpha X^3 + \alpha^{11}X^7$
(c) $e(X) = \alpha^8X^5$

5.3 (a) $g(X) = X^6 + \alpha^{10}X^5 + \alpha^{14}X^4 + \alpha^4X^3 + \alpha^6X^2 + \alpha^9X + \alpha^6$ (b) $(15, 9)$

5.4 $e(X) = \alpha^7X^3 + \alpha^3X^6 + \alpha^4X^{12}$

5.5 (a) 2 (b) (1111111)

5.6 (a) 25, $\boldsymbol{G} = \begin{bmatrix} 1 & 0 & 4 & 3 \\ 0 & 1 & 2 & 3 \end{bmatrix}$, 3 (b) (1234)

5.7 (a) 0.6, 3 (b) Yes, (c) Fifth position, α

5.8 (a) $g(X) = X^4 + \alpha^{13}X^3 + \alpha^6X^2 + \alpha^3X + \alpha^{10}$,
$c(X) = \alpha^5X^7 + \alpha^7X^5 + \alpha^4X^4 + \alpha^5X^2 + \alpha X + \alpha^9$
(b) $c(X) = \alpha^5X^{11} + \alpha^7X^9 + \alpha^4X^8 + \alpha^5X^6 + \alpha X^5 + \alpha^9X^4$
(c) $e(X) = X^7 + X^9$, the decoder adds two errors,
$e(X) = X^5 + X^2$, successful decoding of the error pattern

5.9 It can correct burst errors of 64 bits

Chapter 6

6.1 (b) 6 (d) No
6.2 (a) 5 (b) Systematic
6.3 (a) $g^{(1)}(D) = 1 + D^2$, $g^{(2)}(D) = D + D^2$ (b) Catastrophic (c) $(10, 01, 11)$
6.4 (a) 2, 4 (b) $(110, 101, 101, 011, 110, 011, 000)$
6.5 (a) 4 (b) $(\ 10, 11, 11)$ (c) $T(X) = X^4 + 2X^5 + 2X^6 + \ldots$ (d) 16×10^{-6}
6.6 (a) $\boldsymbol{m} = (101000\ldots)$
6.7 (a) $(11, 10, 01, 00, 11)$ (b) $(00, 20, 20, 00, 00\ldots)$
6.8 See Section 6.6
6.9 (a) 0.5, 5 (b) Non-systematic
6.10 $\boldsymbol{m} = (1110010\ldots)$

Chapter 7

7.1 (a) $\boldsymbol{H}^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$, $d_{\min,\mathrm{CC}} = 4$

(b) $d_{\min,\mathrm{BC}} = 3$, $d_{\min,\mathrm{conc}} = 5$

(c) (11010000) decoded by the convolutional code, (110100) passed to the block code, (100)decoded message vector

7.2 (a) 1/7, $d_{\min,\mathrm{conc}} = 12$ (b) $d_{\min,\mathrm{conc}} = 12 = 3 \times 4 = d_{\min,(3,1)} \times d_{\min,(7,3)}$ (c) Same as (b)
7.3 (a) 0.5, 3 (b) (0000)
7.4 3, 5
7.5 (a)

$$\boldsymbol{m} = \left(\begin{array}{l} -1-1, -1-1, -1-1, +1-1, -1+1, +1-1, -1+1, -1-1, +1-1, \\ +1-1, -1-1, +1+1, +1-1, -1-1, +1-1, -1-1 \end{array} \right)$$

(b) Message successfully decoded in three iterations

Chapter 8

8.1 (a) There are at least seven cycles of length 4; the‘1’s involved in these cycles are seen in matrix $\boldsymbol{H}$ below:

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

(b) In order to maintain $s = 3$, '1's should be moved along the columns by replacement of '0's by '1's and vice versa, in the same column. Every position of a '0' in the above matrix cannot be filled by replacement with a '1' unless another cycle is formed.

(c) In general, it will correspond to another code.

8.2 (a) 4, $\begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$, $\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$

(b) 3, 12/7, 13/4, 13/7, 0.429, 4 (c) 6, 4, that of cycle length 6

8.3 (a) (1011100) (b) cyclic graph, decoded vectors in successive iterations: (1011000) , (1011001) , (1011101) , 1011100, successful decoding.

Systematic graph, decoded vectors in successive iterations: (1011000) , (1011011) , (1011011) , 1011000, 10110101, 1011101, unsuccessful decoding in six iterations.

8.4 (a) The decoder fluctuates between the two code vectors $\boldsymbol{c} = (00000)$ and $\boldsymbol{c} = (10100)$, which are those closest to the received vector $\boldsymbol{r} = (10200)$.

(b) Connections between symbol nodes and parity check nodes are not enough for effective belief propagation.

Index

Essentials of Error-Control Coding Jorge Castiñeira Moreira and Patrick Guy Farrell