Improved Error Control Techniques for Data Transmission

Steven Robert Marple

Submitted for the degree of

Doctor of Philosophy.

University of Lancaster.

March 2000.

To my parents, Rob and Gill Marple.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to take this opportunity to thank my supervisor, Professor B. Honary, for his guidance, support and encouragement. Without this help I could not have completed this work.

I would also like to thank my colleagues, both past and present, in the Communications Research Centre, Department of Communications, Lancaster University.

Finally, I would like to thank my parents and Judith for their support.

Declaration

This Thesis and the work described in it are my own, except where stated otherwise. The work was carried out at the University of Lancaster between January 1994 and December 1999. No part of this Thesis has been submitted for the award of a higher degree, either at the University of Lancaster or elsewhere. Some parts of this Thesis have appeared in publications, which are listed below and in the References.

B. Honary, G. Markarian, and S. Marple. Trellis decoding of block codes: Practical approach. In *ICCT '96*, volume 1, pages 525–527, Beijing, China, May 1996.

Relevant Thesis sections: 4.1.2, 4.1.4, 4.2.5, 4.3, 6.1.2, 6.2, and 6.4.

B. Honary, G. Markarian, and S. R. Marple. Trellis decoding of the Reed-Solomon codes: Practical approach. In *3rd International Symposium on Communication Theory and Applications*, pages 12–16, Ambleside, UK, July 1995.

Relevant Thesis sections: 4.1.2, 4.1.4, 4.2.5, 4.3, and 6.4.

B. Honary, G. Markarian, and S. R. Marple. Two-stage trellis decoding of RS codes based on the Shannon product. In *Proceedings of the 1996 International Symposium on Information Theory and Its Applications*, pages 282–285, Victoria, BC, Canada, September 1996.

Relevant Thesis sections: 4.1.2, 4.1.4, 4.2.5, 4.3, 6.1.2, 6.2 and 6.4.

B. Honary, G. Markarian, and S. R. Marple. Trellis decoding of the Reed-Solomon codes: A practical approach. In B. Honary, M. Darnell, and P. G. Farrell, editors, *Communications Coding and Signal Processing, Communications Systems Techniques and Applications Series*, pages 133–147. John Wiley & Sons, New York, London, Sydney, 1997. ISBN 0 86380 221 4.

Relevant Thesis sections: 4.1.2, 4.1.4, 4.2.5, 4.3, 6.1.2, 6.2, and 6.4.

B. Honary and S. R. Marple. Investigation of sequence segment keying (SSK) and its application in CDMA systems. In *ITEC 96*, Leeds, UK, April 1996.

Relevant Thesis sections: 4.2.5, 4.3 and 6.4.

Symbols and Abbreviations

ARC	alternating row-column [decoding]
AWGN	additive white Gaussian noise
$\mathbb{B}$	branch profile
BCH	Bose-Chaudhuri-Hocquenghem
BCH(n, k, d)	BCH code with given parameters
BCJR	Bahl-Cocke-Jelinek-Raviv [trellis]
BER	bit error rate
BM	Berlekamp-Massey [decoding]
BPSK	binary phase shift keying
BSC	binary symmetric channel
$\mathcal{C}$	code
CCSDS	Consultative Committee on Space Data Systems
CPU	central processing unit
d	minimum distance
DMC	discrete memoryless channel
$\mathbf{e}$	error vector

e	error symbol
$e(x)$	error polynomial
E_b	bit energy
ESA	European Space Agency
FSK	frequency shift keying
FSR	feedback shift-register
$\mathbf{G}$	generator matrix
$g(x)$	generator polynomial
GAC	generalised array code
GF	Galois field
$\mathbf{H}$	parity-check matrix
HD	hard decision
HDMLD	hard decision maximum-likelihood decoding
HSSBS	high-speed step-by-step [decoding]
K	constraint length of a convolutional code
k	number of information symbols in block code
$\mathbb{L}$	label profile
LL	log likelihood
MAP	maximum *a posteriori*
MDS	maximum distance separable
ML	maximum-likelihood
$\mathbb{N}$	state profile
n	number of code symbols in block code

N_0	noise spectral density
NASA	National Aeronautics and Space Administration
P_b	probability of bit error
PDF	probability density function
P_M	probability of block error
P_s	probability of symbol error
q	alphabet size used by a code
QPSK	quaternary phase-shift keying
$\mathbf{r}$	received codeword vector
r	received codeword symbol
r	degree of generator polynomial
$r(x)$	received codeword polynomial
RM	Reed-Muller
RM(n, k, d)	Reed-Muller code with given parameters
RS	Reed-Solomon
RS(n, k, d)	Reed-Solomon code with given parameters
$\mathbf{S}$	syndrome vector
$S(x)$	syndrome polynomial
SD	soft decision
SDMLD	soft decision maximum-likelihood decoding
SNR	signal to noise ratio
SOVA	soft-output Viterbi algorithm
T	trellis

t	maximum number of correctable errors
TSD	two-stage decoding
$\mathbf{u}$	dataword vector
u	dataword symbol
$u(x)$	dataword polynomial
$\mathbf{v}$	codeword vector
v	codeword symbol
$v(x)$	codeword polynomial
VA	Viterbi algorithm
α	primitive element of Galois field
Δ	discrepancy
κ	complexity
$\Lambda(x)$	error-locator polynomial
$\lambda(x)$	correction polynomial for $\Lambda(x)$
ν	number of correctable errors
$\Omega(x)$	error-evaluator polynomial
$\omega(x)$	correction polynomial for $\Omega(x)$

Abstract

Error control coding is frequently used to minimise the errors which occur naturally in the transmission and storage of digital data. Many methods for decoding such codes already exist. The choice falls mainly into two areas: hard-decision algebraic decoding, a computationally-efficient method, and soft-decision combinatorial decoding, which although more complex offers better error-correction.

The work presented in this Thesis is intended to provide practical decoding algorithms which can be implemented in real systems.

Soft-decision maximum-likelihood decoding of Reed-Solomon codes can be obtained by using the Viterbi algorithm over a suitable trellis. Two-stage decoding of Reed-Solomon codes is presented. It is an algorithm by which near-optimum performance may be achieved with a complexity lower than the Viterbi algorithm.

The soft-output Viterbi algorithm (SOVA) has been investigated as a means of providing soft-decision information for subsequent decoders. Considerations of how to apply SOVA to multi-level codes are given. The use of SOVA in a satellite downlink channel is discussed. The results of a computer simulation, which showed a 1.8 dB improvement in coding gain for only a 20% increase in decoding complexity, are presented.

SOVA was also used to improve the decoding performance when applied to an RS product code. Several different decoding methods were evaluated, including cascade decoding, and a method where the row and columns were decoded alternately.

A complexity measurement was developed which allows accurate comparisons of decoding complexity for trellis-based and algebraic decoders. With this technique the decoding complexity of all the algorithms implemented are compared. Also included in the comparison are the Euclidean and Berlekamp-Massey algorithms.

Chapter 1

Introduction

Chapter 1

Introduction

The twentieth century has seen an explosion in the use and availability of communication systems. They are now placed in and on many devices which were not even invented one hundred years ago. Such widespread use has placed high demands on engineers. Mobile telephones are expected to operate for long periods and with clear reception. Digital television and weather images from space are expected to be clear of speckles. Music from compact discs must be free of clicks and pops which frequently troubled the vinyl records which they are now quickly replacing. As the storage size of computer memories and disks increase the access times plummet. As these technologies are reliant upon computer hardware, which is still following Moore's 'law',[1] the rapid increase in technology looks set to continue. The uniting factor in all of these diverse applications is that they use error control coding to protect valuable digital data and enhance the service they provide.

[1]Moore, founder of Intel, suggested that the number of transistors on integrated circuits for computers doubles approximately every 18–24 months [Moore, 1965].

In its early days error control coding could only be afforded by the 'super-rich'—the military and organisations such as NASA. Even so, the codes used then (often Reed-Muller codes) are considered by today's standards as weak and simple to decode. Today, error control coding is widespread and cheap. Probably most popular are the Reed-Solomon codes. They are, however, a two-edged sword; providing greater error protection but are also many orders of magnitude more difficult to decode. Although efficient hard-decision RS decoders exist the holy grail is an efficient soft-decision algorithm, which will provide optimum performance with simplicity.

The twenty-first century will surely see an increase in the use of error control coding as current technologies are miniaturised further, and new ones invented. Thus the demand for fast and efficient decoding algorithms will only increase. This Thesis presents new work which is aimed at both improving upon hard-decision decoding performance while reducing complexity from the soft-decision case.

Chapter two introduces the background topics used in this work. Included are the theory and important properties of linear and cyclic block codes. Attributes of convolutional codes are discussed. The concept of concatenated codes and important definitions regarding trellis diagrams and trellis decoding are introduced. The channel models used in the computer simulations are also described.

Algebraic decoding of RS codes is examined in Chapter three. The Berlekamp-Massey, Euclidean and high-speed step-by-step algorithms are explained, both mathematically and with the aid of examples. Chapter four details trellis construction techniques, for both syndrome (BCJR/Wolf) and coset trellises. The Viterbi algorithm is described and an example decoding used to illustrate the procedure. A novel low-complexity, near-optimum decoding algorithm, two-stage decoding, is presented and a worked example given.

In Chapter five concatenated decoding is used as a means of combining good error control performance with low complexity. The soft-output Viterbi algorithm is demonstrated as a method by which the outer decoder is able to perform better by taking advantage of soft-decision information. The Viterbi decoding example shown in Chapter four is extended to give a clear demonstration of how SOVA operates. The application of SOVA over non-binary trellises is discussed. Product codes may be thought of as a form of concatenated coding. Various algorithms for decoding product codes are described, and the application of SOVA to each is considered.

Chapter six presents results on both decoding complexity and performance. As this Thesis includes both algebraic and combinatorial (trellis) decoders a unified practical method, by which all the decoders implemented may be compared, was sought. How this was achieved is explained. Results of all the simulated systems are given, including a weather satellite image distribution system. Trellis decoding complexity for the Viterbi algorithm was analysed in a mathematical manner, applicable to all linear codes (and also separable non-linear codes). Also, the analysis is expanded for the soft-output Viterbi algorithm. Following this, decoding complexity for all the simulated codes is given, using the same set of benchmarks. Decoding performance is not forgotten, and Chapter six also includes decoding performance curves for all the decoding algorithms demonstrated. Where possible the same channel model was used.

Concluding remarks are made in Chapter seven. The unified approach to decoding complexity allows comparisons to be made regarding the complexity of the various systems. Where appropriate, comparisons of the decoding performance are also made. The improved performance of the weather satellite image distribution system is discussed, and commercial benefits of increased coding gain are highlighted. Finally, the References, and for the benefit of the reader, a citation index and general index are located at the back of this work.

Chapter 2

Background

Chapter 2

Background

2.1 Block Codes

2.1.1 Linear Block Codes

In a block code the message symbols are sectioned into blocks of fixed length, k, before being passed to the encoder. The input block or *dataword* contains k data symbols over an alphabet of size q. The encoder output is a *codeword* containing n code symbols, also over an alphabet of size q. For the case $q = 2$ the code is described as *binary*. Block codes may be divided into two categories, *linear* block codes and *non-linear* block codes. Only linear block codes are considered.

For a useful code the datawords $\mathbf{u}$ must form a one-to-one mapping with the set of q^k possible input sequences. For an (n, k) linear code $\mathcal{C}$ the codewords $\mathbf{v}$ must form a k-dimensional subspace of the n-dimensional *codespace* over the field GF(q) [Lin and Costello, 1983, p. 52], i.e., the *dimension* of the code is k. Since the codewords are restricted to a k-dimensional subspace of the codespace the linear sum of any two codewords is also restricted to the k-dimensional subspace and must therefore also be a codeword.

The Generator Matrix

For a code $\mathcal{C}$ there exists k linearly independent codewords $\mathbf{g}_0,\ \mathbf{g}_1,\ \ldots,\ \mathbf{g}_{k-1}$ so that every codeword $\mathbf{v}$ in $\mathcal{C}$ is a linear combination of these codewords, i.e.,

$$\mathbf{v} = u_0\mathbf{g}_0 + u_1\mathbf{g}_1 + \ldots + u_{k-1}\mathbf{g}_{k-1} \tag{2.1}$$

where $(u_0,\ u_1,\ \ldots,\ u_{k-1})$ are symbols in the input sequence $\mathbf{u}$ represented by elements in GF(q). The k linearly independent codewords can be arranged as rows in a $k \times n$ matrix:

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_{k-1} \end{bmatrix} = \begin{bmatrix} g_{0,0} & g_{0,1} & \cdots & g_{0,n-1} \\ g_{1,0} & g_{1,1} & \cdots & g_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k-1,0} & g_{k-1,1} & \cdots & g_{k-1,n-1} \end{bmatrix} \tag{2.2}$$

where $\mathbf{g}_i = (g_{i,0},\ g_{i,1},\ \ldots,\ g_{i,n-1})$ and $k = 0,\ 1,\ \ldots,\ k$. A dataword $\mathbf{u}$ can be mapped to its corresponding codeword $\mathbf{v}$ by [Lin and Costello, 1983, p. 53]

$$
\begin{aligned}
\mathbf{v} &= \mathbf{u} \cdot \mathbf{G} \\
&= (u_o,\ u_1,\ \ldots,\ u_{k-1}) \cdot \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_{k-1} \end{bmatrix} \\
&= u_0\mathbf{g}_0 + u_1\mathbf{g}_1 + \ldots + u_{k-1}\mathbf{g}_{k-1}
\end{aligned} \tag{2.3}
$$

From (2.3) it can be seen that the matrix $\mathbf{G}$ generates codewords of $\mathcal{C}$ given a dataword, and is known as the *generator matrix*. If the encoding procedure of a linear block code preserves the input sequence $\mathbf{u}$ within the output sequence of $\mathbf{v}$ the code is *systematic*. Systematic codes enable simplifications to the decoding algorithm, and are especially important for array codes. This useful property can be identified in the generator matrix, if k consecutive columns of $\mathbf{G}$ form the $k \times k$ identity matrix the code is systematic. By column reordering the generator matrix for systematic codes can always be arranged to *reduced-echelon* form

$$\mathbf{G} = \left[\begin{array}{ccccc:c} 1 & 0 & 0 & \dots & 0 & \\ 0 & 1 & 0 & \dots & 0 & \\ 0 & 0 & 1 & & 0 & \mathbf{P} \\ \vdots & \vdots & & \ddots & & \\ 0 & 0 & 0 & & 1 & \end{array}\right] \tag{2.4}$$

where $\mathbf{P}$ represents the parity checks. An important point to note is that every valid codeword is a multiple of the generator matrix. The importance of this will become apparent when the decoding of a received codeword which contains errors is considered.

The Parity-check Matrix

It is useful to be able to express the code in a manner which highlights the parity checks. For a $k \times n$ generator matrix $\mathbf{G}$ there is an $(n-k) \times n$ *parity-check matrix*, $\mathbf{H}$. The relationship of the parity-check matrix to the generator matrix, $\mathbf{G}$, is given by

$$\mathbf{G}\mathbf{H}^T = \mathbf{0} \tag{2.5}$$

where $\mathbf{H}^T$ is the transpose of $\mathbf{H}$ and $\mathbf{0}$ is an all-zeros matrix. For any codeword, $\mathbf{v}$ in the code the parity checks sum to zero and thus

$$\mathbf{v}\mathbf{H}^T = \mathbf{0} \tag{2.6}$$

2.1.2 Syndrome Vector

Consider the case of the parity checks when the received codeword, **r**, is in error. The parity check values, or *syndromes*, are non-zero. The *syndrome vector*, **S**, is defined by

$$\begin{gathered}\mathbf{S} = \mathbf{r}\mathbf{H}^T \\ \text{where } \mathbf{S} = [S_1,\ S_2,\ \ldots\ ,\ S_{2t}]\end{gathered} \tag{2.7}$$

For the case when the received codeword is correct (i.e., $\mathbf{r} \equiv \mathbf{v}$) Equation 2.7 reduces to Equation 2.6. However, when the received codeword is in error **r** is given by $\mathbf{r} = \mathbf{v} + \mathbf{e}$, where **e** is the error vector. Substituting into Equation 2.7 gives

$$\begin{aligned}\mathbf{S} &= \mathbf{r}\mathbf{H}^T \\ &= (\mathbf{v} + \mathbf{e})\mathbf{H}^T \\ &= \mathbf{v}\mathbf{H}^T + \mathbf{e}\mathbf{H}^T \\ &= \mathbf{e}\mathbf{H}^T\end{aligned} \tag{2.8}$$

since $\mathbf{v}\mathbf{H}^T = \mathbf{0}$ (Equation 2.6). From Equation (2.8) it is clear that the syndrome is dependent only upon the error pattern, **e**, and not the transmitted codeword, **v**.

There is a one-to-one mapping from correctable error patterns to the syndromes. For simple codes error correction can be achieved by a table lookup of the syndromes and a GF(q) subtraction of the corresponding error value. This method is not practical for useful codes as the table size is too large to store (e.g., for RS$(255, 223, 33)$ the table would contain $256^{32} - 1 = 1.16 \times 10^{77}$ entries!).

2.1.3 The Singleton Bound

The Singleton bound [Singleton, 1964] provides an upper limit on the minimum distance, $d_{\min}$, between codewords in a linear block code. It thus provides an important goal good codes should attain. Codes which satisfy the Singleton bound with equality are termed *maximum distance separable*. The bound is given by

$$d_{\min} \leq n - k + 1 \tag{2.9}$$

2.1.4 Array Codes

Array codes were introduced by Elias [Elias, 1954]. They are constructed from linear component codes in two or more dimensions. The simplest array code is a two-dimensional code with an (n_1, k_1, d_1) vertical code, $\mathcal{C}_1$, and an (n_2, k_2, d_2) horizontal code, $\mathcal{C}_2$. The resulting code, $\mathcal{C}$, is an $(n_1 n_2, k_1 k_2, d_1 d_2)$ code (Figure 2.1). The term *product code* is sometimes applied to *array codes*; the two terms are interchangeable.

General Properties of Array Codes

Theorem 2.1 The minimum weight for the product of two codes is the product of the minimum weights of the codes.

Proof 2.1 [Elias, 1954]. The minimum weights of the component codes, $\mathcal{C}_1$ and $\mathcal{C}_2$ are d_1 and d_2. Each column containing a non-zero element must have at least d_1 non-zero

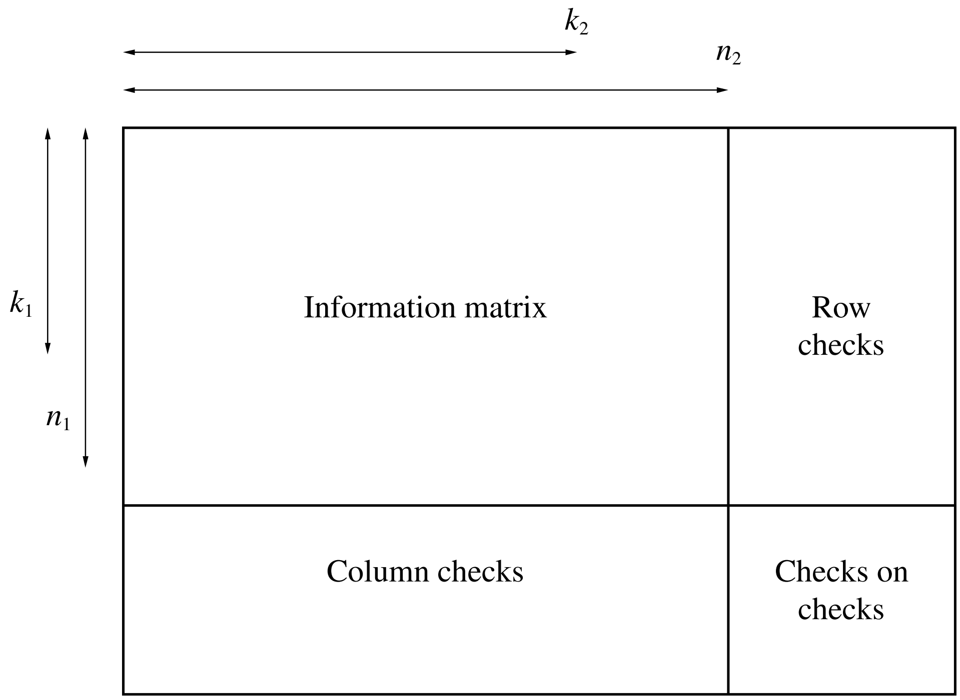


Figure 2.1: Array code.

elements, and each row containing a non-zero element must have at least d_2 non-zero elements. Therefore if the product code $\mathcal{C}$ contains any non-zero elements it must contain at least d_1 non-zero rows and d_2 non-zero columns. The minimum number of non-zero elements is therefore $d_1 d_2$ and thus the minimum weight of $\mathcal{C}$ is $d_1 d_2$. ■

Encoding and decoding are greatly simplified when the component codes are systematic. For a two-dimensional product code with systematic component codes the generator matrix can be shown to be the Kronecker product (denoted by $\otimes$) of the generator matrices [Slepian, 1960]. For the case when the component codes are single parity-check codes (i.e., $k_1 = n_1 - 1$, $k_2 = n_2 - 1$) the generator matrix reduces to the simpler case as the Tensor product of the component codes [Wolf, 1965]. Higher dimensions are possible. As both row and column codes are linear it is not important whether the row or column encoding is performed first, the checks-on-checks will be identical in either case [Peterson and Weldon, 1972, p. 132]. Similarly, the decoding order is not important.

If serial transmission of the codeword symbols is assumed (which is the normal case) the two-dimensional codeword, $\mathbf{v}$

$$\mathbf{v} = \begin{bmatrix} v_{0,0} & v_{0,1} & \cdots & v_{0,n_2-1} \\ v_{1,0} & v_{1,1} & \cdots & v_{1,n_2-1} \\ \vdots & \vdots & \ddots & \vdots \\ v_{n_1-1,0} & v_{n_1-1,1} & \cdots & v_{n_1-1,n_2-1} \end{bmatrix} \tag{2.10}$$

must be converted to a one-dimensional vector before transmission over the channel. The mapping of $v_{i,j}$ where $0 \leq i < n_1$ and $0 \leq j < k_2$ to v_l where $0 \leq l < n_1 n_2 - 1$ may be accomplished by a number of methods. The *canonical ordering* is achieved by choosing i and j as the quotient and remainder when l is divided by n_2 [Berlekamp, 1968, p. 338]. Canonical ordering gives the array

$$\mathbf{v} = \begin{bmatrix} v_0 & v_1 & \cdots & v_{n_2-1} \\ v_{n_2} & v_{n_2+1} & \cdots & v_{2n_2-1} \\ \vdots & \vdots & \ddots & \vdots \\ v_{(n_1-1)n_2} & v_{(n_1-1)n_2+1} & \cdots & v_{n-1} \end{bmatrix} \tag{2.11}$$

For the case when n_1 and n_2 are relatively prime there is (by the Chinese remainder theorem) a unique integer l in the range $0 \leq l < n_1 n_2 - 1$ for the pair (i, j) such that

$l \equiv i \bmod n_2$ and $l \equiv j \bmod n_1$ [Burton and Weldon, 1965]. This mapping of i and j to l is known as *cyclic ordering*. The reasoning behind the name is clearly seen from the example below where $n_1 = 3$ and $n_2 = 5$

$$\mathbf{v} = \begin{bmatrix} v_0 & v_6 & v_{12} & v_3 & v_9 \\ v_{10} & v_1 & v_7 & v_{13} & v_4 \\ v_5 & v_{11} & v_2 & v_8 & v_{14} \end{bmatrix} \tag{2.12}$$

If $\mathcal{C}_1$ and $\mathcal{C}_2$ are cyclic codes and n_1 and n_2 are relatively prime then the product $\mathcal{C} = \mathcal{C}_1 \otimes \mathcal{C}_2$ is also cyclic, however not all cyclic codes are array codes [MacWilliams and Sloane, 1978, p. 570].

Decoding Array Codes

Many decoding algorithms for array codes exist, both algebraic and trellis-oriented. The simplest method for decoding a canonically-ordered array code is to *cascade decode* the component codes one at a time, decoding the row code, $\mathcal{C}_2$, and then the column code $\mathcal{C}_1$. Particularly for memoryless channels, the cascade decoding algorithm is inefficient. There exist error patterns for which the code is capable of correcting but the algorithm fails to correct. If the minimum distances, d_1 and d_2, of the component codes are odd then there are error patterns of weight $(d_1 + 1)(d_2 + 1)/4$ which are incorrectly decoded, although the code is capable of correcting errors up to weight $(d_1 d_2 - 1)/2$ [Berlekamp, 1968, p. 340].

2.1.5 Generalised Array Codes

As the name suggests GACs are a generalisation of the array codes introduced by Elias. Unlike standard array codes GACs may have different component codes along a given dimension, with the restriction of the code length being invariant. The total number of code symbols is given by $n = n_1 n_2$, where n_1 and n_2 are the number of rows and columns, respectively. The total number of information symbols is given by $k = \sum_{i=1}^{n_1} k_p$, where k_p is the number of information symbols in p-th row [Honary and Markarian, 1997, p. 11].

The technique provides a simple design procedure for constructing many different block codes, BCH, Hamming, Golay, RM etc. [Honary and Markarian, 1993a,b; Honary *et al.*, 1995a]. It is important as it allows minimal trellises to be designed in a straightforward manner.

2.1.6 Cyclic codes

Cyclic codes are an important subclass of linear block codes, not only because many prominent codes are cyclic e.g., BCH, RS, but also because they are used in the construction of other error-correcting codes e.g., Kerdock and Preparata codes. The inherent algebraic structure of cyclic codes allows the formation of many practical decoding methods; Euclidean (Section 3.3), Berlekamp-Massey (Section 3.4), step-by-step (Section 3.5) and others.

General Properties of Cyclic Codes

A code $\mathcal{C}$ is *cyclic* if it is linear and every cyclic shift of every codeword $\mathbf{v}$ is also a codeword. i.e., if $(v_0,\ v_1,\ \ldots,\ v_{n-1})$ is a codeword in $\mathcal{C}$ then $(v_{n-1},\ v_0,\ \ldots,\ v_{n-2})$ is also a codeword in $\mathcal{C}$ [MacWilliams and Sloane, 1978, p. 188]. Cyclic codes are commonly defined in terms of polynomials, where the coefficients of the polynomial are the symbols in $\mathbf{v}$. The notation $v(x)$ will be used to denote a code polynomial. The relationship between a codeword $\mathbf{v}$ and the polynomial is

$$\mathbf{v} = (v_0,\ v_1,\ \ldots,\ v_{n-1}) \tag{2.13}$$

$$v(x) = v_0 + v_1 x + v_2 x^2 + \ldots + v_{n-1} x^{n-1} \tag{2.14}$$

It can be seen that the maximum degree of $v(x)$ is $n-1$. Algebraically, $v(x)$ is defined to be a polynomial modulo $X^n - 1$. This leads to the important identity

$$x^n \equiv 1 \tag{2.15}$$

From (2.15) it can be shown that a multiplication of $v(x)$ by x is a cyclic shift

$$\begin{aligned} x.v(x) = \quad & v_0 x + v_1 x^2 + v_2 x^3 + \ldots + v_{n-2} x^{n-1} + v_{n-1} x^n \\ = \ & v_{n-1} + v_0 x + v_1 x^2 + v_2 x^3 + \ldots + v_{n-2} x^{n-1} \end{aligned} \tag{2.16}$$

It can be shown [Wicker, 1994, p. 101; Wilson, 1996, pp. 443–444] that there exists a unique polynomial, $g(x)$, such that every code polynomial can be expressed as a product of the generator polynomial

$$v(x) = u(x)\,g(x) \tag{2.17}$$

where $u(x)$ is a polynomial of degree $k-1$ or less and is known as the message polynomial. Equation (2.17) indicates a method by which a message polynomial, $u(x)$, can be encoded to its corresponding codeword polynomial. An alternative approach is based on matrices. It is shown in equation (2.2) that a generator matrix can be constructed from k linearly independent codewords. Since the generator polynomial is itself a codeword polynomial (corresponding to the case $u(x) = 1$) the k codewords can be arranged as k cyclic shifts of $g(x)$.

$$\mathbf{G} = \begin{bmatrix} g(x) \\ xg(x) \\ x^2g(x) \\ \vdots \\ x^{k-1}g(x) \end{bmatrix} = \begin{bmatrix} g_0 & g_1 & \ldots & g_r & 0 & \ldots & 0 & 0 \\ 0 & g_0 & g_1 & \ldots & g_r & 0 & \ldots & 0 \\ & & \ddots & \ddots & & \ddots & & \\ 0 & \ldots & 0 & g_0 & g_1 & \ldots & g_r & 0 \\ 0 & 0 & \ldots & 0 & g_0 & g_1 & \ldots & g_r \end{bmatrix} \tag{2.18}$$

The encoding procedure given in (2.17) does not produce a systematic codeword. Normally, systematic codewords are preferred as they simplify the decoding procedure. The normal method by which systematic codewords are generated is [Michelson and Levesque, 1985, p. 133]

$$v(x) = \left[x^{n-k}u(x) \bmod g(x)\right] + x^{n-k}u(x) \tag{2.19}$$

It can be seen that this does result in a valid codeword, the remainder when $v(x)$ is divided by $g(x)$ is zero, and the maximum degree of $v(x)$ is $n - 1$.

Syndrome Polynomial

For cyclic codes (e.g., BCH and RS) the calculation of the syndromes can be performed more efficiently by using the cyclic properties of the code. Though the syndrome vector can be calculated by $\mathbf{S} = \mathbf{rH}^T$ (2.7) this calculation can also be performed as

$$\begin{aligned} S(x) &= \frac{r(x)}{g(x)} \\ &= r(x) \bmod g(x) \end{aligned} \tag{2.20}$$

$$\text{where } S(x) = S_1 + S_2x + S_3x^2 + \ldots + S_{2t}x^{2t-1} \tag{2.21}$$

where $S(x)$ is the *syndrome polynomial*. Thus the syndrome polynomial may be defined as the remainder when an erroneous codeword is divided by the generator polynomial, $g(x)$. The proof is given in [Peterson and Weldon, 1972, p. 231]. For the case of the received codeword being equal to the transmitted codeword, $v(x)$, the syndrome polynomial is zero since valid codewords are exactly divisible by the generator polynomial (2.17).

It can be shown that the syndrome polynomial is dependent upon the error polynomial, $e(x)$, and not the transmitted codeword, $v(x)$, by substituting $r(x) = v(x) + e(x)$

into (2.20)

$$
\begin{aligned}
S(x) &= r(x) \bmod g(x) \\
&= [v(x) + e(x)] \bmod g(x) \\
&= v(x) \bmod g(x) + e(x) \bmod g(x) \\
&= e(x) \bmod g(x)
\end{aligned}
\tag{2.22}
$$

since $v(x) \bmod g(x) = 0$ (from (2.17)).

2.1.7 Bose-Chaudhuri-Hocquenghem Codes

Bose-Chaudhuri-Hocquenghem codes were discovered independently by Hocquenghem [Hocquenghem, 1959] in 1959 and Bose and Ray-Chaudhuri [Bose and Ray-Chaudhuri, 1960a,b] in 1960. BCH codes are a generalisation of the cyclic Hamming codes for correcting multiple errors. Peterson [Peterson, 1960] proved that BCH codes are cyclic. Gorenstein and Zierler [Gorenstein and Zierler, 1961] generalised the BCH codes for non-binary alphabets of size p^m. Their wide choice of block lengths, code rates and symbol alphabets, coupled with efficient decoding algorithms, has made BCH codes a popular choice for many communication systems.

General Properties of BCH Codes

When constructing an arbitrary cyclic code there is no guarantee of the minimum distance of the code produced [Wicker, 1994, p. 176]. It is necessary to conduct a computer search of all the non-zero codewords to find the minimum-weight codeword and thus the minimum distance of the code. For BCH codes this procedure is not required, the BCH bound places lower limit on the minimum distance of the code. An understanding of these codes requires a knowledge of finite field arithmetic [McEliece, 1987].

Theorem 2.2 The BCH Bound [Wicker, 1994]

Let $\mathcal{C}$ be a q-ary (n, k) cyclic code with generator polynomial $g(x)$. Let m be the multiplicative order of q mod n. (GF(q^m) is thus the smallest extension field of GF(q) which contains a primitive n-th root of unity.) Let α be a primitive n-th root of unity.

Select $g(x)$ to be a minimal-degree polynomial in GF(q)[x], where GF(q)[x] denotes the collection of all polynomials $a_0 + a_1x + a_2x^2 + \cdots + x^n$ of arbitrary degree with coefficients $\{a_i\}$ in the finite field GF(q) [Wicker, 1994, p. 40], such that

$$g(\alpha^b) = g(\alpha^{b+1}) = g(\alpha^{b+2}) = \cdots = g(\alpha^{b+\delta-2}) = 0 \tag{2.23}$$

for some integers $b \geq 0$ and $\delta \geq 1$. The roots of the generator polynomial $g(x)$ are $\delta - 1$ consecutive powers of α. The code $\mathcal{C}$ defined by $g(x)$ has minimum distance $d \geq \delta$.

Proof of Theorem 2.2 can be found in [MacWilliams and Sloane, 1978; Peterson and Weldon, 1972; Wicker, 1994]. The BCH bound can be used to produce a BCH code with a given design distance. However, since the weight distributions of most BCH codes are not known the actual minimum distance may be greater than the design distance.

Generator Polynomial

The generator polynomial for a BCH code has $2t$ roots, which are consecutive powers of α (from Theorem 2.2). Therefore

$$g(x) = \prod_{i=0}^{2t-1}(x - \alpha^{b+i}) \tag{2.24}$$

For the case when $b = 1$ the code is termed *narrow sense* [MacWilliams and Sloane, 1978, p. 203].

2.1.8 Reed-Solomon Codes

Reed-Solomon codes were discovered by Reed and Solomon in 1960 and are a special subclass of non-binary BCH codes [Reed and Solomon, 1960]. RS codes exhibit additional properties to BCH codes which make them very much more powerful than BCH codes. Their powerful error-correcting abilities have made them possibly the most important codes. RS codes are multi-level, therefore $\log_2 q$ binary bits are commonly mapped to one RS symbol. This process provides some burst-error correction. They have many and widespread applications which include the compact disc, satellite communications and digital video broadcasting.

General Properties of RS Codes

Reed-Solomon codes are cyclic and so profit from the many useful characteristics cyclic codes offer. They are normally generated in systematic form using (2.19).

Theorem 2.3 An (n, k) RS code has minimum distance $(n - k + 1)$.

Proof 2.2 Let $\mathcal{C}$ be an (n, k) RS code. The Singleton bound (Section 2.1.3) gives an upper bound of $d \leq n - k + 1$ to all (n, k) codes. The BCH bound provides a lower bound. The generator polynomial $g(x)$ is of degree $n - k$, so it contains $n - k = \delta - 1$ consecutive powers of a primitive n-th root of unity. Therefore $d \geq n - k + 1$. Combining these two results gives

$$d = n - k + 1 \tag{2.25}$$

■

Theorem 2.3 and its proof are important for two reasons. Firstly it shows that RS codes can be designed such that their designed minimum distance is *always* the actual minimum distance (unlike BCH codes). Secondly, RS codes satisfy the Singleton bound with equality, so they are maximum distance separable.

Theorem 2.4 RS codes are invertible.

A code is said to be *invertible* if any k symbols can be used as information symbols in a systematic representation. This follows from the MDS property, proof is given in [Wicker, 1994, p. 189].

2.2 Convolutional Codes

2.2.1 Overview

The name *convolutional code* was coined by Elias [Elias, 1955] to describe a code which is the output sequence of a linear mapping of an input sequence with a discrete-time, finite-alphabet convolution of the input and encoder's impulse response [Wilson, 1996, p. 551]. Such codes are sometimes termed trellis codes, but this name is misleading because block codes may also be represented by trellises. The concept of encoding an input sequence without segmenting it is very different to that used by block codes (Section 2.1).

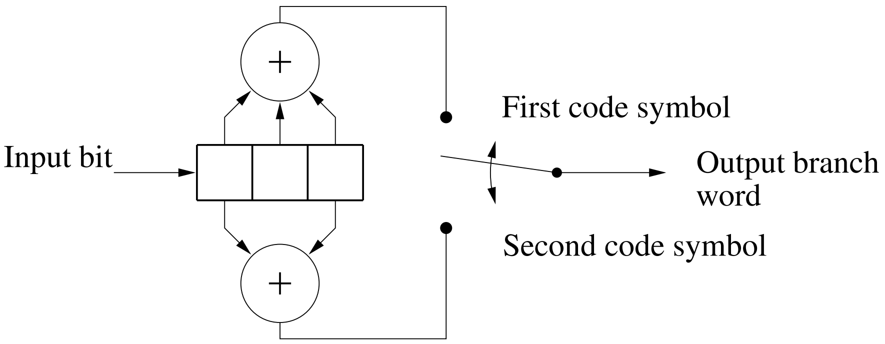


Figure 2.2: Encoder for (2, 1, 3) convolutional code.

2.2.2 General Properties

A convolutional code over GF(q) is usually described by the parameters (n, k, K), where k is the number of q-ary symbols (simultaneously) input to the decoder and n is the number of q-ary symbols (simultaneously) output from the decoder. As with a block code, the rate is given by n/k. The *constraint length* of the code, K, is defined as the number of consecutive symbols in the output stream affected by any input symbol. It is also the *memory* of the code.

2.3 Concatenated Codes

2.3.1 Overview

Concatenated coding was introduced in 1966 by Forney [Forney, 1966]. It is a powerful technique for creating error-correcting codes by combining two (or more) codes sequentially. The primary reason for using concatenated codes is to achieve a low error rate with an overall implementation complexity which is less than that which would be required by a single decoding operation [Sklar, 1988, p. 365]. Figure 2.3 shows a concatenated coding scheme. The data stream is first encoded with the *outer code* (in this case an RS block code). The output of the outer code is then re-encoded with the *inner code*, (here a convolutional code) before transmission. At the receiver the decoding order must be reversed, and so the inner code is decoded first. Any errors from the output of the inner decoder are likely to appear as bursts, hence it is usual to include an *interleaver* and *de-interleaver* between the inner and outer codes. The purpose of the interleaver is to rearrange the symbols so that errors do not occur in bursts but are spread through several outer codewords to allow correct decoding.

Convolutional codes are a natural choice for the inner code. With a suitable Viterbi decoder SD information can be used for maximum performance. RS codes are frequently used for the outer code. They are powerful and when combined with a

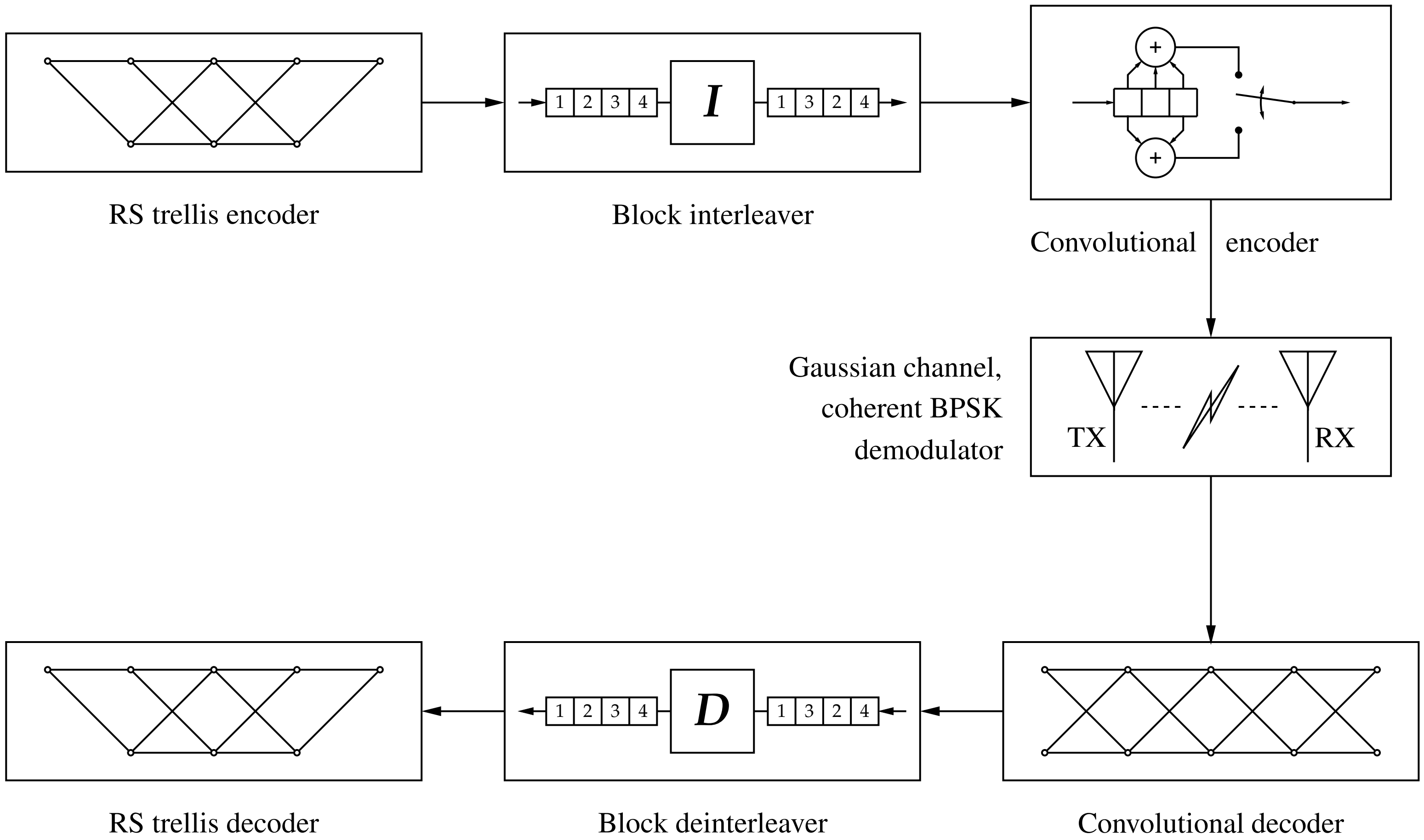


Figure 2.3: Concatenated coding example.

binary convolutional code the burst-error performance of the RS code helps minimise errors. There are, however, many possible configurations for a concatenated coding scheme and flexibility is one of its many advantages. For example the compact disc coding system uses a concatenated system based upon RS$(32, 28, 5)$ and RS$(28, 24, 5)$ shortened RS codes.

2.3.2 General Properties of Concatenated Codes

The rate of a concatenated code is the product of the rates of its component codes.

Consider a concatenated code $\mathcal{C}$ with an (n_1, k_1, d_1) inner code $\mathcal{C}_1$ and an (n_2, k_2, d_2) outer code $\mathcal{C}_2$. An input sequence of k_1k_2 symbols is passed to the outer encoder. The output is k_1n_2 symbols. This new block is sent to the inner decoder which results in n_1n_2 output symbols. The rate is thus $\frac{k_1k_2}{n_1n_2}$.

The minimum distance of a concatenated code is d_1d_2 [Lin and Costello, 1983, p. 279].

The proof is the same as for an array code, see Proof 2.1.

2.4 Trellises

2.4.1 Introduction

A *trellis diagram* (commonly called a *trellis*) is an *acyclic edge-labelled directed graph* [Muder, 1988], with one start point (the *root*) and one end point (the *goal*). The horizontal axis of a trellis diagram indicates the passage of time. The trellis can be viewed as a two-dimensional representation of the q^k codewords through the n-dimensional codespace. To enable an accurate description of the properties of a trellis it is first necessary to introduce some common definitions.

2.4.2 Definitions

Nodes of the graph represent possible encoder *states*, $S_{i,t}$, where i is the state number and t is the time index. The nodes are decomposed into a union of disjoint subsets, called *vertices* or *levels*. The levels are numerated $0,\ 1,\ \ldots,\ N_c$ ($N_c \leq n+1$) and the t-th level consists of N_t nodes, $(S_{1,t}),\ (S_{2,t}),\ \ldots,\ (S_{N_t,t})$.

Between states in adjacent levels, $S_{i,t}$ and $S_{j,t+1}$, there may be directed *branches* (also called *edges*), $B(S_{i,t} \rightarrow S_{j,t+1})$, which indicate possible changes in state. Branches are only permitted to connect adjacent levels. The set of branches between level $t-1$ and t is called the t-th depth. In some texts the set of branches at a given depth is termed a *stage* [Wicker, 1994, p. 292]. To prevent confusion (e.g., with "two-stage decoding") such terminology is avoided in this work. Associated with every branch is a *label* denoting the output (or code) symbol(s) given when that branch is taken, and a *branch metric*, B_m, which indicates the likelihood of a given branch being selected. For certain trellises an additional input (or data) label may exist. Its purpose is to allow the same trellis structure to be used for both encoding and decoding operations. The code label is a l_t-dimensional vector of q-ary symbols, $(l_1,\ l_2,\ \ldots,\ l_t)$. The code label associated with the branch $B(S_{i,t} \rightarrow S_{j,t+1})$ is denoted by $L(S_{i,t}, S_{j,t+1})$.

Using the notation introduced above the root can be more precisely defined as $S_{1,0}$, and the goal as S_{1,N_c}. A *path* is a continuous sequence of branches, and is denoted by $\mathcal{P}(S_{i,t} \rightarrow S_{j,t+1} \rightarrow S_{k,t+2} \rightarrow \ldots \rightarrow S_{z,t+\delta})$. The term *partial path* is sometimes used to denote a sequence of branches for which decoding is incomplete, thus the sequence starts from the root but does not reach the goal. For certain codes (generally convolutional codes) it is necessary to *truncate* the trellis to lessen decoding delay. Frequently these trellises are shown with multiple roots and goals (see Figure 4.7 for an example). Though this does not strictly match the definition of a trellis, the truncated section is often considered to be a trellis in its own right.

A trellis is called a *code trellis* of the code $\mathcal{C}$ if there is a one-to-one mapping between the codewords of the code $\mathcal{C}$ and all paths between $S_{1,0}$ and S_{1,N_c} (i.e., all $\mathcal{P}(S_{1,0} \rightarrow \ldots \rightarrow S_{1,N_c})$).

Let $\mathbb{N}(t) = [\mathbb{N}_0,\ \mathbb{N}_1,\ \ldots,\ \mathbb{N}_{N_c}]$ be the *state profile* of the trellis T and $\mathbb{B}(t) = [\mathbb{B}_1,\ \mathbb{B}_2,\ \ldots,\ \mathbb{B}_{N_c}]$ be the *branch profile*, where $\mathbb{N}_i$ is the number of states at the i-th level and $\mathbb{B}_j$ is the number of branches at the j-th depth [Forney and Trott, 1993; Honary and Markarian, 1997, p. 161]. Let $\mathbb{L}(t) = [\mathbb{L}_1,\ \mathbb{L}_2,\ \ldots,\ \mathbb{L}_{N_c}]$ be the *label size profile* of the trellis where $\mathbb{L}_j$ is the number of symbols used for labelling the j-th depth.

Figure 2.4 shows a trellis for the $(7, 4, 3)$ Hamming code annotated with the definitions described above. The state profile, branch profile and code label size profile of

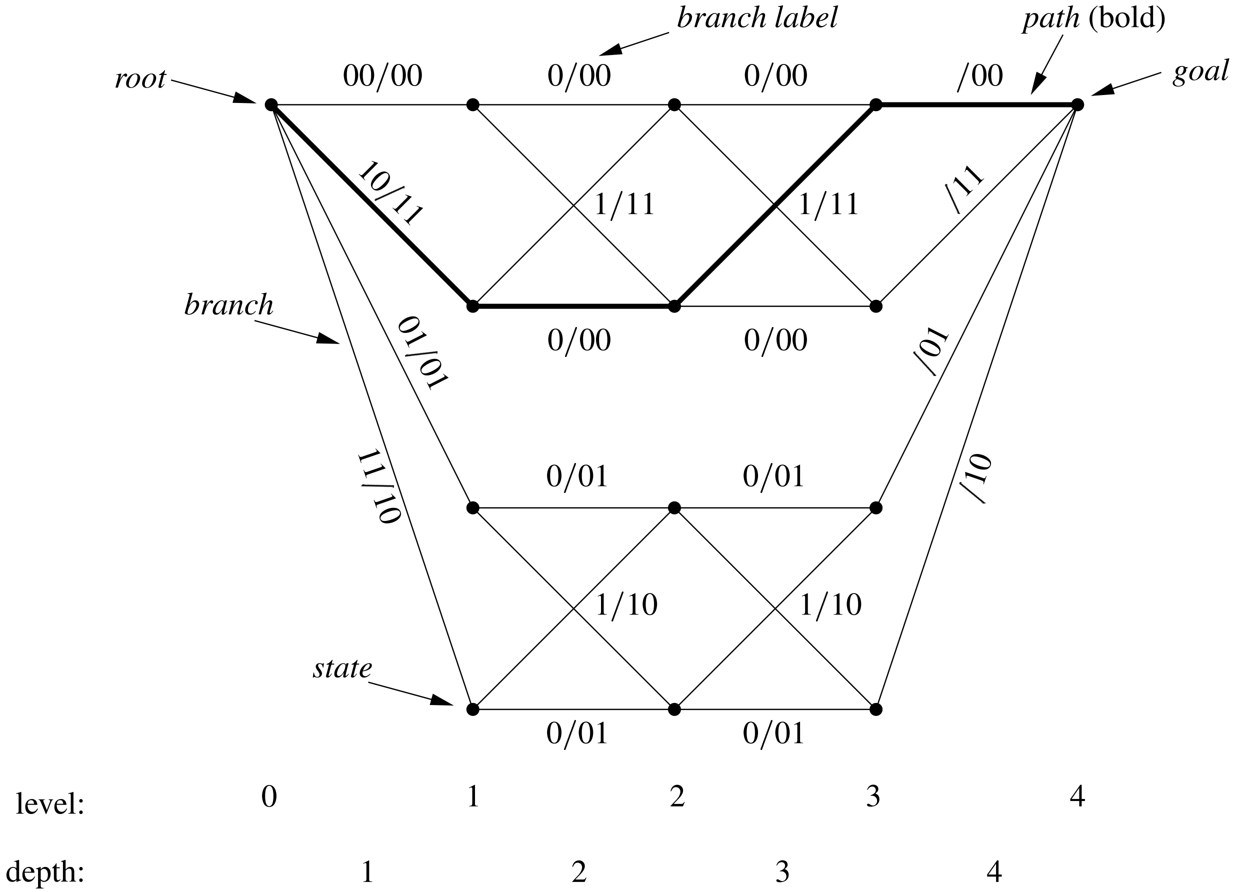


Figure 2.4: Trellis for RM(8, 4, 4) with definitions.

this trellis are

$$\begin{aligned}\mathbb{N}(t) &= [\mathbb{N}_0, \mathbb{N}_1, \mathbb{N}_2, \mathbb{N}_3, \mathbb{N}_4] \\ &= [1, 4, 4, 4, 1]\end{aligned} \tag{2.26}$$

$$\begin{aligned}\mathbb{B}(t) &= [\mathbb{B}_1, \mathbb{B}_2, \mathbb{B}_3, \mathbb{B}_4] \\ &= [4, 8, 8, 4]\end{aligned} \tag{2.27}$$

$$\begin{aligned}\mathbb{L}(t) &= [\mathbb{L}_1, \mathbb{L}_2, \mathbb{L}_3, \mathbb{L}_4] \\ &= [2, 2, 2, 1]\end{aligned} \tag{2.28}$$

2.4.3 Properties

Proper A trellis where all the branches (edges) leaving any state (vertex) have distinct labels. Unless otherwise stated all references to a trellis will imply a proper trellis.

Observable A trellis with a one-to-one mapping between all codewords of the code $\mathcal{C}$ and all paths between $S_{1,0}$ and S_{1,N_c} (i.e., all $\mathcal{P}(S_{1,0} \to \ldots \to S_{1,N_c})$).

Since an unobservable trellis contains more than one path through the trellis for at least one codeword this may cause difficulties for encoders and for sub-optimum decoders operating with a reduced-search Viterbi algorithm, or similar methods.

Minimal trellis Many definitions of a minimal trellis exist due to varying interpretations of minimality. The definition used within this text will be that given by

Muder [Muder, 1988]. The trellis T is a minimal trellis of the code $\mathcal{C}$ if for every other trellis T' of $\mathcal{C}$, $\mathbb{N}_i \leq \mathbb{N}'_i$ [Honary and Markarian, 1997, p. 61].

***State-Oriented* form** The trellis state number is directly correlated with the encoder state.

2.5 Channel Models

2.5.1 Discrete Memoryless Channel

A *discrete memoryless channel* features discrete input and output alphabets. The set of conditional probabilities relating the output to the input is given by $p(j \mid i)$ where i $(1 \leq i \leq M)$ is a modulator M-ary input symbol and j $(1 \leq j \leq q)$ is a demodulator q-ary output symbol. Thus $p(j \mid i)$ is the probability of receiving j if i was transmitted.

The output symbol depends only on the input symbol, not on the existencre of any previous errors (hence the channel has no *memory*). For an input sequence $\mathbf{U} = u_1, u_2, \ldots, u_N$ the conditional probability of the output sequence $\mathbf{Z} = z_1, z_2, \ldots, z_N$ is [Sklar, 1988, p. 261]

$$p(\mathbf{Z} \mid \mathbf{U}) = \prod_{m=1}^{N} p(z_m \mid u_m) \tag{2.29}$$

2.5.2 Binary Symmetric Channel

A *binary symmetric channel* is a special case of the discrete memoryless channel. The input alphabet size is 2, containing the binary elements "0" and "1". In addition, the conditional probabilities are symmetric:

$$p(0 \mid 1) = p(1 \mid 0) = p \tag{2.30}$$

$$p(0 \mid 0) = p(1 \mid 1) = 1 - p \tag{2.31}$$

The channel *transition probabilities* (2.30, 2.31) give the probability that a transmitted symbol is received incorrectly. The demodulator makes no attempt to indicate how well a symbol is received, it merely outputs a "0" or "1". This type of output is termed *hard decision*.

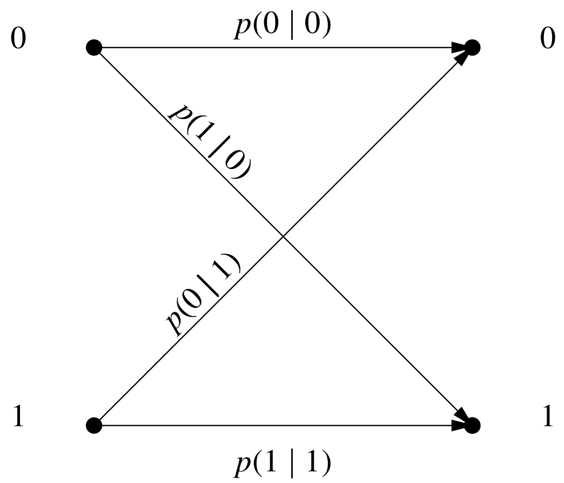


Figure 2.5: Binary symmetric channel model.

An upper bound for the probability of an incorrect decoding by a linear error-correcting code operating over a BSC can be obtained by considering the probability that $> t$ symbol errors occur [Lin and Costello, 1983]

$$P_M \leq \sum_{i=t+1}^{n} \binom{n}{i} P_s^{\,i} (1 - P_s)^{n-i} \tag{2.32}$$

2.5.3 Binary Symmetric Erasure Channel

The *binary symmetric erasure channel* may be viewed as a special case of the DMC, or as an extension of the BSC. Like the BSC the input alphabet size is 2, containing the binary elements. However the output alphabet size is increased to 3, and contains “0”, “1” and an *erasure* (denoted by “?”). For times when the demodulator is not able to clearly identify a “0” or “1” it may signal its uncertainty by outputting an erasure. The decoder is then aware that the symbol is unreliable. The symmetric conditional probabilities are

$$p(0 \mid 1) = p(1 \mid 0) = p \tag{2.33}$$

$$p(0 \mid ?) = p(1 \mid ?) = q \tag{2.34}$$

$$p(0 \mid 0) = p(1 \mid 1) = 1 - p - q \tag{2.35}$$

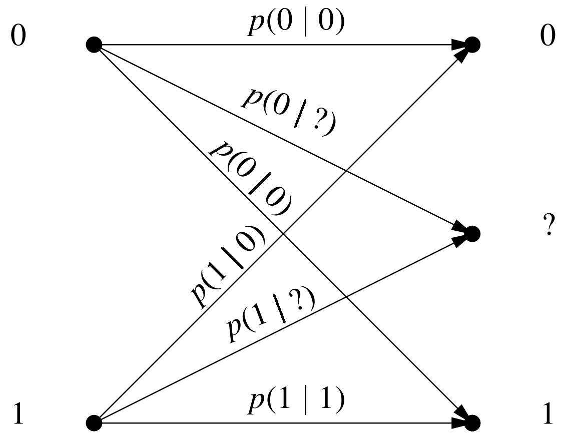


Figure 2.6: Binary symmetric erasure channel model.

2.5.4 Additive White Gaussian Noise Channel

In many cases channels are not discrete but feature a continuous output alphabet over the range $(-\infty \rightarrow +\infty)$. An AWGN channel is an example of such a case. The output is the input with broadband Gaussian noise added. The channel contains no memory (as defined in Section 2.5.1). This type of channel is an accurate channel model of many communication systems, such as satellite, deep-space and line-of-sight links.

White Gaussian noise is a random process, with a zero mean and a Gaussian PDF with variance σ^2. The power spectral density is flat over all frequencies $(-\infty \leq f \leq +\infty)$. The channel corrupts the transmitted signal with noise. The probability density function, y, of the noise value, x, is Gaussian and in the frequency domain the noise is wideband (or white).

$$y = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right\} \tag{2.36}$$

2.5.5 Quantised AWGN Channel

While channels with continuous output alphabets are a natural phenomenon they are impossible to use with SD decoding (due to requiring infinite precision to store the soft value). AWGN channels are frequently 'approximated' to a channel with a fixed number of noise values. Such a channel is termed a *quantised additive white Gaussian noise channel*. Like the standard AWGN channel it contains no memory. The quantised AWGN channel is discussed further in Section 5.1.2, where the relative merits (or *metrics*) of each quantisation level are calculated and their variation with E_b/N_0 for BPSK modulation is plotted.

Chapter 3

Algebraic Decoding of Reed-Solomon Codes

Chapter 3

Algebraic Decoding of Reed-Solomon Codes

3.1 Introduction

While RS codes are constructed by a few well-defined methods (Section 2.1.8) a large variety of decoding methods have been proposed. An important distinction is between those which are *algebraic* and those which are *combinatorial* in origin. Algebraic decoders attempt to correct errors and/or erasures by (algebraically) solving an equation to find the lowest weight error word. The fundamental method by which most RS algebraic decoders operate is by attempting to solve the *key equation*. However, not all algorithms use this approach, notable exceptions being Peterson's direct method [Peterson, 1960] and Massey's step-by-step algorithm [Massey, 1965]. In contrast to algebraic decoders, combinatorial decoders operate on more probabilistic methods to find the codeword which most closely matches the received word. Whilst many algebraic decoders can be adapted for error-and-erasures decoding it is true to say that they are unable to make maximum use of soft-decision information in the way that combinatorial decoders can. Most combinatorial decoding algorithms are trellis-based (e.g., the Fano algorithm and the Viterbi algorithm). Combinatorial decoders are discussed in Chapter 4.

Some important algebraic decoding algorithms include Euclidean decoding, based on Euclid's algorithm and the Berlekamp-Massey algorithm. In 1967 Berlekamp introduced an iterative method for decoding binary BCH codes [Berlekamp, 1967]. In Peterson's method the decoding complexity is proportional to the square of the errors corrected while for Berlekamp's algorithm the decoding complexity increases linearly with the number of errors corrected [Wicker, 1994, p. 211]. Thus Berlekamp's algorithm is much more suited for decoding long block codes where many errors may be corrected. In 1969 Massey described a "shift-register" synthesis of the Berlekamp algorithm [Massey, 1969]. The algorithm is now commonly called the Berlekamp-Massey algorithm in joint commemoration of their findings. In 1975 Sugiyama *et al.* showed how Euclid's algorithm, for finding the greatest common divisor of two integers or polynomials, can be used to solve the key equation and decode BCH and RS codes [Sugiyama *et al.*, 1975].

This Chapter begins with a discussion of the key equation (Section 3.2). The syndrome, error-locator and error-evaluator polynomials are defined. A method by which the error values can be calculated is given, along with its proof. Euclidean decoding is described in Section 3.3, and illustrated with an example. Section 3.4 describes the

Berlekamp-Massey algorithm. The same decoding example is repeated for the case of BM decoding. Finally, high-speed step-by-step decoding is shown. Again, the same decoding example is used to illustrate the algorithm. All the algorithms are described using the same form of the key equation, and have been generalised for the case when the BCH sense is not narrow (Section 2.1.7).

3.2 The Key Equation

3.2.1 Syndrome Calculation

Many common algebraic decoding algorithms for parity-check block codes start by testing if the syndromes (Section 2.1.2) of the received codeword are non-zero (i.e., if the parity checks fail). If the syndrome vector or polynomial is non-zero the received codeword is in error and error-correction is begun.

Let the received codeword be represented by the polynomial $r(x) = r_0 + r_1 x + \ldots + r_{n-1}x^{n-1}$. Let the error word corrupting the received word be represented by the polynomial $e(x) = e_0 + e_1 x + \ldots + e_{n-1}x^{n-1}$. The syndrome values[1] S_j, $(j = 1, 2, \ldots, 2t)$ is the value of the received polynomial evaluated at the $2t$ roots used to define the RS (or BCH) code [Wilson, 1996, p. 471]. Thus, for a narrow sense BCH code $(b = 1)$ the j-th syndrome is calculated by substituting α^j for x in $r(x)$. For the

[1]Note that these syndromes are not the same as the syndrome described in Section 2.1.6

more general case

$$
\begin{aligned}
S_j &= r(\alpha^{j+b-1}) \\
&= \sum_{i=0}^{n-1} (r_i)(\alpha^{j+b-1})^i \\
&= \sum_{i=0}^{n-1} (v_i + e_i)(\alpha^{j+b-1})^i \\
&= \sum_{i=0}^{n-1} (e_i)\alpha^{i(j+b-1)} \\
&\quad \text{where } j = 1,\ 2,\ \ldots\ ,\ 2t
\end{aligned}
\tag{3.1}
$$

3.2.2 Error-locator Polynomial

Let the received codeword contain ν ($\nu \leq t$) correctable errors. Let the locations of the errors be given by time indices $i_1,\ i_2,\ \ldots\ ,\ i_\nu$. For each symbol in error define an *error-locator*, X_i such that

$$X_i = \alpha^i \tag{3.2}$$

Noting that only symbols received in error contribute to the syndrome values it is possible to rewrite (3.1) in terms of the error-locators

$$S_j = \sum_{l=1}^{\nu} e_{i_l} X_l^{(j+b-1)} \tag{3.3}$$

The *error-locator polynomial*, $\Lambda(x)$, is defined as a polynomial whose inverse roots are the error locators [Wicker, 1994, p. 205] i.e., the inverses of the error locations are the roots of $\Lambda(x)$.

$$\Lambda(x) \triangleq \prod_{i=1}^{\nu}(1 - X_i x) \tag{3.4}$$

From (3.4) it can be seen that $\deg \Lambda(x) = \nu$, and that for no errors ($\nu = 0$) then $\Lambda(x) = 0$. For binary codes it is sufficient to find the location of an error, since the error value is always 1. However for RS and other multi-level codes it is necessary to find the value of each error in addition to the location of each error. It is therefore necessary to define an additional polynomial to find the value of the error(s).

3.2.3 Error-evaluator Polynomial

The *error-evaluator polynomial* is a polynomial which when evaluated at an error location gives the value of the error. It is defined as follows.[2] The syndrome polynomial is an infinite degree polynomial, however only the first $2t$ coefficients of x are known

$$\begin{aligned} 1 + S(x) &= 1 + \sum_{j=1}^{\infty} S_j x^j \\ &= 1 + \sum_{j=1}^{\infty}\left(\sum_{l=1}^{\nu} e_{i_l} X_l^{(j+b-1)}\right) x^j \\ &= 1 + \sum_{l=1}^{\nu} e_{i_l} \sum_{j=1}^{\infty}\left(X_l^{(j+b-1)} x^j\right) \end{aligned} \tag{3.5}$$

The summation $\sum_{j=1}^{\infty} X_l^{(j+b-1)} x^j$ can be simplified to a rational expression by noting it is a geometric series of the form $a + ar^1 + ar^2 + \ldots$, for which the simplified form is

[2]This is similar to [Wicker, 1994, p. 221], but has been generalised for the case when $b \neq 1$.

$S_\infty = \frac{a}{1-r}$

$$
\begin{aligned}
\sum_{j=1}^{\infty} X_l^{(j+b-1)} x^j &= \sum_{j=1}^{\infty} X_l^{b-1} (X_l x)^j \\
&= -X_l^{b-1} + \sum_{j=0}^{\infty} X_l^{b-1} (X_l x)^j \\
&= \frac{X_l^{b-1} + X_l^b x}{1 - X_l x} + \frac{X_l^{b-1}}{1 - X_l x} \\
&= \frac{X_l^b x}{1 - X_l x}
\end{aligned}
\tag{3.6}
$$

Therefore (3.5) becomes

$$1 + S(x) = 1 + \sum_{l=1}^{\nu} e_{i_l} \frac{X_l^b x}{1 - X_l x} \tag{3.7}$$

Multiplying both sides of (3.7) by $\Lambda(x)$ (Equation 3.4) produces the definition of the error-evaluator polynomial

$$
\begin{aligned}
\Lambda(x)\left[1 + S(x)\right] &= \Lambda(x) \left[1 + \sum_{l=1}^{\nu} \frac{e_{i_l} X_l^b x}{1 - X_l x}\right] \\
&= \Lambda(x) + \sum_{l=1}^{\nu} \left[\frac{e_{i_l} X_l^b x}{1 - X_l x} \prod_{i=1}^{\nu} (1 - X_i x)\right] \\
&= \Lambda(x) + \sum_{l=1}^{\nu} \left[e_{i_l} X_l^b x \prod_{\substack{i \neq l \\ i=1}}^{\nu} (1 - X_i x)\right] \\
&\triangleq \Omega(x)
\end{aligned}
\tag{3.8}
$$

As the decoder is only able to calculate the first $2t$ coefficients of $S(x)$ then $S(x)$ is unknown, though the decoder does know the value of $S(x) \bmod x^{2t+1}$. Thus the *key* *equation* which the decoder must solve is

$$\Lambda(x)\,[1 + S(x)] \equiv \Omega(x) \bmod x^{2t+1} \tag{3.9}$$

An alternative definition of the key equation sometimes found [Clark and Cain, 1981] is

$$\Lambda(x)S_{\text{alt}}(x) \equiv \Omega_{\text{alt}}(x) \bmod x^{2t+1} \tag{3.10}$$

While the error-locator polynomial is defined in the same way (3.4) it is important to note that the syndrome polynomial, $S_{\text{alt}}(x)$, in (3.10) is not the same as defined in (3.5); they are related by $S(x) = xS_{\text{alt}}(x)$. As the error-evaluator is also defined in a different manner the initial conditions for Euclidean and Berlekamp-Massey decoding also differ, as does the equation for calculating the error values from $\Lambda(x)$ and $\Omega_{\text{alt}}(x)$.

After the solution to the key equation has been found the error locations and values must still be found. Future references to the key equation will be to (3.9) only.

3.2.4 Locating the Errors

The error-locator polynomial, $\Lambda(x)$, contains $\nu, (0 \leq \nu \leq t)$ unique roots $\{X_1^{-1}, X_2^{-1}, \ldots, X_\nu{}^{-1}\}$, corresponding to error locations $\{i_1, i_2, \ldots, i_\nu\}$. The roots may be found by exhaustive substitution or Chien search [Chien, 1964]. The roots must be unique since an error can only occur in one position once. By definition of $\Lambda(x)$ (3.4), $\{X_1^{-1}, X_2^{-1}, \ldots, X_\nu{}^{-1}\} \in \mathrm{GF}(q)$. This implies $\deg \Lambda(x) = \nu$. If any of these conditions are not met the decoder should abort error-correction and declare a *decoder failure* to indicate that the codeword contains an uncorrectable number of errors. The reason why the decoder is able to identify a codeword containing $> t$ errors is that BCH and RS codes are not perfect [Wicker, 1994, p. 76] and there exist words in the codespace which are greater than t Hamming distance from the nearest codeword.

3.2.5 Calculation of the Error Values

Theorem 3.1 The Forney algorithm.

From the Forney algorithm [Forney, 1965] the error values $e_{i_l} = \{e_1,\ e_2,\ \ldots\ ,\ e_\nu\}$ are given by

$$e_{i_l} = \frac{-X_l^{2-b}\Omega(X_l^{-1})}{\Lambda'(X_l^{-1})} \tag{3.11}$$

where $X_l = \alpha^l$ is the error-locator (Section 3.2.2).

Proof 3.1 The proof given here is similar to [Wicker, 1994, p. 222], but has been extended for the case $b \neq 1$. Take the formal derivative of the error-locator polynomial, $\Lambda(x)$, noting the identity $[uv]' = u'v + uv'$.

$$\begin{aligned}\Lambda'(x) &= \left[\prod_{l=1}^{\nu}(1 - X_l x)\right]' \\ &= -\sum_{l=1}^{\nu}\left[X_l \prod_{\substack{j \neq l \\ j=1}}^{\nu}(1 - X_j x)\right]\end{aligned} \tag{3.12}$$

At the error location X_l the error-locator polynomial is $\Lambda(x) = 0$, with the root $x = X_l^1$.

Therefore substitute X_l^{-1} for x in (3.12).

$$\begin{aligned}
\Lambda'(X_l^{-1}) &= \\
\Lambda'(X_l^{-1}) &= -\sum_{l=1}^{\nu}\left[X_l\prod_{\substack{j\neq l\\ j=1}}^{\nu}(1-X_jX_l^{-1})\right] \\
&= -X_l\prod_{\substack{j\neq l\\ j=1}}^{\nu}(1-X_jX_l^{-1})
\end{aligned} \tag{3.13}$$

Similarly, substitute X_l^{-1} for x in the error-evaluator polynomial (3.8).

$$\Omega(X_l^{-1}) = e_{i_l}X_l^bX_l^{-1}\prod_{\substack{i\neq l\\ i=1}}^{\nu}(1-X_iX_l^{-1}) \tag{3.14}$$

Dividing (3.14) by (3.13) gives

$$\begin{aligned}
\frac{\Omega(X_l^{-1})}{\Lambda'(X_l^{-1})} &= \frac{e_{i_l}X_l^bX_l^{-1}\prod_{\substack{i\neq l\\ i=1}}^{\nu}(1-X_iX_l^{-1})}{-X_l\prod_{\substack{j\neq l\\ j=1}}^{\nu}(1-X_jX_l^{-1})} \\
&= \frac{-e_{i_l}X_l^bX_l^{-1}}{X_l}
\end{aligned} \tag{3.15}$$

$\therefore$

$$e_{i_l} = \frac{-X_l^{2-b}\Omega(X_l^{-1})}{\Lambda'(X_l^{-1})} \tag{3.16}$$

■

For narrow sense RS or BCH codes the error values are given by [Wicker, 1994, p. 222]

$$e_{i_l} = \frac{-X_l \Omega(X_l^{-1})}{\Lambda'(X_l^{-1})} \tag{3.17}$$

It can be seen (3.17) is equivalent to (3.11) for the case $b = 1$.

3.3 Euclidean Decoding

3.3.1 Euclid's Algorithm

Euclid's algorithm is a recursive method for finding the greatest common divisor of two elements in a Euclidean domain [Wicker, 1994, p. 50]. Examples of Euclidean domains include integer numbers and polynomials whose coefficients are elements in the same Galois field.

1. Let a and b represent two integers or polynomials, where $a > b$ if they are integers or $\deg a \geq \deg b$ if they are polynomials.

2. Initialise the time-indexed variable $r^{(i)}$ with the values $r^{(-1)} = a$ and $r^{(0)} = b$.

3. If $r^{(i-1)} \neq 0$, or for polynomials if $\deg r^{(-1)}(x) > \deg b(x)$, then define $r^{(i)}$ by

$$r^{(i)} = r^{(i-2)} - q^{(i)} r^{(i-1)} \tag{3.18}$$

   Repeat until $r^{(i)} = 0$. The greatest common divisor is given by $r^{(i-1)}$.

Example 3.1 Calculate the greatest common divisor for $a = 93$ and $b = 33$.

The values for $r^{(i)}$ and $q^{(i)}$ for each iteration i are given in Table 3.1. The algorithm terminates with $i = 4$, therefore $\gcd(93, 33) = r^{(3)} = 3$.

i	$q^{(i)}$	$r^{(i)}$	$r^{(i-2)} = q^{(i)}r^{(i-1)} + r^{(i)}$
-1	—	93	—
0	—	33	—
1	2	27	$93 = 2 \times 33 + 27$
2	1	6	$33 = 1 \times 27 + 6$
3	4	**3**	$27 = 4 \times 6 + 3$
4	2	0	$6 = 2 \times 3 + 0$

Table 3.1: Solution to Example 3.1.

□

3.3.2 Extended Version of Euclid's Algorithm

The extended version of Euclid's algorithm is used to find two values in the Euclidean domain such that

$$\gcd(a, b) = sa + tb \tag{3.19}$$

Since they are linearly related the same recursive algorithm can be used to find $s^{(i)}$ and $t^{(i)}$. That is

$$s^{(i)} = s^{(i-2)} - q^{(i)} s^{(i-1)} \tag{3.20}$$

$$t^{(i)} = t^{(i-2)} - q^{(i)} t^{(i-1)} \tag{3.21}$$

At each step of the algorithm $r^{(i)} = s^{(i)}t + t^{(i)}b$. The initial conditions are found as follows

$$r^{(-1)} = a = s^{(-1)}a + t^{(-1)}b \tag{3.22}$$

$\therefore$

$$s^{(-1)} = 1 \tag{3.23}$$

$$t^{(-1)} = 0 \tag{3.24}$$

$$r^{(0)} = b = s^{(0)}a + t^{(0)}b \tag{3.25}$$

$\therefore$

$$s^{(0)} = 0 \tag{3.26}$$

$$t^{(0)} = 1 \tag{3.27}$$

When the algorithm terminates s and t are given by $s^{(i-1)}$ and $t^{(i-1)}$ respectively.

Example 3.2 Calculate the values of s and t for $a = 93$ and $b = 33$.

The values for $r^{(i)}$, $q^{(i)}$, $s^{(i)}$ and $t^{(i)}$ for each iteration i are given in Table 3.2. The algorithm terminates with $i = 4$, therefore $s = 5$ and $t = -14$.

i	$q^{(i)}$	$r^{(i)}$	$s^{(i)}$	$t^{(i)}$
−1	—	93	1	0
0	—	33	0	1
1	2	27	1	−2
2	1	6	−1	3
3	4	3	**5**	−**14**
4	2	0	−11	31

Table 3.2: Solution to Example 3.2.

□

3.3.3 Euclid's Algorithm for Solving the Key Equation

The applicability of Euclid's algorithm for solving the key equation can be shown most clearly by re-expressing the key equation (3.9) as

$$\Lambda(x)\,[1 + S(x)] \equiv \Omega(x) \bmod x^{2t+1} \tag{3.28}$$

$$\Rightarrow \Theta(x)x^{2t+1} + \Lambda(x)\,[1 + S(x)] = \Omega(x) \tag{3.29}$$

Comparing (3.29) with (3.19) it can be seen that

$$a \equiv x^{2t+1} \tag{3.30}$$

$$b \equiv [1 + S(x)] \tag{3.31}$$

$$s \equiv \Theta(x) \tag{3.32}$$

$$t \equiv \Lambda(x) \tag{3.33}$$

$$\gcd(a, b) \equiv \Omega(x) \tag{3.34}$$

Therefore the extended form of Euclid's algorithm can be used to solve the key equation. As $\Theta(x)$ is not useful the iterations for $s^{(i)}$ are ignored. The Euclidean decoding algorithm is presented below:

1. Calculate the syndrome polynomial, $S(x)$.

2. Initialise the algorithm variables, let

$$r^{(-1)}(x) = x^{t+1} \tag{3.35}$$

$$r^{(0)}(x) = 1 \tag{3.36}$$

$$t^{(-1)}(x) = 0 \tag{3.37}$$

$$t^{(0)}(x) = 1 \tag{3.38}$$

3. If $r^{(i-1)}(x) \neq 0$ then define

$$r^{(i)}(x) = r^{(i-2)}(x) - q^{(i)}(x)r^{(i-1)}(x) \tag{3.39}$$

4. Define

$$t^{(i)}(x) = t^{(i-2)}(x) - q^{(i)}(x)r^{(i-1)}(x) \tag{3.40}$$

5. If $\deg r^{(i)}(x) > t$ go to step 3.

6. $\Lambda(x) = t^{(i)}(x)$ and $\Omega(x) = r^{(i)}(x)$. Find the roots of $\Lambda(x)$ and determine the error locations.

7. Find the error magnitudes using the Forney algorithm (3.11) and correct the errors.

Example 3.3 To correct an erroneous codeword from $\mathrm{RS}(15, 9, 7)$.

The code $\mathrm{RS}(15, 9, 7)$ is a triple error-correcting code over GF(16). Let the primitive polynomial for GF(16) be $1 + \alpha + \alpha^4$. Table 3.3 gives the elements of the field in polynomial representation. The identity element may also be represented by

$$\alpha^{15} = 1 \tag{3.41}$$

Let the sense of the code be $b = 3$. The generator polynomial, $g(x)$, of the code is (2.24)

$$\begin{aligned} g(x) &= (x - \alpha^3)(x - \alpha^4)(x - \alpha^5)(x - \alpha^6)(x - \alpha^7)(x - \alpha^8) \\ &= \alpha^3 + \alpha^4 x + \alpha^{14} x^2 + \alpha^{10} x^3 + \alpha^3 x^4 + \alpha^{12} x^5 + x^6 \end{aligned} \tag{3.42}$$

Element	Polynomial representation			
	α^2	α	1	0
0	0	0	0	0
1	0	0	0	1
α	0	0	1	0
α^2	0	1	0	0
α^3	1	0	0	0
α^4	0	0	1	1
α^5	0	1	1	0
α^6	1	1	0	0
α^7	1	0	1	1
α^8	0	1	0	1
α^9	1	0	1	0
α^{10}	0	1	1	1
α^{11}	1	1	1	0
α^{12}	1	1	1	1
α^{13}	1	1	0	1
α^{14}	1	0	0	1

Table 3.3: Galois field elements for GF(16).

Let the data polynomial be

$$u(x) = \alpha^3 x^6 \tag{3.43}$$

The systematic codeword is calculated from Equation 2.19, which gives

$$v(x) = \alpha^7 + \alpha^4 x + \alpha^{12} x^2 + \alpha^4 x^3 + \alpha^{11} x^4 + \alpha^9 x^5 + \alpha^3 x^{12} \tag{3.44}$$

After transmission the received codeword, $r(x)$, is corrupted with errors, $e(x)$, where

$$r(x) = v(x) + e(x) \tag{3.45}$$

Let

$$e(x) = \alpha^{13} + \alpha^{11}x + \alpha^3 x^{12} \tag{3.46}$$

therefore

$$r(x) = \alpha^5 + \alpha^{13}x + \alpha^{12}x^2 + \alpha^4 x^3 + \alpha^{11}x^4 + \alpha^9 x^5 \tag{3.47}$$

The syndromes are calculated as

$$S_j = 1.(\alpha^j) + \alpha^7.(\alpha^j)^2 + \alpha.(\alpha^j)^3 + \alpha^{10}.(\alpha^j)^4 + \alpha^{10}.(\alpha^j)^5 \tag{3.48}$$
$$\text{where } j = 1,\ 2,\ \ldots,\ 6$$

and are shown in Table 3.4. From (2.21) and Table 3.4

$$[1 + S(x)] = 1 + \alpha^{11}x + \alpha^{10}x^3 + \alpha^3 x^4 + \alpha^9 x^5 + \alpha^2 x^6 \tag{3.49}$$

The initial conditions for Euclid's algorithm are $\Omega^{(-1)}(x) = x^{2t+1} = x^7$, $\Omega^{(0)}(x) =$

S_j	Value
1	α^{11}
2	0
3	α^{10}
4	α^3
5	α^9
6	α^2

Table 3.4: Syndrome values.

$[1 + S(x)] = 1 + \alpha^{11}x + \alpha^{10}x^3 + \alpha^3x^4 + \alpha^9x^5 + \alpha^2x^6$, $\Lambda^{(-1)}(x) = 0$ and $\Lambda^{(0)}(x) = 1$. The error-locator polynomial, $\Lambda^{(i)}(x)$, the error-evaluator polynomial, $\Omega^{(i)}(x)$, and the quotient, $q^{(i)}$ are given in Table 3.5 for each iteration, i, of the algorithm.

The solution to the key equation is found after 3 iterations, the error-locator and error-evaluator are

$$\Lambda(x) = \alpha^5 + \alpha^{11}x + \alpha^3x^3 \tag{3.50}$$

$$\Omega(x) = \alpha^5 + \alpha^6x + \alpha^7x^2 + \alpha^{14}x^3 \tag{3.51}$$

The roots of the error-locator polynomial can be found by either exhaustive substitution or by a Chien search [Chien, 1964]. The roots are 1, α^3 and α^{14}, or in their inverse form, from (3.41), α^0, α^{-12} and α^{-1}. Thus $X_l = \{1, \alpha, \alpha^{12}\}$, corresponding to errors located at $\{x^0, x^1, x^{12}\}$.

All that remains is to calculate the value of the errors at the error locations and subtract the error polynomial from the received codeword. As the code is not narrow sense the error values must be found from (3.11). The formal derivative of $\Lambda(x)$ is

$$\begin{aligned}\Lambda'(x) &= \left[\alpha^5 + \alpha^{11}x + \alpha^3x^3\right]' \\ &= \alpha^{11} + 3(\alpha^3x^2) \\ &= \alpha^{11} + \alpha^3x^2\end{aligned} \tag{3.52}$$

i	$\Lambda^{(i)}(x)$	$\Omega^{(i)}(x)$	$q^{(i)}(x)$
-1	0	x^7	—
0	1	$S(x)$	—
1	$\alpha^5 + \alpha^{13}x$	$\alpha^5 + \alpha^{12}x + \alpha^9x^2 + x^3 + \alpha^7x^5$	$\alpha^5 + \alpha^{13}x$
2	$\alpha^9 + \alpha^8x^2$	$\alpha^9 + \alpha^5x + \alpha^8x^2 + \alpha^{12}x^4$	$\alpha^9 + \alpha^5x + \alpha^8x^2 + \alpha^{12}x^4$
3	$\alpha^5 + \alpha^{11}x + \alpha^3x^3$	$\alpha^5 + \alpha^6x + \alpha^7x^2 + \alpha^{14}x^3$	$\alpha^{10}x$

Table 3.5: Solution of the Key Equation using Euclid's algorithm.

The error values are calculated thus

$$
\begin{aligned}
e_{i_l} &= \frac{X_l^{2-b}\Omega(X_l^{-1})}{\Lambda'\left(X_l^{-1}\right)} \\
&= \frac{X_l^{-1}\left[\alpha^5 + \alpha^6\left(X_l^{-1}\right) + \alpha^7\left(X_l^{-1}\right)^2 + \alpha^{14}\left(X_l^{-1}\right)^3\right]}{\alpha^{11} + \alpha^3(X_l^{-1})^2} \\
&= \begin{cases} \alpha^{13} & \text{for } X_l = 1, \\ \alpha^{11} & \text{for } X_l = \alpha, \\ \alpha^3 & \text{for } X_l = \alpha^{12}. \end{cases}
\end{aligned}
\tag{3.53}
$$

Therefore the error polynomial is

$$
e(x) = \alpha^{13} + \alpha^{11}x + \alpha^3x^{12} \tag{3.54}
$$

The transmitted codeword was

$$
\begin{aligned}
v(x) &= r(x) - e(x) \\
&= \alpha^7 + \alpha^4x + \alpha^{12}x^2 + \alpha^4x^3 + \alpha^{11}x^4 + \alpha^9x^5 + \alpha^3x^{12}
\end{aligned}
\tag{3.55}
$$

□

3.4 Berlekamp-Massey Decoding

3.4.1 Introduction

Massey [Massey, 1969] recognised that the problem of finding the minimum-degree solution to the key equation is the same as finding the minimum-length feedback shift-register which generates the first $2t$ terms of the syndrome polynomial, $S(x)$. The Berlekamp-Massey algorithm may be used for decoding binary and multi-level codes. An overview of the algorithm is given below:

1. Find the shortest FSR which will predict S_2 from S_1.

2. Test the FSR to see if it will also predict S_3.

   If the test is successful (i.e., there is a *discrepancy* of 0) continue until the test fails (or all $2t$ syndromes have been generated).

   If the test fails use the discrepancy to modify the connections to the FSR so that

   (i) the next syndrome is correctly predicted,

   (ii) previous (correct) predictions are not changed,

   (iii) the FSR is increased by the smallest possible amount (to find the minimum weight error).

3. Continue until all $2t$ syndromes can be predicted correctly.

The connections to the FSR produce the error-locator polynomial, $\Lambda(x)$. Since $\Lambda(x)$ can be found from $\Omega(x)\,[1+S(x)]$ it follows that $\Omega^{(i)}(x)$ obeys the same recursive relationship as $\Lambda^{(i)}(x)$. Therefore a second FSR can be simultaneously constructed to find $\Omega(x)$. Both $\Lambda(x)$ and $\Omega(x)$ share the same discrepancy but need their own correction polynomials ($\lambda(x)$ and $\omega(x)$ respectively).

3.4.2 The Berlekamp-Massey decoding algorithm

The steps below show the procedure for Berlekamp-Massey decoding [Berlekamp, 1967; Massey, 1969] of an arbitrary BCH or RS code. This follows from [Wicker, 1994, p. 219], but with adaptations to work for any value of b, and to generate both the error-locator and error-evaluator polynomials.

1. Calculate the syndrome values, $S_1, S_2, \ldots, S_{2t}$ for the received word.

2. Initialise the algorithm variables, let

$$i = 0 \tag{3.56}$$

$$L = 0 \tag{3.57}$$

$$\Lambda^{(0)}(x) = 1 \tag{3.58}$$

$$\lambda(x) = x \tag{3.59}$$

$$\Omega^{(0)}(x) = 1 \tag{3.60}$$

$$\omega(x) = 0 \tag{3.61}$$

3. Increment the iteration counter, $i = i + 1$. Compute the discrepancy $\Delta^{(i)}$ by subtracting the i-th output of the FSR defined by $\Lambda^{(i-1)}(x)$ from the i-th syndrome.

$$\Delta^{(i)} = S_i - \sum_{j=1}^{L} \Lambda_j^{(i-1)} S_{i-j} \tag{3.62}$$

4. Test the new discrepancy. If $\Delta^{(i)} = 0$, then go to step 8.

5. Modify the connection polynomials to remove the discrepancy.

$$\Lambda^{(i)}(x) = \Lambda^{(i-1)}(x) - \Delta^{(i)}\lambda(x) \tag{3.63}$$

$$\Omega^{(i)}(x) = \Omega^{(i-1)}(x) - \Delta^{(i)}\omega(x) \tag{3.64}$$

6. Test the FSR length. If $2L \geq i$, then go to step 8.

7. Change the FSR length and update the correction polynomials.

$$L = i - L \tag{3.65}$$

$$\lambda(x) = \frac{\Lambda^{(i-1)}(x)}{\Delta^{(i)}} \tag{3.66}$$

$$\omega(x) = \frac{\Omega^{(i-1)}(x)}{\Delta^{(i)}} \tag{3.67}$$

8. Take into account the new length of the FSR.

$$\lambda(x) = x\lambda(x) \tag{3.68}$$

$$\omega(x) = x\omega(x) \tag{3.69}$$

9. Check if all the syndrome values have been used. If $i < 2t$, then go to step 3.

10. Determine the roots of $\Lambda(x) = \Lambda^{(2t)}(x)$.

11. If the roots are distinct and lie in GF(q) then calculate the error magnitudes (using Equation 3.11) for each error location. $v(x) = r(x) - e(x)$. STOP.

12. If the roots are not distinct, or do not lie in GF(q) then the calculated error-locator does not agree with its definition (3.4), declare a decoder failure. STOP.

Note that it is trivial to implement a decoder which can correct t_c errors and detect an extra t_d errors, where $2t_c + t_d = d - 1$. The syndromes are reduced in number to $2t_c$ and the condition in step 9 becomes $i < 2t_c$. It is also a simple matter to add error-and-erasure decoding to the BM algorithm [Wicker, 1994].

Example 3.4 To correct an erroneous codeword from RS$(15, 9, 7)$.

Let the code and received codeword be as defined in Example 3.3. Recall that the code has $b = 3$, and the received codeword is $r(x) = \alpha^5 + \alpha^{13}x + \alpha^{12}x^2 + \alpha^4x^3 + \alpha^{11}x^4 + \alpha^9x^5$. The values of the algorithm variables for each iteration i are given in Table 3.6.

The values of $\Lambda(x)$ and $\Omega(x)$ are identical to those found by Euclidean decoding (within a common constant factor, as found by [Clark and Cain, 1981, pp. 207–208]). The common constant factor is removed in the evaluation of (3.11). Hence the error polynomial is identical to (3.54). □

i	S_i	$\Lambda^{(i)}(x)$	Δ	L	$\lambda^{(i)}(x)$	$\Omega^{(i)}(x)$	$\omega^{(i)}(x)$
0	—	1	—	0	x	1	0
1	α^{11}	$1+\alpha^{11}x$	α^{11}	1	$\alpha^4 x$	1	$\alpha^4 x$
2	0	1	α^7	1	$\alpha^4 x^2$	$1+\alpha^{11}x$	$\alpha^4 x^2$
3	α^{10}	$1+\alpha^{14}x^2$	α^{10}	2	$\alpha^5 x$	$1+\alpha^{11}x+\alpha^{14}x^2$	$\alpha^5 x+\alpha x^2$
4	α^3	$1+\alpha^8 x+\alpha^{14}x^2$	α^3	2	$\alpha^5 x^2$	$1+\alpha^7 x+\alpha^9 x^2$	$\alpha^5 x^2+\alpha x^3$
5	α^9	$1+\alpha^8 x+\alpha^7 x^2$	α^{11}	3	$\alpha^4 x+\alpha^{12}x^2+\alpha^3 x^3$	$1+\alpha^7 x+\alpha^3 x^2+\alpha^{12}x^3$	$\alpha^4 x+\alpha^{11}x^2+\alpha^{13}x^3$
6	α^2	$1+\alpha^6 x+\alpha^{13}x^3$	α^{10}	3	$\alpha^4 x^2+\alpha^{12}x^3+\alpha^3 x^4$	$1+\alpha x+\alpha^2 x^2+\alpha^9 x^3$	$\alpha^4 x^2+\alpha^{11}x^3+\alpha^{13}x^4$

Table 3.6: Solution of the Key Equation using BM algorithm.

3.5 High-speed Step-by-step Decoding

3.5.1 Introduction

The step-by-step algorithm differs considerably in its approach from that of the Euclidean and Berlekamp-Massey algorithms, which both solve the key equation to compute the error values and their locations. Step-by-step decodes every possible error location for the correct error value individually and without solving the key equation. The original step-by-step algorithm was introduced by Massey [Massey, 1965] in 1965. However, in this section the modification proposed by Wei and Wei [Wei and Wei, 1993] is discussed as it is more efficient than the original.

The basic algorithm is as follows. Suppose a method exists to calculate the error weight of a received codeword without decoding it. First the initial error weight of the received word is calculated. A symbol is temporarily changed and the error-weight of the temporarily-changed word calculated. If the error-weight is increased the original symbol is correct. If the error weight has decreased then the new symbol is correct. Otherwise the symbol is in error but the new symbol is incorrect, and another value should be tried until the correct one is found. The process is repeated for all information symbols. It is not necessary to correct the parity symbols, though if desired they can be corrected in the same way.

For the case where the received codeword contains $t - 1$ or less errors, it is necessary to distinguish up to t errors since it is likely a correct symbol will be replaced with an incorrect one while searching for the first error location. However, if the full error-correction capability of the code is to be utilised then it is necessary to correct a codeword containing exactly t errors. Therefore it is necessary to distinguish between $t-1$ and $t+1$ errors.

3.5.2 Calculating the Error Weight

Of great importance to the step-by-step algorithm is the ability to efficiently calculate the error-weight of a received codeword. This can be achieved from the interdependence of the syndromes of RS codes using Theorem 9.9 of [Peterson and Weldon, 1972]. This states that for any RS code the matrix

$$\mathbf{N}_j = \begin{bmatrix} S_1 & S_2 & \dots & S_j \\ S_2 & S_3 & \dots & S_{j+1} \\ \vdots & \vdots & \ddots & \vdots \\ S_j & S_{j+1} & \dots & S_{2j-1} \end{bmatrix} \quad (3.70)$$
$$\text{where } j = 1,\ 2,\ \dots,\ n$$

is singular[3] if the weight of $e(x)$ is $j-1$ or less, and is non-singular if the weight is j. If the weight of $e(x)$ is greater than j, the result is not determinable. The proof is given in [Peterson and Weldon, 1972].

It was previously stated that an error weight of up to $t+1$ must be distinguishable by the decoder. From forming the matrix $\mathbf{N}_{t+1}$ it can be seen that the syndrome value $2t+1$ is required, which is not calculable for a t error-correcting code. However, in the calculation of $\det(\mathbf{N}_{t+1})$ the cofactor of S_{2t+1} is $\det(\mathbf{N}_t)$. Therefore if $\det(\mathbf{N}_t)$ is

[3]A matrix is said to be singular if its determinant is zero.

zero, then $\det(\mathbf{N}_{t+1})$ is independent of S_{2t+1} and can be calculated. Thus the problem of requiring S_{2t+1} can be overcome by defining a modified syndrome matrix, $\overline{\mathbf{N}}_{t+1}$ where S_{2t+1} is replaced by zero. Note that the result of $\det(\overline{\mathbf{N}}_{t+1})$ is only valid when $\det(\mathbf{N}_t)$ is zero.

For the purpose of calculating the error weight only the singularity of the syndrome matrix is important, the actual value if non-zero is discarded. Consequently the results may be expressed as a sequence of binary decision bits

$$h_j = \begin{cases} 1 & \text{if } \det(\mathbf{N}_j) = 0, \\ 0 & \text{if } \det(\mathbf{N}_j) \neq 0. \end{cases} \tag{3.71}$$

$$\text{where } j = 0,\ 1,\ \ldots,\ t$$

$$h_{t+1} = \begin{cases} 1 & \text{if } \det(\overline{\mathbf{N}}_{t+1}) = 0, \\ 0 & \text{if } \det(\overline{\mathbf{N}}_{t+1}) \neq 0. \end{cases} \tag{3.72}$$

As the error weight is dependent on all the $t+1$ binary decision bits it is useful to combine them into a decision vector, **D**.

$$\mathbf{D} = (h_1,\ h_2,\ \ldots,\ h_{t+1}) \tag{3.73}$$

From (3.70)

$$
\begin{array}{ll}
\text{Weight of } e(x) & \\
0 & \mathbf{D} \in \phi_0 = \{(1^{t+1})\} \qquad (3.74) \\
1 & \mathbf{D} \in \phi_1 = \{(0, 1^t)\} \qquad (3.75) \\
w & \mathbf{D} \in \phi_w = \{(X^{w-1}, 0, 1^{t-w+1})\} \qquad (3.76) \\
 & \text{where } w = 2,\ 3,\ \ldots,\ t-1 \\
t & \mathbf{D} \in \phi_t = \{(X^{t-1}, 0, X)\} \qquad (3.77) \\
t+1 & \mathbf{D} \in \phi_{t+1} = \{(X^{t-1}, 0, X), (X^{t-1}, 1, 0)\} \qquad (3.78)
\end{array}
$$

where $\{(1^{t+1})\}$ indicates $t+1$ consecutive "1"s and X indicates "0" or "1".

The conditions above show that $t+1$ errors are distinguishable from $t-1$ errors. It is not possible to distinguish t and $t+1$ errors in all cases. To account for this it is necessary to make a small modification to the basic algorithm described previously.

3.5.3 Calculating the Syndrome Values

Wei and Wei use a concept of updated syndrome values. Since the changes to the received codeword are known it is possible to only fully calculate the syndrome values once and from there on keep a running total. This concept is important in reducing the complexity of the algorithm. For a code with sense $b = 1$ the syndromes for a trial codeword with a trial error value β at location x^0 can be computed from [Wei and Wei, 1993]

$$S_i^1 = S_i^0 + \beta \quad \text{where } \beta = \alpha^j \quad j \in \{0,\ 1,\ \ldots,\ n-1\} \tag{3.79}$$

The above equation will now be generalised for codes other than narrow sense ($b > 1$), and for trial locations other than x^0. Let the received codeword be represented by

$$r(x)^0 = r_{n-1}x^{n-1} + r_{n-2}x^{n-2} + \cdots + r_1 x + r_0 \tag{3.80}$$

The syndromes for a BCH code are given by (from 3.1)

$$S_j = \sum_{i=0}^{n-1} r_i \alpha^{i(j+b-1)} \tag{3.81}$$

The original syndrome values are

$$S_1^0 = r_{n-1}\alpha^{(n-1)b} + r_{n-2}\alpha^{(n-2)b} + \cdots + r_2\alpha^{2b} + r_1\alpha^b + r_0 \tag{3.82}$$

$$S_2^0 = r_{n-1}\alpha^{(n-1)(b+1)} + r_{n-2}\alpha^{(n-2)(b+1)} + \cdots + r_2\alpha^{2(b+1)} + r_1\alpha^{(b+1)} + r_0 \tag{3.83}$$

$$\vdots$$

$$S_{2t}^0 = r_{n-1}\alpha^{(n-1)(b+1)} + r_{n-2}\alpha^{(n-2)(b+2t)} + \cdots + r_2\alpha^{2(b+2t)} + r_1\alpha^{(b+2t)} + r_0 \tag{3.84}$$

The implementation described in [Wei and Wei, 1993] cyclically shifts the codeword before correcting each symbol at the x^0 location. As RS codes are cyclic, any cyclic shift of a valid codeword is also a valid codeword. Hence cyclic shifts do not alter the error weight of the received word. To calculate the error weight cyclic shifts may be ignored, instead modifying the symbol at a variable position p. This gives

$$\begin{aligned} r(x)^1 &= r(x)^0 \\ &= r_{n-1}x^{n-1} + r_{n-2}x^{n-2} + \cdots + r_1 x + r_0 + \beta x^p \\ &\quad \text{where } \beta = \alpha^j \\ &\qquad j = 0,\ 1,\ \ldots,\ n-2 \\ &\qquad p = 0,\ 1,\ \ldots,\ n-1 \end{aligned} \tag{3.85}$$

The modified syndrome values are

$$S_1^0 = r_{n-1}\alpha^{(n-1)b} + r_{n-2}\alpha^{(n-2)b} + \cdots + r_2\alpha^{2b} + r_1\alpha^b + r_0 + \beta\alpha^{pb} \tag{3.86}$$

$$S_2^0 = r_{n-1}\alpha^{(n-1)(b+1)} + r_{n-2}\alpha^{(n-2)(b+1)} + \cdots + r_2\alpha^{2(b+1)} + r_1\alpha^{p(b+1)} + r_0 \tag{3.87}$$

$$\vdots$$

$$S_{2t}^0 = r_{n-1}\alpha^{(n-1)(b+1)} + r_{n-2}\alpha^{(n-2)(b+2t)} + \cdots + r_2\alpha^{2(b+2t)} + r_1\alpha^{p(b+2t)} + r_0 \tag{3.88}$$

Therefore the updated syndromes are generated by

$$
\begin{aligned}
S_i^1 &= S_i^0 + \beta\alpha^{p(i+b-1)} \\
&\text{where } \beta = \alpha^j \\
&\qquad i = 1,\ 2,\ \ldots,\ 2t \\
&\qquad j \in \{0,\ 1,\ \ldots,\ n-1\} \\
&\qquad p \in \{0,\ 1,\ \ldots,\ n-1\}
\end{aligned}
\tag{3.89}
$$

For the case $p = 0$ (3.89) reduces to (3.79).

3.5.4 The Algorithm

The algorithm described below is a slightly modified version of that given in [Wei and Wei, 1993]. In particular, the received codeword is not cyclically shifted, instead the position of the symbol under test moves. For the high-level implementation tested (Section 6.3.2) this approach was more efficient. Annotations to the algorithm are shown *thus*.

1. Calculate the initial syndrome values S_i^0 where $i = 0,\ 1,\ \ldots,\ 2t$ and then obtain $\mathbf{D}^0$.

2. Let $p = n - k$

   Only the information symbols require correction; the parity symbols occupy the least significant $2t$ *symbol positions in the received codeword.*

3. Let $j = 0$.

4. Let $\beta = \alpha^j$ then obtain $S_i^0 + \beta\alpha^{ip}$.

   Keep a running total of the temporarily-changed syndrome values.

5. If $\mathbf{D}^0 \in \phi_w$ and $\mathbf{D}^1 \in \phi_{w+1}$, where $0 \leq w < t$, then go to step 10.

   The number of errors has increased, therefore the original symbol value is correct.

6. If $\mathbf{D}^0 \in \phi_t$ and $\mathbf{D}^1 \in \{(X^{t-1}, 1, 0)\}$, then go to step 10.

   The temporarily-changed received codeword contains $t+1$ errors. Therefore the original codeword must have contained t errors. Hence the original symbol value is correct.

7. If $\mathbf{D}^0 \in \phi_w$ and $\mathbf{D}^1 \in \phi_{w-1}$, where $1 \leq w < t$, then add βx^p to the received word. Replace S_i^0 by S_i^1 and $\mathbf{D}^0$ by $\mathbf{D}^1$. Go to step 10.

   The number of errors has decreased by one. Therefore the temporary value of β is correct so update the algorithm variables.

8. If $\mathbf{D}^0 \notin \{\phi_0,\ \phi_1,\ \ldots,\ \phi_{t-1}\}$ and $\mathbf{D}^1 \in \phi_{t-1}$, then add βx^p to the received word. Replace S_i^0 by S_i^1 and $\mathbf{D}^0$ by $\mathbf{D}^1$. Go to step 10.

   The original codeword contained t errors; $t-1$ errors remain.

9. If $j < q-1$ then set $j = j+1$ and go to step 4.

   The symbol being tested may be incorrect and the correct symbol has not yet been found, try another value. Alternatively, $\mathbf{D}^1 \in \{X^{t-1}, 0, X\}$, hence it is not possible to ascertain whether the received codeword contains t or $t+1$ errors. For the case when $\mathbf{D}^1 \in \{X^{t-1}, 0, X\}$ the only course of action is to try different symbol values until either $\mathbf{D}^1 \in \{X^{t-1}, 1, 0\}$ or $\mathbf{D}^1 \in \phi_{t-1}$.

10. If $p < n-1$ then set $p = p+1$ and go to step 3.

    The current symbol has been tested, move onto the next symbol.

11. All the k symbols have been checked and corrected. The decoded information symbols are in the most significant k locations of the received codeword.

Example 3.5 To correct an erroneous codeword from RS$(15, 7, 9)$.

Let the code and received codeword be as defined in Example 3.3. The code has $b = 3$, and the received codeword is $r(x) = \alpha^5 + \alpha^{13}x + \alpha^{12}x^2 + \alpha^4x^3 + \alpha^{11}x^4 + \alpha^9x^5$.

For the key decoding stages, Table 3.5 shows the active decoding position, p, the trial error value, β, and the before and after error weights, $\mathbf{D}^0$ and $\mathbf{D}^1$ respectively. Also shown is the trial codeword. In this Example errors in the parity symbols are not corrected, so the first decoding position is $p = 6$, i.e., x^6. The decoder has correctly established that three symbols are in error ($|\mathbf{D}^0| = 3$). At location x^6 successive elements from GF(16) are tried in turn as the possible error value. For $\beta = \alpha^3$ the trial codeword has increased to distance 4 from the nearest correct codeword, thus the decoder is able to identify that the original symbol value was correct. This pattern is repeated up to $p = 11$. Note that for $p = 9$ the decoder cannot detect an increase in the error-weight. This means for every possible error pattern at location x^9 the distance to the nearest valid codeword is only $2t$, whereas for other locations the distance is $2t$ or $2t + 1$. As the decoder has not detected any decrease in error weight the original symbol must be correct, provided the original error weight was $\leq 2t$.

At location x^{12} the decoder has found and corrected an error—the error value is α^3 and is signalled by the decrease in error weight. Two more errors remain, the decoder searches the remaining two locations but the errors are not found since they are in the parity symbols. The algorithm required 52 attempts to correct the single error in the information symbols. □

				Trial information symbols								
p	β	$\lvert\mathbf{D}^0\rvert$	$\lvert\mathbf{D}^1\rvert$	x^6	x^7	x^8	x^9	x^{10}	x^{11}	x^{12}	x^{13}	x^{14}
6	1	3	3	**1**	0	0	0	0	0	0	0	0
6	α	3	3	$\boldsymbol{\alpha}$	0	0	0	0	0	0	0	0
6	α^2	3	3	$\boldsymbol{\alpha}^{\mathbf{2}}$	0	0	0	0	0	0	0	0
6	α^3	3	4	$\boldsymbol{\alpha}^{\mathbf{3}}$	0	0	0	0	0	0	0	0
7	1	3	3	0	**1**	0	0	0	0	0	0	0
7	α	3	3	0	$\boldsymbol{\alpha}$	0	0	0	0	0	0	0
7	...	...	...					...				
7	α^4	3	3	0	$\boldsymbol{\alpha}^{\mathbf{4}}$	0	0	0	0	0	0	0
7	α^5	3	4	0	$\boldsymbol{\alpha}^{\mathbf{5}}$	0	0	0	0	0	0	0
8	1	3	3	0	0	**1**	0	0	0	0	0	0
8	α	3	3	0	0	$\boldsymbol{\alpha}$	0	0	0	0	0	0
8	...	...	...					...				
8	α^{13}	3	3	0	0	$\boldsymbol{\alpha}^{\mathbf{13}}$	0	0	0	0	0	0
8	α^{14}	3	4	0	0	$\boldsymbol{\alpha}^{\mathbf{14}}$	0	0	0	0	0	0
9	1	3	3	0	0	0	**1**	0	0	0	0	0
9	α	3	3	0	0	0	$\boldsymbol{\alpha}$	0	0	0	0	0
9	...	...	...					...				
9	α^{13}	3	3	0	0	0	$\boldsymbol{\alpha}^{\mathbf{13}}$	0	0	0	0	0
9	α^{14}	3	3	0	0	0	$\boldsymbol{\alpha}^{\mathbf{14}}$	0	0	0	0	0
10	1	3	3	0	0	0	0	**1**	0	0	0	0
10	α	3	3	0	0	0	0	$\boldsymbol{\alpha}$	0	0	0	0
10	α^2	3	3	0	0	0	0	$\boldsymbol{\alpha}^{\mathbf{2}}$	0	0	0	0
10	α^3	3	3	0	0	0	0	$\boldsymbol{\alpha}^{\mathbf{3}}$	0	0	0	0
10	α^4	3	4	0	0	0	0	$\boldsymbol{\alpha}^{\mathbf{4}}$	0	0	0	0
11	1	3	4	0	0	0	0	0	**1**	0	0	0
12	1	3	3	0	0	0	0	0	0	**1**	0	0
12	α	3	3	0	0	0	0	0	0	$\boldsymbol{\alpha}$	0	0
12	α^2	3	3	0	0	0	0	0	0	$\boldsymbol{\alpha}^{\mathbf{2}}$	0	0
12	α^3	3	2	0	0	0	0	0	0	$\boldsymbol{\alpha}^{\mathbf{3}}$	0	0
13	1	2	3	0	0	0	0	0	0	α^3	**1**	0
14	1	2	3	0	0	0	0	0	0	α^3	0	**1**
—	—	2	—	0	0	0	0	0	0	α^3	0	0

Table 3.7: High-speed step-by-step decoding of RS$(15, 9, 7)$.

Chapter 4

Trellis Decoding

Chapter 4

Trellis Decoding

4.1 Trellis Construction Methods

4.1.1 Introduction

Techniques for designing block code trellises have been investigated ever since they were first proposed for error-correction in 1974 [Bahl *et al.*, 1974]. Wolf [Wolf, 1978] has shown that such a trellis, known as a *syndrome trellis* (also called a *Wolf* or *BCJR* trellis), can be used for maximum-likelihood decoding of an arbitrary linear block code. McEliece [McEliece, 1994] proved syndrome trellises are minimal and proposed a technique to obtain an optimal reordering of the generator matrix of the code.

In 1988 Forney [Forney, 1988b] introduced the concept of a *coset code* and its *coset trellis*. These trellises have a regular structure, composed of a number of identical subtrellises which differ only in the labelling of the trellis branches. This important advancement in trellis design allows a reduction in both the decoder complexity and storage requirements of a corresponding Viterbi decoder [Honary *et al.*, 1995b]. Muder [Muder, 1988] proved that coset trellises are minimal and that the number of states in the trellis diagram can be minimised by an appropriate reordering of the symbols in the codeword. A vast amount of literature has accumulated on the design of a minimal trellis for a given block code [Berger and Be'ery, 1993; Honary and Markarian, 1993a,b; Honary *et al.*, 1993; Kasami *et al.*, 1993a,b; Kschischang and Sorokine, 1995; Wu *et al.*, 1994; Zyablov and Sidorenko, 1993]. Optimal reorderings have been found for certain binary codes [Berger and Be'ery, 1993; Forney, 1988b; Honary *et al.*, 1995b; Kasami *et al.*, 1993a,b]. The general solution to this problem, and its extension for non-binary block codes, is however unsolved and remains a complex analytical task.

Trellises are usually constructed in their state-oriented form (Section 2.4.3). It is a simplification used for trellis construction, but it is not a requirement. Indeed, for some codes (e.g., non-linear codes) state-oriented form is not possible. Both syndrome and coset trellises are based upon their state-oriented form.

4.1.2 Shannon Product of Trellises

Shannon [Shannon, 1956] described a product of two channels which "corresponds to a situation where both channels are used each unit of time". A similar product exists for two (or more) trellises which are combined in such a manner that they share the same time indexes. This product is known as the *Shannon product* of trellises [Sidorenko *et al.*, 1995, 1996]. It is denoted by '$\star$'.

Proposition 4.1 Consider two codes $\mathcal{C}'$ and $\mathcal{C}''$ with the same label profile (and it follows, the same code length n). Let T' be a trellis of the code $\mathcal{C}'$ and T'' be a trellis of the code $\mathcal{C}''$. The Shannon product

$$T = T' \star T'' \tag{4.1}$$

of these trellises is the trellis of the code $\mathcal{C} = \mathcal{C}' + \mathcal{C}''$.

Proof 4.1 It is necessary to prove a one-to-one mapping between codewords of code $\mathcal{C}$ and the paths in T.

The sum of $\mathcal{C}' + \mathcal{C}''$ produces the set of all possible sums $\mathbf{v}' + \mathbf{v}''$. Let $\mathbf{v}' + \mathbf{v}'' \in \mathcal{C}$. Associated with $\mathbf{v}'$ is a path $\mathcal{P}'$ with labels $(l'_1,\ l'_2,\ \ldots,\ l'_n)$, while associated with $\mathbf{v}''$ is a path $\mathcal{P}''$ with labels $(l''_1,\ l''_2,\ \ldots,\ l''_n)$. By definition, in T there is a path with labels

$$\begin{aligned}(l'_1 + l''_1,\ l'_2 + l''_2,\ \ldots,\ l'_n + l''_n) &= (l'_1,\ l'_2,\ \ldots,\ l'_n) + (l''_1,\ l''_2,\ \ldots,\ l''_n) \\ &= \mathbf{v}' + \mathbf{v}'' \in \mathcal{C}' + \mathcal{C}''\end{aligned} \tag{4.2}$$

for any path $\mathcal{P}$ in the trellis T. By definition, there exists in T' a path $\mathcal{P}'$ with labels $(l'_1,\ l'_2,\ \ldots,\ l'_n)$ which is the path for codeword $\mathbf{v}'$ ($\mathbf{v}' \in \mathcal{C}'$). Likewise, there exists in T'' a path $\mathcal{P}''$ with labels $(l''_1,\ l''_2,\ \ldots,\ l''_n)$ which is the path for codeword $\mathbf{v}''$ ($\mathbf{v}'' \in \mathcal{C}''$).

For trellises which are labelled with both data and code symbols the Shannon product can be extended, and performed on each set of labels, where code labels are summed and data labels concatenated. ■

A code $\mathcal{C}$ having a sum structure can be constructed from the (linear) sum of its subcodes. There are many codes having the property of a sum structure, e.g., RS, RM and the Nordstrom-Robinson code. Alternatively, the trellis for any such code can be deconstructed into the trellises of its subcodes.

4.1.3 Syndrome Trellises

In proving syndrome trellises are minimal McEliece [McEliece, 1994] showed that the maximum number of states in a syndrome trellis can be estimated using Wolf's bound.

$$N_{\max} \leq \min\left\{q^k, q^{(n-k)}\right\} \tag{4.3}$$

The minimum number of states at the i-th level of the minimal trellis can be obtained as [Forney, 1988b; Zyablov and Sidorenko, 1993]:

$$\begin{gathered} N_i = \frac{q^k}{q^{k_{\text{past}}} q^{k_{\text{future}}}} \\ i = 0,\ 1,\ \ldots\ ,\ n \end{gathered} \tag{4.4}$$

where

$$k_{\text{past}} = \dim(\mathcal{C}_{\text{past}}) \tag{4.5}$$

$$k_{\text{future}} = \dim(\mathcal{C}_{\text{future}}) \tag{4.6}$$

$$\mathcal{C}_{\text{past}} = (i, k_{\text{past}}, d) \tag{4.7}$$

$$\mathcal{C}_{\text{future}} = (n-i, k_{\text{future}}, d) \tag{4.8}$$

Theorem 4.1 The maximum number of states in the minimal syndrome trellis of an RS code is defined as:

$$N_{\max} = \min\left\{q^{k}, q^{n-k}\right\} \tag{4.9}$$

Proof 4.2 To prove Theorem 4.1 consider the i-th and $(i+1)$-th vertices of the trellis such that $i = (n-1)/2$ and $i+1 = (n+1)/2$. Consider also the two different types of code:

(i) Low-rate RS code: $n - k > k$

It is easy to show since for RS codes $k = n - d + 1$ (Equation (2.25))

$$\frac{n}{2} < d - 1 \tag{4.10}$$

and

$$\frac{n+1}{2} < d \tag{4.11}$$

hence $k_{\text{future}} = k_{\text{past}} = 0$, and

$$N_{\max} = \frac{q^{k}}{q^{0}q^{0}} = q^{k} \tag{4.12}$$

(ii) High-rate RS code: $n - k < k$

Similar to (i) it follows that $N_{\max} = q^{n-k}$.

■

To obtain a syndrome trellis for an RS(n, k, d) code over GF(q). Let $\mathbf{u} = (u_1, u_2, \ldots, u_k)$ be a vector of information symbols and $\mathbf{v} = (v_1, v_2, \ldots, v_n)$ be the encoded vector where $u_i \in$ GF(q) and $v_i \in$ GF(q). Let $\mathbf{G}$ be the generator matrix of the RS code in cyclic form [Vardy and Be'ery, 1991]. $\mathbf{G}$ is given in the format:

$$\mathbf{G} = \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_k \end{bmatrix} \tag{4.13}$$

where g_i, $(i = 1, 2, \ldots, k)$ is the i-th row of the generator matrix $\mathbf{G}$. The desired RS code can be constructed as a sum of k codes:

$$\mathcal{C} = \sum_{j=1}^{k} \mathcal{C}_j \tag{4.14}$$

where the j-th code, $\mathcal{C}_j$, is an $(n, 1, d)$ code over GF(q) generated by $\mathbf{G}_j = \left[g_j\right]$. It is apparent that q codewords, $\mathbf{v}_j$, in $\mathcal{C}_j$ can be obtained as

$$\mathbf{v}_j = u_j \left[g_j\right] \tag{4.15}$$

$$j = 0, 1, \ldots, q-1 \tag{4.16}$$

since u_j is able to take any of q values. The corresponding trellises, T_j, $j = 1, 2, \ldots, k$ start from the root $(S_{1,0})$, and finish at the goal $(S_{1,n+1})$. The trellises have $n + 1$

vertices and the number of states in the t-th vertex is defined as follows [McEliece, 1994]:

$$N_0 = N_n = 1 \tag{4.17}$$

$$N_t = \begin{cases} q & \text{if } g_j^t \neq 0 \text{ and } g_j^{t+1} \neq 0 \\ 1 & \text{for all other cases} \end{cases} \tag{4.18}$$

where $t = 0,\ 1,\ \ldots,\ n$ and g_j^t is the t-th element of g_j.

Proposition 4.2 Let T_j be a syndrome trellis of an elementary code $\mathcal{C}_j$, generated by g_j, where $j = 1,\ 2,\ \ldots,\ k$. Then $T = T_1 \star T_2 \star \ldots \star T_k$ is the syndrome trellis of the code $\mathcal{C}$, generated by $\mathbf{G}$.

Proof 4.3 Since $\mathcal{C} = \sum_{j=1}^{k} \mathcal{C}_j$, proof follows from Proposition 4.1. The state profile of the designed syndrome trellis can be obtained as follows:

$$\begin{aligned} N_0 &= N_n = 1 \\ N_t &= q^m \end{aligned} \tag{4.19}$$

where m is a number of non-zero elements in the t-th column of $\mathbf{G}$ followed by any other non-zero element. ■

Theorem 4.2 The designed trellis is a minimal syndrome trellis.

Proof 4.4 In order to prove Theorem 4.2 it is necessary to show the maximum number of states in the designed trellis is given according to Theorem 4.1. Thus calculate the parameter m for two different cases.

Consider a row of **G**. It has d consecutive non-zero elements from which $(d-1)$ elements are followed by a non-zero element.

(i) Low-rate RS code: $(n-k>k)$

the maximum number of non-zero elements which are followed by any other non-zero element is given by $m_{\max}=k$.

(ii) High-rate RS code: $(n-k<k)$

the maximum number of non-zero elements which are followed by any other non-zero element is given by $m_{\max}=n-k$.

Thus the maximum number of states in the designed trellis is defined as

$$N_t^{\max}=\min\left\{q^k,q^{n-k}\right\} \tag{4.20}$$

and from Theorem 4.1 it follows that the designed trellis is minimal. ■

Example 4.1 To design the syndrome trellis for the narrow-sense RS$(7,5,3)$ code with symbols taken from GF(8).

The generator polynomial is

$$\begin{aligned} g(x) &= (x-\alpha)(x-\alpha^2) \\ &= x^2+\alpha^4x+\alpha^3 \end{aligned} \tag{4.21}$$

from which the generator matrix can be obtained as follows:

$$\mathbf{G} = \begin{bmatrix} g_1 \\ g_2 \\ g_3 \\ g_4 \\ g_5 \end{bmatrix} = \begin{bmatrix} \alpha^3 & \alpha^4 & 1 & 0 & 0 & 0 & 0 \\ 0 & \alpha^3 & \alpha^4 & 1 & 0 & 0 & 0 \\ 0 & 0 & \alpha^3 & \alpha^4 & 1 & 0 & 0 \\ 0 & 0 & 0 & \alpha^3 & \alpha^4 & 1 & 0 \\ 0 & 0 & 0 & 0 & \alpha^3 & \alpha^4 & 1 \end{bmatrix} \tag{4.22}$$

The overall code, $\mathcal{C}$, is represented as

$$\mathcal{C} = \sum_{j=1}^{5} \mathcal{C}_j \tag{4.23}$$

where $\mathcal{C}_j$ is a $(7, 1, 3)$ code generated by g_j:

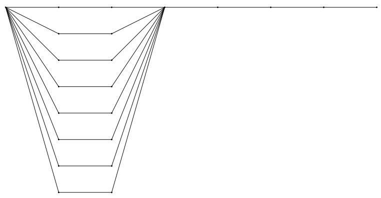
$$\mathcal{C}_1 = u_1[g_1] = (\alpha^3 u_1,\ \alpha^4 u_1,\ u_1,\ 0,\ 0,\ 0,\ 0) \tag{4.24}$$

$$\mathcal{C}_2 = u_2[g_2] = (0,\ \alpha^3 u_2,\ \alpha^4 u_2,\ u_2,\ 0,\ 0,\ 0) \tag{4.25}$$

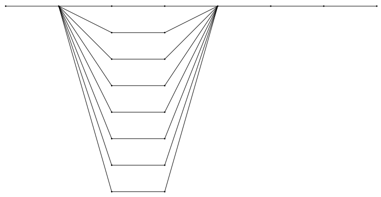
$$\vdots$$

$$\mathcal{C}_5 = u_5[g_5] = (0,\ 0,\ 0,\ 0,\ \alpha^3 u_5,\ \alpha^4 u_5,\ u_5) \tag{4.26}$$

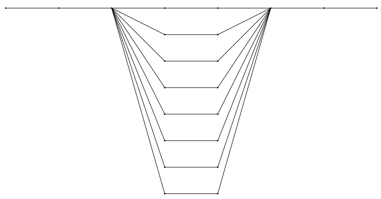
The component trellises have a very simple structure (as shown in Figure 4.1) with the



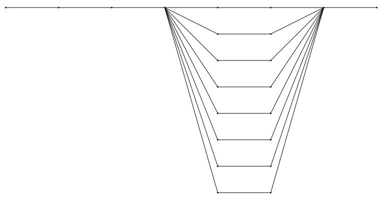
(a) T_1



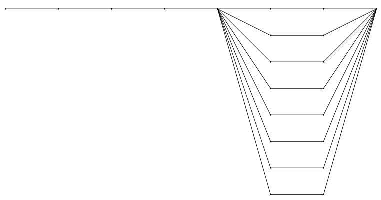
(b) T_2



(c) T_3



(d) T_4



(e) T_5

Figure 4.1: Component Syndrome Trellises for RS$(7, 5, 3)$.

state profiles defined according to (4.19):

$$N_1(t) = (1,\ 8,\ 8,\ 1,\ 1,\ 1,\ 1,\ 1) \tag{4.27}$$

$$N_2(t) = (1,\ 1,\ 8,\ 8,\ 1,\ 1,\ 1,\ 1) \tag{4.28}$$

$$N_3(t) = (1,\ 1,\ 1,\ 8,\ 8,\ 1,\ 1,\ 1) \tag{4.29}$$

$$N_4(t) = (1,\ 1,\ 1,\ 1,\ 8,\ 8,\ 1,\ 1) \tag{4.30}$$

$$N_5(t) = (1,\ 1,\ 1,\ 1,\ 1,\ 8,\ 8,\ 1) \tag{4.31}$$

Applying the procedure outlined above, the Shannon product of the trellises, $T = T_1 \star T_2 \star T_3 \star T_4 \star T_5$ has the following state profile:

$$N(t) = (1,\ 8,\ 64,\ 64,\ 64,\ 64,\ 64,\ 8,\ 1) \tag{4.32}$$

and the overall syndrome trellis for the $\mathrm{RS}(7,5,3)$ code is shown in Figure 4.2. Similar trellises can be obtained using the techniques described in [Wolf, 1978]. However, the technique described here allows one to label the designed trellis with both information and encoded symbols and is easier to implement. Using the technique proposed in [Forney, 1988b; Zyablov and Sidorenko, 1993] it is easy to show that the minimal trellises for $\mathrm{RS}(7,5,3)$ must have 64 states and the designed trellis is isomorphic to the minimal trellis of the code. Therefore the designed trellis is a minimal trellis.

The isomorphism of the trellis is demonstrated in Figure 4.3, which is instead constructed from the Shannon product $T = (T_4 \star T_2) \star ((T_1 \star T_5) \star T_5)$ and clearly shows the cosets of the underlying RS code. □

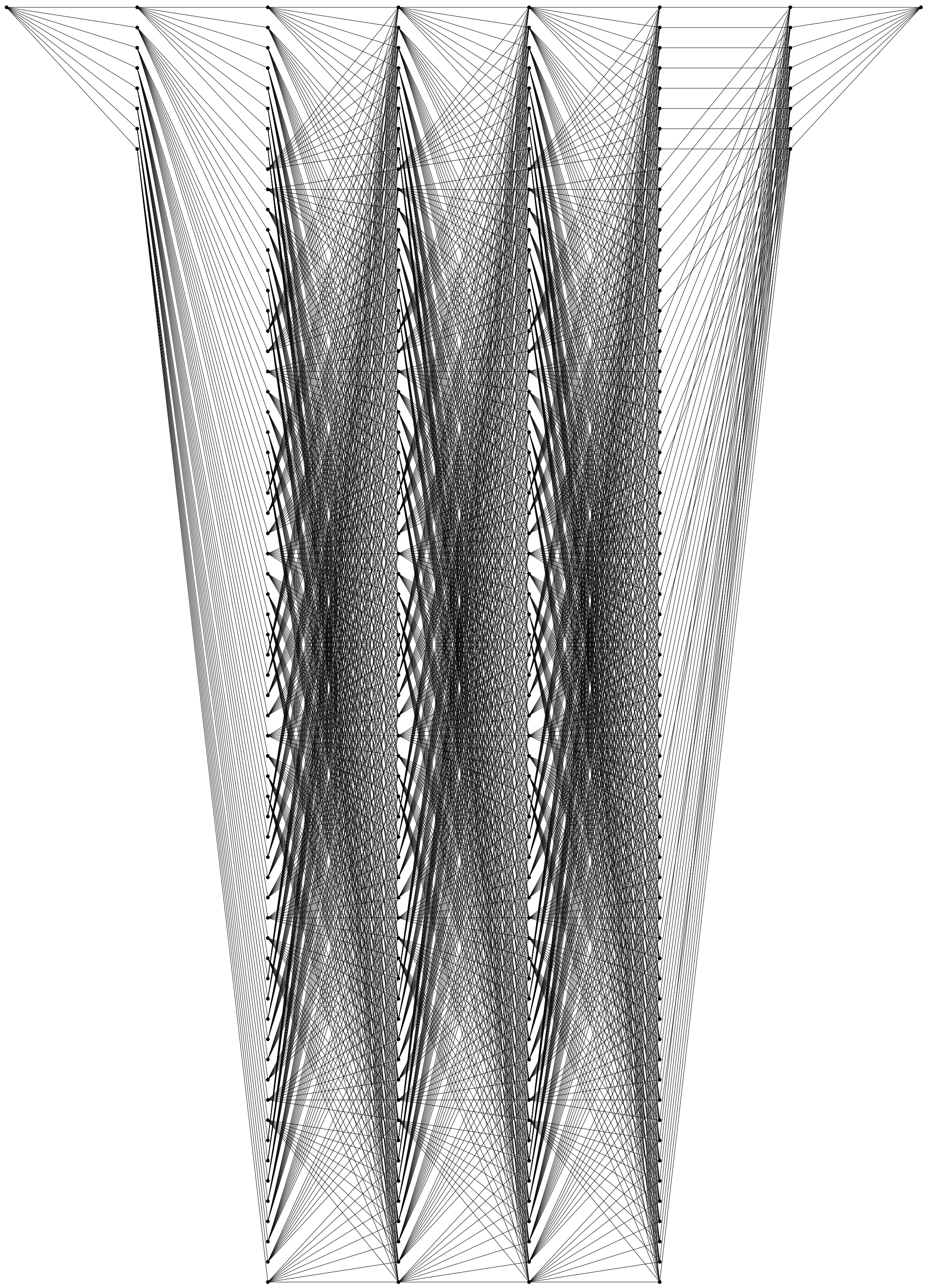


Figure 4.2: Syndrome trellis for RS$(7, 5, 3)$.

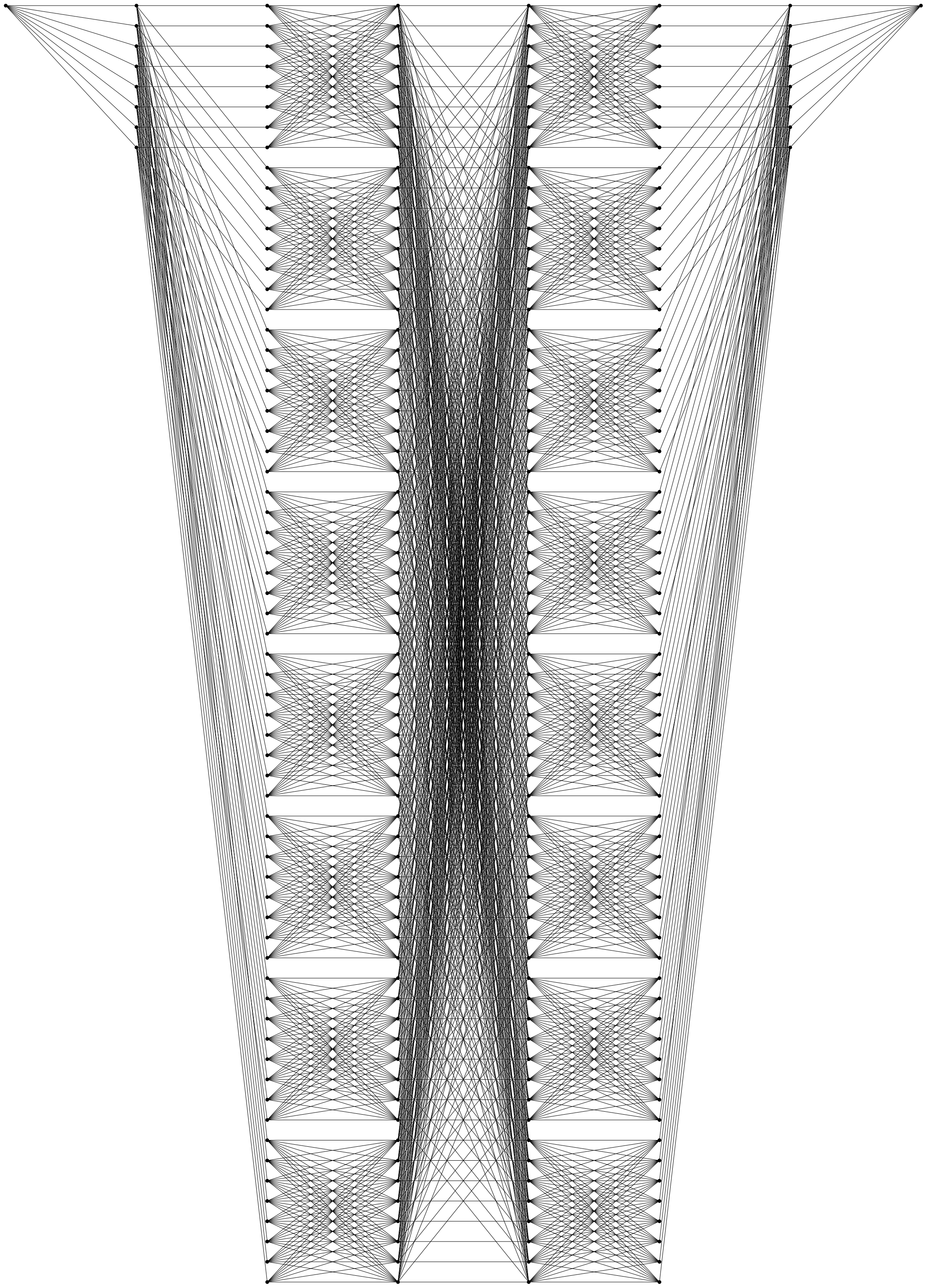


Figure 4.3: A trellis isomorphic to the RS$(7, 5, 3)$ syndrome trellis.

4.1.4 Coset Trellises

A coset trellis contains a set of parallel subtrellises. Its highly regular structure enables the storage requirements to be reduced since each subtrellis is identical in structure and only the branch labels differ. Moreover, the labels differ by a fixed offset which is the value of the coset leader. Decoding algorithms which are able to take advantage of its regular structure (e.g., two-stage decoding) are of lower complexity than the Viterbi algorithm (Section 6.4.2). Alternatively, a suitable decoder can take advantage of the parallel subtrellises to perform most of the decoding operations in parallel and thus operate at a higher throughput than a decoder over an irregular trellis.

The Shannon product of trellises (Section 4.1.2) can be applied to the design of minimal coset trellises of RS codes. To design a coset trellis it is first necessary to calculate the state profile of the minimal syndrome trellis for the RS(n,k,d) code:

$$N_{\text{synd}} = [N_0,\ N_1,\ \ldots\ ,\ N_n] \tag{4.33}$$

The state profile can be obtained from the calculation of the minimal number of states for every splitting point of the trellis (4.4). From the calculated N_{synd} choose splitting points which have the same number of states and define the state and label profiles of the desired trellis:

$$N_{\text{coset}} = [1,\ N_1,\ N_2,\ \ldots\ ,\ N_{N_c-1},\ 1] \tag{4.34}$$

$$L_{\text{coset}} = [l_1,\ l_2,\ \ldots\ ,\ l_{N_c-1}] \tag{4.35}$$

where N_c is the number of columns (vertices) in the desired coset trellis. Since all vertices have the same number of states:

$$N_i = n_j \tag{4.36}$$

$$i,\ j = 1,\ 2,\ \ldots,\ N_c - 1 \tag{4.37}$$

and in general

$$l_i \neq l_j \tag{4.38}$$

$$i,\ j = 1,\ 2,\ \ldots,\ N_c - 1 \tag{4.39}$$

At the next stage represent the generator matrix **G** in the following format:

$$\mathbf{G} = \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_k \end{bmatrix} = \begin{bmatrix} G_1 & G_2 & \ldots & G_{N_c-1} \end{bmatrix} \tag{4.40}$$

where $G_i, i = 1,\ 2,\ \ldots,\ N_c - 1$, has l_i columns and k rows. Each row of **G** is used to design the trellis diagram of the $(n, 1, d)$ code over GF(q) with the label size profile given as in (4.35), and the overall trellis diagram can be obtained as the Shannon product of k designed component trellises. The designed trellis is a minimal coset trellis [Honary and Markarian, 1997].

Example 4.2 To design a coset trellis for RS$(7, 3, 5)$ with symbols taken from GF(8).

The generator polynomial is

$$\begin{aligned} g(x) &= (x-\alpha)(x-\alpha^2)(x-\alpha^3)(x-\alpha^4) \\ &= \alpha^3 + \alpha x + x^2 + \alpha^3 x^3 + x^4 \end{aligned} \tag{4.41}$$

from which the generator matrix can be obtained as follows:

$$\mathbf{G} = \begin{bmatrix} \alpha^3 & \alpha & 1 & \alpha^3 & 1 & 0 & 0 \\ 0 & \alpha^3 & \alpha & 1 & \alpha^3 & 1 & 0 \\ 0 & 0 & \alpha^3 & \alpha & 1 & \alpha^3 & 1 \end{bmatrix} \tag{4.42}$$

Following the procedure outlined above, the state profile of the trellis is obtained as $N_{\text{synd}} = [N_0,\ N_1,\ \ldots,\ N_7]$, where

$$N_0 = \frac{q^3}{q^0 q^3} = 1 \tag{4.43}$$

$$N_1 = \frac{q^3}{q^0 q^2} = 8 \tag{4.44}$$

$$N_2 = \frac{q^3}{q^0 q^1} = 64 \tag{4.45}$$

$$N_3 = \frac{q^3}{q^0 q^0} = 512 \tag{4.46}$$

$$N_4 = \frac{q^3}{q^0 q^0} = 512 \tag{4.47}$$

$$N_5 = \frac{q^3}{q^1 q^0} = 64 \tag{4.48}$$

$$N_6 = \frac{q^3}{q^2 q^0} = 8 \tag{4.49}$$

$$N_7 = \frac{q^3}{q^3 q^0} = 1 \tag{4.50}$$

and $N_{\text{synd}} = [1,\ 8,\ 64,\ 512,\ 512,\ 64,\ 8,\ 1]$. It is apparent that for a given RS$(7,3,5)$

code it is possible to design a number of different (but isomorphic) minimal trellises. Three possible solutions, each with different state and label profiles, are given below:

(i)

$$N = [1,\ 8,\ 8,\ 1] \tag{4.51}$$

$$L = [1,\ 5,\ 1] \tag{4.52}$$

(ii)

$$N = [1,\ 64,\ 64,\ 1] \tag{4.53}$$

$$L = [2,\ 3,\ 2] \tag{4.54}$$

(iii)

$$N = [1,\ 512,\ 512,\ 1] \tag{4.55}$$

$$L = [3,\ 1,\ 3] \tag{4.56}$$

Choosing solution (ii) the generator matrix of the code is:

$$\mathbf{G} = \left[\begin{array}{ccc} G_1 & G_2 & G_3 \end{array}\right] = \left[\begin{array}{cc:ccc:cc} \alpha^3 & \alpha & 1 & \alpha^3 & 1 & 0 & 0 \\ 0 & \alpha^3 & \alpha & 1 & \alpha^3 & 1 & 0 \\ 0 & 0 & \alpha^3 & \alpha & 1 & \alpha^3 & 1 \end{array}\right] \tag{4.57}$$

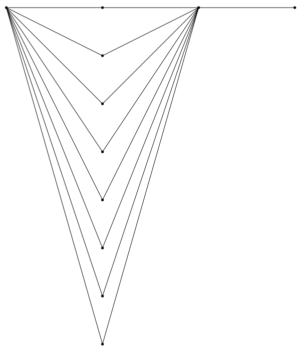
The overall trellis diagram, T, can be obtained as the Shannon product of three trellises, $T = T_1 \star T_2 \star T_3$, each one corresponding to a $(7, 1, 5)$ code, generated by its respective row of $\mathbf{G}$. These trellises are presented in Figure 4.4 and the overall trellis diagram is shown in Figure 4.5. As follows from Figure 4.5, the minimal coset trellis for RS$(7, 3, 5)$ consists of 8 identical, parallel subtrellises which differ only in their labelling. Each such subtrellis has 4 vertices, and a maximum 8 states. $\square$

4.2 Decoding Algorithms

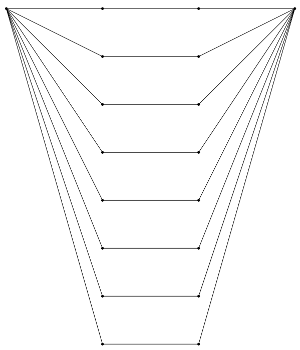
4.2.1 Introduction

The aim of a trellis decoder is to choose the *best* path through the trellis, either by maximizing the similarity or minimizing the difference between the received sequence and one of the codewords. Depending upon which metrics are in use *best* may mean the largest or smallest path metric. If the trellis is labelled with both data and code symbols the decoding algorithm can usually be configured to output either data or code symbols. This is true for the Viterbi and soft-output Viterbi algorithms, and two-stage decoding (Section 4.2.3). While the dataword is normally the required output some product code decoding algorithms may require the most likely codeword (Section 5.3.4).

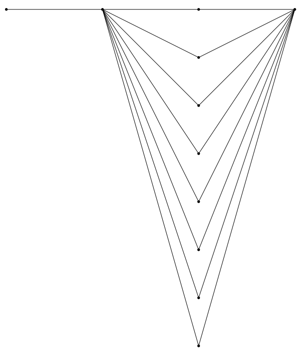
Trellises for block codes are fixed in length, while the length of a convolutional trellis is related to the message length. Since this can result in exceedingly long trellises convolutional decoding algorithms normally truncate the trellis early to reduce the decoding delay and memory to a finite, known value. For this reason convolutional



(a) T_1



(b) T_2



(c) T_3

Figure 4.4: Component Coset trellises for RS$(7, 3, 5)$.

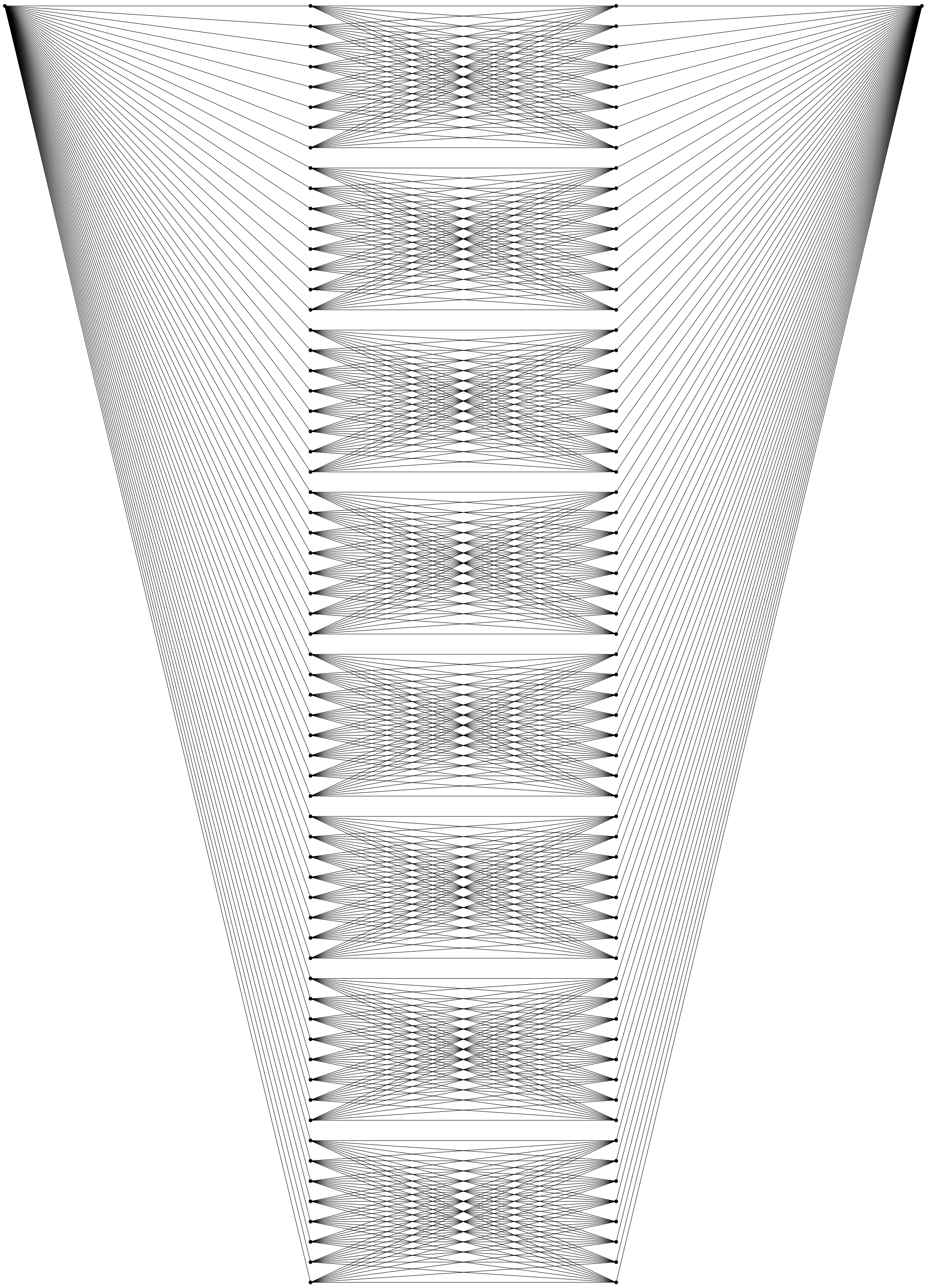


Figure 4.5: Coset trellis For RS$(7, 3, 5)$.

code trellises are frequently seen with multiple start and end points, reflecting the fact that the decoding sequence may begin and end at any state. Heller and Jacobs have shown that the length of the truncated trellis should be 4 or 5 times the code constraint length, by which time it can be assumed that all surviving paths have merged with the ML path [Heller and Jacobs, 1971]. Forney's more conservative result [Forney, 1970] sets the minimum decoding length at $5.8K$.

4.2.2 Viterbi Decoding

An asymptotically-optimum decoding algorithm was proposed by Viterbi [Viterbi, 1967] in 1967. It was later shown [Omura, 1969] that the VA provides a ML decoding solution for convolutional codes, and has since been used for ML decoding of block codes also. It reduces the computational load by taking advantage of the trellis structure. It calculates a series of *path metrics* which are a measure of the similarity (or difference) between the received sequence and the possible transmitted sequences. The VA eliminates paths which cannot possibly form part of the ML path. This is performed when two or more branches enter a node; the partial path having the best metric is chosen to become the *surviving path*. This continues until the end of the trellis is reached and a surviving path selected. The VA is usually implemented in one of two methods, either *register-exchange mode* or *trace-back mode*.

The Viterbi algorithm is mostly easily explained with the aid of an example decoding.

Example 4.3 Viterbi decoding of a $(2, 1, 3)$ convolutional code.

An encoder for this code is given in Figure 2.2. The trellis (see Figure 4.7) contains four states at each level. Suppose the uncoded data sequence was

$$\mathbf{u} = [\ldots, 0, 1, 0, 0, 1, 1, 1, 0, \ldots] \tag{4.58}$$

and that the encoder shown in Figure 2.2 was in state S_3. The encoded sequence was thus

$$\mathbf{v} = [\ldots, 01, 00, 10, 11, 11, 01, 10, 01, \ldots] \tag{4.59}$$

After transmission over a discrete symmetric channel as shown in Figure 4.6 the receiver assigns one of four values to each received symbol. The ‘$\underline{0}$’ and ‘$\underline{1}$’ indicate the reception of a good signal, while ‘0’ and ‘1’ indicate reception of a weaker signal. For the channel shown in Figure 4.6 log likelihood functions [Wicker, 1994, p. 294] are used to compute the set of bit metrics used in the decoding process. The bit metrics are given in Table 4.1. Assume that the received sequence is

$$\mathbf{r} = [\ldots, 0\underline{1}, \overline{1}0, 1\underline{0}, \underline{11}, \overline{\underline{0}}\underline{1}, 0\underline{1}, 1\underline{0}, \underline{01}, \ldots] \tag{4.60}$$

In (4.60) overlining is used to highlight which symbols are in error.

		received symbol			
		$\underline{0}$	0	1	$\underline{1}$
required	0	5	4	2	0
symbol	1	0	2	4	5

Table 4.1: Channel metrics for the Viterbi decoding example.

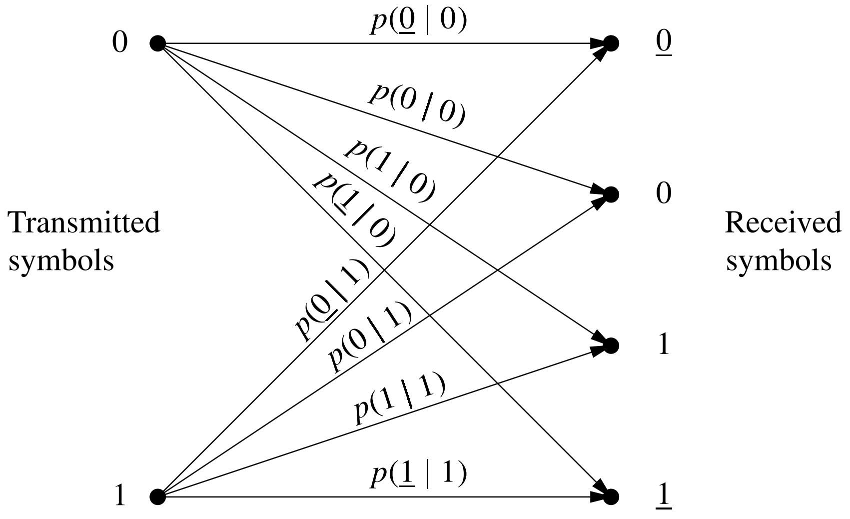


Figure 4.6: A discrete symmetric channel model.

Figure 4.7 shows the (truncated) trellis diagram for the code $(2, 1, 3)$. At each level there are $2^{K-1} = 4$ states. Each state has two branches leading in and out. The branches are labelled with their data and code symbols (*data/code*). The value in parentheses is the SD metric for that particular branch. The value above or below each node is the state metric, which is a measure of the likelihood of any state being part of the transmitted sequence. The state metrics can be found recursively from the sum of an input branch and its preceding state metric. In this Example the metrics are a measure of similarity, therefore the best metric is the largest one.

Trellis decoding starts at state $S_{0,1}$, that is state 0 at time $t = 1$. The best path back to $t = 0$ is $\mathcal{P}(S_{0,1} \to S_{2,0})$ and is indicated by a solid black line. The decoding metric for state $S_{0,1}$ is 8. This process is repeated for all other states at time $t = 1$. Moving onto $t = 2$, state $S_{0,2}$ has a choice of two paths, $\mathcal{P}(S_{0,2} \to S_{2,1} \to S_{3,0})$ or $\mathcal{P}(S_{0,2} \to S_{0,1} \to S_{2,0})$ with metrics of 17 and 14 respectively. The best path is

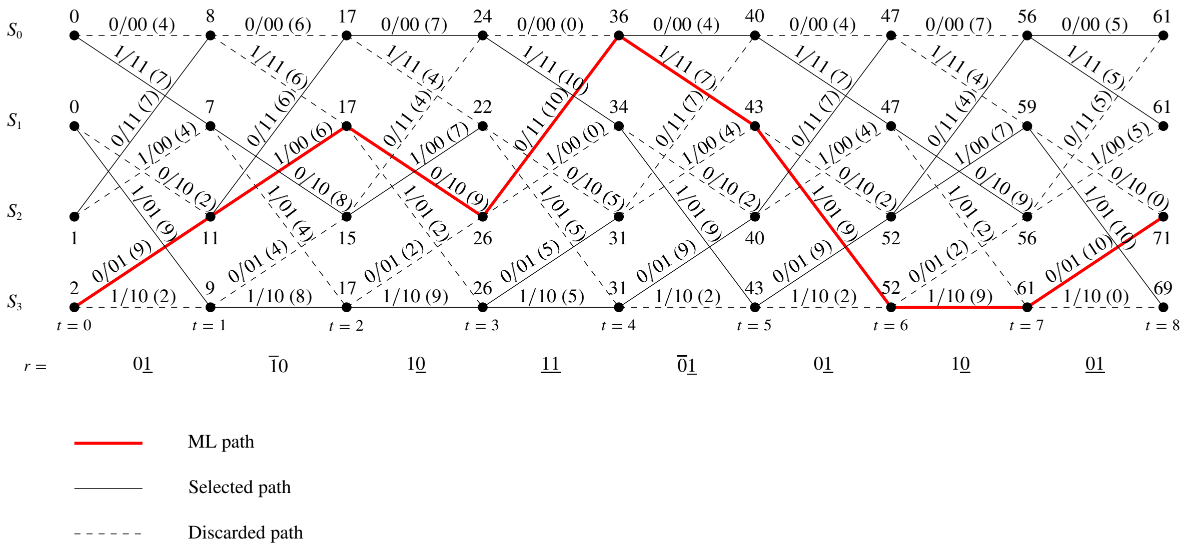


Figure 4.7: Soft-decision Viterbi decoding example.

$\mathcal{P}(S_{0,2} \rightarrow S_{2,1} \rightarrow S_{3,0})$. The process is repeated up to $t = 8$. If any two paths have the same metric one is chosen arbitrarily. Note that if each state stores its own metric it is never necessary to trace back more than one level.

The ML path, denoted by thick, solid lines, can be seen by starting at the state at time $t = 8$ with the best metric (i.e., $S_{2,8}$) and tracing back along the best path. The data symbol output from the decoder is the data label on the earliest branch of the ML path, i.e., $B(S_{2,1} \rightarrow S_{3,0})$. Therefore the decoder output is '0', and is in agreement with the first data symbol of $\mathbf{x}$. Subsequent decoding attempts will output more recent symbols in the trellis. The trace-back can be avoided by keeping track of the output label each state would give if it is on the ML path.

It is important that the path metrics for the states at time $t = 1$ are not lost. These states will become the earliest states in the next decoding attempt. Only the relative difference in metrics is important, a feature which can be exploited to avoid numeric overflow. Therefore on the next decoding attempt the state metrics for $S_{0,0}$, $S_{1,0}$, $S_{2,0}$ and $S_{3,0}$ will be 1, 0, 4 and 2 respectively.

Note that the surviving path from each state at time $t = 8$ has merged with the ML path by $t = 1$. Therefore, regardless of which node had been chosen the correct data for $t = 0$ would have resulted. This indicates that the truncated trellis was (just!) long enough so that all possible paths had merged with the ML path. In practice a trellis of depth 8 for a code of constraint length 3 is not sufficiently long to reliably ensure all possible paths merge with the ML path. A flowchart showing the decoding stages is given in Figure 4.8. □

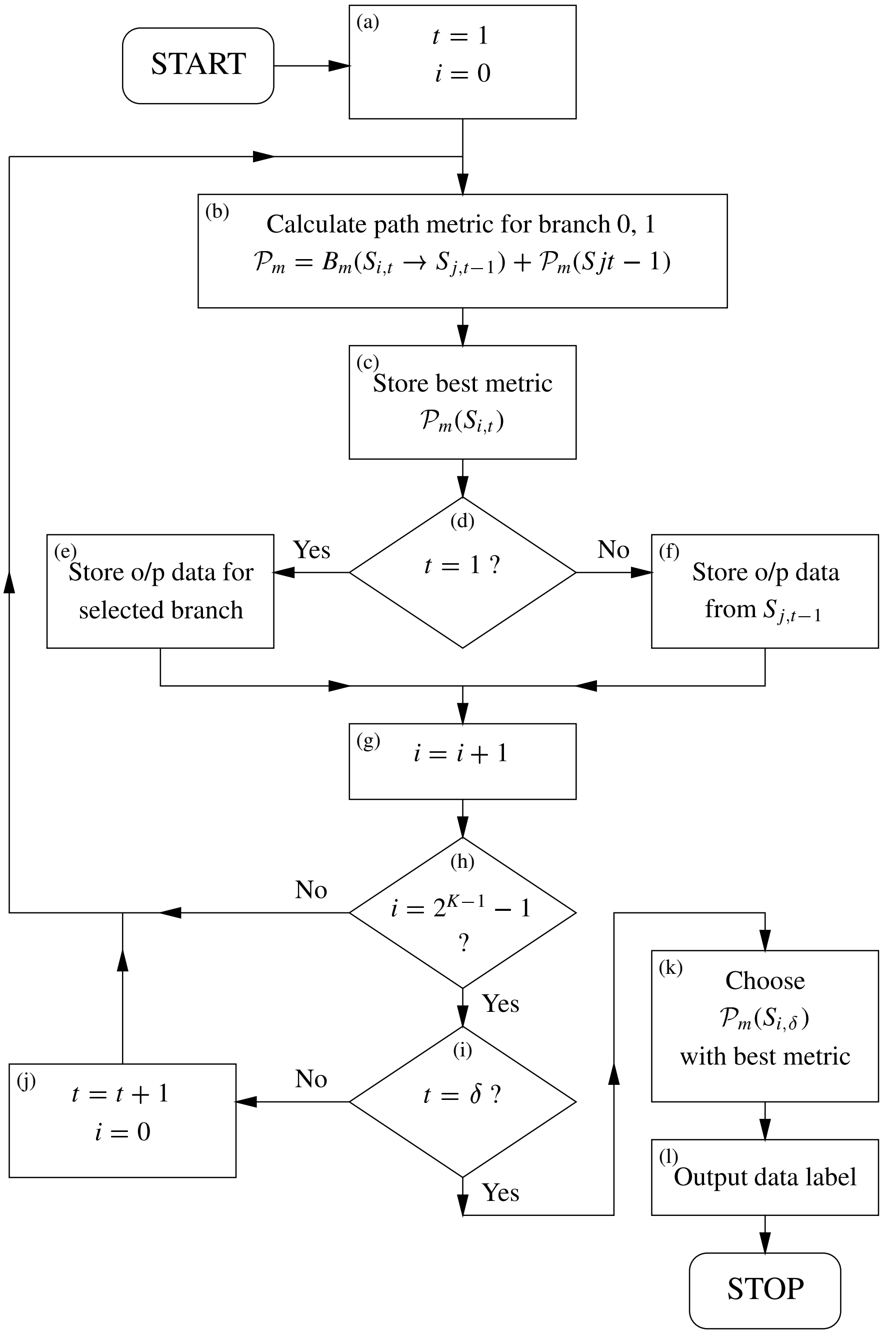


Figure 4.8: Flowchart of the Viterbi decoding algorithm.

4.2.3 Two-stage Trellis Decoding

It has been shown [Omura, 1969] that SDMLD of RS codes can be achieved by use of the Viterbi algorithm [Viterbi, 1967] over a suitable trellis. For short RS codes Viterbi decoding provides a practical and optimum (SDMLD) error-correcting performance. The designed coset trellises are isomorphic to the minimal trellis, and are thus themselves minimal (Section 4.1.4). However, for long RS codes Viterbi decoding becomes infeasible due to its considerable decoding complexity and storage requirements. It is therefore necessary to use a different decoding method. The algebraic techniques described in Chapter 3 are well-known but unlike trellis decoding are unable to take advantage of any SD information the channel may provide.

The Shannon product of trellises (Section 4.1.2) indicates a manner in which a trellis for a code with an inherent sum structure may be decomposed into its component trellises, T' and T''. It should be noted that the technique is not constrained to a maximum of two component trellises. If the decoding complexity of the component trellises T' and T'' is κ' and κ'' respectively, then the decoding complexity of the trellis T is (approximately) $\kappa'\kappa''$. However, if the decoding operation can be performed on the two component trellises the complexity is reduced to $\approx \kappa' + \kappa''$. The storage requirement is reduced in much the same way. Hence both major hurdles to trellis decoding for long RS codes have been reduced.

The decoding procedure consists of two major steps:

1. Identify in which subtrellis the maximum-likelihood path lies.

2. Apply the Viterbi decoding algorithm only to the subtrellis indicated at step 1.

If the overall trellis is viewed as the Shannon product of two trellises, T' and T'', with corresponding codes $\mathcal{C}'$ and $\mathcal{C}''$, then codewords from $\mathcal{C}''$ can be viewed as coset leaders which generate the cosets of $\mathcal{C}$.

Two-stage decoding is a type of reduced search Viterbi algorithm. However, unlike most reduced search algorithms the paths to be decoded are decided *before* trellis decoding (proper) begins, i.e., at the end of stage one, when the most likely subtrellis(es) have been identified. The usual behaviour of reduced search algorithms (e.g., [Shin and Sweeney, 1994] or [Aguado and Farrell, 1998]) is to select the candidate paths as trellis decoding progresses.

4.2.4 Two-stage Decoding of Reed-Muller Codes

Reed-Muller codes are highly regular, a feature which can be used to good effect in their construction [Wilson, 1996, p. 429]. Two-stage decoding of RM codes is also able to make good use of their regular structure, and was first employed by Wu *et al.* who found the decoding performance was only 0.2–0.5 dB away from SDMLD [Wu *et al.*, 1994]. For two-stage decoding of Reed-Muller codes a trellis can be used to identify which subtrellis(es) to decode. However, RM codes have few subtrellises and in many instances it is more efficient to select the subtrellis(es) by algebraic means, (using 'soft' Galois field algebra (Section 4.3) if soft information is available). A simple example is presented to demonstrate the TSD technique.

Example 4.4 Two-stage decoding of the RM$(8, 4, 4)$ code.

Consider the code generated from a generalised array code (Section 2.1.5). The construction of the code can be found in [Honary *et al.*, 1995a]. The code $\mathcal{C}$ is the linear sum of two component array codes, $\mathcal{C}_1$ and $\mathcal{C}_2$, i.e., $\mathcal{C} = \mathcal{C}_1 + \mathcal{C}_2$ where

$$\mathcal{C}_1 = \begin{bmatrix} u_1 & p_1 \\ u_2 & p_2 \\ u_3 & p_3 \\ u_4 & p_4 \end{bmatrix} \tag{4.61}$$

$$\mathcal{C}_2 = \begin{bmatrix} 0 & u_4 \\ 0 & u_4 \\ 0 & u_4 \\ 0 & u_4 \end{bmatrix} \tag{4.62}$$

$$\mathcal{C} = \begin{bmatrix} u_1 & u_1 \oplus u_4 \\ u_2 & u_2 \oplus u_4 \\ u_3 & u_3 \oplus u_4 \\ p_4 & p_4 \oplus u_4 \end{bmatrix} = \begin{bmatrix} v_1 & v_2 \\ v_3 & v_4 \\ v_5 & v_6 \\ v_7 & v_8 \end{bmatrix} \tag{4.63}$$

$$\begin{aligned} \text{where} \quad p_j &= u_j \\ j &= \{1, 2, 3\} \\ p_4 &= u_1 + u_2 + u_3 \end{aligned} \tag{4.64}$$

Addition over GF(2) is denoted by '$\oplus$'. The trellis for this code is shown in Figure 4.9. From 4.63 it can be seen that the cosets of the RM code are generated by $\mathcal{C}_1$ and the coset leaders are generated by $\mathcal{C}_2$.

The first stage of the decoding process is to identify in which subtrellis the codeword lies. This is achieved by decoding $\mathcal{C}_2$ to find the value of u_4. At this stage the values of the other data symbols, u_1, u_2 and u_3 are not known. However four independent predictions of the value of u_4 can be made from the four rows in $\mathcal{C}$. If $u_4 = 0$ then the left and right columns of a row should have the same value, and if $u_4 = 1$ then the columns should have opposite values. This can be shown by rearranging (4.63).

$$\widehat{u}_4 = \left\{ \begin{array}{c} (v_1 \oplus v_2), \\ (v_3 \oplus v_4), \\ (v_5 \oplus v_6), \\ (v_7 \oplus v_8) \end{array} \right\} \tag{4.65}$$

where $\widehat{u}_4$ is the set of symbol predictors for u_4. If hard-decision values of the received symbols $v_1, v_2, \ldots, v_8$ were used $\widehat{u}_4$ should be evaluated with a majority-vote.[1] If soft-decision information is available a better method is to use 'soft' Galois field arithmetic (Section 4.3) to preserve as much information as possible. Having found u_4 the corresponding subtrellis of Figure 4.9 (top for $u_4 = 0$, bottom for $u_4 = 1$) is decoded, using the Viterbi algorithm. □

4.2.5 Two-stage Decoding of Reed-Solomon Codes

Although the coset leader is the direct output of the T'' trellis it is not possible in the case of RS codes to simply pass the received codeword through T''; the received codeword contains encoded information from T' which appears as errors to T''. Therefore

[1]For a result of 2 : 2 an arbitrary decision must be made.

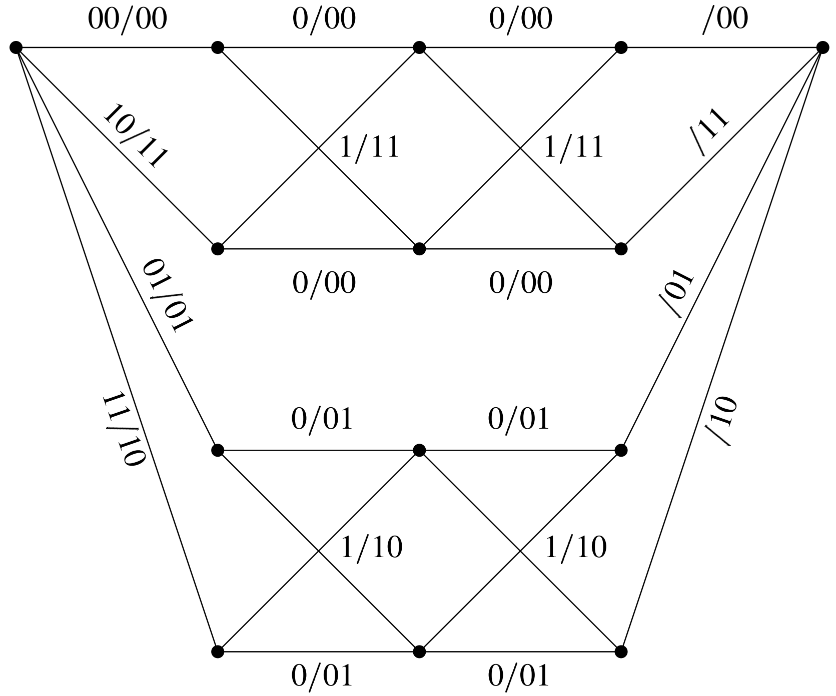


Figure 4.9: Trellis for RM(8, 4, 4).

an algebraic method is used to predict which subtrellis to decode. The subtrellis prediction can be improved with the inclusion of SD information, such as by using 'soft' Galois field arithmetic (Section 4.3).

Stage 1—To Find the Most Likely Subtrellis

To decode $\mathcal{C}''$ each information symbol, u_j (where $j = 1, 2, \ldots, k$), in $\mathcal{C}''$ is predicted independently by a set of *symbol predictors*, $\widehat{u}_j$. In general k symbols are required to predict u_j since there are k unknowns (the k information symbols). Any k symbols can be used as RS codes are invertible (Section 2.1.8).

Form a unique set $\mathbf{S}$ from (almost) any k out of n symbols. There are $\binom{n}{k}$ sets. For the set $\mathbf{S} = \{v_1, v_2, \ldots, v_k\}$ take a weighted sum of every symbol in the set.

$$c_{1_j}v_1 + c_{2_j}v_2 + \ldots + c_{k_j}v_k = f_{1_j}u_1 + f_{2_j}u_2 + \ldots + f_{k_j}u_k \tag{4.66}$$

To find the coefficients $c_{1_j}, c_{2_j}, \ldots, c_{k_j}$ note that

$$f_{i_j} = \begin{cases} 0 & \text{for } i \neq j \\ 1 & \text{for } i = j \end{cases} \tag{4.67}$$

From the generator matrix $\mathbf{G}$ form a $k \times k$ matrix, $\boldsymbol{\beta}$, from the k columns which relate to the encoded symbols in set $\mathbf{S}$. That is, for symbols $\{s_1, s_2, \ldots, s_k\}$ form $\boldsymbol{\beta}$ from columns $1,\ 2,\ \ldots,\ k$ of $\mathbf{G}$.

$$\begin{bmatrix} \beta_{11} & \beta_{12} & \ldots & \beta_{1k} \\ \beta_{21} & \beta_{22} & \ldots & \beta_{2k} \\ \vdots & & \ddots & \\ \beta_{k1} & \beta_{k2} & & \beta_{kk} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_k \end{bmatrix} = \begin{bmatrix} f_{1_j} \\ f_{2_j} \\ \vdots \\ f_{k_j} \end{bmatrix} \tag{4.68}$$

Solve (4.68) by Gaussian elimination.

Earlier it was stated that almost any k symbols could be used. It is important that only the *minimum set* of received symbols is used to predict a given information symbol. A minimum set, $\mathbf{S}_{\min}$, is defined such that no subset of $\mathbf{S}_{\min}$ exists from which it is possible to predict u_j. The inclusion of unnecessary symbols degrades the quality of the prediction, since for small P_s the probability of a prediction using an incorrect symbol is proportional to both P_s and the number of symbols used. It should be noted that the calculation of the coefficients $c_{1_j},\ c_{2_j},\ \ldots,\ c_{k_j}$ (Equation 4.68) is a design operation, while the evaluation of the weighted sums (4.66) is a decoding operation. From the requirement of minimum sets it can be shown that the number of predictions, N_{u_j}, available for an information symbol u_j, $j = 1,\ 2,\ \ldots,\ k$ is given by

$$N_{u_j} = \begin{cases} \binom{n-1}{k} + 1 & \text{where } \begin{cases} j = 1 \\ j = k \end{cases} \\ \binom{n}{k}\left[1 - \prod_{i=0}^{j-1} \frac{k-i}{n-1} - \prod_{i=0}^{k-j} \frac{k-i}{n-i}\right] + 2 & \text{where } j = 2,\ 3,\ \ldots,\ k-1 \end{cases} \tag{4.69}$$

The number of predictions of u_j is maximised when $j = (k+1)/2$. Thus performance is maximised by ensuring $\mathcal{C}''$ is composed from the most central $(k-1)/2$ component codes, i.e.,

$$\mathcal{C}' = \begin{cases} \mathcal{C}_1 + \mathcal{C}_2 + \ldots + \mathcal{C}_{\frac{k+3}{4}} & \\ \quad + \mathcal{C}_{\frac{3k+5}{4}} + \mathcal{C}_{\frac{3k+9}{4}} + \ldots + \mathcal{C}_k & \text{where } \frac{k+1}{2} \text{ odd} \\ \mathcal{C}_1 + \mathcal{C}_2 + \ldots + \mathcal{C}_{\frac{k+1}{4}} & \\ \quad + \mathcal{C}_{\frac{3k+3}{4}} + \mathcal{C}_{\frac{3k+7}{4}} + \ldots + \mathcal{C}_k & \text{where } \frac{k+1}{2} \text{ even} \end{cases} \tag{4.70}$$

$$\mathcal{C}'' = \begin{cases} \mathcal{C}_{\frac{k+7}{4}} + \mathcal{C}_{\frac{k+11}{4}} + \ldots + \mathcal{C}_{\frac{3k+1}{4}} & \text{where } \frac{k+1}{2} \text{ is odd} \\ \mathcal{C}_{\frac{k+5}{4}} + \mathcal{C}_{\frac{k+9}{4}} + \ldots + \mathcal{C}_{\frac{3k-1}{4}} & \text{where } \frac{k+1}{2} \text{ is even} \end{cases} \tag{4.71}$$

Stage 2—Finding the ML Path Through a Given Subtrellis

Having identified the subtrellis to decode it can be generated, on-the-fly if necessary, by adding the appropriate codeword from $\mathcal{C}''$ (i.e., coset leader of $\mathcal{C}$) to the coset of $\mathcal{C}$ containing the all-zeros codeword. The subtrellis is then decoded with the Viterbi algorithm. A substantial improvement in the performance of two-stage decoding can be obtained by decoding more than one subtrellis. The subtrellises decoded are chosen on the basis of the highest confidences from the output of stage one. The final output is the one with highest confidence from the output of stage two.

Example 4.5 Design the symbol predictors for two-stage decoding of the RS$(7,3,5)$

code.

Let the trellis be designed according to Example 4.2. Thus the generator matrix is

$$\mathbf{G} = \begin{bmatrix} g_{T_1} \\ g_{T_2} \\ g_{T_3} \end{bmatrix} = \left[\begin{array}{cc:ccc:cc} \alpha^3 & \alpha & 1 & \alpha^3 & 1 & 0 & 0 \\ 0 & \alpha^3 & \alpha & 1 & \alpha^3 & 1 & 0 \\ 0 & 0 & \alpha^3 & \alpha & 1 & \alpha^3 & 1 \end{array}\right] \tag{4.72}$$

The component trellises are shown in Figure 4.4, and the complete trellis in Figure 4.5. From (4.70) and (4.71) it is apparent that the selection of the subtrellises should be based upon the 'central' code (i.e., $\mathcal{C}_2$) to maximise the number of predictions available (and thus ensure the best possible performance). Although it is possible to use isomorphic trellises where the subtrellis decision is based upon $\mathcal{C}_1$ or $\mathcal{C}_3$ this will result in less subtrellis predictions.

Calculation of the Symbol Predictors

The symbols of $\mathcal{C}_2$ are dependent upon u_2 alone, therefore only a single symbol predictor, $\widehat{u}_2$, is required. Whilst in general the minimum number of received symbols required to predict $\mathcal{C}_2$ is $k = 3$ there exists two minimum sets requiring only 2 received symbols, $\{v_1, v_2\}$ and $\{v_6, v_7\}$.

For the minimum set $\mathbf{S} = \{v_1, v_2\}$ the calculation of the symbol predictor proceeds as follows. The weighted sum of the received symbols is (from Equation 4.66)

$$\begin{aligned} c_1 v_1 + c_2 v_2 &= f_1 u_1 + f_2 u_2 + f_3 u_3 \\ &= u_2 \end{aligned} \tag{4.73}$$

The coefficients c_1 and c_2 can be found by taking columns 1 and 2 from the generator matrix (4.72), as in accordance with (4.68). It follows that

$$\begin{bmatrix} \alpha^3 & \alpha \\ 0 & \alpha^3 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \tag{4.74}$$

Solving (4.74) by Gaussian elimination gives

$$c_1 = \alpha^2 \tag{4.75}$$

$$c_2 = \alpha^4 \tag{4.76}$$

Therefore the first independent calculation for the value of u_2 is given by $\alpha^2 v_1 + \alpha^4 v_2$. However, the transmitted symbols $\{v_1,\ v_2,\ \ldots,\ v_n\}$ are not known by the receiver. Instead the received symbols $\{r_1,\ r_2,\ \ldots,\ r_n\}$ must be used. As they may be subject to errors the calculation is weakened to a *prediction* of the value of u_2. However as 27 independent predictions are available the overall prediction is much less sensitive to errors. The full set of 27 symbol predictors are given in Table 4.2. Having calculated the symbol predictors all the design stages are complete.

□

Example 4.6 Decode a received codeword of RS$(7, 3, 5)$ using two-stage decoding.

Consider that the dataword, $\mathbf{u} = [\ 0 \quad 0 \quad 0\ ]$, was transmitted. The transmitted

Number	Symbol predictor
1	$\alpha^2 r_1 + \alpha^4 r_2$
2	$r_6 + \alpha^3 r_7$
3	$\alpha^4 r_1 + \alpha^3 r_3 + \alpha^5 r_4$
4	$r_1 + \alpha^2 r_3 + \alpha^5 r_5$
5	$\alpha r_1 + \alpha^4 r_3 + \alpha^4 r_6$
6	$\alpha^3 r_1 + \alpha^6 r_3 + \alpha^2 r_7$
7	$\alpha^6 r_1 + \alpha^2 r_4 + \alpha^3 r_5$
8	$\alpha^3 r_1 + \alpha^3 r_4 + \alpha r_6$
9	$r_1 + r_4 + \alpha r_7$
10	$\alpha^5 r_1 + \alpha r_5 + \alpha^5 r_6$
11	$\alpha r_1 + \alpha^4 r_5 + \alpha^4 r_7$
12	$r_2 + \alpha^4 r_3 + \alpha^6 r_4$
13	$\alpha^5 r_2 + \alpha^5 r_3 + \alpha r_5$
14	$\alpha r_2 + \alpha^2 r_3 + \alpha^2 r_6$
15	$\alpha^2 r_2 + \alpha^3 r_3 + \alpha^6 r_7$
16	$\alpha^3 r_2 + \alpha^4 r_4 + \alpha^5 r_5$
17	$\alpha^2 r_2 + r_4 + \alpha^5 r_6$
18	$\alpha^5 r_2 + \alpha^3 r_4 + \alpha^4 r_7$
19	$\alpha^6 r_2 + r_5 + \alpha^4 r_6$
20	$\alpha r_2 + \alpha^2 r_5 + \alpha^2 r_7$
21	$\alpha^6 r_3 + r_4 + \alpha^4 r_5$
22	$r_3 + \alpha^4 r_4 + \alpha^6 r_6$
23	$\alpha^5 r_3 + \alpha^2 r_4 + r_7$
24	$\alpha^3 r_3 + \alpha^3 r_5 + \alpha r_6$
25	$r_3 + r_5 + \alpha r_7$
26	$\alpha^6 r_4 + \alpha^2 r_5 + \alpha^3 r_6$
27	$\alpha^5 r_4 + \alpha r_5 + \alpha^5 r_7$

Table 4.2: Symbol predictors for Two-stage decoding of $\mathrm{RS}(7, 3, 5)$.

codeword is given by

$$\begin{aligned}
\mathbf{v} &= \mathbf{u}.\mathbf{G} \\
&= [\ 0\ \ 0\ \ 0\ ] \begin{bmatrix} \alpha^3 & \alpha & 1 & \alpha^3 & 1 & 0 & 0 \\ 0 & \alpha^3 & \alpha & 1 & \alpha^3 & 1 & 0 \\ 0 & 0 & \alpha^3 & \alpha & 1 & \alpha^3 & 1 \end{bmatrix} \\
&= [\ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ ]
\end{aligned} \tag{4.77}$$

For simplicity consider only hard-decision decoding of $\mathcal{C}_2$. Let the received codeword be $\mathbf{r} = \mathbf{v} + \mathbf{e}$, where $\mathbf{e} = [\ 0 \quad \alpha^6 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0\ ]$.

$$\begin{aligned}
\mathbf{r} &= [\ r_1 \quad r_2 \quad r_3 \quad r_4 \quad r_5 \quad r_6 \quad r_7\ ] \\
&= [\ 0 \quad \alpha^6 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0\ ]
\end{aligned} \tag{4.78}$$

The prediction of the value of u_2, and thus of which subtrellis to decode, is obtained by substituting (4.78) into the symbol predictors (Table 4.2) and choosing the most likely value. The results of evaluating the symbol predictors are given in Table 4.3. It can be seen that the most likely value of u_2 is zero, and therefore the subtrellis to decode is the one generated by $u_2 = 0$. If multiple subtrellises are to be decoded, then the subtrellises generated by $u_2 = 1$, $u_2 = \alpha$ and $u_2 = \alpha^4$ are the next most likely candidates. The chosen subtrellis(es) are decoded with normal Viterbi decoding (Section 4.2.2).

□

Value	Votes cast	Symbol predictor number(s)
0	17	{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 21, 22, 23, 24, 25, 26, 27}
1	2	{14, 20}
α	2	{15, 17}
α^2	1	{16}
α^3	1	{1}
α^4	2	{13, 18}
α^5	1	{19}
α^6	1	{12}

Table 4.3: Symbol predictor results.

4.3 'Soft' Galois Field Arithmetic

Reference has been made to 'soft' Galois field arithmetic. In this Section an explanation is given as to how soft GF arithmetic may be implemented.

Traditional GF arithmetic operates on GF variables whose value must be defined precisely. This is often accomplished by using a HD output from the demodulator. In reaching a precise definition the useful soft information is discarded. By including the soft information better error-correction can be achieved.

For the binary[2] GF(q) operations $\oplus$, $\ominus$ and $\otimes$ all q^2 combinations must be evaluated. There are q different results $(0,\ 1,\ \alpha,\ \ldots\ ,\ \alpha^{q-2})$ and each result occurs q times. Therefore, each of the q output values has an associated probability which is the sum of q probabilities. A similar approach can also be used for $\oslash$, although there is also an error state due to division by zero.

[2] i.e., taking two arguments.

Example 4.7 The addition of two GF(4) values using soft GF arithmetic.

For the elements $\{0, 1, \alpha, \alpha^2\}$ let the confidences of a be $\{0.75, 0.10, 0.10, 0.05\}$ respectively, and for b $\{0.25, 0.25, 0.50, 0.00\}$ respectively. Consider the result $a \oplus b = 0$. There are four ways by which this outcome may be achieved: $0 \oplus 0$, $1 \oplus 1$, $\alpha \oplus \alpha$ and $\alpha^2 \oplus \alpha^2$. The probability that $a \oplus b = 0$ is given by

$$\begin{aligned} p(a \oplus b = 0) &= p(a = 0,\ b = 0) + p(a = 1,\ b = 1) \\ &\quad + p(a = \alpha,\ b = \alpha) + p(a = \alpha^2,\ b = \alpha^2) \\ &= p(a = 0).p(b = 0) + p(a = 1).p(b = 1) \\ &\quad + p(a = \alpha).p(b = \alpha) + p(a = \alpha^2).p(b = \alpha^2) \end{aligned} \tag{4.79}$$

Table 4.4 illustrates how the output confidences for all outcomes are computed. The sum of the output probabilities is one. □

Element	Confidence				
	Computation				Total
0	0.1875 $(0 \oplus 0)$	0.0250 $(1 \oplus 1)$	0.0500 $(\alpha \oplus \alpha)$	0.0000 $(\alpha^2 \oplus \alpha^2)$	0.2625
1	0.1875 $(0 \oplus 1)$	0.0250 $(1 \oplus 0)$	0.0000 $(\alpha \oplus \alpha^2)$	0.0250 $(\alpha^2 \oplus \alpha)$	0.2375
α	0.3750 $(0 \oplus \alpha)$	0.0250 $(\alpha \oplus 0)$	0.0000 $(1 \oplus \alpha^2)$	0.0125 $(\alpha^2 \oplus 1)$	0.4125
α^2	0.0000 $(0 \oplus \alpha^2)$	0.0125 $(\alpha^2 \oplus 0)$	0.0500 $(1 \oplus \alpha)$	0.0250 $(\alpha \oplus 1)$	0.0875

Table 4.4: Soft GF arithmetic for GF(4).

4.4 Discussion

In this Chapter techniques for constructing minimal trellises have been demonstrated. For low-rate codes this is served by the coset trellises, while for high-rate codes syndrome trellises should be used. The Shannon product of trellises is important not only for trellis construction but also for the decomposition of trellises into simpler forms. Two-stage decoding of RS codes is a new method which can take advantage of simpler, regular trellises to provide low-complexity sub-optimum decoding. To improve the decoding performance of TSD a new procedure for including soft information in the evaluation of Galois field algebra was presented. The decoding performance and complexity of both Viterbi and two-stage decoding has been measured by computer simulation. Results are presented in Chapter 6.

Chapter 5

Improved Decoding of Concatenated Codes

Chapter 5

Improved Decoding of Concatenated Codes

5.1 Concatenated Codes

5.1.1 Introduction

Concatenated codes are frequently used to implement a low-complexity, low error-rate channel. The basic concepts of concatenated coding were introduced in Section 2.3. In such a system the inner decoder is able to take advantage of any SD information from the channel. For maximum performance the outer decoder requires SD information from the inner decoder. Traditional decoders are unable to fulfill this requirement.

A number of decoding algorithms which provide soft-decision outputs exist, e.g., SOVA [Hagenauer and Hoeher, 1989] and MAP [Bahl *et al.*, 1974]. SOVA was identified as a useful method as it is a generalisation of the Viterbi algorithm, the *de facto* trellis decoding algorithm. Only SOVA is considered in this work.

Massey stated that convolutional codes should be used as the first stage of decoding because they can easily accept soft decisions and channel state information [Massey, 1984]. Many concatenated coding schemes exist which do just that for the very reasons stated. However, a decoding failure in a convolutional code normally produces a burst of errors [Hagenauer *et al.*, 1994, p. 243]. Reed-Solomon codes are well known for their burst error correction capability when $\log_2 q_{\mathrm{RS}}$ binary bits are mapped into one RS symbol (Section 2.1.8). A concatenated system comprising a convolutional inner code and an RS outer code with an interleaver/de-interleaver operating on RS symbols provides very good performance and is used by NASA and ESA for space communications [Dai, 1995; Wicker, 1994]. While the binary to multi-level mapping provides burst error correction it is a mixed blessing, as the bit error probabilities must somehow be transformed into symbol error probabilities. For an RS trellis decoder this can be avoided if the trellis is labelled with binary bits.

Section 5.2 of this Chapter describes the soft output Viterbi algorithm. The Viterbi decoding example (4.3) is extended to incorporate SOVA, outputting reliability information in addition to the most likely symbol(s). Product codes can be viewed as a type of concatenated code, with the row and column codes forming the inner and outer codes. Section 5.3 describes various decoding algorithms for decoding product codes constructed with RS subcodes.

5.1.2 Calculation of the Log Likelihood Metrics

It can be shown [Gallager, 1968; Wozencraft and Jacobs, 1965] that for any channel if all input sequences are equally likely the decoder which minimises the error probability is one which compares the conditional probabilities (or likelihood functions), $p(\mathbf{r} \mid \mathbf{v})$, of the received sequence, $\mathbf{r}$, and all possible transmitted sequences, $\mathbf{v}$, and selects the maximum, $\mathbf{v}'$ [Viterbi, 1971]. Such a decoder is termed *maximum likelihood*. For

$$\mathbf{r} = r_t,\ r_{t+1},\ r_{t+2} \tag{5.1}$$

$$\mathbf{v}' = v'_t,\ v'_{t+1},\ v'_{t+2} \tag{5.2}$$

the decoder must calculate the probability

$$p(\mathbf{r} \mid \mathbf{v}') = p(r_t \mid v_t{}') \cdot p(r_{t+1} \mid v_{t+1}{}') \cdot p(r_{t+2} \mid v_{t+2}{}') \tag{5.3}$$

For most channels the inputs to the receiver are real values and thus require infinite precision. This is not possible and some loss of precision must be accepted by quantising the received signal to a finite number of values. Simulation studies [Heller and Jacobs, 1971] have shown that 8-level quantisation resulted in only 0.25 dB reduction in coding gain with respect to the unquantised case, much less than the gains made possible by using SD decoding, typically 2 dB [Hagenauer *et al.*, 1994; Marple, 1998]. See also Chapter 6.

The transition probabilities may then be computed by considering the area under the PDF for each quantisation level. In (5.3) it can be seen that the conditional probability that sequence $\mathbf{r}$ was received is dependent upon multiplication operations. For almost all implementations summations are preferred to multiplications. This can be achieved by using logarithms. Equation (5.3) may then be rewritten without multiplication as the *log likelihood function*

$$\log p(\mathbf{r} \mid \mathbf{v}') = \log p(r_t \mid v'_t) + \log p(r_{t+1} \mid v'_{t+1}) + \log p(r_{t+2} \mid v'_{t+2}) \tag{5.4}$$

Since $\log p(\mathbf{r} \mid \mathbf{v}')$ increases monotonically with $p(\mathbf{r} \mid \mathbf{v}')$ the decoder is able to maximise $\log p(\mathbf{r} \mid \mathbf{v}')$ instead of $p(\mathbf{r} \mid \mathbf{v}')$ with the same result. Logarithms of any base may be used, the only difference is a scaling factor. Unless stated otherwise natural logarithms are used.

Integer arithmetic is typically several times faster than floating point arithmetic and is often preferred for reasons of both speed and reduced complexity of the hardware required. The log likelihood functions can be transformed into integer *log likelihood metrics*, ℓ [Wicker, 1994, p. 294]

$$\ell(r_i \mid v'_i) = \langle a \left[\log p(r_i \mid v)_i) + b\right] \rangle \tag{5.5}$$

where a and b are real numbers chosen to scale the LL functions into a suitable range, and $\langle x \rangle$ denotes the closest integer to x. Rounding errors can be minimised by choosing appropriate values for a and b. When a is positive the decoder should select v'_i to maximise $\ell(r_i \mid v'_i)$ and when a is negative v'_i is selected to minimise $\ell(r_i \mid v'_i)$.

Example 5.1 Calculate the set of metrics to be used for a coherently-demodulated BPSK system over an AWGN channel operating at $E_b/N_0 = -3\,\text{dB}$ with 8 quantisation levels.[1]

Let the received bit energy in noiseless conditions be E_b and let the levels have an equal spacing of $\frac{1}{2}\sqrt{E_b}$. For simplicity assume unity bit energy; let a "0" be represented by -1 and a "1" by $+1$. Thus the 7 transition points are -1.5, -1.0, -0.5, 0.0, $+0.5$, $+1.0$ and $+1.5$. If "0" was transmitted it could be received in any one of the 8 levels, and the probability of each level being received is (generally) different and dependent upon E_b/N_0, the signalling scheme used and the noise PDF. Figure 5.1 shows the 7 transition levels, and the signalling values with the superimposed Gaussian PDF (2.36) at $E_b/N_0 = -3\,\text{dB}$. The area under the PDF is given by

$$A = \int \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right\} dx \tag{5.6}$$

Thus the probability of receiving "0" in any region is given by evaluating the integral in (5.6) between the limits of the quantisation level. For coherently demodulated BPSK over an AWGN channel the standard deviation, σ, is given by

$$\sigma = \sqrt{\frac{E_b}{2N_0}} \tag{5.7}$$

For $E_b/N_0 = -3\,\text{dB}$, $\sigma = 0.9988$. The logarithmic transition probabilities can be scaled into the range $0 \to 15$ by choosing $b = 5.08$ and $a = 15/(-1.18 + 5.08) =$

[1]A similar example appears in [Wilson, 1996, p. 307].

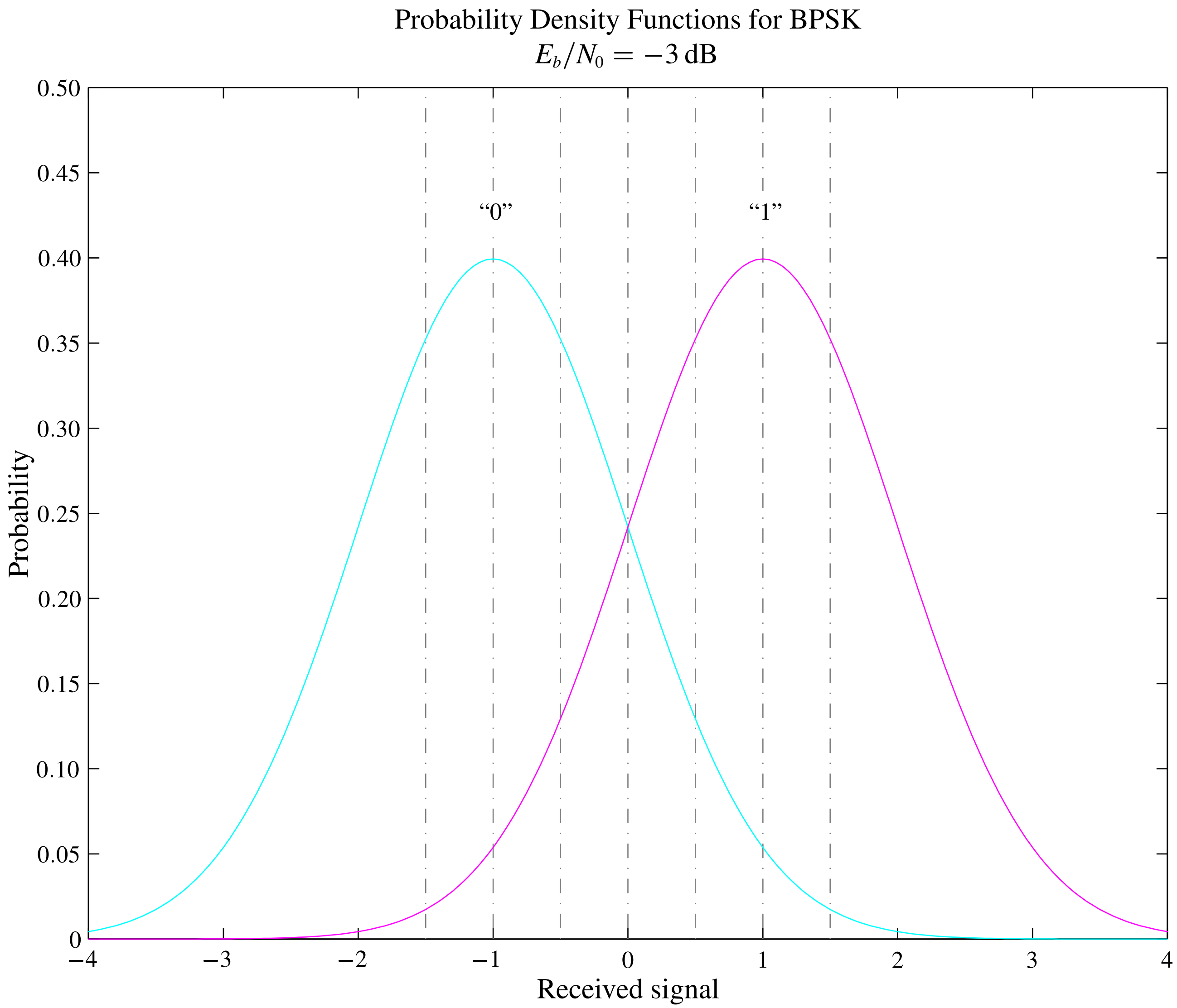


Figure 5.1: Signalling values with PDFs for $E_b/N_0 = -3$ dB superimposed.

3.8642. The transition probabilities and LL metrics for "1" are computed in the same way. The transition probabilities are given in Table 5.1, where the symmetry of the channel is reflected in the symmetry of the LL metrics. The metrics are shown graphically in Figure 5.2. In the area of indecision (received signal ≈ 0) the metrics are approximately equal while further from zero the metrics display an increasingly strong bias to "0" or "1". In Figure 5.1 it can be seen that there is a disproportionately large probability of receiving signals in the $-\infty \to -1.5$ and $+1.5 \to +\infty$ regions, which leads to the apparent discontinuities at the boundaries of Figure 5.2. This is due to the poor SNR of the channel as the probability of receiving a high confidence value is larger than over a good channel. □

Variation of the Log Likelihood Metrics with E_b/N_0

The LL metrics are dependent upon E_b/N_0, but the receiver can only estimate this ratio, its measurement is subject to error. It is therefore necessary to consider the sensitivity of the metrics with variation in E_b/N_0. Figure 5.3 shows the LL metrics for symbol "0" against E_b/N_0 over the range $-6\,\text{dB} \to +6\,\text{dB}$. The metrics are scaled to fit the range $0 \to 15$ using the method outlined above. The dotted lines show the metrics truncated to integer values as would be used in a hardware implementation of SOVA. For the region where the noise dominates the signal ($E_b/N_0 < 0\,\text{dB}$) the metrics are sensitive to changes in E_b/N_0. In such noisy conditions coding would not be used because uncoded operation results in fewer errors. For $E_b/N_0 > 2\,\text{dB}$ there is very little change in the scaled LL values, indeed, for the integer metrics there is no change. This shows that for the area of interest the metrics are not particularly

level	lower limit	upper limit	transition probability		log of transition probability		scaled LL value		LL metric	
			$p(r \mid 0)$	$p(r \mid 1)$	$\log p(r \mid 0)$	$\log p(r \mid 1)$	"0"	"1"	"0"	"1"
0	$-\infty$	-1.5	0.3083	0.0062	-1.18	-5.09	15.00	0.00	15	0
1	-1.5	-1.0	0.1917	0.0165	-1.65	-4.11	13.18	3.77	13	4
2	-1.0	-0.5	0.1917	0.0440	-1.65	-3.12	13.18	7.53	13	8
3	-0.5	$+0.0$	0.1500	0.0918	-1.90	-2.39	12.24	10.36	12	10
4	$+0.0$	$+0.5$	0.0918	0.1500	-2.39	-1.90	10.36	12.24	10	12
5	$+0.5$	$+1.0$	0.0440	0.1917	-3.12	-1.65	7.53	13.18	8	13
6	$+1.0$	$+1.5$	0.0165	0.1917	-4.11	-1.65	3.77	13.18	4	13
7	$+1.5$	$+\infty$	0.0062	0.3083	-5.09	-1.18	0.00	15.00	0	15

Table 5.1: Transition probabilities and LL metrics for a coherently-demodulated BPSK channel at $E_b/N_0 = 3$ dB.

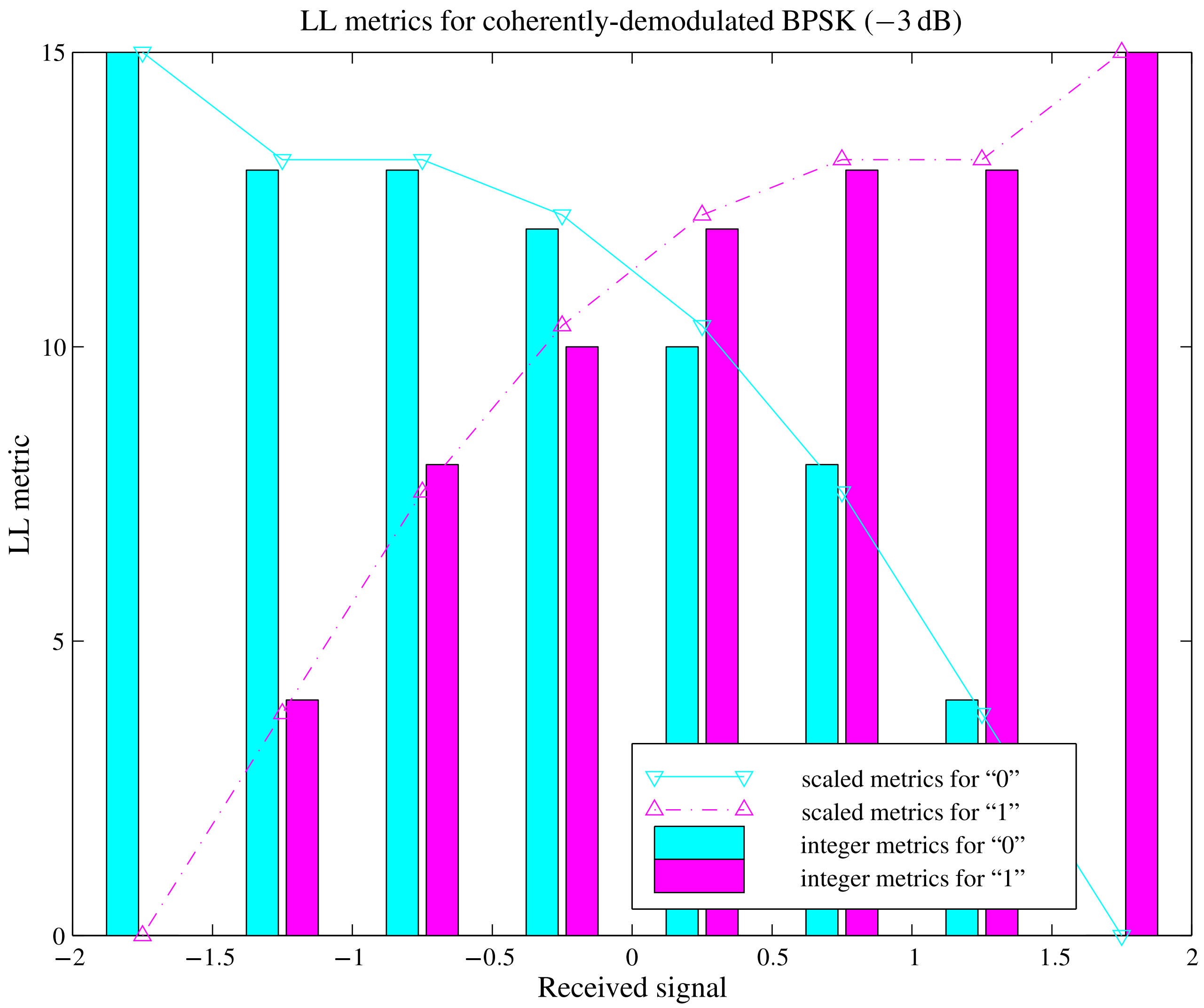


Figure 5.2: Scaled LL metrics for coherently-demodulated BPSK at $E_b/N_0 = -3$ dB.

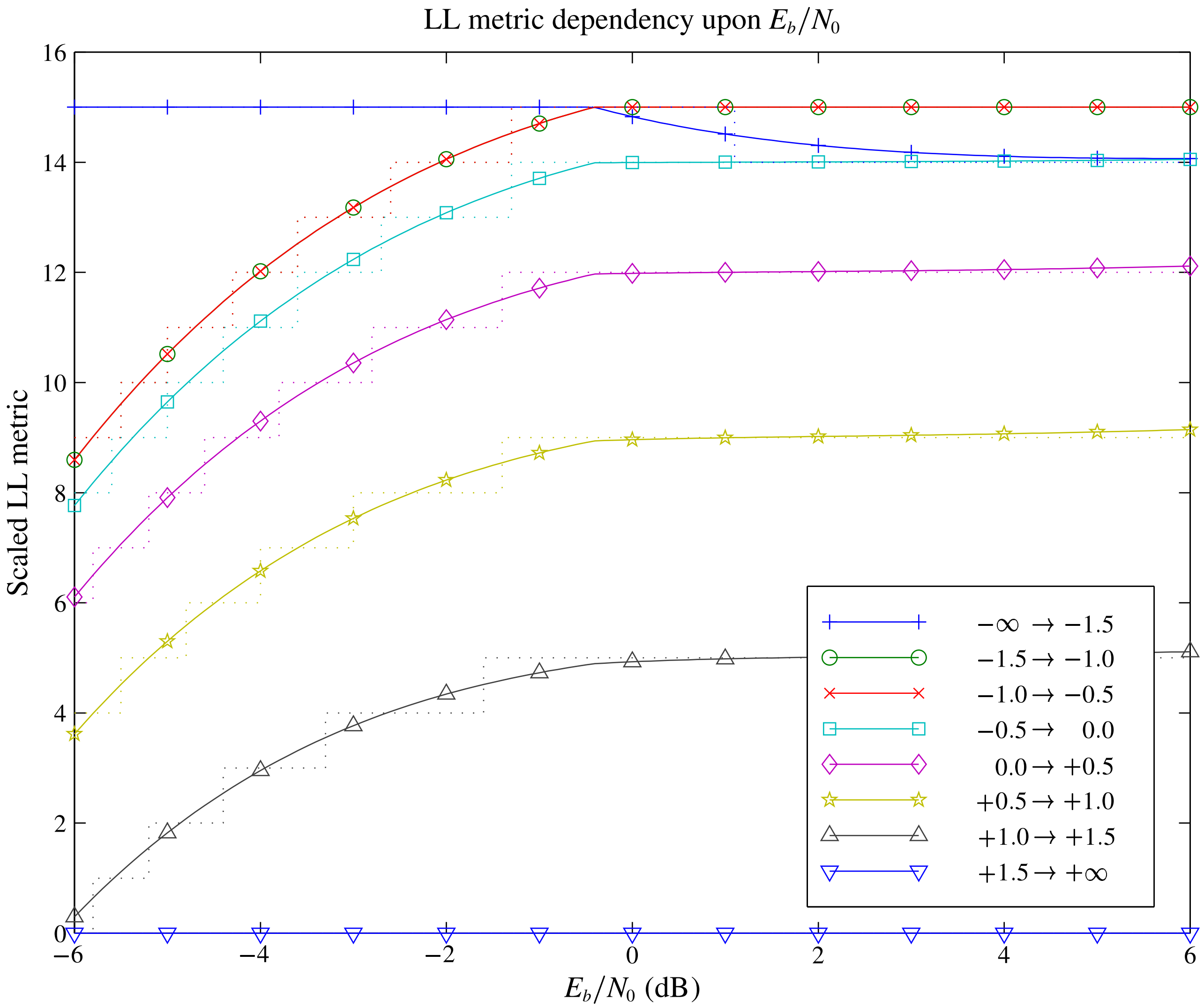


Figure 5.3: Variation of LL metrics for a coherently-demodulated BPSK channel.

sensitive to E_b/N_0 and may remain constant with no impact on performance.

5.2 Soft Output Viterbi Algorithm

5.2.1 Introduction

As its name suggests the soft output Viterbi algorithm [Hagenauer and Hoeher, 1989; Hagenauer *et al.*, 1994, 1996] is a modification to the 'standard' Viterbi algorithm [Viterbi, 1967]. It can decode using soft or hard decision information and provides a single reliability measure for its output sequence, which is the closest codeword. Alternatively, if the trellis is labelled with both data and code symbols then the output sequence can be the dataword corresponding to the closest codeword.

One of the main areas in which SOVA has been used is for iterative or turbo decoding.[2] An immense amount of literature has recently been written on turbo decoding, but the work involving SOVA described in this Chapter is intended for the purpose of concatenated decoding and therefore iterative methods have not been applied.

5.2.2 Differences Between the Standard and Soft-Output Viterbi Algorithms

The soft output Viterbi algorithm differs from the standard model by keeping track of the reliability of its decisions. However, only decisions which affect the outcome are considered, that is decisions which lie along the surviving path. Decisions which

[2] Turbo decoding is also known as turbo coding, this is misleading since the 'turbo' (feedback) analogy applies to the decoding, not the codes themselves [Hagenauer *et al.*, 1996].

affect other paths are discarded at the same time the path is discarded. Consider the trellis segment for a binary code shown in Figure 5.4. For each trellis state, $S_{j,t}$, the

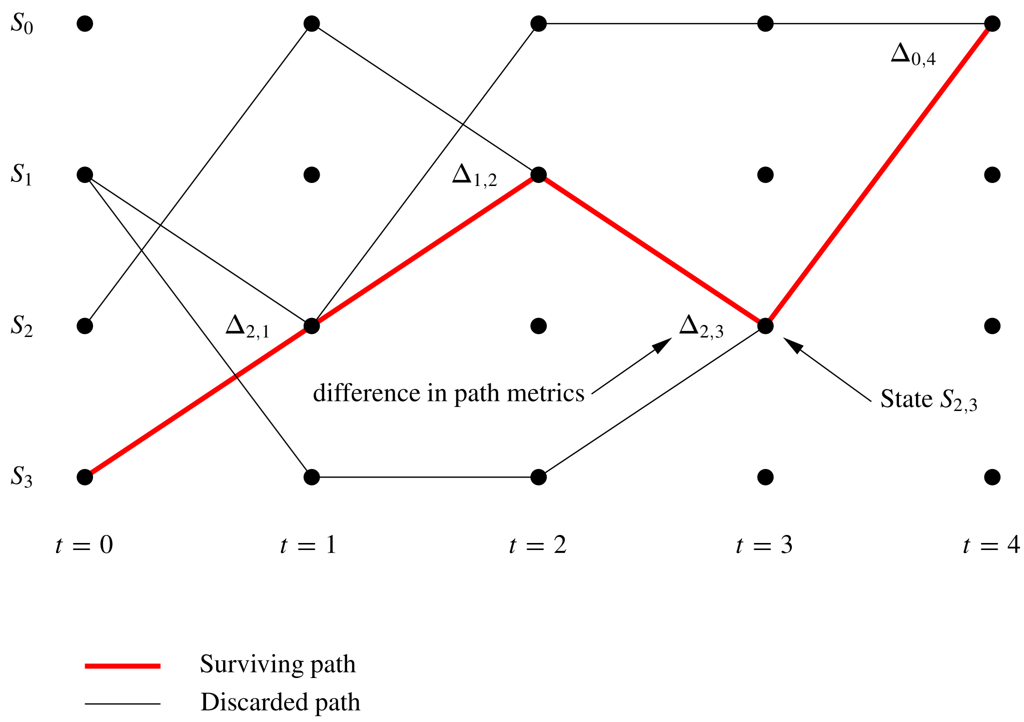


Figure 5.4: Example trellis with metric differences for traceback SOVA.

Viterbi algorithm chooses the branch $B(S_{i,t-1} \rightarrow S_{j,t})$ to select the best partial path metric, $\mathcal{P}_m(S_{i,t-1} + B_m(S_{i,t-1} \rightarrow S_{j,t}))$.

At time index t the partial paths $\mathcal{P} = \mathcal{P}(\ldots \rightarrow S_{i,t-1} \rightarrow S_{j,t})$ and $\mathcal{P}' = \mathcal{P}(\ldots \rightarrow S_{i',t-1} \rightarrow S_{j,t})$ merge, with metrics M and M' respectively. Let $\mathcal{P}$ be the surviving path. Define the metric difference as [Hagenauer *et al.*, 1996]

$$\Delta_{j,t} = M - M' \tag{5.8}$$

The probability that the decision made at this point was correct is given by [Hagenauer *et al.*, 1996]

$$p(\text{correct}) = \frac{p(\mathcal{P})}{p(\mathcal{P}) + p(\mathcal{P}')} \tag{5.9}$$

$$= \frac{e^{M}}{e^{M} + e^{M'}} \tag{5.10}$$

$$= \frac{e^{\Delta_{j,t}}}{1 + e^{\Delta_{j,t}}} \tag{5.11}$$

Therefore the log likelihood ratio of this binary path decision is $\Delta_{j,t}$ because

$$\log \frac{p(\text{correct})}{1 - p(\text{correct})} = \Delta_{j,t} \tag{5.12}$$

This shows that when two paths merge and either would give rise to the same output the LL reliability of the decision is ∞, since no mistake would be made at that point in the trellis. If the data output would not be the same then the reliability is given by the difference in the partial path metrics of the surviving and discarded paths. The reliability of the output sequence is given by product of the reliabilities for the decisions affecting the output (equivalent to the sum of log likelihood metrics). Hagenauer shows that the sum can be approximated to the smallest log likelihood decision reliability of the terms [Hagenauer, 1995]. While the channel information gives some indication as to the most likely transmitted sequence the additional reliability information gleaned from the decoding process is termed the *extrinsic information*.

It is important to note at this point that the behaviour of the algorithm differs between convolutional and block trellises. This is due to the differing ways in which each are decoded. A convolutionally-encoded data stream typically imposes an intolerable delay in the decoding process. For this reason a truncated trellis (typically 4 to 6 times the constraint length) is decoded once per k output symbols, whereas a block code trellis is decoded once for the entire codeword. This affects the algorithm in two ways. Firstly, for convolutional decoding independent passes through the trellis are made for each k output symbols. Usually $k = 1$ and thus each symbol has a unique metric. This cannot be done for the fixed length block code trellis. The ML sequence *is* the codeword, and thus all output symbols in the codeword must share the same reliability metric. While the MAP algorithm [Hagenauer *et al.*, 1996] is able to provide reliability metrics for each encoded symbol this is not helpful for concatenated schemes where metrics for the reliability of output *data* symbols is sought. (MAP is also much more complex.) The second point to note is that when decoding over a truncated convolutional trellis the output is the data label(s) on the first branch, whereas for the block code trellises described in Chapter 4 the output is the sequence of data labels (or occasionally code labels) found on all branches forming the ML path. This means all decisions along the ML path of a block code trellis are important since they all will affect the outcome. Conversely, for convolutional decoding fewer decisions will be important as decoding progresses—the length of the truncated convolutional trellis is chosen with the assumption that all surviving paths have merged with the ML path by the time decoding is terminated.

The differences between the standard and soft output Viterbi algorithms are most easily seen with the aid of a decoding example. They are also illustrated in Figure 5.5, where the flowchart given in Chapter 4 (Figure 4.8) is expanded to include the addi-

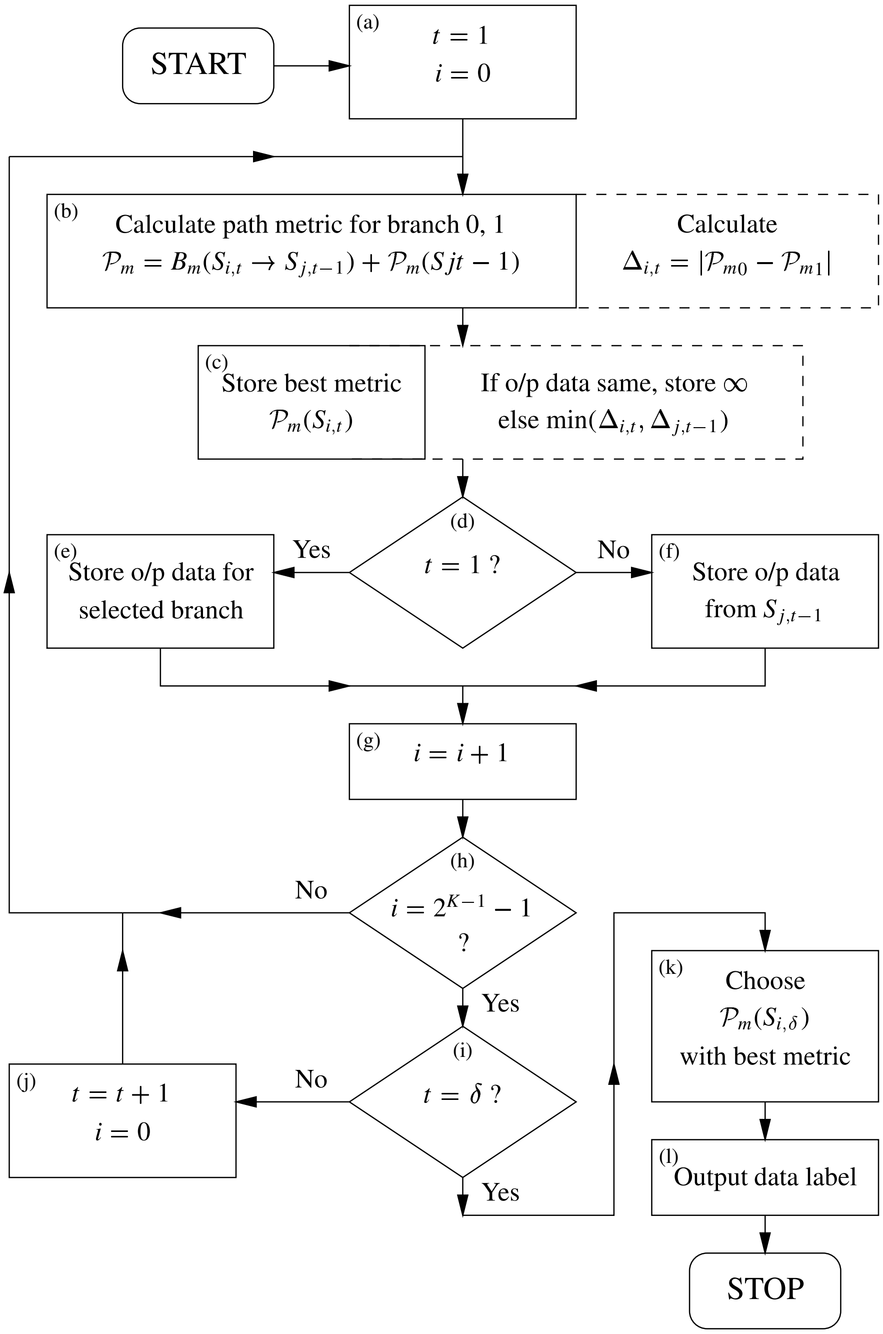


Figure 5.5: Flowchart of the soft output Viterbi decoding algorithm.

tional steps for SOVA.

Example 5.2 Soft-output Viterbi decoding of a $(2, 1, 3)$ convolutional code.

Example 4.3 will now be extended to include SD outputs. Figure 5.6 shows the annotated trellis. Every decision-making state i.e., those for which time $t > 0$, is now annotated with two values, the partial path metric to time $t = 0$ (as in Example 4.3) and the new LL reliability of the decision made at that node (in parentheses).

Trellis decoding starts at state $S_{0,1}$. The best path from $t = 0$ is $\mathcal{P}(S_{2,0} \to S_{0,1})$ and is indicated by a solid black line. The decoding metric for state $S_{0,1}$ is 8. Since both paths give the same output data, "1", it is clear that no bit error would have occurred had the discarded path been selected. Therefore the LL reliability of this decision is ∞. This process is repeated for all other states at time $1 < t \leq 8$.

At state $S_{0,3}$ the decision of which path to discard will influence the decoder output should that node be on the ML path. The reliability metric is the difference between the metrics of the selected path and the discarded path, i.e., $24 - 19 = 5$. Should the decision metric be zero this shows that the choice of best path was tied; therefore the value of the data output was dependent upon an arbitrary decision and should not be relied upon.

The reliability is found by tracing back along the ML path and taking the minimum of all the decision metrics on the ML path. Thus the data output is "0" with reliability 7 (from state $S_{2,3}$). Trace-back can be avoided if each state keeps track of the (minimum) reliability metric.

It was noted in Example 4.3 that the trellis was just long enough for the surviving paths to merge with the ML path. This is seen by tracing the paths back in time, and

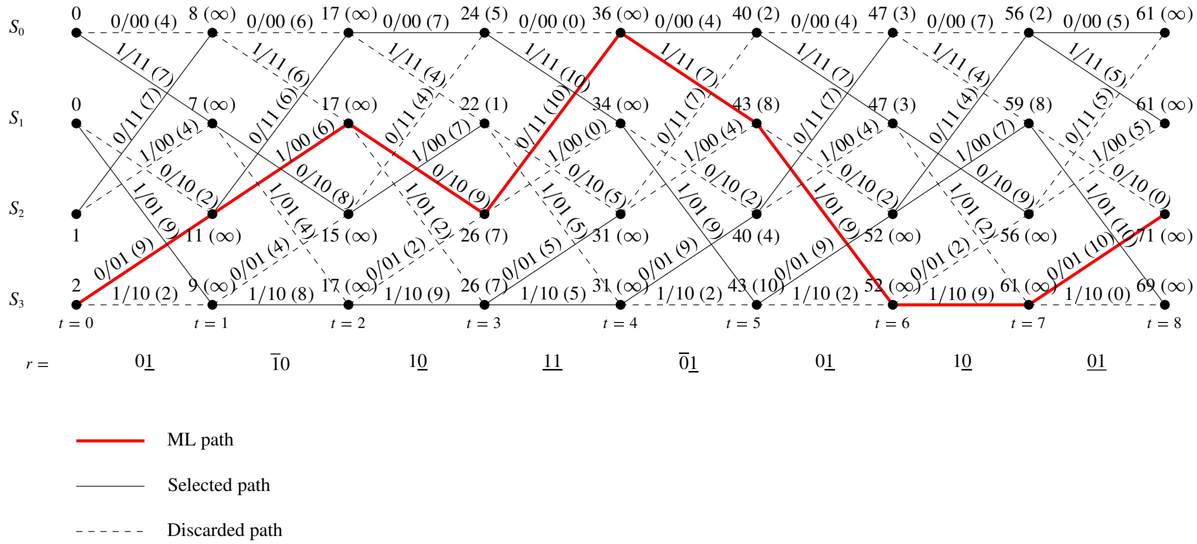


Figure 5.6: Soft-decision Viterbi decoding example with soft outputs.

also by noting the reliability of all the decisions made at $t = 8$ is ∞, indicating no difference in output data. $\square$

In Section 6.5 SOVA is used to improve the performance of a satellite link using a standard concatenated coding system. Unlike the iterative decoding described in [Hagenauer *et al.*, 1996] where the extrinsic information is used as the *a priori* information to the next *iteration*, the extrinsic information is used as the *a priori* information to the next *decoder*.

5.2.3 SOVA and Non-binary Trellises

SOVA considers only the weakest decision made, and thus considers only the ML path and the best discarded path. Thus SOVA is analogous to a MAP decoder with 2 codewords [Bahl *et al.*, 1974]. The SOVA algorithm described above has been extended to work over non-binary convolutional code trellises (and also binary code trellises where $k \neq 1$).

At each node in the trellis the reliability of the decision is given by considering the best and next-best paths, ignoring the metrics on the other discarded paths. As before, if the output data would be identical the reliability of the decision is ∞, otherwise it is the difference in metrics of the best and next-best paths; should an arbitrary decision be made the decision metric is zero. Thus at the least reliable node on the ML path the extended SOVA algorithm decides between two codewords. Whilst in principle it is possible to consider multiple codewords the complexity is correspondingly higher.

5.3 Reed-Solomon Product Codes

5.3.1 Introduction

Product codes can be considered a form of concatenated codes, where the row and column codes form the inner and outer codes. If constructed with linear codes then the encoding and decoding order is not important (Section 2.1.4) and may be reversed if desired. The decoding of row and column codewords can even be alternated [Bate *et al.*, 1986; Farrell *et al.*, 1986]. This is in contrast with a true concatenated coding scheme (Section 2.3) where decoding order must be the reverse of the encoding order.

For optimal performance it is desirable to use soft-decision decoding. Many possible methods exist; the Fano [Fano, 1963], stack [Jelinek, 1969; Zigangirov, 1966], Chase [Chase, 1972], and Viterbi [Viterbi, 1967] algorithms, and also MAP [Bahl *et al.*, 1974; Hagenauer and Hoeher, 1989]. Equally desirable are soft outputs, so that the second decoder of the concatenated system can perform optimally. It is of course important to balance optimality with complexity, so that a realiseable solution may be implemented, either in hardware or software. For this reason the extended SOVA described in Section 5.2.3 was selected. The MAP algorithm with its symbol-by-symbol reliability metrics is also applicable but the complexity of SOVA is considerably lower for only minor degradation in performance [Hagenauer *et al.*, 1996]. Any appropriate block code trellis may be used. The block code trellises used in [Hagenauer *et al.*, 1996] are based upon the parity-check matrix (**H**) and are thus irregular in structure. Regular trellises allow reductions in decoder complexity and storage requirements, thus the GAC construction methods based upon the generator matrix (**G**) described in

Chapter 4 are used here. RS codes were chosen as the block codes, they are MDS and so provide the greatest possible distance for a given n and k. This is important for a product code as the overall distance is the product of the distances of the subcodes.

5.3.2 Creation of Systematic Trellises Using GAC Construction

For some product code decoding algorithms it is required that the subcode codewords are systematic. One such algorithm is the alternating row/column method described in Section 5.3.7 or other algorithms which may terminate early when all codewords containing data symbols have been decoded. More precisely, the data symbols should have a one-to-one correspondence with k code symbols—the actual order of the symbols is not important nor do they need to be consecutive (such a symbol reordering is trivial) provided the data symbols are identifiable.

Consider the RS$(7, 5, 3)$ trellis constructed in Example 4.1. The generator matrix (4.22) is not systematic as it is not in reduced-echelon form, nor can it be rearranged to be. Only u_5 is unchanged after the data word $\mathbf{u}$ is multiplied by the generator matrix. Table 5.2 shows a few of the 32768 codewords from the RS$(7, 5, 3)$ trellis. For clarity the symbols are given in decimal form. It clearly shows the lack of a one-to-one correspondence between the data symbols and the code symbols (u_5 excepted). RS codes are invertible (Section 2.1.8) and therefore any k symbols can be chosen as data symbols, the remaining $n - k$ symbols form the parity checks. Since the trellis is labelled with independent data and code labels it is possible to re-map the data symbols on the trellis to match the first k code symbols. Table 5.3 shows the same datawords with the new systematic mapping.

dataword					codeword						
u_1	u_2	u_3	u_4	u_5	v_1	v_2	v_3	v_4	v_5	v_6	v_7
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	4	5	1
0	0	0	0	2	0	0	0	0	5	6	2
		⋮						⋮			
0	0	7	7	7	0	0	6	3	2	5	7
0	1	0	0	0	4	5	0	1	0	0	0
0	1	0	0	1	4	5	0	1	4	5	1
		⋮						⋮			
7	7	7	5	6	3	6	6	2	7	2	6
7	7	7	5	7	3	6	6	2	4	7	7
7	7	7	6	0	3	6	3	2	1	6	0
		⋮						⋮			

Table 5.2: Sample data and codewords of the non-systematic RS$(7,5,3)$ trellis.

dataword					codeword						
u_1	u_2	u_3	u_4	u_5	v_1	v_2	v_3	v_4	v_5	v_6	v_7
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	2	5
0	0	0	0	2	0	0	0	0	2	3	6
		⋮						⋮			
0	0	7	7	7	0	0	7	7	7	3	4
0	1	0	0	0	0	1	0	0	0	1	1
0	1	0	0	1	0	1	0	0	1	4	6
		⋮						⋮			
7	7	7	5	6	7	7	7	5	6	3	0
7	7	7	5	7	7	7	7	5	7	0	6
7	7	7	6	0	7	7	7	6	0	2	4
		⋮						⋮			

Table 5.3: Sample data and codewords of the systematic RS$(7,5,3)$ trellis.

5.3.3 Cascade Decoding

Many different methods for decoding product codes exist. It is useful to define the simplest as a standard against which the performance of more 'intelligent' algorithms are compared. For this purpose cascade decoding (Section 2.1.4, p. 15) is used as the reference method. It is expected that the channel can supply soft-decision information to the row decoder, hence the first decoding stage is soft-decision. However the standard decoder does not make use of advanced techniques such as SOVA (Section 5.2), so it is unable to supply soft information to the column decoder (second stage). The block diagram of the decoder is shown in Figure 5.7.

5.3.4 Modification of the Channel Metrics with the SOVA Metric

The SOVA decoder detailed in Section 5.2 was configured to produce two outputs, a codeword and a metric indicating the reliability of the chosen codeword. For non-cascade decoding algorithms the full codeword symbols are required. For cascade decoding only the dataword is strictly required, however this is easily obtained from the first k symbols of the systematic codeword. In order for the SOVA metric to influence later decodings its value must be somehow incorporated into the buffer storing the received channel metrics. The method used for this was common to both product code decoding schemes using SOVA (Sections 5.3.5 and 5.3.7) and so will be described separately below.

Remembering that SOVA on a block code trellis outputs only one metric (Section 5.2.2) necessitates the assumption that the metric applies equally to all symbols.

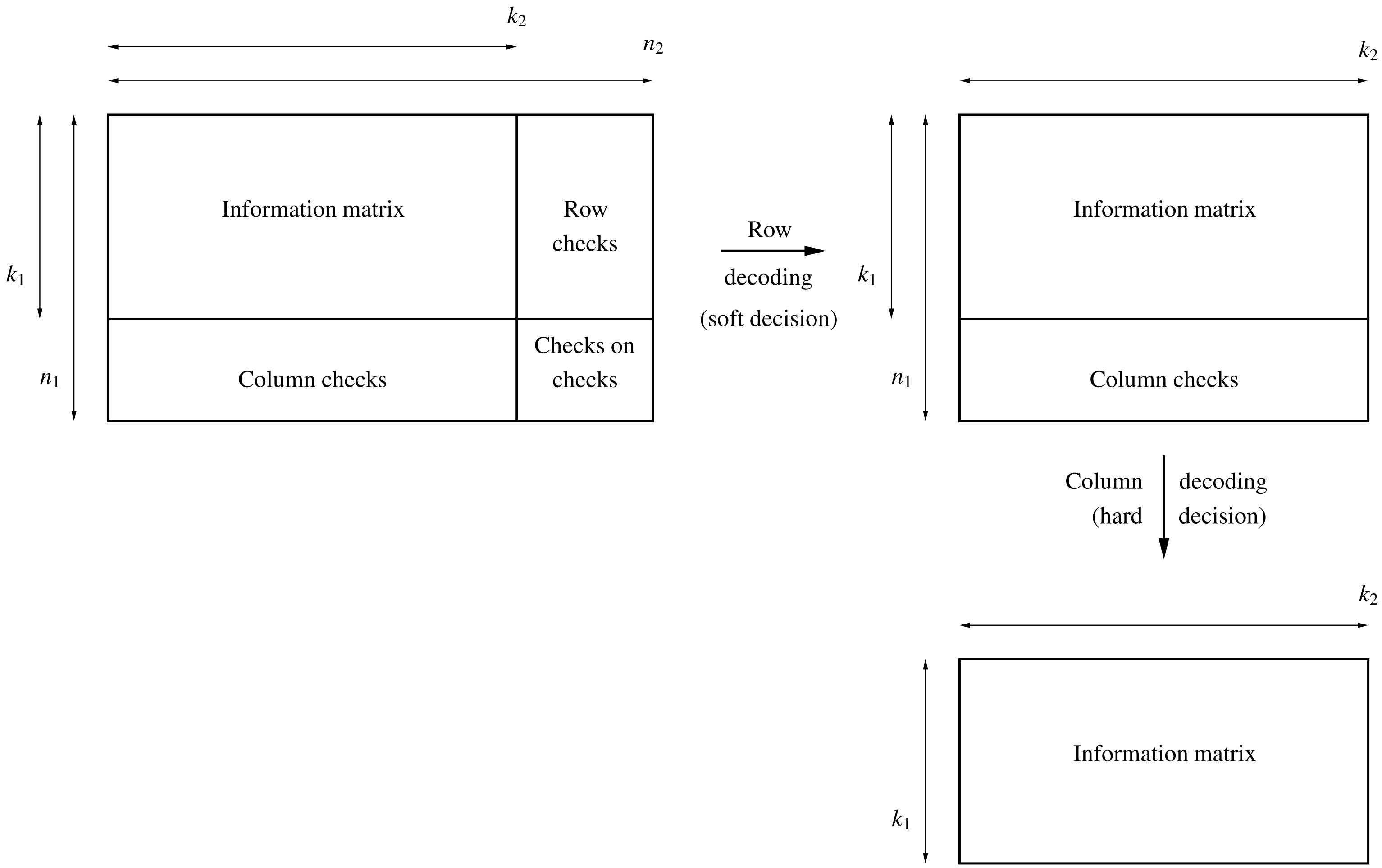


Figure 5.7: Cascade decoding algorithm for product code.

Nor does SOVA indicate the next most likely value for a symbol from a non-binary alphabet. Therefore it must be assumed that the discarded values for each symbol are equally improbable. Plainly this is not the case but without more information no better assumptions can be drawn.

In all cases the received symbol metrics (channel metrics) were stored in a buffer of the same dimensions as the transmitted codeword ($n_1 \times n_2$). When decoding a row/column the corresponding row/column metrics were passed to the SOVA decoder. After decoding it is not desirable to completely replace the channel information with the SOVA metric as it applies to only a subset of all possible symbol values. SOVA may also have incorrectly decoded the received codeword. Instead, the SOVA metric is accumulated to the M buffered metrics for the symbol values which correspond to those selected by SOVA (where $M = n_1$ for rows and $M = n_2$ for columns). Thus the extrinsic information derived by SOVA is accumulated to the existing channel state information. An example will help demonstrate the method used.

Example 5.3 Consider a $(9, 4, 4)$ product code, whose subcodes are $(3, 2, 2)$ parity check codes over GF(4). Using SOVA decode the first row code and modify the symbol metrics.

Let the received metrics be

$$\begin{array}{ccc|c|c|c|}
 & & & \multicolumn{3}{c}{\text{column}} \\
 & & & \multicolumn{1}{c}{1} & \multicolumn{1}{c}{2} & \multicolumn{1}{c}{3} \\
\cline{4-6}
\text{row} & & 1 & \begin{array}{rr} 0: & 19 \\ 1: & 28 \\ \alpha: & 4 \\ \alpha^2: & 13 \end{array} & \begin{array}{rr} 0: & 23 \\ 1: & 28 \\ \alpha: & 8 \\ \alpha^2: & 13 \end{array} & \begin{array}{rr} 0: & 23 \\ 1: & 25 \\ \alpha: & 14 \\ \alpha^2: & 16 \end{array} \\
\cline{4-6}
 & & 2 & \ddots & \ddots & \ddots \\
\cline{4-6}
 & & 3 & \ddots & \ddots & \ddots \\
\cline{4-6}
\end{array} \qquad (5.13)$$

SOVA indicates that the best codeword is "110", with a reliability of 3. Therefore add 3 to the metrics corresponding to a row 1 codeword "110". The new metrics (emboldened) are

$$\begin{array}{ccc|c|c|c|}
 & & & \multicolumn{3}{c}{\text{column}} \\
 & & & \multicolumn{1}{c}{1} & \multicolumn{1}{c}{2} & \multicolumn{1}{c}{3} \\
\cline{4-6}
\text{row} & & 1 & \begin{array}{rr} 0: & 19 \\ 1: & \mathbf{31} \\ \alpha: & 4 \\ \alpha^2: & 13 \end{array} & \begin{array}{rr} 0: & 23 \\ 1: & \mathbf{31} \\ \alpha: & 8 \\ \alpha^2: & 13 \end{array} & \begin{array}{rr} 0: & \mathbf{26} \\ 1: & 25 \\ \alpha: & 14 \\ \alpha^2: & 16 \end{array} \\
\cline{4-6}
 & & 2 & \ddots & \ddots & \ddots \\
\cline{4-6}
 & & 3 & \ddots & \ddots & \ddots \\
\cline{4-6}
\end{array} \qquad (5.14)$$

□

It can be seen that if the reliability of the decision made by SOVA is zero, i.e., the decision was arbitrary, then no change is made to the metrics. Conversely, a positive decision by SOVA will result in a large modification in the LL metrics. Summing the same value to all symbols obeys the assumption stated that all symbols are equally reliable; not modifying the discarded symbol values maintains the second assumption that the discarded symbol values are equally improbable. At no point is the *a priori* channel information discarded, but is modified with the extrinsic information from each nonarbitrary decoding.

5.3.5 Cascade Decoding Algorithm with SOVA

A logical extension of the cascade decoder is to apply SOVA decoding to the first decoding stage. The second stage is therefore able to use SD information. In other studies [Hagenauer *et al.*, 1994; Marple, 1998] SOVA provided a performance improvement of about 2 dB over an AWGN channel for a true concatenated system.

The received symbol metrics are stored in a buffer of the same dimensions as the transmitted codeword ($n_1 \times n_2$). The rows are decoded using the extended SOVA algorithm detailed in Section 5.2.3, taking the soft information from the buffer. After decoding each row the SOVA metric is used to modify the metrics stored in the buffer (Section 5.3.4). The second stage, decoding columns, then follows. The Viterbi decoder is able to make use of SD information from the first stage, along with the *a priori* channel information which preceded the first decoding stage.

5.3.6 Alternative Decoding Strategies

Alternative decoding strategies exist which aim to minimise the number of errors. Bate *et al.* considered various methods based upon decoding rows and columns alternatively [Bate *et al.*, 1986]. The row/column subcodes were decoded in decreasing order of confidence. The algorithms investigated for subcode decoding were hard decision decoding, soft decision decoding using successive erasures decoding [Chase, 1972] and combined soft/hard decision decoding. In [Bate *et al.*, 1986] the method used to compute the row/column confidences was

$$C = \sum_{i=1}^{M \log_2 q} \left| \log \frac{p(r_i \mid 0)}{p(r_i \mid 1)} \right| \tag{5.15}$$

where

$$M = \begin{cases} n_1 \text{ for rows} \\ n_2 \text{ for columns} \end{cases}$$

By rearranging (5.15) it can be shown how LL metrics can be used instead of conditional probabilities.

$$C = \sum_{i=1}^{M \log_2 q} \Big| \log p(r_i \mid 0) - \log p(r_i \mid 1) \Big| \tag{5.16}$$

$$C' = \sum_{i=1}^{M \log_2 q} |\ell_{0_i} - \ell_{1_i}| \tag{5.17}$$

where ℓ_{0_i} is the LL metric for symbol "0" for the i-th bit and ℓ_{1_i} is the metric for symbol

"1" for the i-th bit. In other words, the confidences can be computed by summing the difference between the LL metrics for "0" and "1". In the case of LL ratios which have been mapped to integer values C' will be related to C by $C' \simeq aC$ (allowing for rounding errors) where a is a scaling factor (Equation 5.5).

5.3.7 Alternating Row-Column Decoding Using SOVA

The alternating row-column decoder in [Bate *et al.*, 1986] was adapted to use SOVA. The channel modelled was binary, though only symbol metrics were available to the decoders (Section 6.6.1). This required a small change to the method used for calculating the received codeword confidence. The symbols represented by the best and worst LL metrics are bit inverses of each other. Therefore (5.17) may be rewritten as

$$C' = \sum_{i=1}^{M} |\ell_{\text{best}_i} - \ell_{\text{worst}_i}| \tag{5.18}$$

where ℓ_{best_i} is the metric corresponding to the most likely value for the i-th symbol and ℓ_{worst_i} is the least likely.

The received symbol metrics are stored in a buffer of the same dimensions as the transmitted codeword (n_1 by n_2). From these buffered values the row and column codeword confidences are computed with (5.18) and sorted into order of decreasing confidence. The row codeword with the highest probability of being correct is decoded first.

After decoding one row codeword the next most likely column codeword is decoded. This process is repeated until all row and column codewords containing data symbols are decoded. By this method each data symbol is decoded twice, once with a row decoding and once with a column decoding. The decodings may occur in either order. However after both decodings the data symbol's metrics cannot be changed. Therefore after the second decoding the value of the symbol (as decided by the SOVA decoder) is transferred to the output buffer.

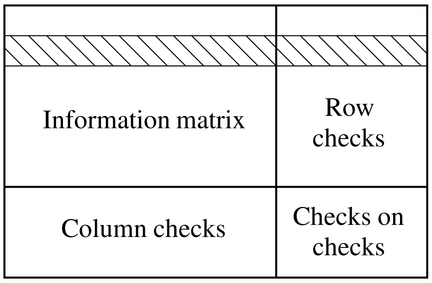
Decoding may terminate early if the remaining row and column codewords do not contain data symbols, their decoding will not affect the output data. With this procedure the average decoding delay and computation are reduced. If the product code is not approximately square, i.e., $n_1 \gg n_2$ or $n_1 \ll n_2$, it may be desirable to decode row and column codewords in some ratio other than $1 : 1$ so that the row and column decodings complete in approximately the same number of cycles.

Bate *et al.* [Bate *et al.*, 1986] recomputed the row and column subcode confidences after each row and column iteration. Re-sorting the confidences can be numerically expensive, even the best sorting algorithms require of the order of several times $N \log_2 N$ operations [Press *et al.*, 1992, p. 329]. The performance of the algorithm described above was tested with and without the re-sorting of codeword confidences.

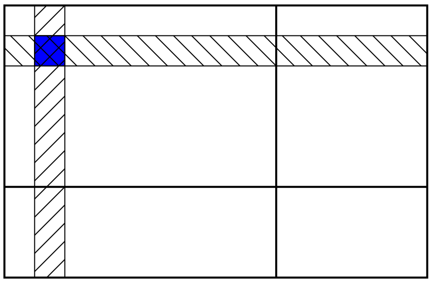
Example 5.4 The decoding of an arbitrary product code with the alternating row/ algorithm.

Figure 5.8 shows the initial decoding stages. Hatching denotes codewords which have been decoded, whilst shading indicates information symbols which have been copied to the output buffer. Each step is explained below.

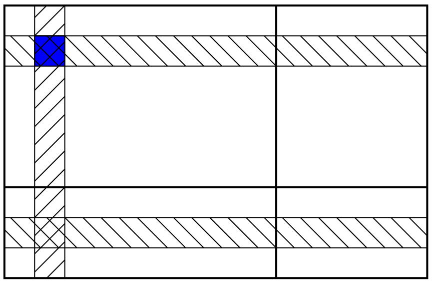
(a) In this case the best row codeword is within the information section of



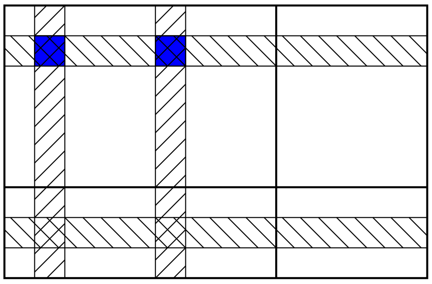
(a) 1st row decoding.



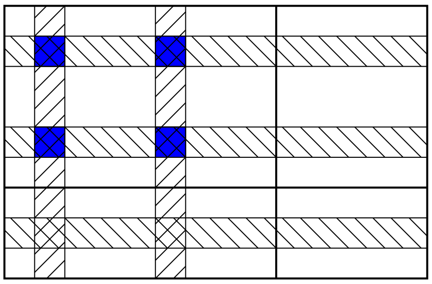
(b) 1st column decoding.



(c) 2nd row decoding.



(d) 2nd column decoding.



(e) 3rd row decoding.

Figure 5.8: Alternating row/column decoding example.

the product code, but no symbols have been decoded twice.

(b) The best column decoding is also within the information section of the product code. One data symbol has been decoded twice and is copied to the output buffer.

(c) The second best row decoding does not affect any data symbols of the product code. However by modifying the confidence metrics of the remaining column codewords it may still affect the final result.

(d) The second best column decoding intersects two completed row decodings. Only the intersecting data symbol is copied to the output buffer.

(e) The third row decoding intersects two previously decoded columns. Two data symbols are copied to the output buffer.

This process continues until all data symbols have been decoded twice and copied into the output buffer. □

5.3.8 Iterative Decoding of Product Codes

Although it was stated earlier that SOVA was applied for concatenated coding techniques it was noticed how readily iterative decoding may be applied to the decoder described in Section 5.3.7. After decoding the channel metric buffer contains the channel metrics plus the extrinsic information from SOVA, ready to be the *a priori* information for the next iteration. Hence repeated decodings on the same channel metric buffer results in an iterative decoding process.

Research is a product of an inquisitive mind (and *vice versa*); given the ease with which iterative decoding may be applied and the additional increase in coding gain possible ($> 1.5\,\mathrm{dB}$ for just 4 iterations with a BCH$(64, 51, 6) \times$ BCH$(64, 51, 6)$ code [Pyndiah, 1998]) the temptation to test this decoder in an iterative fashion could not be resisted. Some encouraging introductory results are given in Section 6.6.4.

Chapter 6

Results and Computer Simulations

Chapter 6

Results and Computer Simulations

6.1 Decoding Complexity Measurements

6.1.1 Implementation Overview

All the decoders discussed in detail in this Thesis were implemented in C++ [Stroustrup, 1997], a high-level, object-oriented programming language. C++ classes allowed rapid development and code reuse of important components such as GF and polynomial arithmetic, channel models and trellis decoders. The trellises themselves were constructed at run-time from a net list contained in a *Trellis Description Language* file. The trellis diagrams in this Thesis were computer-generated from the trellis description files. A high-level implementation allowed a great degree of flexibility in the number and type of codes which could be implemented.

It should be noted that all the simulations described in this Chapter have been performed in full, using random data and random errors. For trellis decoding full implementations of the trellis were used. This is important since it allows real implementations of the decoders described. Simulations which consist of transmitting the all-zeros codeword and/or matched-filters over a limited subset of codewords are useful performance tools but cannot be used in real systems where the full set of codewords may be transmitted, for which the trellis diagram is used to exploit the redundancy in the tree diagram.

6.1.2 Algebraic Decoding Complexity Measurements

Various measurements of complexity exist. One method is to count the number of codewords decoded in a given time. This is only accurate if all the decoders in the trial are implemented equally well. Subsequent comparisons can only be made by using the same hardware, which may not be appropriate for all algorithms, and which may not always be available. A more formal method is the use of O-notation.[1] While O-notation is a helpful tool for algorithm designers its use is not without problems [Sedgewick, 1988, pp. 71–76]. It is a worst-case bound, the constants c_0 and N_0 are unknown and may be large. Without knowledge of the constants c_0 and N_0 only the asymptotic performance may be compared.

A more practical method was chosen instead, that of counting the number of important mathematical operations. For high-level simulations this is a comparatively easy task. The relative execution time of algebraic operations is dependent upon the hardware selected, but, by choosing the appropriate parameters the method can be ap-

[1] A function $g(N)$ is said to be $O(f(n))$ if there exist constants c_0 and N_0 such that $g(N) < c_0 f(N)$ for all $N > N_0$ [Sedgewick, 1988, p. 72].

plied to any implementation. To enable valid comparisons it was assumed that each decoder could be implemented on an AT & T DSP32C digital signal processor. Each mathematical operation (add (+), subtract (−), multiply (×), divide (÷) and compare (=)) was assigned a cost in terms of the number of CPU cycles required for its execution. Table 6.1 shows the cost of integer, floating-point (real) and Galois field arithmetic, taken from a DSP32C implementation of a minimum-weight RS decoder.[2] The DSP32C does not contain instructions for integer multiply, integer divide or floating-point divide. However, these instructions were not needed by any of the decoders implemented.

It was assumed that the GF arithmetic would be implemented using the same polynomial basis as used by the minimum-weight decoder. For fields of characteristic 2 addition and subtraction are identical and can be performed with an exclusive-OR logical operation. For the polynomial basis multiplication and division are most easily implemented by table look-up, which is the method used in Table 6.1. To enable comparisons between different codes the decoding complexity is converted to a *bit complexity*, κ_{bit}, which is the total decoding complexity divided by the number of bits output from one decoding.

[2]From I. Martin, private communication.

data type	Operation				
	$+$	$-$	$\times$	$\div$	$=$
int	1	1	n/a	n/a	3
float	2	2	2	n/a	6
GF	1	1	5	5	3

Table 6.1: Relative complexity of algebraic operations (for a DSP32C).

6.2 Trellis Decoding Complexity

6.2.1 Introduction

The complexity of a trellis has previously been measured by various means, number of states [Muder, 1988], number of vertices [Kasami *et al.*, 1993a,b] and number of edges [McEliece, 1996]. However, none of these methods allow trellis decoding complexity to be compared against algebraic decoding complexity. To do so, one method is to calculate the number of arithmetic operations required [Honary, Markarian, and Marple, 1995c, 1996, 1997]. Therefore, the trellis decoding complexity can be compared directly with algebraic decoding using the method described in Section 6.1.2. It should be noted that for integer and floating-point numbers the term *comparison* includes the tests $>$, $<$, $\geq$ and $\leq$ in addition to equality ($=$) and their logical inverses. For compactness comparison is denoted by simply "$=$".

It is first assumed that trellis decoding will be performed with the Viterbi algorithm, and then later extended to include SOVA. In the analysis presented here it is also assumed that log likelihood metrics are used to avoid multiplication operations (p. 119); a similar analysis is possible for Euclidean distance metrics, which were used in [Honary, Markarian, and Marple, 1995c, 1996, 1997]. Following these assumptions the complexity can be calculated with the steps shown below. Note that only algebraic operations involving the log likelihood metrics are counted, other operations are designated *overheads*. The overheads, which are very dependent on the actual hardware or software implementation, are not usually included in considerations of complexity. One important consequence is that memory accesses to the stored metrics can be made for free. However, no assumption as to the type of the log likelihood metric (i.e., integer or floating-point) is made.

6.2.2 Complexity of the Viterbi Algorithm

Branch Labels

Consider a trellis with a state profile $\mathbb{N}(t) = [\mathbb{N}_0, \mathbb{N}_1, \ldots, \mathbb{N}_{N_c}]$, branch profile $\mathbb{B}(t) = [\mathbb{B}_1, \mathbb{B}_2, \ldots, \mathbb{B}_{N_c}]$ and a (code) label profile $\mathbb{L}(t) = [\mathbb{L}_1, \mathbb{L}_2, \ldots, \mathbb{L}_{N_c}]$. To calculate the metric associated with one branch at depth j requires the addition of $\mathbb{L}_j$ log likelihood metrics, a process which needs $\mathbb{L}_j - 1$ additions. This is repeated for all branches at depth j, and for all depths $j = 1, 2, \ldots, N_c$. Therefore, in calculating the branch metrics the total number of additions, N^+, is given by

$$N^+ = \sum_{j=1}^{N_c} \mathbb{B}_j(\mathbb{L}_j - 1) \tag{6.1}$$

Note that trellises for simpler codes, such as RM and single error-correcting Hamming codes, may 'share' branch labels. At a given depth, j, more than one branch may be labelled with the same code symbols, i.e., $L(S_{i,t}, S_{j,t+1}) = L(S_{i',t}, S_{j',t+1})$ where $\mathbb{L}_t > 1$ and at least one of the inequalities $i \neq i'$, $j \neq j'$ holds (i.e., the start and/or end vertices must differ). An example of such a trellis is Figure 2.4, where for instance $L(S_{1,1}, S_{1,2}) = L(S_{2,1}, S_{2,2}) = [0, 0]$. The branch metrics associated with such branches will always be identical so it is possible to optimise decoding by saving the temporary result to memory after the first calculation. Note that this saving is irrelevant for branches with a code label size of 1; from (6.1) the complexity of evaluating such a branch is zero. This optimisation is not possible with RS codes and so will not be considered further.

State metrics

The state metric is the metric for the best partial path from the root to the state in question. Consider the trellis section given in Figure 6.1. Let the number of branches entering state $S_{j,t}$ be N_b. The first partial path metric can be calculated as the sum of the state metric from the preceding level, $S_{i,t-1}$, and the metric of the branch linking the two states. This requires one addition. The process is repeated for the remaining $N_b - 1$ branches. The best metric is stored as the state metric, finding the best metrics requires $N_b - 1$ comparisons. For a linear trellis the number of branches entering each state $S_{j,t}$ is the same for all $j = 1,\ 2,\ \ldots,\ \mathbb{N}_t$ states at depth t. Therefore N_b is given by

$$N_b = \frac{\mathbb{B}_t}{\mathbb{N}_t} \tag{6.2}$$

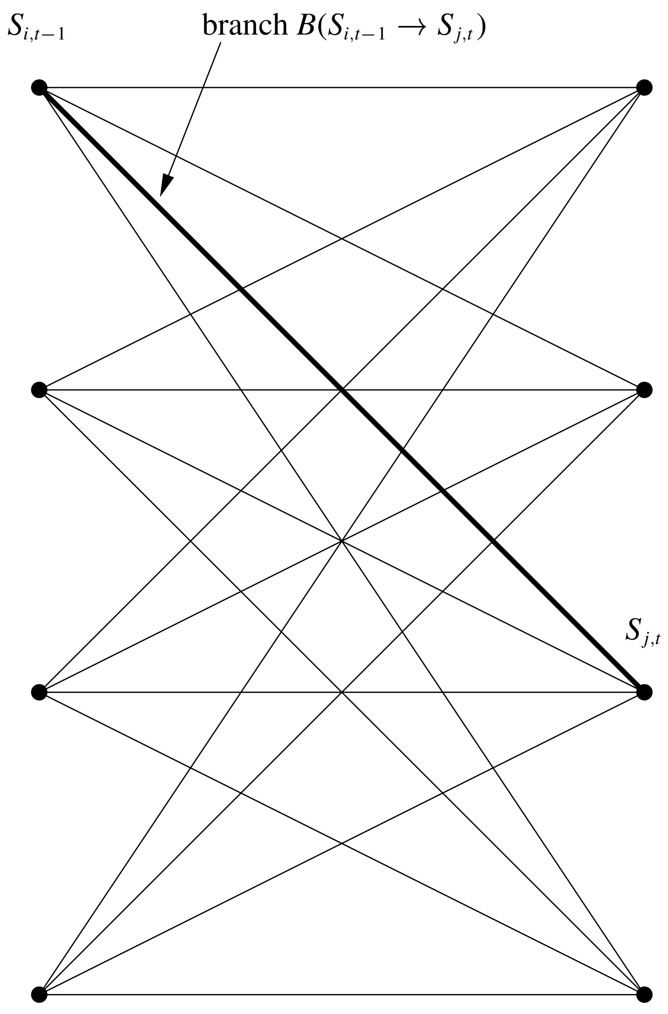


Figure 6.1: Calculating the state metric.

The total number of comparisons for all states is given by

$$\begin{aligned} N^{=} &= \sum_{t=1}^{N_c} (N_b - 1)\,\mathbb{N}_t \\ &= \sum_{t=1}^{N_c} \left(\frac{\mathbb{B}_t}{\mathbb{N}_t} - 1 \right) \mathbb{N}_t \\ &= \sum_{t=1}^{N_c} (\mathbb{B}_t - \mathbb{N}_t) \end{aligned} \tag{6.3}$$

For rectangular non-linear codes the same technique can be applied, except that N_b may not be constant over all vertices at a given depth, requiring an extra summation over all vertices at depth t.

In some cases it is possible to optimise the state metric calculation. For trellises which are never truncated the state metric of the root is always zero, therefore no addition is required in the calculation of the state metric at depth 1. Generally this is not true for convolutional code trellises (see Example 4.3) since the first states in the trellis will not have a zero metric. The number of additions required to calculate the state metrics is given by

$$\begin{aligned} N^{+} &= \sum_{t=t_s}^{N_c} N_b \mathbb{N}_t \\ &= \sum_{t=t_s}^{N_c} \mathbb{B}_t \end{aligned} \tag{6.4}$$

where $t_s = 2$ for block code trellises and $t_s = 1$ for truncated (convolutional) trellises.

Total Complexity

The total complexity for trellis decoding using the VA is obtained by combining (6.1), (6.3) and (6.4).

$$N_{\mathrm{VA}}^{+} = \sum_{j=1}^{N_c} \mathbb{B}_j(\mathbb{L}_j - 1) + \sum_{t=t_s}^{N_c} \mathbb{B}_t \tag{6.5}$$

$$N_{\mathrm{VA}}^{=} = \sum_{t=1}^{N_c} (\mathbb{B}_t - \mathbb{N}_t) \tag{6.6}$$

6.2.3 Complexity of SOVA

Using the same technique as above it is possible to calculate the additional complexity for SOVA decoding. The calculation of the branch metrics is unchanged. During the calculation of the state metric the reliability metric must also be determined. After finding the best partial path from the N_b possible choices the next best path must be found, i.e., the best out of the remaining $N_b - 1$ possibilities. However, by arranging the selection as a binary tree, less than $\lceil \log_2 N_b \rceil$ comparisons are needed [Knuth, 1973, pp. 142–143]. Restricting N_b to be an integer power of 2, only $\log_2 N_b - 1$ additional comparisons are needed at each decision-making state.[3] A decision-making state is defined by $N_b > 1$.

Finally, SOVA must store the difference between the best and next best paths, requiring one subtraction per decision-making state. SOVA must also test if the output data from the best and next best paths is the same. If not, the difference between the

[3]For linear codes where q is an integer power of 2, N_b is always an integer power of 2. This restriction is met for the majority of all useful codes.

best and next best paths must be calculated and stored, which requires one subtraction per decision-making state. For the case that both paths would result in the same output ∞ is stored. The comparison of output data is declared to be part of the overheads for two reasons. Firstly, it is an integer comparison, not necessarily the same type of comparison as may be used for comparing two log likelihood metrics. Secondly, for block code trellises the output data is always different, thus the comparison always fails and can be eliminated. Therefore the additional number of comparisons and subtractions is given by

$$
\begin{aligned}
N^{=} &= \sum_{t=1}^{N_c} \begin{cases} 0 & \text{if } \mathbb{N}_t = \mathbb{B}_t \\ \mathbb{N}_t \left(\log_2 N_b - 1\right) & \text{otherwise} \end{cases} \\
&= \sum_{t=1}^{N_c} \begin{cases} 0 & \text{if } \mathbb{N}_t = \mathbb{B}_t \\ \mathbb{N}_t \left(\log_2 \dfrac{\mathbb{B}_t}{\mathbb{N}_t} - 1\right) & \text{otherwise} \end{cases}
\end{aligned}
\tag{6.7}
$$

$$
N^{-} = \sum_{t=1}^{N_c} \begin{cases} 0 & \text{if } \mathbb{N}_t = \mathbb{B}_t \\ \mathbb{N}_t & \text{otherwise} \end{cases}
\tag{6.8}
$$

The total complexity of decoding a trellis using SOVA is therefore given by combining

(6.5), (6.6), (6.7) and (6.8)

$$N^{+}_{\text{SOVA}} = \sum_{j=1}^{N_c} \mathbb{B}_j(\mathbb{L}_j - 1) + \sum_{t=t_s}^{N_c} \mathbb{B}_t \tag{6.9}$$

$$N^{=}_{\text{SOVA}} = \sum_{t=1}^{N_c} \begin{cases} 0 & \text{if } \mathbb{N}_t = \mathbb{B}_t \\ \mathbb{B}_t + \mathbb{N}_t \left(\log_2 \dfrac{\mathbb{B}_t}{\mathbb{N}_t} - 2 \right) & \text{otherwise} \end{cases} \tag{6.10}$$

$$N^{-}_{\text{SOVA}} = \sum_{t=1}^{N_c} \begin{cases} 0 & \text{if } \mathbb{N}_t = \mathbb{B}_t \\ \mathbb{N}_t & \text{otherwise} \end{cases} \tag{6.11}$$

Example 6.1 The RS$(7, 3, 5)$ trellis decoding complexity using VA and SOVA are compared for the case of an AT & T DSP32C digital signal processor.

The trellis (Figure 4.5) is decoded using integer LL metrics. Properties of the trellis are as below:

$$N_c = 3 \tag{6.12}$$

$$\mathbb{N}(t) = [1, 64, 64, 1] \tag{6.13}$$

$$\mathbb{B}(t) = [64, 512, 64] \tag{6.14}$$

$$\mathbb{L}(t) = [2, 3, 2] \tag{6.15}$$

$$t_s = 2 \tag{6.16}$$

Using the operation cost as given in Table 6.1 the relative complexities are shown in Table 6.2.

□

algorithm	$+$	$-$	$\times$	$\div$	$=$	total complexity	bit complexity
VA	1728	0	0	0	511	3261	362.33
SOVA	1728	65	0	0	644	3725	413.89

Table 6.2: Comparison of VA and SOVA decoding complexity for RS(7, 3, 5).

Example 6.2 Similar to Example 6.1, calculate the VA and SOVA decoding complexity for the RS(7, 5, 3) trellis (Figure 4.2). Properties of the trellis are as below:

$$N_c = 7 \tag{6.17}$$

$$\mathbb{N}(t) = [1, 8, 64, 64, 64, 64, 8, 1] \tag{6.18}$$

$$\mathbb{B}(t) = [8, 64, 512, 512, 512, 64, 8] \tag{6.19}$$

$$\mathbb{L}(t) = [1, 1, 1, 1, 1, 1, 1] \tag{6.20}$$

$$t_s = 2 \tag{6.21}$$

The decoding complexities are shown in Table 6.3.

algorithm	$+$	$-$	$\times$	$\div$	$=$	total complexity	bit complexity
VA	1672	0	0	0	1407	5893	392.87
SOVA	1672	201	0	0	1809	7300	486.67

Table 6.3: Comparison of VA and SOVA decoding complexity for RS(7, 5, 3).

□

6.3 A Comparison of Algebraic Decoders

6.3.1 Error-correction Performance

The algebraic decoders implemented were Euclidean, Berlekamp-Massey and high-speed step-by-step. All are HDMLD, their performance is therefore upper-bounded by (2.32). The probability of bit error can be converted from the probability of symbol error over an M-ary orthogonal signal set by (adapted from [Sklar, 1988, Equation 3.127, p. 180])

$$P_b = \frac{M}{2(M-1)} P_s \tag{6.22}$$

Figure 6.2 compares the error-correction performance of the algebraic decoders for the RS$(7, 5, 3)$ code over a coherently-demodulated BPSK channel. The BM BER is about half the bound. (The bound assumes that an incorrect decoding will result in all bits erroneous, whereas on average half are correct.) The Euclidean decoding implementation behaved slightly differently to that of Berlekamp-Massey and HSSBS in the case of decoder failures. The performance is slightly worse but still within the HDMLD bound.

6.3.2 Decoding Complexity

The decoder complexity was measured by using the C++ Galois field class to count the number of $+$, $-$, $\times$, $\div$ and $=$ (compare) operations. The decoders were presented with the same $t + 1$ sets of codewords, where each set contained $0,\ 1,\ \ldots,\ t$ errors. The

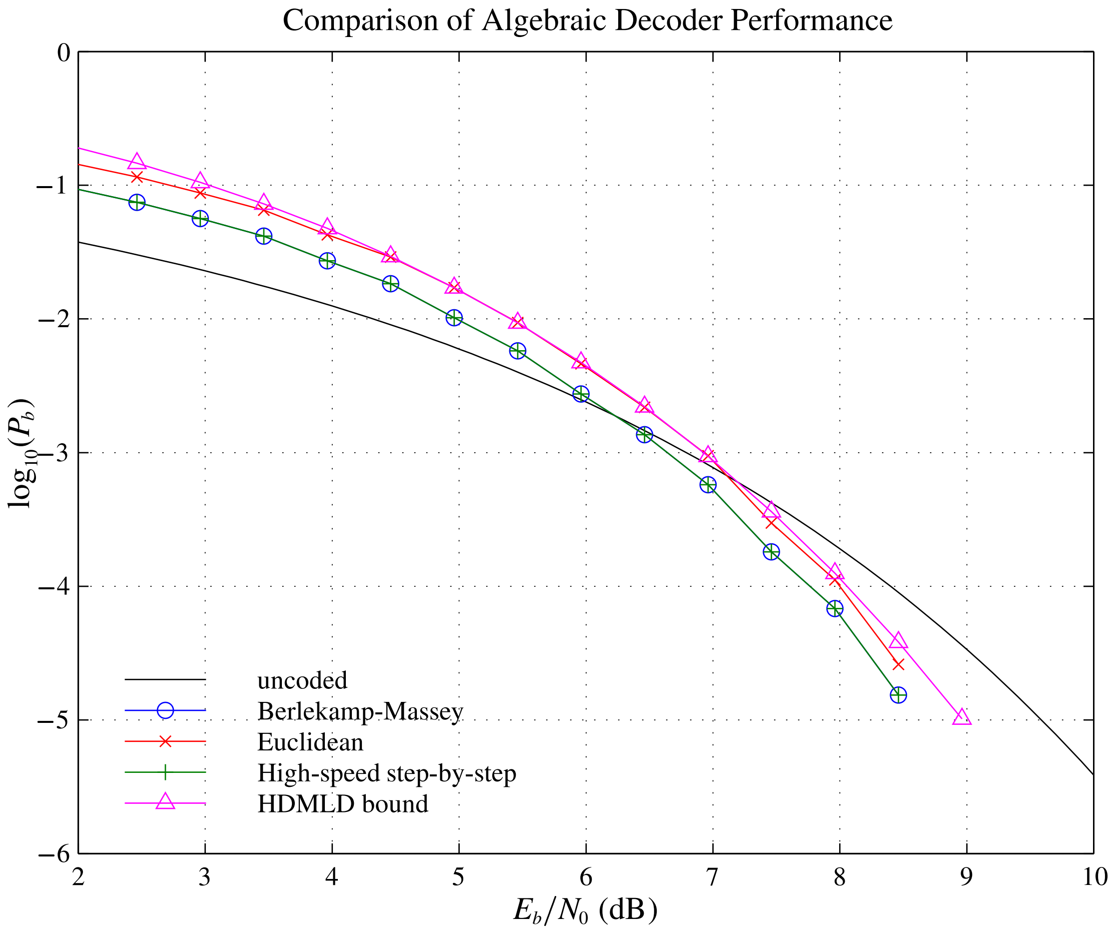


Figure 6.2: Algebraic decoding performance for RS(7, 5, 3) over a coherently-demodulated BPSK channel.

average decoding complexity for one codeword is shown for $\mathrm{RS}(7,3,5)$ (Table 6.4), $\mathrm{RS}(7,5,3)$ (Table 6.5), $\mathrm{RS}(63,55,9)$ (Table 6.6) and $\mathrm{RS}(255,223,33)$ (Table 6.7).[4] For the case of no errors there is no difference in complexity. This is to be expected since they are all syndrome-based algorithms where the first step is the syndrome calculation, for which they share a common method (3.1). On finding no errors no further work is necessary.

It can be seen that HSSBS decoding is the least efficient of the three algebraic decoders implemented. For codes over small alphabets (Tables 6.4 and 6.5) the performance is tolerable. With the chosen complexity criteria HSSBS is particularly heavily penalised for its high use of multiplications. With a different choice of basis for the GF arithmetic (e.g., logarithmic), where GF multiplications can be implemented more simply than a table look-up, its complexity performance would improve. Whatever basis is chosen, the implementation complexity of addition and subtraction are likely to be approximately equal to each other,[5] as are multiplication and division. However, combining the number of additions with subtractions and multiplications with divisions reveals that HSSBS will always be more complex than either Euclidean or Berlekamp-Massey decoding.

For codes over large alphabets the situation worsens dramatically (see Table 6.6); there are an exponentially increasing number of possible error values for a trial-and-error method to search. It was not possible to include complexity results for HSSBS in Table 6.7. Though HSSBS is an improvement over the original step-by-step algo-

[4]In the tables the values are printed with limited precision, but the complexity is based upon the full numerical precision.

[5]Exactly equal for fields of characteristic 2.

number of errors	$+$	$-$	$\times$	$\div$	$=$	total complexity	bit complexity
0	24.53	0.00	21.03	0.00	0.00	129.69	14.41
1	33.21	9.69	28.66	5.00	8.41	236.44	26.27
2	58.87	21.46	49.12	12.21	25.66	463.91	51.55

(a) Berlekamp-Massey.

number of errors	$+$	$-$	$\times$	$\div$	$=$	total complexity	bit complexity
0	24.53	0.00	21.03	0.00	0.01	129.72	14.41
1	40.52	23.00	48.13	8.00	14.89	388.84	43.20
2	72.21	42.87	73.91	16.14	51.03	718.42	79.82

(b) Euclidean.

number of errors	$+$	$-$	$\times$	$\div$	$=$	total complexity	bit complexity
0	24.53	0.00	21.03	0.00	0.01	129.72	14.41
1	50.44	19.60	90.64	0.00	15.11	568.57	63.17
2	92.14	47.32	194.38	0.00	51.32	1265.33	140.59

(c) High-speed step-by-step.

Table 6.4: Complexity for decoding RS$(7, 3, 5)$.

number of errors	$+$	$-$	$\times$	$\div$	$=$	total complexity	bit complexity
0	12.22	0.00	10.48	0.00	0.00	64.61	4.31
1	18.84	7.73	16.10	5.00	8.47	157.47	10.50

(a) Berlekamp-Massey.

number of errors	$+$	$-$	$\times$	$\div$	$=$	total complexity	bit complexity
0	12.22	0.00	10.48	0.00	0.00	64.61	4.31
1	22.11	15.00	25.51	8.00	8.88	231.28	15.42

(b) Euclidean.

number of errors	$+$	$-$	$\times$	$\div$	$=$	total complexity	bit complexity
0	12.22	0.00	10.48	0.00	0.00	64.61	4.31
1	41.28	15.21	69.34	0.00	31.05	496.34	33.09

(c) High-speed step-by-step.

Table 6.5: Complexity for decoding RS$(7, 5, 3)$.

number of errors	$+$	$-$	$\times$	$\div$	$=$	total complexity	bit complexity
0	498	0	490	0	6	2962	8.98
1	506	14	498	5	13	3072	9.31
2	609	26	572	13	62	3743	11.34
3	685	45	634	24	142	4445	13.47
4	866	70	774	38	287	5855	17.74

(a) Berlekamp-Massey.

number of errors	$+$	$-$	$\times$	$\div$	$=$	total complexity	bit complexity
0	498	0	490	0	6	2962	8.98
1	522	39	537	8	32	3385	10.26
2	653	87	639	17	132	4414	13.38
3	759	132	732	27	292	5562	16.85
4	961	173	893	38	536	7393	22.40

(b) Euclidean.

number of errors	$+$	$-$	$\times$	$\div$	$=$	total complexity	bit complexity
0	498	0	490	0	6	2962	8.98
1	2912	4204	11474	0	276	65314	197.92
2	4467	6895	18513	0	719	106084	321.47
3	6219	9929	26449	0	1358	152470	462.03
4	20149	34070	89589	0	3546	512802	1553.95

(c) High-speed step-by-step.

Table 6.6: Complexity for decoding RS$(63, 55, 9)$.

number of errors	$+$	$-$	$\times$	$\div$	$=$	total complexity	bit complexity
0	8115	0	8086	0	50	48696	27.30
1	8159	38	8126	5	58	49022	27.48
2	8475	51	8354	13	166	50858	28.51
3	8875	70	8653	24	394	53508	29.99
4	9417	95	9074	38	766	57368	32.16
5	10303	126	9782	55	1344	63647	35.68
6	11096	163	10441	75	2088	70103	39.30
7	12260	205	11425	98	3049	79227	44.41
8	12912	254	12005	124	4127	86189	48.31
9	13740	309	12728	153	5358	94526	52.99
10	13877	369	12879	184	6626	99441	55.74
11	16338	435	15012	218	8279	117763	66.01
12	17019	509	15657	257	10033	127199	71.30
13	18689	585	17150	296	12029	142591	79.93
14	19479	670	17872	341	14164	153709	86.16
15	21750	760	19934	387	16587	173877	97.46
16	23987	855	21966	435	19294	194727	109.15

(a) Berlekamp-Massey.

Table 6.7: Complexity for decoding RS$(255, 223, 33)$.

number of errors	$+$	$-$	$\times$	$\div$	$=$	total complexity	bit complexity
0	8115	0	8086	0	50	48696	27.30
1	8222	135	8285	8	149	50265	28.18
2	8687	328	8663	17	524	53985	30.26
3	9234	492	9110	27	1190	58979	33.06
4	9920	655	9674	38	2165	65632	36.79
5	10937	810	10514	50	3496	75054	42.07
6	11857	966	11302	63	5139	85064	47.68
7	13148	1126	12413	77	7143	98154	55.02
8	13915	1279	13108	92	9393	109375	61.31
9	14851	1427	13940	108	11915	122260	68.53
10	15097	1580	14200	125	14594	132083	74.04
11	17656	1722	16429	143	17762	155523	87.18
12	18403	1850	17144	161	21104	170090	95.34
13	20184	2011	18747	181	24804	191251	107.20
14	21064	2157	19559	203	28736	208238	116.73
15	23401	2291	21689	224	33021	234320	131.35
16	25699	2417	23781	247	37649	261204	146.41

(b) Euclidean.

Table 6.7: Complexity for decoding RS$(255, 223, 33)$.

rithm [Massey, 1965] it is not well suited to decoding multi-level codes over large alphabets. For these reasons further work on HSSBS was not pursued.

The average decoding complexity is a function of the probability of symbol errors, which in turn is dependent on the modulation scheme, the type of noise and the ratio E_b/N_0. Figure 6.3 compares the decoding complexities as a function of E_b/N_0 for the RS$(7,5,3)$ code over a coherently-demodulated BPSK channel in the presence of AWGN. At high values of E_b/N_0, where received symbol errors are uncommon, the complexity is almost constant and is dominated by the cost of the syndrome calculation.

6.4 Two-Stage Decoding

6.4.1 Decoder Performance

The decoding performance of TSD has been evaluated by computer simulation for RS$(7,3,5)$ and RS$(7,5,3)$ codes. For each case the performance of both HD and SD subtrellis prediction was measured. The simulated modulation scheme was non-coherently demodulated 8FSK, over an AWGN channel. Unlike the other simulations Euclidean distance metrics were used, represented by floating-point values.

Figure 6.4 shows the performance of RS$(7,3,5)$ with HD choice of subtrellis. The goal of this near-optimal decoder is to improve upon the performance of HDMLD, but with a complexity lower than SDMLD. To obtain an appreciable coding gain the majority of the subtrellises must be decoded (≈ 6) so that the complexity saving is reduced. (It should be noted however that a less noisy channel is required for HDMLD

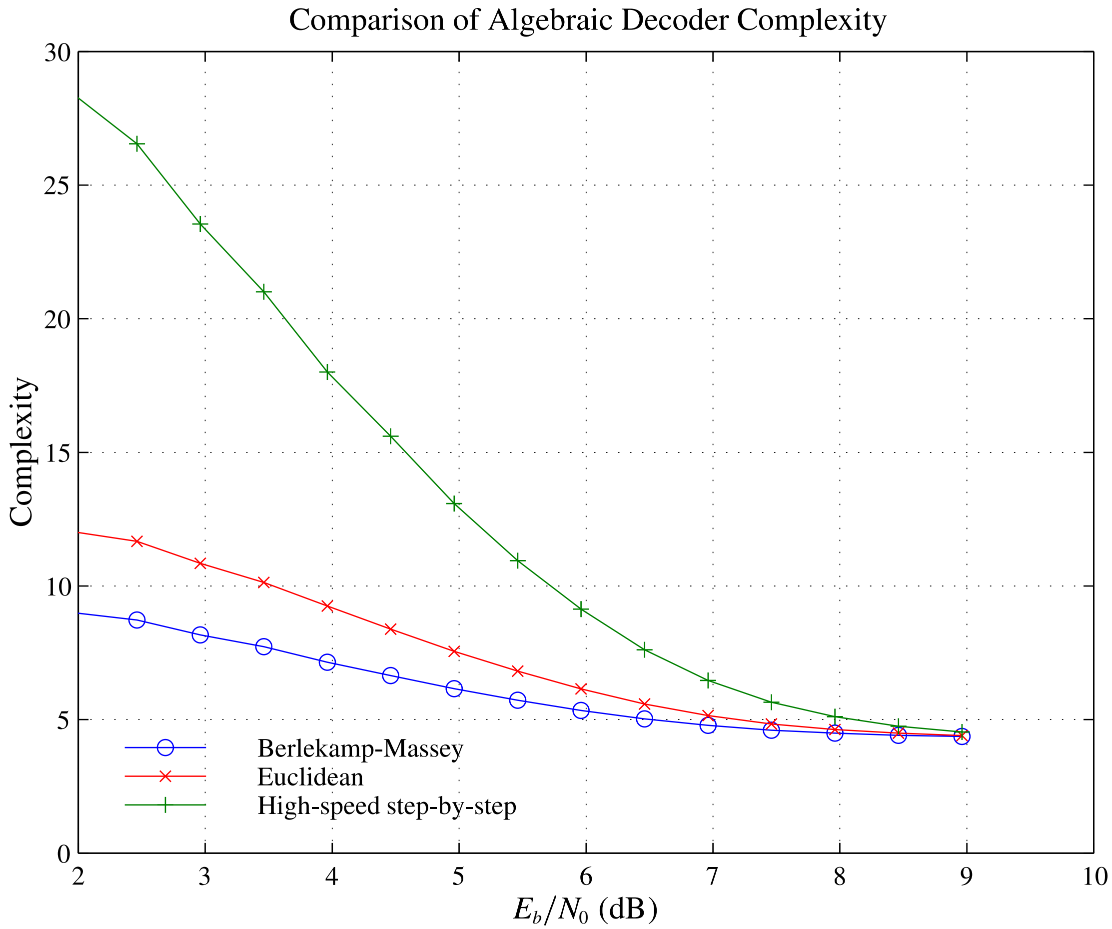


Figure 6.3: Algebraic decoding complexity for RS(7, 5, 3) over a coherently-demodulated BPSK channel.

to give *any* coding gain.) Figure 6.5 shows the performance when SD information is incorporated into the subtrellis selection, by using "soft GF" algebra. The error-correction performance is much improved, solely as a result of more reliable subtrellis predictions. Decoding just 3 of the 8 subtrellises results in a performance very close to optimum.

Similarly Figures 6.6 and 6.7 show the performance of RS$(7, 5, 3)$ for HD and SD choice of subtrellis respectively. For this code subsets of the trellis are selected based on the prediction of two information symbols, so it is of little surprise that $> q$ symbols are required for good performance. (With $\leq q$ symbols there may not be the opportunity to try a second symbol value for the more confident prediction.) Again, evaluation of the subtrellis predictors with soft GF arithmetic considerably improves the result.

6.4.2 Decoder Complexity

Two-stage decoding has both algebraic and combinatorial operations. The first stage uses GF arithmetic. By inspection of the subtrellis symbol predictors the number of addition and multiplication operations is easily obtained. Since the subtrellis predictors are known at design time, the prediction process can easily be optimised to remove unnecessary multiplications when the coefficient is 1. By inspection of Table 4.2 the first decoding stage for RS$(7, 3, 5)$ needs 52 GF additions and 67 GF multiplications. Thus the complexity for stage 1 is $52 \times 1 + 67 \times 5 = 387$ CPU cycles.

The second decoding stage is the Viterbi decoding of N_{st} subtrellises. It is assumed that the trellis will be decoded using the log likelihood values, which may be expressed either in floating-point numbers or converted to integer metrics (Section 5.1.2). For a coset trellis, such as Figure 4.5, the decoding implementation and calculation of its complexity is relatively straightforward. Each subtrellis is independent of the others so they can be decoded in isolation (in parallel even, if required).

Although the two-stage decoding simulations did not use LL metrics, the complexity analysis presented below is for the case of LL metrics. This enables the complexity measurements to be compared with the other decoders simulated in this Chapter.

Table 6.8 shows the complexity to decode the RS$(7, 3, 5)$ coset trellis (Figure 4.5), for the case of HD subtrellis prediction. Note that selecting the best subtrellis from the N_{st} decoded subtrellises requires $N_{\mathrm{st}} - 1$ comparisons. Table 6.8 includes the additional $N_{\mathrm{st}} - 1$ comparisons. The bit complexity to decode the full minimal RS$(7, 3, 5)$ trellis is 362.33 (Table 6.2); for $N_{\mathrm{st}} \leq 7$ decoding is simplified.

For a syndrome trellis, such as Figure 4.2, the decoding complexity is more difficult to analyse because the trellis does not contain independent subtrellises. Prior knowledge (or prediction) of information symbols can, however, be used to limit decoding to only a subset of the trellis. The trellis vertices at depth 3 can be thought of as storing the values of u_2 and u_3. From the generator matrix (4.22) it can be seen that the branches $B(S_{3,i} \rightarrow S_{4,j})$ are labelled with

$$v_4 = u_2 + \alpha^4 u_3 + \alpha^3 u_4 \tag{6.23}$$

The new information, responsible for the trellis branching, is u_4, while the information from past subcodes (u_2 and u_3) was effectively stored in the vertex number. When the

N_{st}	$+$	$-$	$\times$	$\div$	$=$	stage 2 complexity	TSD total	TSD total per bit
1	272	0	0	0	63	461	848	94.22
2	544	0	0	0	127	925	1312	145.78
3	816	0	0	0	191	1389	1776	197.33
4	1088	0	0	0	255	1853	2240	248.89
5	1360	0	0	0	319	2317	2704	300.44
6	1632	0	0	0	383	2781	3168	352.00
7	1904	0	0	0	447	3245	3632	403.56
8	2176	0	0	0	511	3709	4096	455.11

Table 6.8: Complexity versus number of subtrellises decoded for TSD of RS$(7, 3, 5)$.

values for u_2 and u_3 are known (or can be predicted) only those paths which pass through the relevant vertex at depth 3 need be decoded. This technique is similar to *forced-state decoding* or *state pinning* [Hagenauer *et al.*, 1994, p. 245], though with the aim of reducing decoder complexity instead of increasing the error-correction performance of a feedback decoder.

Figure 6.8 shows the possible trellis paths for the case $u_2 = 0$ and $u_3 = 0$, while the possible paths for the case $u_2 = 1$ and $u_3 = 2$ are shown in Figure 6.9. Some branches are common whatever values of u_2 and u_3 are selected. This is true for branches at depths 1, 6 and 7, which require decoding only once. Figure 6.10 highlights the common branches. In the worst case, as shown, no branches at depths $2, 3, \ldots, 5$ are common. The complexity of decoding depths $2, 3, \ldots, 5$ for one combination of u_3, u_4 has been considered as $\frac{1}{64}$ of the total complexity of decoding depths $2, 3, \ldots,$ 5. When 8 or more combinations of u_3, u_4 are decoded some of the central branches are guaranteed to be common, so that the worst-case analysis over-emphasizes the complexity. The number of GF additions required for the prediction of u_2 and u_3 are 36 and 40, respectively, while the number of multiplications are 39 and 44, respectively. Table 6.9 shows the decoding complexity using the worst-case analysis for selected values of N_{st}.

N_{st}	$+$	$-$	$\times$	$\div$	$=$	stage 2 complexity	TSD total	TSD total per bit
1	97	0	0	0	28.88	183.62	674.62	44.98
2	122	0	0	0	51.75	277.25	768.25	51.22
8	272	0	0	0	189.00	839.00	1330.00	88.67
16	472	0	0	0	372.00	1588.00	2079.00	138.60
24	672	0	0	0	555.00	2337.00	2828.00	188.53
32	872	0	0	0	738.00	3086.00	3577.00	238.47
44	1172	0	0	0	1012.50	4209.50	4700.50	313.37
56	1472	0	0	0	1287.00	5333.00	5824.00	388.27
64	1672	0	0	0	1470.00	6082.00	6573.00	438.20

Table 6.9: Complexity versus number of subtrellises decoded for TSD of $\mathrm{RS}(7,5,3)$.

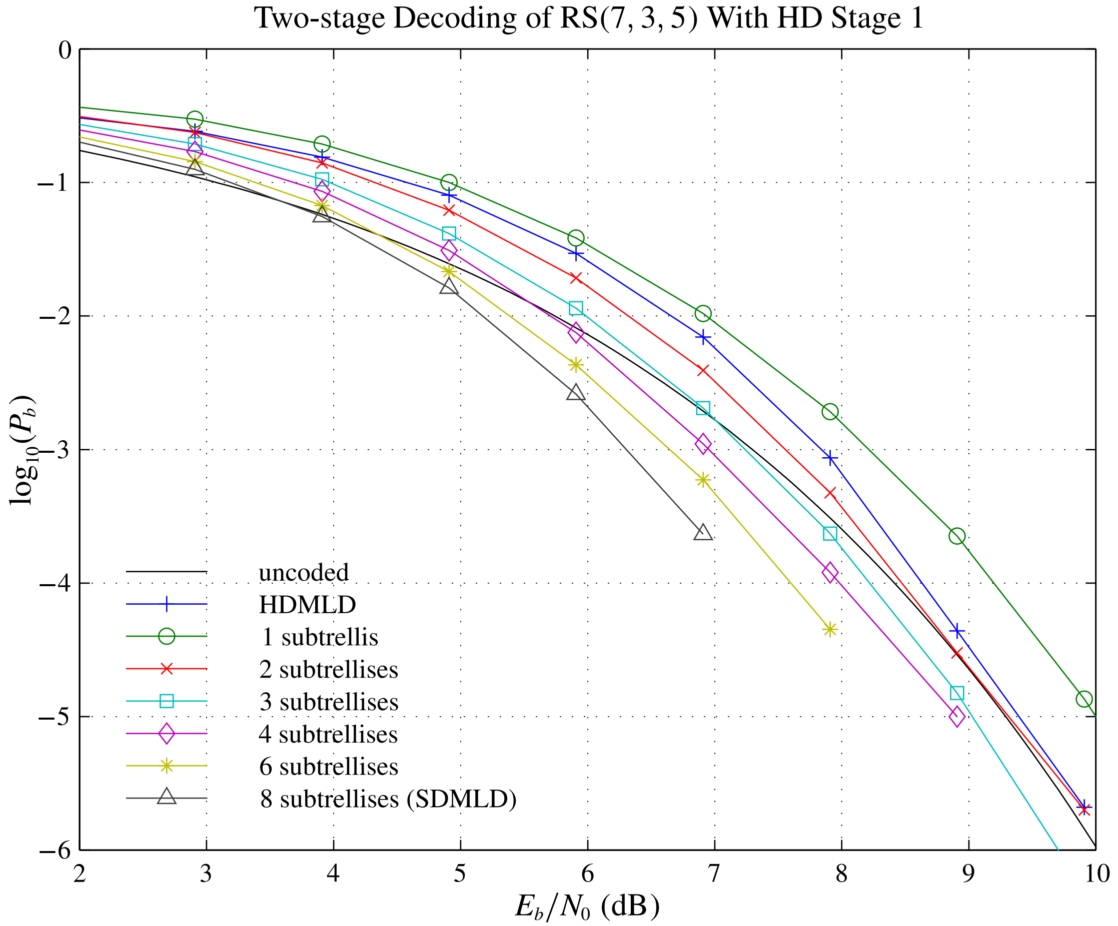


Figure 6.4: Two-stage decoding of RS(7, 3, 5), with HD choice of subtrellis, over 8-FSK channel.

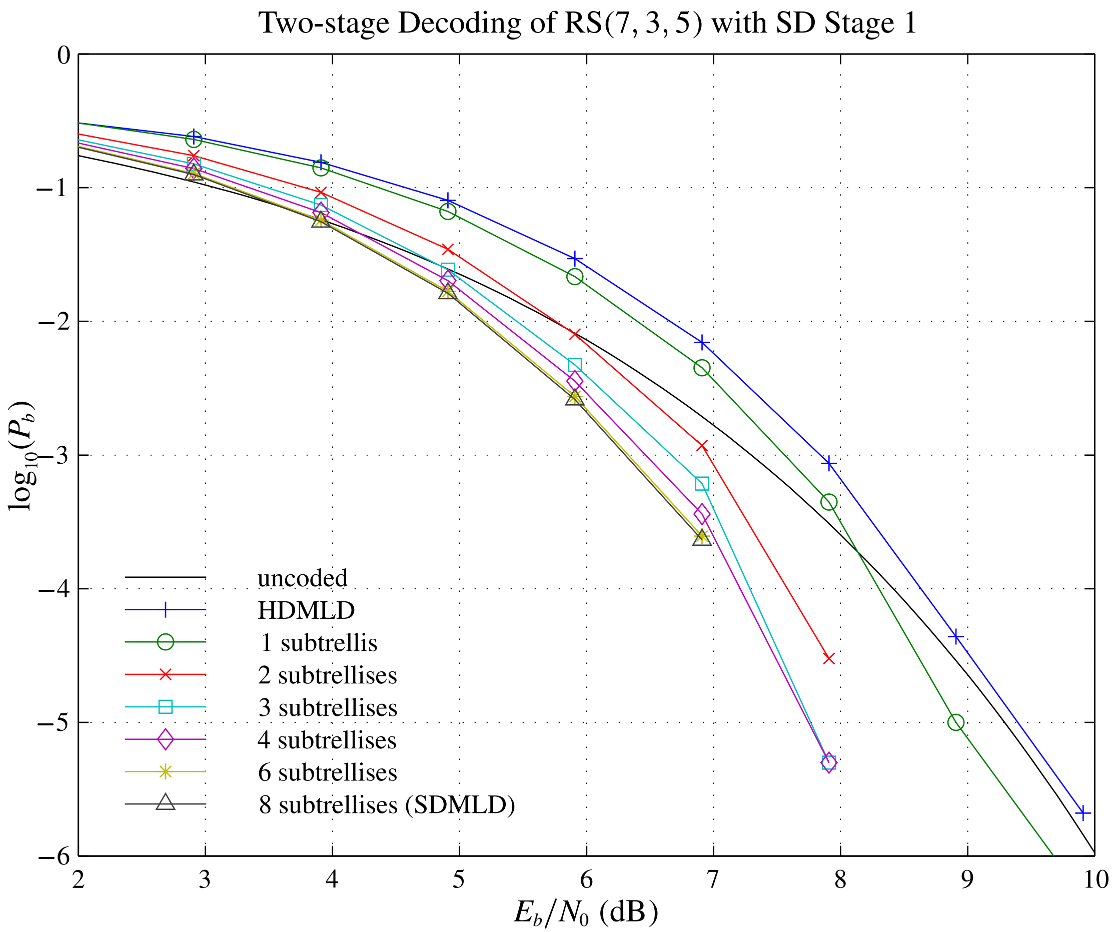


Figure 6.5: Two-stage decoding of RS$(7, 3, 5)$, with SD choice of subtrellis, over 8-FSK channel.

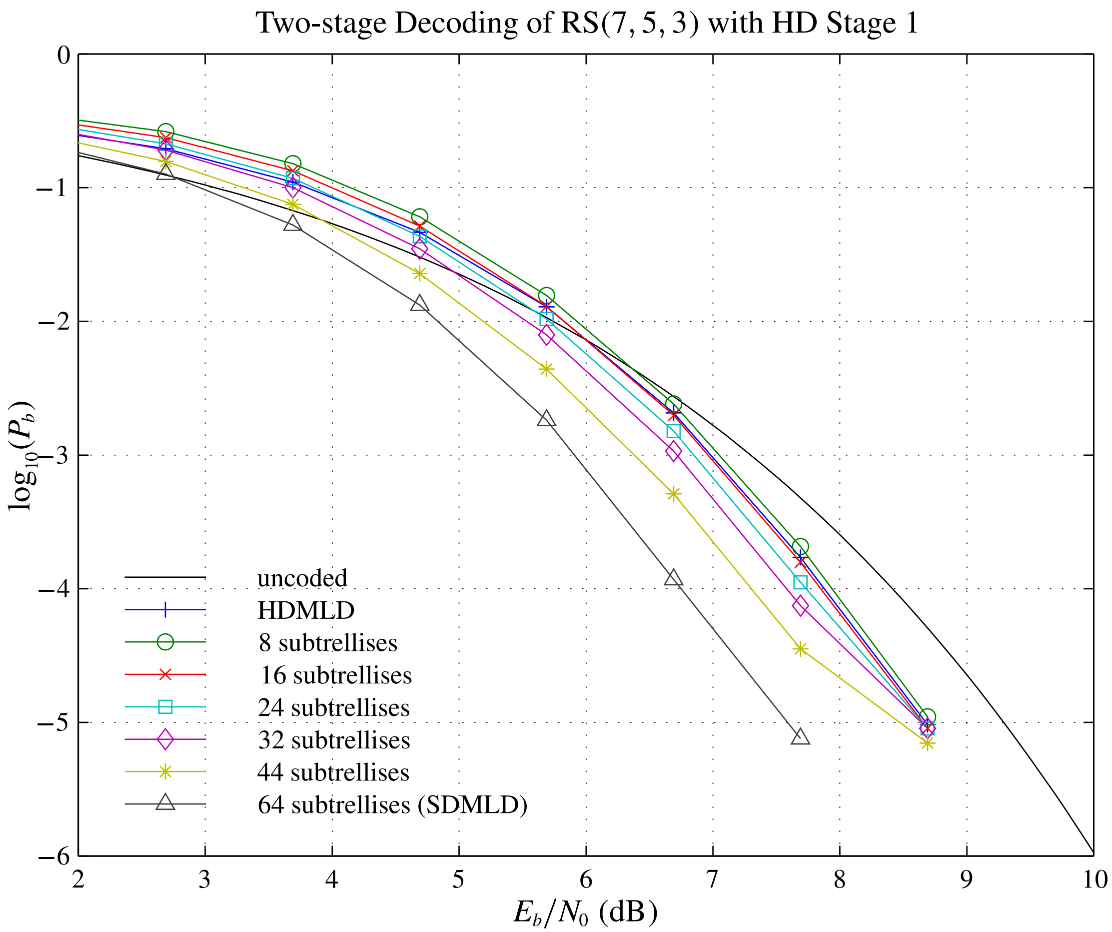


Figure 6.6: Two-stage decoding of RS(7, 5, 3), with HD choice of subtrellis, over 8-FSK channel.

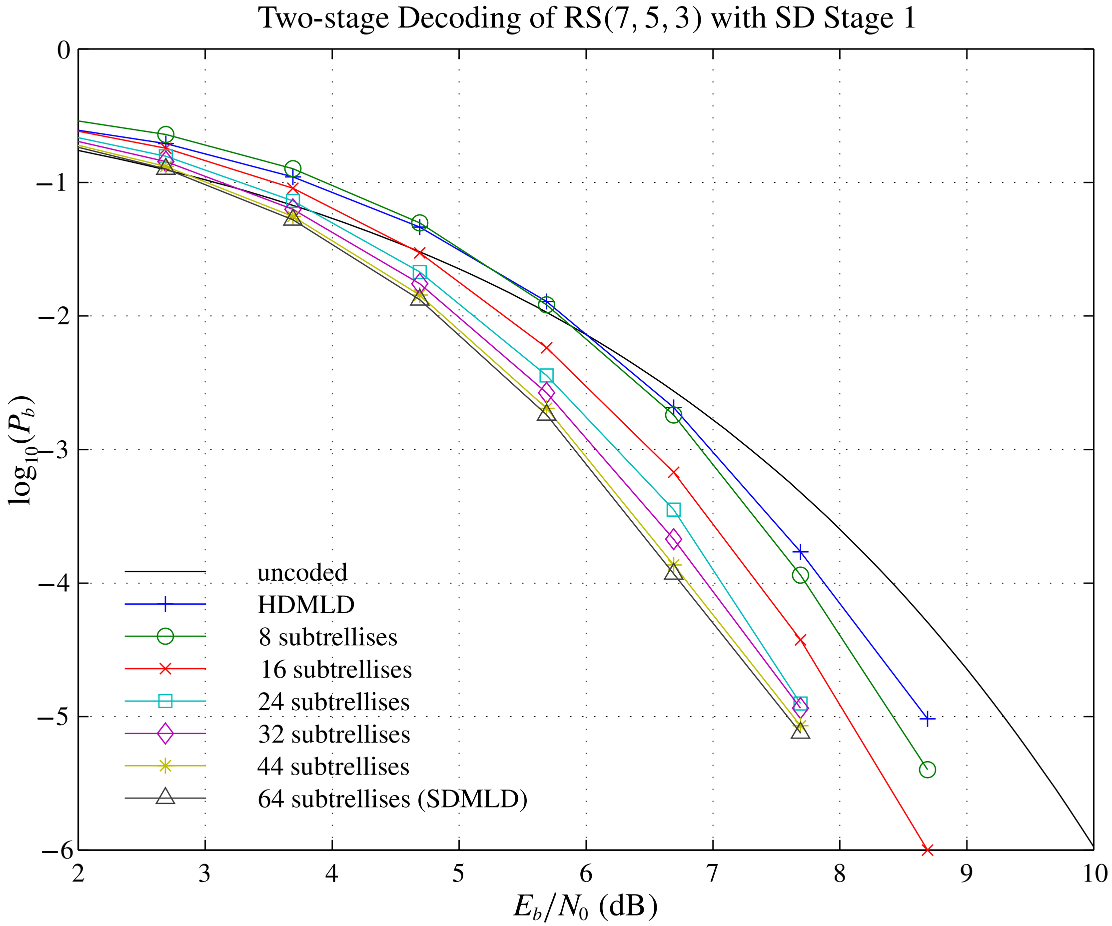


Figure 6.7: Two-stage decoding of RS(7, 5, 3), with SD choice of subtrellis, over 8-FSK channel.

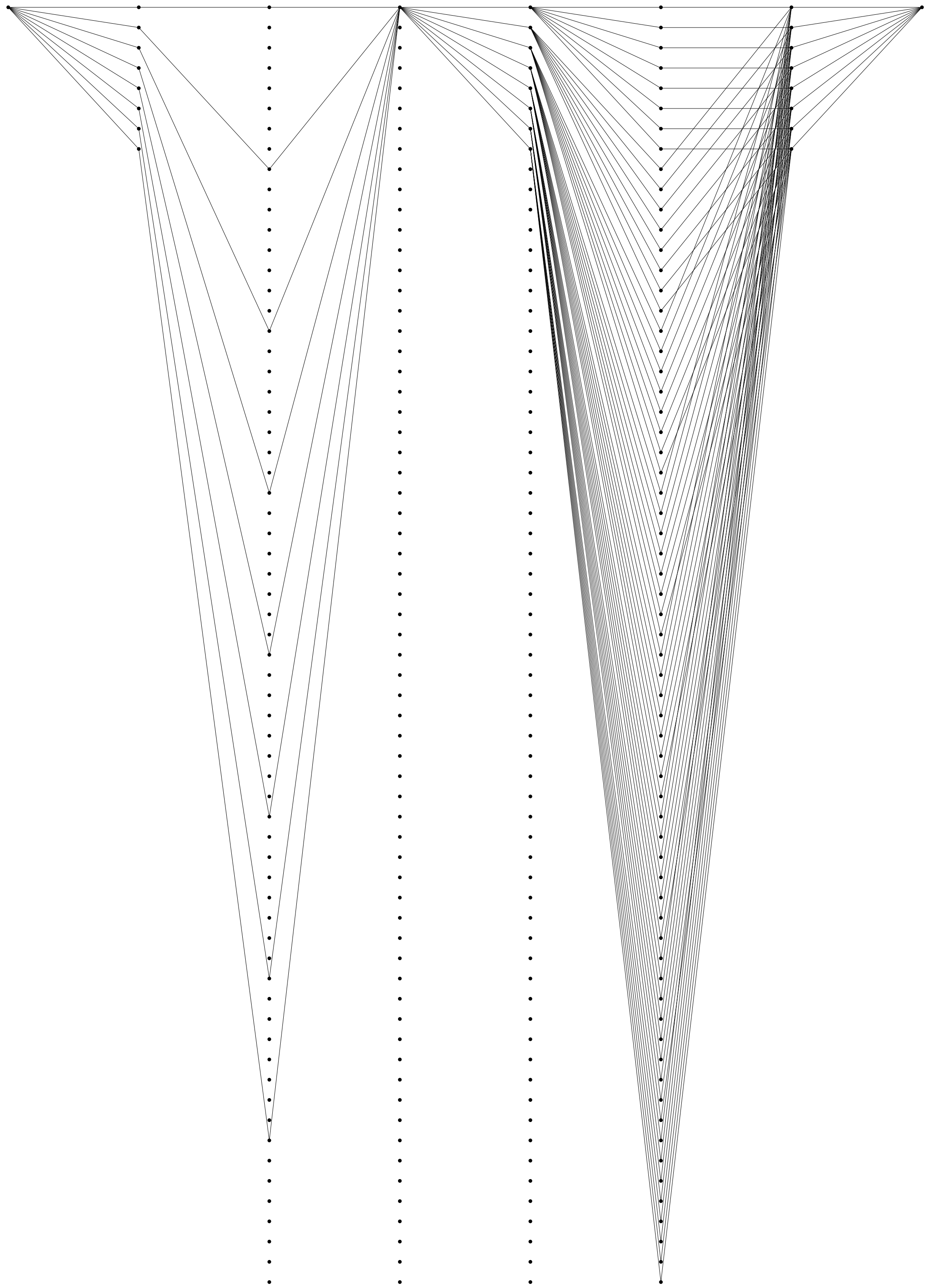


Figure 6.8: Subset of RS$(7, 5, 3)$ trellis for $u_2 = 0, u_3 = 0$.

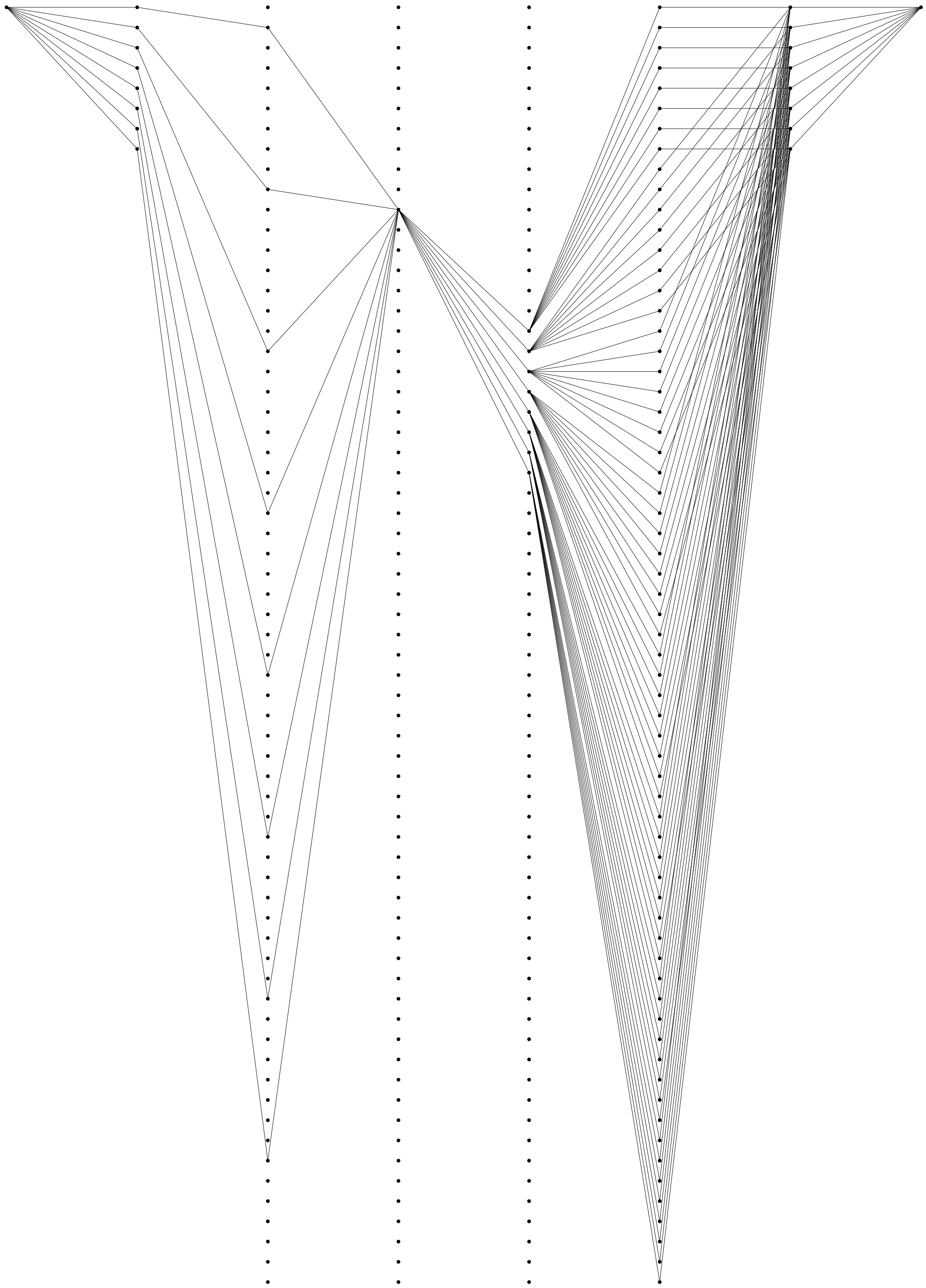


Figure 6.9: Subset of RS$(7, 5, 3)$ trellis for $u_2 = 1, u_3 = 2$.

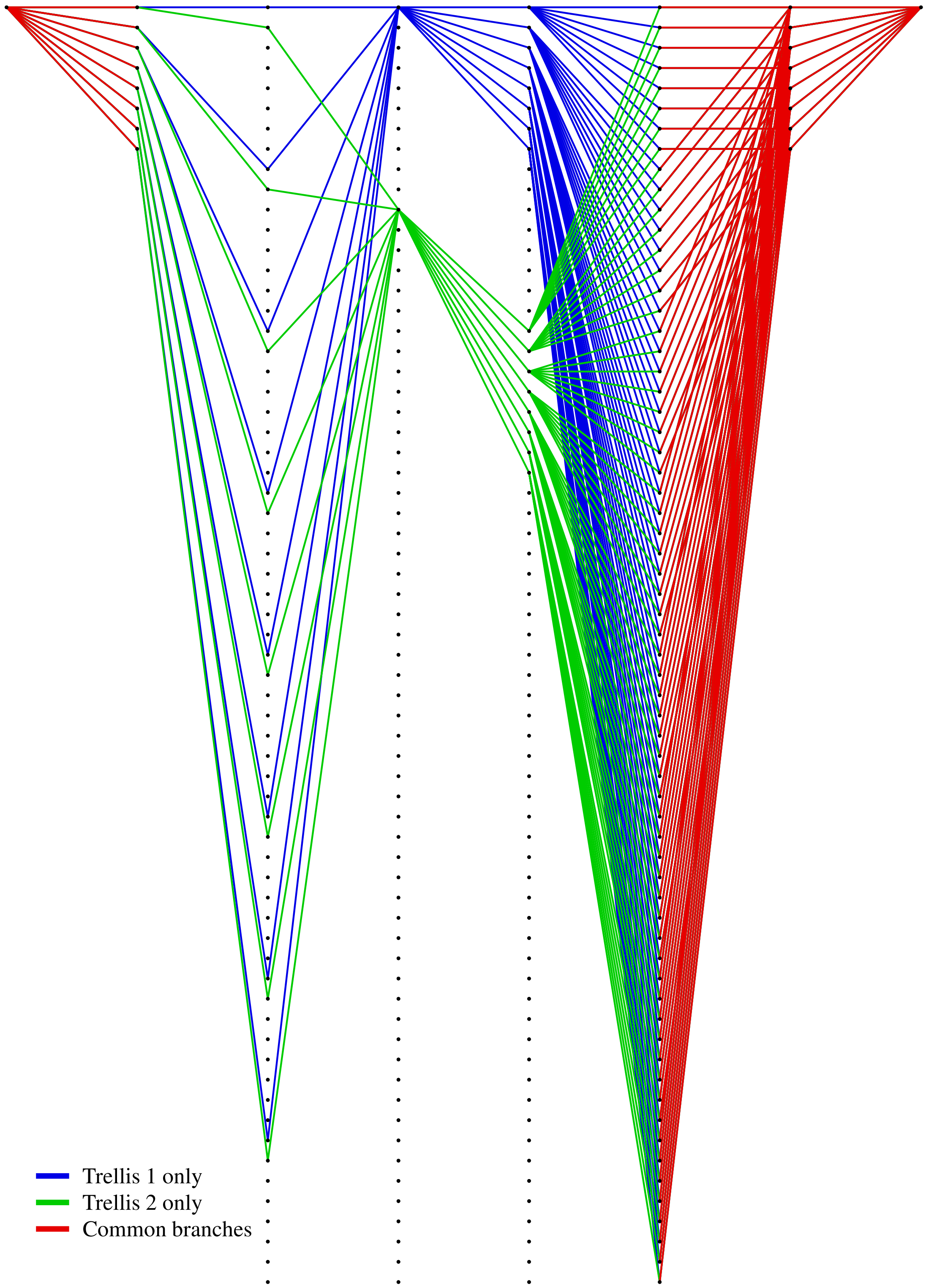


Figure 6.10: Subsets of RS$(7, 5, 3)$ trellis for $u_2 = \{0, 1\}$, $u_3 = \{0, 2\}$.

6.5 SOVA Applied to the Meteosat II Satellite System

6.5.1 Introduction

The European Space Agency is funding a replacement series of geosynchronous satellites, Meteosat II, which will be used for meteorological purposes. The satellites will be responsible for transmission of weather images to a ground station. The processed information is then transmitted back up to the satellites for retransmission to fee-paying end users. In its simplest form the Meteosat II – Earth retransmission channel can be viewed as a concatenated coding system, with an $\mathrm{RS}(255, 223, 33)$ outer code and a $(2, 1, 7)$ convolutional inner code as shown in Figure 2.3. An interleaver of depth $d = 4$ is used between inner and outer codes. The proposed system also includes encryption, compression, randomisation and synchronisation. None of these affect the performance of the error control coding and can be ignored. This concatenated code is also used by NASA for the Planetary Data Standard.

Potential users are spread over a wide geographical area. Users at the edges of the defined service area are those most likely to have most difficulty in reception. There is of course a trade-off between the cost of increasing the transmitter power, increasing the complexity of the users' receiving equipment and the defined geographical limits for which satisfactory reception can be expected. If a means can be found to improve the channel error-rate for users in marginal locations greater commercial benefits exist, either by enlarging the service area or reducing the transmitter power. An investigation was made into the benefits of applying SOVA to improve the system performance. Replacing the inner Viterbi decoder with a soft-output Viterbi decoder allowed SD decoding of the outer (RS) code. The original system is denoted by *SD-HD* and the modified one by *SD-SD*.

6.5.2 Simulated System Details

The performance of SOVA in the Meteosat II – Earth high/low rate user station link was measured by computer simulation. An overview of the simulated system is shown in Figure 2.3 and the specifications are given in Table 6.10. The differences between the simulated and actual systems are discussed below.

Convolutional code:	n	2 bits
	k	1 bit
	K	7
	generator polynomial:	
	$g_1(x)$	$1+x+x^2+x^3+x^6$
	$g_2(x)$	$1+x^2+x^3+x^5+x^6$
	trellis length	28, 35 or 42
RS code:	n	7 symbols
	k	5 symbols
	d	3 symbols
	$g(x)$	$1+\alpha^4 x+\alpha^3 x^2$
	GF size	8
	GF primitive polynomial	$1+x$
	symbol width	3 bits
Block interleaver:	depth	4
	width	7 RS symbols (21 bits)
Modulation/ demodulation:	type	BPSK
	channel	AWGN
	demodulation	coherent
	quantisation resolution	3 bits

Table 6.10: Specifications of simulated Meteosat II system.

Modulation/Demodulation

The simulated system modelled the transmission of equally-likely random data over a BSC with AWGN and coherent BPSK demodulation. The system proposed by ESA is switchable between BPSK and QPSK modulation. However, BPSK and QPSK have identical BER performance [Sklar, 1988, p. 172] so only BPSK modulation was modelled. The simulated demodulator output was quantised into 8 levels.

Synchronisation

The simulated system assumed synchronisation. This assumption can be made since without synchronisation no error control coding can be applied. Whilst the error control decoders are important for ensuring and maintaining synchronisation, introduction of SOVA will not worsen the synchronisation behaviour.

Data randomisation

Data randomisation is recommended for the following reasons [Dai, 1995, p. 3.2-18]:

- adequate symbol transition density in the data stream
- smooth spectrum shaping
- with QPSK modulation: standard I/Q demultiplexing scheme possible
- does not introduce degradation
- reliable demodulator/synchroniser performance

Data randomisation was not included in the simulated system since synchronisation was assumed. The reasons for including data randomisation are for synchronisation purposes only, it has no effect on the error control codes.

Convolutional Code and Decoder

The convolutional code used for the computer simulation was the $(2, 1, 7)$ code specified by the CCSDS. The decoder used was either a SOVA decoder, or a conventional VA trellis decoder, both operated in the trace-back implementation. Decoding was performed over trellises of depths 28 ($4K$), 35 ($5K$) and 42 ($6K$). A single section of the trellis is shown in Figure 6.11. The quantisation resolution modelled was 8 levels, the maximum resolution given by the frame synchroniser output [Dai, 1995, p. 3.2-13]. Greater resolution would provide little extra performance, simulation studies [Heller and Jacobs, 1971] have shown that 8 level quantisation resulted in only 0.25 dB reduction in coding gain with respect to the unquantised case.

Reed-Solomon Code and Decoder

To measure the coding gain SOVA can produce it is necessary that the RS decoder used is capable of SDMLD. Failing to use such a decoder will not produce an independent measure of the coding gain possible by introducing SOVA, but instead a combination of the gain by SOVA and the loss from the sub-optimal decoder.

For the simulation the RS$(255, 223, 33)$ code over GF(256) was replaced by the RS$(7, 5, 3)$ code over GF(8). This code was chosen because it is a similar rate to RS$(255, 223, 33)$ and readily decoded with VA over the trellis shown in Figure 4.2.

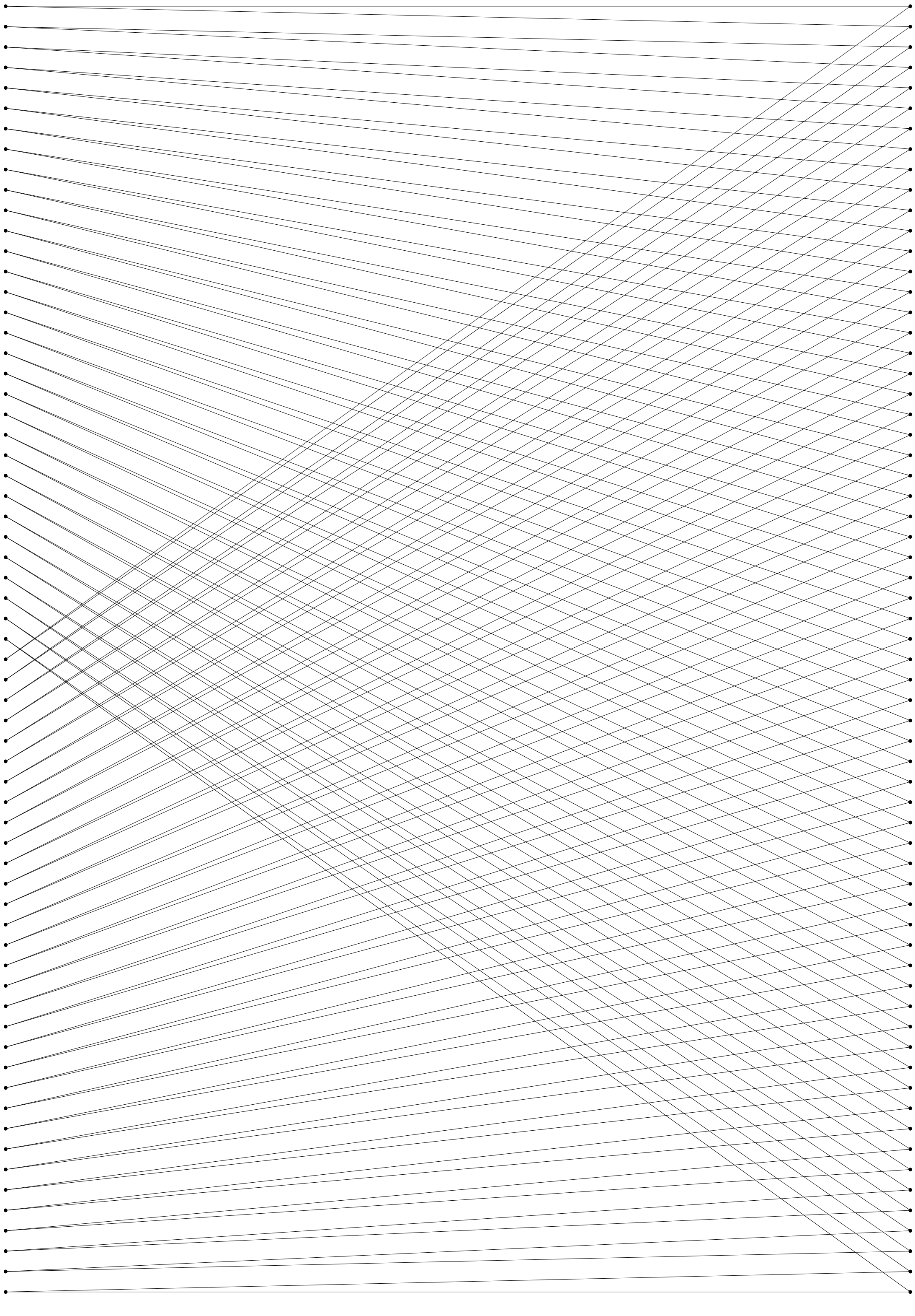


Figure 6.11: One section of the (2, 1, 7) convolutional code trellis.

The trellis was labelled with the binary mapping of the RS symbols (using polynomial representation), thus avoiding the need to map symbols and symbol reliabilities from binary to GF(8). The RS Viterbi decoder operated in trace-back mode. For HD decoding of the outer code a Berlekamp-Massey decoder was used.

Interleaver and De-interleaver

The interleaver/de-interleaver specified by the CCSDS has depth $d = 4$. The width of the interleaver was reduced from 255 8-bit symbols to 7 3-bit symbols to conform with the change of outer (RS) code. The interleaver was arranged to operate on the RS symbols, not on individual bits, and thereby preserved the burst-error correction capability of the RS code.

6.5.3 Simulation Results

Results from the simulated system are presented over the E_b/N_0 range 0–7 dB. The CCSDS coding standard does not specify the path storage to be used for a Viterbi decoder (nor even the decoding method to use!). It is assumed that the trellis length will be in the range $4K$ to $6K$ [Heller and Jacobs, 1971]. Results are given for convolutional trellis lengths 28 ($4K$), 35 ($5K$) and 42 ($6K$).

The concatenated coding scheme provides extremely high error-correction capability above 4.5 dB, requiring some results to be extrapolated. On the graphs dotted lines indicate results obtained from extrapolated data. The uncoded curve was calculated by theoretical means, and is in agreement with the simulated uncoded curve (not shown). The probability of bit error for uncoded data transmitted with BPSK over a

BSC containing AWGN is given by [Sklar, 1988, p. 166]

$$P_b = Q\left(\sqrt{\frac{2E_b}{N_0}}\right) \tag{6.24}$$

where $Q(x)$ is the complementary error function.

	Additional coding gain (dB)		
BER	depth = 28	depth = 35	depth = 42
10^{-4}	0.7	0.9	1.0
10^{-5}	0.9	1.1	1.4
10^{-6}	1.1	1.5	1.8

Table 6.11: Additional coding gain achieved through use of SOVA.

Figure 6.12 shows the performance of both SOVA and VA decoding for a convolutional trellis of depth 28. At a BER of 10^{-5} SOVA decoding provides a coding gain increase of approximately 0.9 dB. The "break-point" at which coding becomes beneficial is reduced from $E_b/N_0 = 3.6$ dB to $E_b/N_0 = 3.1$ dB. The same results for a convolutional trellis of depth 35 are shown in Figure 6.13. At a BER of 10^{-5} SOVA displays a coding gain over the VA of 1.1 dB, while the break-point is reduced from $E_b/N_0 = 3.4$ dB to $E_b/N_0 = 2.8$ dB. For a trellis of depth 42 (Figure 6.14), the difference in coding gain has risen to 1.4 dB at a BER of 10^{-5}. The break-point is reduced from $E_b/N_0 = 3.3$ dB to $E_b/N_0 < 2.8$ dB.

6.5.4 Decoder Complexity

The total system complexity was calculated from the sum of the complexities for each of the component decoders. Only operations directly associated with decoding were

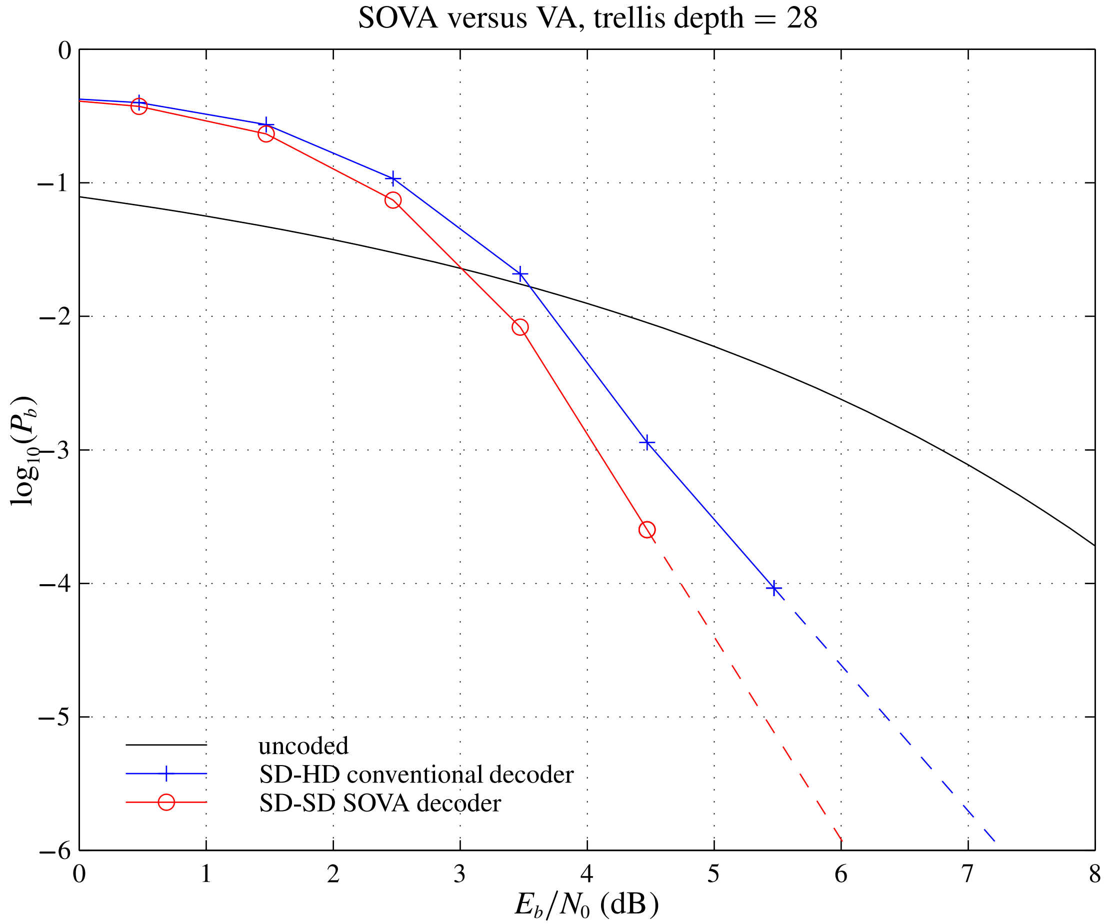


Figure 6.12: Simulation results for RS$(7, 5, 3)$ and $(2, 1, 7)$ over a trellis of depth 28.

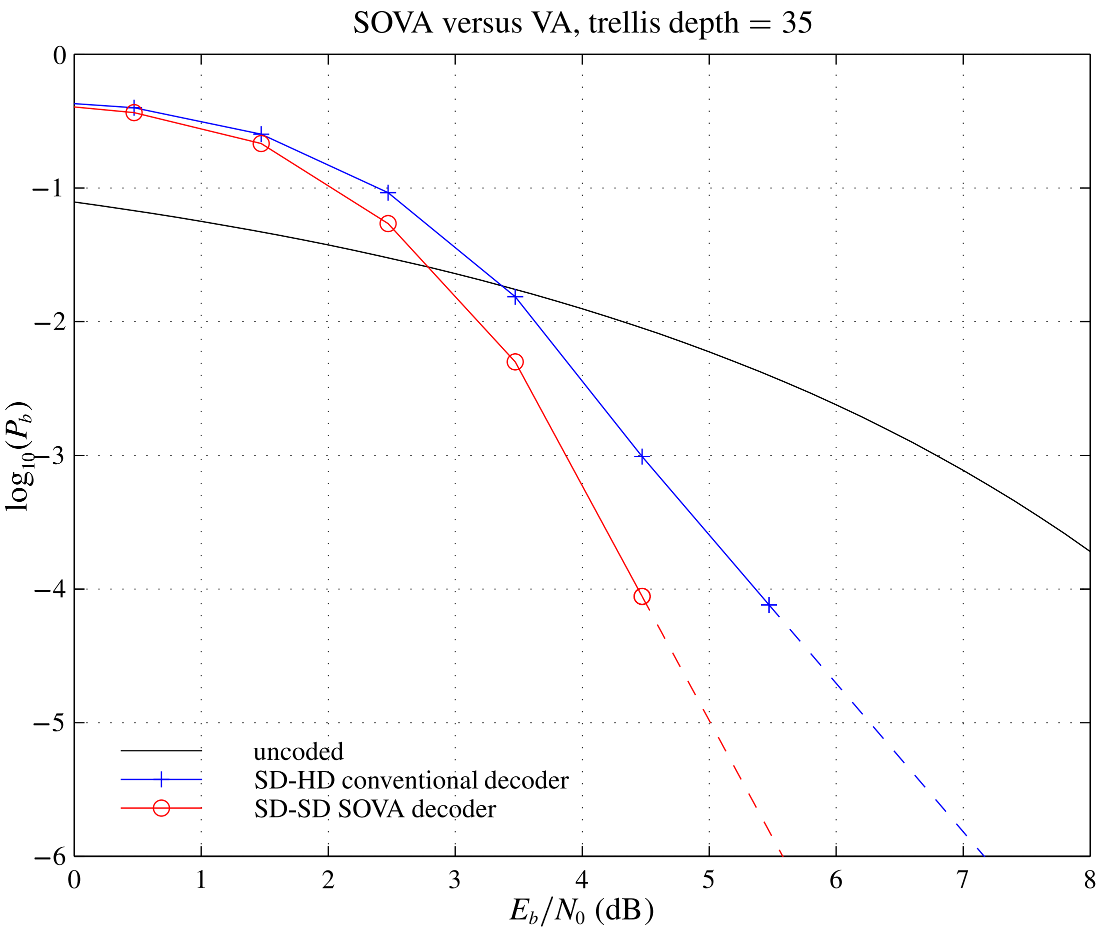


Figure 6.13: Simulation results for RS$(7, 5, 3)$ and $(2, 1, 7)$ over a trellis of depth 35.

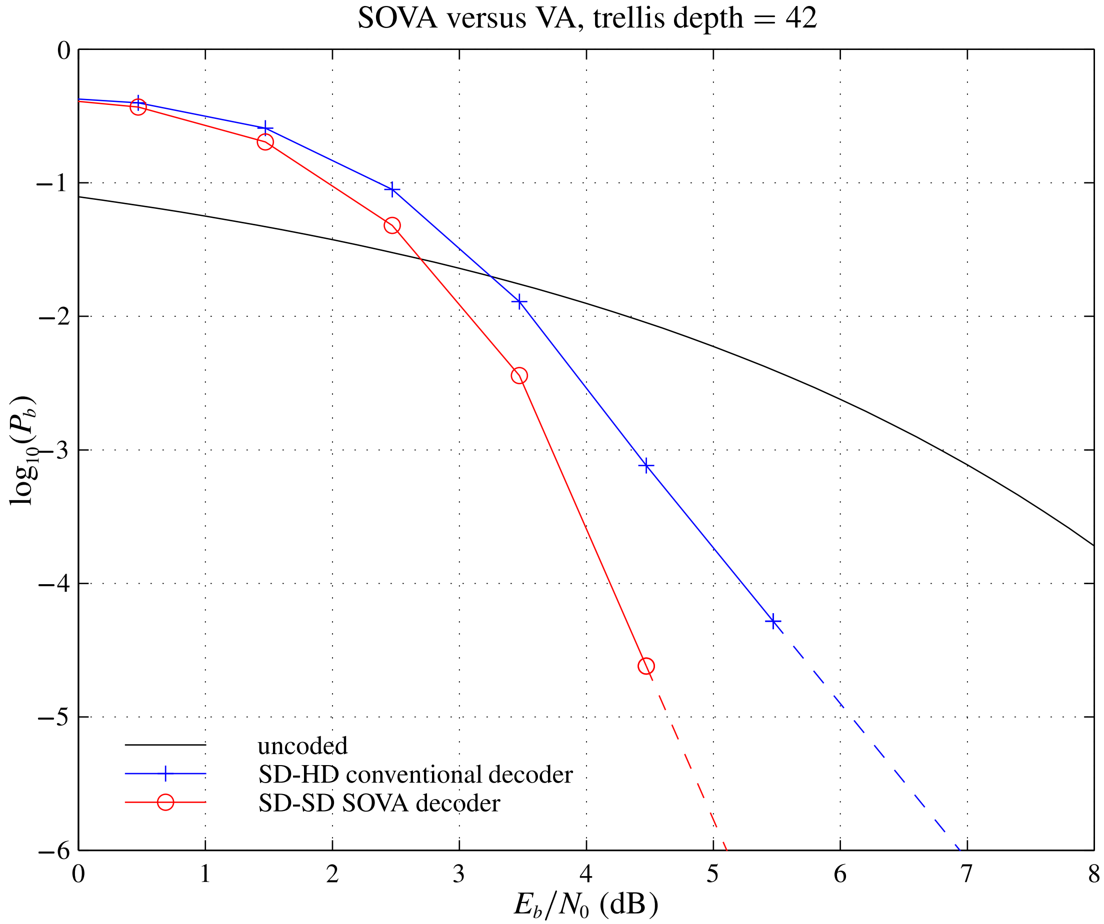


Figure 6.14: Simulation results for RS$(7, 5, 3)$ and $(2, 1, 7)$ over a trellis of depth 42.

included. Additional work, such as that performed by the de-interleaver, was termed overheads because the exact complexity is very dependent on the exact hardware or software implementation.

The decoding complexity for the inner $(2, 1, 7)$ convolutional code was calculated using the procedure given in Section 6.2. Table 6.12 details the VA complexity for trellises truncated to depths 28, 35 and 42. Similarly, the decoding complexity using SOVA is shown in Table 6.13. Each decoding operation produces just one binary bit of information.

The complexity for VA and decoding of the outer $\mathrm{RS}(7, 5, 3)$ code was calculated in Example 6.2. A BM decoder was used for HD decoding of $\mathrm{RS}(7, 5, 3)$, for which the complexity is given in Figure 6.3. The complexity of the BM decoder is dependent upon E_b/N_0; the results are quoted for an output BER of 10^{-4}.

The complexity of decoding the outer $\mathrm{RS}(7, 5, 3)$ code with the VA can be obtained from Table 6.3, while the complexity for the original BM decoder can be calculated from Figures 6.2 and 6.3. At a BER of 10^{-4} the bit complexity of the BM decoder is 4.52. Each outer decoding operation results in 15 binary bits of data. On average, $\frac{n_2}{k_2} = \frac{7}{5}$ inner decoding operations are required for every (binary) data bit output by the concatenated system. The total complexity, for both the original system (SD-HD) and the improved system (SD-SD) is shown in Table 6.14.

trellis length	$+$	$-$	$\times$	$\div$	$=$	total	bit complexity
28	3808	0	0	0	1792	9184	9184
35	4760	0	0	0	2240	11480	11480
42	5712	0	0	0	2688	13776	13776

Table 6.12: Complexity for VA decoding of the convolutional $(2, 1, 7)$ code.

trellis length	$+$	$-$	$\times$	$\div$	$=$	total	bit complexity
28	3808	1792	0	0	1792	10976	10976
35	4760	2240	0	0	2240	13720	13720
42	5712	2688	0	0	2688	16464	16464

Table 6.13: Complexity for SOVA decoding of the convolutional $(2, 1, 7)$ code.

trellis	bit complexity	
length	SD-HD	SD-SD
28	12862	15759
35	16077	19601
42	19291	23442

Table 6.14: Comparison of decoding complexity for the Meteosat II concatenated coding scheme.

6.6 RS Product Code Decoding

6.6.1 The Transmission Channel

The transmission channel used in the simulated system was a coherently-demodulated binary phase-shift keying channel with additive white Gaussian noise, transmitting equally-likely data. At the demodulator the soft outputs were quantised to 8 levels. To simplify the implementation the $\mathrm{RS}(7,5,3) \times \mathrm{RS}(7,5,3)$ product code operated on RS symbols, not binary bits. Unlike the SOVA system described in Section 6.5 it was therefore not possible to use a binary-mapped RS trellis. However the same binary mapping and combination of LL metrics which would have been performed by the trellis was instead performed at the output of the demodulator. This mapping is shown in Table 6.15.

GF(8) value	bit representation	LL metric construction
0	000	$\ell_{0_1} + \ell_{0_2} + \ell_{0_3}$
1	001	$\ell_{0_1} + \ell_{0_2} + \ell_{1_3}$
α	010	$\ell_{0_1} + \ell_{1_2} + \ell_{0_3}$
α^2	011	$\ell_{0_1} + \ell_{1_2} + \ell_{1_3}$
α^3	100	$\ell_{1_1} + \ell_{0_2} + \ell_{0_3}$
α^4	101	$\ell_{1_1} + \ell_{0_2} + \ell_{1_3}$
α^5	110	$\ell_{1_1} + \ell_{1_2} + \ell_{0_3}$
α^6	111	$\ell_{1_1} + \ell_{1_2} + \ell_{1_3}$

Table 6.15: Binary to GF(8) mapping for LL metrics.

The RS decoders were presented with symbols over GF(q). For each symbol there were q LL metrics. Other than for the implementation changes described above this channel is identical to that used by the SOVA system (Section 6.5.2), and later, comparisons between the two systems will be made. Likewise, the simulated system assumed synchronisation.

6.6.2 Comparison of Cascade Decoding Algorithms

Decoding Performance

Four different variations on the cascade decoder were implemented. The first and most basic is HD decoding of rows, followed by HD decoding of columns (denoted by *cascade HD-HD*). As expected this decoder performed worst. A standard improvement is to add SD decoding to the first decoding stage (cascade SD-HD). By considering the channel state information an improvement of approximately 1.5 dB over cascade HD-HD was obtained (Figure 6.15).

By introducing advanced techniques such as SOVA (Section 5.2) the column decoding stage may also use SD information and reap the benefits by reducing the BER further still. Two variations of the cascade SD-SD decoder were implemented. In the first (cascade SD-SDa), the column decoding used only the soft output from SOVA (the extrinsic information) whilst the second used the sum of the SOVA metric and the channel state information (i.e., extrinsic + channel state information). As cascade SD-SDa does not make use of all the information available it performed worse than cascade SD-SDb, but it is shown in Figure 6.15 to indicate the relative gains due to the extrinsic information and the channel state information.

By taking advantage of the extrinsic information available from SOVA and the channel state information the cascade SD-SDb decoder is able to perform best of all the cascade decoders implemented. It has a coding gain of > 1 dB over cascade SD-HD at a BER of 10^{-6} (Figure 6.15), and with respect to cascade HD-HD the coding gain is approximately 2.2 dB.

Decoding Complexity

The cascade decoding complexity was measured by combining the decoding complexities of the BM, VA and SOVA decoders. The complexity values are quoted for a BER of 10^{-4}, because the complexity of the algebraic decoders (Section 6.3.1) is a function of the input E_b/N_0 ratio. BM decoding was chosen because it is the least complex HD decoder implemented in this work. Table 6.16 details the complexity for the cascade decoders described above. Only operations performed by the row or column decoders are included in Table 6.16, other operations are deemed overheads and are not included. The complexity of SD-SDa and SD-SDb are almost identical, SD-SDb is *slightly* more complex because it requires extra additions to sum the channel state and extrinsic information.

Decoder	Number of decodings				
version	BM	VA	SOVA	total	bit total
HD-HD	12	0	0	812	10.83
SD-HD	5	7	0	41589	554.53
SD-SDa	0	5	7	80565	1074.20
SD-SDb	0	5	7	80600	1074.67

Table 6.16: Comparison of complexity for cascade decoding algorithms.

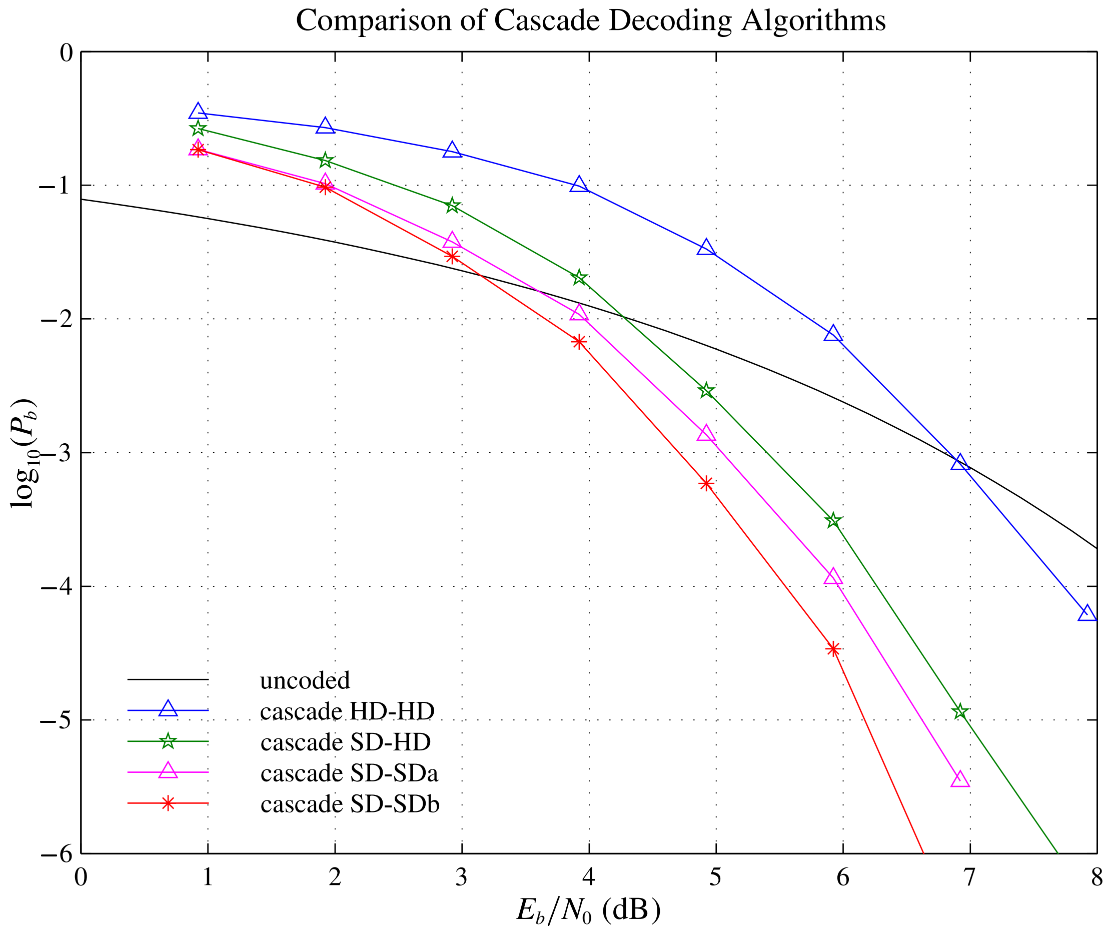


Figure 6.15: Comparison of cascade decoding algorithms.

6.6.3 Comparison of Alternating Row-Column Algorithms

Decoding Performance

Two variants of the alternating row-column decoder (Section 5.3.7) were implemented and evaluated. Firstly, with just an initial sort of the row and column codeword confidences (i.e., no re-sort), and also with the row/column codeword confidences recalculated and re-sorted after each row/column decoding. Both variants used SOVA, not the successive erasures decoding used in [Bate *et al.*, 1986]. In Figure 6.16 the error-correction performance is compared with uncoded transmission and the 'standard' cascade SD-HD decoder. It can be seen that the ARC decoders provide an extra 0.8 dB coding gain relative to the standard cascade SD-HD decoder. Interestingly, no significant difference in performance of the two variants is discernible. This shows that recalculation and re-sorting of the row and column codeword confidences is not necessary.

Decoding Complexity

The decoding complexity may be computed by considering the average number of row and column decodings required. The average number of row and column decodings is given by $\frac{n_1+k_1}{2}$ and $\frac{n_2+k_2}{2}$ respectively. To decode a product code with RS$(7,5,3)$ row/column codes the bit complexity is 1168.00, not significantly different from that of the cascade SD-SDb decoder ($\kappa_{\text{bit}} = 1074.67$).

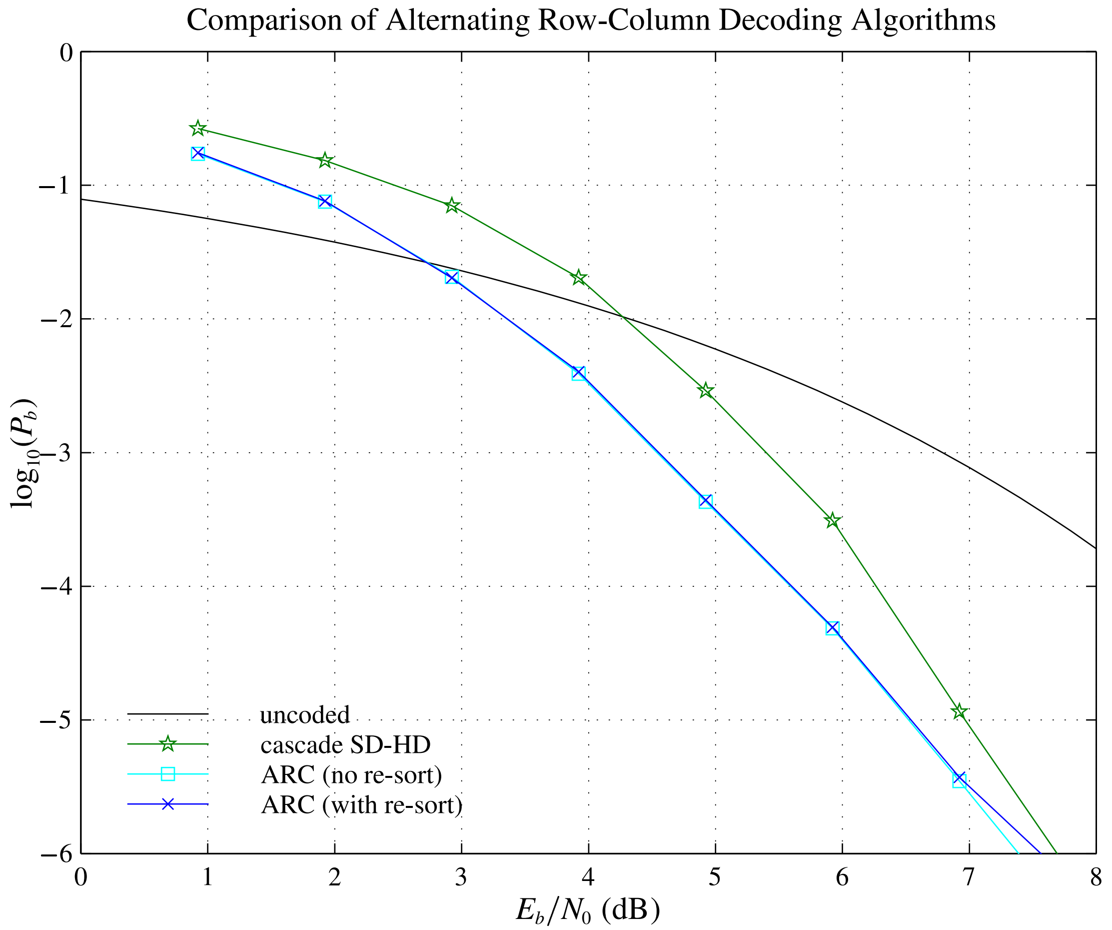


Figure 6.16: Comparison of alternating row-column decoding algorithms.

6.6.4 Results on Iterative Decoding of Product Codes

In Figure 6.17 the decoding performance of ARC decoding is compared for the cases of one and four iterations. Again, no benefit is found from re-sorting the row and column codeword confidences after each decoding. It can be seen that the effect of four iterations is to provide an extra 1.1 dB coding gain. The decoding complexity is simply i times greater, i.e., for the decoder in Figure 6.17 with 4 iterations the bit complexity is 4672.

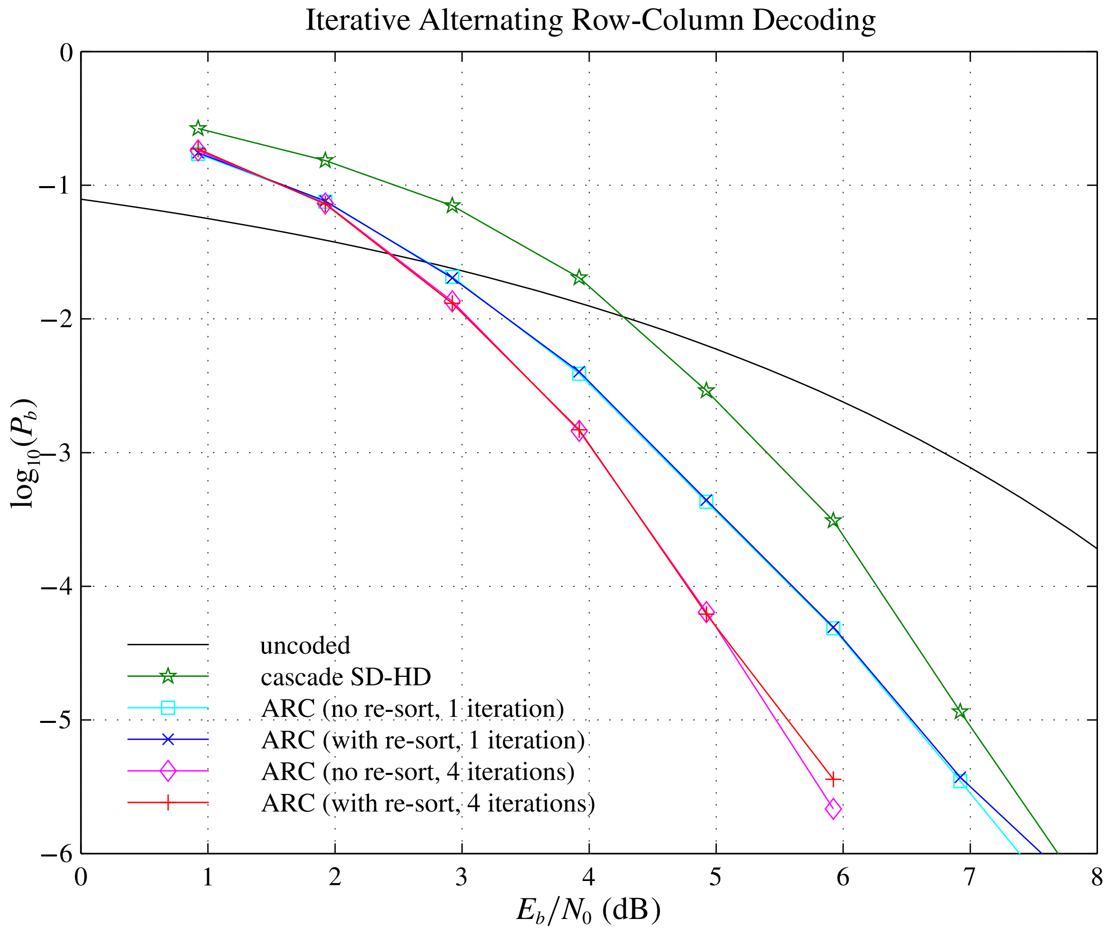


Figure 6.17: Iterative alternating row-column decoding.

Chapter 7

Conclusions and Further Work

Chapter 7

Conclusions and Further Work

7.1 Original Contributions

I claim the following areas of research as original contributions:

- Two-stage decoding of RS codes (Section 4.2.5).
- Soft Galois field arithmetic (Section 4.3).
- Alternating row-column decoding of product codes using SOVA (Section 5.3.7).
- Trellis decoding complexity measured by the number of additions and comparisons (Section 6.2).

  While McEliece has also introduced a similar method [McEliece, 1996] the results of my work were first published in 1995 (initially for Euclidean metrics). Subsequently the work was extended to LL metrics and presented in the form of trellis parameters. This work was independent of [McEliece, 1996]. Further-more, Section 6.2 details both optimisations not given in [McEliece, 1996] and also the decoding complexity for SOVA.

- The computer simulation of the new decoding algorithms to measure both decoder performance and complexity. In addition, the computer simulation of the Meteosat II – Earth retransmission channel using SOVA.

7.2 Decoding Complexity

In Section 6.1.2 a technique was introduced to measure and compare decoding complexity for algebraic decoders. It was successfully applied to the Berlekamp-Massey, Euclidean and high-speed step-by-step decoders. The difference in complexity for the Berlekamp-Massey and Euclidean decoders was small, when correcting errors the Berlekamp-Massey algorithm was typically 20–25% more efficient. Both [Wicker, 1994, p. 225] and [MacWilliams and Sloane, 1978, p. 369] cite the fact that the Berlekamp algorithm is the slightly more efficient of the two.

Transforming trellis complexities into the number of algebraic operations required for its computation (Section 6.2) has been a very successful approach for comparing trellis and algebraic decoding techniques. In particular, without such a method it would not have been possible to ascertain what complexity benefits actually existed in two-stage decoding. Nor would the comparisons between the Meteosat II system and the product code algorithms, as they are considerably different in their approach. McEliece has used a similar approach to calculating trellis complexity [McEliece, 1996]. However, his published results apply to the BCJR trellis only while the method described in Section 6.2 is applicable to any linear code trellis. With trivial modification (as suggested) the work in this Thesis can also be applied to rectangular non-linear codes. McEliece also applies equal weighting to the operations "addition" and "taking the minimum", whereas it was shown (Table 6.1) that such operations are not necessarily of equal complexity. Another significant difference is the inclusion of certain important optimisations, such as shared labels and decoding non-truncated trellises.

7.2.1 Shared Labels and Trellis Complexity

It would appear from the literature that no work has been carried out on the subject of reducing trellis (decoding) complexity by using the concept of shared branch labels (p. 153). McEliece states [McEliece, 1996, p. 1077]

> It can happen that different codewords will produce common edges, i.e., edges with the same values of init(e), fin(e) and $\lambda(e)$.[1] Such "shared" edges are only counted once in the trellis. It is this sharing of edges that makes the BCJR trellis an efficient graphical representation of the code.

This is however a feature of applying separability[2] [Sidorenko *et al.*, 1999] to a code (be it linear or non-linear) to produce the optimum trellis; it is *not* the same as the label sharing described on p. 153, where the start and/or end nodes differ. It was stated that codes such as RM and single error-correcting Hamming codes could share labels (which is apparent by inspection of their trellises) but that it is not possible for

[1] McEliece uses the notation init(e), fin(e) and $\lambda(e)$ to refer to the start state, end state and labelling, respectively, of a trellis branch (edge).

[2] Also termed *rectangularity*.

RS codes. No proof was offered for either statement and this is an area where further work is required.

7.3 High-speed Step-by-step Decoding

Example 3.5 highlights the tortuous route by which HSSBS decoding sometimes operates, particularly when compared with the efficiency in which both Euclidean and Berlekamp-Massey algorithms perform the same task (Examples 3.3 and 3.4 respectively). HSSBS decoding is very inefficient at correcting the t-th error because of its dependence of finding a codeword at a distance of $2t+1$ from the received codeword. It should be noted that if HSSBS had corrected all symbol locations only 40 attempts would have been required to correct all errors (instead of 52). For sake of completeness, further study into the trade-off between decoding all symbols and decoding only the information symbols, as a function of E_b/N_0, could be made. However, HSSBS is not (and probably never will be) an efficient algorithm for decoding multi-level codes over large alphabets, as the results in Section 6.3.1 clearly show.

7.4 Two-stage Decoding

The premise of two-stage decoding was that the decoding complexity could be reduced from $\kappa'\kappa''$ to $\kappa'+\kappa''$, with only minor loss of performance. For the case of RS codes it was shown that a trellis-based system for the selection of second-stage subtrellises was not possible. This unfortunately increased the complexity of the first decoding stage, but complexity savings are still possible (Section 6.4.2). For the case of HD subtrellis prediction it also reduced the performance. A trellis-type method would seem a more natural method to use for the selection of which subtrellis(es) to decode.

It is suggested that for two-stage decoding of RS codes future work might be directed towards the use of the max-sum algorithm [Forney, 1997]. The subtrellis prediction can therefore employ Tanner graphs [Tanner, 1981] as a means of preserving SD information. Investigation should be carried out to see if such a scheme is any more efficient that the ‘soft’ GF arithmetic implemented in Section 4.3.

An alternative direction is to consider the possibility of decoding a variable number of subtrellises, instead of the fixed but adjustable number currently decoded. Such a scheme would require a termination condition. Aguado and Farrell employed the Fano metric [Fano, 1963] for their reduced search trellis decoding algorithm while Shin and Sweeney used their own reference path metric to discard candidate paths. A third option for deciding when to stop decoding subtrellises might be the SOVA reliability metric. Whatever metric were to be employed the advantages of such a scheme could be improved error control performance and reduced complexity.

7.5 Comparison of Decoders for Concatenated Codes

For the case of the modified Meteosat II system it can be seen that SOVA provides a gain of 1.1–1.8 dB, dependent upon the depth of the trellis decoded for the convolutional code (Figures 6.12–6.14). The commercial benefits of a 1.8 dB increase in coding gain are considerable. For example, the geostationary satellite can transmit a signal 1.8 dB weaker, which requires less power and thus smaller solar cells and batteries. The weight saved translates to cheaper launch costs for the satellite. Alternatively, the receiving dish be may reduced in area by 33% (so that a 0.8 m dish can be used where a 1 m dish was previously required). Another option is to increase the defined service area and accrue greater revenue from an enlarged number of users. For the system simulated, the increase in decoding complexity to achieve such a large gain was very modest indeed, only 22%. An alternative comparison is that SD-SD decoding over a convolutional trellis of length 28 has approximately the same complexity as SD-HD decoding over a trellis of length 35, but performed better than the most complex SD-HD decoder evaluated.

The simulated transmission channel for both the Meteosat II satellite system and the RS product code was coherently-demodulated BPSK. Therefore their performance may be compared directly. The best of the Meteosat II, and RS product code decoders have been included together in Figure 7.1.

Firstly, it can be seen that the performance increase in applying SOVA to the (modified) Meteosat II system is 1.8 dB at a BER of 10^{-6}. For cascade decoding of a product code the increase is only just over 1 dB. This may be explained by two factors. The inner code of the Meteosat II system is stronger (but more complex to decode). Also, the outer RS$(7, 5, 3)$ code of the modified Meteosat II system was allocated a reliability metric for each input bit. For the cascade decoder it was necessary to make the assumptions that the SOVA metric applied equally to all symbols, and that the discarded values were equally unlikely (p. 137). This lead to only 7 SOVA reliability metrics being available, which were ‘recycled’ for each of the 5 column decodings.

It is interesting to note that the Meteosat II decoder performs better than the RS product code, for both SD-HD and SD-SD cases. Massey's assertion (p. 117) that convolutional codes should be used as the first stage of decoding has been shown to be correct for the cases simulated. Such increase in performance comes at a price of higher complexity, which must not be forgotten. The Meteosat II SD-SD decoder shown in Figure 7.1 is ≈ 20 times more complex than one iteration of the ARC SD-SD decoder.

7.6 Iterative Decoding of Product Codes

The brief foray into iterative decoding (Sections 5.3.8 and 6.6.4) has provided some excellent results. Without any additional work the alternating row-column decoder was instructed to execute four decodings instead of one. At a BER of 10^{-5} the coding gain increased by > 1.1 dB. While the complexity also increased by a factor of four the bit complexity was still less than that of the modified Meteosat II system by a factor of ≈ 5. Further work should be undertaken to ascertain the optimum number of iterations.

The increase in coding gain was less than that found in [Pyndiah, 1998], where a $\mathrm{BCH}(64,51,6) \times \mathrm{BCH}(64,51,6)$ code provided an increase of > 1.5 dB for just 4 iterations. There are several reasons for this, not just the change of code or increase in code alphabet size. Pyndiah used a *weighting factor*, α, to scale the amount of extrinsic information included by each iteration, m, where

$$\alpha(m) = \begin{bmatrix} 0.0 & 0.2 & 0.3 & 0.5 & 0.7 & 0.9 & 1.0 & 1.0 \end{bmatrix} \tag{7.1}$$

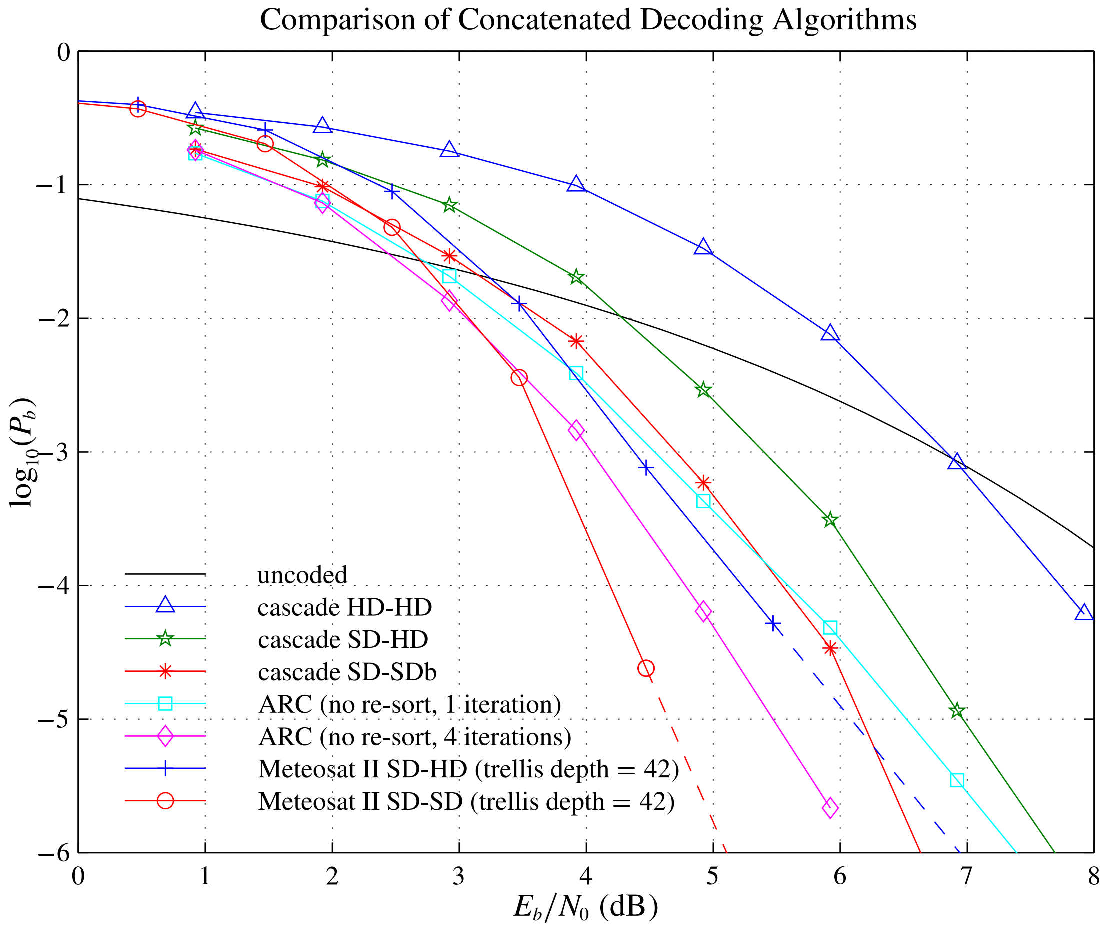


Figure 7.1: Comparison of concatenated decoding algorithms.

With such a scheme it ensures the extrinsic information is added slowly, to help the decoding process to converge upon the correct solution. Adding a weighting factor to the iterative decoding described in Section 5.3.8 would be a trivial extension. It should also be remembered that the soft-output Viterbi algorithm employed is not entirely optimum at generating its reliability metric. (However, the selection of the output codeword is optimum.) There is neither any indication as to the next-best symbol, nor the relative reliabilities of the decoded symbols. Nonetheless SOVA is shown to be a very useful and efficient decoding algorithm for this purpose. Interesting further work on this topic would be to simulate the $\mathrm{BCH}(64, 51, 6) \times \mathrm{BCH}(64, 51, 6)$ code used by Pyndiah with the iterative decoder described in Section 5.3.8. This is at the limit of what is currently possible to decode with a trellis.

References

L. E. Aguado and P. G. Farrell. On hybrid stack decoding algorithms for block codes. *IEEE Transactions on Information Theory*, 44(1):398–409, January 1998.

L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimising symbol error rate. *IEEE Transactions on Information Theory*, IT-20: 284–287, March 1974.

S. D. Bate, B. K. Honary, and P. G. Farrell. Soft and hard decision decoding of product codes for communication systems. In *Systems Science*, volume 12, pages 79–85. 1986.

Y. Berger and Y. Be'ery. Bounds on the trellis size of linear block codes. *IEEE Transactions on Information Theory*, 39(1):203–209, January 1993.

E. R. Berlekamp. Nonbinary BCH decoding. In *1967 International Symposium on Information Theory*, San Remo, Italy, 1967.

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

R. C. Bose and D. K. Ray-Chaudhuri. Further results on error correcting binary group codes. *Information and Control*, 3(3):279–290, September 1960a.

R. C. Bose and D. K. Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and Control*, 3(1):68–79, March 1960b.

H. O. Burton and E. J. Weldon, Jr. Cyclic product codes. *IEEE Transactions on Information Theory*, IT11:433–439, 1965.

D. A. Chase. A class of algorithms for decoding block codes with channel measurement information. *IEEE Transactions on Information Theory*, IT-18:170–182, 1972.

R. T. Chien. Cyclic decoding procedure for the Bose-Chaudhuri-Hocquenghem codes. *IEEE Transactions on Information Theory*, IT-10:357–363, October 1964.

G. C. Clark, Jr. and J. B. Cain. *Error-Correction Coding for Digital Communications*. Plenum Press, New York and London, 1981. ISBN 0-306-406-15-2.

Daimler-Benz Aerospace. *Meteosat Second Generation: HRUS / LRUS Detailed Design Document*, 1995. Doc. MSGUS-DOR-DDD-0001, Issue 2, Rev. 0, Date 27/10/95.

P. Elias. Error-free coding. *IEEE Transactions on Information Theory*, IT-4:29–37, 1954.

P. Elias. Coding for noisy channels. *IRE Convention Record*, 3(4):37–47, 1955.

R. M. Fano. A heuristic discussion of probabilistic decoding. *IEEE Transactions on Information Theory*, IT-9:64–74, April 1963.

P. G. Farrell, B. K. Honary, and S. D. Bate. Adaptive product codes with soft/hard decision decoding. In *Proceedings of the IMA Conference on Cryptography and Coding*, pages 95–111, Cirencester, UK, December 1986.

G. D. Forney, Jr. On decoding BCH codes. *IEEE Transactions on Information Theory*, IT-11:549–557, October 1965.

G. D. Forney, Jr. *Concatenated Codes*. MIT Press, Cambridge, MA, 1966.

G. D. Forney, Jr. Convolutional codes I: Algebraic structure. *IEEE Transactions on Information Theory*, IT-16:720–738, November 1970.

G. D. Forney, Jr. Coset codes—part I: Introduction and geometrical classification. *IEEE Transactions on Information Theory*, 34(5):1123–1151, September 1988a.

G. D. Forney, Jr. Coset codes—part II: Binary lattices and related codes. *IEEE Transactions on Information Theory*, 34(5):1152–1187, September 1988b.

G. D. Forney, Jr. On iterative decoding and the two-way algorithm. In *International Symposium on Turbo Codes*, pages 12–25, Brest, France, 1997.

G. D. Forney, Jr. and M. D. Trott. The dynamics of group codes: State spaces, trellis diagrams, and canonical encoders. *IEEE Transactions on Information Theory*, 39 (9):1491–1513, September 1993.

R. G. Gallager. *Information Theory and Reliable Communication*. John Wiley & Sons, New York, London, Sydney, 1968.

D. Gorenstein and N. Zierler. A class of cyclic linear error-correcting codes in p^m symbols. *Journal of the Society of Industrial and Applied Mathematics*, 9:207–214, June 1961.

J. Hagenauer. Source-controlled channel decoding. *IEEE Transactions on Communications*, 43(9):2449–2457, September 1995.

J. Hagenauer and P. Hoeher. A Viterbi algorithm with soft-decision outputs and its applications. In *Proceedings of the IEEE Globecom Conference*, pages 1680–1686, Dallas, Texas, November 1989.

J. Hagenauer, E. Offer, and L. Papke. Matching Viterbi decoders and Reed-Solomon decoders in a concatenated system. In S. B. Wicker and V. K. Bhargava, editors, *Reed-Solomon Codes and Their Applications*, pages 242–271. IEEE Press, New York, 1994. ISBN 0-7803-1025-X.

J. Hagenauer, E. Offer, and L. Papke. Iterative decoding of binary block and convolutional codes. *IEEE Transactions on Information Theory*, 42(2):429–445, March 1996.

J. A. Heller and I. W. Jacobs. Viterbi decoding for satellite and space communication. *IEEE Transactions on Communications Technology*, COM-19(5):835–848, October 1971.

A. Hocquenghem. Codes correcteurs d'erreurs. *Chiffres*, 2:147–156, 1959.

B. Honary and G. Markarian. Low-complexity trellis decoding of Hamming codes. *Electronics Letters*, 29(12):1114–1116, June 1993a.

B. Honary and G. Markarian. New simple encoder and trellis decoder for Golay codes. *Electronics Letters*, 29(25):2170–2171, 1993b.

B. Honary and G. Markarian. *Trellis Decoding of Block Codes*. Kluwer Academic Publishers, Boston, Dordrecht, London, 1997. ISBN 0-7923-9860-2.

B. Honary, G. Markarian, and M. Darnell. Low-complexity trellis decoding of linear block codes. *IEE Proceedings on Communications*, 142(4):201–209, August 1995a.

B. Honary, G. Markarian, and M. Darnell. Trellis decoding for linear block codes. In P. G. Farrell, editor, *Codes and Ciphers*. IMA Press, 1995b. ISBN 0905091035.

B. Honary, G. Markarian, and S. R. Marple. Trellis decoding of the Reed-Solomon codes: Practical approach. In *3rd International Symposium on Communication Theory and Applications*, pages 12–16, Ambleside, UK, July 1995c.

B. Honary, G. Markarian, and S. R. Marple. Two-stage trellis decoding of RS codes based on the Shannon product. In *Proceedings of the 1996 International Symposium on Information Theory and Its Applications*, pages 282–285, Victoria, BC, Canada, September 1996.

B. Honary, G. Markarian, and S. R. Marple. Trellis decoding of the Reed-Solomon codes: A practical approach. In B. Honary, M. Darnell, and P. G. Farrell, editors, *Communications Coding and Signal Processing, Communications Systems Techniques and Applications Series*, pages 133–147. John Wiley & Sons, New York, London, Sydney, 1997. ISBN 0 86380 221 4.

B. Honary, G. S. Markarian, and P. G. Farrell. Generalised array codes and their trellis structure. *Electronics Letters*, 29(6):541–542, March 1993.

F. Jelinek. A fast sequential decoding algorithm using a stack. *IBM Journal of Research and Development*, 13(6):675–685, November 1969.

T. Kasami, T. Takata, T. Fujiwara, and S. Lin. On complexity of trellis structure of linear block codes. *IEEE Transactions on Information Theory*, 39(3):1057–1064, May 1993a.

T. Kasami, T. Takata, T. Fujiwara, and S. Lin. On the optimum bit orders with respect to the state complexity of trellis diagrams for binary linear codes. *IEEE Transactions on Information Theory*, 39(1):242–245, January 1993b.

D. E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, USA, second edition, 1973. ISBN 0-201-03803-X.

F. R. Kschischang and V. Sorokine. On the trellis structure of block codes. *IEEE Transactions on Information Theory*, 41(6):1924–1937, November 1995.

S. Lin and J. Costello, Jr. *Error Control Coding: Fundamentals and Applications*. Prentice-Hall, Englewood Cliffs, NJ, 1983. ISBN 0-13-283796-X.

F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*. North Holland-Elsevier Science Publishers, Amsterdam, New York, Oxford, 1978. ISBN 0-444-85193-3.

S. R. Marple. Feasibility study of applying soft-output Viterbi algorithm decoding to

METSAT II. Technical report, Communications Research Centre, Lancaster University, UK, 1998.

J. L. Massey. Step-by-step decoding of the Bose-Chaudhuri-Hocquenghem codes. *IEEE Transactions on Information Theory*, IT-11(4):580–585, October 1965.

J. L. Massey. Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, IT-15(1):122–127, January 1969.

J. L. Massey. The how and why of channel coding. In *Proceedings of the 1984 Zurich Seminar on Digital Communications*, number 84 CH 1998-4, pages 67–73, Zurich, Switzerland, 1984. IEEE.

R. J. McEliece. *Finite Fields for Computer Scientists and Engineers*. Kluwer Academic Publishers, Boston, Dordrecht, London, 1987. ISBN 0898381916.

R. J. McEliece. The Viterbi decoding complexity of linear block codes. In *Proceedings of the IEEE International Symposium on Information Theory*, page 341, Trondheim, Norway, 1994.

R. J. McEliece. On the BCJR trellis for linear block codes. *IEEE Transactions on Information Theory*, 42(4):1072–1092, July 1996.

A. M. Michelson and A. H. Levesque. *Error-Control Techniques for Digital Communications*. John Wiley & Sons, New York, London, Sydney, 1985. ISBN 0-471-88074-4.

G. E. Moore. Cramming more components onto integrated circuits. *Electronics*

Magazine, 38(8):114–117, April 1965. See also `http://www.intel.com/intel/museum/25anniv/hof/moore.htm`.

D. J. Muder. Minimal trellises for block codes. *IEEE Transactions on Information Theory*, 34(5):1049–1053, September 1988.

J. K. Omura. On the Viterbi decoding algorithm. *IEEE Transactions on Information Theory*, IT-15:177–179, January 1969.

W. W. Peterson. Encoding and error-correction procedures for the Bose-Chaudhuri codes. *IRE Transactions on Information Theory*, IT-6:459–470, September 1960.

W. W. Peterson and E. J. Weldon, Jr. *Error-Correcting Codes*. MIT Press, Cambridge, MA, second edition, 1972. ISBN 0-262-16-039-0.

W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: The Art of Numerical Computing*. Cambridge University Press, Cambridge, UK, second edition, 1992. ISBN 0-521-43108-5. `http://cfatab.harvard.edu/nr/bookc.html`.

R. M. Pyndiah. Near-optimum decoding of product codes: Block turbo codes. *IEEE Transactions on Communications*, 46(8):1003–1010, August 1998.

I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society of Industrial and Applied Mathematics*, 8:300–304, June 1960.

R. Sedgewick. *Algorithms*. Addison-Wesley, Reading, MA, USA, second edition, 1988. ISBN 0-201-06673-4.

C. E. Shannon. The zero error capacity of a noisy channel. *IRE Transactions on Information Theory*, 2:s8–s19, 1956.

S. K. Shin and P. Sweeney. Soft decision decoding of Reed-Solomon codes using trellis methods. *IEE Proceedings on Communications*, 141(5):303–308, October 1994.

V. Sidorenko, G. Markarian, and B. Honary. Code trellises and the Shannon Product. In *Seventh Joint Swedish-Russian International Workshop on Information Theory*, pages 220–224, June 1995.

V. Sidorenko, G. Markarian, and B. Honary. Minimal trellis design for linear codes based on the Shannon Product. *IEEE Transactions on Information Theory*, 42(6): 2048–2053, November 1996.

V. Sidorenko, I. Martin, and B. Honary. On the rectangularity of nonlinear block codes. *IEEE Transactions on Information Theory*, 45(2):720–725, March 1999.

R. C. Singleton. Maximum distance q-nary codes. *IEEE Transactions on Information Theory*, IT-10:116–118, 1964.

B. Sklar. *Digital Communications*. Prentice-Hall, Englewood Cliffs, NJ, 1988. ISBN 0-13-212713-X.

D. Slepian. Some further theory of group codes. *Bell Systems Technical Journal*, 39: 1219–1252, September 1960.

B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, Reading, MA, USA, third edition, 1997. ISBN 0-201-88954-4.

Y. Sugiyama, Y. Kasahara, S. Hirasawa, and T. Namekawa. A method for solving key equation for decoding Goppa codes. *Information and Control*, 27(1):87–99, January 1975.

R. M. Tanner. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, IT-27:533–547, September 1981.

A. Vardy and Y. Be'ery. Bit level soft decision decoding of Reed-Solomon codes. *IEEE Transactions on Information Theory*, 39(3):440–444, 1991.

A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, IT-13:260–269, April 1967.

A. J. Viterbi. Convolutional codes and their performance in communication systems. *IEEE Transactions on Communications Technology*, COM-19(5):751–772, October 1971.

S. W. Wei and C. H. Wei. High-speed decoder of Reed-Solomon codes. *IEEE Transactions on Information Theory*, 41:1588–1593, November 1993.

S. Wicker. *Error Control Systems for Digital Communication and Storage*. Prentice-Hall, Englewood Cliffs, NJ, 1994. ISBN 0-13-200809-2.

S. G. Wilson. *Digital Modulation and Coding*. Prentice-Hall, Englewood Cliffs, NJ, 1996. ISBN 0-13-210071-1.

J. K. Wolf. On codes derivable from the tensor product of check matrices. *IEEE Transactions on Information Theory*, 11:281–284, April 1965.

J. K. Wolf. Efficient maximum likelihood decoding of linear block codes using a trellis. *IEEE Transactions on Information Theory*, IT-24(1):76–80, January 1978.

J. M. Wozencraft and I. M. Jacobs. *Principles of Communication Engineering*. John Wiley & Sons, New York, London, Sydney, 1965.

J. Wu, S. Lin, T. Kasami, T. Fujiwara, and T. Takata. An upper bound on the effective error coefficient of two-stage decoding, and good two-level decompositions of some Reed-Muller codes. *IEEE Transactions on Communications*, 42(2/3/4):813–1053, February/March/April 1994.

K. Zigangirov. Some sequential decoding procedures. *Problemi Peredachi Informatii*, 2(4):13–25, 1966. (Russian).

V. Zyablov and V. Sidorenko. Bounds of complexity of trellis decoding of linear block codes. In *Proceedings of the Swedish-Russian Workshop on Information Theory*, August 1993.

Citation Index

Index

Colophon

This Thesis was typeset in Computer Modern and Adobe® Times-Roman by $\LaTeX\,2_\varepsilon$. The generosity of Leslie Lamport in making $\LaTeX\,2_\varepsilon$ freely available, and also Donald Knuth for $\TeX$, is acknowledged. The success of $\TeX$ and $\LaTeX\,2_\varepsilon$ is due to many volunteers in the $\TeX$ community who have contributed numerous support packages for free.