Trellis Decoding of Reed Solomon and Related Linear Block Codes

By

Werner Heinrich Büttner

A dissertation submitted in partial fulfillment of the requirements for the degree

Master of Engineering (Electronic)

in the

Department of Electrical and Electronic Engineering
in the
Faculty of Engineering

at the

UNIVERSITY OF PRETORIA

Advisor: Professor L.P. Linde

October 1999

Abstract

TRELLIS DECODING OF REED-SOLOMON AND RELATED LINEAR BLOCK CODES

by

Werner Heinrich Büttner

Advisor: Prof. L. P. Linde

Department of Electrical and Electronic Engineering

Master of Engineering (Electronics)

Reed-Solomon codes, a subset of multilevel non-binary cyclic codes with powerful burst error correcting capabilities, are known to be computationally efficient when algebraic decoding techniques are applied. They may however give weaker performance compared to convolutional coding techniques, at least at moderate bit error rates (around 10^{-5} to 10^{-6}) on the AWGN channel. This disadvantage mainly results from the lack of a general applicable method for soft-decision decoding. The aim is to construct a trellis from the generator matrix of a Reed-Solomon code and to show that apart from the historical Berlekamp Massey frequency domain techniques, other techniques usually reserved for convolutional code decoding, such as Maximum Likelihood (ML) and the Maximum A-Posteriori (MAP) techniques, can be successfully applied in the decoding process. Consequently, the main objective of this dissertation is to analyse, design and implement a soft-input, soft-output Reed-Solomon ML or MAP trellis decoding algorithm with performance and complexity comparable to conventional algebraic block decoding methods.

The main reason why trellis decoding is not often used for cyclic block codes, is the complexity of the decoder, especially in the case of long codes with high redundancy. In fact, it will be almost impossible to implement the Viterbi decoder for Reed-Solomon codes with moderate redundancy, considering the fact that the Viterbi decoder becomes computationally unfeasible and practically intractable for convolutional codes of constraint length greater than 10 to 12. Therefore, in order to reduce trellis complexity, a search for minimal block trellis decoding techniques is launched through:

- manipulation of the generator matrix with a view of obtaining minimal trellis structures, i.e., minimising the number of states in a specific block code trellis;

- devising methods to simplify trellis construction of large block codes (i.e., codes with large block size n and redundancy n-k);

- considering only a certain number of trellis paths which are most likely to be transmitted, in stead of all possible paths. This may for example be achieved through a expurgation process on the original trellis (i.e., eliminating paths not terminating in the zero state) or by applying maximum likelihood (ML) or maximum a-posteriori (MAP) decoding methods such as the Viterbi algorithm which results in significant computational savings through its 'survival path' mechanism. The expurgation process is block code specific, i.e., the existence of unterminated paths may differ for each code and is a function of each code's inherent algebraic structure.

The objective of this reduced search method is therefore to optimise the performance of the code while minimising trellis decoding complexity and the corresponding decoding delay.

In the process of constructing minimal trellis structures for cyclic block codes, a novel topological branch interconnecting (trellis branch indexing or labelling) scheme for block codes, and specifically Reed-Solomon codes, is proposed and developed. The technique identifies unique interconnecting patterns in the branch structures of sub-trellises, by which the remaining parts of the trellis may be uniquely defined without having to resort to complicated trellis branch calculations. It is shown that the complexity of the trellis construction process may be reduced by orders in magnitude (for relatively short block lengths), by exploiting the well defined cyclic trellis patterns inherent to the trellis structures of individual block codes.

After having established methods for efficient block trellis construction and the corresponding minimal trellis coder and decoder design, it is next shown how ML convolutional decoding techniques, such as the Viterbi decoding algorithm, can be successfully employed in the decoding process of block codes which could traditionally only be decoded by means of algebraic techniques. The study then investigates the error performance achievable using the trellis as a means of decoding. It is shown that the performance of ML (Viterbi) trellis block decoding with soft decisions matches the

Abstract

performance rendered by soft decision algebraic block decoding techniques in all respects.

Key Words: Trellis, Complexity Reduction, Bit Error Rate, Reed-Solomon, Topology.

Centre for Radio and Digital Communication (CRDC)
Department of Electrical and Electronic Engineering
University of Pretoria

Opsomming

TRELLIS DEKODEERING VAN REED-SOLOMON EN VERWANTE LINEÊRE BLOK KODES

deur

Werner Heinrich Büttner

Studieleier: Prof. L. P. Linde

Departement van Elektiese en Elektoniese Ingenieurswese

Meesters in Ingenieurswese (Elektronies)

Reed-Solomon kodes, 'n onderafdeling van multivlak nie-binêre sikliese kodes met kragtige foutkorreksie eienskappe, word geken as rekenkundig effektief wanneer algebraiese dekoderingstegnieke gebruik word. Hulle kan egter swakker vaar in vergelyking met konvolusie koderingstegnieke, minstens by gemiddelde bisfoutwaarskynlikhede (omtrent 10^{-5} tot 10^{-6}) in 'n AWGN kanaal. Hierdie nadeel spruit uit die afwesigheid van 'n algemene methode vir sagte-beslissing-dekodering. Die doel is die konstruksie van 'n trellis vanaf die generatormatriks van 'n Reed-Solomon kode en om aan te toon dat benewens die historiese Berlekamp Massey frekwensie tegniek, dit moontlik is om ander tegnieke wat normaalweg uitsluitlik vir die dekodering van konvolusie kodes dien, soos die "Maximum Likelihood" (ML) en die Maksimum A-Posteriori (MAP) tegnieke, suksesvol te benut in die dekoderingsproses. Gevolglik is die hoofdoelstelling van hierdie verhandeling die analise, ontwerp en implementering van 'n sagte-inset, sagte-uitset Reed-Solomon ML of MAP trellis dekoderingsalgoritme met foutkorreksie-eienskappe en kompleksiteit vergelykbaar met konvensionele algebraiese blokdekoderingstegnieke.

Die hoofrede hoekom trellis dekodering nie vrylik gebruik word vir sikliese blokkodes nie, is die kompleksiteit van die dekodeerder, veral in die geval van lang kodes met 'n hoë oortolligheid. Dit word amper 'n rekenkundige onmoontlikheid om 'n Vieterbi dekodeerder vir Reed-Solomon kodes met 'n gemiddelde oortolligheid te implementeer, siende dat dit rekenkundig en prakties onmoontlik is vir konvolusiekodes met 'n beperkings lengte groter as 10 tot 12. Ten einde trellis kompleksiteit te beperk, word

Opsomming

'n soektog na minimale blok trellise geloods deur:

- manipulasie van die generatormatriks met die doel om 'n minimale trellis struktuur te verkry, met ander woorde om die aantal toestande in 'n spesefieke blokkode trellis te minimiseer;

- ontwikkeling van metodes om trellis konstruksie vir groot blok kodes (kodes met groot bloklengte n en oortolligheid n-k) te vereenvoudig;

- om slegs 'n sekere aantal trellis paaie in berekening te bring as een van die mees waarskynlik gestuurde kode, in plaas daarvan om almal in berekening te bring. Hierdie kan byvoorbeeld bewerkstellig word deur 'n proses, waar sekere paaie wat nie in die nul toestand termineer nie, weg te laat. Dit kan vêrder ook gedoen word deur die ML of MAP dekoderingstegnieke soos die Viterbi algoritme te gebruik, wat 'n groot besparing in bewerkings teweegbring deur gebruikmaking van die oorblywende pad meganisme. The proses van uitlating van paaie verskil vir elke blokkode en is 'n funksie van elke kode se inherente algebraiese struktuur.

Die doel van hierde gereduseerde soektog metode is dus om die werkverrigting van die kode te optimiseer terwyl die dekoderingskompleksiteit en die saamhangende dekoderingsvertraging geminimiseer word.

In die konstruksie proses van minimale trellis strukture vir sikliese blokkodes, word 'n nuwe topologiese tak interkonneksie (trellis tak indeksering) skema vir blok kodes, en meer spesifiek Reed-Solomon kodes, voorgestel en ontwikkel. Hierdie tegniek identifiseer unieke interkonneksiepatrone in die takstruktuur van sub-trellise, waarmee die res van die trellis uniek gedefinieer kan word sonder om ingewikkelde trellis takbewerkings te verrig. Daar word aangetoon dat die kompleksiteit van die trellis konstruksieproses ordes vereenvoudig kan word (vir relatiewe klein bloklengtes), deur die goed gedefinieerde sikliese trellispatrone inherent in die trellisstruktuur van individuele blokkodes te gebruik.

Nadat metodes vir effektiewe blok trelliskonstruksie en die minimale trellis kodeerder en dekodeerder bepaal is, word daar aangetoon hoe ML konvolusie dekoderingstegnieke soos die Viterbi algoritme suksesvol benut kan word in die dekoderingsproses van blok kodes, wat normaalweg slegs deur middel van algebraiese tegnieke dekodeer kon word.

Opsomming

Die studie ondersoek dan die foutkorreksieverrigting wat verkry word indien die trellis tegniek as dekoderingsproses benut word.

Daar word aangetoon dat die verrigting van ML (Viterbi) trellis blokdekodering met sagte beslissings in alle aspekte dieselfde is as die verrigting verkry deur sagte beslissing algebraïese tegnieke.

Sleutelwoorde: **T**rellis, **K**ompleksiteit **V**ermindering, **B**isfoutwaarskynlikheid, **R**eed-**S**olomon, **T**opologie.

Dedication

This dissertation is dedicated to L. du Preez.

Your support and effort will forever be remembered.

Acknowledgments

The following persons have given invaluable input, without which this dissertation would not have reached the state which it is in now.

Firstly I would like to thank Professor L.P. Linde, who as my mentor and project leader has always been there when needed. Without his expertise and time, this project would not have become what it is today.

Secondly, I would like to thank L. Staphorst, who has sacrificed enormous amounts of time to assist with various parts of the project. His help is greatly appreciated.

Sincere thanks also goes out to D. J. van Wyk, who had the initial drive to start this project.

Several persons have sacrificed their time to proofread this dissertation. Again, thanks goes out to Professor L.P. Linde, who has on more than one occasion set aside time to read the dissertation. The other people who have contributed in this aspect are amongst other B.H. Waldeck and Y. Iglauer.

Thanks also goes out to S. Swanepoel, who was always available to listen to problems and provide solutions.

I would also like to take this opportunity to thank various other people, the list being to long to reproduce here, who have given their utmost to help make this project a success.

And, last but not least, thanks goes to my parents and my girlfriend L. du Preez who have helped and assisted me throughout the whole project.

List of Acronyms and Abbreviations

List of Acronyms

AWGN	Additive White Gaussian Noise
BCH	Bose-Chaudhuri-Hocquenghem
BER	Bit Error Rate
BPSK	Binary Phase Shift Keying
CTD	Code Trellis Diagram
GF	Galois Field
HD	Hard-Decision
MAP	Maximum A-Posteriori Probability
ML	Maximum-Likelihood
MLD	Maximum-Likelihood Decoder
RS	Reed Solomon
SD	Soft-Decision
SDML	Soft-Decision-Maximum-Likelihood
SP	Trellis Sub-Part

List of Abbreviations

dB	Decibel
E_b	Energy per Bit
E_{bit}/N_0	Energy to Noise Ratio
G	Generator Matrix
H	Parity Check Matrix
h_x	Column x of Parity Matrix H
I	Unit Matrix
M	Encoder Memory Arrangement
N	Number of States
N_0	Single Sided Noise Power Spectral Density
P(x)	Probability
P_{Bit}	Bit Error Probability
P_{Block}	Block Error Probability

List of Figures

List of Tables

Table of Contents

Chapter 1

Introduction

This chapter provides the framework on which the rest of the dissertation is based.

To start off, a historical overview is presented stretching back to the dawn of modern telecommunication over half a century ago. Following this historical account, a clear breakdown of the topics examined and contributions made is provided. This will provide the reader with an overview of the work following the introduction.

In the next section, the goals and objectives of the dissertation are outlined. They are again evaluated in the summary at the end of the dissertation, and should be read together with this introductory chapter.

1.1 Historical Overview

Since the epochal work by Claude E. Shannon in 1948 [23], various different channel encoder and channel decoding techniques have been developed for the transmission of data over noisy communication channels. The aim of all these techniques are the same: Transmit as much data as possible given a certain time frame and channel characteristic with the least amount of errors. This simple statement caused an uproar in the communications and mathematical fields when Shannon derived a bound specifying the maximum capacity that can be achieved at various energy to noise ratios for a given channel. Since 1948 this bound is commonly known as the *Shannon Bound* [23]. It has become the dream of many a designer and researcher to cross this theoretical bound discovered by Shannon. To date, the bound has not been reached nor exceeded, although the latest techniques come to within dB's of this elusive goal.

The various coding techniques are abound as sand in the Sahara desert. All channel coding techniques can be divided into two main groups, namely the class of convolutional codes and the class of block codes. Many hybrid systems have also evolved over the years, but in principle, the two above mentioned groups are the two main fields on which

research is currently done.

The enormous difference between the properties of these two groups of codes has always lead to either block codes or convolutional codes being researched, but efforts involving combinations of these codes were few and far between. This phenomenon mainly sprouted from the completely different decoding procedures used to decode the two groups of codes. For block codes, pure algebraic techniques involving correlative and comparative processes were developed, whereas elegant soft decision techniques were discovered for convolutional codes. As the convolutional soft decision decoding techniques improved with time, the use of block codes in systems became less attractive. This spiraling effect led to a situation were almost all of the effort was being focused on convolutional codes and their decoding techniques.

The decoding techniques for convolutional codes rely on the fact that all convolutional codes can be described by a state-time variant diagram termed a *trellis* diagram. This sparked the idea that if a trellis diagram could be obtained for a block code, the same elegant convolutional decoding techniques could be applied in the decoding process of block codes.

In 1974 Bahl, Cocke, Jelinek and Raviv [2] proposed a possible method for the construction of a block code trellis diagram. Their valuable work was used as a base by J.K. Wolf [16] in 1978 in order to develop a procedure for the construction of block code trellis diagrams. Forney [5] also elaborated on the topic, and introduced the world to *coset trellises*, which are described in the chapters to follow.

Parallel to this discovery, a new type of block code was introduced on the 21st of January 1959. This non-binary block code became known as the Reed-Solomon[20] code, named after its discoverers, Irvine Reed and Gus Solomon. The Reed-Solomon code has become the most widely used block code and has the best error correcting capability of all block codes. Many different techniques, apart from the algebraic ones, have been developed to decode Reed-Solomon codes (the Berlekamp Massey [21] technique to name just one). Nevertheless, the reasons why this group of codes has never enjoyed the advantages of convolutional decoding techniques can firstly be attributed to the enormous size of the codes and secondly to the fact that it falls into the category of a block code. This dissertation will investigate methods for soft decision maximum likelihood decoding of Reed-Solomon codes. A quote from a paper by Cooper [21] is found to be very relevant

here: "*The Holy Grail of Reed-Solomon decoding research is the soft decision maximum likelihood decoder. There is however still a great deal of work to be done before the Grail is found and the knights can go home.*"

Now, 50 years later, more and more effort is channeled into the development of novel channel encoding techniques employing both block codes and convolutional codes. In this manner, it is aimed to combine the best of two worlds in order to get closer to the ultimate goal in communication theory - the Shannon Bound.

1.2 Goals and Objectives

The dissertation is mainly concerned with various decoding techniques of block codes. The main objective is to decode block codes with the same decoders that are employed in the decoding process of convolutional codes, namely the Viterbi and MAP algorithms.

From this main goal several secondary goals emanate which are listed below. Each secondary goal is considered to be as important as the original goal.

- This dissertation aims at providing viable trellis construction techniques for block codes, their success being evaluated and judged in terms of complexity.
- Soft decision maximum likelihood techniques, traditionally reserved for the decoding of convolutional codes, are applied to the decoding process of block codes.
- Comparisons between the Viterbi decoding techniques and the algebraic techniques are made. The comparisons are done for both hard decision and soft decision decoding.
- The trellis construction techniques are then extended to include the construction of trellises for the Reed-Solomon family of codes.
- A novel trellis construction technique for Reed-Solomon codes is presented which reduces the amount of calculations required during the construction of such a trellis.

- Decoding of the Reed-Solomon codes is attempted employing the Viterbi algorithm.

- The dissertation shows that maximum likelihood decoding can be achieved with the techniques mentioned above, and that it compares favorably with the traditional algebraic decoding results.

- An algorithm is investigated which decreases trellis complexity, making decoding of larger block codes possible with the Viterbi algorithm. Simulation results are presented to verify the decoding performance using the lower complexity trellises.

1.3 Outline of Dissertation

In this chapter, the *Introduction*, a framework is provided on which the entire dissertation is based.

The second chapter, *Foundations of Channel Coding*, deals with important coding and information theory concepts. The emphasis in this chapter is placed on coding theory, and all the relevant topics are discussed. The most important aspect of the coding theory as far as this dissertation is concerned, is the discussion of block codes and their properties. This discussion will provide the foundation on which all other chapters are based. Throughout the remainder of the work, reference will be made to the second chapter of the dissertation.

Decoding of Block Codes is the third chapter of the dissertation, and employs the technique discussed in the previous chapter to realise a method for decoding block codes. Here techniques normally reserved for the decoding of convolutional codes are adapted and used for the decoding of block codes. The main focus in this chapter is the creation of trellis diagrams for block codes. This allows the aforementioned decoding processes to be used for block codes. Not all the known trellis construction techniques are discussed in this chapter, but reference is made to the appendixes at the end of the dissertation for more information.

The chapter, *Performance Issues of Various Block Codes*, investigates the performance of a selected group of block codes, namely the Hamming, Reed-Muller and BCH families

of codes. Error rate curves of the block codes obtained by using the Viterbi algorithm are compared to curves obtained by traditional algebraic decoding techniques. These curves are then also compared to the theoretical curves obtained by calculation. It is shown that maximum likelihood decoding is achieved by employing the convolutional decoding techniques on block codes.

Trellis Construction for Reed-Solomon Codes is discussed in chapter 5. The importance of this group of error correcting block codes has already been pointed out. Very little information is available on this topic in literature, and therefore a great amount of detail is presented in this chapter. Again, the first step for Viterbi decoding of Reed-Solomon codes is the construction of a trellis diagram for the code. As Reed-Solomon codes are huge compared to any other type of code, their trellis diagrams are too large to illustrate schematically. For this reason a small code was chosen for illustration purposes. A topological analysis is done on the constructed trellis, and several novel findings are made, resulting in a novel "topological" trellis construction technique for Reed-Solomon codes. The performance of this family of codes, when used together with convolutional decoders, is examined. Error rate curves are plotted in order to demonstrate that decoding of Reed-Solomon codes employing either the Viterbi or MAP algorithms is in fact possible. This ultimately means that Reed-Solomon codes can be maximum likelihood soft decision decoded by the same algorithms normally used for convolutional codes.

Chapter 6 presents an introduction to trellis complexity, which is accompanied by a technique for the successful reduction of trellis complexity. As mentioned earlier, the only limiting factor in the trellis decoding process is the actual complexity of the trellis. The larger the trellis, the more complex the decoding, and this will ultimately make the decoding of large block codes impractical. A simple example is used to show that this proposed technique is in fact able to reduce the trellis complexity by a significant factor. The effect of this technique is that larger codes can now be decoded with the same amount of effort required to decode smaller codes, since the trellis size of the larger codes can be reduced significantly. This counteracts the apparent drawback of block code decoding with the Viterbi Algorithm. Simulation results are presented which show that there is no degradation in error correcting performance when such a reduced trellis is used in the decoding process.

A *Summary* is provided in chapter 7. As mentioned earlier, this chapter should be read

together with the current one in order to determine which of the goals and objectives have been achieved. An objective stance is taken and the goals are evaluated in the last chapter of the dissertation.

Apart from the chapters listed above, there are a total of eight Appendixes which provide more information regarding several topics under discussion to the interested reader.

1.4 Major Contributions

The major contributions made in this work are as follows:

- Trellis complexity resolution.
- Novel trellis construction techniques based on Wolf's method.
- Simulation results for verification of techniques.

Chapter 2

Foundations of Channel Coding

2.1 Introduction

This chapter is dedicated to describing the general principles of channel coding and error correcting codes. The two main groups of error correcting codes that will be discussed here, are block codes [3], [4], [8] and convolutional codes. In order to describe exactly how a trellis for a block code is constructed in the following chapter, it is essential that several definitions and descriptions be given in this chapter. For this reason, this chapter is divided into four parts. The first part gives a general description of the fundamentals of block coding. This is then followed by a description of convolutional codes. The third part illustrates several decoding principles, along with a detailed derivation of metrics. At the end of this section, a short description of general methods for block code decoding will be given. The last section will detail the Viterbi algorithm [14], as this is, besides the MAP algorithm, still the most important algorithm available for decoding block code trellises.

2.2 Block Codes

Besides providing a mathematical framework for block codes[3], [4], [8], [15], this section mainly contains a description of block codes. Several coding terms such as Hamming-weight, Hamming-distance, minimal-distance and error correcting capability, which are used freely throughout this work and other standard works on this topic, will be described and illustrated. The prior knowledge of material contained in this chapter is of utmost importance before attempting to master any additional information and methods contained within this work.

2.2.1 Linear Block Code, Generator Matrix and Parity Check Matrix

The vectors of the n-dimensional vector space $GF(q)^n$ contain n components, which are taken from the symbol alphabet $GF(q) = \{0,1,...,q-1\}$. If any arbitrary subset of this vector space contains q^k vectors, then this set is termed a (n, k)-block code C with code rate $R = k/n$ and its elements are the codewords $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1})$. If the addition of two codewords and the multiplication of one codeword with an element of $GF(q)^n$ gives another codeword, the code will be a subset of $GF(q)^n$ termed a *linear block code*. From the above reasoning, it follows that the zero vector **0** has to be contained within every linear block code.

If the possible 2^k information vectors $\boldsymbol{i} = (i_0, i_1, \ldots, i_k)$ are each assigned to a codeword C in a specific way, a code book is formed. The specific manner of assigning these vectors is of utmost importance, as this is what differentiates one code in the space from another one. A linear assignment proves very effective with linear block codes, as the code can then be described by a multiplication of the information vector **i** and a $(k \times n)$-matrix G:

$$\boldsymbol{c} = \boldsymbol{i} \cdot \boldsymbol{G} \tag{2.1}$$

The matrix **G** with elements from $GF(q)$ has a rank k and is termed the *generator matrix*. The row vectors $\mathbf{g}^i$ form a possible base for the subset of $GF(q)^n$ described by C, in that every codeword of C is a linear combination of the row vectors $\mathbf{g}^i$ of **G**, with $i = 0,1,...,k$. The vectors $\mathbf{g}^i$ can be described as either a column vector or a row vector, depending on whether a subscript or superscript notation is used, i.e.

$$\textit{Column vectors} \quad g_j = \begin{pmatrix} g_{1j} \\ g_{2j} \\ \vdots \\ g_{kj} \end{pmatrix}, \qquad j = 1,\ldots,n \tag{2.2}$$

$$\textit{Row vectors} \quad g^i = (g_{i1}, g_{i2}, \ldots, g_{in}), \quad i = 1,\ldots,k \tag{2.3}$$

with

$$\boldsymbol{G} = (g_{ij}) = \begin{pmatrix} g_{11} & g_{12} & \cdots & g_{1n} \\ g_{21} & g_{22} & \cdots & g_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k1} & g_{k2} & \cdots & g_{km} \end{pmatrix} = (g_1, g_2, \ldots, g_n) = \begin{pmatrix} g^1 \\ g^2 \\ \vdots \\ g^k \end{pmatrix}. \tag{2.4}$$

The set of all vectors, which are orthogonal to the vectors of C, the so called orthogonal complement $C_\perp$ of C, is called the dual code of C. The dimensions of this code is n - k. If any n - k base vectors of $C_\perp$ are randomly chosen for the rows $\boldsymbol{h}^i$ of the $((n - k) \times n)$-matrix **H**, then the following holds

$$\boldsymbol{c} \cdot \boldsymbol{H}^T = 0, \tag{2.5}$$

where 0 is the zero vector of length n - k. The matrix **H**, which already identifies the linear block code C without the use of the accompanying generator matrix **G**, is termed the *parity check* matrix.

Since the rows $\boldsymbol{g}^i$ of the generator matrix **G** are possible codewords, it follows from equation (2.5), that

$$\boldsymbol{G} \cdot \boldsymbol{H}^T = 0, \tag{2.6}$$

where **0** is the $(k \times (n - k))$-zero vector.

2.2.2 Equivalent Codes and Systematic Codes

If a (n,k) block code C is defined by its generator matrix **G**, an equivalent code with the same parameters can be constructed if the following operations are performed[8]:

- Elementary row operations:
  - Exchanging two rows
  - Multiplication of a row with a field element $\alpha \in (GF(q) \setminus \{0\})$
  - Replacing a row by the addition of that same row and a multiple of any other row
- Exchanging any column vectors

If a chosen number of operations are performed in succession, then the resulting manipulation can be described by a matrix multiplication $\mathbf{T}_G \cdot \mathbf{G}$, where $\mathbf{T}_G$ is a regular $(k \times k)$ matrix with elements taken from $GF(q)$. With this type of manipulation, it is possible, without changing the original code C (the arrangement of the code vectors $\mathbf{c}^i$ do however change), to rearrange any generator matrix into a form where k of the n columns are represented by unit vectors of length k. By exchanging the appropriate k rows, the generator matrix can be transformed to:

$$\boldsymbol{G}' = (\boldsymbol{I}_k | \boldsymbol{A}). \tag{2.7}$$

In the above equation $\mathbf{I}_k$ is the $(k \times k)$ unit matrix and $\mathbf{A}$ is a $(k \times (n - k))$ matrix. A code with a generator matrix in this form is termed a systematic code. The first k code symbols correspond with the information vector $\mathbf{i}$, since

$$\boldsymbol{c} = \boldsymbol{i} \cdot \boldsymbol{G}' = (\boldsymbol{i} | \boldsymbol{i} \cdot \boldsymbol{A}) = (i_0, i_1, \ldots, i_{k-1}, c_k, c_{k+1}, \ldots, c_{n-1}). \tag{2.8}$$

where the $\{c_j\}$, $j = k,\ldots,n - k$ denote the parity bits of the codeword C.

Although not explicitly proven here, it is shown that every block code can be transformed into a corresponding systematic code.

If the generator matrix $\mathbf{G}$ is transformed into the systematic form as per equation (2.7), then the corresponding parity check matrix $\mathbf{H}$ can be found accordingly:

$$\boldsymbol{H}' = (-\boldsymbol{A}^T | \boldsymbol{I}_{n-k}), \tag{2.9}$$

where $\mathbf{I}_{n-k}$ indicates the $((n - k) \times (n - k))$ unit matrix. This is fairly simple to prove, since

$$\boldsymbol{G}' \cdot \boldsymbol{H}'^T = (\boldsymbol{I}_k | \boldsymbol{A}) \cdot \begin{pmatrix} -\boldsymbol{A} \\ \boldsymbol{I}_{n-k} \end{pmatrix} = -\boldsymbol{A} + \boldsymbol{A} = 0 \tag{2.10}$$

as per equation (2.6).

Just like the row and column operations performed on the generator matrix **G** do not affect the code, so does the performing of row and column operations on the parity check matrix not affect the code at all. The entire transformation process of the parity check matrix **H** can be described, as before, by a matrix multiplication $\mathbf{T}_H \cdot \mathbf{H}$, where $\mathbf{T}_H$ is a regular $((n - k) \times (n-k))$ matrix with elements taken from $GF(q)$. Similar to the generator matrix, the exchange of columns again provides an equivalent (but not the same) code.

Often an equivalent form of the original code C may facilitate a far simpler decoding procedure. It is often also desirable (for several reasons) to describe a block code C in its systematic form C'. Simple decoding procedures have also been developed for systematic error correcting block codes.

2.2.3 Cyclic Codes - Generator and Parity Check Polynomials

When any shifted version of a linear block code C produces another valid codeword in C, then this code is labeled *cyclic*. As shown in a few standard works, any cyclic code can be described by a generator polynomial [8] in the form:

$$g(x) = g_0 + g_1 \cdot x + g_2 \cdot x^2 + \ldots + g_{n-k} \cdot x^{n-k} \tag{2.11}$$

In relation to this, the coefficients of the code polynomial

$$c(x) = c_0 + c_1 \cdot x + c_2 \cdot x^2 + \ldots + c_{n-1} \cdot x^{n-1} \tag{2.12}$$

are taken from the components of the code vector $c = (c_0, c_1, \ldots, c_{n-1})$.

A very simple relationship exists between the code vector $c(x)$ and the information vector $i(x)$, namely that one can directly multiply the information polynomial $i(x)$ of degree $\leq k - 1$ with the generator polynomial $g(x)$, and as a result obtain the code polynomial $c(x)$ of degree $\leq n - 1$.

$$c(x) = i(x) \cdot g(x) \tag{2.13}$$

The resulting generator matrix is given below:

$$\boldsymbol{G} = \begin{pmatrix} g_0 & g_1 & g_2 & \cdots & g_{n-k-1} & g_{n-k} & & & & 0 \\ & g_0 & g_1 & g_2 & \cdots & g_{n-k-1} & g_{n-k} & & & \\ & & g_0 & g_1 & g_2 & \cdots & g_{n-k-1} & g_{n-k} & & \\ & & & g_0 & g_1 & g_2 & \cdots & g_{n-k-1} & g_{n-k} & \\ 0 & & & & g_0 & g_1 & g_2 & \cdots & g_{n-k-1} & g_{n-k} \end{pmatrix} \quad (2.14)$$

Similar to equation (2.5), there exists a so called parity check polynomial $h(x)$ of degree k for cyclic codes for which the following holds true:

$$c(x) \cdot h(x) = 0 \mod(x^n - 1) \quad \textit{for all } c(x) \in C. \quad (2.15)$$

This leads to an expression:

$$g(x) \cdot h(x) = x^n - 1. \quad (2.16)$$

For any possible h(x), the parity check matrix could be given as follows:

$$\boldsymbol{H} = \begin{pmatrix} h_k & h_{k-1} & h_{k-2} & \cdots & h_2 & h_1 & h_0 & & & 0 \\ & h_k & h_{k-1} & h_{k-2} & \cdots & h_2 & h_1 & h_0 & & \\ & & h_k & h_{k-1} & h_{k-2} & \cdots & h_2 & h_1 & h_0 & \\ & & & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & & & & h_k & h_{k-1} & h_{k-2} & \cdots & h_2 & h_1 & h_0 \end{pmatrix}. \quad (2.17)$$

2.2.4 Syndrome

If a codeword is transmitted over a noisy channel, it is possible that some of the symbols of the transmitted code are in error. In other words, an error vector **e** is added to the code vector **c**:

$$\boldsymbol{r} = \boldsymbol{c} + \boldsymbol{e} \quad (2.18)$$

If the received vector r is multiplied with the transposed parity check matrix $\mathbf{H}^T$, then the answer is 0 only if $\mathbf{e} = \mathbf{0}$ (error free transmission), or if $\mathbf{e} \in C$, which in term translates $\mathbf{r}$ into another valid codeword. The expression $\mathbf{r} \cdot \mathbf{H}^T$ is called the *syndrome* s, and is not only dependant on the error vector $\mathbf{e}$, but also on the transmitted codeword c, since it can be shown with the help of equation (2.5) and (2.18) that the following holds true:

$$s = r \cdot H^T = (c + e) \cdot H^T = e \cdot H^T \tag{2.19}$$

2.3 Convolutional Codes

Similar to the previous section, this section presents the basic fundamental concepts of convolutional codes [5], [6]. Convolutional codes are the second large group of error correcting codes. Their structure is vastly different from block codes, and the generation of these codes is also done in a completely different way.

2.3.1 Convolutional Encoder

In the previous section it was shown that the n symbols of a block code were just dependant on the k information symbols of the relevant information vector. However, for a convolutional code, an additional parameter is required, namely M. The parameter M is called the *arrangement of memory* of the encoder, or in short, the *memory* of the convolutional encoder. The n code symbols of the convolutional code are thus a function of M previous information vectors.

An example of a convolutional encoder defined over GF(q) with $q = 2$, $R = k/n = 2/3$ and $M = 1$ is shown in **Figure 1**. For each branch i, $k = 2$ information symbols from the symbol alphabet $GF(q = 2)$ are shifted into the shift register. These are then combined with the previous information vector ($M = 1$) to form a codeword $\mathbf{c}_i$. The shift registers are loaded with zeros at the start of the encoding process.

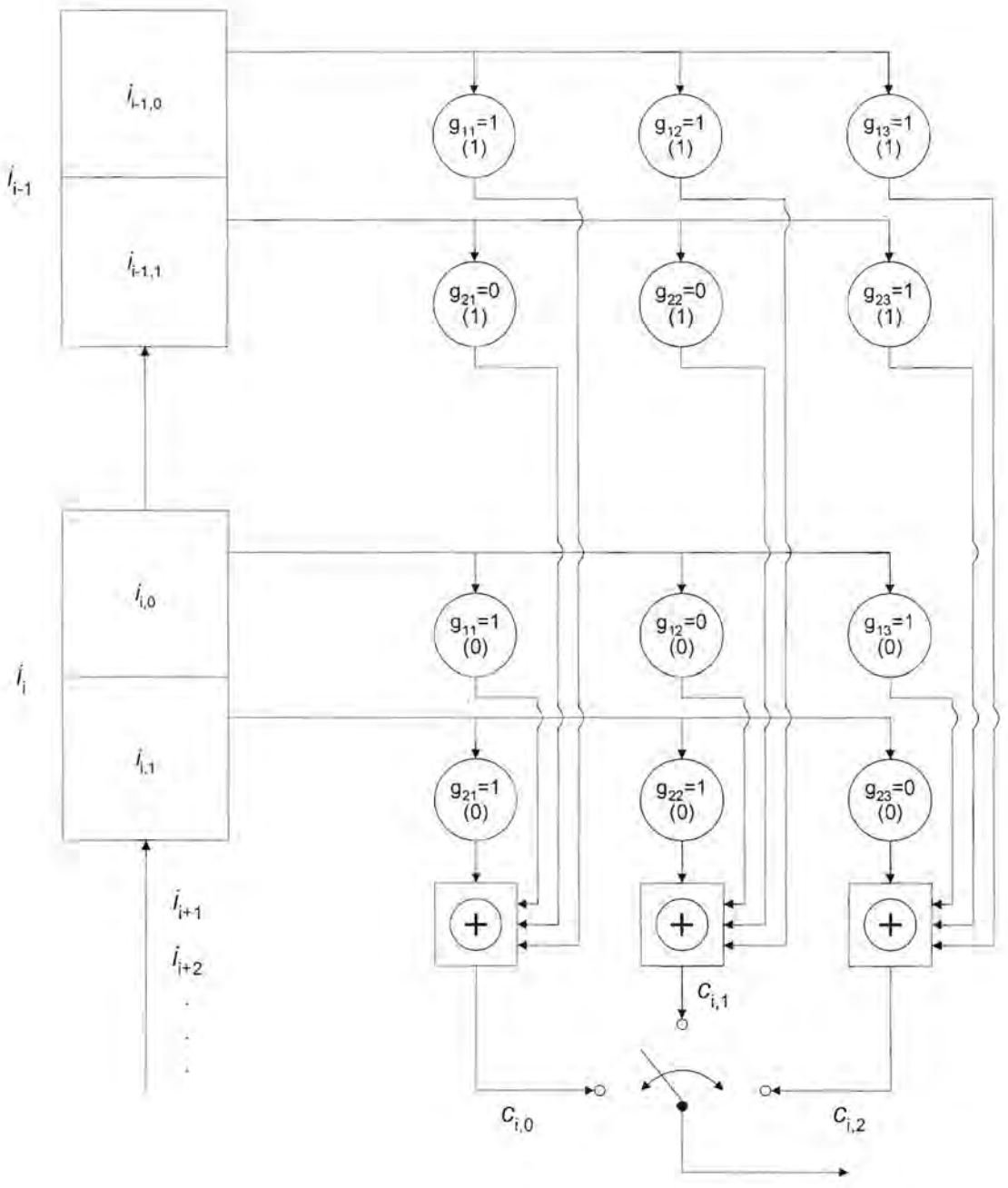

Figure 1 Example of a Convolutional Encoder with R = 2/3 and M = 1

Several methods of describing convolutional codes and convolutional encoders [5], [14] have been developed. For the purposes of this study, only the matrix representation and the representation in trellis form will be described. The above example of a convolutional encoder will be used to illustrate the representation forms.

2.3.2 Matrix Representation of Convolutional Codes

The mapping of the information sequence

$$\boldsymbol{i} = (i_0, i_1, i_2, \ldots, i_l, \ldots) \tag{2.20}$$

into the code sequence

$$\boldsymbol{c} = (c_0, c_1, c_2, \ldots, c_l, \ldots) \tag{2.21}$$

can be described (analog to equation (2.1)) by

$$\boldsymbol{c} = \boldsymbol{i} \cdot \boldsymbol{G} \tag{2.22}$$

where the generator matrix has the following semi-infinite form:

$$\boldsymbol{G} = \begin{pmatrix} G_0 & G_1 & G_2 & \cdots & G_{M-1} & G_M & & 0 \\ & G_0 & G_1 & G_2 & \cdots & G_{M-1} & G_M & \\ & & G_0 & G_1 & G_2 & \cdots & G_{M-1} & G_M \\ 0 & & & \ddots & \ddots & \ddots & \ddots & \ddots \end{pmatrix}. \tag{2.23}$$

The (k × n) sub-matrixes

$$\boldsymbol{G}_l = (g_{ij}^{(l)}) = \begin{pmatrix} g_{11}^{(l)} & g_{12}^{(l)} & \cdots & g_{1n}^{(l)} \\ g_{21}^{(l)} & g_{22}^{(l)} & \cdots & g_{2n}^{(l)} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k1}^{(l)} & g_{k2}^{(l)} & \cdots & g_{kn}^{(l)} \end{pmatrix} \tag{2.24}$$

can be read directly from **Figure 1**. In this example, the matrix is as follows:

$$\boldsymbol{G} = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 1 & & & & & & \cdots & & \cdots & \cdots \\ 1 & 1 & 0 & 0 & 0 & 1 & & & \mathbf{0} & & & & \cdots & & \\ & & & 1 & 0 & 1 & 1 & 1 & 1 & & & & & & \\ & \mathbf{0} & & 1 & 1 & 0 & 0 & 0 & 1 & & \mathbf{0} & & \cdots & \cdots & \\ & & & & & & 1 & 0 & 1 & 1 & 1 & 1 & & & \\ & \mathbf{0} & & & \mathbf{0} & & & & & & & & \mathbf{0} & \cdots & \\ & & & & & & 1 & 1 & 0 & 0 & 0 & 1 & & & \\ & & & & & & & & & & \ddots & & \ddots & & \end{pmatrix} \tag{2.25}$$

with

$$\boldsymbol{G}_0 = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \tag{2.26}$$

and

$$\boldsymbol{G}_1 = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.27}$$

The generator matrix can also be represented in a polynomial form, as follows:

$$\boldsymbol{G}(D) = (g_{ij}(D)) = \begin{pmatrix} g_{11}(D) & g_{12}(D) & \cdots & g_{1n}(D) \\ g_{21}(D) & g_{22}(D) & \cdots & g_{2n}(D) \\ \vdots & \vdots & \ddots & \vdots \\ g_{k1}(D) & g_{k2}(D) & \cdots & g_{kn}(D) \end{pmatrix}. \tag{2.28}$$

The elements $g_{ij}(D)$ of the above matrix are polynomials of degree $M_{ij} \leq M$

$$g_{ij}(D) = g_{ij}^{(0)} + g_{ij}^{(1)} \cdot D + g_{ij}^{(2)} \cdot D^2 + \ldots + g_{ij}^{(M_{ij}-1)} \cdot D^{M_{ij}-1} + g_{ij}^{(M_{ij})} \cdot D^{M_{ij}} \tag{2.29}$$

with the coefficients $g_{ij}^{(l)}$ of the matrixes $\mathbf{G}_l$ from equations (2.23) and (2.24).

$\mathbf{G}(D)$ can thus be written as

$$\boldsymbol{G}(D) = \boldsymbol{G}_0 + \boldsymbol{G}_1 \cdot \boldsymbol{D} + \boldsymbol{G}_2 \cdot \boldsymbol{D}^2 + \ldots + \boldsymbol{G}_{M-1} \cdot \boldsymbol{D}^{M-1} + \boldsymbol{G}_M \cdot D^M. \tag{2.30}$$

The so called *memory size* of a convolutional encoder is defined as follows:

$$\nu_G = \sum_{i=1}^{k} \max_j M_{ij} \leq M \cdot k \tag{2.31}$$

In this example this would translate to:

$$\boldsymbol{G}(D) = \begin{pmatrix} 1+D & D & 1+D \\ 1 & 1 & D \end{pmatrix} \tag{2.32}$$

and

$$\nu_G = \sum_{i=1}^{2} \max_j M_{ij} = \max(1,1,1) + \max(0,0,1) = 1 + 1 = 2. \tag{2.33}$$

The memory size ν_G exactly determines the amount of registers used in the shift register structure of the convolutional encoder.

2.3.3 Trellis Diagram Representation

If one defines the contents of the registers ν_G containing the previous information symbols, which are used to calculate the current codewords, as states of the convolutional encoder, then a series of nodes develop. These series of nodes allow the representation of the q^{ν_G} possible states, and emanating branches that represent the

q^k possible information vectors at a certain instant in time. This graph of topologically arranged nodes and branches is called a trellis diagram. In short, it can be describes as the time equivalent representation of the state diagram of the convolutional encoder.

For the convolutional encoder presented in **Figure 1** of this section, the corresponding trellis diagram is given in **Figure 2** below.

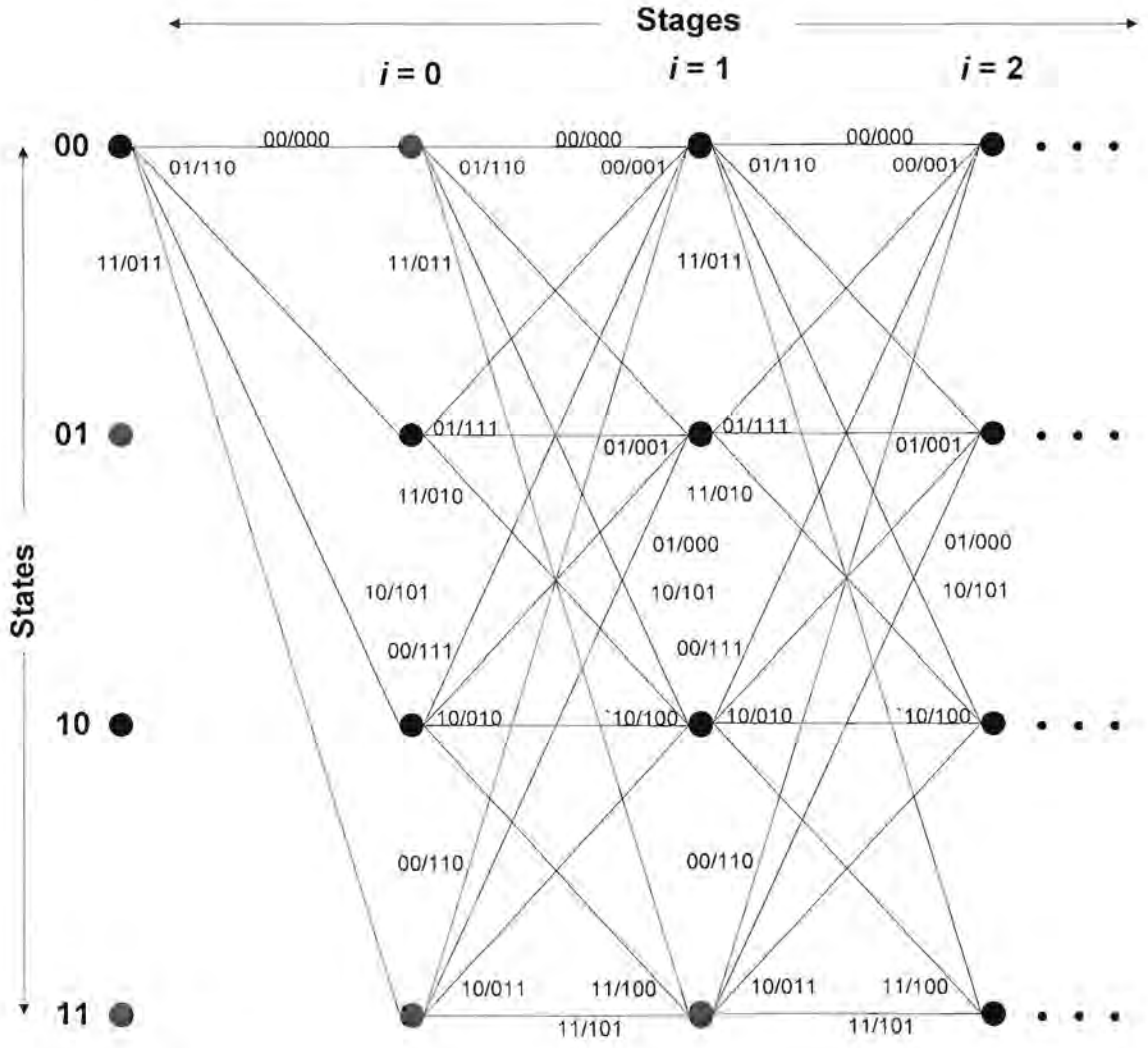

Figure 2 Trellis Diagram of the Convolutional Encoder from Figure 1

The vectors on the left hand side of the figure indicate the state that the specific encoder is in. On the branches, the relevant information vectors along with their respective code vectors are indicated. The notation that is used on the branches is a general

input/output notation, and this notation will be used throughout the dissertation. This means that the information vectors are listed on the left hand side and the resulting output or code vectors are listed on the right hand side.

After M stages, when the information vector has reached the last register block, the trellis is fanned out. This means that all the branch transitions from that stage will be the same as the branch transitions from any stage after that particular stage where fan-out has occurred.

2.4 On the Decoding of Block and Convolutional Codes

In this section, several decoding principles of error correcting codes will be presented. This will be the basis on which the latter chapters on decoding will be based.

2.4.1 General Decoding Principles

In **Figure 3**, a simple structure for a digital transmission system [10] utilizing a channel encoder and a channel decoder is shown. At the encoder, k information symbols (an information vector $\mathbf{i}$) are mapped onto n code symbols (a codeword $\mathbf{c}$). The code sequence is then encoded modulation specifically in the block "*Transmitter*" and transformed into a transmission signal (*modulation*). This signal is then transmitted over the "Additive White Gaussian Noise" channel, where transmission errors are being introduced to the original error free signal. In the "*Receiver*" block, the received signal is demodulated to produce the estimated received sequence $\mathbf{r}$. The decoder establishes an estimated equivalent $\hat{\mathbf{i}}$ of the transmitted information sequence $\mathbf{i}$.

For simplicity reasons, a BPSK-modulation [10] is assumed. Furthermore, no inter-symbol interference is found in the channel. If this is the chase, then the blocks labeled "*Transmitter*", "*AWGN channel*" and "*Receiver*" in **Figure 3**, can be combined into a time-discrete memory-free channel, in which the j^{th} component of the received sequence $\mathbf{r}$ is only dependant on the j^{th} component of the code sequence $\mathbf{c}$.

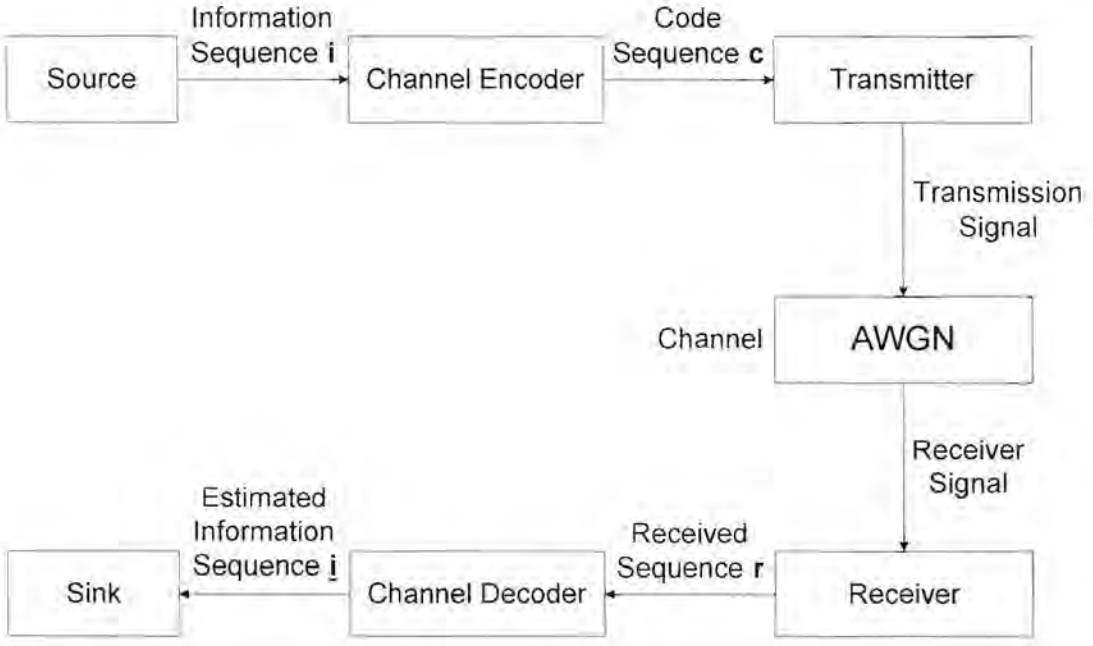

Figure 3 Structure of a Digital Transmission System

There are two possible ways how the receiver passes on the information (obtained from the demodulated received sequence) to the decoder:

- The receiver decides for every received code symbol the corresponding code symbol that was most likely sent. This is then passed on to the decoder without sending any other information about the certainty or likelihood of the decision. This method of decoding is termed *hard decision decoding*. This means that for the previously mentioned time-discrete memory-less channel, the components of the error vector **e** and the received vector **r** have the same range of values as the code symbols. In this binary case this would translate to elements from GF(2).
- The decoder passes an additional value to the decoder. This additional value expresses the reliability of the preceding decision for a particular code symbol. With a binary code alphabet this would mean that the real value and not just a one or zero is passed on to the decoder. For the case of a hard decision decoding process, the real valued received vector would be transformed into the closest matching binary symbol, and passed on to the decoder. With *soft decision decoding* the decoder receives the real valued symbols, and can use this additional information, often referred to as channel information, in the decoding

process. The effect of soft decision decoding, is that for the same bit error rate, a low code rate code needs about 2 dB E_{bit}/N_0 less using this method, than with hard decision decoding. With higher code rates, this difference becomes smaller. It is intuitively assumed that the components of the error vector **e** and the received vector **r** are real valued.

Independent of which of the above mentioned schemes is used, the result of the decoding process can be grouped into either one of the following categories:

- *Correct Decoding*, which implies that the decoder has identified the errors introduced by the channel, and has corrected them. The decoder thus delivers the same information that was initially sent to the information sink.
- *Incorrect Decoding*, which means that the decoder does not deliver to the information sink the same information that was initially sent.
- *Decoding Failure*, implies that the decoder does not deliver any information vector to the information sink.

The following methods are envisaged in the decoding process:

- *Error Detection*
  Here the decoder only checks whether or not the received vector represents a valid codeword. If this is not the case, then the decoder does not perform any operation aimed at correcting such an inherently incorrect vector. In the case where $\mathbf{e} \in C$, the decoder is not even able to recognize the error seeing that $\mathbf{r} \in C$ even though $\mathbf{r} \neq \mathbf{c}$.
- Maximum Likelihood Decoding
  If the decoder makes a decision (for any given received vector r) about which one of the codewords $\hat{\mathbf{c}}$ was most likely sent, in other words,

$$P(\hat{c}|r) = \max_{c \in C} P(c|r) \tag{2.34}$$

then it is called the *Maximum-A-Posteriori* (MAP) decision rule. According to Bayes theorem, the following holds true:

$$P(c|r) = \frac{P(r|c) \cdot P(c)}{P(r)}. \tag{2.35}$$

Since we are dealing with the received vector **r**, the probability of it being the one out of all the possible codewords,

$$P(r) = \sum_{all\ c \in C} P(r|c) \cdot P(c) \tag{2.36}$$

is constant, as long as it is the vector under investigation. If the codewords **c** are all equiprobable (in other words, all the possible codewords *A-Priori-Probabilities* $P(c) = 1/q^k$ are the same), then the following decision rule applies:

$$P(r|\hat{c}) = \max_{c \in C} P(r|c) \tag{2.37}$$

If the decoding is done according to this principle, then it is termed *Maximum Likelihood* decoding.

- *Bounded Minimal Distance Decoding*
  In this method, not all the possible received vectors r are decoded, but only those in which a bounded amount of errors exist. A possible undesirable outcome of this type of decoding, is decoding failure.

For the different decoding methods [10], error probabilities can be defined as follows:

- *Block Error Probability*

$$P_{Block} = \lim_{\mu \to \infty} \frac{Number\ of\ incorrectly\ decoded\ codewords}{Number\ \mu\ of\ sent\ codewords} \tag{2.38}$$

- *Bit Error Probability*

$$P_{Bit} = \lim_{v \to \infty} \frac{Number\ of\ incorrectly\ decoded\ information\ bits}{Number\ v\ \ of\ sent\ information\ bits} \tag{2.39}$$

The optimal decoding (*i.e.*, the error probability is minimized) of a binary block code or convolutional code with the incorporation of reliability information, (*i.e.*,soft decision decoding) will be investigated in more detail for the BPSK modulation system transmitting over a AWGN-channel.

To derive an expression for the decoding strategy and the metric, the detailed system representation of **Figure 4** is used. The already described time discrete, memory-less channel adds to the transmitted sequence a real valued error vector **f**. This is not to be confused with the error vector *e* with components from *GF*(2). As explained in the definition of soft decision decoding, the channel decoder uses this real valued received sequence **y** for decoding, and also assumes the function of the modulation specific decoding.

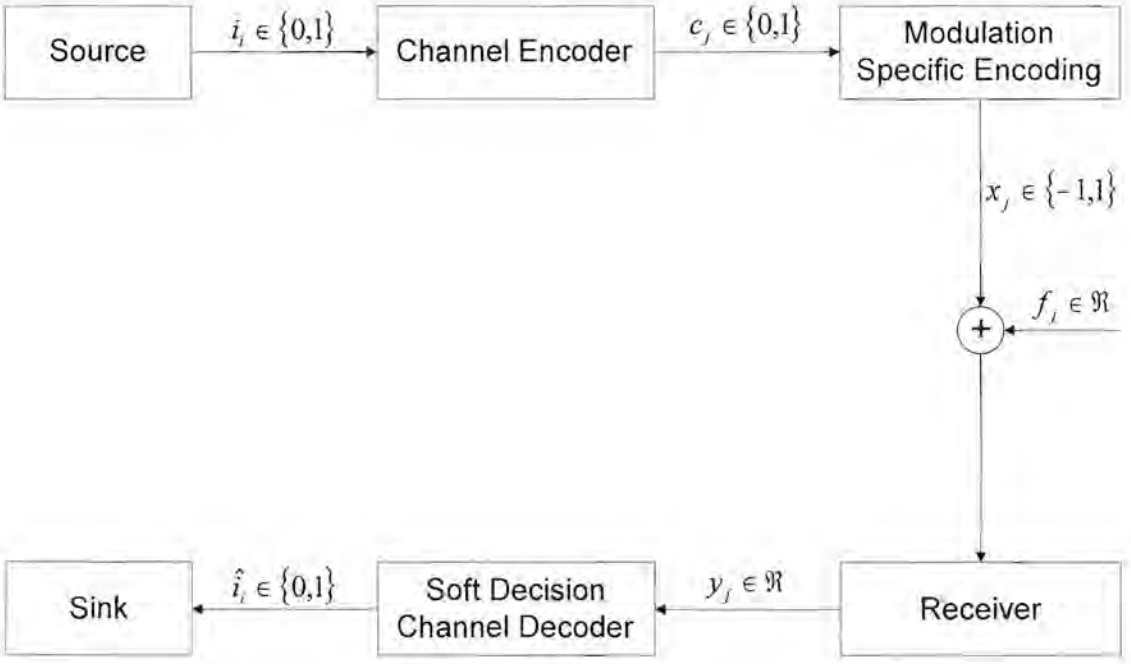

Figure 4 System Model for the Derivation of Decoding Strategy

The components f_j of the error vector **f** are (given the present assumptions) normally distributed random variables with the following density function:

$$\varphi\left(f_j\right) = \frac{1}{\sqrt{2\pi}\cdot\sigma}\cdot e^{-\frac{1}{2}\left(\frac{f_j}{\sigma}\right)^2} \tag{2.40}$$

with $\sigma^2 = N_0$ where N_0 is the single sided noise power spectral density. Analog to the previous discussion, the components y_i of the received vector **y** are also normally distributed random variables with average values of -1 and 1, dependant on the sent value x_j of the transmission sequence **x**. It follows that the density function of y_j is also dependant on the variable x_j.

$$\varphi\left(y_j\middle|x_j\right) = \frac{1}{\sqrt{2\pi}\cdot\sigma}\cdot e^{-\frac{1}{2}\left(\frac{y_j - x_j}{\sigma}\right)^2}. \tag{2.41}$$

The two density functions[10] $\varphi\left(y_j\middle|x_j = -1\right)$ and $\varphi\left(y_j\middle|x_j = 1\right)$ are shown in **Figure 5**.

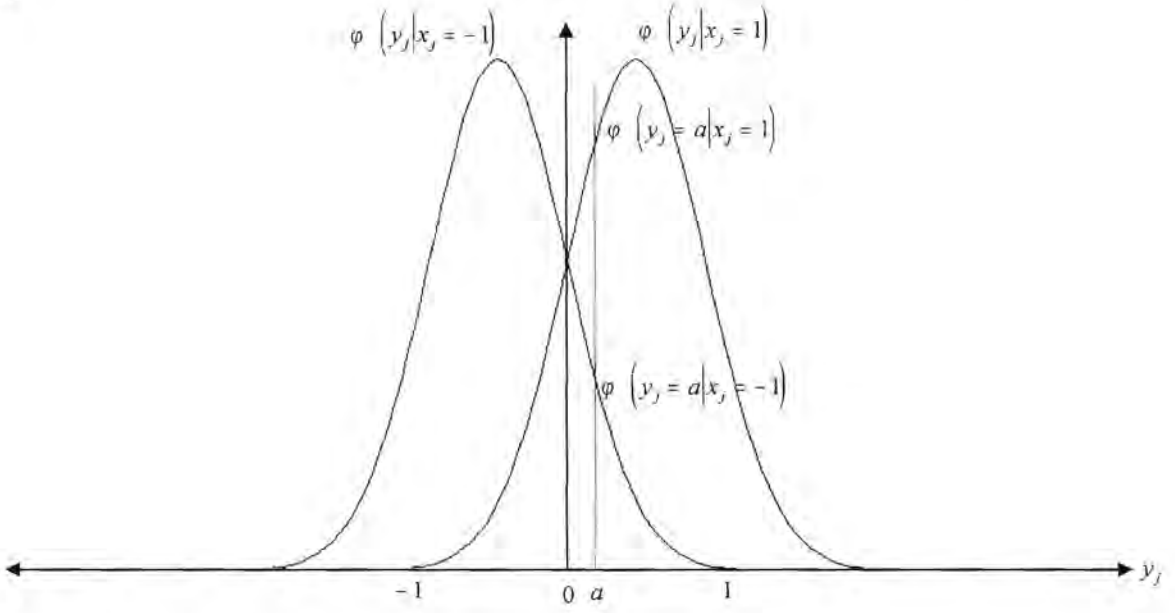

Figure 5 Probability Density Functions

Since f_j and y_j are stationary probability values (i.e. their density functions

$$\Phi\left(f_j\right) = \int_{s=-\infty}^{f_j} \varphi(t) \cdot dt \quad i.e. \quad \Phi\left(y_i \middle| x_j\right) = \int_{s=-\infty}^{y_j} \varphi\left(t \middle| x_j\right) \cdot dt \tag{2.42}$$

are stationary), the probability $P\left(y_j = a \middle| x_j\right)$ is zero for any arbitrary value of a.

For the purpose of the following discussion, the MAP decision rule of equation (2.34) is considered. If, instead of the received vector **r** with discrete values for the components r_j, the vector **y** with real valued components y_i is used, then equation (2.34) changes to:

$$P\left(\hat{x} \middle| y\right) = \max_{all\ x} P\left(x \middle| y\right). \tag{2.43}$$

Since, as explained above, no more discrete probabilities P(**y**|**x**) exist, the terms P(**r**|**c**) and P(**r**) from equation (2.34) have to be replaced with φ(**y**|**x**) and φ(**y**) respectively.

$$P\left(x \middle| y\right) = \frac{\varphi\left(y \middle| x\right) \cdot P(x)}{\varphi\left(y\right)} \tag{2.44}$$

where

$$\varphi\left(y\right) = \sum_{all\ x} \varphi\left(y \middle| x\right) \cdot P(x), \tag{2.45}$$

which for a given **y**, just as with P(r) from equation (2.34), results in a constant value. Due to the before mentioned fact, the decision rule as stated in equation (2.43) can be rewritten into the following decision rule:

$$\varphi\left(y \middle| \hat{x}\right) = \max_{all\ x} \varphi\left(y \middle| x\right) \tag{2.46}$$

This means, that the most likely sent vector $\hat{x}$ for equally likely transmitted vectors **x** is the one which maximizes the above probability density function.

The expression $\varphi(y|x)$ to be maximized, is transformed with the assistance of equation (2.41). The vectors **x** and **y** each have n components in this case.

$$\varphi(y|x) = \prod_{l=1}^{n} \varphi(y_l|x_l) = \prod_{l=1}^{n} \frac{1}{\sqrt{2\pi}\cdot\sigma} \cdot e^{-\frac{1}{2}\left(\frac{y_l - x_l}{\sigma}\right)^2} = \left(\frac{1}{\sqrt{2\pi}\cdot\sigma}\right)^n \cdot e^{-\frac{1}{2\sigma^2}\sum_{l=1}^{n}(y_l - x_l)^2} \quad \textbf{(2.47)}$$

To maximize this expression, it is necessary to minimize the sum in the exponential function.

$$\sum_{l=1}^{n}(y_l - x_l)^2 = \sum_{l=1}^{n}(y_l^2 - 2x_l y_l + x_l^2) = \sum_{l=1}^{n} y_l^2 - 2\sum_{l=1}^{n} x_l y_l + \sum_{l=1}^{n} x_l^2 \quad \textbf{(2.48)}$$

From the above expression it follows that $\sum y_l^2$ (given the received vector **y**) as well as $\sum x_l^2 = n$ are constant, which implies that in order to minimize the given sum, only $\sum x_l y_l$ has to be maximized.

In conclusion, it can be stated that with BPSK transmission of n code symbols over an AWGN channel, the Maximum-Likelihood-Decoder maximizes the following sum:

$$\sum_{l=1}^{n} x_l y_l = x \cdot y^T \quad \textbf{(2.49)}$$

In other words, the scalar product of the sent vector x and the received vector y is maximized by the Maximum-Likelihood-Decoder.

2.4.2 Decoding of Block Codes

For any given code, there usually exists a multitude of different decoding procedures [10], [1], [2]. The choice of any given decoding procedure above any other is usually governed by the type of application. Other factors which contribute to the choice are, for instance, the availability of computing power, the speed of the specific decoding and the ease of implementation of given algorithm. These factors obviously also have to be considered when choosing a code, since these factors will impose restrictions on the type of code to be used, and also limit the parameter choices for the code (e.g. the possible

values of n and k may be severely limited by the non-availability of decoding power).

If in addition, channel state information also has to be considered in the decoding process (Soft-Decision Decoding) and the decoding done via a Maximum-Likelihood process (the procedure will be termed SDML decoding in future), then there are only very few decoding procedures that allow for a practical decoding of given block code. Theoretically, every linear block code can be decoded by a SDML decoding process, in that every received vector is compared to all q^k possible sent vectors and the most likely one chosen as per equation (2.43). It can be seen that in a decoding procedure utilizing the above outlined method, q^k scalar products have to be calculated, and the maximum chosen. Since the number of calculations increase exponentially with k, it is not a very viable decoding procedure for codes with large k. The main topic of this work is to provide procedures that are able to provide more efficient decoding for high rate codes with small n-k. The following chapters will be dedicated to the development and fine-tuning of such procedures.

2.4.3 Decoding of Convolutional Codes

The fact that with convolutional codes the n code symbols are not only dependant on the k information symbols of the current information vector, but also on M previous information vectors, is the reason why this decoder has a lot more information available to do the decoding on than a block decoder. The above comparison holds true for all codes with comparable parameters such as n and k. This then also explains why the decoding results for memory-less channels obtained by the use of a convolutional decoder are superior to those obtained with block decoders.

In order to perform SDML decoding on a convolutional code with a code set C of T code blocks, it is necessary to compare this with all the possible $q^{T \cdot k}$ code sequences as describes in equation (2.43). Viterbi showed in 1967 that in the Trellis diagram of **Figure 2**, it is not necessary to compare all qk paths with the received vector, but that it is sufficient, at any given time instant i, to only consider the incoming path into a state having the "best" metric. By applying equation (2.49) the assumption is made that the decoder is in a state which best corresponds to the most likely sent code sequence. This path is termed the survivor path.

The following figure outlines the process proposed by Viterbi [14].

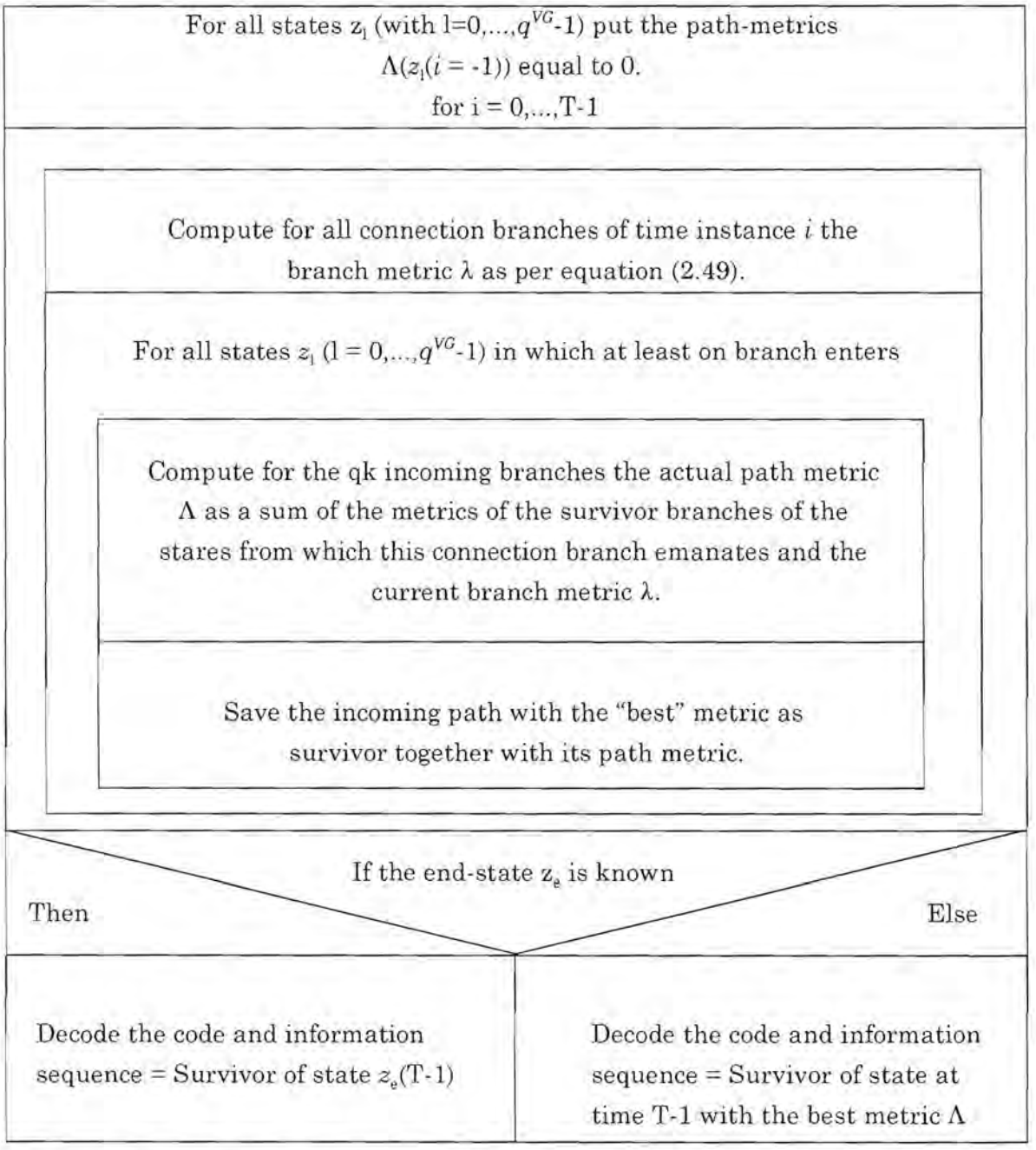

Figure 6 Schematic Representation of the Viterbi Algorithm

In the next chapter, the principles given are used in order to provide SDML methods for the decoding of block code trellises.

Chapter 3

Decoding of Block Codes

3.1 Introduction

In the previous chapter it was explained how every linear block code can theoretically be Soft-Decision-Maximum-Likelihood decoded (SDML-decoded) by comparing the received vector with all possible q^k sent vectors, and the most likely one chosen. In this chapter the two most popular non-algebraic decoding methods for block codes are presented. The first one considered is the method suggested by Bahl, Cocke, Jelinek and Raviv in 1974 [2], and the second one the procedure described and outlined by Wolf [16]. Both methods reduce the number of comparisons required dramatically. In **Section 3.1**, the term *Code Trellis Diagram* [5] is introduced and explained. A special case of the code trellis diagram is termed the *Syndrome Trellis Diagram* [5], the construction of which is described in **Subsection 3.2**. In the above mentioned code trellis diagram SDML-decoding can be performed by applying the Viterbi algorithm as described in the last section of **Chapter 2**. Firstly however, the method to obtain the parity check matrix **H**, required to construct the code trellis diagram from the generator matrix **G**, is explained in **Subsection 3.3**.

Throughout the chapter, a number of practical examples are given to illustrate various aspects of block code trellis construction.

3.2 Block Codes

A true code trellis (also termed a true code trellis diagram) of a code C is a trellis diagram,described in **Subsection 2.3.3**, with the property that a unique mapping between the codewords of the code C and the paths of the trellis diagram exists. That implies that for the case of a binary linear block code, as described in this section, the following holds true:

- The trellis diagram has a starting state (states will from time to time also be described as nodes) z_0 (i = -1) and an ending state z_e ($i = n$ - 1).
- Each connecting branch is assigned a binary value (i.e. a code bit), under the condition that two branches leaving the same state or node cannot have the same binary value assigned to them.
- A path from z_0 (i = -1) to z_e ($i = n$ - 1) with assigned code bits $c_0, c_1, \ldots, c_{n-1}$ exists when $c = (c_0, c_1, \ldots, c_{n-1}) \in C$.

The term "*State*" means exactly the same as the term "*Node*", and the naming of states and their ordering into a plane of states is completely arbitrary. In **Figures 7** and **8**, two possible "*Untrue*" trellis diagrams are shown, for the (3,2)-Parity Check Code with generator matrix:

$$\boldsymbol{G} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \tag{3.1}$$

The reason why they are called "*Untrue Trellis Diagrams*" is the fact that the second condition listed above is not met for each of the first nodes.

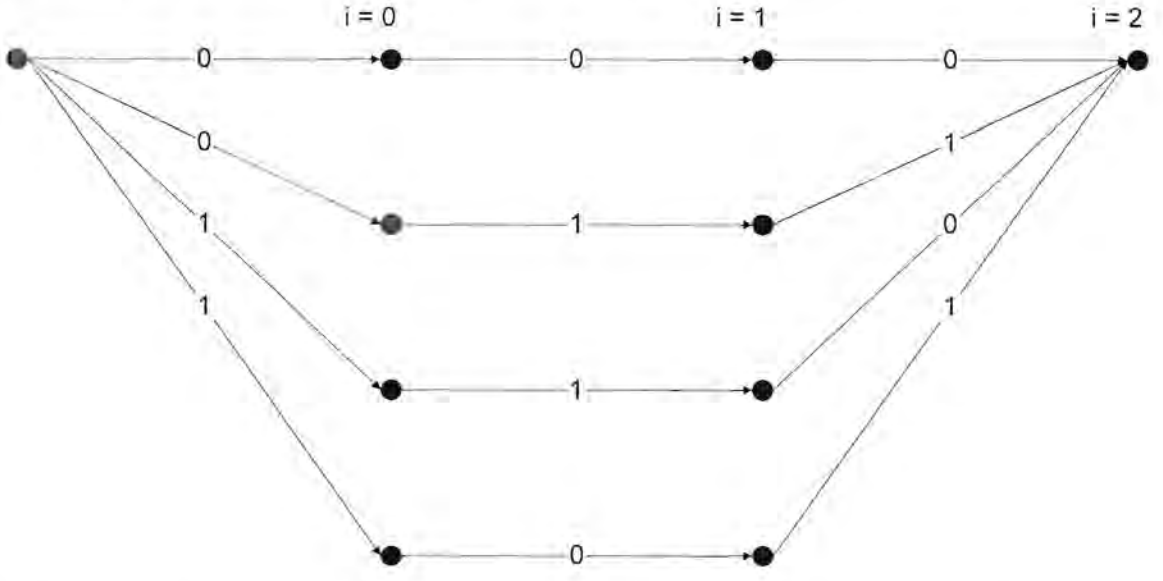

Figure 7 Untrue Trellis Diagram A for the (3,2)-Parity-Check-Code

It may never happen that after stepping through the trellis diagram one code symbol at a time one has to traverse backwards in order to find the correct path. The trellis diagram in **Figure 7** is called a trivial trellis diagram of a block code.

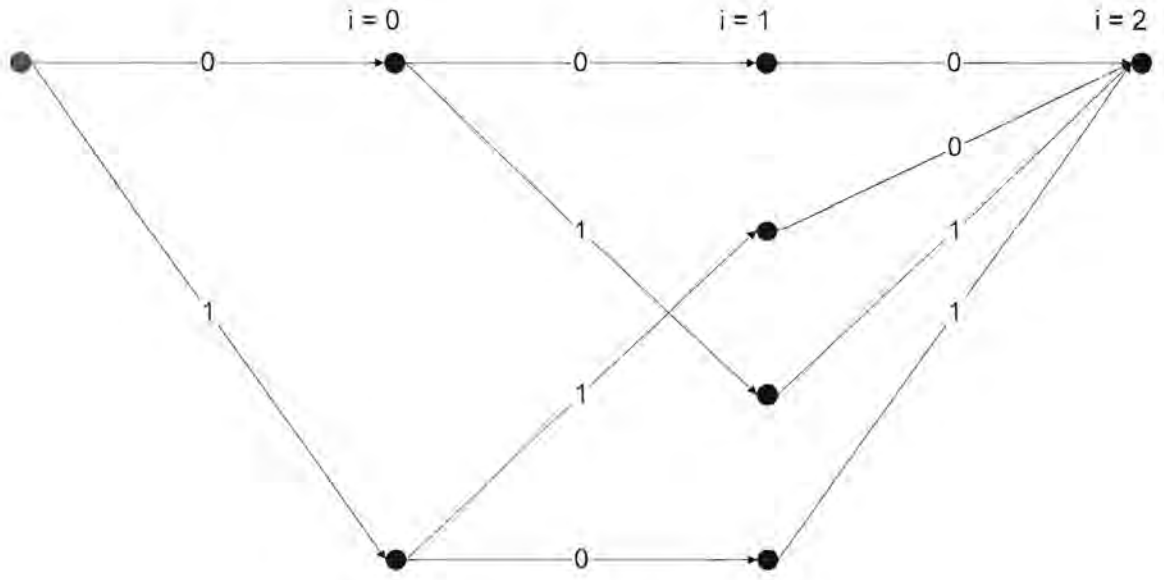

Figure 8 True Trellis Diagram A for the (3,2)-Parity-Check-Code

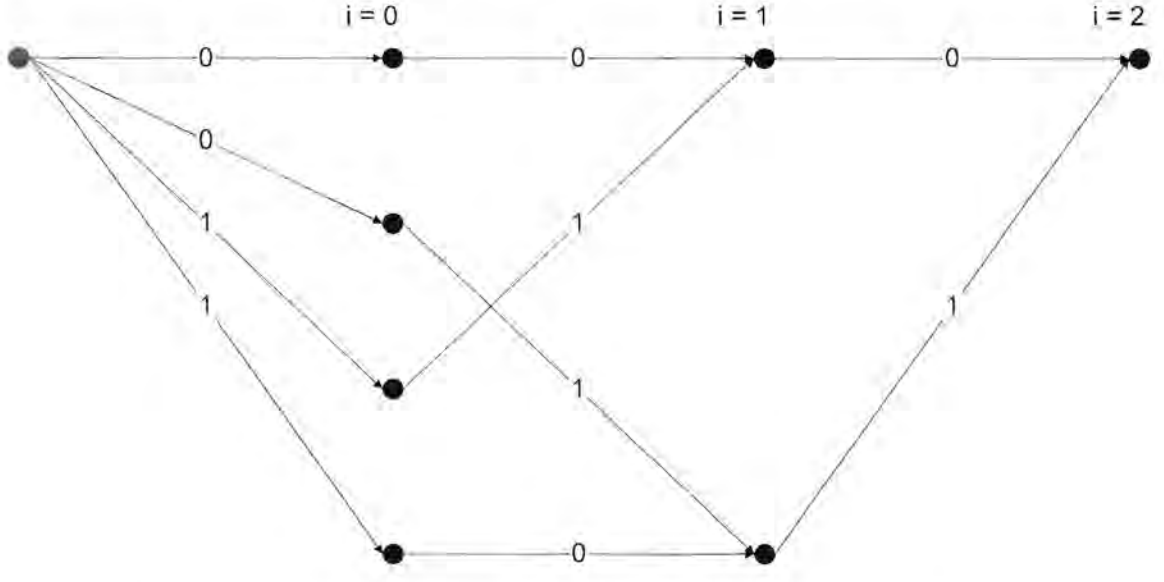

Figure 9 Untrue Trellis Diagram B for the (3,2)-Parity-Check-Code

Two code trellis diagrams are "*Isomorph*" if the one can be transformed into the other by changing some or all planes of states. Examples of isomorph trellises are shown in **Figures 10** and **11**, both of which are also valid or true code trellis diagrams.

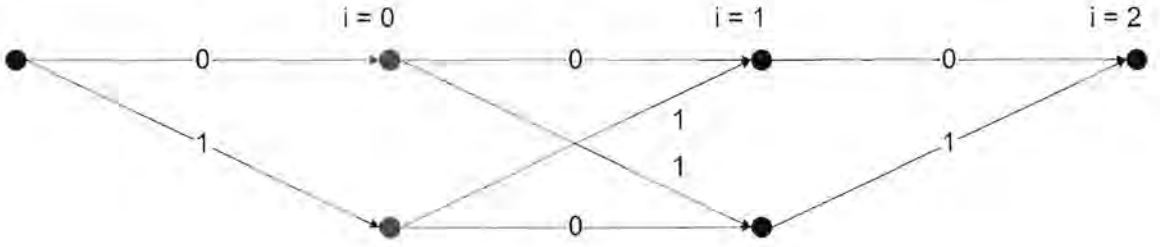

Figure 10 True Trellis Diagram C for the (3,2)-Parity-Check-Code

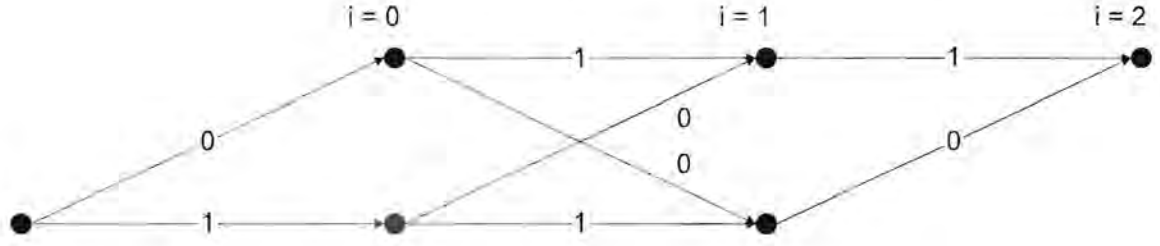

Figure 11 True Trellis Diagram B for the (3,2)-Parity-Check-Code

A code trellis is termed minimal, if for all planes of states the number of nodes in the given trellis is minimal compared to all other possible code trellis diagrams for the specific code C. Both the code trellis diagrams in **Figures 10** and **11** are minimal code trellis diagrams for the (3,2)-Parity-Check-Code. This is true, since there exists no other code trellis diagram for the same code C in which the planes $i = 0$ and $i = 1$ deliver less nodes. Furthermore, a minimal code trellis diagram for a code C is unique except for the isomorph variations in the trellis diagrams of the same code C. This means that two minimal trellis diagrams for the same code C which are not also isomorph cannot exist.

Of utmost importance for the following sections, is the fact that the Viterbi Algorithm described in **Subsection 2.4.3** can now be applied to a true code trellis diagram for the SDML decoding of block codes, without the necessity to compare the received vector with all of the possible q^k sent vectors.

3.3 Syndrome Trellis

A linear block code is specified through its parity check matrix [10] $\boldsymbol{H} = \left(h_{ij}\right) = \left(h_1, h_2, \ldots, h_n\right)$ as described in **Section 2.1.1**. **Equation 2.5** can be rewritten in the following manner:

$$\boldsymbol{c} \cdot \boldsymbol{H}^T = \boldsymbol{c}_0\boldsymbol{h}_1^T + \boldsymbol{c}_1\boldsymbol{h}_2^T + \ldots + \boldsymbol{c}_{n-1}\boldsymbol{h}_n^T = \boldsymbol{0} \tag{3.2}$$

or

$$\boldsymbol{c}_0\boldsymbol{h}_1 + \boldsymbol{c}_1\boldsymbol{h}_2 + \ldots + \boldsymbol{c}_{n-1}\boldsymbol{h}_n = \boldsymbol{0}^T \tag{3.3}$$

where $\mathbf{0}$ is the zero row vector of length n - k and $\mathbf{0}^T$ the corresponding column vector.

A state $z^T(i)$ is defined as

$$z^T(i) = \boldsymbol{c}_0\boldsymbol{h}_1 + \boldsymbol{c}_1\boldsymbol{h}_2 + \ldots + \boldsymbol{c}_i\boldsymbol{h}_{i+1} = \sum_{p=0}^{i} \boldsymbol{c}_p\boldsymbol{h}_{p+1} \tag{3.4}$$

where $z^T(i)$ (as $\mathbf{0}^T$) is a column vector with n - k components.

From this, it is possible to construct a valid code trellis diagram, termed a *syndrome trellis*, diagram defined by:

$$z^T(i-1) + \alpha \cdot h_{i+1} = z^T(i) \tag{3.5}$$

Equation 3.5 is interpreted as follows:

- At the time instance i = -1 only one state exists. This state is called the starting state $z_0^T(i = -1) = 0^T$.
- For all i = 0, 1, ..., n - 1:
  - $z^T(i)$ is exactly then a state if at time i a connecting branch enters and for any $\alpha \in$ GF(q). This means that the number of states at time i is computed by inserting all $\alpha \in$ GF(q) into all states $z^T(i$ - 1) of **Equation**

3.5.

- At the same time a connection branch is constructed from the state $z^T(i-1)$ to the corresponding state $z^T(i)$ and labeled $\alpha = c_i$, according to **Equation 3.5.**

- Finally, all nodes without a path ending in the final state $z_e^T(1=n-1)=0^T$ are removed as well as all other nodes which branch off these removed nodes. This can be seen from **Equations 3.3** and **3.4**. This expurgating or "cleaning" of the code trellis diagram is not absolutely necessary for the decoding process. The decoding can also be done on a non-expurgated trellis, but then only paths which do end in the state $z_e^T(1=n-1)=0^T$ must be considered, and the others ignored.

The syndrome trellis diagram obtained by means of the above method consists of a total of $n+1$ planes each comprising a maximum of q^{k-n} states. The q^k paths of the trellis diagram starting at state $z_0^T(i=-1)=0^T$ and ending in state $z_e^T(1=n-1)=0^T$ represent the q^k codewords.

Since the syndrome trellis diagram only contains q^k paths, another upper bound for the maximum number of states in a plane exists, since at any time i no more than q^k states can be reached by traversing q^k paths. This then defines the upper bound on the number of states Ni at time i as:

$$N_i \leq q^{\min(n-k,k)} \tag{3.6}$$

In order to illustrate the process outlined above, the syndrome trellis diagram for the (7,4)-Hamming-Code is constructed. The parity check matrix for the above mentioned code is as follows:

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \tag{3.7}$$

Beginning at the starting state

$$z_0^T(i=-1) = 0^T = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \tag{3.8}$$

two connecting branches, one for $\alpha = 0$ and one for $\alpha = 1$ respectively are constructed to the states

$$z^T(i=0) = z^T(i=-1) + 0 \cdot \boldsymbol{h}_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + 0 \cdot \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \tag{3.9}$$

and

$$z^T(i=0) = z^T(i=-1) + 1 \cdot \boldsymbol{h}_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + 1 \cdot \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}. \tag{3.10}$$

According to **Equations 3.9** and **3.10**, at time $i = 0$ the states mentioned above are possible. Which one of them actually occurs is dependant on the first code bit c_0.

At each one of these two states, two new connecting branches emanate. In order to determine the two new destination states, the following two expressions are added to $z^T(i = 0)$:

$$0 \cdot \boldsymbol{h}_2 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \qquad \text{and} \qquad 1 \cdot \boldsymbol{h}_2 = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}.$$

For each level or plane in the syndrome trellis diagram the process is repeated for each active or connected node. The correct column vector of the parity check matrix has to be used for every iteration.

The result of this iterative construction is depicted in **Figure 12**. The solid lines depict the case for which $\alpha = 0$ and the dotted lines the case $\alpha = 1$. It has to be noted that when

$\alpha = 0$, the trellis will always move from a certain state j in plane m to the same state k in plane n. This allows for fast trellis construction, when the system is implemented in hardware, as half of the connecting branches are calculated by default. This saves a tremendous amount of calculating time, which in term means more effective transmission systems.

If the trellis is expurgated as described above, a trellis diagram representation is obtained as depicted in **Figure 13**.

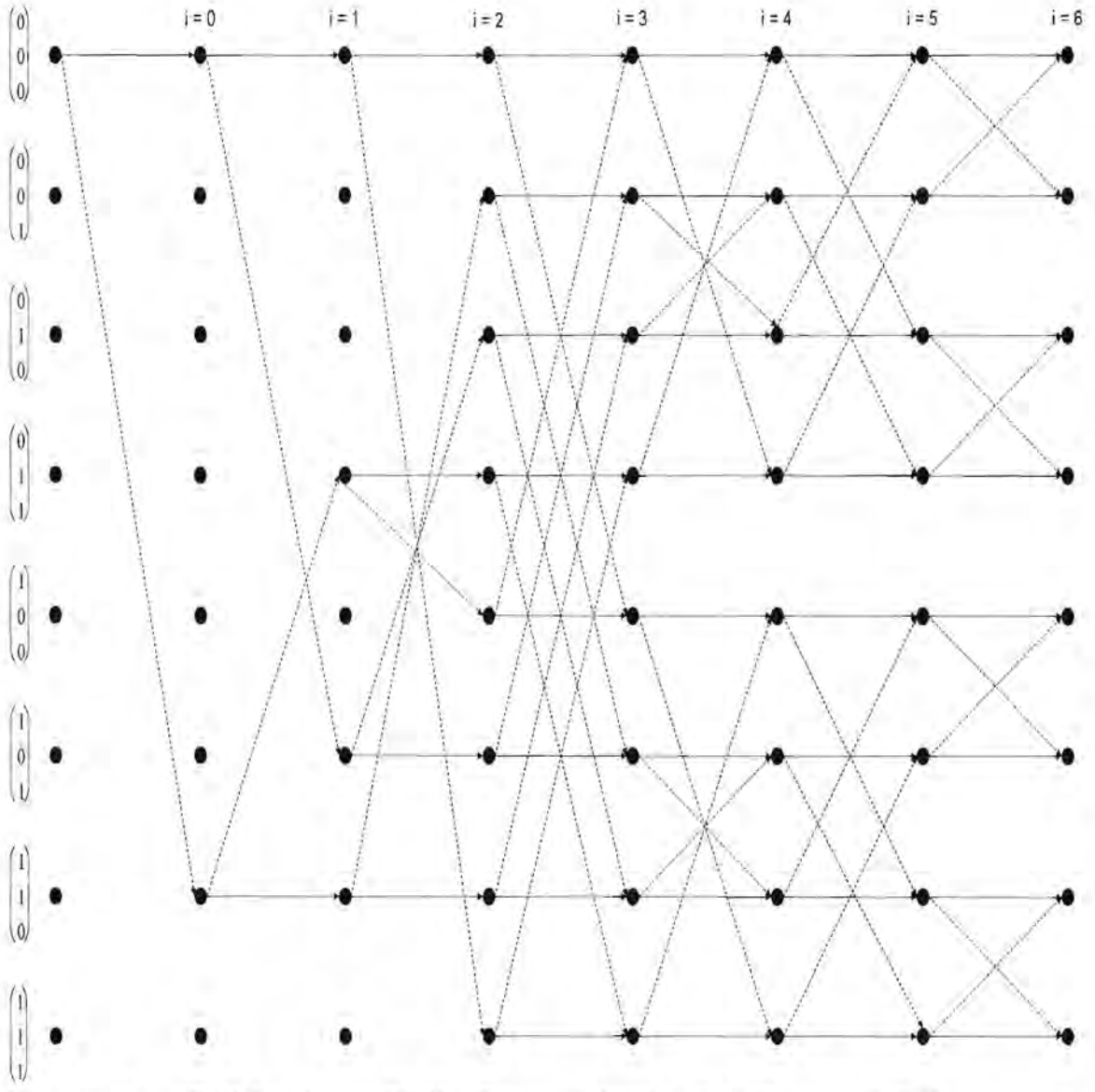

Figure 12 Full Syndrome Trellis Diagram for the (7,4)-Hamming-Code

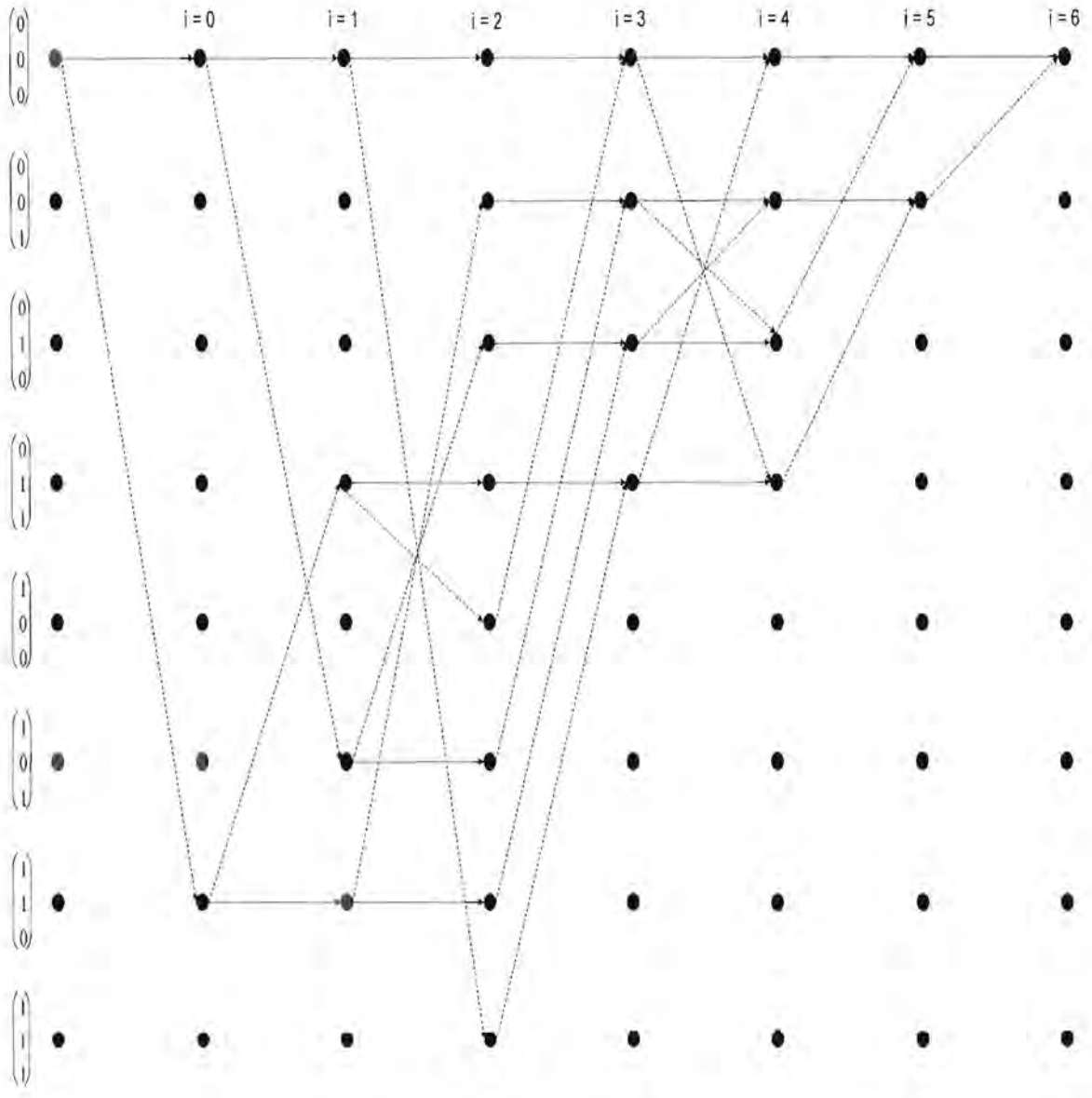

Figure 13 Expurgated Trellis Diagram for the (7,4)-Hamming-Code

As shown in **Section 3.2**, it is indeed possible to SDML decode a block code by utilizing its syndrome trellis diagram (a special case of a true code trellis diagram). The complexity of the decoding is dependant on the number of nodes in the trellis. This is governed by the following equation:

$$N_{\Sigma} = \sum_{i=-1}^{n-1} N_i \tag{3.11}$$

From the definition of the minimal code trellis diagram in **Section 3.2**, it follows that a code trellis diagram is minimal when N_{Σ} is minimal for all code trellis diagrams of C. The syndrome trellis diagram of a given linear block code is a minimal code trellis diagram. This means, that there exist no other non-isomorph code trellis diagrams with fewer nodes than the code trellis diagram obtained through the use of the syndrome construction procedure outlined above.

However, this does not hold for equivalent codes. As was shown in **Subsection 2.2.2**, an equivalent code can be obtained from a code C, by performing matrix operations on the parity check matrix of the original code C. For such an equivalent code it is very possible and also likely, that syndrome trellis diagrams can exist which have a different number of nodes. An example for such a case is the (5,3)-Code with the following parity check matrix.

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{pmatrix} \tag{3.12}$$

The syndrome trellis diagram for this code is depicted in **Figure 14**.

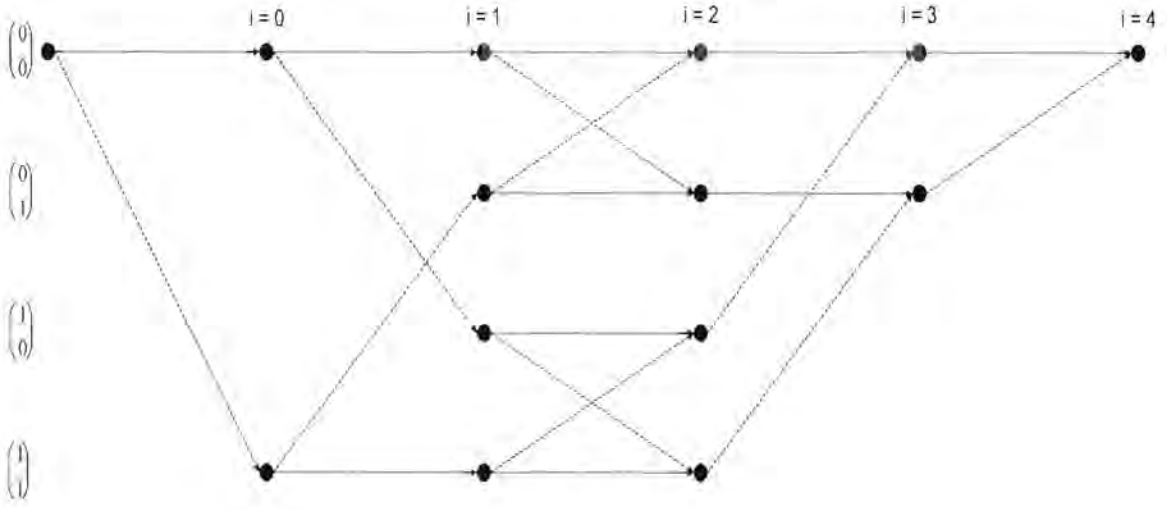

Figure 14 Syndrome Trellis Diagram A for the (5,3)-Code

If columns of the parity check matrix of **Equation 3.12** are interchanged, an equivalent code is obtained with the following parity check matrix.

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix} \tag{3.13}$$

The syndrome trellis diagram for this code is depicted in **Figure 15**.

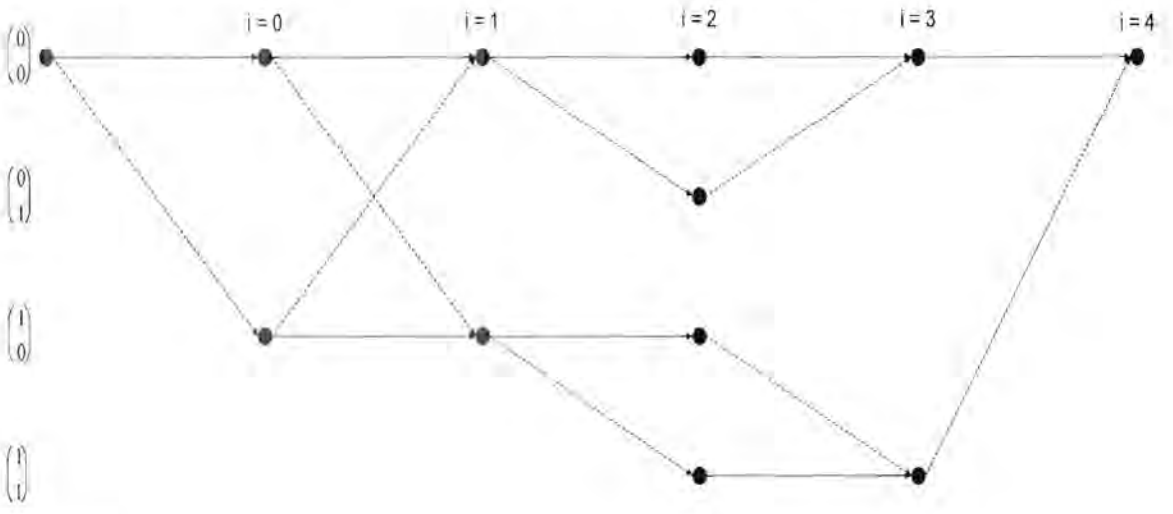

Figure 15 Syndrome Trellis Diagram B for the (5,3)-Code

As can be seen from **Figures 14** and **15** above, the number of nodes differ for the two equivalent codes. In **Figure 14**, N_{Σ} equals 14, but in **Figure 15**, N_{Σ} only equals 12. It can be seen that in both cases $q^k = 8$ paths traverse the code trellis diagrams. These 8 paths represent the $q^k = 8$ codewords. In the second diagram however, the topological distribution of these paths is more favorable than the distribution in the second diagram. The reason for this being that the reduced number of nodes imply a much simpler decoding complexity. To summarize, it can be said that the less nodes a syndrome trellis diagram contains, the simpler the eventual decoding becomes.

For cyclic codes an alternative code trellis diagram construction method exists, which differs completely from the syndrome trellis diagram construction technique described previously. It is done by using the content of the shift registers (which are used in the coding and decoding process) as states in the trellis diagram. This method will however not be considered here, as it does not produce a minimal trellis diagram. Another

method of obtaining a syndrome trellis diagram is considered in **Appendix 1**, where it is included as an example.

As mentioned in **Section 3.3**, the decoding complexity relates to the number of nodes in the code trellis diagram. It should be clear that a good trellis construction is the procedure that will result in a trellis being minimal, i.e. the trellis should have the least number of nodes N_{Σ} possible. This is indeed an immense task, since $n!$ permutations of the column vectors of the parity check matrix will have to be processed in order to find the equivalent code trellis diagram with the least amount of nodes, i.e. the minimal trellis representation. For smaller codes, this process can be done by hand, but for a large code ($n>>$), this task becomes nearly intractable.

In a following chapter, methods are investigated which reduce the number of nodes in the code trellis diagram.

3.4 Derivation of the Parity Check Matrix

In **Section 3.3** the parity check matrix **H** was assumed known for the calculation of the syndrome trellis diagram [16]. A decoder would normally only have the generator matrix **G** available for the specific code to be decoded. In order to perform the decoding, the decoder will have to determine the topological structure of the trellis diagram, for which it will need to have the parity check matrix **H**, obtainable from the generator matrix **G**.

For generator matrixes in the systematic form **G'** defined in **Equation 2.7**, it is fairly elementary to find the parity check matrix according to **Equation 2.9**. For the general non-systematic form of the generator matrix, this process is however a bit more complicated.

A method that presents itself, is the direct solving of **Equation 2.6** ($\mathbf{G}\cdot\mathbf{H}^T = \mathbf{0}$) after fixing the values of $(n - k)^2$ unknowns. This approach is not a trivial process since it is not just a matter of fixing any of the $(n - k)^2$ unknowns, but also which of the unknowns to fix. This is explained in more detail in the following outline:

- For non-systematic block codes, it is also possible to obtain a matrix representation in the form of **Equation 2.9**. The $(n - k)^2$ chosen unknowns would

be the elements of the ((n - k) × (n - k)) unit matrix in $\mathbf{H}' = (-\mathbf{A}^T|\mathbf{I}_{n-k})$. If the given generator matrix G is split up into the two sub-matrixes $\mathbf{G}_{left}$ and $\mathbf{G}_{right}$ where $\mathbf{G}_{left}$ is a (k × k) matrix and $\mathbf{G}_{right}$ a (k × (n - k)) matrix, i.e. $\mathbf{G} = [\mathbf{G}_{left} \mid \mathbf{G}_{right}]$, then **Equation 2.6** can be rewritten as

$$\boldsymbol{G} \cdot \boldsymbol{H}'^{T} = \left(\boldsymbol{G}_{left} \middle| \boldsymbol{G}_{right}\right) \cdot \begin{pmatrix} -\boldsymbol{A} \\ \hline \boldsymbol{I}_{n-k} \end{pmatrix} = -\boldsymbol{G}_{left} \cdot \boldsymbol{A} + \boldsymbol{G}_{right} = \boldsymbol{0} \tag{3.14}$$

or

$$\boldsymbol{G}_{left} \cdot \boldsymbol{A} = \boldsymbol{G}_{right} \tag{3.15}$$

The following are the n - k linear equations for the calculation of the n - k column vectors of the matrix $\boldsymbol{A} = (a_1, a_2, \ldots, a_{n-k})$:

$$\begin{aligned} \boldsymbol{G}_{left} \cdot a_1 &= g_{right,1} \\ \boldsymbol{G}_{left} \cdot a_2 &= g_{right,2} \\ &\vdots \\ \boldsymbol{G}_{left} \cdot a_{n-k} &= g_{right,n-k} \end{aligned} \tag{3.16}$$

Each of these (inhomogeneous) equations only has a solution if the rank of the extended matrix $[\mathbf{G}_{left} \mid \mathbf{G}_{right,i}], i = 1,2,\ldots,n - k$, is not larger than the rank of $\mathbf{G}_{left}$. This is however not necessarily fulfilled, since the first k columns of the matrix G, which forms the matrix $\mathbf{G}_{left}$, can be linearly dependant, which in turn could make the rank of $\mathbf{G}_{left}$ smaller than k. By extending the matrix $\mathbf{G}_{left}$ by adding one column from $\mathbf{G}_{right}$, the rank can be increased but then the equations are not uniquely solvable.

This shows that it is not possible to choose and fix any of the $(n - k)^2$ unknowns.

- A possible solution to **Equation 2.6** can be found from a matrix $\widetilde{\boldsymbol{H}}$ with n - k

identical row vectors identical. This can be any particular code vector from $C^{\perp}$. $(n - k)^2$ unknowns can be chosen and fixed in such a manner that the matrix $\widetilde{H}$ solves **Equation 2.6**. However, the matrix obtained by the procedure above, is indeed not a parity check matrix, since it has the rank 1 and not $n - k$, which does not comply with the definition of a parity check matrix given in **Section 2.2**.

Since this approach does not render a tractable solution, the parity check matrix $\widetilde{H}$ has to be calculated on the following basis:

- Firstly the generator matrix **G** is, as described in **Section 2.2**, is transformed by elementary row matrix operations so that unit vectors of length k are present in k of the n columns of the matrix.
- By swapping columns, the transformed generator matrix is changed into its systematic form **G'** given by **Equation 2.7**.
- The corresponding parity check matrix **H'** is then determined from **Equation 2.9**.
- The column swapping procedure in step two above is then applied in reverse order on **H'** to so obtain a possible parity check matrix **H**.

An example of this procedure is given in **Appendix B**.

3.5 Decoding of the Trellis Diagram

As already mentioned several times, the Viterbi algorithm outlined **Figure 6** of **Subsection 2.4.3**, can be used to theoretically SDML-decode any linear block code through its syndrome trellis. This means that for every codeword C all the steps depicted in **Figure 7** have to be performed once with the end state $z_e^T(1 = n - 1) = 0^T$ known.

However, unlike the normal Viterbi algorithm the most likely sent codeword $\hat{c}$ and not

the most likely sent information vector **i** is obtained. The output is a valid codeword that satisfies **Equation 2.5**. Therefore, all that needs to be done is to reverse the mapping between the information vector and the code vector to obtain the most probable transmitted information vector. This is done with a lookup table or by solving **Equation 2.1**. In order for the decoder to work on an arbitrary block code, of which only the generator matrix is know to the decoder initially, the second method of obtaining the information vector from the code vector is preferred.

Chapter 4

Performance Issues of Various Block Codes

4.1 Introduction

In the previous chapter, the syndrome method of constructing the trellis diagram of a block code was presented, as well as techniques to decode block codes by means of these trellis diagrams. This chapter will concentrate on performance issues relevant to this decoding method. Several different block codes are investigated, and their simulation results given. All the simulation software was developed from scratch, and was used to achieve the given simulation results.

4.2 Simulation Results

The following simulations results were obtained for both soft-decision and hard-decision block decoding. The generator matrixes that were used for the simulations are given in **Appendix C**. A simulation was done with a (7,4)-Hamming code for both soft decision and hard decision decoding in an AWGN channel. The simulation setup is the same as described in **Figure 4** of **Chapter 2** and consists of a transmitter transmitting over an AWGN channel. The achieved results for the (7,4)-Hamming code are compared to both the theoretical curves and the error curves obtained for the traditional analytical decoding methods. The results prove that the same (maximum likelihood) performance is obtained with the Viterbi algorithm than with conventional algebraic methods. This holds for both soft decisions as well as hard decisions. As all the curves, including the bounds, cluster closely together, this representation (i.e. all curves on one graph) is not used in subsequent simulation results in this dissertation. It is clear form the results in the example (**Figure 16**) that the results fall well within the limits defined by the BER bounds for the decoding methods under discussion. The following equation was used in determining the upper bound for the decoding.

$$P_E \leq \sum_{m=\left\lfloor \frac{d_{\min}-1}{2} \right\rfloor}^{n} \binom{n}{m} p^m (1-p)^{n-m} \qquad (4.1)$$

The following two graphs provide the results achieved for the (7,4)-Hamming code.

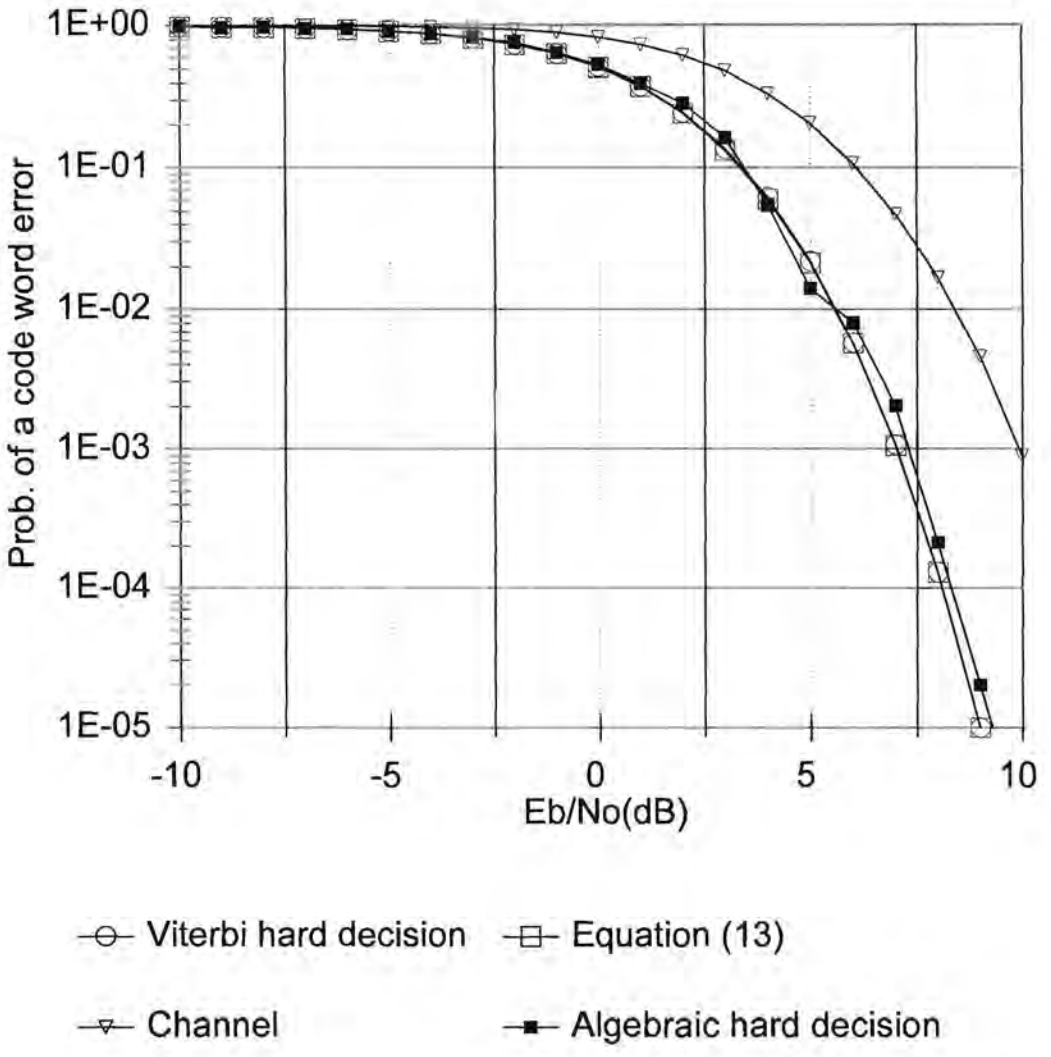

Figure 16 BER Comparison of Various Hard Decision Viterbi Decoding Methods of the (7,4) Hamming Block Code

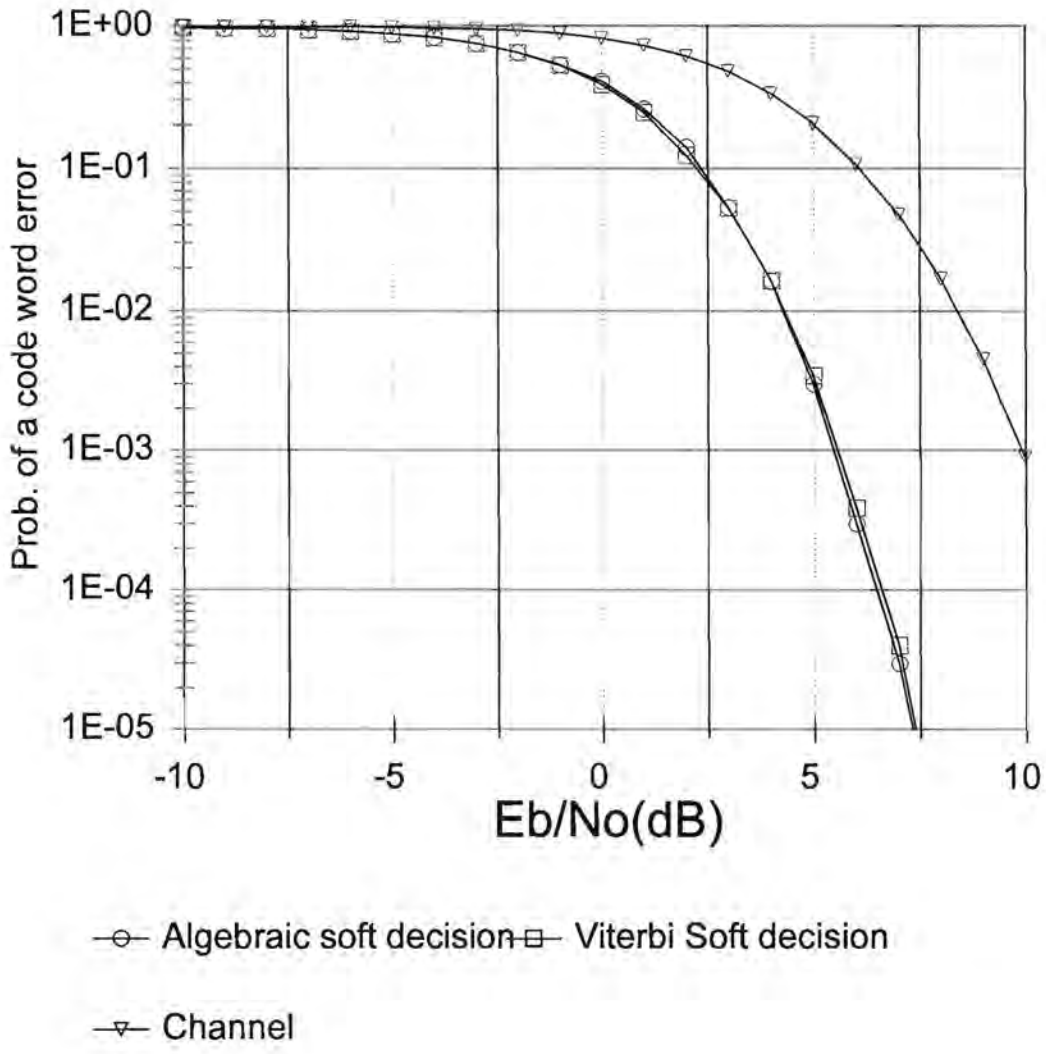

Figure 17 BER Comparison of Various Soft Decision Viterbi Decoding Methods of the (7,4) Hamming Block Code

On the following BER graphs curves for various families of codes are depicted. For the reasons explained above, the error rate curves will not be compared to the algebraic methods, since these curves lie on top of each other. In the figures, a graph termed *Bipolar Reference* is however included. This curve represents the BER performance of the uncoded case under identical channel conditions.

4.2.1 Bit Error Rate Performance of the(7,4)-Hamming-Code with Viterbi Trellis Decoding

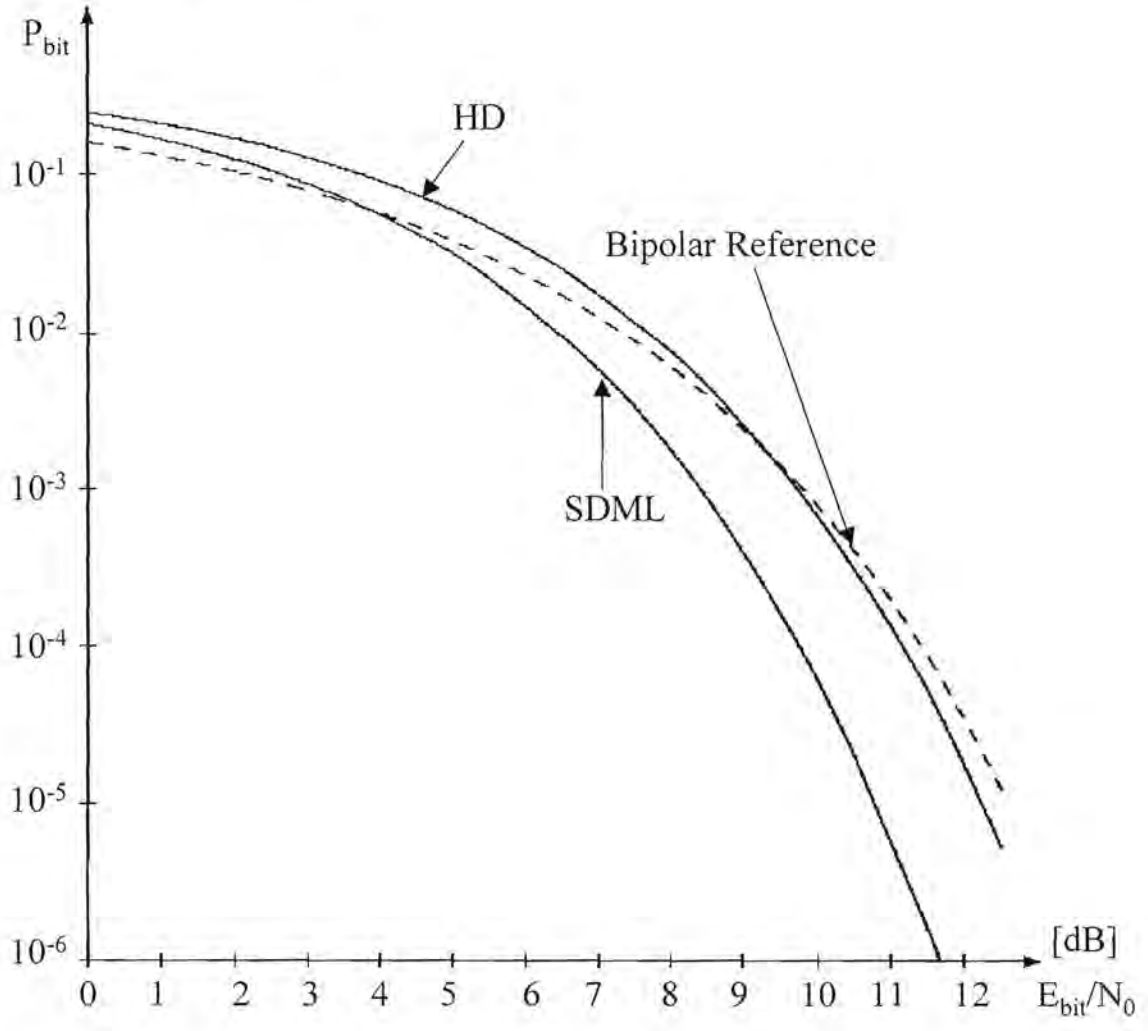

Figure 18 Bit Error Rates of the (7,4)-Hamming-Code in an AWGN Channel

Figure 18 presents the BER performance of a (7,4)-Hamming-Code. As can be seen from the figure, SDML-decoding becomes viable for channels having and E_{bit}/N_0 ratio of 4 dB and higher. Below this cross over value (3.5 dB in this case), SDML-decoding fairs no better than the normal uncoded transmission. The HD-decoding only crosses the uncoded graph at 9.2 dB. It is also only marginally better than the uncoded curve for E_{bit}/N_0 larger than 9.2 dB. At $P_{bit} = 10^{-5}$ the difference between SDML and HD-decoding is 1.4 dB. This corresponds with the calculated [10] value for the (7,4)-Code.

4.2.2 Block Error Rate Performance of the (7,4)-Hamming-Code with Viterbi Trellis Decoding

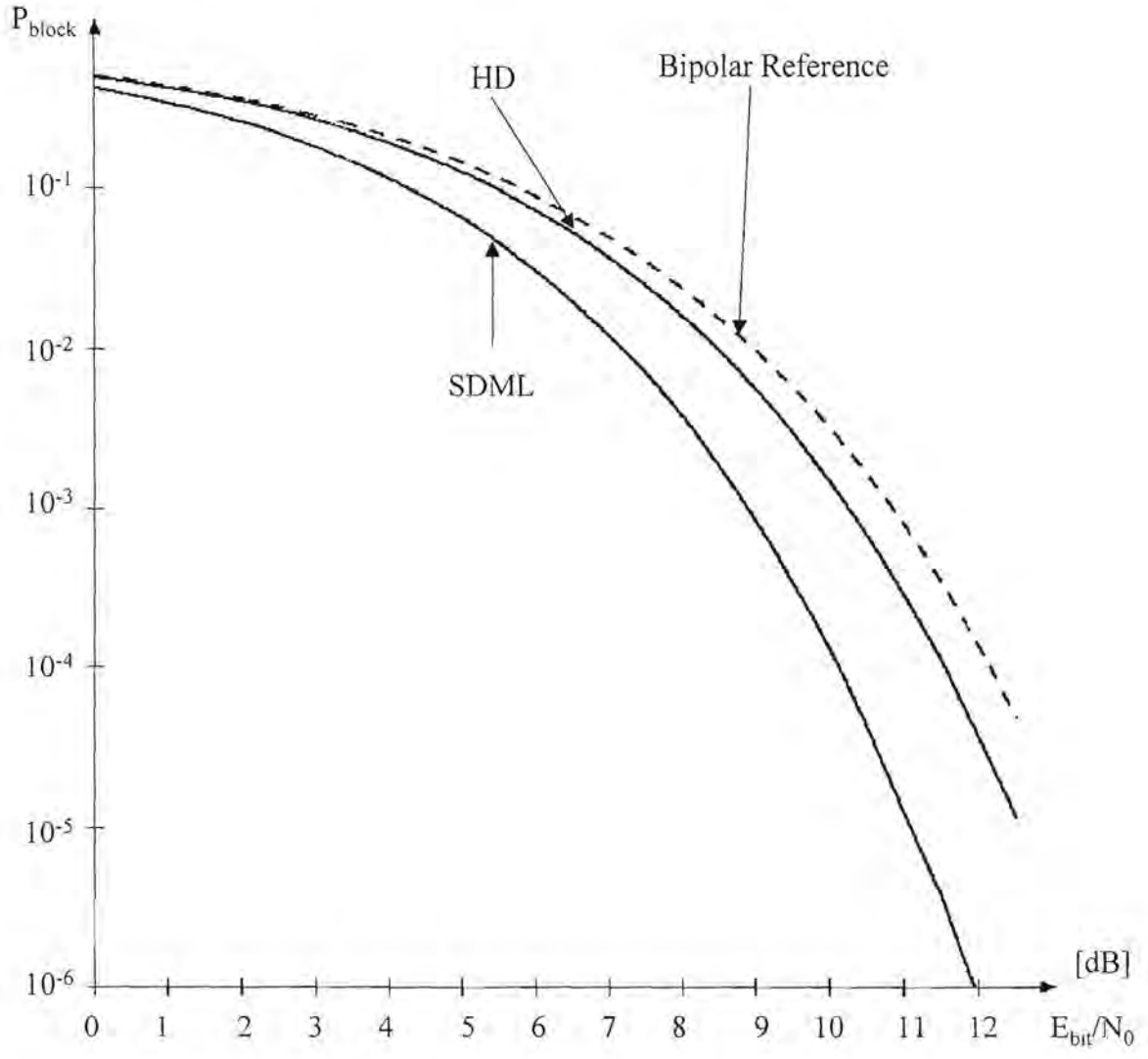

Figure 19 Block Error Rate of the (7,4)-Hamming-Code in an AWGN Channel

In **Figure 19** above, the block error probability of the (7,4)-Hamming-Code is given. The difference between the BER and the block error rate is as follows: Whereas several bit errors may only result in one block error, the BER counts all bit errors. The latter are statistically dependant on a few parameters such as the burst error statistics of the given channel. For instance, a codeword of length 7 bits is decoded incorrectly. This constitutes only one block error, but any number of bits (1 to 7) may be incorrect.

4.2.3 Bit Error Rate Performance of the (16,11)-Reed-Muller-Code with Viterbi Trellis Decoding

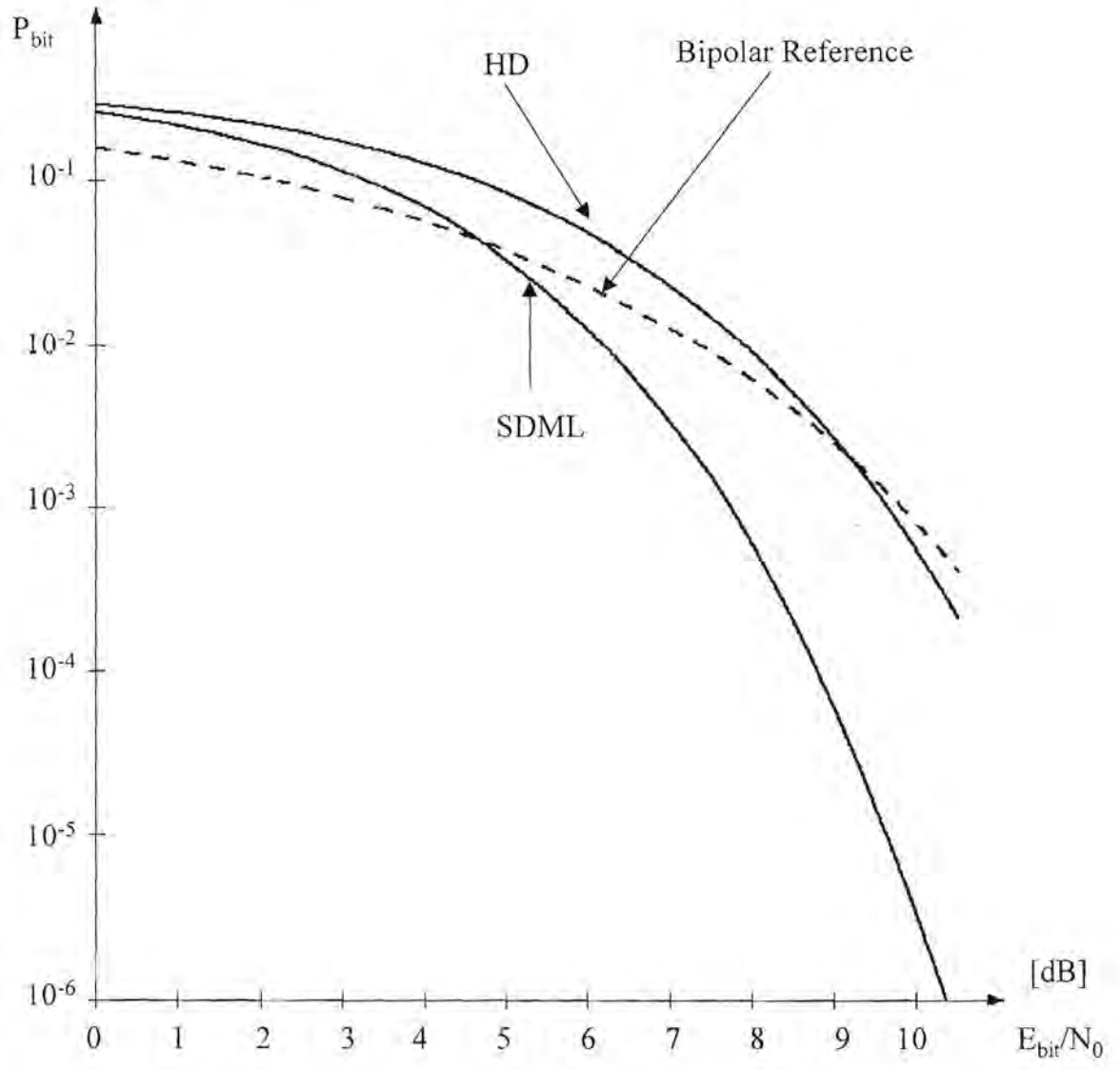

Figure 20 Bit Error Rate of the (16,11)-Reed-Muller-Code in an AWGN Channel

The above curve is very similar to the curve for the (7,4)-Hamming-Code depicted in **Figure 18**. The difference between the two codes is as follows: To achieve a bit error rate of 10^{-6} with the Reed-Muller-Code, an E_{bit}/N_0 ratio of only 10.3 dB is required, whereas the Hamming-Code requires an E_{bit}/N_0 ratio of 11.7 dB. The Reed-Muller code has a 1.4 dB coding advantage on the (7,4)-Hamming code in this particular example.

4.2.4 Bit Error Rate Performance of the (31,21)-BCH-Code with Viterbi Trellis Decoding

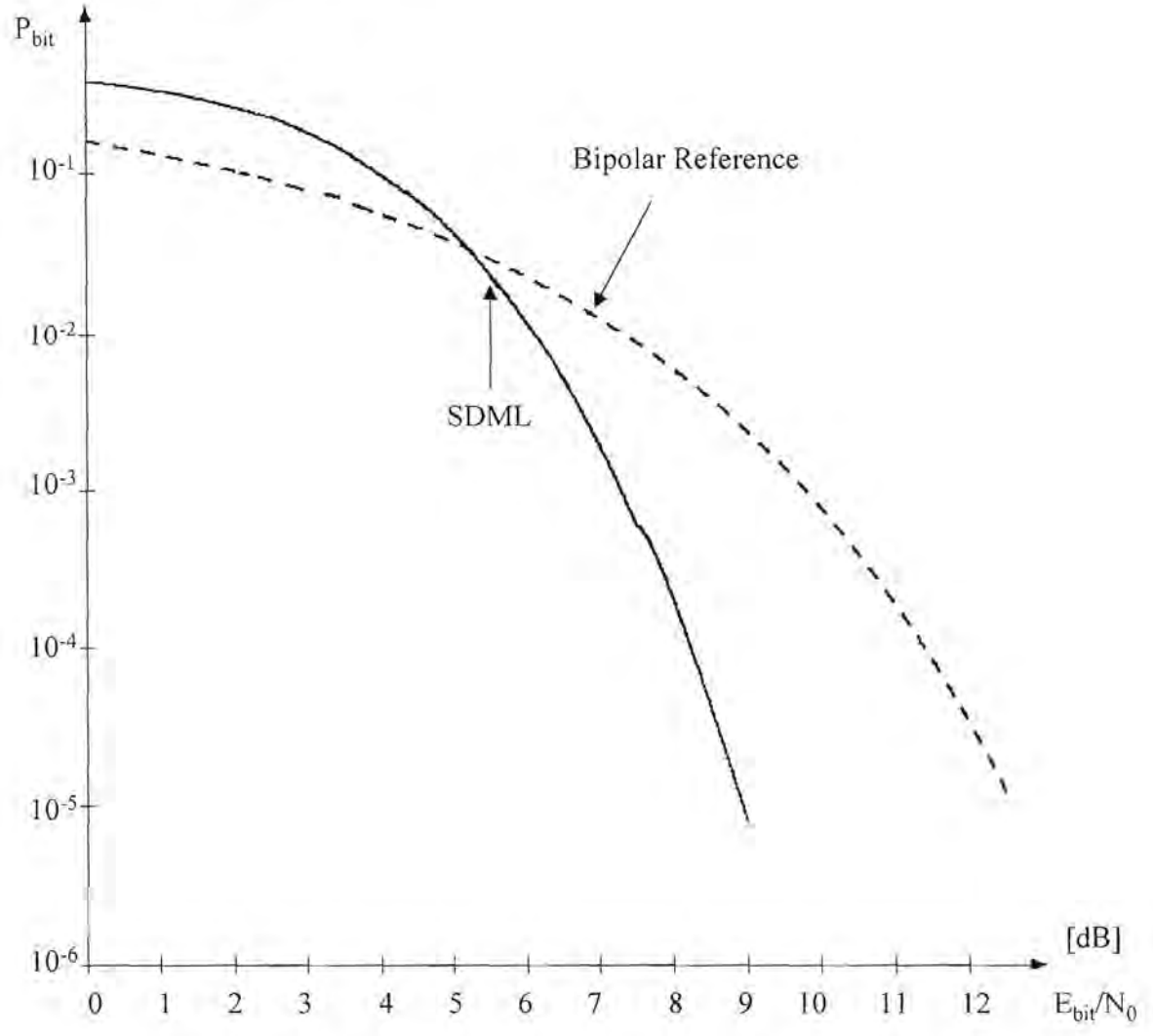

Figure 21 Bit Error Rate of the (31,21)-BCH-Code in an AWGN Channel

The above figure displays the Bit Error Rate of a (31,21)-BCH-Code. It can be seen that this code renders good performance at high values of E_{bit}/N_0. A Bit Error Rate of 10^{-5} is achieved at only 9 dB. In **Figure 22** another BCH-code is displayed, and in **Figure 23**, a comparison between both the BCH-codes and the (7,4)-Hamming-Code is made.

4.2.5 Bit Error Rate Performance of the (15,7)-BCH-Code with Viterbi Trellis Decoding

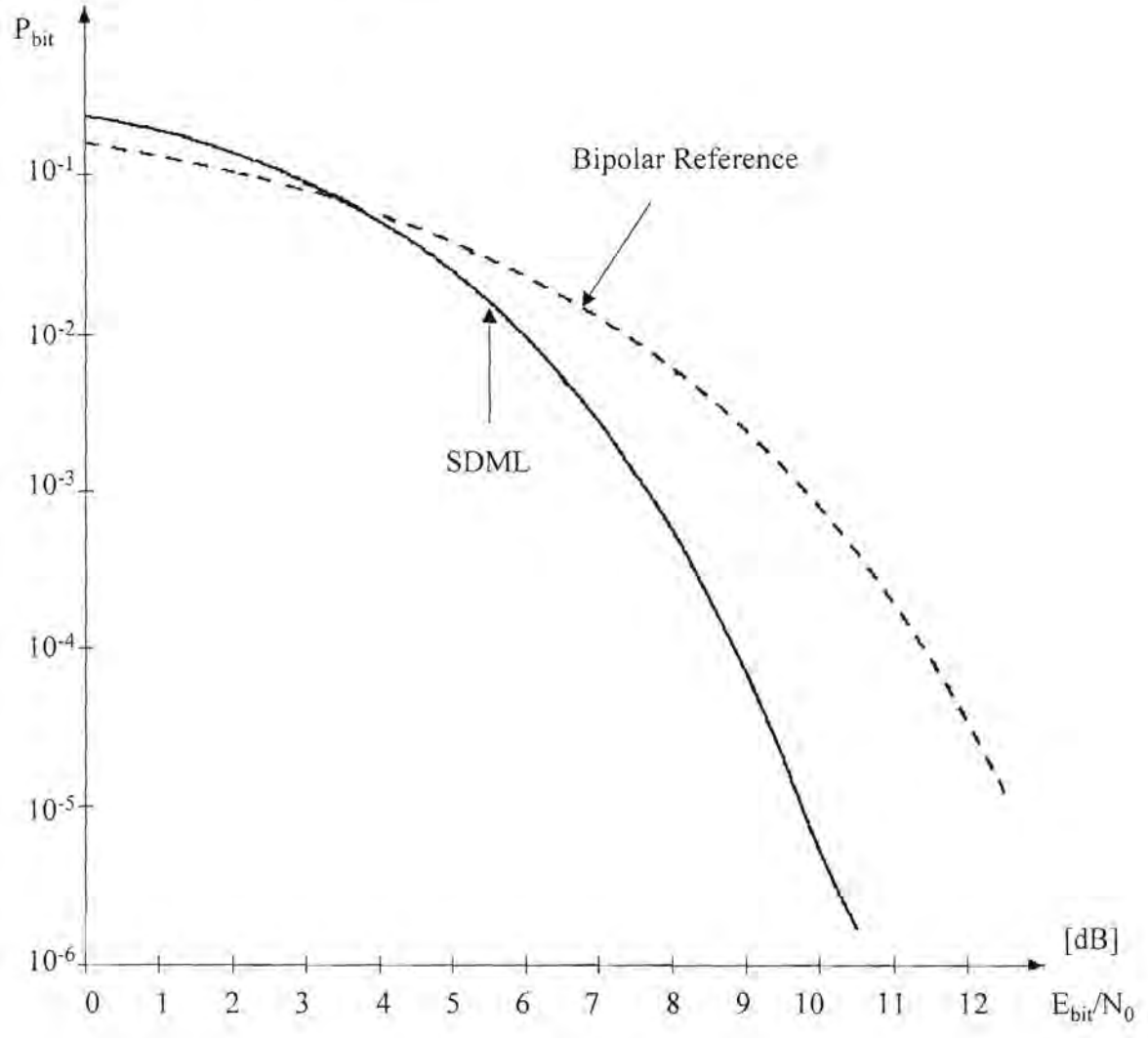

Figure 22 Bit Error Rate of the (15,7)-BCH-Code in an AWGN Channel

In **Figure 22**, the Bit Error Rate of a (15,7)-BCH-Code is displayed. Again, it can be seen that the code provides a relatively good performance at high values of E_{bit}/N_0. To gain some insight into the relative performance of the block codes considered, the previous two figures are superimposed on the (7,4)-Hamming-Code. The result is shown in **Figure 23**.

4.2.6 Performance Comparison of BCH-Codes with (7,4)-Hamming-Code with Viterbi Trellis Decoding

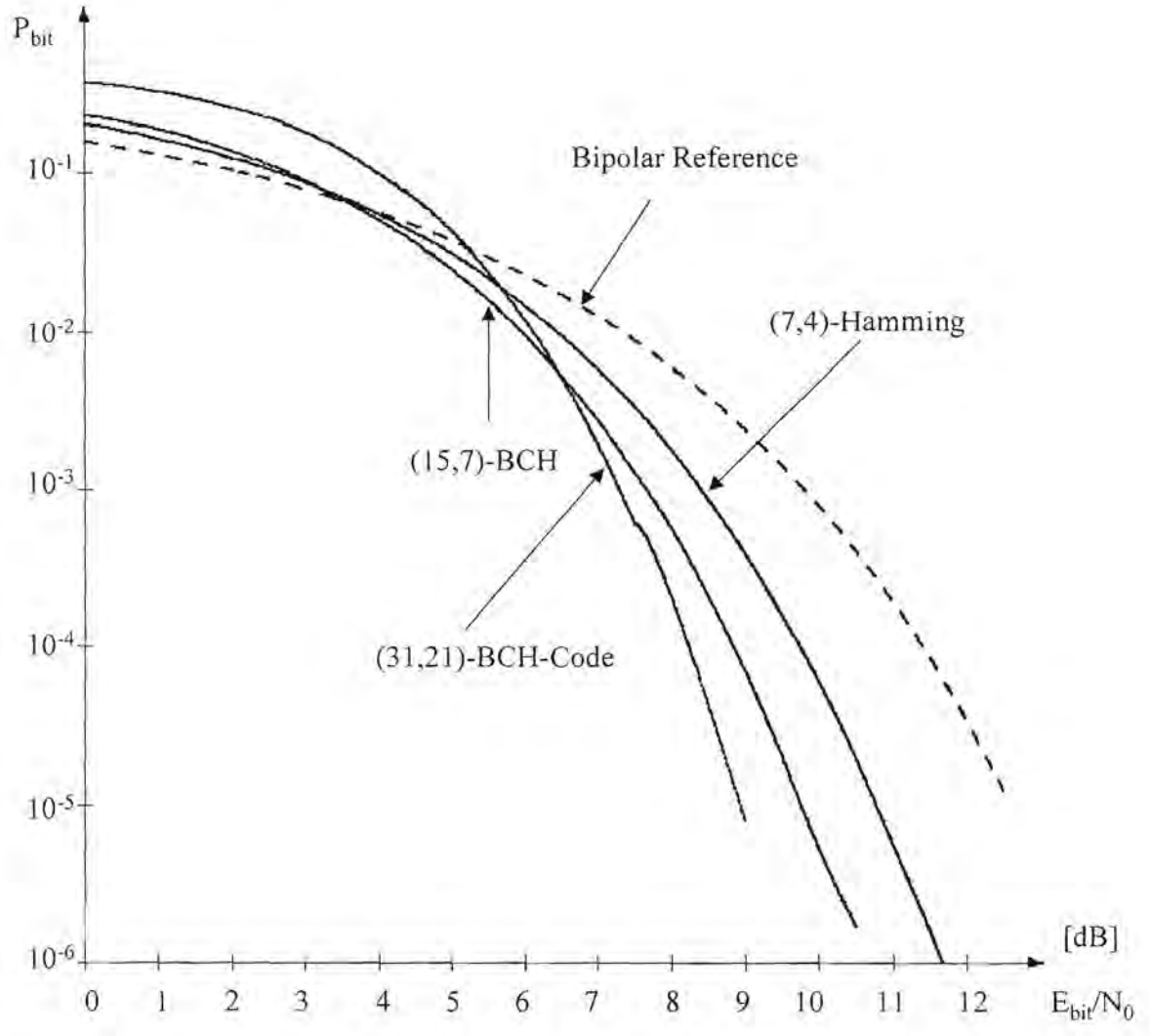

Figure 23 Comparison of BCH-Codes with (7,4)-Hamming-Code in an AWGN Channel

From **Figure 23**, it can be seen that all of the codes perform worse than the uncoded bipolar reference system at low values of E_{bit}/N_0. The cross over point, where the codes begin to perform according to their theoretical specification, is at about 5.5 dB. From this point onward, the larger codes progressively yield better performance as the block length increases. It should be noted that a number of the factors have to be considered when deciding on a code for a particular channel.

4.2.7 Bit Error Rate Performance of the (16,5)-Reed-Muller-Code *R*(1,4) with Viterbi Trellis Decoding

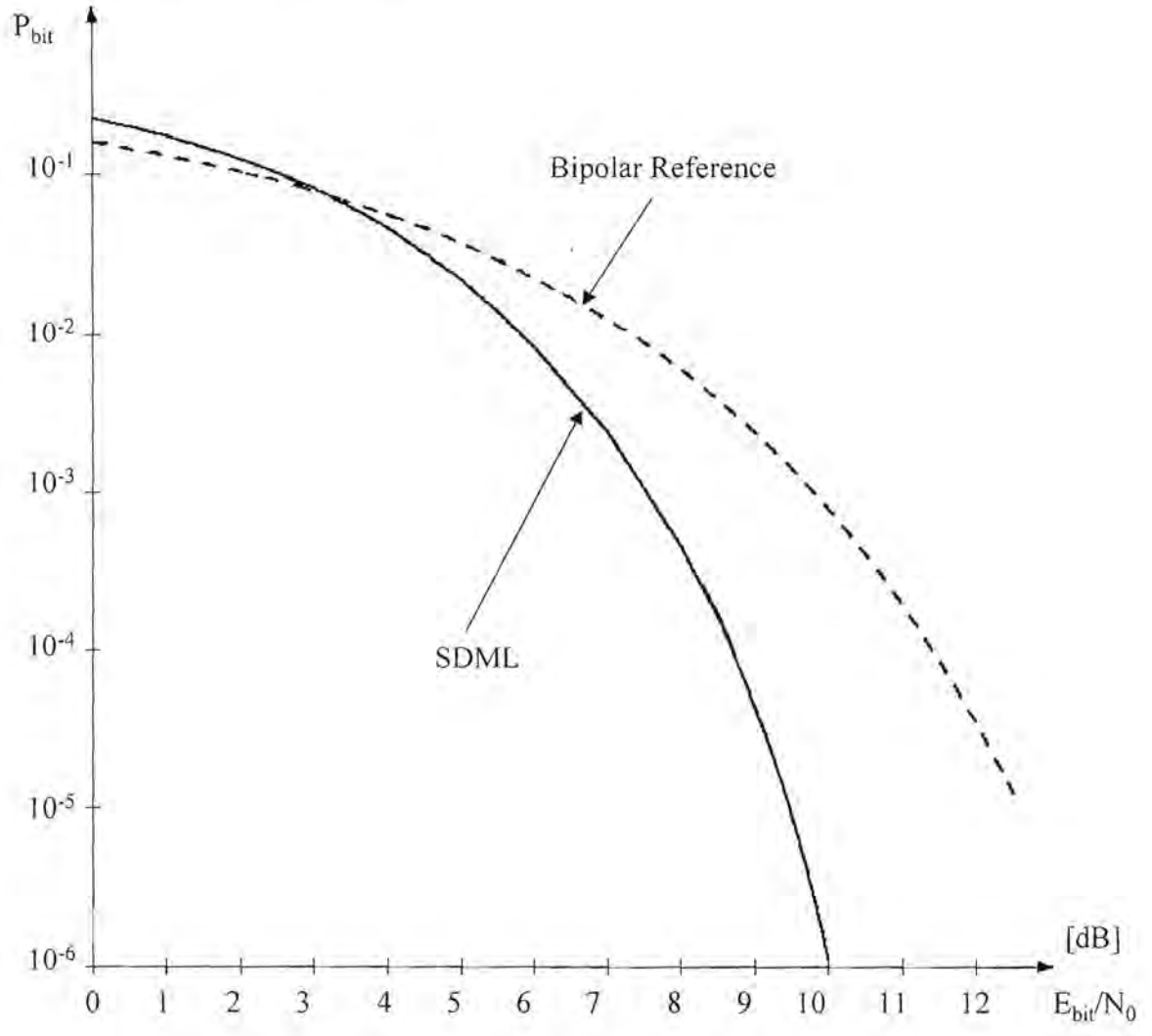

Figure 24 Bit Error Rate of the (16,5)-Reed-Muller-Code $R(1,4)$ in an AWGN Channel

4.2.8 Bit Error Rate Performance of the (16,11)-Reed-Muller-Code *R*(2,4) with Viterbi Trellis Decoding

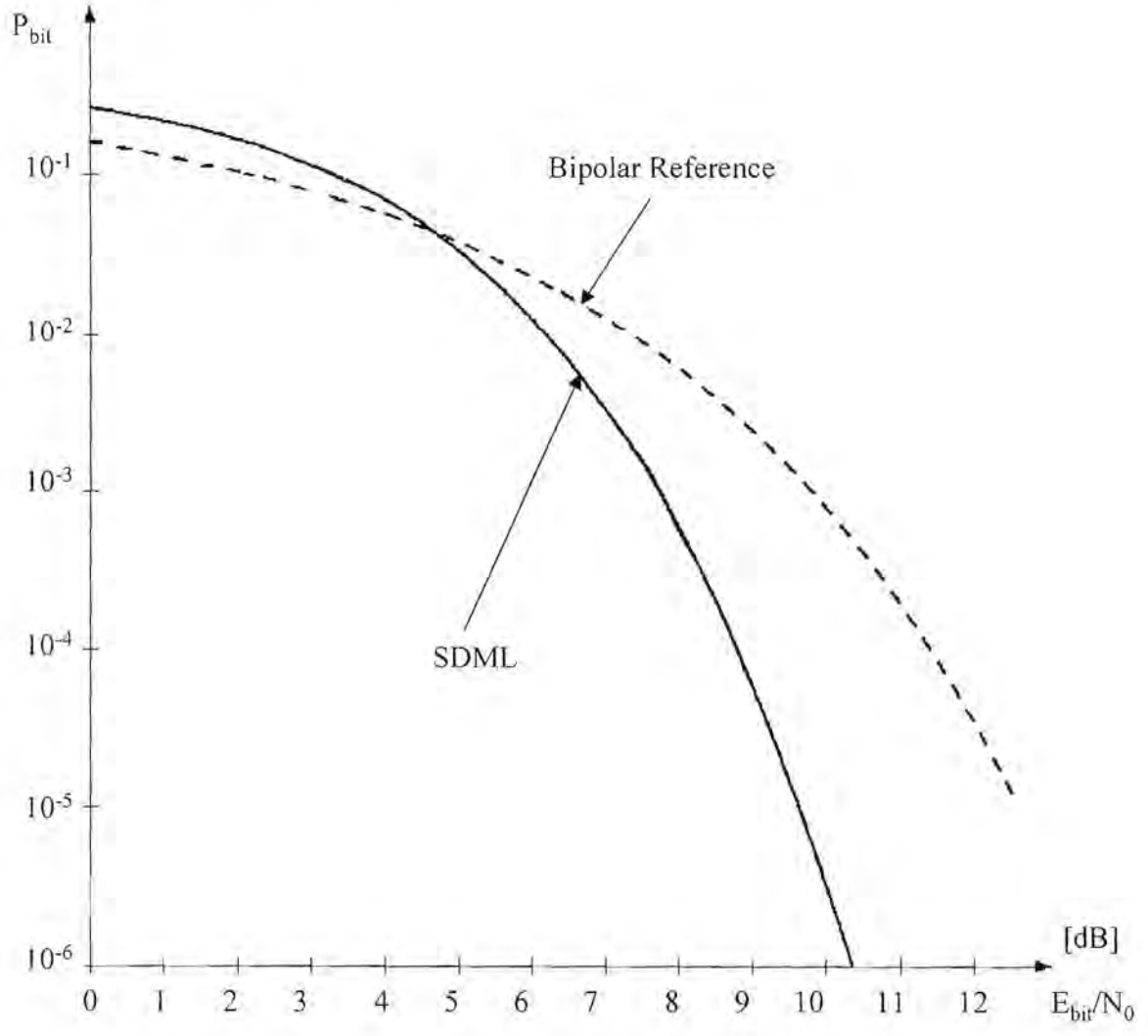

Figure 25 Bit Error Rate of the (16,11)-Reed-Muller-Code $R(2,4)$ in an AWGN Channel

4.2.9 Bit Error Rate Performance of the (32,16)-Reed-Muller-Code *R*(2,5) with Viterbi Trellis Decoding

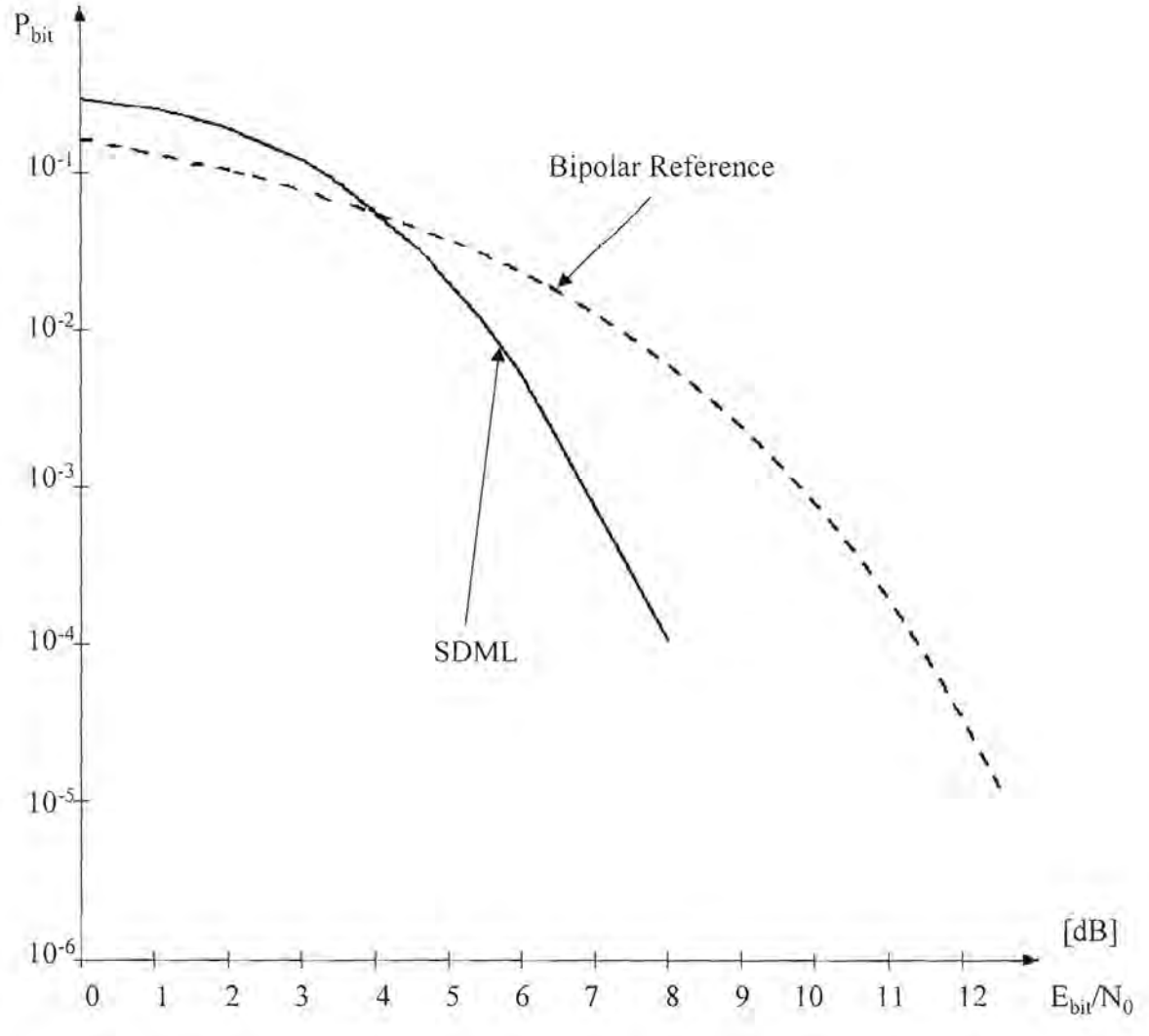

Figure 26 Bit Error Rate of the (32,16)-Reed-Muller-Code *R*(2,5) in an AWGN Channel

4.2.10 Bit Error Rate Performance of the (32,26)-Reed-Muller-Code *R*(3,5) with Viterbi Trellis Decoding

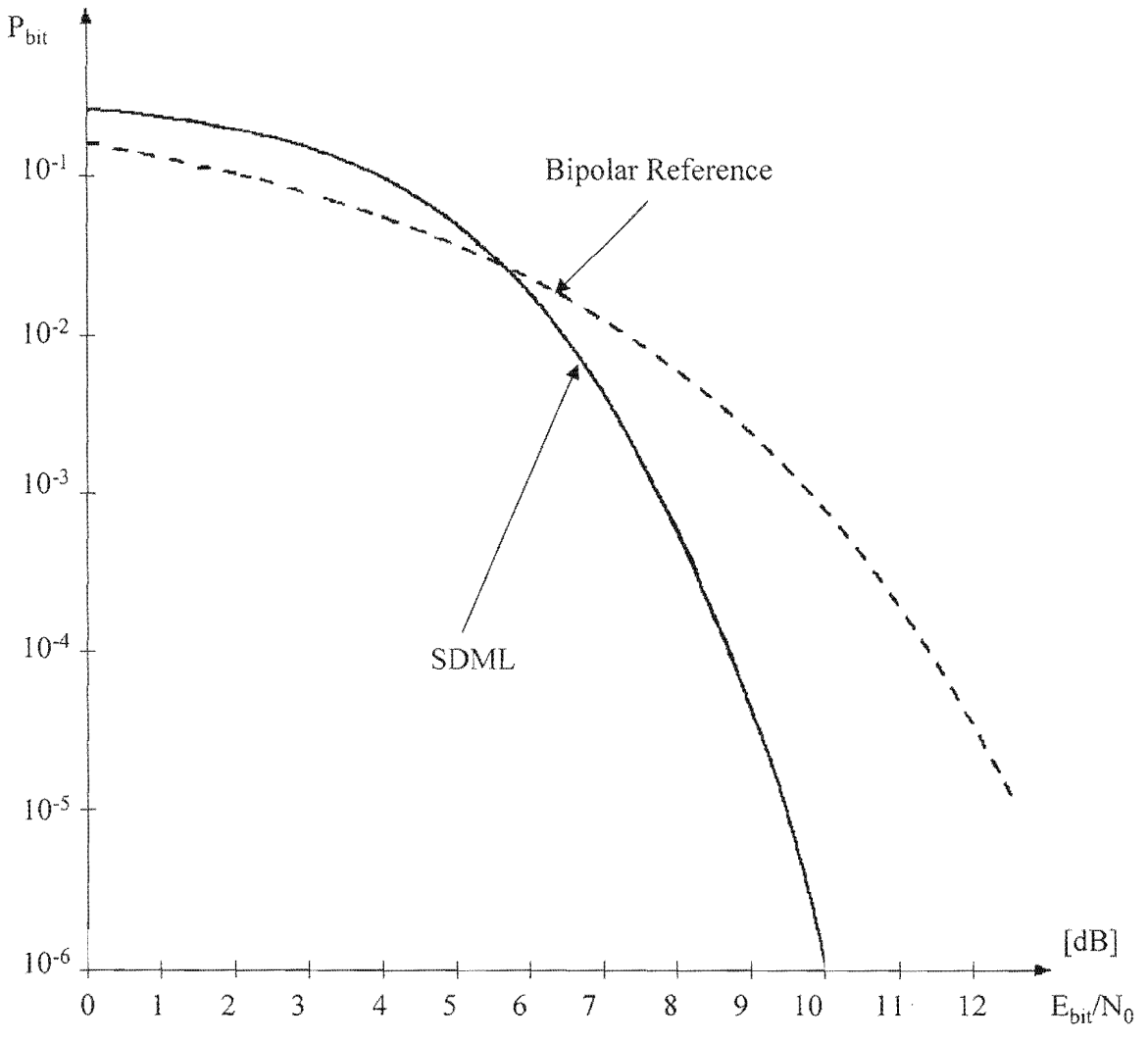

Figure 27 Bit Error Rate of the (32,26)-Reed-Muller-Code *R*(3,5) in an AWGN Channle

4.2.11 Bit Error Rate Performance of the (32,31)-Reed-Muller-Code *R*(4,5) with Viterbi Trellis Decoding

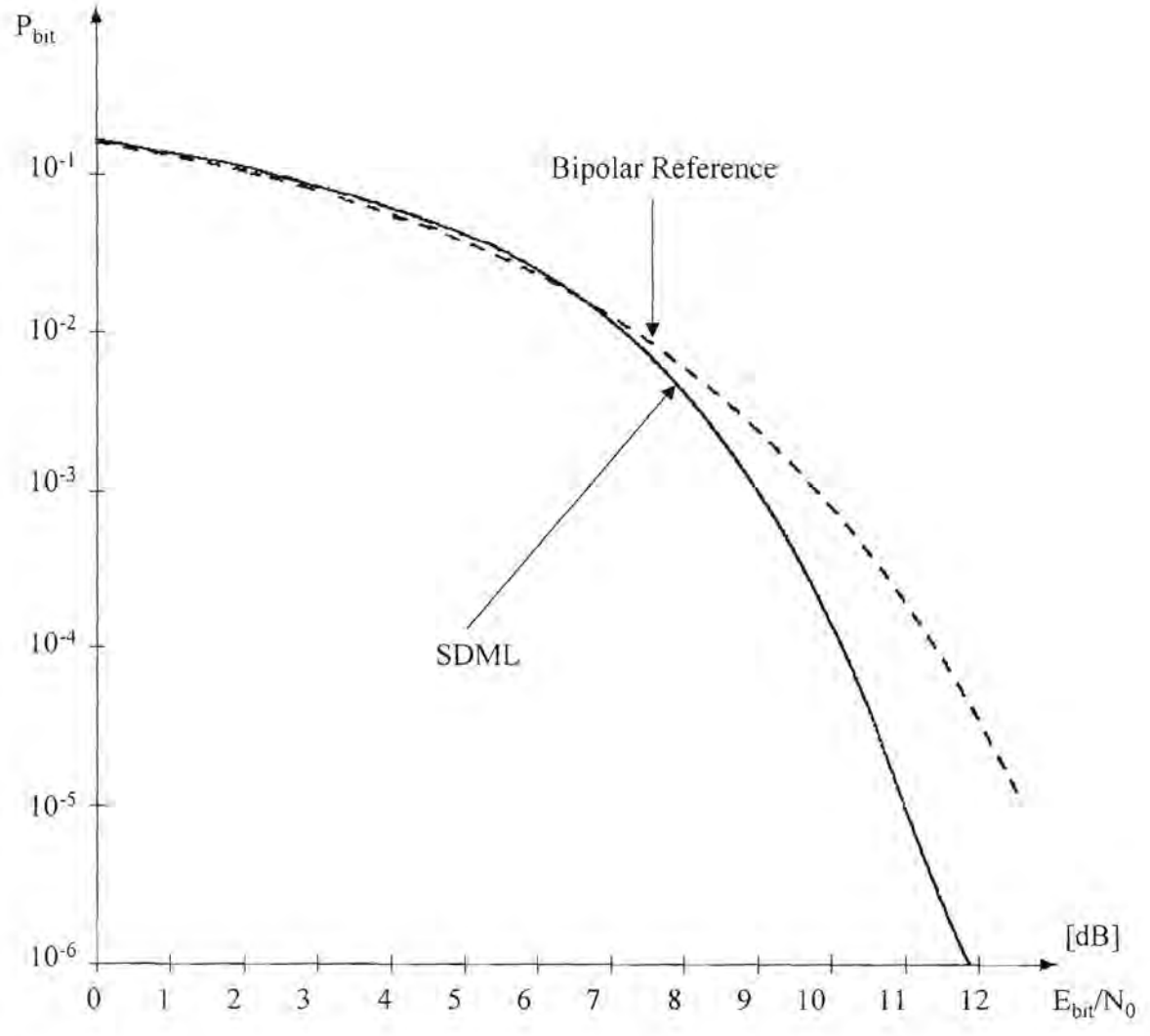

Figure 28 Bit Error Rate of the (32,31)-Reed-Muller-Code *R*(4,5) in an AWGN Channel

4.2.12 Performance Comparison of Reed-Muller-Codes *R(x,y)* for different (*n,k*) with Viterbi Trellis Decoding

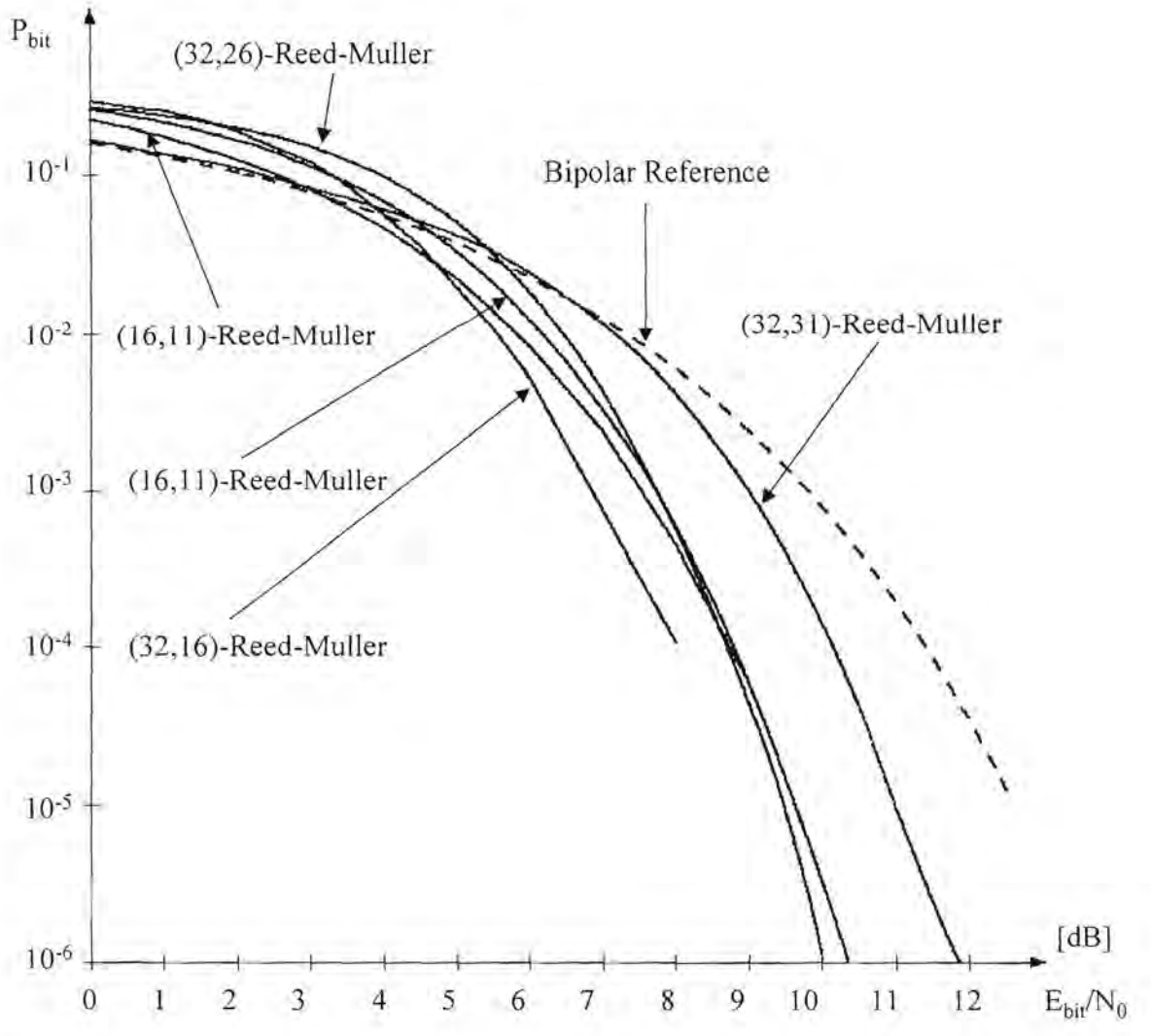

Figure 29 Comparison of different Reed-Muller-Codes

The above figure compares all the different simulation results which were obtained for Reed-Muller codes. What is apparent from the above results, is that the performance achieved is not only dependent on the size of the code, but also on the parameters (*n,k*). This is illustrated by the fact that the (32,31)-Reed-Muller-Code performs a lot worse than the (16,11)-Reed-Muller-Code. Again, the choice of the block code is a very critical step in the design of a error correcting code.

4.2.13 Tabulation of Results

The results achieved by simulation are presented in the table below.

Code	Cross Over HD	Cross Over SDML	Asymptotic Gain HD	Asymtotic Gain SDML
(7,4)-Hamming	9.2 dB	4.0 dB	0.75 dB	2.0 dB
(16,11)-Reed-Muller	9 dB	4.5 dB	1.0 dB	2.5 dB
(16,5)-Reed-Muller	-	3.5 dB	-	4.0 dB
(32,16)-Reed-Muller	-	4.0 dB	-	3.5 dB
(32,26)-Reed-Muller	-	5.9 dB	-	4.0 dB
(32,31)-Reed-Muller	-	7.0 dB	-	1.5 dB
(31,21)-BCH	-	5.1 dB	-	3.5 dB
(15,7)-BCH	-	3.5 dB	-	3.0 dB

Table 1 Tabulation of Results achieved with Viterbi Trellis Decoding

4.2.14 General Discussion of Results

In the above simulations it was shown that it is indeed possible to employ the Viterbi algorithm to decode block codes. Furthermore, it is proven that all block codes which are decoded via their trellis diagram representations conform to the term maximum likelihood decoding. This means that the bit error rate curves that are obtained for the Viterbi decoding of block codes matches the bit error rate curves obtained by analytical decoding - which is a maximum likelihood technique. No coding gain is thus lost when block codes are decoded employing their trellis diagrams. Put differently, it can be said that no sacrifice or penalty is paid in transforming a block code into its respective trellis representation. It is also shown that the simulation results are supported by the theoretical analyses which were presented for reference purposes.

Chapter 5

Trellis Construction and Decoding of Non-Binary Reed-Solomon Codes

5.1 Introduction to Reed Solomon Codes

On the 21st of January 1959 Irvine Reed and Gus Solomon [21] submitted a dull sounding title for a paper to the "Journal of the Society for Industrial and Applied Mathematics"[20]. The paper was accepted and published in June of 1960 under the title "Polynomial Codes over Certain Finite Fields". Little did the unsuspecting world know at that stage what dramatic impact this paper would have on the coding ideology known to man. The enormous contribution that this class of error correcting codes, appropriately called Reed-Solomon Codes[22], has made to the communications sphere, cannot be described in this short paragraph. Reed-Solomon codes have proved their value in channels with a predominantly bursty error characteristic.

5.2 Algebraic Reed-Solomon Code Generation

The class of Reed-Solomon [1], [10],[18],[19],[20,[21],[22] codes are constructed and decoded through the use of finite field arithmetic. These fields are sometimes also referred to as Galois Fields [24][25], after their discoverer. They too have a very wide application in modern communication systems. In order to provide the reader with a more complete set of information, **Appendix F** gives a brief overview of finite fields. The reader may find it necessary to consult this appendix first before proceeding with the following chapter, since none of the information given in **Appendix F** is repeated here.

The original approach to the construction of the Reed-Solomon codes over finite fields was very simple indeed. Assume that k information symbols

$$\left[m_0, m_1, \ldots, m_{k-2}, m_{k-1}\right] \tag{5.1}$$

exist over the finite field GF(q). From these information symbols, the following polynomial can be constructed

$$P(x) = m_0 + m_1 \cdot x + \ldots + m_{k-2} \cdot x^{k-2} + m_{k-1} \cdot x^{k-1} \tag{5.2}$$

As with any code, Reed-Solomon codes have a limited set of code words for every specific Reed-Solomon code. These code words are found by evaluating the polynomial of **Equation 5.2** at each of th q elements of the finite Galois field GF(q). The code words will thus be in the following form

$$c = \left(c_0, c_1, c_2, \ldots, c_{q-1}\right) = \left(P(0), P(\alpha), P\left(\alpha^2\right), \ldots, P\left(\alpha^{q-1}\right)\right) \tag{5.3}$$

The full set of code words can be formed by allowing the k information symbols to assume all possible combinations of values over GF(q). The information symbols are taken from GF(q), resulting in each being able to assume q different values. There is no need to explain why a Reed-Solomon code has a huge set of possible code words. A Reed-Solomon code will have q^k unique code words. From the discussion and definitions given in Chapter 3, it follows that a Reed-Solomon code is linear in nature.

It is quite common to call the parameter k of the Reed-Solomon code the dimension of the code, since the value k forms a vector space of dimension k over the Galois field GF(q). The length of a Reed-Solomon code is commonly termed n in accordance with block code conventions.

Due to the polynomial nature of Reed-Solomon codes, it is possible to revert each Reed-Solomon code to a system of q linear equations with k parameters each. A generic system is given in the equation below:

$$\begin{aligned}
&P(0) = m_0 \\
&P(\alpha) = m_0 + m_1 \cdot \alpha + m_2 \cdot \alpha^2 + \ldots + m_{k-1} \cdot \alpha^{k-1} \\
&P\left(\alpha^2\right) = m_0 + m_1 \cdot \alpha^2 + m_2 \cdot \alpha^4 + \ldots + m_{k-1} \cdot \alpha^{2(k-1)} \\
&\vdots \\
&P\left(\alpha^{q-1}\right) = m_0 + m_1 \cdot \alpha^{q-1} + m_2 \cdot \alpha^{2(q-1)} + \ldots + m_{k-1} \cdot \alpha^{(k-1)(q-1)}
\end{aligned} \tag{5.4}$$

From normal algebra principles, it follows that any k of these expressions can be used in order to construct a system of k equations with k variables. As an example, the first

k expressions of **Equation 5.4** form the following system:

$$\begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & \alpha & \alpha^2 & \dots & \alpha^{(k-1)} \\ 1 & \alpha^2 & \alpha^{42} & \dots & \alpha^{2\cdot(k-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{k-1} & \alpha^{2\cdot(k-1)} & \dots & \alpha^{(k-1)\cdot(k-1)} \end{pmatrix} \cdot \begin{pmatrix} m_0 \\ m_1 \\ m_2 \\ \vdots \\ m_{k-1} \end{pmatrix} = \begin{pmatrix} P(0) \\ P(\alpha) \\ P(\alpha^2) \\ \vdots \\ P(\alpha^{k-1}) \end{pmatrix} \tag{5.5}$$

It can be shown that this system has a unique solution for the k information symbols by computing the determinant of the following coefficient matrix:

$$\begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & \alpha & \alpha^2 & \dots & \alpha^{(k-1)} \\ 1 & \alpha^2 & \alpha^{42} & \dots & \alpha^{2\cdot(k-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{k-1} & \alpha^{2\cdot(k-1)} & \dots & \alpha^{(k-1)\cdot(k-1)} \end{pmatrix} \tag{5.6}$$

It can be shown that the above matrix can be reduced to a Vandermonde matrix, and that all Vandermonde matrixes are non-singular. From this, it follows that any k of the expressions in **Equation 5.4** can be used to solve the system.

In the modern construction of Reed-Solomon codes, a construction technique is employed in which a generator matrix approach is used. It is however not necessary at this point to go into the detail of this construction method, as only an overview of Reed-Solomon codes is provided here.

5.3 Conventional Decoding Methods for Reed-Solomon Codes

After the discovery of Reed-Solomon codes, a search for an efficient decoding algorithm commenced. None of the standard decoding techniques at the time were useful for this purpose, for example, simple codes can be decoded through the use of a syndrome look-

up table. This approach is however out of the question for the standard Reed-Solomon codes. A (65, 53, 10) Reed-Solomon code, capable of correcting 5 errors, will have to have a look-up table of 10^{20} symbols. No hardware would be able to handle such immense data sizes.

In their 1960 paper, Gus Reed and Irvine Solomon proposed a decoding algorithm based on the solution of sets of simultaneous equations as described above in **Section 5.2** [60]. Although more useful than the look-up table approach, this techniques only allows for the decoding of very small Reed-Solomon codes. During the 1960's a large amount of work was done towards finding effective decoding algorithms for Reed-Solomon codes. Some of the contributors were people like Peterson, Chien and Forney [26],[27],[28],[29],[30]. Although several techniques were devised, no major improvement on the earlier decoding techniques were found.

The eventual breakthrough came late in 1967 when Berlekamp [32] devised an effective decoding algorithm for nonbinary BCH and Reed-Solomon codes. Following the good work of Berlekamp, Massey [21] showed in 1968 that the decoding problem is equivalent to the generation of linear feedback shift registers which generate the Reed-Solomon codes. The original Berlekamp decoding technique was adapted by Massey to conform to his theory of decoding by linear feedback shift registers. This decoding algorithm is suitably termed the Berlekamp-Massey [21] decoding algorithm.

A lot of different decoding procedure exist, but these will not be described or mentioned here, as this would serve no purpose at all.

The "Holy Grail" of Reed-Solomon decoding research is the maximum likelihood soft decision decoder. A soft decision decoder accepts analog values directly from the channel. The demodulator is not forced to decide which of the q possible symbols a given signal is supposed to represent. The decoder is thus able to make decisions based on the quality of a received signal. For example, the decoder is more willing to assume that a noisy value represents an incorrect symbol than a clean, noise-free signal. All of the information on the "noisiness" of a particular received signal is lost when the demodulator assigns a symbol to the signal prior to decoding.

A lot of work has been done on soft decision maximum likelihood decoders for Reed-Solomon codes, but as to yet, no practical solution has been found. This dissertation

attempts to provide a possible solution for more effective SDML decoding for Reed-Solomon codes.

5.4 Error Correcting Characteristics of Reed-Solomon Codes

The Reed-Solomon code's error correcting capability and performance is now presented. For the purpose of this discussion, it is assumed that t of the code word coordinates are corrupted during transmission, and are received incorrectly. This state of affairs would lead to an incorrect representation in the system of **Equation 5.4**, and would lead to an incorrect solution if **Equation 5.5** is solved. Assuming that there is no knowledge of where the errors are, all possible constructions of systems from **Equation 5.4** are made. There are in fact $\binom{q}{k}$ such systems, $\binom{t+k-1}{k}$ of which produce, when solved, incorrect information symbols. Correct information bits are received as long as the following holds true:

$$\binom{t+k-1}{k} < \binom{q-t}{k} \tag{5.7}$$

This condition only applies when $t + k - 1 < q - t$, which in itself only holds true if the following condition is met $2t < q - k + 1$. A Reed-Solomon code of length q and dimension k can correct up to t errors, where t is given as:

$$t = \left(\frac{q-k+1}{2} \right) \tag{5.8}$$

It was proved in 1964 by Singleton [31] that the family of Reed-Solomon codes provides the best error correction capability for any code with the same length and dimension.

5.5 Various Trellis Construction Techniques

Trellis construction techniques for Reed-Solomon codes can basically be divided into two groups [21],[22],[33],[34]. The first group is termed "Syndrome Trellises of Reed-Solomon Codes" and the second group "Coset Trellises of Reed Solomon Codes". Both these methods aim at obtaining a trellis diagram which would be considered as being minimal. In the following subsections, a brief overview on both these methods is presented. Another method also exists. This method is called the "Shannon Product of Trellises". Due to the way in which the dissertation is organized, this method is presented in Appendix D, since it is referred to by several chapters. It is advisable that the reader familiarize himself with above mentioned procedure, as it is essential to the understanding of the discussion below.

5.5.1 Syndrome Trellis Design for Reed-Solomon Codes

Wolf's method of trellis construction was already presented in **Chapter 3**. This method of trellis construction is under investigation in more detail in **Section 5.6**, and is not pursued further.

A method which is very similar to Wolf's [16] method is presented here as another form of syndrome trellis construction.

A few years back, McElice [19] has proven that trellises found via syndrome techniques are minimal. According to a bound calculated by Wolf [16], the maximum number of states in a syndrome trellis can be estimated as:

$$N_{synd}^{\max} \leq \min\left\{q^{k}, q^{(n-k)}\right\} \tag{5.9}$$

Similar to this, Forney has shown that the minimum number of states at the i-th level of the trellis is equal to:

$$N_i = \frac{q^k}{q_{past}^k \cdot q_{future}^k} \quad i = 1,2,\ldots,n \tag{5.10}$$

where

$$\begin{aligned} k_{past} &= \dim\left(C_{past}\right) \\ k_{future} &= \dim\left(C_{future}\right) \\ C_{past} &= (i, k_{past}, d) \\ C_{future} &= (i, k_{future}, d) \end{aligned} \tag{5.11}$$

As shown before, it is necessary to find the syndrome trellis in a certain Galois field. To be as general as possible, the determination of the trellis diagram is done in GF(q). Firstly, a few definitions are in order at this stage.

The vector of information symbols is defined as

$$X = \left(x_1, x_2, \ldots, x_k\right) \qquad x_i \in GF(q) \tag{5.12}$$

and the encoded vector as

$$Y = \left(y_1, y_2, \ldots, y_n\right) \qquad y_i \in GF(q) \tag{5.13}$$

with $i = 1,2,\ldots,k$ and $j = 1,2,\ldots,n$.

Let **G** be the generator matrix of the Reed-Solomon code in the cyclic form. (see **Chapter2**):

$$\boldsymbol{G} = \begin{pmatrix} g_1 \\ \vdots \\ g_k \end{pmatrix} = \begin{pmatrix} g_1 \\ cs_1(g_1) \\ cs_2(g_2) \\ \vdots \\ cs_{k-1}(g_k) \end{pmatrix} \tag{5.14}$$

In the above equation, $g_i \quad i = 1,2,\ldots,k$ is the i-th row of the generator matrix **G**, and $cs_j(g_1)$ denotes j cyclic shifts of g_1.

The desired Reed-Solomon code can be constructed as a sum of k codes:

$$C = \sum_{j=1}^{k} C_j \tag{5.15}$$

where the j-th code C_j is an (n,1,d) code (k = 1) over GF(q) generated by $\mathbf{G}_j=[g_j]$.

It is relatively easy to show that q code words in C_j can be obtained as:

$$C_j = x_j\left[g_j\right] \tag{5.16}$$

The corresponding sub-trellises T_j, where $j=1,2,...,k$, start at the generic node called the root node, and terminate in the finishing node called the destination node. These sub-trellises have $(n+1)$ vertices and the number of states in the i^{th} vertice is defined as being:

$$\begin{aligned} N_0 &= N_n = 1 \\ N_t &= \begin{cases} q & \text{if } g_j^i \neq 0 \text{ and } g_j^{i+1} \neq 0 \\ 1 & \text{for all other cases} \end{cases} \end{aligned} \tag{5.17}$$

where $t = 0,1,2,3,...,n$ and represents the t^{th} element of g_j.

If T_j is a syndrome trellis of the elementary code C_j generated by g_i then the combined trellis can be obtained from $T = T_1 \cdot T_2 \cdot \ldots \cdot T_k$. In other words, the syndrome trellis of the code C generated by G is given as T.

The state profile of the syndrome trellis can be obtained from:

$$\begin{aligned} N_0 &= N_n = 1 \\ N_t &= q^m \quad t = 1,2,\ldots,n-1 \end{aligned} \tag{5.18}$$

where m is the number of non-zero elements in the t^{th} column of **G** which are followed by any other non-zero element.

The above process is illustrated by means of an example in **Appendix E**.

5.5.2 Coset Trellis Design for Reed-Solomon Codes

A coset trellis represents a set of parallel sub-trellises [5], each corresponding to one of the cosets of the basic code. Such a trellis allows a reduction in the decoder complexity, since all the sub-trellises have identical structure and differ only in the labeling of their respective trellis branches. Again, the reader is referred to **Appendix D**, since this method also involves Shannon's "Product of Trellises" method. Below, it is now shown how the Shannon product of trellises can be successfully employed to find a minimal coset trellis representation of a Reed-Solomon code.

The first step in obtaining the coset trellis representation, is the calculation of the state profile from the minimal syndrome trellis representation of the Reed-Solomon code. This profile can be obtained by calculating the minimal number of states at every node of the trellis.

From the above calculated profile, N_{synd} splitting points are chosen which have a similar number of states. The next step is defining the state and label size profiles of the desirable trellis:

$$N_{\text{coset}} = \left[1, N_1, N_2, \ldots, N_c - 1, 1\right] \tag{5.19}$$

$$l_{\text{coset}} = \left[l_1, l_2, \ldots, l_{N_c-1}\right] \tag{5.20}$$

where N_c is a number of columns (vertices) in the desired coset trellis, and all vertices have a similar number of states:

$$N_i = N_j \qquad i, j = 1,2,\ldots,N_c - 1 \tag{5.21}$$

It is apparent, that in the general case the following applies:

$$l_i \neq l_j \qquad i, j = 1,2,\ldots,N_c - 1 \tag{5.22}$$

Finally, the overall generator matrix of the complete code can be presented in the following form:

$$\boldsymbol{G} = \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_k \end{pmatrix} = \begin{pmatrix} \boldsymbol{G}_1 & \boldsymbol{G}_2 & \cdots & \boldsymbol{G}_{N_c-1} \end{pmatrix} \tag{5.23}$$

where $\mathbf{G}_i$, with $i = 1,2,...,N_c - 1$, has l columns and k rows. Each row of **G** is used to design the trellis diagram of the (n,l,d) code over GF(q) with the label size profile given by **Equation 5.20**. The overall trellis diagram can be obtained as the Shannon product of k constituent trellises.

Again, as mentioned before, examples to illustrate the process are given in the second part of **Appendix E**.

5.5.3 Modified Trellis Design Procedure for Reed-Solomon Codes

This next method is basically a variation on Wolf's method. The basic procedure outlined in **Chapter 3** still applies to the trellis design procedure for Reed-Solomon codes. It is however essential to understand all the implications of using a higher order Galois field than GF(2).

In order to provide a very clear example of the procedure and also due to the large complexity, a very simple example is given, namely a (7,5) RS-code in GF(4). Since it was decided to keep the Galois field set as small as possible (not counting the binary case),a Galois field of GF(4) was chosen. In order to have a full understanding on this issue, the reader is referred back to the previous sections of this chapter, in which the basics of Reed-Solomon codes and Galois fields are presented.

The following definitions are in order here.

The Galois field that was chosen is:

$$GF(2^2) = GF(4) \tag{5.24}$$

The number of states in the trellis are:

$$N = (2^{GF(2^2)})^{n-k} = (2^2)^2 = (4)^2 = 16 \tag{5.25}$$

The following generator polynomial was chosen for the example:

$$\begin{aligned} g(x) &= (x+\alpha)\cdot(x+\alpha^2) \\ &= (x^2 + \alpha\cdot x + \alpha^2\cdot x + \alpha^3) \\ &= (1\cdot x^2 + (\alpha^2+\alpha)\cdot x^1 + 1\cdot x^0) \\ &= (1\cdot x^2 + 1\cdot x^1 + 1\cdot x^0) \end{aligned} \tag{5.26}$$

The following generator matrix can be constructed from the above given generator polynomial. Note that the generator matrix is in cyclic form, as each row is just a shifted replica of the generator polynomial coefficients.

$$\boldsymbol{G} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \tag{5.27}$$

From this generator matrix, the following information is obtained:

The trellis depth equals:

$$T_d = Num(\boldsymbol{G}_{columns}) = n = 7 \tag{5.28}$$

The number of possible code words are:

$$N_{CW} = (GF(2^2))^n = (2^2)^7 = 4^7 = 16384 \tag{5.29}$$

The number of branches emanating from each node is determined by the number of symbols in the code. Thus, for the binary case, the number of symbols would be 2 and the number of associated branches also 2. However, for a code in GF(4), there are 4 branches associated with the following 4 symbols:

$$Symbols = \{0,\ \ 1,\ \ \alpha,\ \ \alpha^2\} \tag{5.30}$$

At this point it can already be said that each node will have the following branch structure depicted in **Figure 30** below:

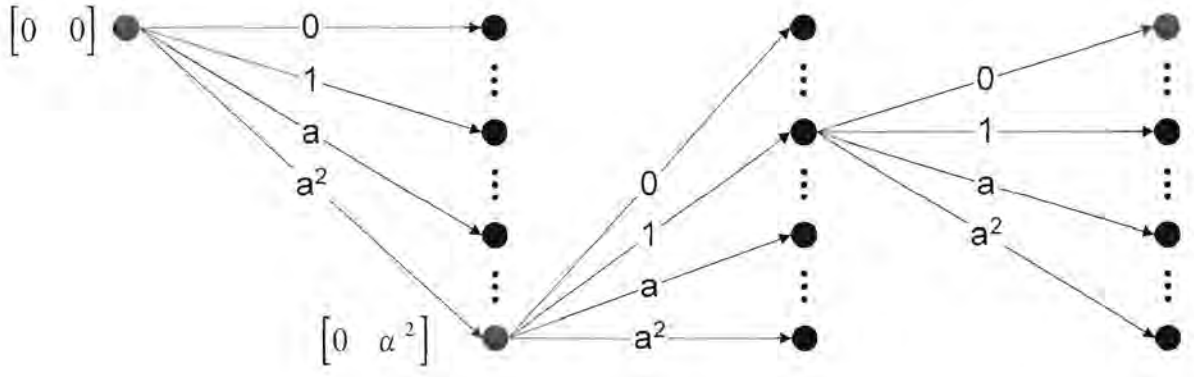

Figure 30 Branch Structure of Reed-Solomon Trellis is Galois Field GF(4)

Next, from the generator matrix **G** given by **Equation 5.27**, the parity check matrix can be calculated. This is done by performing elementary matrix operations on the matrix as outlined in **Chapter 2**. For a complete discussion on this topic, the reader is referred to **Appendix B**. Only a few steps are shown below.

$$\boldsymbol{G} = \left(\begin{array}{ccccc|cc} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{array}\right) = \left(\begin{array}{ccccc|cc} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{array}\right) \tag{5.31}$$

This is further transformed into:

$$\boldsymbol{G} = \left(\begin{array}{ccccc|cc} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{array}\right) \tag{5.32}$$

From the above systematic generator matrix **G** it is possible to read off the parity check matrix in the following form:

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix} \tag{5.33}$$

Next a lookup table can be constructed to simplify the derivation of the trellis diagram. The lookup table is based on elementary Galois field mathematics presented above in **Section 5.2**.

The lookup table for $GF(2^2)$ is divided into 4 separate lookup tables, one for each symbol in the Galois field. These tables are constructed using the primary polynomial in GF(4). This polynomial can be expressed as:

$$\alpha^2 = \alpha + 1 \tag{5.34}$$

The table for code symbol "0":

$$\begin{vmatrix} 0+0=0 \\ 0+1=1 \\ 0+\alpha=\alpha \\ 0+\alpha^2=\alpha^2 \end{vmatrix} \tag{5.35}$$

The table for code symbol "1":

$$\begin{vmatrix} 1+0=1 \\ 1+1=0 \\ 1+\alpha=\alpha^2 \\ 1+\alpha^2=\alpha \end{vmatrix} \tag{5.36}$$

The table for code symbol "α ":

$$\begin{vmatrix} \alpha+0=\alpha \\ \alpha+1=\alpha^2 \\ \alpha+\alpha=0 \\ \alpha+\alpha^2=1 \end{vmatrix} \tag{5.37}$$

The table for code symbol "α^2":

$$\begin{vmatrix} \alpha^2+0=\alpha^2 \\ \alpha^2+1=\alpha \\ \alpha^2+\alpha=1 \\ \alpha^2+\alpha^2=0 \end{vmatrix} \tag{5.38}$$

Since the number of states in the trellis is 16, they can be divided into 4 sub-parts containing 4 states each. The significance of this is explained later. The 4 sub-parts along with their respective 4 states each are numbered as follows:

Sub-part 1 along with first set of four states:

$$SP_1 = \left\{\begin{bmatrix}0 & 0\end{bmatrix},\begin{bmatrix}0 & 1\end{bmatrix},\begin{bmatrix}0 & \alpha\end{bmatrix},\begin{bmatrix}0 & \alpha^2\end{bmatrix}\right\} \tag{5.39}$$

Sub-part 2 along with second set of four states:

$$SP_2 = \left\{\begin{bmatrix}1 & 0\end{bmatrix},\begin{bmatrix}1 & 1\end{bmatrix},\begin{bmatrix}1 & \alpha\end{bmatrix},\begin{bmatrix}1 & \alpha^2\end{bmatrix}\right\} \tag{5.40}$$

Sub-part 3 along with third set of four states:

$$SP_3 = \left\{\begin{bmatrix}\alpha & 0\end{bmatrix},\begin{bmatrix}\alpha & 1\end{bmatrix},\begin{bmatrix}\alpha & \alpha\end{bmatrix},\begin{bmatrix}\alpha & \alpha^2\end{bmatrix}\right\} \tag{5.41}$$

Sub-part 4 along with fourth set of four states:

$$SP_4 = \left\{\left[\alpha^2 \quad 0\right],\left[\alpha^2 \quad 1\right],\left[\alpha^2 \quad \alpha\right],\left[\alpha^2 \quad \alpha^2\right]\right\} \tag{5.42}$$

The next step in the design process is to identify trellis branches associated with the individual code words. This is done by finding all the connecting branches between nodes of the trellis. The starting node is taken as state [0,0], and paths are calculated from here onwards. The reader is again reminded that this process is very similar to the one provided in **Chapter 3**. It should also be noted that paths emanating from nodes are only calculated for states that have at least one incoming branch. An exception to the rule is the above mentioned starting state.

As shown in **Figure 20**, each symbol in the Galois field causes a branch to emanate from each node, implying, at first glance that 4 calculations will have to be done in order to find the 4 branches emanating from every node or state.

For the first section of the trellis, the first column of the parity check matrix is used for all calculations.

For reference, the parity check matrix is repeated. For completeness, the unit matrix has been appended to the end:

$$\boldsymbol{H} = \left(\begin{array}{ccccc|cc} 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{array}\right) \tag{5.43}$$

The following calculations yield the first 4 branches emanating from node [0 0] where each branch is labeled with a specific input symbol, 0, 1, α or α^2:

$$h_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \tag{5.44}$$

$$\begin{aligned}
\begin{bmatrix}0 & 0\end{bmatrix} \rightarrow 0: &\quad \begin{bmatrix}0 & 0\end{bmatrix} + 0\cdot\begin{bmatrix}1 & 1\end{bmatrix} = \begin{bmatrix}0 & 0\end{bmatrix} \\
1: &\quad \begin{bmatrix}0 & 0\end{bmatrix} + 1\cdot\begin{bmatrix}1 & 1\end{bmatrix} = \begin{bmatrix}1 & 1\end{bmatrix} \\
\alpha: &\quad \begin{bmatrix}0 & 0\end{bmatrix} + \alpha\cdot\begin{bmatrix}1 & 1\end{bmatrix} = \begin{bmatrix}\alpha & \alpha\end{bmatrix} \\
\alpha^2: &\quad \begin{bmatrix}0 & 0\end{bmatrix} + \alpha^2\cdot\begin{bmatrix}1 & 1\end{bmatrix} = \begin{bmatrix}\alpha^2 & \alpha^2\end{bmatrix}
\end{aligned} \tag{5.45}$$

It can be seen that quite a number of calculations are involved in order to find the nodes that are connected by a specific connection branch associated with a specific symbol. In order to still give a good overview of the process, a shorthand notation is employed to summarise all the relevant results of the calculations.

It can be seen that the next batch of calculations will deliver a set of 16 active nodes, due to the fact that 4 branches exit any active node. An active node is any node which has at least one trellis branch entering it. When the trellis is "fanned-out", in other words such that each node or state has at least one branch entering it, the calculations will yield 64 branches interconnecting the various nodes.

The method used to obtain the trellis branches, is based on the same structure used in **Equation 5.45**, but by only displaying the starting and ending nodes of each branch. The details of the calculations are left out. These may be readily filled in by the reader without a lot of effort.

For each depth of the trellis, the next column of the parity check matrix is used. This means, that at the last section of the trellis, the last column of the parity check matrix is employed to base the calculations on and so on.

On the next few pages, the complete results of all the calculations are presented.

The results for the first section of the trellis are presented on this page.

The relevant column of the parity check matrix is the following:

$$h_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

The following equation was used to calculate the table below:

$$NewState = OldState + Symbols \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Old State	New States	Old State	New States	Old State	New States	Old State	New States
[0 0]	[0 0]	[1 0]	-	[α 0]	-	[α² 0]	-
	[1 1]		-		-		-
	[α α²]		-		-		-
	[α² α²]		-		-		-
[0 1]	-	[1 1]	-	[α 1]	-	[α² 1]	-
	-		-		-		-
	-		-		-		-
	-		-		-		-
[0 α]	-	[1 α]	-	[α α]	-	[α² α]	-
	-		-		-		-
	-		-		-		-
	-		-		-		-
[0 α²]	-	[1 α²]	-	[α α²]	-	[α² α²]	-
	-		-		-		-
	-		-		-		-
	-		-		-		-

Table 2 Nodes for State 1 of (7,5) RS-Code in GF(4)

The results for the second section of the trellis are presented on this page.

The relevant column of the parity check matrix is the following:

$$\boldsymbol{h}_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

The following equation was used to calculate the table below:

$$NewState = OldState + Symbols \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Old State	New States	Old State	New States	Old State	New States	Old State	New States
[0 0]	[0 0]	[1 0]	-	[α 0]	-	[α^2 0]	-
	[1 0]		-		-		-
	[α 0]		-		-		-
	[α^2 0]		-		-		-
[0 1]	-	[1 1]	[1 1]	[α 1]	-	[α^2 1]	-
	-		[0 1]		-		-
	-		[α^2 1]		-		-
	-		[α 1]		-		-
[0 α]	-	[1 α]	-	[α α]	[α α]	[α^2 α]	-
	-		-		[α^2 α]		-
	-		-		[0 α]		-
	-		-		[1 α]		-
[0 α^2]	-	[1 α^2]	-	[α α^2]	-	[α^2 α^2]	[α^2 α^2]
	-		-		-		[α α^2]
	-		-		-		[1 α^2]
	-		-		-		[0 α^2]

Table 3 Nodes for State 2 of (7,5) RS-Code in GF(4)

The results for the third section of the trellis are presented on this page.

The relevant column of the parity check matrix is the following:

$$\boldsymbol{h}_3 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

The following equation was used to calculate the table below:

$$NewState = OldState + Symbols \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Old State	New States	Old State	New States	Old State	New States	Old State	New States
[0 0]	[0 0]	[1 0]	[1 0]	[α 0]	[α 0]	[α^2 0]	[α^2 0]
	[0 1]		[1 1]		[α 1]		[α^2 1]
	[0 α]		[1 α]		[α α]		[α^2 α]
	[0 α^2]		[1 α^2]		[α α^2]		[α^2 α^2]
[0 1]	[0 1]	[1 1]	[1 1]	[α 1]	[α 1]	[α^2 1]	[α^2 1]
	[0 0]		[1 0]		[α 0]		[α^2 0]
	[0 α^2]		[1 α^2]		[α α^2]		[α^2 α^2]
	[0 α]		[1 α]		[α α]		[α^2 α]
[0 α]	[0 α]	[1 α]	[1 α]	[α α]	[α α]	[α^2 α]	[α^2 α]
	[0 α^2]		[1 α^2]		[α α^2]		[α^2 α^2]
	[0 0]		[1 0]		[α 0]		[α^2 0]
	[0 1]		[1 1]		[α 1]		[α^2 1]
[0 α^2]	[0 α^2]	[1 α^2]	[1 α^2]	[α α^2]	[α α^2]	[α^2 α^2]	[α^2 α^2]
	[0 α]		[1 α]		[α α]		[α^2 α]
	[0 1]		[1 1]		[α 1]		[α^2 1]
	[0 0]		[1 0]		[α 0]		[α^2 0]

Table 4 Nodes for State 3 of (7,5) RS-Code in GF(4)

The results for the fourth section of the trellis are presented on this page.

The relevant column of the parity check matrix is the following:

$$\boldsymbol{h}_4 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

The following equation was used to calculate the table below:

$$NewState = OldState + Symbols \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Old State	New States	Old State	New States	Old State	New States	Old State	New States
[0 0]	[0 0]	[1 0]	[1 0]	[α 0]	[α 0]	[α^2 0]	[α^2 0]
	[1 1]		[0 1]		[α^2 1]		[α 1]
	[α α^2]		[α^2 α]		[0 α]		[1 α]
	[α^2 α^2]		[α α^2]		[1 α^2]		[0 α^2]
[0 1]	[0 1]	[1 1]	[1 1]	[α 1]	[α 1]	[α^2 1]	[α^2 1]
	[1 0]		[0 0]		[α^2 0]		[α 0]
	[α α^2]		[α^2 α^2]		[0 α^2]		[1 α^2]
	[α^2 α]		[α α]		[1 α]		[0 α]
[0 α]	[0 α]	[1 α]	[1 α]	[α α]	[α α]	[α^2 α]	[α^2 α]
	[1 α^2]		[0 α^2]		[α^2 α^2]		[α α^2]
	[α 0]		[α^2 0]		[0 0]		[1 0]
	[α^2 1]		[α 1]		[1 1]		[0 1]
[0 α^2]	[0 α^2]	[1 α^2]	[1 α^2]	[α α^2]	[α α^2]	[α^2 α^2]	[α^2 α^2]
	[1 α]		[0 α]		[α^2 α]		[α α]
	[α 1]		[α^2 1]		[0 1]		[1 1]
	[α^2 0]		[α 0]		[1 0]		[0 0]

Table 5 Nodes for State 4 of (7,5) RS-Code in GF(4)

The results for the fifth section of the trellis are presented on this page.

The relevant column of the parity check matrix is the following:

$$\boldsymbol{h}_5 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

The following equation was used to calculate the table below:

$$NewState = OldState + Symbols \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Old State	New States	Old State	New States	Old State	New States	Old State	New States
[0 0]	[0 0]	[1 0]	[1 0]	[α 0]	[α 0]	[α^2 0]	[α^2 0]
	[1 0]		[0 0]		[α^2 0]		[α 0]
	[α 0]		[α^2 0]		[0 0]		[1 0]
	[α^2 0]		[α 0]		[1 0]		[0 0]
[0 1]	[0 1]	[1 1]	[1 1]	[α 1]	[α 1]	[α^2 1]	[α^2 1]
	[1 1]		[0 1]		[α^2 1]		[α 1]
	[α 1]		[α^2 1]		[0 1]		[1 1]
	[α^2 1]		[α 1]		[1 1]		[0 1]
[0 α]	[0 α]	[1 α]	[1 α]	[α α]	[α α]	[α^2 α]	[α^2 α]
	[1 α]		[0 α]		[α^2 α]		[α α]
	[α α]		[α^2 α]		[0 α]		[1 α]
	[α^2 α]		[α α]		[1 α]		[0 α]
[0 α^2]	[0 α^2]	[1 α^2]	[1 α^2]	[α α^2]	[α α^2]	[α^2 α^2]	[α^2 α^2]
	[1 α^2]		[0 α^2]		[α^2 α^2]		[α α^2]
	[α α^2]		[α^2 α^2]		[0 α^2]		[1 α^2]
	[α^2 α^2]		[α α^2]		[1 α^2]		[0 α^2]

Table 6 Nodes for State 5 of (7,5) RS-Code in GF(4)

The results for the sixth section of the trellis are presented on this page.

The relevant column of the parity check matrix is the following:

$$\boldsymbol{h}_6 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

The following equation was used to calculate the table below:

$$NewState = OldState + Symbols \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Old State	New States	Old State	New States	Old State	New States	Old State	New States
[0 0]	[0 0]	[1 0]	[1 0]	[α 0]	[α 0]	[α^2 0]	[α^2 0]
	[1 0]		[0 0]		[α^2 0]		[α 0]
	[α 0]		[α^2 0]		[0 0]		[1 0]
	[α^2 0]		[α 0]		[1 0]		[0 0]
[0 1]	[0 1]	[1 1]	[1 1]	[α 1]	[α 1]	[α^2 1]	[α^2 1]
	[1 1]		[0 1]		[α^2 1]		[α 1]
	[α 1]		[α^2 1]		[0 1]		[1 1]
	[α^2 1]		[α 1]		[1 1]		[0 1]
[0 α]	[0 α]	[1 α]	[1 α]	[α α]	[α α]	[α^2 α]	[α^2 α]
	[1 α]		[0 α]		[α^2 α]		[α α]
	[α α]		[α^2 α]		[0 α]		[1 α]
	[α^2 α]		[α α]		[1 α]		[0 α]
[0 α^2]	[0 α^2]	[1 α^2]	[1 α^2]	[α α^2]	[α α^2]	[α^2 α^2]	[α^2 α^2]
	[1 α^2]		[0 α^2]		[α^2 α^2]		[α α^2]
	[α α^2]		[α^2 α^2]		[0 α^2]		[1 α^2]
	[α^2 α^2]		[α α^2]		[1 α^2]		[0 α^2]

Table7 Nodes for State 6 of (7,5) RS-Code in GF(4)

The results for the seventh section of the trellis are given on this page.

The relevant column of the parity check matrix is the following:

$$\boldsymbol{h}_7 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

The following equation was used to calculate the table below:

$$NewState = OldState + Symbols \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Old State	New States	Old State	New States	Old State	New States	Old State	New States
[0 0]	[0 0]	[1 0]	[1 0]	[α 0]	[α 0]	[α^2 0]	[α^2 0]
	[0 1]		[1 1]		[α 1]		[α^2 1]
	[0 α]		[1 α]		[α α]		[α^2 α]
	[0 α^2]		[1 α^2]		[α α^2]		[α^2 α^2]
[0 1]	[0 1]	[1 1]	[1 1]	[α 1]	[α 1]	[α^2 1]	[α^2 1]
	[0 0]		[1 0]		[α 0]		[α^2 0]
	[0 α^2]		[1 α^2]		[α α^2]		[α^2 α^2]
	[0 α]		[1 α]		[α α]		[α^2 α]
[0 α]	[0 α]	[1 α]	[1 α]	[α α]	[α α]	[α^2 α]	[α^2 α]
	[0 α^2]		[1 α^2]		[α α^2]		[α^2 α^2]
	[0 0]		[1 0]		[α 0]		[α^2 0]
	[0 1]		[1 1]		[α 1]		[α^2 1]
[0 α^2]	[0 α^2]	[1 α^2]	[1 α^2]	[α α^2]	[α α^2]	[α^2 α^2]	[α^2 α^2]
	[0 α]		[1 α]		[α α]		[α^2 α]
	[0 1]		[1 1]		[α 1]		[α^2 1]
	[0 0]		[1 0]		[α 0]		[α^2 0]

Table 8 Nodes for State 7 of (7,5) RS-Code in GF(4)

From the above tables, it is possible to construct a complete trellis by connecting all relevant branches on an empty trellis grid.

Before this is done however, a few important comments are in order. It is obvious that quite a lot of calculations are needed in order to be able to draw the final trellis diagram. It is easy to see that the complexity will increase as n and k of the code increases. However, as the order of the Galois field increases an even greater increase in complexity is affected.

One possible solution to overcome the complexity is to employ a lookup table, which alleviates the need for extensive branch calculations.

Observing the tables that have been constructed a definite pattern emerges. The practical implication of this is, that once a point in the lookup table is found, the next outcomes may be uniquely determined from the entries of those tables which yielded the last output result. The other elements can just be read of in sequence. In this way, the derivation of the complete trellis is vastly simplified.

The next step in the design process is to produce a graphical display of the trellis diagram from the calculated data. It is a fairly simple process, in which a starting and ending node or state are connected with an interconnecting branch. The branches in the above tables of results are defined by listing the starting and ending states of a particular section in the trellis. The associated symbol, which is the output for that particular branch, may also be found from above the tables, since for each given node, the first result was obtained from the first symbol, the second result was obtained from the second symbol and so on. This is illustrated on the next few pages, where trellis patterns for subsequent sections of the trellis diagrams are displayed consecutively.

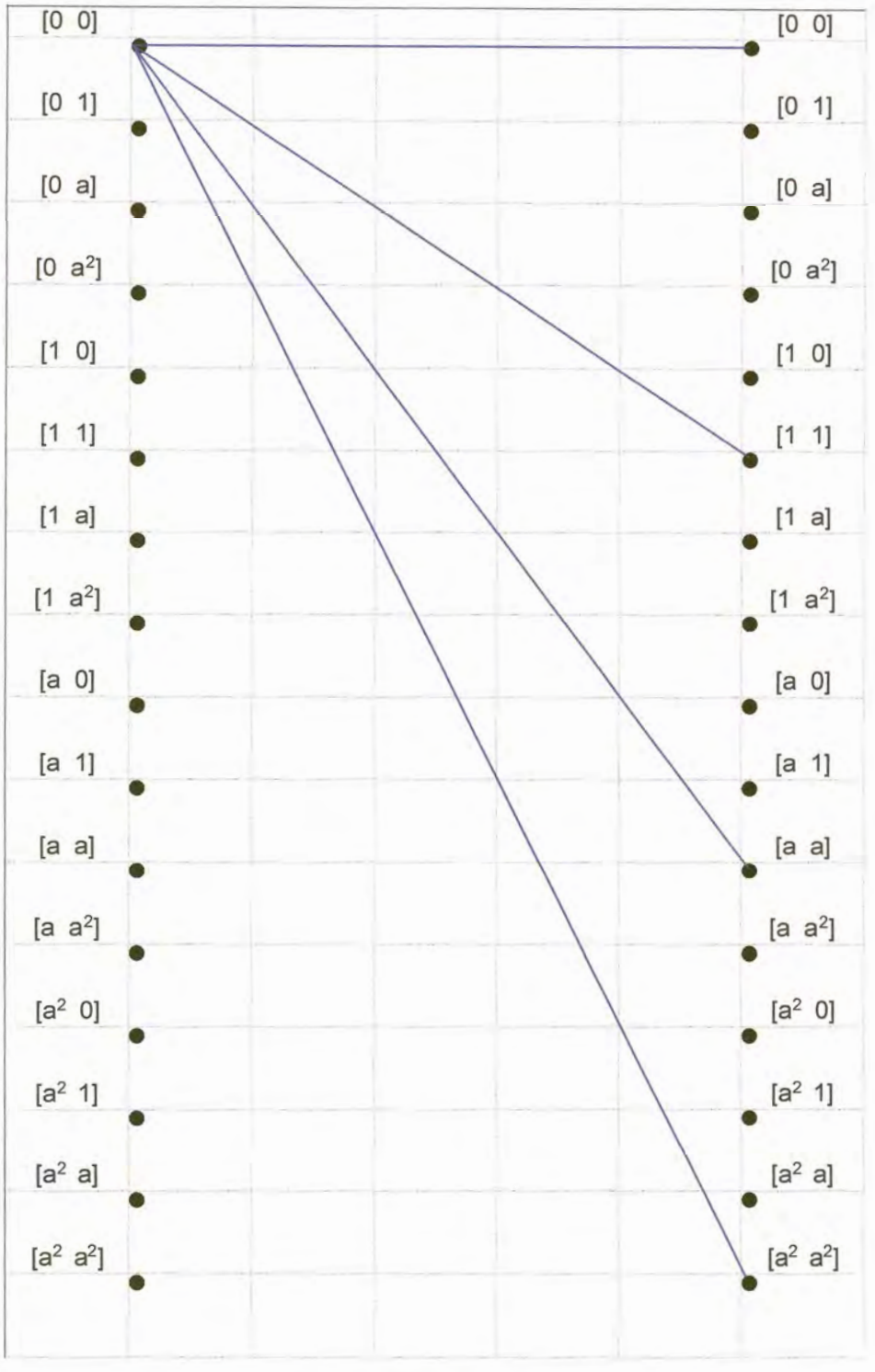

Figure 31 Section 1 of (7,5) Reed-Solomon Trellis in GF(4).

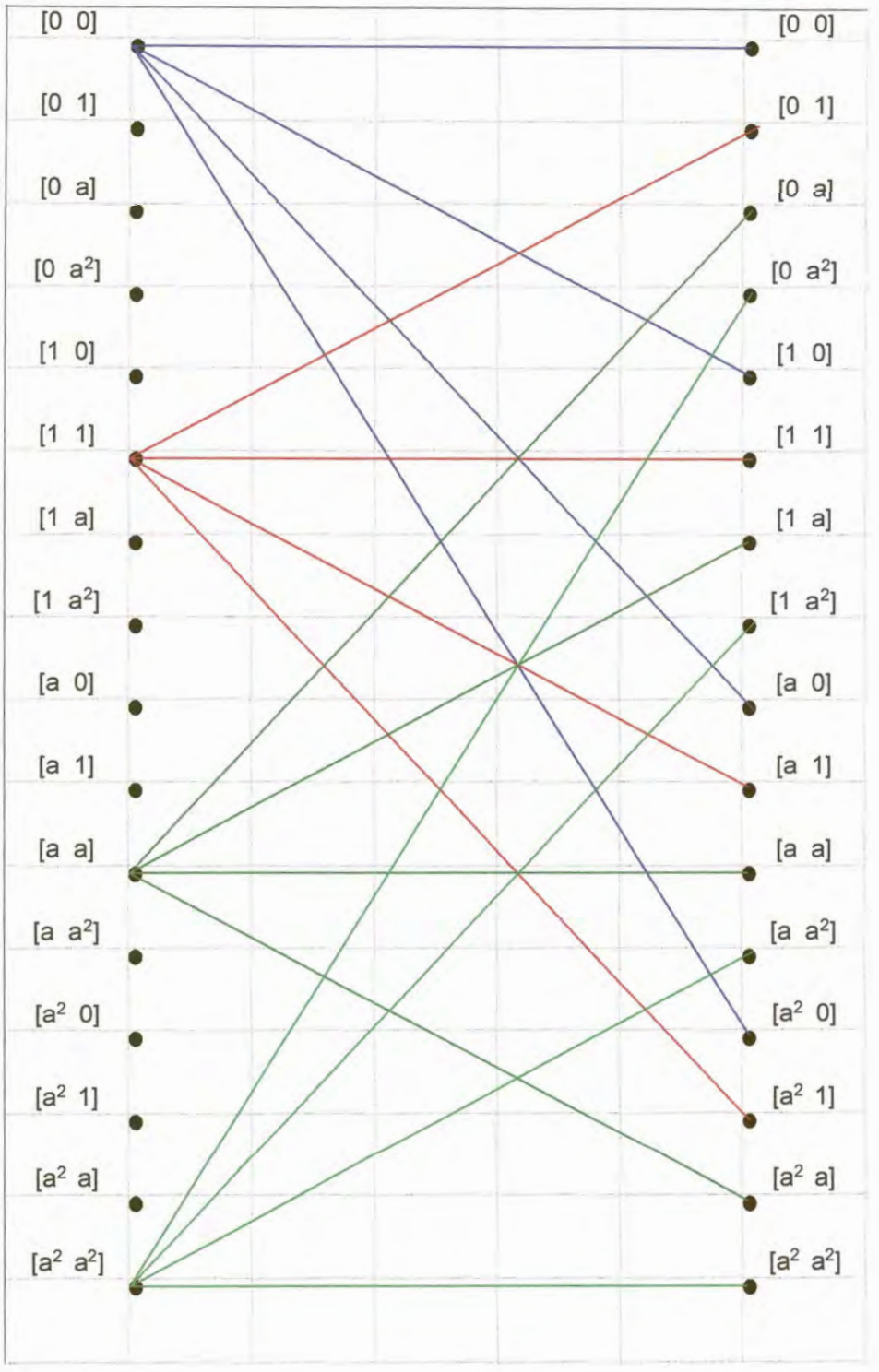

Figure 32 Section 2 of (7,5) Reed-Solomon Trellis in GF(4).

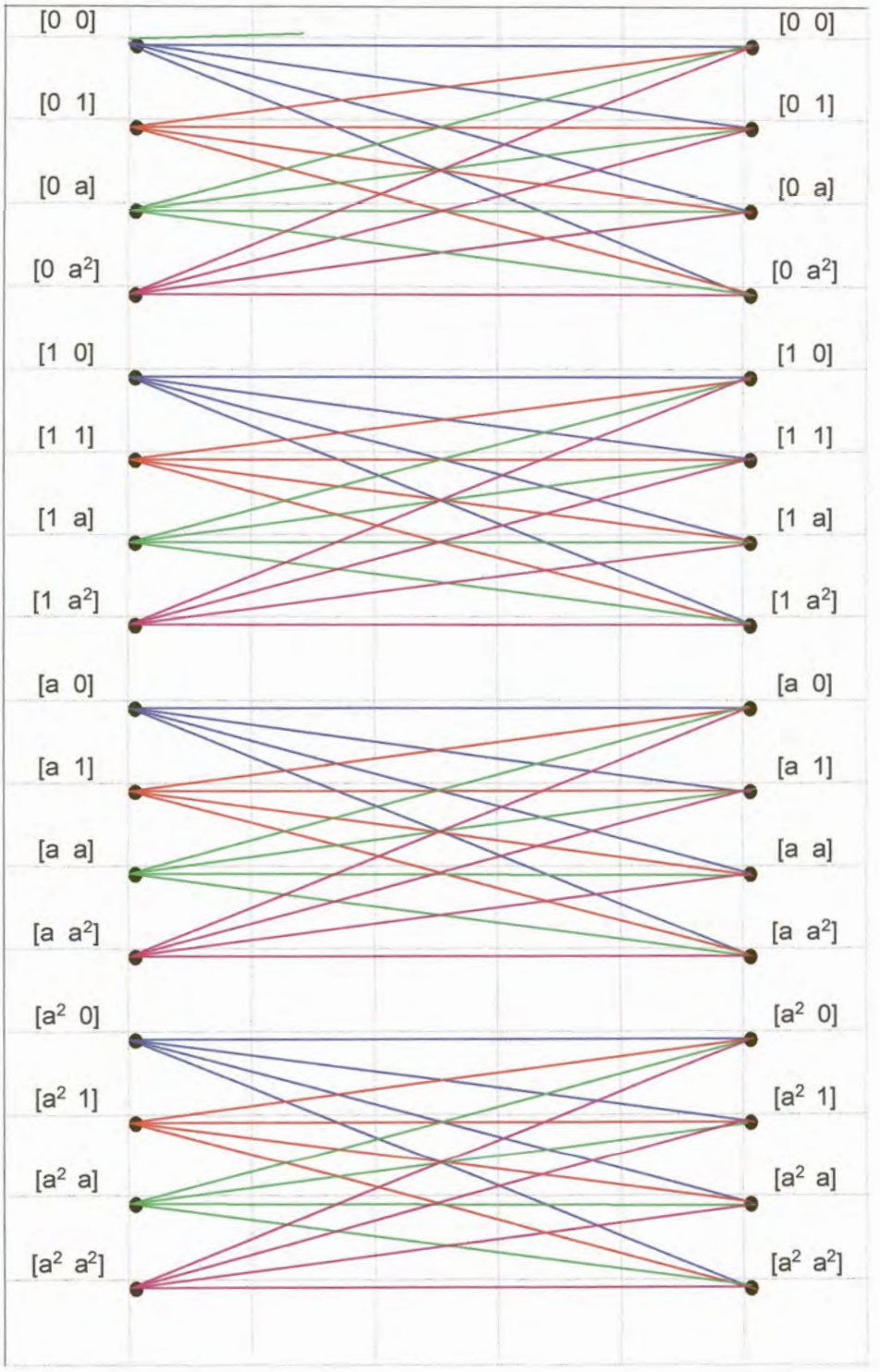

Figure 33 Section 3 of (7,5) Reed-Solomon Trellis in GF(4)

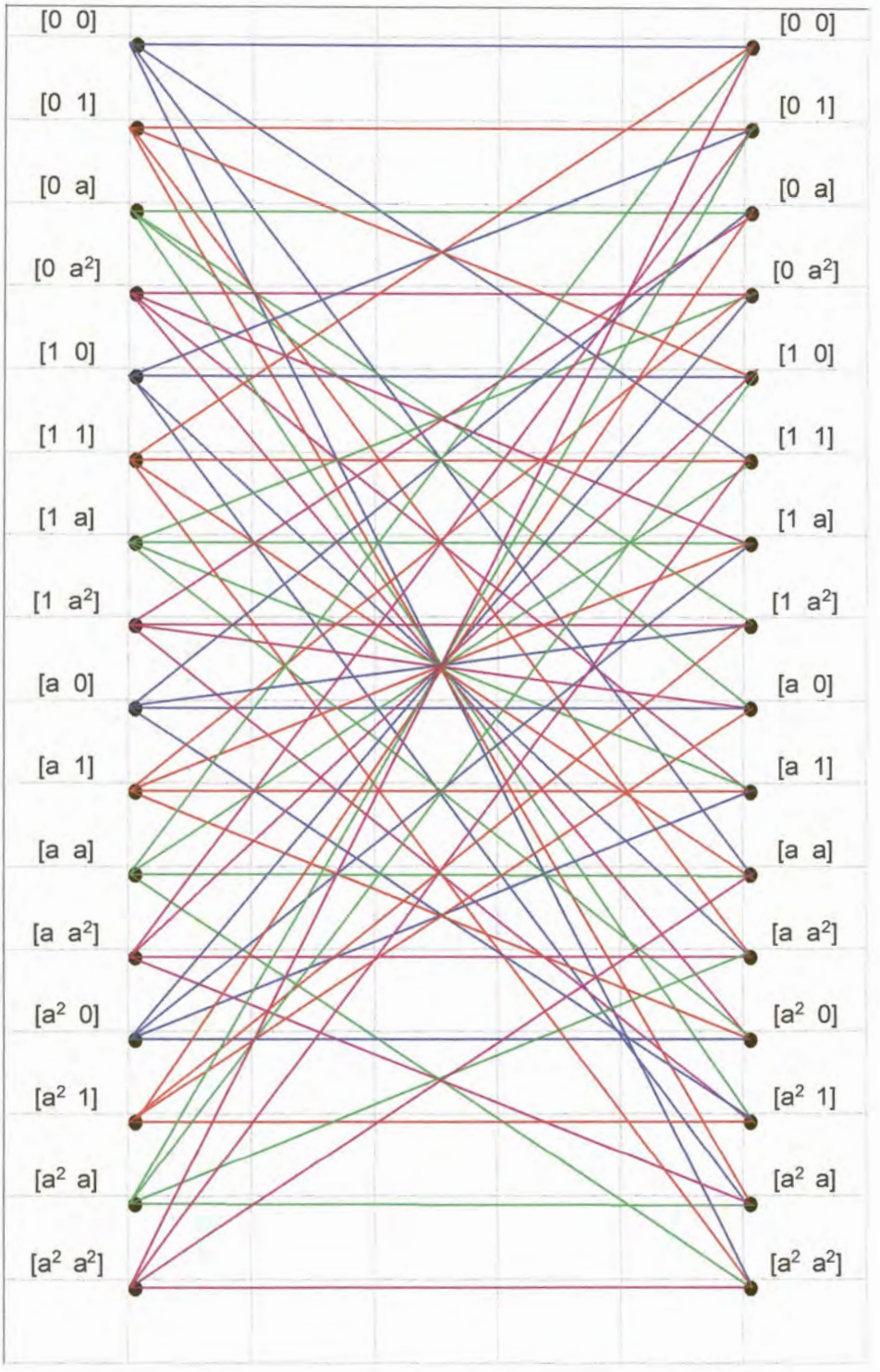

Figure 34 Section 4 of (7,5) Reed-Solomon Trellis in GF(4).

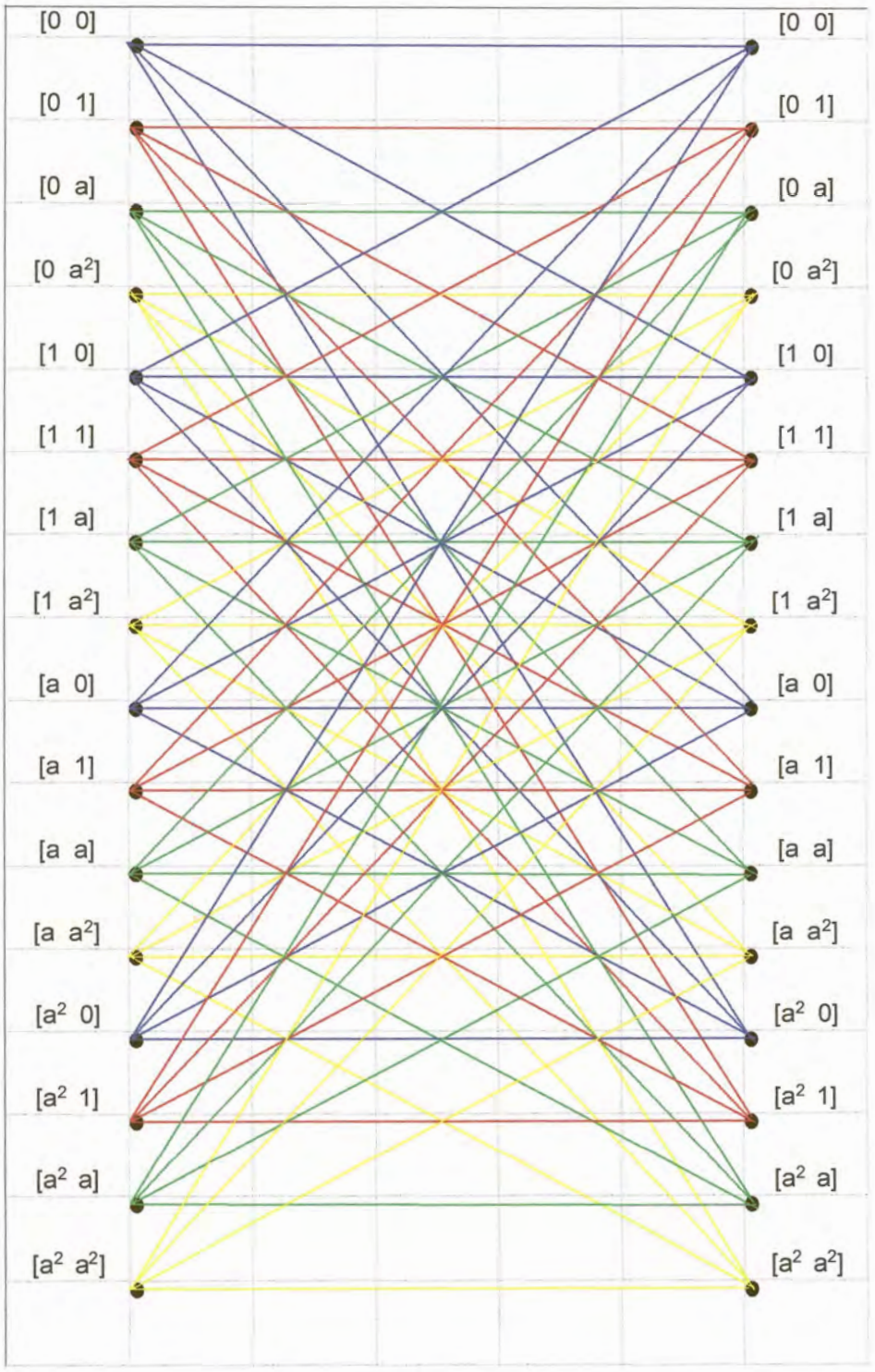

Figure 35 Section 5 of (7,5) Reed-Solomon Trellis in GF(4).

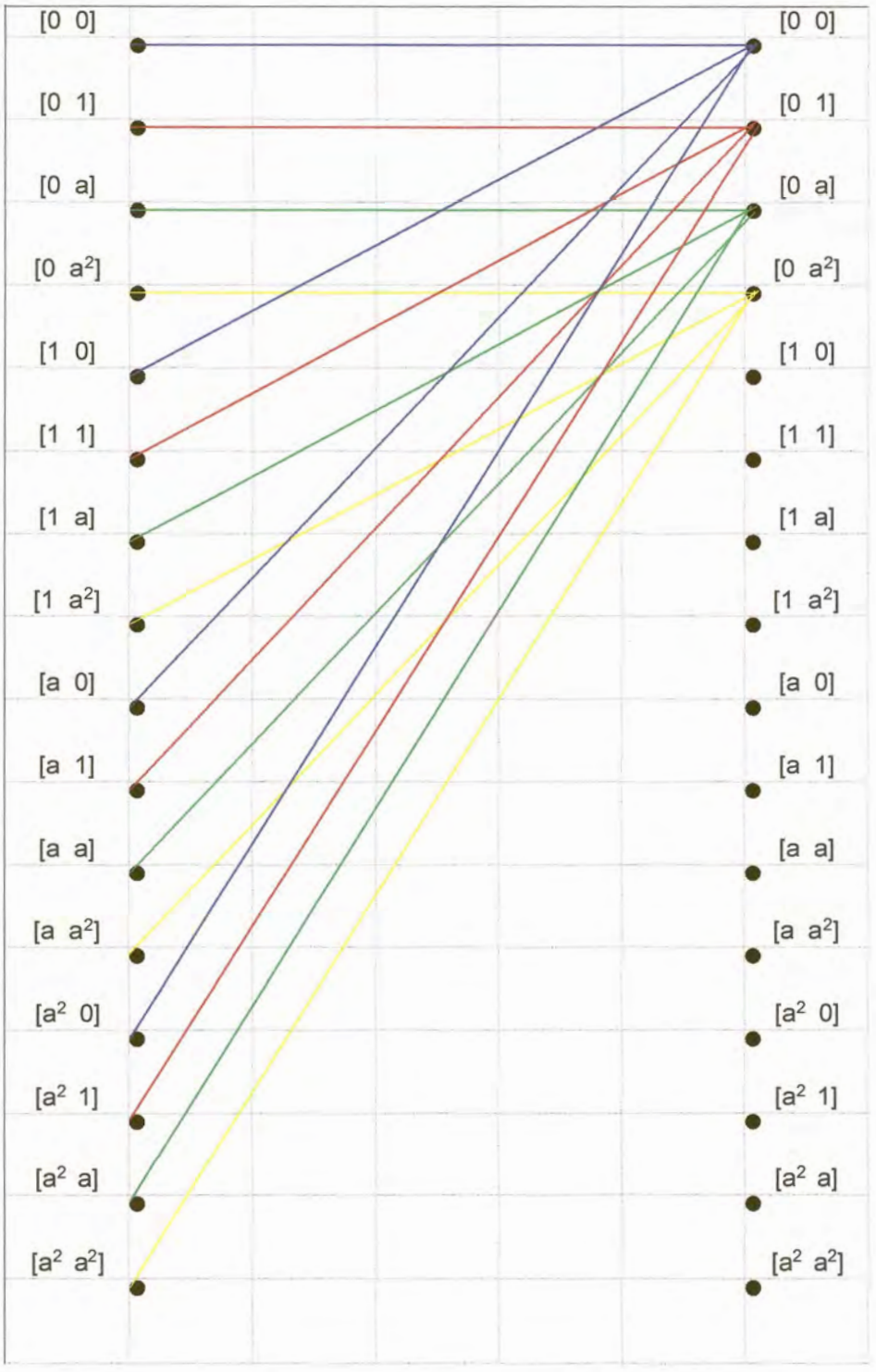

Figure 36 Section 6 of (7,5) Reed-Solomon Trellis in GF(4).

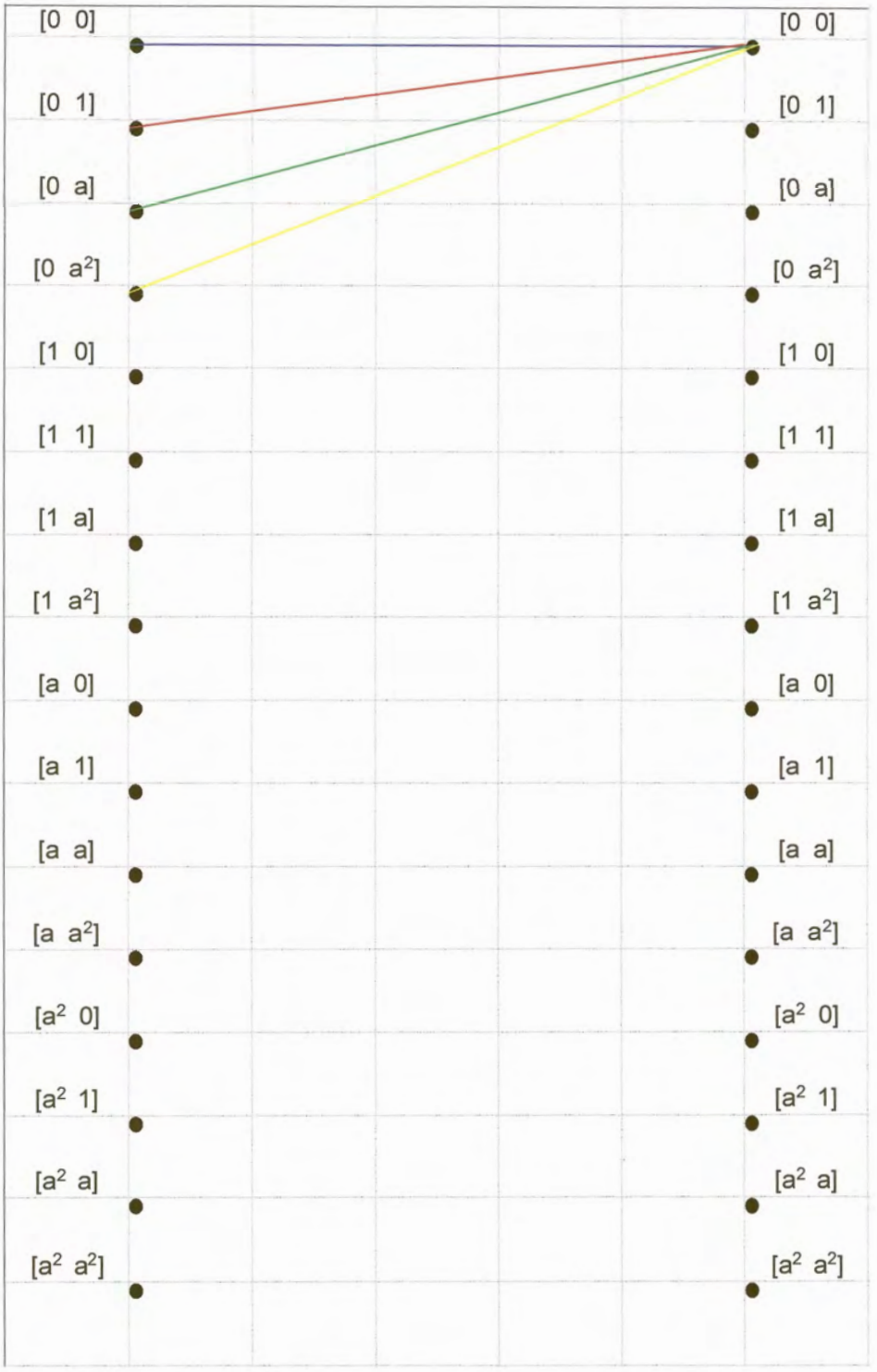

Figure 37 Section 7 of (7,5) Reed-Solomon Trellis in GF(4).

5.6 A Topological Trellis Construction Scheme for Reed-Solomon Codes

As mentioned before, one of the largest restrictions on the use of a Reed-Solomon trellis, is the large complexity involved. The small code described above, viz the (7,5) RS-code, already had a fairly huge trellis structure. This of course makes the decoding process complex and slow to perform.

The first task in the decoding process is the construction of the trellis diagram which is used by the decoder. The realtime construction of the trellis can be omitted if the trellis is hard coded in the decoder. This however imposes a serious limitation on the versatility of the encoder/decoder, as a new decoder/encoder will have to be designed for every new application. It would be very advantageous if an elegant trellis construction technique could be developed.

As the base of the Galois field increases, the number of possible symbols in the code increases. This in turn has a negative impact on the complexity of the system. The number of branches per node in the trellis expands exponentially. A method is required which would limit the required number of calculations involved in obtaining a trellis diagram. Such a method would lead to large savings in computational complexity.

In this discussion, the code to be considered is the one used in **Subsection 6.2.3**, namely the (7,5) RS-code. This code has a parity check matrix in which the number of values in a column is 2. The whole procedure can easily be expanded in order to accommodate larger parity matrixes and codes.

This is where the previously mentioned partitioning of the states into 4 sub-parts is applied. As the state numbering follows a binary sequence, it can be seen that each state is numbered by a unique ordering of the 4 symbols in the field. Due to this, each set of four sub-parts has the same unique symbol in the first position of this state numbering scheme. The second symbol is then just one of the four remaining symbols. In other words, for each unique symbol in the first position of a state which identifies the sub-part of the states, any of the four other symbols are employed to identify a state within the sub-parts. This might seem very insignificant, but will prove invaluable later on.

As can be seen from the trellis diagrams, the first node of each sub-part is coded in a specific colour, which it retains in all other states as well. Due to the fact that 4 nodes or states exist within each sub-part, 4 colours are used. These colours are given in the order blue, red, green and yellow. Again, this seems very unimportant, but together with the first observation, it will provide a very practical method by which the complexity of trellis design can be simplified and confined.

The next important observation is the parity check matrix. Every column of the parity check matrix determines the connecting branches for a specific section of the trellis, viz the first column specifies the first section of the trellis, the second column the second section, and so on.

The values inside the parity check matrix depends on the Galois field used. In other words, if GF(2) is used, the only possible values in the matrix are the binary values 0 and 1. If however GF(4) is used, then the possible values the parity check matrix can assume are the 4 symbols in this field, namely $0,1,\alpha$ and α^2. At the moment all that needs to be remembered, is that there is a different construction associated with the element 0, than with the non zero elements 1, α and α^2.

The examples show that the calculations made to obtain the branches of the trellis seem to follow a cyclic pattern. This can be explained as follows. A lookup table for each symbol can be constructed, as was done before. This lookup table gives the result of adding any possible valid symbol to the symbol of the lookup table. When the first calculation for a node is made, a specific position in the lookup table will provide the answer. If this answer is found, no more calculations need to be made, since the remaining branch connections can be read off from the subsequent positions of the lookup table i.e. the specification of trellis branches follow a very specific topological pattern. The table has to be wrapped when the bottom of the table is reached before all branches have been specified.

The tricky task now is to formalize all these findings, in order to provide a systematic algorithm for trellis construction with low complexity.

The foregoing findings may be summarized point-wise as follows:

- The trellis state numbering is divided into several sub-parts, in which the first

element of the numbering stays the same symbol in the sub-part.

- Colour coding a specific node is important. The colour of a node or state has to remain the same in each of the sub-parts.
- The parity check matrix's columns can be divided into those that have a zero element in the first position and those that do not.
- The symbol in the second position of the parity check matrix column, is important as far as the selection of subsequent points is concerned.
- The results of the calculations are cyclic. The answers can thus be read off from a lookup table as soon as the starting point in the table has been established.

The discussion will commence by describing and explaining the last few steps in the above list based on the (7,5) RS-example. It is said that the results and calculations are cyclic. This can be illustrated by using the first node, and the first column of the parity check matrix to calculate the 4 branches that emanate from this active starting node. The node is labeled [0 0]. In order to find the 4 nodes to which it is connected, the first column of the parity check matrix is multiplied by the symbols and added to the starting node [0 0]. In this way four new nodes in the second section of the trellis are obtained namely [0 0], [0 1], [0 a] and [0 a^2]. Ignoring the first element of the node, the second one changes cyclically. The first calculation can be solved by using the first row of the lookup table of **Equation 5.35**. The answer of the lookup is 0. In order to find the next three nodes, all that has to be done is to continue reading off the rows of the lookup table in **Equation 5.35**. If the bottom of the table is reached prematurely, i.e. before the desired amount of nodes have been numbered, then the table entries are wrapped around. The cyclic procedure therefore follows a mod N cyclic pattern where N denotes the number of branches emanating from a specific node in the code trellis. This wrap-around does not occur since it only occurs when the first element is read off a row other than the first one. The sequence of branch labels obtained when reading the elements off the lookup table in order are 0, 1, a and a^2. Comparing this with the results obtained by calculation, it can be seen that they correspond precisely.

This is the first step in constructing a trellis with low complexity.

The next step is to describe how the values in the columns of the parity check matrix affect the structure of the trellis diagram. Recall, that the columns of the parity check matrix can be divided into two large groups. The first group includes those columns that have the zero element in the first position and the second group those columns that do

not have a zero element in the first position. At this point, the state division into 4 sub-parts becomes important. Two possibilities now exist. If the parity check matrix column has a zero in the first position, then a branch emanating from a node in a specific sub-part will terminate in a node in the same sub-part. If however a column of the parity check matrix is non-zero, then a branch emanating from a node in a specific sub-part, will have a termination in a node from each sub-part of the trellis diagram. As an example of the first case, consider **Figure 26**. It can be seen that the branches emanating from the [0 0] node all end in the same sub-part. This holds true for all instances. The second case is illustrated in **Figure 27**. Here it can be seen that the 4 branches emanating from node [0 0] terminate in each of the 4 sub-parts. This is due to the cyclic phenomenon described in the previous paragraph. Here it is just applicable to the second position of the column of the parity check matrix. It is important to note that the sub-part in which a branch terminates is also determined by the same cyclic principle described in the above paragraph on cycling. If the first branch enters sub-part 3, then the second branch will enter sub-part 4, the third branch sub-part 1 and the last branch sub-part 2. This becomes extremely important when numbering the branches with their corresponding symbols from the Galois field set.

After discussing all of the important issues, a description of the colour coding part is now in order. The colour coding displays the culmination of all the above mentioned steps. **Figure 28** is used as a starting point. This sub-trellis was formed using the parity check column [1 1]. As discussed before, the fact that the first element is a 1, forces the branches leaving a node in a specific sub-part to enter nodes in different sub-parts. This can be seen by just observing the node [0 0]. The branches enter node [0 0] in sub-part 1, node [1 0] in sub-part 2, [α 0] in sub-part 3 and [α^2 0] in sub-part 4. These 4 branches are all marked with blue. Now the interesting part reveals itself. The first node of the second sub-part is also coloured in blue, and it can be seen that they enter the nodes activated by blue branches leaving node [0 0]. This phenomenon occurs throughout the figure. In order to draw the trellis, only the branches of the first sub-part have to be calculated. Hereafter, due to the cyclic phenomenon, all that is required to uniquely identify and label subsequent branches in this section of the trellis.

There is just one other form of topology, which occurs if the first element of the column of the parity check matrix is zero. In this case, as illustrated by **Figure 26**, all the branches emanating from a node in a specific sub-part activate all other nodes in the same sub-part. All that needs to be calculated for this section of the trellis is the first

section. Hereafter all remaining trellis branches may be labeled according to the cyclic pattern described.

In fact, it should be noticed that all that is really necessary to calculate are the branches of the first node in the trellis. This, together with the two types of topologies and the cycling phenomenon determines the trellis uniquely and completely.

For the above trellis, it would normally be necessary to do 404 complex calculations in GF(4). If the cyclic topology is exploited, the whole trellis can be calculated with only 28 complex calculations in GF(4) a saving in the order of a factor of 14.

This method provides an enormous step forward in the whole trellis design procedure for Reed-Solomon codes. Note that the proposed algorithm does not provide a minimal (i.e. minimum number of states and branches) trellis, but only simplifies the trellis construction. A procedure to find a minimal trellis is given in the following chapter.

On the next page, a flow diagram of the topological trellis design procedure is presented.

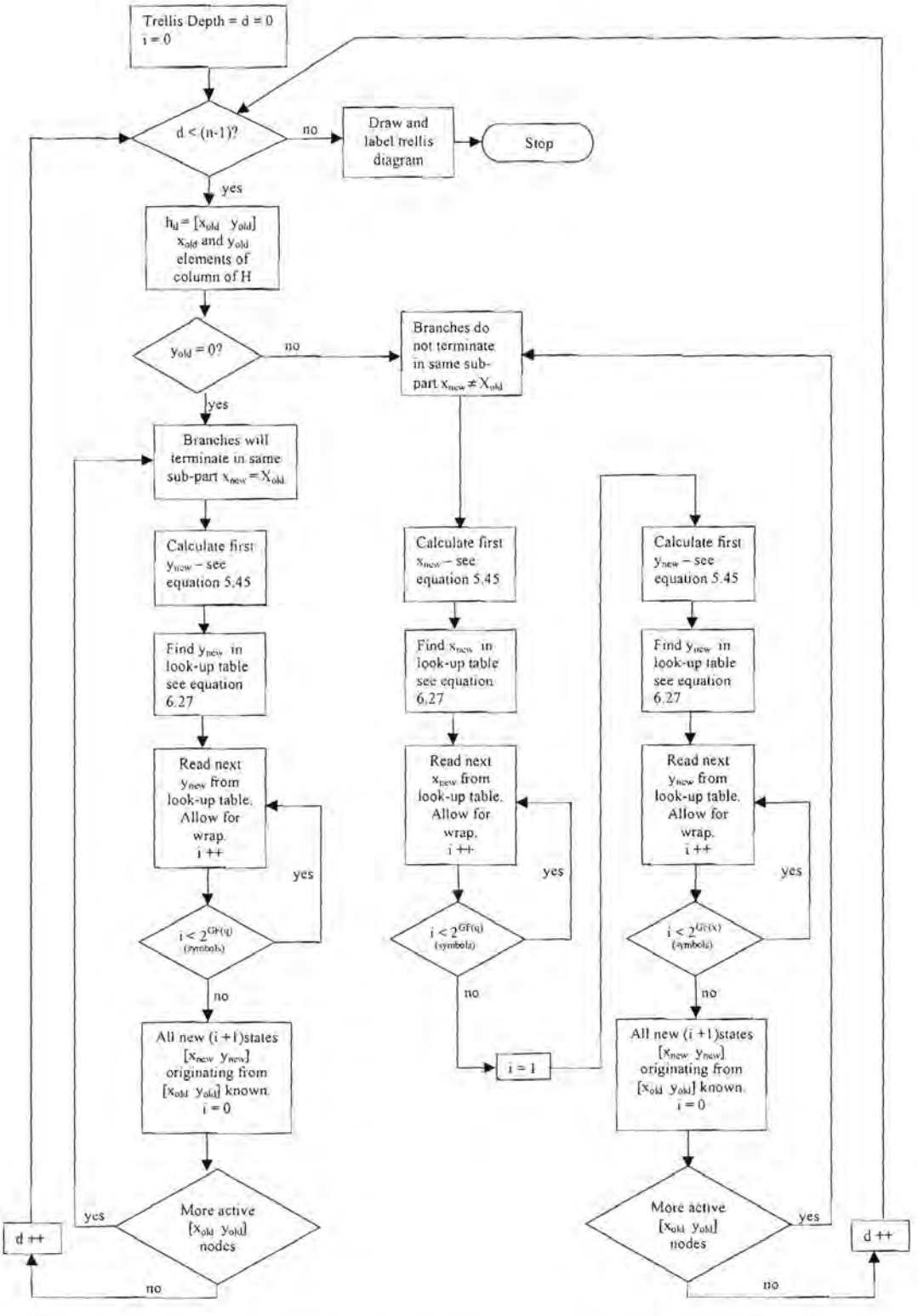

Figure 38 Flow Diagram of the Topological Trellis Design Algorithm

5.7 Symbol Error Rate of Reed-Solomon Code

In this section, the performance of Reed-Solomon codes is investigated. The main task at hand is to show that Reed-Solomon codes can be decoded with the Viterbi Algorithm. The decoding performance is shown to be the same as for algebraic techniques, since the Viterbi algorithm is a maximum likelihood technique.

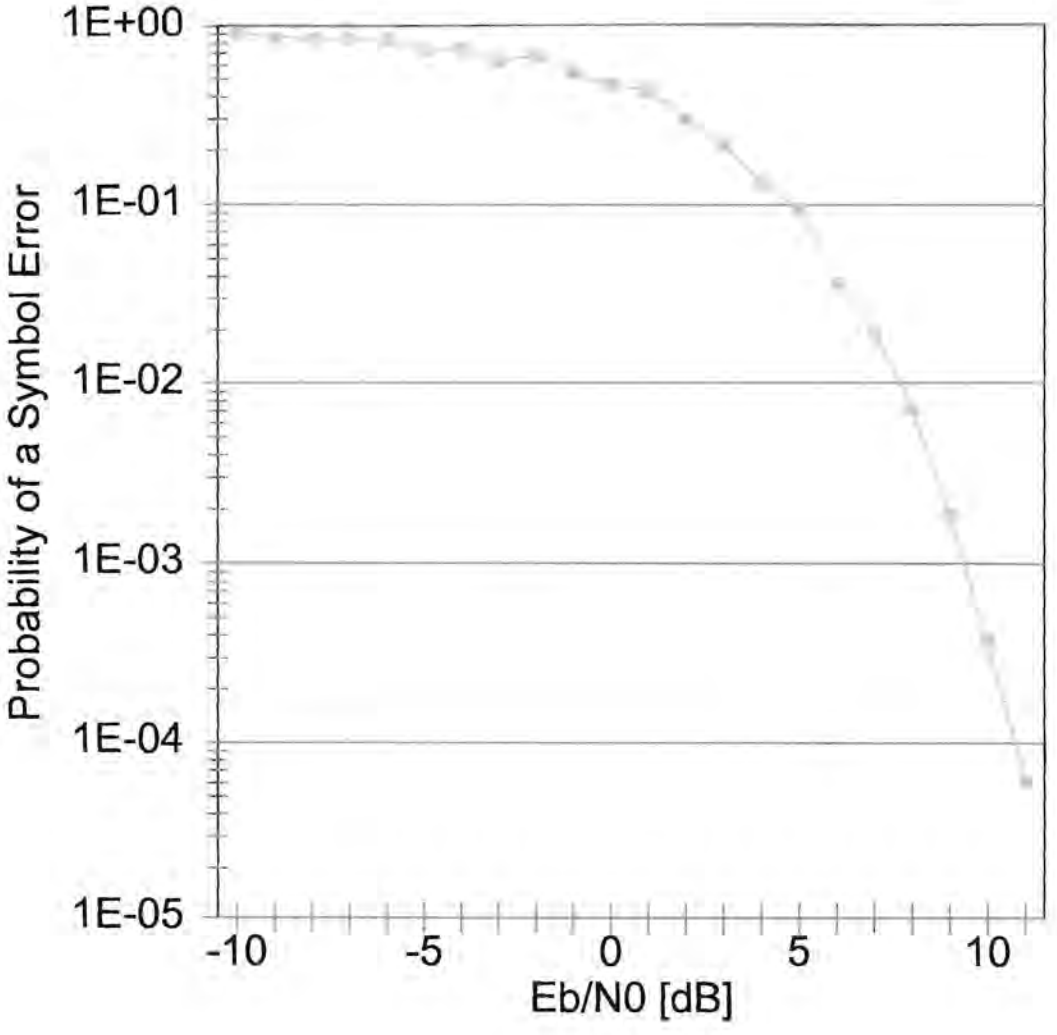

Figure 39 Simulated Symbol Error Rate of (7,5)-Reed-Solomon Code

As mentioned and shown previously, the trellis diagrams for Reed-Solomon codes are very large. This is the main problem with Reed-Solomon codes. As it was possible to construct a trellis for a Reed-Solomon code in the previous chapter, it is now possible to Viterbi decode the Reed-Solomon code. This is illustrated on a relatively small code, as it takes quite a long time to construct a symbol error rate curve. The same principles of course apply to larger codes also.

Again consider the(7,5)-Reed-Solomon code from the previous chapter. The simulation setup employs a Reed-Solomon encoder, the output of which is transmitted over an AWGN channel. In the decoder, a SDML Viterbi decoder is employed. The simulation setup is the same as depicted in **Chapter 2**.

On the previous page, the symbol error rate for the Reed-Solomon code is shown.

5.8 General Discussion on Reed-Solomon Performance

Reed-Solomon codes were traditionally decoded in the frequency domain using the Berlekamp Massey algorithm [20]. This method uses Fourier transforms to do the decoding. Due to the Fourier transform, the decoding is only an approximation, and marginally better decoding results can be obtained employing a maximum likelihood decoding technique such as the Viterbi algorithm. Although the scope of this dissertation was not to show that Viterbi decoding is better than Berlekamp Massey decoding of Reed-Solomon codes, an attempt was made to use existing standard software to find symbol error rate curves for a Berlekamp Massey algorithm. This was possible, but these software packages do not provide for such small codes as the (7,5)-Reed-Solomon code used here. Reed-Solomon codes usually have typical specifications such as $n = 255$ and $k = 251$ or the likes. This meant that a Berlekamp Massey algorithm would have to be re-written in order to do this comparison. The other problem was that the trellis complexity sky-rockets with high values of n and k and that a normal personal computer would not be able to calculate the required symbol error rates for the large block lengths. This does not even take into account the Galois fields which are commonly used in VHDL Reed-Solomon coders. These often run in fields as high as $GF(128)$ or $GF(256)$ compared to the $GF(4)$ field used in the Viterbi simulation presented here.

The comparison which seemed possible at first proved not to be possible with the current state of computer hardware available. The solution to this problem would be the VHDL implementation of such a Viterbi Reed-Solomon decoder. This could then be compared with the normal Berlekamp Massey decoders using a bit error rate analyzer, but this falls beyond the scope of this dissertation.

What has been shown is that it is indeed possible to decode Reed-Solomon codes with decoding techniques usually reserved for convolutional code decoding. This opens many possibilities for integrating coding schemes employing both powerful Reed-Solomon codes and convolutional codes. It is possible to use the same decoder in the decoding process by deriving the resultant trellis of the concatenation of all codes employed.

Chapter 6

Introduction to Trellis Complexity and Trellis Complexity Reduction

6.1 Introduction

In this Chapter, an introduction to the complexity of a block code trellis is given, and a method is examined which yields trellises with lower complexity. Ultimately, the goal is to reduce the trellis complexity by magnitudes in order to achieve a more efficient decoding process.

The trellis complexity of a block code [7], [11], [18] is mainly determined by the number of states and branches in a trellis. The state complexity is usually measured by its state space dimension profile, and the branch complexity by the total number of branches present in the trellis diagram. In **Chapter 2** it was shown that for every generator matrix **G** of a block code, there exists a trellis with a minimum number of states and branches, called a minimal trellis. It was however also shown, that there exist several equivalent codes, described by there relevant generator matrixes. These equivalent trellises derived from their respective generator matrixes, describe the same set of code words. For each equivalent code, there exists a number of trellis representations, only one of which is minimal. The aim is thus to find the equivalent code, or generator matrix permutation, that will deliver a minimal trellis with a fewer number of branches and/or states than any other minimal trellis for that code.

6.2 State Complexity

For a binary (N, K) linear block code C, the state complexity [35] of an N-section bit-level code trellis is measured by its state space dimension profile. The state space dimension profile can be defined as follows

$$\left(\rho_0, \rho_1, \rho_2, \ldots, \rho_N\right) \tag{6.1}$$

where for $0 \le i \le N$ th following holds

$$\rho_i = \log_2 |\Sigma_i(C)|. \tag{6.2}$$

The maximum value among the state space dimension of a particular code trellis can be defined as

$$\rho_{\max}(C) = \max_{0 \le i \le N}(\rho_i). \tag{6.3}$$

At this point, it is necessary to give a mathematical formulation of the state space of a N-section trellis for the above mentioned code. A generator matrix **G** is assumed for the code.

At time i, $0 \le i \le N$, the rows of **G** are divided into three disjoint subsets:

- $\boldsymbol{G}_i^p$ consists of those rows of $\boldsymbol{G}$ whose spans are contained in the interval $[1,i]$
- $\boldsymbol{G}_i^f$ consists of those rows of **G** whose spans are contained in the interval $[i+1,N]$.
- G_i^s consists of those rows of **G** whose active spans contain i.

Let A_i^p, A_i^f and A_i^s denote the subsets of information bits that correspond to the rows of $\boldsymbol{G}_i^p$, $\boldsymbol{G}_i^f$ and G_i^s respectively. The information bits in A_i^p do not affect the encoder outputs after time i, and hence they become the past with respect to time i. The information bits in A_i^f only affect the encoder outputs after time i. Since the active spans of the rows in G_i^s contain the time instant i, the information bits in A_i^s affect not only the past encoder outputs up to time i, but also the future encoder outputs beyond time i. It can be said that the information bits in A_i^s define a encoder state for the code C at time i. Each state is defined by a specific combination of the ρ_i information bits in A_i^s. The parameter ρ_i is the dimension of the state space $\Sigma_i(C)$.

The dimension of the state space can be defined as [21][22]

$$\rho_i(C) = \left|\boldsymbol{G}_i^s\right| = K - \left|\boldsymbol{G}_i^p\right| - \left|\boldsymbol{G}_i^f\right| = K - k\left(C_{0,i}\right) - k\left(C_{i,N}\right) \tag{6.4}$$

where

$$k\left(C_{0,i}\right) \quad and \quad k\left(C_{i,N}\right) \tag{6.5}$$

denote the dimensions of the past and future sub-codes with respect to time i.

Due to the fact that the parameters in **Equation 6.5** are non-negative, it can be said that

$$\rho_{\max}(C) \leq K. \tag{6.6}$$

It follows from the uniqueness of a state label that

$$\left|\Sigma_i(C)\right| \leq 2^{N-K} \tag{6.7}$$

Furthermore, it follows that

$$\rho_i \leq N - K \tag{6.8}$$

for $0 \leq i \leq N$.

Combining the above equations, results in the following bound:

$$\rho_{\max}(C) \leq N - K \tag{6.9}$$

From **Equations 6.6** and **6.9** it follows that the upper bound on the maximum state complexity is given by:

$$\rho_{\max}(C) \leq \min\{K, N - K\}. \tag{6.10}$$

This bound was first proved by J.K. Wolf [16]. In general, this bound is fairly loose. However, for cyclic codes, this bound gives the exact state complexity. For non-cyclic codes, tighter upper bounds have been obtained.

If the Viterbi algorithm is applied to the N-section trellis of a code, then the maximum

number of survivors and path metrics to be stored are both $2^{\rho_{max}(C)}$.

From this it follows that the state space dimension is a key measure of the trellis complexity and thus also the decoding complexity of a specific block code. It was also proven, although not repeated here, that a code C and its dual have the same state complexity[21],[22].

6.3 Branch Complexity

The branch complexity of an N-section trellis diagram for an (N, K) linear block code C is defined as the total number of branches in the trellis. This complexity determines the number of additions required in a trellis based decoding algorithm to decode a received sequence.

The branch complexity can easily be calculated by summing all the branches leaving active states.

6.4 Overall Complexity

In order to design more efficient and faster decoding algorithms for trellis based systems, methods have to be developed to limit the branch and state complexity as described in **Sections 6.3** and **6.4**.

It should be obvious that reducing state complexity is the most efficient complexity reduction process, since it implies that branches will also be reduced in the process. It thus serves a dual purpose.

6.5 Generator Matrix Permutations

In the previous paragraph, an introduction to trellis complexity was given. The most important parameter for the determination of trellis complexity is the state space

dimension $\rho_{\max}(C)$. The lower this value, the lower the actual decoding complexity involved [21][22].

This section will show how a good choice of a generator matrix **G** can influence the eventual decoding complexity. The method that is considered is a non-systematic search through all $n!$ permutations of generator matrixes, in order to find the equivalent code that delivers the smallest state space complexity.

In order to specify that a given permutation of the code symbols will actually deliver a noticeable change in the state space complexity and hence reduce the decoding complexity, a number of codes have to be examined.

As an example, a (7, 4)-Hamming code is considered. The code has the following generator matrix:

$$\boldsymbol{G} = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix} \tag{6.11}$$

This generator matrix has the following state space profile:

$$\rho(C) = (0,\ 1,\ 2,\ 3,\ 2,\ 2,\ 1,\ 0) \tag{6.12}$$

The trellis for this code has the following number of nodes:

$$N_{\Sigma} = \sum_{i=0}^{N} q^{\rho_i(C)} = 1+2+4+8+4+4+2+1 = 26 \tag{6.13}$$

The maximum state space for the code is:

$$\rho_{\max}(C) = 3 \tag{6.14}$$

In order to test the postulate that equivalent codes might lead to reduced state space complexity, the first and the last column of the generator matrix is swapped. This then

delivers a new generator matrix G', which in turn yield an equivalent code with generator matrix:

$$\boldsymbol{G}' = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix} \tag{6.15}$$

The state space is then found to be

$$\rho(C') = (0, \quad 1, \quad 2, \quad 3, \quad 3, \quad 2, \quad 1, \quad 0) \tag{6.16}$$

The trellis for this code has the following number of nodes:

$$N'_{\Sigma} = \sum_{i=0}^{N} q^{\rho_i(C)} = 1+2+4+8+8+4+2+1 = 30 \tag{6.17}$$

The maximum state space for the code is

$$\rho_{\max}(C') = 3 \tag{6.18}$$

As can be seen from **Equation 6.18** and **Equation 6.14** that the maximum state space dimension has remained unchanged. Notice however that the number of nodes present in the trellises has changed. The first code has 26 nodes in its trellis diagram, and the second equivalent code has 30. It is apparent that the first code will therefore be considerably simpler to decode.

Therefore, at this point it has to be said that the state space dimension is a very important factor in the complexity of the trellis, but that other factors such as number of nodes also have an effect.

In order to find out if any equivalent code to the generator matrix of **Equation 6.11** yields a smaller state space dimension, all the permutations of the generator matrix have to be considered.

There are in total $n! = 7! = 5040$ permutations of the generator matrix in **Equation 6.11** to consider. When this is done, the following results are obtained.

- There exist 2016 generator matrixes that have 26 nodes and a state space dimension of 3.
- There exist 3024 generator matrixes that have 30 nodes and a state space dimension of 3.

It is thus not possible to find a equivalent code by permutations of the generator matrix that has a trellis with less than 26 nodes, and a state space dimension of less than 3 for the (7,4)-Hamming code.

Next consider the (5, 3)-Code that Wolf has extensively examined in [16]. The parity check matrix of this code is given as:

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{pmatrix} \tag{6.19}$$

For this code, the state space is given as:

$$\rho(C) = \begin{pmatrix} 0, & 1, & 2, & 2, & 1, & 0 \end{pmatrix} \tag{6.20}$$

From this it follows that the maximum state space dimension is:

$$\rho_{\max}(C) = 2 \tag{6.21}$$

The number of nodes in the trellis can be computed as

$$N_{\Sigma} = \sum_{i=0}^{N} q^{\rho_i(C)} = 1 + 2 + 4 + 4 + 2 + 1 = 14 \tag{6.22}$$

If all the $n! = 5! = 120$ permutations are again considered, the following results are obtained:

- There exist 8 generator matrixes that have 10 nodes and a state space dimension of 1.
- There exist 32 generator matrixes that have 12 nodes and a state space dimension of 2.
- There exist 80 generator matrixes that have 14 nodes and a state space dimension of 2.

This implies that there is a generator matrix that delivers an equivalent code, with a state space dimension of 1 and just 10 nodes in the trellis diagram. The amount of decoding time and effort saved if this alternate trellis is used is significant.

Both the reduced trellis and the original trellis are shown below.

As can be seen from the figures above, the first trellis apparently has a much higher complexity than the second one. The decoder operating on the second trellis would be considerably less complex than the decoder of the first trellis.

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix} \tag{6.23}$$

The Parity Check Matrix of the second trellis is
From the above proof it can be said that there exist equivalent codes, obtainable through permutations of generator matrixes, which yield lower trellis complexity than all other equivalent codes.

The best way to find these optimum generator matrixes is the brute force permutation approach. This method evaluates all possible permutations of the generator matrix under investigation, and selects the best one.

This approach is however in most cases not a very viable approach. Many codes have a fairly large value for n. Assuming n to be only 64, which is in fact still small, a total of $n! = 64 = 1.267E89$ permutations have to be considered. For a Reed-Solomon code of n = 512 the value becomes unwieldy high.

There is no doubt that a reduction in trellis complexity is an essential contribution for soft-decision maximum likelihood decoding of block codes. In the example above, the decoding time may be shortened considerably, by employing minimum state trellis structures with fewer nodes and correspondingly fewer branches.

The following simulation results are needed to prove that the performance of a specific code does not decrease if a reduced trellis is used for the decoding. It would be of no use if a trellis could be simplified, but at the cost of performance. The chapter uses a (5,3)-Code as described in **Chapter 9**.

6.6 Decoding in the Trellis Diagrams

The following simulation results are presented in order to show that the performance of a specific code does not decrease if a reduced trellis is used for the decoding. It would

be of no use if a trellis could be simplified, but at the cost of performance.

The code under investigation is the (5,3)-code [16]. This code was used in the previous section to derive and construct a reduced trellis. The parity check matrix of the original code is given by:

$$\boldsymbol{H}_1 = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{pmatrix} \tag{6.24}$$

This can then be transformed into the generator matrix of this code. See **Appendix B** for a detailed description of this process. The generator matrix is given by:

$$\boldsymbol{G}_1 = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix} \tag{6.25}$$

Using the process described in the previous chapter, a simplified trellis can be found by using the following parity check matrix:

$$\boldsymbol{H}_2 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix} \tag{6.26}$$

Again, this parity check matrix has to be transformed into a generator matrix in order to obtain the codewords needed for decoding. The process of obtaining the generator matrix is a bit more involved, since the code is no longer systematic and any attempt to transform it into a systematic code, will destroy the inherent properties of the code, nullifying the ability of the matrix to produce a reduced trellis.

The following property of matrices is used to find the generator matrix.

$$\boldsymbol{G} \cdot \boldsymbol{H}^T = \underline{\boldsymbol{0}} \tag{6.27}$$

The above equation delivers 3 equations with 15 unknowns. But due to the nature of the

equations, all unknowns that cannot be determined can be chosen randomly.

$$\boldsymbol{G}_2 = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \end{pmatrix} \tag{6.28}$$

These two parity check matrices will produce the trellises given on the next page. The diagrams were found using simulation software written in C++. The code is partially reproduced in Appendix G, but a full version can be obtained from the authors.

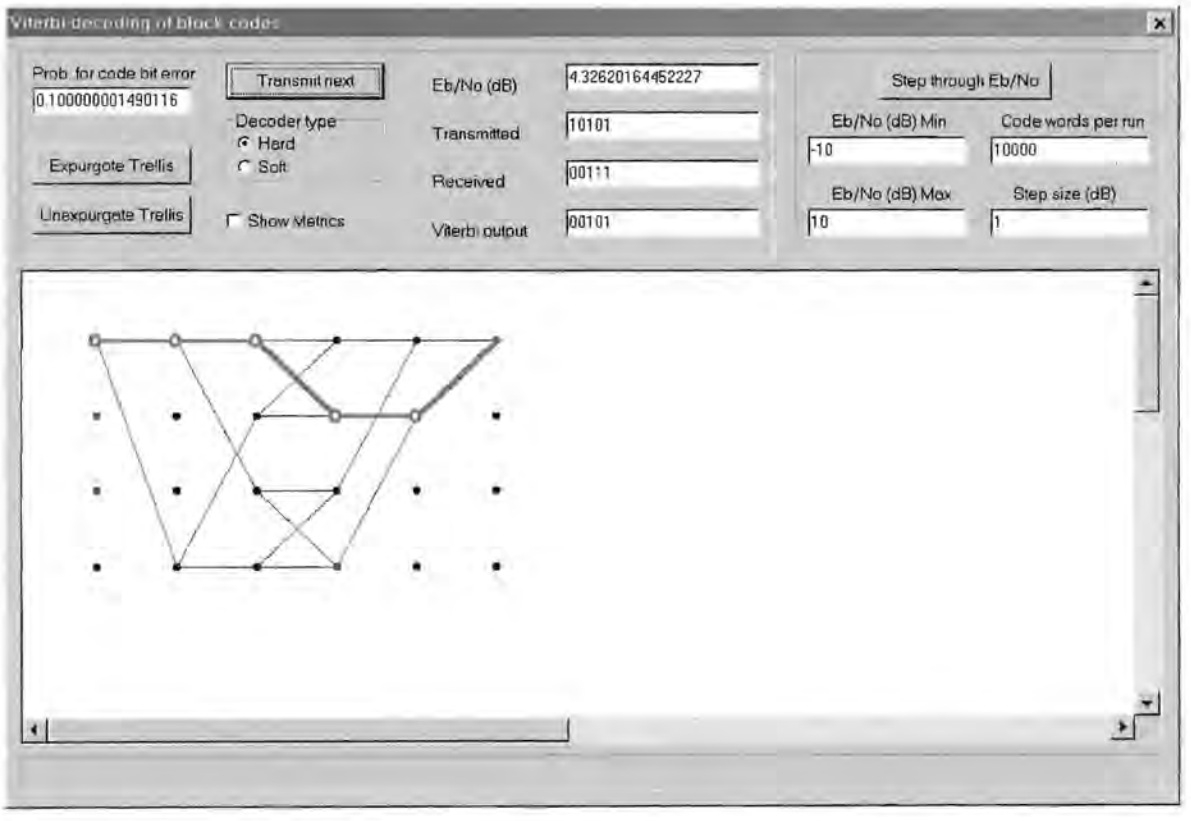

Figure 42 (5,3) Code Trellis for Parity Check matrix H_1 Produced by Trellis Simulation Software

As can be seen from the above figure, SDML-decoding and HD-decoding can be obtained with the software, as well as expurgation. The transmitted and received vectors are displayed together with the output of the Viterbi decoder and the channel E_{bit}/N_0.

The path on which the Viterbi decoder has decided on is highlighted in green - the so-called survivor path. Branches with weight 0 are marked in blue and branches with weight 1 in red.

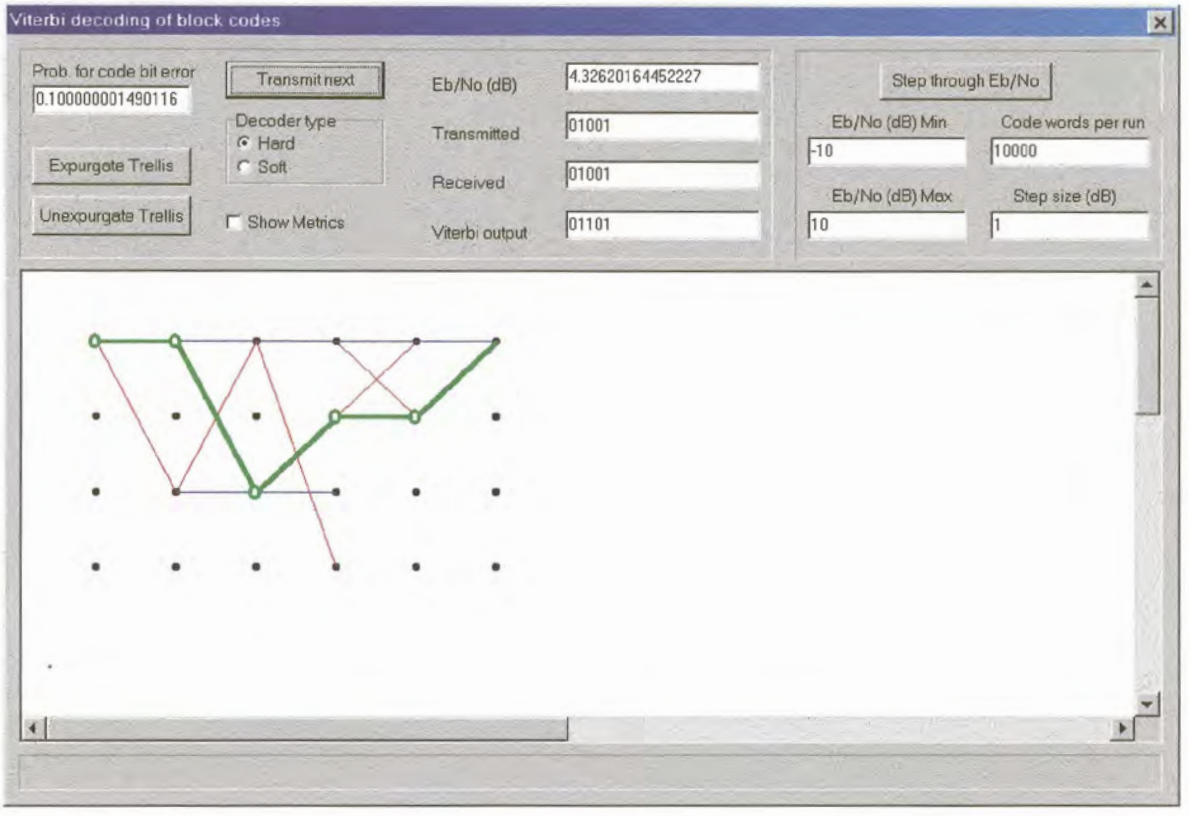

Figure 43 (5,3) code Trellis for Parity Check Matrix H_2 produced by Trellis Simulation Software

The above figure shows the trellis for the second code. Due to the fact that the reduced trellis has one less state than the original trellis, the expurgation procedure cannot cope with the reduced number of states (one less than in the previous example). A new approach has to be employed to remove the paths into a state from which no branches emanate. This procedure was written after this screen dump was produced.

The above two figures prove that the reduced trellis is able to decode the codewords properly. What has to be demonstrated however, is that the overall performance of the reduced trellis equals the performance of the original trellis under identical channel conditions.

6.7 Bit Error Rate Calculations

The two trellises in the preceding paragraph were used in the simulation software to find the BER graphs for both the reduced and normal trellises.

The following configuration files have to be set up for the simulator in order to calculate the BER.

The first configuration file for the original trellis is called "*Original (5,3) Code.bcc*" and contains the parameters and matrix in order for the simulator to compute the trellis and its accompanying weights.

Message_length_(k):
3

Code_length_(n):
5

Galois_field_prime_value:
2

Generator_matrix:
1 0 0 1 1
0 1 0 1 0
0 0 1 0 1

Table 9 Original (5,3) Code.bcc File

The second configuration file for the original trellis is called "*Original (5,3) Code Trellis.btf*" and is created by the simulator, but it can also be entered if the above data is not known.

Number_of_states_in_the_trellis:
4

Depth_of_the_trellis:

6

Active_nodes_in_the_trellis

1	*1*	*1*	*1*	*1*	*1*
0	*0*	*1*	*1*	*1*	*0*
0	*0*	*1*	*1*	*0*	*0*
0	*1*	*1*	*1*	*0*	*0*

Branch0_destinations:

0	*0*	*0*	*0*	*0*	*-1*
-1	*-1*	*1*	*1*	*-1*	*-1*
-1	*-1*	*2*	*-1*	*-1*	*-1*
-1	*3*	*3*	*-1*	*-1*	*-1*

Branch1_destinations:

3	*2*	*1*	*-1*	*-1*	*-1*
-1	*-1*	*0*	*-1*	*0*	*-1*
-1	*-1*	*3*	*0*	*-1*	*-1*
-1	*1*	*2*	*1*	*-1*	*-1*

Branch0_weights:

-1	*-1*	*-1*	*-1*	*-1*	*-1*
-1	*-1*	*-1*	*-1*	*-1*	*-1*
-1	*-1*	*-1*	*-1*	*-1*	*-1*
-1	*-1*	*-1*	*-1*	*-1*	*-1*

Branch1_weights:

1	*1*	*1*	*-1*	*-1*	*-1*
-1	*-1*	*1*	*-1*	*1*	*-1*
-1	*-1*	*1*	*1*	*-1*	*-1*
-1	*1*	*1*	*1*	*-1*	*-1*

Table 10 Original (5,3) Code Trellis.btf File

The first configuration file for the reduced trellis is called "Reduced *(5,3) Code.bcc*" and contains the parameters and matrix in order for the simulator to compute the trellis and

its accompanying weights.

Message_length_(k):
3

Code_length_(n):
5

Galois_field_prime_value:
2

Generator_matrix:
1 0 1 0 1
1 0 1 1 0
1 1 0 1 1

Table 11 Reduced (5,3) Code.bcc File

The second configuration file for the original trellis is called "Reduced *(5,3) Code Trellis.btf*" and is created by the simulator, but it can also be entered if the above data is not known.

Number_of_states_in_the_trellis:
4

Depth_of_the_trellis:
6

Active_nodes_in_the_trellis

1	*1*	*1*	*1*	*1*	*1*
0	*0*	*0*	*1*	*1*	*0*
0	*1*	*1*	*0*	*0*	*0*
0	*0*	*0*	*0*	*0*	*0*

Branch0_destinations:

0	*0*	*0*	*0*	*0*	*-1*

-1	*-1*	*-1*	*1*	*-1*	*-1*
-1	*2*	*-1*	*-1*	*-1*	*-1*
-1	*-1*	*-1*	*-1*	*-1*	*-1*

Branch1_destinations:

2	*2*	*-1*	*1*	*-1*	*-1*
-1	*-1*	*-1*	*0*	*0*	*-1*
-1	*0*	*1*	*-1*	*-1*	*-1*
-1	*-1*	*-1*	*-1*	*-1*	*-1*

Branch0_weights:

-1	*-1*	*-1*	*-1*	*-1*	*-1*
-1	*-1*	*-1*	*-1*	*-1*	*-1*
-1	*-1*	*-1*	*-1*	*-1*	*-1*
-1	*-1*	*-1*	*-1*	*-1*	*-1*

Branch1_weights:

1	*1*	*-1*	*1*	*-1*	*-1*
-1	*-1*	*-1*	*1*	*1*	*-1*
-1	*1*	*1*	*-1*	*-1*	*-1*
-1	*-1*	*-1*	*-1*	*-1*	*-1*

Table 12 Reduced (5,3) Code Trellis.btf File

For a complete description of the files the reader is referred to the help file included with the software. The above files are just given so that the reader can reproduce the results obtained with ease.

The simulator is set up to use a AWGN channel, and the channel energy to noise ratio E_{bit}/N_0 is varied from -10 dB to 13 dB. Simulations were run until a hundred errors were found for each increment of E_{bit}/N_0.

The results that were obtained were stored in a file for each case. The files were computed in order to obtain the desired BER. For the purpose of comparison, the *Block Error,* also known as *Word Error Rate,* is displayed in the graphs. By doing so, the statistical dependance is removed (as explained in **Chapter 4**) and a pure comparison

can be made.

The numerical results are also repeated below for both cases.

-10.000000	*0.769231*
-9.000000	*0.625000*
-8.000000	*0.769231*
-7.000000	*0.588235*
-6.000000	*0.588235*
-5.000000	*0.454545*
-4.000000	*0.555556*
-3.000000	*0.384615*
-2.000000	*0.303030*
-1.000000	*0.303030*
0.000000	*0.294118*
1.000000	*0.270270*
2.000000	*0.147059*
3.000000	*0.119048*
4.000000	*0.068966*
5.000000	*0.078740*
6.000000	*0.046512*
7.000000	*0.018553*
8.000000	*0.019455*
9.000000	*0.005388*
10.000000	*0.010834*
11.000000	*0.001283*
12.000000	*0.000764*
13.000000	*0.000126*

Table 13 Original Results.res File

-10.000000	0.757576
-9.000000	0.699301
-8.000000	0.588235
-7.000000	0.591716
-6.000000	0.540541

-5.000000	0.469484
-4.000000	0.450450
-3.000000	0.369004
-2.000000	0.352113
-1.000000	0.298507
0.000000	0.247525
1.000000	0.223714
2.000000	0.151976
3.000000	0.140056
4.000000	0.105708
5.000000	0.065876
6.000000	0.050429
7.000000	0.027541
8.000000	0.017794
9.000000	0.009327
10.000000	0.004335
11.000000	0.001692
12.000000	0.000590
13.000000	0.000160

Table 14 Reduced Results.res File

On the next page the results for the original and reduced trellis performance are superimposed on one graph.

From the figure below, it can be seen that the error performance of the two trellises are practically identical, thus verifying the statements made and results achieved in the previous paragraphs. The assumption made here is that if there is no performance degradation for a small code, then there will be no performance degradation for larger codes.

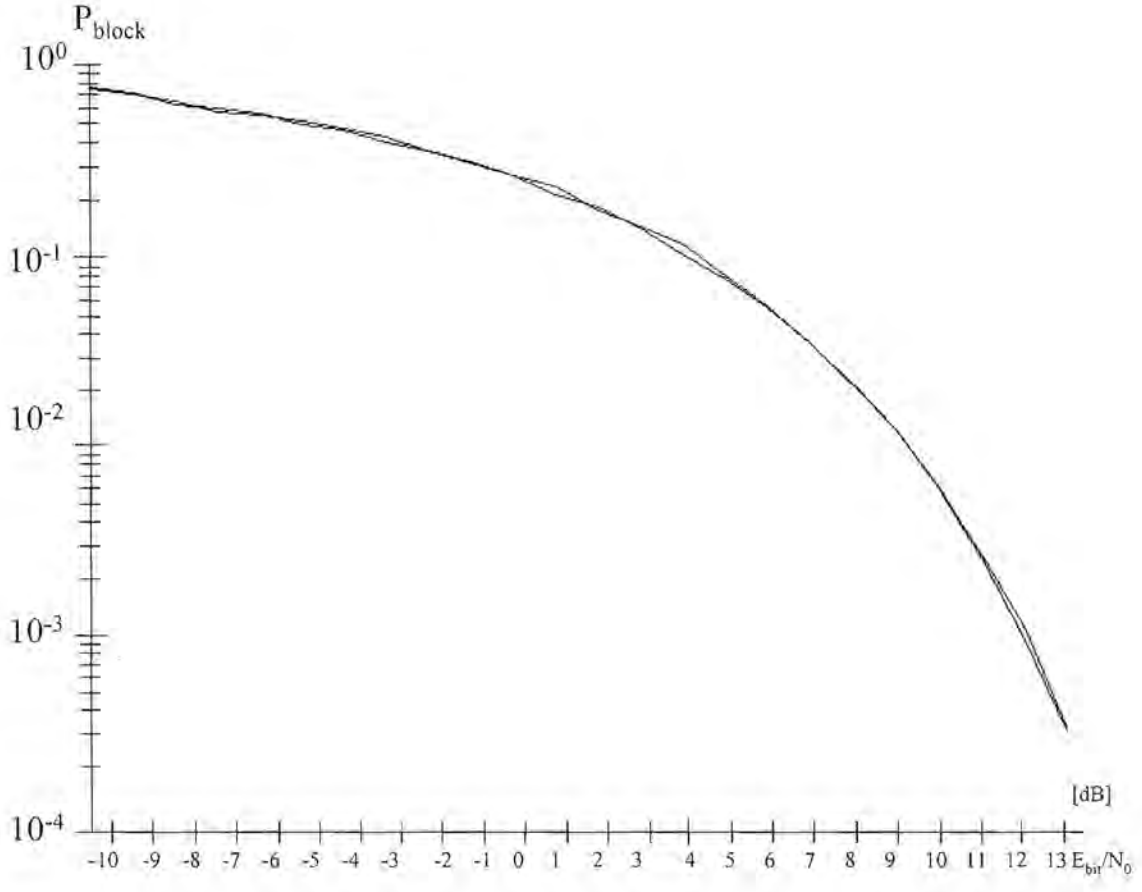

Figure 44 Comparison of Reduced and Original Block Error Rates

6.8 Algorithms for the Determining Minimal Trellis Diagram Representations

In **Section 6.5** an algorithm which used a brute force search in order to find the minimal trellis representation from the permutations of the generator matrix was presented. This algorithm is of course ideal when small codes are considered, since it will most certainly find the minimum trellis representation. However, as soon as large

codes, or even just medium length codes are considered, the algorithm becomes intractable. This stems from the fact that $n!$ permutations have to be considered in order to find the permutation of the generator matrix which will lead to a minimal trellis construction. This creates the need for better algorithms, some of which will be discussed here.

6.8.1 Terminated Brute Force Search Algorithm

As mentioned above, the brute force search algorithm becomes unviable for large codes. An obvious alternative would be to consider as many permutations of the generator matrix as possible. This search could be limited by a certain amount of permutations, time or processing power of the platform used. This however boils down to luck and is not a very good approach.

However, the initial idea of a terminated brute force search does however hold merit. All which needs to be done is to determine good end of search criteria. A good option for doing this presents itself in the form of bounds of the state space profile. As described in **Section 6.5**, a generator matrix which would deliver a minimal trellis representation can be identified from the state space profile and the amount of nodes the trellis would produce. It is thus necessary to establish a bound to the state space profile. This would allow the brute force search to continue until the bound is reached or approached to a certain extent.

Consider a linear block code (n,k) C with code symbols from the symbol alphabet $GF(q)$. $I = \{0,1,2,...,n - 1\}$ is the set of the indexes i of the code symbols c_i. From this J can be defined as being a subset of I such that $J \subseteq I$ with $J \leq I$.

A partial code is defined as a code having undergone a sub-dividing operation. This operation T_J involves setting all code symbols c_i to 0 whose indexes are not contained in J but only in I.

$$T_J(c) = \begin{cases} c_i & for \quad I \in J \\ 0 & for \quad I \notin J \end{cases} \tag{6.29}$$

The partial code $T_J(C)$ i found by applying the operation above on every code word.

$$T_J(C) = \left\{ T_J(c) | c \in C \right\} \tag{6.30}$$

Furthermore, a sub-code C_J is defined as the subset of code words c of C whose components at the indexes $(I - J)$ are equal to zero.

$$C_J = \left\{ c \in C | c_i = 0 \quad for \quad i \notin J \right\} \tag{6.31}$$

An example will be given here to illustrate the principles discussed above. Consider again the (7,4)-Hamming code with generator matrix:

$$G = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix} \tag{6.32}$$

This generator matrix yields the following 16 code words presented in tabular format below:

i	c	i	c
(0 0 0 0)	(0 0 0 0 0 0 0)	(1 0 0 0)	(0 0 1 1 1 0 0)
(0 0 0 1)	(1 0 1 0 1 1 0)	(1 0 0 1)	(1 0 0 1 0 1 0)
(0 0 1 0)	(1 0 1 0 0 0 1)	(1 0 1 0)	(1 0 0 1 1 0 1)
(0 0 1 1)	(0 0 0 0 1 1 1)	(1 0 1 1)	(0 0 1 1 0 1 1)
(0 1 0 0)	(1 1 0 0 1 0 0)	(1 1 0 0)	(1 1 1 1 0 0 0)
(0 1 0 1)	(0 1 1 0 0 1 0)	(1 1 0 1)	(0 1 0 1 1 1 0)
(0 1 1 0)	(0 1 1 0 1 0 1)	(1 1 1 0)	(0 1 0 1 0 0 1)
(0 1 1 1)	(1 1 0 0 0 1 1)	(1 1 1 1)	(1 1 1 1 1 1 1)

Table 15 Code Words of the (7,4)-Hamming Code in $GF(2)$

As an example, choose J = {1, 3, 6}. If the sub-division operation is performed on the code, then the partial code is found to be the following set of code words.

(0 0 0 0 0 0 0)
(0 0 0 0 0 0 1)
(0 1 0 0 0 0 0)
(0 1 0 0 0 0 1)
(0 0 0 1 0 0 0)
(0 0 0 1 0 0 1)
(0 1 0 1 0 0 0)
(0 1 0 1 0 0 1)

Table 16 Partial Code Words $T_J(C)$ with $J = \{1, 3, 6\}$

When the selection criteria for the sub-codes are applied, the following result is obtained.

(0 0 0 0 0 0 0)
(0 1 0 1 0 0 1)

Table 17 Sub-Codes C_J with $J = \{1, 3, 6\}$

Given the above definitions, it is now possible to define the dimensional distribution:

$$K(C) = \left\{k_i(C) \quad with \quad 0 \le i \le n\right\} \tag{6.33}$$

where

$$k_i(C) = \max_J \left\{k(C_J) \quad with \quad |J| = i\right\} \quad for \quad 0 \le i \le n \tag{6.34}$$

From the principles of duality between $T_J(C)$ and C_J, it follows that the inverse dimensional distribution can be given as:

$$\widetilde{K}(C) = \left\{ \widetilde{k}_i(C) \quad with \quad 0 \le i \le n \right\} \tag{6.35}$$

where

$$\widetilde{k}_i(C) = \min_J \left\{ k(T_J(C)) \quad with \quad |J| = i \right\} \quad for \quad 0 \le i \le n \tag{6.36}$$

The information above can now be used to formulate a terminated brute force search algorithm. To start off, a (15,7)-BCH code is chosen as an example. The generator matrix of the code is:

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix} \tag{6.37}$$

Just as in **Section 6.5**, permutations of the generator matrix are examined in order to find the trellis diagram with the least amount of nodes, and the lowest state space profile. As no criteria are known at this stage for the termination of the algorithm, a run lasting eight days (Pentium III processor) was performed. In this time, a total of 2^{32} permutations of the generator matrix were considered, but this only constitutes about 0.08% of the total $n!$ = 15! = 1307674368000 possibilities. The results obtained are presented in tabular form below.

N_Σ	$\rho_{max}(C) = 5$	$\rho_{max}(C) = 6$	$\rho_{max}(C) = 7$
206	3600	-	-
214	86712	-	-
222	320904	-	-
230	476976	-	-
238	2989824	21600	-
246	-	295728	-
254	6473592	818472	-
262	-	1870896	-
270	-	9492504	-
278	-	496320	-
286	-	24652824	-
294	-	4688640	-
302	-	22750126	-
318	-	63860208	-
326	-	6441624	-
334	-	34550356	-
350	-	132288532	-
382	-	165657036	-
390	-		7416529
398	-		34507084
414	-		90973995
446	-		217439332
510	-	-	245168364

Table 18 Results for N_Σ and $\rho_{max}(C)$ for (15,7)-BCH code after 2^{32} permutations

The figure below represents the data obtained in graphical format.

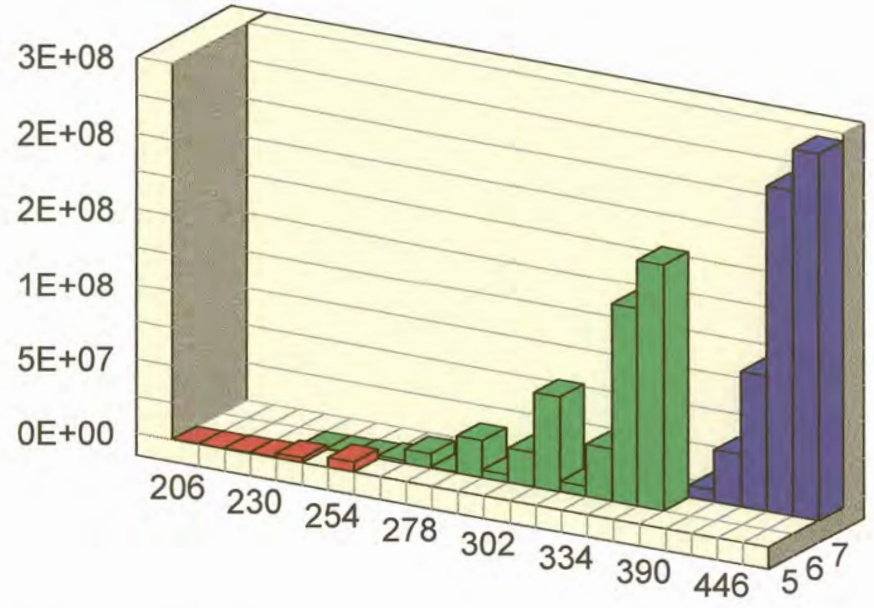

Figure 45 Graphical representation of N_Σ and ρ(C) for (15,7)-BCH code

Forney calculated a lower bound for the state space profile which will be used a criterion for the termination of the brute force search[30] [31]. The lower bound for the state space profile is given by:

$$\rho_{\max}(C) \geq \tilde{k}(C) - k(C) \tag{6.38}$$

Applied to the (15,7)-BCH code, the following results are obtained. The dimensional distribution is found to be:

$$k(C) = \{0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 2 \quad 2 \quad 3 \quad 4 \quad 4 \quad 5 \quad 6 \quad 7\} \tag{6.39}$$

Dual to this, the inverse dimensional distribution is:

$$\tilde{k}(C) = \{0 \quad 1 \quad 2 \quad 3 \quad 3 \quad 4 \quad 5 \quad 5 \quad 6 \quad 6 \quad 6 \quad 7 \quad 7 \quad 7 \quad 7 \quad 7\} \tag{6.40}$$

Using the definition of the Forney bound on the state space profile, the following result is obtained:

$$\rho_{\max}(C) \geq \{0 \quad 1 \quad 2 \quad 3 \quad 3 \quad 3 \quad 4 \quad 4 \quad 4 \quad 4 \quad 3 \quad 3 \quad 3 \quad 2 \quad 1 \quad 0\} \geq 4 \quad \textbf{(6.41)}$$

Also from Forney [30] [31] a lower bound for the number of nodes in the trellis diagram can be calculated from:

$$N_{\Sigma} \geq \sum_{i=0}^{n} q^{\tilde{k}_i(C)-k_i(C)} = 1+2+4+8+8+16+16+16+16+8+8+8+4+2+1 \geq 126 \quad \textbf{(6.42)}$$

From the above two bounds and the results calculated, it is possible to specify that the minimum values for N_{Σ} and ρ(C) lie in the region of:

$$4 \leq \rho(C) \leq 5 \quad \textbf{(6.43)}$$

and

$$126 \leq N_{\Sigma} \leq 206 \quad \textbf{(6.44)}$$

From the above analysis it can be said that the double lower bound criteria for the termination of the brute force search provides good approximated results for the minimal trellis representation. In the example above, the state space profile was approximated as being 5, although, with all probability, it is in fact 4. The number of nodes which the minimal trellis would contain was approximated as being 206, although there are most probably only 126 nodes present in the minimal trellis diagram. As mentioned earlier, a full brute search would take 100 days to complete. With the test run which was done, it took 8 days to accumulate enough data to use for an analysis. However, with the two bounds which can now be used to terminate the brute force search, it is quite possible, that a good approximation can be found within a few hours.

6.8.2 Systematic Search Algorithm

The following search algorithm will attempt to arrange the generator matrix in such a way, so that the probability that this specific permutation of the generator matrix will produce a minimal trellis diagram, is increased. The whole algorithm is based on a statement by Forney [30]:

"*In order to minimise the state space complexity, the dimension buildup has to increase as fast as possible.*"

The dimension buildup is given below:

$$b_i(C) = k(C_J) = k - k(T_{I-J}(C)) = k - RANK(G_{I-J}) \tag{6.45}$$

with

$$J = \{0,1,\ldots,i-1\} \tag{6.46}$$

and

$$k = k_i(C) + \tilde{k}_{n-i}(C) \tag{6.47}$$

In order for the dimension buildup to increase rapidly, it is necessary for the rank of the sub-matrices $\boldsymbol{G}_{I,J}$ to decrease as fast as possible. If this is not possible, then the increase of the rank should be kept as small as possible.

The most elegant way to accomplish this, is to arrange the columns of the generator matrix from right to left such that the sub-matrixes $\boldsymbol{G}_{I,J}$ with $I - J = \{i, i+1, \ldots, n\}$ for $i = n, n-1, \ldots, 1, 0$ do not increase in rank with each added column.

The algorithm can be formalised as follows:

9. Choose a random column of the generator matrix as being $\hat{g}_n$.
10. Set the counter $i = n - 1$

- Search through the remaining columns g_j in order to find the next $\hat{g}_i$ so that:

$$Rank\left(g_i, \hat{g}_{i+1}, \hat{g}_{i+2}, \ldots, \hat{g}_{n-1}, \hat{g}_n\right) = Rank\left(\hat{g}_{i+1}, \hat{g}_{i+2}, \ldots, \hat{g}_{n-1}, \hat{g}_n\right) \quad \textbf{(6.48)}$$

- If at least one such a column exists, use it as $\hat{g}_i$ for the new generator matrix $\hat{\mathbf{G}}$.
- If $Rank\left(\hat{g}_{i+1}, \hat{g}_{i+2}, \ldots, \hat{g}_{n-1}, \hat{g}_n\right) > 1$ use any remaining column of $\mathbf{G}$ as $\hat{g}_i$.
- Decrease i by one.
- Repeat as long as $Rank\left(g_i, \hat{g}_{i+1}, \hat{g}_{i+2}, \ldots, \hat{g}_{n-1}, \hat{g}_n\right) < k$.

3. Use the remaining columns of the generator matrix as $\hat{g}_1, \hat{g}_2, \hat{g}_{i-1}$ *and* $\hat{g}_i$ of $\hat{\mathbf{G}}$.

The results obtained for 4 different (11,7)-Hamming code are given below. A comparison is made between the complete brute force search, and the systematic search algorithm. The lower bound is also given in the table.

Code	Given **G** Matrix	Lower Bound	Syst. Search	Brute Force
(11,7)-Hamming A	ρ=4 N_Σ=94	ρ=3 N_Σ=38	ρ=4 N_Σ=84	ρ=3 N_Σ=38
(11,7)-Hamming B	ρ=4 N_Σ=94	ρ=2 N_Σ=28	ρ=4 N_Σ=94	ρ=2 N_Σ=28
(11,7)-Hamming C	ρ=4 N_Σ=94	ρ=3 N_Σ=54	ρ=4 N_Σ=74	ρ=3 N_Σ=54

Table 19 Comparison of Results for N_Σ and $\rho(C)$ for (11,7)-Hamming codes.

As can be seen from the results above, the systematic search does not deliver wonderful results, but it can be seen that a reduction in the trellis complexity does occur, requiring only a minimal amount of computations. Another test was done with three (15,5)-BCH

codes. In this test run, the brute force search was terminated according to the criteria presented in the previous section.

Code	Given **G** Matrix	Lower Bound	Syst. Search	Term. B. F.
(15,5)-BCH A	ρ=5 N_Σ=254	ρ=4 N_Σ=126	ρ=4 N_Σ=134	ρ=4 N_Σ=134
(15,5)-BCH B	ρ=7 N_Σ=510	ρ=4 N_Σ=126	ρ=6 N_Σ=254	ρ=5 N_Σ=206
(15,5)-BCH C	ρ=4 N_Σ=158	ρ=3 N_Σ=86	ρ=4 N_Σ=138	ρ=4 N_Σ=110

Table 20 Comparison of Results for N_Σ and $\rho_{max}(C)$ for (15,5)-BCH codes.

From the above table of results, it can be seen, that when it is impossible to perform a complete brute force search, the systematic search algorithm compares very favorably with the terminated brute force search. In its favor as well is the fact the systematic algorithm required three minutes to find a solution, whereas the terminated brute force search was running for 9 hours.

It will now be attempted to optimise the systematic search algorithm.

6.8.3 Optimised Systematic Search Algorithm

After analysing the systematic search algorithm, two apparent problems were seen, which limited the effectiveness of the search algorithm.

- With each arrangement of the columns in order to limit the rank of the sub-matrices, the column which increases the rank the least should be chosen. Most of the time, especially for large codes, a choice has to be made between two or more columns. It is possible that a wrong choice at this point could place a

drastic limit on the performance of the algorithm.

- This is especially true for the initial choice of the starting column. This is done arbitrarily, since at this point the rank of all the columns are 1.

A solution to this would be to consider all permutations again. This would mean that each one of the 15 columns should be chosen as a starting column for the new matrix. The same procedure as in the previous section is applied, but when a choice has to be made between 2 columns which would not increase the rank of the sub-matrix much, it is not made randomly. All of the columns which would provide such a solution are considered. Of course, the further on the algorithm is, the more permutations there are to consider. If the codes are too large, this algorithm can be applied together with the lower bounds on the state space profile and number of nodes in the minimal trellis. The following results were obtained for the three (11,7)-Hamming codes and three (15,5)-BCH codes.

Code	Given **G** Matrix	Lower Bound	Syst. Search	Brute Force
(11,7)-Hamming A	ρ=4 N_Σ=94	ρ=3 N_Σ=38	ρ=3 N_Σ=38	ρ=3 N_Σ=38
(11,7)-Hamming B	ρ=4 N_Σ=94	ρ=2 N_Σ=28	ρ=2 N_Σ=28	ρ=2 N_Σ=28
(11,7)-Hamming C	ρ=4 N_Σ=94	ρ=3 N_Σ=54	ρ=3 N_Σ=58	ρ=3 N_Σ=54

Table 21 Comparison of Results for N_Σ and ρ(C) for (11,7)-Hamming codes obtained with the Optimised Search Algorithm

Code	Given G Matrix	Lower Bound	Syst. Search	Term. B. F.
(15,5)-BCH A	ρ=5 N_Σ=254	ρ=4 N_Σ=126	ρ=4 N_Σ=134	ρ=4 N_Σ=134
(15,5)-BCH B	ρ=7 N_Σ=510	ρ=4 N_Σ=126	ρ=6 N_Σ=218	ρ=5 N_Σ=206
(15,5)-BCH C	ρ=4 N_Σ=158	ρ=3 N_Σ=86	ρ=4 N_Σ=114	ρ=4 N_Σ=110

Table 22 Comparison of Results for N_Σ and ρ(C) for (15,5)-BCH codes obtained with the Optimised Search Algorithm

As can be seen from the above results, the optimised systematic search approaches the terminated brute force search and the lower bounds. It however requires far fewer computations in order to find a solution.

The optimised systematic search provides an elegant algorithm in order to find a minimal trellis approximation with the minimum amount of computations required.

Chapter 7

Conclusion

7.1 Conclusion

The main objective of this dissertation was to show that it is indeed possible to utilize decoding techniques traditionally reserved for convolutional code decoding for the decoding of block codes. This main goal was extended even further when it was attempted to decode one of the most powerful families of block codes, namely the Reed-Solomon codes, with the Viterbi algorithm. This opens the way for endless possibilities employing block codes and convolutional codes combined together into one encoding scheme.

Apart from this main objective, various techniques for trellis construction were gathered, developed and evaluated. This should prove helpful in further research projects to follow on this dissertation. Again, the basic idea of trellis construction was extended to include non-binary Reed-Solomon codes. A novel technique was found which utilizes the topological structure of the Reed-Solomon codes in order to simplify and streamline the trellis construction procedure. This ensures that trellis construction does not have to be hard-coded in a hardware implementation. A re-programmable Reed-Solomon trellis construction integrated circuit can be developed, which can then be used in conjunction with the standard Viterbi algorithm.

Early on during the research it became clear that trellis size remains a stumble block of trellis decoders for block codes. A method needed to be devised in order to reduce the trellis complexity. A technique was found which reduces trellis complexity significantly, namely the manipulation of the generator matrix. It facilitates the Viterbi decoding of large block codes by reducing trellis complexity.

All the research was backed up with mathematical and simulation results. This proves that maximum likelihood decoding employing techniques such as Viterbi, SOVA and MAP are a viable means of decoding block codes of considerable size.

A large library of simulation software was written, which allowed for various simulations to be run. Amongst others, a Viterbi decoder and a trellis construction tool was written. All these software modules were kept generic, so that they could be applied to both binary and non-binary codes.

Another direct contribution made via this dissertation is the novel topological trellis construction technique for Reed-Solomon codes. Additionally, it was shown that Viterbi decoding is viable for block codes, even for the most complex of codes, such as the non-binary Reed-Solomon code family. Simulation software was produced which can be used in the development of numerous novel coding techniques employing both block coders and convolutional decoders. This was possible before, but now it is possible to have just one standard maximum likelihood decoder. There are also possibilities for combining the trellises of both block and convolutional codes, creating even more opportunities for successful hybrid systems.

Design and development of suitable hardware solutions for the trellis decoding techniques in this dissertation is left as a possible future research project.

It can be said, that the "Holy Grail" of soft decision block code decoding has as yet not been found, but this work should in itself be a contribution to the quest and serve as a valuable platform for further research.

References

The following references were used in the process of creating this dissertation. The work of the different authors is acknowledged in this section.

[1] M Honary, *Trellis Decoding of Block Codes*, Kluwer Academic Publishers 1998.

[2] LR Bahl, JCocke, F Jelinek and J Raviv,. "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate", *IEEE Transactions on Information Theory*, Vol. 20 pp 284-287.

[3] RE Blahut, *Theory and Practice of Error Control Codes*, Addison Wesley 1984.

[4] S Bate, B Honary and PG Farell, "Error Control Techniques Applicable to HF channels", *IEEE Proceedings*, Vol. 136, No. 1, February 1989, pp. 57-63.

[5] GD Forney, "Convolutional Codes I: Algebraic Structure", *IEEE Transactions on Information Theory*, Vol. 16, 1970, pp. 720-738.

[6] R Johannesson and Z Wan, "A Linear Algebra Approach to Minimal Convolutional Encoders", *IEEE Transactions of Information Theory*, Vol. 39, 1993, pp. 1219-1233.

[7] T Kasami and T Fujiwara, "On Complexity of Trellis Structure of Linear Block Codes", *IEEE Transactions on Information Theory*, Vol. 39, 1993, pp 1057-1064.

[8] FJ MacWilliams and NJ Sloane, "The Theory of Error Correcting Codes", Amsterdam: North-Holland, 1983.

[9] DJ Muder, "Minimal Trellises for Block Codes", *IEEE Transactions on Information Theory*, Vol. 34, 1988, pp. 1049-1053.

[10] JG Proakis, *Digital Communications*, New York, McGraw and Hill, 1989.

[11] VR Sidorenko and VV Zyablov, "Bounds on Complexity of Trellis Decoding of Linear Codes", *Proceedings of the Sixth Joint Swedish-Russian International Workshop on Information Theory*, 1993, pp. 164-168.

[12] VR Sidorenkoand VV Zyablov, "Decoding of Convolutional Codes using Syndrome Trellsis", *Proceedings of the Sixth Joint Swedish-Russian International Workshop on Information Theory*, 1993, pp. 46-50.

[13] B Sklar, *Digital Communications*, Englewood Cliffs, New York, Prentice Hall, 1988.

[14] AJ Viterbi, "Error Bounds for Convolutional Codes and an Asymtotically Optimum Decoding Algorithm", *IEEE Transactions on Information Theory*, Vol. 13, 1967, pp. 260-269.

[15] AJ Viterbi and JK Omura, "Principles of Digital Communication and Coding", *New York, McGraw and Hill*, 1979.

[16] JK Wolf, "Efficient Maximum Likelihood Decoding of Linear Block Codes using a Trellis", *IEEE Transactions on Information Theory*, Vol. 24, 1978, pp. 76-80.

[17] R Rotman, *Galois Theory*, Berlin, Springer Verlag, 1990.

[18] S Lin, T Kasami and T Fujiwara "Trellises and Trellis based Decoding Algorithms for Linear Block Codes", *Kluwer Academic Publishers*, 1998.

[19] S Lin and DJ Costello, *Error Control Coding: Fundamentals and their Applications*, Prentice Hall, 1983.

[20] IS Reed and G Solomon, "Polynomial Codes over Certain Finite Fields", *SIAM Journal of Applied Mathematics*, Vol. 8, 1960, pp. 300-304

[21] SB Wicker and VK Bhargava, *Reed-Solomon Codes and their Applications*, IEEE Press, New York, 1994.

[22] JK Wolf, "On Codes derivable from the Tensor Product of Check Matrices", *IEEE*

Transactions on Information Theory, Vol. 11, April 1965, pp. 281-284, April 1965.

[23] CE Shannon, "The Zero Capacity of Noisy Channels", *IRE Transactions on Information Theory,* vol. 2, September 1956, pp. 8-19.

[24] E Biglieri, D Divsalar, PJ McLane and MK Simon, *Introduction to Trellis-Coded Modulation with Applications*, MacMillan Pull. Co, New York, 1991.

[25] RJ McEliece, *Finite Fields for Computer Scientists and Engineers,* Boston: Kluwer Academic, 1987.

[26] GD Forney and AR Calderbank "Coset Codes for Partial Response Channels; or, Coset Codes with Spectral Nulls", *IEEE Transactions on Information Theory,* vol. 35, 1989, pp. 925-943.

[27] GD Forney, "Coset Codes- Part1: Introduction and Geometrical Classification", *IEEE Transactions on Information Theory*, vol. 34, 1988, No5, pp. 1123-1151.

[28] GD Forney, "Coset Codes- Part 2: Binary Lattices and Related Codes", *IEEE Transactions on Information Theory,* vol. 34, No. 5, 1988, pp. 1152-1187.

[29] GD Forney and MD Trott, "The Dynamics of Group Codes: State Spaces, Trellis Diagram and Canonical Encoders", *IEEE Transactions on Information Theory*, vol. 39, 1993, pp. 1491-1513.

[30] GD Forney, "Dimension/Length Profiles and Trellis Complexity of Linear Block Codes", *IEEE Transactions on Information Theory,* vol. 40, 1994, pp. 1741-1752.

[31] GD Forney, "Density/Length Profiles and Trellis Complexity of Lattices", *IEEE Transactions on Information Theory,* vol. 40, 1994, pp. 1753-1772.

[32] E Berlekamp, *Algebraic Coding Theory,* Agean Park Press, Laguna Hills, Revised edition, 1984.

[33] RJ McEliece, "The Viterbi Decoding Complexity of Linear Block Codes" *Proceedings of IEEE International Symposium on Information Theory,*

Trondheim, Norway, 1994, p. 341

[34] S Sonander, *Generalized Array Code Construction Algorithm for Reed-Solomon Codes,* Msc Thesis, Lancaster University, 1994.

[35] O Ytrehus, "On the Trellis Complexity of Certain Binary Block Codes", *IEEE Transactions on Information Theory,* vol. 41, 1995, No. 2, pp. 559-560.

Appendix A

Another Syndrome Trellis Construction Technique

A.1 Introduction

In **Section 3.3** of **Chapter 3** a method of constructing a syndrome trellis diagram as proposed by Wolf was described. In this Appendix, another method of constructing the syndrome trellis diagram will be investigated. The method is similar to the one described by Wolf, but in this method, the syndromes of the parity check matrix are directly used to construct the trellis diagram.

In order to ensure complete understanding of this method, an example to illustrate the various steps is provided.

A.2 Generation of the Code Book

Consider the following binary linear block code (5,3,2) with the following parity check matrix:

$$\boldsymbol{H} = \left(\begin{array}{ccc|cc} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{array}\right) = \left(\begin{array}{c|c} -\boldsymbol{P}^{t} & \boldsymbol{I}_{n-k} \end{array}\right) \qquad \textbf{(A.1)}$$

with $n = 5$, $k = 3$ and $(n - k) = 2$.

By using the above mentioned values for n, k and $(n - k)$ together with definitions from **Chapter 2**, the following parameters can be calculated. Firstly, the number of code words is,

$$2^k = 2^3 = 8 = M \tag{A.2}$$

and the number of states are:

$$2^{n-k} = 2^{5-3} = 2^2 = 4 \tag{A.3}$$

As shown in **Appendix B**, it is possible to obtain the generator matrix **G** from the parity check matrix **H** of a given block code C. The reverse is of course also possible, provided that the parity check matrix is in the systematic form, or transformed into the systematic form first.

For further information on this transformation, **Appendix 3** should be consulted.

As mentioned above, the reverse process (generator matrix **G** from parity check matrix **H**) would look as follows. Note that the parity check matrix was already in systematic form which means that there is no need to first transform the parity check matrix into systematic form before the generator matrix is calculated.

$$\boldsymbol{G} = \left(\boldsymbol{I}_k \;\middle|\; \boldsymbol{P} \right) = \left(\begin{array}{ccc|cc} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{array} \right) \tag{A.4}$$

The code book can now be constructed by calculating the result of taking all the possible linear combinations of the generator matrix.

In other words, each row is Modula-2 added to every other row. The resulting output vector is one of the codewords in the code book. Noting that it is a linear code, it is also necessary to consider the all zero state. The following is a list of all the codewords and their associated code bits.

$$
\begin{aligned}
C_1 &= 1\ 0\ 0\ 1\ 1 = c_{11}\ c_{12}\ c_{13}\ c_{14}\ c_{15} \\
C_2 &= 0\ 1\ 0\ 1\ 0 = c_{21}\ c_{22}\ c_{23}\ c_{24}\ c_{25} \\
C_3 &= 0\ 0\ 1\ 0\ 1 = c_{31}\ c_{32}\ c_{33}\ c_{34}\ c_{35} \\
C_4 &= 1\ 1\ 0\ 0\ 1 = c_{41}\ c_{42}\ c_{43}\ c_{44}\ c_{45}\ [1 \oplus 2] \\
C_5 &= 1\ 0\ 1\ 1\ 0 = c_{51}\ c_{52}\ c_{53}\ c_{54}\ c_{55}\ [1 \oplus 3] \\
C_6 &= 0\ 1\ 1\ 1\ 1 = c_{61}\ c_{62}\ c_{63}\ c_{64}\ c_{65}\ [2 \oplus 3] \\
C_7 &= 1\ 1\ 1\ 0\ 0 = c_{71}\ c_{72}\ c_{73}\ c_{74}\ c_{75}\ [1 \oplus 2 \oplus 3] \\
C_8 &= 0\ 0\ 0\ 0\ 0 = c_{81}\ c_{82}\ c_{83}\ c_{84}\ c_{85}\ [All\ 0]
\end{aligned}
\tag{A.5}
$$

The above list contains all the possible codewords. We are now ready to proceed to the next step.

A.3 Calculation of the Syndromes

The next step in the process is to calculate the syndromes for each of the code words.

In order to evaluate the syndromes, some definitions would be in order here.

Syndrome S_{1n} is defined as follows:

$$S_{1n} = \breve{C}_{1n} \cdot \boldsymbol{H}^T \tag{A.6}$$

The transformed codeword is defined below:

$$
\begin{aligned}
\breve{C}_{11} &= \begin{pmatrix} c_{11} & 0 & 0 & 0 & 0 \end{pmatrix} \\
\breve{C}_{12} &= \begin{pmatrix} c_{11} & c_{12} & 0 & 0 & 0 \end{pmatrix} \\
&\ \ \vdots \\
\breve{C}_{1n} &= \begin{pmatrix} c_{11} & c_{12} & \cdots & \cdots & c_{1n} \end{pmatrix}
\end{aligned}
\tag{A.7}
$$

- Consider codeword $C_1 = \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} & c_{15} \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \end{bmatrix}$

  Start with the first transformed codeword. This is used to calculate the first syndrome.

$$S_{11} = \tilde{C}_{11} \cdot \boldsymbol{H}^T = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \end{pmatrix} = \xi_3 \qquad \textbf{(A.8)}$$

  Using the other transformed codewords, the other syndromes are calculated.

$$S_{12} = \tilde{C}_{12} \cdot \boldsymbol{H}^T = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \end{pmatrix} = \xi_3 \qquad \textbf{(A.9)}$$

$$S_{13} = \tilde{C}_{13} \cdot \boldsymbol{H}^T = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \end{pmatrix} = \xi_3 \qquad \textbf{(A.10)}$$

$$S_{14} = \breve{C}_{14} \cdot \boldsymbol{H}^T = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \end{pmatrix} = \xi_1 \qquad \textbf{(A.11)}$$

$$S_{15} = \breve{C}_{15} \cdot \boldsymbol{H}^T = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \qquad \textbf{(A.12)}$$

- Consider codeword $C_2 = \begin{pmatrix} c_{21} & c_{22} & c_{23} & c_{24} & c_{25} \end{pmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \end{bmatrix}$

  Due to space limitations, only the calculated syndromes for the rest of the codes will be displayed.

$$S_{21} = \breve{C}_{21} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \qquad \textbf{(A.13)}$$

$$S_{22} = \breve{C}_{22} \cdot \boldsymbol{H}^T = \begin{pmatrix} 1 & 0 \end{pmatrix} = \xi_2 \qquad \textbf{(A.14)}$$

$$S_{23} = \breve{C}_{23} \cdot \boldsymbol{H}^T = \begin{pmatrix} 1 & 0 \end{pmatrix} = \xi_2 \qquad \textbf{(A.15)}$$

$$S_{24} = \breve{C}_{24} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \qquad \textbf{(A.16)}$$

$$S_{25} = \tilde{C}_{25} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \quad \textbf{(A.17)}$$

- Consider codeword $C_3 = \begin{pmatrix} c_{31} & c_{32} & c_{33} & c_{34} & c_{35} \end{pmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \end{bmatrix}$

The syndromes for this codeword are given below.

$$S_{31} = \tilde{C}_{31} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \quad \textbf{(A.18)}$$

$$S_{32} = \tilde{C}_{32} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \quad \textbf{(A.19)}$$

$$S_{33} = \tilde{C}_{33} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 1 \end{pmatrix} = \xi_1 \quad \textbf{(A.20)}$$

$$S_{34} = \tilde{C}_{34} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 1 \end{pmatrix} = \xi_1 \quad \textbf{(A.21)}$$

$$S_{35} = \tilde{C}_{35} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \quad \textbf{(A.22)}$$

- Consider codeword $C_4 = \begin{pmatrix} c_{41} & c_{42} & c_{43} & c_{44} & c_{45} \end{pmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 \end{bmatrix}$

The syndromes are given below.

$$S_{41} = \tilde{C}_{41} \cdot \boldsymbol{H}^T = \begin{pmatrix} 1 & 1 \end{pmatrix} = \xi_3 \quad \textbf{(A.23)}$$

$$S_{42} = \tilde{C}_{42} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 1 \end{pmatrix} = \xi_1 \quad \textbf{(A.24)}$$

$$S_{43} = \tilde{C}_{43} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 1 \end{pmatrix} = \xi_1 \quad \textbf{(A.25)}$$

$$S_{44} = \breve{C}_{44} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 1 \end{pmatrix} = \xi_1 \quad \textbf{(A.26)}$$

$$S_{45} = \breve{C}_{45} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \quad \textbf{(A.27)}$$

- Consider codeword $C_5 = \begin{pmatrix} c_{51} & c_{52} & c_{53} & c_{54} & c_{55} \end{pmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \end{bmatrix}$

  The appropriate syndromes for this code are listed below.

$$S_{51} = \breve{C}_{51} \cdot \boldsymbol{H}^T = \begin{pmatrix} 1 & 1 \end{pmatrix} = \xi_3 \quad \textbf{(A.28)}$$

$$S_{52} = \breve{C}_{52} \cdot \boldsymbol{H}^T = \begin{pmatrix} 1 & 1 \end{pmatrix} = \xi_3 \quad \textbf{(A.29)}$$

$$S_{53} = \breve{C}_{53} \cdot \boldsymbol{H}^T = \begin{pmatrix} 1 & 0 \end{pmatrix} = \xi_2 \quad \textbf{(A.30)}$$

$$S_{54} = \breve{C}_{54} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \quad \textbf{(A.31)}$$

$$S_{55} = \breve{C}_{55} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \quad \textbf{(A.32)}$$

- Consider codeword $C_6 = \begin{pmatrix} c_{61} & c_{62} & c_{63} & c_{64} & c_{65} \end{pmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \end{bmatrix}$

  The syndromes for this codeword are given below.

$$S_{61} = \breve{C}_{61} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \quad \textbf{(A.33)}$$

$$S_{62} = \breve{C}_{62} \cdot \boldsymbol{H}^T = \begin{pmatrix} 1 & 0 \end{pmatrix} = \xi_2 \quad \textbf{(A.34)}$$

$$S_{63} = \breve{C}_{63} \cdot \boldsymbol{H}^T = \begin{pmatrix} 1 & 1 \end{pmatrix} = \xi_3 \quad \textbf{(A.35)}$$

$$S_{64} = \breve{C}_{64} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 1 \end{pmatrix} = \xi_1 \quad \textbf{(A.36)}$$

$$S_{65} = \breve{C}_{65} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \quad \textbf{(A.37)}$$

- Consider codeword $C_7 = \begin{pmatrix} c_{71} & c_{72} & c_{73} & c_{74} & c_{75} \end{pmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \end{bmatrix}$

  The syndromes are given below.

$$S_{71} = \breve{C}_{71} \cdot \boldsymbol{H}^T = \begin{pmatrix} 1 & 1 \end{pmatrix} = \xi_3 \quad \textbf{(A.38)}$$

$$S_{72} = \breve{C}_{72} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 1 \end{pmatrix} = \xi_1 \quad \textbf{(A.39)}$$

$$S_{73} = \breve{C}_{73} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \quad \textbf{(A.40)}$$

$$S_{74} = \breve{C}_{74} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \quad \textbf{(A.41)}$$

$$S_{75} = \breve{C}_{75} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \quad \textbf{(A.42)}$$

- Consider codeword $C_8 = \begin{pmatrix} c_{81} & c_{82} & c_{83} & c_{84} & c_{85} \end{pmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

  The syndromes for this codeword are as below.

$$S_{81} = \breve{C}_{81} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \quad \textbf{(A.43)}$$

$$S_{82} = \breve{C}_{82} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \quad \textbf{(A.44)}$$

$$S_{83} = \breve{C}_{83} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \tag{A.45}$$

$$S_{84} = \breve{C}_{84} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \tag{A.46}$$

$$S_{85} = \breve{C}_{85} \cdot \boldsymbol{H}^T = \begin{pmatrix} 0 & 0 \end{pmatrix} = \xi_0 \tag{A.47}$$

After having calculated the relevant syndromes for the codewords, one can now proceed to draw the syndrome trellis diagram.

A.4 Constructing the Syndrome Trellis Diagram

Each one of the previously calculated syndromes is associated with a node in the syndrome trellis diagram. The time element of the trellis diagram is enclosed in the number of the specific syndrome. The first syndrome of a code would be the first in time, and the last syndrome last in time (further down the trellis).

- Construct single code trellis for $C_1 = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \end{pmatrix}$

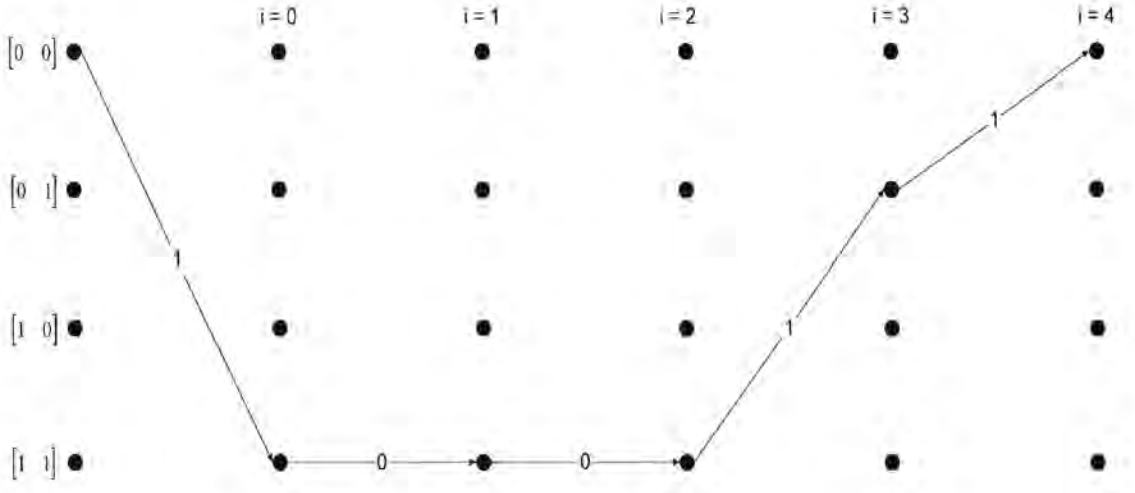

Figure 46 Single Code Trellis for Codeword 1

- Construct single code trellis for $C_2 = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \end{pmatrix}$

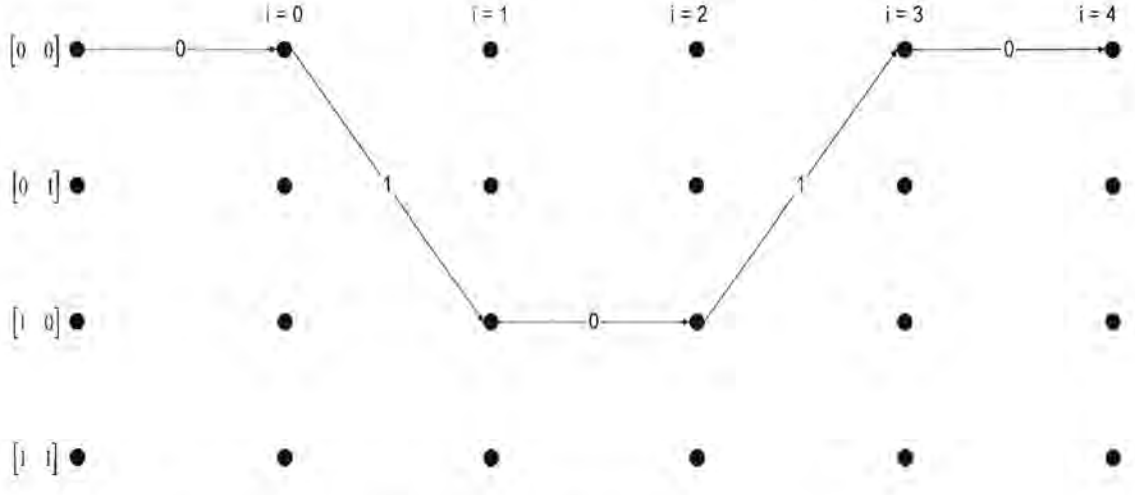

Figure 47 Single Code Trellis for Codeword 2

- Construct single code trellis for $C_3 = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 \end{pmatrix}$

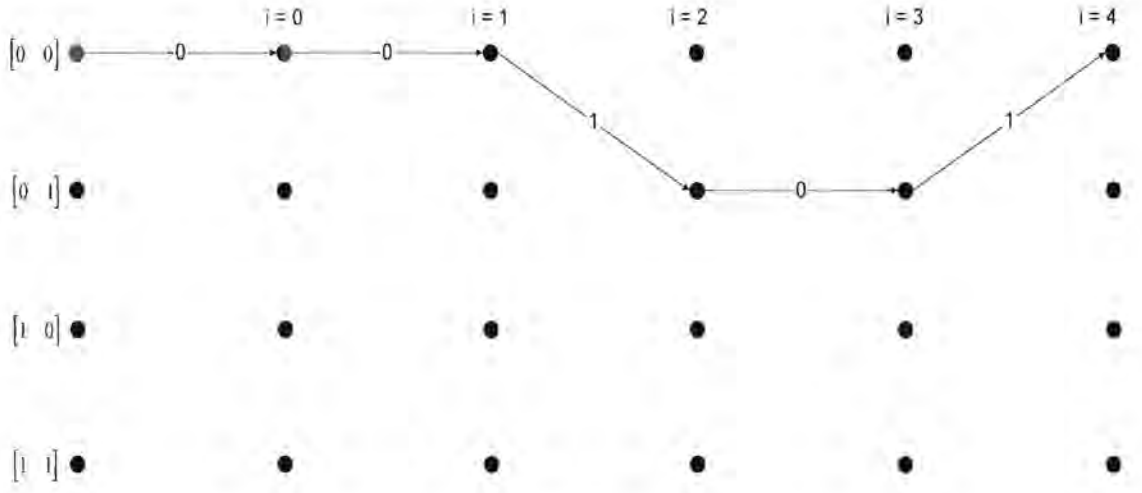

Figure 48 Single Code Trellis for Codeword 3

- Construct single code trellis for $C_4 = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 \end{pmatrix}$

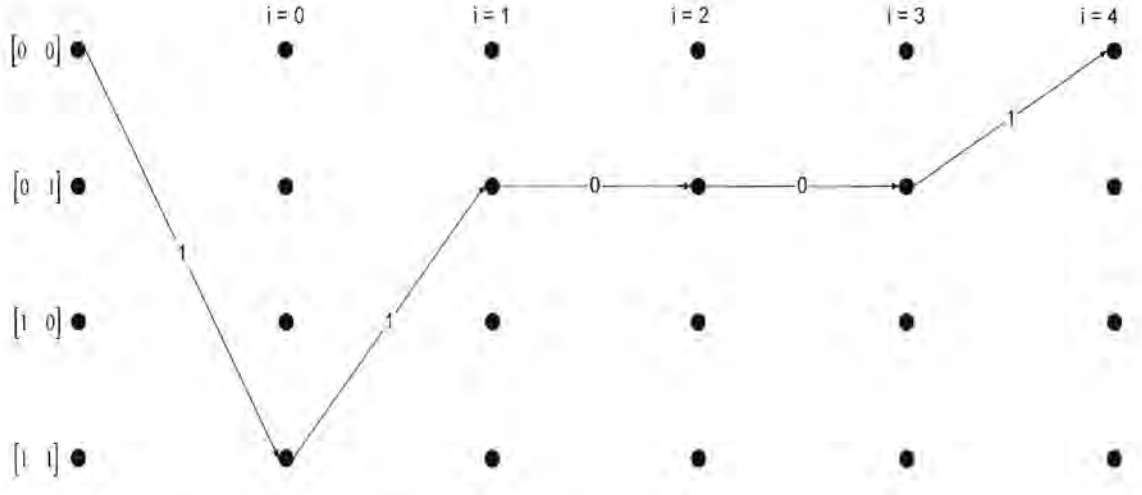

Figure 49 Single Code Trellis for Codeword 4

- Construct single code trellis for $C_5 = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \end{pmatrix}$

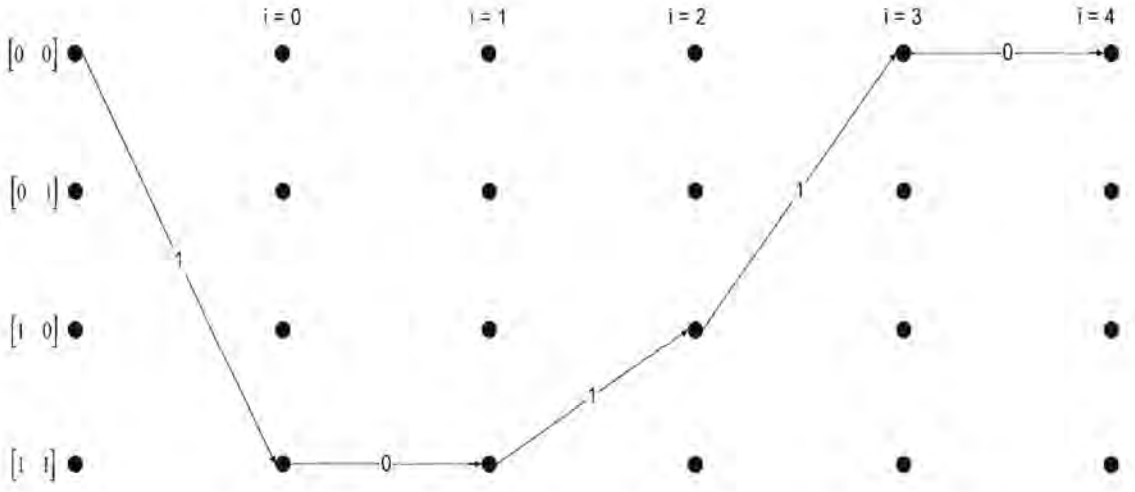

Figure 50 Single Code Trellis for Codeword 5

- Construct single code trellis for $C_6 = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \end{pmatrix}$

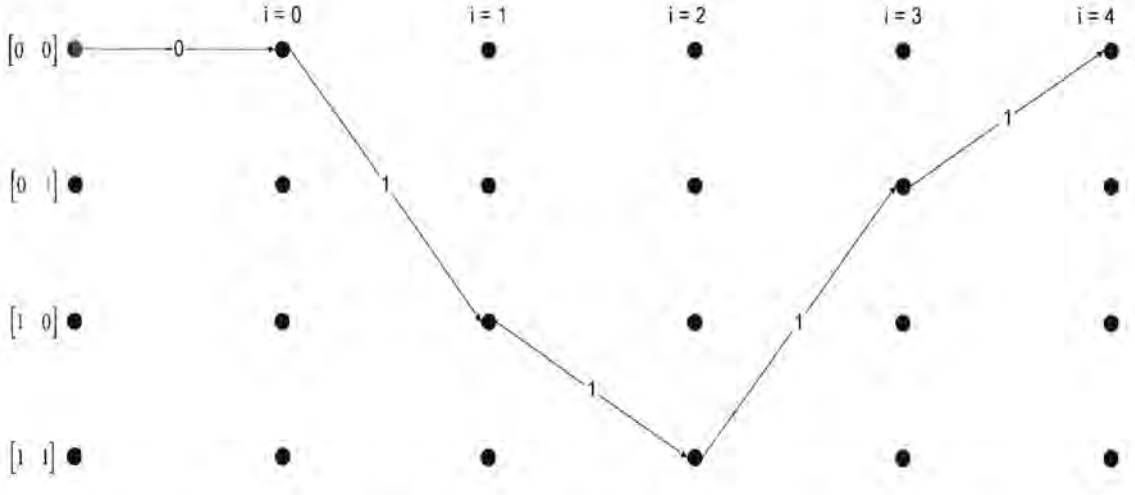

Figure 51 Single Code Trellis for Codeword 6

- Construct single code trellis for $C_7 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \end{pmatrix}$

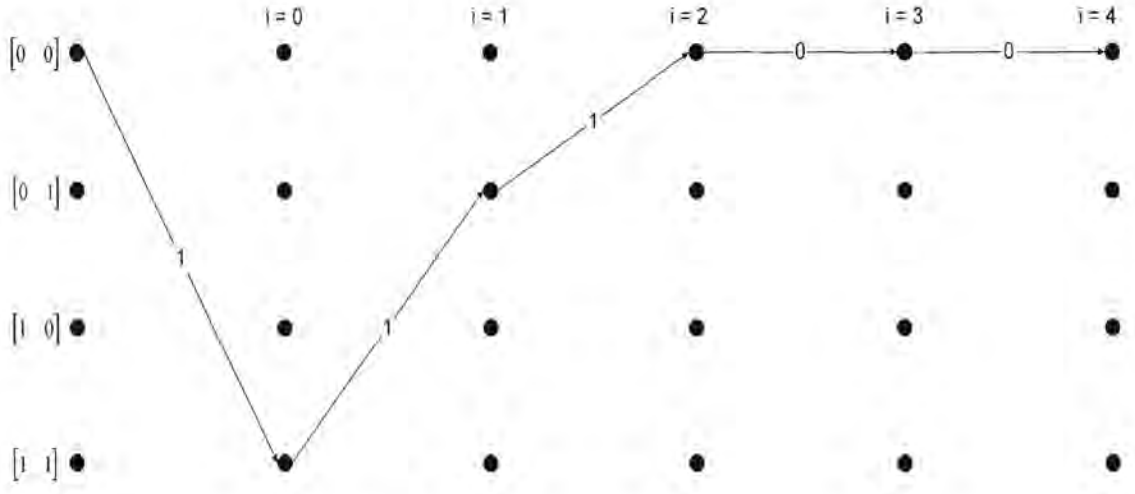

Figure 52 Single Code Trellis for Codeword 7

- Construct single code trellis for $C_8 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \end{pmatrix}$

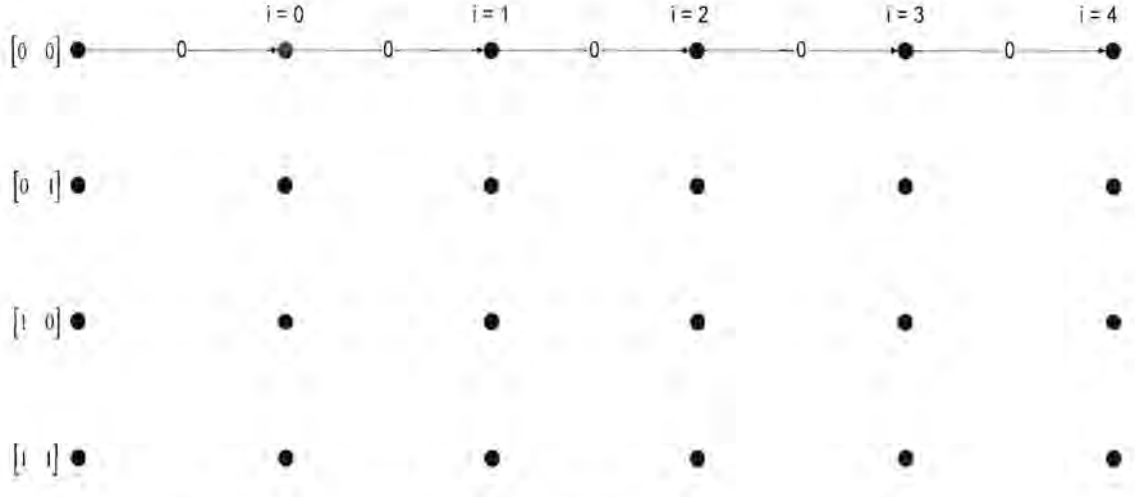

Figure 53 Single Code Trellis for Codeword 8

- Combination of all single code trellis diagrams

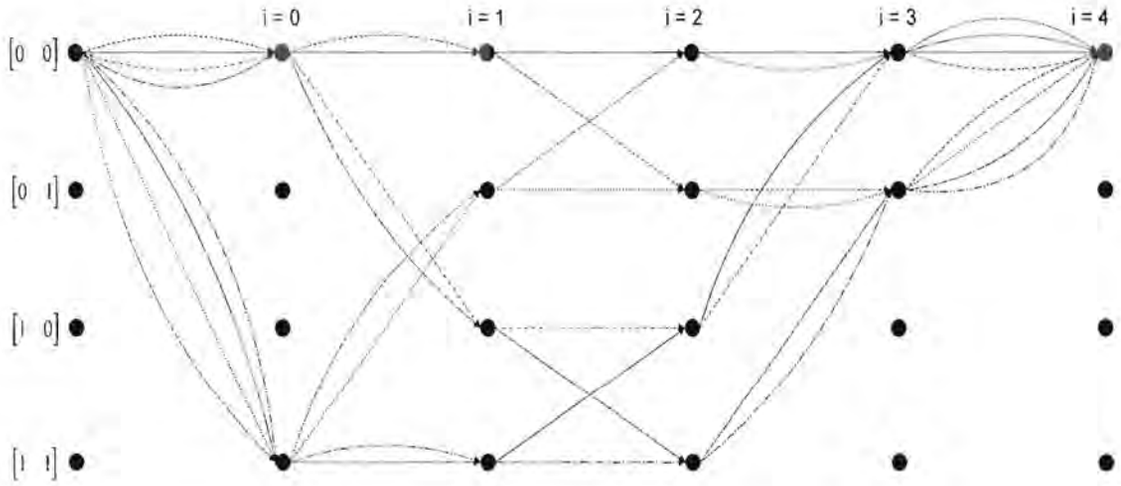

Figure 54 Combined Single Code Trellis Diagram

- Create the final syndrome trellis diagram

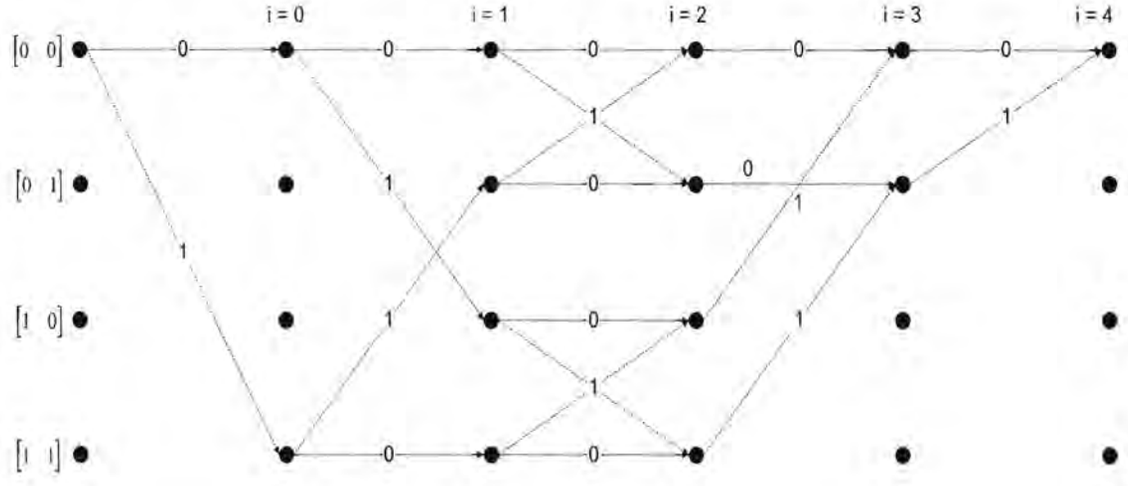

Figure 55 Final Syndrome Trellis Diagram

Appendix B

Computation of the Parity Check Matrix from the Generator Matrix

B.1 Introduction

In **Section 3.4** it was described analytically how the parity check matrix **H** of a linear block code was generated from the generator matrix **G**. An example to illustrate the proposed method is given below:

B.2 Explanation by Example

Given the following generator matrix

$$\boldsymbol{G} = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix} \tag{B.1}$$

of a (7,4)-Hamming-Code.

Firstly, we will try and transform a column of this generator matrix G in such a manner that it represents the first unit vector after transformation

$$\boldsymbol{e}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{B.2}$$

The first step in achieving this is the swapping of the first and second row. Note that according to **Chapter 2** such elementary matrix operations are allowed.

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix} \tag{B.3}$$

In order to complete the task, the third and fourth rows are replaced by their respective sums with the first row. When this is done, the first column is the first unit vector.

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{B.4}$$

For the further transformation, the element of the second row in the second column has to be a 1. This is done by swapping the second and the third row.

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{B.5}$$

The elements of the first and the fourth row in the second column are now made 0, in that the second row is added to these two rows.

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \tag{B.6}$$

In the third row of the third column there is already a 1 present. This makes the next operation easier since all that is require is to make the rest of the elements in this column 0. This is done by adding the third row to the particular rows.

$$\begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \tag{B.7}$$

Preferably, only a 1 in the fourth row of the 4 column should be present. There is however a zero. There is also no lower row with a 1 in the fourth column, which can be exchanged with the fourth row in order to meet the requirement. Rows further up in the matrix can also not be used to perform the row swapping operation, since this would also disturb the unit vectors already present in several of the columns. The only possible solution is not to use the fourth column, but to move along to another column further to the right of the matrix. The column that we choose for this is the fifth one. This one is chosen due to the fact that it already has a one in the fourth row. This makes it a lot easier. The other ones present in this column will also have to be made 0, and this is accomplished by adding the fourth row to the particular rows. By doing this, we obtain the following generator matrix which still produces exactly the same code. It does however have a different mapping between the information vectors $\boldsymbol{i}$ and the code vectors $\boldsymbol{c}$ (for clarification on this, the reader is referred back to **Section 2.1** in **Chapter 2**).

$$\begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \tag{B.8}$$

In the first, second, third and fifth columns the following unit vectors are present. This completes the first step in finding the parity check matrix.

$$\boldsymbol{e}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \qquad \boldsymbol{e}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \tag{B.9}$$

$$\boldsymbol{e}_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \qquad \boldsymbol{e}_4 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{B.10}$$

To bring this matrix into the systematic form **G'** it is necessary to exchange the fourth and the fifth columns. In the systematic form matrix **G'**, the following matrix

$$\boldsymbol{A} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} \tag{B.11}$$

is contained in the fourth, sixth and seventh columns. In these columns of the corresponding parity check matrix **H**, the unit matrix $\mathbf{I}_{n-k}$ is present. The remaining columns (first, second, third and fifth columns) make up the transposed matrix

$$-\boldsymbol{A}^T = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \tag{B.12}$$

This results in the parity check matrix that was used to construct the syndrome trellis diagram of **Chapter 3**. It can easily be shown that all the relevant equations in **Section 3.4** are satisfied.

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \tag{B.13}$$

Appendix C

Generator Matrixes of Simulated Codes

C.1 Introduction

For reasons of completeness and also so that simulation results can be repeated by other people basing their work on this piece, this **Appendix C** will contain most of the generator matrixes **G** of the codes used in this work. In order to obtain the parity check matrix **H** for a particular code, the generator matrix **G** as given in this appendix can be transformed using the process described in **Chapter 3** and illustrated in **Appendix B**.

C.2 Generator Matrixes of given BCH-Codes

The (15,7)-BCH-Code has the following generator polynomial given in the exponential form below.

$$g(x) = 1 + x^4 + x^6 + x^7 + x^8 \qquad \textbf{(C.1)}$$

The generator matrix can be obtained from the polynomial and is as follows.

$$\boldsymbol{G} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix} \qquad \textbf{(C.2)}$$

For the (31,21)-BCH-Code, the generator polynomial and generator matrix is as given below.

$$g(x) = 1 + x^3 + x^5 + x^6 + x^8 + x^9 + x^{10} \tag{C.3}$$

$$G = \begin{pmatrix}
1&0&0&1&0&1&1&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&1&0&1&1&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&1&0&1&1&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&1&0&1&1&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&1&0&1&1&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&1&0&1&1&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&1&0&1&1&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&1&0&1&1&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&1&0&1&1&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&1&0&1&1&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&1&1&0&1&1&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&1&1&0&1&1&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&1&1&0&1&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&1&1&0&1&1&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&1&1&0&1&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&1&1&0&1&1&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&1&1&0&1&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&1&1&0&1&1&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&1&1&0&1&1&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&1&1&0&1&1&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&1&1&0&1&1&1
\end{pmatrix} \tag{C.4}$$

C.3 Generator Matrixes of given Reed-Muller-Codes

Here follow several Reed-Muller-Codes and their generator matrixes.

- (16,5)-Reed-Muller-Code $R(1,4)$

$$\boldsymbol{G} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \quad \textbf{(C.5)}$$

- (16,11)-Reed-Muller-Code $R(2,4)$

$$\boldsymbol{G} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \quad \textbf{(C.6)}$$

- (32,31)-Reed-Muller-Code $R(4,5)$ (Parity-Check-Code)

$$G = \begin{pmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1
\end{pmatrix} \quad \textbf{(C.7)}$$

- (32,26)-Reed-Muller-Code $R(3,5)$

$$G = \begin{pmatrix}
1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0\\
1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0\\
0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0\\
0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&1&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&1&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0\\
1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1
\end{pmatrix} \quad \text{(C.8)}$$

- (32,16)-Reed-Muller-Code $R(2,5)$

$$G = \begin{pmatrix}
1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0\\
1&0&1&0&0&0&0&0&1&0&1&0&0&0&0&0&1&0&1&0&0&0&0&0&1&0&1&0&0&0&0&0\\
1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0\\
0&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0\\
1&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0\\
1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1&0&0\\
0&0&0&0&0&0&0&0&1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0\\
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1
\end{pmatrix} \quad \textbf{(C.9)}$$

Appendix D

Shannon Product of Trellises

D.1 Introduction

In **Chapter 5**, extensive use of the Shannon Product of trellises theory was made. This Appendix focuses on this unique trellis "combination" method developed by Shannon.

it should however be noted that the Shannon product of trellises is in itself also a fully functional method of designing a trellis diagram from first principles. It does however not always deliver a minimal trellis representation.

D.2 Analytical Approach to the Shannon Trellis Product

The method will first be described on an analytical basis, and will be illustrated with an example afterwards.

Let C_1 and C_2 be binary linear block codes with parameters (N,K_1,D_1) and (N,K_2,D_2) respectively. The two generator matrixes for the codes will be $\mathbf{G}_1$ and $\mathbf{G}_2$. It is further assumed that C1 and C2 only have the all-zero code word in common. From this would follow that:

$$C_1 \cap C_2 = \{0\} \tag{D.1}$$

Their direct sum is defined as follows:

$$C \stackrel{\Delta}{=} C_1 \oplus C_2 \stackrel{\Delta}{=} \{u+v : u \in C_1, v \in C_2\} \tag{D.2}$$

From this it would follow that C is an (N,K_1+K_2,d) linear block code with minimum distance equal to

$$d_{\min} = \min\{d_1, d_2\} \tag{D.3}$$

and generator matrix

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{G}_1 \\ \boldsymbol{G}_2 \end{bmatrix}. \tag{D.4}$$

Let T_1 and T_2 be N-section trellis diagrams for C_1 and C_2 respectively. Then a N-section trellis T for the direct-sum code C can be constructed by taking the Shannon product of T_1 and T_2.

Let

$$T = T_1 \times T_2 \tag{D.5}$$

denote the Shannon product of T_1 and T_2.

The state spaces of $T1$ and $T2$ at time i are given as

$$\Sigma_i(C_1) \quad and \quad \Sigma_i(C_2) \quad for\ 0 \le i \le N \tag{D.6}$$

respectively.

The construction of the Shannon product T is carried out according to the following procedure:

- For all i, the state space product from of $C1$ and C2 is obtained according to **Equation D.6**. This product can then be given as

$$\Sigma_i(C_1) \times \Sigma_i(C_2) = \left\{ \left(\sigma_i^{(1)}, \sigma_i^{(2)} \right) : \sigma_i^{(1)} \in \Sigma_i(C_1), \sigma_i^{(2)} \in \Sigma_i(C_2) \right\} \tag{D.7}$$

  Then $\Sigma_1(C_1) \times \Sigma_2(C_2)$ forms the state space of $T_1 \times T_2$ at time i. In other words, the two-tuples from $\Sigma_1(C_1) \times \Sigma_2(C_2)$ form the nodes of T at level i.

- A state $\left(\sigma_i^{(1)}, \sigma_i^{(2)} \right) \in \Sigma_i(C_1) \times \Sigma_i(C_2)$ is adjacent to a state $\left(\sigma_{i+1}^{(1)}, \sigma_{i+1}^{(2)} \right) \in \Sigma_{i=1}(C_1) \times \Sigma_{i+1}(C_2)$ if and only if $\sigma_i^{(1)}$ is adjacent to $\sigma_{i+1}^{(1)}$ and $\sigma_i^{(2)}$ is adjacent to $\sigma_{i+1}^{(2)}$. Let $l\left(\sigma_i^{(1)}, \sigma_{i+1}^{(1)} \right)$ denote the label of the branch that

connects $\sigma_i^{(1)}$ to $\sigma_{i+1}^{(1)}$ in trellis T_1 and $l\left(\sigma_i^{(2)},\sigma_{i+1}^{(2)}\right)$ denote the label of the branch that connects $\sigma_i^{(2)}$ to $\sigma_{i+1}^{(2)}$ in trellis T_2. Then two adjacent states $\left(\sigma_i^{(1)},\sigma_i^{(2)}\right)$ and $\left(\sigma_{i+1}^{(1)},\sigma_{i+1}^{(1)}\right)$ in trellis T are connected by a branch with label

$$l\left(\sigma_i^{(1)},\sigma_{i+1}^{(1)}\right)+l\left(\sigma_i^{(2)},\sigma_{i+1}^{(2)}\right) \tag{D.8}$$

For $0 \le i \le N$, let $\rho_i\left(C_1\right)$ and $\rho_i\left(C_2\right)$ be the dimensions of the state spaces given in Equation D.6 respectively. Then the state space dimension profile of $T_1 \times T_2$ is given by

$$\left(\rho_0\left(C_1\right)+\rho_0\left(C_2\right),\rho_1\left(C_1\right)+\rho_1\left(C_2\right),\ldots,\rho_N\left(C_1\right)+\rho_N\left(C_2\right)\right) \tag{D.9}$$

D.3 Shannon Product of Trellises by Example

The following example will illustrate the technique as described in **Section D.2**.
The two block codes C_1 and C_2 are assumed to be linear, and have generator matrixes in the following form:

$$\boldsymbol{G}_1 = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \end{pmatrix} \tag{D.10}$$

and

$$\boldsymbol{G}_2 = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{D.11}$$

From the discussion in **Section D.2** it follows that

$$C_1 \cap C_2 = \{0\} \tag{D.12}$$

and that the direct-sum is given as

$$G = \begin{pmatrix} G_1 \\ G_2 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}. \tag{D.13}$$

The following two figures show the trellis diagrams for the two codes C_1 and C_2 respectively.

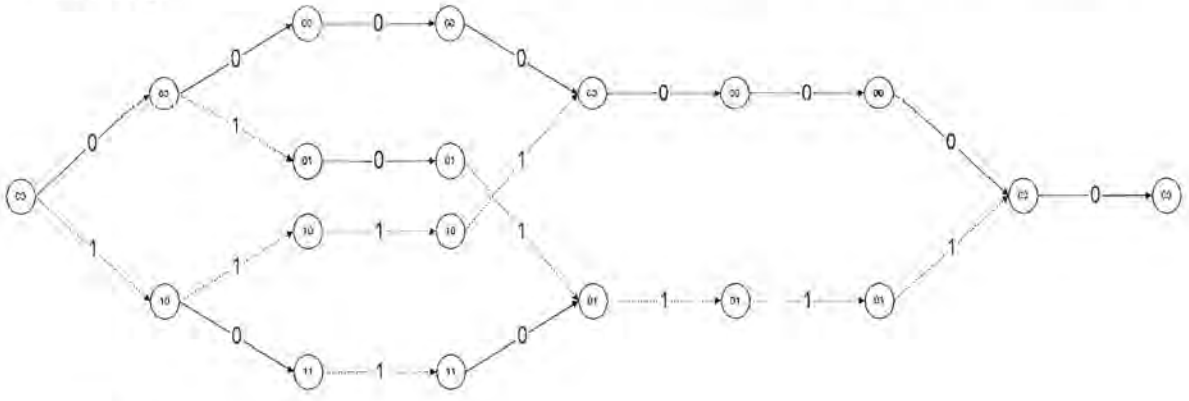

Figure 56 Trellis Diagram for Code C_1

These trellises are then combined with the Shannon procedure as described in **Section D.2**. The result is given in the figure below.

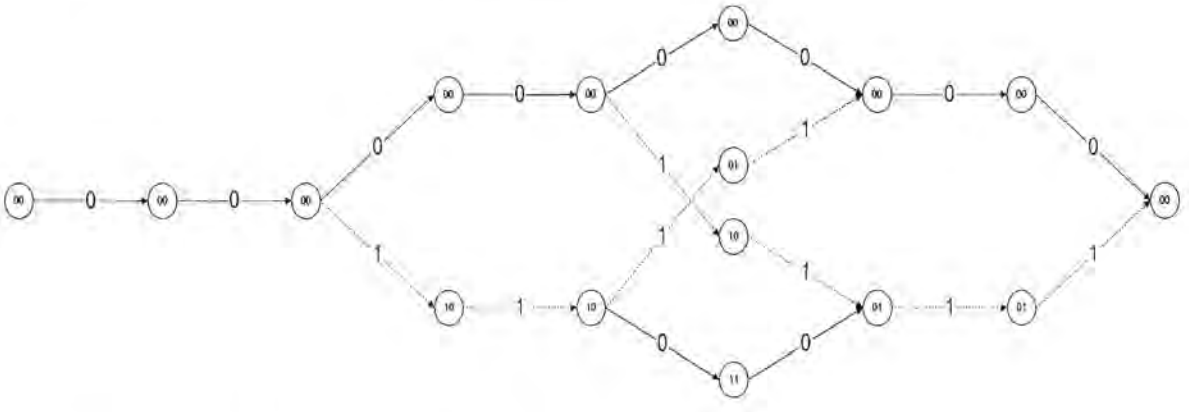

Figure 57 Trellis Diagram for Code C_2

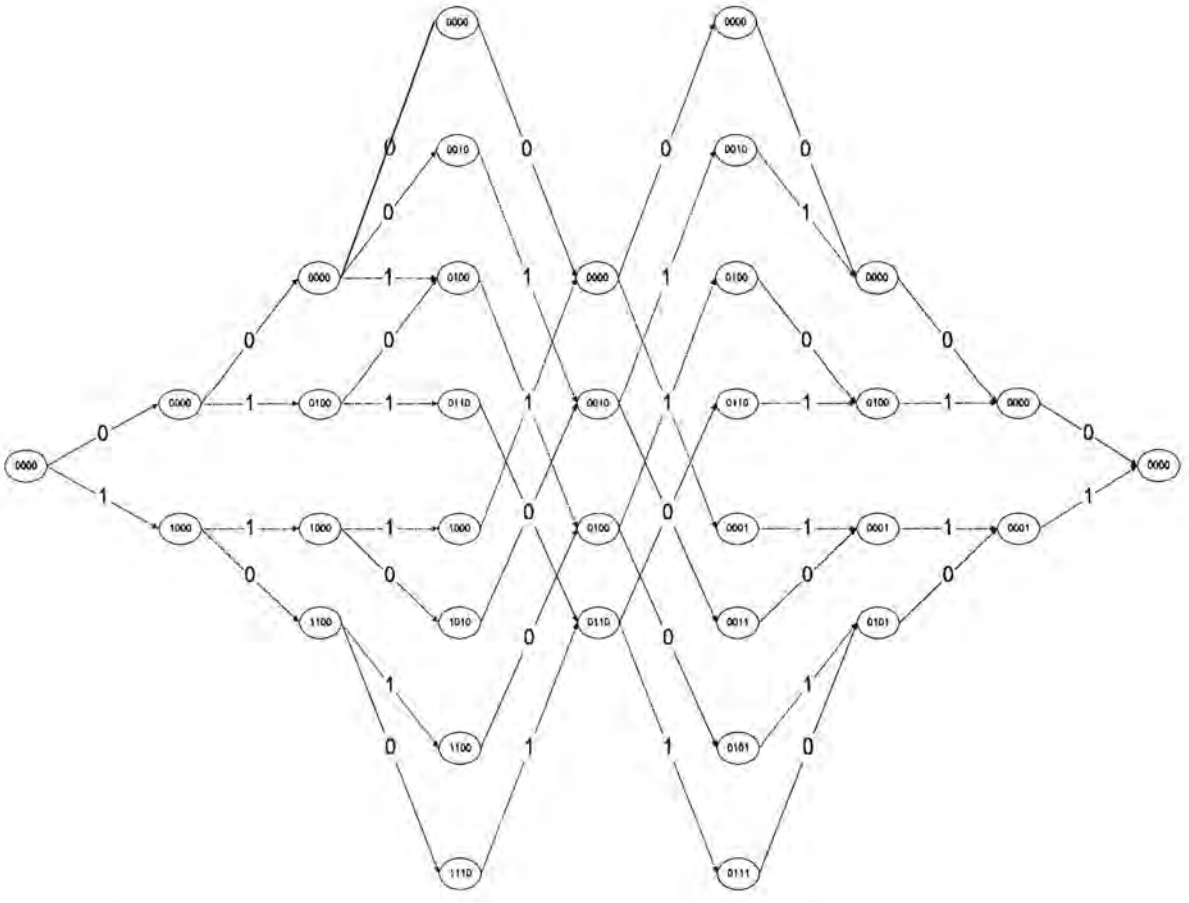

Figure 58 Combined Shannon Product Trellis Diagram

Appendix E

Coset and Syndrome Trellises by Example

E.1 Introduction

In this Appendix, the two techniques for Reed-Solomon trellis construction as presented in **Section 5.5** of **Chapter 5**, are illustrated by example. The first part is concerned with the syndrome construction technique, and the second part is concerned with the coset trellis design technique.

E.2 Syndrome Construction

This example will illustrate the use of the syndrome technique in obtaining a trellis diagram for a (7, 5, 3) Reed-Solomon code, with symbols taken from GF(8). The generator polynomial for this specific Reed-Solomon code is given as

$$g(X) = \alpha^3 + \alpha^4 \cdot X + X^2 \tag{E.1}$$

from which the generator matrix **G** can be constructed.

$$\boldsymbol{G} = \begin{pmatrix} g_1 \\ g_2 \\ g_3 \\ g_4 \\ g_5 \end{pmatrix} = \begin{pmatrix} \alpha^3 & \alpha^4 & 1 & 0 & 0 & 0 & 0 \\ 0 & \alpha^3 & \alpha^4 & 1 & 0 & 0 & 0 \\ 0 & 0 & \alpha^3 & \alpha^4 & 1 & 0 & 0 \\ 0 & 0 & 0 & \alpha^3 & \alpha^4 & 1 & 0 \\ 0 & 0 & 0 & 0 & \alpha^3 & \alpha^4 & 1 \end{pmatrix} \tag{E.2}$$

The overall Reed-Solomon code can be presented in the following form:

$$C = \sum_{j=1}^{5} C_j \tag{E.3}$$

In **Equation E.3** C_j with j=1,2,...,5 is a (7, 1, 3) sub-code generated by g_j. These sub-

codes can be expressed in the following format:

$$\begin{aligned}
C_1 &= x_1 \cdot g_1 = \left(\alpha^3 x_1,\ \alpha^4 x_1,\ x_1,\ 0,\ 0,\ 0,\ 0\right) \\
C_2 &= x_2 \cdot g_2 = \left(0,\ \alpha^3 x_2,\ \alpha^4 x_2,\ x_2,\ 0,\ 0,\ 0\right) \\
C_3 &= x_3 \cdot g_3 = \left(0,\ 0,\ \alpha^3 x_3,\ \alpha^4 x_3,\ x_3,\ 0,\ 0\right) \\
C_4 &= x_4 \cdot g_4 = \left(0,\ 0,\ 0,\ \alpha^3 x_4,\ \alpha^4 x_4,\ x_4,\ 0\right) \\
C_5 &= x_5 \cdot g_5 = \left(0,\ 0,\ 0,\ 0,\ \alpha^3 x_5,\ \alpha^4 x_5,\ x_5\right)
\end{aligned} \tag{E.4}$$

The component trellises have a very simple structure with the following state profiles:

$$\begin{aligned}
N_1(t) &= \left(1,\ 8,\ 8,\ 1,\ 1,\ 1,\ 1,\ 1\right) \\
N_2(t) &= \left(1,\ 1,\ 8,\ 8,\ 1,\ 1,\ 1,\ 1\right) \\
N_3(t) &= \left(1,\ 1,\ 1,\ 8,\ 8,\ 1,\ 1,\ 1\right) \\
N_4(t) &= \left(1,\ 1,\ 1,\ 1,\ 8,\ 8,\ 1,\ 1\right) \\
N_5(t) &= \left(1,\ 1,\ 1,\ 1,\ 1,\ 8,\ 8,\ 1\right)
\end{aligned} \tag{E.5}$$

The state profiles from **Equation E.5** are represented in graphical format below.

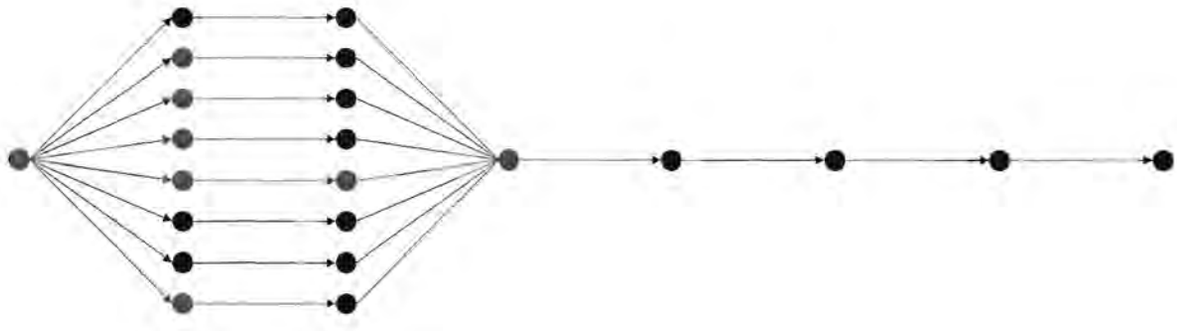

Figure 59 Sate Profile for C_1

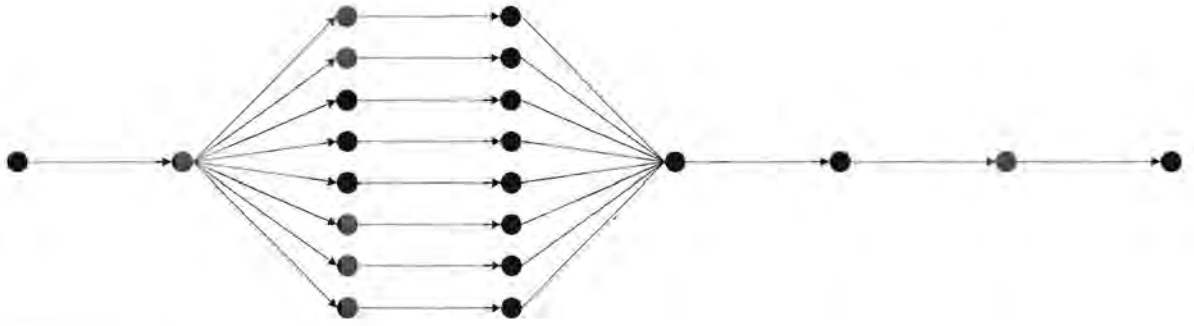

Figure 60 State Profile for C_2

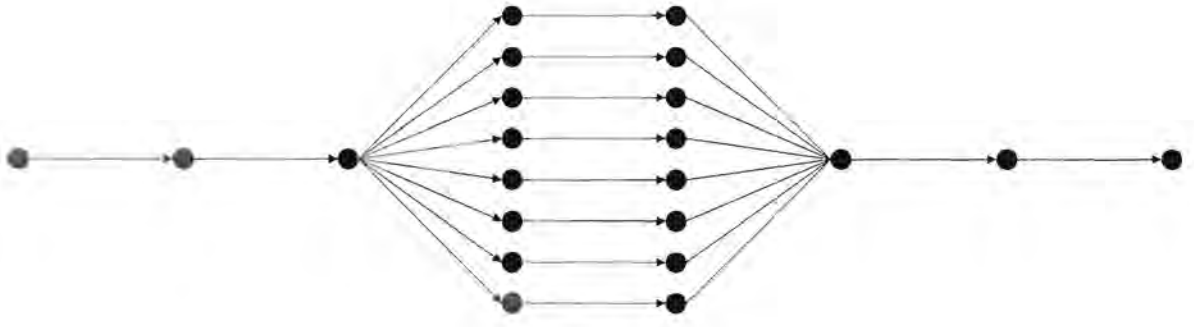

Figure 61 State Profile for C_3

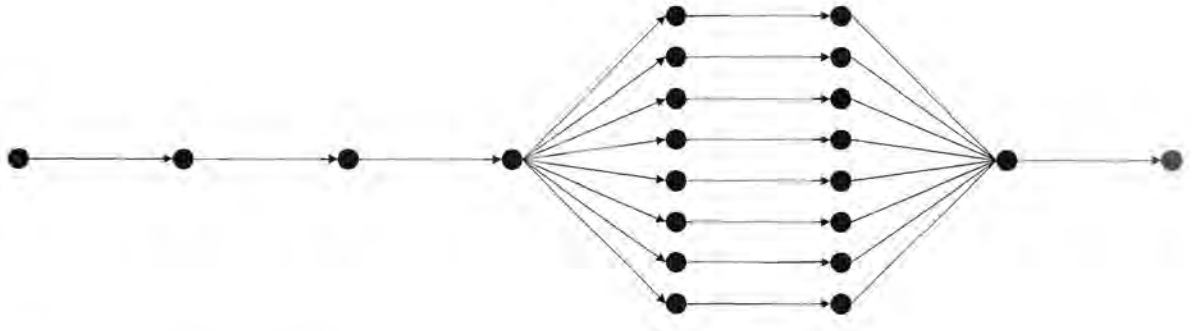

Figure 62 State Profile for C_4

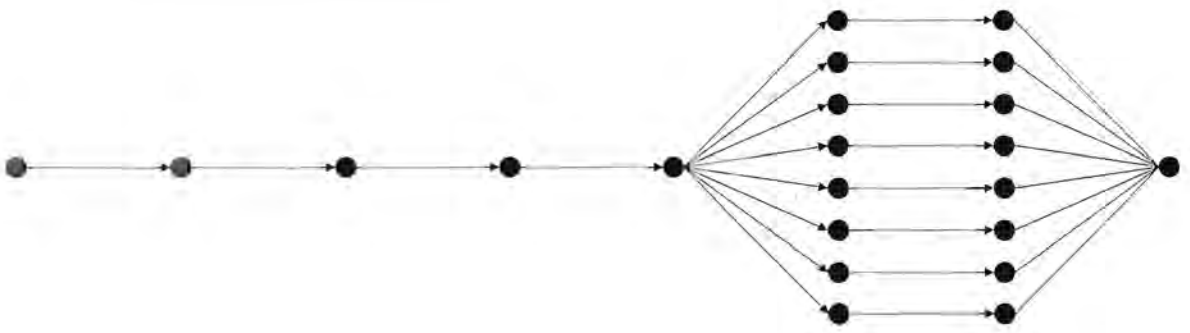

Figure 63 State Profile for C_5

When the procedure as described in **Section 6.2** is applied to the above state profiles, then a new state profile for the entire code can be written as:

$$N(t) = \{1,\ 8,\ 64,\ 64,\ 64,\ 64,\ 64,\ 8,\ 1\} \qquad \textbf{(E.6)}$$

The result of this procedure is given in the figure on the next page. Due to the huge amount of states and branches, no labeling is done on the graph itself. It should be noted that several branches have been omitted in state 3, state 4 and state 5.

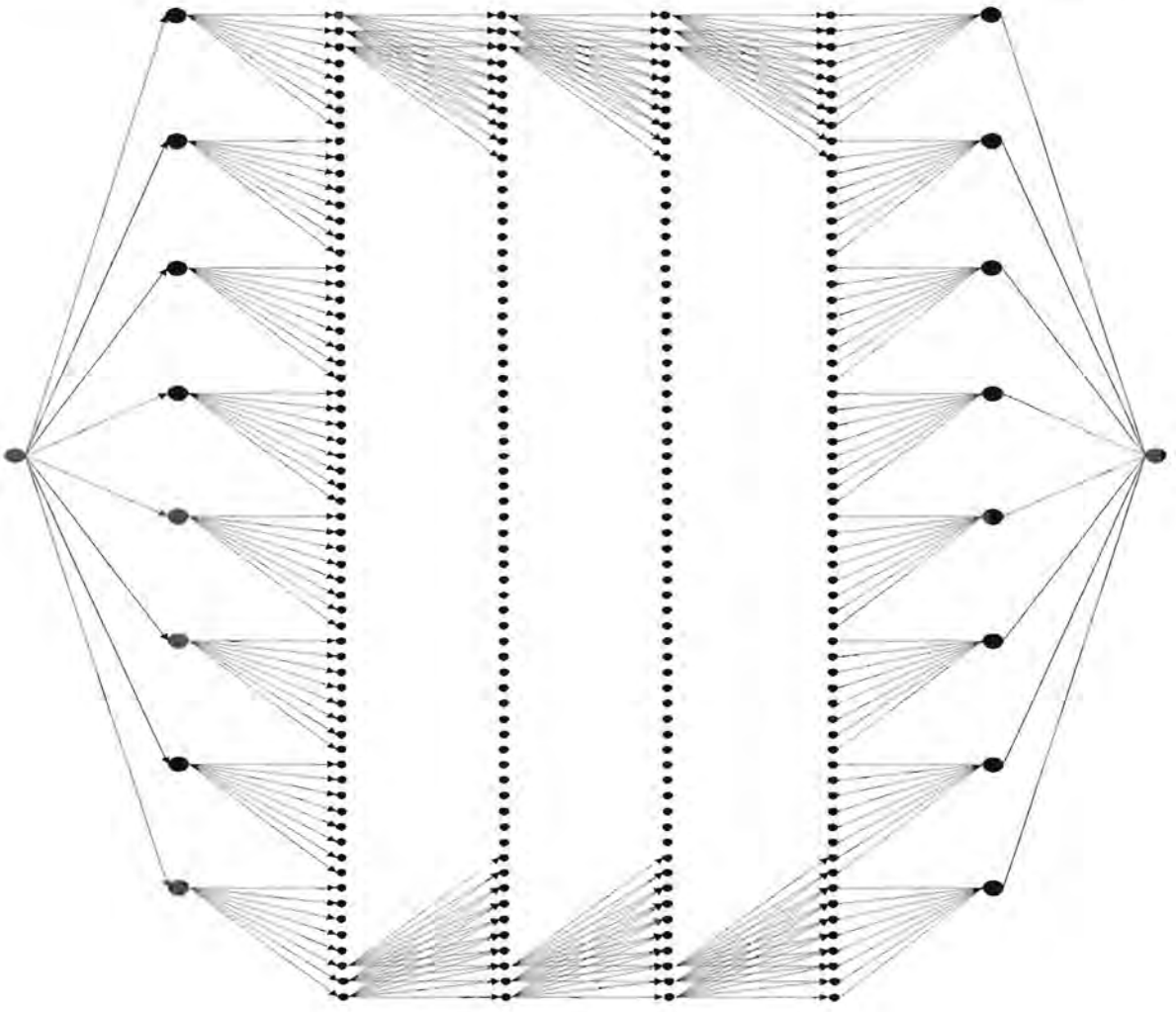

Figure 64 Syndrome Trellis Diagram for the (7, 5, 3) Reed-Solomon Code

E.3 Coset Construction

For this example, a (7, 5, 3) Reed-Solomon code is chosen with symbols taken from GF(8). The chosen code has the following generator polynomial

$$g(X) = \alpha^3 + \alpha \cdot X + X^2 + \alpha^3 \cdot X^3 + X^4 \qquad \textbf{(E.7)}$$

From the above generator polynomial, the generator matrix can be constructed as follows:

$$\boldsymbol{G} = \begin{pmatrix} \alpha^3 & \alpha & 1 & \alpha^3 & 1 & 0 & 0 \\ 0 & \alpha^3 & \alpha & 1 & \alpha^3 & 1 & 0 \\ 0 & 0 & \alpha^3 & \alpha & 1 & \alpha^3 & 1 \end{pmatrix} \tag{E.8}$$

The state profile can be obtained as given below:

$$\begin{aligned} N_0 &= \frac{q^3}{q^0 \cdot q^3} = 1 \\ N_1 &= \frac{q^3}{q^0 \cdot q^2} = q = 8 \\ N_2 &= \frac{q^3}{q^0 \cdot q} = q^2 = 64 \\ N_3 &= \frac{q^3}{q^0 \cdot q^0} = q^3 = 512 \\ N_4 &= \frac{q^3}{q^0 \cdot q^0} = q^3 = 512 \\ N_5 &= \frac{q^3}{q^1 \cdot q^0} = q^2 = 64 \\ N_6 &= \frac{q^3}{q^2 \cdot q^0} = q = 8 \\ N_7 &= \frac{q^3}{q^3 \cdot q^0} = 1 \end{aligned} \tag{E.9}$$

From this point onwards, a number of different but isomorphic trellis diagrams exist. The one chosen depends solely on the designer, and the procedure as given in **Section 5.5** is applied. The trellis is then again the Shannon product of a number of sub-trellises. Due to the large complexity of even this small code, the resulting trellis diagrams will not be presented here.

Appendix F

Galois Field Arithmetic

F.1 Introduction

This appendix will give a brief introduction into the arithmetic of fields. A field is a set of elements in which one is able to do addition, subtraction, multiplication and division without ever having to leave the set. Addition and multiplication must satisfy the commutative, associative and distributive laws.

F.2 Fields

Let F be a set of elements on which two binary operations called addition and multiplication are defined. The set F together with the two binary operations mentioned above is a field if the following conditions are satisfied:

- F is a commutative group under addition . The identity element with respect to addition is called the zero element or the additive identity of F and is denoted by 0.
- The set of nonzero elements in F is a commutative group under multiplication. The identity element with respect to multiplication is called the unit element or the multiplicative identity of F and is denoted by 1.
- Multiplication is distributive over addition, that is, for any three elements a, b and c in F the following holds:

$$a \cdot (b + c) = a \cdot b + a \cdot c. \qquad \textbf{(F.1)}$$

It follows from the above definition, that a field consists of at least two elements, the additive identity and the multiplicative identity.

The number of elements in a field is called the order of the field. A field with a finite number of elements is called a finite field. In a field, the additive inverse of an element a is denoted by $-a$ and the multiplicative inverse of a is denoted by a^{-1}, provided that a

is not equal to zero. Subtracting a field element b from another field element a is defined as adding the additive inverse. If b is a nonzero element dividing by b is defined as multiplying by the multiplicative inverse.

A number of basic properties of fields exist and are given below.

- For every element a in a field $a \cdot 0 = 0 \cdot a = 0$
- For any two nonzero elements a and b in a field $a \cdot b \neq 0$
- $a \cdot b = 0$ and $a \neq 0$ implies that $b = 0$
- For any two elements a and b in a field $-(a \cdot b) = (-a) \cdot (b) = (a) \cdot (-b)$
- For $a \neq 0$, $a \cdot b = a \cdot c$ implies that $b = c$

Fields are normally indicated by GF(q). This notation is used in honor of Galois who discovered these fields. The binary field in which most everyday calculations are done is called GF(2). Every Galois field contains a given number of symbols. In GF(4) the set is limited to 4 symbols. The symbols can be expressed in exponential or polynomial form, which makes addition and multiplication easier depending on which representation is used. A table is given below to illustrate the above process with 8 symbols.

No.	Exponential Representation	Polynomial Representation
1	0	0
2	1	1
3	α	α
4	α^2	α^2
5	α^3	$\alpha^3 = \alpha + 1$
6	α^4	$\alpha^3 \cdot \alpha = (\alpha + 1) \cdot \alpha = \alpha^2 + \alpha$
7	α^5	$\alpha^4 \cdot \alpha = \ldots = \alpha^2 + \alpha + 1$
8	α^6	$\alpha^5 \cdot \alpha = \ldots = \alpha^2 + 1$

Table 23 Galois Symbol Representation in Exponential and Polynomial Form

This discussion was just a very brief introduction to the study of finite fields and Galois fields. For further information, the reader is referred to standard works[21],[24],[25] on this topic.

Appendix G

Simulation Code

G.1 Introduction

This is just a partial reproduction of the simulation code used in the dissertation. The full code is available on request from the authors.

G.2 Simulation Code

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop
#include <stdio.h>
#include <math.h>
#include <vcl/clipbrd.hpp>
#include "main.h"
//---------------------------------------------------------------------------
#pragma resource "*.dfm"
TForm1 *Form1;

//---------------------------------------------------------------------------
// Constructor
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
        // Setup the system
  ReadSpecs();
  states = pow(2,(float)(n-k));

  // 20-stage PN Generator
  int data_source_feedback[2] = {2,19};
  DataSource = new PN_Gen(20,2,data_source_feedback);

        // Create the coder
  HammingCoder = new Cyclic(n,k,gen_pol);

  // Create channel
```

```
    Pe = 0.1;
    Channel = new BSC(1,-1,Pe);
    No = Channel->Compute_No_needed_for_Pe();
    AWGN = new Noise(0,No);

    // Create the block code's trellis
    HammingTrellis = new Trellis(states,(n+1));
    HammingTrellis->BlockTrellis(n,k,parity);

    // Create the Viterbi decoder
    HammingDecoder = new Viterbi(states,(n+1));

    // Create a few buffers
    message = new Matrix(1,k);
    received = new Matrix(1,n);
    code = new Matrix(1,n);

    // Display startup Pe and associated Eb/No
    Edit1->Text=(10*log10((float)(n)/(float)(k*No)));
    Edit5->Text=Pe;

    // Initial node spacing factors
    zi = (Image1->Width)/(4*(n+2));
    zo = (Image1->Height)/(4*(states+1));

    // Display the startup trellis
    DrawTrellis();
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
void TForm1::ReadSpecs()
{
        FILE *specs;
    char temp[30];

    OpenDialog1->Filter = "Block code specs (*.bcs)|*.BCS";
        if (OpenDialog1->Execute())
    {
        specs = fopen((OpenDialog1->FileName).c_str(),"rt");

            // Read n from file
            fscanf(specs,"%s",temp);
            fscanf(specs,"%i",&n);

            // Read k from file
            fscanf(specs,"%s",temp);
            fscanf(specs,"%i",&k);

            // Read generator polynomial from file
```

```
            fscanf(specs,"%s",temp);
            gen_pol = new int[n-k+1];
            for (int i=0; i<(n-k+1); i++)
            {
                  fscanf(specs,"%s",temp);
                  gen_pol[i] = atoi(temp);
            }

            // Read the H matrix from file
            fscanf(specs,"%s",temp);
            VECALLOC(parity,(n-k),n);
            for (int i=0; i<(n-k); i++)
            {
                        for (int j=0; j<n; j++)
               {
                  fscanf(specs,"%s",temp);
                              parity[i][j] = atoi(temp);
                     }
                  }

                  fclose(specs);
   }
   else exit(0);
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Display the trellis information
void TForm1::DrawTrellis()
{
   // Determine spacing between nodes
   int x_step = zi;
   int y_step = zo;

   // Refresh the canvas
         Image1->Canvas->Pen->Color=clBlack;
         Image1->Canvas->Brush->Color=clWhite;
   Image1->Canvas->Rectangle(0,0,Image1->Width,Image1->Height);

   // Draw nodes
   Image1->Canvas->Brush->Color=clBlack;
   for (int m=1; m<=(n+1); m++)
   {
         for (int l=1; l<=states; l++)
         Image1->Canvas->Ellipse((m*x_step-3),(l*y_step-3),(m*x_step+3),(l*y_step+3));
   }
   Image1->Canvas->Brush->Color=clWhite;

   // Draw the branches of the trellis
```

```
  for (int j=0; j<n; j++)
  {
          for (int m=0; m<states; m++)
          {
                for (int l=0; l<states; l++)
              {
                Image1->Canvas->MoveTo((j+1)*x_step, y_step*(m+1));
                if (HammingTrellis->OutgoingBranches(j,m,l)==1)
                { if (HammingTrellis->ReturnBranchCause(j,m,l)==0)
                  Image1->Canvas->Pen->Color=clBlue;
                  else Image1->Canvas->Pen->Color=clRed;
                  Image1->Canvas->LineTo((j+2)*x_step, y_step*(l+1));
                        }
              }
          }
  }

  Image1->Canvas->Pen->Color=clBlack;
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Display the computed metrics
void TForm1::ShowMetrics()
{
        char metric[4];
        Image1->Canvas->Pen->Color=clBlack;

   // Determine the spacing between metrics
        int x_step = zi;
   int y_step = zo;

   // Write metris to the canvas
   for (int m=1; m<=(n+1); m++)
   {
        for (int l=1; l<=states; l++)
     {
        if (HammingTrellis->ReturnStateActive(m-1,l-1))
        {
        sprintf(metric,"%2.2f",(HammingDecoder->ReturnMetric(l-1,m-1)));
          Image1->Canvas->TextOut((m*x_step),(l*y_step-20),metric);
        }
     }
   }
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
```

```
// Display the most probable path
void TForm1::DrawPath()
{
        // Redraw the trellis
        DrawTrellis();
        Image1->Canvas->Pen->Color=clGreen;
   Image1->Canvas->Pen->Width=3;

   // Draw best path
   int x_step = zi;
   int y_step = zo;

   for (int j=0; j<n; j++)
   {
           for (int m=0; m<states; m++)
           {
           for (int l=0; l<states; l++)
              {
                 Image1->Canvas->MoveTo((j+1)*x_step, y_step*(m+1));
                 i                                                    f
(HammingDecoder->ReturnPath(m,j)*HammingDecoder->ReturnPath(l,j+1)*HammingTrellis-
>OutgoingBranches(j,m,l))
              Image1->Canvas->LineTo((j+2)*x_step, y_step*(l+1));
        if  (HammingDecoder->ReturnPath(m,j)==1)
Image1->Canvas->Ellipse(((j+1)*x_step-5),((m+1)*y_step-5),((j+1)*x_step+3),((m+1)*y_step+5));
              }
           }
   }

   Image1->Canvas->Pen->Color=clBlack;
   Image1->Canvas->Pen->Width=1;
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Transmit and receive a single code word
void __fastcall TForm1::Button1Click(TObject *Sender)
{
   AnsiString c;
   AnsiString r;
   AnsiString v;
   float receive[15];
   float noise;

   // Flush shift register based coder
        HammingCoder->Flush();

   // Construct message vector
   for (int x=0; x<k; x++)
   {
```

```
        message->ChangeElement(0,x,DataSource->Output(0));
        DataSource->Shift();
    }

    // Construct code word
    HammingCoder->Compute(message);

    // Buffer the code word that is going to be transmitted
    for (int y=0; y<n; y++)          code->ChangeElement(0,y,HammingCoder->Return_code_bit(y));

    // Transverse channel
    for (int y=0; y<n; y++)
    {
          noise = AWGN->Output(1);

received->ChangeElement(0,y,Channel->Transverse_Channel((float)(2*code->ReturnElement(0
,y)-1)+noise));

        // Differentiate between hard decision and soft decision decoding approaches
        if (RadioGroup1->ItemIndex==0) receive[y] = (float)received->ReturnElement(0,y);
            if (RadioGroup1->ItemIndex==1) receive[y] = 2*code->ReturnElement(0,y)-1+noise;
    }

    // Decode the received code word
    HammingDecoder->Decode(receive,HammingTrellis);

    // Draw the best path as determined by the Viterbi decoder
    DrawPath();

    // Display computed metrics if necessary
    if (CheckBox1->State==1) ShowMetrics();

    // Display all info
          for (int y=0; y<n; y++)
    {
          c += code->ReturnElement(0,y);
                if (RadioGroup1->ItemIndex==0)  r += ((received->ReturnElement(0,y)+1)/2);
             else  r = "";
        v += HammingDecoder->ReturnCodeBit(y);
    }

          Edit2->Text=c.c_str();
    Edit3->Text=r.c_str();
    Edit4->Text=v.c_str();
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Expurgate the trellis
void __fastcall TForm1::Button2Click(TObject *Sender)
```

```
{
        HammingTrellis->Expurgate();
  DrawTrellis();
  DrawPath();
  if (CheckBox1->State==1) ShowMetrics();
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Unexpurgate the trellis
void __fastcall TForm1::Button3Click(TObject *Sender)
{
  // Re-create the block code's trellis
  HammingTrellis->BlockTrellis(n,k,parity);

  // Draw the unexpurgated trellis
  DrawTrellis();
  DrawPath();

        // Display metrics if necesaary
  if (CheckBox1->State==1) ShowMetrics();
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Compute the Pe associated with a give noise power
float TForm1::Get_Pe(float N)
{
  float pe=0;
  float y=0;

  for (int i=0; i<1000; i++)
  {
    pe += (0.01/sqrt(3.1415*N))*exp(-(y+1)*(y+1)/N);
    y += 0.01;
  }
  return pe;
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Transmit x code words at different Eb/No values
void __fastcall TForm1::Button6Click(TObject *Sender)
{
  float receive[30];
  float noise;
  float error;
  int temp;
```

```
FILE *output;

output = fopen("results.txt","wt");

// Turn displays off
Edit2->Enabled = false;
Edit2->Update();
Edit3->Enabled = false;
Edit3->Update();
Edit4->Enabled = false;
Edit4->Update();
Edit5->Enabled = false;
Edit5->Update();

        // Step through the given range of Eb/No
          for (float step=atof(Edit6->Text.c_str()); step<=atof(Edit7->Text.c_str());
step+=atof(Edit8->Text.c_str()))
  {
        No = ((float)(n)/(float)(k))/(pow(10,(step/10)));
        AWGN->Change_variance(No/2);
     Pe=Get_Pe(No);
        error=0;
     Edit1->Text=(10*log10((float)(n)/(float)(k*No)));
     Edit1->Update();

     // Step through the given amount of code words
          for (float i=0; i<atoi(Edit9->Text.c_str()); i++)
          {
                ProgressBar1->Position = (i/atoi(Edit9->Text.c_str()))*100;

        // Flush the shift register based coder
                        HammingCoder->Flush();

                // Construct message vector
                for (int x=0; x<k; x++)
                {
                   message->ChangeElement(0,x,DataSource->Output(0));
                   DataSource->Shift();
                }

                // Construct code word
                HammingCoder->Compute(message);
                              for (int y = 0; y < n; y++)
code->ChangeElement(0,y,HammingCoder->Return_code_bit(y));

                // Transverse channel
                for (int y=0; y<n; y++)
                {
                      noise = AWGN->Output(1);

received->ChangeElement(0,y,Channel->Transverse_Channel((float)(2*code->ReturnElement(0
```

```
,y)-1)+noise));

        if (RadioGroup1->ItemIndex==0) receive[y] = (float)received->ReturnElement(0,y);
                                        if   (RadioGroup1->ItemIndex==1)   receive[y] =
2*code->ReturnElement(0,y)-1 + noise;
                }

        // Decode the received code word
        HammingDecoder->Decode(receive,HammingTrellis);

        // Count errors
        temp = 0;
                                        for   (int   y=0;   y<n;   y++)   temp   +=
HammingDecoder->ReturnCodeBit(y)^(code->ReturnElement(0,y));
        error += (temp>0);
           }
     fprintf(output,"%f %f\n",10*log10((float)(n)/(float)(k*No)),error/atof(Edit9->Text.c_str()));
}
  fclose(output);
        Edit2->Enabled = true;
  Edit3->Enabled = true;
  Edit4->Enabled = true;
  Edit5->Enabled = true;
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Compute the new Eb/No when Pe is changed
void __fastcall TForm1::Changing_Pe(TObject *Sender, WORD &Key, TShiftState Shift)
{
        if (Key == 13)
  {
        Pe = atof(Edit5->Text.c_str());
           Channel->Change_Pe(Pe);
           No = Channel->Compute_No_needed_for_Pe();
           Edit1->Text=(10*log10((float)(n)/(float)(k*No)));
  }
}
//---------------------------------------------------------------------------



//---------------------------------------------------------------------------
// Refresh shown Pe
void __fastcall TForm1::Exit_Pe_Change(TObject *Sender)
{
        Edit5->Text = Pe;
}
//---------------------------------------------------------------------------
```

```
//---------------------------------------------------------------------------
// Toggles between displaying the metrics and not displaying the metrics
void __fastcall TForm1::Metric_State(TObject *Sender)
{
        if (CheckBox1->State == 1) ShowMetrics();
  else {DrawTrellis(); DrawPath();}
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Zoom in
void __fastcall TForm1::Test1Click(TObject *Sender)
{
        // Adjust spacing between nodes
        zi *= 2;
  zo *= 2;

  // Redisplay all
  DrawTrellis();
  DrawPath();
        if (CheckBox1->State == 1) ShowMetrics();
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Zoom out
void __fastcall TForm1::ZoomOut1Click(TObject *Sender)
{
        if ((zo>10)*(zi>10))
  {
        zi /= 2;
    zo /= 2;
  }

  DrawTrellis();
  DrawPath();
        if (CheckBox1->State == 1) ShowMetrics();
}
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Copy current displayed trellis, path and metrics to the clipboard
void __fastcall TForm1::Dowhat1Click(TObject *Sender)
{
        Clipboard()->Assign(Image1->Picture);
}
//---------------------------------------------------------------------------
```