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.

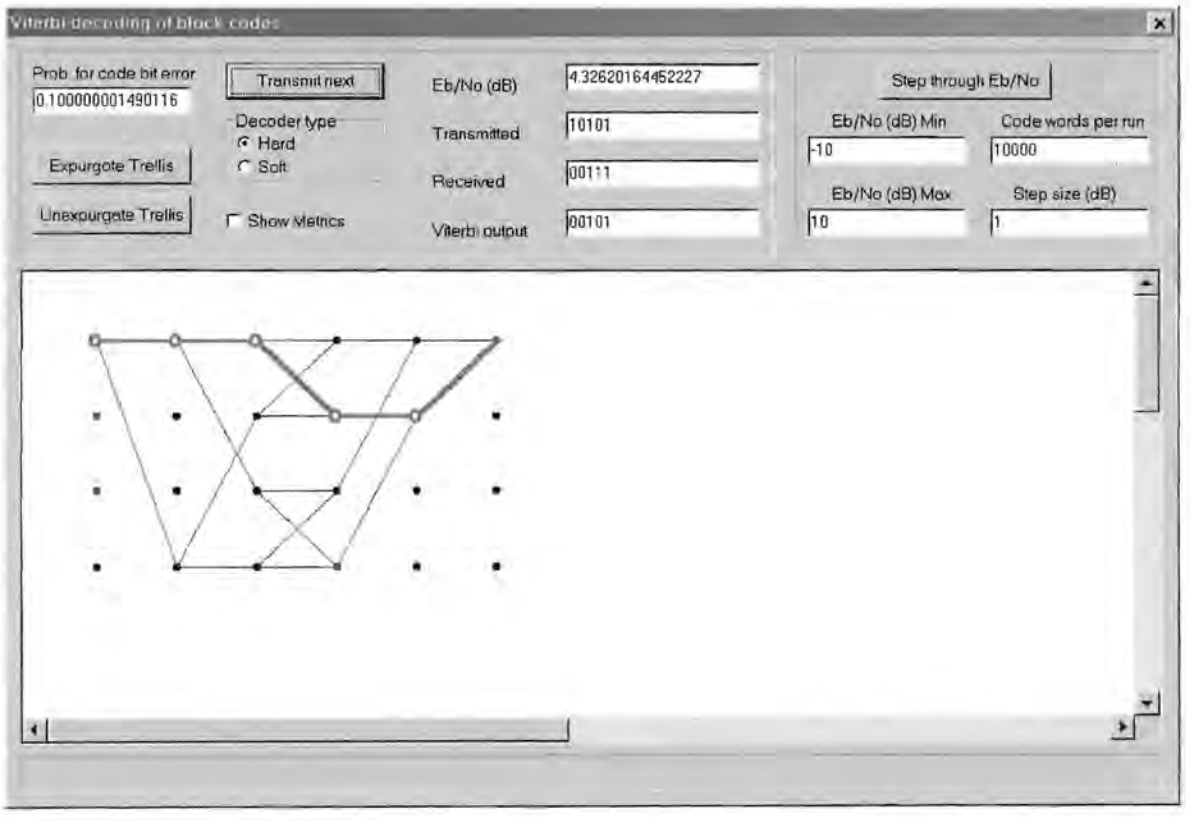

Figure 42 (5,3) Code Trellis for Parity Check matrix H_1 Produced by Trellis Simulation Software

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.

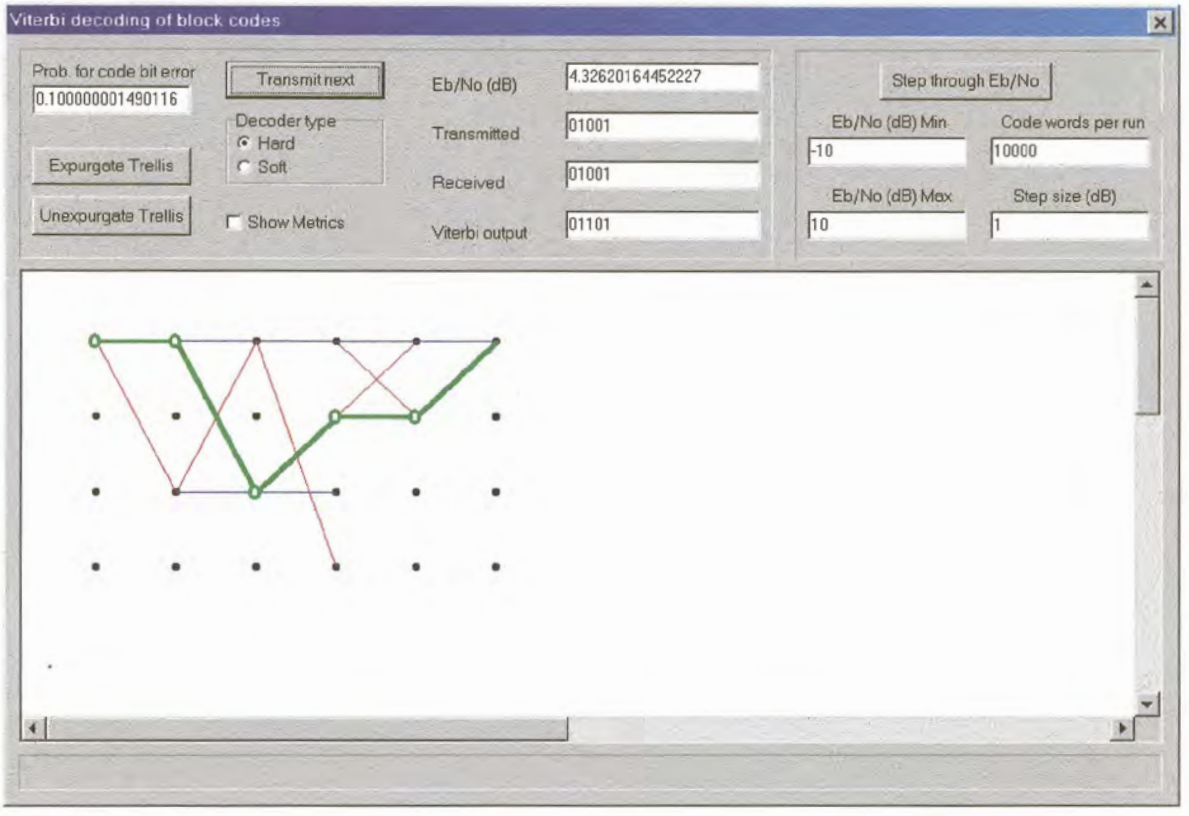

Figure 43 (5,3) code Trellis for Parity Check Matrix H_2 produced by Trellis Simulation Software

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.

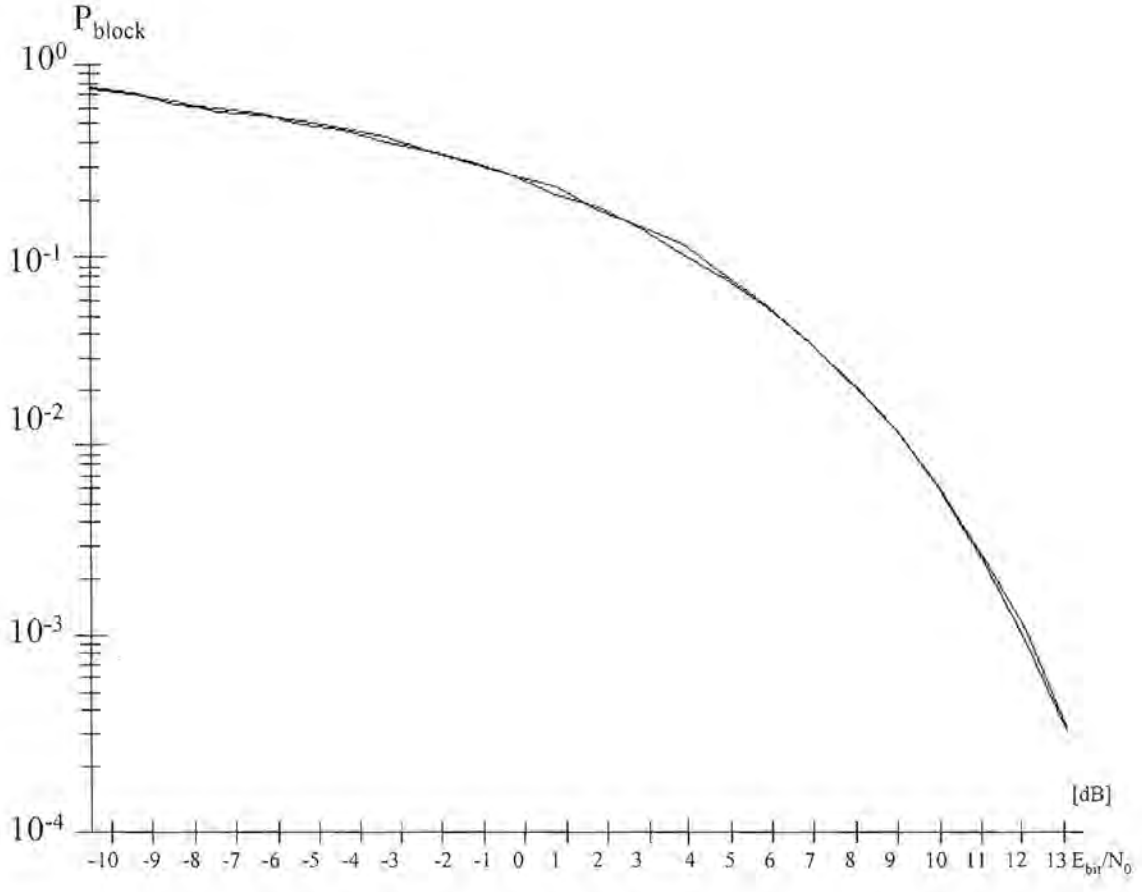

Figure 44 Comparison of Reduced and Original Block Error Rates

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.

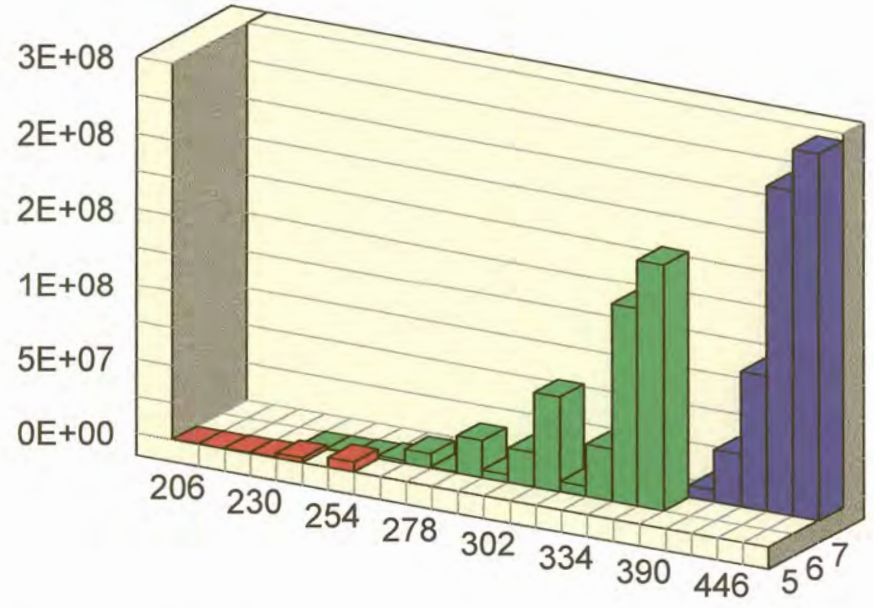

Figure 45 Graphical representation of N_Σ and ρ(C) for (15,7)-BCH code

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.