APPENDICES

APPENDIX A

A TUTORIAL ON TCM

This appendix presents a tutorial on trellis codes as applied to Q^2PSK. Specifically, the general concepts of trellis coded modulation are discussed, including some fundamental principles, decoding and evaluation.

A.1 INTRODUCTION

In this appendix the techniques of combined modulation and coding, known as Trellis Coded Modulation (TCM) are introduced. Over the last 2 decades TCM has emerged as an efficient and reliable coding scheme used for data communication with the purpose of gaining noise immunity over uncoded transmission without altering the data rate. Stated in other words, TCM is a combined coding and modulation scheme for improving the reliability of a digital transmission system without increasing the transmitted power or the required bandwidth. In a power-limited environment, the desired system performance should be achieved with the smallest possible power. One solution is the use of error-correcting codes, which increase the power efficiency by adding extra bits to the transmitted symbol sequence. This procedure requires the modulator to operate at a higher data rate and hence requires a larger bandwidth. In a bandwidth-limited environment, increased efficiency in frequency utilization can be obtained by choosing higher-order modulation schemes (e.g. 8-PSK instead of QPSK), but larger signal power would be needed to maintain the same signal separation and hence the same error probability.

The idea is indeed not new, since multilevel modulation of convolutionally encoded symbols was a known concept before the introduction of TCM. The innovative aspect of TCM is the concept that convolutional encoding and modulation should not be treated as separate entities, but rather, as a combined operation. The parameter governing the performance of the transmission system is consequently not the *free Hamming distance* of the convolutional code, but rather, over the additive Gaussian noise channel, the *free Euclidean distance* between transmitted signal sequences. Thus

the optimization of the TCM design will be based on Euclidean distances rather than on Hamming distances, so that the choice of the code and of the signal constellation will both be performed separately. Finally, the detection process will involve *soft* rather than *hard* decisions (i.e., received signals are processed before making decisions as to which transmitted symbol they correspond.)

A.1.1 Fundamentals of TCM

Assume here a discrete-time, continuous-amplitude model for the transmission of data on the additive white Gaussian noise channel. In this communications model, independently introduced by the Russian scholar Kotel'nikov and by C.E. Shannon, the messages to be delivered to the user are represented by points, or vectors, in an N-dimensional Euclidean space $\mathbf{R}^N$, called the *signal space*. When the vector x is transmitted, the received signal is represented by the vector

$$z = x + \nu \tag{A.1}$$

where ν is a noise vector whose components are independent Gaussian random variables with mean zero and the same variance $N_o/2$. The vector x is chosen from Ω' consisting of M' vectors, known as the *signal constellation*. The average square length

$$E' = E_{s,u} = \frac{1}{M'} \sum_{x \epsilon \Omega'} \| x \|^2 \tag{A.2}$$

will be referred to as the average signal *energy* of the uncoded transmission. This parameter is very important when analyzing TCM schemes, since it is used in quantifying the effective coding gain of the coding scheme.

Consider now the transmission of a sequence $\{x_i\}_{i=1}^{L}$ of L signals, where the subscript i denotes discrete time. The receiver that minimises the average error probability over the sequences operates as follows. It first observes the received signal $y_1, \ldots, y_L$, and then decides that $X_1, \ldots, X_L$ was transmitted if the squared Euclidean distance

$$d^2 = \sum_{i=1}^{L} \| y_i - x_i \|^2 \tag{A.3}$$

is minimised for $x_i = X_i, i = 1, \ldots, L$, or, in other words, if the sequence $X_1, \ldots, X_L$ is closer to the received sequence than to any other allowable signal vector sequence. The resulting error probability, as well as the symbol error probability, is upper bounded, at least for high SNR, by a decreasing function of ratio d^2_{min}/N_o, where d^2_{min} is the Minimum Squared Euclidean Distance (MSED) between two allowable signal vector sequences.

One way of improving the system performance is that of removing the assumption that the signals are independent. This can be done by restricting the transmitted sequences to a subset of Ω'^K. Now, to do this, the transmission rate will also be reduced. To avoid this unwanted reduction, one may choose to increase the size of Ω'. For example, if Ω' is changed into $\Omega \supset \Omega'$ and M' into $M.M'$, and select M'^K sequences as subset of Ω^K, one can have sequences that are less tightly packed and hence increase the minimum distance between them.

In conclusion, a minimum distance d_{free} between two possible sequences is obtained that turns out to be greater than the minimum distance d_{min} between signals in Ω', i.e., the constellation from which they were drawn. Hence use of ML sequence detection will yield a *distance gain* d^2_{free}/d^2_{min}.

On the other hand, to avoid a reduction of the value of the transmission rate, the constellation is expanded from Ω' to Ω. This may entail an increase in the average signal energy from $E' = E_{s,u}$ to $E = E_{s,c}$, and hence an "energy loss" $E/E' = E_{s,c}/E_{s,u}$. Thus the *asymptotic coding gain* of a TCM scheme is defined as

$$\gamma_c = \frac{d^2_{free,c}/E_{s,c}}{d^2_{free,u}/E_{s,u}} = 10 \log_{10} \frac{d^2_{free,c}}{d^2_{free,u}} \tag{A.4}$$

where $E_{s,u}$ and $E_{s,c}$ are the average energies associated with uncoded and coded transmission, respectively.

The introduction of interdependencies among the signals and the expansion of the signal set are two of the basic ideas underlying trellis-coded modulation, another is *set partitioning* introduced by Ungerboeck [27, 29].

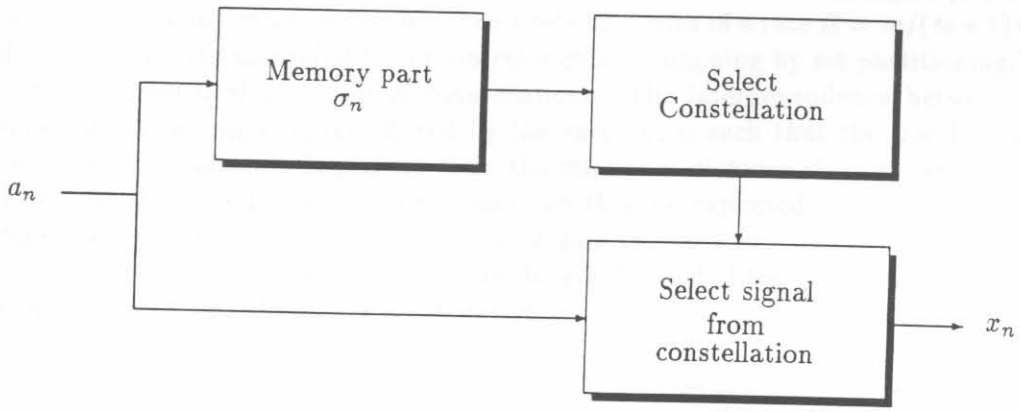

Figure A.1: General model for TCM.

Assume that the signal x_n transmitted at discrete time n depends not only on the source symbol a_n transmitted at the same time instant (as it would be with memoryless modulation), but also on a finite number of previous source symbols:

$$x_n = f(a_n, a_{n-1}, \ldots, a_{n-L}). \tag{A.5}$$

By defining

$$\sigma_n = (a_{n-1}, \ldots, a_{n-L}). \tag{A.6}$$

as the *state* of the encoder at time n, can be rewritten (A.5) in the more compact form

$$\begin{aligned} x_n &= f(a_n, \sigma_n) \\ \sigma_{n+1} &= g(a_n, \sigma_n). \end{aligned} \tag{A.7}$$

Equations A.7 can be interpreted as follows. The function $f(\cdot,\cdot)$ describes the fact that each channel symbol depends not only on the corresponding source symbol, but also on the encoder state parameter, σ_n. In other words, at any time instant the transmitted symbol is chosen from a constellation that is selected by the encoder state, σ_n at time instant n. The function $g(\cdot,\cdot)$ describes the *memory part* of the encoder and shows the evolution of the modulator states. The general model for TCM is shown in Figure A.1.

A.2 HEURISTIC AND ANALYTIC REPRESENTATION OF TCM

In this section the two different approaches to code design are examined, the first proposed by Ungerboeck, consisting of modeling the memory part of the TCM encoder through a convolutional encoder [27, 29], and the second introduced by Calderbank and Mazo [31, 30], where the output channel signals are directly expressed in terms of a sliding block of input bits.

A.2.1 Ungerboeck Codes

Ungerboeck proposed a channel coding technique that achieves remarkable coding gains, without sacrificing the data rate or expanding the bandwidth of the transmitted signal [27, 29]. The basic idea is to encode m information bits into $m+1$ bits by means of a rate $R = m/(m+1)$ convolutional encoder, which selects according to certain rules called "mapping by set partitioning," points from one of the conventional 2^{m+1} signal constellations. The interdependence between the resulting sequences of channel signals, introduced by the encoder, is such that the free ED, d_{free} between any two possible sequences is greater than the minimum distance d_{min} between any two points in the 2^m signal constellation. Such memory can then be exploited by the ML decoder, yielding a coding gain, γ_c defined in (A.4). The coding gain (γ_c) is a function of the amount of memory introduced by the encoder, i.e., the constraint length L_c, and of the positioning of the signal points in the signal space, i.e., the signal constellation.

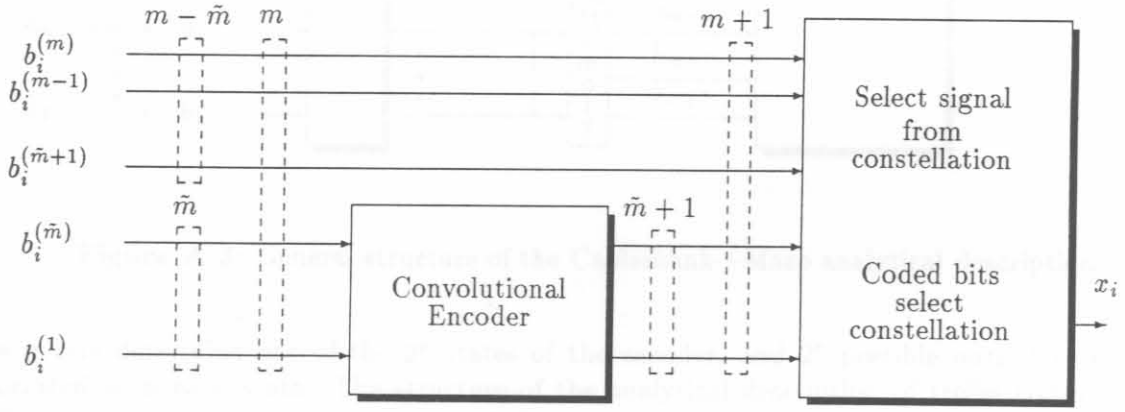

Figure A.2: Block diagram of an Ungerboeck code ($m = \log_2 M$).

It is convenient to represent an Ungerboeck code by the scheme depicted in Figure A.2. At every time instant i, the rate $\tilde{m}/(\tilde{m}+1)$ convolutional encoder receives $\tilde{m}$ input bits and generates $\tilde{m}+1$ coded bits. These, in return, determine the subconstellation from which the transmitted signal has to be chosen. Furthermore, these coded bits move the encoder to the next state. In the figure, the $m - \tilde{m}$ source bits left uncoded are explicitly shown. The presence of uncoded bits causes parallel transitions.

Ungerboeck's codes or variations of these codes have received a great deal of attention in that they are readily implemented, and considered as standards for high-speed modems on the voiceband

data channel.

A.2.2 Calderbank—Mazo Codes

The analytic description of the trellis codes was developed for one dimensional modulation, and channel signals have to be real numbers for the analytic process. Basically, the output channel signals are directly expressed in terms of a sliding block of input bits, with the intermediate step of output coded bits being irrelevant for analytical described trellis codes.

When a trellis code is used to encode data at a rate of r bit per channel symbol, each channel input x will depend not only on the most recent block of k bits that enter the encoder but will also depend on the ν bits preceding this block. Formally [31, 30]:

$$X_j = x(a_{jr}, a_{jr-1}, \ldots, a_{jr-(r-1)}; a_{(j-1)r}, \ldots, a_{(j-1)-(\nu-1)}) \tag{A.8}$$

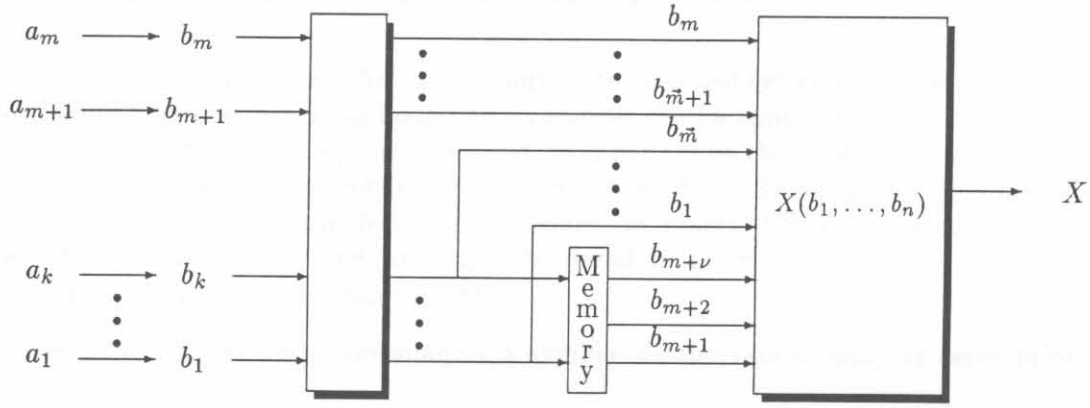

Figure A.3: General structure of the Calderbank—Mazo analytical description

The ν bits determine one of the 2^ν states of the encoder, and 2^r possible output symbols are associated with each state. The structure of the analytical description of trellis codes is shown in Figure A.3, and then the output symbol may be determined by solving the set of equations resulting from all the input bit combinations:

$$\begin{aligned} x(b_1, b_2, \ldots, b_n) &= \sum_{i=1}^{n} d_i b_i + \sum_{i,j=1;j>i}^{n} d_{ij} b_i b_j + \\ &+ \sum_{i,j,l=1;l>j>i}^{n} d_{ijl} b_i b_j b_l + \ldots + d_{12\ldots n} b_1 b_2 \ldots b_n \end{aligned} \tag{A.9}$$

or by solving its equivalent matrix equation, describing the encoder function as follows:

$$X = Bd \tag{A.10}$$

where $b_i = 1 - 2a_i$, X is the channel signal matrix, B is the Hadamard matrix with elements ± 1 and d the matrix of constants which determines the code.

In order to calculate the coefficients of the matrix d, the matrix of the channel symbols is calculated from the trellis diagram and the solution for d is obtained from:

$$d = \frac{B^T X}{2^n} \tag{A.11}$$

where X is the channel signal matrix, B is the Hadamard matrix with elements ± 1, and d is the matrix of constants to be determined. In (A.10) one has $X = X(b_1, b_2, \ldots, b_n)$ which present the modulation output X for the input information and memory bits $b_1, b_2, \ldots, b_n$, which denote the k-th row of the $(1 \times n')$ matrix B given by $B_k = (b_1, b_2, \ldots, b_n, b_1 b_2, \ldots, b_1 b_2 \ldots b_n)$; d^T will be a $(1 \times n')$ matrix given by $d^T = (d_1, d_2, \ldots, d_n, d_{12}, \ldots, d_1 d_2 \ldots d_n)$, where $n' = 2^n - 1$, n being the number of input bits per interval plus the number of memory bits of the encoder.

The discussion of the Calderbank—Mazo algorithm, together with the C software listing can be found in Appendix B.

A.3 PERFORMANCE EVALUATION

In order to obtain bounds on the error performance for coded systems used on memoryless channels, the Chernoff or Bhattacharyya bounding techniques can be applied to trellis coded communication systems [30]. In the following, the upper and lower bounds on the error probability of a coded system are discussed, based on the transfer function approach [75, 33, 30]. Channel State Information (CSI) may also be available in the decoding procedure. In general CSI is the information derived from the channel that can be used to design the decoding metric to give improved performance for communication over fading channels [81].

In order to simplify performance analysis, a number of important simplifying assumptions are made:

- First, assume perfect coherent detection and consider the effects of amplitude fading only,
- In order to ensure independent fading of adjacent demodulated signals, infinite interleaving/deinterleaving depth is assumed. This permits a memoryless channel approximation, which in turn makes *analytical* performance analysis feasible.
- Third, an infinite decoding delay is assumed in the decoding (Viterbi) process.
- Finally, it is assumed that fading over a single signalling interval may be represented by a single fading amplitude.

The assumptions made here are consistent with those in [61].

A.3.1 Analytical Upper Bound to the Error Probability of Q^{2}PSK

The discussion is started by expressing a coded sequence of transmitted Q^{2}PSK channel signals of length L by $\mathbf{x_L} = (\overline{x_1}, (\overline{x_2}, \ldots, (\overline{x_L})$, where $\overline{x_k}$, $k = 1, 2, \ldots, L$, are Four-Dimensional (4D) representations of the allowable Q^{2}PSK channel signals. Corresponding to $\mathbf{x_L}$, is the channel output sequence $\mathbf{y_L} = (\overline{y_1}, (\overline{y_2}, \ldots, (\overline{y_L})$ where $\overline{y_k}$, $k = 1, 2, \ldots, L$, is the 4D continuous random variable,

$$\overline{y_k} = \overline{\rho_k}\,\overline{x_k} + \overline{n_k} \tag{A.12}$$

where $\overline{\rho_k}$ represents a (normalised) random fading amplitude, and where $\overline{n_k}$ is a 4D additive Gaussian noise process with zero mean and variance σ^2 in each of the four dimensions.

Based on a suitable decoding metric, the receiver will make a decision on the transmitted sequence $\mathbf{x_L}$, which is called $\mathbf{x'_L}$, which leads to the definition of a pairwise error probability [61],

$$P_e(\mathbf{x_L} \rightarrow \mathbf{x'_L}) = P_r(m(\mathbf{y_L}, \mathbf{x'_L}) \geq m(\mathbf{y_L}, \mathbf{x_L}) \mid \mathbf{x_L}) \tag{A.13}$$

where

$$\begin{aligned} m(\mathbf{y_L}, \mathbf{x_L}) &= \sum_{k=1}^{L} m(\overline{y_k}, \overline{x_k}) \\ m(\overline{y_k}, \overline{x_k}) &= \begin{cases} -\mid \overline{y_k} - \overline{\rho_k}\,\overline{x_k} \mid^2 & \text{, ideal CSI} \\ -\mid \overline{y_k} - \overline{x_k} \mid^2 & \text{, no CSI} \end{cases} \end{aligned} \tag{A.14}$$

is the decoding metric [61, 30].

In the case of ideal CSI, the receiver is assumed to have exact knowledge of $\overline{\rho_k}$, $k = 1, 2, \ldots, L$, and the decoding metric is then ML. On the other hand, when no CSI is available at the receiver, the decoding metric is no longer ML and this introduces an additional weakness into the bounding procedure [94, 95].

A tight upper bound on the average bit error probability can be obtained from

$$P_b \leq \frac{k_0}{bN} T(D) \mid_{D=Z} \tag{A.15}$$

where N is the number of trellis states; b is the number of information symbols associated with each branch in the trellis; k_0 is a factor that depends on the type of channel, the type of demodulation and the code structure; and $Z = \exp\{-E_s/4N_o\}$ is the so-called Bhattacharyya parameter. For AWGN with optimum coherent demodulation, k_0 is given by

$$k_0 = Q\left\{\sqrt{\frac{E_s}{N_0} d^2_{free}}\right\} D^{-d^2_{free}} \tag{A.16}$$

which means to compute k_0, knowledge of d_{free} is required, the minimum free distance. As a consequence, availability of fast algorithms to evaluate d_{free} is of crucial importance. The software listing of the program written in C to calculate d_{free} is presented in Appendix B.

In (A.15), $T(D)$ a function of D, is the (scalar, closed loop) *transfer function* of the error state diagram, , where its power generally represents the distance (Hamming or Euclidean) between two code words. In this study a new algebraic algorithm for the derivation of the transfer function for a trellis encoder, originally introduced by Chan and Norton [79], is utilised. This algebraic algorithm is discussed in detail in Appendix C. Alternatively, the computational algorithm introduced by Biglieri et al. [30], based on matrix-algebra could have been used with equally good affect.

From the analysis by Biglieri et al. [30] for the symbol error probability, the SEP is given as

$$P_s \leq T(D) \mid_{D=Z} \tag{A.17}$$

where $T(D) = \sum_{\overline{e}} W(\overline{e})$, $Z = exp(-E_b/N_o)$ is the so-called Bhattacharyya parameter, and $\overline{e}$ is the error vector. $T(D)$ is called the (scalar) *transfer function* of the error-state diagram.

Define $W(\overline{e})$ as the *weight profile* given by

$$W(\overline{e}) = \frac{1}{M} \sum_{\overline{c}_i}^{M} D^{d^2(\overline{c}_i, \overline{c}_i \oplus \overline{e})} \tag{A.18}$$

where M is the number of Q^{2}PSK symbols in a subset after set-partitioning, $\overline{c}_i$ denotes the M symbol vectors constituting the subset, and $d^2(\cdot)$ is the MSED between two symbol vectors separated by the error vector $\overline{e}$.

A better upper bound on P_s than (A.17) can be derived by knowledge of the free distance, d_{free} of the code and is given by

$$P_s \leq Q\left\{\sqrt{\frac{d_{free}^2 E_b}{N_o}}\right\} exp\left(\frac{d_{free}^2 E_b}{4N_o}\right) T(D)\,|_{D=Z} \tag{A.19}$$

From (A.17) the upper bound on the BEP is easily found and given by

$$P_b \leq \frac{1}{b} T(D)\,|_{D=Z} \tag{A.20}$$

where b is the number of source (input) bits per trellis transition.

Yet, another tighter upper bound can be found from (A.19), given by

$$P_b \leq \frac{1}{b} Q\left\{\sqrt{\frac{d_{free}^2 E_b}{N_o}}\right\} exp\left(\frac{d_{free}^2 E_b}{4N_o}\right) T(D)\,|_{D=Z} \tag{A.21}$$

A discussion of the Chan—Norton algebraic algorithm utilised for generating the transfer function of the trellis encoder may be found in Appendix B.

A.3.2 Lower Bound to Error Probability

An lower bound to the average bit error probability is given by

$$P_b \geq \frac{\psi}{m} Q\left\{\frac{d_{free}}{\sqrt{N_o}}\right\} \tag{A.22}$$

where ψ is the probability that at any time, a trellis path chosen at random has another path splitting from it at that time, and remerging later, such that the Euclidean distance is d_{free}. In (A.22) m denotes the number of information bits at any node.

A.3.3 Union Bound techniques

Before proceeding with the union bound, consider the case of comparing the correct path with another path that differs from it in q positions. Denote the probability of error in this comparison by P_q. For the BSC with channel symbol error rate ρ, the probability of error in this computation is simply the probability that more than $q/2$ errors occur plus $1/2$ the probability that $q/2$ error occur [24].

An union bound on the probability of a first-event error, P_e at branch i may be obtained by summing the error probabilities for all possible paths that merge with the all-zero path at this point. This overbound is then given by

$$P_e < \sum_{q=0}^{\infty} n_q P_q \tag{A.23}$$

Furthermore, an union bound on the probability of bit error, P_b may be obtained from(A.23) by weighting each term by the corresponding number of bit errors, i.e. the information weight for each path). However, for a rate-k/n code there are k symbols decoded on each branch. Thus, P_b is bounded by

$$P_b < \frac{1}{k} \sum_{q=0}^{\infty} w_q P_q \tag{A.24}$$

The evaluation of (A.23) and (A.23) requires knowledge of the path weight distribution of the code. One can also perform various manipulations with the transfer functions to provide approximate closed-form expressions for bounds on P_e and P_b. Observe that (A.23) and (A.24) would be identical if the term P_q could be written in the form a^q. This is possible for a few channels. For example, it can be shown that for the BSC, P_q is over-bounded by [96]:

$$P_j < \left\{2[\rho(1-\rho]^{1/2}\right\}^q \tag{A.25}$$

Thus, from (A.23) and (A.25) one may write the bound on the symbol error probability as

$$P_e < T(D,N)|_{N=1,D=2[\rho(1-\rho)]^{1/2}} \tag{A.26}$$

In a similar fashion, from (A.24) and (A.25) one can write the bit error probability as

$$P_b < \frac{1}{k} \left.\frac{\delta T(D,N)}{\delta N}\right|_{N=1,D=2[\rho(1-\rho)]^{1/2}} \tag{A.27}$$

A union bound which is valid for demodulator soft decisions may be similarly established [96]. Bounds similar to (A.26) and (A.27) can also be obtained for more general channel models than the BSC. For a binary input AWGN channel with no output quantization, the bounds respectively become

$$P_e < T(D,N)|_{N=1,D=e^{R_c E_b/N_o}} \tag{A.28}$$

and

$$P_b < \frac{1}{k} \left.\frac{\delta T(D,N)}{\delta N}\right|_{N=1,D=e^{R_c E_b/N_o}} \tag{A.29}$$

where R_c is the code rate.

A.4 DECODING TCM

In this section the Viterbi algorithm as applied to decoding TCM signals is considered. If the TCM signal is described by using a trellis, whose branches are associated with transitions between encoder states and with signals transmitted over the channel, the task of the TCM decoder is to

estimate the path that the encoded signal sequence traverses through the trellis. This is done by associating with each branch of the trellis a number, called the *branch metric*, and looking for the path whose total metric is minimum. This path corresponds to the transmitted sequence. Thus the decoding problem can be split into two parts:

- Definition of a branch metric, and its computation based on the observed values of the received signal.
- Evaluation of the minimum-metric path.

In this section, the Viterbi Algorithm (VA) is considered for decoding TCM. This decoding algorithm, introduced by Viterbi in 1967 [96], uses the trellis structure of the code and determines the ML estimate of the transmitted sequence that has the largest metric.

ML decoding implies finding the path with largest metric through the trellis by comparing the metrics of all branch paths entering each state with the corresponding level elements of the received sequence in an iterative manner. In the decoding process, if at some level it is found that a path cannot possibly yield the largest metric, then the path is discarded by the decoder. In this manner, a decoder that compares the metrics of all paths entering a state and only retains the survivor path at that state will yield a most likely path if the operation is repeated for all distinct states at each level. This simple iterative process is known as the Viterbi Algorithm (VA), described by Dr. A.J. Viterbi in the late 1960's [96].

Viterbi decoding consists fundamentally of three processes. The first step in the decoder process is to generate a set of correlation measurements, known as *branch metrics*, for each n-tuple of codewords input from the communication channel (where k and n are, respectively the number of bits input and output to a rate k/n convolutional coder. These branch metric values indicate the correlation between received codewords and the 2^m possible codeword combinations.

The Viterbi decoder determines the state of the L_c-bit (where L_c is the constraint length) memory at the encoder using a maximum likelihood technique. Once the value of the encoder is determined, the original information is known, since the encoder memory is simply information that is a function of the state (value) of the encoder. To determine the encoder state, the second step in the Viterbi algorithm generates a set of 2^{L_c-1} *state metrics* which are measures of the occurrence probability for each of the 2^{L_c-1} possible encoder memory states. For this reason, the exponential growth of the decoding effort is related to the encoder memory order, $L_c - 1$. As the state metrics are computed, a binary decision is formed for each of the 2^{L_c-1} possible states as to the most probable path taken to arrive at that particular state. These binary decision are stored in a *path memory*.

Step three computes the decoded output data. To do this, the *path* from the current state to some point in the finite past is traced back by *chaining* the binary decisions stored in the path memory during step 2 from state to state. The effects caused by noise to the one and only correct result are mitigated as the paths within the *chainback* memory converge after some history. The greater the *decoding depth* of the chainback process the more likely that the final decoded result will be error free. As a result, higher code rates and constraint lengths require longer chainback depths for best performance. The chainback memory of the Viterbi decoder traces the history of the previous states to arrive at the most probable state of the encoder in the past, and thus determine the transmitted data.

A.5 CONCLUDING REMARKS: APPENDIX A

In the classical approach to channel encoding, the two functions of coding and modulation are regarded as separate operations. It was seen that in TCM these two functions are combined at the trellis decoder. However, the code design goal is to exploit the redundancy (whether in the form of parity bits or a enlarged signal space) to maximize the minimum distance (Hamming or Euclidean) between different codewords.

APPENDIX B

TCM CODE DESIGN — Utility Software

The evaluation of the coefficients of the analytical description and finding d_{free} are the two main tasks in the design procedure, and compliments the results presented in Chapter 7. In this appendix a discussion of the analytical description of trellis codes proposed by Calderbank and Mazo, and the utility software developed for computing the coefficients of the analytical equation are presented. In the second part of this appendix the computational algorithm proposed by Mulligan and Wilson for the computation of d_{free} is considered. The software code written in *C* is discussed.

B.1 Calderbank—Mazo Description of trellis codes

The general theory of the Calderbank—Mazo [31] formulation of trellis codes was presented in Appendix A The structure of the analytical description of trellis codes may be determined by solving its equivalent matrix equation, describing the encoder function as follows [30]:

$$X = BD \tag{B.1}$$

where $b_i = 1 - 2a_i$, X is the channel signal matrix, B is the Hadamard matrix with elements ± 1 and D the matrix of constants which determines the code. Calderbank and Mazo show that B is an orthogonal matrix. Therefore, in order to calculate the coefficients of the matrix D, the matrix of the channel symbols is calculated from the trellis diagram and the solution for D is obtained from:

$$D = \frac{B^T X}{2^n} \tag{B.2}$$

where X is the channel signal matrix, B is the Hadamard matrix with elements ± 1, and D is the matrix of constants to be determined. Here $n = k + \nu$, where k is the number of input bits and ν is the number of memory elements for the trellis code.

In the following the computer software to calculate the coefficients of D is presented. The program is based on simple matrix algebra, from which an analytical description can be derived by means of a trellis representation of a code, by knowing the input bits and the corresponding signals in the modulation signal set.

```
1  // ======================================
2  // Program Name: ANALYTIC.CPP
3  //
4  // Description:  EVALUATION OF
5  //               CALDERBANK-MAZO ANALYTICAL
6  //               DESCRIPTION OF TCM
7  //
8  // ======================================

9  #include <iostream.h>
10 #include <conio.h>
11 #include <stdio.h>
12 #include <stdlib.h>
13 #include "matrix_d.h"        // Includes the matrix algebra routines
14 #include "mathstat.h"        // Includes general mathematical and statistical
15                                    // functions

16 void main(int argc, char *argv[])
17 {
18     Matrix  B, A;
19     Matrix  f, d;
20     FILE     *Config;   // Configuration file

21     int             i[K], P[K];
22     int             a, b, m, n, o, p, q, s, t, r;
23     int             N, R, C;
24     double  element;

25     clrscr();
26     if (argc != 2)
27     {
28         cout << "Usage: Analytic  Configuration_File";  exit(1);
29     }
30     Config = fopen(argv[1],"rt");
31     fscanf(Config, "%d", &N);
32     R = pow(2,N);
33     for (a = 0; a < K; a++)   P[a] = 999;
```

```
34      C = 0;
35      for (a = 1; a <= N; a++)
36      {
37          // C is the number of columns in matrix B
38          // (i.e. the number of possible solutions of D)
39          // It is given by
40          //      Sum of all the combinations of N and a, where a=1,...,N

41          P[a-1] = Combination(N,a);
42          C += P[a-1];
43      }
44      // Initialisation of matrices
45      B.Set_dimension(R,C);
46      A.Set_dimension(C,R);
47      f.Set_dimension(R,1);
48      d.Set_dimension(1,C);

49      // Obtain trellis structure from configuration file
50      for (a = 1; a <= R; a++)
51      {
52          fscanf(Config,"%d", &t);
53          f(a,1) = t;
54      }
55      fcloseall();  // Close configuration file
56      r = 1;
57      for (a = 0; a < N; a++)         // initialise the sliding block input-output
58                                          // bits to initial condition (-1,-1,....,-
59          i[a] = -1;
60      clrscr();
61      cout << "CALDERBANK-MAZO: Analytical Description of TCM coders (c)dvw96\n\n";
62      cout << "C: = " << C << "\n";
63      for (a = 0; a < N; a++)
64          cout << P[a] << " + ";
65      cout << "\n\n";

66      //================================
67      // INITIALISATION OF MATRIX --- B
68      //================================

69      for (a = R; a > 0; a--)
70      {
71          // L = 1
72          for (m = 0; m < N; m++)
73              B(a,r++) = i[m];
74          // L = 2
75          for (m = 0; m < N-1; m++)
76              for (n = m+1; n < N; n++)
77                  B(a,r++) = i[m]*i[n];
78          // L = 3
```

```
79          for (m = 0; m < N-2; m++)
80              for (n = m+1; n < N-1; n++)
81                  for (o = n+1; o < N; o++)
82                      B(a,r++) = i[m]*i[n]*i[o];
83          // L = 4
84          for (m = 0; m < N-3; m++)
85              for (n = m+1; n < N-2; n++)
86                  for (o = n+1; o < N-1; o++)
87                      for (p = o+1; p < N; p++)
88                          B(a,r++) = i[m]*i[n]*i[o]*i[p];
89          // L = 5
90          for (m = 0; m < N-4; m++)
91              for (n = m+1; n < N-3; n++)
92                  for (o = n+1; o < N-2; o++)
93                      for (p = o+1; p < N-1; p++)
94                          for (q = p+1; q < N; q++)
95                              B(a,r++) = i[m]*i[n]*i[o]*i[p]*i[q];
96          // L = 6
97          for (m = 0; m < N-5; m++)
98              for (n = m+1; n < N-4; n++)
99                  for (o = n+1; o < N-3; o++)
100                     for (p = o+1; p < N-2; p++)
101                         for (q = p+1; q < N-1; q++)
102                             for (s = q+1; s < N; s++)
103                                 B(a,r++) = i[m]*i[n]*i[o]*i[p]*i[q]*i[s];
104         // L = 7
105         for (m = 0; m < N-6; m++)
106             for (n = m+1; n < N-5; n++)
107                 for (o = n+1; o < N-4; o++)
108                     for (p = o+1; p < N-3; p++)
109                         for (q = p+1; q < N-2; q++)
110                             for (s = q+1; s < N-1; s++)
111                                 for (t = s+1; t < N; t++)
112                                     B(a,r++) = i[m]*i[n]*i[o]*i[p]*i[q]*i[s]*i[t];
113         for (m = 0; m < N; m++)
114         {
115             i[m] += 2;
116             if (i[m] == 1) break;
117             else i[m] = -1;
118         }
119         r = 1;
120     }
121     //====================
122     // SOLVE MATRIX --- D
123     //====================

124     d = T(T(B)*f/R);  // Solve for D
125     printf("\n\nSolution for coefficients of D:\n* ");
126     for (a = 1; a <= C; a++)
```

```
127          cout << d(1,a);
128       cout << "\n\n*** FIN ***";
129  }
```

B.2 Mulligan—Wilson Algorithm for computation of d_{free}

In this study it was shown to what extent the Euclidean free distance of a TCM and MTCM defines the asymptotic coding gain of the scheme. Furthermore, it was shown that d_{free} plays a central role in determining tighter bounds on system performance. It is stated in the book by Biglieri *et al.* that if a single parameter is to be used to assess the quality of a trellis coded scheme, d_{free} is the only sensible one that comes to mind. Therefore, it makes sense to look at an algorithm to compute this parameter.

The algorithm for computation of d_{free} is described, as derived by Mulligan and Wilson [80]. Our presentation here follows from [30] (pp. 128-131). Consider the trellis description of the TCM scheme. Every pair of branches in a section of the trellis defines one distance between the signals labeling the branches. If there are parallel transitions, every branch will be associated with an entire subconstellation (partitioned subset). In this case, only the minimum distance between any two signals extracted from the pair of subconstellations will be used. The squared distance between the signal sequences associated with two paths in the trellis is obtained by summing the individual squared distances. The algorithm is based on the update of the entries of a matrix D, which are the minimum squared distances between all pairs of paths diverging from any initial state and reaching a specific pair of states at discrete time n.

In the following we present the computer program to calculate d_{free}. The program requires the number of states N_s, the number of branches leaving a state or re-emerging at a state and the number of parallel transitions. The signals are sequentially assigned in an configuration input file, with the signal order corresponding exactly to the signal assignment of the trellis. Following the signals, there are N_s additional entries, specifying the trellis structure (e.g. fully, half- or quarter-connected trellis).

```
1  // ======================================
2  // Program Name: DMIN.CPP
3  //
4  // Description:  EVALUATION OF MINIMUM FREE
5  //               EUCLIDEAN DISTANCE
6  //
7  // ======================================

8  #include <conio.h>
9  #include <stdio.h>
10 #include <stdlib.h>
11 #include <iostream.h>
12 #include <math.h>
13 #include <mem.h>
```

```
14  #define    FALSE        0
15  #define    TRUE         1

16  // =======
17  // GLOBALS
18  // =======
19  float        **X, ***PT, **CDT, **TEMP;

20  // ==============
21  // PROCEDURE MAIN
22  // ==============
23  void main(int argc, char *argv[])
24  {
25      FILE     *Config;
26      int        S, K, KT, D;
27      int        TOTAL1, TOTAL2, TOTAL3, TOTAL4, TOTAL5, TOTAL6;
28      unsigned int    FIRST, STILL;
29      int        TD, SN;
30      int        C1, C2, C3;            // Counters
31      int        p, q, Y, Z, ROWPT, ROW, COL;
32      float        MIN, DIST, DMIN, TEMPO;
33      int        i, j, k, l, m, n, r, Itemp;
34      float        Ftemp;

35      clrscr();
36      if (argc != 2)
37      {
38          cout << "Usage: Dmin  Configuration_File";
39          exit(1);
40      }
41      cout << "Mullligan & Wilson Computational Algorithm";
42      cout << " - (c) dvw96";
43      cout << "\n\nAssuming Configuration File: " << argv[1];
44      Config = fopen(argv[1],"rt");
45      fscanf(Config, "%d", &S);
46      fscanf(Config, "%d", &K);
47      fscanf(Config, "%d", &KT);
48      fscanf(Config, "%d", &D);

49      // ==============
50      // INITIALISATION
51      // ==============
52      TOTAL1 = S*pow(2,K);
53      TOTAL2 = S*pow(2,KT);
54      TOTAL3 = pow(2,KT);
55      TOTAL4 = pow(2,K-KT);
56      TOTAL5 = pow(2,2*KT);
57      TOTAL6 = S*S;
```

```
58      // =================
59      // MEMORY ALLOCATION
60      // =================
61      Memory_Allocation();

62      // X[TOTAL1][D]
63      // CDT[S][S]
64      // TEMP[S][S]
65      // PT[TOTAL6][TOTAL5][2]


66      Y = Z = 0;          // Position indexes

67      // ===================================
68      // Input channel signals into matrix X
69      // ===================================

70      for (i = 0; i < TOTAL1; i++)
71          for (j = 0; j < D; j++)
72          {
73              fscanf(Config, "%f", &Ftemp);
74              X[i][j] = Ftemp;
75          }
76      for (i = 0; i < S; i++)
77          for (j = 0; j < S; j++)
78          {
79              if (i == j)
80              {
81                  CDT[i][j]  = 0.0;
82                  TEMP[i][j] = 0.0;
83              }
84              else
85              {
86                  CDT[i][j]  = 1000.0;
87                  TEMP[i][j] = 1000.0;
88              }
89          }
90      // ================================
91      // Initialize Pair-state Table (PT)
92      // ================================
93      TD = 0;  // Trellis-Depth = 0
94      C1 = 0;
95      for (n = 0; n < S; n++)
96      {    // 10
97          fscanf(Config, "%d", &C2);
98          C2 = C2 - 1;
99          C3 = 0;
100         SN = 0;
101         j  = 0;  // loop-control variable for column
```

```
102         for (m = 0; m < TOTAL2; m++)
103         {    // 20
104             for (l = 0; l < TOTAL3; l++)
105             {    // 30
106                 MIN = 1000.0;
107                 for (p = 0; p < TOTAL4; p++)
108                 {    // 35
109                     for (q = 0; q < TOTAL4; q++)
110                     {    // 38
111                         DIST = 0.0;
112                         for (r = 0; r < D; r++)
113                             DIST = DIST + pow(X[Y+p][r] - X[Z+q][r],2.0);
114                         if (DIST < MIN) MIN = DIST;
115                     }    // 38
116                 }    // 35
117                 Z = Z + TOTAL4;
118                 if (Z >= TOTAL1)
119                 {
120                     Z = 0;
121                     Y = Y + TOTAL4;
122                 }
123                 PT[C1 * S + SN][C3 * TOTAL3 + l][0] = MIN;
124                 PT[C1 * S + SN][C3 * TOTAL3 + l][1] = C2*S+(j+1);
125                 j = j + 1;
126                 if (j == S) j = 0;
127             }    // 30
128             SN = SN + 1;
129             if (SN == S)
130             {
131                 SN = 0;
132                 C2 = C2 + 1;
133                 C3 = C3 + 1;
134             }
135         }    // 20
136         C1 = C1 + 1;
137     }    // 10

138     // ======================================
139     // Compute TEMP for the next trellis depth
140     // ======================================

141     FIRST = TRUE;
142     DMIN  = 1000.0;
143     do
144     {
145         for (i = 0; i < S; i++)
146         {    // 70
147             for (j = 0; j < S; j++)
148             {    // 80
```

```
149                 if ((CDT[i][j] != 0.0 || FIRST == TRUE) &&
150                      (CDT[i][j] <  1000.0 ))
151                 {   // If 1
152                     for (l = 0; l < TOTAL5; l++)
153                     {    // 90
154                         ROWPT = i*S + j;
155                         TEMP0 = CDT[i][j] + PT[ROWPT][l][0];
156                         ROW   = (int) (PT[ROWPT][l][1] - 1.0)/S;
157                         COL   = (((int) PT[ROWPT][l][1]) % S)-1;
158                         if (COL == -1) COL = S-1;
159                         if ((TEMP0 < TEMP[ROW][COL]) || (TEMP[ROW][COL] == 0.0))
160                             TEMP[ROW][COL] = TEMP0;
161                         if ((ROW == COL) && (TEMP[ROW][ROW] > 0.0) && (TEMP[ROW][ROW]
162                             DMIN = TEMP[ROW][COL];
163                     }    // 90
164                 }    // If 1
165             }    // 80
166         }    // 70
167         // ====================
168         // Update CDT from TEMP
169         // ====================

170         STILL = FALSE;
171         FIRST = FALSE;

172         for (i = 0; i < S; i++)
173         {    // 120

174             for (j = 0; j < S; j++)
175             {    // 130

176                 CDT[i][j] = TEMP[i][j];

177                 if (i == j)    TEMP[i][j] = 0.0;
178                 else TEMP[i][j] = 1000.0;

179                 if (CDT[i][j] < DMIN)    STILL = TRUE;

180             }    // 130
181         }    // 120

182         TD = TD + 1;
183     } while ((STILL == TRUE) && (TD < 100));

184     // ======================
185     // Presentation of Results
186     // ======================
187     highvideo();
188     cprintf("\r\n\nResults:\n\r");
```

```
189       if (TD >= 100)
190       {
191           cout << "The code appears to be catastrophic after a ";
192           cout << "trellis depth of 100\n";
193       }
194       else
195       {
196           cout << "Minimum Squared Euclidean Distance, Dmin: " << DMIN << "\n";
197           cout << "Trellis Depth = " << TD << "\n";
198       }
199       fcloseall();

200       // ===========
201       // Free memory
202       // ===========
203       Free_Memory();
204   }
```

APPENDIX C

TRANSFER FUNCTION DERIVATION

This appendix the Chan—Norton algebraic algorithm for generating the transfer function of a trellis code is presented. The algebraic method is presented as an alternative to the better known transfer function derivation by means of graphical means, i.e. by graph-reduction of Mason's rule.

The Chan—Norton algorithm for transfer function derivation is presented [79]. In order to find the transfer function representing all paths (except the loop at 0, i.e. the modified transfer function is considered) starting from state 0 and ending at state $0'$. Firstly the trellis diagram of the encoder is drawn. Then each branch of the trellis is labeled. The labels can be in general expressed as a finite sum $\sum_i a_i x^{n_i}$, which is the weight profile (defined in Chapter 6). The summation accounts for parallel transitions, x is a dummy variable whose exponent n_i is the number of bits in error for that transition and a_i is a scalar which could be a function of the SNR or Euclidean distance.

The transfer function $T(x)$ is a quotient of two polynomials, $T(x) = f(x)/g(x)$, say. From the discussion in [79], the successive coefficients of the transfer function form a linear recurring sequence, and the denominator of minimal degree can be used using the Extended Euclidean Algorithm (XEA). The next step is to generate the *adjacency matrix* $A = A(x)$ of the encoder from the trellis diagram. For an N_s state encoder, A is matrix of dimension $(N_s+1)\times(N_s+1)$, and A_{ij} represents the paths with starting state i, ranging from states $0, 1, 2, \ldots, N_s, 0'$, and ending in state j also ranging from states $0, 1, 2, \ldots, N_s, 0'$.

Importantly, the entries of the matrix will be the labels on the trellis diagram. If the two states are not joined on the trellis diagram, the entry will be zero in the adjacency matrix. The column $0'$ represents the *first output column*. The $A_{00'}$ entry represents the path starting from state 0 and ending at state $0'$. (Note that the first column and last row in A are zero.)

If p is a polynomial, let δp denote the degree of p. Then determine the upper bound for $\delta\ det\ (I-A)$,

where I is the identity matrix. An upper bound is found by taking the sum of the maximum of the degrees of each of the columns in $I - A$, and adding one. Denote this upper bound by M. Next all the paths starting from state 0 and ending at state 0′ with degree of x less than $2M$, i.e., $T(x)\ mod\ x^{2M}$ need to be summed. One method is by repeated multiplication of A with the output column vector. The sum of the first entries in the resulting column vectors is the answer we are looking for. Let $e(x) = T(x)\ mod\ x^{2M}$. Apply the XEA to x^{2M} and $e(x)$ until $\delta\ r_{n+1} < M$; letting $r_{-1} = x^{2M}$, $r_0 = e(x)$, $r_{i+1} = r_{i-1} - q_{i+1} \cdot r_i$ is obtained, where $\delta\ r_{i+1} < \delta r_i$, i.e., q_{i+1} is the quotient obtained from dividing r_{i-1} by r_i with remainder r_{i+1} for $0 \leq i \leq n$. The transfer function is then given by $T(x) = r_{n+1}/t_{n+1}$.

It is important to note that the general complexity of this algebraic algorithm, as is the case with the graph-reduction method, increases exponentially with the number of states of the encoder. In the following detail concerning the derivation of the transfer function of the rate-3/4 8-state trellis code given in Chapter 6 is presented.

The trellis code structure for the fully-connected rate-3/4 trellis code designed in Chapter 6, is repeated here for convenience. The trellis diagram is illustrated in Figure C.1.

b_6	b_5	b_4	$(b_3\ b_2\ b_1)$							
0	0	0	0	15	3	12	5	10	9	6
0	0	1	1	14	2	13	4	11	8	7
0	1	0	15	0	12	3	10	5	6	9
0	1	1	14	1	13	2	11	4	7	8
1	0	0	3	12	0	15	9	6	5	10
1	0	1	2	13	1	14	8	7	4	11
1	1	0	12	3	15	0	6	9	10	5
1	1	1	13	2	14	1	7	8	11	4

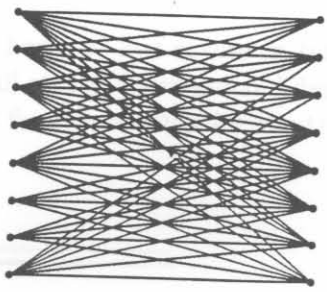

Figure C.1: Fully-connected $R = 3/4$ 8-state trellis code.

The adjacency matrix, of dimensions (9×9) for the 8-state trellis code in Figure C.1 is

$$A(x) = \begin{bmatrix} S_0 & S_1 & S_2 & S_3 & S_4 & S_5 & S_6 & S_7 & S_{0'} \\ 0 & D_{15}x & D_3x & D_{12}x^2 & D_5x & D_{10}x^2 & D_9x^2 & D_6x^3 & D_0 \\ 0 & D_{14}x & D_2x & D_{13}x^2 & D_4x & D_{11}x^2 & D_8x^2 & D_7x^3 & D_1 \\ 0 & D_0x & D_{12}x & D_3x^2 & D_{10}x & D_5x^2 & D_6x^2 & D_9x^3 & D_{15} \\ 0 & D_1x & D_{13}x & D_2x^2 & D_{11}x & D_4x^2 & D_7x^2 & D_8x^3 & D_{14} \\ 0 & D_{12}x & D_0x & D_{15}x^2 & D_9x & D_6x^2 & D_5x^2 & D_{10}x^3 & D_3 \\ 0 & D_{13}x & D_1x & D_{14}x^2 & D_8x & D_7x^2 & D_4x^2 & D_{11}x^3 & D_2 \\ 0 & D_3x & D_{15}x & D_0x^2 & D_6x & D_9x^2 & D_{10}x^2 & D_5x^3 & D_{12} \\ 0 & D_2x & D_{14}x & D_1x^2 & D_7x & D_8x^2 & D_{11}x^2 & D_4x^3 & D_{13} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{(C.1)}$$

where $D_0 \ldots D_{15}$ are nonzero scalar labels of the trellis diagram. An upper bound, M for $\delta det(I-A)$ is easily found as the sum of the maximum of the degrees of each of the columns in $(I - A)$, and adding one. The bound is $M = (1+1+2+1+2+2+2+3+0)+1 = 14$. Therefore, we need to

find all the paths from S_0 to $S_{0'}$, that have degree less than $2M = 28$. This is found by repeated multiplication of matrix A with an output column vector, for instance

$$C_1 = A \cdot \begin{bmatrix} D_0 \\ D_1 \\ D_{15} \\ D_{14} \\ D_3 \\ D_2 \\ D_{12} \\ D_{13} \\ 0 \end{bmatrix} \tag{C.2}$$

We continue this multiplication until the order of the polynomial in the first row is of order $\delta < 2M$. Since, the computational complexity of this specific code design is quite high it is necessary to consider the use of computer utilities to assist in the computations.

After some mathematics (computer mathematics by means of "Mathematica" [97]) the transfer function of this trellis code is given by (see (5.14))

$$T(D) = \frac{D^{20} + 24D^{36} - 16D^{18} + 48D^{8} + 24D^{24}}{2 + 18D^{4} - 8D^{8} + D^{12}} \tag{C.3}$$

By substituting $D = exp(-E_b/N_o)$ in equation (C.3), the transfer function is readily obtained.

In conclusion, it is stated that the foregoing algebraic algorithm, however less severe in terms of required effort compared to the graph-reduction methods, is not at all straight forward and demands still a lot of work by the trellis code designer in order to obtain the final expression for the transfer function.

APPENDIX D

SET PARTITIONING FOR Q^{2}PSK/MTCM CODE DESIGN

In this appendix the subsets obtained from the first and second partition levels, for the multiplier integer $n = 11$ as required in the MTCM analysis of Chapter 6, section 6.1.1, are presented. Only the subsets obtained for a multiplicity, $k = 2$ are presented. Recall, that the subsets for the codes of higher multiplicity is easily derived from the subsets with code multiplicity of 2.

D.1 CODE CARDINALITY OF 16

$$A_0 \otimes B_0 = \begin{bmatrix} 0 & 0 & 8 & 8 \\ 1 & 11 & 9 & 3 \\ 2 & 6 & 10 & 14 \\ 3 & 1 & 11 & 9 \\ 4 & 12 & 12 & 4 \\ 5 & 7 & 13 & 15 \\ 6 & 2 & 14 & 10 \\ 7 & 13 & 15 & 5 \end{bmatrix} \quad A_0 \otimes B_1 = \begin{bmatrix} 0 & 1 & 8 & 9 \\ 1 & 12 & 9 & 4 \\ 2 & 7 & 10 & 15 \\ 3 & 2 & 11 & 10 \\ 4 & 13 & 12 & 5 \\ 5 & 8 & 13 & 0 \\ 6 & 3 & 14 & 11 \\ 7 & 14 & 15 & 6 \end{bmatrix} \tag{D.1}$$

$$A_0 \otimes B_2 = \begin{bmatrix} 0 & 2 & 8 & 10 \\ 1 & 13 & 9 & 5 \\ 2 & 8 & 10 & 0 \\ 3 & 3 & 11 & 11 \\ 4 & 14 & 12 & 6 \\ 5 & 9 & 13 & 1 \\ 6 & 4 & 14 & 12 \\ 7 & 15 & 15 & 7 \end{bmatrix} \quad A_0 \otimes B_3 = \begin{bmatrix} 0 & 3 & 8 & 11 \\ 1 & 14 & 9 & 6 \\ 2 & 9 & 10 & 1 \\ 3 & 4 & 11 & 12 \\ 4 & 15 & 12 & 7 \\ 5 & 10 & 13 & 2 \\ 6 & 7 & 14 & 13 \\ 7 & 0 & 15 & 8 \end{bmatrix} \tag{D.2}$$

$$A_0 \otimes B_4 = \begin{bmatrix} 0 & 4 & 8 & 12 \\ 1 & 15 & 9 & 7 \\ 2 & 10 & 10 & 2 \\ 3 & 5 & 11 & 13 \\ 4 & 0 & 12 & 8 \\ 5 & 11 & 13 & 3 \\ 6 & 6 & 14 & 14 \\ 7 & 1 & 15 & 9 \end{bmatrix} \quad A_0 \otimes B_5 = \begin{bmatrix} 0 & 5 & 8 & 13 \\ 1 & 0 & 9 & 8 \\ 2 & 11 & 10 & 3 \\ 3 & 6 & 11 & 14 \\ 4 & 1 & 12 & 9 \\ 5 & 12 & 13 & 4 \\ 6 & 7 & 14 & 15 \\ 7 & 2 & 15 & 10 \end{bmatrix} \tag{D.3}$$

$$A_0 \otimes B_6 = \begin{bmatrix} 0 & 6 & 8 & 14 \\ 1 & 1 & 9 & 9 \\ 2 & 12 & 10 & 4 \\ 3 & 7 & 11 & 15 \\ 4 & 2 & 12 & 10 \\ 5 & 13 & 13 & 5 \\ 6 & 8 & 14 & 0 \\ 7 & 3 & 15 & 11 \end{bmatrix} \quad A_0 \otimes B_7 = \begin{bmatrix} 0 & 7 & 8 & 15 \\ 1 & 2 & 9 & 10 \\ 2 & 13 & 10 & 5 \\ 3 & 8 & 11 & 0 \\ 4 & 3 & 12 & 11 \\ 5 & 14 & 13 & 6 \\ 6 & 9 & 14 & 1 \\ 7 & 4 & 15 & 12 \end{bmatrix} \tag{D.4}$$

$$A_0 \otimes B_8 = \begin{bmatrix} 0 & 8 & 8 & 0 \\ 1 & 3 & 9 & 11 \\ 2 & 14 & 10 & 6 \\ 3 & 9 & 11 & 1 \\ 4 & 4 & 12 & 12 \\ 5 & 15 & 13 & 7 \\ 6 & 10 & 14 & 2 \\ 7 & 5 & 15 & 13 \end{bmatrix} \quad A_0 \otimes B_9 = \begin{bmatrix} 0 & 9 & 8 & 1 \\ 1 & 4 & 9 & 12 \\ 2 & 15 & 10 & 7 \\ 3 & 10 & 11 & 2 \\ 4 & 5 & 12 & 13 \\ 5 & 0 & 13 & 8 \\ 6 & 11 & 14 & 3 \\ 7 & 6 & 15 & 14 \end{bmatrix} \tag{D.5}$$

$$A_0 \otimes B_{10} = \begin{bmatrix} 0 & 10 & 8 & 2 \\ 1 & 5 & 9 & 13 \\ 2 & 0 & 10 & 8 \\ 3 & 11 & 11 & 3 \\ 4 & 6 & 12 & 14 \\ 5 & 1 & 13 & 9 \\ 6 & 12 & 14 & 4 \\ 7 & 7 & 15 & 15 \end{bmatrix} \quad A_0 \otimes B_{11} = \begin{bmatrix} 0 & 11 & 8 & 3 \\ 1 & 6 & 9 & 14 \\ 2 & 1 & 10 & 9 \\ 3 & 12 & 11 & 4 \\ 4 & 7 & 12 & 15 \\ 5 & 2 & 13 & 10 \\ 6 & 13 & 14 & 5 \\ 7 & 8 & 15 & 0 \end{bmatrix} \tag{D.6}$$

$$A_0 \otimes B_{12} = \begin{bmatrix} 0 & 12 & 8 & 4 \\ 1 & 7 & 9 & 15 \\ 2 & 2 & 10 & 10 \\ 3 & 13 & 11 & 5 \\ 4 & 8 & 12 & 0 \\ 5 & 3 & 13 & 11 \\ 6 & 14 & 14 & 6 \\ 7 & 9 & 15 & 1 \end{bmatrix} \quad A_0 \otimes B_{13} = \begin{bmatrix} 0 & 13 & 8 & 5 \\ 1 & 8 & 9 & 0 \\ 2 & 3 & 10 & 11 \\ 3 & 14 & 11 & 6 \\ 4 & 9 & 12 & 1 \\ 5 & 4 & 13 & 12 \\ 6 & 15 & 14 & 7 \\ 7 & 10 & 15 & 2 \end{bmatrix} \tag{D.7}$$

$$A_0 \otimes B_{14} = \begin{bmatrix} 0 & 14 & 8 & 6 \\ 1 & 9 & 9 & 1 \\ 2 & 4 & 10 & 12 \\ 3 & 15 & 11 & 7 \\ 4 & 10 & 12 & 2 \\ 5 & 5 & 13 & 13 \\ 6 & 16 & 14 & 8 \\ 7 & 11 & 15 & 3 \end{bmatrix} \quad A_0 \otimes B_{15} = \begin{bmatrix} 0 & 15 & 8 & 7 \\ 1 & 10 & 9 & 2 \\ 2 & 5 & 10 & 13 \\ 3 & 0 & 11 & 8 \\ 4 & 11 & 12 & 3 \\ 5 & 6 & 13 & 14 \\ 6 & 1 & 14 & 9 \\ 7 & 12 & 15 & 4 \end{bmatrix} \tag{D.8}$$

D.2 CODE CARDINALITY OF 8

$$C_0 \otimes D_{0a} = \begin{bmatrix} 0 & 0 & 4 & 8 \\ 1 & 6 & 5 & 14 \\ 2 & 12 & 6 & 4 \\ 3 & 2 & 7 & 10 \end{bmatrix} \quad C_0 \otimes D_{0b} = \begin{bmatrix} 0 & 11 & 4 & 3 \\ 1 & 1 & 5 & 9 \\ 2 & 7 & 6 & 15 \\ 3 & 13 & 7 & 5 \end{bmatrix} \tag{D.9}$$

$$C_0 \otimes D_{1a} = \begin{bmatrix} 0 & 1 & 4 & 9 \\ 1 & 7 & 5 & 15 \\ 2 & 12 & 6 & 5 \\ 3 & 3 & 7 & 11 \end{bmatrix} \quad C_0 \otimes D_{1b} = \begin{bmatrix} 0 & 12 & 4 & 4 \\ 1 & 1 & 5 & 10 \\ 2 & 8 & 6 & 0 \\ 3 & 14 & 7 & 6 \end{bmatrix} \tag{D.10}$$

$$C_0 \otimes D_{2a} = \begin{bmatrix} 0 & 2 & 4 & 10 \\ 1 & 8 & 5 & 0 \\ 2 & 14 & 6 & 6 \\ 3 & 4 & 7 & 12 \end{bmatrix} \quad C_0 \otimes D_{2b} = \begin{bmatrix} 0 & 13 & 4 & 5 \\ 1 & 3 & 5 & 11 \\ 2 & 9 & 6 & 1 \\ 3 & 15 & 7 & 7 \end{bmatrix} \tag{D.11}$$

$$C_0 \otimes D_{3a} = \begin{bmatrix} 0 & 3 & 4 & 11 \\ 1 & 9 & 5 & 1 \\ 2 & 15 & 6 & 7 \\ 3 & 0 & 7 & 8 \end{bmatrix} \quad C_0 \otimes D_{3b} = \begin{bmatrix} 0 & 14 & 4 & 6 \\ 1 & 4 & 5 & 9 \\ 2 & 10 & 6 & 2 \\ 3 & 0 & 7 & 8 \end{bmatrix} \tag{D.12}$$

$$C_0 \otimes D_{4a} = \begin{bmatrix} 0 & 4 & 4 & 12 \\ 1 & 10 & 5 & 2 \\ 2 & 0 & 6 & 8 \\ 3 & 6 & 7 & 14 \end{bmatrix} \quad C_0 \otimes D_{4b} = \begin{bmatrix} 0 & 15 & 4 & 7 \\ 1 & 5 & 5 & 13 \\ 2 & 11 & 6 & 3 \\ 3 & 1 & 7 & 9 \end{bmatrix} \tag{D.13}$$

$$C_0 \otimes D_{5a} = \begin{bmatrix} 0 & 5 & 4 & 13 \\ 1 & 11 & 5 & 3 \\ 2 & 1 & 6 & 9 \\ 3 & 7 & 7 & 15 \end{bmatrix} \quad C_0 \otimes D_{5b} = \begin{bmatrix} 0 & 0 & 4 & 8 \\ 1 & 6 & 5 & 14 \\ 2 & 12 & 6 & 4 \\ 3 & 2 & 7 & 10 \end{bmatrix} \tag{D.14}$$

$$C_0 \otimes D_{6a} = \begin{bmatrix} 0 & 6 & 4 & 14 \\ 1 & 12 & 5 & 4 \\ 2 & 2 & 6 & 10 \\ 3 & 8 & 7 & 0 \end{bmatrix} \quad C_0 \otimes D_{6b} = \begin{bmatrix} 0 & 1 & 4 & 9 \\ 1 & 7 & 5 & 15 \\ 2 & 13 & 6 & 5 \\ 3 & 3 & 7 & 11 \end{bmatrix} \tag{D.15}$$

$$C_0 \otimes D_{7a} = \begin{bmatrix} 0 & 7 & 4 & 15 \\ 1 & 13 & 5 & 5 \\ 2 & 3 & 6 & 11 \\ 3 & 9 & 7 & 1 \end{bmatrix} \quad C_0 \otimes D_{7b} = \begin{bmatrix} 0 & 2 & 4 & 10 \\ 1 & 8 & 5 & 0 \\ 2 & 14 & 6 & 6 \\ 3 & 4 & 7 & 12 \end{bmatrix} \tag{D.16}$$

$$C_0 \otimes D_{8a} = \begin{bmatrix} 0 & 8 & 4 & 0 \\ 1 & 14 & 5 & 6 \\ 2 & 4 & 6 & 12 \\ 3 & 10 & 7 & 2 \end{bmatrix} \quad C_0 \otimes D_{8b} = \begin{bmatrix} 0 & 3 & 4 & 11 \\ 1 & 9 & 5 & 1 \\ 2 & 15 & 6 & 7 \\ 3 & 5 & 7 & 13 \end{bmatrix} \tag{D.17}$$

$$C_0 \otimes D_{9a} = \begin{bmatrix} 0 & 9 & 4 & 1 \\ 1 & 15 & 5 & 7 \\ 2 & 5 & 6 & 13 \\ 3 & 11 & 7 & 3 \end{bmatrix} \quad C_0 \otimes D_{9b} = \begin{bmatrix} 0 & 4 & 4 & 12 \\ 1 & 10 & 5 & 2 \\ 2 & 0 & 6 & 8 \\ 3 & 6 & 7 & 14 \end{bmatrix} \tag{D.18}$$

$$C_0 \otimes D_{10a} = \begin{bmatrix} 0 & 10 & 4 & 2 \\ 1 & 0 & 5 & 8 \\ 2 & 6 & 6 & 14 \\ 3 & 12 & 7 & 4 \end{bmatrix} \quad C_0 \otimes D_{10b} = \begin{bmatrix} 0 & 5 & 4 & 13 \\ 1 & 11 & 5 & 3 \\ 2 & 1 & 6 & 9 \\ 3 & 7 & 7 & 15 \end{bmatrix} \tag{D.19}$$

$$C_0 \otimes D_{11a} = \begin{bmatrix} 0 & 11 & 4 & 3 \\ 1 & 1 & 5 & 9 \\ 2 & 7 & 6 & 15 \\ 3 & 13 & 7 & 5 \end{bmatrix} \quad C_0 \otimes D_{11b} = \begin{bmatrix} 0 & 6 & 4 & 14 \\ 1 & 12 & 5 & 4 \\ 2 & 2 & 6 & 10 \\ 3 & 8 & 7 & 0 \end{bmatrix} \tag{D.20}$$

$$C_0 \otimes D_{12a} = \begin{bmatrix} 0 & 12 & 4 & 4 \\ 1 & 2 & 5 & 10 \\ 2 & 8 & 6 & 0 \\ 3 & 14 & 7 & 6 \end{bmatrix} \quad C_0 \otimes D_{12b} = \begin{bmatrix} 0 & 7 & 4 & 15 \\ 1 & 13 & 5 & 5 \\ 2 & 3 & 6 & 11 \\ 3 & 9 & 7 & 1 \end{bmatrix} \tag{D.21}$$

$$C_0 \otimes D_{13a} = \begin{bmatrix} 0 & 13 & 4 & 5 \\ 1 & 3 & 5 & 11 \\ 2 & 9 & 6 & 1 \\ 3 & 15 & 7 & 7 \end{bmatrix} \quad C_0 \otimes D_{13b} = \begin{bmatrix} 0 & 8 & 4 & 0 \\ 1 & 14 & 5 & 6 \\ 2 & 4 & 6 & 12 \\ 3 & 10 & 7 & 2 \end{bmatrix} \tag{D.22}$$

$$C_0 \otimes D_{14a} = \begin{bmatrix} 0 & 14 & 4 & 6 \\ 1 & 4 & 5 & 12 \\ 2 & 10 & 6 & 2 \\ 3 & 0 & 7 & 8 \end{bmatrix} \qquad C_0 \otimes D_{14b} = \begin{bmatrix} 0 & 9 & 4 & 1 \\ 1 & 15 & 5 & 7 \\ 2 & 5 & 6 & 13 \\ 3 & 11 & 7 & 3 \end{bmatrix} \tag{D.23}$$

$$C_0 \otimes D_{15a} = \begin{bmatrix} 0 & 15 & 4 & 7 \\ 1 & 5 & 5 & 13 \\ 2 & 11 & 6 & 3 \\ 3 & 1 & 7 & 9 \end{bmatrix} \qquad C_0 \otimes D_{15b} = \begin{bmatrix} 0 & 10 & 4 & 2 \\ 1 & 0 & 5 & 8 \\ 2 & 6 & 6 & 14 \\ 3 & 12 & 7 & 4 \end{bmatrix} \tag{D.24}$$