

# **An Investigation into the Performance Capabilities of Multi-h CPFSK Digital Modulation**

by

**Garth Airey**

BSc(Eng) *Cape Town*

Submitted to the Department of Electrical Engineering in partial fulfilment of the requirements for the degree of

MSc(Eng)

at the

UNIVERSITY OF CAPE TOWN

February 1997

© University of Cape Town 1997

The University of Cape Town has been given the right to reproduce this thesis in whole or in part. Copyright is held by the author.

The copyright of this thesis vests in the author. No quotation from it or information derived from it is to be published without full acknowledgement of the source. The thesis is to be used for private study or non-commercial research purposes only.

Published by the University of Cape Town (UCT) in terms of the non-exclusive license granted to UCT by the author.

**Declaration by the Author:**

I hereby declare that the work contained in this document is my own original work and that every attempt has been made to indicate and reference any exceptions or external sources. I also state that, prior to the submission of this dissertation, the work presented here has never been published in any form, nor has it been submitted to any educational institution for the purposes of obtaining any degree or diploma or for any other reason.

Signed:

Signed by candidate

Garth Airey (Author)

# Acknowledgements

I wish to thank the following people for their valuable contributions to this project:

- Dr R.M. Braun for supervising, and for his many good ideas and suggestions.
- Transnet Bursaries for their financial support.
- Eugène Nel for the use of his house, his computer and his printer. Also for all that encouragement to "get on with it" - I needed that!
- My family, for always being there for me when I needed them (and for putting up with all my nonsense). What would I do without them?

And then, above all, I would like to express my sincere gratitude to God for his never-failing love and support throughout seventeen years of education. It is only through Him that all of this has been made possible.

# Synopsis

This thesis sets out the procedures and results of an investigation into the performance capabilities of multi-h CPFSK digital modulation. Multi-h CPFSK is a coded modulation scheme. It exhibits high bandwidth efficiency and good envelope properties and is therefore well suited to use in satellite communications channels. However, the power efficiency of multi-h CPFSK signals is largely dependent on the particular multi-h code used, as well as the optimality of the decoder.

Optimal decoding of multi-h CPFSK signals is achieved by means of the Viterbi Algorithm, applied to the multi-h phase trellis diagram. The Viterbi path metric is given by the squared euclidean distance between the particular path and the received signal. The probability of error for optimal decoding is approximated by

$$P_e \approx Q\left(\sqrt{\frac{E_b \cdot d_{min}^2}{N_0}}\right)$$

where  $d_{min}^2$  is known as the normalised minimum squared euclidean distance of the code. The larger the value of  $d_{min}^2$ , the better the power efficiency. Power efficiency is usually referenced to MSK which has a  $d_{min}^2$  of 2. A code with a  $d_{min}^2$  of greater than 2 is said to achieve improved power efficiency, or coding gain, over MSK. Sequential decoders may also be used to decode multi-h signals, but these are not seen to be a viable option as they offer severely degraded performance.

For optimal decoding, Anderson, Aulin and Sundberg [2] have found theoretical bounds on performance for  $h$ -sets of sizes 1, 2, 3 and 4, but have not documented practical codes which achieve these maximum coding gains. Cuthbert [4] has conducted a PBIL search for high performance codes of sizes 1 to 8, but subsequent evaluation of his results has revealed that his performance measurements are incorrect and his results are therefore of little value. For this reason an attempt has been made to repeat/improve the search process performed

by Cuthbert and to carefully verify all results by means of a computer BER simulation.

A modified PBIL search algorithm has been used to conduct the search for high performance  $h$ -sets. The search software has been implemented in 'C' for  $h$ -set sizes of 2 to 8 with  $h$ -set denominators ranging between 8 and 256. For the purposes of the search, the minimum squared euclidean distance for each trial code has been approximated by Viterbi decoding a random modulated bit stream and noting the differences in path metrics between that bitstream and all paths merging with it in the phase trellis.

The results of the search indicate that coding gains of 4.5 dB (over MSK) and greater are achievable using multi- $h$ . The highest coding gain found by the search was 4.87 dB for an  $h$ -set of size 7 with a denominator of 256. For the  $h$ -set sizes of 2, 3 and 4, the results fall within 0.5% of the theoretical maxima found by [2], thus verifying the near-optimality of the all of the results. A general trend observed in the results is that the coding gains increase with  $h$ -set size and denominator size. However, the law of diminishing returns applies and it is therefore unlikely that multi- $h$  codes with performance characteristics much better than those found in this thesis exist. As a further verification of the results, two of the codes have been implemented by computer simulation, and BER curves have been found for Viterbi decoding of the simulated signals. It is evident from plots of these curves that the simulated BERs correspond closely to the theoretical curves at high SNR.

Additional issues relating to Viterbi decoding of multi- $h$  CPFSK have also been addressed. These include correlator receiver structure, calculation of branch metrics, choice of truncation depth and memory management issues. However, these topics have not been studied in great detail.

Based on the work completed in this dissertation, recommendations for future research are: to consider various synchronisation methods for extracting carrier timing, bit timing and superbaud timing from multi- $h$  CPFSK signals; to investigate the practical implementation of the Viterbi Decoder for multi- $h$  CPFSK; to build a physical simulation test-bed in order to determine the real-world performance of multi- $h$  CPFSK, and to investigate the effects of real-world channels on multi- $h$  CPFSK systems.

# Table of Contents

<b>Acknowledgements</b>	<b>i</b>
<b>Synopsis</b>	<b>ii</b>
<b>List of Figures</b>	<b>vii</b>
<b>List of Tables</b>	<b>ix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background.....	1
1.2 Objectives of Thesis.....	3
1.3 Organisation of Thesis.....	4
<b>2 The Theory of Multi-h CPFSK</b>	<b>5</b>
2.1 Multi-h CPFSK and the Digital Communications System.....	5
2.2 Multi-h CPFSK in relation to Other Digital Modulation Schemes.....	6
2.3 Mathematical Representation of Multi-h CPFSK Signals.....	8
2.4 Graphical Representation of Multi-h CPFSK Signals .....	10
2.5 Satellite Channels and Multi-h CPFSK.....	11
2.5.1 Constraints placed on the choice of modulation scheme.....	12
2.5.2 Suitability of multi-h CPFSK to satellite channels.....	13

<b>3</b>	<b>Optimal Decoding of Multi-h Signals in the presence of AWGN</b>	<b>17</b>
3.1	The Viterbi Algorithm .....	17
3.2	The Maximum Likelihood Detector.....	18
3.2.1	Derivation of the maximum likelihood detector .....	18
3.2.2	Implementation as a correlator receiver.....	23
3.3	The Complete Viterbi Decoder for Multi-h CPFSK.....	24
3.3.1	The Viterbi decoding process.....	25
3.3.2	Calculation of branch metrics for multi-h CPFSK.....	28
3.4	Total Probability of Error for the Viterbi Decoder.....	28
<b>4</b>	<b>A Modified PBIL Search for High Performance H-sets</b>	<b>31</b>
4.1	Performance Criteria and Desired Properties of $h$ -sets.....	31
4.2	Calculation of $d_{min}^2$ .....	32
4.3	The PBIL Algorithm.....	33
4.4	PBIL and Multi-h CPFSK.....	35
4.5	Modifications to the PBIL Algorithm .....	36
4.6	Modified PBIL Search Results.....	39
<b>5</b>	<b>Measurement of BER Curves for Selected Multi-h Codes</b>	<b>43</b>
5.1	Software Implementation of the Viterbi Algorithm.....	43
5.1.1	Choice of truncation depth.....	44
5.1.2	Memory structure for path storage .....	45
5.2	Simulation Results.....	47

<b>6</b>	<b>Sequential Decoding of Multi-h Signals</b>	<b>49</b>
6.1	General Description of Sequential Algorithms.....	49
6.2	Error Performance of Sequential Decoders.....	50
<b>7</b>	<b>Summary and Recommendations</b>	<b>52</b>
<b>A</b>	<b>Gaussian Probability Functions</b>	<b>54</b>
<b>B</b>	<b>Derivation of Probability of Error for the Correlator Receiver</b>	<b>55</b>
<b>C</b>	<b>PN Sequence Generator for Search and Simulation Software</b>	<b>59</b>
<b>D</b>	<b>'C' Code Listing for Modified PBIL Search Software</b>	<b>60</b>
<b>E</b>	<b>'C' Code Listing for Viterbi Decoding Software</b>	<b>71</b>

# List of Figures

1-1	The position of MSK relative to the channel capacity boundary.....	2
1-2	Thesis organisation and chapter layout.....	4
2-1	Block diagram of a digital communications system.....	5
2-2	Simplified digital modulation tree.....	7
2-3	Phase trellis diagram for $\{h_1, h_2\} = \{1/4, 2/4\}$ .....	11
2-4	Baseband-equivalent power spectra for MSK and QPSK.....	14
2-5	Probability of error curves for multi-h CPFSK ( $d_{min}^2 = 6$ ) and MSK.....	15
3-1	Communications system for derivation of maximum likelihood detector.....	19
3-2	Example of signal space representation for $s_1(t)$ and $s_2(t)$ .....	20
3-3	Example of probability distribution function for $X(k)$ .....	21
3-4	Example of rotated signal space for probability of error calculation.....	22
3-5	Phase trellis diagram for $\{h_1, h_2\} = \{1/4, 2/4\}$ , plotted betw. $-3\pi/2$ and $\pi$ .....	25
3-6	Step one of the Viterbi decoding process.....	25
3-7	Step two of the Viterbi decoding process.....	26
3-8	Step three of the Viterbi decoding process.....	26
3-9	Step-by-step Viterbi decoding.....	27

3-10 Correlator bank structure for the calculation of branch metrics.....	29
4-1 PBIL flowchart.....	34
4-2 Modified PBIL flowchart.....	38
4-3 3D surface plot of modified PBIL search results.....	39
4-4 Modified PBIL search results w.r.t. the channel capacity boundary.....	41
5-1 Build up of distance between paths over the truncation depth (worst case).....	44
5-2 Memory structure for simulation path storage.....	45
5-3 More efficient memory structure for path storage.....	46
5-4 BER curve for a set size of 4, denominator 32.....	47
5-5 BER curve for a set size of 6, denominator 64.....	48
6-1 Paths dominating the probability of error for sequential decoding.....	50
B-1 Correlator receiver structure.....	55
C-1 PN sequence generator for search and simulation software.....	59

# List of Tables

4-1	Representation of $h$ -set numerator values for a denominator of 16.....	37
4-2	Modified PBIL search results.....	40
4-3	Theoretical performance limits for multi-h CPFSK obtained by [2].....	42
4-4	Comparison between modified PBIL results and theoretical maxima.....	42

# Chapter 1

## Introduction

### 1.1 Background

It is difficult to imagine what life in today's world would be like without easy access to reliable, economic and efficient means of communication. Recent years have seen a period of explosive growth for the communications industry, driven by the constant demand for smaller, cheaper, faster and more portable devices. Amazing new technologies, such as digital cellular telephony and satellite television, have been introduced. In fact, the need for highly efficient communication systems has begun to work its way into almost every facet of modern society, from personal entertainment and long distance telephone conversation, to business, banking and finance, to live news and sports coverage, aircraft navigational systems, the Internet - the list continues.

The predominant focus in providing for these needs has been moving steadily away from old analogue technologies and onto cheaper, more reliable digital solutions, often involving satellites where wide-area coverage is required. As an example, there is a huge push at the present time for a global network of low earth orbiting (LEO) satellites to cater for the needs of the digital cellular industry and the Internet. In terms of a digital modulation scheme required for use in such a project, the most suitable technique available from current technology is probably that of Minimum Shift Keying (MSK). MSK is generally considered to be an efficient modulation scheme, attractive for use in satellite applications [3].

As an indication of the *absolute* efficiency of MSK, however, it is instructive to review the theory of channel capacity published by Claude Shannon in 1948 [6]. In this work, Shannon presents a mathematical proof giving the maximum bit rate,

$R$ , which can be supported by a communications channel of bandwidth  $B$ , at a signal to noise ratio,  $E_b/N_0$ , without incurring any transmission errors. This maximum bit rate is known as the channel capacity,  $C$ , and is given by the expression

$$C = B \log_2 \left( 1 + \frac{E_b}{N_0} \left( \frac{C}{B} \right) \right) \quad (1.1)$$

where  $C$  is in bits/sec,  $B$  is in Hz,  $E_b$  is the bit energy in J, and  $N_0$  is the single-sided noise power spectral density in W/Hz. According to this formula, an ideal communications system is defined as one in which the data can be transmitted without error at the maximum rate of  $R = C$  bits/sec. By re-arranging (1.1) and solving for  $E_b/N_0$ , one can obtain an explicit relationship between  $E_b/N_0$  and  $C/B$ .

$$\frac{E_b}{N_0} = \frac{2^{C/B} - 1}{C/B} \quad (1.2)$$

The resultant curve is shown in Figure 1-1, and the position of MSK (for a bit error rate of  $10^{-6}$ ) relative to the channel capacity boundary is indicated.

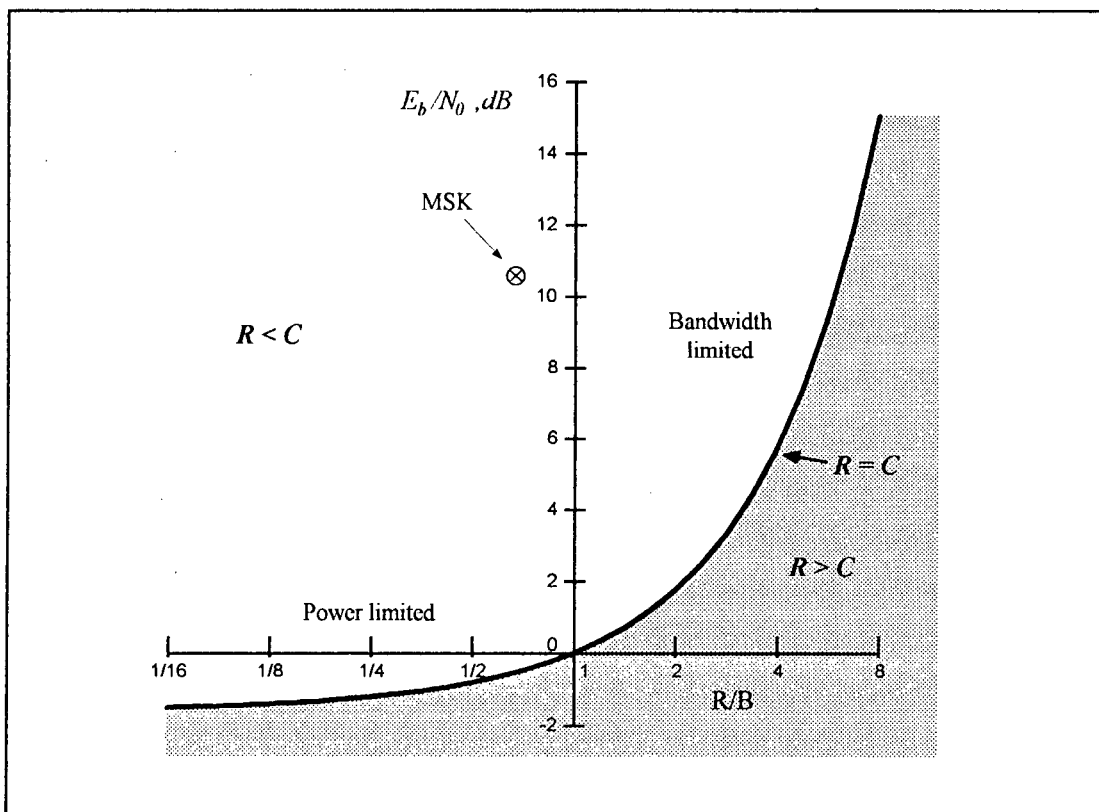


Figure 1-1: The position of MSK relative to the channel capacity boundary.

It can clearly be seen from Figure 1-1 that MSK falls approximately 11dB short of the Shannon capacity limit in terms of its power efficiency. This indicates that the scheme is actually very inefficient. The question must therefore be asked whether there is not another modulation scheme with similar properties to MSK which can be used for satellite channels (or any other channels) at an improved power efficiency over MSK. The answer to this question is yes. One such digital modulation scheme is a generalised form of Frequency Shift Keying (FSK) known as multi-h Continuous Phase Frequency Shift Keying (CPFSK). It has been shown by Anderson, Aulin and Sundberg [2] that coding gains of at least 4.5dB over MSK are achievable using this scheme. However, specific  $h$ -sets which achieve these gains have not been discovered by [2], nor have  $h$ -sets of size greater than four been investigated.

In 1996 Cuthbert [4] set about trying to find high performance  $h$ -sets of size 1 to 8 using a Population Based Incremental Learning (PBIL) search algorithm. Subsequent evaluation of his results, however, has revealed that his performance measurements were incorrect and his results therefore useless. This has left the questions regarding the performance capabilities of multi-h CPFSK largely unanswered.

With the above in mind, the main aim of this dissertation was to conduct a detailed investigation into the theoretical performance capabilities of multi-h CPFSK digital modulation, and to carefully verify all results. It is hoped that the work presented in this document will provide a solid basis for future research into the practical implementation of multi-h CPFSK.

## 1.2 Objectives of Thesis

Based on the above-stated aim, the objectives of this thesis were as follows:

- To acquire a good understanding of multi-h CPFSK and its performance characteristics.
- To investigate thoroughly the optimal decoding of multi-h CPFSK signals and the associated probability of error.
- To obtain some insight into the practical considerations involved in the implementation of the Viterbi Decoder.
- To conduct a search for high-performance  $h$ -sets, and to verify the theoretical coding gains of these  $h$ -sets by means of a computer simulation of the Viterbi Decoder.
- To consider the performance of sequential decoding algorithms.

### 1.3 Organisation of Thesis

The chapter layout for this document is depicted in Figure 1-2. Wherever possible, an attempt has been made to follow the thesis objectives in an orderly and coherent fashion.

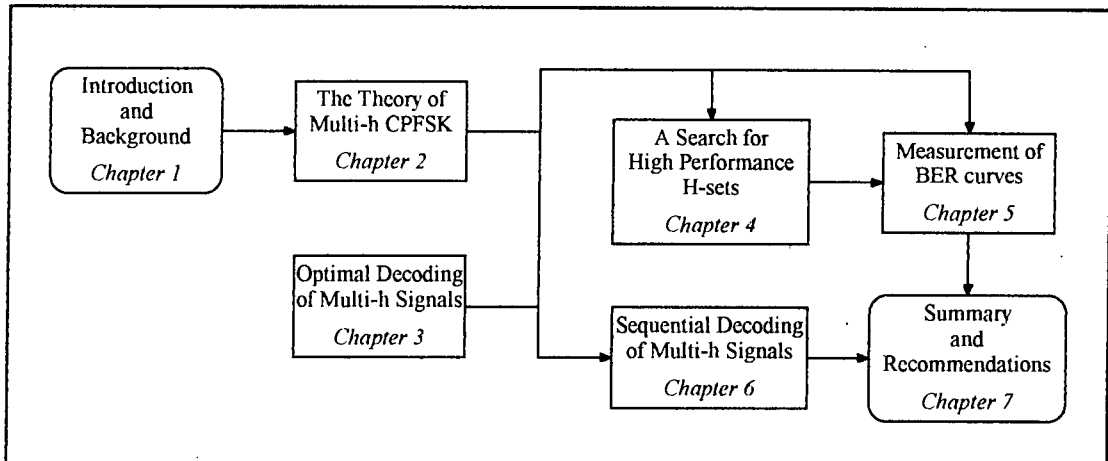


Figure 1-2: Thesis organisation and chapter layout.

Chapter 2 begins with the general theory of multi-h CPFSK and a discussion regarding the performance considerations for satellite channels. In Chapter 3 a full development of the optimal decoder is presented and some important implementation issues such as receiver structure and calculation of branch metrics are addressed. Chapter 4 then deals with the search for high-performance  $h$ -sets, including an in-depth examination of the search algorithm itself. The results of the search are tabulated and discussed. In Chapter 5 the coding gain results of Chapter 4 are verified by means of a computer simulation of the Viterbi Decoder. This chapter includes a description of the software implementation of the Viterbi Decoder as well as some related issues such as truncation depth and memory management. The results are plotted and discussed. Following this, Chapter 6 deals briefly with the performance of sequential decoding algorithms. Finally, the document is concluded in Chapter 7 with a brief summary of the work done and results obtained and recommendations are made for further research.

## Chapter 2

# The Theory of Multi-h CPFSK

In this chapter we review the structure of a digital communications system and the way in which multi-h CPFSK fits into that structure. We then consider the relationship between multi-h CPFSK and some of the other common digital modulation schemes, before presenting a mathematical and a graphical description of the multi-h signalling format. Finally we investigate the suitability of multi-h CPFSK for use in satellite communication channels.

### 2.1 Multi-h CPFSK and the Digital Communications System

The primary function of the transmitter/receiver pair in a digital communications system is to transfer digitally encoded information across a noisy analogue channel, in a manner that is both reliable and efficient. A block diagram of the system elements required to perform this task is presented in Figure 2-1.

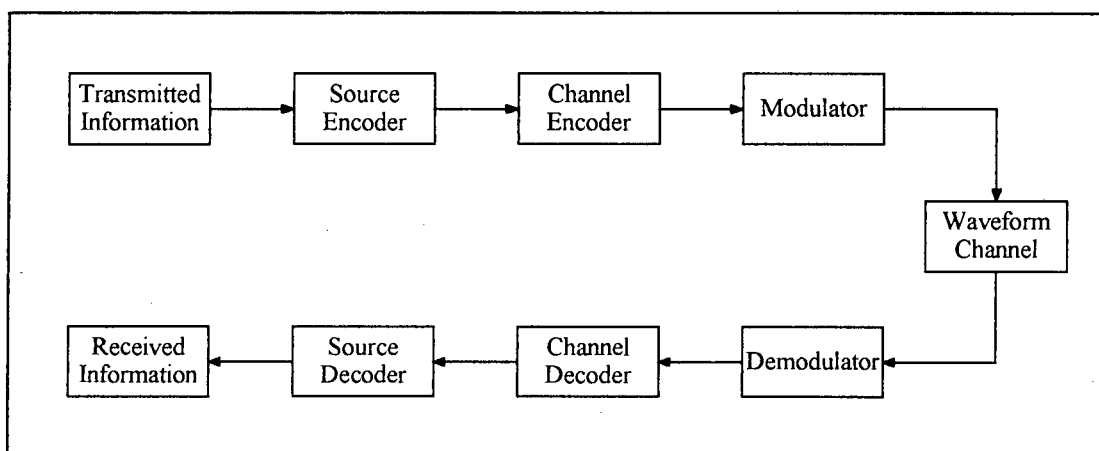


Figure 2-1: Block diagram of a digital communications system.

In the above figure, the source encoder is responsible for mapping the original transmitted symbol stream into a form in which redundancy has been removed. The output of the source encoder is a stream of binary digits (bits), representing the original information. A controlled amount of redundancy is then reintroduced to the signal by means of the channel encoder. The purpose of this added redundancy is to provide protection for the information against errors which may be introduced by noise in the channel. Finally, the task of the modulator is to convert the resulting digital data stream into an analogue waveform, suitable for transmission across the communications channel.

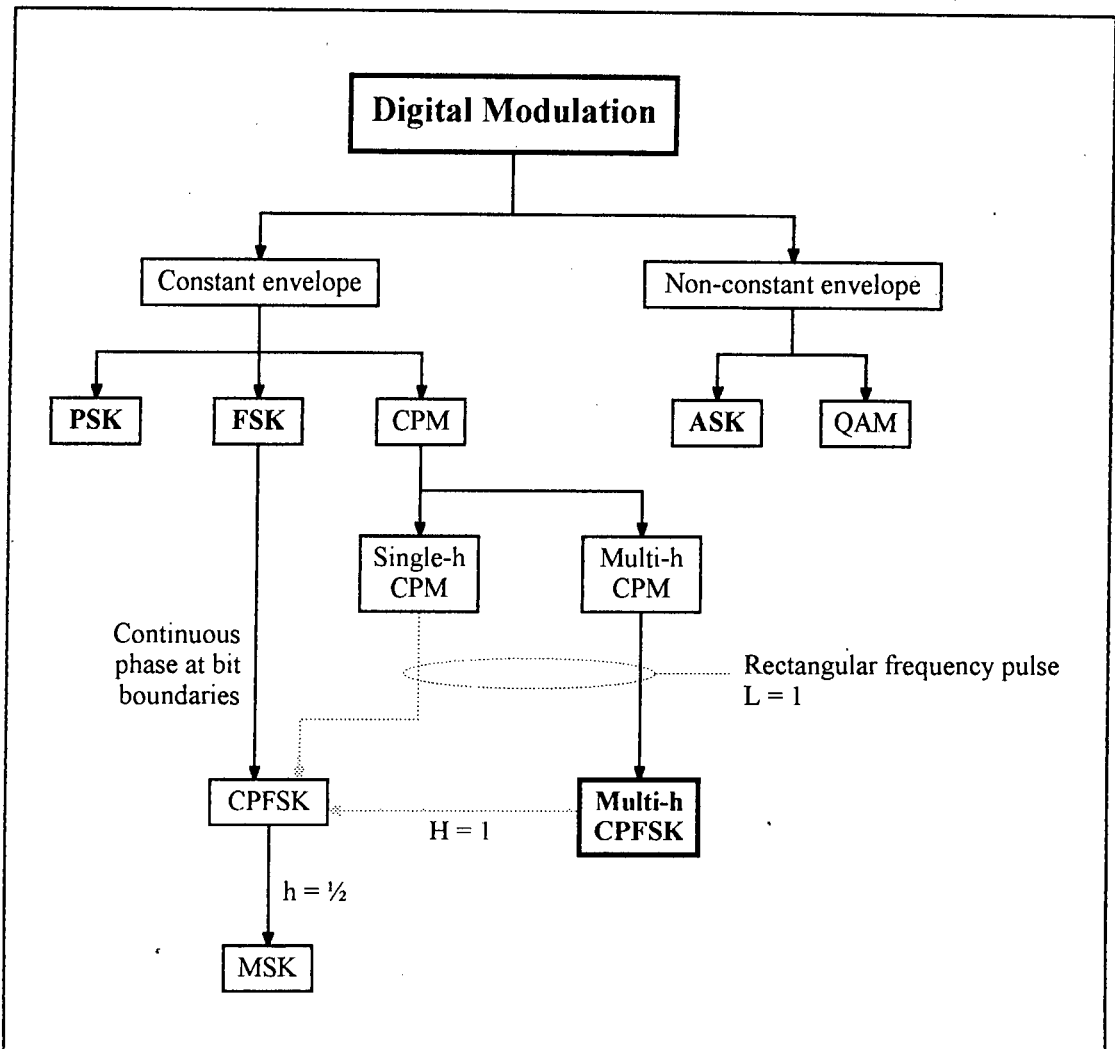
The communications channel places a number of restrictions on the analogue waveform, the most important of which are restrictions on signal power, signal bandwidth and signal envelope characteristics (these will be discussed in more detail in Section 2.5). The communications channel is also responsible for adding noise and distortions to the analogue waveform. For a non-fading, linear channel, the noise is best modelled as Additive White Gaussian Noise (AWGN).

At the receiver, the demodulator estimates the most likely encoded information that would produce the waveform actually received. The demodulated data stream is then decoded by the channel decoder and the source decoder. The channel decoder may incorporate error correction methods that make use of the redundancy introduced by the channel encoder.

Over the past few decades, the majority of research in digital modulation theory has been in the area of combined channel encoding and modulation (using schemes such as Trellis coded modulation and, more recently, multi-h CPFSK). By combining these two elements, it has been found that substantial coding gains may be achieved, without the increased bandwidth requirements of conventional channel encoding methods (such as convolutional coding). This means that a multi-h modulator can operate at a lower transmit power than an equivalent uncoded scheme such as FSK, but at the same time exhibit identical bandwidth efficiency in terms of the number of bits per second that are transmitted per hertz of bandwidth.

## **2.2 Multi-h CPFSK in relation to Other Digital Modulation Schemes**

Before describing multi-h CPFSK itself, it is useful to illustrate its relationship to other well known digital modulation schemes. Figure 2-2 (adapted from [3]) shows a simplified digital modulation tree, comprised of the main categories of modulation schemes. On the first level of this tree a distinction is made between



**Figure 2-2:** Simplified digital modulation tree.  $L$  is the length of the frequency shaping pulse in bit periods,  $H$  is the number of  $h$  values in the  $h$ -set,  $h$  is the modulation index.

constant and non-constant envelope schemes. It is important to note that, in general, the constant envelope property of a modulated signal holds only while that signal is unfiltered: any signal with phase discontinuities will have a non-constant envelope after filtering, and only continuous-phase signals retain an approximately constant envelope after bandlimiting. The reason for making this distinction will become apparent in the discussion on the importance of the constant envelope property (Section 2.5.1).

On the second level of the digital modulation tree are the three most basic modulation schemes. A sinusoidal signal can be modulated in amplitude, phase or frequency and this produces the fundamental Amplitude Shift Keying (ASK), Frequency Shift Keying (FSK) and Phase Shift Keying (PSK) techniques. Continuous Phase Modulation (CPM) is a term used to describe any phase modulation technique which does not generate phase discontinuities in the

modulated waveform. Quadrature Amplitude Modulation (QAM) is a general form of ASK in which the digital data is demultiplexed into two channels which amplitude-modulate both an in-phase and a quadrature-phase carrier.

Finally, the tree shows two ways of arriving at multi-h CPFSK: via CPM as a special case of multi-h CPM, and via FSK as a generalised form of CPFSK.

### ***Multi-h CPFSK as a special case of CPM***

Multi-h CPFSK may be derived from CPM (for a full description of CPM the reader is referred to [2]) by using a rectangular frequency pulse (LREC) with a pulse length of  $L = 1$ . For  $L = 1$ , the length of time during which the carrier phase is influenced by a particular data bit is 1 bit period.

### ***Multi-h CPFSK as a generalised form of CPFSK***

Multi-h CPFSK may also be derived from FSK as a generalised form of CPFSK. CPFSK is the same as FSK in that it represents the data with sinusoids of different frequencies. However, in CPFSK, the modulator is constructed in such a way as to ensure phase continuity of the modulated waveform at the bit boundaries. In multi-h CPFSK, the modulation index,  $h$ , is changed in a cyclical manner from bit period to bit period.

## **2.3 Mathematical Representation of Multi-h CPFSK Signals**

A full mathematical representation of multi-h CPFSK is complex and rather confusing and will not, therefore, be presented here. However, the reader is referred to Anderson, Aulin and Sundberg [2] for a detailed explanation of the subject. For the purposes of this dissertation, a simplified, more intuitive representation will be given. It should be noted that only the transmission of binary data will be considered, as this is conceptually easy and is also readily generalised to an M-level system. The digital data stream will be represented by  $d_i$

$$d_i \equiv d_0, d_1, d_2, \dots \quad (2.1)$$

where  $i$  is the number of the current data bit and  $d_i$  is defined by

$$\begin{aligned} d_i &= -1 && \text{for a binary '0'} \\ d_i &= +1 && \text{for a binary '1'} \end{aligned} \quad (2.2)$$

Since multi-h CPFSK is a form of frequency shift keying, we shall begin this discussion with a definition of FSK. Frequency Shift Keying (FSK) is a class of digital modulation in which the two binary symbols, '0' and '1', are represented by sinusoids of different frequencies. The modulated waveform,  $s(t)$ , for FSK is given by

$$s(t)_{FSK} = \sqrt{\frac{2E_b}{T_b}} \cos(2\pi(f_c + d_i \Delta f)t) \quad (2.3)$$

where  $E_b$  is the bit energy,  $T_b$  is the bit period,  $f_c$  is the carrier frequency in hertz and  $\Delta f$  is the offset frequency. The ratio of the difference between the two signalling frequencies and the bit period is defined as the modulation index,  $h$ .

$$h = \frac{2\Delta f}{T_b} \quad (2.4)$$

As mentioned in Section 2.2, Continuous Phase Frequency Shift Keying (CPFSK) is a special case of FSK (with continuous phase at the bit boundaries), and multi-h CPFSK is a generalised form of CPFSK. In multi-h CPFSK, the modulation index,  $h$ , is replaced by  $h_i$  which is chosen in a cyclic fashion from a set of indices.

$$h_i \in \{h_0, h_1, h_2, \dots, h_{H-1}\} \\ h_{i+H} = h_i \quad (2.5)$$

In practice,  $h_i$  is restricted to be of the form  $h_i = p_i/q$  where  $p_i$  and  $q$  are integers and  $h_i$  is therefore rational. To conserve bandwidth, the restriction  $p_i < q$  is placed on all  $p_i$ .

Mathematically, multi-h CPFSK can be expressed as

$$s(t)_{multi-h} = \sqrt{\frac{2E_b}{T_b}} \cos\left(2\pi\left(f_c + \frac{d_i h_i T_b}{2}\right)t + \theta_i + \theta_0\right) \quad (2.6)$$

where  $\theta_0$  is the initial carrier phase and  $\theta_i$  represents the accumulated phase of the modulated signal up to the beginning of bit period  $i$ . The latter term is necessary to ensure continuous phase between successive CPFSK bit periods, and is given by

$$\theta_i = \sum_{k=0}^{i-1} 2\pi\left(f_c + \frac{d_k h_k T_b}{2}\right)T_b - 2\pi i\left(f_c + \frac{d_i h_i T_b}{2}\right)T_b \quad (2.7)$$

At this point it is useful to limit the general definition of multi-h CPFSK by making some assumptions. The purpose of these assumptions is purely to simplify the mathematics of the modulation scheme, as they have no effect on its performance characteristics. The assumptions are as follows:

- 1) Bit energy,  $E_b = 1$  joule
- 2) Bit period,  $T_b = 1$  second
- 3) Carrier frequency,  $f_c$ , is an integer multiple of the bit rate.
- 4) Initial carrier phase (at time  $t = 0$ ),  $\theta_0 = 0$

From assumptions (3) and (4) it is clear that the phase of the carrier waveform at the beginning of each bit period is restricted to zero. This means that the total modulated signal,  $s(t)$ , can now be easily represented, on a bit by bit basis, as a series of delayed waveform pulses,  $w_i(t)$ , each defined over the period  $0 < t < T_b$  and corresponding to data bit  $d_i$ . The mathematical expression for  $s(t)$  becomes

$$s(t)_{multi-h} = \sum_i w_i(t - iT_b) \quad (2.8)$$

where  $w_i(t)$  is given (from equation (2.6) and assumptions (1), (2) and (4)) by

$$w_i(t) = \begin{cases} \sqrt{2} \cos\left(2\pi\left(f_c + \frac{d_i h_i}{2}\right)t + \theta_i\right) & 0 < t < T_b \\ 0 & \text{elsewhere} \end{cases} \quad (2.9)$$

and  $\theta_i$  may be expressed (from equation (2.7) and assumption (3)) as

$$\theta_i = \sum_{k=0}^{i-1} \pi d_k h_k \quad (2.10)$$

Equations (2.8) to (2.10) give a useful mathematical model of multi-h CPFSK for the calculation of its performance and for simulation purposes. The model has therefore been used extensively for the work presented in the later chapters of this dissertation, where these subjects are addressed in detail.

## 2.4 Graphical Representation of Multi-h CPFSK Signals

Since every successive bit in a CPFSK signal relies on the frequency and phase of the bits preceding it, a certain amount of memory is introduced into the multi-h

CPFSK waveform. This memory is expressed mathematically by the term  $\theta_i$  in equation (2.10) and is best illustrated in a graphical form by means of a phase trellis diagram. The phase trellis diagram is a plot of the excess phase of the modulated waveform (with respect to the carrier waveform) for every possible transmitted sequence of bits, starting at time  $t = 0$ .

As an example, the specific set of  $h$ -values  $\{h_1, h_2\} = \{1/4, 2/4\}$  is considered and its resultant phase trellis diagram is pictured in Figure 2-3.

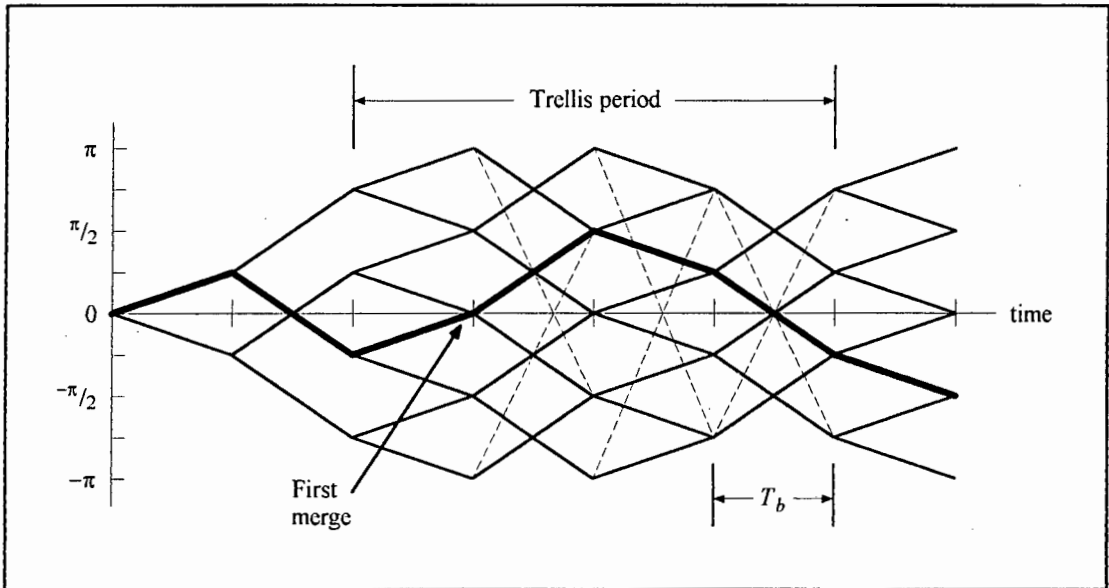


Figure 2-3: Phase trellis diagram for  $\{h_1, h_2\} = \{1/4, 2/4\}$ .

The path shown in bold in the diagram corresponds to the bit pattern '1011000'. Note that, due to the inherent memory of the multi-h signalling format, this path is unique until the end of the third bit period, where it merges with the path for the bit sequence '010[1000]'. This means that the decoder has three bit periods instead of one in which to decide which of the two bit sequences is more likely to have been sent. Note also that the excess phase at the end of each bit period in Figure 2-3 takes on values from a discrete set. This will only happen for a set of  $h_i$  which is rational.

## 2.5 Satellite Channels and Multi-h CPFSK

To understand why multi-h CPFSK is attractive for use on satellite channels it is instructive to review, firstly, the constraints that these channels place on the choice of modulation scheme, and then to determine how multi-h CPFSK meets these constraints and to see how it compares to other schemes.

### **2.5.1 Constraints placed on the choice of modulation scheme**

In general, the constraints placed on the choice of modulation scheme are based on the following criteria (extracted from [3]):

#### ***Bandwidth efficiency***

As the demand for wireless communication services increases, the frequency spectrum becomes more and more congested. Most communications systems are forced to operate within a limited slice of the frequency spectrum and this calls for bandwidth efficient modulation schemes which transmit as much information as possible in the given bandwidth.

Bandwidth efficiency is defined as the ratio between the bit rate,  $R_b$ , and the bandwidth required to sustain transmission at that rate. This ratio depends somewhat on the definition of system bandwidth. Three commonly used figures are the Nyquist bandwidth, the null-to-null bandwidth and the 99% energy constraint bandwidth [1].

#### ***Power efficiency***

The number of bit errors made in receiving a modulated signal depends on the transmitted signal power and the noise power introduced by the channel. Clearly, modulation schemes that minimise the probability of error for a given signal-to-noise ratio are preferred.

Power efficiency is thus defined as the bit-energy-to-noise-density ratio ( $E_b/N_0$ ) required to receive data at a given bit error probability in an AWGN channel.

#### ***Synchronization***

Closely related to the issue of power efficiency is the type of detection used at the demodulator. Coherent detection, where the receiver is synchronised to the transmitter, yields better power efficiency than non-coherent detection in a non-fading AWGN channel.

In order to synchronise the receiver, one must recover carrier frequency and phase, as well as bit timing information. The ease of synchronising depends on the modulation scheme, but the most efficient schemes are those which are self-synchronising. Such schemes have characteristics that enable one to recover the carrier and clock from the modulated signal itself, without the need for a discrete signal component at the carrier frequency or a pilot tone, which would be wasteful

of transmitter power and channel bandwidth.

### ***Envelope characteristics***

In systems where available power is limited, it is common to use non-linear power amplifiers because of their high efficiency. Solid state class C amplifiers, for example, are operated in or near saturation of the devices and are thus highly non-linear, but they can achieve efficiencies of up to 90%. Travelling-wave tube amplifiers are also highly non-linear, and are often used on satellite systems.

The use of such amplifiers calls for modulation schemes with a constant envelope, as any non-linearities applied to a signal with amplitude fluctuations creates severe distortion and out-of-band sidelobes.

### ***System complexity***

In the implementation of a digital communications system, the overall system complexity depends on the requirements in terms of each of the factors mentioned above. In general, systems requiring high bandwidth efficiency, high power efficiency and coherent detection are far more complex to implement than systems with less stringent requirements.

## ***2.5.2 Suitability of multi-h CPFSK to satellite channels***

The most restrictive aspect of operating a satellite transmitter is that the available power is severely limited. Typical low earth orbiting (LEO) microsatellite transmitter powers are in the region of 5 to 10W, and this means that it is important to employ a modulation scheme with good power efficiency. Also, non-linear power amplifiers are invariably used because of their efficiency, which means that a constant envelope scheme is required. This immediately limits the choice of modulation schemes to one of the CPM schemes. A secondary, but by no means unimportant, consideration is that of bandwidth efficiency.

Of all the CPM schemes, those based on CPFSK are the simplest to implement. In fact, most practical satellite systems at the present time employ either standard FSK/CPFSK modulation (coherently or non-coherently detected) or the CPFSK-based Minimum Shift Keying (MSK). Unfortunately, multi-h CPFSK suffers a complexity disadvantage over the simpler FSK or MSK schemes. However, in terms of power efficiency, good multi-h CPFSK codes perform significantly better than the single-h techniques.

As a qualitative assessment of the suitability of multi-h CPFSK to satellite

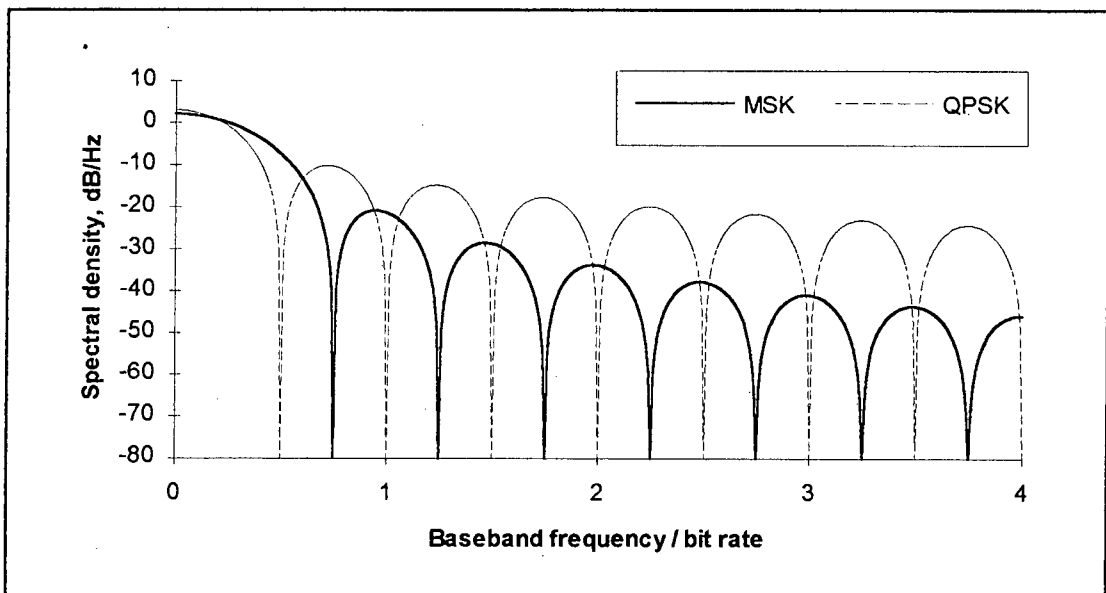
channels, consideration is given to each of the modulation scheme constraints and the corresponding multi-h CPFSK performance characteristics.

### **Bandwidth efficiency of multi-h CPFSK**

Bandwidth efficiency is defined in terms of the power spectrum for a particular modulation scheme. The power spectrum for multi-h CPFSK is dependent on the specific code used, and therefore cannot be plotted for the general case (for a basic discussion regarding power spectrum calculations for multi-h codes, the reader is referred to Ziemer and Peterson [9]). However, some comments can be made about the general spectral characteristics.

The most important feature of the multi-h CPFSK spectrum is a fast spectral roll-off. This is due to the fact that there are no phase discontinuities in the modulated signal (although the first derivative is discontinuous), so that the power spectral envelope decays as  $f^{-4}$  or at a rate of 12 dB/octave. This is identical to the decay rate for MSK (a single-h scheme,  $h = 1/2$ ). As a comparison, the decay rate for a non-continuous phase scheme such as PSK is 6 dB/octave. The other main feature of the power spectrum is the width of the first spectral lobe in Hz per bits/second. For multi-h CPFSK, this is generally in the region of 0.8, again a similar value to MSK. In other words, the first spectral null occurs at a frequency deviation of approximately 0.8 times the bit rate from the centre frequency. For a comparative non-CPM scheme such as QPSK, the first spectral null occurs at a value 0.5.

As an illustration of the approximate power spectrum for multi-h CPFSK, the spectrum for MSK is plotted in Figure 2-4. A comparison with QPSK is provided.



**Figure 2-4:** Baseband-equivalent power spectra for MSK and QPSK.

### Power efficiency of multi-h CPFSK

Power efficiency is determined by the signal-to-noise ratio required to give a particular probability of error. For optimal decoding, the probability of error for multi-h CPFSK is given by

$$P_e \approx Q \left( \sqrt{\frac{E_b \cdot d_{min}^2}{N_0}} \right) \quad (2.11)$$

where  $E_b$  is the bit energy,  $N_0$  is the single-sided noise power spectral density, and  $d_{min}^2$  is known as the normalised minimum squared euclidean distance of the code. The value of  $d_{min}^2$  for MSK and QPSK is equal to 2. For multi-h CPFSK it may be as high as 6 or greater (see Section 4.6). Thus, for a particular probability of error, multi-h CPFSK requires a lower bit energy than that required for MSK or QPSK.

For comparison, the probability of error curves for multi-h (with  $d_{min}^2 = 6$ ) and MSK are plotted in Figure 2-5.

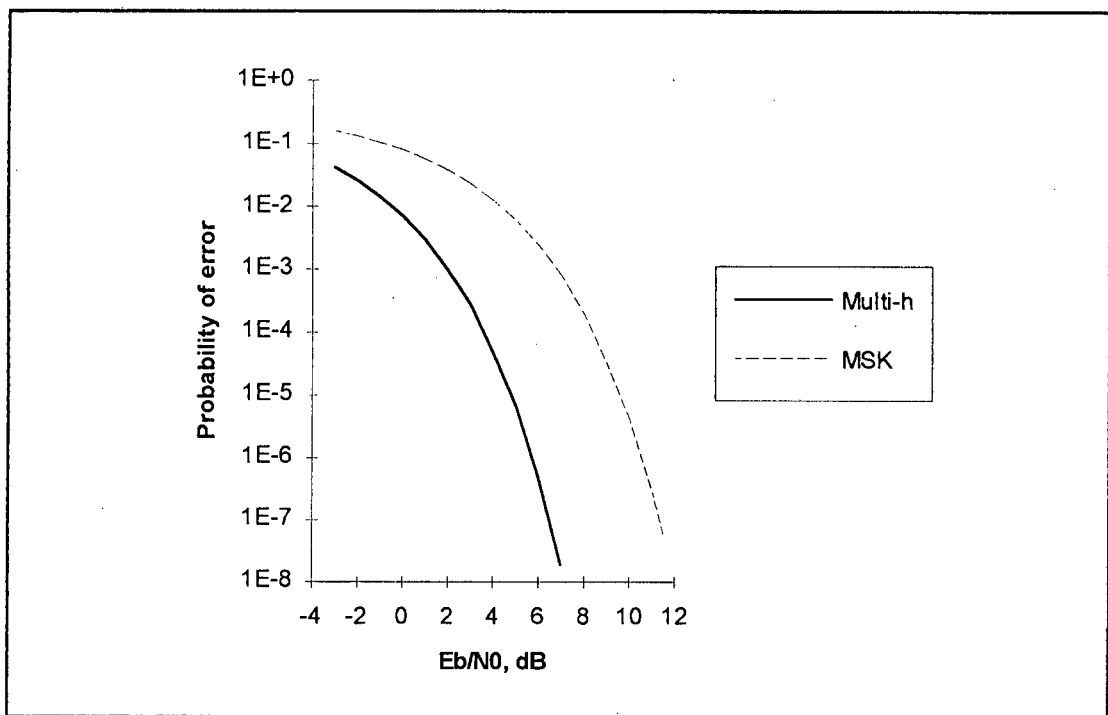


Figure 2-5: Probability of error curves for multi-h CPFSK ( $d_{min}^2 = 6$ ) and MSK.

### Synchronisation issues for multi-h CPFSK

In order to take advantage of the coding gain inherent to multi-h CPFSK, it is necessary to employ coherent detection. This means that the multi-h receiver requires complex synchronisation circuitry, a disadvantage for satellite use.

The complexity of the synchronisation circuitry is directly proportional to the length of the  $h$ -set and to the size of the  $h$ -set denominator. Whereas a coherent MSK receiver requires only two synchronised oscillators, one for the upper frequency and one for the lower, a multi- $h$  receiver for an  $h$ -set of length 4 requires a total of eight (4 times 2) coherent oscillators. Furthermore, a multi- $h$  receiver has the added complexity of having to extract so-called superbaud timing from the signal, a clock to indicate the start of each  $h$ -set cycle.

#### ***Envelope characteristics of multi- $h$ CPFSK***

As mentioned previously, multi- $h$  CPFSK belongs to the broad class of CPM schemes which display constant envelope properties. This is a huge advantage for use in satellite communications as it allows the use of efficient, non-linear power amplifiers.

## Chapter 3

# Optimal Decoding of Multi-h Signals in the presence of AWGN

In this chapter a full development of the maximum likelihood decoder for multi-h CPFSK is presented. The chapter begins with a discussion of the Viterbi decoding algorithm, followed by an investigation into the implementation of and the probability of error for the maximum likelihood detector. Finally, the complete Viterbi Decoder is examined and its probability of error is found.

### 3.1 The Viterbi Algorithm

The Viterbi Algorithm (VA) is an elegant method for performing maximum likelihood decoding of convolutional-type codes. It was first described in mathematical terms by A.J. Viterbi [8] in 1967. Due to the fact that convolutional codes exhibit a trellis structure similar to the phase trellis of multi-h CPFSK, it is possible to apply the principles of Viterbi decoding to the maximum likelihood decoding of multi-h CPFSK.

The function of the Viterbi Algorithm is, essentially, to select the most likely transmitted data sequence out of the many possible paths in a trellis diagram. It does so by keeping track of a so-called path metric, or likelihood function, for each path originating at time  $t = 0$  in the phase trellis. The path metric is an indication of the likelihood that a particular path is, in fact, the path that was transmitted. At every point in the phase trellis where multiple paths meet, all but the most likely path arriving at that point are discarded.

The method is optimal due to the fact that a path is only discarded at that point

where it meets with a more likely path. No further information can be gained about the likelihood of a particular path being the transmitted path after it has merged with another path, as the paths become indistinguishable from one another after the merge.

### 3.2 The Maximum Likelihood Detector

The process of calculating the path metric and using it to choose the most likely transmitted path is known as maximum likelihood detection. Expressed in simple terms, the maximum likelihood problem is as follows:

*Given two or more possible transmitted signals and a certain received signal comprising of one of the possible transmitted signals plus AWGN, how does one compare the received signal to the possible transmitted signals so that the likelihood of choosing the signal which was actually transmitted is maximised?*

It turns out that the most likely signal is that signal which has the smallest squared euclidean distance,  $D^2$ , from the received signal, where  $D^2$  is defined as

$$D^2 = \int (y(t) - x(t))^2 dt \quad (3.1)$$

for two signals  $x(t)$  and  $y(t)$ . The proof of this equation will be given in the next section in conjunction with a calculation for the resulting probability of error.

#### 3.2.1 Derivation of the maximum likelihood detector

We begin by considering two possible transmitted signals,  $s_1(t)$  and  $s_2(t)$ , both of which are lowpass bandlimited to  $f_s/2$  Hz. The transmitter transmits one of  $s_i(t)$  through an AWGN channel where white noise with a double-sided power spectral density of  $N_0/2$  watts per hertz is added to the signal. We shall denote the transmitted signal by  $s(t)$  and the added noise component by  $n(t)$ . The waveform emerging from the channel,  $x(t)$ , may then be expressed as

$$x(t) = s(t) + n(t) \quad (3.2)$$

At the receiver,  $x(t)$  is passed through an ideal low-pass filter with cut-off frequency  $f_s/2$  Hz. The purpose of the filter is to limit the bandwidth of the noise component,  $n(t)$ , to the same cut-off frequency as the signal component, thereby

reducing the noise power to a minimum. The resulting waveform,  $x'(t)$ , is sampled at a sampling rate of  $f_s$  Hz (sampling period  $T_s$  sec) and passed on to the maximum likelihood detector. The entire system is depicted in Figure 3-1.

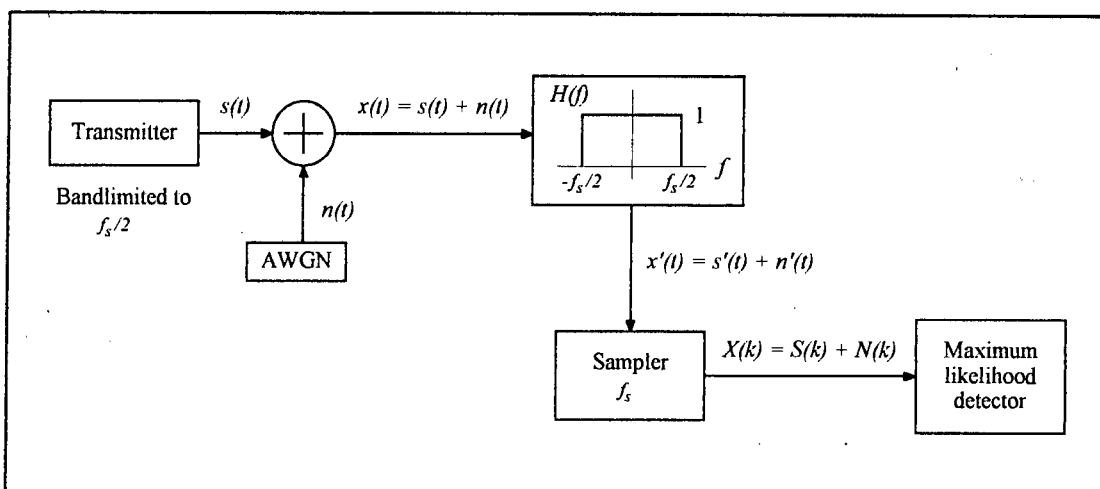


Figure 3-1: Communications system for derivation of maximum likelihood detector.

It should be noted that, since the sampling rate is equal to twice the bandwidth of filtered waveform, the Nyquist sampling criterion is satisfied and no aliasing or loss of information occurs. The sampled waveform is given by

$$X(k) = S(k) + N(k) \quad k = 0, 1, 2, \dots \quad (3.3)$$

where the simplification

$$X(k) \equiv x'(kT_s) \cdot \delta(t - kT_s) \quad (3.4)$$

has been used.

We now investigate the terms  $S(k)$  and  $N(k)$  more closely. Firstly, we note that the transmitted signal,  $s(t)$ , is not affected by the receiver filter,  $H(f)$ , as it has no frequency components outside of the filter bandwidth. We can therefore say that the sampled signal,  $S(k)$  ( $k = 0, 1, 2, \dots$ ), is equal to the instantaneous value of  $s(t)$  at time  $t = kT_s$ .

$$S(k) = s(kT_s) \cdot \delta(t - kT_s) \quad k = 0, 1, 2, \dots \quad (3.5)$$

For the noise component, we are interested in the noise variance, or power, at the output of the receiver filter (Gaussianly distributed noise retains its Gaussian distribution properties when passed through a linear filter). The noise power,  $\sigma_n^2$ ,

is given by

$$\sigma_n^2 = \int_{-\infty}^{\infty} S_n(f) |H(f)|^2 df \quad (3.6)$$

where  $S_n(f)$  is the input noise power spectral density, and  $H(f)$  is the transfer function of the linear system. Substituting for  $S_n(f)$  and  $H(f)$  in (3.6), we obtain for the noise power

$$\sigma_n^2 = \int_{-\frac{f_s}{2}}^{\frac{f_s}{2}} \frac{N_0}{2} df = \frac{N_0 f_s}{2} \quad (3.7)$$

Since the successive samples of bandlimited white noise are uncorrelated with one another for Nyquist sampling, the sampled noise component,  $N(k)$  ( $k = 0, 1, 2, \dots$ ), is a random Gaussian variable with zero mean and variance given by  $\sigma_n^2$  in equation (3.7).

At this point we turn our attention to the maximum likelihood detection of  $s_i(t)$ . We shall assume that the possible transmitted signals,  $s_1(t)$  and  $s_2(t)$ , are each of length  $nT_s$  seconds ( $n$  an integer), and that they produce sampled signal components  $S_1(k)$  and  $S_2(k)$  ( $k = 0, 1, 2, \dots, n-1$ ) when transmitted through our communications system. Using these  $n$  samples, each of the possible transmitted signals may be represented by a unique point in an  $n$ -dimensional signal space. This concept is illustrated for a simple 2-dimensional case in Figure 3-2.

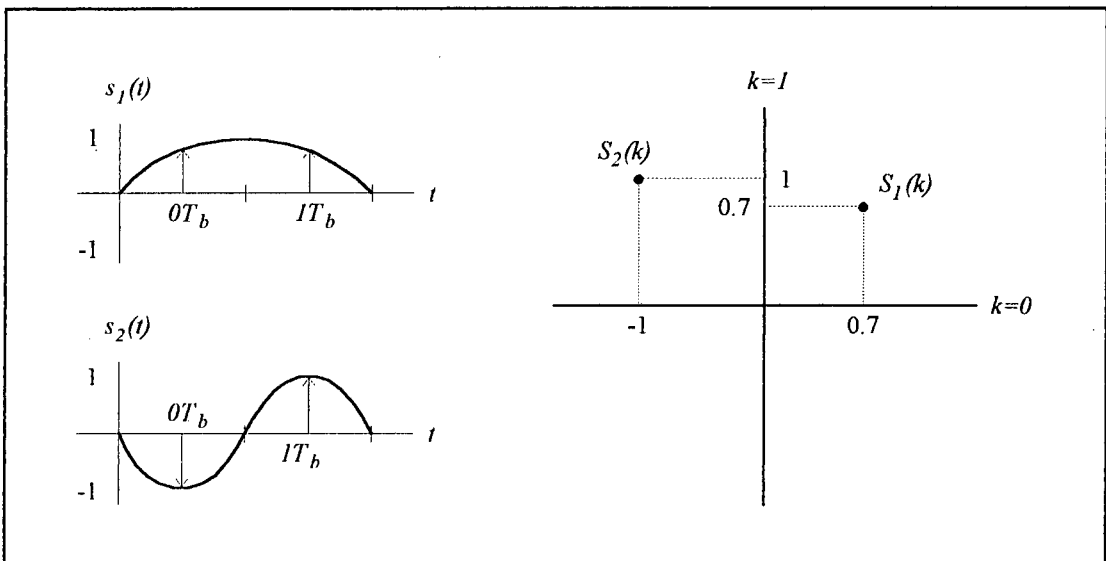


Figure 3-2: Example of signal space representation for  $s_1(t)$  and  $s_2(t)$ .

The maximum likelihood detector receives the sampled signal,  $X(k)$ , comprising of  $S_1(k)$  or  $S_2(k)$  plus a noise component,  $N(k)$ . As mentioned previously, the successive samples of  $N(k)$  are uncorrelated. Therefore, in terms of our signal space,  $N(k)$  may be thought of as an  $n$ -dimensional Gaussian probability distribution function with zero mean and variance  $\sigma_n^2$  given by equation (3.7).  $X(k)$  is defined by the same Gaussian distribution function, but with a mean of  $S_1(k)$  or  $S_2(k)$ . The signal space representation of  $X(k)$  for the 2-dimensional example is pictured in Figure 3-3.

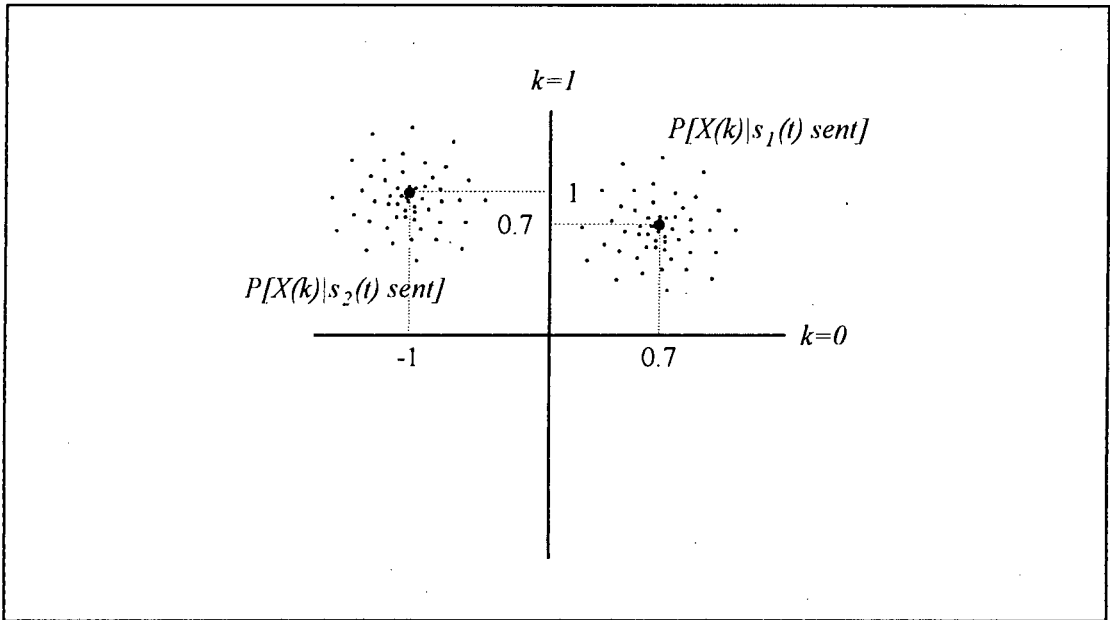


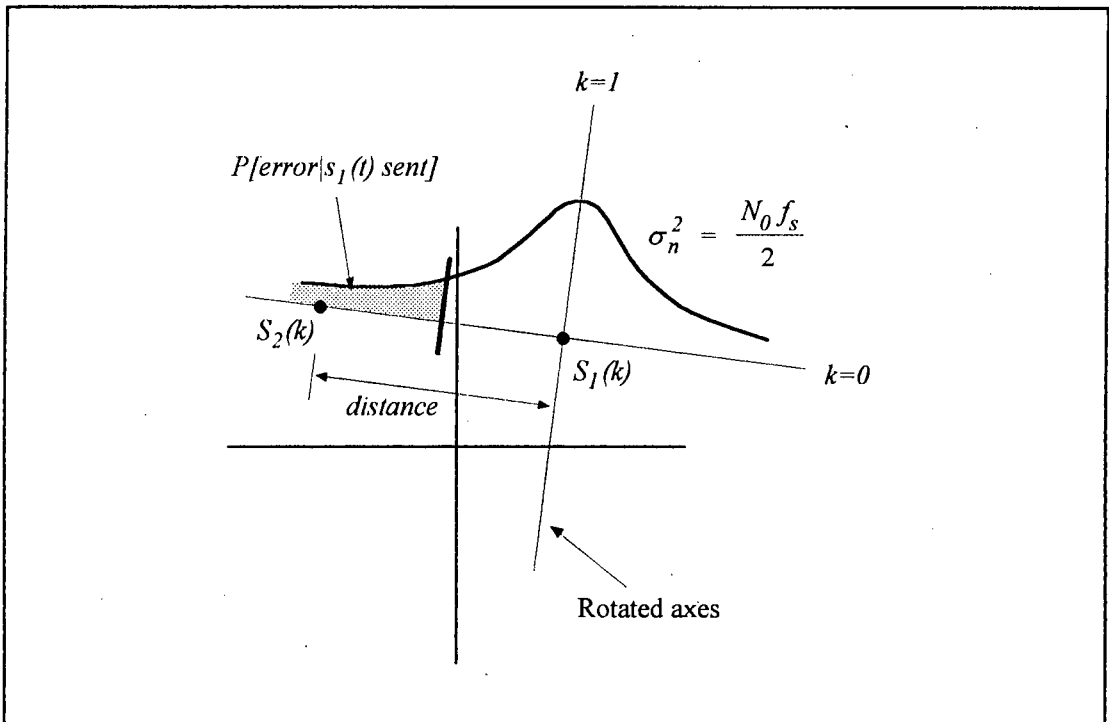
Figure 3-3: Example of probability distribution function for  $X(k)$ .

From basic probability theory, the most likely transmitted signal is that signal for which the probability distribution function of  $X(k)$  is largest for the received signal  $X(k)$ . Since the  $n$ -dimensional Gaussian distribution function is a monotonically decreasing function of  $X(k)$  as  $X(k)$  moves away from  $S_1(k)$  or  $S_2(k)$ , the maximum likelihood transmitted signal is that signal which is closest to the received signal in the signal space representation, i.e. that signal which has the smallest squared euclidean distance from the received signal. The squared euclidean distance between two signals  $X(k)$  and  $Y(k)$  is defined by the sum

$$distance^2 = \sum_{k=0}^{n-1} (Y(k) - X(k))^2 \quad (3.8)$$

Given the above maximum likelihood rule, the probability of making a detection error is equal to the probability that  $X(k)$  is closer to  $S_1(k)$  given that  $S_2(k)$  was sent, or equivalently that  $X(k)$  is closer to  $S_2(k)$  given that  $S_1(k)$  was sent.

In order to find a quantitative expression for the probability of error, it is necessary to rotate the signal space axes so that one of the axes is parallel to a straight line connecting  $S_1(k)$  and  $S_2(k)$ . This has no effect on the probability of error itself, as the  $n$ -dimensional Gaussian distribution is symmetric over all  $n$  dimensions. However, it does simplify the condition under which an error will occur. For the rotated signal space, an error will occur only if  $X(k)$  is closer to the incorrect  $S_i(k)$  along the direction of the parallel axis. In other words, the probability of error becomes dependent on just a single noise sample in the direction of the parallel axis. This noise sample may be treated as a 1-dimensional Gaussian probability distribution as illustrated in Figure 3-4.



**Figure 3-4:** Example of rotated signal space for probability of error calculation.

The probability of error is then given by

$$P_e = Q\left(\frac{\text{distance}}{2\sigma_n}\right) = Q\left(\sqrt{\frac{\text{distance}^2}{2N_0f_s}}\right) \quad (3.9)$$

where  $\text{distance}^2$  is defined as the squared distance between  $S_1(k)$  and  $S_2(k)$  according to equation (3.8),  $\sigma_n$  is obtained by taking the square root of equation (3.7) and the  $Q(x)$  function is as defined in Appendix A.

All that now remains is to relate the signal space quantity,  $\text{distance}^2$ , to the continuous squared euclidean distance,  $D^2$ , defined in equation (3.1). This may

be achieved by noting that  $D^2$  can be approximated by means of the rectangular rule of integration as follows:

$$D^2 = \int (y(t) - x(t))^2 dt \approx \sum_{k=0}^{n-1} (Y(k) - X(k))^2 T_s \quad (3.10)$$

However, the right-hand side of equation (3.10) is simply equal to  $distance^2$  multiplied by  $T_s$ . Furthermore, if we let the sampling frequency,  $f_s$ , tend to infinity (sampling period  $T_s \rightarrow 0$ ), then the approximation in equation (3.10) becomes an exact equality (from the basic definition of an integral) and our sampled receiver becomes equivalent to the continuous case. This also removes the restriction that the transmitted signals be bandlimited, as well as removing the need for a low-pass filter at the input to the receiver. Thus, the relationship between  $distance^2$  and  $D^2$  (as  $f_s \rightarrow \infty$ ) may be given by

$$distance^2 = \frac{D^2}{T_s} = D^2 f_s \quad (3.11)$$

Substituting the above result into the maximum likelihood detector rule, we find that the maximum likelihood transmitted signal is that signal which has the smallest squared euclidean distance,  $D^2$ , from the received signal, where  $D^2$  is defined by equation (3.1). The associated probability of error is found by substituting (3.11) into (3.9), giving

$$P_e = Q\left(\sqrt{\frac{D^2}{2N_0}}\right) \quad (3.12)$$

Unfortunately, the squared euclidean distance,  $D^2$ , is difficult to calculate in a practical implementation of the maximum likelihood detector, and therefore an equivalent quantity is required. As it turns out, for multi-h CPFSK signals the correlation between the received signal and the possible transmitted signals gives an optimum indication of the most likely transmitted signal. The resulting detector is known as a correlator receiver, and will be discussed in the following section.

### 3.2.2 Implementation as a correlator receiver

The task of the maximum likelihood detector is to choose the maximum likelihood transmitted signal from two or more possible transmitted signals. In the previous section it was shown that the optimum way in which to do this is to choose the

signal with the smallest squared euclidean distance,  $D^2$ , from the received signal. However, since  $D^2$  is difficult to calculate, a simplification is required. The correlator receiver is derived by expanding and simplifying the equation for squared euclidean distance.

Consider, once again, a set of possible transmitted signals,  $s_i(t)$  ( $i = 1, 2, 3, \dots$ ), and a certain received signal,  $x(t)$ , where all the signals are of the same length. In order to find the most likely transmitted signal, the quantity  $D^2$  must be minimised over all  $i$ , where  $D^2$  is given by

$$D^2 = \int (s_i(t) - x(t))^2 dt \quad (3.13)$$

Expanding (3.13), we obtain

$$D^2 = \int (s_i(t)^2 - 2s_i(t)x(t) + x(t)^2) dt \quad (3.14)$$

We now note that the quantity  $x(t)^2$  has no dependence on the minimisation parameter  $i$ . Also, for multi-h CPFSK (a constant envelope scheme) the quantity  $\int s_i(t) dt$  is constant over all  $i$ . These two terms may therefore be ignored. The minimisation of  $D^2$  thus reduces to minimising the quantity

$$\int (-2s_i(t)x(t)) dt \quad (3.15)$$

Equivalently, we may maximise

$$\int s_i(t)x(t) dt \quad (3.16)$$

where the integral in (3.16) is simply the correlation between  $s_i(t)$  and  $x(t)$ . The probability of error for the correlator receiver remains identical to that for the maximum likelihood detector. A proof of this is provided in Appendix B.

### 3.3 The Complete Viterbi Decoder for Multi-h CPFSK

In the preceding two sections we have examined the Viterbi Algorithm and the method of calculating the path metrics for the algorithm. In this section we expand on the above by investigating the complete Viterbi decoding process, as well as the physical structure required for the decoding of multi-h CPFSK.



calculates the branch metrics for the second bit period. These are added to the stored path metrics to produce the results shown in Figure 3-7.

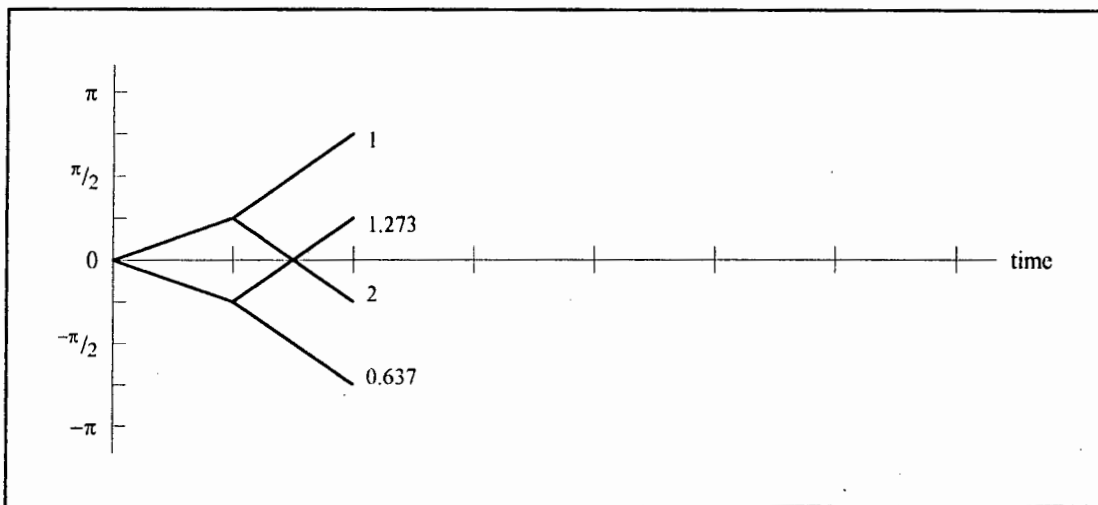


Figure 3-7: Step two of the Viterbi decoding process.

After the third bit period, merges occur in the phase trellis. At every node in the trellis where two paths meet, the respective path metrics are compared. The path with the larger path metric is the more likely of the two paths, and is therefore stored as the survivor path, while the other path is discarded. Figure 3-8 illustrates this concept. Discarded paths are shown in grey.

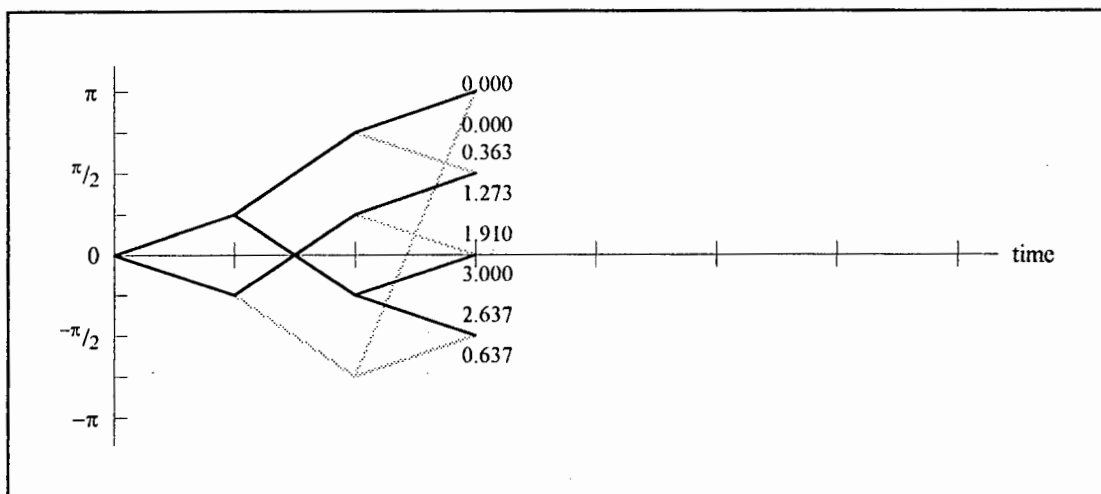


Figure 3-8: Step three of the Viterbi decoding process.

The remainder of the decoding process proceeds in a similar fashion, as shown in Figure 3-9. The single path extracted from the phase trellis diagram is the maximum likelihood transmitted path.

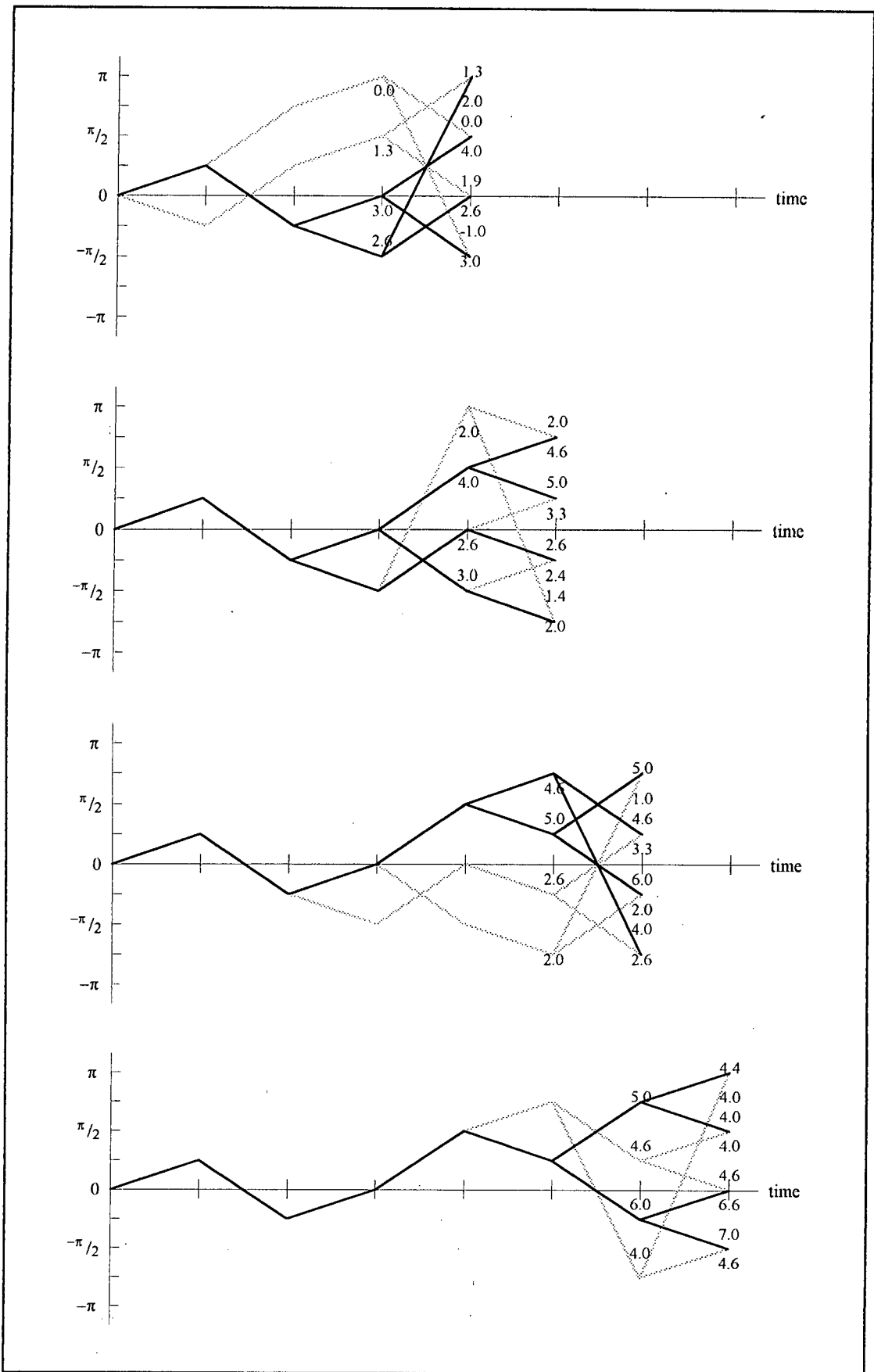


Figure 3-9: Step by step Viterbi decoding.

### 3.3.2 Calculation of branch metrics for multi-h CPFSK

The structure required for the calculation of branch metrics in multi-h CPFSK may be derived from equation (3.16). According to (3.16), the branch metric ( $BM$ ) for a particular trellis path  $s(t)$ , during bit period  $i$ , and a received signal  $x(t)$  is given by the correlation

$$BM = \int_{iT_b}^{(i+1)T_b} s(t)x(t)dt \quad (3.17)$$

Substituting for  $s(t)$ , this becomes

$$BM = \int_{iT_b}^{(i+1)T_b} \cos(2\pi ft + \phi)x(t)dt \quad (3.18)$$

where  $f$  is the frequency of the path and  $\phi$  is the path phase. For rational values of  $h_i$  ( $h_i = p_i/q$ ), the phase term,  $\phi$ , is restricted to the finite set of values

$$\phi \in \frac{n\pi}{q} \quad (3.19)$$

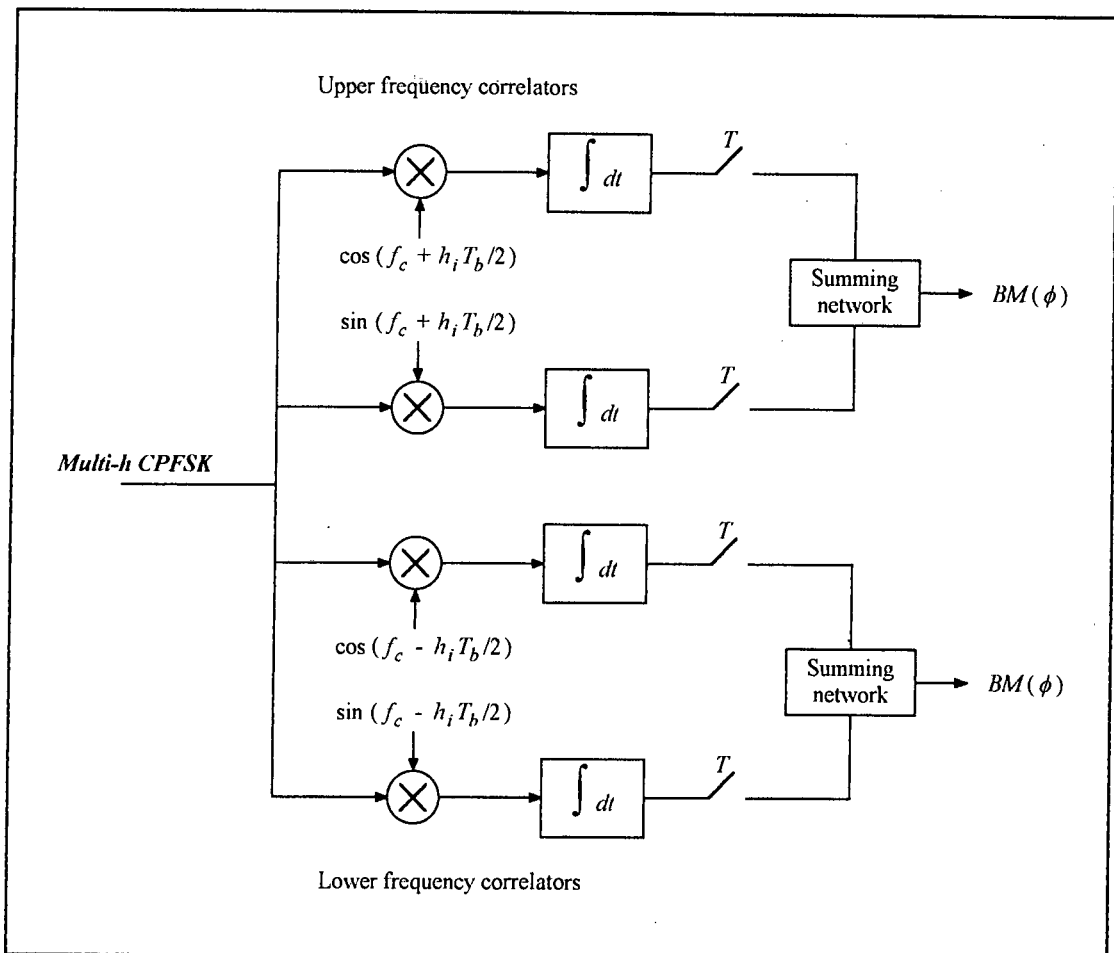
where  $n$  is an integer between  $-q+1$  and  $q$ . Equation (3.18) may be further simplified by means of the identity  $\cos(A+B) = \cos A \cos B - \sin A \sin B$ , to give

$$\begin{aligned} BM = & \cos(\phi) \int_{iT_b}^{(i+1)T_b} x(t) \cos(2\pi ft) dt \\ & - \sin(\phi) \int_{iT_b}^{(i+1)T_b} x(t) \sin(2\pi ft) dt \end{aligned} \quad (3.20)$$

We thus see that only two correlators (a  $\cos$  and a  $\sin$ ) are needed per transmit frequency in order to calculate all of the branch metrics for that frequency. If we consider that for each modulation index,  $h_i$ , we have two possible transmit frequencies, a high frequency and a low frequency, we can deduce that a bank of four correlators is required for each modulation index. The resultant physical structure of the correlation banks is depicted in Figure 3-10 overleaf.

## 3.4 Total Probability of Error for the Viterbi Decoder

For a large signal-to-noise ratio (SNR), the total probability of error for Viterbi decoding is dominated by the minimum squared euclidean distance,  $D_{min}^2$ , among all possible paths in the phase trellis which originate at a common node and end at a subsequent merge.



**Figure 3-10:** Correlator bank structure for the calculation of branch metrics. A separate correlator bank is required for each modulation index.

From (3.12) the probability of error,  $P_e$ , may therefore be approximated by

$$P_e \approx Q\left(\sqrt{\frac{D_{\min}^2}{2N_0}}\right) \quad (3.21)$$

Since the term  $D^2$  is dependent on the power of the received signal, this value is generally normalised according to the bit energy,  $E_b$ . The normalised squared euclidean distance,  $d^2$ , is defined by

$$d^2 = \frac{D^2}{2E_b} \quad (3.22)$$

The resultant probability of error may then be expressed as

$$P_e \approx Q\left(\sqrt{\frac{E_b \cdot d_{\min}^2}{N_0}}\right) \quad (3.23)$$

It should be noted that the signal-to-noise ratio ( $E_b/N_0$ ) required to give a particular  $P_e$  for CPFSK codes is generally referenced to the standard set by Minimum Shift Keying (MSK) which has a minimum squared euclidean distance of 2. Thus, any code with a  $d_{min}^2$  of greater than 2 will achieve improved power efficiency (or coding gain) over MSK. The coding gain (in dB) is given by

$$\text{coding gain} = 10 \log_{10} \left( \frac{d_{min}^2}{2} \right) \quad (3.24)$$

## Chapter 4

# A Modified PBIL Search for High Performance H-Sets

One of the main objectives of this project was to conduct a search for high performance multi-h CPFSK  $h$ -sets, and to measure the performance of these  $h$ -sets by means of a computer simulation of the Viterbi decoder. This chapter is devoted to the search process and to the results obtained. The work presented in this chapter is based on research undertaken by J. Cuthbert [4] in which a PBIL search algorithm was used to discover "good"  $h$ -sets. Unfortunately, an investigation into Cuthbert's results has shown that his measure of performance was inaccurate, and his documented  $h$ -sets are therefore useless. For this reason an attempt has been made to repeat/improve the search process and to carefully verify the results by means of the Viterbi decoding simulation.

The chapter thus begins with a discussion of the performance criteria and the desired properties for good  $h$ -sets. This is followed by a description of the PBIL search algorithm, and an explanation of modifications made to the algorithm in order to improve its search speed. Finally, the results of the search are presented for  $h$ -sets of size 2 to 8, and a comparison is made to Shannon's channel capacity theorem and to the optimal results found by Anderson, Aulin and Sundberg [2] for set sizes of 2, 3 and 4.

### 4.1 Performance Criteria and Desired Properties of $h$ -sets

The basic measure of performance for multi-h CPFSK  $h$ -sets is the probability of error for a given signal-to-noise ratio. The lower the probability of error, the higher the  $h$ -set performance. Since the probability of error is proportional to

$Q(\sqrt{d_{min}^2})$ , and the  $Q(x)$  function is a monotonically decreasing function of  $x$ , we see that the probability of error may be minimised by maximising  $d_{min}^2$ . Hence, the primary performance criterion for multi-h CPFSK  $h$ -sets is given by the minimum squared euclidean distance,  $d_{min}^2$ .

The desired properties of the  $h$ -sets are as follows:

- Modulation indices,  $h_i$ , confined to  $0 < h_i < 1$ . This ensures high bandwidth efficiency.
- Modulation indices rational, i.e.  $h_i = p_i/q$  ( $p_i$  and  $q$  integers) so as to make implementation possible.
- Small denominator,  $q$ , to allow ease of synchronisation and decoding.
- Small set size to keep modulator/demodulator complexity within manageable limits.

## 4.2 Calculation of $d_{min}^2$

The minimum squared euclidean distance,  $d_{min}^2$ , for a given  $h$ -set may be easily calculated by Viterbi decoding a random modulated bitstream (with no noise) and noting the differences in path metrics between the bitstream and all paths merging with it in the phase trellis. The validity of this method can be shown as follows:

Assume a transmitted signal  $s_t(t)$  corresponding to the random bitstream, and another signal  $s_a(t)$  corresponding to a path in the phase trellis which merges with the transmitted path. Both signals have a total energy of  $E$  joules and a bit energy,  $E_b$ , of 1 joule. Then, from equations (3.22) and (3.1), the squared euclidean distance,  $d^2$ , between the paths is given by

$$\begin{aligned}
 d^2 &= \frac{1}{2E_b} \int (s_t(t) - s_a(t))^2 dt \\
 &= \frac{1}{2} \int (s_t(t)^2 - 2s_t(t)s_a(t) + s_a(t)^2) dt \\
 &= \frac{1}{2} \left( 2E - 2 \int s_t(t)s_a(t) dt \right) \\
 &= E - \int s_t(t)s_a(t) dt
 \end{aligned} \tag{4.1}$$

Using the above result, the path metrics for the transmitted path,  $PM_{transmit}$ , and for the other path,  $PM_{another}$ , may be expressed as

$$PM_{transmit} = \int s_t(t)s_t(t) dt = E \tag{4.2}$$

and

$$PM_{another} = \int s_i(t)s_a(t)dt = E - d^2 \quad (4.3)$$

This gives a difference between the two path metrics of

$$PM_{transmit} - PM_{another} = E - (E - d^2) = d^2 \quad (4.4)$$

where  $d^2$  is the quantity to be calculated.

Thus, for a random modulated bitstream with bit energy,  $E_b$ , equal to 1 joule, the smallest value obtained for  $d^2$  by the above method is equal to the minimum squared euclidean distance for the particular  $h$ -set. Simulations have shown that this method converges quickly towards the absolute minimum value for  $d^2$ , and a bitstream of length 1000 bits is more than adequate for the purposes of obtaining a reliable result.

### 4.3 The PBIL Algorithm

The Population Based Incremental Learning (PBIL) Algorithm is an efficient numerical search algorithm used for approximating the global optimum of a multidimensional function. It is modelled, to some extent, on the principle of evolution observed in nature, in which a population of samples is improved over a number of generations by employing some sort of selection process. In PBIL, the selection process consists of finding the best sample in a random population, and learning from that sample so as to increase the probability of good samples being reproduced in the following generation.

Every sample in a PBIL search is made up of a number of parameters which govern the performance of that sample. Since it is important that no meaning be placed on the individual parameters of a sample, each sample is written as a string of binary digits (bits). Samples are generated in a random fashion by means of a probability vector, which gives the probability of each bit in the sample being a zero or a one.

The PBIL algorithm commences by initialising all values in the probability vector to 0.5 (equal probability of obtaining a zero or one), and using this vector to produce a set of samples (a population) for the first generation. The performance of each sample is evaluated by means of a fitness function and the best sample is found. This sample is then used to incrementally adjust the probability vector in such a way as to increase the probability of the best sample being regenerated in

the next generation. The process is repeated for a number of generations until the algorithm converges on a good solution. Random mutations are also introduced into the probability vector between generations in order to prevent the PBIL algorithm from focussing on local maxima. In this way the search is widened to cover as much of the search space as possible.

A flow chart of the PBIL algorithm is presented in Figure 4-1.

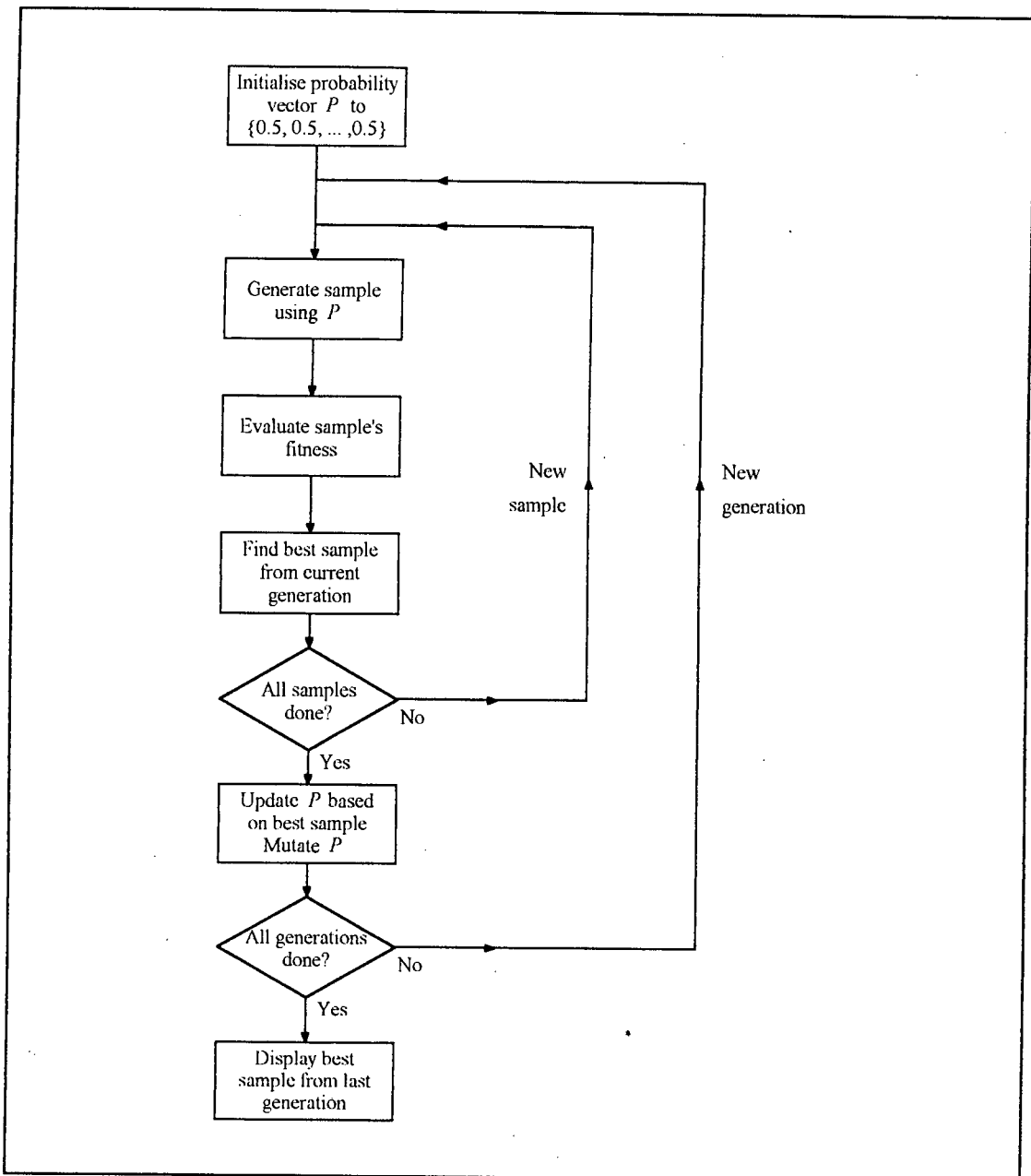


Figure 4-1: PBIL flowchart.

#### 4.4 PBIL and Multi-h CPFSK

The PBIL search parameters for multi-h CPFSK are the  $h$ -set numerators,  $p_i$ , for a particular denominator  $q$ . The fitness function is the minimum squared euclidean distance,  $d_{min}^2$ , where a high value of  $d_{min}^2$  indicates a good  $h$ -set.

Each PBIL sample is made up of the  $h$ -set numerators written one after the other in binary form. The number of bits representing each numerator is chosen according to the maximum value that can be used. For instance, if the denominator is 16, the desirable range for the numerator is from 1 to 15, and hence four bits per numerator are required. As an example case, consider the search for an  $h$ -set of size 2 with a denominator of 16. If the first random parameter set generated is  $\{^{13}/_{16}, ^7/_{16}\}$ , it will be represented by the initial vectors

vector	$h_1$				$h_2$			
sample	1	1	0	1	0	1	1	1
$P$	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

where  $P$  is the probability vector. The actual generation of the PBIL samples is carried out by producing a random number between 0 and 1 for each sample bit and comparing it to the corresponding probability vector element  $P_i$ . If the random number is greater than  $P_i$ , then the sample bit generated is a '1', otherwise it is a '0'.

At the end of each generation the probability vector is adjusted according to the best sample of that generation. The amount by which it is adjusted is given by the learning rate, and the direction of adjustment is down for a '1' and up for a '0'. This is so that the probability of reproducing the best sample in the next generation is increased. Cuthbert [4] suggests a learning rate of 0.05, which means that, assuming the above sample to be the best sample of the generation, the probability vector would be updated as

vector	$h_1$				$h_2$			
sample	1	1	0	1	0	1	1	1
$P$	0.45	0.45	0.55	0.45	0.55	0.45	0.45	0.45

Mutations are occasionally introduced into the probability vector at the end of a generation, based on a mutation probability. As with the probability vector, the mutation probability is a number between 0 and 1 (suggested value by Cuthbert is 0.02) which is compared to a random number between 0 and 1. If the random

number is smaller than the mutation probability, each element of the probability vector is adjusted in a random direction by an amount given by the mutation shift. Cuthbert suggests a mutation shift of 0.05.

## 4.5 Modifications to the PBIL Algorithm

In order to streamline the performance of the PBIL algorithm for multi-h, it is appropriate to make some modifications to the basic algorithm. These modifications are discussed below. A modified flowchart is given in Figure 4-2.

### *Representation of h-set numerators*

According to Anderson, Aulin and Sundberg [2], all good  $h$ -sets tend to have modulation indices which alternate around an average value of approximately 0.65. Should the  $h$ -set numerators be represented by their direct binary equivalents, this can present a problem for the search procedure.

Consider the case where a good  $h$ -set is found with a certain index equal to just below 0.5. Then the direct binary representation of this index would have a '0' in its most significant bit and a '1' in most of the other bits. However, it is highly probable that the optimum  $h$ -set has a value of just greater than 0.5 for the same index. This would be represented by a '1' in the most significant bit and a '0' in most of the other bits. Thus, in order for the search algorithm to move from its present state into the optimal state, it would need to adjust the binary representation in almost every bit simultaneously. If just the most significant bit were flipped, the index would become equal to approximately 1 and this would certainly not produce a good  $h$ -set. The chance of both the most significant bit and its neighbouring bits being flipped to produce a good  $h$ -set is rather small. Therefore, the algorithm would have a large probability of getting stuck in its sub-optimal state without being able to move across the 0.5 boundary.

For this reason it makes sense to change the representation of the  $h$ -set numerators so that most significant bit is a '1' over the entire range from 0.65-0.25 to 0.65+0.25. This can be achieved by making each binary number represent the numerator given by

$$\text{numerator} = \left( \text{binary number} + \frac{7q}{8} \right) \bmod q + 1 \quad (4.5)$$

where  $q$  is the  $h$ -set denominator.

As an example, the resultant binary representation for the numerator values,  $p_i$ , with a denominator,  $q$ , of 16 is presented in Table 4-1.

Numerator $p_i$	Binary representation	Index $h_i$
16	0001	1.00
15	0000	0.94
14	1111	0.88
13	1110	0.81
12	1101	0.75
11	1100	0.69
10	1011	0.63
9	1010	0.56
8	1001	0.50
7	1000	0.44
6	0111	0.38
5	0110	0.31
4	0101	0.25
3	0100	0.19
2	0011	0.13
1	0010	0.06

Table 4-1: Representation of  $h$ -set numerator values for a denominator of 16.

### **Updating of probability vector**

Instead of updating the probability vector based on the best sample in the current generation, it has been found by experiment that the search process proceeds more quickly when the update is based on the best sample out of all generations. The reason for this appears to be that the  $h$ -set search surface is generally a smooth, multi-peaked function where the peaks are of similar height (although there are many sharp dips in the surface due to catastrophic codes). Therefore, once a good  $h$ -set been identified, all that is necessary is for the search algorithm to narrow its search around that  $h$ -set in the hope of finding the corresponding peak, as the peak is probably near-optimal. This modification seems to eliminate unnecessary wandering around the search space in the early stages of the search.

### **Asymptotic learning rate**

In order to prevent the probability vector from converging too quickly on the current best  $h$ -set, it is useful to asymptotically decrease the PBIL learning rate as the probability vector moves away from 0.5 towards 0 or 1. The expression for the learning rate becomes

$$\text{asymptotic learning rate} = 2|1 - P_i - \text{sample bit}| \cdot \text{constant rate} \quad (4.6)$$

where the result is clipped to lie within the region  $0 < \text{result} < \text{constant rate}$ .

### Mutations

Due to the properties of the  $h$ -set search surface and the modified method for updating the probability vector, small random mutations seem to be of little value in widening the search area. Therefore, the random mutation function is discarded. However, without the mutations, as the search algorithm converges towards the

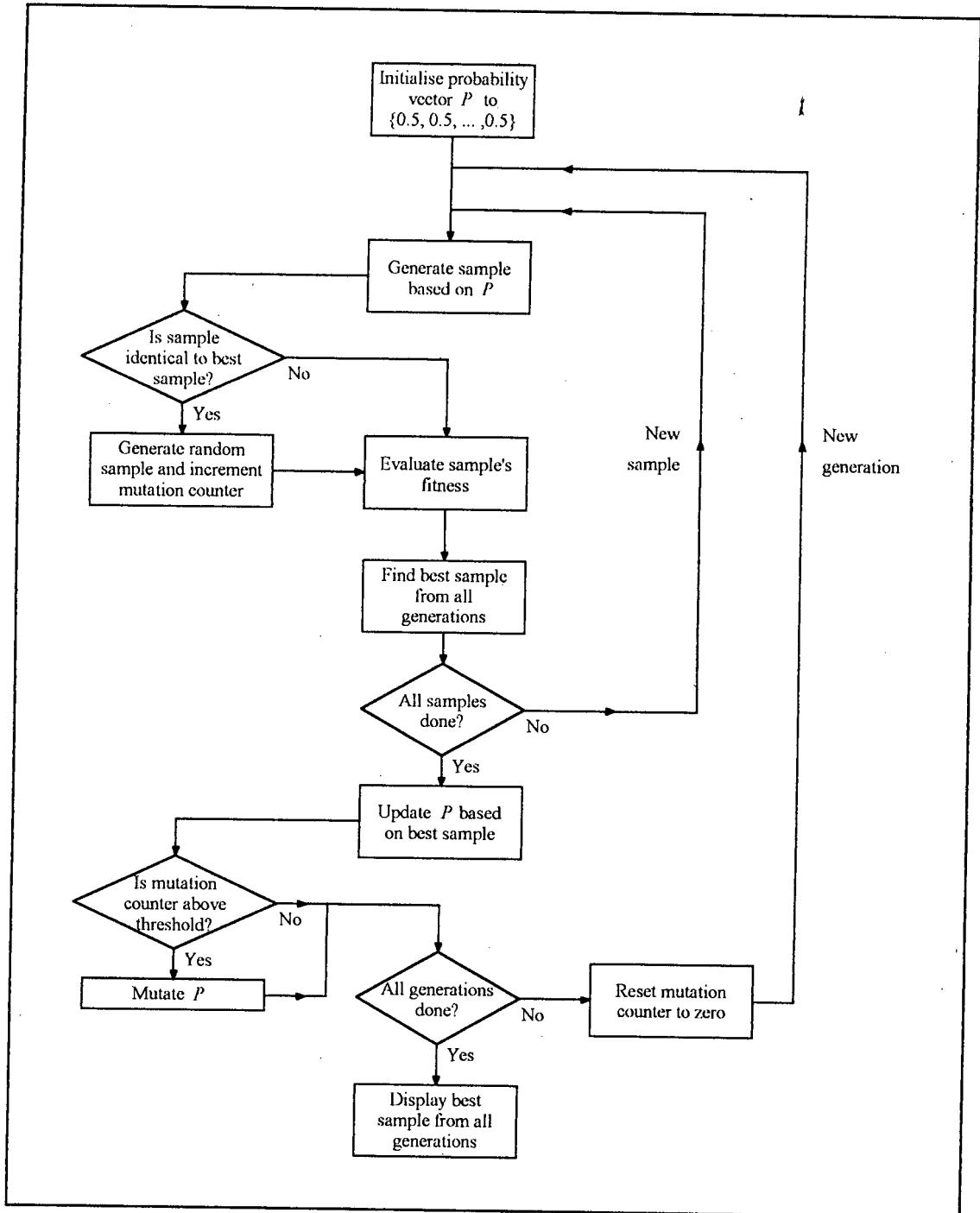


Figure 4-2: Modified PBIL flow chart.

current best  $h$ -set, more and more samples are generated which are identical to the best  $h$ -set.

In order to combat this problem, every time such a sample is generated, it is replaced by a completely random sample (produced by an all 0.5 probability vector) and a counter is incremented. If the counter is incremented more than a set number of times within a particular generation, then at the end of that generation a large mutation is forced on the probability vector. This system proves to be quite effective for a mutation shift of 0.25.

## 4.6 Modified PBIL Search Results

A 'C' implementation of the modified PBIL algorithm has been used to conduct a search for high-performance  $h$ -sets of size 2 to 8 with denominators,  $q$ , ranging between 8 and 256. A listing of the code is provided in Appendix D.

The results of the search are set out in Table 4-2. As a graphical aid the values obtained for  $d_{min}^2$  are presented in a 3D surface plot below.

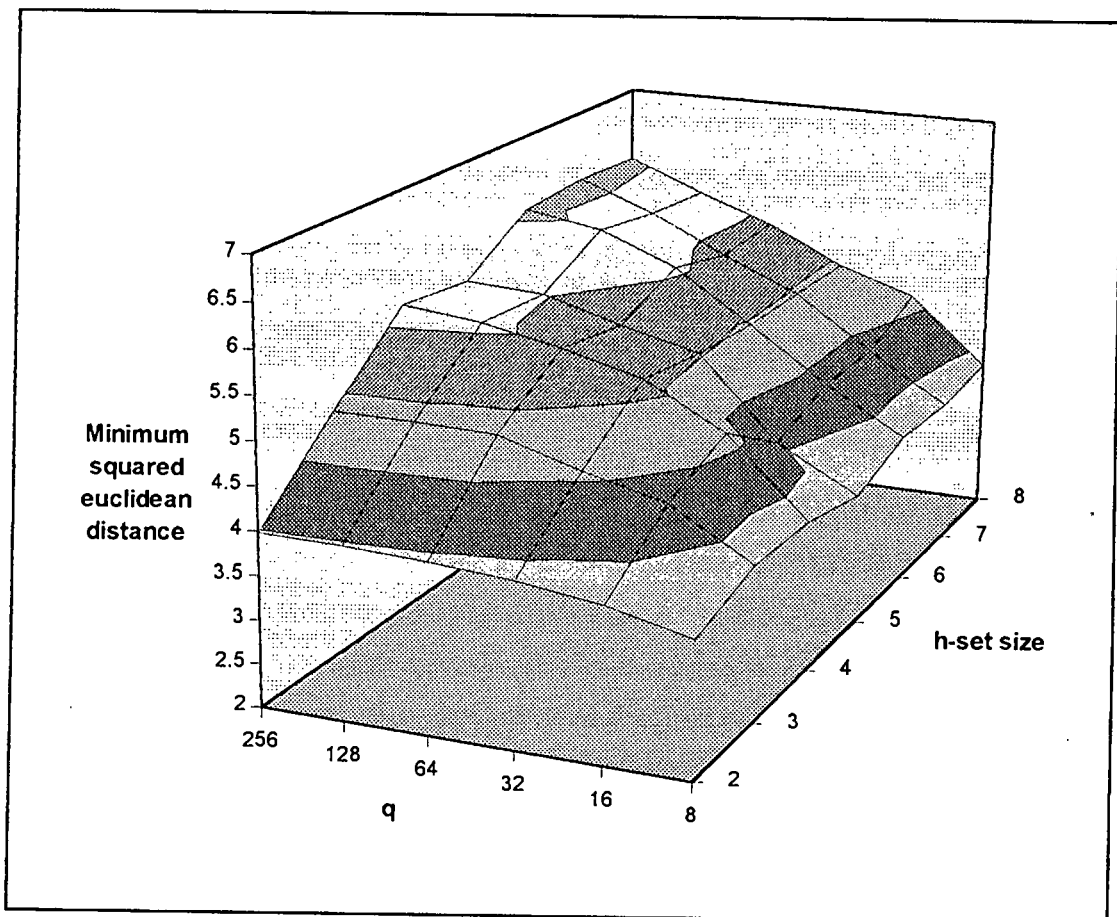
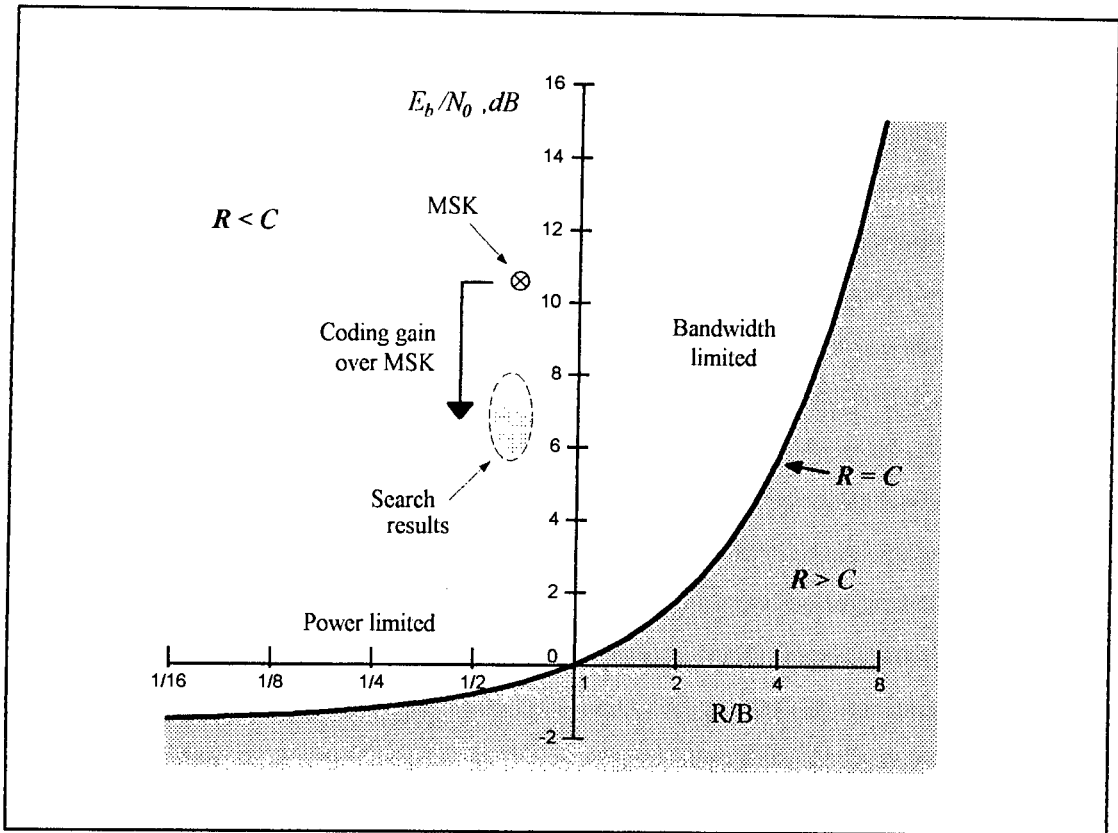


Figure 4-3: 3D surface plot of modified PBIL search results.

Set size	$q$	$p_i$	$d_{min}^2$	Gain over MSK in dB
2	8	5, 4	3.55	2.49
	16	9, 8	3.76	2.74
	32	17, 16	3.88	2.88
	64	33, 32	3.94	2.94
	128	65, 64	3.97	2.98
	256	129, 128	3.98	2.99
3	8	5, 4, 6	3.79	2.78
	16	11, 12, 10	4.34	3.36
	32	23, 22, 21	4.57	3.59
	64	44, 43, 45	4.85	3.85
	128	89, 91, 82	4.87	3.86
	256	163, 177, 183	4.87	3.86
4	8	4, 6, 4, 5	3.75	2.73
	16	10, 11, 8, 12	4.65	3.66
	32	24, 20, 23, 18	5.16	4.12
	64	37, 48, 39, 47	5.40	4.31
	128	72, 94, 74, 95	5.58	4.46
	256	187, 144, 188, 146	5.68	4.53
5	8	5, 4, 5, 4, 6	3.55	2.49
	16	10, 8, 10, 12, 9	4.07	3.09
	32	22, 20, 25, 16, 24	5.03	4.01
	64	44, 32, 48, 41, 50	5.26	4.20
	128	69, 99, 81, 100, 86	5.52	4.41
	256	200, 153, 195, 137, 171	5.57	4.45
6	8	6, 5, 4, 6, 5, 4	3.79	2.78
	16	12, 11, 10, 12, 11, 10	4.34	3.36
	32	26, 23, 18, 25, 22, 20	4.97	3.95
	64	33, 42, 51, 33, 50, 43	5.57	4.45
	128	102, 64, 100, 78, 97, 70	5.93	4.72
	256	152, 190, 155, 201, 162, 199	6.09	4.84
7	8	6, 4, 5, 4, 6, 5, 4	3.75	2.73
	16	12, 10, 11, 8, 12, 10, 11	4.34	3.36
	32	26, 23, 22, 20, 24, 16, 21	4.96	3.94
	64	42, 47, 36, 42, 48, 32, 52	5.35	4.27
	128	105, 89, 72, 99, 81, 102, 68	5.79	4.62
	256	153, 192, 142, 200, 169, 132, 193	6.14	4.87
8	8	4, 5, 6, 4, 5, 4, 6, 5	3.84	2.83
	16	10, 11, 8, 12, 10, 11, 8, 12	4.65	3.66
	32	23, 20, 22, 24, 17, 26, 22, 24	4.95	3.94
	64	37, 48, 39, 47, 37, 48, 39, 47	5.40	4.31
	128	84, 97, 81, 101, 69, 107, 81, 103	5.73	4.57
	256	132, 194, 166, 207, 180, 147, 200, 169	6.09	4.84

Table 4-2: Modified PBIL search results.

The significance of the results with respect to Shannon's channel capacity theorem is illustrated in Figure 4-4.



**Figure 4-4:** Modified PBIL search results with respect to the channel capacity boundary.  $R$  is the bit rate,  $C$  is the channel capacity,  $B$  is the bandwidth.

An examination of the above figure reveals that the multi- $h$  CPFSK codes found using the modified PBIL search lie about 35% closer to the channel capacity boundary than MSK. This represents a significant step forward from the already "efficient" MSK scheme. However, it should also be noted from Figure 4-3 that, although the performance of the  $h$ -sets improves consistently with  $h$ -set size and denominator, the law of diminishing returns applies. Therefore, the existence of multi- $h$  CPFSK codes with performance characteristics much better than those presented in Table 4-2 is unlikely.

The optimality of the results in Table 4-2 may be examined by means of a comparison with the theoretical limits obtained by Anderson, Aulin and Sundberg [2] after exhaustive search. The limiting cases are listed in Table 4-3.

Table 4-4 compares these performance limits to the modified PBIL search results for  $h$ -set sizes 2,3 and 4 with a denominator of 256. Since the search results approximate the theoretical maxima to within 0.5% for set sizes 2,3 and 4, it is

$h$ -set size	$h_i$	$d_{min}^2$	Gain over MSK in $dB$
1	0.715	2.43	0.85
2	0.5, 0.5	4.00	3.01
3	0.620, 0.686, 0.714	4.88	3.87
4	0.73, 0.55, 0.73, 0.55	5.69	4.54

**Table 4-3:** Theoretical performance limits for multi-h CPFSK obtained by [2].

reasonable to assume that the results for the larger  $h$ -sets are also close to their respective (and as yet unknown) maxima.

$h$ -set size	PBIL results		Theoretical maxima	
	$h_i$	Gain over MSK in $dB$	$h_i$	Gain over MSK in $dB$
2	0.50, 0.50	2.99	0.50, 0.50	3.01
3	0.64, 0.69, 0.71	3.86	0.62, 0.69, 0.71	3.87
4	0.73, 0.56, 0.73, 0.57	4.53	0.73, 0.55, 0.73, 0.55	4.54

**Table 4-4:** Comparison between modified PBIL results and theoretical maxima.

## Chapter 5

# Measurement of BER Curves for Selected Multi-h Codes

In this chapter the coding gain results of Chapter 4 are verified by means of a computer simulation of the Viterbi Decoder. The verification is performed by employing the Viterbi Algorithm to measure the bit error rate (BER) curves for selected  $h$ -sets, and then comparing these curves to the theoretical probabilities of error for the respective codes.

The  $h$ -sets used in the simulation are those sets from Table 4-2 deemed to be the most promising for practical implementation. According to the desired properties of  $h$ -sets laid down in Section 4.1, such sets have a high  $d_{min}^2$ , a small denominator,  $q$ , and a small set size. Two  $h$ -sets fitting these requirements, which immediately stand out from the 3D surface plot in Figure 4-3, are those with a set size of 4, denominator 32, and with a set size of 6, denominator 64. Hence, these are the codes examined.

The chapter begins with a discussion of the software implementation of the Viterbi Algorithm. This is followed by the simulation results for the two selected  $h$ -sets.

### 5.1 Software Implementation of the Viterbi Algorithm

There are two important considerations in the practical implementation of the Viterbi Algorithm, namely the storage of survivor paths and their associated path metrics, and the truncation depth for the stored paths. It is obvious from the discussion of the Viterbi decoding process in Section 3.3 that a complete decoder requires a path memory ranging all the way from time  $t = 0$  up until the current bit

period. This is because the Viterbi survivor paths may extend backward by an infinite number of bit periods without merging into a single maximum-likelihood path. However, this is not a practical requirement for a real-world implementation of the decoder. For this reason, the survivor paths are usually truncated to a finite depth. This means that each time a bit is dropped off the end of the survivor paths, a bit must be output by the decoder. The decision as to which bit should be output is made by choosing the bit from the survivor path with the largest path metric.

### 5.1.1 Choice of truncation depth

The minimum truncation depth required to ensure zero loss of performance is dependent on the distance properties of the  $h$ -set being decoded. The condition for optimal decoding performance is that no path which splits from the transmitted path at the truncation depth, and terminates on any node in the phase trellis at the current bit period, must be allowed to have a squared euclidean distance of less than  $d_{min}^2$  from the transmitted path.

With this in mind, the worst case scenario for the selected multi-h  $h$ -sets occurs for a transmitted bit sequence of '1?0?? ...', where '?' represents either a '1' or a '0', and another path in the phase trellis corresponding to '0s1sss ...', where each 's' is the same as the respective '?'. Figure 5-1 illustrates the build-up of distance between two such paths.

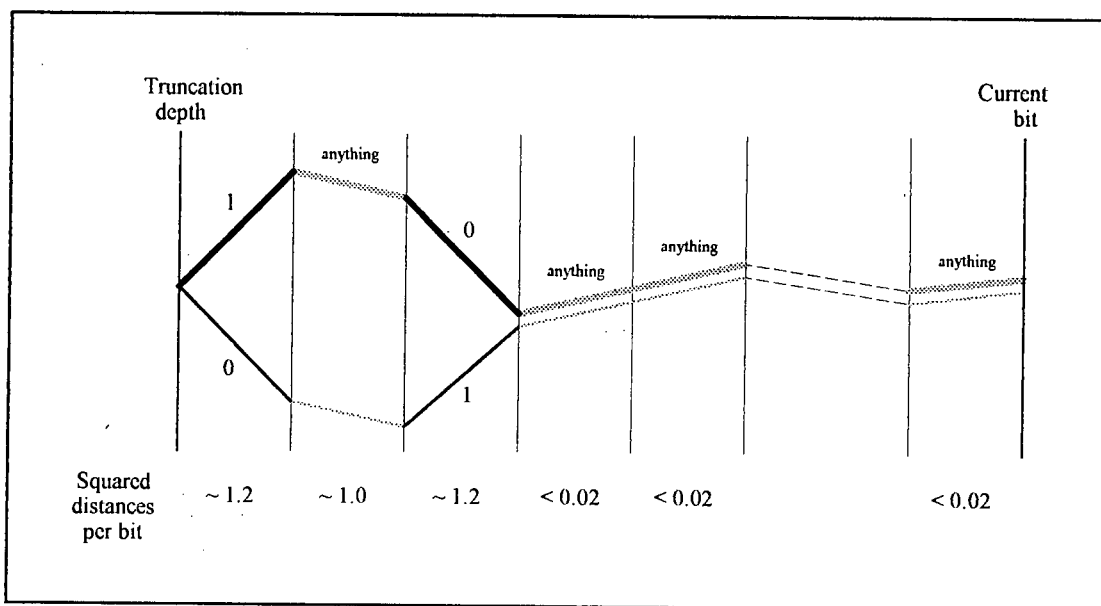


Figure 5-1: Build up of distance between paths over the truncation depth (worst case).

In order for the total squared euclidean distance between the paths to be greater than the minimum squared euclidean distance for the respective  $h$ -sets, the

following truncation depths are required:

<i>h</i> -set size	Truncation depth
4	100 bits
6	500 bits

### 5.1.2 Memory structure for path storage

The memory structure used by the computer simulation for the storage of survivor trellis paths consists of an array of data structures, one for each of the possible vertical nodes in the phase trellis diagram. The data structures contain storage space for an "occupied" flag (to indicate whether a path terminates on that particular node or not), for the path metric and for the associated path. Each path is represented by its corresponding bitstream.

The paths are entered into the path memory in a cyclic fashion so as to avoid the need for shifting all the existing paths backward by one bit period every time the decoder advances to a new bit. A pointer is thus required to keep track of the current bit in the memory space. The memory structure is illustrated in Figure 5-2.

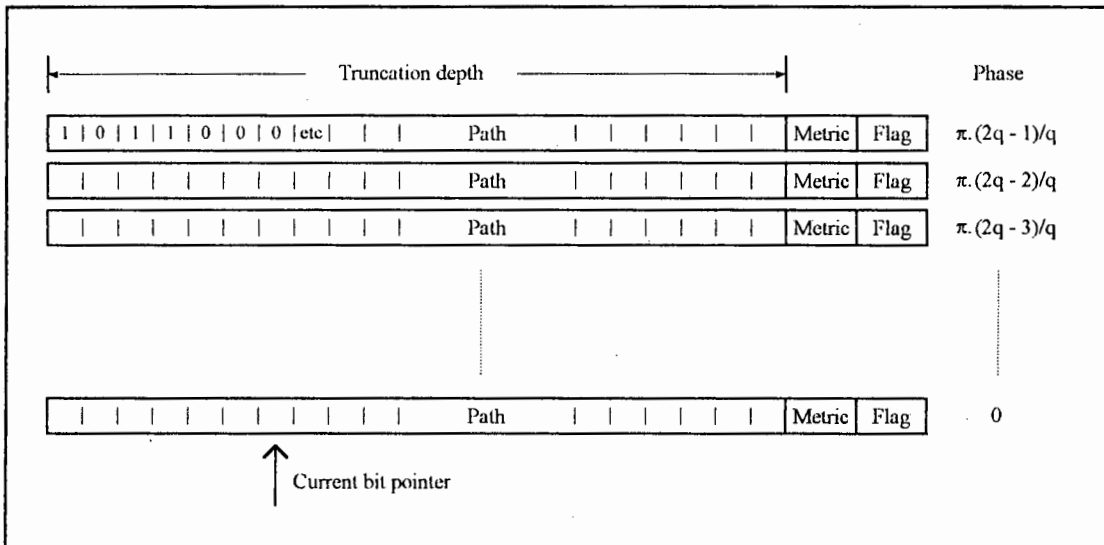


Figure 5-2: Memory structure for simulation path storage.

A simple way in which to visualise the use of the memory structure in Figure 5.2 is to think of it as being a window of data sliding over the phase trellis diagram, representing the corresponding phase trellis paths inside that window. At the end of every bit period, the window is advanced by one bit period and the new branch



## 5.2 Simulation Results

The Viterbi decoding simulation is based on the description of the Viterbi decoder given in Section 3.3 and on the above discussion. A 'C' code listing for the simulation software is provided in Appendix E.

The simulation results are plotted in Figures 5-4 and 5-5, along with the theoretical BER curves for the respective  $h$ -sets and for MSK. Due to project time limitations, only bit error rates of greater than  $10^{-4}$  have been considered.

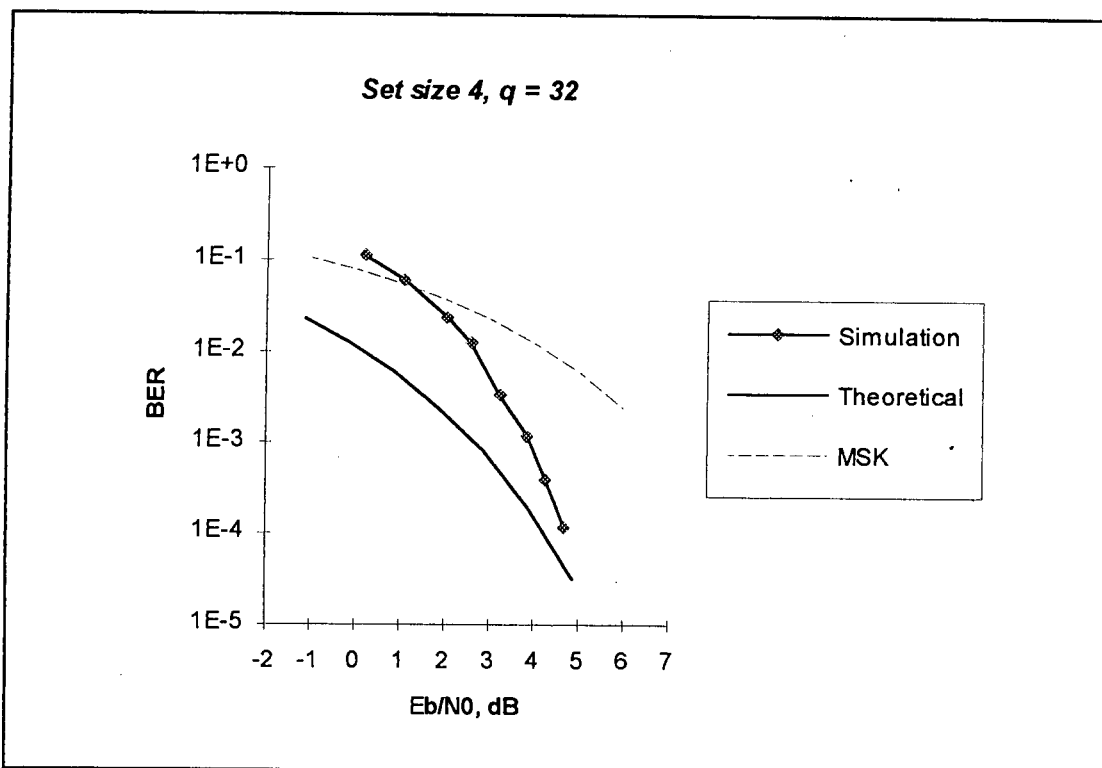
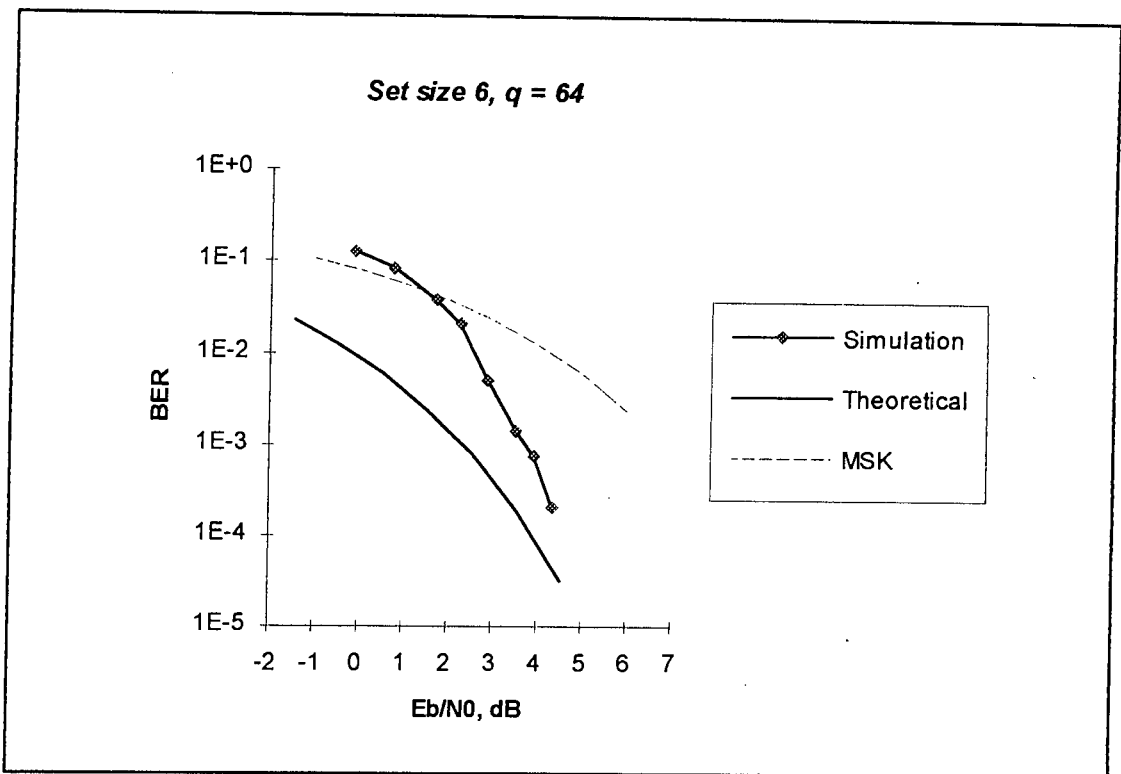


Figure 5-4: BER curve for a set size of 4, denominator 32.

An examination of the graphs in Figures 5-4 and 5-5 reveals that the simulation BER curves approach the theoretical results asymptotically as the signal-to-noise ratio increases. This is as predicted in Section 3.4. Thus, at high signal-to-noise ratios, the theoretical coding gain of  $10\log(d_{min}^2/2)$  dB over MSK is achieved. However, at low signal-to-noise ratios the performance of the multi- $h$  signal is worse than that for MSK.

The reason for the large discrepancy between the simulation and theoretical results at low signal-to-noise ratios is due to two main factors. The first is that the probability of error formula for multi- $h$  CPFSK gives the probability of choosing an incorrect *path* in the phase trellis diagram, not an incorrect *bit*. Since an



**Figure 5-5:** BER curve for a set size of 6, denominator 64.

incorrect path consists of many incorrect bits, the bit error rate is higher than the value given by  $P_e$ . The second reason is that at low signal-to-noise ratios there are many incorrect paths which have a high probability of being mistaken for the transmitted path. This is in contrast to the situation at high signal-to-noise ratios where the probability of path error is dominated by just one incorrect path. Once again, this means that the actual probability of error is higher than the value given by  $P_e$ .

## **Chapter 6**

# **Sequential Decoding of Multi-h Signals**

A final topic of relevance to this dissertation is that of sequential decoding of multi-h CPFSK signals. The sequential algorithms are of interest due to the fact that they are far less computationally intensive than the Viterbi Decoder, and are therefore more suited to high-speed implementations [4]. However, it turns out that sequential decoders are extremely sub-optimal for high-performance multi-h CPFSK codes. This chapter discusses the performance issues involved.

The chapter thus begins with a general description of sequential decoding algorithms. This is followed by an investigation into their error performance characteristics.

### **6.1 General Description of Sequential Algorithms**

Sequential algorithms differ from the Viterbi Algorithm in that only a single path in the phase trellis is considered at a time. For Viterbi decoding all paths are considered simultaneously. The initial path examined by the sequential decoder is chosen in an intelligent fashion by means of a sub-optimal estimation of the transmitted path, for example by taking the output of a simple non-coherent detector. If it so happens that the path being pursued by the decoder builds up an unlikely amount of distance between itself and the received signal, the receiver back-tracks to an earlier split in the phase trellis and tries another path. In this way the decoder searches through all of the obviously promising paths in order to find one which satisfies its performance criteria. This resulting path is given as the decoder output.

## 6.2 Error Performance of Sequential Decoders

The probability of error associated with sequential decoders is dominated by the same two paths which influence the choice of truncation depth for Viterbi Decoding. These paths are shown in Figure 6-1.

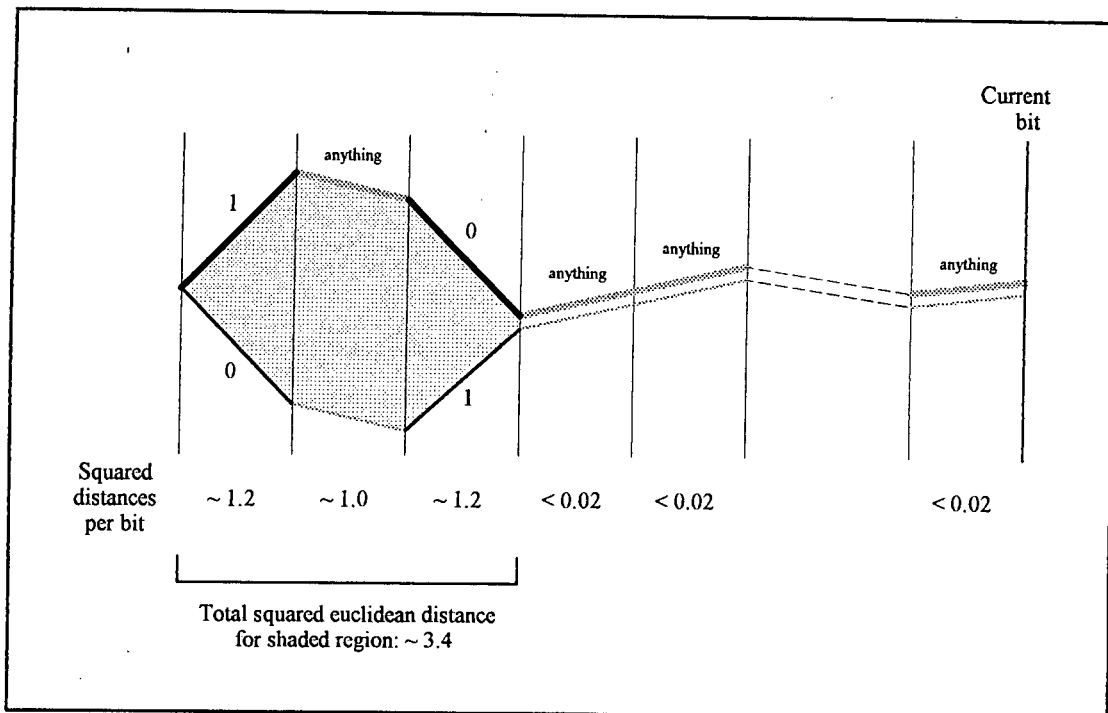


Figure 6-1: Paths dominating the probability of error for sequential decoding.

An examination of Figure 6-1 reveals that the majority of squared euclidean distance between the paths is accumulated during the first three bit periods following the split. This distance is indicated by the shaded region in the diagram. After the third bit period, however, the rate of distance build-up is minimal. Therefore, supposing that the transmitted path in the figure is the upper path (shown in bold), and that the path pursued by the decoder is the incorrect lower path, once the decoder advances past the third bit period, the difference between the distance build-up rates (from the received signal) for the correct and incorrect paths is almost identical. This means that, after the third bit period, it becomes practically impossible to identify the incorrect path as being an incorrect path under noisy conditions. The result is that the probability of error becomes dependent only on that squared euclidean distance which is accumulated during the first three bit periods. This distance is approximately equal to 3.4 regardless of the  $h$ -set used.

Thus, from equation (3.24), the upper bound for the coding gain over MSK for sequential decoders may be calculated to be approximately 2.3 dB. This is

substantially lower than for optimal decoding. Furthermore, the actual gains achievable are even less than this due to the fact that the decoding process itself is sub-optimal.

Therefore it may be concluded that, although the computational requirements for sequential decoders are less demanding than for optimal decoders, the sub-optimal performance of sequential decoding algorithms renders them unsuitable for use with multi-h CPFSK.

## Chapter 7

# Summary and Recommendations

The main aim of this dissertation has been to discover the theoretical performance capabilities of multi- $h$  CPFSK digital modulation, viewed as a coded modulation scheme. Such an investigation is of interest to the communications world due to the high bandwidth efficiency and good envelope properties displayed by multi- $h$  CPFSK signals, which make them extremely attractive for use in satellite communication channels.

The above goal has been achieved by means of a detailed examination of the multi- $h$  CPFSK signalling format, followed by an in-depth look at optimal detection and decoding methods. Using the knowledge obtained, the efficient PBIL search algorithm has been modified and employed to search for near-optimal high performance  $h$ -sets of sizes 2 to 8. The validity of the PBIL results has been proven by comparing them to the theoretical performance limits found by [2], and by measurement of BER curves using a computer simulation of the Viterbi decoder. Finally, the subject of sequential decoding has been addressed, with the conclusion being that sequential decoding of multi- $h$  CPFSK is extremely sub-optimal.

The results of this dissertation are best summarised by Figure 4-3 and Table 4-2 (pages 39 and 40), where the  $h$ -sets discovered using the modified PBIL search algorithm, and their respective coding gains over MSK, are clearly displayed. These results show that for well-chosen  $h$ -sets it is theoretically possible to achieve coding gains over MSK of 4.5 dB and greater. It should be noted, however, that the claimed coding gains are only applicable for high signal-to-noise ratios. At low signal-to-noise ratios (below about 5 dB), the performance of multi- $h$  CPFSK is severely degraded to the point of eventually becoming worse than that for MSK (at a SNR of approximately 1.5 dB).

The significance of the search results with respect to Shannon's channel capacity theorem is illustrated by the graph in Figure 4-4. Basically, this shows that the multi-h CPFSK codes lie approximately 35% closer to the channel capacity boundary than MSK. This is a large step forward from MSK. However, due to the fact that increased code complexity from the larger set sizes appears to produce very little increase in performance, it is unlikely that much further performance gain over MSK can be achieved using multi-h CPFSK digital modulation.

Some of the other issues addressed in this thesis are those relating to the implementation of the Viterbi decoder, namely the correlator receiver structure, the calculation of branch metrics, choice of truncation depth and memory management issues. However, it must be stressed that the purpose of this dissertation has not been to investigate the practical issues of multi-h CPFSK implementation. Rather, it has simply been to provide a solid theoretical basis on which additional research and development can be grounded.

Recommendations for future investigation are thus

- Firstly, to consider various synchronisation methods for extracting carrier timing, bit timing and superbaud timing from the multi-h CPFSK signals. It is of great importance that the phase locking accuracies of the synchronisation loops be extremely tight as the performance of the decoder is very sensitive to phase error.
- Should it be found that adequate synchronisation techniques exist, to investigate in detail the practical implementation of the Viterbi Decoder for multi-h CPFSK. The investigation should include calculations of the expected performance results for the Viterbi Decoder given the quality of the synchronisation structures involved.
- To build a physical simulation test-bed in order to determine the real-world performance of multi-h CPFSK and the associated implementation losses.
- To investigate the effects of real-world channels on multi-h CPFSK systems.

## Appendix A

# Gaussian Probability Functions

The standard gaussian distribution function with zero mean and unit variance is

$$p(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \quad (\text{A.1})$$

The probability that the gaussian random variable  $X$  is greater than a given value  $x$  is given by the function  $Q(x)$ , defined by

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^{\infty} e^{-z^2/2} dz \quad (\text{A.2})$$

Tabulated values of equation (A.2) are available in most communications textbooks. The reader is referred to [7] for such a table. For large values of  $x$ ,  $Q(x)$  can be approximated by

$$Q(x) \approx \frac{1}{x\sqrt{2\pi}} \left(1 - \frac{1}{x^2}\right) e^{-x^2/2} \quad (\text{A.3})$$

According to [7], the error in this approximation is about -2% for  $x = 3$ , -1% for  $x = 4$ , and the approximation becomes increasingly better for larger values of  $x$ .

## Appendix B

# Derivation of Probability of Error for the Correlator Receiver

We begin by considering two possible transmitted signals,  $s_1(t)$  and  $s_2(t)$ , both with an energy of  $E$  joules. The signal  $s_1(t)$  is transmitted through an AWGN channel, and a noise component,  $n(t)$ , with a double-sided power spectral density of  $N_0/2$  W/Hz is added to it. The received signal,  $x(t)$ , is thus equal to

$$x(t) = s_1(t) + n(t) \quad (\text{B.1})$$

At the receiver,  $x(t)$  is correlated with each of  $s_1(t)$  and  $s_2(t)$ . The signal which produces the largest correlation output is chosen as the maximum likelihood transmitted signal. An error therefore occurs if the correlation for  $s_2(t)$  is larger than that for  $s_1(t)$ . The receiver structure is illustrated in Figure B-1.

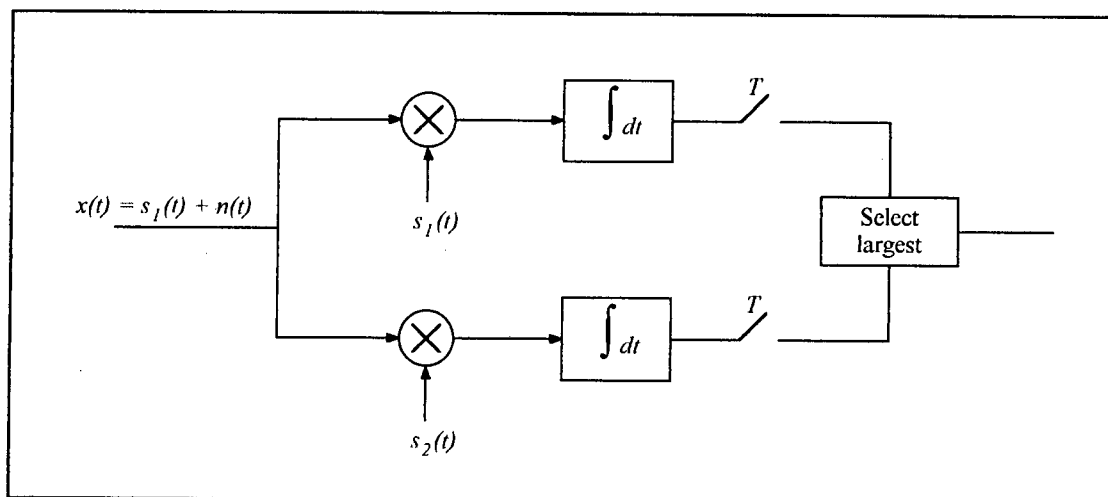


Figure B-1: Correlator receiver structure.

We then define the squared euclidean distance,  $D^2$ , between  $s_1(t)$  and  $s_2(t)$ . This is given by

$$\begin{aligned}
 D^2 &= \int (s_1(t) - s_2(t))^2 dt \\
 &= \int (s_1(t)^2 - 2s_1(t)s_2(t) + s_2(t)^2) dt \\
 &= 2E - 2 \int s_1(t)s_2(t) dt
 \end{aligned} \tag{B.2}$$

Thus the correlator outputs for the upper and lower arms of the receiver can be expressed (from Figure B-1 and equation (B.2)) as

$$\begin{aligned}
 \text{upper arm output} &= \int (s_1(t) + n(t))s_1(t) dt \\
 &= \int s_1(t)s_1(t) dt + \int n(t)s_1(t) dt \\
 &= E + \int n(t)s_1(t) dt \\
 &= E + N_{upper}
 \end{aligned} \tag{B.3}$$

$$\begin{aligned}
 \text{lower arm output} &= \int (s_1(t) + n(t))s_2(t) dt \\
 &= \int s_1(t)s_2(t) dt + \int n(t)s_2(t) dt \\
 &= \left(E - \frac{D^2}{2}\right) + \int n(t)s_2(t) dt \\
 &= \left(E - \frac{D^2}{2}\right) + N_{lower}
 \end{aligned} \tag{B.4}$$

where  $N_{upper}$  and  $N_{lower}$  are gaussian random variables with variances  $\sigma_{upper}^2$  and  $\sigma_{lower}^2$  equal to

$$\sigma_{upper}^2 = \sigma_{lower}^2 = \frac{EN_0}{2} \tag{B.5}$$

Returning to the condition for obtaining an error, we see from equations (B.3) and (B.4) that, in order for the lower arm output to be larger than the upper arm output (condition for error), we require

$$\left(E - \frac{D^2}{2}\right) + N_{lower} > E + N_{upper} \tag{B.6}$$

which gives

$$N_{lower} - N_{upper} > \frac{D^2}{2} \tag{B.7}$$

It is extremely important to note that  $N_{lower}$  in equation (B.7) has a common part with  $N_{upper}$ , dependent on the correlation between  $s_1(t)$  and  $s_2(t)$ . Using the simplified notation for the correlation integral,  $S_1 S_2 \equiv \int s_1(t) s_2(t) dt$ , the common part,  $N_{common}$  is given by

$$N_{common} = \frac{S_1 S_2}{E} N_{upper} \equiv \frac{\int s_1(t) s_2(t) dt}{E} N_{upper} \quad (\text{B.8})$$

with a corresponding variance,  $\sigma_{common}^2$ , of

$$\begin{aligned} \sigma_{common}^2 &= \left( \frac{S_1 S_2}{E} \right)^2 \sigma_{upper}^2 \\ &= \left( \frac{S_1 S_2}{E} \right)^2 \frac{E N_0}{2} \\ &= \frac{(S_1 S_2)^2 N_0}{2E} \end{aligned} \quad (\text{B.9})$$

This means that the left-over, independent part of  $N_{lower}$ , denoted by  $N_{independent}$ , has variance,  $\sigma_{independent}^2$  equal to

$$\begin{aligned} \sigma_{independent}^2 &= \sigma_{lower}^2 - \sigma_{common}^2 \\ &= \frac{E N_0}{2} - \frac{(S_1 S_2)^2 N_0}{2E} \\ &= \frac{(E^2 - (S_1 S_2)^2) N_0}{2E} \end{aligned} \quad (\text{B.10})$$

Hence, if we subtract  $N_{common}$  from  $N_{lower}$  and  $N_{upper}$  in equation (B.7), we obtain for the error condition

$$N_{independent} - \left( N_{upper} - N_{common} \right) > \frac{D^2}{2} \quad (\text{B.11})$$

Substituting for  $N_{common}$  from equation (B.8), this reduces to

$$\begin{aligned} N_{independent} - \left( N_{upper} - \frac{S_1 S_2}{E} N_{upper} \right) &> \frac{D^2}{2} \\ N_{independent} - \left( \frac{E - S_1 S_2}{E} N_{upper} \right) &> \frac{D^2}{2} \end{aligned} \quad (\text{B.12})$$

A further simplification can be achieved by use of equation (B.2) to give

$$\text{Condition for error: } N_{\text{independent}} - \frac{D^2}{2E} N_{\text{upper}} > \frac{D^2}{2} \quad (\text{B.13})$$

Finally, in order to calculate the probability of error, we need to find the total variance,  $\sigma_{\text{total}}^2$ , for the left-hand side of (B.13). This is equal to (from equations (B.10) and (B.5))

$$\begin{aligned} \sigma_{\text{total}}^2 &= \frac{(E^2 - (S_1 S_2)^2) N_0}{2E} + \left(\frac{D^2}{2E}\right)^2 \frac{E N_0}{2} \\ &= \frac{(E^2 - (E - D^2/2)^2) N_0}{2E} + \frac{D^4}{4E^2} \frac{E N_0}{2} \\ &= \frac{(E D^2 - D^4/4) N_0}{2E} + \frac{D^4 N_0}{8E} \\ &= \frac{D^2 N_0}{2} - \frac{D^4 N_0}{8E} + \frac{D^4 N_0}{8E} \\ &= \frac{D^2 N_0}{2} \end{aligned} \quad (\text{B.14})$$

Thus the probability of error for the correlator receiver,  $P_e$ , is given (from equations (B.13) and (B.14)) by

$$\begin{aligned} P_e &= Q\left(\frac{D^2/2}{\sigma_{\text{total}}}\right) \\ &= Q\left(\frac{D^2/2}{\sqrt{D^2 N_0/2}}\right) \\ &= Q\left(\sqrt{\frac{D^2}{2N_0}}\right) \end{aligned} \quad (\text{B.15})$$

where the  $Q(x)$  function is as defined in Appendix A.

## Appendix C

# PN Sequence Generator for Search and Simulation Software

The pseudo-noise generator used by the search and simulation software to produce long random bit sequences is a 20-stage maximal-length sequence generator with feedback taps [3,20] (obtained from Dixon [5]). The structure of the generator is shown in Figure C-1.

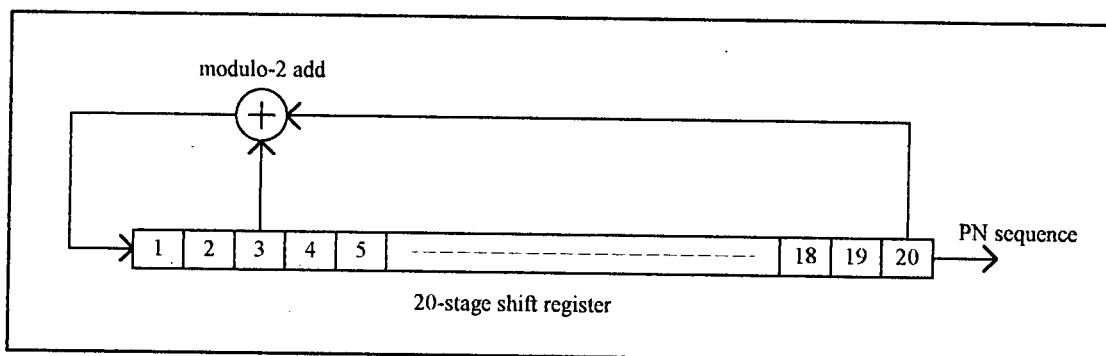


Figure C-1: PN sequence generator for search and simulation software.

The period of the generator is given by

$$\text{Period} = 2^{20} - 1 = 1048575 \quad (\text{C.1})$$

This period has been checked by means of a software implementation of the generator (which loads the shift register with all '1's and then clocks the device until the all '1' pattern re-appears). The period has been found to be correct at 1048575.

## Appendix D

# 'C' Code Listing for Modified PBIL Search Software

```
/*
*****
/* Modified PBIL search algorithm to find good symetrical h-sets. The
/* fitness function for the search algorithm is the minimum squared
/* euclidean distance of the h-set.
/*
/* d^2(min) is calculated for each h-set by Viterbi decoding a modulated
/* random bit sequence, and recording the minimum accumulated squared
/* distance for all paths in the phase trellis which merge with the
/* transmitted path.
/*
*****
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

/*
*****
/* Data structure definition for struct NODE. This structure holds
/* information for a node in the phase trellis diagram.
*****
typedef struct node
{
    int occupied; /* Flag set if node is occupied */
    double metric; /* Path metric for node */
} NODE;

/*
***** Seed shift register *****
/* Function to produce a random seed for the PN generator shift register.
/*
/* int *sreg: pointer to shift register
*****
```

```

void seedsreg (int *sreg)
{
    int counter;

    randomize();

    for (counter = 0;counter < 20;counter++)
        sreg[counter] = random(2);

    sreg[0] = 1;
}

/***** Bit generator *****/
/* Function to produce a random bit using the PN generator. Function */
/* returns the random bit and advances the shift register. The shift */
/* register is a 20 stage register with feedback taps 20 and 3. It */
/* produces a pseudonoise sequence of period 1048575. */
/* */
/* int *sreg: pointer to shift register */
/*****

int bitgen (int *sreg)
{
    int bit;                /* Random bit */
    int temp;

    bit = sreg[19];

    temp = (sreg[19]+sreg[2]) % 2;

    sreg[19] = sreg[18];
    sreg[18] = sreg[17];
    sreg[17] = sreg[16];
    sreg[16] = sreg[15];
    sreg[15] = sreg[14];
    sreg[14] = sreg[13];
    sreg[13] = sreg[12];
    sreg[12] = sreg[11];
    sreg[11] = sreg[10];
    sreg[10] = sreg[9];
    sreg[9] = sreg[8];
    sreg[8] = sreg[7];
    sreg[7] = sreg[6];
    sreg[6] = sreg[5];
    sreg[5] = sreg[4];
    sreg[4] = sreg[3];
    sreg[3] = sreg[2];
    sreg[2] = sreg[1];
    sreg[1] = sreg[0];
    sreg[0] = temp;

    return (bit);
}

/***** Initialise node *****/
/* Function to initialise the array of nodes. All node metrics are set */
/* to zero. All occupied flags are set to zero except for node zero. */
/* */
/* NODE *node: pointer to array of nodes */
/* int hd: denominator of h-set */
/*****

void initnode (NODE *node,int hd)

```

```

{
    int counter;

    for (counter = 0; counter < 2*hd; counter++)
    {
        node[counter].occupied = 0;
        node[counter].metric = 0;
    }
    node[0].occupied = 1;
}

/***** Correlate *****/
/* Function to calculate the correlations (over a single bit period) */
/* between the transmitted path and cos and sin waves for the two */
/* possible modulation frequencies. */
/* */
/* int *h: Pointer to h-set */
/* double hd: Denominator of h-set */
/* int i: h-set index */
/* int bit: Current transmitted bit */
/* int bitnode: Starting node for current bit */
/* double *corr*cos0: Pointer to correlation with '0' frequency cos */
/* double *corr*sin0: Pointer to correlation with '0' frequency sin */
/* double *corr*cos1: Pointer to correlation with '1' frequency cos */
/* double *corr*sin1: Pointer to correlation with '1' frequency sin */
/*****

void correlate (int *h, double hd, int i, int bit, int bitnode,
               double *corr*cos0, double *corr*sin0,
               double *corr*cos1, double *corr*sin1)
{
    if (bit+1)
    {
        *corr*cos0 = (sin((2*h[i] + bitnode)/hd*M_PI) -
                     sin(bitnode/hd*M_PI)) / (2*h[i]/hd*M_PI);
        *corr*sin0 = (sin((2*h[i] + bitnode+hd/2)/hd*M_PI) -
                     sin((bitnode+hd/2)/hd*M_PI)) / (2*h[i]/hd*M_PI);
        *corr*cos1 = cos(bitnode/hd*M_PI);
        *corr*sin1 = cos((bitnode+hd/2)/hd*M_PI);
    }
    else
    {
        *corr*cos0 = cos(bitnode/hd*M_PI);
        *corr*sin0 = cos((bitnode+hd/2)/hd*M_PI);
        *corr*cos1 = (sin((-2*h[i] + bitnode)/hd*M_PI) -
                     sin(bitnode/hd*M_PI)) / (-2*h[i]/hd*M_PI);
        *corr*sin1 = (sin((-2*h[i] + bitnode+hd/2)/hd*M_PI) -
                     sin((bitnode+hd/2)/hd*M_PI)) / (-2*h[i]/hd*M_PI);
    }
}

/***** dmin generate *****/
/* Function to calculate the minimum squared euclidean distance for a */
/* given h-set. Function returns the minimum squared euclidean distance. */
/* In the case of the minimum squared distance being less than the best */
/* minimum squared distance, the function aborts the calculation and sets */
/* the crash flag. */
/* */
/* int *sreg: Pointer to shift register */
/* int *h: Pointer to h-set */
/* double hd: Denominator of h-set */
/* int setsize: Size of h-set */
/* double bestdmin: Best minimum squared euclidean distance */
/* int *crash: Pointer to crash indicator */
/*****

```

```

double dmingen (int *sreg,int *h,double hd,int setsize,double bestdmin,
               int *crash)
{
    NODE *node;                /* Pointer to an array of nodes */
    NODE *temp;               /* Pointer to a temporary array */
    NODE *swap;               /* Pointer used for swapping */

    int snode0;               /* Starting node for a binary '0' */
    int snode1;               /* Starting node for a binary '1' */
    double corr0;             /* Path metric for '0' path */
    double corr1;             /* Path metric for '1' path */
    double corrcos0;          /* Correlation with '0' cos */
    double corrsin0;          /* Correlation with '0' sin */
    double corrcos1;          /* Correlation with '1' cos */
    double corrsin1;          /* Correlation with '1' sin */
    double dmin;              /* Minimum squared distance */

    int bit;                  /* Current transmitted bit */
    int bitnode;              /* Starting node for current bit */

    int i;                    /* h-set index */
    int bitcount;             /* Bit counter */
    int nodecount;            /* Node counter */
    int numnodes;             /* Number of nodes in node array */

    node = (NODE *)malloc(2*hd*sizeof(NODE));
    temp = (NODE *)malloc(2*hd*sizeof(NODE));

    initnode (node,hd);

    numnodes = 2*hd;
    bitnode = 0;
    dmin = 100;
    *crash = 0;

    for (bitcount = 0;bitcount < 1000 && dmin > bestdmin+0.001;
         bitcount++)
    {
        i = fmod (bitcount,setsize);
        bit = bitgen (sreg) * 2 - 1;

        correlate (h,hd,i,bit,bitnode,&corrcos0,&corrsin0,&corrcos1,&corrsin1);

        gotoxy (1,1);
        cprintf ("%d  %lf ",bitcount,dmin);

        bitnode = fmod (bitnode + bit*h[i] + numnodes,numnodes);

        for (nodecount = 0;nodecount < numnodes;nodecount++)
        {
            snode0 = fmod(nodecount + h[i] + numnodes,numnodes);
            snode1 = fmod(nodecount - h[i] + numnodes,numnodes);

            corr0 = node[snode0].metric + cos(snode0/hd*M_PI)*corrcos0
                    - sin(snode0/hd*M_PI)*corrsin0;
            corr1 = node[snode1].metric + cos(snode1/hd*M_PI)*corrcos1
                    - sin(snode1/hd*M_PI)*corrsin1;

            if ((corr0 > corr1 || !node[snode1].occupied) &&
                node[snode0].occupied)
            {
                if (node[snode1].occupied && node[snode0].occupied)
            {

```

```

        if (nodecount == bitnode && corr0-corr1 < dmin)
            dmin = corr0-corr1;
    }
    temp[nodecount].occupied = 1;
    temp[nodecount].metric = corr0;
}

    else if ((corr1 >= corr0 || !node[snode0].occupied) &&
            node[snode1].occupied)
    {
        if (node[snode0].occupied && node[snode1].occupied)
        {
            if (nodecount == bitnode && corr1-corr0 < dmin)
                dmin = corr1-corr0;
        }
        temp[nodecount].occupied = 1;
        temp[nodecount].metric = corr1;
    }

    else
    {
        temp[nodecount].occupied = 0;
        temp[nodecount].metric = 0;
    }
}
swap = node;
node = temp;
temp = swap;
}

if (bitcount < 1000)
    *crash = 1;

free (node);
free (temp);

return (dmin);
}

```

```

/***** Random number generate *****/
/* Function to generate a random number between 0 and 1, using a 10 bit */
/* sequence of random bits from the PN generator. Function returns the */
/* random number. */
/* */
/* int *sreg: Pointer to shift register */
/*****

```

```

float randnumgen (int *sreg)
{
    float randnum; /* Random number between 0 and 1 */

    randnum = bitgen (sreg) * 512 +
        bitgen (sreg) * 256 +
        bitgen (sreg) * 128 +
        bitgen (sreg) * 64 +
        bitgen (sreg) * 32 +
        bitgen (sreg) * 16 +
        bitgen (sreg) * 8 +
        bitgen (sreg) * 4 +
        bitgen (sreg) * 2 +
        bitgen (sreg);

    randnum += 0.5;
    randnum /= 1024;

    return (randnum);
}

```

```

/***** Initialise probability vector *****/
/* Function to initialise the PBIL probability vector to 0.5. */
/* */
/* float *pvector: Pointer to PBIL probability vector */
/* int samplesize: Size of PBIL sample */
/*****

void initpvector (float *pvector,int samplesize)
{
    int counter;

    for (counter = 0;counter < samplesize;counter++)
    {
        pvector[counter] = 0.5;
    }
}

/***** Learn probability vector *****/
/* Function to implement learning of the PBIL probability vector, using a */
/* PBIL sample from which to learn. Learning rate is 0.05 and this */
/* decreases asymptotically as the probability vector moves from 0.5 */
/* towards 0 or 1. */
/* */
/* int *sample: Pointer to PBIL sample */
/* int *pvector: Pointer to PBIL probability vector */
/* int samplesize: Size of PBIL sample */
/*****

void learnpvector (int *sample,float *pvector,int samplesize)
{
    int counter;
    float adjust; /* Adjustment to vector */

    for (counter = 0;counter < samplesize;counter++)
    {
        adjust = (1-pvector[counter]-sample[counter]) * 0.1;
        if (adjust > 0.05)
            adjust = 0.05;
        if (adjust < -0.05)
            adjust = -0.05;
        pvector[counter] += adjust;
    }
}

/***** Mutate probability vector *****/
/* Function to implement mutation of the PBIL probability vector. Each */
/* bit of the probability vector is mutated by an amount 0.25 in a random */
/* direction. */
/* */
/* int *sreg: Pointer to shift register */
/* float *pvector: Pointer to PBIL probability vector */
/* int samplesize: Size of PBIL sample */
/*****

void mutatepvector (int *sreg,float *pvector,int samplesize)
{
    int counter;
    float randnum;

    for (counter = 0;counter < samplesize;counter++)
    {
        randnum = randnumgen (sreg);

```

```

    if (randnum > 0.5)
        pvector[counter] += 0.25;
    else
        pvector[counter] -= 0.25;
}
}

```

```

/***** Clip probability vector *****/
/* Function to clip the PBIL probability vector to the region (0,1). */
/* float *pvector: Pointer to PBIL probability vector */
/* int samplesize: Size of PBIL sample */
/*****

```

```

void clippvector (float *pvector,int samplesize)
{
    int counter;

    for (counter = 0;counter < samplesize;counter++)
    {
        if (pvector[counter] > 1)
            pvector[counter] = 1;
        if (pvector[counter] < 0)
            pvector[counter] = 0;
    }
}

```

```

/***** Sample generate *****/
/* Function to generate a sample bit sequence using the PBIL probability */
/* vector. */
/* int *sreg: Pointer to shift register */
/* int *sample: Pointer to PBIL sample */
/* float *pvector: Pointer to PBIL probability vector */
/* int samplesize: Size of PBIL sample */
/*****

```

```

void samplegen (int *sreg,int *sample,float *pvector,int samplesize)
{
    int counter;
    float randnum;

    for (counter = 0;counter < samplesize;counter++)
    {
        randnum = randnumgen (sreg);

        if (randnum > pvector[counter])
            sample[counter] = 1;
        else
            sample[counter] = 0;
    }
}

```

```

/***** h-set generate *****/
/* Function to generate the h-set values from a PBIL sample bit sequence. */
/* int *h: Pointer to h-set */
/* int *sample: Pointer to PBIL sample */
/* int hd: Denominator of h-set */
/* int setsize: Size of h-set */
/* int samplesize: Size of PBIL sample */
/*****

```

```

void hgen (int *h,int *sample,int hd,int setsize,int samplesize)
{
    int multiplier;
    int counter;
    int temp;

    temp = samplesize/setsize;

    for (counter = 0;counter < samplesize;counter++)
    {
        if (!fmod(counter,temp))
        {
            h[counter/temp] = 0;
            multiplier = 1;
        }
        h[counter/temp] += multiplier*sample[counter];
        multiplier *= 2;
    }

    gotoxy (1,3);
    cprintf ("currh:  ");
    for (counter = 0;counter < setsize;counter++)
    {
        h[counter] = fmod(h[counter]+hd*7/8,hd) + 1;
        cprintf ("h%d = %d  ",counter,h[counter]);
    }
}

/***** Compare h-set *****/
/* Function to compare two h-sets. Function returns a 1 if h-sets are
/* the same, or a 0 if h-sets are different.
/*
/* int *h1: Pointer to h-set 1
/* int *h2: Pointer to h-set 2
/* int setsize: Size of h-set
/*****/

int compareh (int *h1,int *h2,int setsize)
{
    int counter;
    int same = 0;          /* Flag set if h-sets are equal */

    for (counter = 0;counter < setsize;counter++)
    {
        if (h1[counter] == h2[counter])
            same += 1;
    }

    if (same == setsize)
        same = 1;
    else
        same = 0;

    return (same);
}

/***** Copy array *****/
/* Function to copy an array of ints from source to destination.
/*
/* int *dest: Pointer to destination array
/* int *source: Pointer to source array
/* int arraysize: Size of array
/*****/

```

```

void copyarray (int *dest,int *source,int arraysize)
{
    int counter;

    for (counter = 0;counter < arraysize;counter++)
        dest[counter] = source[counter];
}

```

```

/***** Main *****/
/* Main procedure */
/*****

```

```

void main()
{
    int sreg1[20];          /* PN generator 1 shift register */
    int sreg2[20];          /* PN generator 2 shift register */

    int setsize;           /* Size of h-set */
    int *h;                 /* Pointer to h-set array */
    int *besth;            /* Pointer to best h-set array */
    int hd;                 /* Denominator of h-set */

    float *pvector;        /* Pointer to probability vector */
    float *stdpvector;     /* Pointer to standard 0.5 vector */
    int *sample;           /* Pointer to PBIL sample */
    int *bestsample;       /* Pointer to best PBIL sample */
    double dmin;           /* Minimum squared distance */
    double bestdmin;       /* Best minimum squared distance */
    int crash;             /* Indicator for dmingen crash */

    int samplesize;        /* Size of PBIL sample */

    int nums;              /* No of samples per generation */
    int numg;              /* Number of PBIL generations */

    int gcount;            /* Generation counter */
    int scount;            /* Sample counter */
    int samecount;         /* Counter h-sets same as best h */
    int mutatecount;       /* Counter for mutates */
}

```

```
clrscr();
```

```

printf ("\n\n\n\n");
printf ("Enter size of h set:          ");
scanf ("%d",&setsize);
printf ("Enter denominator of h set: ");
scanf ("%d",&hd);
printf ("\nEnter number of samples per PBIL generation: ");
scanf ("%d",&nums);
printf ("\nEnter number of PBIL generations:          ");
scanf ("%d",&numg);

```

```

hd = log(hd+1)/log(2);
samplesize = hd*setsize;
hd = pow(2,hd);

```

```

printf ("\n\n");
printf ("\nsetsize = %d",setsize);
printf ("\nhd = %d",hd);
printf ("\nnums = %d",nums);
printf ("\nnumg = %d",numg);
printf ("\n\n\n");

```

```
getch();
```

```

clrscr();

pvector = (float *)malloc(samplesize*sizeof(float));
stdpvector = (float *)malloc(samplesize*sizeof(float));
sample = (int *)malloc(samplesize*sizeof(int));
bestsample = (int *)malloc(samplesize*sizeof(int));

h = (int *)malloc(setsize*sizeof(int));
besth = (int *)malloc(setsize*sizeof(int));

seedsreg (sreg1);
seedsreg (sreg2);
initpvector (pvector,samplesize);
initpvector (stdpvector,samplesize);

for (scount = 0;scount < setsize;scount++)
{
    besth[scount] = 0;
}

bestdmin = 0;
mutatecount = 0;

for (gcount = 0;gcount < numg;gcount++)
{
    samecount = 0;
    for (scount = 0;scount < nums;scount++)
    {
        gotoxy (1,18);
        cprintf ("oddballs:  %d  ",samecount);
        samplegen (sreg2,sample,pvector,samplesize);
        hgen (h,sample,hd,setsample,samplesize);
        if (compareh(h,besth,setsample))
        {
            samplegen (sreg2,sample,stdpvector,samplesize);
            hgen (h,sample,hd,setsample,samplesize);
            samecount++;
        }
        dmin = dmingen (sreg1,h,hd,setsample,bestdmin,&crash);
        if (!crash)
        {
            bestdmin = dmin;
            gotoxy (55,8);
            cprintf ("last update:  %d  ",gcount);
            copyarray (bestsample,sample,samplesize);
            copyarray (besth,h,setsample);
        }
        gotoxy (1,8);
        cprintf ("generation:  %d          best dmin =  %lf",
                gcount,bestdmin);
    }
    learnpvector (bestsample,pvector,samplesize);
    if (samecount > nums/10-1)
    {
        mutatepvector (sreg2,pvector,samplesize);
        mutatecount++;
        gotoxy (1,20);
        cprintf ("mutates:  %d",mutatecount);
    }
    clippvector (pvector,samplesize);
    gotoxy (1,10);
    cprintf ("besth:  ");
    for (scount = 0;scount < setsize;scount++)
    {
        cprintf ("h%d = %d  ",scount,besth[scount]);
    }
}

getch();

```

```
free (pvector);  
free (stdpvector);  
free (sample);  
free (bestsample);  
free (h);  
free (besth);  
}
```

## Appendix E

# 'C' Code Listing for Viterbi Decoding Software

```
/*
*****
/* Viterbi decoding algorithm to calculate the bit error rate (BER) for
/* a user-defined noise power, NO. The bit energy, Eb, is fixed at 1J.
/*
*****

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <alloc.h>
#include <math.h>
#include <time.h>

#define IA 16807
#define IM 2147483647
#define AM (1.0/IM)
#define IQ 127773
#define IR 2836
#define NTAB 32
#define NDIV (1+(IM-1)/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)

int  FREQ;
int  SAMPLES;
int  TRUNCDEPTH;

float NO;

/*
*****
/* Data structure definition for struct NODE. This structure holds
/* information for a node in the phase trellis diagram.
/*
*****

typedef struct node
```

```

{
    int occupied;                /* Flag set if node is occupied */
    double metric;              /* Path metric for node */
    char path[500];            /* Path bit sequence */
} NODE;

/***** Random 1 *****/
/* Function to produce a random number between 0.0 and 1.0. Function */
/* returns the random number. Initialises if called with idum a negative */
/* number. */
/* Source: Numerical Recipes in C - Press, Teukolsky, Vetterling, */
/* Flannery */
/* long *sreg: pointer to seed */
/*****

float ran1 (long *idum)
{
    int j;
    int k;
    static long iy=0;
    static long iv[NTAB];
    float temp;

    if (*idum <= 0 || !iy)
    {
        if (-(*idum) < 1)
            *idum = 1;
        else
            *idum = -(*idum);
        for (j = NTAB+7; j >= 0; j--)
        {
            k = (*idum)/IQ;
            *idum = IA*(*idum-k*IQ)-IR*k;
            if (*idum < 0)
                *idum += IM;
            if (j < NTAB)
                iv[j] = *idum;
        }
        iy = iv[0];
    }
    k = (*idum)/IQ;
    *idum = IA*(*idum-k*IQ)-IR*k;
    if (*idum < 0)
        *idum += IM;
    j = iy/NDIV;
    iy = iv[j];
    iv[j] = *idum;
    if ((temp=AM*iy) > RNMX)
        return RNMX;
    else
        return temp;
}

/***** Gaussian deviate *****/
/* Function to produce a normally distributed random number with zero */
/* mean and unit variance. Function returns the random number. */
/* Source: Numerical Recipes in C - Press, Teukolsky, Vetterling, */
/* Flannery */
/* long *sreg: pointer to seed */
/*****

```

```

float gasdev (long *idum)
{
    float ran1 (long *idum);
    static int iset = 0;
    static float gset;
    float fac, rsq, v1, v2;

    if (iset == 0)
    {
        do
        {
            v1 = 2.0*ran1(idum)-1.0;
            v2 = 2.0*ran1(idum)-1.0;
            rsq = v1*v1+v2*v2;
        }
        while (rsq >= 1.0 || rsq == 0.0 );

        fac = sqrt (-2.0*log(rsq)/rsq);
        gset = v1*fac;
        iset = 1;
        return v2*fac;
    }
    else
    {
        iset=0;
        return gset;
    }
}

/***** Seed shift register *****/
/* Function to produce a random seed for the PN generator shift register. */
/*
/*   int *sreg: pointer to shift register
/*
/*****

void seedsreg (int *sreg)
{
    int counter;

    randomize();

    for (counter = 0;counter < 20;counter++)
        sreg[counter] = random(2);

    sreg[0] = 1;
}

/***** Bit generator *****/
/* Function to produce a random bit using the PN generator. Function
/* returns the random bit and advances the shift register. The shift
/* register is a 20 stage register with feedback taps 20 and 3. It
/* produces a pseudonoise sequence of period 1048575.
/*
/*   int *sreg: pointer to shift register
/*
/*****

int bitgen (int *sreg)
{
    int bit;
    int temp;

    bit = sreg[19];

    temp = (sreg[19]+sreg[2])% 2;

```

```

sreg[19] = sreg[18];
sreg[18] = sreg[17];
sreg[17] = sreg[16];
sreg[16] = sreg[15];
sreg[15] = sreg[14];
sreg[14] = sreg[13];
sreg[13] = sreg[12];
sreg[12] = sreg[11];
sreg[11] = sreg[10];
sreg[10] = sreg[9];
sreg[9] = sreg[8];
sreg[8] = sreg[7];
sreg[7] = sreg[6];
sreg[6] = sreg[5];
sreg[5] = sreg[4];
sreg[4] = sreg[3];
sreg[3] = sreg[2];
sreg[2] = sreg[1];
sreg[1] = sreg[0];
sreg[0] = temp;

return (bit);
}

/***** Initialise node *****/
/* Function to initialise the array of nodes. All node metrics are set */
/* to zero. All occupied flags are set to zero except for node zero. */
/* */
/* NODE *node: pointer to array of nodes */
/* int hd: denominator of h-set */
/*****/

void initnode (NODE *node,int hd)
{
    int counter1;
    int counter2;

    for (counter1 = 0;counter1 < 2*hd;counter1++)
    {
        node[counter1].occupied = 0;
        node[counter1].metric = 0;
        for (counter2 = 0;counter2 < TRUNCDEPTH;counter2++)
            node[counter1].path[counter2] = 0;
    }
    node[0].occupied = 1;
}

/***** Correlate *****/
/* Function to calculate the correlations (over a single bit period) */
/* between the transmitted signal and cos and sin waves for the two */
/* possible modulation frequencies. */
/* */
/* int *h: Pointer to h-set */
/* double hd: Denominator of h-set */
/* int i: h-set index */
/* int bit: Current transmitted bit */
/* int bitnode: Starting node for current bit */
/* long *idum: Seed for Gaussian noise generator */
/* double *corr*cos0: Pointer to correlation with '0' frequency cos */
/* double *corr*sin0: Pointer to correlation with '0' frequency sin */
/* double *corr*cos1: Pointer to correlation with '1' frequency cos */
/* double *corr*sin1: Pointer to correlation with '1' frequency sin */
/*****/

void correlate (int *h,double hd,int i,int bit,int bitnode,long *idum,

```

```

        double *corrcos0, double *corrsin0,
        double *corrcos1, double *corrsin1)
{
    double wave;
    double phase;
    float noiseampl;

    int counter;

    *corrcos0 = 0;
    *corrsin0 = 0;
    *corrcos1 = 0;
    *corrsin1 = 0;

    for (counter = 0; counter < SAMPLES; counter++)
    {
        phase = (FREQ*hd*2 + bit*h[i]) * (double)counter/SAMPLES;
        noiseampl = sqrt(N0*SAMPLES/2.0);
        wave = cos((phase+bitnode)/hd*M_PI)*sqrt(2) + gasdev(idum)*noiseampl;

        phase = (FREQ*hd*2 - h[i]) * (double)counter/SAMPLES;
        *corrcos0 += wave * cos(phase/hd*M_PI);
        *corrsin0 += wave * sin(phase/hd*M_PI);

        phase = (FREQ*hd*2 + h[i]) * (double)counter/SAMPLES;
        *corrcos1 += wave * cos(phase/hd*M_PI);
        *corrsin1 += wave * sin(phase/hd*M_PI);
    }
    *corrcos0 /= SAMPLES;
    *corrsin0 /= SAMPLES;
    *corrcos1 /= SAMPLES;
    *corrsin1 /= SAMPLES;
}

/***** Copy path *****/
/* Function to copy a node path from source to destination. */
/* */
/* int *dest: Pointer to destination node */
/* int *source: Pointer to source node */
/*****

void copypath (char *temp, char *path)
{
    int counter;

    for (counter = 0; counter < TRUNCDEPTH; counter++)
        temp[counter] = path[counter];
}

/***** Main *****/
/* Main procedure */
/*****

void main()
{
    long idum; /* Gaussian number generator seed */
    int sreg[20]; /* PN generator shift register */

    int setsize; /* Size of h-set */
    int *h; /* Pointer to h-set array */
    int hd; /* Denominator of h-set */

    NODE *node; /* Pointer to an array of nodes */
    NODE *temp; /* Pointer to a temporary array */
    NODE *swap; /* Pointer used for swapping */
}

```

```

int snode0;          /* Starting node for a binary '0' */
int snode1;          /* Starting node for a binary '1' */
double corr0;        /* Path metric for '0' path      */
double corr1;        /* Path metric for '1' path      */
double corrcos0;     /* Correlation with '0' cos      */
double corrsin0;     /* Correlation with '0' sin      */

double corrcos1;     /* Correlation with '1' cos      */
double corrsin1;     /* Correlation with '1' sin      */
double biggest;      /* Biggest path metric           */
int outnode;         /* Node with biggest path metric */

int bit;             /* Current transmitted bit       */
int bitnode;         /* Starting node for current bit */
char sentpath[500]; /* Transmitted path              */

int i;               /* h-set index                    */
int bitcount;        /* Bit counter                     */
int nodecount;       /* Node counter                     */
int numbits;         /* Number of bits to decode       */
int numnodes;        /* Number of nodes in node array  */
int errorcount;     /* Number of errors                */

FILE *fpo;

if (!(fpo = fopen ("viterbi.txt","w")))
{
    fprintf (stderr,"ERROR: couldn't open file viterbi.txt for writing\n");
    exit (1);
}

clrscr();

printf ("\n\n\n\n");
printf ("Enter number of bits to decode:   ");
scanf ("%d",&numbits);
printf ("Enter modulation center frequency: ");
scanf ("%d",&FREQ);
printf ("Enter number of samples per bit:   ");
scanf ("%d",&SAMPLES);
printf ("Enter viterbi truncation depth:    ");
scanf ("%d",&TRUNCDEPTH);
printf ("Enter noise PSD, N0:   ");
scanf ("%f",&N0);

printf ("\n");

printf ("Enter h-set denominator:   ");
scanf ("%d",&hd);
printf ("Enter h-set size:           ");
scanf ("%d",&setsize);

printf ("\n");

printf ("\nnumbits = %d",numbits);
printf ("\nFREQ = %d",FREQ);
printf ("\nSAMPLES = %d",SAMPLES);
printf ("\nTRUNCDEPTH = %d",TRUNCDEPTH);
printf ("\nN0 = %f",N0);

printf ("\n");

printf ("\nhd = %d",hd);
printf ("\nsetsize = %d",setsize);

```

```

h = (int *)malloc(setsize*sizeof(int));

printf ("\n\n\n");

for (i = 0;i < setsize;i++)
{
    printf ("Enter h[%d]:  ",i);
    scanf ("%d",&h[i]);
}

printf ("\n");

for (i = 0;i < setsize;i++)
{
    printf ("\nh[%d] = %d",i,h[i]);
}

getch();

node = (NODE *)malloc(2*hd*sizeof(NODE));
temp = (NODE *)malloc(2*hd*sizeof(NODE));

printf ("\n\n\n");

randomize();
idum = random(10) - 10;

seedsreg (sreg);
initnode (node,hd);

numnodes = 2*hd;
bitnode = 0;
errorcount = 0;

for (bitcount = 0;bitcount < numbits;bitcount++)
{
    i = fmod (bitcount,setsize);
    bit = bitgen (sreg) * 2 - 1;

    sentpath[fmod(bitcount,TRUNCDEPTH)] = (bit+1)/2;

    correlate (h,hd,i,bit,bitnode,&idum,&corrcos0,&corrsin0,
               &corrcos1,&corrsin1);

    bitnode = fmod (bitnode + bit*h[i] + numnodes,numnodes);

    for (nodecount = 0;nodecount < numnodes;nodecount++)
    {
        snode0 = fmod(nodecount + h[i] + numnodes,numnodes);
        snode1 = fmod(nodecount - h[i] + numnodes,numnodes);

        corr0 = node[snode0].metric + cos(snode0/(double)hd*M_PI)*corrcos0
              - sin(snode0/(double)hd*M_PI)*corrsin0;
        corr1 = node[snode1].metric + cos(snode1/(double)hd*M_PI)*corrcos1
              - sin(snode1/(double)hd*M_PI)*corrsin1;

        if ((corr0 > corr1 || !node[snode1].occupied) &&
            node[snode0].occupied)
        {

```

```

temp[nodecount].occupied = 1;
temp[nodecount].metric = corr0;
copypath (temp[nodecount].path,node[snode0].path);
temp[nodecount].path[fmod(bitcount,TRUNCDEPTH)] = 0;
}
else if ((corr1 >= corr0 || !node[snode0].occupied) &&
        node[snode1].occupied)
{
temp[nodecount].occupied = 1;
temp[nodecount].metric = corr1;
copypath (temp[nodecount].path,node[snode1].path);
temp[nodecount].path[fmod(bitcount,TRUNCDEPTH)] = 1;
}
else
{
temp[nodecount].occupied = 0;
temp[nodecount].metric = 0;
}
}
swap = node;
node = temp;
temp = swap;

cprintf ("\n\r%d",bitcount);

if (bitcount+1 > TRUNCDEPTH-1)
{
biggest = NULL;
outnode = 0;
for (nodecount = 0;nodecount < 2*hd;nodecount++)
{
if (node[nodecount].occupied)
{
if (biggest == NULL || node[nodecount].metric > biggest)
{
biggest = node[nodecount].metric;
outnode = nodecount;
}
}
}
}
cprintf ("  %d",node[outnode].path[fmod(bitcount+1,TRUNCDEPTH)]);

if (node[outnode].path[fmod(bitcount+1,TRUNCDEPTH)] !=
sentpath[fmod(bitcount+1,TRUNCDEPTH)])
{
errorcount++;
}
cprintf ("  errors: %d",errorcount);
}
}

printf ("\n\n");
printf ("\nNumber of bits:  %d",numbits-TRUNCDEPTH+1);
printf ("\nNumber of errors: %d",errorcount);
printf ("\n\nBER = %f\n", (float)errorcount/(numbits-TRUNCDEPTH+1));

getch();

free (node);
free (temp);
free (h);

fclose (fpo);
}

```

# References

- [1] **Amoroso, F.** 'The Bandwidth of Digital Data Signals'. *IEEE Communications Magazine*, November 1980, pp.13-24.
- [2] **Anderson, JB. Aulin, T. and Sundberg, C.** *Digital Phase Modulation*. Plenum Press, 1986.
- [3] **Böhm, B.** *DSP Implementation and Performance Testing of the Massey-Hodgart MSK Demodulator*. MSc thesis, Dept Elec.Eng. University of Cape Town, June 1995.
- [4] **Cuthbert, J.** *High Performance Multi-h CPFSK Modulator and Demodulator Design using Population Based Incremental Learning Search Methods*. PhD thesis, Dept Elec.Eng. University of Cape Town, December 1996.
- [5] **Dixon, RC.** *Spread Spectrum Systems with Commercial Applications - Third Edition*. John Wiley & Sons, 1994.
- [6] **Shannon, CE.** 'A Mathematical Theory of Communications'. *Bell Systems Technical Journal*, vol.27, pp.379-423, July 1948, and pp.623-656, October 1948.
- [7] **Stremmer, FG.** *Introduction to Communication Systems - Third Edition*. Addison-Wesley, 1990.
- [8] **Viterbi, AJ.** 'Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm'. *IEEE Transactions on Communication Technology*, vol.COM-19, pp.751-772.
- [9] **Ziener, RE. and Peterson, RL.** *Introduction to Digital Communication*. Macmillan Publishing Company, 1992.