SOME APPLICATIONS OF MATHEMATICS TO CODING THEORY

Elizabeth Jane Dunscombe

Royal Holloway and Bedford New College

(University of London)

Submitted for the degree of Ph.D.

ABSTRACT

This thesis deals with the transmission of data over a channel that is subject to noise, or interference. There are many different methods of trying to achieve reliable communication of data in the presence of noise. This thesis considers some of these methods, in particular, those aspects involving the use of error-correcting codes. A number of specific applications are considered, as well as some more general theory.

One general class of codes is that of cyclic codes (where every cyclic shift of a codeword is also a codeword). Chapter 2 of this thesis reviews a decoding scheme for cyclic codes proposed by Professor P.M. Cohn. The scheme is a modification of standard array syndrome decoding. It is shown that Cohn's scheme does not perform as well as the original version of syndrome decoding.

Chapter 3 considers Cyclotomically Shortened Reed Solomon codes (a class of codes introduced by J.L. Dornstetter) and their relationship with the Chinese Remainder Theorem codes of J.J. Stone. The blocklength and dimension of these codes is established, together with the best possible lower bound on the minimum distance. The notion of cyclotomic shortening is then extended to Alternant codes.

Chapter 4 deals with the subject of interleaving for channels that are subject to bursts of errors. An optimal solution is given to a problem posed by Inmarsat when interleaving is used with a convolutional code. It is shown how to improve the method of interleaving which feeds data column-wise into an array and then transmits row-wise, by careful selection of the order in which the rows are transmitted.

The final chapter discusses the concept of an error-correcting code with two different codeword lengths. Some general results about such codes are presented. A method of forming these codes is given for the case when one wordlength is twice the other. A specific example of this type of code is considered. Both theoretical and simulated performance results are presented for the example.

ACKNOWLEDGEMENTS

I wish to thank Professor F.C. Piper for his guidance and support throughout this research and for always being right about the next phase of post-graduate life. My thanks also to Dr. M. Walker of Racal Research Ltd. for his help and for giving me his time despite a busy schedule.

I am grateful to everyone in the Mathematics Department at RHBNC for making my stay as a post-graduate such a pleasant one. Particular thanks to Dr. P. Wild for spending many hours listening to my ideas (both good and bad) and for his helpful suggestions. Also thanks to Dr. J.D. Knowles for his help with Statistics.

I wish to thank my husband, Chris, whose faith in my abilities always exceeded my own, for his constant encouragement and support.

I gratefully acknowledge the financial support for this research from the Science and Engineering Research Council and Racal Research Ltd.

I lift my eyes to the hills –

from where does my help come ?

My help comes from the Lord,

the Maker of heaven and earth.

Psalm 121, v. 1 - 2.

CONTENTS

Table of Figures

Chapter 1

INTRODUCTION

§1.1.1 Introduction

This chapter gives an overview of the subject of Coding Theory from a mathematical viewpoint. It pays particular attention to those topics which will be considered further in this thesis. However, only those results which are widely presented in the literature are discussed here, the lesser known results will be detailed in the introduction to the chapter to which they are relevant.

For the world to function as it does today, there has to be a large amount of information transferred from one place to another, this may be done verbally or by written text of some form. Information is transferred through a *communication system*. In general, a communication system consists of an *information source*, a *communication channel* and a *receiver*. The communication channel can take many forms, for example a public telephone circuit between Glasgow and Bangor or the link between a satellite and a ground station.

The large amount of data that is transferred, together with the nature of the physical world, results in most communication channels being subject to 'noise'.

Noise is some event that occurs whilst data is passing through a channel so that the data that is received is not exactly the data that was sent. The effects of the noise on the data will be described as *errors* on the channel. If redundancy (extra data that is derived in some way from the original information) is added to the data before it is transmitted over the channel, then at the receiver it may be possible to use this redundancy to detect or even correct the errors that have occurred. As a rough guide, the more redundancy that is added the more errors that can be dealt with. The term *information* will be used to describe that which the information source produces. The term *data* will be used to describe that which is actually transmitted over the communication channel, i.e. the information with the redundancy added.

A *code* is a set of sequences of data symbols, called *codewords*, that contain redundancy. The process called *encoding* is the assigning of codewords to information sequences. To enable the information to be recovered at the receiver, the code must contain a unique codeword for every possible information sequence.

In summary, a source produces information which is encoded to codewords, these codewords are then transmitted across a communication channel and errors occur. A codeword with errors added is received, this will be called the *received word*. The process known as decoding uses the redundancy in the received word to try to recover the transmitted codeword. Once a codeword is obtained it is then mapped to its corresponding information sequence, but this is not usually considered as part of the decoding process.

The speed at which the information is transmitted between the information source and the receiver is often important and, assuming that data symbols can only be transmitted across the channel at a fixed rate, the more redundancy that is added, the slower the receiver collects the information.

Thus the aim of Coding Theory is to try to balance the conflicting requirements of transmitting data accurately (i.e. with as few errors as is possible) and efficiently (i.e. as fast as possible).

Throughout this thesis, the symbol q will be used to represent a prime power and it will be assumed, unless otherwise stated, that the information source produces a sequence of symbols from the Galois field $GF(q)$.

§1.2.1 Block Codes

A block code divides the information produced by the source into fixed length blocks of symbols and encodes each block to a fixed length codeword independently of the surrounding blocks.

Let M be the number of different sequences of k symbols from $GF(q)$ that the information source can produce.

Definition 1.1 A *block code* of size M over $GF(q)$ is a set of M codewords of length n over $GF(q)$. □

To use a block code it is necessary to define a one-to-one map from the set of M sequences of length k produced by the information source to the set of codewords. Encoding is then performed by taking the first k consecutive symbols produced by the information source, mapping them to the corresponding codeword and repeating the process with the next k consecutive symbols, until all the information has been encoded. Each codeword is transmitted as it is produced.

Often $M = q^k$, that is the information source may produce all possible sequences of length k over $GF(q)$. In this case the code is called an <u>(n, k) code</u>, n is the *<u>blocklength</u>* of the code and k is the *<u>dimension</u>* of the code. Notice that the codewords may be considered as vectors over $GF(q)^n$.

<u>Definition 1.2</u> A *<u>linear code</u>* is an (n, k) code over $GF(q)$, where the set of codewords is a subspace of $GF(q)^n$ of dimension k. □

A vector space of dimension k can be generated by any set of k linearly independent vectors from the space. This result enables the encoding procedure for a linear code to be defined in terms of an $k \times n$ matrix, called the *<u>generator matrix</u>* of the code. The rows of the generator matrix are taken to be any k linearly independent codewords. Encoding is then performed by representing the k information symbols as a row vector, i, in $GF(q)^k$ and then

$$\mathbf{c} = \mathbf{i}\mathbf{G}$$

where **c** is a codeword vector and **G** is the generator matrix of the code.

Let C be a code, then

Definition 1.3 The orthogonal complement, $C^{\perp}$, of C is the set of all vectors which are orthogonal to every vector in C . $C^{\perp}$ can also be thought of as a code and is called the dual code of C .□

The dual code of a linear (n, k) code is linear with dimension $n-k$ and therefore any $n-k$ linearly independent vectors of $C^{\perp}$ generate the dual code. Let $\mathbf{H}$ be a generator matrix for the dual code, then for any $\mathbf{c} \in C$,

$$\mathbf{c}\mathbf{H}^{\mathbf{T}} = \mathbf{0} \tag{1.1}$$

because $\mathbf{c}$ is orthogonal to every row of $\mathbf{H}$. In fact, any vector over $GF(q)^n$ is a codeword iff it satisfies (1.1). $\mathbf{H}$ is called the parity-check matrix of C .

An important parameter in Coding Theory is the distance between codewords, the most frequently used distance measure is given by the following four definitions.

Definition 1.4 The Hamming distance , $d(\mathbf{x}, \mathbf{y})$, between two n-tuples $\mathbf{x}$ and $\mathbf{y}$ is the number of places in which they differ. □

Definition 1.5 The minimum Hamming distance, d, of a code C , is the least Hamming distance between any pair of codewords. □

Definition 1.6 The Hamming Weight, $w(\mathbf{c})$, of a codeword $\mathbf{c}$, is the number of non-zero entries in $\mathbf{c}$.□

Definition 1.7 The minimum Hamming weight, w, of a code C is the smallest Hamming weight of any non-zero codeword contained in C . □

The following theorem shows the relationship between Hamming distance and Hamming weight for linear codes.

<u>*Theorem 1.1*</u> *For a linear code, the minimum Hamming distance is equal to the minimum Hamming weight. That is*

$$d = w$$

□

<u>*Proof*</u> [Bl], page 46 (Thm 3.1.3). □

In this thesis, Hamming distance will be refered to simply as distance and Hamming weight simply as weight, except where two distance measures are in use at one time, when the full name for each distance will be used to avoid confusion.

For a linear code, the minimum distance can be found very easily from the parity-check matrix, **H**.

<u>*Theorem 1.2*</u> *The minimum distance, d, of a code, C, is the largest value of x, such that every set of $x - 1$ columns of* **H** *is linearly independent.* □

<u>*Proof*</u> [Bl], page 48 (Corollary 3.2.3). □

There is a well-known upper bound on the value of d for given n and k for a linear code.

Theorem 1.3 (The Singleton Bound) *The minimum distance of any (n,k) linear code satisfies*

$$d \leq n - k + 1$$

□

Proof [Bl], page 50 (Theorem 3.2.6). □

Definition 1.8 Any linear code whose minimum distance satisfies

$$d = n - k + 1$$

is called maximum distance separable. □

Lastly, a measure of how much using an (n,k) code slows down the speed at which information is transmitted over the communication channel is introduced.

Definition 1.9 The data rate of an (n,k) code is the quantity k/n. □

This section has described block codes and also linear block codes. An encoding method has been introduced for linear block codes and the Hamming distance measure has been defined. The majority of work in coding theory has been performed on linear codes. These codes have a strong structure which helps in the search for good codes (i.e. those which correct many errors whilst maintaining a high data rate). Also linear codes are the only block codes for which practical decoding algorithms have been found. The next section looks at decoding methods for linear block codes.

§1.2.2 <u>Decoding Linear Block Codes</u>

There are several different families of linear block codes, formed by imposing restrictions on either the generator matrix or the parity-check matrix, specific decoding methods have been developed for each of these families. However, this section will consider only general decoding methods that could be used for any linear block code.

Most decoding schemes aim to decode the received word to the closest codeword, that is the codeword that is the least Hamming distance away (with a choice being made in the event of a tie). The object of a good decoding scheme is to reduce the number of codewords that have to be searched through before the closest one can be found.

<u>Correct decoding</u> occurs when the received word is decoded to the codeword that was transmitted, otherwise *<u>erroneous decoding</u>* takes place. Suppose that t symbols in the received word are in error (that is different from the ones that were sent), then correct decoding occurs if all codewords (other than the one transmitted) are at distance $> t$ from the received word. This occurs if

$$d \geq 2t + 1$$

Thus a code can be guaranteed to decode a received word correctly if it contains at most $\lfloor \frac{d-1}{2} \rfloor = t$ errors. Such a code is called *<u>t-error-correcting</u>* and t is called the *<u>error-correcting capability</u>* of the code. Erroneous decoding occurs if so many errors have occurred that the received word is closer to a codeword other than the one that was sent, that is the error-correcting capability of the code has been exceeded.

Suppose a word y $\in GF(q)^n$ has been received, then it is required that c $\in \mathcal{C}$, the closest codeword to y, is found. Perhaps the simplest algorithm for doing this is using a <u>standard array</u>. A standard array has dimensions $(q)^{n-k} \times (q)^k$ and one method of forming it is as follows :

(i) Take the first row to be the set of codewords, with the all-zero codeword in the first position (the remaining codewords may be in any order). Set $i = 2$.

(ii) To obtain the i^{th} row of the array, form a set of vectors which consists of all vectors in $GF(q)^n$ which do not already appear in the array. Select from this set a vector of least weight (in the event of a tie, a free choice may be made). This vector is placed in the first position in the i^{th} row. Each of the remaining positions are filled with the sum, over $GF(q)$, of the chosen vector and the codeword at the top of the column in question. Set $i = i + 1$.

(iii) Repeat Step (ii) until $i = (q)^{n-k} + 1$

To decode y, find y in the standard array and decode it to the codeword at the head of the column in which y is found.

The standard array contains all the elements of $GF(q)^n$, that is the set of all possible received words. Each column in the array consists of those received words that have the codeword at the top of the column as their closest codeword. The reader is refered to [Bl], § 3.3 for further details.

It is clear that for large n and k, the standard array would be too large to store or to list. However, there is a variation on standard array decoding that requires only the first column of the array to be stored, along with another column of vectors of length $n - k$ over $GF(q)$. For this decoding method it is necessary to introduce the concept of the syndrome of the received word.

Definition 1.10 For any received word $\mathbf{y}$, the syndrome, $\mathbf{s}$, of $\mathbf{y}$ is defined by

$$\mathbf{s} = \mathbf{y}\mathbf{H}^{\mathbf{T}}$$

□

Now $\mathbf{y} = \mathbf{c} + \mathbf{e}$, where $\mathbf{c}$ is a codeword and $\mathbf{e}$ is the error pattern. Thus

$$\begin{aligned} \mathbf{s} &= (\mathbf{c} + \mathbf{e})\mathbf{H}^{\mathbf{T}} \\ &= \mathbf{c}\mathbf{H}^{\mathbf{T}} + \mathbf{e}\mathbf{H}^{\mathbf{T}} \\ &= \mathbf{e}\mathbf{H}^{\mathbf{T}} \end{aligned}$$

Therefore the syndrome of a received word depends only on the error pattern. Notice that all the elements in a row of the standard array have the same syndrome, because each row consists of all the codewords with the same vector added to each. Thus decoding may be performed by listing the first column of the standard array together with the syndrome of each vector in this column. Then decoding may be performed by calculating the syndrome of the received word, finding the syndrome in the table and taking the corresponding vector as the error pattern. Subtracting the error pattern from the received word gives the required codeword.

Syndrome decoding may be used for any linear block code and the principle is used in many decoding schemes for specific codes (although not always explicitly). The next section examines the sub-class of linear block codes called cyclic codes.

§1.2.3 Cyclic Codes

Definition 1.11 A linear (n, k) code over $GF(q)$ is a cylic code if, when $\mathbf{c}$ is a codeword, every cyclic shift of $\mathbf{c}$ is also a codeword. □

Cyclic codes are best described by using polynomials. First, recognise that each vector in $GF(q)^n$ can be represented as a polynomial in x of degree $\leq n-1$. The components of the vector become the coefficients of the polynomial. The set of polynomials formed from all the vectors of $GF(q)^n$ forms the ring $GF(q)[x]/(x^n - 1)$. Thus any cyclic code is a subset of this ring. Suppose the polynomial $c(x)$ is a codeword, then a cyclic shift of this codeword can be written in terms of multiplication within the ring as follows

$$xc(x) = R_{x^n-1}[xc(x)]$$

where $R_{x^n-1}[p(x)]$ denotes $p(x)$ modulo $x^n - 1$. It is recognising that a cyclic shift can be written in this form that leads to the following theorem.

Theorem 1.4 *In the ring $GF(q)[x]/(x^n - 1)$, a subset is a cyclic code iff the subset is an ideal of the ring.* □

Proof [Bl], page 97 (Theorem 5.2.1). □

Now, in the subset of polynomials that forms a cyclic code there exists a unique, non-zero, monic polynomial of least degree ($= n - k$), ([Bl], page 97). This polynomial may be used to generate the code (see Theorem 1.5) and hence is called the generator polynomial of the code and is denoted by $g(x)$. For $g(x)$ to generate the code, the information symbols are represented as a polynomial, $a(x)$, of degree $\leq k-1$ over $GF(q)$.

Theorem 1.5 *A cyclic code consists of all multiples of $g(x)$ by polynomials of degree $k-1$ or less.* □

Proof [Bl], page 98 (Theorem 5.2.2). □

The generator polynomial replaces the generator matrix for an ordinary linear block code. In the same way, it is possible to define a parity-check polynomial, but first Theorem 1.6 is needed.

Theorem 1.6 *There is a cyclic code of blocklength n, with generator polynomial $g(x)$ iff $g(x)$ divides $x^n - 1$.* □

Proof [Bl], page 98 (Theorem 5.2.3). □

Definition 1.12 The parity-check polynomial, $h(x)$, is given by

$$g(x)h(x) = x^n - 1$$

□

The definitions of standard array decoding and syndrome decoding can be written in terms of polynomials for cyclic codes. This will be considered in Chapter 2, where the effects of choosing a different vector from each row to be placed in the first column will be examined. The next section deals with a particular class of cyclic codes called Reed Solomon codes.

§1.3.1 Reed Solomon Codes

Reed Solomon codes are defined over the extension field $GF(q^m)$, so, for this section, an information source producing elements of $GF(q^m)$ will be assumed. The codes are defined by describing how to form the generator polynomial.

Definition 1.13 Let α be a primitive n^{th} root of unity in $GF(q^m)$ and j_0 a positive integer. Then a *t-error-correcting Reed Solomon code* over $GF(q^m)$ of blocklength $n \mid q^m - 1$ is a cyclic code with generator polynomial

$$g(x) = (x - \alpha^{j_0})(x - \alpha^{j_0+1}) \cdots (x - \alpha^{j_0+2t-1})$$

and has dimension $n - 2t$. □

Reed Solomon codes are maximum distance separable ([Bl], page 175 (Theorem 7.3.1)). Encoding and decoding processes for Reed Solomon codes are well-known and the interested reader is refered to [Bl], [PW] or [MWS], as these processes are not relevant to this thesis.

It is also possible to define Reed Solomon codes in terms of Galois-field Fourier Tranforms, which are the subject of the next definition.

Definition 1.14 Let $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ be a vector of $GF(q)^n$, where $n \mid q^m - 1$ for some m, and let α be an element of $GF(q^m)$ of order n. The _Galois-Field Fourier Transform_,(GFFT), of v is the vector

$\mathbf{V} = (V_0, V_1, \ldots, V_{n-1})$ given by

$$V_j = \sum_{i=0}^{n-1} \alpha^{ij} v_i \qquad j = 0, 1, \ldots, n-1.$$

□

Definition 1.14 gives the GFFT of a vector over $GF(q)$as a vector over $GF(q^m)$. The following theorem takes a vector over $GF(q^m)$ and shows when its inverse GFFT is a vector over $GF(q)$.

Theorem 1.7 *Let* $\mathbf{V}$ *be a vector of length* n *over* $GF(q^m)$ *where* $n \mid q^m - 1$. *Then the inverse GFFT* v *is a vector of elements of* $GF(q)$ *iff the following equations are satisfied :*

$$V_j^q = V_{((qj))} \qquad j = 0, \ldots, n-1.$$

where the double brackets indicate modulo $n-1$ *arithmetic.* □

Proof [Bl], page 221 (Theorem 8.2.1). □

Theorem 1.7 plays an important role in Chapter 3 of this thesis. The only further definition that will be presented here is the GFFT definition of Reed

Solomon codes. The theory behind this definition and of GFFT as a whole may be found in [Bl], Chapter 8.

Definition 1.15 A *t-error-correcting Reed Solomon* code over $GF(q^m)$, of blocklength $n \mid q^m - 1$ and dimension $n-2t$, may be formed by taking a vector of length n, setting $2t$ consecutive terms equal to zero, and filling the remaining places with the information symbols. The codeword is then the inverse GFFT of this vector. To form the complete set of codewords, the $2t$ consecutive zeros must occur in a fixed place. □

Thus far, only block codes have been discussed. The next section introduces the other fundamental group of codes that are part of Coding Theory, that is *convolutional codes*.

§1.4.1 Convolutional Codes

Unlike block codes, convolutional codes do not break up the sequence of information symbols into blocks and encode each block independently. The sequence is split into blocks (called *frames*) of k_0 symbols, where k_0 is small (and is often equal to one). At each time interval a frame is fed into the encoder and $n_0 > k_0$ symbols are output from the encoder, however these n_0 symbols will depend upon the input frame and also the previous $k-1$ frames (k is called the *constraint length* of the code).

Perhaps the best way to describe a convolutional code is in terms of a state transition diagram. Each state is labelled with the values in the k current frames, two states are joined if the input of a single frame can take the first state to the second. The branches are labelled with the n_0 symbols that the encoder outputs given that this transition takes place.

<u>*Example 1.1*</u> Consider a binary convolutional code with $k_0 = 1, n_0 = 2$ and $k = 3$, and state transition diagram as in Figure 1.1.

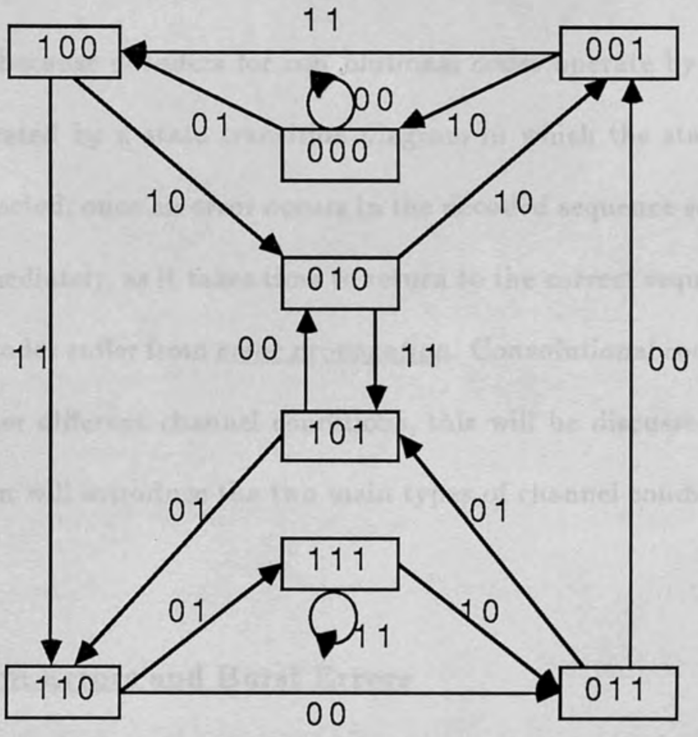


Figure 1.1

Then an information sequence 1101 will be encoded to 01110001. (Note : the encoder starts in state 000 and all sequences of bits are read from left to right).

□

To decode a convolutional code, in principle the whole received sequence must be compared with every possible sequence that the encoder could produce, the closest one being chosen. However, in practice this is impossible for any reasonable length message as the number of possible sequences grows rapidly with the length of the message. So a decoder that only searches through some of the possible sequences must be found. The best known such decoder is the Viterbi decoder ([Bl], pages 377 - 382), which considers, at any one time, sequences of length only $\alpha k n_0$, where α is a small positive integer.

Observe that, because decoders for convolutional codes operate by selecting a sequence generated by a state transition diagram in which the states are not fully interconnected, once an error occurs in the decoded sequence several more will follow immediately, as it takes time to return to the correct sequence. Thus convolutional codes suffer from *error propagation*. Convolutional codes perform differently under different channel conditions, this will be discussed in §1.5.2, the next section will introduce the two main types of channel conditions.

§1.5.1 Random Errors and Burst Errors

The term 'channel conditions' refers to the type of errors a channel is subject to. Consider a channel that has the binary digits, 0 and 1, as the data symbols that it can transmit. On most practical channels, it is waveforms of energy that are transmitted over the channel. Suppose that 0 and 1 are transmitted as the waveforms shown in Figure 1.2, this may be thought of as positive and negative energy.

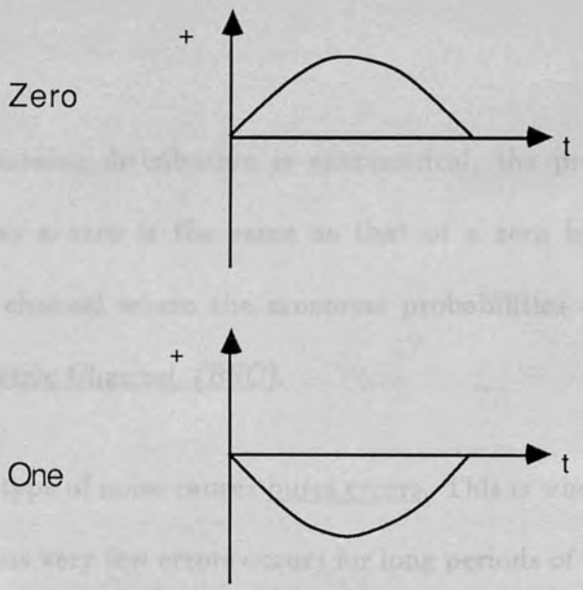


Figure 1.2

At the receiver, if positive energy is received in a time interval, then this is interpreted as a zero, in the same way negative energy is interpreted as a one. The noise that causes errors comes from other waveforms passing through the same region as the channel. Suppose that the effect of all these other waveforms is compounded together and can be considered as a single noise wave. The data waveform and the noise waveform are added by the principle of superposition. An error occurs in a given time interval when the superimposed waveforms have energy of a different sign from that of the data waveform.

The channel conditions are determined by the statistical behaviour of the noise waveform. One type of noise is *Additive White Gaussian Noise, (AWGN)*, this occurs when the average energy level of the noise waveform in each time interval follows a Gaussian distribution, with mean zero and the variance dependent on the amount of noise. This produces *random errors* and the probability of a received bit being in error can be easily calculated from the Gaussian distribution.

Because the Gaussian distribution is symmetrical, the probability of a one being received as a zero is the same as that of a zero being received as a one. A binary channel where the crossover probabilities are equal is called a _Binary Symmetric Channel, (BSC)_.

The other main type of noise causes _burst errors_. This is when the noise level is very low (and thus very few errors occur) for long periods of time, but these are interspersed with quite long bursts of continuous high level noise, producing a long burst of erroneous bits. An example of such a channel is a mobile telephone link from a moving vehicle. The low noise level occurs when the vehicle is moving through open ground and a burst may be caused by the vehicle travelling through a tunnel. The maximum number of consecutive bits that may be received in error is called the _maximum burst length, b_.

The above ideas have been expressed in terms of binary transmission, but similiar ideas extend to the transmission of other types of data symbols.

§1.5.2 Block Codes vs Convolutional Codes

The two different types of errors described above have different effects on error-correcting codes. When random errors occur with a low probability of a symbol being in error, then the number of errors that occur is usually within the error-correcting capability of the code. As the probability of a symbol being in error increases, the error-correcting capability of the code with be exceeded more often and more decoding errors will occur.

Convolutional codes are relatively slow to decode even with a Viterbi decoder. However they do perform well in the presence of AWGN when it is acceptable for the received information to contain a few errors. Block codes can provide a higher data integrity under AWGN, but may have a slower data rate, which again may be offset by a faster decoder. Thus, under AWGN, each individual situation must be considered before deciding whether a block code or a convolutional code would be best.

Burst errors, however, provide long periods of time when virtually no error-correction is needed. But a burst of errors will normally exceed the error-correcting capability of the code. Convolutional codes do not perform well with burst errors. This is because a long burst of errors will cause the decoded sequence to deviate a long way from the transmitted sequence and recovery may take a long period of time.

In the presence of burst errors, block codes always perform better than convolutional codes. This is because the nature of block codes restricts the decoding errors that result from the burst to just those codewords affected by the burst, that is there is no error propagation. The longer the blocklength of the code the better able it is to cope with burst errors. For example, given 100 bits to transmit, it is advantageous to use a 10-error-correcting code with blocklength 100 than a 1-error-correcting code with blocklength 10. Although both codes can correct 10 errors in 100 bits, the first code will correct this many errors even if they all occur together in a block of ten whereas the second code can correct 10 errors only if one error occurs in every 10 bits.

[BPP] provides a very good comparison between block codes and convolutional codes and expands on this section. The next section looks at a technique for randomising burst errors.

§1.6.1 Interleaving

The previous section demonstrated that, supposing a fixed number of errors where to occur during the transmission of a message, better data integrity may be obtained if they are spread out rather than all occurring in a block. However, as data sometimes has to be sent over a channel that is subject to burst errors, is there anything that can be done to transform a bursty chanel to make it look like a channel that produces random errors ? This section describes a technique called *interleaving*, which is fairly successful at randomising bursts of errors.

In this section a codeword will be either a block code codeword or a sequence of some given length produced by a convolutional code.

The principle behind interleaving is to transmit all the symbols of a codeword at widely spaced (compared with the maximum burst length of the channel) intervals with the intervening spaces being similarly filled by symbols of other codewords. At the receiving end, the symbols are placed back in their codewords and decoded in the normal way. In this manner a burst of errors appears as random errors spread over many codewords.

One method of interleaving is to feed the data into the columns of an $n \times m$ array and then transmit row by row. At the receiving end, the inverse process is performed. This ensures that a given burst of errors can affect at most one symbol in any column, provided that the maximum number of symbols that a single burst can affect (the maximum burst length) is less than or equal to m.

However interleaving introduces a time delay into the system because the whole array has to be filled before transmission can begin and, more importantly, the whole array has to be received before decoding can commence. Therefore it may not be practically possible to make the rows of the array sufficiently long so that m is greater than or equal to the maximum burst length. Chapter 4 will consider a method of overcoming this problem which involves transmitting the rows of the array in non-sequential order.

§1.7.1 Variable Length Codes

When transmitting information across a channel, one important parameter is the data rate. Much work has been done on improving the data rate for transmission. Consider first a noiseless, binary channel, then to transmit a message of English text each character of the text must be given a unique binary representation. Suppose each character is represented by the same number of bits, then, if just the 26 alphabetic characters are considered, each must be represented by at least 5 bits. But in English text some characters occur more frequently than others (e.g. the letter E occurs about twice as often as the letter S). If the condition that each character is represented by the same number of bits is removed and the frequently occuring characters are represented by fewer bits than the less common ones, then the overall length of the message may be reduced. Such codes are called Variable Length codes.

§1.7.2 Discussion of Variable Length Codes and Noiseless Channels

Consider a set of information symbols

$$\mathcal{S} = \{s_0, s_1, \ldots, s_m\}$$

associated with each symbol, s_i, is a probability, p_i, that the symbol will be transmitted at any given instant. Denote the set of codewords by

$$\mathcal{C} = \{c_0, c_1, \ldots, c_m\}$$

where each c_i consists of w_i symbols from some code alphabet, $\mathcal{A}$. For a variable length code to be useful, there must be no confusion as to how any sequence of symbols from the code alphabet is to be split into codewords. (Assuming that the sequence is formed from a series of codewords). Thus any useful code must satisfy Definition 1.16.

Definition 1.16 A Variable Length code is uniquely decodable if a sequence of symbols from $\mathcal{A}$ can be split up into codewords in at most one way. □

Example 1.2 If $\mathcal{C} = \{1, 0, 11, 10\}$, then this code is not uniquely decodable, e.g. the sequence 1011 could be split as 1 0 11 or 10 11 or 1 0 1 1 or 10 1 1. □

The main problem with variable length codes is synchronization at the receiver. That is knowing where codewords begin and end as they arrive. (Unique decodability only guarantees that once the whole message is received, there is only one way of splitting it into codewords). One solution is to have a very distinct sequence of symbols, which is transmitted in between the transmission of each codeword. However, this adds extra symbols to the message, when the object is to decrease the message length. The usual method of maintaining synchronization is to ensure that none of the smaller codewords form the start of a longer codeword. Any code satisfying Definition 1.17 is called _instantaneously decodable_.

Definition 1.17 A code is called a _prefix code_ if no codeword is a prefix for any other. □

Example 1.3 If $C = \{01, 11, 011\}$ then this code is not a prefix code because the codeword 01 is a prefix of the codeword 011. However, the code $C = \{0, 10, 110\}$ is a prefix code. □

Lemma 1.8 *A prefix code is uniquely decodable.* □

Proof [McE], page 239. □

The _average wordlength_ of a variable length code is the average number of code symbols that are used to represent a source symbol. An _optimal_ variable length code is one that minimizes the average wordlength for a particular source. Given a source and its associated p_i's, then Huffmann codes, [McE], pages 243 - 248, are always optimal.

This section has assumed a noiseless channel. The next section will consider the problems that occur when errors may be present on the channel.

§1.7.3 Variable Length Codes and Noisy Channels

The problem here is that an error may cause a group of short codewords to appear as a long codeword and vice-versa. If the decoder makes an error in the length of the codeword it outputs, this is called a *synchronization error*. When the decoder looks at the next sequence of received symbols, having made a synchronization error, it is probably *out-of-synch*, because it is unlikely to be looking at the beginning of a codeword. The decoder will remain out-of-synch until some event occurs so that it aligns once again with the beginning of a codeword. This event may be, the occurrence of another error, outside intervention or the decoder may decode some out-of-synch symbols to a codeword of such a length that the decoder regains synchronization 'by itself'.

Work has been done on finding codes with good synchronization recovery properties. One good example of such codes is Titchener's T-codes, [Ti].

However, there appears to be little work done on error-correcting variable length codes. That is codes that correct data errors rather than just recovering from synchronization errors. One paper on this topic, [BS], has some nice theoretical results, but the conditions imposed on the channel (namely that the error ranges of codewords of different lengths be distinct) are unrealistic as far as practical implementation goes. Other work, [Ha], has been done using group theoretic ideas, but again this seems to have little practical relevance.

The purpose of Chapter 5 is to find a variable length code with some error-correcting capability, which has a shorter average wordlength than the block-length of a block code with comparable error-correcting capability. This code should also have a feasible means of implementation.

§1.8.1 Summary

This chapter has outlined the basic results of coding theory that are relevant to this thesis and has tried to demonstrate some of the aims of Coding Theory. The following chapters examine some work of others and then go on to tackle two major issues, interleaving and variable length error-correction coding. Interleaving is important for block codes and convolutional codes. As stated in [BPP], the world is becoming less Gaussian and more bursty, therefore overcoming bursts of errors has increasing importance for Coding Theory.

Error-correcting variable length codes seem to combine most effectively the two aims of Coding Theory, efficient and reliable communication.

Chapter 2

A STUDY OF PROFESSOR P.M.COHN'S DECODING SCHEME FOR CYCLIC CODES

§2.1.1 Introduction

Cyclic codes are usually decoded by an algorithm which, in some way, makes use of the syndrome of the received word. One such algorithm uses the standard array, as was described in §1.2.2. In an early version of a chapter for an algebraic text book, Cohn proposed a decoding scheme with an alternative method of choosing the row leaders for standard array decoding. The work described in this chapter examines in what way the different choice of row leaders affects the error correcting capabilities of standard array decoding. The conclusion reached is that Cohn's scheme gives much poorer performance in most practical situations. Professor Cohn was notified of these conclusions and has, as a result, withdrawn the decoding algorithm from the book.

More recently, Dr. R. Hill (Salford University) has also analysed Cohn's scheme and has derived even more conclusive arguments to expose its weakness. Hill's argument is also included.

§2.1.2 Notation

The notation of §1.2.3 is followed in this chapter, but is stated explicitly here for completeness. $\mathcal{C}$ is a cyclic code, over GF(q), of blocklength n and dimension k that can correct up to t errors.

$\mathcal{C}$ has generator polynomial $g(x)$ of degree $n-k$ and parity-check polynomial $h(x)$, such that

$$g(x)h(x) = x^n - 1$$

$c(x)$ is a polynomial of degree less than or equal to n whose coefficients form a codeword in $\mathcal{C}$.

$v(x)$ is a polynomial of degree less than or equal to n representing a received word.

$S(x)$ is the syndrome polynomial.

$a(x)$ is the message polynomial and has degree less than or equal to k.

$e(x)$ is the error polynomial and has degree less than or equal to n.

§2.2.1 Standard Array Decoding

This section considers standard array decoding written in polynomial notation.

Definition 2.1 The syndrome polynomial is given by

$$S(x) = v(x)h(x) \bmod (x^n - 1)$$

□

The dependence of the syndrome on the error pattern is now demonstrated in terms of the polynomial notation.

$$v(x) = c(x) + e(x)$$

Thus $S(x)$ may derived as follows

$$\begin{aligned} S(x) &= [c(x) + e(x)]h(x) \bmod (x^n - 1) \\ &= [c(x)h(x) + e(x)h(x)] \bmod (x^n - 1) \\ &= [a(x)g(x)h(x) + e(x)h(x)] \bmod (x^n - 1) \\ &= [h(x)e(x)] \bmod (x^n - 1) \end{aligned}$$

Thus the syndrome depends only on the error pattern and not on the particular codeword that was transmitted. Therefore the syndromes may be used to partition the set of all possible received words into cosets. A standard array is formed by taking the rows to be the cosets. The first row is the coset containing all the codewords, the all-zeros codeword being placed first in the row. For each of the remaining cosets one element is chosen to be the coset leader, this element is placed first in the row. The remaining elements in the coset are placed so that each one is the sum of the coset leader and the codeword at the head of the column in which it is placed. A received word is decoded to the codeword at the head of the column in which the received word is found. In practice this is done by calculating the syndrome of the received word and then subtracting from the word the coset leader associated with the calculated syndrome. Thus an error pattern is correctable if and only if it is a coset leader. Hence the way the coset leaders are chosen greatly affects the error-correcting capabilities of this method of decoding. The traditional way to choose the coset leaders is to take the polynomial whose coefficients have least weight, as in §1.2.2. Therefore a t-error-correcting code has all words of weight t or less as coset leaders (n.b. some words of weight greater than t may also appear as coset leaders).

§2.2.2 <u>Cohn's Scheme</u>

This scheme differs from the above only in the way the coset leaders are chosen. In this scheme a coset leader is taken to be the polynomial of least degree, rather than that of least weight. Now there exists at most one polynomial of degree less than $n - k = r$ in each coset, because any two elements in the same coset differ by a codeword, which has degree greater than or equal to r. There are q^k codewords and hence q^k elements in each coset. There are q^n possible received words and therefore $q^{n-k} = q^r$ cosets. There are q^r polynomials of degree less than r, thus the set of coset leaders is the set of polynomials of degree less than r.

It can be seen from the above, that Cohn's scheme corrects an error pattern if and only if it has zeros in the last k places.

§2.3.1 <u>Background Results</u>

This section introduces some results not related to syndrome decoding, but which are used to prove later results.

Define the function:

$$f(x) = (1 + x)^{n-k} - \sum_{i=0}^{t} \binom{n}{i} x^i$$

Consider the following results:

(1)

$$f(0) = 1^{n-k} - \binom{n}{0}$$
$$= 1 - 1$$
$$= 0$$

(2)

$$\frac{d}{dx}\sum_{i=o}^{t}\binom{n}{i}x^i = \sum_{i=o}^{t} i\binom{n}{i}x^{i-1}$$
$$= \sum_{i=1}^{t} n\binom{n-i}{i-1}x^{i-1}$$
$$= n\sum_{i=0}^{t-1}\binom{n-1}{i}x^i$$

(3)

$$f'(x) = (n-k)(1+x)^{n-k-1} - n\sum_{i=0}^{t-1}\binom{n-1}{i}x^i$$
$$\Rightarrow \quad f'(0) = (n-k) - n$$
$$= -k$$
$$\Rightarrow \quad f'(0) < 0$$

(4)

$$(1+x)\sum_{i=0}^{s}\binom{m}{i}x^i = 1 + \sum_{i=1}^{s}\left[\binom{m}{i} + \binom{m}{i-1}\right]x^i + \binom{m}{s}x^{s+1}$$
$$= \sum_{i=0}^{s}\binom{m+1}{i}x^i + \binom{m}{s}x^{s+1}$$

(5)

$$(1+x)\sum_{i=0}^{t-1}\binom{n-1}{i}x^i = \sum_{i=0}^{t-1}\binom{n}{i}x^i + \binom{n-1}{t-1}x^t$$
$$= \sum_{i=o}^{t}\binom{n}{i}x^i + \left[\binom{n-1}{t-1} - \binom{n}{t}\right]x^t$$
$$= \sum_{i=0}^{t}\binom{n}{i}x^i + \binom{n-1}{t}x^t$$

(6) Consider $f'(x)$ when $f(x) = 0$. From (3)

$$f'(x) = \frac{1}{1+x}\left[(n-k)(1+x)^{n-k} - n(1+x)\sum_{i=0}^{t-1}\binom{n-1}{i}x^i\right]$$
$$= \frac{1}{1+x}\left[(n-k)(1+x)^{n-k} - n\sum_{i=0}^{t}\binom{n}{i}x^i + n\binom{n-1}{t}x^t\right]$$
$$= \frac{1}{1+x}\left[-k(1+x)^{n-k} + n\binom{n-1}{t}x^t\right]$$

(7) Consider

$$h(x) = \frac{x^t}{(1+x)^{n-k}}$$

then

$$h'(x) = \frac{tx^{t-1}}{(1+x)^{n-k}} - (n-k)\frac{x^t}{(1+x)^{n-k+1}}$$
$$= \frac{x^{t-1}}{(1+x)^{n-k+1}}\left[t + tx - nx + kx\right]$$
$$= \frac{x^{t-1}}{(1+x)^{n-k+1}}\left[t - (n-k-t)\,x\right]$$

It can be seen that, starting from $x = 0$, $h(x)$ increases to a single maximum and then decreases to zero as $x \to \infty$.

(8) It has been shown that:

(i) $f(0) = 0$,

(ii) $f'(0) < 0$.

Now, because $n - k > t$, $f(x) \to \infty$ as $x \to \infty$. Hence $f(x)$ must cross the positive x-axis an odd number of times (and at least once). Suppose that $x_1 < x_2 < x_3$ are the first three values of x greater than zero for which $f(x) = 0$. Then it follows that:

$$f'(x_1) \geq 0, \quad f'(x_2) \leq 0, \quad f'(x_3) \geq 0.$$

That is

$$-k(1+x_1)^{n-k} + n\binom{n-1}{t}x_1^t \geq 0$$
$$-k(1+x_2)^{n-k} + n\binom{n-1}{t}x_2^t \leq 0$$
$$-k(1+x_3)^{n-k} + n\binom{n-1}{t}x_3^t \geq 0$$

This gives

$$\frac{x_1^t}{(1+x_1)^{n-k}} \geq \frac{k}{n\binom{n-1}{t}} \geq \frac{x_2^t}{(1+x_2)^{n-k}}$$

and

$$\frac{x_3^t}{(1+x_3)^{n-k}} \geq \frac{k}{n\binom{n-1}{t}}$$

However, (7) shows that this is impossible, hence there is only one point $x_1 > 0$ such that $f(x_1) = 0$. Therefore, the curve of $f(x)$ has the form shown in Figure 2.1.

§2.4.1 Probability of Correct Decoding

The probability of correct decoding is the probability that the codeword produced by the decoder is the codeword that was transmitted.

A comparison is now made between the probabilities of correct decoding of traditional standard array decoding and Cohn's scheme. Let

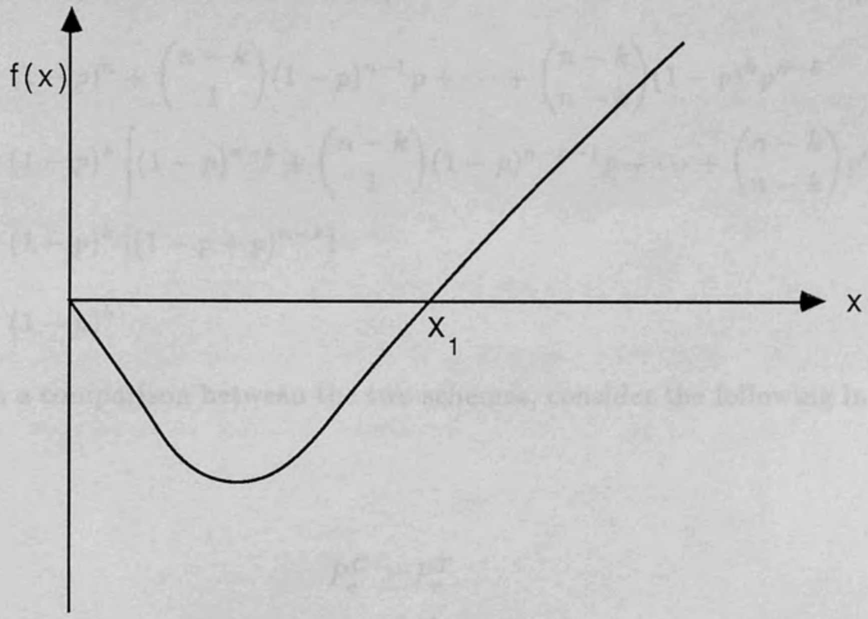


Figure 2.1

P_c^T denote the probability of correct decoding for the traditional scheme.

P_c^C denote the probability of correct decoding for Cohn's scheme.

Consider a Binary Symmetric Channel (BSC) with cross-over probability p, then

$$P_c^T \geq (1-p)^n + \binom{n}{1}(1-p)^{n-1}p + \cdots + \binom{n}{t}(1-p)^{n-t}p^t$$

The RHS of this inequality is the sum of the probability of occurrence of each error pattern of weight $\leq t$. (There is an inequality, rather than equality, because more error patterns than those of weight t may be corrected). P_c^C is given by the probability of an error pattern with all zeros in the last k places.

$$\begin{aligned}
P_c^C &= (1-p)^n + \binom{n-k}{1}(1-p)^{n-1}p + \cdots + \binom{n-k}{n-k}(1-p)^k p^{n-k} \\
&= (1-p)^k \left[(1-p)^{n-k} + \binom{n-k}{1}(1-p)^{n-k-1}p + \cdots + \binom{n-k}{n-k}p^{n-k}\right] \\
&= (1-p)^k \left[(1-p+p)^{n-k}\right] \\
&= (1-p)^k
\end{aligned}$$

To form a comparison between the two schemes, consider the following inequality:

$$\begin{aligned}
&P_c^C \geq P_c^T \qquad (2.1)\\
\Rightarrow \quad &(1-p)^k \geq \sum_{i=0}^{t} \binom{n}{i}(1-p)^{n-i}p^i
\end{aligned}$$

Determining the values of n, k and t for which this inequality is satisfied will give the parameters of possible codes for which Cohn's scheme is as good as or better than the traditional scheme. (n.b. The measure of 'goodness' has been taken to be the probability of correct decoding, this is not the only possibility.)

Set

$$x = \frac{p}{1-p}$$

then (1) becomes

$$(1+x)^{n-k} \geq \sum_{i=0}^{t} \binom{n}{i} x^i$$

Recall the function $f(x)$ of §2.3.1, then (2.1) is satisfied iff $f(x) \geq 0$. Thus, from Figure 2.1, the inequality is not satisfied for some range of x, $0 < x < x_1$ and then is always satisfied for $x \geq x_1$.

§2.5.1 Threshold Values for Cohn's Scheme

This section shows that when comparing P_c^C and P_c^T, it is not necessary to consider all values of p between [0,1].

Lemma 2.1 *If for some given values of n,k and t, $P_c^C < P_c^T$ for all $p \leq \frac{1}{2}$, then, for these values of n, k and t, it is always better to use the traditional scheme (in some cases with bit inversion).* □

Proof

Case 1: $p \leq \frac{1}{2}$ and no bit inversion.

By assumption $P_c^C < P_c^T$, therefore it is better to use the traditional scheme.

Case 2: $p > \frac{1}{2}$ and bit inversion.

Inverting every bit takes the probability of bit error from p to $q = 1 - p$, but $q \leq \frac{1}{2}$ and this reverts to Case 1.

Case 3: $p > \frac{1}{2}$ and no bit inversion.

The assumption does not rule out the situation that $P_c^C > P_c^T$ for $p = p_1 > \frac{1}{2}$. However P_c^C is strictly monotonic decreasing in the range $[0,1]$, therefore $P_c^C(p_1) < P_c^C(q_1)$, where $q_1 = 1 - p_1$. But $P_c^C(q_1) < P_c^T(q_1)$, by the assumption, therefore better can be acheived by using bit inversion.

Case 4: $p \leq \frac{1}{2}$ and bit inversion.

Probability of bit error again maps to q, in this case $q > \frac{1}{2}$. Therefore $P_c^C(q) > P_c^T(q)$ may occur, but, by similar reasoning to Case 3, it can be seem that this is not as good as using the traditional scheme and the original p. $\Box$

Thus, if bit inversion is possible, then to prove that the traditional scheme is better for all probabilities, p, it is sufficient to prove that $P_c^C < P_c^T$ for $p \leq \frac{1}{2}$

Combining Lemma 2.1 and §2.4.1 shows that $P_c^C < P_c^T$ for all probabilities, p, iff $f(1) < 0$ ($x = 1$ corresponds to $p = \frac{1}{2}$).

§2.5.2 Computer Results

The Pascal program of Appendix 1 finds and prints, for given n, k and t, the range of x, of width 0.01, in which $f(x)$ becomes positive, providing that this range occurs before $x = 1$ (i.e. $p = \frac{1}{2}$). Otherwise the program prints a * to denote that $P_c^C < P_c^T$ for all $P \leq \frac{1}{2}$ It does this for all values of n, k and t in the following ranges

$$3 \leq n \leq 100$$

$$\left[\frac{n}{2}\right] \leq k \leq n-2$$

$$1 \leq t \leq \left[\frac{n-k}{2}\right]$$

Examination of the results file showed that, for given n and k, mainly stars appeared when t was large, but ranges could be found when t was small. An exhaustive computer search through the results file showed that

<u>Result 2.2</u> *For any fixed n and k in the above ranges, $f(1) > 0$ only if*

$$t \leq \left[\frac{n-k}{4}\right] + 1$$

□

<u>Corollary 2.3</u> *For given n and k, and for t in the range*

$$\left[\frac{n-k}{4}\right] + 1 < t \leq \left[\frac{n-k}{2}\right]$$

it is always better to use the traditional scheme. □

For most practical coding schemes, only bit-error probalities in the range

$$0 \leq p \leq 0.1$$

are considered. If the above calculations are repeated replacing $x = 1$ with $x = 0.11 \Rightarrow p \simeq 0.1$, the following result is obtained.

<u>Result 2.4</u> *For any fixed n and k in the above ranges, $f(0.11) > 0$ only if*

$$t \leq \left[\frac{n-k}{16}\right] + 1$$

□

<u>Corollary 2.5</u> *For given n and k, for p in the range $0 \leq p < 0.1$, and for t in the range*

$$\left[\frac{n-k}{16}\right] + 1 < t \leq \left[\frac{n-k}{2}\right]$$

it is always better to use the traditional scheme. □

§2.6.1 Summary

Cohn's scheme and traditional syndrome decoding differ only in the way that the coset leaders are chosen. In Cohn's scheme this choice is made in a more mathematically succinct way than for the traditional scheme. However, it has been shown above that Cohn's scheme does not perform well unless t is much less than the Singleton bound. Certainly, for all practical values of p (that is $p \leq \frac{1}{2}$), only codes with very poor error-correcting ability perform better under Cohn's scheme. Also, there is no apparent advantage in the practical implementation of Cohn's scheme over the traditional one.

Hence, in all practical circumstances and for fixed n and k, a code could be chosen and decoded using the traditional scheme and perform better than it would be possible to achieve using Cohn's scheme and the same n and k.

Postscript

In written correspondence, Dr. R. Hill (Salford), has remarked that a stronger result than that presented here can be obtained from the following observations.

Section 2.2.2 showed that Cohn's scheme will decode correctly iff the last k bits of the codeword are received correctly. Hence, the probability of correct decoding for Cohn's scheme is

$$P_c^C = (1-p)^k$$

(as was obtained in a slightly more roundabout way in §2.4.1). This is just the probability of an uncoded k-bit message being received without error. Thus, Cohn's scheme is useless, because the same result can be achieved, with less time and decoding effort, by simply transmitting the message uncoded.

Dr. Hill has also shown that $P_c^T \geq P_c^C$ always. If it were possible for $P_c^T < P_c^C$, then this would imply that the use of a linear code (with nearest neighbour decoding) could give a lower probability of correct decoding than using no coding at all.

Observe that the worst way of adding redundancy to k-bit messages is to add $n-k$ zeros to every message vector. The coset leaders for this code are precisely the 2^{n-k} vectors having zeros in the last k places. Thus, the use of this code is equivalent to the use of Cohn's decoding scheme for any (n,k) code and hence,

$$P_c^T = (1-p)^k$$

for this code.

Consider the 'standard form' generator matrix

$$[A|I]$$

then the 2^{n-k} vectors having zeros in the last k positions are all in different cosets (since no two of them add up to a codeword), though they will not in general be the coset leaders. This leads straight away to

$$P_c^T \geq (1-p)^k$$

for any code (with equality iff the 2^{n-k} vectors with zeros in the last k places are all cosets leaders). Thus the result follows.

Chapter 3

A STUDY OF REED SOLOMON CODES, CHINESE REMAINDER THEOREM CODES AND CYCLOTOMIC SHORTENING

§3.1.1 Introduction

This chapter was motivated by the work of Dr. J.L. Dornstetter, [Do] and [Do,85], and its relationship with Reed Solomon codes and Chinese Remainder Theorem codes, [St] and [PW]. In [Do], Dornstetter describes (albeit briefly) a class of codes called Cyclotomically Shortened Reed Solomon codes (hereafter denoted as CSRS codes). This chapter reiterates the definitions of [Do] and establishes values for the dimension and blocklength of CSRS codes, together with a (best possible) lower bound for the minimum distance.

It can be shown (see [PW]) that Reed Solomon codes can be defined in terms of a particular class of Chinese Remainder Theorem codes (hereafter denoted as CRT codes). Here it is shown that CSRS codes are equivalent to a different (although related) class of CRT codes.

Dornstetter's patent application,[Do,85], is for a decoder for a class of CRT codes and it is implied in [Do] that this class of CRT codes is equivalent to CSRS codes, however Lemma 3.14 shows that this is not the case.

Lastly this chapter will consider the possibility of cyclotomically shortening Alternant codes. It is shown that although the shortening is possible, the resulting codes are not apparently useful.

Note that this chapter assumes an information source that produces symbols from $GF(q^m)$.

§3.1.2 Introduction to CRT codes

The remainder, $r_j(x)$, of a polynomial, $i(x)$, when divided by a polynomial, $m_j(x)$, gives little information about $i(x)$. However, given increasing numbers of residues of $i(x)$ modulo different polynomials, there comes a point when these residues determine $i(x)$ uniquely. The precise conditions for this will be given in Theorem 3.1. First some notation.

Let $i(x)$ be a polynomial of degree $\leq k-1$ over $GF(q^m)$

Let $m_j(x)$ be a polynomial of degree d_j over $GF(q^m)$, for $j = 0, 1, \ldots, n-1$.

Theorem 3.1 *The polynomial $i(x)$ can be reconstructed from the remainders*

$$r_j(x) \equiv i(x) \bmod m_j(x) \qquad\qquad j = 0, 1, \ldots, n-1,$$

provided the $m_j(x)$ are relatively prime in pairs and that

$$\sum_{j=0}^{n-1} d_j > k-1$$

□

Proof This is just the Chinese Remainder Theorem for polynomial rings. For a formal proof, see [PW]. □

Theorem 3.1 may be used to form a block code of dimension k and blocklength n. Blocklength is used a little loosely here, as, if the $m_j(x)$ do not all have the same degree, not all the n symbols will have the same size. The codes will be introduced by describing the method of encoding.

To encode the k information symbols (from $GF(q^m)$), write them as the coefficients of a polynomial $i(x)$ of degree $\leq k-1$. Select n polynomials, $m_j(x)$ that satisfy the requirements of Theorem 3.1 and find the residues of $i(x)$ modulo each of the $m_j(x)$. The codeword is formed by placing the coefficients of each residue in turn in a vector of length

$$n' = \sum_{j=0}^{n-1} d_j$$

over $GF(q^m)$. Thus the codeword contains n' elements of $GF(q^m)$, but the blocklength, n, is taken as the number of component residues.

Now consider what happens at the receiver. Suppose a codeword is transmitted and received with no errors. If the residue polynomials are reconstructed from the codeword, then Theorem 3.1 guarantees that the polynomial $i(x)$ can be recovered and hence the information symbols may be obtained. The codes formed in this way are called *Chinese Remainder Theorem* codes.

The requirements of Theorem 3.1 allow the recovery of the information in the absence of errors. However, if the $m_j(x)$ are chosen so that

$$\sum_{j=0}^{n-1} d_j \gg k-1$$

and $s > 1$ subsets of the $m_j(x)$ are such that the sum of the degrees of their elements is $> k-1$, then some error-correction may be performed. Suppose that one $GF(q^m)$ symbol is received in error, then one of the residues will also be wrong. If each of the s subsets is used separately to construct a polynomial of degree $\leq k-1$, then each of the subsets that contains $m_e(x)$, the $m_j(x)$ corresponding to the residue that is in error, will produce a polynomial. The remaining subsets (not containing $m_e(x)$) produce the correct polynomial. Now, if $m_e(x)$ appears in less than half the s subsets and a majority decision is used on which polynomial to accept, the error can be corrected.

Thus, if each $m_j(x)$ appears in less than half the s subsets, one error can be corrected. Note that this is one erroneous residue, up to all the $GF(q^m)$symbols that form this residue may be in error.

It follows that, if each $m_j(x)$ appears in less than one quarter of the subsets, then two erroneous residues may be corrected. Thus, the error correcting capability of the code can be increased by restricting each $m_j(x)$ to a smaller proportion of the subsets. It may also be possible to increase the error-correcting capabilities of the code by restricting the occurrences of pairs, triples etc. of the $m_j(x)$, but this is not the subject of this work.

A more formal definition of CRT codes is now given

Definition 3.1 A *Chinese Remainder Theorem (CRT) code* over $GF(q^m)$ of dimension k and blocklength n is formed by writing the k information symbols as the coefficients of a polynomial of degree $\leq k-1$. The codeword is formed from the n residues mod $m_j(x)$, where the $m_j(x)$ are polynomials of degree d_j over $GF(q^m)$ that are relatively prime in pairs and are such that

$$\sum_{j=1}^{n} d_j > k-1$$

□

An example of CRT codes is given in the next section, where it is shown that Reed Solomon codes can be defined in terms of CRT codes.

§3.1.3 Reed Solomon Codes

There are many ways of defining Reed Solomon codes, two of which were given in §1.3.3. A third definition can be made in terms of CRT codes. Reed Solomon codes are a subclass of CRT codes, formed by a particular choice of the $m_j(x)$.

Definition 3.2 A *t-error-correcting Reed Solomon code* over $GF(q^m)$, having blocklength n and dimension $n-2t$ is formed by taking the information symbols to be the coefficients of the polynomial $i(x)$ in the definition of a CRT code. The $m_j(x)$ are taken to be

$$m_j(x) = (x - \alpha^j)$$

That is, the Reed Solomon codeword is formed from the set of residues

$$r_j(x) \equiv i(x) \bmod (x - \alpha^j)$$

where α is a primitive n^{th} root of unity in $GF(q^m)$. □

The details of this definition are given in [PW], page 263. In this particular CRT code, all the $m_j(x)$ have degree 1 and thus $n = n'$.

This chapter will deal with Reed Solomon codes of blocklength $2^m - 1$ and will sometimes use the notation $2t = r$.

§3.1.4 Introduction to CSRS codes

Given a code, $\mathcal{C}$, it is possible to form a subcode, $\mathcal{S}$, by selecting those codewords of $\mathcal{C}$ which satisfy

$$f(c_0, c_1, \ldots, c_{2^m-2}) = 0$$

where f is some function. If f is carefully chosen, then the codewords of $\mathcal{S}$ satisfy an equivalence relation, r. This relation may be between groups of the c_i, if a knowledge of any particular group determines the remainder of the codeword. The choice of a specific f leads to the definition of CSRS codes. These codes are now introduced.

Let $\mathcal{C}$ be a $(2^m - 1, 2^m - 1 - r, r + 1)$ Reed Solomon code . Denote the codewords of $\mathcal{C}$ by

$$\mathbf{c} = (c_0, c_1, c_2, \ldots, c_{2^m-2})$$

Let $\mathcal{S}$ be a subset of $\mathcal{C}$ defined by the constraint

$$\mathbf{c} \in \mathcal{S} \quad \text{iff} \quad c_{2i} = c_i^2 \quad \forall i \in [0, \ldots, 2^m - 2] \tag{3.1}$$

where the indices are calculated modulo $2^m - 1$.

In the notation above, this gives $f(c_i, c_{2i}) = c_{2i} - c_i^2$ and the equivalence relation $c_i \; r \; c_{2i}$ Each codeword contained in $\mathcal{S}$ can be considered to have its elements partitioned into equivalence classes, called cosets, by the constraint (3.1), with any element in the coset determining the remainder of the coset. The term coset is used here because of the link between the equivalence classes and cyclotomic cosets, which will become apparent later. Because any element of a coset determines the remainder of the coset, the codewords of $\mathcal{S}$ could be shortened so as to contain just one element from each coset. This gives

<u>Definition 3.3</u> A <u>Cyclotomically Shortened Reed Solomon code</u> consists of the codewords from a $(2^m - 1, 2^m - 1 - r, r + 1)$ Reed Solomon code that satisfy constraint (3.1), shortened so as to consist of one element from each coset. □

<u>Example 3.1</u> Let $m = 4$ and $\mathcal{C}$ = RS$(15, 7, 9)$. Then cosets defined by the constraint (3.1) are as follows

$$\{c_0\}$$

$$\{c_1, c_2, c_4, c_8\}$$

$$\{c_3, c_6, c_9, c_{12}\}$$

$$\{c_5, c_{10}\}$$

$$\{c_7, c_{11}, c_{13}, c_{14}\}$$

So to form a CSRS code, C', the codewords in C which satisfy (3.1) are selected and then these are shortened by taking only the elements

$$c_0 c_1 c_3 c_5 c_7$$

Thus the codewords of C' consist of five $GF(16)$ symbols, that is twenty bits. □

§3.2.1 Preliminary Results

This section examines the properties of GFFT of vectors with certain constraints on their elements.

Lemma 3.2 *A vector over $GF(2^m)$ of length 2^m-1 that satisfies constraint (3.1) has a GFFT that consists entirely of zeros and ones.* □

Proof Refer to Theorem 1.7. The proof of this Theorem remains valid if $\mathbf{V}$ and $\mathbf{v}$ are interchanged and the GFFT is taken rather than its inverse. Using this second version and setting $q = 2$ yields the desired result. □

However, from Definition 1.15, each Reed Solomon codeword has r consecutive zeros in its GFFT. This gives

Lemma 3.3 *The GFFT of a codeword $\mathbf{c} \in S$ has r consecutive terms that are zero and the remaining $2^m - 1 - r$ terms can take the value either zero or one.* □

§3.3.1 Dimension of CSRS codes

The dimension of the CSRS code is determined by the number of Reed Solomon codewords that satisfy the constraint (3.1). From Definition 1.15 a $(2^m-1, 2^m-1-r, r+1)$ Reed Solomon code consists of <u>all</u> vectors of length 2^m-1 over $GF(2^m)$ whose GFFT has r specific consecutive elements equal to zero and the remaining elements members of $GF(2^m)$. Lemma 3.3 shows that the set S consists of <u>all</u> vectors of length 2^m-1 over $GF(2^m)$ whose GFFT has r specific consecutive elements equal to zero and the remaining elements contained in $GF(2)$. There exists a unique CSRS codeword for each element of S, therefore

<u>Lemma 3.4</u> *The dimension of a CSRS code, derived from a* $(2^m-1, 2^m-1-r, r+1)$ *Reed Solomon code, is* 2^m-1-r *over* $GF(2)$. □

Thus, if a GFFT method of encoding is used for the Reed Solomon code, the set S consists of all those codewords produced to binary input to the Reed Solomon encoder.

<u>Example 3.2</u> The CSRS code, C', of Example 3.1, has dimension 7 over $GF(2)$, i.e. the codewords of C' consist of the first, second, fourth, sixth and eighth elements of all RS(15,7,9) codewords produced by binary input to an appropriate GFFT encoder. □

§3.3.2 Blocklength of CSRS codes

The blocklength, N, of a CSRS code in terms of symbols from $GF(2^m)$ is determined by the number of cosets into which the constraint (3.1) partitions the integers modulo $2^m - 1$. Now the coset

$$\{i_j, i_j.2, i_j.2^2, \ldots\}$$

where $i_j \in \mathbf{Z}_{2^m-1}$ is called a <u>cyclotomic coset</u>. Hence the constraint (3.1) partitions the indices of the codeword elements into cyclotomic cosets. Thus, to find the blocklength of a CSRS code, the number of cyclotomic cosets of $GF(2^m)$ over $GF(2)$ should be calculated.

Any element in a coset can be used to determine the remainder of the coset, so let s be the coset representative for a given cyclotomic coset.

<u>*Definition 3.4*</u> The <u>minimum polynomial over $GF(2)$</u> of $\beta \in GF(2^m)$ is the polynomial of least degree over $GF(2)$ that has β as a root. □

The following argument uses results from [MWS] and finds an expression for the number of cyclotomic cosets of $GF(2^m)$ over $GF(2)$. This expression then determines the blocklength of a CSRS code.

<u>*Lemma 3.5*</u> *All the members of a cyclotomic coset have the same minimum polynomial.* □

<u>*Proof*</u> See Property M6, page 103 of [MWS]. □

Denote this polynomial by $M^{(s)}(x)$. Then

<u>*Lemma 3.6*</u>

$$x^{2^m-1} - 1 = \prod_s M^{(s)}(x)$$

□

<u>Proof</u> Put $p = 2$ into Property M7, page 105 of [MWS]. □

All minimum polynomials are irreducible (Property M1, page 99) and have degree $\leq m$ (Property M4, page 100).

<u>Lemma 3.7</u> *$x^{2^m} - x$ = the product of all irreducible polynomials over $GF(2)$, whose degree divides m.* □

<u>Proof</u> Put $p = 2$ in Theorem 10, page 107 of [MWS]. □

Let $I_2(m)$ be the number of polynomials of degree m, irreducible over $GF(2)$, then

<u>Lemma 3.8</u>

$$I_2(m) = \frac{1}{m} \sum_{e|m} \mu(e) 2^{m|e}$$

where

$$\mu(j) = \begin{cases} 1 & \textit{if } j = 1; \\ (-1)^r & \textit{if } j \textit{ is the product of } r \textit{ distinct primes;} \\ 0 & \textit{otherwise.} \end{cases}$$

□

<u>Proof</u> Put $q = 2$ in Theorem 5, page 115 of [MWS]. □

<u>Lemma 3.9</u> *The number of irreducible polynomials over $GF(2)$, whose degree divides m is*

$$\sum_{d|m} I_2(d)$$

□

<u>*Proof*</u> This follows directly from the definition of $I_2(m)$. □

The blocklength of a CSRS code can now be stated

<u>*Theorem 3.10*</u> *The blocklength, N, of a CSRS code, formed from a $(2^m - 1, 2^m - 1 - r, r+1)$ Reed Solomon code, is given by*

$$N = \sum_{d|m} I_2(d) - 1$$

□

<u>*Proof*</u> Notice that $x^{2^m} - x = x(x^{2^m - 1} - 1)$ and x is a polynomial irreducible over $GF(2)$, whose degree divides m. Hence, the number of irreducible polynomials whose degree divides m that make up the factors of $x^{2^m-1} - 1$ is

$$\sum_{d|m} I_2(d) - 1 \tag{3.2}$$

(from Lemma 3.9). Hence, the number of minimum polynomials is also given by (3.2) (using Lemma 3.6). Thus the result follows. □

The following example evaluates N for a specific code. Figure 3.1 gives the values of N for a range of m.

<u>*Example 3.3*</u> Consider again the code, C', of Example 3.1. In this case, $m = 4$ and thus, Theorem 3.10 gives

$$\begin{aligned} N &= \sum_{d|4} I_2(d) - 1 \\ &= I_2(1) + I_2(2) + I_2(4) - 1 \end{aligned}$$

Now

$$I_2(1) = \sum_{e|1} \mu(e) 2^{1|e} = 2\mu(1) = 2$$

$$I_2(2) = \frac{1}{2} \sum_{e|2} \mu(e) 2^{2|e} = \frac{1}{2}(4\mu(1) + 2\mu(2)) = \frac{1}{2}(4 - 2) = 1$$

$$I_2(4) = \frac{1}{4} \sum_{e|4} \mu(e) 2^{4|e} = \frac{1}{4}(16\mu(1) + 4\mu(2) + \mu(4)) = \frac{1}{4}(16 - 4 + 0) = 3$$

Hence

$$N = 2 + 1 + 3 - 1 = 5$$

as was shown in Example 3.1. □

Table of possible blocklengths

m	$\sum_{d \backslash m} I_2(d) - 1$
1	1
2	2
3	3
4	5
5	7
6	13
7	19
8	35

Figure 3.1

§3.3.3 Minimum Distance of CSRS codes

This section states a lower bound for the minimum distance of CSRS codes. This bound is derived by using the minimum distance, $r + 1$, of the Reed Solomon code which was used to form the CSRS code.

Lemma 3.11 *The minimum distance, D, of a CSRS code formed from a $(2^m - 1, 2^m - 1 - r, r + 1)$ Reed Solomon code is such that*

$$D \geq \left\lceil \frac{r+1}{m} \right\rceil$$

where $\lceil x \rceil$ denotes the least integer $\geq x$. □

Proof Consider the set S , which consists of all those Reed Solomon codewords that satisfy constraint (3.1). The minimum distance of S is at least $r+1$, as this is the minimum distance of the Reed Solomon code. Take $\mathrm{s}, \mathrm{s}' \in S$, then if

$$s_i = s_i'$$

$$\Rightarrow \quad s_i^2 = (s_i')^2$$

$$\Rightarrow \quad s_{2i} = s_{2i}'$$

Thus, for each pair of codewords, the elements of the same coset either all agree or all disagree. Now, between each pair of codewords in S , there must be sufficient cosets which disagree to give at least $r+1$ symbols that disagree. The minimum number of cosets in disagreement to give a fixed number of symbols in disagreement, occurs when it is the largest cosets that disagree. Thus,

$$\text{Minimum number of cosets that disagree } = \left\lceil \frac{r+1}{m} \right\rceil$$

Now the number of cosets that disagree between two codewords in S, determines the number of $GF(2^m)$ symbols that disagree in the resulting CSRS codeword. Hence

$$D \geq \left\lceil \frac{r+1}{m} \right\rceil$$

□

The following example shows that this bound is the best possible.

Example 3.4 Suppose that the RS(15,7,9) code is encoded using a GFFT decoder. This encoder takes the inverse GFFT of a vector in which the first eight places are zeros and the remaining seven places contain the information symbols. Suppose that the encoder uses the Galois field formed with $p(z) = z^4 + z + 1$ and $\alpha = z$ as the primitive element.

The code, C', of Example 3.1 is formed by cyclotomically shortening those Reed Solomon codewords formed from binary input. Now, binary input 0000000 gives the codeword 00000 and binary input 1111101 gives the codeword $0\alpha^3\alpha^3 10$. These codewords differ in 3 symbols. From Lemma 3.11, for the CSRS code in question

$$D \geq \left\lceil \frac{9}{4} \right\rceil = 3$$

thus, the code C' of Example 3.1 satisfies the bound with equality. □

§3.3.4 CSRS Codes as Binary Codes

It is possible to consider CSRS codes as binary codes. It has already been shown that CSRS codewords are derived from binary input to a GFFT Reed Solomon encoder. The $GF(2^m)$ symbols that make up a CSRS codeword can be written in binary notation. Hence, a CSRS code is an $(mN, 2^m - 1 - r)$ code over $GF(2)$. When considered over $GF(2)$, CSRS codes are linear. Also the minimum distance of the code in terms of bits is still bounded below by $\lceil \frac{r+1}{m} \rceil$, because each disagreement between a pair of $GF(2^m)$ symbols may be caused by a difference of only 1 bit, as the following example shows.

Example 3.5 The code, C', is a (20,7) code. Consider the binary expansions of the two codewords given in Example 3.4, they are

$$00000000000000000000$$

$$00001000100000010000$$

The distance between these words is 3, hence, even as a binary code, C' reaches the lower bound on the minimum distance. □

Notice that the shortest blocklength, single error-correcting, binary code of dimension 7, has blocklength 11. Thus, this CSRS code is a long way from the best that can be achieved.

Examples 3.4 and 3.5 demonstrate that the lower bound for the minimum distance of CSRS codes, either over $GF(2^m)$ or $GF(2)$, cannot be improved without further constraints on the codes.

§3.4.1 CSRS codes and CRT codes

It is now shown that there is an equivalence between CSRS codes and a particular class of CRT codes. For clarity in the proof of Theorem 3.11, this section will use slightly different notation from other sections. This new notation will now be introduced.

Let the integers modulo $2^m - 1$ be partitioned into N cyclotomic cosets of the form

$$C_j = \{i_j, i_j.2, i_j.2^2, \ldots\} \qquad\qquad j = 0, 1, \ldots, N-1.$$

Let α be a primitive element of $GF(2^m)$ and define

$$A_j = \{\alpha^c \mid c \in C_j\}$$

Let $M_j(x)$ be the minimum polynomial associated with the coset C_j, then

$$M_j(x) = \prod_{c \in C_j} (x - \alpha^c)$$

and $M_j(x)$ is the minimum polynomial $\forall\, \beta \in A_j$.

The following, alternative definition of CSRS codes uses the CRT code definition of Reed Solomon codes.

Definition 3.5 A _Cyclotomically Shortened Reed Solomon code_, C' may be defined as follows :

Select a representative $\alpha_j \in A_j$ for each $j \in [0, \ldots, N-1]$. Use the k information bits to form the coefficients of a polynomial, $i(x)$, of degree $\leq k - 1$. The codeword is the vector $(c_0, \ldots, c_{N-1})$, over $GF(2^m)$, where

$$c_j = i(x) \bmod (x - \alpha_j) = i(\alpha_j)$$

□

The following example gives another CRT code, using the $M_j(x)$ as the $m_j(x)$. This code is then used in Theorem 3.11 to show that CSRS codes can be defined in terms of CRT codes.

Example 3.6 The CRT code C'' is formed by encoding the binary polynomial $i(x)$ (of degree $\leq k-1$) to the following sequence of elements of $GF(2)[x]$

$$(c_0(x), \ldots, c_{N-1}(x))$$

where

$$c_j(x) = i(x) \bmod M_j(x)$$

$\square$

Theorem 3.12 *The codes C' and C'' are equivalent codes.* $\square$

Proof Define the mapping

$$f : C'' \longrightarrow C'$$

by

$$f(c_0(x), \ldots, c_{N-1}(x)) = (c_0(\alpha_0), \ldots, c_{N-1}(\alpha_n))$$

This map is well defined because $\exists\, i(x)$ with

$$i(x) = q(x)M_j(x) + c_j(x)$$

by definition of C''. Hence

$$i(x) \bmod (x - \alpha_j) = i(\alpha_j)$$

$$= c_j(\alpha_j)$$

because $M_j(\alpha_j) = 0$.

This map, f, is onto because given $i(x)$ such that

$$c_j = i(x) \bmod (x - \alpha_j) = i(\alpha_j)$$

set

$$i(x) = q(x)M_j(x) + c_j(x)$$

then

$$c_j = i(\alpha_j) = c(\alpha_j)$$

f is a $GF(2)$ homomorphism because

$$\begin{aligned} f((c_0(x),\ldots,c_{N-1}(x)) &+ (c'_0(x),\ldots,c'_{N-1}(x))) \\ &= f(c_0(x)+c'_0(x),\ldots,c_{N-1}(x)+c'_{N-1}(x)) \\ &= (c_0(\alpha_0)+c'_0(\alpha_1),\ldots,c_{N-1}(\alpha_n)+c'_{N-1}(\alpha_n)) \\ &= (c_0(\alpha_1),\ldots,c_{N-1}(\alpha_n)) + (c'_0(\alpha_0),\ldots,c'_{N-1}(\alpha_n)) \\ &= f(c_0(x),\ldots,c_{N-1}(x)) + f(c'_0(x),\ldots,c'_{N-1}(x)) \end{aligned}$$

Therefore, to prove that f is an isomorphism, it remains only to show that f is one-to-one. Suppose that $c_j(\alpha_j) = c'_j(\alpha_j)$ then this implies that α_j is a root of

$$c_j(x) - c'_j(x) \in GF(2)[x]$$

But $M_j(x)$ is the minimum polynomial of α_j, so

$$c_j(x) - c'_j(x) = 0 \bmod M_j(x)$$

$$\Rightarrow c_j(x) = c'_j(x)$$

because both $c_j(x)$ and $c'_j(x)$ have degree less than that of $M_j(x)$. Thus, the codes are equivalent. □

The following example gives a specific case of this mapping.

Example 3.7 Take $m = 4$, then the five C_j are

$$C_0 = \{0\}$$

$$C_1 = \{1, 2, 4, 8\}$$

$$C_2 = \{3, 6, 9, 12\}$$

$$C_3 = \{5, 10\}$$

$$C_4 = \{7, 11, 13, 14\}$$

Suppose that $GF(2^4)$ is generated using $p(z) = z^4 + z + 1$ and $\alpha = z$ is a primitive element. Then the A_j are

$$A_0 = \{\alpha^0\}$$

$$A_1 = \{\alpha^1, \alpha^2, \alpha^4, \alpha^8\}$$

$$A_2 = \{\alpha^3, \alpha^6, \alpha^9, \alpha^{12}\}$$

$$A_3 = \{\alpha^5, \alpha^{10}\}$$

$$A_4 = \{\alpha^7, \alpha^{11}, \alpha^{13}, \alpha^{14}\}$$

The coset representatives, α_j, for each A_j will be taken as the first element in the coset. Thus, the minimum polynomials are given by (remembering that $+1 = -1$ over $GF(2)$)

$$\begin{aligned}
M_0(x) &= (x - 1) \\
M_1(x) &= (x - \alpha)(x - \alpha^2)(x - \alpha^4)(x - \alpha^8) \\
&= x^4 + x + 1 \\
M_2(x) &= (x - \alpha^3)(x - \alpha^6)(x - \alpha^9)(x - \alpha^{12}) \\
&= x^4 + x^3 + x^2 + x + 1 \\
M_3(x) &= (x - \alpha^5)(x - \alpha^{10}) \\
&= x^2 + x + 1
\end{aligned}$$

$$M_4(x) = (x-\alpha^7)(x-\alpha^{11})(x-\alpha^{13})(x-\alpha^{14})$$

$$= x^4 + x^3 + 1$$

Take $i(x) = x^6 + x^4 + x^2 + 1$ as the information polynomial to be encoded. From Definition 3.5, it can be seen that the corresponding codeword in C' is given by

$$(i(\alpha^0), i(\alpha^1), i(\alpha^3), i(\alpha^5), i(\alpha^7)) = (0, \alpha^9, \alpha^9, 1, \alpha^9)$$

written in binary notation, this becomes

$$(00001010101000011010)$$

Now look at the code C''

$$c_0(x) = x^6 + x^4 + x^2 + 1 \bmod (x-1)$$

$$= 0$$

$$c_1(x) = x^6 + x^4 + x^2 + 1 \bmod (x^4 + x + 1)$$

$$= x^3 + x$$

$$c_2(x) = x^6 + x^4 + x^2 + 1 \bmod (x^4 + x^3 + x^2 + x + 1)$$

$$= x^3$$

$$c_3(x) = x^6 + x^4 + x^2 + 1 \bmod (x^2 + x + 1)$$

$$= 1$$

$$c_4(x) = x^6 + x^4 + x^2 + 1 \bmod (x^4 + x^3 + 1)$$

$$= x + 1$$

The equivalence between the two codewords comes by evaluating $c_i(\alpha_i)$, which gives

$$(c_0(\alpha^0), c_1(\alpha^1), c_2(\alpha^3), c_3(\alpha^5), c_4(\alpha^7)) = (0, \alpha^9, \alpha^9, 1, \alpha^9)$$

A similar routine may be followed for all information polynomials. □

§3.5.1 Encoding and Decoding

This section will look at encoders and decoders for the three classes of codes mentioned above.

There are well-known encoders and decoders for Reed Solomon codes, see [Bl], [PW] or [MWS], and these will not be described here.

Turning attention to CRT codes, the general method of encoding CRT codes was implicit in their definition (see Definition 3.1). The general decoding method for CRT codes was outlined in Section 3.1.2 and is stated explicitly here.

Decoding Scheme for CRT codes

Take the set of $m_j(x)$ that was used to form the codewords, form all distinct subsets of this set such that the sum of the degrees of the elements of the set is $> k-1$. Each $m_j(x)$ is associated with a position in the codeword, i.e. the position in which the residue $i(x) \bmod m_j(x)$ is placed. For each received word, $i(x)$ is calculated using each of the distinct subsets in turn and the symbols in the received word in the associated positions of the $m_j(x)$ of the subset in question. The received word is decoded to the coefficients of the $i(x)$ which occurs most often, some predefined choice can be made in the event of a tie. □

The problem with this decoding scheme (as with most schemes that involve the calculation of several possible decodings followed by some form of comparison) is that for most practical situations any implementation will take too long to run. To the writer's knowledge there are no implementations of encoders and decoders for arbitrary CRT codes. However, more feasible encoders/ decoders have been developed for certain specific classes of CRT codes (e.g. Reed Solomon codes and the codes to be described in Section 3.6.1).

Lastly, consideration is given to CSRS codes. There is one obvious method of encoding and decoding CSRS codes, that is to use the encoder and decoder for the Reed Solomon code from which the CSRS code was derived (the base Reed Solomon code).

Encoding and Decoding scheme for CSRS codes

The encoding scheme is as follows. CSRS codes encode binary information, so take the k information bits and write each of them as a $GF(2^m)$ symbol. Encode these $GF(2^m)$ symbols using a GFFT encoder for the base Reed Solomon code. Cyclotomically shorten the resulting Reed Solomon codeword using constraint (3.1) and transmit the resulting CSRS codeword. (It was shown in §3.3.1 that binary input guaranteed that a Reed Solomon codeword would satisfy constraint (3.1) and can therefore be shortened).

Decoding is then performed by lengthening the received word, using constraint (3.1), and then using a GFFT decoder for the base Reed Solomon code. It would be necessary to adapt a standard Reed Solomon decoder so that it could decode only to those codewords that satisfy constraint (3.1). □

The major problem with this scheme is that it introduces severe error propagation. This is because any channel error that causes one received symbol to be in error may produce up to m symbols in error in the word that enters the Reed

Solomon decoder. The number of additional errors caused depends on the size of the cyclotomic coset in which the error occurred. It is clear that this scale of error propagation will seriously degrade the performance of the system.

The question of an encoder and decoder for CSRS codes will be returned to at the end of Section 3.6.2.

§3.6.1 Dornstetter Codes

In [Do,85], Dornstetter describes an encoder and decoder for a family of codes, which will be refered to as Dornstetter codes. These codes are defined as follows

Definition 3.6 A Dornstetter code, C , of blocklength n and dimension k, has codewords that consist of the residues

$$c_j = i(x) \bmod P_j(x) \qquad\qquad j = 0, 1, 2, \ldots, n-1.$$

where $i(x)$ is a binary polynomial of degree $\leq k-1$ whose coefficients are the k information bits. The $P_j(x)$ consist of all the irreducible factors of $x^{2^m-1} - 1$ over $GF(2)$ of degree m and any polynomials of degree m that can be formed from the remaining factors of $x^{2^m-1} - 1$ such that the $P_j(x)$ are all mutually coprime in pairs. Note that the blocklength, n, of the code is determined by the number of $P_j(x)$ that can be formed. □

Example 3.8 of the next section gives a specific example of a Dornstetter code.

§3.6.2 <u>CSRS codes and Dornstetter codes</u>

The following Lemma compares Dornstetter codes with CSRS codes by using the equivalent CRT codes. The Lemma is followed by a specific example, to clarify the distinction between the two sets of codes.

<u>Lemma 3.13</u> *CSRS codes and Dornstetter codes do not define the same set of codewords.* □

<u>Proof</u> Compare the code of Example 3.2, which Theorem 3.11 showed to be equivalent to a CSRS code and the Dornstetter code of Definition 3.6. The two sets of $m_j(x)$ used in the definitions are different. The set of $m_j(x)$ in Definition 3.6 takes the set of $m_j(x)$ of Example 3.2, leaves some of them unchanged, takes the product of others and discards the remainder so that all the $m_j(x)$ of Definition 3.6 have degree m. □

In the following example a larger value of m is taken than for previous examples, this is necessary to demonstrate clearly the distinction between the two sets of codewords.

<u>Example 3.8</u> Let $m = 8$, then the base code is the Reed Solomon $(255, 255-r, r+1)$ code. Theorem 3.10 gives that the blocklength, N, of the derived CSRS code is given by

$$\begin{aligned} N &= \sum_{d|8} I_2(d) - 1 \\ &= I_2(1) + I_2(2) + I_2(4) + I_2(8) - 1 \end{aligned}$$

Recall the values of $I_2(1) - I_2(4)$ from Example 3.3.

$$\begin{aligned} I_2(8) &= \frac{1}{8}\sum_{e|8} \mu(e) 2^{8|e} = \frac{1}{8}(256\mu(1) + 16\mu(2) + 4\mu(4) + \mu(8)) \\ &= \frac{1}{8}(256 - 16 + 0 + 0) = 30 \end{aligned}$$

Therefore

$$N = 2 + 1 + 3 + 30 - 1 = 35$$

So, the blocklength of the derived CSRS code (over $GF(256)$) is 35.

Now consider the Dornstetter code for $m = 8$. The irreducible factors of $x^{255} - 1$ can be found in [PW], Appendix C. There are 30 of degree 8, 3 of degree 4, 1 of degree 2 and 1 of degree 1. (Notice that these are the minimum polynomials of the cyclotomic cosets as defined in the CSRS code). However, any 2 of the degree 4 polynomials may be combined to give another degree 8 polynomial that is mutually prime to the original 30 degree 8 polynomials, but no more mutually prime degree 8 polynomials can be created. Therefore, the Dornstetter code with $m = 8$ has blocklength 31. □

The Patent Application [Do,85] is essentially for a decoder, however it is unclear from Dornstetter's work whether this decoder is for CSRS codes or Dornstetter codes (or even both). Dornstetter appears to make no distinction between the two sets of codes. However, Lemma 3.14 shows that they are in fact different. The decoder in [Do,85] uses an adaptation of the Berlekamp Massey algorithm and works for a particular class of CRT codes because of 'the pertinent choice of the $m_j(x)$'. It is therefore unlikely that the decoder can be adapted to decode many (if any) other classes of CRT codes.

§3.7.1 Alternant Codes

These codes are very closely related to Reed Solomon codes and this final section will consider the application of the ideas of the previous sections to Alternant codes. Firstly, Alternant codes are defined.

Definition 3.7 An *Alternant Code*, $\Omega(C, h)$, is defined as follows. Let $h = (h_0, h_1, \ldots, h_{n-1})$ be a vector in $GF(q^m)^n$, which has every coordinate distinct from zero. Then Ω consists of all $c = (c_0, c_1, \ldots, c_{n-1}) \in GF(q)^n$ which satisfy

$$ch = (c_0h_0, c_1h_1, \ldots, c_{n-1}h_{n-1}) \in C$$

where C is a Reed Solomon code over $GF(q^m)$, as defined in Definition 1.15. □

This definition may be expressed in terms of Galois-Field Fourier Transforms. Let $H = (H_0, H_1, \ldots, H_{n-1})$ be the GFFT of h and $C = (C_0, C_1, \ldots, C_{n-1})$ the GFFT of c. Then Ω may be defined as the set of all words whose GFFT, $(C_0, C_1, \ldots, C_{n-1})$ over $GF(q^m)$ satisfies

$$(i) \sum_{k=0}^{n-1} C_k H_{j-k} = 0 \qquad j = j_0, \ldots, j_0 + 2t - 1$$

$$(ii) C_{(kq)} = (C_k)^q$$

(i) ensures that ch is a Reed Solomon codeword and (ii) ensures that c is over $GF(q)$ (refer to Theorem 1.7)

Consideration is now given to the possiblity of cyclotomically shortening Alternant codes. The codewords of Ω are over $GF(q)$, where q is a power of 2. If constraint (3.1) is applied to an Alternant code then it partitions the elements of a codeword into cyclotomic cosets and the codewords that satisfy constraint (3.1) may be shortened in the same manner as before. Note that the blocklength of the Alternant code is the same as the Reed Solomon code, C , which is used in its formation. Thus, a Cyclotomically Shortened Alternant (CSA) code, will have the same blocklength as the CSRS code obtain from C . More formally

<u>Theorem 3.14</u> *The blocklength, n, of a CSA code derived from an Alternant code of blocklength $2^m - 1$ is given by*

$$n = \sum_{d|m} I_2(d) - 1$$

where

$$I_2(d) = \frac{1}{d} \sum_{e|d} \mu(e) 2^{d|e}$$

and where

$$\mu(e) = \begin{cases} 1 & \text{if } j = 1; \\ (-1)^r & \text{if } j \text{ is the product of } r \text{ distinct primes;} \\ 0 & \text{otherwise.} \end{cases}$$

□

For the dimension of a CSA code, the GFFT definition of the codes is used. The dimension, k, of an Alternant code is not known precisely in general, but it is known to satisfy the following inequality.

<u>Lemma 3.15</u> *The dimension, k, of an Alternant code formed using a Reed Solomon code of designed distance $2t + 1$ is such that*

$$k \geq q^m - 1 - 2tm$$

(Note the dimension is over $GF(q)$). □

Proof   See [Bl], page 230. □

By the same argument used for CSRS codes the dimension of a CSA code is given by

Lemma 3.16   *The dimension of a CSA code, derived from an Alternant code of dimension k over $GF(q)$, is also k, but over $GF(2)$.* □

The minimum distance of a CSA code is bounded below in the same way as for CSRS codes.

Lemma 3.17   *The minimum distance, D, of a CSA code formed from an Alternant code with minimum distance d satisfies*

$$D \geq \left\lceil \frac{d}{m} \right\rceil$$

. □

Proof   As for CSRS codes. □

§3.8.1 Summary

This study has introduced Chinese Remainder Theorem codes and several methods of defining Reed Solomon codes (one in terms of CRT codes). Cyclotomically Shortened Reed Solomon codes have been described and expressions given for their dimension and blocklength. A lower bound for the minimum distance of CSRS codes has been given, together with a code that satisfies the lower bound with equality. The equivalence between CSRS codes and a particular class of CRT codes has been demonstrated. Another class of CRT codes, called Dornstetter codes has been defined and it has been shown that CSRS code and Dornstetter codes are non-identical.

Encoding and decoding schemes have been examined for the above mentioned codes and the confusion over Dornstetter's Patent Application discussed.

Lastly, the possibility of cyclotomically shortening Alternant codes has been examined. Whilst it is obviously possible to cyclotomically shorten any code formed over $GF(2^m)$, there is no apparent gain in general, unless the resulting codes are equivalent to a class of CRT codes and Dornstetter's decoder can be adapted for use with them.

Chapter 4

INTERLEAVING FOR CONVOLUTIONAL CODING

§4.1.1 Introduction

The concept of interleaving using an $n \times m$ array was introduced in §1.6.1. The situation considered here is that when the maximum burst length exceeds m. This problem was posed by Mr. E. Jones of Inmarsat at the 4th RSRE coding meeting, Manchester, May 1987. Inmarsat (International Maritime Satellite Organisation) were considering the requirements of a mobile satcoms operator. A part of the communications system involved interleaving using an $n \times m$ array, data being fed into the columns and transmitted row-wise. The nature of the system was such that it was not feasible to make the row length, m, greater than the maximum expected burst length. Inmarsat considered sending the rows of the array in non-sequential order. They had found, ad hoc, the order in which to send the rows of an array with $n = 64$ to achieve the maximum spreading of errors for their particular channel conditions. The ad hoc approach was adopted because they could find no theoretical results for the problem. The purpose of this chapter is to rectify this situation, by providing a general solution to the problem. A more precise description of the problem is now given.

Suppose that the maximum length burst of errors is such that it can affect at most a rows (either completely or in part). Then, to minimize the effect of such a burst of errors, rows that are within a of each other in the array (that is rows that are close enough together to be affected by the same burst, if the rows were transmitted in sequential order) should be transmitted as far apart as possible. Although this does not change the number of errors in each column, it does space them throughout the whole column rather than them occurring in one solid block.

Now, if the interleaving is being performed on data encoded with a block code whose blocklength is greater than or equal to n, the spacing of the errors has no effect because most block codes can correct a fixed number of errors irrespective of whether they occur spaced out or in a block. However, if the blocklength of the code is less than n or convolutional coding is being used, this spacing of the errors can improve the performance of the system. In the block coding case the improvement occurs because the number of errors the burst causes in a column may be beyond the error-correcting capabilties of a single codeword, but the spacing could spread them over two (or possibly more) codewords, each codeword being able to correct the reduced number of errors. In the case of convolutional coding, it is well known that convolutional codes perform significantly better in the presence of random errors than they do in the presence of burst errors.

§4.1.2 Objectives

This chapter has two main objectives. Firstly, to find the optimal scheme with which to transmit the rows of the array so that any two rows that are within a of each other are sent as far apart as possible. Secondly, to give some recommendations about the usage of this optimal scheme.

§4.1.3 Notation

Let n be the number of rows in the array.

Let a be the maximum number of rows that can be affected (either totally or in part) by a single burst of errors.

Let the rows of the array be labelled $0, 1, \ldots, n-1$. (Note: a row will always be identified by its original position in the array.)

Let $n = \alpha a + b$, where $\alpha, b \in \mathbf{Z}$ and $\alpha > 0, 0 \leq b < a$.

Let t_{ij} be the number of rows transmitted between the transmission of row i and row j. For example, if the following sequence of rows is transmitted

$$0\;3\;6\;9\;2\;5\;8\;1\;4\;7$$

then $t_{35} = 3$.

Two rows i and j are within a of each other if and only if

$$\mid i-j \mid \leq a-1$$

Let $t = \min\limits_{|i-j| \leq a-1} t_{ij}$. That is t is the minimum number of rows transmitted between any two rows that are within a of each other in the array.

The first objective is therefore to find a scheme that maximizes t. But first a simpler problem will be considered.

§4.2.1 A Simpler Problem

Consider the rows of the array to be placed in the following cosets.

$$C_0 = \{0, 1, 2, \cdots, a-1\}$$

$$C_1 = \{a, a+1, a+2, \cdots, 2a-1\}$$

$$C_2 = \{2a, 2a+1, 2a+2, \cdots, 3a-1\}$$

$$\vdots$$

$$C_{\alpha-1} = \{(\alpha-1)a, (\alpha-1)a+1, \cdots, \alpha a-1\}$$

$$C_\alpha = \{\alpha a, \alpha a+1, \cdots, \alpha a+b-1\}$$

Cosets $C_0, C_1, \ldots, C_{\alpha-1}$ have size α, coset C_α has size b.

Notice that each element of a coset is within a of every other element in that coset and that some elements within different cosets are also within a of each other. Now consider the simpler problem of transmitting elements of the same coset as far apart as possible. Let t' be the minimum number of elements transmitted between the transmission of two elements from the same coset. Theorem 4.1 gives the maximum values of t' and a scheme which can achieve these values.

Theorem 4.1 *If the cosets $C_0, C_1, \ldots, C_\alpha$ are as defined above, then the maximum value for t' is as follows:*

$$\text{If } b = a - 1 \text{ then } \max t' = \alpha$$

$$\text{else } \max t' = \alpha - 1$$

□

Proof The optimal scheme for transmitting elements from the cosets so as to transmit elements in the same coset as far apart as possible is the following: "Starting with coset C_0, cycle through the cosets in numerical order, sending one element (which has not already been sent) from each coset in turn." Thus, between the transmission of two elements from one coset, exactly one element is transmitted from each of the other cosets. This is the optimal scheme because to increase the number of elements transmitted between two elements, c_{i1} and c_{i2}, from coset C_i, an extra element would have to be sent from some coset C_j and this would decrease the number of elements transmitted between two elements from the coset C_j. (There could be at most 1 element from each of $\alpha - 1$ cosets, rather than α cosets).

To determine t' for this scheme it is necessary to consider two cases:

Case 1: $b = a - 1$.

In this case $\exists$ α cosets of size a and 1 coset of size $a-1$. During the first $a-1$ cycles through the cosets, an element (which has not already been transmitted) can be chosen from each of the cosets (because there are at least $a - 1$ elements in each coset). During the last cycle it is not possible to send an element from coset C_α, because there are none left.

However, because the first cycle starts with the coset C_0, the coset C_α is always the last in a cycle. Thus the sequence of transmitted elements is just 1 element 'short' at the very end of the sequence. Now there are $\alpha + 1$ cosets, therefore there are α elements transmitted between the transmission of two elements from the same coset (the fact that the sequence is one element short at the end does not affect the distance between the previous elements). Therefore, in this case, $t' = \alpha$.

Case 2: $b \leq a - 2$.

In this case the coset C_α will 'run out' of elements before the last cycle, so some cycles will contain only α elements. Therefore, in this case, $t' = \alpha - 1$.

Clearly, t' for the optimal scheme is equivalent to $\max t'$. □

Corollary 4.2 *If $b = a - 1$ then $t \leq \alpha$ otherwise $b \leq a - 2$ and $t \leq \alpha - 1$.* □

Proof This problem tries to transmit some elements that are within a of each other as far apart as possible, but not all such elements. Therefore the constraints on this problem are weaker than those on the problem of §4.1.2 and so $\max t'$ provides an upper bound for the maximum value of t. Thus t is bounded above as shown. □

§4.2.2 Discussion of the Case b = a − 1

This section shows that the upperbound for t when $b = a - 1$ can be reduced from α to $\alpha - 1$. The full interleaving problem of § 4.1.2 is considered once again.

Theorem 4.3 *If for $n = \alpha a + b$ there is a scheme with $t = \alpha$, then there exists a scheme for $n' = (\alpha - 1)a + b$ with $t = \alpha - 1$.* □

Proof Take the scheme for n and remove the rows labelled with the a highest numbers, i.e. if $n = 7.3 + 2 = 23$ and the scheme is described as a sequence of the rows in the order in which they are to be transmitted (e.g. 0 3 6 9 12 15 18 21 1 4 7 10 13 16 19 22 2 5 8 11 14 17 20) then remove from this sequence the numbers 20, 21 and 22 (to give 0 3 6 9 12 15 18 1 4 7 10 13 16 19 2 5 8 11 14 17). This new sequence then has $t = \alpha - 1$. The reason for this is as follows:

Let the numbers to be removed form the set S , then each element of S is within a of every other element of S, therefore in the original scheme, each element of S must be separated by at least α terms from any other element of S . Now consider two rows i and j, not in S , such that $| i - j | \leq a - 1$, in the original scheme they are separated by at least α terms and up to a of these terms could be contained in S . Suppose one term between i and j was in S , then, when this term is removed, i and j are separated by at least $\alpha - 1$ terms. Now suppose that two terms between i and j were contained in S, then, when these two terms are removed, the number of terms between i and j decreases by two. However, the two terms in S were separated by at least α terms, therefore there must have been at least $\alpha + 2$ terms between i and j in the original sequence. Thus, on the removal of two terms, they are still at least α apart. In general, if $x \geq 2$ terms between i and j were from S, they were each separated from every other by at least α terms, therefore there must have been at least $(x-1)\alpha + x$ terms between i and j. Thus, on the removal of x terms, i and j are still separated by at least $(x-1)\alpha \geq \alpha$ terms.

Thus, in all possible cases, i and j are still separated by at least $\alpha - 1$ terms.

So, the sequence, which forms a scheme with $t = \alpha$ for $n = \alpha a + b$, with the a highest numbers removed forms a scheme with $t = \alpha - 1$ for $n' = (\alpha - 1)a + b$.

□

<u>Corollary 4.4</u> *If for $n = \alpha a + (a-1) \exists$ a scheme with $t = \alpha$ (i.e. acheiving the upper bound in the corollary to Theorem 4.1) then $\exists$ a scheme for $n' = (\alpha-1)a + (a-1)$ with $t = \alpha - 1$ (also achieving the upper bound of the corollary).*

□

<u>*Proof*</u> Substitute $b = a - 1$ in Theorem 4.3. □

<u>Lemma 4.5</u> *There is no scheme with $t = 1$ for $n = 2a - 1$.* □

<u>*Proof*</u> The row labelled a is within a of all the other rows in the array (there are $a - 1$ rows each side of it). Therefore, wherever the row labelled a is placed in the new scheme, it has to have at least one other row placed next to it, that is a row that is within a of it must be placed next to it, hence $t = 0$. Therefore there is no scheme with $t = 1$. □

Theorem 4.6 *If $b = a-1$ then the upper bound $t = \alpha$ can never be achieved.*

$\square$

Proof Corollary 4.4 shows that if there exists a scheme achieving the upper bound for $n = \alpha a + (a-1)$ then there exists a scheme achieving the upper bound for $n' = (\alpha - 1)a + (a-1)$. The contrapositive of this statement shows that if there does not exist a scheme that achieves the upper bound for $n' = (\alpha-1)a + (a-1)$ then there does not exist a scheme which achieves the upper bound for $n = \alpha a + (a-1)$. Lemma 4.5 shows that there does not exist a scheme achieving the upper bound for $n = 2a - 1 = a + (a-1)$. Therefore, by induction, there does not exist a scheme achieving the upper bound for any value of α with $b = a - 1$. $\square$

Corollary 4.7 *For all values of b, $t \leq \alpha - 1$.* $\square$

Proof Combine Corollary 4.2 and Theorem 4.6. $\square$

§4.3.1 An Optimal Scheme

Corollary 4.7 shows that $t \leq \alpha - 1$ for all values of n, a and b, therefore, if a scheme can be found that achieves $t = \alpha - 1$, then this scheme is optimal.

Scheme 1

If $(n, a) = d$, then at time interval i, send the row R_i as determined by the following:

```
(1) Set $i = 0$.
(2) Evaluate
        For $j = 0$ to $d - 1$ do
            For $k = 0$ to $(\frac{n}{d} - 1)$ do
                $(R_i = (ak + j) \bmod n,\ i = i + 1)$
```

□

It is now shown that this scheme achieves $t = \alpha - 1$, providing $b > 0$.

Theorem 4.8 *If $n = \alpha a + b, b > 0$ and $(n, a) = d$, then Scheme 1 sends all n rows of the array and achieves $t = \alpha - 1$.* □

Proof Firstly, it is shown that Scheme 1 does send each row of the array. Now for each of the d values of j, k takes $\frac{n}{d}$ values, hence there are n rows transmitted. The same row is not transmitted twice because d is the highest common factor of n and a and $j \leq d - 1$.

Secondly, the value of t is found. Consider the sequence of the row numbers in the order in which the rows are transmitted, i.e.

$$0, a, 2a, \ldots, \alpha a, a - b, 2a - b, \ldots$$

Split this sequence into ordered blocks, each block commencing with an element $< a$ and consisting of the immediately following terms in the sequence (in order) up to, but not including, the next element $< a$. Whilst j is fixed and k is incrementing, any block has the following form:

$$a-\gamma$$
$$2a-\gamma$$
$$\vdots$$
$$\alpha a-\gamma$$
$$(\alpha+1)a-\gamma?$$

call this Block Y. The ? after the last element denotes that this element may or may not be contained in Block Y, depending on the ratio of a and γ to n. Notice that any two elements in the same block are **not** within a of each other. Now look at the distances between elements of consecutive blocks by considering Block Y and Block (Y+1). There are two cases to consider.

Case 1: $(\alpha+1)a-\gamma$ is not contained in Block Y.

Now the first element in Block (Y+1) is

$$(\alpha+1)a-\gamma-(\alpha a+b)=a-b-\gamma$$

Consider how many elements of Block Y are within a of the first element of Block (Y+1). Now $a-\gamma$ and $a-b-\gamma$ are within a of each other. Also

$$\mid 2a-\gamma-(a-b-\gamma)\mid=\mid a+b\mid\geq a$$

So $2a-\gamma$ is not within $a-b-\gamma$ As the elements of Block Y increase in size, no other element in Block Y is within a of $a-b-\gamma$. In this case , Block Y contains α elements , therefore the number of elements transmitted between the transmission of $a-\gamma$ and $a-b-\gamma$ is $\alpha-1$.

The elements in every block are all the same fixed distance apart, i.e. the third and fourth and elements in a block are the same distance apart as the fourth and fifth and this is true (with the same distance) for any block. Hence there are $\alpha - 1$ elements transmitted between any element in Block Y and the first in Block (Y+1) that is within a of this element.

Case 2: Block Y does contain $(\alpha+1)a-\gamma$.

In this case, the first element in Block (Y+1) is

$$(\alpha+2)a-\gamma-(\alpha a+b) = 2a-b-\gamma$$

Now $2a-b-\gamma$ and $a-\gamma$ are within a of each other. Also

$$\mid 2a-b-\gamma-(2a-\gamma)\mid = \mid -b \mid \leq a-1$$

therefore $2a-b-\gamma$ and $2a-\gamma$ are within a of each other. But,

$$\mid 2a-b-\gamma-(3a-\gamma)\mid = \mid -a-b \mid > a$$

thus $2a-\gamma$ is the last element in Block Y that is within a of $2a-b-\gamma$. However, in this case , Block Y contains $\alpha+1$ elements and therefore the number of elements transmitted between the transmission of $2a-\gamma$ and $2a-b-\gamma$ is $\alpha-1$. Hence, by the same argument as in Case 1, there are $\alpha-1$ elements transmitted between any element in Block Y and the first element in Block (Y+1) that is within a of this element.

Thus, whilst j is fixed and k is incrementing, at least $\alpha-1$ rows are transmitted between the transmission of any two rows that were originally within a of each other. When j is incremented, this affects which block is transmitted next, but not the structure of the block. It is now shown that incrementing j does not affect the distances between elements in consecutive blocks.

Consider increasing j to $j+1$. Let Block Y$'$ be the block before j is incremented and Block (Y$'$+1) the block after. Let the first element of Block Y$'$ be $a-\gamma$, the first element of Block (Y$'$+1) is $j+1$. The last element of Block Y$'$ is either $\alpha a-\gamma$ or $(\alpha+1)a-\gamma$, but this element is equal in value to $(\alpha-1)a+b+j$.

$$\begin{array}{ll}
a-\gamma & j+1 \\
2a-\gamma & a+j+1 \\
\vdots & \vdots \\
\alpha a-\gamma & \\
(\alpha+1)a-\gamma\ ? & \\
\text{Block Y}' & \text{Block (Y}'\text{+1)}
\end{array}$$

Again there are two cases to consider:

Case 1 : $(\alpha-1)a+b+j=\alpha a-\gamma$

which gives $\gamma=a-b-j$

As before, consider the number of elements of Block Y$'$ that are within a of the first element of Block (Y$'$+1). Now $a-\gamma$ and $j+1$ are within a of each other because $j+1<d<a$ and $a-\gamma<a$. Also

$$\mid 2a-\gamma-(j+1)\mid=\mid 2a-(a-b-j)-(j+1)\mid=\mid a+b-1\mid\geq a$$

because $b>0$. Hence $2a-\gamma$ and $j+1$ are not within a of each other. Thus Blocks Y$'$ and (Y$'$+1) follow the same pattern as Blocks

Y and (Y+1) in case 1. Therefore, rows within a of each other are transmitted at least $\alpha - 1$ apart.

Case 2 : $(\alpha-1)a + b + j = (\alpha+1)a - \gamma$

which gives $\gamma = 2a - b - j$

Again, $a - \gamma$ and $j+1$ are within a of each other. Also

$$| 2a - \gamma - (j+1) | = | 2a - 2a + b + j - j - 1 | = | b - 1 | < a$$

and thus $2a - \gamma$ and $j+1$ are within a of each other. But

$$| 3a - \gamma - (j+1) | = | a + b - 1 | \geq a$$

Hence, $2a - \gamma$ is the last element in Block Y′ that is within a of $j+1$.

Again, Blocks Y′ and (Y′+1) follow the same pattern as Blocks Y and (Y+1) in case 2.

Thus, incrementing j does not affect the number of rows transmitted between any two rows originally within a of each other. Hence, Scheme 1 achieves $t = \alpha - 1$. □

Notice that this scheme, in effect, interleaves the rows of the array into an $(\alpha+1) \times a$ array. However, the last row of this array is incomplete. Asking whether or not a column of the $(\alpha+1) \times a$ array contains an element in the $(\alpha+1)^{th}$ row is equivalent to asking if a Block Y (in the proof of Theorem 4.8) contains the ? element.

If a divides n, then Scheme 1 achieves $t = \alpha - 2$. In this case it is suggested that the value of a used should be the least integer greater than the actual a, such that this new value does not divide n.

The next section considers when Scheme 1 may be used.

§4.4.1 Recommendations for Usage of Scheme 1

The situation being considered is one in which interleaving is performed using an $n \times m$ array and m is less than the maximum length burst of errors, this maximum length burst affecting at most a rows. The object of the previous sections has been to develop a scheme (this being the order in which the rows are to be transmitted) which enabled rows within a of each other in the original array to transmitted as far apart as possible.

The purpose of this Section is to consider how far apart it is **necessary** to transmit such rows and to make some recommendations about the use of the scheme when the characteristics of the channel, over which the information is transmitted, are unknown.

§4.4.2 Interleaving and Random Errors

Consider first an interleaving scheme in which the rows are transmitted in sequential order. This scheme in effect performs a permutation of mn data symbols and the reverse process at the receiving end performs the inverse permutation. Now supose that the errors occur at random, that is, there is a fixed probability that a symbol is in error and this probability is independent from symbol to symbol. Hence the probability of a given error pattern depends only on the number of errors in the pattern and not their positions. Therefore permuting the data does not alter the probability of any given error pattern, as permutation affects the positions of the errors and not the number of them.

Now, if interleaving is used where the rows are not transmitted in sequential order, this merely performs an second permutation on the data symbols. Repeating the above argument shows that with this scheme again, the probability of a given error pattern is the same as when no interleaving takes place. Thus, if random errors occur, the probability of a given error pattern is independent of whether or not interleaving is performed and, if interleaving is performed, whether or not the rows are transmitted in sequential order.

§4.4.3 Using Scheme 1 with Unknown a

The object of interleaving is to randomize bursts of errors. Now the maximum number of rows a single burst can affect is a, thus, if there exists a scheme such that no two rows that were within a of each other in the original array are transmitted within a of each other, then this achieves sufficient randomization. This is because, when the rows are considered sequentially again at the receiver, any block of a rows can contain at most one row affected by any given burst.

Therefore, once $t = a - 1$ can be achieved, it is not necessary to increase t further.

Now consider the situation where the value of a is unknown, what is the best policy to adopt ?

<u>Policy 1</u> *If interleaving is performed using Scheme 1, where the dimensions of the array are fixed (i.e. n is fixed), but the maximum number of rows that can be affected by a single burst is unknown, then choose a to be the greatest integer such that $a \leq \alpha$ and $n = \alpha a + b, b < a$ and $b \in \mathbf{Z}$. (This gives $a = \lfloor \sqrt{n-1} \rfloor$).* $\square$

Suppose Policy 1 is used, what happens if the maximum length of burst that can occur is not the chosen a ?

Case 1: The maximum burst length is less than a, say a'.

In this case, the transmission scheme is forcing apart rows that it is not necessary to separate. Now if Scheme 1 had been used with the knowledge that the maximum burst length was a', then it is possible that a larger value of t could have been obtained. However, Policy 1 has $t \geq a - 1$ and as $a' < a$, $t > a'$. Hence, no advantage could be gained by having a larger value of t.

Case 2: The maximum burst length is greater than a, say a''.

In this case, the transmission scheme does not force apart all rows that are within a'' of each other. However, using Scheme 1 with a knowledge of a'' would give $t < a'' - 1$ and therefore some rows originally within a'' of each other will be transmitted within a'' of each other. Thus a knowledge of a'' does not improve the situation. This case demonstrates that (as was to be expected), once a becomes larger than a certain value, Scheme 1 has decreasing effectiveness.

§4.5.1 Conclusion

An optimal solution has been found for the problem proposed by Inmarsat. A policy for using the given scheme when the channel characteristics are unkown has been presented. The scheme provide a means of increasing the effectiveness of interleaving for convolutional coding, without introducing extra time delay. It is noted however, that Scheme 1 has a cut-off point after which it has a decreasing effect on performance.

Chapter 5

Binary Two-Length Codes with Error Correction

§5.1.1 Introduction

The subject of variable length codes was introduced in §1.7.1. Variable length coding is of use for a message over any set of characters where the characters are not all equally likely to occur, e.g. English text. However, there has been very little work done on error-correcting variable length codes. This chapter studies the concept of an error-correcting code with two different wordlengths.

First some theoretical results are presented. These include considerations of linearity and a distance metric for two-length codes. Also, a general decoding scheme is given for such codes.

It is intuitive that taking one wordlength to be a multiple of the other will produce better synchronization properties. This work considers the case when one wordlength is twice the other (this choice is also justified). The idea of using a known, linear block code to form the two-length code is presented, together with necessary bounds on the blocklength and dimension of this block code.

Theoretical results are given for the expected synchronization properties of two-length codes when one wordlength is twice the other. These are quite encouraging. The remainder of the Chapter considers a specific two-length code, formed from the Hamming (7,4) code. It seemed natural to test the performance of this code on the English alphabet, as frequency tables are readily available.

The performance of the two-length code was simulated, along with that of 5-bit ASCII, a (9,5) block code and an appropriate Titchener code. The results of the simulations are in line with the theoretical results. The chosen example code has good synchronization properties and produces relatively low character error rates in the received message.

§5.1.2 Objectives

The aim of this chapter is to develop a theory which will then enable the construction of a binary, two-length code which is capable of correcting bit errors that occur on the channel, so that more information characters are correctly decoded and some synchronization errors are averted. This code should have a faster data rate over a fixed-length, 1-error-correcting block code and should be such that there is no transmission of a fixed sequence to mark the end of a codeword. Thus the code must be uniquely decodable in the absence of errors. It is also desired that the code have some form of recovery ability, that is when a synchronization error occurs, the decoder remains out-of-synch for some period of time and then regains word synchronization. Recovery should occur because of some inherent property of the code, rather than manual alteration of the decoder.

§5.1.3 Initial Ideas

In the codes considered in this chapter, each character will be encoded to a unique binary sequence called a _codeword_, the number of bits in the codeword will be called the _wordlength_. A variable length code may consist of codewords of several different wordlengths. For example, one of the codes in [Ti] has codewords of wordlengths 4,5,6,7,8,9,10,11,12,13,14 and 15 bits. However, for convenience, the codes considered here will have codewords of only two different wordlengths. Let these wordlengths be n_1 and n_2 where $n_1 < n_2$. Such codes will be called _two-length codes_. The length n_1 codewords will be called _short codewords_ and the length n_2 words, _long codewords_. Each letter of the alphabet will be associated with either a short codeword or a long codeword, therefore the probability of sending a particular length codeword may be calculated. Let p_1 and p_2 be the probability of sending a short and long codeword respectively.

Let the first n_1-bits of a long codeword be called the _prefix_ and the remaining $(n_2 - n_1)$-bits the _addon_.

It was seen in §1.7.3 that one of the problems with variable length codes is maintaining word synchronization. When using fixed-length block codes, providing that no bits are lost on the channel, maintaining word synchronization is not a problem because the receiver always knows exactly how many bits form a codeword. When using variable length codes, the receiver has to decide how to split the incoming sequence of bits into the varying length codewords. For two-length codes, the term *word synchronization* will be used to describe the state at a receiver when a received word is correctly identified as either a short codeword or a long codeword. Because this work deals with transmission over noisy channels, the received message will contain errors which may cause confusion between prefices and short codewords, thus it may not always be possible to maintain word synchronization. The term *synchronization error* will be used to describe the situation where a codeword of a different length from that which was sent is decoded. The term *out-of-synch* will be used to describe the state of the decoding process once a synchronization error has occurred and before the decoder returns to decoding codewords of the same length as those which were sent. Whilst the decoder is out-of-synch it is usual for character errors to occur (but see [Ti] for an exception). Now, a synchronization error may be caused by a single bit error and once the synchronization error has occurred the decoder may remain out-of-synch for some time. Thus, variable length codes suffer from error propagation.

To ensure that the two-length code is uniquely decodable in the absence of errors it is chosen to be a prefix code according to the Definition 1.17. However, in the presence of errors, this is not very robust. Unless the code is carefully designed, it may take just one bit error to interchange the prefix of a long codeword and a short codeword. Based on the definitions of t-error-correcting and minimum

Hamming distance from fixed-length block codes, the concept of a prefix code is generalised for the purposes of this work as follows:

<u>Definition 5.1</u> A two-length code is a <u>t-prefix code</u> if any prefix to a long codeword and any short codeword differ in at least $2t+1$ places. □

Notice that, if t is large, a decoder is unlikely to confuse a prefix and a short codeword, hence synchronization errors are unlikely to occur. This will prevent error propagation. Clearly much more structure is required in the code if it is to prevent character errors. This will be discussed later when a specific two-length code is considered, the next few sections are concerned with what can be said in general about two-length binary codes.

§5.2.1 Linearity Considerations

The main area of successful work on block codes has been with linear block codes. It is linear codes for which practical encoding and decoding algorithms exist. Once codewords can have different lengths, it is not possible to talk of a code being linear. This is because the linearity of a block code is defined in terms of a vector space and it is meaningless to talk about vector spaces when more than one length of vector is under consideration.

However, the codewords of a two-length code break down easily into smaller sets, each containing vectors of only one length. If some of these sets can be made linear, then this will give the two-length code some structure, which may be beneficial either for forming a code or for encoding and decoding schemes.

The purpose of this section is to determine which sets can be linear and the effect the linearity of any particular set has on the other sets. The sets that will be considered are : the set of short codewords, the set of long codewords, the set of prefices and the set of addons.

Some notation for these sets is now presented :

Let SC be the set of short codewords and let s denote a member of SC.

Let LC denote the set of long codewords, LC_p the set of prefices and LC_a the set of addons.

If $\mathbf{p} \in LC_p$ and $\mathbf{a} \in LC_a$ are the prefix and addon for a long codeword, then denote this member of LC by $(\mathbf{p} : \mathbf{a})$

Note that it is not assumed that every prefix is paired with every addon in LC, but this is not excluded either, thus it is not necessarily true that $LC = LC_p \times LC_a$ (where $\times$ denotes the Cartesian product).

It is possible to make any one of the four sets linear, but now consider how the linearity of one set affects that of another.

The prefix condition demands that at least $SC \cap LC_p = \emptyset$, therefore SC and LC_p cannot both be linear as they cannot both contain the all-zero vector. However, it is not necessary that $SC \cap LC_a = \emptyset$ or that $LC_p \cap LC_a = \emptyset$. Hence it is possible for both SC and LC_a to be linear and similarly for LC_p and LC_a.
If the addition of long codewords is taken to be ordinary vector addition, then it is clear that

$$LC \text{ linear} \quad \Rightarrow \quad LC_p \text{ linear and } LC_a \text{ linear}$$

The contrapositive of this gives

$$LC_p \text{ not linear or } LC_a \text{ not linear} \quad \Rightarrow \quad LC \text{ not linear}$$

The converse statement

$$LC_p \text{ linear and } LC_a \text{ linear} \quad \Rightarrow \quad LC \text{ linear}$$

is true if $LC = LC_p \times LC_a$, otherwise no conclusion may be drawn.

Applying the above gives the following conditions on the linearity of the sets that form the two-length code.

(i) If SC is linear, then this forces LC_p to be non-linear and therefore LC is non-linear. Also zero, one or two of the following may be true

  (a) $SC \cup LC_p$ is linear.

  (b) LC_a is linear.

(ii) If LC_p is linear then SC is non-linear. This forces one of (a) or (b) to be true and furthermore (c) may or may not be true.

  (a) LC is linear and therefore LC_a is linear.

  (b) LC is non-linear and therefore nothing can be said as to whether or not LC_a is linear.

  (c) $SC \cup LC_p$ is linear.

(iii) If LC_a is linear this forces no constraint on the other sets.

The next section considers a distance measure for two-length codes.

§5.2.2 A Distance Measure

The object of this section is to find a distance metric for a two-length code. A distance measure $d(\mathbf{x},\mathbf{y})$ is a metric if it satisfies the following three axioms.

(i) $d(\mathbf{x},\mathbf{y}) \geq \mathbf{0}$ with equality iff $\mathbf{x} = \mathbf{y}$.

(ii) $d(\mathbf{x},\mathbf{y}) = d(\mathbf{y},\mathbf{x})$.

(iii) $d(\mathbf{x},\mathbf{y}) + d(\mathbf{y},\mathbf{z}) \geq d(\mathbf{x},\mathbf{z})$.

The Hamming distance is a metric for block codes and it therefore seems natural to try to extend it to two-length codes. Denote Hamming distance by d_H and Hamming weight by w_H. The distance measure, d_t, as given below appears an obvious choice for the two-length code. Take $\mathbf{s_1}, \mathbf{s_2} \in SC$ and $(\mathbf{p_1} : \mathbf{a_1}), (\mathbf{p_2} : \mathbf{a_2}) \in LC$, then

$$d_t(\mathbf{s_1},\mathbf{s_2}) = d_H(\mathbf{s_1},\mathbf{s_2})$$

$$d_t((\mathbf{p_1} : \mathbf{a_1}),(\mathbf{p_1} : \mathbf{a_1})) = d_H((\mathbf{p_1} : \mathbf{a_1}),(\mathbf{p_2} : \mathbf{a_2}))$$

$$d_t(\mathbf{s_1},(\mathbf{p_1} : \mathbf{a_1})) = d_H(\mathbf{s_1},\mathbf{p_1})$$

Unfortunately, d_t is not a metric because it does not always satisfy (iii). (Take, for example $\mathbf{x}$ and $\mathbf{z}$ to be short codewords and $\mathbf{y}$ to be a long codeword). It does however satisfy (i) (the distance between a long codeword and a short codeword can never be zero because $SC \cap LC_p = \emptyset$) and (ii).

The next distance measure introduced overcomes this problem and is a metric, as will be shown in Theorem 5.1. But first the formal definition.

<u>*Definition 5.2*</u> The distance measure, d_T, of a two-length code is defined as :

$$d_T(\mathrm{s}_1, \mathrm{s}_2) = d_H(\mathrm{s}_1, \mathrm{s}_2)$$

$$d_T((\mathrm{p}_1 : \mathrm{a}_1), (\mathrm{p}_2 : \mathrm{a}_2)) = d_H((\mathrm{p}_1 : \mathrm{a}_1), (\mathrm{p}_2 : \mathrm{a}_2))$$

$$d_T(\mathrm{s}_1, (\mathrm{p}_1 : \mathrm{a}_1)) = d_T((\mathrm{p}_1 : \mathrm{a}_1), \mathrm{s}_1) = d_H(\mathrm{s}_1, \mathrm{p}_1) + w_H(\mathrm{a}_1)$$

□

<u>Theorem 5.1</u> *The distance measure d_T is a metric on the set of codewords.* □

<u>*Proof*</u> Consider the set, $\mathcal{S}\,^*$, of n_2-bit words that consists of all the short codewords, each with $(n_2 - n_1)$ zeros added on the end, and all the long codewords. Then d_T on the two-length code is equivalent to d_H on $\mathcal{S}\,^*$, and therefore d_T is a metric on the two-length code. □

Note that the definition of d_T extends to length n_1 and length n_2 words that are outside the set of codewords. However, if it is possible for an n_1-bit word to be the prefix of an n_2-bit word and LC_a contains the all-zero word, then d_T is no longer a metric, because the distance between a long codeword and a short codeword may be zero. Thus, d_T is not necessarily a metric on the set of received words, if the two-length code is used on a noisy channel.

It is also possible to define the minimum distance of a two-length code.

<u>*Definition 5.3*</u> The <u>minimum distance</u> , d', of a two-length code is the minimum value of $d_T(\mathrm{x}, \mathrm{y})$ between any pair of codewords x and y. □

The next section will present a decoding scheme which guarantees to correct t errors provided that the code used is a t-prefix code and that $d' \geq 2t+1$.

§5.2.3 A t-Error-Correcting Decoding Scheme

Given a t-prefix code such that $d' \geq 2t+1$, then the following decoding algorithm may be used and guarantees to correct t errors in a single codeword, whether long or short. The scheme is a general one and certain processes may be reduced for specific codes. The scheme assumes that no bits can be lost or gained on the channel, that is the same number of bits are received as were sent, but this is no more than is required when using a block code. The scheme decodes the bits in sequence as they are received.

Decoding Scheme

Step 1 Compare the next n_1-bits with a short codeword, if the distance between them is $\leq t$ then decode to this codeword, otherwise repeat until all short codewords have been compared. If a codeword has been found then halt, otherwise proceed to Step 2.

Step 2 Take the n_1-bits of Step 1 and also the next $(n_2 - n_1)$-bits from the received sequence. Compare these n_2-bits with a long codeword, if the distance between them is $\leq t$ then decode to this codeword, otherwise repeat until all long codewords have been compared. If a codeword has been found halt, otherwise proceed to Step 3.

Step 3 Decode to the codeword that was found to be nearest in all the comparisons of both Step 1 and Step 2. Halt

A flow diagram of this scheme is given in Figure 5.1.

Theorem 5.2 *The decoding scheme described above will correct up to t errors per codeword for either length codeword, if the code used is a t-prefix code with minimum distance $\geq 2t + 1$.* $\square$

Proof There are two cases to consider. Firstly, assume that a short codeword, s, was sent. During transmission, s was corrupted by a binary error vector, e, of length n_1 and weight at most t. Thus, $\mathrm{s} + \mathrm{e}$ is received. Step 1 calculates the distance between $\mathrm{s} + \mathrm{e}$ and all short codewords until possibly one is found at distance $\leq t$ away. Suppose that s is the last of the short codewords to be compared with $\mathrm{s} + \mathrm{e}$. Then for any $\mathrm{s}' \in SC$, $\mathrm{s}' \neq \mathrm{s}$

$$d_T(\mathrm{s}+\mathrm{e}, \mathrm{s}') = d_H(\mathrm{s}+\mathrm{e}, \mathrm{s}')$$

$$\text{now} \quad d_H(\mathrm{s}, \mathrm{s}+\mathrm{e}) + d_H(\mathrm{s}+\mathrm{e}, \mathrm{s}') \geq d_H(\mathrm{s}, \mathrm{s}')$$

$$\Rightarrow \quad d_H(\mathrm{s}+\mathrm{e}, \mathrm{s}') \geq d_H(\mathrm{s}, \mathrm{s}') - d_H(\mathrm{s}, \mathrm{s}+\mathrm{e})$$

$$\Rightarrow \quad d_H(\mathrm{s}+\mathrm{e}, \mathrm{s}') \geq t + 1$$

as $d_H(\mathrm{s}, \mathrm{s}') = d_T(\mathrm{s}, \mathrm{s}') \geq 2t + 1$ and $d_H(\mathrm{s}, \mathrm{s}+\mathrm{e}) = w_H(\mathrm{e}) \leq t$. Thus $\mathrm{s} + \mathrm{e}$ will not be decoded to a short codeword that is not s, and it will be decoded to s because $d_T(\mathrm{s}, \mathrm{s}+\mathrm{e}) = d_H(\mathrm{s}, \mathrm{s}+\mathrm{e}) \leq t$. Hence, if a short codeword is sent and $\leq t$ errors occur, correct decoding occurs.

Secondly, assume that a long codeword, $(\mathrm{p} : \mathrm{a})$, was sent. During transmission this is corrupted by a binary error vector $(\mathbf{e_1} : \mathbf{e_2})$ of length n_2 and such that

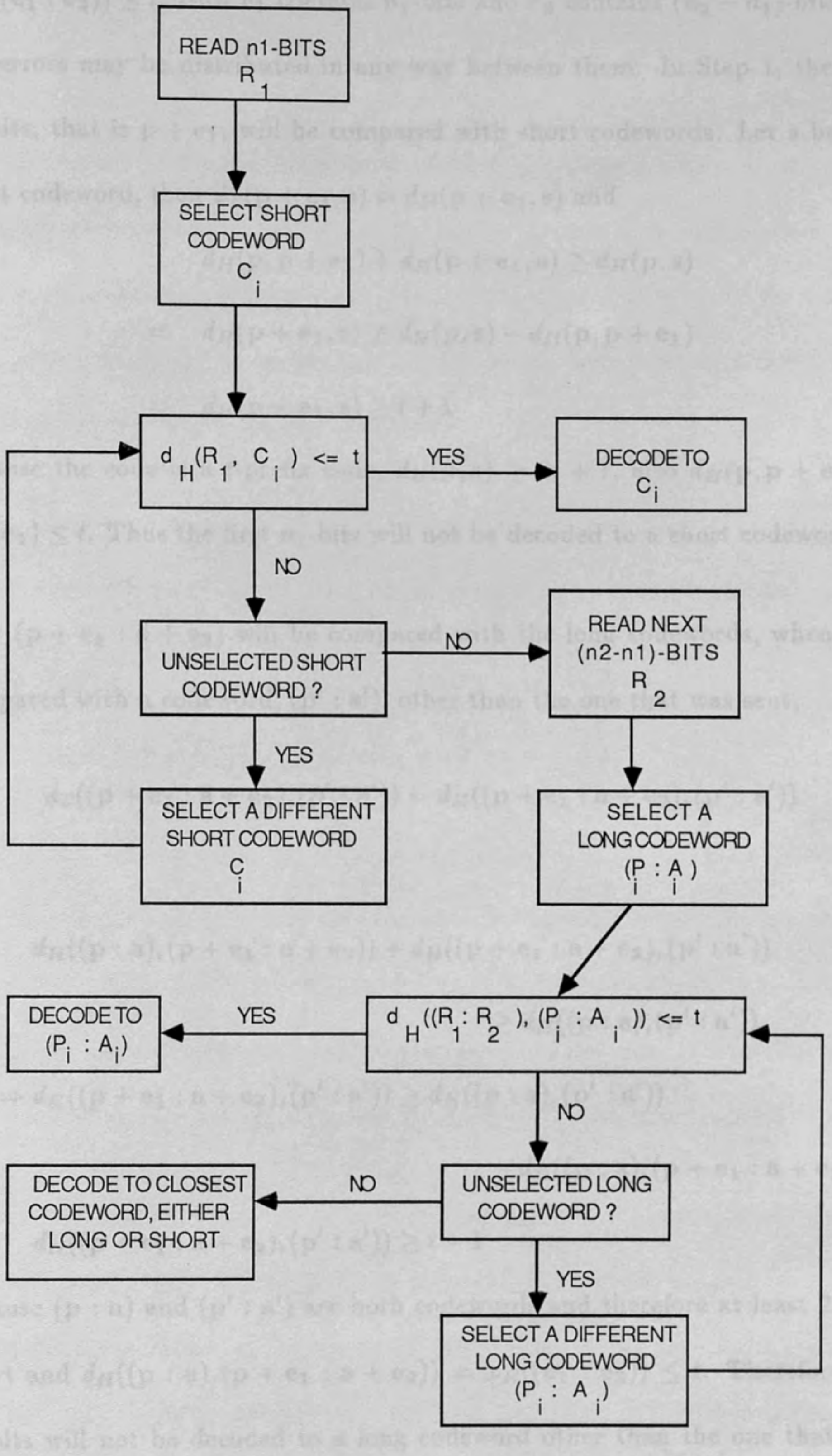

READ n1-BITS R_1
SELECT SHORT CODEWORD C_i
$d_H(R_1, C_i) <= t$
YES
DECODE TO C_i
NO
UNSELECTED SHORT CODEWORD ?
NO
READ NEXT (n2-n1)-BITS R_2
YES
SELECT A DIFFERENT SHORT CODEWORD C_i
SELECT A LONG CODEWORD $(P_i : A_i)$
DECODE TO $(P_i : A_i)$
YES
$d_H((R_1 : R_2), (P_i : A_i)) <= t$
NO
DECODE TO CLOSEST CODEWORD, EITHER LONG OR SHORT
NO
UNSELECTED LONG CODEWORD ?
YES
SELECT A DIFFERENT LONG CODEWORD $(P_i : A_i)$

Figure 5.1

$w_H((\mathbf{e_1} : \mathbf{e_2})) \leq t$. Now $\mathbf{e_1}$ contains n_1-bits and $\mathbf{e_2}$ contains $(n_2 - n_1)$-bits and the errors may be distributed in any way between them. In Step 1, the first n_1-bits, that is $\mathbf{p} + \mathbf{e_1}$, will be compared with short codewords. Let s be any short codeword, then $d_T(\mathbf{p} + \mathbf{e_1}, \mathbf{s}) = d_H(\mathbf{p} + \mathbf{e_1}, \mathbf{s})$ and

$$d_H(\mathbf{p}, \mathbf{p} + \mathbf{e_1}) + d_H(\mathbf{p} + \mathbf{e_1}, \mathbf{s}) \geq d_H(\mathbf{p}, \mathbf{s})$$

$$\Rightarrow \quad d_H(\mathbf{p} + \mathbf{e_1}, \mathbf{s}) \geq d_H(\mathbf{p}, \mathbf{s}) - d_H(\mathbf{p}, \mathbf{p} + \mathbf{e_1})$$

$$\Rightarrow \quad d_H(\mathbf{p} + \mathbf{e_1}, \mathbf{s}) \geq t + 1$$

because the code is a t-prefix code, $d_H(\mathbf{p}, \mathbf{s}) \geq 2t + 1$, also $d_H(\mathbf{p}, \mathbf{p} + \mathbf{e_1}) = w_H(\mathbf{e_1}) \leq t$. Thus the first n_1-bits will not be decoded to a short codeword.

Now $(\mathbf{p} + \mathbf{e_1} : \mathbf{a} + \mathbf{e_2})$ will be compared with the long codewords, when it is compared with a codeword, $(\mathbf{p'} : \mathbf{a'})$, other than the one that was sent,

$$d_T((\mathbf{p} + \mathbf{e_1} : \mathbf{a} + \mathbf{e_2}), (\mathbf{p'} : \mathbf{a'})) = d_H((\mathbf{p} + \mathbf{e_1} : \mathbf{a} + \mathbf{e_2}), (\mathbf{p'} : \mathbf{a'}))$$

and

$$d_H((\mathbf{p} : \mathbf{a}), (\mathbf{p} + \mathbf{e_1} : \mathbf{a} + \mathbf{e_2})) + d_H((\mathbf{p} + \mathbf{e_1} : \mathbf{a} + \mathbf{e_2}), (\mathbf{p'} : \mathbf{a'})) \geq d_H((\mathbf{p} : \mathbf{a}), (\mathbf{p'} : \mathbf{a'}))$$

$$\Rightarrow d_H((\mathbf{p} + \mathbf{e_1} : \mathbf{a} + \mathbf{e_2}), (\mathbf{p'} : \mathbf{a'})) \geq d_H((\mathbf{p} : \mathbf{a}), (\mathbf{p'} : \mathbf{a'})) - d_H((\mathbf{p} : \mathbf{a}), (\mathbf{p} + \mathbf{e_1} : \mathbf{a} + \mathbf{e_2}))$$

$$\Rightarrow \quad d_H((\mathbf{p} + \mathbf{e_1} : \mathbf{a} + \mathbf{e_2}), (\mathbf{p'} : \mathbf{a'})) \geq t + 1$$

because $(\mathbf{p} : \mathbf{a})$ and $(\mathbf{p'} : \mathbf{a'})$ are both codewords and therefore at least $2t + 1$ apart and $d_H((\mathbf{p} : \mathbf{a}), (\mathbf{p} + \mathbf{e_1} : \mathbf{a} + \mathbf{e_2})) = w_H((\mathbf{e_1} : \mathbf{e_2})) \leq t$. Therefore the n_2-bits will not be decoded to a long codeword other than the one that was sent. They will be decoded to the one that was sent, because $d_T((\mathbf{p} + \mathbf{e_1} : \mathbf{a} + \mathbf{e_2}), (\mathbf{p} : \mathbf{a})) = d_H((\mathbf{p} + \mathbf{e_1} : \mathbf{a} + \mathbf{e_2}), (\mathbf{p} : \mathbf{a})) = w_H((\mathbf{e_1} : \mathbf{e_2})) \leq t$. Thus, if a long codeword is sent and less than t errors occur, correct decoding will take place. Therefore the theorem is proved. □

The discussion so far has been involved with decoding a single word. If the scheme's performance is to be assessed, then it must not be forgotten that once synchronization is lost, there will be knock-on errors to be dealt with. How synchronization errors are handled is seen in the decoding scheme of the specific example, which follows later.

So far, only general results about two-length codes have been given, attention is now turned towards a specific method of forming two-length codes.

§5.3.1 Two-Length Binary Codes with $n_2 = 2n_1$

This section will use the same notation as previous sections for short codewords, long codewords etc. It will specify a method of forming two-length codes with $n_2 = 2n_1$. The reason for this choice of codeword lengths is to permit the decoder to regain synchronization without outside intervention, why this occurs will become apparent in §5.4.1. First consider the construction of the code.

Let $n_2 = 2n_1$ and choose n_1 to be the blocklength of a known, linear, binary t-error-correcting block code, S. This code will be called the <u>base code</u>. Choose some of the codewords of S to be short codewords and some to form the set of prefices, LC_p. This choice must be such that $SC \cap LC_p = \emptyset$ and thus the code is a t-prefix code. The set of addons, LC_a, is also made up of codewords from S, with no restriction on the choice that can be made, thus some codewords from the base code may appear in LC_a and one of LC_p and SC.

The long codewords are formed by pairing prefices and addons, each addon may be paired with more than one prefix and vice versa. The size of the sets must be such that a unique codeword exists for each character in the source alphabet.

Thus a two-length code is formed from one existing block code, such codes will be called Derived codes. Each letter of the source alphabet is assigned a codeword, either long or short, depending on the relative probability of the letter being sent. Encoding can then be performed by using a look-up table, that is a table that lists each source letter together with its codeword.

At the decoder the received sequence is first considered as n_1-bit blocks and passed through a decoder for the base code. This decoder will, by definition, correct t errors in every n_1-bit block. The second process is to split the n_1-bit blocks into codewords, this is done sequentially. That is, if the first block is in SC then it is considered as a short codeword and if the next block is in LC_p, then this block and the next block are considered to be a long codeword. Note that if the third block is not an addon for the second block, i.e. they do not form a codeword, then a ? is output. The decoding process continues in a like manner for the whole received sequence.

If p_1 and p_2 are as in §5.1.3, then the average wordlength, N, obtained by using a two-length code is given by

$$N = p_1 n_1 + p_2 n_2$$

but $p_2 = 1 - p_1$ and for Derived codes, $n_2 = 2n_1$, which gives

$$\begin{aligned} N_D &= p_1 n_1 + 2n_1(1 - p_1) \\ &= n_1(2 - p_1) \qquad (5.1) \end{aligned}$$

One of the aims of using two-length codes is to reduce the average wordlength to less than the blocklength of a comparable fixed-length block code, the next section examines some bounds on the blocklength and dimension of the base code for this aim to be achievable.

§5.3.2 Bounds on Blocklength and Dimension

There are well tabulated bounds on the blocklength, dimension and minimum distance of block codes, see [Ve], with improvements by R. Hill and K.L. Traynor (to be published). Thus for a given alphabet, it would be easy to find the shortest blocklength code with sufficient codewords that would correct t errors. Let the blocklength and dimension of such a code be n and k respectively. Let n_1 and k_1 be the blocklength and dimension of the base code used to form the Derived code of §5.3.1. Suppose that the source alphabet has α letters, each with probability f_i of being output by the source, with the letters arranged so that $f_i \geq f_{i+1}\ \forall i$. Let $S = \mid SC \mid$ and $L = \mid LC_p \mid$, then

$$p_1 = \sum_{i=1}^{S} f_i$$

Now, for a Derived code to be worth using it is required that $N_D < n$, thus

$$n_1(2 - p_1) < n$$

$$\text{thus} \quad n_1 < \frac{n}{2 - \sum_{i=1}^{S} f_i} \qquad (5.2)$$

It is clear that $n_1 < n$, otherwise the Derived code can never do better than the block code.

Now S codewords from the base code have been used as short codewords, but any of the remaining $2^{k_1} - S$ codewords may be used as prefices and any of the 2^{k_1} codewords from the base code may be used as addons. Thus the maximum number of codewords for the Derived code is

$$S + (2^{k_1} - S)2^{k_1}$$

and this must be at least as great as the number of letters in the alphabet. Therefore

$$\begin{aligned}
& S + (2^{k_1} - S)2^{k_1} \geq \alpha \\
\Rightarrow\quad & (2^{k_1})^2 - S2^{k_1} + (S - \alpha) \geq 0 \\
\Rightarrow\quad & (2^{k_1} - \frac{S + \sqrt{S^2 - 4S + 4\alpha}}{2}) \times (2^{k_1} - \frac{S - \sqrt{S^2 - 4S + 4\alpha}}{2}) \geq 0 \\
\Rightarrow\quad & 2^{k_1} \geq \frac{S + \sqrt{S^2 - 4S + 4\alpha}}{2} \\
\text{or}\quad & 2^{k_1} \leq \frac{S - \sqrt{S^2 - 4S + 4\alpha}}{2}
\end{aligned}$$

However, $\alpha > S, \Rightarrow S^2 - 4S + 4\alpha > S^2$, therefore the second bound on 2^{k_1} cannot be true, thus

$$k_1 \geq \log_2(S + \sqrt{S^2 - 4S + 4\alpha}) - 1 \tag{5.3}$$

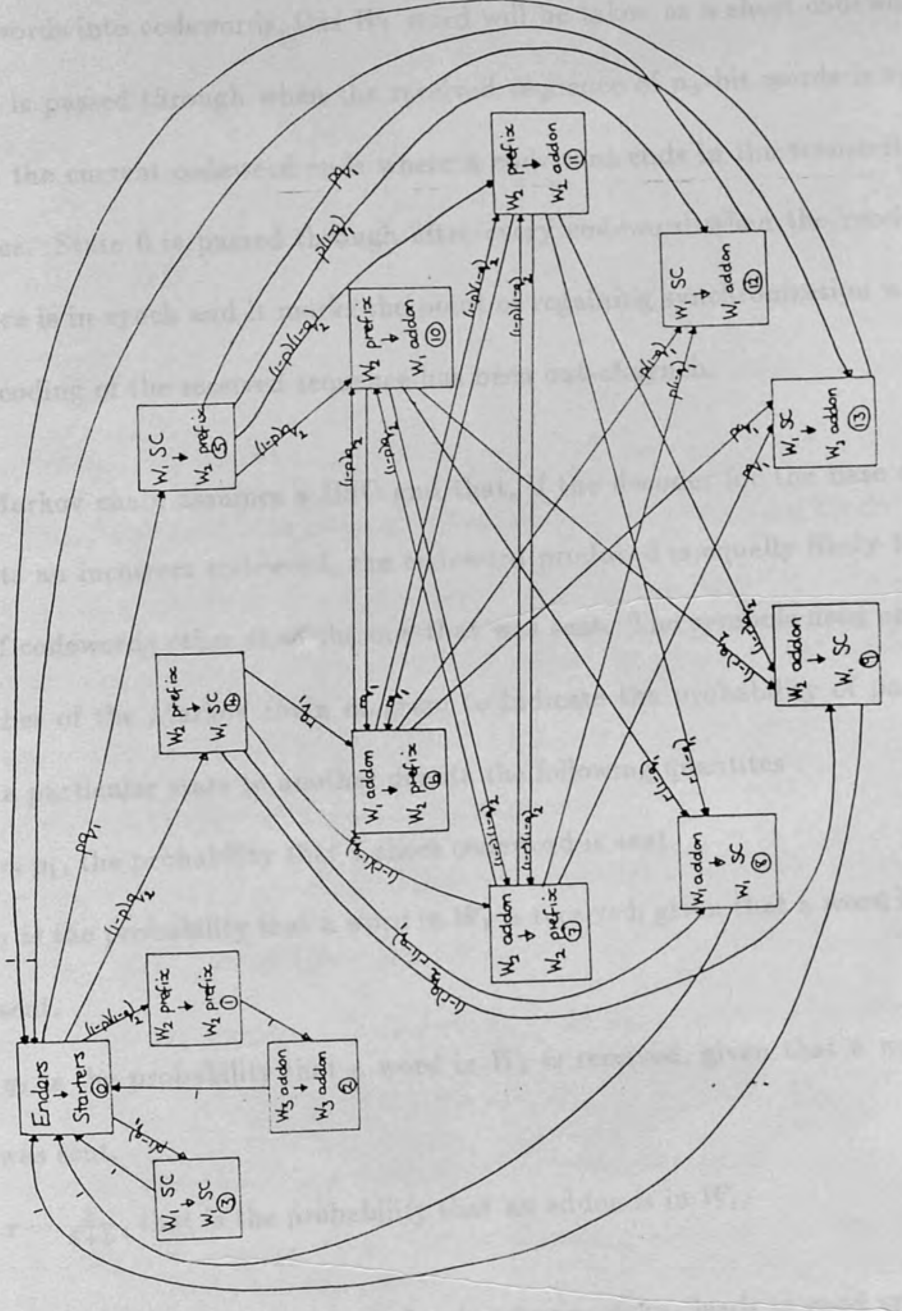
Now, because $n_1 < n$ and both codes have minimum distance at least $2t + 1$ and n is the shortest blocklength code with dimension k, it is clear that $k_1 < k$.

Equation (5.2) provides an upper bound for n_1 and Equation (5.3) provides a lower bound for k_1, which is upper bounded by k. Both (5.2) and (5.3) are dependent on S and for a given S it may not be possible to find n_1, k_1 that both satisfy the bounds. The next section will examine the effects of synchronization errors on Derived codes.

§5.4.1 Investigation of the Synchronization Properties of Binary Two-Length Codes with $n_2 = 2n_1$

This section is concerned only with synchronization errors and not with character errors. Thus it is the length of a codeword that is important, not the character that it represents. The possible sequences of n_1- bit blocks that can be transmitted and those that could be received can be regarded as a Markov chain. Consider the following specific example. (Note that it is assumed that the decoder for the base code always produces a codeword). Let all the codewords of the base code be used as either a short codeword or a prefix, let $LC_a = SC \cup LC_p$ and let $LC = LC_P \times LC_a$. These codes will be called complete Derived codes. The Markov chain for such a code is given in Figure 5.2.

The sequences that are sent and received look like sequences of codewords from the base code. Let W_1 be the set of those codewords from the base code which are short codewords and W_2 the set of those codewords which are prefices. Then a word received that is in W_1 may be a short codeword or an addon, similarly a word received that is in W_2 may be the start of a long codeword or the end of one. In Figure 5.2, the states are numbered for ease of identification. For states 1 to 13, the top line describes the word that was sent and the bottom line the word that was received, e.g. state 4 indicates that the current n_1-bits were sent



Enders → Starters ⓪
W₂ prefix → W₂ prefix ①
W₃ addon → W₃ addon ②
W₁ SC → W₁ SC ③
W₂ prefix → W₁ SC ④
W₁ SC → W₂ prefix ⑤
W₁ addon → W₁ SC ⑥
W₂ addon → W₂ prefix ⑦
W₁ addon → W₁ SC ⑧
W₁ addon → W₂ prefix ⑨
W₂ prefix → W₁ addon ⑩
W₂ prefix → W₁ addon ⑪
W₁ SC → W₁ addon ⑫
W₁ SC → W₁ addon ⑬

Figure 5.2

as a prefix to a long codeword, because of channel errors the base code decoder has output a word from W_1 and, because of the current position in splitting the n_1-bit words into codewords, this W_1 word will be taken as a short codeword. State 0 is passed through when the received sequence of n_1-bit words is split so that the current codeword ends where a codeword ends in the transmitted sequence. State 0 is passed through after every codeword when the received sequence is in-synch and it marks the point of regaining synchronization when the decoding of the received sequence has been out-of-synch.

The Markov chain assumes a BSC and that, if the decoder for the base code outputs an incorrect codeword, the codeword produced is equally likely to be any of codewords other than the one that was sent. The symbols used on the branches of the Markov chain diagram to indicate the probability of passing from a particular state to another denote the following quantites :

(i) $p = p_1$, the probability that a short codeword is sent.

(ii) q_1 is the probability that a word in W_2 is received, given that a word in W_1 was sent.

(iii) q_2 is the probability that a word in W_1 is received, given that a word in W_2 was sent.

(iv) $r = \frac{S}{S+L}$, that is the probability that an addon is in W_1.

The Markov chain demonstrates why choosing $n_2 = 2n_1$ leads to good synchronization recovery. For example, the paths 0,4,8,0 and 0,4,9,0 only exist because the addon of a long codeword can also be a short codeword.

A path through the Markov chain which leaves state 0 will pass through one of two distinct state sets, the out-of-synch states $\{4,5,6,7,8,9,10,11,12,13\}$ or the in-synch states $\{1,2,3\}$ before returning to state 0. One property of a variable length coding scheme that it is desirable to know is the expected length of time that the decoder will be out-of-synch, given that a synchronization error occurs. That is, the expected length of time before synchronization is regained. (There is another parameter which could be considered, that is the average time-out-of-synch over the whole transmission). For the Markov chain in question, passing through each state (except state 0) represents the decoding of one n_1-bit block. Thus, it is possible to calculate the average number of n_1-bit blocks that will be out-of-synch, given that a synchronization error occurs. This is done by finding the average length of the path through the out-of-synch states between two consecutive visits to state 0, assuming that the out-of-synch states where entered on leaving state 0. Consider the following :

E(time out-of-synch, given that a synchronization error occurs) = E (time in out-of-synch states between two successive visits to state 0, given that the out-of-synch states are entered) = E(time out-of-synch between two successive visits to state 0)/p(go out-of-synch from state 0).

Let S_E be the expected time out-of-synch between two successive visits to state 0. Then S_E may be found from the stationary distribution of the Markov chain. The following is taken from [GS], §6.2, pages 119 - 125. The Markov chain in question is finite, irreducible and therefore recurrent with stationary distribution $\mathbf{\Pi}$, where $\mathbf{\Pi P} = \mathbf{\Pi}$. The transition matrix, $\mathbf{P}$, of the Markov chain is the matrix of probabilities of transfer from each state to every other. Now $\Pi_i = \mu_i^{-1}$, where μ_i is the mean recurrence time of state i. Let $\rho_i(k)$ be the mean number of visits of the chain to state i between two successive visits to state k. Then, if k is a non-null state of an irreducible, persistant chain, then there is a stationary distribution $\mathbf{\Pi}$ with $\Pi_i = \rho_i(k)/\mu_k$. Now the mean of a sum equals the sum of the means and therefore

$$\begin{aligned} S_E &= \sum_{i=4}^{13} \rho_i(0) \\ &= \sum_{i=4}^{13} \frac{\Pi_i}{\Pi_0} \\ &= \frac{1}{\Pi_0} \sum_{i=4}^{13} \Pi_i \end{aligned}$$

The Markov chain of Figure 5.2 satisfies all the necessary constraints and has transition matrix $\mathbf{P}$ as in Figure 5.3.

To find the stationary distribution it is neceessary to solve $\mathbf{\Pi P} = \mathbf{\Pi}$ from the following set of equations :

$$\Pi_0 = \Pi_2 + \Pi_3 + \Pi_8 + \Pi_9 + \Pi_{12} + \Pi_{13} \tag{1}$$

$$\Pi_1 = (1-p)(1-q_2)\Pi_0 \tag{2}$$

$$\Pi_2 = \Pi_1 \tag{3}$$

$$\Pi_3 = p(1-q_1)\Pi_0 \tag{4}$$

$$\Pi_4 = (1-p)q_2\Pi_0 \tag{5}$$

$$\Pi_5 = pq_1\Pi_0 \tag{6}$$

$$\Pi_6 = rq_1(\Pi_4 + \Pi_{10} + \Pi_{11}) \tag{7}$$

$$\Pi_7 = (1-r)(1-q_2)(\Pi_4 + \Pi_{10} + \Pi_{11}) \tag{8}$$

$$\Pi_8 = r(1-q_1)(\Pi_4 + \Pi_{10} + \Pi_{11}) \tag{9}$$

$$\Pi_9 = (1-r)q_2(\Pi_4 + \Pi_{10} + \Pi_{11}) \tag{10}$$

$$
P = \begin{pmatrix}
0 & (1-p)(1-q2) & 0 & p(1-q1) & (1-p)q2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & rq1 & (1-r)(1-q2) & r(1-q1) & (1-r)q2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (1-p)q2 & (1-p)(1-q2) & p(1-q1) & pq1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (1-p)q2 & (1-p)(1-q2) & p(1-q1) & pq1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (1-p)q2 & (1-p)(1-q2) & p(1-q1) & pq1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & rq1 & (1-r)(1-q2) & r(1-q1) & (1-r)q2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & rq1 & (1-r)(1-q2) & r(1-q1) & (1-r)q2 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
$$

Figure 5.3

$$\Pi_{10} = (1-p)q_2(\Pi_5 + \Pi_6 + \Pi_7) \tag{11}$$

$$\Pi_{11} = (1-p)(1-q_2)(\Pi_5 + \Pi_6 + \Pi_7) \tag{12}$$

$$\Pi_{12} = p(1-q_1)(\Pi_5 + \Pi_6 + \Pi_7) \tag{13}$$

$$\Pi_{13} = pq_1(\Pi_5 + \Pi_6 + \Pi_7) \tag{14}$$

As the Π_i form a probability distribution, there is the following additional constraint :

$$\sum_{i=0}^{13} \Pi_i = 1 \tag{15}$$

Because the Markov chain is finite and irreducible the above 15 equations have a unique solution. Hence the first fourteen equations have solutions forming a 1-dimensional vector space. Thus, wlog, set $\Pi_0 = 1$ and consider the first fourteen equations only, then use equation (15) to find the particular solution. Setting $\Pi_0 = 1$ gives

$$\Pi_2 + \Pi_3 + \Pi_8 + \Pi_9 + \Pi_{12} + \Pi_{13} = 1$$

$$\Pi_1 = (1-p)(1-q_2)$$

$$\Pi_2 = (1-p)(1-q_2)$$

$$\Pi_3 = p(1-q_1)$$

$$\Pi_4 = (1-p)q_2$$

$$\Pi_5 = pq_1$$

from equations (1) - (6) respectively. Combining (7) and (8) and substituting for Π_4 gives

$$\Pi_6 + \Pi_7 = [(1-r)(1-q_2) + rq_1][(1-p)q_2 + \Pi_{10} + \Pi_{11}]$$

Combining (11) and (12) and substituting for Π_5 gives

$$\Pi_{10} + \Pi_{11} = (1-p)(pq_1 + \Pi_6 + \Pi_7)$$

Combining these last two equations gives

$$\Pi_6 + \Pi_7 = [(1-r)(1-q_2) + rq_1][(1-p)q_2 + (1-p)(pq_1 + \Pi_6 + \Pi_7)]$$

Rearranging gives

$$\Pi_6 + \Pi_7 = \frac{[(1-r)(1-q_2) + rq_1][(1-p)(q_2 + pq_1)]}{1 - (1-p)[(1-r)(1-q_2) + rq_1]} \qquad (16)$$

Substituting (16) into the equation for $\Pi_{10} + \Pi_{11}$ and rearranging gives

$$\Pi_{10} + \Pi_{11} = (1-p)\left\{\frac{pq_1 + [(1-r)(1-q_2) + rq_1](1-p)q_2}{1 - (1-p)[(1-r)(1-q_2) + rq_1]}\right\} \qquad (17)$$

It is now possible to remove all the Π_i from the LHS of (15), therefore to obtain the particular solution it is necessary to solve

$$\lambda\left\{3 - p + pq_1 + \frac{[(1-r)(1-q_2) + rq_1](1-p)(q_2 + pq_1)}{1 - (1-p)[(1-r)(1-q_2) + rq_1]} + \frac{(1-p)(pq_1 + [(1-r)(1-q_2) + rq_1](1-p)q_2}{1 - (1-p)[(1-r)(1-q_2) + rq_1]}\right\} = 1$$

Rearranging gives

$$\lambda\left\{\frac{1 + (2-p)(1 + pq_1) + (1-p)[(1-r)(1-q_2) + rq_1][p + 2q_2 - pq_2 - 3]}{1 - (1-p)[(1-r)(1-q_2) + rq_1]}\right\} = 1 \qquad (18)$$

To obtain the particular solution, put $\Pi_0 = \lambda$ and solve for the remaining Π_i. However recall that

$$S_E = \frac{\sum_{i=4}^{13} \Pi_i}{\Pi_0} = \frac{1 - (\Pi_0 + \Pi_1 + \Pi_2 + \Pi_3)}{\Pi_0}$$

Thus, from equations (2),(3) and (4)

$$S_E = \frac{1 - \lambda[3 + 2q_2(p-1) - p(q_1 + 1)]}{\lambda}$$

Substituting for λ from (18) and rearranging gives

$$S_E = p(q_1 - q_2) + \frac{(pq_1 + q_2)(2-p)}{1-(1-p)[(1-r)(1-q_2)+rq_1]}$$

It can be seen from Figure 5.2, that the probability of entering the out-of-synch states is $pq_1 + (1-p)q_2 = p(q_1 - q_2) + q_2$ and therefore

Theorem 5.3 *The expected number of n_1-bit blocks that are out-of-synch, given that a synchronization error occurs, is E_{os} where*

$$E_{os} = \frac{p(q_1 - q_2)}{p(q_1 - q_2) + q_2} + \frac{(pq_1 + q_2)(2-p)}{(1-(1-p)[(1-r)(1-q_2)+rq_1])(p(q_1-q_2)+q_2)}$$

$\square$

Notice that it is not surprising that q_1 and q_2 appear in the formula for E_{os}, even though they are associated with going out-of-synch. This is because one of the ways to regain synchronization is for (in effect) another synchronization error to occur.

This section has developed a measure of the average time the decoder will be out-of-synch once a synchronization error occurs for complete Derived codes. The next section considers E_{os} when less severe restrictions are placed on the sets LC and LC_a.

§5.4.2 E_{os} when $LC_a \subset SC \cup LC_p$ and $LC \subset LC_p \times LC_a$

The codes of this section are Derived codes, with each codeword of the base code being either a short codeword or a prefix in the two-length code. But, unlike complete Derived codes, the set of addons does not consist of all the prefices and all the short codewords, i.e. $LC_a \subset SC \cup LC_p$. Also the constraint that the long codewords be the Cartesian product of the prefices and the addons is relaxed. Instead, it is insisted that each prefix has the same number of addons associated with it, x of these addons come from SC and y come from LC_p, where x and y are fixed for all prefices. These codes are called *fixed-ratio Derived codes*.

In this case it is possible to decode an n_1-bit block to a prefix and for the next n_1-bit block not to be a possible addon. In this situation specify that the decoder will consider the two blocks to be a long codeword, but that this word will be flagged as uncertain.

With all the above assumptions, the Markov chain for the decoding process for this code is the same as for complete Derived codes, if r is defined as follows

$$r = \frac{x}{x+y}$$

This is because the Markov chain deals with synchronization errors, not character errors and no different synchronization errors can occur with the codes of this section as from the codes of the previous section. Hence, the formula for E_{os} is also valid for fixed-ratio Derived codes.

The next section introduces a specific two-length code for the 26 characters of the English alphabet, based on the ideas of §5.3.1 - §5.4.2.

§5.5.1 A Binary Two-Length Code for the English Alphabet

The binary, linear, 1-error-correcting block code with the shortest blocklength that has at least 26 codewords, is a (9,5) code. Consider forming a Derived code for the English alphabet as used in normal text. The relative frequency of the occurrence of each letter is given in Figure 5.4.

If the chosen base code has minimum distance 3, then this forces the prefices and the short codewords to have a minimum distance of 3. Thus the Derived code formed will be a t-prefix code with $t = 1$. Refer to equation (5.3), the least value of S is 1, otherwise the code is not two-length. If $S = 1$ then $k_1 \geq \log_2(1 + \sqrt{101}) - 1$, thus the least value of k_1 is 3. The binary, linear, 1-error-correcting block code of dimension 3 with the shortest blocklength is a (6,3) code. With $n = 9$ and $n_1 = 6$, to satisfy (5.2), it is required that

$$\sum_{i=1}^{S} f_i \geq 2 - \frac{n}{n_1} = \frac{1}{2}$$

from Figure 5.4 it can be seen that this requires $S \geq 6$. Now the (6,3) code has 8 codewords, if 6 become short codewords there are a maximum of 2 prefices. All the codewords of the (6,3) code can be addons, therefore such a Derived code can have at most $6 + (2 \times 8) = 22$ codewords and thus is insufficient for the English alphabet.

Lengthening the blocklength of the base code merely makes things worse, therefore consider a base code of dimension 4. For this code to be 1-error-correcting it must have blocklength 7. Can a Derived code be formed from 16 codewords

Position	Letter	Relative Frequency	Cumulative Relative Frequency
1	E	12.702	12.702
2	T	9.056	21.758
3	A	8.167	29.925
4	O	7.507	37.432
5	I	6.966	44.398
6	N	6.749	51.147
7	S	6.327	57.474
8	H	6.094	63.568
9	R	5.987	69.555
10	D	4.253	73.808
11	L	4.025	77.833
12	C	2.782	80.615
13	U	2.758	83.373
14	M	2.406	85.779
15	W	2.360	88.139
16	F	2.228	90.367
17	G	2.015	92.382
18	Y	1.974	94.356
19	P	1.929	96.285
20	B	1.492	97.777
21	V	0.978	98.755
22	K	0.772	99.527
23	J	0.153	99.680
24	X	0.150	99.830
25	Q	0.095	99.925
26	Z	0.074	100.00

Figure 5.4

of length 7 ? Equation (5.2) gives

$$\sum_{i=1}^{S} f_i \geq \frac{5}{7}$$

Figure 5.4 shows that this requires $S \geq 10$. A (7,4) code has 16 codewords, if 10 become short codewords then there are at most 6 prefices, but each prefix can have up to 16 addons. Thus such a code has a maximum of $10 + (6 \times 16) = 106$ codewords, which is quite sufficient for the English language! It is therefore possible to have more short codewords than 10, and thus reduce the average wordlength of the code further, even 15 short codewords and 1 prefix can still provide sufficient codewords for the English alphabet.

Suppose that it is desired to form a fixed-ratio Derived code for the English alphabet, then an exhaustive search through all possible ways of splitting the codewords of the base code into short codewords and prefices shows that only an 11:5 or a 14:2 split produces a code where each prefix has the same number of addons. §5.4.2 also requires that each prefix have a given number of addons from SC. If an 11:5 split is chosen, then each prefix could have 0, 1, 2 or 3 addons from SC and if the 14:2 split is chosen each prefix may have 5 or 6 addons from SC.

The remainder of this chapter will examine the performance of fixed-ratio Derived codes formed by choosing the Hamming (7,4) code as the base code and looking at all six of the splits described in the previous paragraph. These fixed-ratio Derived codes will be denoted as _(S,x) D-codes_, where S is the number of short codewords and x is the number of short codewords used as addons for each prefix. The Hamming (7,4) code is _perfect_ because the vector space $GF(2)^7$ can be partitioned into disjoint sets, each set containing a single codeword and those words that are distance one from it. Hence, a decoder for the Hamming (7,4) code always outputs a codeword. The next section looks at the values of E_{os} for the six codes to be considered.

§5.5.2 Synchronization Properties for Two-Length Codes Formed from the Hamming (7,4) code

Recall the formula for E_{os} and consider sending messages of English text. For a given code, the number of short codewords, S , determines p (this being the sum of the probabilities of occurrence of the first S letters in Figure 5.4), but r is determined by the number of short codewords that are used as addons for each prefix. The probabilities q_1 and q_2 depend upon the error statistics for the channel in use. Assume a BSC with bit error probability p_b, then the probability that the decoder for the base code ouputs an incorrect codeword is given by

$$q_0 = 1 - (1 - p_b)^{n_1} - n_1 p_b (1 - p_b)^{n_1 - 1}$$

Now, because it was assumed (§5.4.1) that when the decoder outputs an erroneous codeword, it is equally likely for this to be any codeword other than the one that was sent, q_1 and q_2 are given by

$$q_1 = \frac{L}{S + L - 1} q_0$$
$$q_2 = \frac{S}{S + L - 1} q_0$$

where $S = \mid W_1 \mid$ and $L = \mid W_2 \mid$.

The formula for E_{os} can be written in terms of p, q_0, r, S and L as follows

$$E_{os} = \frac{p(L-S)}{p(L-S)+S} + \frac{(pL+S)(2-p)}{(p(L-S)+S)(1-(1-p)[1-r+\frac{q_0(r(S+L)-S)}{S+L-1}])}$$

This section will obtain upper bounds for E_{os} for each of the (S,x) D-codes. There are six cases to consider.

(1) (11,0) D-code ($p = 0.77833$).

$$E_{os} = \frac{-6p}{-6p+11} + \frac{(5p+11)(2-p)}{(11-6p)(1-(1-p)[1-\frac{11}{15}q_0])}$$

To maximise E_{os} it is necessary to minimize q_0. Thus,

$$E_{os} \leq 2.955$$

(2) (11,1) D-code ($p = 0.77833$).

$$E_{os} = \frac{-6p}{-6p+11} + \frac{(5p+11)(2-p)}{(11-6p)(1-(1-p)[\frac{2}{3}-\frac{17}{45}q_0])}$$

To maximize E_{os} it is necessary to minimize q_0. Thus

$$E_{os} \leq 2.635$$

(3) (11,2) D-code ($p = 0.77833$).

$$E_{os} = \frac{-6p}{-6p+11} + \frac{(5p+11)(2-p)}{(11-6p)(1-(1-p)[\frac{1}{3}-\frac{1}{45}q_0])}$$

As before, minimize q_0 to give

$$E_{os} \leq 2.365$$

(4) (11,3) D-code ($p = 0.77833$).

$$E_{os} = \frac{-6p}{-6p+11} + \frac{(5p+11)(2-p)}{(11-6p)(1-(1-p)[\frac{1}{3}q_0])}$$

This time, to maximize E_{os}, maximize q_0. Thus

$$E_{os} \leq 2.365$$

(5) (14,5) D-code ($p = 0.85779$).

$$E_{os} = \frac{-12p}{14-12p} + \frac{(2p+14)(2-p)}{(14-12p)(1-(1-p)[\frac{1}{6} - \frac{2}{45}q_0]}$$

To maximize E_{os}, minimize q_0. Thus

$$E_{os} \leq 2.183$$

(6) (14,6) D-code($p = 0.85779$).

$$E_{os} = \frac{-12p}{14-12p} + \frac{(2p+14)(2-p)}{(14-12p)(1-(1-p)[\frac{2}{15}q_0]}$$

To maximize E_{os}, maximize q_0. Thus

$$E_{os} \leq 2.159$$

The above bounds on E_{os} show that these (S,x) D-codes have very good expected synchronization recovery properties. Remember that E_{os} represents the number of n_1-bit blocks that are out-of-synch, the number of characters will be less because some blocks will be paired into long codewords.

§5.5.3 Why Restrict Attention to $n_2 = 2n_1$?

For two-length codes, it is intuitive that taking $n_2 = \gamma n_1$, where γ is an integer, will produce better synchronization properties than if n_2 is not a multiple of

n_1. But why choose $\gamma = 2$? The choice to study codes with $\gamma = 2$ was made because it was felt that such a ratio of wordlengths would provide the best synchronization properties. (The better the synchronization properties, the less error propagation and so, hopefully, the better performance). This feeling can be justified by using the results of §5.5.2. If $n_2 = \gamma n_1$ then, if a synchronization error occurs, the minimum number of n_1-bit blocks that are out-of-synch is γ. This is because when a synchronization error occurs, it must somehow involve a long codeword and thus the length of a long codeword determines the minimum number of n_1-bit blocks that are out-of-synch. Therefore, the expected number of n_1-bit blocks out-of-synch, $E_{os}(\gamma) \geq \gamma$. §5.5.2 shows that $E_{os}(2) < 3$ for certain values of p, q, r. Now $E_{os}(3) \geq 3$ for any p, q, r and so on for larger γ. Thus, as it is known that there exist two length codes which achieve a lower average out-of-synch burst length than the best that can be hoped for by choosing a larger γ, these seemed the most appropriate codes to study.

§5.6.1 Simulation of Code Performance

The performance of each of the six (S,x) D-codes of the previous section has been simulated along with that of a 5-bit ASCII code, a Titchener code, [Ti], and a (9,5) block code for the 26 letters of the English language. The simulation was done using the Pascal programs of Appendix 2, together with a 73 line file of English text taken from a magazine article.

Figure 5.5 gives the parity-check matrix for the Hamming (7,4) code used to form the (S,x) D-codes. Also given is the parity-check matrix for the (9,5) block

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Parity-check matrix
of the Hamming (7,4) code.

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Parity-check matrix
of the (9,5) blockcode

E = 100
T = 101
A = 0000
O = 0001
I = 0011
N = 0101
S = 0111
H = 1111
R = 00100
D = 00101
L = 01101
C = 11100
U = 11101
M = 010000
W = 010001
F = 010011
G = 110000
Y = 110001
P = 110011
B = 110101
V = 110111
K = 0100100
J = 0100101
X = 1100100
Q = 1100101
Z = 1101100

Codewords of the
T-code.

Figure 5.5

code. The order in which the codewords of both these codes are used does not affect the simulation. The T-code (also given in Figure 5.5) was formed as in the method of code construction ([Ti], page 1), at each stage it was the first element in the list that was sacrificed as the prefix. A comparison was made between T-codes and D-codes because T-codes are relatively new and seem to show good synchronization properties.

There is one program for each different type of code. Each program reads the data from the English text (the message), encodes the data in the appropriate code and errors are then added to the encoded data. The program then decodes the corrupted data and produces the received version of the message file.

The errors are added to the encoded data by reading bit-wise from a file of zeros and ones, where a one occurs with a fixed probability, and adding these bits (modulo 2) to the bits of the encoded message. The error files were generated using the Random Bernoulli function on the Minitab system. This function produces the results of Bernoulli trials with a specified probability of a one occurring.

The simulation for each code produced a value for the probability of a character being in error in the received version of the message. For the fixed length codes this was obtained simply by comparing the received message and the transmitted message character by character and recording an error if the two characters differ. For the variable length codes, the message and the received data do not always contain the same number of characters and regaining character synchronization often requires skipping more characters in one file than the other.

The number of character errors was taken to be the number of characters in the transmitted message that could not be recovered from the received message. For example, if the top line is the transmitted message and the bottom line what is received, the following has a string of three errors and then a single error.

```
YOULLFINDMEVERYDIFFICULTANNOUNCES
YBELFINDMEVERYTEIFFICULTANNOUNCES
```

This can be seen by replacing erroneous characters in the received message by spaces so that character synchronization is maintained, thus the above example becomes

```
YOULLFINDMEVERYDIFFICULTANNOUNCES
Y   LFINDMEVERY IFFICULTANNOUNCES
```

The errors were counted in this way, as it appeared the most accurate method of imitating how a person would read the message. For each received file the number of character errors was counted using an interactive Pascal program. This program also recorded the maximum number of consecutive character errors (the maximum burst length of character errors) and also the average burst length of character errors.

For the (S,x) D-codes, the amount of time out-of-synch is also measured. This is done by creating two extra files each consisting of the characters 'S' and 'LO'. The first file is created as the message is encoded, each time a short codeword is produced an 'S' is written to the file and each time a long codeword is produced 'LO' is written. A similar process is performed at the decoding end as the final file of text is written. The two files are read character by character, whilst the files agree the received data is in-synch, when they disagree it is out-of-synch and the number of characters which disagree is the number of n_1-bit blocks that are out-of-synch. Both the average and the maximum number of n_1-bit blocks out-of-synch are recorded for each received message.

Thus, the simulation produces two measures for (S,x) D-codes, time out-of-synch and character error burst length. Both these measures are useful. The amount the data is out-of-synch is a measure of how much error propagation is caused by the fact that the code is not of fixed blocklength. The error burst length is important to the user, who is essentially interested in how much data he can recover.

Note, the synchronization properties of T-codes are not measured here as Titcheners own results are given in [Ti]. The next section gives the results of the simulations.

§5.6.2 Simulation Results

The results are presented in two forms, graphs and tables.

Figure 5.6 compares the performance of the four (11,x) D-codes. It can be seen that the more short codewords that are used as addons the better the performance. This follows because word-synchronization is regained quicker and there is thus less error propagation. Word-synchronization is recovered quicker because :

(i) Suppose that a prefix is mistaken for a short codeword, but its addon is received as it was sent. If the addon is from the set of short codewords, then it will be decoded as a short codeword and word-synchronization is regained after two n_1-bit blocks. However if the addon is from the set of prefices then it will be decoded to the prefix of a long codeword and the next n_1-bit block, which was sent as a short codeword or as a prefix, will be taken as an addon, thus word-synchronization is not regained for at least three n_1-bit blocks.

(ii) Suppose that a short codeword is mistaken for a prefix, but the following n_1-bit blocks are received as sent. The next n_1-bit block will be taken as an addon for the prefix. If the next block is a short codeword then word-synchronization is regained after two n_1-bit blocks. If the next block is the prefix to a long codeword and the next is an addon from the set of short codewords, then word-synchronization is regained after three n_1 bit blocks. If the next block is a prefix with an addon from the set of prefices then this addon is taken to be a prefix and word-synchronization is not regained for at least four n_1-bit blocks.

Figure 5.7 compares the performance of the two (14,x) D-codes. (There are in fact two lines on this graph). Although there is little difference between them, the one with the most short codewords as addons gives the better performance for the same reasons as above. Figure 5.8 compares the best (11,x) D-code with the best (14,x) D-code. The one with 14 short codewords gives the best performance. This follows from similar reasoning to the above, the more blocks from the set of short codewords, the easier synchronization is regained.

Figure 5.9 compares the performance of the best (S,x) D-code (the (14,6) D-code) with that of the 9-bit block code, the 5-bit ASCII code and the T-code. This graph does not account for the fact that they all have different data rates. Under this method of comparison, the graph shows that D-codes give the better performance.

Figures 5.10, 5.11 and 5.12 compare the various codes, compensating for the different data rates. The comparison was done by assuming a BSC with AWGN, the relationship between the bit-error probability and the quantity E_b/N_0 is given in [Pr], page 146. Such a comparison is necessary if the data is being transmitted over a bandwidth limited channel.

The reason for this necessity is that for binary signalling, the required bandwidth is inversely proportional to the time taken to send one bit on the channel. Now, to achieve the same information rate with codes of different average wordlengths, the bits of the longer codes must be sent quicker than the bits of the shorter codes. Hence, the longer codes require more bandwidth. This causes problems if the amount of bandwidth available is limited. Plotting the character error probabilities against E_b/N_0 in effect, compares the character error rates when the same bandwidth is used for bits from each code. Further details may be obtained from [Pr], page 156.

Figure 5.10 shows that T-codes do appear better on bandwidth limited channels. However, Figure 5.11 shows that the (S,x) D-code does much better than the 9-bit block code, whereas Figure 5.12 shows that the T-code does not have such a significant gain over the 5-bit ASCII code. From Figure 5.9, the (S,x) D-code is better than the 9-bit block code, even when the differing data rates are not compensated for. Whereas the 5-bit ASCII performs better than the T-code until the differing data rates are compensated for.

Figure 5.13 compares the average character burst error properties for (S,x) D-codes and T-codes and the first table of Figure 5.15 gives the maximum character burst error length for these codes. Figure 5.14 plots the average number of n_1-bit blocks before synchronization is regained for (S,x) D-codes and the second table of Figure 5.15 gives the maximum value. The points of Figure 5.14 compare favourably with the theoretical results of §5.5.2, remembering that each point comes from just one observation whereas the theoretical result is the expected value.

The simulations gave the following average wordlengths, for the T-code $N = 4.34956$ bits per character, for the (11,x) D-code, $N_D = 8.59197$ bits per character and for the (14,x) D-code $N_D = 8.02591$ bits per character. The values for the (S,x) D-codes are close to the expected average wordlengths given by equation (5.1). This equation gives, for the 11 short codewords, $N_D = 8.55169$ and for the 14 short codewords, $N_D = 7.99547$. The expected average wordlength for the T-code used is $N = 4.32539$

§5.7.1 Conclusions

The concept of an error-correcting two-length code has been introduced, together with some linearity considerations and a distance measure. Also, a general decoding scheme was presented. Two-length codes with $n_2 = 2n_1$ have been discussed and theoretical results on their synchronization properties have been given.

Next the concept of forming two-length codes with $n_2 = 2n_1$ from a known code was introduced. One specific example for a given alphabet was then considered, but the method of formation could be used with any known code and for any alphabet where the characters are not all equally likely to occur.

A two-length code formed from the Hamming (7,4) code was studied in detail. This code was shown to have good synchronization properties. It also improved both the average wordlength and the output character error rate over the nearest comparable block code. The last of these is particularly pleasing, because most variable length codes sacrifice data integrity for data rate.

The writer suggests that this method of forming two-length codes could have wide-ranging applications and that these initial results give indications that good performance may be achieved. Also, the idea of using wordlengths that are multiples of 2 of the shortest wordlength could be extended to multi-length codes. For example, a k-length code with wordlengths $n_1, n_2, \ldots, n_k$ where $n_i = 2n_{i-1}$, $i = 2, \ldots, k$. This is perhaps an area for future research.

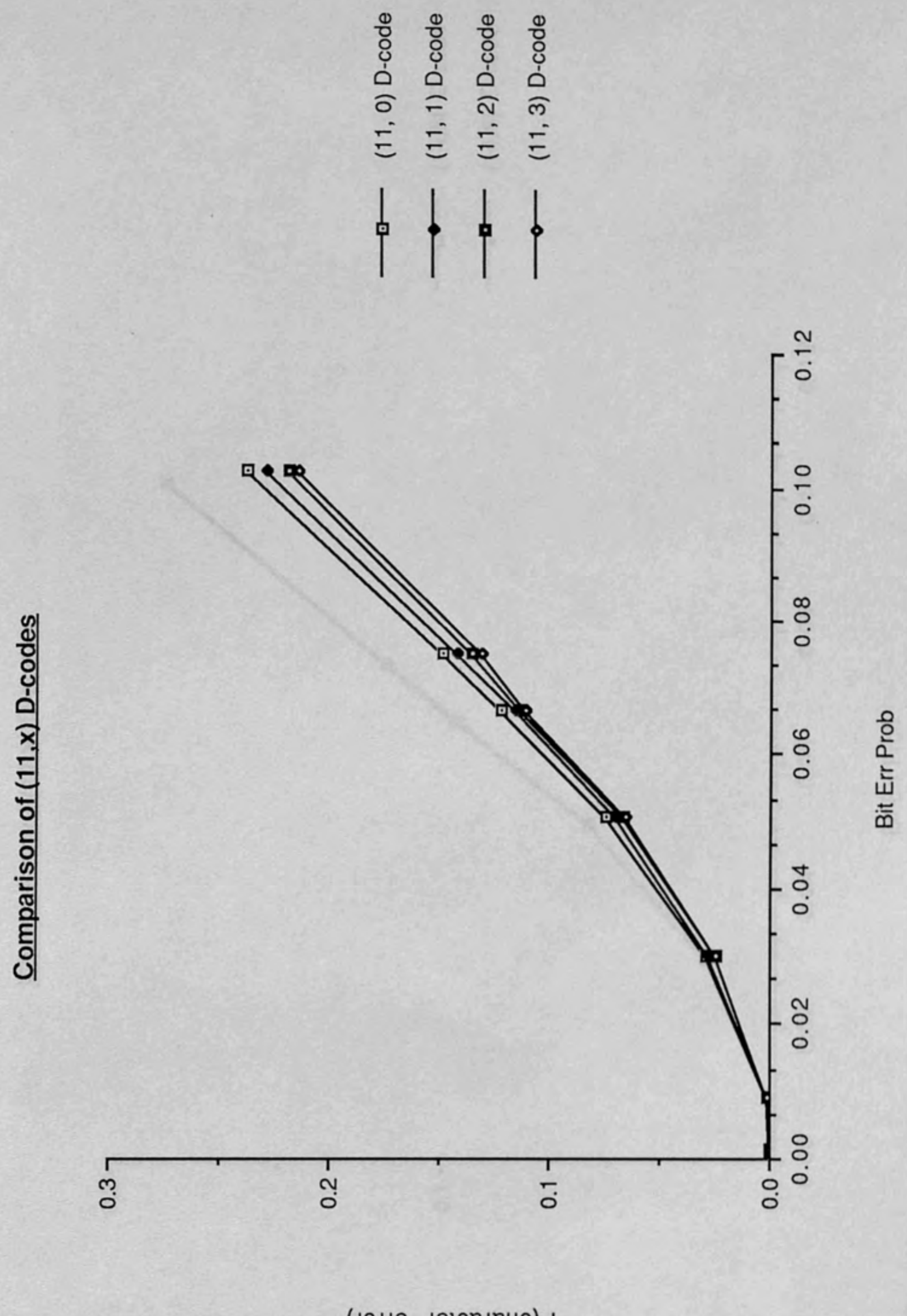

Comparison of (11,x) D-codes
P(character error)
0.0
0.1
0.2
0.3
0.00
0.02
0.04
0.06
0.08
0.10
0.12
Bit Err Prob
(11, 0) D-code
(11, 1) D-code
(11, 2) D-code
(11, 3) D-code

Figure 5.6

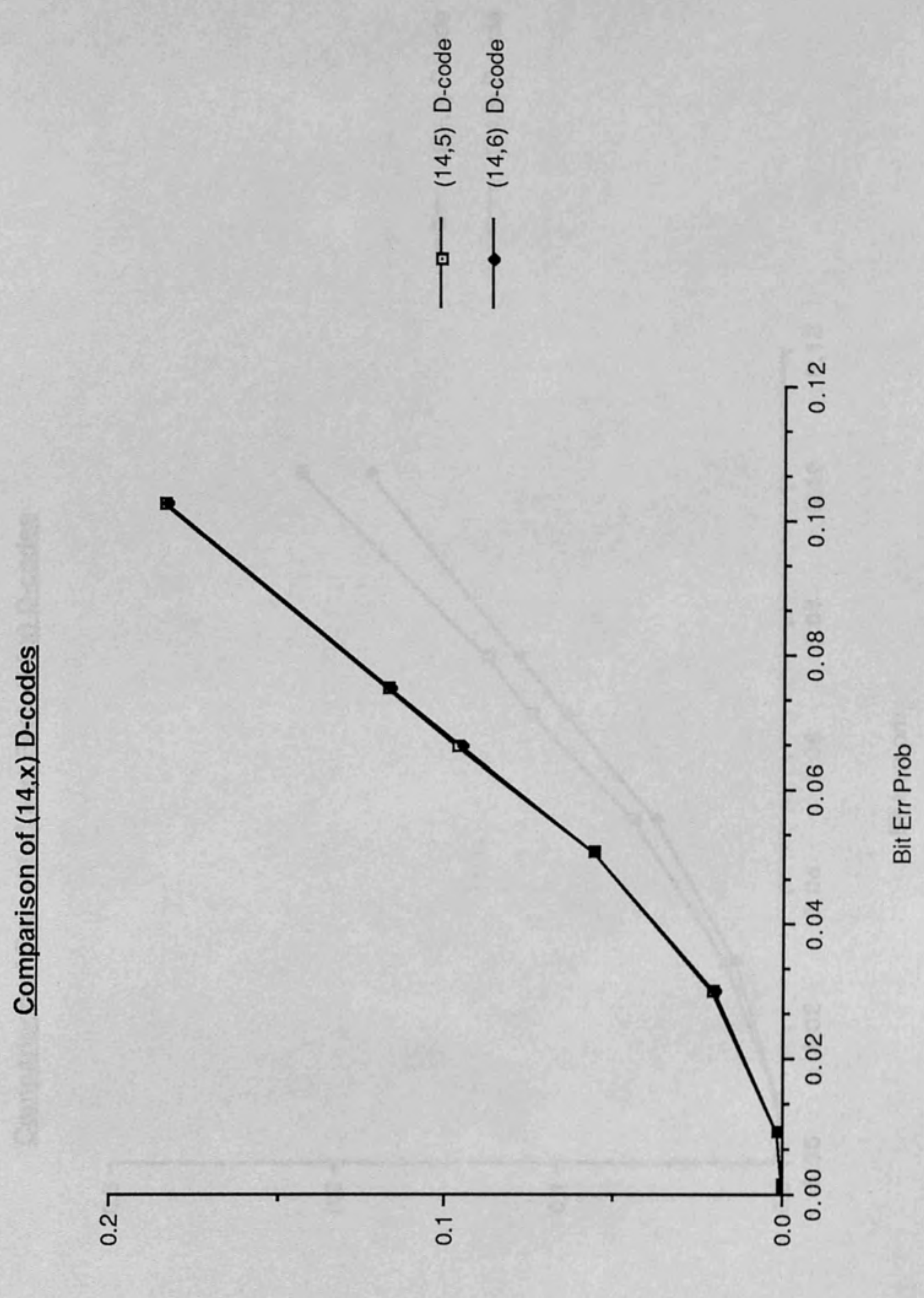
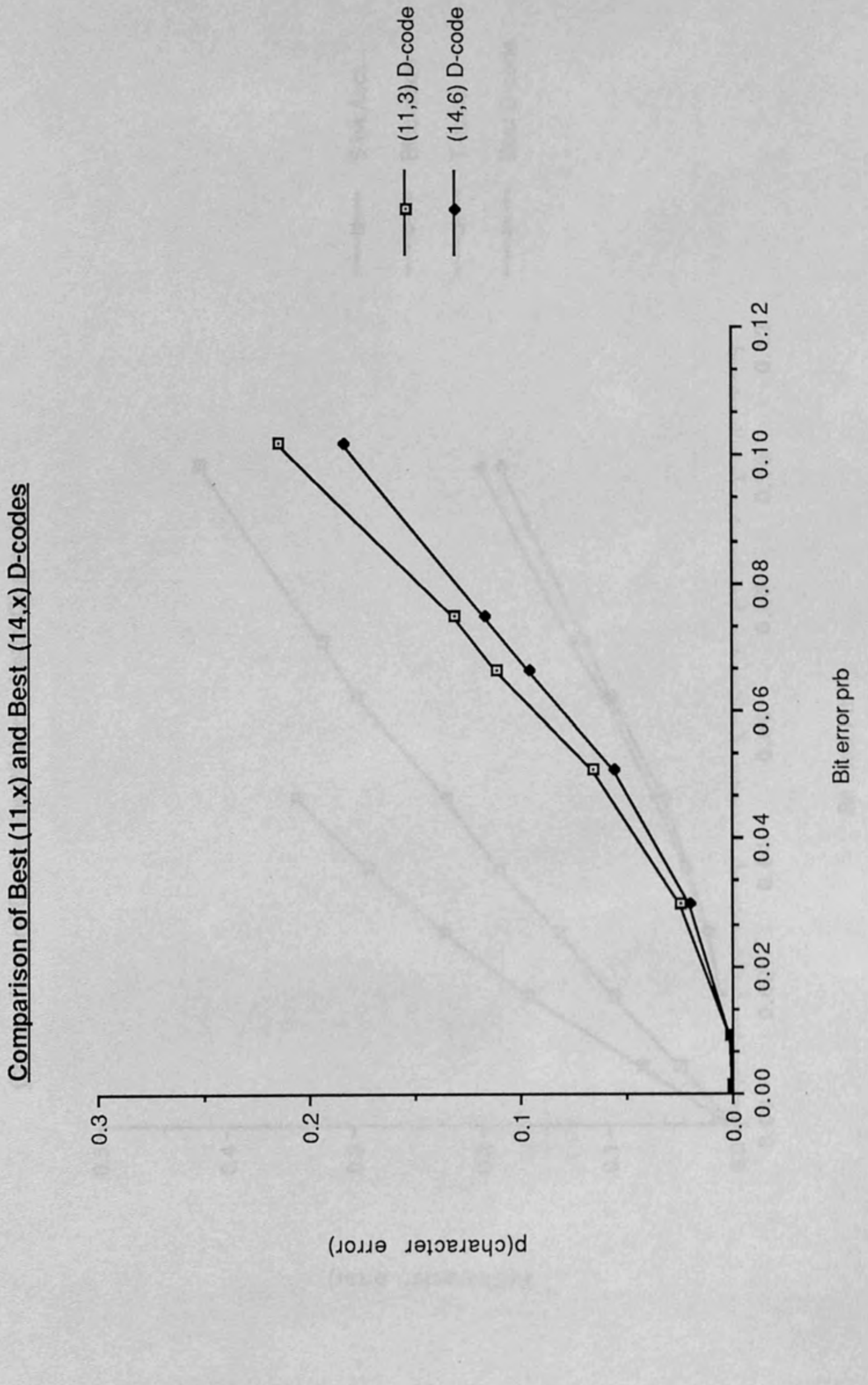

Comparison of (14,x) D-codes
(14,5) D-code
(14,6) D-code
P(character error)
0.0
0.1
0.2
Bit Err Prob
0.00
0.02
0.04
0.06
0.08
0.10
0.12

Figure 5.7



Comparison of Best (11,x) and Best (14,x) D-codes
(11,3) D-code
(14,6) D-code
p(character error)
0.0
0.1
0.2
0.3
Bit error prb
0.00
0.02
0.04
0.06
0.08
0.10
0.12

Figure 5.8

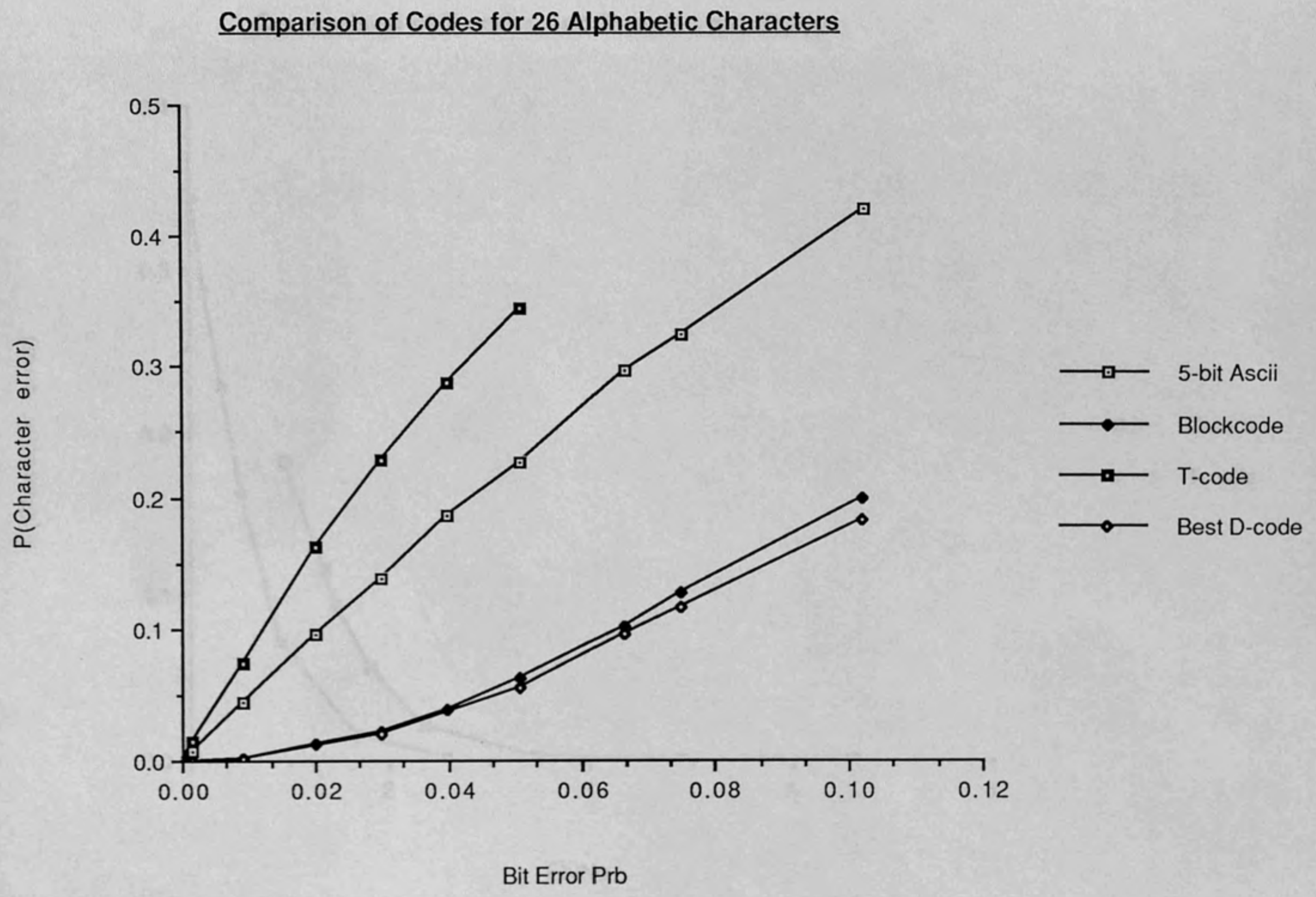

Comparison of Codes for 26 Alphabetic Characters
P(Character error)
0.5
0.4
0.3
0.2
0.1
0.0
0.00
0.02
0.04
0.06
0.08
0.10
0.12
Bit Error Prb
5-bit Ascii
Blockcode
T-code
Best D-code

Figure 5.9

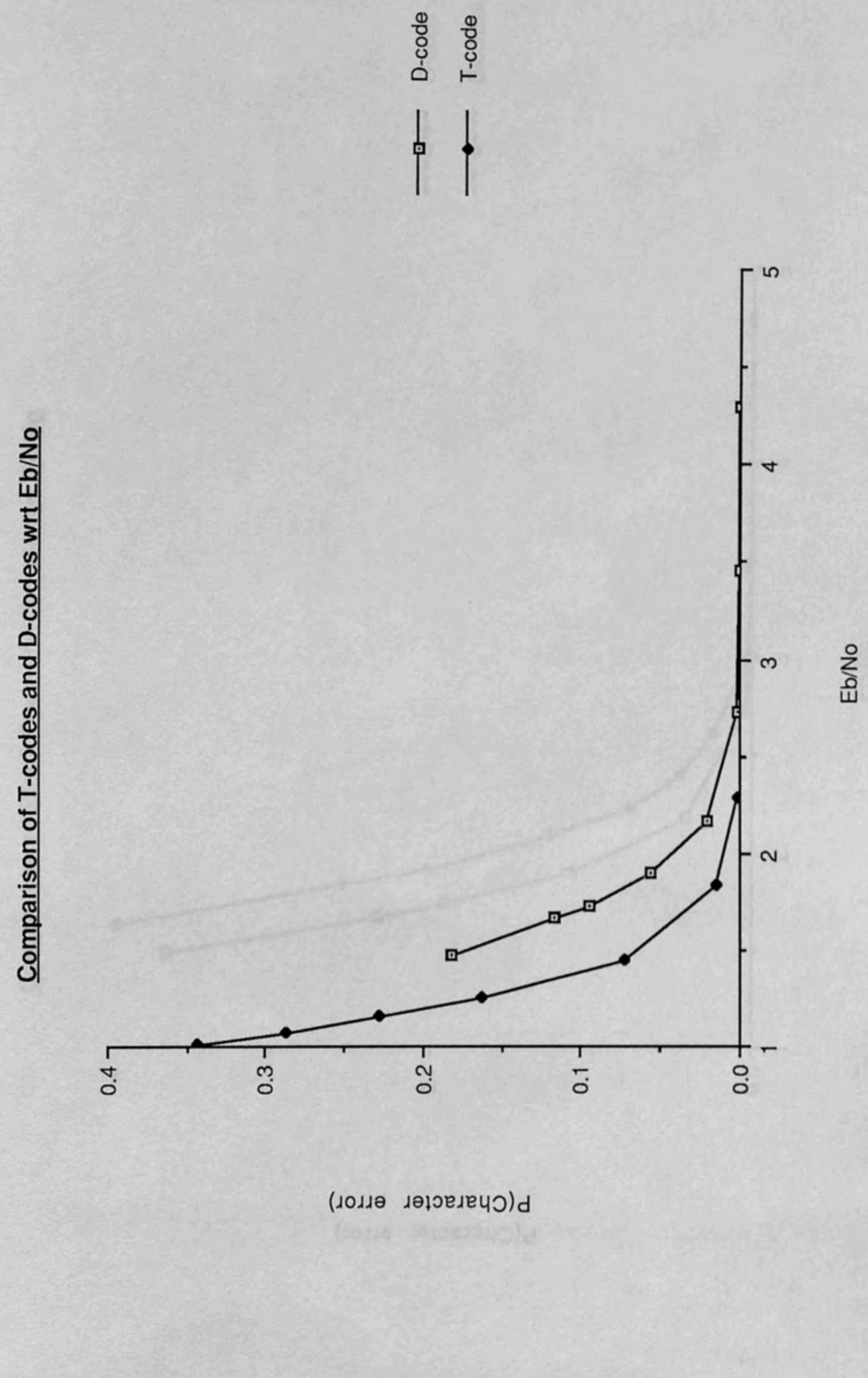

Comparison of T-codes and D-codes wrt Eb/No
D-code
T-code
P(Character error)
0.4
0.3
0.2
0.1
0.0
1
2
3
4
5
Eb/No

Figure 5.10

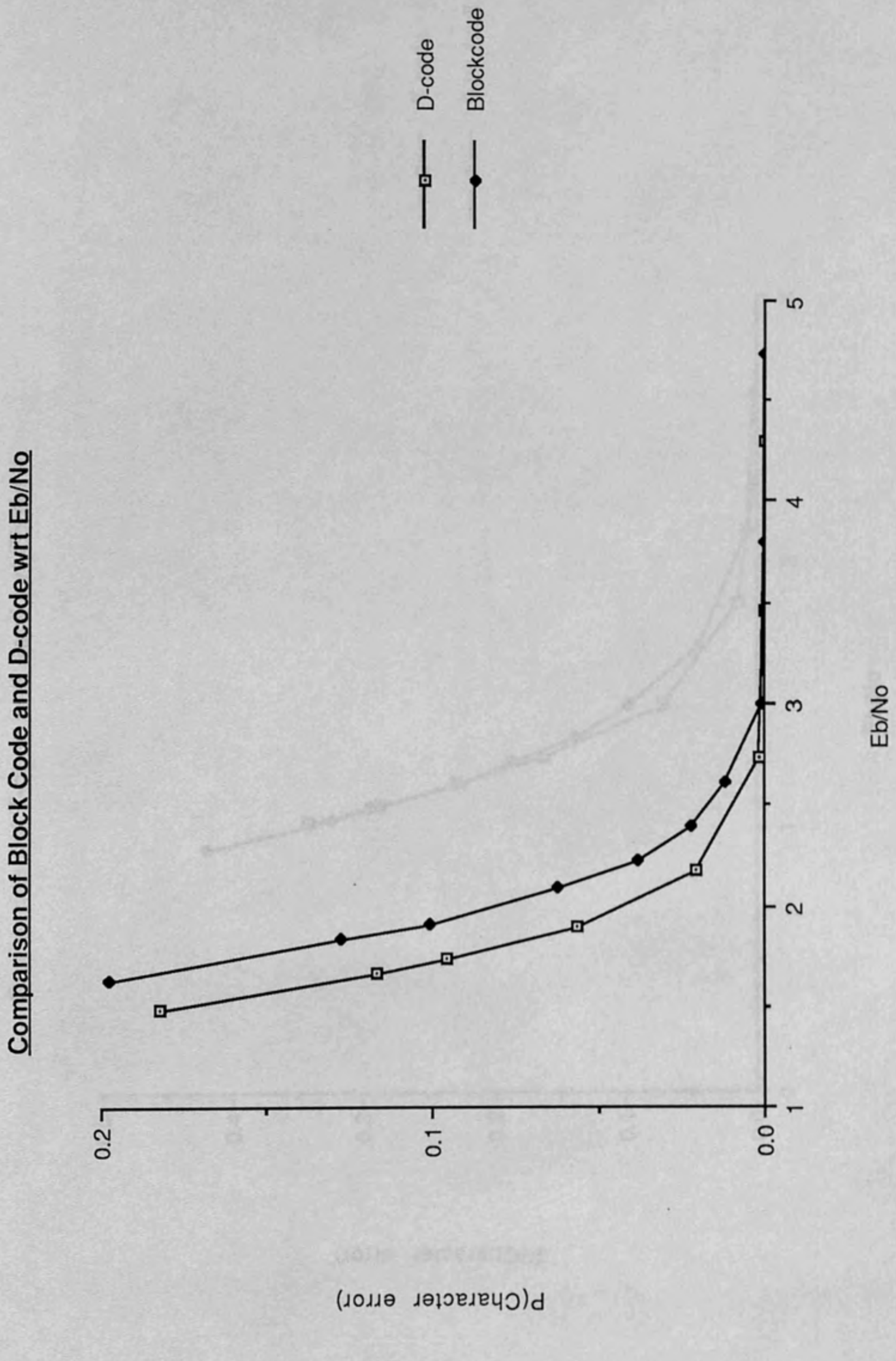

Comparison of Block Code and D-code wrt Eb/No
P(Character error)
0.2
0.1
0.0
1
2
3
4
5
Eb/No
D-code
Blockcode

Figure 5.11

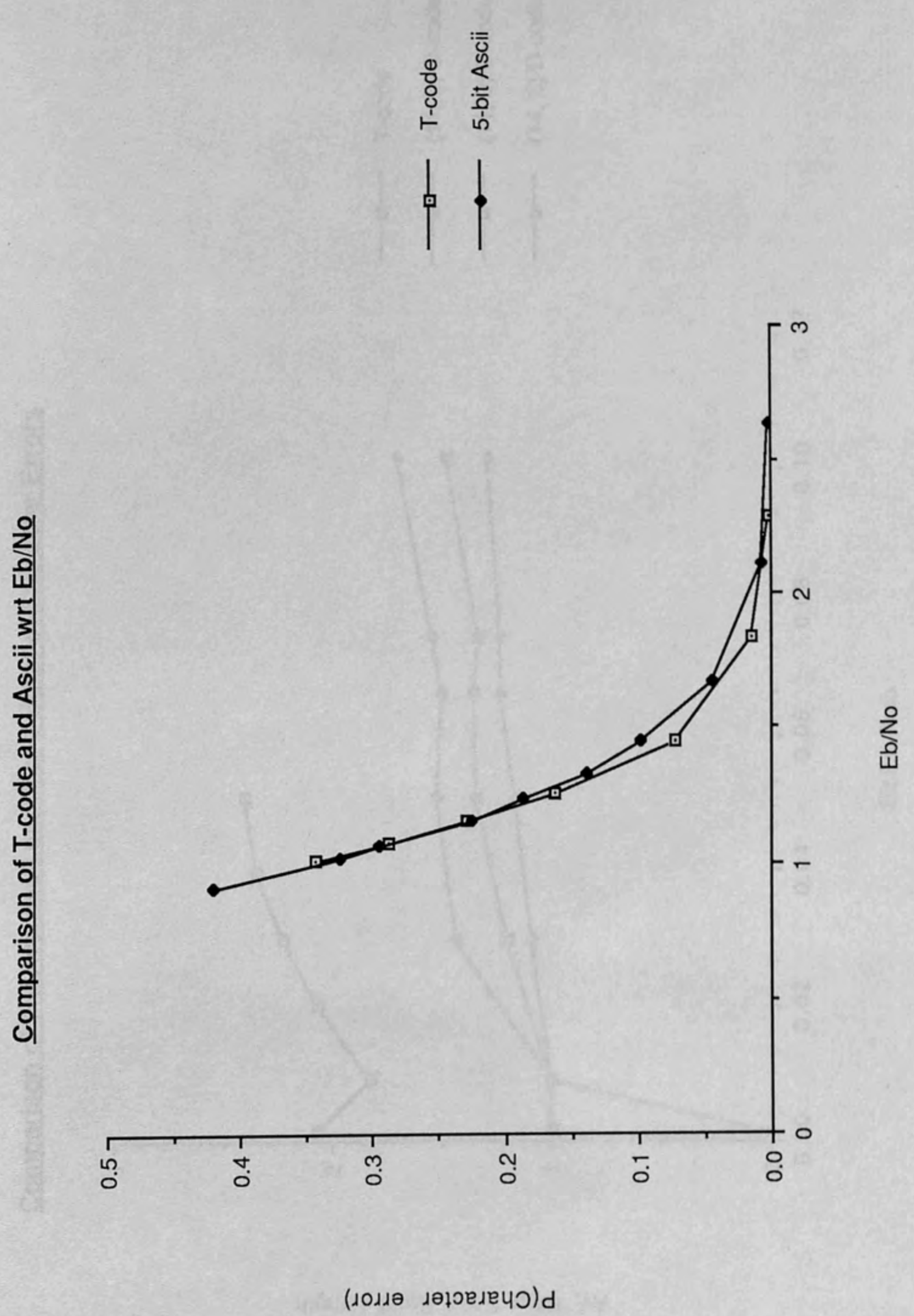

Comparison of T-code and Ascii wrt Eb/No
P(Character error)
0.5
0.4
0.3
0.2
0.1
0.0
0
1
2
3
Eb/No
T-code
5-bit Ascii

Figure 5.12

Comparison of the Average Length of a Burst of Character Errors

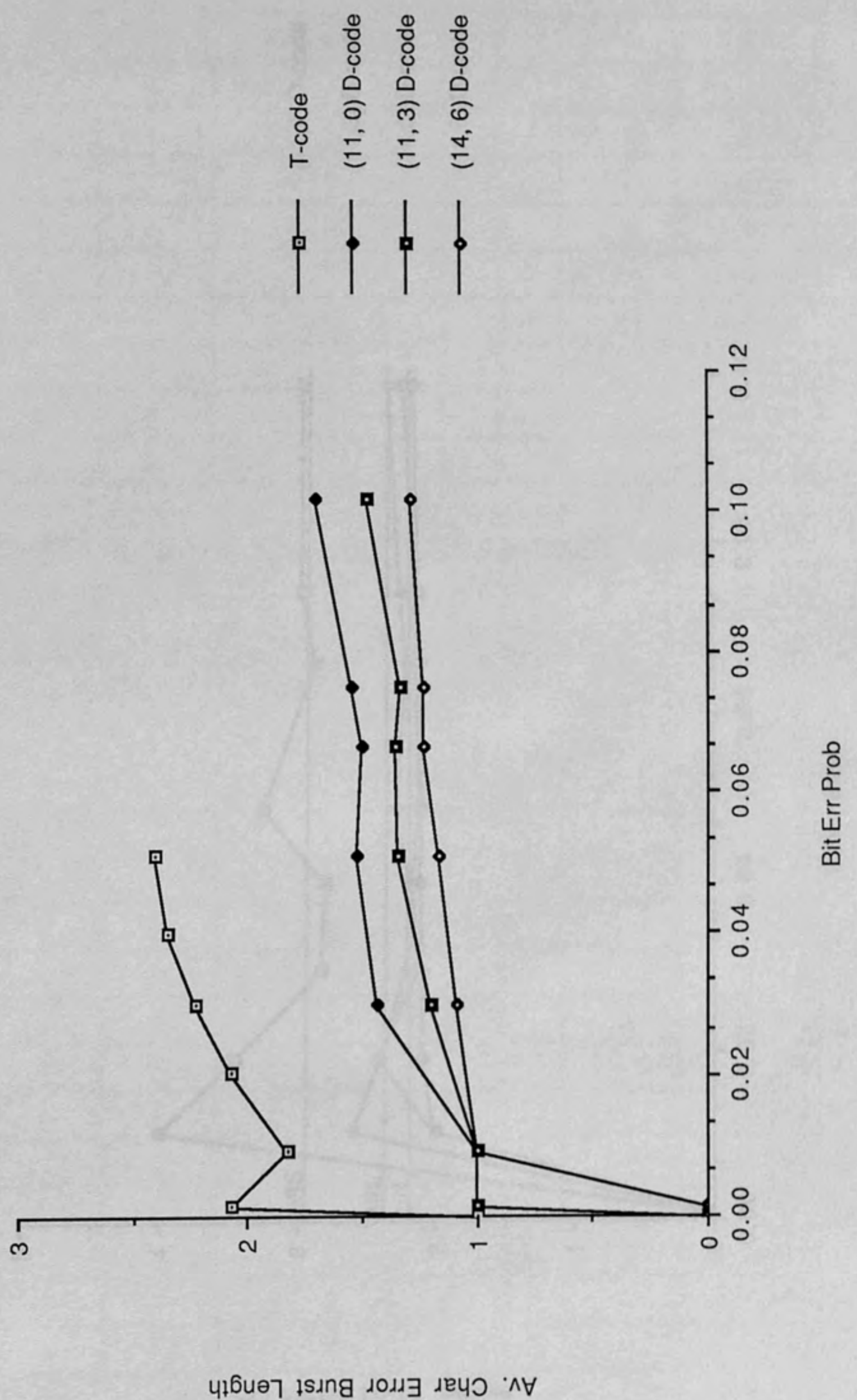


Figure 5.13

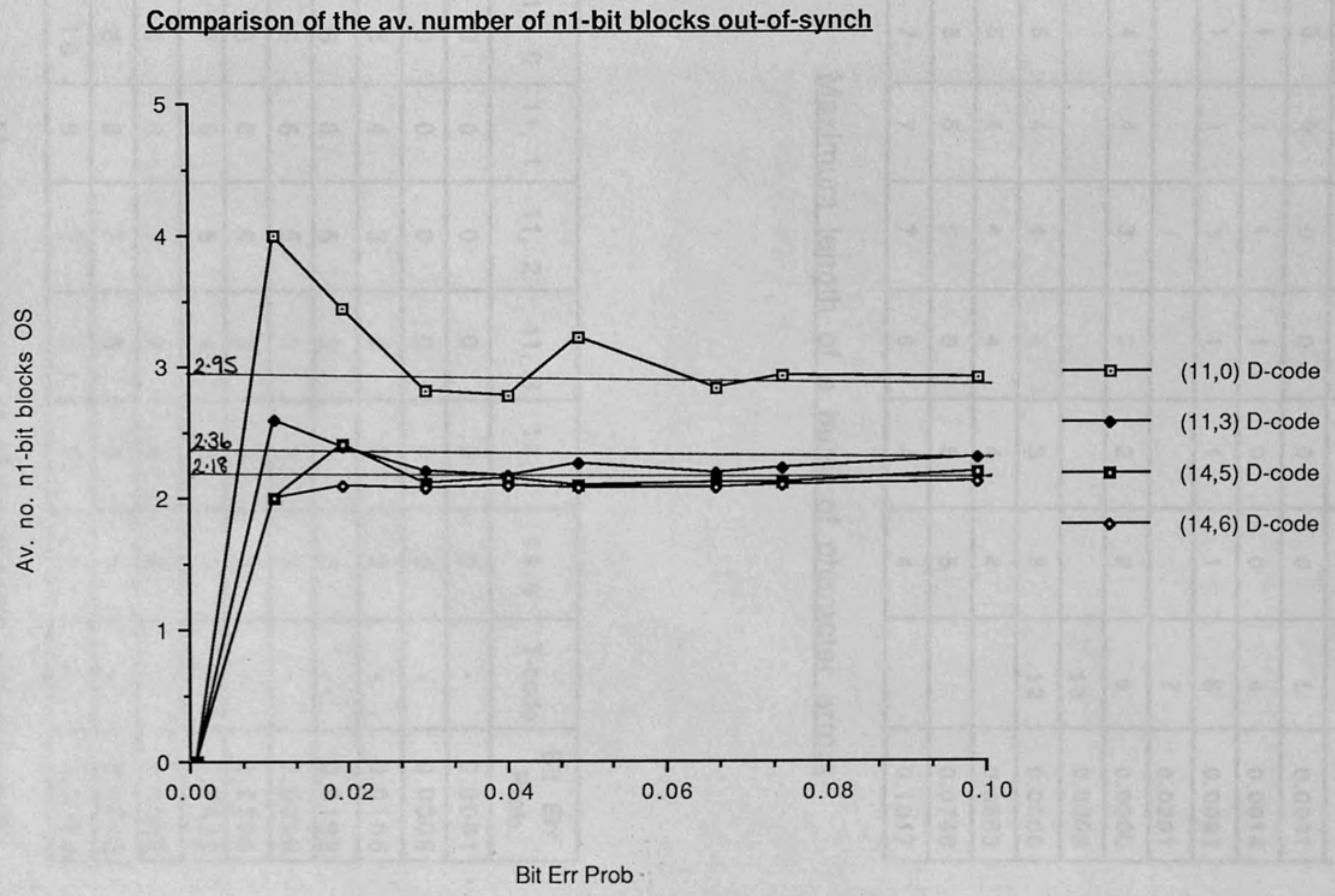

Comparison of the av. number of n1-bit blocks out-of-synch
Av. no. n1-bit blocks OS
Bit Err Prob
0
1
2
3
4
5
0.00
0.02
0.04
0.06
0.08
0.10
2·95
2·36
2·18
(11,0) D-code
(11,3) D-code
(14,5) D-code
(14,6) D-code

Figure 5.14

11,0	11,1	11,2	11,3	14,5	14,6	T-code	Bit Err Prob
0	0	0	0	0	0	1	0.0001
1	1	1	1	0	0	4	0.0014
1	1	1	1	1	1	5	0.0093
						7	0.0201
4	4	3	2	2	2	9	0.0300
						13	0.0398
5	4	4	4	3	3	12	0.0506
5	4	4	4	4	4		0.0663
8	5	5	5	5	5		0.0748
7	7	7	6	4	4		0.1017

Maximum length of a burst of character errors

11, 0	11, 1	11, 2	11, 3	14,5	14,6	T-code	Bit Err Prob
0	0	0	0	0	0	-	0.0001
0	0	0	0	0	0	-	0.0008
6	4	3	3	2	2	-	0.0106
6	8	5	3	4	3	-	0.0192
6	6	5	5	3	3	-	0.0298
6	8	5	3	3	3	-	0.0396
7	6	5	4	3	3	-	0.0490
6	8	7	7	4	4	-	0.0662
8	8	7	5	4	3	-	0.0743
10	8	7	5	5	5	-	0.0988

Maximum number of n1-bit blocks out-of-synch.

Figure 5.15

REFERENCES

[BS] *Some Combinatorial Results on Variable Length Error-Correcting Codes*, M.A. Bernard and B.D. Sharma, BCC 1987.

[Bl] *Theory and Practice of Error Control Codes*, R. E. Blahut, Addison Wesley.

[BPP] *The Application of Error Control to Communications*, E.R. Berlekamp, R.E. Peile and S.P. Pope, IEEE Comms Magazine, Vol.25, No.4, April 1987, p.44-57.

[Do] *The Digital Cellular SHF 900 System*, J. L. Dornstetter.

[Do,85] *French Patent Application No. 8508098*, J. L. Dornstetter, filed 30/05/85.

[GGW] *Comma-free codes*, S.W. Golomb, B. Gordon and L.R. Welch, Can. J. Math., 1958, 10, p.202-209.

[GS] *Probability and Random Processes*, G.R. Grimmett and D.R. Stirzaker, Oxford Science Publications, 1982.

[Ha] *Foundations of Coding Theory*, W.E. Hartnett (ed), D. Reidling Publishing Co., Dordretcht, Holland, 1974.

[McE] *The Theory of Information and Coding*, R.J. McEliece, Encyclopedia of Mathematics and it Applications, vol 3, Addison Wesley.

[MWS] *The Theory of Error-Correcting Codes*, F. J. MacWilliams and N. J. A. Sloane, North Holland.

[Pr] *Digital Communications*, J.G. Proakis, McGraw-Hill, 1983.

[PW] *Error-Correcting Codes*, W. W. Peterson and E. J. Weldon, MIT Press.

[St] *Multiple-Burst Error Correction with the Chinese Remainder Theorem*, J. J. Stone, J. Soc. Indust. Appl. Math., Vol. 11, No. 1, March 1963, p.74-81.

[Ti] *Digital Encoding by means of new T-Codes to provide improved data synchronization and message integrity*, M.R. Titchener, IEE Proceedings, vol.131, Pt.E, No.4, July 1984, p.151-153.

[Ve] *Updated Table of Upper and Lower Bounds on $d_{MAX(n,k)}$*, T. Verhoeff, IEEE Trans. IT, vol. IT-33, no. 5, Sept. 1987, p.666.

```
program cohn (output,resfile);

(* This program evaluates the probability of correct    *)
(* decoding for both traditional standard array         *)
(* decoding and a variation on this by Prof. P.M.Cohn   *)
(* Final Version : 10th December 1987  *)

type
  matrix = array [3..100,0..100] of real;

var
  ncr                               : matrix;
  n,k,t,mink,maxt                   : integer;
  Cohn,Trad,x,lastx,minp,maxp       : real;
  resfile                           : text;
  satisfied                         : boolean;

procedure calcncr;

(* Calculates and places in the array ncr the values *)
(* of n choose r for all n and r between 3 and 100.  *)

var
  n,r : integer;

begin {ncr}
  for n:= 3 to 100 do
    begin
      ncr[n,0]:=1;
      for r:=1 to n do
        ncr[n,r]:=ncr[n,r-1] * (n-r+1)/r
    end
end; {ncr}

procedure calcrhs (n,k : integer; x:real;
                   var Pc : real);

(* Calculates the probability of correct decoding *)
(* for Cohn's scheme  *)

var
  i : integer;

begin {calcrhs}
  Pc:=1;
  for i:=1 to (n-k) do
    Pc:=Pc * (1+ x)
end; {calcrhs}

procedure calclhs (n,t : integer; x:real;
                   var Pc : real);
```

```
procedure calclhs (n,t : integer; x:real;
                   var Pc : real);

(* Calculates the probability of correct decoding for *)
(* the traditional scheme *)

var
  i : integer;
  power : real;

begin {calclhs}
  Pc :=1;
  power := 1;
  for i:= 1 to t do
    begin
      power:=power*x;
      Pc := Pc + ncr[n,i]*power
    end
end; {calclhs}

begin {main}
  calcncr;
  rewrite (resfile);
  writeln (resfile,'n':4,'k':4,'t':4,'Min p':7,
                                                 'Max p':7);
  for n:=3 to 100 do
    begin
      mink:= n div 2;
      for k :=mink to (n-2) do
        begin
          maxt:= (n-k) div 2;
          for t:=1 to maxt do
            begin
              lastx:=0;
              x:=0.01;
              satisfied:=false;
              while ( not(satisfied) and (x<=(1/9))) do
                begin
                  calclhs (n,t,x,Trad);
                  calcrhs (n,k,x,Cohn);
                  if Cohn > Trad then
                    begin
                      satisfied:=true;
                      minp:= lastx/(1 + lastx);
                      maxp:= x/(1+x);
                      writeln (resfile,n:4,k:4,t:4,
                                              minp:7,maxp:7)
                    end;
                  lastx:=x;
                  x:=x+0.01
                end
            end
        end
    end;
  close (resfile);
  writeln (chr(7))
end.
```

```
program asciisim (input,ascicode,message,datasent,
           letterorder,datarcvd,ascifinal,errors,output);

(* This program encodes the English text of the file   *)
(* message in 5-bit ASCII, writing the resulting bits  *)
(* in the file datasent. The contents of this file are *)
(* then added sequentially to the bits in the file     *)
(* errors, the bitwise sums being placed in the file   *)
(* datarcvd. The bits of this file are decoded in 5-bit*)
(* blocks to the appropriate English character.        *)

type
  col       = array [1..26] of char;
  word      = packed array [1..5] of char;
  codearray = array [1..26] of word;

var
  ascicode,message,errors,datasent,datarcvd,
                      ascifinal,letterorder : text;
  codewords                                 : codearray;
  c                                         : col;
  bitcount,charcount                        : integer;
  biterrp                                   : real;

procedure progheader;

{writes initial info to the screen}

begin{procedure progheader}
  writeln('This program simulates a 5-bit Ascii code for
                                             the English');
  writeln('capital letters');
  writeln('Please supply the bit error probability of the
                                      current error file');
  readln(biterrp)
end;{procedure progheader}

procedure setupcode (var infile : text;
                     var code : codearray);

{copies the codewords from infile to the array code}

var
  i,count : integer;

begin{procedure setupcode}
  reset(infile);
  count:=0;
  while not(eof(infile)) do
    begin
      count:=count + 1;
      for i:= 1 to 5 do
         read(infile,code[count,i]);
      readln(infile);
    end
end;{procedure setupcode}

procedure writew (var outfile : text; cwno : integer);

{writes a single codeword to outfile}

var
  i : integer;

begin{procedure writecw}
  for i:= 1 to 5 do
     write(outfile,codewords[cwno,i])
end;{procedure writecw}

procedure writecword (var outfile :text; ch:char);

{writes the codeword corresponding to ch to outfile}

begin{procedure writeword}
  case ch of
    'E': writew(outfile,1);
    'T': writew(outfile,2);
    'A': writew(outfile,3);
    'O': writew(outfile,4);
    'I': writew(outfile,5);
    'N': writew(outfile,6);
    'S': writew(outfile,7);
    'H': writew(outfile,8);
    'R': writew(outfile,9);
    'D': writew(outfile,10);
    'L': writew(outfile,11);
    'C': writew(outfile,12);
    'U': writew(outfile,13);
    'M': writew(outfile,14);
    'W': writew(outfile,15);
    'F': writew(outfile,16);
    'G': writew(outfile,17);
    'Y': writew(outfile,18);
    'P': writew(outfile,19);
    'B': writew(outfile,20);
    'V': writew(outfile,21);
    'K': writew(outfile,22);
    'J': writew(outfile,23);
    'X': writew(outfile,24);
    'Q': writew(outfile,25);
    'Z': writew(outfile,26);
  end{case}
end;{procedure writecword}

procedure encode (var infile,outfile : text);

{reads an English message from infile and writes to}
{outfile a codeword for each alphabetic character}

var
```

```
count : integer;
  ch    : char;

begin{procedure encode}
  reset(infile);
  rewrite(outfile);
  bitcount:=0;
  charcount:=0;
  while not(eof(infile)) do
   begin
     count:=0;
     repeat
       read(infile,ch);
       if (ch in ['a'..'z']) then
         ch:=chr(ord(ch)-32);
       if (ch in ['A'..'Z']) then
         begin
           charcount:=charcount+1;
           writecword(outfile,ch);
           count:=count+1
         end
     until ((count=10) or (eof(infile)));
     writeln(outfile);
   end;
  close(outfile)
end;{procedure encode}

procedure adderrors(var infile1,infile2,outfile : text);

{reads bitwise from infile1 and infile2 and writes the}
{mod 2 sum of each pair to outfile}

var
  ch1,ch2 : char;

begin
  reset(infile1);
  reset(infile2);
  rewrite(outfile);
  while not(eof(infile1)) do
    begin
      while not((eoln(infile1)) or (eoln(infile2))) do
        begin
          read(infile1,ch1);
          read(infile2,ch2);
          if (ch1=ch2) then
            write(outfile,'0')
          else
            write(outfile,'1')
        end;
      if eoln(infile1) then
        begin
          readln(infile1);
          writeln(outfile)
        end
      else
```

```
        readln(infile2)
      end;
    close(outfile)
  end;

procedure setletord (var order:col);

{ Reads the alphabetic characters in order of }
{ decreasing probability }

var
  i : integer;

begin{procedure setletord}
  reset(letterorder);
  for i:=1 to 26 do
    readln(letterorder,order[i])
end;{procedure setletord}

function compare(w1,w2:word):boolean;

begin{function compare}
  compare:=w1=w2
end;{function compare}

procedure decode (var infile,outfile :text);

{Reads bits from infile in 5-bit blocks and writes the}
{corresponding character to outfile}

var
  recword  : word;
  i,j,count,linecount   : integer;
  match                 : boolean;

begin
  reset(infile);
  rewrite(outfile);
  writeln(outfile,' Bit error probability = ', biterrp);
  writeln(outfile, 'Average wordlength = ',
                                          bitcount/charcount);
  match:=false;
  for i:=1 to 5 do
    read(infile,recword[i]);
  linecount:=0;
  while not(eof(infile)) do
    begin
      i:=0;
      repeat
        i:=i+1;
        match:=compare(recword,codewords[i])
      until ((match) or (i=26));
      if match then
        begin
          write(outfile,c[i]);
          match:=false
          end
        else
          write(outfile,'?');
        linecount:=linecount+1;
        if (linecount = 79) then
          begin
            linecount:=0;
            writeln(outfile)
          end;
        if not(eof(infile)) then
          begin
            if eoln(infile) then
              readln(infile);
            if not(eof(infile)) then
              for i:= 1 to 5 do
                read(infile,recword[i])
          end
      end;
    close(outfile)
  end;

  begin{main body}
    progheader;
    setupcode(ascicode,codewords);
    encode(message,datasent);
    adderrors(datasent,errors,datarcvd);
    setletord(c);
    decode(datarcvd,ascifinal);
    writeln(chr(7))
  end.
```

```
program blocksim (input,blockcode,message,trans,
                  letterorder,mat2, received,firstdecode,
                              errors,blockfinal,output);

(* This program encodes the English text of the file   *)
(* message in a (9,5) blockcode, writing the resulting *)
(* bits in the file datasent. The contents of this file*)
(* are then added sequentially to the bits in the file *)
(* errors, the bitwise sums being placed in the file   *)
(* datarcvd. The bits of this file are decoded in 9-bit*)
(* blocks to a codeword which is then decoded to the   *)
(* appropriate English character.                      *)


type
  col       = array [1..26] of char;
  word      = packed array [1..9] of char;
  codearray = array [1..26] of word;

var
  blockcode,message,errors,trans,received,mat2,
      blockfinal,firstdecode,letterorder      : text;
  codewords                                   : codearray;
  c                                           : col;
  bitcount,charcount                          : integer;
  biterrp                                     : real;


procedure progheader;

{writes initial info to the screen}

begin{procedure progheader}
  writeln('This program simulates a 1-error-correcting
                                          blockcode');
  writeln('for the English capital letters');
  writeln('Please supply the bit error probability of the
                                  current error file');
  readln(biterrp)
end;{prcedure progheader}

procedure setupcode (var infile : text;
                      var code : codearray);

{copies the codewords from infile to the array code}


var
  i,count : integer;

begin{procedure setupcode}
  reset(infile);
  count:=0;
  while not(eof(infile)) do
    begin
      count:=count + 1;
      for i:= 1 to 9 do
         read(infile,code[count,i]);
      readln(infile);
    end
end;{procedure setupcode}

procedure writew (var outfile : text; cwno : integer);

{writes a single codeword to outfile}

var
  i : integer;

begin{proceure writecw}
  for i:= 1 to 9 do
     write(outfile,codewords[cwno,i]);
  bitcount:=bitcount+9
end;{procedure writecw}

procedure writecword (var outfile :text; ch:char);

{writes the codeword corresponding to ch to outfile}

begin{procedure writeword}
  case ch of
    'E':writew(outfile,1);
    'T':writew(outfile,2);
    'A':writew(outfile,3);
    'O':writew(outfile,4);
    'I':writew(outfile,5);
    'N':writew(outfile,6);
    'S':writew(outfile,7);
    'H':writew(outfile,8);
    'R':writew(outfile,9);
    'D':writew(outfile,10);
    'L':writew(outfile,11);
    'C':writew(outfile,12);
    'U':writew(outfile,13);
    'M':writew(outfile,14);
    'W':writew(outfile,15);
    'F':writew(outfile,16);
    'G':writew(outfile,17);
    'Y':writew(outfile,18);
    'P':writew(outfile,19);
    'B':writew(outfile,20);
    'V':writew(outfile,21);
    'K':writew(outfile,22);
    'J':writew(outfile,23);
    'X':writew(outfile,24);
    'Q':writew(outfile,25);
    'Z':writew(outfile,26);
  end{case}
end;{procedure writecword}
```

```
procedure encode (var infile,outfile : text);

{reads an English message from infile and writes to
{outfile a codeword for each alphabetic character}

var
  count : integer;
  ch    : char;

begin{procedure encode}
  reset(infile);
  rewrite(outfile);
  bitcount:=0;
  charcount:=0;
  while not(eof(infile)) do
    begin
      count:=0;
      repeat
        read(infile,ch);
        if (ch in ['a'..'z']) then
          ch:=chr(ord(ch)-32);
        if (ch in ['A'..'Z']) then
          begin
            charcount:=charcount+1;
            writecword(outfile,ch);
            count:=count+1
          end
      until ((count=10) or (eof(infile)));
      writeln(outfile);
    end;
  close(outfile)
end;{procedure encode}

procedure adderrors(var infile1,infile2,outfile : text);

{reads bitwise from infile1 and infile2 and writes the
{mod 2 sum of each pair to outfile}

var
  ch1,ch2 : char;

begin
  reset(infile1);
  reset(infile2);
  rewrite(outfile);
  while not(eof(infile1)) do
    begin
      while not((eoln(infile1)) or (eoln(infile2))) do
       begin
          read(infile1,ch1);
          read(infile2,ch2);
          if (ch1=ch2) then
            write(outfile,'0')
          else
            write(outfile,'1')
        end;
      if eoln(infile1) then
        begin
          readln(infile1);
          writeln(outfile)
        end
      else
        readln(infile2)
    end;
  close(outfile)
end;

procedure fstdecode (var infile,outfile : text);

{reads a block of 9 bits from infile, simulates a}
{decoder and outputs the resulting codeword to outfile.}
{Repeats for whole infile}

var
 recword          : array [1..9] of integer;
 mat              : array [1..4,1..9] of integer;
 s                : array [1..4] of integer;
 found            : boolean;
 i,j,count        : integer;
 ch               : char;

procedure setupmatrix;

{reads the parity check matrix of the block code into
{the array mat}

var
  i,j : integer;

begin{procedure setupmatrix}
  reset(mat2);
  for i:=1 to 4 do
    begin
      for j:= 1 to 9 do
        read(mat2,mat[i,j]);
      readln(mat2)
    end
end;{procedure setupmatrix}

procedure correct (i,j : integer);

begin{procedure correct}
  recword[i]:=(recword[i]+1) mod 2;
  recword[j]:=(recword[j]+1) mod 2
end;{procedure correct}

begin{procedure fstdecode}
  reset(infile);
  rewrite(outfile);
  setupmatrix;
  count:=0;
```

```
while not(eof(infile)) do
   begin
     while not(eoln(infile)) do
       begin
         for i:= 1 to 9 do
           begin
             read(infile,ch);
             recword[i]:= ord(ch) - ord('0')
           end;
         for i := 1 to 4 do
           s[i]:=0;
         for i:= 1 to 4 do
           begin
             for j:= 1 to 9 do
               s[i]:=s[i] + (mat[i,j] * recword[j]);
             s[i]:=s[i] mod 2
           end;
         count:=count+1;
         if ((s[1]=0) and (s[2]=0) and (s[3]=0) and
                                       (s[4]=0)) then
           for i:=1 to 9 do
             write(outfile,recword[i]:1)
         else
           begin
             found:=false;
             i:=0;
             repeat
               i:=i+1;
               if ((mat[1,i]=s[1]) and (mat[2,i]=s[2])
                   and
                     (mat[3,i]=s[3]) and (mat[4,i]=s[4]))
               then
                found:=true
             until (found or (i=9));
             if found then
               recword[i]:=(recword[i] + 1) mod 2
             else
                 if ((s[1]=1) and (s[2]=1) and(s[3]=0)
                                              and (s[4]=1))
                 then
                   correct(1,9)
                 else
                   if ((s[1]=0) and (s[2]=1) and
                                 (s[3]=1) and (s[4]=1))
                   then
                     correct(4,5)
                   else
                     if ((s[1]=0) and (s[2]=0) and
                                 (s[3]=1) and (s[4]=1))
                     then
                       correct(8,9)
                     else
                       if ((s[1]=1) and (s[2]=0) and
                                 (s[3]=1) and (s[4]=1))
                       then correct(2,9)
                       else
```

```
                              if ((s[1]=0) and (s[2]=1) and
                                  (s[3]=0) and (s[4]=1))
                              then correct(1,5)
                              else
                                correct(3,5);
              for i:=1 to 9 do
                write(outfile,recword[i]:1)
              end;
            if (count=8) then
              begin
                writeln(outfile);
                count:=0
              end;
          end;
          readln(infile)
      end;
    writeln(outfile);
    close(outfile);

end;{procedure fstdecode}

procedure setletord (var order:col);


var
  i : integer;

begin{procedure setletord}
  reset(letterorder);
  for i:=1 to 26 do
    readln(letterorder,order[i])
end;{procedure setletord}

function compare(w1,w2:word):boolean;

begin{function compare}
  compare:=w1=w2
end;{function compare}

procedure decode (var infile,outfile :text);

{Matches codewords to English characters}

var
  recword  : word;
  i,j,count,linecount   : integer;
  match                 : boolean;

begin
  reset(infile);
  rewrite(outfile);
  writeln(outfile,' Bit error probability = ', biterrp);
  writeln(outfile, 'Average wordlength = ',
 bitcount/charcount);
  match:=false;
  for i:=1 to 9 do
    read(infile,recword[i]);
  linecount:=0;
  while not(eof(infile)) do
    begin
      i:=0;
        repeat
          i:=i+1;
          match:=compare(recword,codewords[i])
        until ((match) or (i=26));
        if match then
          begin
            write(outfile,c[i]);
            match:=false
          end
        else
          write(outfile,'?');
          linecount:=linecount+1;
        if (linecount = 79) then
            begin
              linecount:=0;
               writeln(outfile)
            end;
          if not(eof(infile)) then
            begin
              if eoln(infile) then
                 readln(infile);
            if not(eof(infile)) then
               for i:= 1 to 9 do
                  read(infile,recword[i])
           end
    end;
  close(outfile)
end;

begin{main body}
  progheader;
  setupcode(blockcode,codewords);
  encode(message,trans);
  adderrors(trans,errors,received);
  fstdecode(received,firstdecode);
  setletord(c);
  decode(firstdecode, blockfinal);
  writeln(chr(7))
end.
```

```
program twolgthsim (input,basecode,letterorder,message,
     trans,errors,received,abmess,abfinal,firstdecode,
                               dfinal,matfile,output);

(* This program asks which (S,x) D-code is required and*)
(* then forms this code from Hamming (7,4) codewords,  *)
(* which are stored in the file basecode.It            *)
(* then encodes the English text of the file message in*)
(* this code, writing the resulting bits in the file   *)
(* datasent. The contents of this file are then added  *)
(* sequentially to the bits in the file errors, the    *)
(* bitwise sums being placed in the file datarcvd. The *)
(* bits of this file are decoded in 7-bit blocks to a  *)
(* Hamming (7,4) codeword. These codewords are then    *)
(* into short and long codewords which are then decoded*)
(* to the appropriate English character.               *)


type

  col       = array [1..26] of char;
  n1bits    = array [1..7] of integer;
  word      = array [1..14] of char;
  wordarray = array [1..26] of word;

var

  basecode,letterorder,message,trans,errors,received,
  firstdecode,dfinal,matfile,abmess,abfinal
                                             : text;
  sizew1,sizew2,r,numbaddons,bitcount,charcount
                                             : integer;
  valid                                      : boolean;
  codewords                                  : wordarray;
  c                                          : col;
  biterrp                                    : real;


procedure getinfo (var size1,size2,ratio,noadds
                                               : integer;
                   var valid : boolean);

{Prints and reads user information and data}

begin {procedure getinfo}
  valid:=true;
  writeln('This program simulates the performance of a
                                              2-length ');
  writeln('error-correcting code for the English
                               language, formed from');
  writeln('the Hamming (7,4) code with the length of the
                                      long codewords');
```

```
    writeln('being twice that of the short codewords.');
    writeln;
    writeln('Please enter the number of short codewords,
                                                  (11 or 14):');
    readln(size1);
    if ((size1 <> 11) and (size1 <> 14)) then
      valid:=false;
    size2:=16 - size1;
    writeln('Please enter the number of short codewords to
                                               be used as addons');
    writeln('for each prefix:');
    readln(ratio);
    writeln('Please supply bit error probability of current
                                                   error file');
    readln(biterrp);
    noadds:= (26 - size1) div size2;
    if ((ratio < 0) or (ratio > noadds)) then
      valid:=false;
    if ((noadds - ratio) > size2) then
      valid:=false
  end; {procedure getinfo}

  procedure formcode (var code : wordarray;
                      var size1,size2,r,noadds : integer);

  {calculates the codewords according to specifications of}
  {getinfo and writes them to the array codewords}

  var
    w1no,i,j,count,next : integer;

  begin{procedure formcode}
    reset(basecode);
    for i:= 1 to size1 do
      begin
        for j:= 1 to 7 do
          read(basecode,code[i,j]);
        for j:= 8 to 14 do
          code[i,j]:=' ';
        readln(basecode)
      end;
    count:=size1 + 1;
    while (count <= 26) do
      begin
        for j:= 1 to 7 do
          read(basecode,code[count,j]);
        readln(basecode);
        for i:= 1 to (noadds-1) do
          for j:= 1 to 7 do
            code[count+i,j]:=code[count,j];
        count:=count+noadds
      end;
    count:=size1 + 1;
    w1no:=1;
    while (count <= 26) do
      begin
      for i:= 1 to r do
        begin
          if (w1no > sizew1) then
            w1no:=1;
          for j:= 8 to 14 do
            code[count,j]:=code[w1no,j-7];
          count:=count+1;
          w1no:=w1no+1
        end;
      next:=count+noadds;
      for i:= (r+1) to noadds do
        begin
          if (next > 26) then
            next:=size1 + next - 26;
          for j:= 8 to 14 do
            code[count,j]:=code[next,j-7];
          next:=next + noadds;
          count:=count+1
        end
    end
end;{procedure formcode}

procedure writeword (var outfile : text;
                         cwno : integer);

{writes to outfile 1 codeword according to its size}

var
  i:integer;

begin {procedure writeword}
  if (cwno <= sizew1) then
    begin
      for i:= 1 to 7 do
        write(outfile,codewords[cwno,i]:1);
      bitcount:=bitcount+7;
      write(abmess,'s')
    end
  else
    begin
      for i:= 1 to 14 do
        write(outfile,codewords[cwno,i]:1);
      bitcount:=bitcount+14;
      write(abmess,'lo')
    end
end;{procedure writeword}

procedure writecw (var outfile:text; ch:char);

{writes out the codeword for a given letter}

begin {procedure writecw}
  case ch of
   'E':writeword(outfile,1);
   'T':writeword(outfile,2);
   'A':writeword(outfile,3);
```

```
    'O':writeword(outfile,4);
    'I':writeword(outfile,5);
    'N':writeword(outfile,6);
    'S':writeword(outfile,7);
    'H':writeword(outfile,8);
    'R':writeword(outfile,9);
    'D':writeword(outfile,10);
    'L':writeword(outfile,11);
    'C':writeword(outfile,12);
    'U':writeword(outfile,13);
    'M':writeword(outfile,14);
    'W':writeword(outfile,15);
    'F':writeword(outfile,16);
    'G':writeword(outfile,17);
    'Y':writeword(outfile,18);
    'P':writeword(outfile,19);
    'B':writeword(outfile,20);
    'V':writeword(outfile,21);
    'K':writeword(outfile,22);
    'J':writeword(outfile,23);
    'X':writeword(outfile,24);
    'Q':writeword(outfile,25);
    'Z':writeword(outfile,26)
  end{case}
end;{procedure writecw}

procedure encode (var infile,outfile : text);

{reads an English message from infile and writes to}
{outfile a codeword for each alphabetic character}

var
  ch    : char;
  count : integer;

begin {procedure encode}
  reset(infile);
  rewrite(outfile);
  rewrite(abmess);
  bitcount:=0;
  charcount:=0;
  while not(eof(infile)) do
    begin
      count:=0;
      repeat
        read(infile,ch);
        if (ch in ['a'..'z']) then
          ch:=chr(ord(ch)-32);
        if (ch in ['A'..'Z']) then
          begin
            writecw(outfile,ch);
            charcount:=charcount+1;
            count:=count+1
          end
      until ((count=5) or (eof(infile)));
      writeln(outfile);
      writeln(abmess)
    end;
  close(outfile);
end;{procedure encode}

procedure adderrors (var infile1,infile2,outfile : text);

{reads bitwise from infile1 and infile2, mod 2 adds each}
{pair of bits and outputs the result to outfile}

var
  b1,b2,i : integer;
        p : real;
       ch : char;

begin
  reset(infile1);
  reset(infile2);
  rewrite(outfile);
  while not(eof(infile1)) do
    begin
      while not((eoln(infile1)) or (eoln(infile2))) do
        begin
          read(infile1,ch);
          b1:=ord(ch)-ord('0');
          read(infile2,ch);
          b2:=ord(ch)-ord('0');
          write(outfile,((b1+b2) mod 2):1)
        end;
      if (eoln(infile1)) then
        begin
          readln (infile1);
          writeln(outfile)
        end
      else
        readln(infile2)
    end;
  close(outfile)
end;{procedure adderrors}

procedure hammingdecode (var infile,outfile : text);

{reads a block of 7 bits from infile, simulates a}
{Hamming decoder and outputs the resulting codeword to}
{outfile. Repeats for whole infile}

var
 recword          : n1bits;
 mat              : array [1..3] of n1bits;
 s                : array [1..3] of integer;
 found            : boolean;
 i,j,count        : integer;
 ch               : char;

procedure setupmatrix;
```

```
{reads the parity check matrix of the Hamming (7,4) code}
{into the array mat}

var
  i,j : integer;

begin{procedure setupmatrix}
  reset(matfile);
  for i:=1 to 3 do
    begin
      for j:= 1 to 7 do
        read(matfile,mat[i,j]);
      readln(matfile)
    end
end;{procedure setupmatrix}

begin{procedure hammingdecode}
  reset(infile);
  rewrite(outfile);
  setupmatrix;
  count:=0;
  while not(eof(infile)) do
    begin
      while not(eoln(infile)) do
        begin
          for i:= 1 to 7 do
            begin
              read(infile,ch);
              recword[i]:= ord(ch) - ord('0')
            end;
          for i:= 1 to 3 do
            s[i]:=0;
          for i:= 1 to 3 do
            begin
              for j:= 1 to 7 do
                s[i]:=s[i] + (mat[i,j] * recword[j]);
              s[i]:=s[i] mod 2
            end;
          count:=count+1;
          if ((s[1]=0) and (s[2]=0) and (s[3]=0)) then
            for i:=1 to 7 do
              write(outfile,recword[i]:1)
          else
            begin
              found:=false;
              i:=0;
              repeat
                i:=i+1;
                if ((mat[1,i]=s[1]) and (mat[2,i]=s[2])
                                    and (mat[3,i]=s[3]))
                then
                 found:=true
              until found;
              recword[i]:=(recword[i] + 1) mod 2;
              for i:=1 to 7 do
              write(outfile,recword[i]:1)
            end;
          if (count=10) then
            begin
              writeln(outfile);
              count:=0
            end;
        end;
        readln(infile)
      end;
    writeln(outfile);
    close(outfile);

  end;{procedure hammingdecode}

  procedure setletord (var order:col);

  {reads into an array the English alphabet in the order}
  {of the respective probabilities of occurrence}

  var
    i : integer;

  begin {procedure setletord}
    reset(letterorder);
    for i:= 1 to 26 do
      readln(letterorder,order[i])
  end;{procedure setletord}

  function compare(w1,w2 : word; start:integer):boolean;

  {returns a value true iff the two supplied 7 bit}
  {sequences are the same}

  var
    check:boolean;
    i    :integer;

  begin{function compare}
    i:=1;
    check:=true;
    while ((i <= 7) and check) do
      begin
        check:=w1[i]=w2[start+i];
        i:=i+1
      end;
    compare:=check
  end;{function compare}

  procedure readnlbits (var cword:word; var finished :
                                                    boolean;
                       var infile:text);

  var
    i : integer;

  begin
  if eof(infile) then
    finished:=true
  else if eoln(infile) then
         readln(infile);
  if eof(infile) then
    finished:=true;
  if not(finished) then
    for i:=1 to 7 do
      read(infile,cword[i])
end; {procedure readn1bits}

procedure englishdecode (var infile,outfile :text);

{translates a file of bits into English letters}

var
  cword           : word;
  i,count,j       : integer;
  match,finished : boolean;

begin
  reset(infile);
  rewrite(outfile);
  rewrite(abfinal);
  writeln(outfile,'Average number of bits per character
                                        = ',bitcount/charcount);
  count:=0;
  finished:=false;
  while not(finished) do
    begin
      readn1bits(cword,finished,infile);
      if not(finished) then
        begin
          count:=count+1;
          match:=false;
          i:=0;
          while not(match) do
            begin
              i:=i+1;
              match:=compare(cword,codewords[i],0)
            end;
          if (i <= sizew1) then
            begin
              write(outfile,c[i]);
              write(abfinal,'s')
            end
          else
            begin
              write(abfinal,'lo');
              readn1bits(cword,finished,infile);
              if not(finished) then
                begin
                  j:=0;
                  match:=false;
                  while (not(match) and
                                        (j <= numbaddons-1)) do
                begin
                      match:=compare(cword,
                                        codewords[i+j],7);
                      j:=j+1
                    end;
                  if match then
                    write(outfile,c[i+j-1])
                  else
                    write(outfile,'?')
                end
             else
               write(outfile,'*')
           end
         end;
         if count=70 then
           begin
             writeln(outfile);
             writeln(abfinal);
             count:=0
           end;
         if count=35 then
           writeln(abfinal)
     end;
  close(outfile)
end;{englishdecode}

begin {main body}
  getinfo(sizew1,sizew2,r,numbaddons,valid);
  if valid then
    begin
      formcode(codewords,sizew1,sizew2,r,numbaddons);
      encode(message,trans);
      adderrors(trans,errors,received);
      hammingdecode(received,firstdecode);
      setletord(c);
      englishdecode(firstdecode,dfinal);
    end
  else
    writeln('Invalid input');
  writeln(chr(7))
end.
```

```
program tcodesim (input,tcode,message,trans,letterorder,
                        received,tfinal,errors,output);

(* This program the English text of the file message *)
(* using a T-code, which is stored in the file tcode.*)
(* The resulting bits are written to the file        *)
(* datasent. The contents of this file are then added*)
(* sequentially to the bits in the fiile errors, the *)
(* bitwise sums being placed in the file datarcvd.   *)
(* The bits of this file are then decoded in order to*)
(* the appropriate English character                 *)

type
  col       = array [1..26] of char;
  word      = packed array [1..7] of char;
  codearray = array [1..26] of word;

var
  tcode,message,errors,trans,received,tfinal,
                              letterorder  : text;
  codewords                                : codearray;
  c                                        : col;
  bitcount,charcount                       : integer;
  biterrp                                  : real;

procedure progheader;

{writes initial info to the screen}

begin{procedure progheader}
  writeln('This program simulates a Titchener code for
                                          the English');
  writeln('capital letters');
  writeln('Please supply the bit error probability of the
                                   current error file');
  readln(biterrp)
end;{prcedure progheader}

procedure setupcode (var infile : text;
                       var code : codearray);

{copies the codewords from infile to the array code}

var
  i,count : integer;

begin{procedure setupcode}
  reset(infile);
  count:=0;
  while not(eof(infile)) do
    begin
      count:=count + 1;
      for i:= 1 to 7 do
```

```
        code[count,i]:=' ';
      i:=0;
      while not(eoln(infile)) do
        begin
          i:=i+1;
          read(infile,code[count,i])
        end;
      readln(infile);
    end
end;{procedure setupcode}

procedure writew (var outfile : text; cwno : integer);

{writes a single codeword to outfile}

var
  i : integer;

begin{proceure writecw}
  for i:= 1 to 7 do
    if (codewords[cwno,i] <> ' ') then
      begin
        bitcount:=bitcount+1;
        write(outfile,codewords[cwno,i])
      end
end;{procedure writecw}

procedure writecword (var outfile :text; ch:char);

{writes the codeword corresponding to ch to outfile}

begin{procedure writeword}
  case ch of
    'E':writew(outfile,1);
    'T':writew(outfile,2);
    'A':writew(outfile,3);
    'O':writew(outfile,4);
    'I':writew(outfile,5);
    'N':writew(outfile,6);
    'S':writew(outfile,7);
    'H':writew(outfile,8);
    'R':writew(outfile,9);
    'D':writew(outfile,10);
    'L':writew(outfile,11);
    'C':writew(outfile,12);
    'U':writew(outfile,13);
    'M':writew(outfile,14);
    'W':writew(outfile,15);
    'F':writew(outfile,16);
    'G':writew(outfile,17);
    'Y':writew(outfile,18);
    'P':writew(outfile,19);
    'B':writew(outfile,20);
    'V':writew(outfile,21);
    'K':writew(outfile,22);
    'J':writew(outfile,23);
    'X':writew(outfile,24);
    'Q':writew(outfile,25);
    'Z':writew(outfile,26);
  end{case}
end;{procedure writecword}

procedure encode (var infile,outfile : text);

{reads an English message from infile and writes to
{outfile a codeword for each alphabetic character}

var
  count : integer;
  ch    : char;

begin{procedure encode}
  reset(infile);
  rewrite(outfile);
  bitcount:=0;
  charcount:=0;
  while not(eof(infile)) do
    begin
      count:=0;
      repeat
        read(infile,ch);
        if (ch in ['a'..'z']) then
          ch:=chr(ord(ch)-32);
        if (ch in ['A'..'Z']) then
          begin
            charcount:=charcount+1;
            writecword(outfile,ch);
            count:=count+1
          end
      until ((count=10) or (eof(infile)));
      writeln(outfile);
    end;
  close(outfile)
end;{procedure encode}
procedure adderrors(var infile1,infile2,outfile : text);

{reads bitwise from infile1 and infile2 and writes the
{mod 2 sum of each pair to outfile}

var
  ch1,ch2 : char;

begin
  reset(infile1);
  reset(infile2);
  rewrite(outfile);
  while not(eof(infile1)) do
    begin
      while not((eoln(infile1)) or (eoln(infile2))) do
        begin
          read(infile1,ch1);
          read(infile2,ch2);
           if (ch1=ch2) then
             write(outfile,'0')
           else
             write(outfile,'1')
         end;
       if eoln(infile1) then
         begin
           readln(infile1);
           writeln(outfile)
         end
       else
         readln(infile2)
     end;
   close(outfile)
 end;

 procedure setletord (var order:col);


 var
   i : integer;

 begin{procedure setletord}
   reset(letterorder);
   for i:=1 to 26 do
     readln(letterorder,order[i])
 end;{procedure setletord}

 function compare(w1,w2:word):boolean;

 begin{function compare}
   compare:=w1=w2
 end;{function compare}

 procedure decode (var infile,outfile :text);

 {Reads from infile bits which are then matched to}
 {codewords and decoded to English characters}

 var
   recword,temp,temp2        : word;
   i,j,count,linecount       : integer;
   match                     : boolean;

 begin
   reset(infile);
   rewrite(outfile);
   writeln(outfile,' Bit error probability = ', biterrp);
   writeln(outfile, 'Average wordlength = ',
                                         bitcount/charcount);
   for i:=1 to 26 do
     begin
       writew(outfile,i);
       writeln(outfile)
     end;
```

```
writeln(outfile);
writeln(outfile);
match:=false;
for i:=1 to 7 do
  read(infile,recword[i]);
linecount:=0;
count:=3;
for i:=1 to 7 do
  temp[i]:=' ';
for i:=1 to count do
  temp[i]:=recword[i];
while not(eof(infile)) do
  begin
        repeat
          i:=0;
          repeat
            i:=i+1;
            match:=compare(temp,codewords[i])
          until ((match) or (i=26));
          if match then
            begin
              write(outfile,c[i]);
              linecount:=linecount+1;
              if (linecount = 79) then
                begin
                  linecount:=0;
                  writeln(outfile)
                end;
            end
          else
            begin
              count:=count+1;
              if (count < 8) then
                temp[count]:=recword[count]
            end
       until (match or (count=8));
       if not(match) then
         begin
           for i:=1 to 6 do
             recword[i]:=recword[i+1];
           if not(eof(infile)) then
             if not(eoln(infile)) then
               read(infile,recword[7])
             else
               begin
                 readln(infile);
                 if not(eof(infile)) then
                 read(infile,recword[7])
               end
           else recword[7]:=' '
         end
       else
         begin
           match:=false;
           for i:=1 to (7 - count) do
             recword[i]:=recword[i+count];
             for i:= (7 - count +1) to 7 do
               if not(eof(infile)) then
                 if not(eoln(infile)) then
                   read(infile,recword[i])
                 else
                   begin
                     readln(infile);
                     if not(eof(infile)) then
                       read(infile,recword[i])
                     else recword[i]:=' '
                   end
               else
                 recword[i]:=' '
             end ;
             count:=3;
             for i:= 1 to 7 do
               temp[i]:=' ';
             for i := 1 to count do
             temp[i]:=recword[i];
   end;
  close(outfile)
end;

begin{main body}
  progheader;
  setupcode(tcode,codewords);
  encode(message,trans);
  adderrors(trans,errors,received);
  setletord(c);
  decode(received,tfinal);
  writeln(chr(7))
end.
```