CANADIAN THESES ON MICROFICHE

I.S.B.N.

THESES CANADIENNES SUR MICROFICHE

National Library of Canada
Collections Development Branch

Canadian Theses on
Microfiche Service

Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Direction du développement des collections

Service des thèses canadiennes
sur microfiche

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us a poor photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de mauvaise qualité.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

NL-339 (r. 82/08)

Canada

ADAPTATION OF ERROR-CONTROL CODING CONCEPTS TO DATA-COMPRESSION

By

Tarek H. Abdel-Nabi

Thesis presented to the School of Graduate Studies in partial fulfilment of the requirements for the degree of Ph.D in Electrical Engineering.

UNIVERSITY OF OTTAWA
OTTAWA, CANADA, 1981

ABSTRACT

The use of binary block codes to compress the output of a discrete memoryless source is considered. As the determination of the resultant distortion is closely related to the problem of enumerating the cosets of the codes to be used, the coset enumeration of certain classes of codes is derived and some of the existing bounds on the covering radius are refined. The use of the well-known error-trapping technique is suggested for the implementation of an instrumentable source encoder. The performance of this procedure is compared to the performance of the step-by-step and majority-logic decoding algorithms. Finally, an extension of error-trapping is proposed, in which two codes of same length but different rates can be used to improve the performance of the source encoding process.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

ACKNOWLEDGEMENTS

The author wishes to express his most sincere appreciation and gratitude to his supervisor, Professor S.G.S. Shiva for his assistance and guidance in this work.

Thanks are also due to various members of the staff of the Electrical Engineering Department for their cooperation.

Special thanks are due to Mrs. Deborah Moyer for her meticulous and skillful typing of the thesis.

Finally, the author wishes to acknowledge financial support from the National Research Council of Canada under Grant A-3371.

CHAPTER I

INTRODUCTION

In the three decades since Shannon's seminal papers of 1948 [1], coding theory has matured into an important segment of communications systems engineering, one that sees increasing applications of relatively standard techniques to reduce system cost and complexity.

The reasons for the increasing applications were numerous: the rapid growth in satellite communications; the revolution in digital integrated circuits with large scale integration; the increasing emphasis on the reliable transmission of digital data; the availability of relatively inexpensive computers for system, algorithm and hardware simulation; the increasing digitization of modems and switches; and the increasing sophistication of the user community.

Developments within the coding field were equally important. The discoveries of new classes of codes and the inventions of powerful decoding algorithms were critical to the significant application of coding theory.

Communications systems can be generally grouped into one of the following two classes: those requiring a very high fidelity of reproduction of the transmitted information,

and those which can tolerate a certain amount of distortion in the received information. Examples of the first class would include a computer communication link or a memory storage system, while a television link or a facsimile transmission system would belong to the second class.

A block diagram model of a communication system is shown in Figure 1. This model is sufficiently general to encompass most communication systems encountered in practice, where in some cases, not all coder/decoder pairs need necessarily appear.

We now proceed to discuss briefly the various blocks that comprise the diagram of Figure 1.

The source generates information that the system must communicate to the user. In certain applications the source, or the user, or both are machines rather than people. We shall henceforth assume that the source is discrete and generates a sequence S(x) of n binary digits per unit time.

The source encoder is the component which categorizes the different 2^n possible output states into a lesser

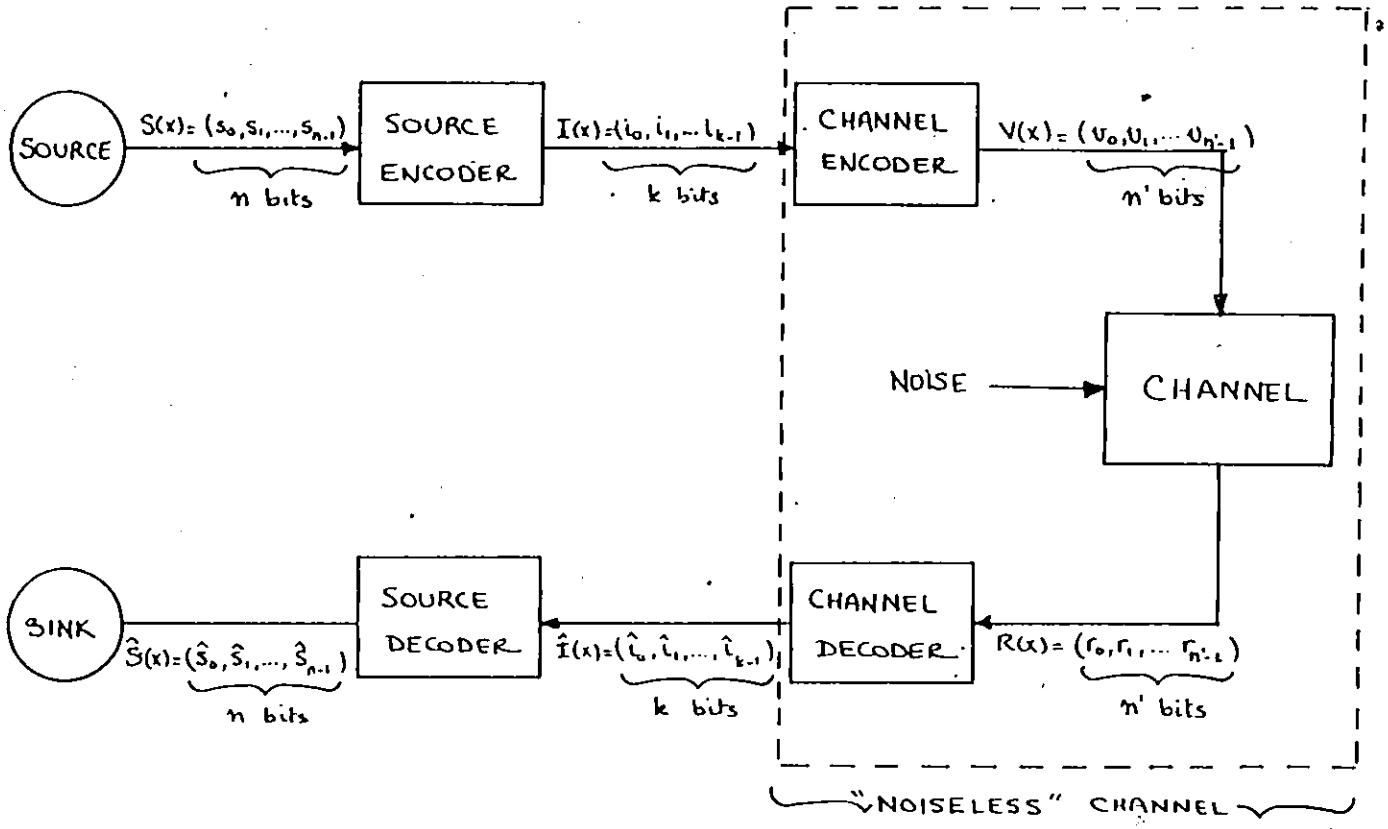


Figure 1 - Block Diagram of a Communication System

number 2^k of states, to remove redundancy from the source output. Redundancy removal results in data compression and a reduction in the transmission rate, for the output of the source encoder now consists of a sequence I(x) of length k, where k<n. Thus a channel of lower capacity can be used, so that data compression is equivalent to bandwidth reduction.

The source encoding step may introduce distortion, but need not. When distortion is allowed, there generally exists a trade-off between the allowed distortion and the amount of data compression. This is discussed in some detail later in this chapter, as one of the purposes of this thesis is to study certain source coding techniques and their corresponding compression-distortion trade-off.

The purpose of the channel encoder is to introduce some structured redundancy to the output I(x) of the source encoder, on the basis of a one-to-one assignment rule that produces a channel codeword V(x) of length n. The added redundancy is used by the channel decoder in its attempt to remove the errors introduced by the noisy channel.

The channel is a spatial link between the source and

the sink such as a cable or a satellite link. It can also be a storage medium such as a magnetic tape, disc or semiconductor memory. In all cases, the channel has the capability of corrupting the information being transmitted or stored.

On the receiver side, the channel decoder can detect or correct the errors caused by the channel, as long as the number of errors per codeword does not exceed the error-detecting or error-correcting capability of the code. The function of the decoder is to categorize the many possible channel outputs into a generally much smaller number of channel decoder outputs. The decoder output is $\hat{I}(x)$, an estimate of the channel encoder input $I(x)$.

Finally, the function of the source decoder is to reverse the assignment $S(x)$ to $I(x)$ made in the source encoder. The output $\hat{S}(x)$ of the source decoder will be identical to the source encoder input $S(x)$ if the source encoding scheme is distortionless and all channel errors are corrected by the channel decoder. If the scheme is not distortionless, or if the number of channel errors exceeds the capability of the code, then the input $\hat{S}(x)$ to the sink will not be identical to the input $S(x)$ of

the source encoder. As it is not generally possible to determine at the receiver which portion of the difference $\hat{S}(x) - S(x)$ is due to the channel errors, it is usual to assume, when source encoding is studied, that the channel is either noiseless or that the channel decoder has successfully corrected all the channel errors.

The main thrust of coding theory has been directed towards finding codes that satisfy Shannon's noiseless coding theorem [1]. The theorem states that the probability of transmission error can be made arbitrarily small as long as the transmission rate is less than the channel capacity. The rate can be reduced to accomodate the channel capacity by adding redundancy to the information prior to its transmission, and removing it at the receiver. This corresponds to channel encoding and decoding respectively. This redundancy is incorporated in a structured fashion such that a given information sequence $I(x)$ always results in the identical redundancy being added to it by the channel encoder.

There have been two fundamentally different approaches to this problem, namely block coding and tree coding. In block coding, the channel encoder partitions the continuous sequence of information digits into k-symbol blocks. It then encodes each block independently and transmits n-symbol

blocks, with n>k, over the channel. The receiver decodes each n-symbol block independently into a block of k symbols. In tree coding, the channel encoder processes the information continuously and associates each long (perhaps semi-infinite) information sequence with a code sequence containing somewhat more digits. The encoder breaks its input sequence into k_o-symbol blocks, where k_o is usually a small number. Then, on the basis of this k_o-tuple and the preceding information symbols, it emits an n_o-symbol section of the code sequence. The name "tree code" stems from the fact that the encoding rules for this type of code are most conveniently described by means of a tree graph. The class of convolutional codes forms a subset of the class of tree codes. These codes are important since they are simpler to implement than other types of tree codes. Powerful decoding algorithms, like Viterbi's decoding [2] have been devised and successfully implemented [3].

With the rapidly increasing range and variety of channel encoding techniques, there has been a correspondingly increasing need for systems designers to assess the benefits of these techniques quantitatively, both for comparison of different codes and to gauge whether, in given circumstances, coding should be used at all. The

concept of Net Coding Gain [4] has been used to express the improvement attained by coding in terms of effective signal-to-noise ratio so that the effect of coding can be compared with other expedients, such as increasing the energy per bit. For a constant information rate, the modulation rate has to be increased if coding is used. This generally leads to reduced energy per bit, and hence a higher error rate during transmission, than for the uncoded case. The comparison to be made is thus between an unprotected system transmitting K bits/sec, and a coded system transmitting N bits/sec, where N>K. The coded system has a higher channel-error rate, but, if adequately designed, should more than compensate for this by the action of the decoder, so that a positive gain can be achieved.

At this point, we wish to mention that the problem of channel encoding for reliable transmission is beyond the scope of this thesis. Here we are concerned with the problem of source encoding for data compression, for which many results related to the former can be exploited. These results can be found in many excellent references, among which are those by Peterson and Weldon [5], Lin [6], Berlekamp [7], Blake and Mullin [8], and MacWilliams and Sloane [9], to name a few.

The proliferation of computer communication networks and teleprocessing applications involves massive transfer of data over long-distance communication links. At the same time, many data processing applications involve storage of large volumes of alphanumeric data such as names, addresses, inventory item descriptions, etc ... . To reduce the data communication costs and/or the storage requirements, there is a need to reduce the redundancy in the data representation, i.e., to compress the data. Data compression also reduces the load on input/output channels in a computer installation. Because of the reduced space requirements of compressed data, it becomes feasible to store data at a higher, and thus faster, level of the storage hierarchy. Since data compression techniques remove some of the redundancy in noncompressed text, they usually contribute to data security. In short, data compression techniques are most useful for large archival files, I/O bound systems with spare CPU cycles, and transfer of voluminous amounts of data over long-distance communication links.

The application of coding theory to the area of data compression also finds its roots in Shannon's work, As we mentioned earlier in this chapter, Shannon's coding theorem states that a source of entropy rate H can be

transmitted reliably over any channel of capacity C>H. This theorem also has a converse devoted to the frequently encountered situation in which the entropy rate of the source exceeds the channel capacity. The theorem states that not all the source data can be recovered reliably when H>C.

Since it is always desirable to recover all the data correctly, one can reduce H either by

(a) slowing down the rate of production of source symbols, or

(b) transmitting the symbols more slowly than they are being produced. However, (a) is often impossible because the source rate is not under the communication system designer's control, and (b) often is impossible both because the data becomes stale, or perhaps even useless, because of long coding delays and because extensive buffering memory is needed.

If H cannot be reduced, then the only other remedy is to increase C. This, however, is usually very expensive in practice. It is therefore reasonable to attempt to preserve those aspects of the source data that

are most crucial to the application at hand in the situation where H>C continues to prevail. In this case, the communication system should be configured to reconstruct the source data at the channel output with minimum possible distortion subject to the requirement that the rate of transmission of information cannot exceed the channel capacity. There is obviously a trade-off between the rate at which information is provided about the output of a data source and the fidelity with which this output can be reconstructed on the basis of this information.

A graphical sketch of a typical rate-distortion tradeoff is shown in Figure 2. If the rate R at which information can be provided about the source output exceeds H, then no distortion need result. As R decreases from H towards zero, the minimum attainable distortion steadily increases from 0 to the value D_{max} associated with the best guess one can make in the total absence of any information about the source outputs.

A satisfactorily defined rate distortion function R(D) possesses the following crucial property. It is possible to compress the data rate from H down to any R>R(D) and still be able to recover the original source

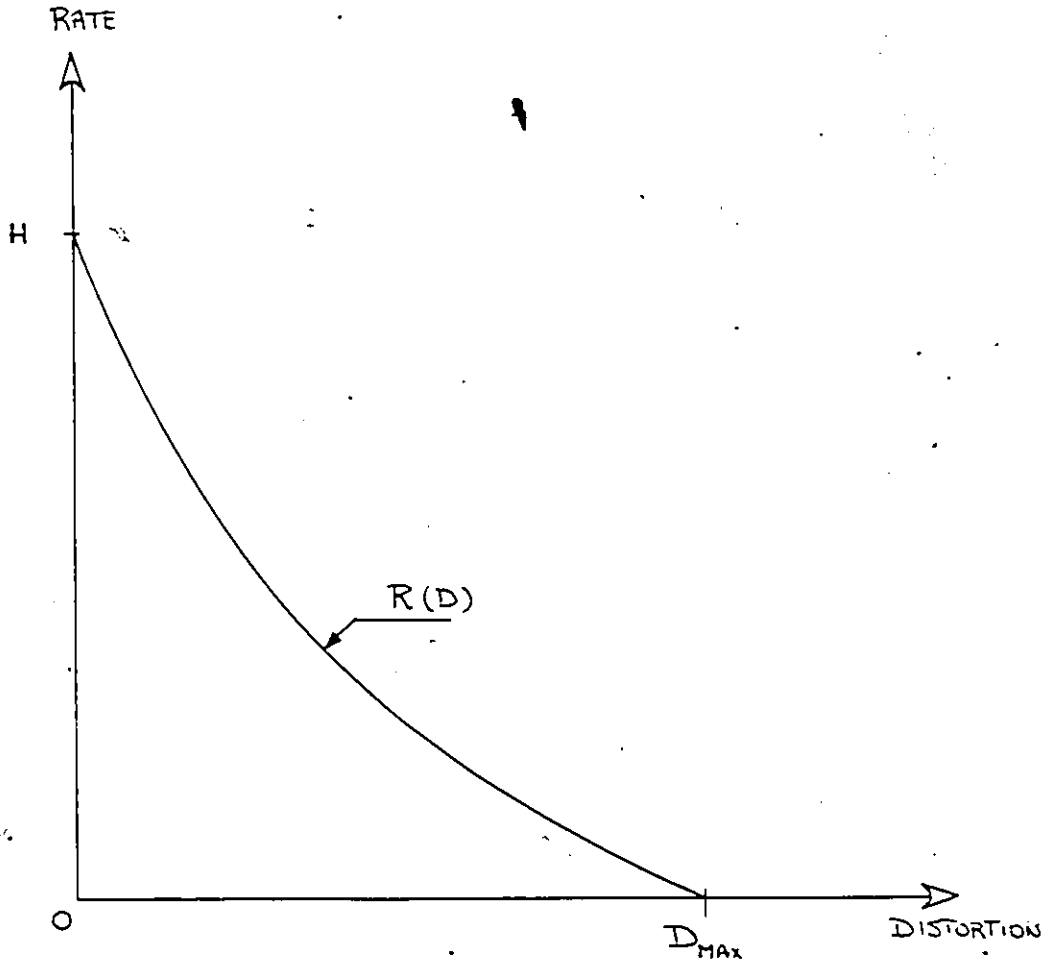


Figure 2 - Typical Rate Distortion Tradeoff

outputs with an average distortion not exceeding D. Conversely, if the compressed data rate R satisfies $R < R(D)$, then it is not possible to recover the original source data from the compressed version with an average distortion D or less.

An R(D) curve that possesses the above property clearly functions as an extension of the concept of entropy. Just as H is the minimum data rate (channel capacity) needed to transmit the source data with zero average distortion, R(D) is the minimum data rate (channel capacity) needed to transmit the data with average distortion D.

We shall now define R(D) for the case of a memoryless source and a context-free distortion measure. A discrete memoryless source generates a sequence $\{X_1, X_2, \ldots\}$ of independent identically distributed random variables. Each X_i assumes a value in the finite set $A = \{a_1, \ldots, a_M\}$ called the source alphabet. The probability that $X_i = a_j$ will be denoted by $P(a_j)$. This probability does not depend on i because the source outputs are identically distributed. Let $\underline{X} = (X_1, \ldots, X_n)$ denote a random vector of n successive source outputs, and let $\underline{x} = (x_1, \ldots, x_n) \in A^n$ denote a value that $\underline{X}$ can

assume. Then the probability $P_n(x)$ that $\underline{X}$ assumes the value $\underline{x}$ is given by

$$P_n(\underline{x}) = \prod_{i=1}^{n} P(x_i)$$

because the source has been assumed to be memoryless.

With reference to the communication system model of Figure 1, let $Y_1, Y_2, \ldots$ denote the sequence of letters received by the user. In general, the Y_i assume values in an alphabet $B = \{b_1, \ldots b_N\}$ of letters that may differ both in value and in cardinality from the source alphabet A, although A = B in most applications.

In order to quantify the rate-distortion trade-off, we must have a means of specifying the distortion that results when $X_i = a_j$ and $Y_i = b_k$. We shall assume that there is given for this purpose a so-called distortion measure ρ, i.e., $\rho(a_j, b_k)$ is the penalty, loss, cost, or distortion that results when the source produces a_j and the system delivers b_k to the user. Moreover, we shall assume that the distortion $\rho_n(x, y)$ that results when a vector $\underline{x} \in A^n$ of n successive source letters is represented to the user as $\underline{y} \in B^n$ is of the form

$$\rho_n(\underline{x}, \underline{y}) = \frac{1}{n} \sum_{i=1}^{n} \rho(x_i, y_i)$$

A family $\{\rho_n, 1 \leq n < \infty\}$ of such vector distortion measures will be referred to as a single letter, or memoryless, fidelity criterion because the penalty charged for each error the system makes does not depend on the overall context in which the error occurs.

Each communication system linking source to user in Figure 1 may be fully described for statistical purposes by specifying for each $j \in \{1, \ldots, M\}$ and $k \in \{1, \ldots N\}$ the probability $Q_{k/j}$ that a typical system output Y will be equal to b_k given that the corresponding input X equals a_j. Contracting notation from $P(a_j)$ to P_j and from $\rho(a_j, b_k)$ to ρ_{jk}, we may associate with each system $Q = (Q_{k/j})$ two functions of extreme importance, namely

$$I(Q) = \sum_{j=1}^{M} \sum_{k=1}^{N} P_j \, Q_{k/j} \log \left(\frac{Q_{k/j}}{Q_k} \right),$$

where $$Q_k = \sum_j P_j Q_{k/j},$$

and $$d(Q) = \sum_{j,k} P_j \, Q_{k/j} \, \rho_{jk}.$$

$I(Q)$ is the average mutual information between source and user for the system Q, and $d(Q)$ is the average distortion with which the source outputs are reproduced for the user by system Q. For situations in which both the source and the fidelity criterion are memoryless, Shannon [10] has defined the rate distortion function $R(D)$ as follows

$$R(D) = \min_{Q:d(Q) \leq D} I(Q),$$

that is, we restrict our attention to those systems Q whose average distortion $d(Q)$ does not exceed a specified level D of tolerable average distortion, and then we search among this set of Q's for the one with the minimum value of $I(Q)$. We minimize rather than maximize $I(Q)$ because we wish to supply as little information as possible about the source outputs provided, of course, that we preserve the data with the required fidelity D. In physical terms, we wish to compress the rate of transmission of information to the lowest value $R_n(D)$ consistent with the requirement that the average distortion may not exceed D.

The rate distortion function $R(D)$ is given its

source coding operational significance by the following two results [11].

RESULT 1: (Converse)

There are no source codes acting on blocks of length n that achieve average distortion D and rate less than $R_n(D)$. There are no source codes of any blocklength achieving distortion D and rate less than R(D).

RESULT 2: (Source Coding Theorem)

For sufficiently long blocklengths there exist source codes that achieve average distortion D and rate arbitrarily close to R(D).

With respect to the rate distortion curve of Figure 2, Result 1 states that the average distortion and rate performance of any source code must lie above the rate distortion curve on the (D, R) plane. Result 2 states that as increasingly longer blocklength codes are considered, performance arbitrarily close to the rate distortion curve is possible. Thus, R(D) specifies the limiting tradeoff between achievable average distortion

and rate. The proof of Result 2 involves a random coding argument which does not yield a method for finding good codes.

Referring to Result 2, the rate $R_n(D)$ of a source code converges to its $R(D)$ as $n \to \infty$. Because such a code can be transmitted with arbitrarily small error over any channel of capacity $C > R_n(D)$, it is therefore potentially advantageous to increase n in order to achieve fidelity D over channels of smaller capacity.

However, the increase in equipment complexity with larger blocklength imposes a limit beyond which a further reduction in the channel capacity requirement would be impractical. It is worth noting that, with larger blocklength, the rate of convergence of the average distortion to its ideal value is much slower than the convergence rate of $R_n(D)$ to the theoretical $R(D)$.

Two fundamental indicators of source encoder complexity, memory requirements and computational requirements have been proposed by Berger [11] in order to make it possible to discuss system

complexity independently of any technological developments. These indicators are used to measure the instrumentability of a source encoder in the following manner. From the observation that the amount of information generated by the source grows only algebraically with blocklength, there arises the hope that the same should be true of the data processing effort. Therefore, we shall say that a source coding scheme is non-instrumentable if its implementation requires either memory or computational effort to grow exponentially with blocklength. If both grow only algebraically, then the scheme is said to be instrumentable.

The brute force approach to source encoding, the code book method, is clearly noninstrumentable. In this method, storage must be provided at the encoder for exp(nR) code words of n letters each. Then the distortion between each codeword and the source output must be computed. Finally, the codeword that minimizes this distortion must be presented to the channel encoder. Both the storage requirement and the computational effort grow exponentially with n. It is therefore necessary to employ codes with structure in order to alleviate the storage and computational

requirements.

With reference to the communication system of Figure 1, a useful analogy exists between source encoders and channel decoders and also between channel encoders and source decoders [11]. Both the source encoder and the channel decoder are relatively complex, decision making devices that examine their inputs carefully and then classify them accordingly. The detailed study of coding for transmission through noisy channels that have been performed to date have produced a wealth of theoretical and practical results regarding the design of channel encoders and decoders. By exploiting the analogies referred to above, it is therefore possible to extend many of these results to the analysis and design of source encoders and decoders. This thesis essentially follows the same broad lines as we shall discuss the adaptation of error-control-coding, particularly block coding, concepts to data-compression. We shall first elaborate on the analogy between source and channel coding to describe the optimum source encoder. Some sub-optimal block encoding schemes will be discussed in later chapters.

An important assumption in this thesis is that the

source outputs consist of independent, equiprobable binary digits, and the distortion is measured by the error fidelity criterion $\rho_{jk} = 1 - \delta_{jk}$. For this binary memoryless source, the rate distortion function R(D) has been shown, by Shannon, to be

$$R(D) = \begin{cases} H_b(p) - H_b(D) & , \quad 0 \leq D \leq D_{max} = p \\ 0 & , \quad D \geq p \end{cases}$$

where $p \leq \frac{1}{2}$ is the probability of the source producing the digit 1 and

$H_b(\cdot)$ is the binary entropy function given by

$$H_b(x) = -x\log_2 x - (1 - x)\log_2(1 - x) \ , \ 0 \leq x \leq 1.$$

In the equiprobable case $p = \frac{1}{2}$, we have

$$R(D) = 1 + D\log_2 D + (1 - D)\log_2(1 - D) \text{ bits/letter.}$$

At this point, we would like to point out that the use of block codes for source coding is one among the numerous techniques suggested to compress the data and reduce the bandwidth requirements. These redundancy removal techniques can be classified in many different

ways depending on their application (distortionless or not), the source type (memoryless or Markovian), or the mapping procedure (fixed length to variable length, fixed length to fixed length, or variable length to fixed length). For example, the entropy coding techniques of Shannon-Fano [12] and Huffman [13] are distortionless and can be used to code the output of sources with unequal message probabilities so as to achieve a minimum average codeword length.

The procedure for constructing codewords with the Shannon-Fano technique is as follows. Given an arbitrary code alphabet made up of N symbols, the source alphabet must be successively subdivided into N equiprobable groups. The first position of all the codewords for the first group will contain the first code symbol, the first position of all the codewords for the second group will contain the second code symbol, and so on until the Nth group. In the Nth group, codewords will have the Nth code symbol in their first position.

The next step consists in subdividing each of the N equiprobable groups into N equiprobable subgroups. Then the first code symbol is assigned to the second

position in all the codewords of the first subgroup, the second code symbol to the second position in all the codewords of the second subgroup, and so on until the Nth subgroup, in which all codewords will have the Nth code symbol in the second position.

The subdivision process is continued until all the subgroups contain exactly one message.

Clearly, the subdivision process can only be carried out exactly if all source message probabilities can be expressed as negative powers of N, and only in this case will the code be optimal.

Huffman's source coding procedure is a systematic procedure which always yields an optimum code, in the sense that no other code has a smaller average codeword length for a given message ensemble, regardless of the probability distribution of the source alphabet.

Briefly, the coding procedure is as follows. Given an ensemble A of M messages $\{a_1, a_2, \ldots a_M\}$ occurring with probabilities $P = \{p_1, p_2 \ldots, p_M\}$, where $p_i > p_{i+1}$, each message is to be expressed as a codeword made up of elements drawn from an alphabet of

N symbols. The messages will be grouped together M_0 messages at a time, where m_0 is the integer satisfying the two requirements

$$2 \leq M_0 \leq N$$

$$\frac{M - m_0}{N - 1} = \text{positive integer.}$$

The m_0 least probable messages are grouped together, and the sum of m_0 message probabilities is used as the probability for this grouped message.

A reduced ensemble of messages is now constructed in order of non-increasing probability, with the grouped message acting as a single message in the reduced ensemble.

New "grouped" messages are then formed by taking the N least probable messages of the reduced ensemble, and using this as a new group to form a new auxiliary ensemble. This is repeated successively until the auxiliary ensemble consists of N grouped messages, which leads to a final auxiliary ensemble consisting of one message with a unit probability.

The steps briefly described above are equivalent to forming a tree of which the original messages are the terminal nodes. If, to each branch of the tree, a code symbol is assigned, then a code word for every message can be written by beginning at the root and following the branches to the terminal node associated with a particular message.

Both Shannon-Fano codes have non-uniform codewords lengths and exhibit the prefix property, i.e., no codeword can be obtained by concatenating terms of one codeword with another. These fixed length to variable length mapping schemes are, however, susceptible to loss of synchronization and buffer overflow.

Run-length encoding [14], [15] is a variable to fixed length data compression technique that maps runs of identical successive source symbols into fixed length codewords. This technique, best suited for a source with highly unsymmetrical symbol probabilities, is a special case of syndrome source coding described in the next chapter. For a binary source, the appearance of a string, or run, of zeros will be terminated by a one, so that a run of length J corresponds to (J - 1) zeros followed by a one. A run of length zero therefore

corresponds to the occurrence of a one. If $p(0) >> p(1)$, it is more efficient to code the run lengths of zeros as opposed to the zeros of the runs. One encoding approach involves representing each possible run up to a certain maximum length by a distinct binary k tuple. Since there are 2^k k-tuples, they could represent runs from length 1 to length $2^k - 1$. The problem then lies in the choice of k: If too small, it will not be able to represent the longer runs. If k is too large, on the other hand, it will be inefficient in representing the shorter runs. This problem may be reduced by letting the k-tuple of all zeros represent an unterminated run of 2^k zeros, so that runs longer than $2^k - 1$ can be represented by a succession of k-tuples.

The transmission of digital images is another area where bandwidth reduction is necessary to compensate for the inefficiencies of the digitizing schemes. In the most basic method of digitizing images, pulse code modulation, the continuous two dimensional image is low pass filtered, then sampled in space at the Nyquist rate and quantized in brightness (grey level). If the image is being scanned in the ordinary manner, i.e., line-by-line, the resulting signal consists of a sequence of amplitude levels chosen from a finite set of allowed

levels. The amplitude levels are then coded as symbols, of which there will be as many as there are levels. For example, assuming we have a television frame to transmit, of 4 MHz nominal bandwidth, then the sampling rate will be 8 MHz, and if 128 grey levels are allowed, then all symbols will be 7 bits long, and each image sample is described by one symbol. Hence, the bit rate will be 56 M bits/sec, which requires a much greater bandwidth (28 MHz) than the 4 MHz originally required. This makes bandwidth reduction very desirable to minimize the transmission costs. Various bandwidth reduction techniques (e.g. Spatial Transform Coding, Adaptive Pulse Code Modulation, Delta Modulation, etc...) have been suggested. They are, however, beyond the scope of this thesis, but more information can be obtained by consulting, as a starting point, some of the review articles that have been published in the IEEE Proceeding and the Transactions on Communications such as [16], [17] and [18].

We now conclude this Introduction with an outline of the thesis.

In Chapter II, we develop the background material necessary for the following chapters. This includes:

a description of the standard array and a review of its properties in connection with channel decoding and source encoding; the class of Bose-Chaudhuri-Hocquenghem (BCH) codes; the coset enumeration of some linear codes, including BCH codes and Reed-Muller codes; interleaved codes; shortened BCH codes; and finally, bounds on the covering radius of linear codes with emphasis on Delsarte's [19] upper bound, and the Gorenstein-Peterson Zierler (GPZ) lower bound for BCH codes [20].

In Chapter III, we give several results related to the cosets of linear codes. First, we present two upper bounds which can be tighter than Delsarte's bound under certain conditions. The GPZ lower bound is then extended to the proper subgroups of a code. The use of these bounds is illustrated in two examples. We then consider the cosets of the Maximal Length Shift Register (MLSR) codes and give an expression for the number of cosets of weight $\frac{n+1}{4}$. The cosets of the shortened-by-1-bit double-error correcting binary BCH codes are analysed next and their coset enumeration is given. We then discuss a bound on the coset enumeration of shortened-by-s-bits codes.

Chapter IV starts with a discussion of the

application of some well known decoding algorithms to source encoding. We examine, in particular, the BCH decoding, the majority logic decoding and the step-by-step decoding algorithms and discuss their respective advantages and weaknesses in the context of source encoding. We then suggest the use of the error-trapping technique as a better alternative and describe the source coding algorithm based on this technique. This algorithm is simulated with a large number of codes to show that its performance is indeed better than the single pass step-by-step and majority-logic algorithms. Some estimates of the upper bound on the maximum distortion are then presented. Finally we discuss a possible extension of the error-trapping technique in which two codes of equal length and different rates can be used for source encoding.

The concluding remarks are given in Chapter V.

Finally, we wish to mention that parts of the thesis have already been reported elsewhere [21], [22].

CHAPTER II

A REVIEW OF CODING RESULTS APPLICABLE TO DATA COMPRESSION

The purpose of this chapter is to develop the background necessary for the future chapters. Here, we review some of the error-control coding results that can be applied to data compression. We elaborate on the analogy between channel and source coding, mentioned in the Introduction, by describing the standard array of a linear code and presenting the expression for the average distortion. We then give the definition of BCH codes and mention their decoding algorithms. Next we survey the results related to the coset enumeration of some linear codes. Finally, we review some existing bounds on the covering radius of a linear code. Delsarte's upper bound and the Gorenstein, Peterson and Zierler lower bound for BCH codes are the two main bounds covered.

2.1 The Standard Array

Let V be an (n, k) linear code and let V_1, V_2 ... V_{2^k} be the 2^k code vectors. A way to partition the space of 2^n n-tuples is as follows. The 2^k code vectors are placed in a row with the zero code vector $V_1 = (0, 0, \ldots 0)$ as the leftmost element. From the remaining $2^n - 2^k$

n-tuples, one, say e_1, is chosen and is placed under the identity vector V_1. The row is then completed by placing under each code vector V_i the sum $V_i + e_1$. Having completed the second row, an unused n-tuple e_2 is chosen from the remaining n-tuples and is placed under V_1 and the row similarly completed. We continue this process until all the n-tuples are used. Then we have an array of rows and columns as shown in Figure 3. Each row consists of 2^k n-tuples. This array is called a standard array for the given (n, k) linear code.

In the standard array, no two n-tuples in the same row are identical and no n-tuple appears in different rows. Each row of the array consists of 2^k distinct n-tuples and all the rows are disjoint. There are $2^n/2^k = 2^{n-k}$ disjoint rows called cosets. The left-most n-tuple e_1 of each row is called a coset leader. The weight of a coset is the weight of its coset leader.

The standard array is useful in analyzing the performance of block codes when they are used for channel decoding (error control) and source encoding (data compression). In the following we will first discuss the case of channel decoding, then we will show the applicability of results to source encoding.

v_1	v_2	v_3	• • •	v_{2^k}
e_1	v_2+e_1	v_3+e_1		$v_{2^k}+e_1$
e_2	v_2+e_2	v_3+e_2		$v_{2^k}+e_2$
•	•	•		•
•				
•				
$e_{2^{n-k}-1}$	$v_2+e_{2^{n-k}-1}$	$v_3+e_{2^{n-k}-1}$		$v_{2^k}+e_{2^{n-k}-1}$

Figure 3 - The Standard Array

For any code vector which is transmitted over the noisy channel, the received vector R may be any of the n-tuples. Any decoding scheme used in the decoder is a rule to partition the 2^n n-tuples into 2^k disjoint subsets S_1, S_2 ... S_2k such that the subset S_i contains only the code vector V_i. If the received vector R is found in the subset S_i, then the decoder identifies V_i as the transmitted code vector. Correct decoding is made if the received vector R is in the subset S_i which corresponds to the actual transmitted code vector. An incorrect decoding results if the received vector R is in the subset which does not correspond to the actual transmitted code vector.

The standard array partitions the 2^n n-tuples into 2^k disjoint subsets C_1, C_2, ... C_2k where

$$C_j = \{V_j, V_j + e_2, V_j + e_3, \ldots, V_j + e_{2n-k}\}.$$

Suppose that the code vector V_j is transmitted over a noisy channel. We can see that the received vector R is in C_j if the error pattern caused by the channel is a coset leader. Then the received vector will be decoded correctly into V_j. On the other hand, if the error pattern caused by the channel is not a coset leader, an

incorrect decoding will result.

In summary, when a standard array is used for decoding, the decoding is correct if and only if the error pattern caused by the channel is a coset leader. The 2^{n-k} coset leaders are called correctable error patterns. In order to minimize the probability of decoding error, the 2^{n-k} coset leaders are chosen to be the error patterns which are most likely to occur. For the binary symmetric channel, the coset leader should be then chosen as a vector with minimum weight from the remaining available vectors. For this choice and if the code vectors are equally likely to be transmitted, the probability of correct decoding is as large as possible. The expression for the probability of correct decoding is given by

$$P(\text{Correct decoding}) = \sum_{i=0}^{\Omega} N_i \, p^i (1-p)^{n-i} \qquad (2.1)$$

where N_i is the number of coset leaders of weight i, p is the channel error probability and Ω is the maximum coset weight or covering radius of the code.

Now suppose that the (n, k) code is the null space

an $(n - k \times n)$ matrix $[H]$. For any received vector R_i, the $(n - k)$-tuple $S = R_i[H]^T$ is called the syndrome. Since the code is the null space of $[H]$, a vector is a codeword if and only if its syndrome is zero. In general, each row of $[H]$ corresponds to a parity check equation that codewords must satisfy. The components of S are zero for all equations that are satisfied and otherwise are non zero. With reference to the standard array, all the 2^k n-tuples of a coset have the same syndrome and the syndromes for different cosets are different. Since the syndrome is an $(n - k)$-tuple, there are 2^{n-k} distinct syndromes. There is therefore a one-to-one correspondence between a coset and an $(n - k)$-tuple syndrome. Or, there is a one-to-one correspondence between a coset leader (a correctable error pattern) and a syndrome.

Using this correspondence, we may form a decoding table which is simpler than the standard array. The table consists of 2^{n-k} coset leaders and their corresponding syndromes. The decoding of a received vector R can then be done as follows. First, calculate the syndrome $R[H]^T$. Next, locate the coset leader e_1 whose syndrome is equal to $R[H]^T$. The n-tuple e_1 is assumed

to be the error pattern caused by the channel. Finally, the code vector $V_j = R + e_1$ is identified as the transmitted code vector. In principle, the table look-up decoding can be used with any (n, k) linear code. But, for large n - k, this decoding becomes very impractical since the decoder has to store 2^{n-k} coset leaders and perform a large number of computations in order to find the coset leader whose syndrome matches the syndrome of the received vector. Thus, practical decoding methods require additional properties in a code other than just the linear property.

For an (n, k) linear code, if it is possible to construct a standard array such that all the error patterns of weight t or less (but not all of weight t + 1 or more) can be used as coset leaders, then the code is a t-error-correcting code whose minimum distance is at least 2t + 1. On the other hand, if an (n, k) linear code has minimum distance 2t + 1, it is possible to construct a standard array such that all error patterns of weight t or less (but not weight t + 1 or more) can be used as coset leaders.

We now examine the properties of the standard array

from the source encoding viewpoint. As discussed in the Introduction the source encoder essentially performs the same function as the channel decoder. Given the source n-tuple S, the source encoder must find a codeword V_j such that the amount of distortion given by the Hamming distance between S and V_j is minimum. The codeword V_j that is nearest to S would naturally have to belong to the same subset C_j as S. The distortion introduced in this case is therefore equal to the weight of the coset leader. The significant difference between source encoding and channel decoding emerges when the coset leader weight is greater than t, the error correcting capability of the code. In this case, there may be more than one word in the coset with the same weight as that of the coset leader. Let one of them be e_i. Then the addition of e_i to S will result in a codeword V_1 that is equally distant from S as V_j. Whereas in channel decoding, this case would be interpreted as incorrect decoding, the choice of V_1 as a codeword for source encoding purposes is legitimate, since it satisfies the minimum distortion criterion. From this standpoint, it can be said that the source encoding requirements are not as stringent as those of channel decoding.

Assuming that the source sequences are all equally

likely to be emitted; the performance measure of the source encoding scheme using an (n, k) linear code can be expressed in terms of the average distortion per bit D_{avg}. From the above discussion, we see that D_{avg} is simply equal to the average normalized weight of the coset leaders in the standard array [11], that is,

$$D_{avg} = \frac{\sum_{i=0}^{\Omega} iN_i}{\left(2^n - k\right) n} . \qquad (2.2)$$

where N_i is the number of coset leaders of weight i.

It is worth noting here that the source encoding schemes we will analyse may not necessarily be efficient if the source is not memoryless and symmetric. For many sources such as the output of measuring devices in space experiments, the information source is highly asymmetric with strong memory constraints. In such cases the conventional scheme may be inefficient since the codewords of simply implemented linear error-correcting codes do not well approximate the likely set of source output sequences. Moreover in any case, this scheme has the disadvantage for many applications that the more complex device, namely the source encoder, is located at the source side where one wishes to use the least

equipment. For these types of applications, an alternative method referred to as syndrome-source-coding has been suggested [23]. The fundamental principle of syndrome-source-coding is that the source output is treated as a channel error pattern, rather than as the received channel codeword. A syndrome former, which is a simple linear device, serves as the source encoder. The syndrome is then taken as the compressed data and there are therefore 1 - R compressed digits per source letter. At the user end, an error pattern estimator for the code i.e., a device which produces a likely (in an appropriate sense) error pattern consistent with the received syndrome, is used as the source decoder. Syndrome-source-coding has the advantage for many applications that the simpler device, namely the syndrome former is located at the source side while the more complex device, namely the error pattern estimator, is located at the user side.

The average distortion for syndrome-source-coding is easily derived. If the binary memoryless source, for which the probability of emitting a one is p, is assumed to behave like a binary symmetric channel with crossover probability p, then the average distortion coincides with the perdigit error probability P_e when the code is

used on the binary symmetric channel. This error probability can be easily computed once the probability of correct decoding is determined. The probability of correct decoding, we recall, is dependent on the value of p and, most importantly, on the coset enumeration of the code.

2.2 BCH Codes

A cyclic code is simply defined as a linear block code in which every cyclic shift of a codeword is also a codeword. A cyclic code of length n over GF(q) consists of all multiples of a generator polynomial g(x) which is the monic polynomial of least degree in the code and is a divisor of $x^n - 1$. To each n-tuple $(a_0, a_1, \ldots, a_{n-1})$ in the cyclic code, there is a polynomial in x of degree less than n, $f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1}$ and multiplication by x (modulo $x^n - 1$) is the same as a cyclic shift. Alternatively the code can be specified completely by the statement that it is the null space of the ideal generated by $h(x) = (x^n - 1)/g(x)$. The polynomial h(x) is referred to as the parity-check polynomial of the code generated by g(x). Since h(x) divides $x^n - 1$, it can be used to generate the dual code.

One of the most important classes of cyclic codes is the class of BCH codes which where found independently by Bose and Ray-Chaudhuri [24] on the one hand, and Hocquenghem [25] on the other. This original work was restricted to binary codes of length $2^m - 1$ for some integer m. It was later generalized by Gorenstein and Zierler [26] to the non-binary case, i.e., GF(q) and any length prime to q. The cyclic BCH codes may be described in the following manner. Let α be any non-zero element of $GF(q^m)$ and take g(x) as the monic polynomial of minimum degree having α^{m_0}, $\alpha^{m_0 + 1}$, ..., $\alpha^{m_0 + d_0 - 2}$ as roots, where m_0 is some integer and $d_0 \geq 2$ is any integer such that the specified roots are all distinct. The resulting code is a cyclic BCH, the length of which is the least common multiple of the orders of the roots. The minimum distance is at least d_0, and d_0 is called the designed distance.

The most important BCH codes are the binary, narrow-sense, primitive codes obtained by letting α be a primitive element of $GF(2^m)$ and letting $m_0 = 1$ and $d_0 = 2t_0 + 1$. For any positive integers m and $t_0 < n/2$, there is a BCH binary code of length $n = 2^m - 1$ which corrects all combinations of t_0 or fewer errors and has no more than mt_0 parity-check symbols.

The first efficient decoding algorithm for binary BCH codes was devised by Peterson [5]. Since then it has been generalized and refined by Gorenstein and Zierler [26], Chien [27], Berlekamp [7], Forney [28] and Massey [29]. A computer implementation of BCH decoders has been described by Michelson [30] and simply implementable decoding algorithms for double- and triple-error-correcting binary BCH codes have been suggested by Davida [31]

2.3 Coset Enumeration of Some Linear Codes

We mentioned in Section 2.1 that the knowledge of the coset enumeration plays a very important role in channel decoding and source encoding. For the former, it permits the calculation of the exact probability of error when maximum likelihood decoding is used. In the latter case, the average distortion, resulting from the compression of the output of a binary discrete memoryless source, can be computed.

In this context, let us recall that a perfect code is a linear code which, for some error-correcting capability t, has all error-patterns of weight t or less and no others as coset leaders. A quasi-perfect code is a code which, for some t, has all patterns of weight t or less, some of weight t + 1, and none of greater weight as coset leaders.

For perfect codes, it has been proved by Tietavainen [32] that there are no perfect Hamming-metric codes over prime power fields except the binary repetition codes, the binary and ternary Golay [33] codes, and the Hamming [34] single-error-correcting codes. There are, however, examples of perfect codes over alphabets of q symbols, q a power of a prime, where the alphabet is not a finite field, which have been reported by Blake and Mullin [8]. Also reported there were Schoenheim's semilinear codes and Vasil'ev's nonlinear codes over finite fields, which are also perfect.

One of theknown classes of quasi-perfect codes is the class of binary two-error-correcting BCH codes. This result was proved in 1960 by Gorenstein, Peterson and Zierler [20] They showed that, for all $m \geq 3$, the (n, k, 2) BCH codes of length $2^m - 1$ have covering radius equal to 3. This was done by proving that given any particular syndrome, the degree of the error locator polynomial associated with it did not exceed 3. The cases of even and odd m were treated separately but this did not affect the outcome of the final result.

For even m, a complete decoding algorithm was given by Berlekamp [7, p. 425]. This algorithm enabled him to count the words of weight 3 in the cosets of weight 2 and 3, with which he determined the number of

codewords of weight 5. Another decoding algorithm was also suggested by Hartmann [35].

For triple-error-correcting binary BCH codes, an extensive study by Berger and Van der Horst [36], [37] showed that the covering radius is five when m is a multiple of four. They also provided strong analytical support for the conjecture that the covering radius is five for all m. The coset enumeration was determined exactly for $m = 5, 8, 9, 10, 11, 12$ and bounded for all $m > 12$.

Assmus and Mattson [38] established that the covering radius is five for odd m, and joined Berger and Van der Horst in conjecturing that it is five for the remaining unsettled case, that is, m even but not a multiple of four. This conjecture was finally proved by Helleseth [39].

In a survey of unsolved problems in coding theory, Berlekamp pointed out that very little was known about the enumeration of cosets of even the simplest families of error-correcting codes. Slepian [40] and Fontaine and Peterson [41] obtained the weight distribution of the coset leaders of some optimum codes of length ≤ 29.

An approximation to the coset distribution of a code was given by Hobbs [42]. Hobbs' technique is a blend of combinational arguments and simulation that approximated fairly well the performance of some short codes. Its main drawback was, however, the absence of any good bounds on the coset enumeration.

Berlekamp and Welch [43] have shown that the enumeration of the cosets of the first order Reed-Muller code is equivalent to the enumeration of Boolean functions under a certain group of transformations, and have thus obtained the coset enumeration of the length 32 first-order RM code.

Sloane and Dick [44] also considered the cosets of first order RM codes. They gave a method for the enumeration of cosets of the shortened first order RM (binary simplex or MLSR) code, the expurgated first order RM (or orthogonal code) and the first order RM (or biorthogonal) code. Their basic idea was to associate with any binary n-tuple Y a single-error correcting code called the structure code. The weight distribution of a coset and the number of cosets with this weight distribution are then obtained in terms of the weight distribution of the structure code of a

coset leader. As an example, they enumerated the cosets of the (15, 4) simplex code and the (16, 5) first order RM code. The applicability of this method was severely limited because of several factors. First, the structure code of a given coset leader is not independent of the choice of that leader within a coset. Second, the characterization of the codes, that appear as structure codes of coset leaders, is not a simple task. Finally, it is not known if many non-equivalent codes have a given weight distribution.

Upper bounds on the covering radius of binary codes were studied by Helleseth, Klove and Mykkeltveit [45]. In particular, they have shown that the covering radius r_m of the $(2^m, m + 1)$ first-order Reed-Muller code satisfied the following relationships

$$r_{2i} = 2^{2i - 1} - 2^{i - 1}, \text{ for even } m = 2i$$

and $2^{2i} - 2^i \leq r_{2i + 1} \leq 2^{2i} - 2^{i - 1}\sqrt{2}$, for odd $m = 2i + 1$.

As the known values of r_1, r_3 and r_5 indicated that $r_{2i + 1} = 2^{2i} - 2^i$ for $i = 0$, 1 and 2, they questioned whether this relationship was generally true. A partial answer was given by Mykkeltveit [46] who showed it to

be also true for i = 3, thus showing that the covering radius of the (128, 8) Reed-Muller code is 56. Verification of this conjecture for larger values of i still remains an open question. (See Note 1).

2.4 Interleaved Codes

In many of the applications to communication technology, interleaving [5] is used as an adjunct to coding for error correction. One technique, which is useful for some types of burst-error channels, is to insert an interleaver between the channel encoder and the channel. The interleaver redistributes the channel symbols to that the symbols from a codeword are mutually separated by somewhat more than the length of a "typical" burst of errors. Thus interleaving effectively makes the channel appear like a random-error channel to the decoder. At the receiver end, the interleaved data stream is then reassembled into the original data format and is checked for errors which are then corrected (within the limitations of the code capability). For some HF channels, interleaving can improve the performance by one to three orders of magnitude [47].

Design parameters for a bit interleaved block code

are the code parameters (n, k, t) and the interleaving factor j. The rate of transmission of information with respect to the channel transmission rate is k/n. Bit-interleaved block coding is easily described as follows. The information stream is presented to the encoder k bits at a time. The encoder then adds the (n - k) parity-check digits. The n-bit codewords are stored in an j x n matrix a row at a time. When the matrix is full, data are read from it a column at a time and the resulting serial stream is then transmitted. For an interleaving factor of j, the successive bits in each codeword are separated in the serial stream by j - 1 bits, one from each of j - 1 other codewords.

The concept of interleaving can also be extended to block-interleaving, which can be viewed as simply adjoining codewords of different blocklengths to get a total blocklength n meeting the requirements of the system designer. If a code i has dimension (n_i, k_i), then the rate of the block-interleaved code is simply

$$\sum_i k_i / \sum_i n_i .$$

For either case, interleaving is a simple way of obtaining a code of long blocklength that retains the

properties of the short component codes in terms of simplicity in encoding and decoding. There is, however, a penalty in terms of time delay equal to twice the total blocklength divided by the channel data rate.

In Chapter III we also propose to discuss the use of interleaved codes for source encoding. This technique is attractive since it provides a means to implement long codes with instrumentable source encoders. Furthermore, the performance of the interleaved codes can be easily derived from the parameters of the component codes.

2.5 Shortened BCH Codes

In many practical applications, some built-in systems constraints may require that the code parameters be set at some particular value other than those which the code designer would prefer. Rather than redesign the entire system from scratch, it usually proves easier to remodel the code. There are six basic ways in which codes can be modified. The block length of the code can be increased by annexing more check digits or more message digits, the block length can be decreased by omitting check digits or omitting message digits; the

block length may remain constant while the number of codewords is increased or decreased. These six types of modifications are respectively called extending [5, p.220] lengthening [7, p.336], puncturing [48], shortening [49], augmenting [50] and expurgating [7, p.335] a code.

The class of codes considered in Chapter III of the thesis is that of the shortened binary BCH codes [5, p.240-243]. They form a class of linear codes which share the mathematical structure and ease of implementation of cyclic codes though they are not actually cyclic. A shortened code has at least as great a minimum distance as the code from which it is derived - the parent code - and since the number of cosets remains unchanged by shortening, the shortened code can correct an appropriately shortened version of any error pattern correctable with the parent code.

An analysis of some of the merits of shortened BCH codes was given by Helgert and Stinaff [49]. They have proved that by shortening certain binary primitive BCH codes they can derive a number of linear error-correcting codes with minimum distances superior to any previously known.

2.6 Bounds on the Covering Radius

The previous section summarized the available results on the coset enumeration of some linear codes. For many classes of codes, the coset enumeration remains an open research problem [9 , p.22]. In the following, we review some of the existing bounds on the covering radius.

2.6.1 Delsarte's Upper Bound

LEMMA 2.2 [19]

The covering radius of a linear code V is at most the number of distinct non zero weights occurring among the vectors of the orthogonal code V*. The equality holds if and only if the code V is perfect.

As the above Lemma clearly shows, Delsarte's upper bound on the covering radius can be applied to any linear code for which the weight distribution of its dual code is known. There is no guarantee, however, that the bound would be reasonably tight in all situations.

To clarify our comment, let us consider the simple

example of the $(n = 2^m - 1, k = n - m)$ Hamming code generated by $g(x)$, and its dual code, the $(n = 2^m - 1, k = m)$ maximal length shift register (MLSR) code generated by $(x^n - 1)/g(x)$.

Using the known fact that the weight spectrum of the MLSR code consists of the all-zero codeword and n codewords of weight $\frac{n + 1}{2}$, then application of Delsarte's bound shows that the covering radius Ω of the Hamming code is at most 1. Since the code is single-error-correcting, then $\Omega \geq 1$. Therefore, $\Omega = 1$ for the perfect Hamming code.

For the reverse situation, the weight spectrum of the Hamming code is known to consist of the all-zero codeword, codewords of weight 3, 4, ..., n - 3, and the all-one codeword. The total number of nonzero weights is therefore (n - 4), which is the upper bound on the covering radius of the MLSR code. This proves to be a very pessimistic estimate of the true covering radius of the MLSR code, as we shall show, in the next chapter, that this covering radius is only $\frac{n - 1}{2}$.

2.6.2 Lower Bounds

The simplest lower bound on the covering radius of a t-error-correcting code is Hamming's sphere packing bound $\Omega \geq t$. The equality holds only for perfect codes and the bound loosens significantly for lower rate (and higher t) codes.

For binary primitive BCH codes, a lower bound on the covering radius was given by Gorenstein, Peterson and Zierler [20]. The bound states that, given a t_1-error-correcting BCH code V_1 and a t_2-error-correcting BCH code V_2 of equal length and $t_2 \quad t_1$, then the code V_2 has a coset of weight at least $2t_1 + 1$.

A different version of Gorenstein, Peterson and Zierler's result was given by Van der Horst [37]. With the definition of C(t, m)* to be a t-error-correcting binary primitive BCH code if it is distinct from the (t - 1)-error-correcting BCH code, the covering radius of C(t, m) is at least 2t - 1. The definition of C(t, m) can be made clearer by noting that C(2, 3) refers to the (7, 1, 3) repetition code of length 7 and that C(3, 3) does not exist. In this context, C(1, m) is the Hamming code that attains the bound, C(2, m) has covering

---

* $m = \log_2(n + 1)$

radius 3 and $C(3, m)$ has covering radius 5. Also, $C(2^m - 2, m)$ is the binary repetition code of length $2^m - 1$, which is known to be trivially perfect.

Now consider $C(t_1, m)$ and $C(t_2, m)$ with $t_2 = t_1 + 1$. The covering radius of $C(t_2, m)$ is therefore at least $2t_2 - 1 = 2t_1 + 1$, which is the form of the bound given by Gorenstein, Peterson and Zierler.

In either form this lower bound can only be used with BCH codes. A more general bound, applicable to a larger family of linear codes is presented in the next chapter.

NOTE 1

Recently obtained results by Kanetkar and Bhargava [51] show that the weight distribution of a coset of a linear code satisfies many linear equations whose coefficients are integers and depend only upon the weights in the dual code. Using the theory developed, they have shown that the covering radius of the (128, 8) RM code is indeed 56 and that 240 is the covering radius of the (512, 10) RM code. The latter result thus proves Mykkeltveit's conjecture for $i = 4$.

CHAPTER III

ON THE COSETS OF CERTAIN CLASSES OF CODES

The cosets of linear codes are the subject of this chapter, where several results will be presented. Additional bounds on the covering radius are given and two examples will illustrate their use. The coset enumeration of two classes of codes, namely the MLSR codes and the shortened-by-1-bit double-error-correcting BCH codes, will be discussed. Bounds on the coset enumeration of shortened-by-s-bits codes are also presented. Finally, we introduce the concept of coset enumerator. Using this notation, we show how to derive the coset enumeration of an interleaved code using the respective coset enumerators of the component codes. We apply this method to the interleaved Hamming and Golay codes and compute the average distortion for these codes.

3.1 Additional Bounds on the Covering Radius

As mentioned in Chapter II, Delsarte's upper bound on the covering radius was not generally tight. Also, the Gorenstein, Peterson and Zierler lower bound was only applicable to the class of BCH codes. In this section we present two upper bounds which can be tighter than Delsarte's bound under certain conditions. We also extend the Gorenstein-Peterson-Zierler bound to cover a larger family of linear codes.

3.1.1 Upper Bounds

Let us define an optimum dimension (n, k) group code V as a code for which the number of codewords is maximum for a given length n and minimum distance d. With this definition, we note that an optimum dimension code is a code for which the Plotkin bound [8, p. 49], namely

$$d \leq \frac{q^k (q-1)n}{(q^k - 1)q} ,$$

is extremely tight.

THEOREM 3.1

If a code V is an optimum dimension code with minimum distance d, then its covering radius Ω is at most equal to d - 1.

PROOF

Suppose that for the code V, there exists a coset C with leader of weight d. The union of V and C would therefore constitute an (n, k + 1) group code with minimum distance d. This contradicts our initial assumption that V is an optimum dimension code. Therefore, the covering radius of V is equal to d - 1 at most.

This concludes the proof of Theorem 3.1.

As a direct application of Theorem 3.1, we note that the MLSR code satisfies the Plotkin bound with equality. It is therefore an optimum dimension code for which the covering radius is upper bounded by

$$\Omega \leq d - 1 = 2^{m-1} - 1 = \frac{n-1}{2} ,$$

whereas, we recall, Delsarte's bound showed the covering

radius to be upperbounded by n - 4.

Now suppose we are given a code which is not optimum and for which the weight spectrum of its dual is not known. We only know, however, that this code contains the all-one codeword. In such a situation, we may use the upper bound given by the following

THEOREM 3.2

For any (n, k) binary group code containing the all-one codeword, the covering radius is at most $\frac{n-1}{2}$.

PROOF

A coset containing a word of weight w must also contain its complement of weight n - w, since both words add up to the all one-codeword. So, if a coset leader has weight $\frac{n-1}{2}$, the coset will also contain a word of weight $\frac{n+1}{2}$. Since the difference between $\frac{n+1}{2}$ and $\frac{n-1}{2}$ is 1, then $\frac{n-1}{2}$ is the maximum value a coset leader can have.

This concludes the proof of Theorem 3.2.

3.1.2 Extended GPZ Lower Bound

Let V_1 be an (n, k_1) binary group code with minimum distance d_1 and let V_2 be an (n, k_2) binary group code with minimum distance d_2, with $d_2 \geq d_1$, such that V_2 is a proper subgroup of V_1. This means that the codewords of V_2 must also belong to V_1.

With this definition, the code denoted by $C(t_2, m)$ in Chapter II is a proper subgroup of $C(t_1, m)$ for $t_2 > t_1$. Also, the $(n, k-1)$ even-weighted subset of an (n, k) binary linear code (with even- and odd-weighted codewords) is a proper subgroup of the parent code.

THEOREM 3.3

The covering radius of the proper subgroup of an (n, k) binary linear code with minimum distance d is at least d.

PROOF

Let $v \neq 0$ be a codeword belonging to V_1 but not V_2. Then v must be in some coset of V_2. Every element of such a coset is nonzero and belongs to V_1. Thus

there is a coset, of V_2, in which the minimum weight is at least d_1. Therefore, the covering radius of V_2 must be at least equal to d_1.

This concludes the proof of Theorem 3.3.

3.1.3 Lower Bound on the Covering Radius of Shortened Codes

The following lower bound applies to the covering radius of shortened-by-1-bit binary cyclic codes and will be used in our analysis, later in this chapter, of the cosets of the shortened-by-1-bit double-error-correcting BCH codes.

THEOREM 3.4

In the $(n - 1, k - 1)$ code V obtained by shortening the (n, k) binary cyclic code V by 1 bit, the covering radius is at least $d - 1$, where d is the minimum distance of V.

PROOF

Since V is cyclic, it has at least one codeword of weight d that starts with 1. This means that the coset

leader X^0 has in its coset a word of weight d - 1 that starts with 0. This in turn implies that V should have at least one coset with leader of weight d - 1.

This concludes the proof of Theorem 3.4.

We now present two examples to illustrate the use of the various bounds given here and in Chapter II.

EXAMPLE 3.1

The covering radius of the $(n = 2^m - 1, k = m + 1, t = \frac{n - 3}{4})$ BCH Code

This code, with minimum distance $d = \frac{n - 1}{2}$, is generated by $g(X) = (1 + X^n)/P(X)(1 + X)$, where P(X) is a primitive polynomial of degree m.

We note that this code is orthogonal to the even-weighted subset of the Hamming Code which contains the even-weighted codewords of weight 4, 6, ... to n - 3 inclusive. Therefore, using Lemma 2.2, we get

$$\Omega \leq \frac{n - 5}{2} .$$

Next we note that the code examined here can be defined, using Van der Horst's terminology, as $C(\frac{n-3}{4}, m)$ for which

$$\Omega \geq 2\left(\frac{n-3}{4}\right) - 1 = \frac{n-5}{2}$$

The covering radius of the $(2^m - 1, m + 1)$ BCH code is therefore $\frac{n-5}{2}$ exactly.

An interesting point to note here is that, for $m = 4$, the code is the 3-error-correcting (15, 5) code. The covering radius of this code is therefore $\Omega = \frac{15-5}{2} = 5$, a result that we report here because it was not explicitly mentioned by Berger and Van der Horst [36] in their analysis of the 3-error-correcting BCH codes. The reason given there was that the (15,5) code was not an $(n, n - 3m)$ code and would have required special treatment.

EXAMPLE 3.2

The covering radius of the (23, 11, d = 8) code

This code is the even-weighted subset of the (23, 12) binary Golay code with minimum distance 7.

According to Theorem 3.3, we have

$$\Omega \geq 7$$

since the (23, 11) code is a proper subgroup of the (23, 12) code.

On the other hand, the (23, 11) code is an optimum dimension code for which we can use the upper bound of Theorem 3.1, i.e.,

$$\Omega \leq 8 - 1 = 7$$

The covering radius of the (23, 11) code is therefore exactly 7.

In this example, we note that the upper bound on Ω could have also been obtained by using Delsarte's result (Lemma 2.2) since the (23, 11) code can also be treated as the dual of the (23, 12) Golay code which has 7 nonzero weights.

3.2 On the Coset Enumeration of MLSR codes

We now consider the problem of enumerating the cosets of the ($n = 2^m - 1$, $k = m$) MLSR codes for

several justifying reasons. The family of MLSR codes is an important family which has many practical applications [52] such as range-finding, synchronizing, modulation, scrambling, performance testing, etc .... We also wish to derive a coset enumeration that is solely dependent on the parameters of the codes, unlike Dick and Sloane's results mentioned in Chapter II. Finally, the analysis gives an insight into the complexity of the coset enumeration problem.

The results we present here, in the form of two theorems, are related to the covering radius of the MLSR codes and the number of cosets of weight t + 1.

THEOREM 3.5

The covering radius of the $(n = 2^m - 1, k = m)$ MLSR code is $\frac{n - 1}{2}$. The coset of weight $\frac{n - 1}{2}$ is unique.

PROOF

We first note that the MLSR code is a proper subgroup of the $(n = 2^m - 1, k = m + 1, t = \frac{n - 3}{4})$ BCH code of example 3.1, with minimum distance $\frac{n - 1}{2}$.

Thus, according to Theorem 3.3,

$$\Omega \geq \frac{n - 1}{2}$$

We also note, that the MLSR code is also an optimum dimension code for which Theorem 3.1 applies, that is,

$$\Omega \leq d - 1 = 2^{m - 1} - 1 = \frac{n - 1}{2}$$

The covering radius of the MLSR code is therefore equal to $\frac{n - 1}{2}$, thus proving the first part of the theorem.

Now let λ_Ω represent a coset leader of weight Ω and let μ_i denote a word in the coset. By definition

$$\mu_i = \lambda_\Omega + v_i$$

where v_i is a codeword belonging to the MLSR code, and

$$\mu_0 = \lambda_\Omega + v_0,$$

where v_0 is the all-zero codeword.

The set $\{v_i \cup u_i,\ i = 0, 1, \ldots, n\}$ is a group code with minimum distance $\frac{n-1}{2}$ and dimension $k = m + 1$. This code is none but the $(n = 2^m - 1,\ k = m + 1,\ t = \frac{n-3}{4})$ BCH code. Since the code is unique, then there must be only one coset of weight $\frac{n-1}{2}$.

This concludes the proof of Theorem 3.5.

COROLLARY 3.6

The coset of weight $\frac{n-1}{2}$ contains n words of weight $\frac{n-1}{2}$ and one word of weight n.

THEOREM 3.7

The number of cosets of weight $\frac{n+1}{4}$ in the MLSR code is given by

$$N_{\frac{n+1}{4}} = \binom{n}{\frac{n+1}{4}} + \frac{1}{3}\binom{n}{2} - \frac{n}{2}\binom{\frac{n+1}{2}}{\frac{n+1}{4}}$$

PROOF

Let N_ω denote the number of cosets of weight ω

and let v_i be any codeword of weight $\frac{n+1}{2}$ in the MLSR code of length n. Since the error-correcting capability t of the code is $t = \frac{d-2}{2} = \frac{n-3}{4}$, therefore

$$N_\omega = \binom{n}{\omega} \text{ for } w = 0, 1, 2, \ldots \frac{n-3}{4} \qquad (3.1)$$

We now estimate the number of cosets with leader of weight $t+1$, i.e., $\omega = \frac{n+1}{4}$. Let λ be such a coset leader. An upper-bound on the number of n-tuples of weight $\frac{n+1}{4}$, including λ, can be estimated using the following version of Johnson's bound [53] given by Freiman [54]: $A(n, u, 2u) \leq \left\lfloor \frac{n}{u} \right\rfloor$

So, in our case we have, at most

$$\left\lfloor \frac{n}{(n+1)/4} \right\rfloor = \left\lfloor \frac{4n}{n+1} \right\rfloor = \left\lfloor \frac{3n+3+n}{n+1} \right\rfloor = 3 \text{ n-tuples}$$

of weight $\frac{n+1}{4}$, including the coset leader λ. In other words, for each λ, there are at most two codewords which can produce words of $\frac{n+1}{4}$ in the coset. Therefore, the cosets with leaders of weight $\frac{n+1}{4}$ can be partitioned into three groups:

- Group A cosets containing three words of weight

$\frac{n+1}{4}$,

- Group B cosets containing two words of weight $\frac{n+1}{4}$,

- Group C cosets containing a unique word of weight $\frac{n+1}{4}$.

For Group A, let λ be the n-tuple of weight $\frac{n+1}{4}$ which is "common" to two codewords v_1 and v_2. This means that $\lambda + v_1 = w_1$ and $\lambda + v_2 = w_2$, where w_1 and w_2 have weight $\frac{n+1}{4}$. We note that w_1 and w_2 do not overlap (no ones in common) and, from the properties of the cosets, their sum is codeword v_3 different from v_1 and v_2. Thus, w_1 is "common" to v_1 and v_3 and w_2 is "common" to v_2 and v_3. Since two codewords can be chosen out of n codewords in $\binom{n}{2}$ ways, and since each λ exhausts three of these choices, hence the number of cosets in Group A is given by

$$N' = \frac{1}{3}\binom{n}{2} \qquad (3.2)$$

For Group B, let μ be an n-tuple of weight $\frac{n+1}{4}$, produced by converting $\frac{n+1}{4}$ ones into zeros in a v_i. Each v_i has a μ in common with only one other v_j.

Thus there are

$$\left[\binom{\frac{n+1}{2}}{\frac{n+1}{4}} - (n-1)\right] n$$

μ's which are not shared by any two codewords. The number of cosets in this group is therefore

$$N'' = \frac{n}{2}\left[\binom{\frac{n+1}{2}}{\frac{n+1}{4}} - (n-1)\right]. \qquad (3.3)$$

Finally, the number of cosets in Group C is simply the difference between the total number of n-tuples of weight $\frac{n+1}{4}$ and the number of words of weight $\frac{n+1}{4}$ accounted for in Groups A and B, that is

$$N''' = \binom{n}{\frac{n+1}{4}} - 3N' - 2N''$$

$$= \binom{n}{\frac{n+1}{4}} - \binom{n}{2} - \left[\binom{\frac{n+1}{2}}{\frac{n+1}{4}} - (n-1)\right] n. \qquad (3.4)$$

The total number of cosets with leaders of weight $\frac{n+1}{4}$ is therefore given by

$$N_{\frac{n+1}{4}} = N' + N'' + N'''$$

$$= \binom{n}{\frac{n+1}{4}} + \frac{1}{3}\binom{n}{2} - \frac{n}{2}\binom{\frac{n+1}{2}}{\frac{n+1}{4}} \qquad (3.5)$$

This concludes the proof of Theorem 3.7 and the analysis of the cosets of MLSR codes.

3.3 Shortened-by-1-bit Double-Error-Correcting Binary BCH Code

3.3.1 Coset Enumeration

It may appear, at first, that enumerating cosets of a shortened code is an easy task, once the coset enumeration of the parent code is known. This is not true, however, because of the simple fact that the weight spectrum of the shortened code will be different from that of the parent code, and so will be the coset enumeration. In this context we recall that the shortened-by-1-bit Hamming code has been shown to be quasi-perfect, rather than perfect [5, p. 142].

Let V be the $(n = 2^m - 1, k = n - 2m)$ binary primitive double-error-correcting BCH code with minimum

distance 5, and let V' be the $(n - 1, k - 1)$ code obtained by shortening V by 1 bit.

From Theorem 3.4, we see that the covering radius of V' should be at least equal 4. The detailed analysis will show that the covering radius of V' is exactly 4. The main result is given in Theorem 3.11 below.

To start the analysis, we first state, without proof the following known lemmas

LEMMA 3.8 [20]

The code V is quasi-perfect.

LEMMA 3.9 [5, Theorem 9.2]

With $m \geq 4$, the code V has 2m check bits, implying that it has $2^{n-k} = 2^{2m} = (n + 1)^2$ cosets.

Combining Lemmas 3.8 and 3.9, we can show that the coset enumeration of V is given by the following Lemma..

LEMMA 3.10

The $(n = 2^m - 1, k = n - 2m)$ binary two-error-correcting BCH code has one coset of weight 0, n cosets of weight 1, $\frac{n(n-1)}{2}$ cosets of weight 2 and $\frac{n(n+3)}{2}$ cosets of weight 3.

Now let us examine in some detail the shortening operation which produces V' from V. This is simply done by making the first leading information symbol identically 0 and omitting it from all code vectors. In other words, we partition the set of 2^k codewords of the code V into two subsets, U_0 and U_1, each containing 2^{k-1} codewords, such that U_0 contains all codewords which have a 0 in the first leading position and U_1 contains all codewords having a 1 in the first leading position. The $(n - 1, k - 1)$ code then consists of the 2^{k-1} elements of U_0 with the first position omitted. All error patterns that do not have a 1 in the first position will be correctable by V'. The error patterns that did have a 1 in the first position will have to be replaced by an error pattern of minimum weight that has the same syndrome and originally had a 0 in the first leading position before being shortened. The complete coset decomposition of

the shortened code will therefore require finding those replacements.

Let us now examine in greater detail the cosets of the code V and let us define

$\lambda_{i,0} \triangleq$ A coset leader of weight i, starting with a 0,

$N_{i,0} \triangleq$ Number of cosets of weight i, starting with a 0,

$\mu_{i,0} \triangleq$ An n-tuple of weight i, starting with a 0, that is not a coset leader but occurs in the cosets,

$v_{i,0} \triangleq$ A codeword of weight i and starting with a 0,

and similarly define $\lambda_{i,1}$, $N_{i,1}$, $\mu_{i,1}$ and $v_{i,1}$ as those parameters starting with a 1.

The elements of the cosets can then be shown in the following manner:

0	$\{v_{5,0}\}$	$\{v_{5,1}\}$	$\{v_{6,0}\}$	$\{v_{6,1}\}$	$\{v_{7,0}\}$	$\{v_{7,1}\}\cdots$
$\{\lambda_{1,0}\}$	$\{\mu_{4,0}\}$	$\{\mu_{4,1}\}$	$\{\mu_{5,0}\}$	$\{\mu_{5,1}\}$	$\{\mu_{6,0}\}$	$\{\mu_{6,1}\}\cdots$
$\{\lambda_{1,1}\}$	$\{\mu_{6,1}\}$	$\{\mu_{4,0}\}$	$\{\mu_{7,1}\}$	$\{\mu_{5,0}\}$	$\{\mu_{8,1}\}$	$\{\mu_{6,0}\}\cdots$
$\{\lambda_{2,0}\}$	$\{\mu_{3,0}\}$	$\{\mu_{3,1}\}$	$\{\mu_{4,0}\}$	$\{\mu_{4,1}\}$	$\{\mu_{5,0}\}$	$\{\mu_{5,1}\}\cdots$
$\{\lambda_{2,1}\}$	$\{\mu_{5,1}\}$	$\{\mu_{3,0}\}$	$\{\mu_{6,1}\}$	$\{\mu_{4,0}\}$	$\{\mu_{7,1}\}$	$\{\mu_{5,0}\}\cdots$
$\{\lambda_{3,0}\}$	$\{\mu_{4,0}\}$	$\{\mu_{4,1}\}$	$\{\mu_{3,0}\}$	$\{\mu_{3,1}\}$	$\{\mu_{4,0}\}$	$\{\mu_{4,1}\}\cdots$
$\{\lambda_{3,1}\}$	$\{\mu_{4,1}\}$	$\{\mu_{4,0}\}$	$\{\mu_{5,1}\}$	$\{\mu_{3,0}\}$	$\{\mu_{6,1}\}$	$\{\mu_{4,0}\}\cdots$

where we have only shown the set of minimum weight n-tuples that result from the operation $\mu = v + \lambda \pmod 2$. To clarify further, let us consider for example $\{v_{5,1}\}$ which is the set of all codewords of weight 5 starting with 1, and $\{\mu_{3,0}\}$, the set of all coset leaders of weight 3 starting with 0. The operation $\{v_{5,1}\} + \{\lambda_{3,0}\}$ in GF(2) may yield $\{\mu_{2,1}\}$ and/or $\{\mu_{4,1}\}$ and/or $\{\mu_{6,1}\}$ and/or $\{\mu_{8,1}\}$. Since $\{\mu_{2,1}\}$ has already been exhausted in the cosets $\{\lambda_{2,1}\}$ and we are only interested in the minimum weight, we list $\{\mu_{4,1}\}$ at the intersection of row $\{\lambda_{3,0}\}$ and column $\{v_{5,1}\}$.

The problem of determining the coset enumeration of the shortened code V' consists therefore of examining the cosets $\{\lambda_{i,1}\}$, $i = 1, 2, 3$ to determine the minimum weight μ in each coset. This examination yields the following result.

THEOREM 3.11

The covering radius of the shortened code V' is 4 for all $m \geq 4$.

For $m > 4$, the coset enumeration of V' is

$$N_0' = 1,\ N_1' = n - 1,\ N_2' = \frac{(n-1)(n-2)}{2},$$

$$N_3' = \frac{n^2 + 5n - 2}{2},\ N_4' = 1,\ N_5' = N_6' = \ldots = 0.$$

For $m = 4$, the coset enumeration of the $(14, 6, 2)$ code is

$$N_0' = 1,\ N_1' = 14,\ N_2' = 91,\ N_3' = 134,\ N_4' = 16.$$

The proof of Theorem 3.11 is given below through a series of lemmas for the sake of clarity. Before proceeding we remark from Theorem 3.11 that for $m > 4$, the shortened code V' is nearly quasi-perfect, since there is only one coset of maximum weight 4. The implication of that remark will become clearer when we compare the data compression performance of V' with that of its parent code V.

We now start the proof of Theorem 3.11 with the following.

LEMMA 3.12

The coset with leader $\lambda_{1,1}$ contains at least one word, of weight 4, starting with 0.

PROOF

The coset leader $\lambda_{1,1}$ has the form X^0 and is therefore unique. In its coset, there must be at least one word of weight 4 starting with 0 since there is at least one codeword $v_{5,1}$, that is, of minimum weight and starting with 1.

This concludes the proof of Lemma 3.12. We now consider the cosets of weight 2.

LEMMA 3.13

For any $m > 4$, the coset with leader $\lambda_{2,1}$ contains at least one word, of weight 3, starting with 0.

PROOF

A coset leader belonging to $\{\lambda_{2,1}\}$ must have the form $1 + X^a$, where $a \in \{I\} = \{1, 2, \ldots, n-1\}$. We associate with $\lambda_{2,1}$ a syndrome $(S_1, S_3) \subset GF(2^m)$ with

$$S_1 = 1 + \alpha^a \tag{3.6}$$

and $$S_3 = 1 + \alpha^{3a}. \tag{3.7}$$

To find a replacement for $1 + X^a$, we must prove the existence of a word of weight 3 of the form $X^c + X^d + X^e$, where (c, d, e) are distinct from (0, a), with the same syndrome (S_1, S_3). We note here that this is the equivalent of proving the existence of a codeword of weight 5 of the form $1 + X^a + X^c + X^d + X^e$ belonging to the parent code V for all values of $a \in \{I\}$.

We must therefore seek a solution for the following set of simultaneous equations

$$S_1 = \alpha^c + \alpha^d + \alpha^e, \tag{3.8}$$

$$S_3 = \alpha^{3c} + \alpha^{3d} + \alpha^{3e}, \tag{3.9}$$

for the given syndrome (S_1, S_3).

With $\alpha^u = S_1^2 + S_1$, we find, using (3.6) and (3.7) that

$$S_3 = S_1^3 + \alpha^u. \tag{3.10}$$

We also define α^f to be

$$\alpha^f = S_1 + \alpha^e. \tag{3.11}$$

We then cube (3.8) and subtract from (3.9) to get, using (3.10) and (3.11),

$$\alpha^c + d = \alpha^{u-f} + \alpha^e(\alpha^e + \alpha^f). \tag{3.12}$$

Finally, we form the quadratic

$$z^2 + \alpha^f z + [\alpha^{u-f} + \alpha^e(\alpha^e + \alpha^f)] = 0 \tag{3.13}$$

the roots of which, α^c and α^d, are a function of S_1, S_3 and α^e. As is well known [55] these roots belong to $GF(2^m)$ if and only if

$$Tr\left[\frac{\alpha^{u-f} + \alpha^e(\alpha^e + \alpha^f)}{\alpha^{2f}}\right] = 0 \tag{3.14}$$

where the trace operator $Tr[.]$, mapping $GF(2^m)$ into

GF(2), is defined by

$$Tr[\gamma] = \sum_{i=0}^{m-1} \gamma^{2^i}, \quad \gamma \in GF(2^m) \tag{3.15}$$

and satisfies the property

$$Tr\left[\gamma^2\right] = Tr[\gamma]^2 = Tr[\gamma]. \tag{3.16}$$

Using (3.16) in (3.14), we see that

$$Tr[\alpha^{u-3f}] = Tr[\alpha^j] = 0 \tag{3.17}$$

where $j = u - 3f$, is the necessary and sufficient condition for the roots α^c and α^d to belong to $GF(2^m)$.

We will now attempt to find a solution to (3.17) by a detailed examination of the even and odd m cases separately.

For even m, we have the following result.

LEMMA 3.14.1

The number of distinct solutions of $Tr[\alpha^{u-3f}] = 0$ in $GF(2^m)$, m even is

$$n_f = \begin{cases} \dfrac{n - 1 \pm 2\sqrt{n+1}}{2} & \text{, when u is a multiple of 3} \\ \dfrac{n - 1 \mp \sqrt{n+1}}{2} & \text{otherwise.} \end{cases}$$

PROOF

For even m, the number of distinct and finite values of j such that $Tr[\alpha^j] = 0$ is known to be

$$\frac{n - 1 + 2\sqrt{n+1}}{6} \qquad (3.18)$$

when j (and u) is a multiple of 3, and

$$\frac{n - 1 \mp \sqrt{n+1}}{3}$$

otherwise, using Welch's theorem [7, p.423]. There, $\pm\sqrt{n+1}$ is defined as $-(-2)^{m/2}$ and both expressions are easily verified to be exponentially increasing functions of m.

From (3.18) and (3.19) we now compute n_f, the number of distinct values of f, which is also the number of distinct solutions of $Tr[\alpha^{u - 3f}] = 0$ for a

given u.

When u is a multiple of 3, then each α^j is a nonzero cube in $GF(2^m)$, yielding three distinct values of f, namely $\left(\frac{u - j}{3}\right) \bmod n$, $\left(\frac{u - j + n}{3}\right) \bmod n$ and $\left(\frac{u - j + 2n}{3}\right) \bmod n$.

When u is not a multiple of 3, then it can be easily shown that only one half of the distinct values of j (given by 3.19) yields an expression $\alpha^{u - j}$ that is a nonzero cube from which three distinct values of f are obtained.

This concludes the proof of Lemma 3.14.1 which dealt with the even m case. For odd m, we have the following result.

LEMMA 3.14.2

The number of distinct solution of $Tr[\alpha^{u-3f}] = 0$ in $GF(2^m)$, m odd, is

$$n_f = \frac{n - 1}{2}$$

PROOF

For odd m, no expressions similar to (3.18) and (3.19) were found. It turns out, however, that there is a simple solution to the trace equation $Tr[\alpha^{u} - 3f] = 0$ in $GF(2^m)$, as shown below.

There are $(n + 1)$ elements in $GF(2^m)$, and exactly one half of these elements has $Tr[.] = 0$ while the other half has $Tr[.] = 1$. Therefore $Tr[\alpha^{j}] = 0$ has $\frac{n+1}{2}$ solutions. Note, however, that $Tr[\alpha^{-\infty}] = [Tr\ 0] = 0$, and since we are only interested in the finite values of j, we are left with $\frac{n-1}{2}$ possible distinct solutions of $Tr[\alpha^{j}] = 0$. We now show that n_f, the number of distinct values of $f = \frac{u-j}{3}$, is also equal to $\frac{n-1}{2}$. Regardless of the particular values of u and j, α^{u-j} has a unique cubic root in $GF(2^m)$ since, with odd m, $n = 2^m - 1$ is not divisible by 3 and only one of the following: $u - j$, $u - j + n$, $u - j + 2n$ can be a multiple of 3. This means that for each value of j, such that $Tr[\alpha^{j}] = 0$, corresponds a unique value of f. If only remains to be shown that these values of f are distinct. Assuming that two different values of j yield the same value of f, we would have

$$u - j_1 + i_1 n = u - j_2 + i_2 n$$

or $\quad j_1 - j_2 = (i_1 - i_2)n,$

implying that $(j_1 - j_2)$ is a multiple of n, which is impossible since both j_1 and j_2 are less than n.

This concludes the proof of Lemma 3.14.2 which dealt with the odd m case.

We now resume the proof of Lemma 3.13, having derived the required number of solutions of $Tr[\alpha^{u} - 3f]$ for even and odd m.

From Lemmas 3.14.1 and 3.14.2, and the translation $\alpha^{e} = S_1 + \alpha^{f}$ (3.11), it immediately follows that n_f is also the number of distinct values of e for which the quadratic (3.13) would have its roots in $GF(2^m)$. It is to be noted, however, that not all n_f values yield a solution satisfying the crucial requirement that 0, a, c, d and e be all distinct. In fact $\alpha^{e} = 0$, $\alpha^{e} = 1$ and $\alpha^{e} = \alpha^{a}$ can all be derived from the set of n_f solutions of $Tr[\alpha^{u} - 3f] = 0$. For example, with $\alpha^{e} = \alpha^{a}$, then $\alpha^{f} = S_1 + \alpha^{e} = S_1 + \alpha^{a} = 1$ since $S_1 = 1 + \alpha^{a}$. So $\alpha^{\frac{u - j}{3}} = 1$. By cubing we get $\alpha^{u - j} = 1$

or $\alpha^u = \alpha^j$. Now, since $\alpha^u = S_3 + S_1^3 = \alpha^a(1 + \alpha^a)$, then $Tr[\alpha^u] = Tr[\alpha^a + \alpha^{2a}] = 0 = Tr[\alpha^j]$. Therefore j is an element of the set of n_f values from which f is obtained. The proof is similar for $\alpha^e = 0$ and $\alpha^e = 1$.

Therefore, considering the fact that e = 0, e = a and e = $-\infty$ lead to the trivial solutions (c, d) = (a, $-\infty$), (0, $+\infty$) and (0, a) respectively, we are left with (n_f - 3) distinct values of e for which 0, a, c, d and e are all distinct. Lemmas 3.14.1 and 3.14.2 show that this result is true for all values of m except for m = 4 and u/3, when there is a unique nonzero field element with Tr[.] = 0, namely $\alpha^0 = 1$. For this case, $n_f = 3$ and no nontrivial solution to (3.13) is to be obtained. Direct verification of the (15, 7, 2) code shows that when $\lambda_{2,1}$ is $1 + X^5$ or $1 + X^{10}$, there are no words of weight 3 in the coset.

This concludes the proof of Lemma 3.13 which dealt with the cosets of weight 2. We have essentially shown that for any m > 4, any coset with leader of the form $\lambda_{2,1}$ contains at least one word of weight 3 starting with 0. From Lemma 3.13 we can also conclude the following result.

COROLLARY 3.15

For any $m > 4$, and for any $a \in \{1, 2, \ldots, n-1\}$, there is at least one codeword of weight 5 in V containing $1 + X^a$.

This completes the analysis of the cosets of weight 2 and we now examine the cosets of weight 3.

LEMMA 3.16

For any $m > 4$, the coset with leader $\lambda_{3,1}$ contains at least one word of weight 3 starting with 0.

PROOF

A coset leader belonging to $\{\lambda_{3,1}\}$ has the form $1 + X^a + X^b$ where a and b are distinct and $(a, b) \in \{I\} = \{1, 2, \ldots, n-1\}$. The syndrome associated with any such coset has components

$$S_1 = 1 + \alpha^a + \alpha^b, \tag{3.20}$$

$$S_3 = 1 + \alpha^{3a} + \alpha^{3b}, \tag{3.21}$$

with $Tr\left[\frac{S_3 + S_1^3}{S_1^3}\right] = 1.$

The latter relationship (3.22) is implied by the fact that its complement, i.e., $Tr[.] = 0$ is true only for the coset leaders of weight 2.

We now follow the same procedure used for the coset leaders $\lambda_{2,1}$, that is, we now search for the words of minimum weight in the coset. Assuming that minimum weight to be 3, we have to prove the existence of a word of weight 3 of the form $X^c + X^d + X^e$, such that 0, a, b, c, d and e are all distinct, with the same syndrome components S_1 and S_3 given by (3.20) and (3.21).

We therefore set

$$1 + \alpha^a + \alpha^b = S_1 = \alpha^c + \alpha^d + \alpha^e, \tag{3.23}$$

$$1 + \alpha^{3a} + \alpha^{3b} = S_3 = \alpha^{3c} + \alpha^{3d} + \alpha^{3e} \tag{3.24}$$

then form the quadratic

$$z^2 + \alpha^f z + \alpha^{u-f} + \alpha^e(\alpha^e + \alpha^f) = 0 \tag{3.25}$$

where $\alpha^u = S_3 + S_1^3 = (\alpha^a + \alpha^b)(1 + \alpha^a + \alpha^b + \alpha^{a+b})$

and $\alpha^f = S_1 + \alpha^e$.

By observing the right-hand sides of (3.23) and (3.24), as well as the quadratic (3.25), we note that they are identical to (3.8), (3.9) and (3.13) respectively. Also, in discussing the solutions of the quadratic (3.13), the knowledge of the explicit values of either S_1, S_3 or α^u was not required. This strongly suggests that the results obtained earlier, in Lemmas 3.14.1 and 3.14.2 can be used here.

For even m we recall that the number of distinct values of j such that $Tr[\alpha^j] = 0$ was given by (3.18) and (3.19). Both expressions were also referred to by Berlekamp [7, pp. 426-428] as the multiplicity of minimum weight elements in the cosets of weight 3, that is, the number of words of weight 3 in a coset with leader of weight 3 (coset leader included). For any $m > 4$, (3.18) and (3.19) yield an integer result greater than unity. Combining this with the fact that in the cosets $\lambda_{3,1}$ any word of weight 3 must start with 0, since otherwise the code V would contain codewords of weight 4, we conclude that for any even m greater than 4, any coset with leader $\lambda_{3,1}$ contains at least

one word of weight 3 starting with 0.

For m = 4, direct verification shows that the minimum weight in the cosets λ_3 is at most 4.

For odd m, and from Lemma 3.14.2, the number of possible values of e was found to be $n_f = \frac{n-1}{2}$. Here, one would assume at first that four values of e, namely 0, a, b and $-\infty$ would have to be discarded in order to get a nontrivial solution for (3.25). However, the value $e = -\infty$ needs not be considered, as we show in the following discussion.

If $e = -\infty$ then $\alpha^{\frac{u-j}{3}} = S_1$, leading to $\alpha^j = \alpha^u / S_1^3$ $\frac{S_3 + S_1^3}{S_1^3}$. Since we are stipulating that $\mathrm{Tr}\left[\frac{S_3 + S_1^3}{S_1^3}\right] = 1$, then $\mathrm{Tr}[\alpha^j] \neq 0$, which contradicts the initial assumption that $\mathrm{Tr}[\alpha^j] = 0$.

We have therefore $\frac{n-7}{2}$ distinct values of e for which 0, a, b, c, d and e are all distinct.

This concludes the proof of Lemma 3.16 which dealt with the cosets of weight 3.

We now summarize the results of Lemmas 3.12, 3.13 and

3.16. In the coset with leader $\lambda_{1,1}$, there is at least one word of weight 4 starting with 0. For the cosets with leaders $\lambda_{2,1}$ and $\lambda_{3,1}$, for all m > 4 each coset contains at least one word of weight 3 starting with 0. For m = 4, the minimum weight is at most 4 for $\lambda_{2,1}$ and $\lambda_{3,1}$. Since the shortening of the code V involves rearranging the cosets to replace those leaders starting with 1 by words of the same coset starting with 0, the above results show that the covering radius of the shortened code V' is equal to 4, thus proving the first part of Theorem 3.11. To prove the remaining part of the Theorem, we now derive the coset enumeration of the shortened code V' from that of V.

The coset enumeration of the parent code V is known to be, from Lemma 3.10

$$N_0 = 1,\ N_1 = n,\ N_2 = \frac{n(n-1)}{2},\ N_3 = \frac{n(n+3)}{2}.$$

The leader of weight 0 remains unchanged in V', therefore $N_0' = 1$. The (n - 1) leaders of weight 1 and the $\frac{(n-1)(n-2)}{2}$ leaders $\lambda_{2,0}$ yield $N_1' = n - 1$ and $N_2' = \frac{(n-1)(n-2)}{2}$ respectively.

On the other hand, the leader $\lambda_{1,1}$ is replaced by a leader of weight 4. For m > 4, each of the (n - 1) coset leaders $\lambda_{2,1}$ and all the $\lambda_{3,1}$ leaders are replaced by words

of weight 3 starting with zero. Therefore $N_3' = \frac{n(n+3)}{2} + n - 1 = \frac{n^2 + 5n - 2}{2}$ and $N_4' = 1$.

This concludes the proof of Theorem 3.11.

3.3.2 Average Distortion per bit

The average distortion per bit resulting from the use of V' for data compression can be obtained by substituting the results of Theorem 3.11 into Equation (2.2).

We therefore have

$$D' = \frac{5n^2 + 11n + 4}{2(n+1)^2 (n-1)} \quad \text{for } n = 2^m - 1,\ m > 4$$

and $D' = 0.18471$ for the (14, 6, 2) code.

In this context we note that the average distortion per bit of the parent code V can be easily shown to be

$$D = \frac{5n + 9}{2(n+1)^2}.$$

Table 1 shows the average distortion for both V and V' as well as their respective compression ratios, computed for moderate values of n. The results shown indicate that the respective average distortion values for both V and V'

Table 1

Average Distortion & Compression Ratios

	Code V				Code V'			
m	n	k	$CR=^{n}/_{k}$	D	n'	k'	$CR=^{n}/_{k}$	D'
4	15	7	2.1429	0.1641	14	6	2.3333	0.18471
5	31	21	1.4762	0.080078	30	20	1.5	0.083822
6	63	51	1.2353	0.039551	62	50	1.24	0.040445
7	127	113	1.1239	0.019653	126	112	1.125	0.019872
8	255	239	1.0669	0.009796	254	234	1.0672	0.009850
9	511	493	1.0365	0.004890	510	492	1.0366	0.004904
10	1023	1003	1.0199	0.002443	1022	1022	1.02	0.002447
11	2047	2025	1.0109	0.001221	2046	2024	1.0109	0.001222
12	4095	4071	1.0059	0.000610	4094	4070	1.0059	0.000611

converge towards a common figure as the length increases, despite the fact that the covering radius of V' is one unit larger than that of V. Since both compression ratios are also comparable, the shortened code V' can practically be used instead of V if such use is imposed by system constraints.

3.3.3 Number of Codewords of Weight 5 in the $(2^m - 1, n - 2m, 2)$ BCH Code with odd m

The result we are presenting here may not be strictly in line with the main topic of the thesis. It is included here, however, because it is a natural consequence of Lemma 3.14.2, thus confirming its validity and stressing its importance.

The number of codewords of weight 5 in the double-error-correcting BCH code with even m is known to be

$$A_5 = \frac{n(n-3)^2}{5!} .$$

This result has been derived in many ways, one of which is due to Berlekamp [7], who used the expressions for the multiplicity of minimum weight elements (3.18) and (3.19).

For the odd m case, a closed form expression for A_d is not as readily available. A straightforward but tedious method of deriving this expression would be to use Kasami's weight enumeration of the orthogonal code [56] with MacWilliam's identities [57].

The method presented here is much simpler. It combines the results of Lemma 3.14.2 with arguments similar to Berlekamp's to yield the desired result, given in the following theorem.

THEOREM 3.17

For odd m, the number of codewords of weight 5 in the $(n = 2^m - 1, k = n - 2m, t = 2)$ BCH code is given by

$$A_5 = \frac{n(n-1)(n-7)}{5!}$$

PROOF

From Lemma 3.14.2 and the proof of Lemma 3.13, we recall that there were $(n - 7)/2$ values of e for which the quadratic (3.13) had a non-trivial solution. Since for each value of e obtained corresponds one pair (c, d), the total number of distinct triplets (c, d, e) is therefore

$(n - 7)/6$. This gives the number of words of weight 3 in a coset of weight 2. For all cosets of weight 2, the total number of words of weight 3 is therefore $\frac{n-7}{6}\binom{n}{2}$, and since each word of weight 3 can be obtained from $\binom{5}{2} = 10$ different codewords of weight 5, the total number of codewords of weight 5 is therefore

$$A_5 = \frac{n(n-1)(n-7)}{5!}$$

for odd m.

This concludes the proof of Theorem 3.17.

3.4 Bounds on the Coset Enumeration of Shortened-by-s-bits Codes

After completing the analysis of the cosets of the shortened-by-1-bit codes in Sec. 3.3, we now consider the possibility of using similar arguments to analyse the cosets of shortened-by-s-bits codes, for $s > 1$. The complexity of the problem, however, quickly grows beyond control for several reasons. The weight distribution of the newly shortened code is once again altered and cannot readily be deduced from that of the parent code. In addition, the covering radius increases to a new value that cannot be estimated with any of the existing bounds. Furthermore, since

the parent code is no longer cyclic, none of the cyclic properties of the original code can be used.

Instead of following a path similar to the one already used in the preceding section, we attempt to use strict combinatorial arguments to estimate the covering radius of the shortened-by-s-bits code. We note here that the method we are about to use is applicable to any binary cyclic code with an odd-weight generator polynomial.

With reference to the (n, k, t) binary cyclic code V_0, let V_s represent the $(n - s, k - s, t_s)$ code obtained from the shortening of V_0 by s bits, where $0 \leq s \leq k - 1$. Let $N_{\omega,s}$ represent the number of cosets of V_s with leaders of weight ω. We assume both the weight W of the generator polynomial of V and the minimum distance d_s of V_s to be odd.

We now discuss an estimate, of $N_{\omega,s}$, obtained starting with V_{k-1}, the $(n - k + 1, 1)$ code consisting of two codewords, the all-zero codeword and g(x) the generator polynomial of V_0 of weight W.

Let us consider the cosets of V_{k-1}. Since the minimum nonzero weight in V_{k-1} is W, then, for $\omega \leq t_{k-1} = \frac{W - 1}{2}$,

$$N_{\omega,\ k-1} = \binom{\upsilon}{w}, \tag{3.26}$$

where $\upsilon = n - k + 1$

For $\omega \geq \frac{W+1}{2}$, we note that V_{k-1} has only one nonzero codeword and that a leader of odd (even) weight produces only words of even (odd) weight in its coset. Combining these facts we see that a leader, of weight $j \geq \frac{W+1}{2}$, produces in its coset $\binom{W}{\frac{W+j-i}{2}}\binom{\upsilon - W}{j - \frac{W+j-i}{2}}$ words of weight i in its coset. This means that, for $\omega \geq \frac{W+1}{2}$,

$$N_{\omega,\ k-1} = \binom{\upsilon}{\omega} - \sum_j \binom{W}{\frac{W-\omega+j}{2}} \binom{\upsilon - W}{j - \frac{W-\omega+j}{2}},$$

where the summation is over appropriate j's. After suitable manipulation we get the following

THEOREM 3.18

For V_{k-1},

$$N_{\omega,\ k-1} = \binom{\upsilon}{w} \quad \text{for } \omega \leq \frac{W-1}{2},$$

and

$$N_{\omega,\ k-1} = \binom{v}{\omega} - \binom{v-W}{0}\binom{W}{W-\omega} + \binom{v-W}{1}\binom{W}{W-\omega+1} +$$

$$+ \binom{v-W}{2}\binom{W}{W-\omega+2} + \ldots + \binom{v-W}{\frac{2\omega-W-1}{2}}\binom{W}{\frac{W-1}{2}}$$

for $\omega \geq \frac{W+1}{2}$.

Indicating the covering radius of V_s by Ω_s, we get, from Theorem 3.18, the following

COROLLARY 3.19

For V_{k-1},

$$\Omega_{k-1} = n - k + 1 - \frac{W+1}{2} \quad \text{and} \quad N_{\Omega_{k-1},\ k-1} = \binom{W}{\frac{W-1}{2}}.$$

Now let us consider the cosets of V_s for $s < k-1$. Clearly, treating the leaders of V_{k-1} as $(n-s)$-tuples, we can construct a set of cosets, of V_s, in which the leader does not necessarily have the minimum weight in its coset.

We note that the words of weight $t_s + 1$ can possibly

be produced by adding leaders of weight t_s to codewords of weight $2t_s + 1$ and by adding leaders of weight $t_s + 1$ to codewords of weight $2t_s + 2$. Also, the number of r-tuples which have weight 2u and have u 1's in the same u positions is at most $\left\lfloor \frac{r - u}{u} \right\rfloor$ where $\lfloor x \rfloor$ indicates the largest integer contained in x. Combining these arguments we get, from Theorem 3.18, the following

COROLLARY 3.20

For V_s with $s < k - 1$,

$$N_{\omega,s} = \binom{n - s}{\omega} \quad \text{for } \omega \leq t_s,$$

$$N_{\omega,s} \geq \frac{\binom{n - s}{t_s + 1} - A_{d_s}\binom{2t_s + 1}{t_s}}{1 + \left\lfloor \frac{n - s - (t_s + 1)}{t_s + 1} \right\rfloor} \quad \text{for } \omega = t_s + 1$$

$$N_{\omega,s} \leq N_{\omega,k-1} \text{ for } \omega \geq t_s + 2,$$

where A_{d_s} is the number of words of weight d_s in V_s.

Corollary 3.20 is clearly not relevant when, for a given V_s, $N_{\omega,s}$'s are known for all ω as, for example, in the case of perfect and quasiperfect codes. It is most

useful when no, or partial information is available about the cosets. The difference between the estimate of the performance of the code, as obtained from Corollary 3.20, and the real value decreases as s is increased. In particular, the estimate is a fairly good indication of the performance if V_0 is a low rate short code.

To illustrate the preceding points we now consider the problem of data-compression. In this connection we note that the average distortion per bit, D_s, for V_s is given by

$$D_s = \frac{\sum_{\omega=0}^{n} \omega N_{\omega,s}}{2^{n-k}(n-s)} \qquad (3.27)$$

By way of an example to show how Corollary 3.20 is applied, let us take the (15, 5, 3) BCH code as V_0 and consider the case of s = 0. This code has

$$G(X) = 1 + X + X^2 + X^4 + X^5 + X^8 + X^{10}$$

so that W = 7 and $\upsilon - W = n - k + 1 - W = 4$. Also $A_{d_s} = 15$. From Corollary 3.20 we get

$$N_{0,0} = 1,$$

$$N_{1,0} = \binom{15}{1} = 15,$$

$$N_{2,0} = \binom{15}{2} = 105,$$

$$N_{3,0} = \binom{15}{3} = 455,$$

$$N_{4,0} \geq \frac{\binom{15}{4} - 15\binom{7}{3}}{1 + \left\lfloor \frac{15 - 4}{4} \right\rfloor} = \frac{1365 - 525}{1 + 2} = 280,$$

$$N_{5,0} \leq \binom{11}{5} - \binom{4}{0}\binom{7}{2} - \binom{4}{1}\binom{7}{3} = 462 - 21 - 140 = 301,$$

$$N_{6,0} \leq \binom{11}{7} - \binom{4}{0}\binom{7}{1} - \binom{4}{1}\binom{7}{2} - \binom{4}{2}\binom{7}{3} =$$

$$462 - 7 - 84 - 210 = 161,$$

$$N_{7,0} \leq \binom{11}{7} - \binom{4}{0}\binom{7}{0} - \binom{4}{1}\binom{7}{1} - \binom{4}{2}\binom{7}{2} - \binom{4}{3}\binom{7}{3} =$$

$$330 - 1 - 28 - 126 - 140 = 35$$

Since V_0 has $2^{10} = 1024$ cosets,

$$N_{5,0} + N_{6,0} + N_{7,0} \leq 1024 - 1 - 15 - 455 - 280 = 168$$

With reference to (3.27)

$$D_0 < \frac{0\cdot1 + 1\cdot15 + 2\cdot105 + 3\cdot455 + 4\cdot280 + 7\cdot35 + 6\cdot(168 - 35)}{1024\cdot15}$$

$$= .2443$$

whereas the actual value of D_0 is known to be 0.2220.

3.5 Interleaved Codes

3.5.1 The Coset Enumerator of Interleaved Codes

Let V_i represent a binary (n_i, k_i) block code, where $i = 1, 2, \ldots, \mu$. Let V represent the $(n = \sum_i n_i, k = \sum_i k_i)$ code obtained by interleaving the codes V_i.

Let $N_{i\omega}$ denote the number of coset leaders of weight ω in the partitioning of the set of 2^{n_i} n_i-tuples of V_i. It is obvious that

$$N_{i0} + N_{i1} + \ldots + N_{i\Omega_i} = 2^{n_i} - k_i \qquad (3.28)$$

where $N_{i0} = 1$ by definition,
and Ω_i is the maximum coset leader weight for V_i.

Let us now redefine $N_{i\omega}$ through the use of a coset enumerator C as

$$N_{i\omega} \triangleq N_{i\omega}\, C^{\omega}, \qquad (3.29)$$

so the coset enumeration of a code V_i will be given by

$$\sum_{j=0}^{\Omega_i} N_{ij} C^j = 1 + N_{i1}C + N_{i2}C^2 + \ldots + N_{i\Omega_i} C^{\Omega_i} \quad (3.30)$$

With respect to the cosets of the interleaved code V, let N_k denote the number of cosets of weight k. This number is simply the total number of ways of adjoining the n_1, n_2, ..., n_μ-tuples to form an n-tuple ($n = \sum_i n_i$) of weight k, that is

$$N_k = \sum_{\substack{k_1+k_2+\ldots+k_\mu=k \\ 0\le k_1\le\Omega_1 \\ 0\le k_2\le\Omega_2 \\ 0\le k_\mu\le\Omega_\mu}} N_{1k_1} N_{2k_2} N_{3k_3} \cdots N_{\mu k_\mu} \quad (3.31)$$

A careful examination of (3.31) reveals that N_k is the coefficient of C^k in the expansion

$$\prod_{i=1}^{\mu} \sum_{j=0}^{\Omega_i} N_{ij} C^j \quad (3.32)$$

Example: Consider the interleaving of the following two codes:

V_1 = (14, 10) shortened-by-1-bit Hamming code

V_2 = (23, 12) Golay code

Using (3.30) the coset enumeration of V_1 and V_2 can be written as

$$1 + 14C + 1\ C^2 \text{ for } V_1$$

and $1 + 23C + 253\ C^2 + 1771\ C^3$ for V_2.

The number of cosets with leaders of weight 4 in the (37, 22) interleaved code is, according to (3.32) the coefficient of C^4 in the expression

$$(1 + 14C + C^2)(1 + 23C + 253\ C^2 + 1771\ C^3),$$

i.e., $14 \times 1771 + 253 = 25047$.

3.5.2 Interleaved Hamming Codes

We now apply the results of the preceding section to the interleaving of Hamming codes. In this context, let V_i represent the $(n_i = 2^{m_i} - 1,\ k_i = n_i - m_i)$ binary Hamming code, where $i = 1, 2, 3, \ldots, \mu$. Let V represent the interleaved code $(n = n_1 + n_2 + \ldots + n_\mu,\ k = k_1 + k_2 + \ldots + k_\mu)$.

If V is used for data-compression, then we get

$$D = \frac{N_1 + 2N_2 + \ldots + \mu N_\mu}{2^{n-k}\ (n_1 + n_2 + \ldots + n_\mu)} \qquad (3.33)$$

where D is the average distortion per bit. The distortion per bit D_i, given by V_i is simply

$$D_i = \frac{n_i}{2^{m_i} n_i} = \frac{1}{n_i + 1} . \tag{3.34}$$

Now, using (3.34), let us consider the sum

$$\sum_{i=1}^{\mu} D_i \frac{n_i}{n_1 + n_2 + \ldots + n_\mu}$$

$$= \frac{B_1 + B_2 + \ldots + B_\mu}{(n_1 + 1)(n_2 + 1) \ldots (n_\mu + 1)(n_1 + n_2 + \ldots + n_\mu)}$$

where

$$B_j = \frac{n_j (n_1 + 1)(n_2 + 1) \ldots (n_\mu + 1)}{(n_j + 1)} \tag{3.35}$$

A careful examination of (3.35) shows that $B_1 + B_2 + \ldots + B_\mu$ is indeed equal to the numerator in (3.33) so that we have the following

Theorem 3.21

In regard to data-compression, V gives

$$D = \frac{n_1}{n_1+n_2+\ldots+n_\mu} D_1 + \frac{n_2}{n_1+n_2+\ldots+n_\mu} D_2 + \ldots + \frac{n_\mu}{n_1+n_2+\ldots n_\mu} D_\mu .$$

As a special case of Theorem 3.21, we see that if a given Hamming code is interleaved to degree μ, then the interleaved code has the same average distortion per bit as the parent code.

Defining D_{max} to be Max $\{D_1, D_2, \ldots, D_\mu\}$ we see, from Theorem 3.21, that $D' \leq D_{max}$, implying that interleaving a short Hamming code with a longer code cannot increase the average distortion per bit.

3.5.3 Interleaved Golay Code

In this section we examine the implication of the fact that the (23, 12, 3) Golay code is perfect with regards to the average distortion obtained when the code is interleaved to degree μ.

Let the coset enumerator of the Golay code be represented as

$$1 + N_1 C + N_2 C^2 + N_3 C^3 \qquad (3.36)$$

where $N_1 = \binom{23}{1}$, $N_2 = \binom{23}{2}$ and $N_3 = \binom{23}{3}$;

The average distortion per bit D_G is given by

$$D_G = \frac{\sum_{i=0}^{3} i\, N_i}{2^{n-k}\, n} = \frac{0\cdot 1 + 1\cdot\binom{23}{1} + 2\cdot\binom{23}{2} + 3\cdot\binom{23}{3}}{2^{11}\cdot 23} = .124023$$

Let us now consider G_μ and $G_{\mu+1}$, the Golay codes interleaved to degree μ and $\mu + 1$ respectively.

For G_μ and $G_{\mu+1}$, the coset enumerators are given by

$$(1+N_1C+N_2C^2+N_3C^3)^\mu = \sum_{i+j+k+l=\mu} \binom{\mu}{i,j,k,l} (1)^i (N_1)^j (N_2)^k (N_3)^l C^{j+2k+3l} \tag{3.37}$$

$$(1+N_1C+N_2C^2+N_3C^3)^{\mu+1} = \sum_{i'+j'+k'+l'=\mu+1} \binom{\mu+1}{i',j',k',l'} (1)^{i'} (N_1)^{j'} (N_2)^{k'} (N_3)^{l'} C^{j'+2k'+3l'} \tag{3.38}$$

Now consider the ratio

$$\frac{D_{G_\mu}}{D_{G_{\mu+1}}} = \frac{\text{Average Distortion per bit for } G_\mu}{\text{Average Distortion per bit for } G_{\mu+1}}$$

which can be written as

$$\frac{D'_{G_\mu}}{D_{G_{\mu+1}}} = \frac{B_\mu / 2^{\mu(n-k)}}{B_{\mu+1}/2^{(\mu+1)(n-k)}} \, {}_{\mu n}{}_{(\mu+1)n} \tag{3.39}$$

where

$$B_\mu = \sum_{i+j+k+l=\mu} (j+2k+3l) \binom{\mu}{i,j,k,l} (1)^i (N_1)^j (N_2)^k (N_3)^l \tag{3.40}$$

and

$$B_{\mu+1} = \sum_{i'+j'+k'+l'=\mu+1} (j'+2k'+3l') \binom{\mu+1}{i',j',k',l'} (1)^{i'} (N_1)^{j'} (N_2)^{k'} (N_3)^{l'} \tag{3.41}$$

It is easy to show, using elementary combinatorial theory that

$$\frac{B_{\mu+1}}{B_\mu} = \frac{\mu+1}{\mu} (1 + N_1 + N_2 + N_3) \tag{3.42}$$

Therefore, using (3.36) in (3.42) we immediately see that $\frac{D_{G_\mu}}{D_{G_{\mu+1}}} = 1$, which proves the following

Theorem 3.22

The average distortion per bit of the Golay code interleaved to degree μ is a constant independent of the degree of interleaving.

3.5.4 Interleaved Hamming and Golay Codes

Let W represent the $(n=2^m-1+23=2^m+22, k=n-m+12)$ binary code obtained by interleaving the $(2^m-1,\ 2^m-1-m)$ Hamming code H andthe (23, 12) Golay code G.

The coset enumerator of W is

$$[1+(2^m-1)C][1+\binom{23}{1}C+\binom{23}{2}C^2+\binom{23}{3}C^3]$$

$$= 1+\left[2^m-1+\binom{23}{1}\right]C+\left[\binom{23}{1}(2^m-1)+\binom{23}{2}\right]C^2+$$

$$+\left[\binom{23}{2}(2^m-1)+\binom{23}{3}\right]C^3+\left[\binom{23}{3}(2^m-1)\right]C^4$$

$$= 1+(2^m+22)C+23(2^m+10)C^2+253(2^m+6)C^3+1771(2^m-1)C^4 \qquad (3.43)$$

Using (2.2) and (3.43), we can then compute the average distortion per bit of the interleaved code W as

$$D_W = \frac{0x1+1x(2^m+22)+2x23(2^m+10)+3x253(2^m+6)+4x1771(2^m-1)}{2^{m-12}.(2^m+22)}$$

which, can also be written in terms of D_G and D_H, the average distortion per bit of the Golay and Hamming codes respectively, as follows

$$D_W = \frac{2^m-1}{2^m+22} D_H + \frac{23}{2^m+22} D_G \qquad (3.44)$$

after simple manipulation

From (3.44) we see that

$$D_W \leq \text{Max } \{D_H, D_G\}.$$

CHAPTER IV

ERROR-CONTROL DECODING METHODS FOR SOURCE ENCODING

The preceding chapter examined the theoretical performance of several classes of block codes from the data compression viewpoint. In this chapter, the practical aspects of using some of the well-known error-control decoding algorithms are examined. The purpose here is to discuss the instrumentability of the source encoder which would implement any of the algorithms that may be used in conjunction with the block code chosen. In this context, we discuss the merits and disadvantages of several decoding algorithms, namely the standard array , the BCH, the majority logic, the step-by-step and the error-trapping decoding algorithms. The latter algorithm is singled out as being easy to implement and is therefore studied in greater detail. Its performance is analysed and compared against the published results related to the step-by-step algorithm. We then discuss some bounds on the average distortion achievable. In concluding this chapter, we present an extension of the error-trapping technique in which two codes of same length and different rates can be used for source encoding.

4.1 A Comparison Between Decoding Algorithms

We have shown, in Chapter II, that the optimum source

encoding scheme for the binary discrete memoryless source is analogous to the well-known standard array decoding algorithm used for error-control purposes.

One way of implementing this algorithm is to store an encoding table that lists each syndrome and its corresponding coset leader. To encode a source sequence one needs merely compute its syndrome, which requires only $n(n - k)$ binary additions, and then form a codeword by adding to the source sequence the corresponding coset leader found in the table. For a large code, however, this task becomes formidable and the encoding table, which occupies $2^{n-k}(2n - k)$ bits of memory, grows exponentially with the block-length n for any given fixed rate. This source encoding scheme, obviously, is not instrumentable.

Recalling that the group property of the source code was the only necessary property in the formation of the standard array, we must therefore consider some suboptimal schemes which make use of more of the code properties.

Intuitively, one such approach would be to use a decoding procedure relevant to the type of source code. Likely choices of easily implemented algorithms are the BCH decoding and the majority logic decoding algorithms

briefly discussed below.

4.11 BCH Decoding Algorithm

Details of the BCH decoding algorithms and its refinements are not given here as they can be found in several textbooks such as [5], [6] and [7]. For the purpose of our discussion, it is sufficient to know that this algorithm is a bounded-distance decoding algorithm that will guarantee, if used for source encoding, that the source encoder will perform acceptably for the source words that lie in cosets of weight $\leq$ t, where t is the error-correcting capability of the code. The difficulty will arise with those source words which lie in higher weight cosets. In this case, the source encoder will normally fail to produce a codeword, as the error-locator polynomial will not have all of its roots in the Galois field in which the operations take place. Even with a great deal of extra logic, such as Hartmann's [58] procedure to decode beyond the BCH bound, this situation can be overcome only partially. This is why both the knowledge of the covering radius of the code, and the existence of a complete decoding algorithm for certain BCH codes, are the deciding factors in the use of the BCH decoding algorithm for source encoding.

In this context, only the single-, double-, and triple-error-correcting BCH codes may be used for data compression. A source encoder implementing their complete decoding algorithm would, in this case, also be optimum.

It is interesting to note here that the (23, 12, 3) binary Golay code can also be considered as a non-primitive BCH code. The generator polynomial of the Golay code is either

$$g_1(X) = 1 + X^2 + X^4 + X^5 + X^6 + X^{10} + X^{11}$$

or $$g_2(X) = 1 + X + X^5 + X^6 + X^7 + X^9 + X^{11}$$

where both $g_1(X)$ and $g_2(X)$ are factors of $X^{23} + 1$. The minimum distance of the code is known to be 7. However, its designed distance, according to the BCH bound, is only 5. This is easily explained as follows. Let α be a primitive element of $GF(2^{11})$ and let $\beta = \alpha^{89}$. Since $2^{11} - 1 = 89 \times 23$, then $\beta^{23} = 1$. In this context, $g_1(X)$ and $g_2(X)$ can be shown to have only β, β^2, β^3, and β^4 as consecutive roots. So, according to the BCH bound, the minimum distance is at least 5, meaning that the BCH decoding algorithm will only be able to correct at most 2 errors.

In the following we present a decoding procedure which is almost the same as the decoding of any binary non primitive BCH code with designed minimum distance 7.

Let V(X) be a codeword belonging to the (23, 12) binary Golay code.

For the given received polynomial

$$S(X) = V(X) + E(X) \tag{4.1}$$

where E(X) is the error of weight 3 at most, let us compute the syndrome

$$S_1 = S(\beta),\ S_3 = S(\beta^3).$$

Since both β and β^3 are roots of the generator polynomial, then if $S_1 = 0$, then $S_3 = 0$. Thus we have

$$\{S_1 = 0\} \leftrightarrow \{E(X) = 0\} \tag{4.2}$$

As usual we also have

$$\{S_1 \neq 0,\ S_3 = S_1^3 = \beta^{3j}\} \leftrightarrow \{E(X) = X^j\} \tag{4.3}$$

Thus, the situations of $|E(X)| = 0$ (no-error) and $|E(X)| = 1$ (single-error) are taken care of by (4.2) and (4.3). Suppose now that we find $|E(X)|$ to be neither 0 nor 1. Then we form, as usual, the quadratic

$$X^2 + S_1 X + \frac{S_3 + S_1^3}{S_1} = 0 \qquad (4.4)$$

Since we are dealing with a non primitive BCH code, there are three possibilities that may arise.

(a) $\mathrm{Tr}\left[\frac{S_3 + S_1^3}{S_1^3}\right]$ is not equal to $\tilde{0}$, in which case the roots of (4.4) do not belong to $GF(2^{11})$.

(b) $\mathrm{Tr}\left[\frac{S_3 + S_1^3}{S_1^3}\right] = 0$ but one or both roots of (4.4) do not have the form $(\alpha^{89})^j = \beta^j$

(c) $\mathrm{Tr}\left[\frac{S_3 + S_1^3}{S_1^3}\right] = 0$ and the roots are $(\alpha^{89})^a = \beta^a$ and $(\alpha^{89})^b = \beta^b$, in which case we conclude that $E(X) = X^a + X^b$.

So, if (a) or (b) occurs we must assume that $|E(X)| = 3$.

Using Newton's identity

$$S_3 + \sigma_1 S_2 + \sigma_2 S_1 + \sigma_3 = 0 \qquad (4.5)$$

with $\sigma_1 = S_1$ and $S_2 = S_1^2$, we form the cubic

$$\sigma(x) = x^3 + S_1 x^2 + \sigma_2 x + (S_3 + S_1^3 + S_1\sigma_2) = 0 \qquad (4.6)$$

For the given S_1 and S_3, the cubic (4.6) has 3 roots of the form β^j only for one value of σ_2, since the code V has d = 7. Suppose that β^p, β^q and β^r are these roots. Then, from (4.6), we have

$$\sigma_2 = \frac{\beta^{3u} + S_1\beta^{2u} + S_3 + S_1^3}{S_1 + \beta^u}, \quad u = p, q, r \qquad (4.7)$$

An implication of (4.7) is as follows: If we compute

$$\phi(\beta^v) \triangleq \frac{\beta^{3v} + S_1\beta^{2v} + S_3 + S_1^3}{S_1 + \beta^v} \qquad (4.8)$$

for v = 0, 1, 2, ..., 22, then there are only 3 values of v for which $\phi(\beta^v)$ turns out to be the same. If these values are v_1, v_2, v_3, then

$$E(X) = X^{v_1} + X^{v_2} + X^{v_3} \tag{4.9}$$

From the preceding discussion it is clear that finding the errors in a 3-error situation in the binary Golay code is almost the same as finding the roots of a cubic in which all of the coefficients σ_i are known. Furthermore, the search of (4.8) is guaranteed to give a solution since all 3-error patterns are coset leaders. This search can be simplified if the idea underlying the Chien-search [27] is applied to the numerator and denominator separately.

4.1.2 Majority Logic Decoding Algorithm

There are several ways of implementing the concept of majority logic decoding [59] to combat channel errors. We can implement the decoder to estimate the error positions in the received sequence. The position in error is then complemented. We can also implement the decoder to recover the n-bits of the transmitted codeword. Finally, we can choose only to extract the K information bits from the received n-tuple. Correct results are guaranteed by all three methods as long as the number of errors does not exceed the error-correcting capability of the code used. Certain error patterns of weight t + 1 can also be corrected if feedback is used or with a second round of estimation

[60]. For higher weight error patterns, there is no guarantee about the validity of the results. The decoder can be made to indicate the occurrence of an uncorrectable error, or simply to ignore such occurrence in which case the result can be accepted as being correct.

In summary, the majority-logic decoding procedure can be thought of as a complete decoding algorithm since the decoder will produce as many results as there are syndromes. The results corresponding to syndromes of error-patterns of weight greater than t will not necessarily be the correct ones.

This "complete decoding" feature, however, is also the reason why majority-logic decoding can be examined in the context of source encoding. Since in source encoding we want to compress an n-bit source sequence into a k-tuple, then the source encoder will be designed to extract k bits only out of the n-tuple, regardless of where in the cosets that n-tuple lies. At the user end, multiplication of the k-bit sequence by g(X) naturally yields a codeword which may, or may not be the true reproduction of the source sequence. Depending on the situation, the distortion introduced in the process can have a value between zero and n - k at most.

The main disadvantage of majority-logic for source encoding, therefore, is the total lack of control over the codeword to be used to represent the source sequence. The second disadvantage we see is that majority-logic is applicable to very few classes of codes, thus severely limiting the choice of an appropriate source code.

Given that the intuitive approach of chosing a source encoding procedure relevant to the type of code does not appear to be the best method, we therefore have to try a different approach.

One such suboptimum encoding scheme has been proposed and analysed by Goblick [op. cit. 11]. This scheme is analogous to the step-by-step channel decoding procedure [op. cit. 5] and is therefore generally applicable to any source code. The algorithm begins by matching the first k digits of the source sequence with a mod 2 sum of appropriate rows of the (k x n) generator matrix [G] of the source code. This process is greatly simplified if [G] is in a systematic form $[I_k P]$, where I_k is the (k x k) identity matrix. Next, row 1 is added to this sum if and only if this reduces the distortion, row 2 is added if and only if this further reduces it, and so on. This process could be iterated through the rows of [G] more

than once, but the law of diminishing returns will soon prevail. During the initial matching process, the distortion is confined within the last (n - k) digits of the source sequence, but starts to spread across the entire n-tuple with the further addition of rows. The maximum distortion that we can expect is obviously n - k.

Goblick has calculated the average distortion achieved by this encoding scheme using a single pass through [G] for ensembles of group codes with randomly chosen generators of various blocklengths. The results for n = 20 are shown in Figure 7. The optimum D is 0.11 at R = 0.5 bits per letter, whereas the curve guarantees that group codes with one-pass, step-by-step encoding exist that achieve $D \leq 0.185$. The curve does not improve substantially for n > 20. In fact, it deteriorates for n > 60, which is clearly a divergence from the results promised by the source coding theorem (Result 2). This deterioration may have probably been lessened with more than one pass. However, any improvement would naturally require more computations and it is doubtful that the improvement will be significant because of the inherent randomness of the step-by-step encoding algorithm.

In this thesis we propose the use, for source encoding,

the well-known error-trapping technique [61]. We assert that it will perform better than the step-by-step and majority-logic algorithms, because of the methodical way in which it operates. It is also attractive because it can be used with a large ensemble of codes. In this connection, we recall that a necessary and sufficient condition for all patterns of t or fewer random errors to be correctable, with an (n, k, t) code, by error-trapping is $t \leq \frac{n}{k}$ [5, p. 238].

A detailed description of the error-trapping technique is given in the next section. We then describe the source coding algorithm based on this technique and compare its performance against step-by-step and majority-logic-decoding.

4.2 Description of the Error-Trapping Decoding Algorithm

The following description is adapted from Shu Lin [6]. Let V(x) be the transmitted codeword of a t-error-correcting (n, k) cyclic code V. Let the received vector be

$$S(x) = V(x) + E(x), \qquad (4.10)$$

where E(x) is the error pattern caused by the channel

disturbance. The syndrome $\rho(x)$ of $S(x)$ is equal to the remainder resulting from dividing the received vector $S(x)$ by the code generator polynomial $g(x)$, i.e.,

$$S(x) = p(x)g(x) + \rho(x) \tag{4.11}$$

where $\rho(x)$ is a polynomial of degree $n - k - 1$ or less. Since $V(x)$ is a code polynomial, it must be a multiple of the generator polynomial $g(x)$, say

$$V(x) = m(x)g(x) \tag{4.12}$$

Combining Eqs (4.10), (4.11) and (4.12) we obtain

$$\begin{aligned} E(x) &= [p(x) + m(x)]g(x) + \rho(x) \\ &= q(x)g(x) + \rho(x), \end{aligned} \tag{4.13}$$

that is, the syndrome of $S(x)$ is also equal to the remainder resulting from dividing the error pattern by the generator polynomial $g(x)$ of the code.

If the errors of $E(x)$ are confined to the $n - k$ parity check positions $1, X, \ldots, X^{n-k-1}$ of $S(x)$, then $E(x)$ is a polynomial of degree $n - k - 1$ or less. It then follows from Eq. (4.13) that $q(x) = 0$ and

$E(x) = \rho(x)$. That is, if the errors in $S(x)$ are confined to the $n - k$ parity positions, then the syndrome of $S(x)$ is identical to the error vector $\rho(x) = E(x)$. Thus, correction can be accomplished simply by adding (modulo 2) the syndrome to the $n - k$ received parity check digits.

Suppose that the errors are not confined to the $n - k$ parity check positions of $S(x)$ but are confined to $n - k$ consecutive positions (including the end around case), say $x^i, x^{i+1}, \ldots, x^{(n-k)+i-1}$. After $n - i$ cyclic shifts of $S(x)$, the errors will be shifted to the $(n - k)$ parity check positions of the cyclically shifted received vector $S^{(n-i)}(x)$. Then the syndrome of $S^{(n-i)}(x)$ is identical to the errors confined to the positions $x^i, x^{i+1}, \ldots, x^{(n-k)+i-1}$ of $S(x)$. As a result, the errors can be corrected.

An error-trapping decoder is shown in Fig. 4. Its operation can be described in the following steps:

STEP 1

Gate 1 is ON; Gates 2 and 3 are OFF. The received vector is read into the syndrome register and simultaneously into the buffer register (since we are

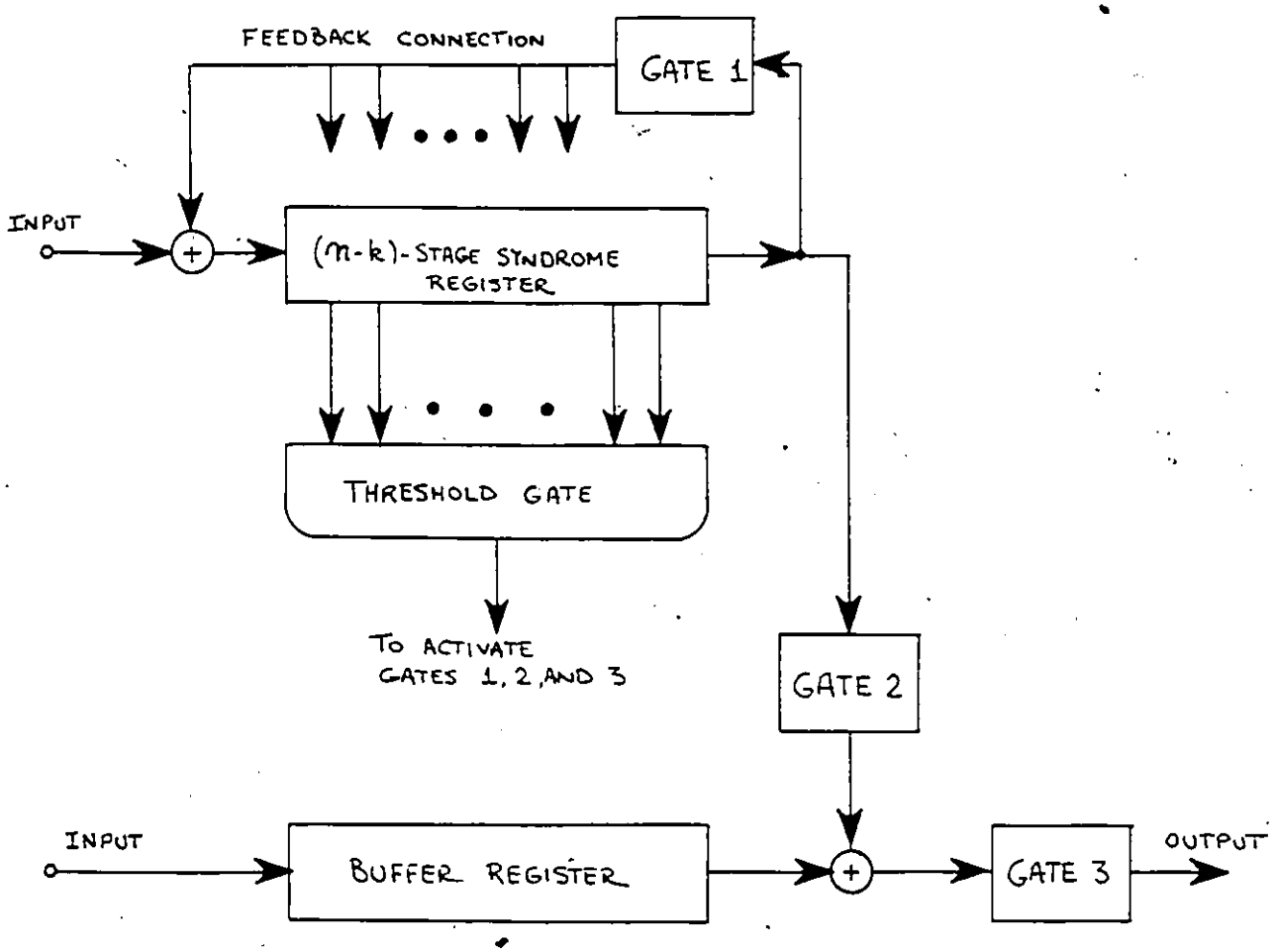


Figure 4-An Error-Trapping Encoder

only interested in the recovery of the k transmitted information digits, the buffer register has only to store the k received information digits). As soon as the entire received vector has been shifted into the syndrome register, the contents of the register are the syndrome of the received vector.

STEP 2

The weight of the syndrome is tested by an (n - k)-input threshold gate. The output of this gate is "1" when t or fewer of its inputs are "1"; otherwise; the output is zero.

STEP 3

(a) If the weight of the syndrome is t or less, the errors are confined to the (n - k) parity check positions X^0, X^1, ..., X^{n-k-1} of the received vector. Thus, the k received information digits in the buffer register are error-free. Gate 3 is then turned ON and the information digits are sent to the data sink (Gate 2 is OFF). The decoding is completed. Return to STEP 1.

(b) If the weight of the syndrome calculated in the first step is greater than t, the syndrome register is then shifted once with Gate 1 ON and Gates 2 and 3 OFF. Go to STEP 4.

STEP 4

The weight of the new contents of the syndrome register is tested.

(a) If the weight is t or less, the errors are confined to the positions x^{n-1}, x^0, x^1, ..., x^{n-k-2} of the received vector. The leftmost digit in the syndrome register matches the error at the position x^{n-1} of the received vector; the other n - k - 1 digits in the syndrome register match the errors at positions x^0, x^1, ..., x^{n-k-2} of the received vector. The output of the threshold gate turns Gate 1 OFF and sets a clock to count from 2. The syndrome register is then shifted (in step with the clock) with Gate 1 turned OFF. As soon as the clock has counted to n - k, the contents of the syndrome register will be (0, 0, ..., 0, 1). The rightmost digit matches the error at the

position X^{n-1} of the received vector. The k received information digits are then read out of the buffer. The first received information digit is corrected by the "1" coming out of the syndrome register. The decoding is thus completed. Return to STEP 1.

(b) If the weight of the contents of the syndrome register (from STEP 3b) is greater than t, the syndrome register is shifted once again with Gate 1 OFF and Gates 2 and 3 OFF. Go to STEP 5.

STEP 5

STEP 4b repeats until the weight of the contents of the syndrome register goes down to t or less. If the weight goes down to t or less after the i^{th} shift, for $1 \leq i \leq n - k$, the clock starts to count from i + 1. At the same time, the syndrome register is shifted with Gate 1 OFF. As soon as the clock has counted to n - k, the rightmost i digits in the syndrome register match the errors in the first i received information digits in the buffer register. The other information digits are error-free. Gates 2 and 3 are turned ON. The

received information digits are read out of the buffer for correction. Return to STEP 1.

STEP 6

If the weight of the contents of the syndrome register never goes down to t or less by the time that the syndrome register has been shifted n - k times (with Gate 1 ON), Gate 3 is then turned ON and the received information digits are read out of the buffer one at a time. At the same time the syndrome register is shifted with Gate 1 ON. As soon as the weight of the contents of the syndrome register goes down to t or less, the contents match the errors in the next n - k digits to come out of the buffer. Gate 2 is then turned ON and the erroneous information digits are corrected by the digits coming out from the syndrome register with Gate 1 OFF. Gate 3 is turned OFF as soon as k information digits have been read out of the buffer.

If the weight of the contents of the syndrome register never goes down to t or less by the time k received information digits have been read out of the buffer, then either an uncorrectable error pattern has occurred or a correctable error pattern with errors not confined to n - k

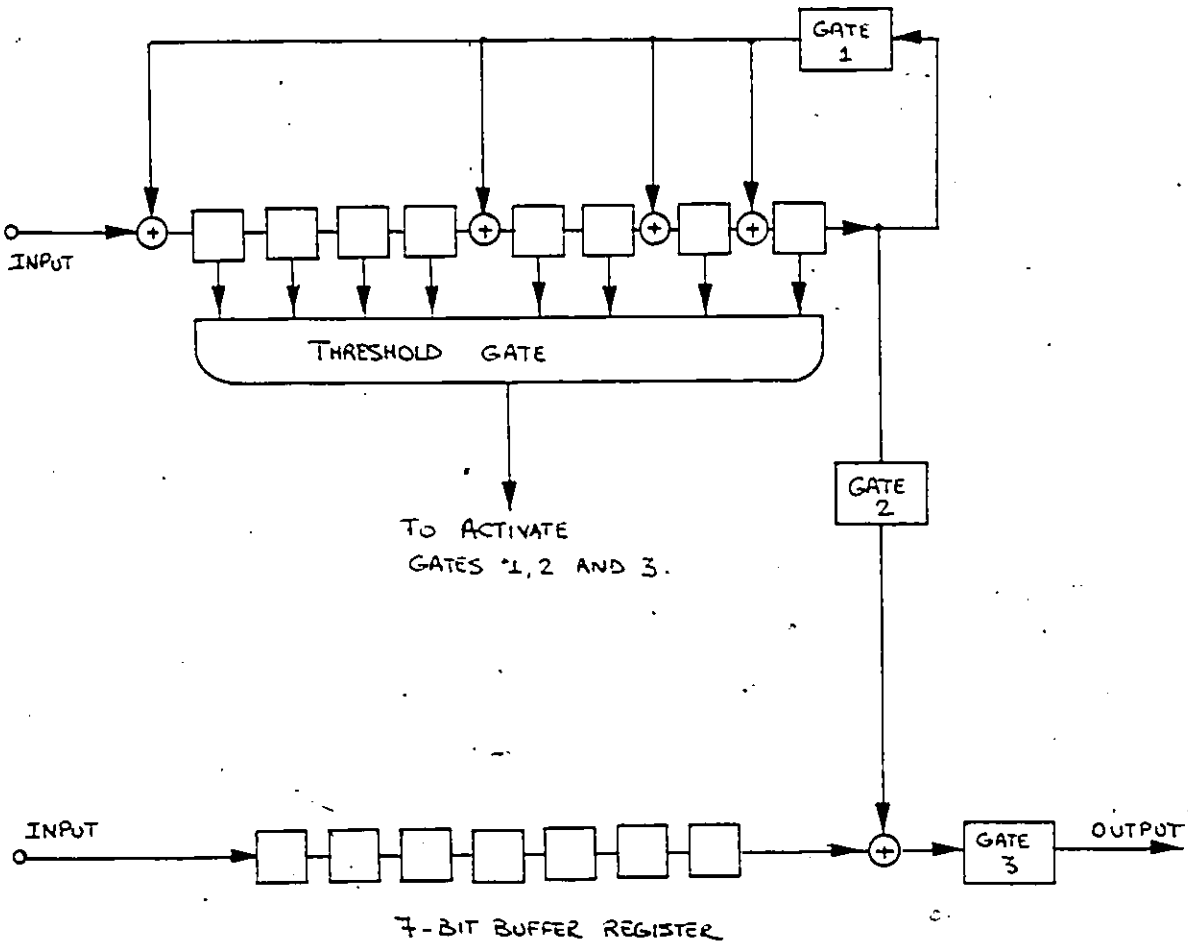


Figure 5 - Error-Trapping Decoder for the (15,7,2) BCH Code

consecutive positions has occurred.

The operation of the error-trapping decoder can best be described with the following example. Let us assume that the code V is the (15, 7, 2) BCH code with generator polynomial $g(x) = 1 + x^4 + x^6 + x^7 + x^8$. The syndrome register is an 8-stage register while the buffer register needs only store 7 digits, as shown in Figure 5. For the received vector $S(x) = 1 + x^2 + x^5 + x^6 + x^7 + x^8 + x^{10} + x^{12} + x^{13}$, the successive contents of the syndrome register, as well as the states of Gates 1, 2 and 3 are shown in Table 2. At the end of the decoding process, the corrected information digits are 1010101.

4.3 Description of the Error Trapping Source Encoding Algorithm

The source encoding algorithm described here is essentially similar to the error trapping decoding scheme surveyed in Sec. 4.2. Slight modifications are, however, required. The buffer register must be able to store the n digits and the threshold gate is replaced by a weight counter.

The main difference in the mode of operation arises

Table 2

The decoding operation of the error-trapping decoder for the (15,7,2) BCH Code

L	Gate Status After L^{th} shift 1	2	3	Syndrome Register Contents after L^{th} shift	Threshold Gate Status after L^{th} shift	Output Symbol
0	ON	OFF	OFF			
1				0		
2				1 0		
3				1 1 0		
4				0 1 1 0		
5				1 0 1 1 0		
6				0 1 0 1 1 0		
7				1 0 1 0 1 1 0		
8				1 1 0 1 0 1 1 0		
9				1 1 1 0 1 0 1 1		
10				0 1 1 1 1 1 1 0		
11				0 0 1 1 1 1 1 1		
12				1 0 0 1 0 1 0 0		
13				1 1 0 0 1 0 1 0		
14				0 1 1 0 0 1 0 1		
15	ON	OFF	OFF	0 0 1 1 1 0 0 1	0	

Table 2 (cont'd)

L	Gate Status After L^{th} shift 1	2	3	Syndrome Register Contents after L^{th} shift	Threshold Gate Status after L^{th} shift	Output Symbol
16	ON	OFF	OFF	1 0 0 1 0 1 1 1	0	
17	ON			1 1 0 0 0 0 0 0	1	
18	OFF			0 1 1 0 0 0 0 0		
19				0 0 1 1 0 0 0 0		
20				0 0 0 1 1 0 0 0		
21				0 0 0 0 1 1 0 0		
22				0 0 0 0 0 1 1 0		
23		OFF	OFF	0 0 0 0 0 0 1 1		
24		ON	ON	0 0 0 0 0 0 0 1		1
25				0 0 0 0 0 0 0 0		0
26				0 0 0 0 0 0 0 0		1
27				0 0 0 0 0 0 0 0		0
28				0 0 0 0 0 0 0 0		1
29				0 0 0 0 0 0 0 0		0
30	OFF	ON	ON	0 0 0 0 0 0 0 0	1	1

from the need to interpret the output result differently. With the channel decoder algorithm, the input is the transmitted codeword, corrupted with a certain number of channel errors. In its attempts to correct these errors the decoder will succeed if their number is less than or equal to the error-correcting capability t of the code used. For a larger number of errors, the decoder will either fail or decode incorrectly. It will also fail if the errors ($e \leq t$) are not confined to $(n - k)$ consecutive positions. On the other hand the task of the source encoder is to map the input n-bit sequence $S(x)$ into a k-bit sequence $I(x)$ such that $I(x)G(x) = V(x)$ is as close as possible (in the Hamming sense) to $S(x)$. That is to say, the encoder must always produce a result for any given input sequence $S(x)$. The amount of distortion introduced by the mapping process will depend on the code used and the encoding method. When an appropriate code is chosen, the error-trapping source encoder will attempt to minimize the overall average distortion in the following way.

START: Given the n-bit source block $S(x)$.

STEP 1: Set a counter $i = 0$.

STEP 2: Compute the remainder $\rho_i(x)$ from the equation

$$\rho(x) = x^i S(x) \bmod G(x).$$

STEP 3: Compute the weight ω_i of $\rho_i(x)$.

If $\omega_i = 0$ then go to STEP 7.

STEP 4: Store i, ω_i and $\rho_i(x)$.

STEP 5: Increment i by 1.

If $i \leq n - 1$ then go to STEP 2,

else go to STEP 6.

STEP 6: Determine the minimum weight ω_a of $\rho_i(x)$ for $i = 0$ to $n - 1$.

STEP 7: Form the codeword $V_\Delta(x) = S(x) + x^{-a}\rho_a(x)$, then divide by $G(x)$ to obtain the compressed k-bit block $I_\Delta(x)$. Transmit $I_\Delta(x)$. Go to START.

A flowchart of this source encoding algorithm Δ is shown in Figure 6.

4.4 Analysis of the Algorithm

4.4.1 Maximum Distortion

It is easy to prove that $V_\Delta(x) = S(x) + x^{-a}\rho_a(x)$ is a codeword. Since $x^i S(x)$ can be expressed, by the Euclidean division algorithm, as $M(x)g(x) + \rho_i(x)$, where $M(x)$ is a polynomial of degree $k - 1$ or less, then $\rho_a(x)$ can be written as

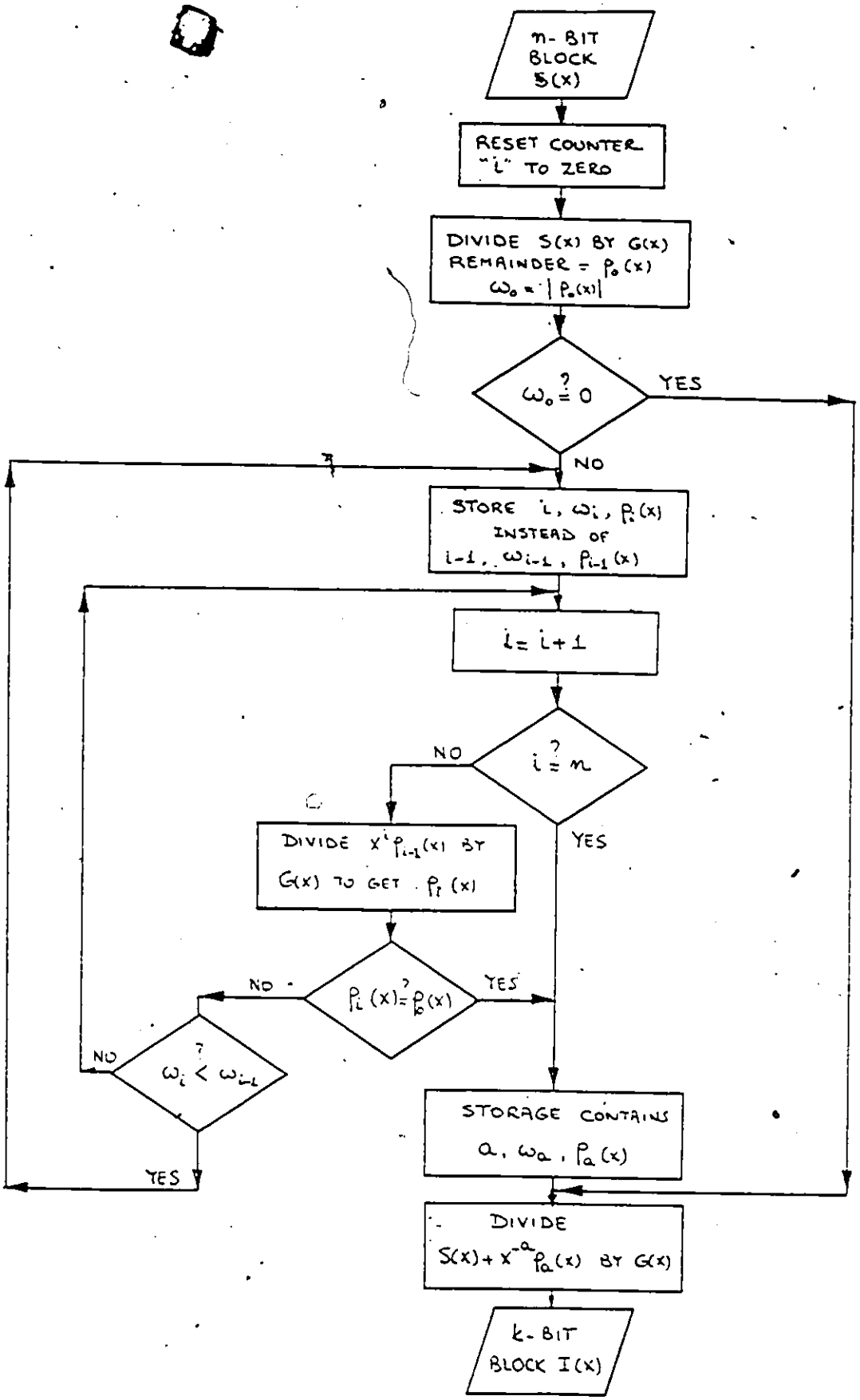


Figure 6 - Flow Diagram of Source Encoder Δ

$$\rho_a(x) = x^a S(x) + M(x)\, g(x). \qquad (4.14)$$

Hence

$$V_\Delta(x) = S(x) + x^{-a}\rho a(x) = x^{-a} M(x)\, g(x), \qquad (4.15)$$

therefore showing that $V_\Delta(x)$ is a multiple of $g(x)$, that is, a codeword belonging to the (n, k) code V with generator polynomial $g(x)$.

Equation (4.15) also shows that a simplification of STEP 7 of the encoding algorithm is possible. Note from Eq. (4.14) that M(x) is the quotient of $\rho_a(x)$ divided by $g(x)$. So, instead of forming the codeword $V_\Delta(x)$ then dividing by $g(x)$ in order to get $I_\Delta(x)$, the compressed k-bit block $I_\Delta(x)$ can be simply obtained by cyclically shifting M(x) "a" positions to the left.

The distance between $V_\Delta(x)$ and S(x) is evidently given by

$$|\rho_a(x)| = \omega_a = \text{Min}\ \{|\rho_0(x)|, |\rho_1(x)|, \ldots, |\rho_{n-1}(x)|\} \qquad (4.16)$$

where, by convention, $|\rho_i(x)|$ is the Hamming weight ω_i of the vector $\rho_i(x)$.

Noting that the degree of $\rho_i(x)$ must be n - k - 1 or less, it is therefore clear that the distortion introduced by the source encoder is confined to (n - k) consecutive positions of the n-bit source block. Hence the maximum distortion cannot, by any means, exceed n - k.

The average distortion per bit D_Δ is derived next.

4.4.2 Average Distortion per Bit D_Δ

The expression for the average distortion per bit D_Δ, resulting from the source encoding algorithm Δ, can be derived as follows. The number of remainders of degree $\leq$ (n - k - 1) is 2^{n-k}. With the assumption that the source emits any n-bit sequence with the likelihood $1/2^n$, then it can also be assumed that the remainders are all equally likely to occur.

Let the set of distinct remainders of $x^0\rho_e(x)$, $x\rho_e(x)$, $x^2\rho_e(x)$, ..., $x^{n-1}\rho_e(x)$ be defined as the remainder class σ_e. The set of 2^{n-k} the remainder classes $\sigma_0, \sigma_1, \sigma_2, \ldots \sigma_e, \ldots, \sigma_u$ for which μ_e is the minimum remainder weight in the class σ_e, that is, $\mu_e = |\rho_e(x)|$ and υ_e is the number of distinct

remainders in σ_e. Noting that $\sum_{e=0}^{u} \upsilon_e = 2^n - k$, the average distortion per bit is given by

$$D_\Delta = \frac{\sum_{e=0}^{u} \upsilon_e \mu_e}{(2^n - k)\, n} \tag{4.17}$$

In Table 3 we list the remainder classes of the (15, 7, 2) BCH code with generator polynomial $g(x) = 1 + x^4 + x^6 + x^7 + x^8$. In this table, we use the following notation for simplicity. A remainder $\rho_i(x) = x^a + x^b + x^c$ in any class σ_j is listed as (a, b, c). To form a remainder class σ_j, we start with a remainder $p_0(x)$ of degree less than $n - k$ that has not occurred before, then successively compute $x^i \rho_i(x)$ mod $g(x)$ for $i = 1$ to $n - 1$. If n is not a prime, then some of the remainder classes may have periods equal to n/e, where e is a factor of n.

To illustrate the use of Table 3 in computing the average distortion D_Δ given by (4.17), we have

$$\mu_0 = 0,\ \upsilon_0 = 0$$

$$\mu_1 = 2,\ \upsilon_1 = 15$$

$$\mu_2 = \mu_3 = \cdots \mu_8 = 3,\ \upsilon_2 = \upsilon_3 = \cdots = \upsilon_8 = 15;$$

$$\mu_9 = \mu_{10} = \cdots \mu_{16} = 3,\ \upsilon_9 = \upsilon_{10} = \cdots = \upsilon_{16} = 15;$$

TABLE 3-REMAINDER CLASSES OF THE (15, 7, 2) BCH CODE

σ_1	σ_2	σ_3
0	1, 0	2, 0
1	2, 1	3, 1
2	3, 2	4, 2
3	4, 3	5, 3
4	5, 4	6, 4
5	6, 5	7, 5
6	7, 6	7, 4, 0
7	6, 4, 0	7, 6, 5, 4, 1, 0
7, 6, 4, 0	7, 5, 1	5, 4, 2, 1, 0
6, 5, 4, 1, 0	7, 4, 2, 0	6, 5, 3, 2, 1
7, 6, 5, 2, 1	7, 6, 5, 4, 3, 1, 0	7, 6, 4, 3, 2
4, 3, 2, 0	5, 2, 1, 0	6, 5, 3, 0
5, 4, 3, 1	6, 3, 2, 1	7, 6, 4, 1
6, 5, 4, 2	7, 4, 3, 2	6, 5, 4, 2, 0
7, 6, 5, 3	7, 6, 5, 3, 0	7, 6, 5, 3, 1

σ_4	σ_5	σ_6
3, 0	4, 0	5, 0
4, 1	5, 1	6, 1
5, 2	6, 2	7, 2
6, 3	7, 3	7, 6, 4, 3, 0
7, 4	7, 6, 0	6, 5, 1, 0
7, 6, 5, 4, 0	6, 4, 1, 0	7, 6, 2, 1
5, 4, 1, 0	7, 5, 2, 1	6, 4, 3, 2, 0
6, 5, 2, 1	7, 4, 3, 2, 0	7, 5, 4, 3, 1
7, 6, 3, 2	7, 6, 5, 3, 1, 0	7, 5, 2, 0
6, 3, 0	2, 1, 0	7, 4, 3, 1, 0
7, 4, 1	3, 2, 1	7, 6, 5, 2, 1, 0
7, 6, 5, 4, 2, 0	4, 3, 2	4, 3, 2, 1, 0
5, 4, 3, 1, 0	5, 4, 3	5, 4, 3, 2, 0
6, 5, 4, 2, 1	6, 5, 4	6, 5, 4, 3, 2
7, 6, 5, 3, 2	7, 6, 5	7, 6, 5, 4, 3

TABLE 3 (CONT'D)

σ_7	σ_8	σ_9
6, 0	7, 0	3, 1, 0
7, 1	7, 6, 4, 1, 0	4, 2, 1
7, 6, 4, 2, 0	6, 5, 4, 2, 1, 0	5, 3, 2
6, 5, 4, 3, 1, 0	7, 6, 5, 3, 2, 1	6, 4, 3
7, 6, 5, 4, 2, 1	3, 2, 0	7, 5, 4
5, 4, 3, 2, 0	4, 3, 1	7, 5, 4, 0
6, 5, 4, 3, 1	5, 4, 2	7, 5, 4, 1, 0
7, 6, 5, 4, 2	6, 5, 3	7, 5, 4, 2, 1, 0
5, 4, 3, 0	7, 6, 4	7, 5, 4, 3, 2, 1, 0
6, 5, 4, 1	6, 5, 4, 0	7, 5, 3, 2, 1, 0
7, 6, 5, 2	7, 6, 5, 1	7, 3, 2, 1, 0
4, 3, 0	4, 2, 0	7, 6, 3, 2, 1, 0
5, 4, 1	5, 3, 1	6, 3, 2, 1, 0
6, 5, 2	6, 4, 2	7, 4, 3, 2, 1
7, 6, 3	7, 5, 3	7, 6, 5, 3, 2, 0

σ_{10}	σ_{11}	σ_{12}
4, 1, 0	5, 1, 0	7, 1, 0
5, 2, 1	6, 2, 1	7, 6, 4, 2, 1, 0
6, 3, 2	7, 3, 2	6, 5, 4, 3, 2, 1, 0
7, 4, 3	7, 6, 3, 0	7, 6, 5, 4, 3, 2, 1
7, 6, 5, 0	6, 1, 0	5, 3, 2, 0
	7, 2, 1	6, 4, 3, 1
	7, 6, 4, 3, 2, 0	7, 5, 4, 2
	6, 5, 3, 1, 0	7, 5, 4, 3, 0
	7, 6, 4, 2, 1	7, 5, 1, 0
	6, 5, 4, 3, 2, 0	7, 4, 2, 1, 0
	7, 6, 5, 4, 3, 1	7, 6, 5, 4, 3, 2, 1, 0
	5, 2, 0	5, 3, 2, 1, 0
	6, 3, 1	6, 4, 3, 2, 1
	7, 4, 2	7, 5, 4, 3, 2
	7, 6, 5, 4, 3, 0	7, 5, 3, 0

TABLE 3 (CONT'D)

σ_{13}

6, 2, 0
7, 3, 1
7, 6, 2, 0
6, 4, 3, 1, 0
7, 5, 4, 2, 1
7, 5, 4, 3, 2, 0
7, 5, 3, 1, 0
7, 2, 1, 0
7, 6, 4, 3, 2, 1, 0
6, 5, 3, 2, 1, 0
7, 6, 4, 3, 2, 1
6, 5, 3, 2, 0
7, 6, 4, 3, 1
6, 5, 2, 0
7, 6, 3, 1

σ_{14}

7, 2, 0
7, 6, 4, 3, 1, 0
6, 5, 2, 1, 0
7, 6, 3, 2, 1
6, 3, 2, 0
7, 4, 3, 1
7, 6, 5, 2, 0
4, 3, 1, 0
5, 4, 2, 1
6, 5, 3, 2
7, 6, 4, 3
6, 5, 0
7, 6, 2
6, 4, 2, 0
7, 5, 3, 1

σ_{15}

5, 3, 0
6, 4, 1
7, 5, 2
7, 4, 3, 0
7, 6, 5, 1, 0
4, 2, 1, 0
5, 3, 2, 1
6, 4, 3, 2
7, 5, 4, 3
7, 5, 0
7, 4, 1, 0
7, 6, 5, 4, 2, 1, 0
5, 4, 3, 2, 1, 0
6, 5, 3, 2, 1
7, 6, 5, 4, 3, 2

σ_{16}

7, 3, 0
7, 6, 1, 0
6, 4, 2, 1, 0
7, 5, 3, 2, 1
7, 3, 2, 0
7, 6, 3, 1, 0
6, 2, 1, 0
7, 3, 2, 1
7, 6, 3, 2, 0
6, 3, 1, 0
7, 4, 2, 1
7, 6, 5, 4, 3, 2, 0
5, 3, 1, 0
6, 4, 2, 1
7, 5, 3, 2

σ_{17}

5, 4, 0
6, 5, 1
7, 6, 2
6, 4, 3, 0
7, 5, 4, 1
7, 5, 4, 2, 0
7, 5, 4, 3, 1, 0
7, 5, 2, 1, 0
7, 4, 3, 2, 1, 0
7, 6, 5, 3, 2, 1, 0
3, 2, 1, 0
4, 3, 2, 1
5, 4, 3, 2
6, 5, 4, 3
7, 6, 5, 4

σ_{18}

7, 3, 1, 0
7, 6, 2, 1, 0
6, 4, 3, 2, [illegible], 0
7, 5, 4, 3, 2, 1
7, 5, 3, 2, 0

σ_{19}

5, 4, 2, 0
6, 5, 3, 1
7, 6, 4, 2
6, 5, 4, 3, 0
7, 6, 5, 4, 1

$$\upsilon_{17} = 4, \ \upsilon_{17} = 5;$$
$$\mu_{18} = \mu_{19} = 4, \ \upsilon_{18} = \upsilon_{19} = 5.$$

This example shows that there are 20 distinct remainder classes, i.e., u = 19. For the remainder class σ_7 (say), the minimum remainder weight μ_7 is 2 and the number of distinct remainders in that class is $\upsilon_7 = 15$. Note that υ_7 is also the period of the remainder class σ_7. From (4.17), the average distortion per bit D_Δ for the (15, 7, 2) BCH code is therefore

$$\frac{0 \cdot 0 + 15 \cdot 1 + 15 \cdot 7 \cdot 2 + 15 \cdot 8 \cdot 3 + 5 \cdot 3 + 5 \cdot 2 \cdot 4}{2^8 \cdot 15} = \frac{1}{6}$$

The average distortion D_Δ can also be expressed in terms of the weight distribution of the remainders, as follows. Let n_i denote the total number of remainders of weight i and let W represent the maximum remainder weight, that is, $W = \text{Max} \{|\rho_e(x)|, e = 0, 1, \ldots u\}$. Then D_Δ is given by

$$D_\Delta = \frac{\sum_{i=0}^{W} i \cdot n_i}{(2^n - k)n} \qquad (4.18)$$

For the example above, the weight distribution of

the remainders is

i	η_i
0	1
1	15
2	105
3	125
4	10

$$\text{and } D_\Delta' = \frac{0\cdot 1 + 1\cdot 15 + 2\cdot 105 + 3\cdot 125 + 4\cdot 10}{15 \times 256} = \frac{1}{6}$$

An important remark can be made by noting the similarity between the expression for

$$D_{AVG} = \frac{\sum_{i=0}^{\Omega} iN_i}{(2^n - k)n}$$

$$\text{and } D_\Delta = \frac{\sum_{i=0}^{W} i\eta_i}{(2^n - k)\, n}$$

given by (2.2) and (4.18) respectively. In the former Ω represented the covering radius of the code and N_i was the number of cosets of weight i. In the latter W is the maximum remainder weight and η_i denotes the total number of remainders of weight i, resulting from the use of

the error-trapping technique. This similarity suggests that, in the absence of any knowledge about the coset enumeration of an (n, k, t) code, W and n_i can be used as estimates for Ω and N_i respectively. Intuitively, the following relation

$$W \geq \Omega$$

would always be true.

4.4.3 Computer Analysis of a Selection of Codes

The performance of the error-trapping source encoding algorithm was analysed by a computer programme for several codes of moderate length, most of them listed in [62]. The programme is listed in Appendix A and the results are tabulated in Table 4.

The function of the programme is to simulate the operation of the source encoder for all possible 2^{n-k} $(n-k)$-tuples that may be produced by an (n, k) binary code. The programme uses a well tested algorithm [63, Ch.3] to generate methodically the $\binom{n-k}{\omega}$ $(n-k)$-tuples of weight ω for the successive values of ω from 1 to $n-k$. This method is described in greater detail in Appendix A.

For each (n - k)-tuple, the remainder class was generated and the minimum weight in that class stored. The final output of the programme[9] consisted of the weight distribution of the remainders as shown in Table 4. As the execution time was found to be of the order of $n \cdot 2^{n-k}$, the programme was ran for codes of moderate lengths ($n \leq 39$) and a small number of parity-checks ($n - k \leq 15$). In spite of these constraints, we were able to test about forty well-known codes, with rates spanning the range between .238 and .838.

For each of the codes analysed, we computed the distortion per bit D_Δ using the expression for D_Δ given by (4.18). The results were then plotted for some of these codes in Figure 7. In this figure, the horizontal axis represents the average distortion D and the vertical axis represents the code rate R. The solid curve denoted by R(D) represents the theoretical rate distortion function of the equiprobable binary discrete memoryless source, described in Chapter I. The second curve shows Goblick's results of the single pass step-by-step encoding procedure described earlier in this chapter.

From Figure 7 we see that the error-trapping algorithm gives consistently better results than the one-pass step-

TABLE 4 - SOME BINARY SOURCE CODES AND THEIR AVERAGE DISTORTION

CODE n, k, d	GENERATOR POLYNOMIAL	WEIGHT DISTRIBUTION OF REMAINDER CLASSES 0	1	2	3	4	5	6	7	8	9	10	11	RATE k/n	AVERAGE DISTORTION
15, 11, 3	23	1	15											.7333	0.0625
15, 10, 4	65	1	15	15	1									.6666	0.1
15, 9, 3	171	1	15	45	3									.6	0.11875
15, 8, 4	213	1	15	60	48	3	1							.5333	0.15416
15, 7, 5	721	1	15	105	125	10								.46666	0.16666
15, 7, 3	673	1	15	90	120	30								.46666	0.17578
15, 6, 4	1315	1	15	105	225	150	15	0	1					.4	0.2056
15, 5, 7	2467	1	15	105	450	425	28							.3333	0.22233
15, 4, 8	7531	1	15	105	455	875	553	43	1					.2666	0.2643
17, 9, 5	727	1	17	119	102	17								.5294	0.1445
17, 8, 6	1171	1	17	136	204	119	34	0	1					.4758	0.17854
21, 16, 3	43	1	21	7	3									.7619	0.06548
21, 15, 4	145	1	21	28	11	3								.71429	0.09077
21, 12, 5	1467	1	21	268	273	49								.57143	0.12760
21, 11, 6	2531	1	21	189	406	322	84	0	1					.52381	0.15495
21, 10, 5	5031	1	21	210	798	777	238	3	0					.47619	0.16627
21, 9, 8	17053	1	21	210	1134	1617	890	220	3					.42857	0.18720
21, 6, 7	173465	1	21	210	1330	5796	14301	10332	777					.2857	0.24203
21, 5,10	214537	1	21	210	1330	5985	17619	25060	13797	1512	0	0	1	.2380	0.27285
23, 12, 7	5343	1	23	230	1035	552	207							.52174	0.14502
23, 11, 8	17445	1	23	253	1265	1679	759	115	1					.47826	0.16487
31, 26, 3	45	1	31											.83871	0.03125
31, 25, 4	157	1	31	31	1									.80645	0.04839
31, 21, 5	3551	1	31	279	620	93								.67742	0.08887
31, 20, 6	4673	1	31	310	868	713	124	0	1					.64516	0.10607
31, 16, 7	107677	1	31	434	2821	8494	16182	4464	341					.51613	0.15103
31, 15, 8	310321	1	31	465	3255	11222	22847	19995	6603	1085	31	0	1	.48387	0.16982
33, 23, 3	3043	1	33	297	396	264	33							.69697	0.08984
33, 22, 6	5145	1	33	330	726	693	265							.66667	0.10310
33, 21, 3	17537	1	33	330	1320	2013	333	66						.63636	0.10924
35, 25, 4	2565	1	35	222	525	206	35							.7143	0.08518
35, 24, 4	7637	1	35	229	560	675	413	119	16					.6857	0.10831
35, 20, 6	147257	1	35	490	3080	10780	13265	5075	35	7				.5714	0.13118
35, 19, 6	251761	1	35	525	3507	13160	23590	18970	5635	112	0	0	1	.5428	0.14664
39, 27, 3	17075	1	39	429	1794	1755	78							.6923	0.08569
39, 26, 6	21105	1	39	468	2145	3393	1911	234	0	0	1			.6666	0.10000
39, 25, 4	57775	1	39	289	1079	2769	5135	6890	182					.6410	0.13023
39, 24, 6	167725	1	39	546	3120	9828	12714	6009	507	0	4			.6154	0.11973

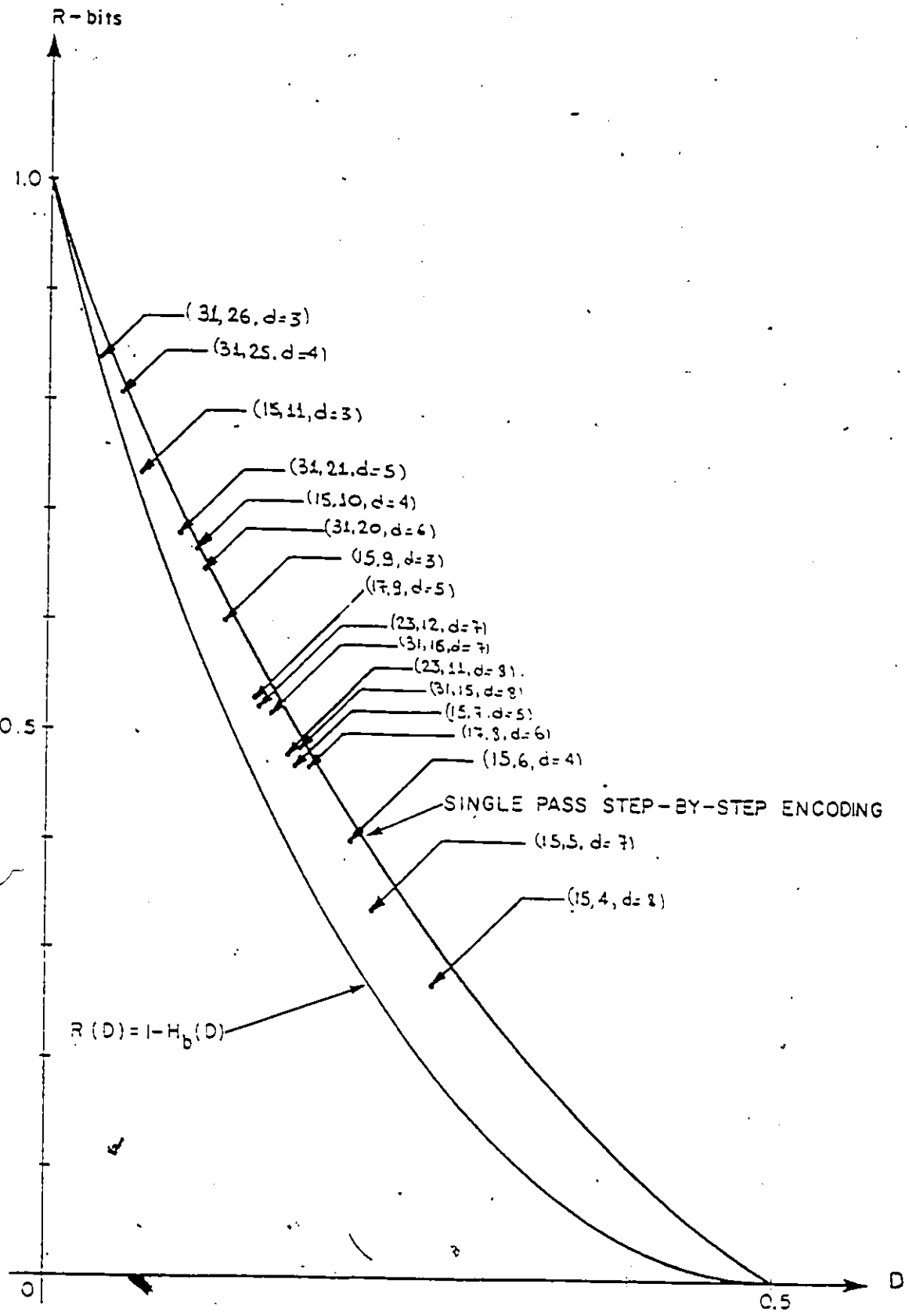


FIGURE 7 - Error Trapping Encoding compared with R(D) and Step-by-Step Encoding.

by-step encoding scheme, thus confirming our assertion. For example, the (21, 10) code with $g(x) = 1 + x^3 + x^4 + x^9 + x^{11}$ has $D_\Delta = .16627$ whereas the curve shows that, with the step-by-step method, an average distortion equal to 0.185 would have resulted. The reason for the better performance is easy to visualize in light of the method of operation of both algorithms. The step-by-step encoding algorithm has been shown earlier to be inherently random. Furthermore, the single-pass step-by-step does not fully exploit the capabilities of the source code since it accepts the results of the encoder after one pass only. With respect to the error-trapping algorithm, there is a payoff because of the orderly manner of its operation. Basically, the algorithm first selects a subset of likely codewords, then chooses, from this subset, the one that introduces the minimum distortion. We also note in passing that the distortion is always confined within $(n - k)$ consecutive digits.

To confirm our assertion with respect to majority-logic decoding, we simulated the operation of a majority-logic source encoder with the (15, 7, 2) BCH code. For this run it was necessary to generate all 2^n possible n-bit source sequences. For each sequence, the compressed k-bit sequence was estimated using the orthogonal estimates

relevant to the (15,7,2) code. This sequence was then multiplied by g(x) and the resulting codeword compared with the source sequence to determine the distortion. The final results were as shown below

Distortion	Number of Source Sequences
0	128
1	1920
2	13440
3	9216
4	6016
5	1152
6	768
7	128

Total $32768 = 2^{15}$.

The average distortion per bit is

$$\frac{0x128+1x1920+2x13440+3x9216+4x6016+5x1152+6x768+7x128}{32768 \times 15}$$

$$= 0.18672.$$

With reference to Table 4, the same code has a

maximum distortion of 4 and an average distortion of 0.1666 ... when the error trapping procedure is used.

We note that the (15, 7, 2) code was the only code for which the majority-logic algorithm was tried because of the relative rarity of majority logic decodable codes of moderate length and the excessive length of execution time necessary to simulate codes of length greater than 15.

We feel, however, that this unique example is sufficient to confirm our initial assertion, namely, that the performance of error-trapping is superior to that of majority-logic with respect to source encoding.

4.5 Upper Bound on the Average Distortion D_Δ

We established in the previous section that the performance of the error-trapping encoding is better than two other suboptimal encoding schemes, namely the single pass step-by-step and the majority-logic schemes. In order to confirm and generalize the results obtained, a tight upper bound on D_Δ is a necessary condition. This problem is as complex as deriving the coset enumeration of a code and generally speaking it is extremely difficult,

even if not impossible, to find such a bound analytically. In this context, the few results given below are helpful in getting rough pessimistic estimates of D_Δ in certain cases.

THEOREM 4.1

With the generator polynomial $G(x) = G_1(x)G_2(x) \ldots G_r(x)$, where each $G_j(x)$ is irreducible, the remainder class containing $\frac{G(x)}{G_j(x)}$ has e_j distinct remainders, e_j being the exponent* of $G_j(x)$.

PROOF

Suppose the source block $s(x)$ and $x^p s(x)$ have the same remainder. Then $s(x)\ (1 + x^p)$ can be expressed as

$$s(x)(1 + x^p) = M(x)G(x) = M(x)G_1(x)G_2(x) \ldots G_r(x).$$

Setting $s(x) = \frac{G(x)}{G_j(x)}$ and $p = e_j$ yields $s(x) \bmod G(x) = s(x)$. This means that the remainder class containing $\frac{G(x)}{G_j(x)}$ has $p = e_j$ distinct remainders.

This concludes the proof of Theorem 4.1.

---

*The exponent e of an irreducible polynomial is the order of its roots, i.e., if an irreducible polynomial belongs to e, then it divides $X^e - 1$ but no polynomial of the form $X^n - 1$ for $n < e$.

A direct result of Theorem 4.1 is the following.

COROLLARY 4.2

With $G(x) = G'(x)(1 + x)$, the remainder class containing $G'(x) = \frac{G(x)}{1 + x}$ has only one remainder.

This corollary provides an explanation to the observed behavior of certain codes given in Table 4, where there is a remainder of weight W and none of weight W - 1. Take, for example, the (17, 8, d = 6) code generated by $G(x) = 1 + x^3 + x^4 + x^5 + x^6 + x^9 = (1 + x)(1 + x + x^2 + x^4 + x^6 + x^7 + x^8)$. The weight of $G'(x) = 1 + x + x^2 + x^4 + x^6 + x^7 + x^8$ is, clearly, 7.

An implication of Corollary 4.2 is the following.

COROLLARY 4.3

With $G(x) = G'(x)(1 + x)$, $W \geq |G'(x)|$, where $W = \text{Max}\{\mu_1, \mu_2, \ldots \mu_u\}$, μ_i being the smallest weight in the remainder class σ_i.

An estimate of the average distortion per bit D_Δ can be found, using Theorem 4.1 for the case of the

($n = n_1 n_2$, $k = n - m_1 - m_2$ code with $G(x) = G_1(x)G_2(x)$, where $n_i = 2^{m_i} - 1$, $(n_1, n_2) = 1$* and $G_i(x)$ is irreducible with exponent n_i, $i = 1, 2$. For this code $\mu_0 = 0$, $\upsilon_0 = 0$; $\mu_1 = 1$, $\upsilon_1 = n_1 n_2$; $\mu_2 \geq |G_1(x)|$, $\upsilon_2 = n_2$; $\mu_3 \geq |G_2(x)|$, $\mu_3 = n_2$; Substituting in Equation (4.17) we get

$$D_\Delta \leq \frac{n_1 n_2 + |G_1(x)| n_2 + |G_2(x)| n_1}{n_1 n_2 \, 2^{m_1 + m_2}}$$

or

$$D_\Delta \leq \frac{n_1 n_2 + |G_1(x)| n_2 + |G_2(x)| n_1}{n_1 n_2 \, (n_1 + 1)(n_2 + 1)} \qquad (4.19)$$

For example, the (21, 16) code generated by $G(x) = 1 + x^4 + x^5$ has $G_1(x) = 1 + x + x^2$ with exponent 3 and $G_2(x) = 1 + x + x^3$ with exponent 7. Substitution in (4.19) gives $D_\Delta \leq 0.0759$ while actual verification (from

*The notation $(n_1, n_2) = 1$ means that the Greatest Common Divisor of n_1 and n_2 is 1, i.e., n_1 and n_2 are relatively prime.

Table 4) shows that $D_\Delta = 0.06548$.

We now supose that $\rho_0(x)$ is a remainder such that $x^j \rho_0(x)$ has degree $n - k$, so that

$$x\,\rho_0(x) = G(x) + \rho_j(x),$$

where $\rho_0(x)$ and $\rho_j(x)$ are in the same remainder class. By examining the maximum weight that $\rho_j(x)$ can have for $|\rho_0(x)| = 1, 2, 3, \ldots n - k$, we find that

$$W \leq n - k - \frac{|G(x)| - 1}{2} \quad \text{for } |G(x)| \text{ odd},$$

and $$W \leq n - k - \left|\frac{G(x) - 2}{2}\right| \quad \text{for } |G(x)| \text{ even}.$$

These results can then be restated, in the context of Corollary 4.3 in the following.

THEOREM 4.4

If $|G(x)|$ is odd, then $W \leq n - k - \frac{|G(x)| - 1}{2}$.

If $|G(x)|$ is even, then $\left|\frac{G(x)}{1 + x}\right| \leq W \leq n - k - \frac{|G(x)| - 2}{2}$

COROLLARY 4.5

If $|G(x)|$ is even and $\left|\frac{G(x)}{1+x}\right| + \left|\frac{G(x)}{2}\right| = n - k + 1$, then $W = \left|\frac{G(x)}{1+x}\right| = n - k - \frac{|G(x)| - 2}{2}$.

For example, the (15, 6) code has $G(x) = 1 + x^2 + x^3 + x^6 + x^7 + x^9$ which can be written as $(1 + x)(1 + x + x^3 + x^4 + x^5 + x^7 + x^8)$. Therefore $G'(x) = 1 + x + x^3 + x^4 + x^5 + x^7 + x^8$,

$$|G'(x)| = 7,$$

$$\left|\frac{G(x)}{2}\right| = 3$$

and $n - k + 1 = 10$, so that from Corollary 4.5, $W = 7$ as can be readily verified from Table 4.

4.6 EXAMPLE: TRANSMISSION OF ALPHABETIC PATTERNS

We present the following example to illustrate the use of the source encoding scheme Δ discussed in this chapter. The binary source is an optical device used to scan alphabetic patterns. The binary output is then fed to the source encoder Δ where the data-compression

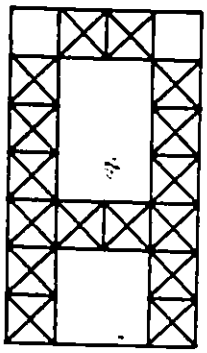
process takes place before transmission. To illustrate the effect of code length, we assume that the resolution of the scanner can be adjusted.

Let us consider the letter A, shown in Figure 8.1(a), drawn within a 7 x 4 rectangle. With a vertical scan, from left to right, the pattern can be decomposed into four binary vectors, namely $S_1(x) = S_4(x) = x + x^2 + x^3 + x^4 + x^5 + x^6$ and $S_2(x) = S_3(x) = 1 + x^4$. Using a (7, 4) code with $G(x) = 1 + x^2 + x^3$ we get:

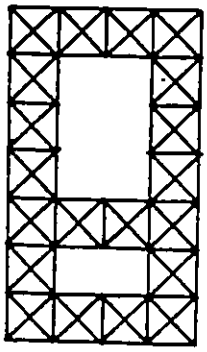
For S_1 and S_4: $\rho_0(x) = 1$ giving $V_1(x) = V_4(x) = 1 + x + x^2 + x^3 + x^4 + x^5 + x^6$

For S_2 and S_3: $\rho_1(x) = 1$ giving $V_2(x) = V_3(x) = 1 + x^4 + x^{-1} = 1 + x^4 + x^6$

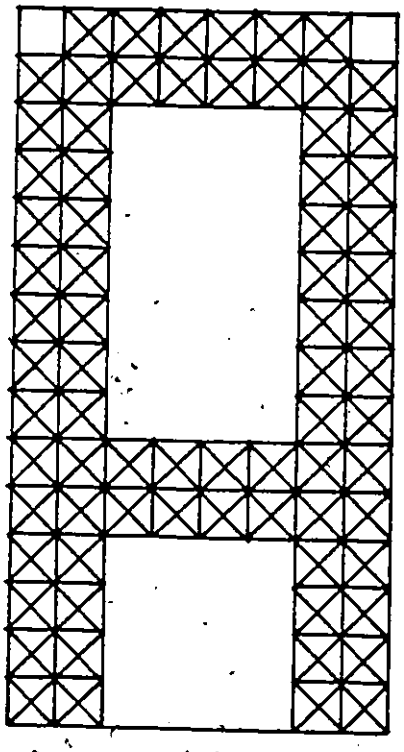
Assuming no channel errors occur during transmission, the received pattern will appear distorted as shown in Figure 8.1(b) and could be mistaken for some other pattern. Let us now use a higher scanner resolution, and assume that the same letter A can be drawn within a 15 x 8 rectangle as shown in Figure 8.1(c). A (15, 11) code with $G(x) = 1 + x + x^4$ gives the pattern of Figure 8.1(d) while the (15, 7) code with $G(x) = 1 + x^4 + x^6 + x^7 + x^8$



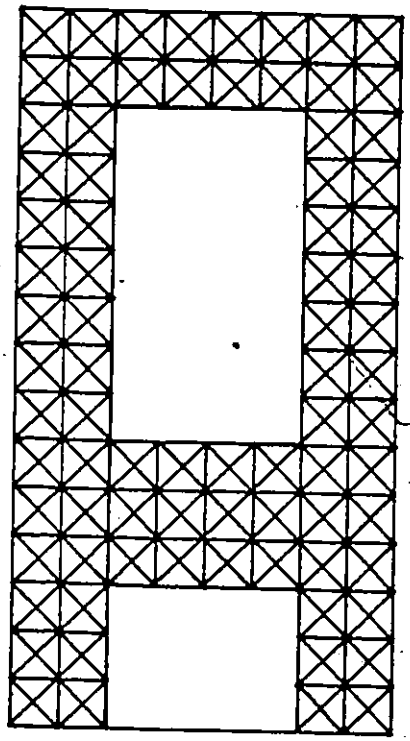
(a)



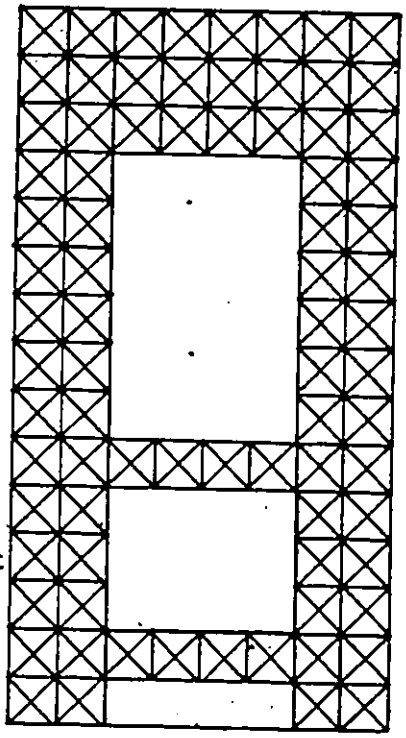
(b)



(c)



(d)



(e)

FIGURE 8.1 - A PATTERN

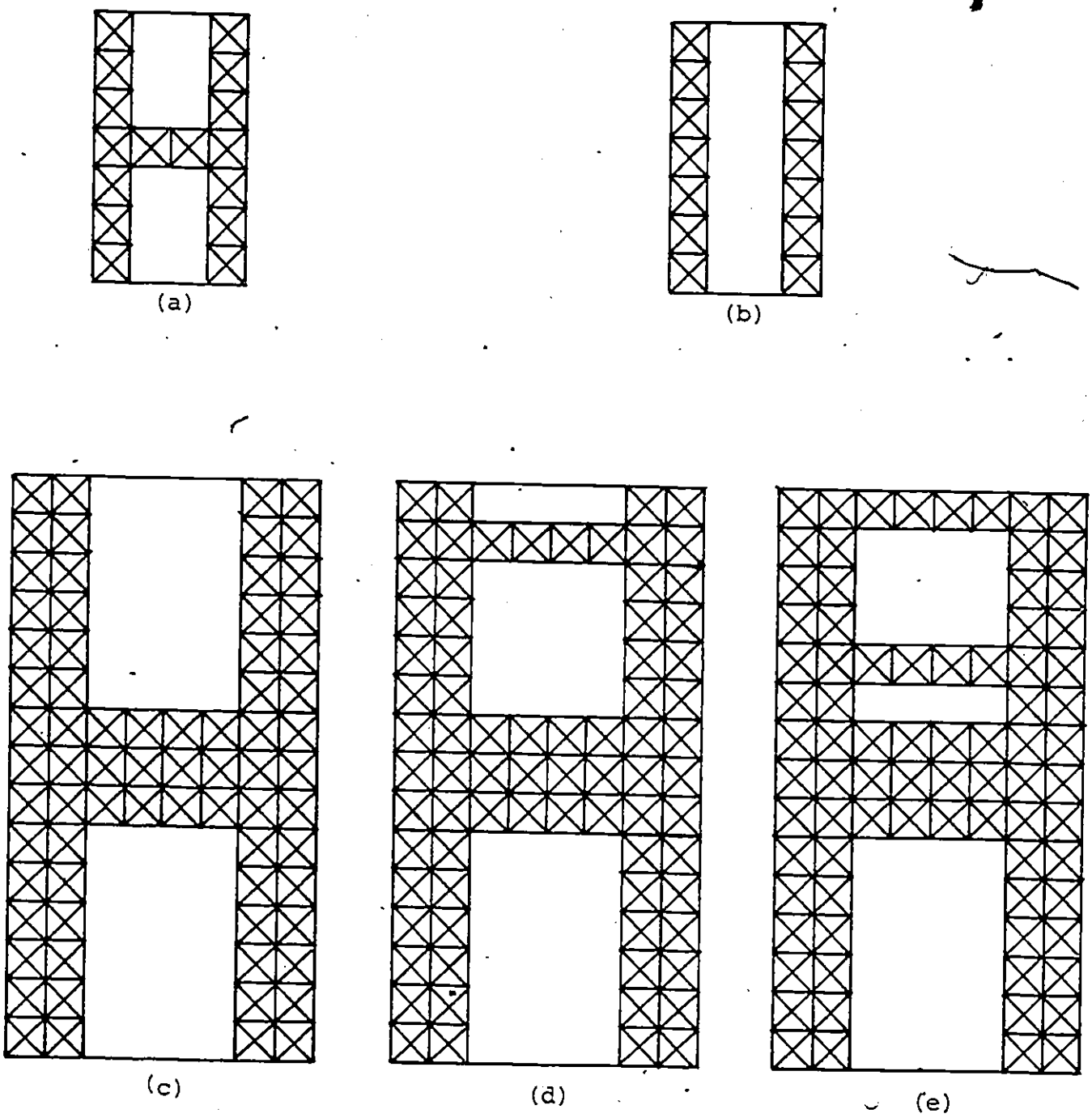


FIGURE 8.2 - H PATTERN

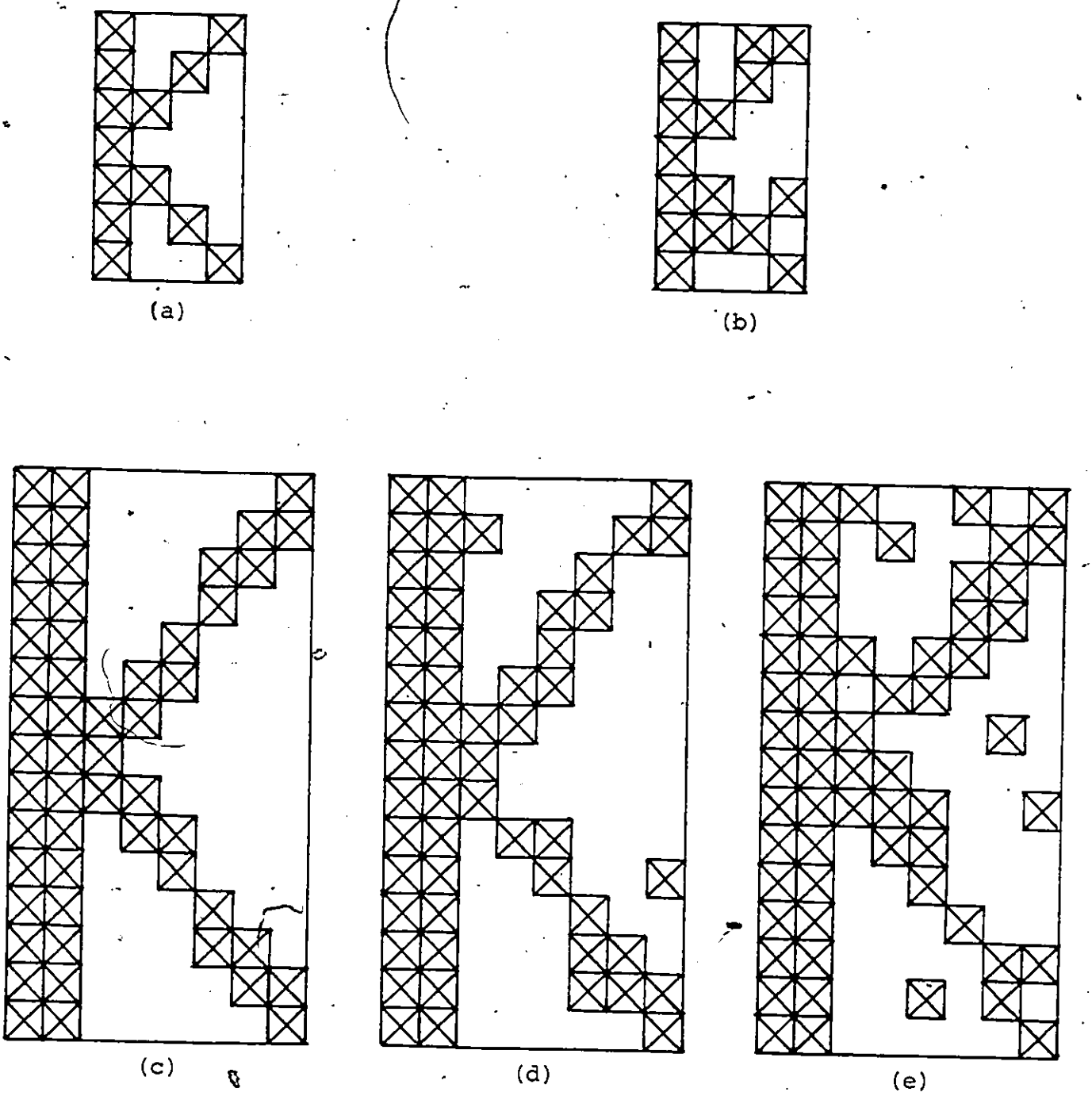


FIGURE 8.3 - K PATTERN

gives the pattern of Figure 8.1(e). As we have shown earlier in Table 4, the (15, 11) code gives a lower average distortion than the (15, 7) code. Clearly, the pattern of Figure 8.1(d) is unequivocally recognized as A, while there may be some doubt about Figure 8.1(e).

Similar exercises have been carried out for the letters H and K, as shown in Figures 8.2 and 8.3. These examples show the need for a tradeoff between the desired compression ratio and the tolerable distortion.

4.7 An Extension of the Error-Trapping Technique

The simple error-trapping procedure described above can be modified to yield a lower average distortion, but at the cost of additional complexity. The modification we propose involves using two codes instead of one in the following manner.

Suppose we use two codes of equal length but different rates, say (n, k_1) and (n, k_2), where $k_1 \neq k_2$, with the respective generator polynomials $g_1(x)$ and $g_2(x)$. Given the n-bit source block $S(x)$, we compute the "nearest" codewords $V_1(x)$ corresponding to $g_1(x)$ and $V_2(x)$ corresponding to $g_2(x)$. If $|S(x) + V_1(x)| < |S(x) + V_2(x)|$, we

transmit the k_1-tuple $V_1(x)/g_1(x)$. Otherwise we transmit the k_2-tuple $V_2(x)/g_2(x)$. If $|S(x) + V_1(x)| = |S(x) + V_2(x)|$, we should naturally transmit the shorter of the k-tuples.

For example, suppose the first code V_1 is (15, 7) with $g_1(x) = 1 + x^4 + x^6 + x^7 + x^8$ and the second code V_2 is (15, 8) with $g_2(x) = 1 + x + x^3 + x^7$. We remark, in passing, that $g_2(x)$ is the reciprocal of the polynomial $h_1(x) = (1 + x^{15})/g_1(x)$, that is, V_2 is the dual of V_1. At this point suppose $S(x) = 1 + x^2 + x^6$. Using the error-trapping procedure with the code V_1, we find $V_1(x) = 0$ whereas, with the code V_2, we find $V_2(x) = 1 + x^2 + x^6 + x^{14}$. Since $|S(x) + V_1(x)| = 3 > 1 = |S(x) + V_2(x)|$, we transmit the 8-tuple $I_2(x) = V_2(x)/g_2(x) = (1 + x^2 + x^6 + x^{14})/(1 + x + x^3 + x^7) = 1 + x + x^3 + x^7$.

Reverting to the general discussion, it is clear from what has been said so far that, if D_i is the average distortion associated with the code V_i, $i = 1, 2$ and $D_1 \leq D_2$, then the average distortion $D_{12} \leq D_1$, when both codes are used. The resultant compression ratio will be somewhere in between the compression ratios of the individual codes. By making k_1 and k_2 nearly equal to each other, the compression ratio can be kept almost

the same with or without modification.

On the other hand, it is also clear that the modified source encoding scheme becomes, in effect, a "variable-packet-length" scheme so far as transmission is concerned, since we must now have a means of determining, at the receiving end, whether the packet belongs to V_1 or V_2.

Also, we might have to overcome a synchronization problem if we do not know the starting time of the received sequence.

Both problems, however, are not hard to overcome if we allow some overhead by enclosing the packet in a frame containing synchronizing bits and a packet identifer. This approach is being used in practice with the well-known Binary Synchronous Communications (BSC) and High Level Data Link Control (HDLC) protocols.

CHAPTER V

CONCLUDING REMARKS

In this thesis, the use of the coset properties of binary cyclic codes has been examined. The knowledge of the coset enumeration of any given code plays an important role in both channel coding and source coding theories. In the former, it allows the calculation of the exact probability of error when maximum likelihood decoding is used. In the latter case, it permits the estimation of the average distortion resulting from the compression of the output of binary discrete memoryless source.

A literature survey has revealed that there are very few classes of codes for which the coset enumeration is known. There are, however, several lower and upper bounds on the maximum coset weight that may, in certain cases, yield the exact value of the covering radius. The best known upper bound is due to Delsarte and it relates the covering radius of a code to the number of nonzero weights in the dual code. This bound is applicable to any linear code, but, as we have shown, is not generally tight. In this connection, we have presented an upper bound that proved to be tighter for codes that satisfy the Plotkin bound. A lower bound on the covering radius of BCH codes

was first derived by Gorenstein, Peterson and Zierler, and a different version was later given by Van der Horst and Berger. We have shown that the GPZ bound is a special case of a more general bound relating the minimum distance of a group code to the covering radius of a proper subgroup of this code.

We used the bounds we derived to show that the covering radius of the $(2^m - 1, m)$ MLSR codes is $\frac{n - 1}{2}$. We then carried the examination of the cosets of the MLSR codes one step further and used several combinatorial arguments to derive an expression for the number of cosets of weight $\frac{n + 1}{4}$. Our purpose in this exercise was twofold. First, the MLSR codes form an important class which has many practical applications. Secondly, the results we were seeking were to be only dependent on the parameters of the codes themselves, unlike Dick and Sloane's results.

In the context of our search for candidate block codes for source encoding purposes we have analyzed the subclass of the shortened-by-1 bit double-error-correcting BCH codes. Starting with the knowledge that the parent code is quasi-perfect, we have examined the cosets of the shortened code, and we have succeeded in deriving the complete coset enumeration of the shortened codes for all $m \geq 4$.

The (14, 6, 2) code was shown to be a special case which was verified separately, but for all other $m > 4$, the $(2^m - 2, k = n - 2m - 1)$ codes were shown to be nearly quasiperfect, in the sense that there is only one coset with maximum weight 4.

An interesting by-product of our analysis was the expression for the number of codewords of weight 5 in the $(2^m - 1, k = n - 2m, t = 2)$ BCH codes for odd m. Our result complements that of Berlekamp who gave the expression for even m. We have verified the validity of our expression by applying MacWilliam's identities to Kasami's weight enumeration of the dual of the code.

A topic for further research in this area is suggested. The 3-error correcting BCH codes were shown to have a covering radius equal to 5. Can the coset enumeration of the shortened codes be obtained from this knowledge? The problem is challenging since it involves the solution of equations of degree 2, 3, 4 and 5 in $GF(2^m)$.

We have also compared the average distortion of both the parent and shortened codes. For long blocklengths, the values obtained are comparable, thus showing that the

increase in the maximum coset weight (from 3 to 4) does not significantly affect the average distortion.

In discussing the use of the standard array for source encoding, we mentioned that, even though this scheme would be optimum in terms of average distortion, the source encoder implementing this scheme would clearly be non instrumentable since the memory requirements grow exponentially with the blocklength. Suboptimum approaches were therefore necessary. In this context, we discussed the use of BCH decoding and majority logic decoding, then concluded that both schemes were impractical. We suggested, as a better alternative, to use the well-known technique of error-trapping as a source encoding scheme. We described the operation of such a source coder then analysed its performance by simulating a number of codes. Our results clearly showed the error-trapping technique to give better average distortion than the one-pass step-by-step encoding technique analysed by Goblick. We also simulated a majority-logic encoder for the (15, 7, 2) BCH code to confirm that it did not perform as well as the error-trapping method. We then attempted to consolidate the simulation results with an analytical derivation of tight upper bound on the distortion. We only succeeded in obtaining some rough pessimistic

estimates. The proof of a tight bound has eluded us and we suggest that this problem be a topic for further research.

Finally, we suggested an extension of the error-trapping technique in which two codes of equal length but different rates can be used to yield an average distortion at most equal to the lower of the average distortions resulting when each code is used individually. We briefly outlined some potential problems associated with the use of such a scheme and discussed a way to solve these problems. Several topics for further research can be suggested. One of them may be to investigate a possible relationship between the two codes that would minimize the distortion. A code and its dual, as used in our example, is one alternative; another alternative would be a code and its even-weighted subset. In the same context, an expression for the average distortion, in terms of the weight distribution of remainder classes of both codes, is a likely research subject. The practical aspects of implementing this extended scheme, such as encoder complexity and amount of overhead should also be investigated. Further extensions of the error-trapping technique, such as the use of multiple codes or the use

of codes of different lengths, could also be the subject of further research.

APPENDIX A

Next k-Subset of an n-Set

Considered here are the combinations of n things taken k at a time. There are $\binom{n}{k}$ k-subsets of an n-set altogether, and the algorithm given here will generate them all, sequentially in "alphabetical order". For example, the list of the ten 3-subsets of {1, 2, 3, 4, 5} will be

{1,2,3}, {1,2,4}, {1,2,5}, {1,3,4}, {1,3,5},
{1,4,5}, {2,3,4}, {2,3,5}, {2,4,5}, {3,4,5}.

The sucessor of a given k-subset $\{a_1, \ldots, a_k\}$ is obtained as follows: search for the smallest h such that $a_{k+1-h} < n + 1 - h$, then increase a_{k+1-h} by 1, and set $a_j \leftarrow a_{j-1} + 1$ $(j = k + 2 - h, k)$.

Algorithm NEXKSB

(A) (First entry) m = 0; h = k; goto (D);

(B) (Later entries) If m = n - h then goto (C);
else h = 0;

(C) h = h + 1; m = a_{k+1-h};

(D) For j = 1,h : $(a_{k+j-h} = m + j)$;
If a_1 = n - k + 1 then goto (E);
else goto (B);

(E) Exit.

PL/I OPTIMIZING COMPILER RESIDU: PROCEDURE OPTIONS (MAIN) REORDER ;

SOURCE LISTING

```
NUMBER LEV NT
    10      0  RESIDU : PROCEDURE OPTIONS (MAIN) REORDER ;                                    RES00010
                                                                                              RES00020
                 /* PARAMETERS */                                                            RES00030
                                                                                              RES00040
    50    1   0  DECLARE ( GEN_DEGREE , MAX_RESIDUE_WEIGHT ) FIXED BINARY INITIAL (50) ;    RES00050
    60    1   0  DECLARE   RESIDUE_WEIGHT                     FIXED BINARY INITIAL (1) ;    RES00060
    70    1   0  DECLARE   RESIDUE_TOTAL                      FIXED BINARY INITIAL (1) ;    RES00070
    80    1   0  DECLARE (RESULT_WEIGHT,MIN_RESIDUE_WEIGHT)  FIXED BINARY INITIAL (0) ;    RES00080
                                                                                              RES00090
                 /* ARRAYS */                                                                RES00100
                                                                                              RES00110
   120    1   0  DECLARE (RESIDUE_VECTOR(0:GEN_DEGREE)) FIXED BINARY                        RES00120
                                  INITIAL ( (1+GEN_DEGREE) (0) ) ;                            RES00130
   140    1   0  DECLARE (TEMP_VECTOR(0:GEN_DEGREE))    FIXED BINARY                        RES00140
                                  INITIAL ( (1+GEN_DEGREE) (0) ) ;                            RES00150
   160    1   0  DECLARE (RESULT_VECTOR(0:GEN_DEGREE))  FIXED BINARY                        RES00160
                                  INITIAL ( (1+GEN_DEGREE) (0) ) ;                            RES00170
   180    1   0  DECLARE (GEN_POLYNOMIAL(0:GEN_DEGREE)) FIXED BINARY                       RES00180
                                  INITIAL ( (1+GEN_DEGREE) (0) ) ;                            RES00190
   200    1   0  DECLARE (SHIFT_VECTOR(0:GEN_DEGREE)) FIXED BINARY                         RES00200
                                  INITIAL ( (1+GEN_DEGREE) (0) ) ;                            RES00210
   220    1   0  DECLARE (RES_COUNT(0:MAX_RESIDUE_WEIGHT)) FIXED BINARY                     RES00220
                                  INITIAL ((1+(MAX_RESIDUE_WEIGHT) (0)) ;                     RES00230
                                                                                              RES00240
                 /* LOCAL VARIABLES */                                                       RES00250
                                                                                              RES00260
   270    1   0  DECLARE ( I, J, K, H, M, COUNTER, TOTAL ) FIXED BINARY INITIAL (0) ;        RES00270
   280    1   0  DECLARE ( FIRST , COMPLETE , TRUE , FLAG )   BIT(1) INITIAL ('0'B) ;       RES00280
   290    1   0  DECLARE  MIN  BUILTIN ;                                                     RES00290
                                                                                              RES00300
                 /* ACCEPT INPUT FROM TERMINAL */                                            RES00310
   320    1   0  PUT SKIP EDIT (' DEGREE OF G(X) : ') (A) ;                                 RES00320
   330    1   0  GET LIST (GEN_DEGREE) ;                                                    RES00330
   340    1   0  PUT SKIP EDIT ( ' ENTER G(X) IN BINARY FORM : ' ) (A) ;                    RES00340
   350    1   0  GET LIST ( (GEN_POLYNOMIAL(K) DO K=0 TO GEN_DEGREE) ) ;                   RES00350
                                                                                              RE600360
                 /* DEFINE AND INITIALIZE PARAMETERS */                                      RES00370
                                                                                              RES00380
   390    1   0  MAX_RESIDUE_WEIGHT = GEN_DEGREE;                                          RES00390
   400    1   0  RESIDUE_WEIGHT = 1 ;                                                       RES00400
   410    1   0  TOTAL = 2 ** GEN_DEGREE ;                                                 RES00410
   420    1   0  COUNTER = 0 ;                                                             RES00420
                                                                                              RES00430
                                                                                              RES00440
                 /* START MAIN PROGRAM */                                                    RES00450
                                                                                              RES00460
   470    1   0  MAIN_LOOP : BEGIN ;                                                       RES00470
   480    2   0                DO WHILE (RESIDUE_WEIGHT <= MAX_RESIDUE_WEIGHT );          RES00480
                                                                                              RES00490
                           /* GENERATE FIRST VECTOR OF WEIGHT EQ. RESIDUE_WEIGHT */        RES00500
   510    2   1                    DO I = 0 TO RESIDUE_WEIGHT - 1 ;                       RES00510
```

```
NUMBER LEV NT

  520    2  2                  TEMP_VECTOR(I) = I + 1 ;
  530    2  2                  END ;
  550    2  1   INTER_LOOP : BEGIN ;
  560    3  1               DO WHILE
                               ((TEMP_VECTOR(0) <= (GEN_DEGREE-RESIDUE_WEIGHT+1))
                               & (TEMP_VECTOR(RESIDUE_WEIGHT-1) <= GEN_DEGREE )) ;

  600    3  2         PROCESS_LOOP : BEGIN ;

  620    4  2         COMPLETE = '0'B;
  630    4  2         MIN_RESIDUE_WEIGHT = RESIDUE_WEIGHT ;

                      /* TRANSFORM TEMP_VECTOR INTO RESIDUE_VECTOR */
  660    4  2         DO I = 0 TO GEN_DEGREE ;
  670    4  3         RESIDUE_VECTOR(I) = 0 ;
  680    4  3         END ;
  690    4  2         DO J = 0 TO RESIDUE_WEIGHT - 1 ;
  700    4  3         RESIDUE_VECTOR( TEMP_VECTOR( J ) - 1 ) = 1 ;
  710    4  3         END ;

                      /* DIVIDE BY GEN_POLYNOMIAL TO FIND RESIDUE */
  740    4  2         TRUE = '1'B ;
  750    4  2         FIRST = '1'B ;
  760    4  2         RESULT_VECTOR = 0 ;
  770    4  2         SHIFT_VECTOR = RESIDUE_VECTOR ;
  780    4  2         DO WHILE (TRUE) ;
  790    4  3            IF SHIFT_VECTOR(GEN_DEGREE) = 0
                            THEN
                               RESULT_VECTOR = SHIFT_VECTOR ;
  820    4  3            ELSE DO I = 0 TO GEN_DEGREE;
  830    4  4                  IF SHIFT_VECTOR(I) = GEN_POLYNOMIAL(I)
  850    4  4                     THEN RESULT_VECTOR(I) = 0 ;
  860    4  4                     ELSE RESULT_VECTOR(I) = 1 ;
                              END ;

                         /* CHECK THE WEIGHT OF THE RESULT VECTOR  */

  900    4  3            RESULT_WEIGHT = 0 ;
  910    4  3            DO I = 0 TO GEN_DEGREE ;
  920    4  4            IF RESULT_VECTOR(I) = 1
                            THEN
                            RESULT_WEIGHT = RESULT_WEIGHT + 1 ;
  950    4  4            END ;

  970    4  3            MIN_RESIDUE_WEIGHT=MIN(MIN_RESIDUE_WEIGHT,RESULT_WEIGHT);

  990    4  3                  DO ;
 1000    4  4                  FLAG = '1'B ;
 1010    4  4                  DO J = 0 TO GEN_DEGREE ;
 1020    4  5                     IF RESULT_VECTOR(J) ¬= RESIDUE_VECTOR(J)
                                     THEN
                                     FLAG = '0'B ;
 1050    4  5                  END ;
```

COLOURED PAPER
PAPIER DE COULEUR

```
PL/I OPTIMIZING COMPILER        RESIDU : PROCEDURE OPTIONS (MAIN) REORDER ;

   NUMBER LEV NT

                                                                                RES01060
     1070   4  4                    IF (FLAG & ¬ FIRST)                         RES01070
                                       THEN DO ;                                RES01080
     1090   4  5                          COMPLETE = '1'B ;                     RES01090
     1100   4  5                          TRUE = ¬ TRUE   ;                     RES01100
     1110   4  5                          END ;                                 RES01110
     1120   4  4                       ELSE DO ;                                RES01120
     1130   4  5                          SHIFT_VECTOR = 0 ;                    RES01130
     1140   4  5                          COUNTER = COUNTER + 1 ;               RES01140
     1150   4  5                             DO J = 0 TO GEN_DEGREE - 1 ;       RES01150
     1160   4  6                                SHIFT_VECTOR(J+1) = RESULT_VECTOR(J) ;RES01160
     1170   4  6                             END ;                              RES01170
                                                                                RES01180
     1190   4  5                          END ;                                 RES01190
     1200   4  4                    END ;                                       RES01200
                                                                                RES01210
                                                                                RES01220
     1230   4  3          FIRST = '0'B ;                                        RES01230
                                                                                RES01240
                                                                                RES01250
     1260   4  3          END ;                                                 RES01260
                                                                                RES01270
                                                                                RES01280
     1290   4  2          IF (COMPLETE) THEN                                    RES01290
                             DO ;                                               RES01300
     1310   4  3             RESIDUE_TOTAL = RESIDUE_TOTAL + 1 ;                RES01310
     1320   4  3             RES_COUNT(MIN_RESIDUE_WEIGHT)=RES_COUNT(MIN_RESIDUE_WEIGHT)+1 ;RES01320
     1330   4  3             END ;                                              RES01330
                                                                                RES01340
                                                                                RES01350
     1360   4  2             END PROCESS_LOOP ;                                 RES01360
                                                                                RES01370
                           /* GENERATE NEXT VECTORS OF WEIGHT RESIDUE_WEIGHT */ RES01380
                                                                                RES01390
     1400   3  2          TRUE = '1'B ;                                         RES01400
     1410   3  2          DO H = 1 TO RESIDUE_WEIGHT WHILE (TRUE) ;             RES01410
     1420   3  3          M = TEMP_VECTOR(RESIDUE_WEIGHT - H)     ;             RES01420
     1430   3  3          IF M ¬= MAX_RESIDUE_WEIGHT - H + 1                    RES01430
                             THEN TRUE = ¬ TRUE                 ;               RES01440
     1450   3  3          END ;                                                 RES01450
                                                                                RES01460
     1470   3  2          DO J = 1 TO H - 1 ;                                   RES01470
     1480   3  3          TEMP_VECTOR(RESIDUE_WEIGHT + J - H) = M + J ;         RES01480
     1490   3  3          END ;                                                 RES01490
                                                                                RES01500
     1510   3  2        END;END INTER_LOOP ;                                    RES01510
                                                                                RES01520
     1530   2  1        RESIDUE_WEIGHT = RESIDUE_WEIGHT + 1 ;                   RES01530
                                                                                RES01540
     1550   2  1      END;END MAIN_LOOP;                                        RES01550
                                                                                RES01560
                       /* OUTPUT DISTRIBUTION OF RESIDUE WEIGHTS */             RES01570
     1590   1  0      RESIDUE_TOTAL = 0 ;                                       RES01580
                                                                                RES01590
```

PL/I OPTIMIZING COMPILER RESIDU : PROCEDURE OPTIONS (MAIN) REORDER ;

NUMBER LEV NT

```
                                                                                    RES01600
                                                                                    RES01610
 1610   1   0  PUT SKIP(4) EDIT (' WEIGHT DISTRIBUTION ') (A) ;                     RES01620
 1620   1   0  PUT SKIP     EDIT (' OF RESIDUE CLASSES  ') (A) ;                   RES01630
 1630   1   0  PUT SKIP     EDIT ('---------------------') (A) ;                   RES01640
 1640   1   0  PUT SKIP ;                                                         RES01650
                                                                                    RES01660
 1660   1   0  DO I = 0 TO MAX_RESIDUE_WEIGHT ;                                   RES01670
 1670   1   1  PUT SKIP EDIT ( I , RES_COUNT(I) ) (X(1) , F(3) , X(4) , F(6) ) ; RES01680
 1680   1   1  RESIDUE_TOTAL = RESIDUE_TOTAL + RES_COUNT(I) ;                     RES01690
 1690   1   1  END ;                                                             RES01700
 1700   1   0  PUT SKIP(2) EDIT (' TOTAL',RESIDUE_TOTAL) (A,X(2),F(6)) ;        RES01710
 1710   1   0  END RESIDU ;
```

REFERENCES

[1] C.E. Shannon, "A Mathematical Theory Of Communication", Bell System Tech. J., vol. 27, pp. 379-423, July 1948 and pp. 623-656, Oct. 1948.

[2] A.J. Viterbi, "Error Bounds for Convolutional Codes and An Asymptotically Optimum Decoding Algorithm", IEEE Trans. Inform Theory, vol. IT-13, pp. 260-269, April 1967.

[3] I.M. Jacobs, "Practical Applications of Coding", IEEE Trans. Inform Theory, vol. IT-20, No. 3, pp. 305-310, May 1974.

[4] J.H. Blythe and K. Edgcombe, "Net Coding Gain of Error Correcting Codes", Proc. IEE, vol. 122, No.6, pp. 609-614, June 1975.

[5] W.W. Peterson and E.J. Weldon, "Error-Correcting Codes", 2nd ed. Cambridge, Mass.: M.I.T. Press, 1972.

[6] S. Lin, "An Introduction to Error-Correcting Codes", Prentice-Hall, Englewood Cliffs, N.J., 1970.

[7] E.R. Berlekamp, "Algebraic Coding Theory", New York: McGraw-Hill, 1968.

[8] I.F. Blake and R.C. Mullin, "The Mathematical Theory of Coding", Academic Press: New York, 1975.

[9] F.J. MacWilliams and N.J.A. Sloane, "The Theory of Error-Correcting Codes", North-Holland Publishing Company, 1977.

[10] C.E. Shannon, "Coding Theorems for a Discrete Source with a Fidelity Criterion", IRE National Conv. Rec., Part 4, pp. 142-163, 1959.

[11] T. Berger, "Rate Distortion Theory, A Mathematical Basis for Data Compression", Englewood Cliffs, N.J.: Prentice-Hall, 1971.

[12] R.A. Fano, "Transmission of Information", M.I.T. Press, N.Y., 1961.

[13] D.A. Huffman, "A Method for the Construction of Minimum Redundancy Codes", Proc. IRE. pp. 1098-1101, Sept. 1952.

[14] J. Capon, "A Probabilistic Model for Run-Length Coding of Pictures" IEEE Trans. Inform. Theory, vol. IT -5, pp. 157-163, December 1959.

[15] J.I. Molinder, "Optimal Coding with a Single Standard Run Length", IEEE Trans. Inform. Theory, vol. IT-20, No.3, pp. 336-343, May 1974.

[16] A.N. Netravali and J.O. Limb, "Picture Coding: A Review", IEEE Proc., vol. 68, No. 3, pp. 366-406, March 1980.

[17] Special Issue on Redundancy Reduction, Proc. IEEE, vol. 5, No. 3, March 1967.

[18] Special Issue on Image Bandwidth Compression, IEEE Trans. Communications, vol. COM-25, No. 11, October 1977.

[19] P. Delsarte, "Four Fundamental Parameters of a Code and their Combinatorial Significance", Inform. and Control, vol. 23, pp. 407-438, 1973.

[20] D. Gorenstein, W.W. Peterson and N. Zierler, "Two-Error-Correcting Bose-Chaudhuri Codes are Quasi-Perfect", Inform. and Control, vol. 3, pp. 292-294, 1960.

[21] T.H. Abdel-Nabi and S.G.S. Shiva, "On the Cosets of Shortened Double-Error-Correcting Binary BCH Codes", Digital Processes, vol. 5, pp. 151-157, 1979.

[22] T.H. Abdel-Nabi and S.G.S. Shiva, "Error Trapping Technique for Data Compression", Proc. International Conference on Communications (ICC '81), Denver, pp. 65.8.1-65.8.5, June 1981.

[23] T.C. Ancheta, Jr., "Syndrome-Source-Coding and its Universal Generalization", IEEE Trans. Inform. Theory, (Corresp.) vol. IT-22, No. 2, pp. 432-436, July 1976.

[24] R.C. Rose and D.K. Ray-Chaudhuri, "On a Class of Error Correcting Binary Group Codes", Inform. and Control, vol. 3, pp. 68-79, March 1960.

[25] A. Hocquenghem, "Codes Correcteurs d'erreurs", Chiffres, vol. 2, pp. 147-156, 1959.

[26] D. Gorenstein and N. Zierler, "A Class of Error-Correcting Codes in p^m Symbols", J. Soc. Ind. Appl. Math., vol. 9; pp. 207-214, June 1961.

[27] R.T. Chien, "Cyclic Decoding Procedure for the Bose-Chaudhuri-Hocquenghem Codes", IEEE Trans Inform. Theory, vol. IT-10, pp. 357-363, October 1964.

[28] G.D. Forney, Jr., "On Decoding BCH Codes", IEEE Trans. Inform. Theory, vol. IT-11, pp. 549-557, October 1965.

[29] J.L. Massey, "Step-by-Step Decoding of the Bose-Chaudhuri-Hocquenghem Codes", IEEE Trans. Inform Theory, vol IT-11, pp. 580-585, October 1965.

[30] A.M. Michelson, "Computer Implementation of Decoders for Several BCH Codes", Symp. on Computer Processing in Communications, Polytechnic Institute of Brooklyn, pp. 401-413, April 1969.

[31] G.I. Davida, "Decoding of BCH Codes", Electronics Letters, vol. 7, No. 22, p. 664, 4th November 1971.

[32] A. Tietavainen, "On the Nonexistence of Perfect Codes over Finite Fields", SIAM J. Appl. Math., vol. 24, No. 1, pp. 88-96, 1973.

[33] M.J.E. Golay, "Notes on Digital Coding", Proc. IRE (Corresp.) vol. 37, p. 657, June 1949.

[34] R.W. Hamming, "Error Detecting and Error Correcting Codes", Bell System Tech. J., vol. 29, pp. 147-160, 1950.

[35] C.R.P. Hartmann, "A Note on the Decoding of Double-Error-Correcting Binary BCH Codes of Primitive Length", IEEE Trans. Inform. Theory (Corresp.), vol. IT-17, pp. 765-766, Nov. 1971.

[36] J.A. Van Der Horst and T. Berger, "Complete Decoding of Triple-Error Correcting Binary BCH Codes", IEEE Trans. Inform. Theory, vol. IT-22, No. 2, pp. 138-147, March 1976.

[37] J.A. Van Der Horst, "Complete Decoding of Some Binary BCH Codes", Ph.D. Dissertation, Dept. Elec. Eng., Cornell Univ., Ithaca, N.Y., 1972.

[38] E.F. Assmus, Jr. and H.F. Mattson, Jr., "Some 3-Error-Correcting BCH Codes Have Covering Radius 5", IEEE Trans. Inform. Theory (Corresp.), vol. IT-22, No. 3, pp. 348-349, May 1976.

[39] T. Helleseth, "All Binary 3-Error-Correcting BCH Codes of Length 2^m-1 Have Covering Radius 5", IEEE Trans. Inform. Theory, vol. IT-24, No. 2, pp. 257-258, March 1978.

[40] D. Slepian, "A Class of Binary Signaling Alphabets", Bell Sys. Tech. J. vol. 35, pp. 203-234, Jan. 1956.

[41] A.B. Fontaine and W.W. Peterson, "Group Code Equivalence and Optimum Codes", IRE Trans. Inform. Theory, vol. IT-5, Special Supplement, pp. 60-70, 1959.

[42] C.F. Hobbs, "Approximating the Performance of a Binary Group Code", IEEE Trans. Inform. Theory, vol. IT-11, pp. 142-144, 1965.

[43] E.R. Berlekamp and L.R. Welch, "Weight Distributions of the Cosets of the (32, 6) Reed-Muller Code", IEEE Trans. Inform. Theory, vol. IT-18, No. 1, pp. 203-207, January 1972.

[44] N.J.A. Sloane and R.J. Dick, "On the Enumeration of Cosets of First Order Reed-Muller Codes", IEEE International Conf. on Communications, pp. 362-366, Montreal, 1971.

[45] T. Helleseth, T. Klove, and J. Mykkeltveit, "On the Covering Radius of Binary Codes", IEEE Trans. Inform. Theory, vol. IT-24, No. 5, pp. 627-628, September 1978.

[46] J.J. Mykkeltveit, "The Covering Radius of the (128, 8) Reed-Muller Code is 56", IEEE Trans. Inform. Theory, vol. IT-26, No. 3, pp. 359-362, May 1980.

[47] K. Brayer and O. Cardinale, "Evaluation of Error Correction Block Encoding for High-Speed HF Data", IEEE Trans Com. Tech., vol. COM-15, pp. 371-382, June 1967.

[48] G. Solomon and J.J. Stiffler, "Algebraically Punctured Cyclic Codes", Inform. and Control, vol. 8, pp. 170-179, 1965.

[49] H.J. Helgert and R.D. Stinaff, "Shortened BCH Codes", IEEE Trans. Inform. Theory (Corresp.), vol. IT-19, pp. 818-820, Nov. 1973.

[50] M. Goldberg, "Augmentation Techniques for a Class of Product Codes", IEEE Trans. Inform. Theory, vol. IT-19, No. 5, pp. 666-672, Sept. 1973.

[51] S.V. Kanetkar and V.K. Bhargava, "Weight Distribution of a Coset and the Outer Radius of a Code", Submitted to the 1982 IEEE Symposium on Information Theory, Les Arques, France. (Private Communication, November 24, 1981).

[52] F.J. MacWilliams and N.J.A. Sloane, "Pseudo-Random Sequences and Arrays", Proc. IEEE, vol. 64, No. 12, pp. 1715-1729, Dec. 1976.

[53] S.M. Johnson, "A New Upper Bound for Error-Correcting Codes", IEEE Trans. Inform. Theory, vol. IT-8, pp. 203-207, 1962.

[54] C.V. Freiman, "Upper Bounds for Fixed-Weight Codes of Specified Minimum Distance", IEEE Trans. Inform. Theory (Corresp.), vol IT-10, pp. 246-248, July 1964.

[55] E.R. Berlekamp, H. Rumsey and G. Solomon, "On the Solution of Algebraic Equations over Finite Fields", Inform. and Control, vol. 10, pp. 553-564, 1967.

[56] T. Kasami, "Weight Distributions of Bose-Chaudhuri-Hocquenghem Codes", in: R.C. Bose and T.A.-Dowling, eds., Combinatorial Math. and its Applications, Univ. of North Carolina Press, Chapel Hill, N.C., 1969.

[57] F.J. MacWilliams, "A Theorem on the Distribution of Weights in a Systematic Code", Bell System Tech. J. vol. 42, pp. 79-94, 1963.

[58] C.R.P. Hartmann, "Decoding Beyond the BCH Bound", IEEE Trans. Inform. Theory (Corresp.), vol. IT-18, pp. 441-444, May 1972.

[59] J.L. Massey, "Threshold Decoding", MIT Press, Cambridge, Mass., 1963.

[60] Z.A. Muscati and S.G.S. Shiva, "1-Step Majority-Logic Decoding with Two Rounds of Estimation", Proc. ICC '81, Denver, pp. 65.7.1-65.7.5, 1981.

[61] L.D. Rudolph and M.E. Mitchell, "Implementation of Decoders for Cyclic Codes", IEEE Trans. Inform. Theory, vol. IT-10, No. 3, pp. 259-260, 1964.

[62] H.D. Goldman et al, "The Weight Structure of some Bose-Chaudhuri Codes", IEEE Trans. Inform Theory (Corresp.), vol. IT-14, pp. 167-169, 1968.

[63] A. Nijenhuis and H.S. Wilf, "Combinatorial Algorithms for Computers and Calculators, 2nd Ed. New York, Academic Press, 1978.