INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

ERROR CORRECTING CAPABILITY OF 1-STEP MAJORITY LOGIC DECODING

by

Claude Laferriere

A thesis submitted to the School of Graduate Studies,
University of Ottawa, in partial fulfilment
of the requirements for the degree of
Master of Applied Science

Department of Electrical Engineering
Faculty of Science and Engineering
University of Ottawa
Ottawa, Canada

June, 1977

Acknowledgments

The author wishes to acknowledge gratefully the guidance and encouragement of his supervisor, Professor S.G.S. Shiva.

Thanks are due to all the members of the department of electrical engineering and to all the graduate students for providing a suitable atmosphere for research.

Finally, as it is customary to mention the work of the person who typed the manuscript, the author would like to extend his thanks to the ATS of the IBM 360/65 of the computer centre for its patient editing and typing of this thesis.

Abstract

One-step majority logic decoding corrects not only all error patterns of weight t or less but also some error patterns of weight greater than t. In this thesis an estimate of the number of error patterns of weight t+1, which 1-step MLD can correct, is given since they have a high probability of occurrence as compared to those of greater weights. General considerations on 1-step MLD are also included.

Table of Contents

INTRODUCTION

0.1 Early examples of coding

As very often pointed out by many, the need to communicate in a reliable fashion over long distances is one of the prerequisites for the existence of any organised civilisation. This need was felt even as early as in the Roman empire when civil servants had to control a vast territory from a central location. They would often send many messengers carrying the same message in the hope that at least one of them would make it. This is, one can say, one of the earliest uses of added redundancy in transmitting information. They were also faced with the dilemma of either sending many normal messengers or only one very strong man who could protect himself more adequately. This was in fact choosing between coding the message or simply increasing the transmitted power. So even in those days, there was a choice between 'to code or not to code'.

0.2 The coding problem

At the present time the problem is still pretty much

the same. Consider figure 0.1, where data in binary form is being transmitted through a channel to eventually reach a sink.

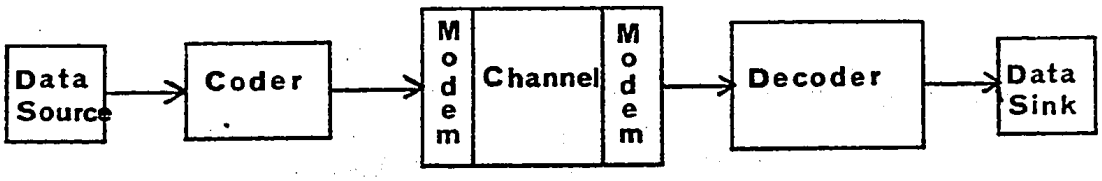


Figure 0.1, A communication system.

If the coder and the decoder are not used, the only way to improve the performance of the transmission is by increasing the power so that the signal will not be affected by noise present in the channel so much that it cannot be detected.The ratio Eb/No is the energy per information bit over the noise spectral density. The performance in the uncoded system can be obtained by evaluating how much of an Eb/No is required to acheive a given error probability.

Shannon [1] demonstrated that, for a hypothetical scheme operating at infinite bandwidth [2], the lower bound on Eb/No was -1.6 dB. He also showed that whenever the information rate R was less than the channel capacity

C, there existed some coding or modulation schemes which were guaranteeing a decoding error probability as low as desired. This thesis is mainly concerned with the use of coding techniques as a way of achieving the performance predicted by Shannon's theorems. These theorems only prove the existence of certain coding schemes but their practical implementation was never defined. This is, what has been called, the coding problem.

0.3 Definition of coding

Broadly speaking, coding is adding redundancy to a message so that, if corrupted by noise, it can be restored to the original by using the extra information provided by the coding scheme. In terms of figure 0.1, describing a communication system using coding, the coding gain is the reduction in Eb/No made possible by the use of coding. However, because of the need to add extra information to the message, the effect of coding will be to increase the rate of transmission. Therefore it would seem that, whenever bandwidth is available and power is scarce, the use of coding is a good way to achieve a satisfactory performance.

The way in which the extra information is added will determine many of the code characteristics. First of all, in order to perform error detection or error correction, codewords have to be at a certain distance of each other. This distance is guaranteed by the redundancy in the message and in the case of block codes is called the Hamming distance. Just to illustrate this point, the Hamming distance is the number of positions in which two words differ. Secondly, adding redundancy using a block structure will result in the creation of block codes. A coder for block coding is shown in figure 0.2. A block of k information bits is encoded into another block of n bits, where n is greater than k, allowing for the extra information.

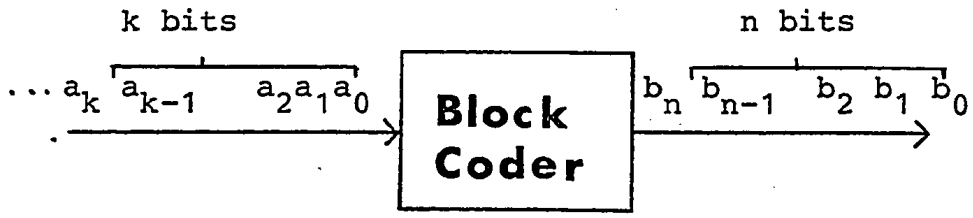


Figure 0.2, A block coder.

Another process is known as convolutional coding and is illustrated in figure 0.3. A convolutional encoder is

a linear sequential circuit consisting of a k-stage shift register and m mod-2 adders. The code is generated by feeding a continuous stream of information bits into the circuit which converts it to a sequence of binary code digits [3].

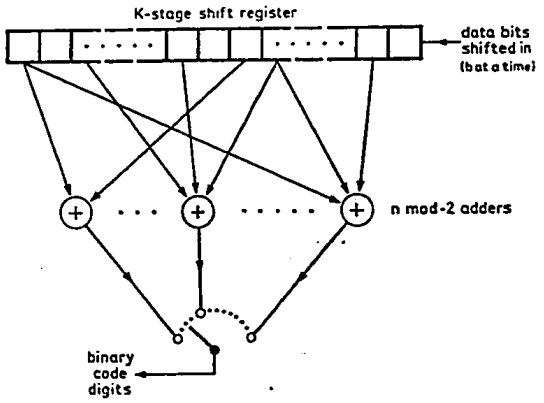


Figure 0.3, A convolutional coder.

Block and convolutional codes derive their error control capabilities from their minimum distance in the case of block codes or free distance for convolutional codes. A code ensuring a minimum distance d will be able to detect the presence of d-1 errors and also to correct the occurence of (d-1)/2 errors, although the decision to

detect or to correct has to be made beforehand. Obviously the detection capability of a code is twice as big as its correction power and this led to a scheme of error detection associated with retransmission of the incorrect data. The choice between detection and correction will be discussed next.

0.4 Detection vs correction

If, for a certain communication facility,

1) a feedback channel is available at reasonable cost and the delay introduced by a request and a retransmission is not too long,

2) the channel is not too noisy,

3) the errors are reasonnably bunched,

then ARQ (Automatic Repeat Request) system is feasible [4]. In an ARQ scheme, the data is arranged into blocks for transmission including a certain amount of redundancy to enable the receiver to detect blocks where errors

crept in. Then if any errors occured in the transmission, the system would send a request via the feedback link for a retransmission of the blocks in error.

If any one of the above conditions is violated, ARQ is not going to be a viable solution as the throughput of the system will be too low. It easy to visualise the implications of the above conditions. If the delay associated with the feedback link is too long, the size of the buffer memory at the sending end will be enormous. If the channel is too noisy, the scheme will spend most of its time retransmitting. On the other hand, if the errors are not reasonably bunched, the system will not operate efficiently. Consider the case where 200 blocks of 200 bits of data have been received with 200 errors. The ideal case for ARQ would be one block with 200 errors and 199 blocks error free. In real life though, a one-error-in-each-block situation might happen and would not be good for ARQ, hence the need for the errors to be bunched.

Nevertheless, when those conditions are met, ARQ performs very well, at a much higher rate than error-correction and is substantially error free [5]. This

would seem to indicate that in practical situations, ARQ would be preferable to error correction which is in general expensive to implement. However there exist situations where ARQ is impractical or unfeasible such as broadcast, deep space or one-way communication or data storage in a computer memory, because of the unavailability of a feedback link [5]. In these situations, error correction can be used advantageously to provide the required performance. Error correction can also be used in any schemes where the necessary conditions for ARQ are not met.

0.5 Error-correction

Error correction as opposed to error detection uses the redundancy in the message to correct errors which occured during transmission. The error correcting capability of a code is, as said before, linked with its minimum distance. In actual fact if a message is coded according to a scheme guaranteeing a minimun distance d(odd), then the error correcting capability t will be (d-1)/2. Although one would like a code to be able to correct as many error patterns as possible, practical error correction will aim at correcting the most likely types

of errors to occur in a particular channel. This is the reason why in order to cater for the various channels in use today, namely the HF and troposcatter radio, the microwave, the wireline and the satellite and deep space links, [6] that different coding and decodind schemes were designed.

The two types of errors that will be encountered on those channels are the random errors and the burst errors. Many block codes have been found for random error correction. For instance, the Hamming single error correcting codes [7], and also the BCH codes [8,9], just to name a few. These codes, if not too long, can be decoded by a method using the properties of the standard array [10], and for longer codes, by BCH decoding [11,12,4]. Furthermore one could use majority logic decoding [13], error trapping [14], or permutation decoding [15] for codes which are so decodable.

Unfortunately these codes have not been used extensively for random error correction on real channels because they were very often outperformed by random error correcting convolutional codes. Consider for example, the case of the satellite channel which can be accurately

modelled as white Gaussian, that is to say that errors due to noise are independent of each other. Random error correcting convolutional codes have been used for this channel in the Spade system.

In the Spade experiment, a coding scheme using threshold decoding and hard decisions, reduced the bit-error-rate from 10^{-4} to 10^{-8}, as confirmed by the Intelsat lll [16]. To illustrate the improvement obtained, take the case of the voice channel, delta modulated and transmitted at a rate of 28 Kbs. For an error-probability of 10^{-4}, a rate 1/2 convolutional code used in conjunction with hard decision and threshold decoding yielded a coding gain of 5 dB. Furthermore if the user is willing to pay more in terms of equipment, soft decisions and Viterbi decoding will give him an 8 dB of coding gain [16].

Although convolutional codes have been proved to outperform block codes over terrestrial and satellite links for error correction [6,17] , block codes have also been used in certain applications such as the mariner experiment in 1969. In the case of the space probe, a (32,26) biorthogonal Reed-Muller block code was used

succesfully [18].

In many instances, errors will occur in burst as is the case over most communication channels. Codes specially designed to handle burst correction will then have to be used. Among these are the Fire codes generated by a polynomial of the form $g(x)=P(x)(X^C+1)$ and with a burst error correcting capability b. P(x) will be a polynomial of degree $p \geq b$ and of exponent ε, and $c=2b-1$. It is also possible to use a high speed decoding algorithm which although less powerful than the standard one, is a lot faster and also detect the error-situations that the standard algorithm would have corrected. For an example of this, consider the Fire code generated by

$$g(x)=(x^{11}+1)(x^6+x+1)=x^{17}+x^{12}+x^{11}+x^6+x+1,$$

with n=693, b=6 and a decoding time of 693 cycles after calculation of the syndromes.

An alternative to that is the Fire code generated by

$$g(x)=(x^{11}+1)(x^3+x+1)(x^4+x+1),$$

with n=1155, b=3 and also correcting most of the burst errors of length 4,5,6, while detecting the rest of the burst errors of the same length, although not correcting them. The decoding time is 26 cycles plus a few multiplications [4].

Besides the Fire codes, the Reed-Solomon codes can be used to correct burst errors. They are a special case of the BCH codes and are generated by

$$g(x)=(x-\alpha^0)(x-\alpha^1)(x-\alpha^2).....(x-\alpha^{2t-1}),$$

where α is a primitive roots over $GF(2^m)$. The code can correct t symbols and has a length $n=2^m-1$ symbols or $m(2^m-1)$ bits so that ultimately the code can correct t in phase bursts of length m [4].

Apart from using codes specially designed for burst error correction, one can also use product codes or interleaved codes. The latter technique is well known among practising engineers and is reliable and easy to implement.

The majority of commonly used channels, as listed previously, are bursty. In fact the only channel

adequately modelled by the white Gaussian noise is the satellite and deep space link. Basically noise in a satellite communication link is primarily due to thermal activity at the receiver front end. Furthermore, other perturbations might be caused by uncertainty in the carrier phase at the demodulator and also by inaccuracies in the receiver AGC (Automatic gain control) [2]. All these contribute to the white Gaussian character of the satellite link.

Certain systems, using PSK (phase shift keying) as their modulation technique, were subjected to phase ambiguity because of PSK. Some solutions were found and among those was differential encoding, which uses the amount of phase shift to transmit information. At the receiving end, the data is recovered by comparing the difference of two consecutive phases. The problem with differential encoding is that, failure to detect one phase properly, will result in a burst error of length 2 as the next phase will be wrongly estimated, even though it might have been received correctly.

Various schemes of error correcting codes have been applied to TDMA (time division multiple access) satellite

communications [19]. Due to the bursty behaviour of differential encoding, it has been found that random error correcting convolutional codes, used in conjunction with N-symbol differential encoding, are best in terms of hardware required for the same performance as burst error correcting convolutional codes. N-symbol differential encoding is a method by which the inevitable second phase error will occur N symbols away from the first one, thus emulating code interleaving in the sense that the burst errors are reduced to random errors.

More specifically, for channels carrying voice encoded PCM (pulse code modulation), single error correcting Wyner-Ash convolutional codes, with N-symbol differential encoding, provides an adequate improvement. For digital data, self orthoganal codes are substituted to the Wyner-Ash codes, to acheive a better performance.

In general, convolutional codes are used more often than block codes over communication channels, because of the former's better rate and simpler implementation. This fact is also well illustrated by the availability of the hardware [20]. However there is one area in which block codes, due to their nature, have near complete supremacy

and that is error control in computer memories.

0.6 Error correction in computer memories

Modern computers are high speed machines and generally they are very sensitive to errors. That is to say, that if the programme instructions or the data words stored in memory are retreived erroneously, the operation of the computer might become erratic. Error correction is widely used to control those errors, both in off-line mass storage and on-line memories. Consider the cicuit of figure 0.4, which represents the CPU (central processing unit) of a digital computer, storing and retreiving information on a mass storage medium or off-line memory. This type of memory is considered first because it illustrates very good conditions for the application of coding.

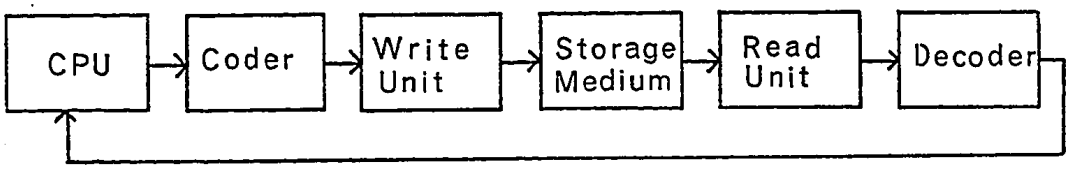


Figure 0.4, Data storage in computers.

Mass storage memories are mainly magnetic tapes, magnetic discs or photo digital memories. In the case of tapes or discs, errors are caused primarily by the non-uniformity of the coating oxide, usually spanning a few symbols and thus resulting in a burst error. Besides that, other causes of error are misalignement of read and write units and noise in the electronic circuitry. On the other hand, photo digital memories tend to have random errors in a proportion of 90% and the rest is mainly multiple burst errors.

Error correcting block codes have been applied to many systems [21,22]. Among these, is the magnetic tape system 2400 tape series of IBM 360 system, where a CRC (cyclic redundancy check) coupled with very little hardware is used to perform extensive error correction. Another example is a high speed disc file for the IBM 370 system. Disc files are subjected to burst errors and furthermore, although segmented in tracks like the tapes, tracks in a disk are independent of each other. The coding scheme has to handle long serial records, and fast decoding has to be acheived. For that particular case, Fire codes with a high speed decoding algorithm were used.

One of the most impressive result obtained by the use of coding is in the case of photo digital mass memory. In such a system data is recorded by an electron beam and read by an optical flying spot scanner. The system has a very large capacity and the error behaviour has already been mentioned earlier. Therefore, to correct both random and multiple burst errors present in the memory, Reed-Solomon codes were used. The decoding algorithm was implemented in software and included a single error and multiple error correction routines. As the data is read in, a check is made to determine whether the block is in error or not. If in error, the computer will perform a single error correction routine and recheck to see if there is still some errors left. This procedure of single error correction and check is applied several times over. Then, if to no avail, the computer switches to a much slower multiple error correction routine. It is interresting to compare the speed of these decoding algorithms. A single-error correction routine is executed in 0.34 ms as compared to 3.5 ms and 16 ms for double and triple error correction.

Because of this flexibility, the actual throughput of the memory is slowed down by only 0.7%. The increase

in reliability gained by the system is impresive. Without coding, 7 bits out of a thousand would be in error. With coding however, only one line out of 2.14 million lines of 378 bits, would contain non-decoded errors [21].

Besides mass storage memories, there also exists the on-line core memory. Until very recently, core memory did not use error correction, as the cost of magnetic ferrite was high. Recent development in RAM (random access memory) semiconductor memory chips made them in many respects a more interresting alternative than core memory for on-line applications.

In this type of memory, packing density as high as 16 Kbits are possible and due to the random nature of the defects occuring on the substrate, these memories are subjected to random errors. An error caused by a defect in a memory cell is permanent and is called a hard error. On the other hand, errors might originate from a false reading of the sense amplifier (figure 0.5) and very often those errors will disappear at a second reading, hence they are soft errors.

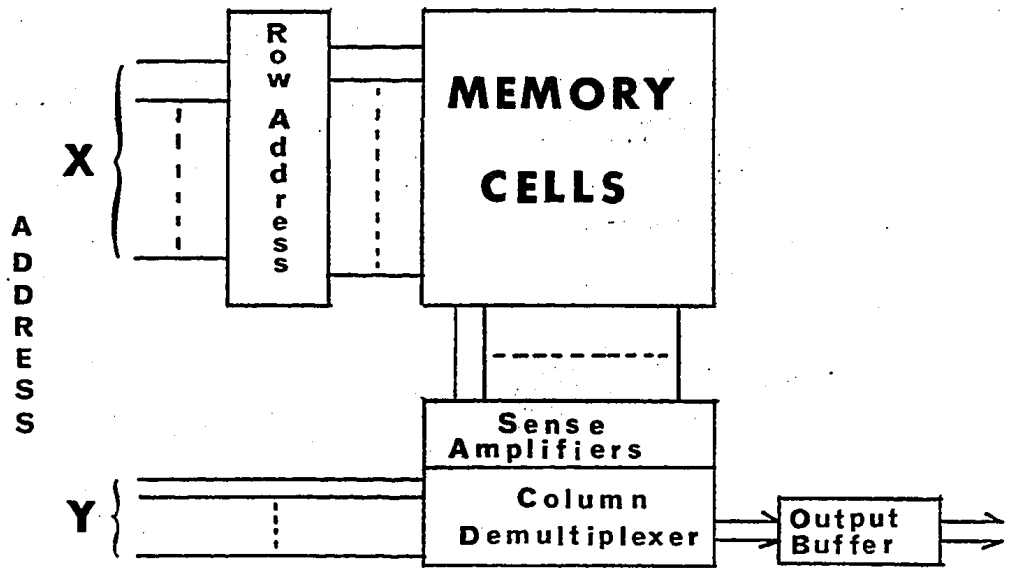


Figure 0.5, A simplified RAM block diagram.

At high density, semiconductor memories will be of a single bit architecture,(e.g. 4 Kbits x 1, or 16 Kbits x 1). For that reason, in order to obtain the desired word length, several chips have to be used in parallel, as shown in figure 0.6. This type of memory lends itself very well to the use of block coding as more chips are easily added. The kind of error one has to cater for is mainly random errors and the use of Hamming single error correcting codes has proved itself adequate for error control [23].

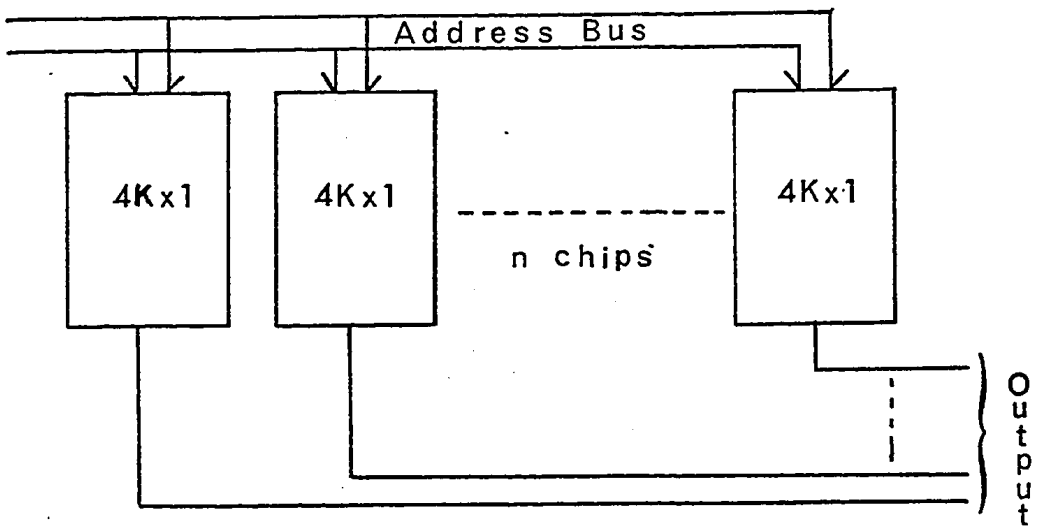


Figure 0.6, A schematic of a 4 Kbit x n memory.

Error correction could be implemented in several ways into a memory system. Figure 0.7 depicts an arrangement that has been proved successful in computer applications.

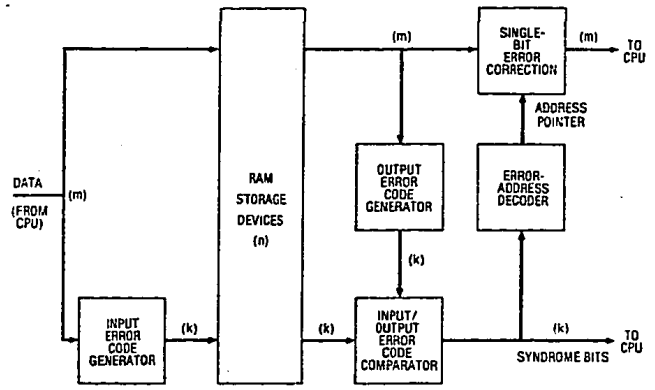


Figure 0.7, Error-correction in semiconductor memory.

In figure 0.7, the data from the CPU is encoded and is fed into RAM memory. After recovery, the parity bits are regenerated and then compared with the stored ones. If there is any disagreement, the single error position is determined and the error is corrected. The syndrome is also given to the CPU which can find out if any one chip is defective in too many positions.

Finally, to justify the use of coding on such memories [23], suffice to say that 4Kbits by 16 bits memory using 1K RAM, with a device failure rate of 0.1% per thousand hours, would have an MTBF (mean time between

failure) of 15,600 hours. However, as the size of the memory increases, the MTBF gets smaller. A 256 Kbit by 64 bit memory using the same 1Kbit chips would have an MTBF of 64 hours, making coding or some other forms of error control just about mandatory.

0.7 Concluding remarks

To sum up briefly, coding, which was until a few years ago a purely theoretical subject, gained in importance as more and more communication facilities implemented some coding schemes. This growth of coding had been foreseen by J.J. Stiffler [24] when he pointed out that with the introduction of large scale integrated circuits, better decoding algorithms and a better understanding of actual channels, coding would be a competitive method to control errors.

This then completes this brief review of some error correcting codes and certain of their applications. A thesis outline is to follow next.

0.8 Thesis outline

This thesis mainly deals with MLD (majority logic decoding) although some chapters are devoted to subjects that arose from the research work. Among these is the decoding error probability of shortened codes which is covered in chapter 1. A few theorems on that topic are given, as well as graphs of the decoding error probability for different amounts of shortening.

The main part of the thesis starts with chapter 2, with a review of MLD codes and decoding techniques. Next is chapter 3 where the MLD algorithm is studied in detail. Also investigated is the extra error correcting capability inherent to the scheme, followed by an estimate on how many error patterns of weight t+1 MLD can correct. A condensed version of section 3.2 and 3.3 has also been accepted for publication [43].

In chapter 4 the practical feasibility of building a microprocessor-based coder-decoder is investigated. Finally chapter 5 contains the concluding remarks together with suggestions for further work with MLD.

Chapter 1

Decoding error probability of shortened codes

1.1 Introduction

As seen in the previous section, error correcting codes are used to improve on the overall error probability. It is often desirable for the system designer to minimise this probability and if coding is used, it is reasonable to expect that the shortening of the code will help in that respect.

In this chapter, the expression used to describe the decoding error probability is stated, followed by a few comments on shortening. Thereafter three theorems characterising the behaviour of the difference of decoding error probability between a shortened code and its parent code are introduced. Finally, graphs of different probabilities and improvements together with some remarks are included, thus completing the first chapter.

1.2 Decoding error probability

With reference to figure 0.1, describing a communication channel where coding is used, the binary data from the source will be encoded according to a scheme guaranteeing a minimum distance d. At the other end, the decoder will execute a BDD (bounded distance decoding) algorithm which means that all error patterns of weight t or less will be corrected. Any error pattern of weight greater than t will be considered as not correctable and the output of the decoder will be assumed to be wrong even in the case where the decoder does not give an answer, indicating error detection.

In such a scheme, the probability of obtaining a correctly decoded result at the output of the decoder can be expressed as

$$P_c(n,t,p) = (1-p)^n \sum_{i=0}^{t} \binom{n}{i} \beta^i, \qquad (1.1a)$$

$$\beta = p/(1-p), \quad 0 \le p \le 0.5,$$

where n is the code block length, t the error correcting capability and p the channel cross over probability. The decoding error probability is

$$P_e(n,t,p)=1-P_c(n,t,p) \qquad (1.1b)$$

but for ease of writing $P_c(n,t,p)$ will be used throughout the first chapter.

1.3 Code shortening

A codeword V(x) belonging to a linear code V can usually be expressed as

$$V(x)=g(x)\ C(x), \qquad (1.2)$$

where g(x) is the generator polynomial and C(x) the information polynomial. Equation 1.2 represents a non-systematic encoding. The codeword has a length of n bits and a maximum degree of n-1. The generator polynomial has a degree m=n-k and C(x) has a degree of k-1 or less.

Here shortening means restraining the degree of C(x) to a value less than k-1 resulting in a shorter block length. This is not however the only type of shortening possible.

1.4 Some theoretical results

In this section three theorems and two corollaries will be introduced. The first theorem to be presented formally proves that shortening a code increases its probability of correct decoding.

Theorem 1.1

Given an (n-s,k-s,t) code, s<k, obtained by shortening an (n,k,t) code,

$$P_c(n-s,t,p) > P_c(n,t,p). \tag{1.3a}$$

Proof:

Without losing generality, s can be assumed to be equal to 1, so that if $P_c(n-1,t,p)$ is proved to be greater than $P_c(n,t,p)$, then by extension $P_c(n-s,t,p)$ is also greater than $P_c(n,t,p)$.

The statement of the theorem is

$$P_c(n-1,t,p) > P_c(n,t,p), \tag{1.3b}$$

which could be rewritten as

$$\frac{P_c(n,t,p)}{P_c(n-1,t,p)} < 1. \tag{1.4}$$

Expanding will yield

$$(1-p)\ \frac{\sum_{i=0}^{t}\binom{n}{i}\beta^i}{\sum_{i=0}^{t}\binom{n-1}{i}\beta^i} < 1. \tag{1.5}$$

It is known that

$$\binom{n}{i} = \binom{n-1}{i-1} + \binom{n-1}{i}, \tag{1.6}$$

so that (1.5) can be expressed as

$$(1-p)\ \frac{\sum_{i=1}^{t}\binom{n-1}{i-1}\beta^i + \sum_{i=0}^{t}\binom{n-1}{i}\beta^i}{\sum_{i=0}^{t}\binom{n-1}{i}\beta^i} < 1\ , \tag{1.8}$$

or

$$(1-p)\left[\frac{\sum_{i=1}^{t}\binom{n-1}{i-1}\beta^{i}}{\sum_{i=0}^{t}\binom{n-1}{i}\beta^{i}}+1\right]<1, \tag{1.8}$$

or

$$(1-p)\left[1+\frac{\beta\sum_{i=0}^{t-1}\binom{n-1}{i}\beta^{i}}{\sum_{i=0}^{t}\binom{n-1}{i}\beta^{i}}\right]<1\ , \tag{1.9}$$

which can be finally reduced to,

$$-p\left[1-\frac{\sum_{i=0}^{t-1}\binom{n-1}{i}\beta^{i}}{\sum_{i=0}^{t}\binom{n-1}{i}\beta^{i}}\right]<0\ , \tag{1.10}$$

or

$$\frac{\sum_{i=0}^{t-1}\binom{n-1}{i}\beta^i}{\sum_{i=0}^{t}\binom{n-1}{i}\beta^i} < 1 , \qquad (1.11)$$

which is true if $\beta>0$, therefore proving (1.3b) and by extension (1.3a).

The second theorem, theorem 1.2, evaluates the gain in probability of correct decoding when a code is shortened by s bits. It is followed by corollary 1.1 which defines the improvement gained with respect to the parent code, in terms of decoded probability, when shortening is used.

Theorem 1.2

Given an (n-s,k-s,t) code, s<k, obtained by shortening an (n,k,t) code,

$$P_c(n-s,t,p)-P_c(n,t,p)= (1-p)^{n-s+1}\ \beta^{t+1}\sum_{i=1}^{s}(1-p)^{s-i}\binom{n-i}{t} . \qquad (1.12)$$

Proof:

Theorem 1.1 was proved for (n-1,n) and then generalised for (n-s,n). This proof is making use of the same argument and therefore the first step will be to establish that

$$P_c(n-1,t,p)-P_c(n,t,p)= (1-p)^{n-t-1}\ p^{t+1}\ \binom{n-1}{t}. \quad (1.13)$$

In the first step, t will be fixed equal to 1 so that

$$P_c(n-1,1,p)-P_c(n,1,p)=(1-p)^{n-1}+(1-p)^{n-2}\ p\binom{n-1}{1}-$$

$$-\left[(1-p)^n+(1-p)^{n-1}\ p\binom{n}{1}\right]. \quad (1.14)$$

Factoring out this expression will give

$$=(1-p)^{n-2}\left[(1-p)+\ p\binom{n-1}{1}-(1-p)^2-(1-p)p\binom{n}{1}\right], \quad (1.15)$$

or

$$=(1-p)^{n-2}\left[1-p+p\binom{n-1}{1}-1+2p-p^2-p\binom{n}{1}+p^2\binom{n}{1}\right]. \quad (1.16)$$

Using the following fact that

$$\binom{n}{i} = \binom{n-1}{i-1} + \binom{n-1}{i}, \quad (1.17)$$

it is possible to rewrite (1.16) as

$$= (1-p)^{n-2}\left[p\binom{n-1}{1} - p^2 + p - p - p\binom{n-1}{1} + p^2 + p^2\binom{n-1}{1}\right], \quad (1.18)$$

which is

$$= (1-p)^{n-2}\ p^2\ \binom{n-1}{1}. \quad (1.19)$$

Then in the second step, t is made equal to 2 so that

$$P_c(n-1,2,p) - P_c(n,2,p) = (1-p)^{n-1} + (1-p)^{n-2}\ p\binom{n-1}{1} +$$

$$(1-p)^{n-3}\ p^2\binom{n-1}{2} - \left[(1-p)^n + p(1-p)^{n-1}\binom{n}{1} + p^2(1-p)^{n-2}\binom{n}{2}\right]. \quad (1.20)$$

or

$$= p^2(1-p)^{n-2}\ \binom{n-1}{1} + \left[p^2(1-p)^{n-3}\binom{n-1}{2} - p^2(1-p)^{n-2}\binom{n}{2}\right]. \quad (1.21)$$

Factoring this expression will give

$$= (1-p)^{n-3}\left[(1-p)p^2\binom{n-1}{1} + p^2\binom{n-1}{2} - (1-p)p^2\binom{n}{2}\right], \quad (1.22)$$

and by using (1.17) it is possible to write

$$= (1-p)^{n-3}\left[(1-p)p^2\binom{n-1}{1} + p^2\binom{n-1}{2} - (1-p)p^2\binom{n-1}{1} - (1-p)p^2\binom{n-1}{2}\right], \quad (1.23)$$

or

$$= (1-p)^{n-3}\left[p^2\binom{n-1}{2} - p^2\binom{n-1}{2} + p^3\binom{n-1}{2}\right], \quad (1.24)$$

which is

$$= (1-p)^{n-3}\ p^3 \binom{n-1}{2} . \qquad (1.25)$$

Finally induction is applied to get an answer independent of t. Assuming that for some t,

$$P_c(n-1,t,p)-P_c(n,t,p)=(1-p)^{n-t-1}\ p^{t+1} \binom{n-1}{t} , \qquad (1.13)$$

then for t+1,

$$P_c(n-1,t+1,p)-P_c(n,t+1,p)=\left[(1-p)^{n-1}+(1-p)^{n-2}\ p\binom{n-1}{1} + \right.$$
$$\left. \ldots+(1-p)^{n-t-1}\ p^t\binom{n-1}{t}+(1-p)^{n-t-2}\ p^{t+1}\binom{n-1}{t+1} \right] - \left[(1-p)^n \right.$$
$$\left. +(1-p)^{n-1}\ p\binom{n}{1} +\ldots+ (1-p)^{n-t}\ p^t \binom{n}{t} + (1-p)^{n-t-1}\ p^{t+1}\binom{n}{t+1}\right] , \qquad (1.26)$$

which is

$$=p^{t+1}(1-p)^{n-t-1}\binom{n-1}{t}+p^{t+1}(1-p)^{n-t-2}\binom{n-1}{t+1} -p^{t+1}(1-p)^{n-t-1}\binom{n}{t+1} . \qquad (1.27)$$

Factoring out this expression will give

$$=p^{t+1}(1-p)^{n-t-2}\left[(1-p)\binom{n-1}{t}+\binom{n-1}{t+1}- (1-p)\binom{n}{t+1}\right], \qquad (1.28)$$

and using (1.17) it is possible to write

$$=(1-p)^{n-t-2}\ p^{t+1}\left\{(1-p)\binom{n-1}{t}+\binom{n-1}{t+1}-(1-p)\left[\binom{n-1}{t+1}+\binom{n-1}{t}\right]\right\}, \qquad (1.29)$$

or finally

$$= (1-p)^{n-t-2}\ p^{t+2}\ \binom{n-1}{t+1}\ , \qquad (1.30)$$

therefore proving that for any t's,

$$P_c(n-1,t,p) - P_c(n,t,p) = (1-p)^{n-t-1}\ p^{t+1}\ \binom{n-1}{t}\ . \qquad (1.13)$$

The above result is for (n-1,n), and at this point it would be interesting to generalise it for (n-s,n). Adding up all the differences yields

$$P_c(n-s,t,p) - P_c(n,t,p) = \left[P_c(n-s,t,p) - P_c(n-s-1,t,p)\right] + ..$$

$$..+\left[P_c(n-1,t,p) - P_c(n,t,p)\right]\ , \qquad (1.31)$$

which can also be expressed as

$$P_c(n-s,t,p) - P_c(n,t,p) = (1-p)^{n-s+1}\ \beta^{t+1}\left[(1-p)^{s-1}\binom{n-1}{t} + \right.$$

$$\left.(1-p)^{s-2}\binom{n-2}{t} + ...+\binom{n-s}{t}\right]\ , \qquad (1.12)$$

or

$$= (1-p)^{n-s+1}\ \beta^{t+1}\ \sum_{i=1}^{s} (1-p)^{s-i}\ \binom{n-i}{t}\ . \qquad (1.12)$$

Using theorem 1.2 it is now possible to evaluate the improvement gained by shortening an (n,k,t) code by s

bits. This is given in corollary 1.1.

Corollary 1.1

Given an (n-s,k-s,t) code, s<k, and its (n,k,t) parent code, the improvement I is

$$I \triangleq \frac{\left[P_c(n-s,t,p)-P_c(n,t,p)\right]}{P_c(n,t,p)} , \qquad (1.32)$$

which can be expressed as

$$I \triangleq \frac{\beta^{t+1}}{(1-p)^{s-1}} \frac{\sum_{i=1}^{s} (1-p)^{s-i} \binom{n-i}{t}}{\sum_{i=0}^{t} \binom{n}{i} \beta^i} . \qquad (1.33)$$

Corollary 1.2 and theorem 1.3 conclude this section. The former proves that as one keeps on shortening, the improvement will increase in value whereas the latter says that if two codes of the same length but with different error correcting capabilities are shortened, the improvement will be greater for the code with the smallest t.

Corollary 1.2

Given an (n-s-1,k-s-1,t) and an (n-s,k-s,t) codes derived from the (n,k,t) parent code,

$$I_{s+1} > I_s \tag{1.34}$$

Proof:

The corollary can be expressed as

$$\frac{\left[P_c(n-s-1,t,p)-P_c(n,t,p)\right]}{P_c(n,t,p)} > \frac{\left[P_c(n-s,t,p)-P_c(n,t,p)\right]}{P_c(n,t,p)}, \tag{1.35}$$

which can be further reduced to

$$\frac{P_c(n-s-1,t,p)}{P_c(n,t,p)} > \frac{P_c(n-s,t,p)}{P_c(n,t,p)} \quad . \tag{1.36}$$

From theorem 1.1, it is clear that $P_c(n-s-1,t,p) > P_c(n-s,t,p)$, implying that $I_{s+1} > I_s$.

Theorem 1.3

Given an (n-s,k-s t),s<k, and an (n-s, k'-s,t+1), s< k' codes and their respective (n,k,t) and (n,k',t+1) parent codes,

$$I_t > I_{t+1} . \tag{1.37}$$

Proof:

Here again, (1.37) can be expressed as

$$\frac{\left[P_c(n-s,t,p)-P_c(n,t,p)\right]}{P_c(n,t,p)} > \frac{\left[P_c(n-s,t+1,p)-P_c(n,t+1,p)\right]}{P_c(n,t+1,p)} . \tag{1.38}$$

This could be rewritten as

$$\frac{P_c(n-s,t,p)}{P_c(n,t,p)} > \frac{P_c(n-s,t+1,p)}{P_c(n,t+1,p)} , \tag{1.39}$$

or

$$\left[\frac{(1-p)^{n-s}}{(1-p)^n}\right]\left[\frac{\sum_{i=0}^{t}\binom{n-s}{i}\beta^i}{\sum_{i=0}^{t}\binom{n}{i}\beta^i}\right] > \left[\frac{(1-p)^{n-s}}{(1-p)^n}\right]\left[\frac{\sum_{i=0}^{t+1}\binom{n-s}{i}\beta^i}{\sum_{i=0}^{t+1}\binom{n}{i}\beta^i}\right]. \tag{1.40}$$

Reducing the inequality yields

$$\frac{\sum_{i=0}^{t}\binom{n-s}{i}\beta^{i}}{\sum_{i=0}^{t}\binom{n}{i}\beta^{i}} > \frac{\sum_{i=0}^{t+1}\binom{n-s}{i}\beta^{i}}{\sum_{i=0}^{t+1}\binom{n}{i}\beta^{i}}, \tag{1.41}$$

which can be expressed as

$$\frac{A}{B} > \frac{A + \binom{n-s}{t+1}\beta^{t+1}}{B + \binom{n}{t+1}\beta^{t+1}}, \tag{1.42}$$

or

$$\frac{\binom{n}{t+1}\beta^{t+1}}{B} > \frac{\binom{n-s}{t+1}\beta^{t+1}}{A}, \tag{1.43}$$

or

$$\frac{\binom{n}{t+1}}{\binom{n-s}{t+1}} - \frac{B}{A} > 0, \tag{1.44}$$

or replacing,

$$\frac{\binom{n}{t+1}}{\binom{n-s}{t+1}} \sum_{i=0}^{t}\binom{n-s}{i}\beta^{i} - \sum_{i=0}^{t}\binom{n}{i}\beta^{i} > 0. \tag{1.45}$$

Rearranging (1.45) by grouping everything under the summation sign yields

$$\sum_{i=0}^{t}\left\{\left[\frac{\binom{n}{t+1}\binom{n-s}{i}}{\binom{n-s}{t+1}}\right]-\binom{n}{i}\right\}\beta^{i}>0\ . \tag{1.46}$$

β is always greater than zero, and so is β^{i}. (1.46) will hold if all the coefficients of β^{i}'s, $0\leq i\leq t$, are greater than zero, so that it possible to write

$$\binom{n}{t+1}\cdot\binom{n-s}{i}>\binom{n}{i}\cdot\binom{n-s}{t+1}\ , \tag{1.47}$$

which is

$$\frac{n!}{(n-t-1)!\,(t+1)!}\cdot\frac{(n-s)!}{(n-s-i)!\;i!}>\frac{n!}{(n-i)!\;i!}\cdot\frac{(n-s)!}{(n-s-t-1)!\,(t+1)!}\ , \tag{1.48}$$

or

$$\frac{(n-i)!}{(n-s-i)!}>\frac{(n-t-1)!}{(n-t-1-s)!}\ , \tag{1.49}$$

or

$$(n-i)(n-i-1)\ldots(n-i-s+1)>(n-t-1)(n-t-2)\ldots(n-t-s)\ . \tag{1.50}$$

Comparing term by term will yield, as $0\leq i\leq t$, and as there are s terms in both sides,

$$
\begin{aligned}
&(n-i) > (n-t-1) \ , \\
&(n-i-1) > (n-t-2) \ , \\
&\quad \vdots \qquad\quad \vdots \\
&(n-i-s+1) > (n-t-s) \ ,
\end{aligned}
\tag{1.51}
$$

proving (1.50) and also (1.37).

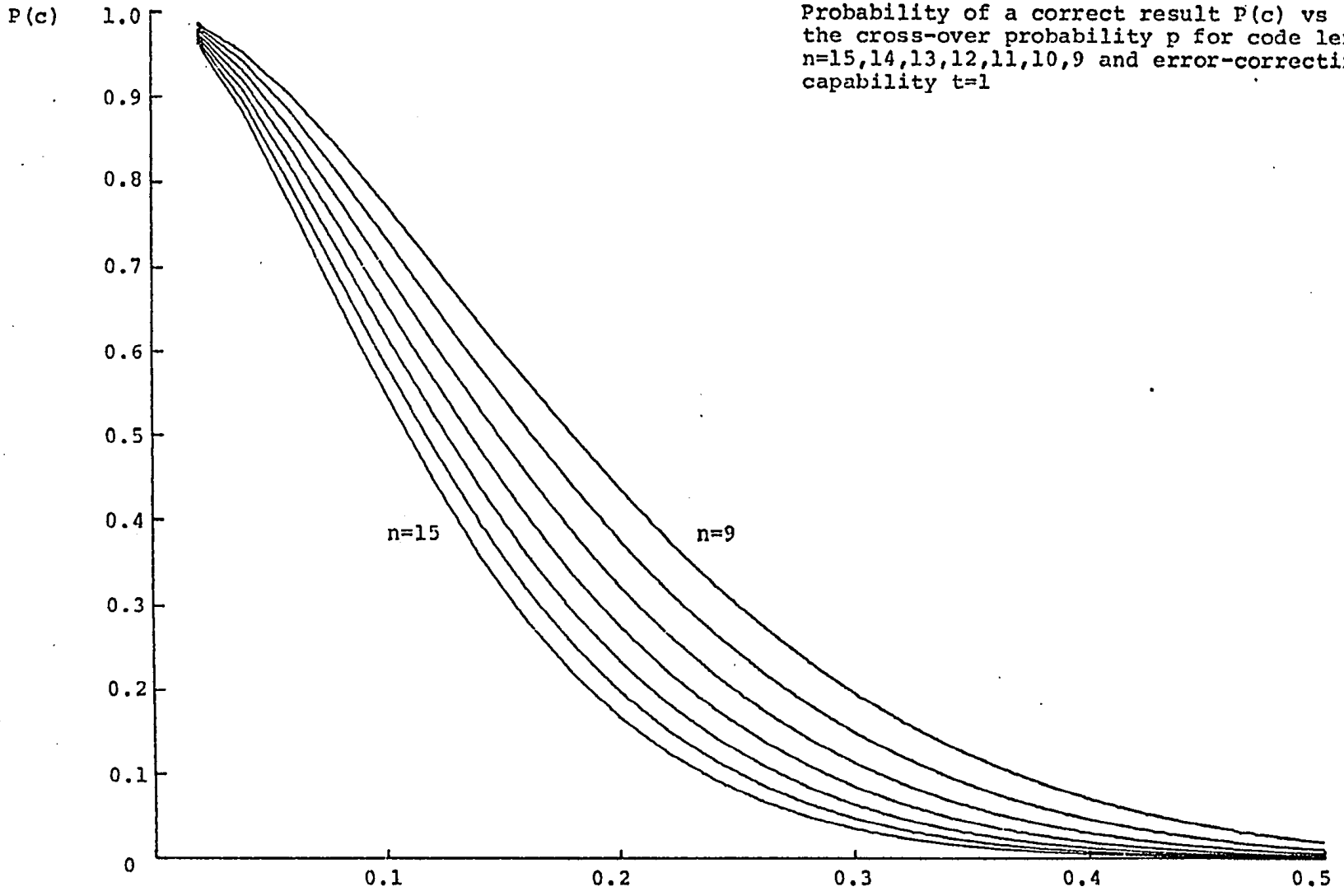
1.5 Remarks on the improvement

The improvement as defined in corollary 1.1 increases in value as one keeps on shortening the code. However its relative value, i.e. $I_s - I_{s-1}$, does not necessarily keep on increasing with s, (figure 1.7 to 1.12).

When codes of the same length but with different error correcting capabilities are used, the most sizable improvement is for the code with the smallest t, although the difference in improvement seems to be very much dependent on the block length, (figure 1.13 to 1.15).

Finally, the results presented here in this chapter might be interesting from a theoretical point view but

are in fact disappointing if one is looking at shortening for improving the error probability of a system exhibiting a fairly good signal to noise ratio, (figure 1.1 to 1.6). Therefore the practical use of shortening for decreasing the decoding error probability is limited.



Probability of a correct result P(c) vs the cross-over probability p for code lengths n=15,14,13,12,11,10,9 and error-correcting capability t=1

Figure 1.1.

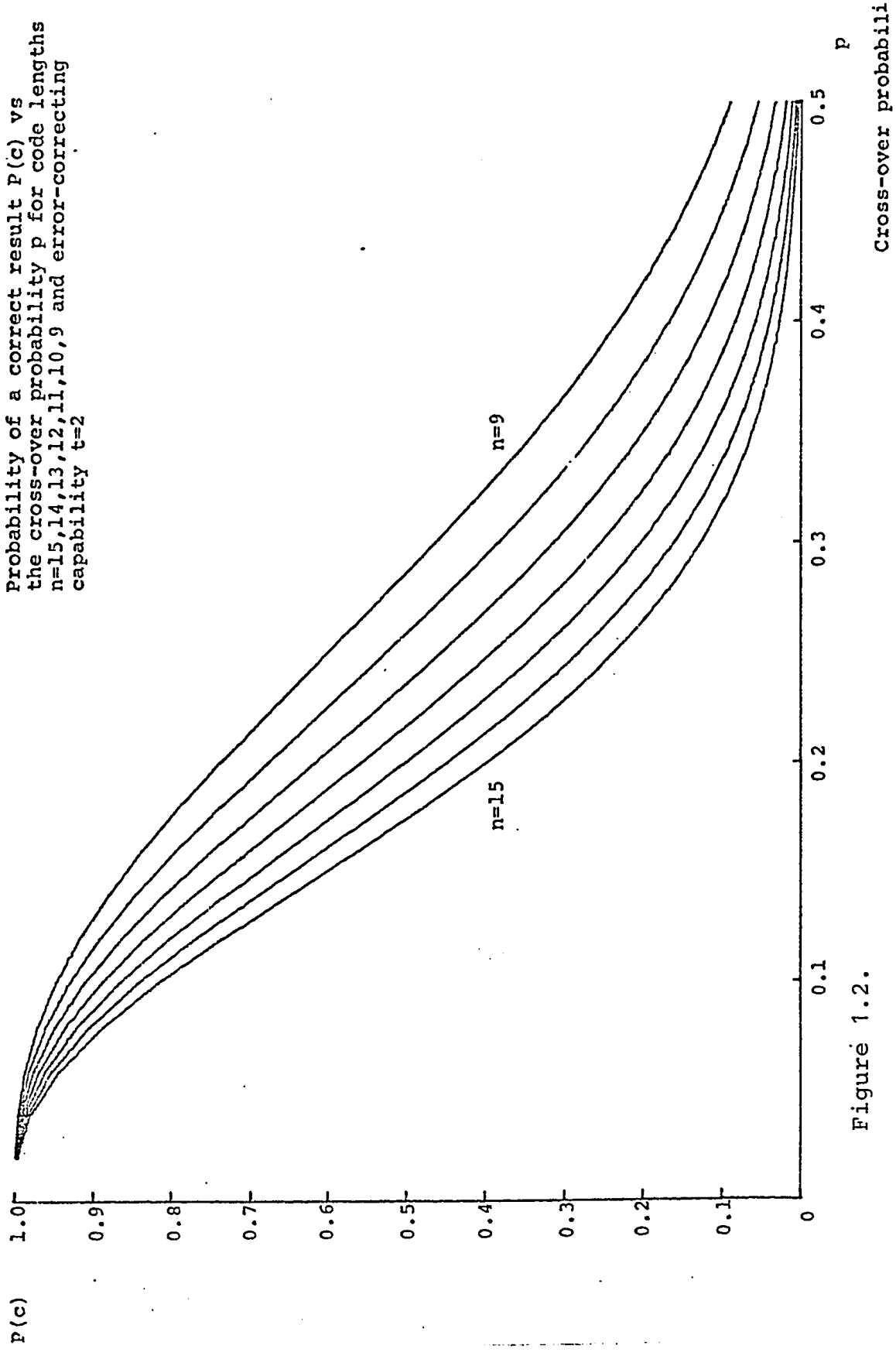

Probability of a correct result P(c) vs
the cross-over probability p for code lengths
n=15,14,13,12,11,10,9 and error-correcting
capability t=2
P(c)
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
n=9
n=15
0.1
0.2
0.3
0.4
0.5
p
Cross-over probability

Figure 1.2.

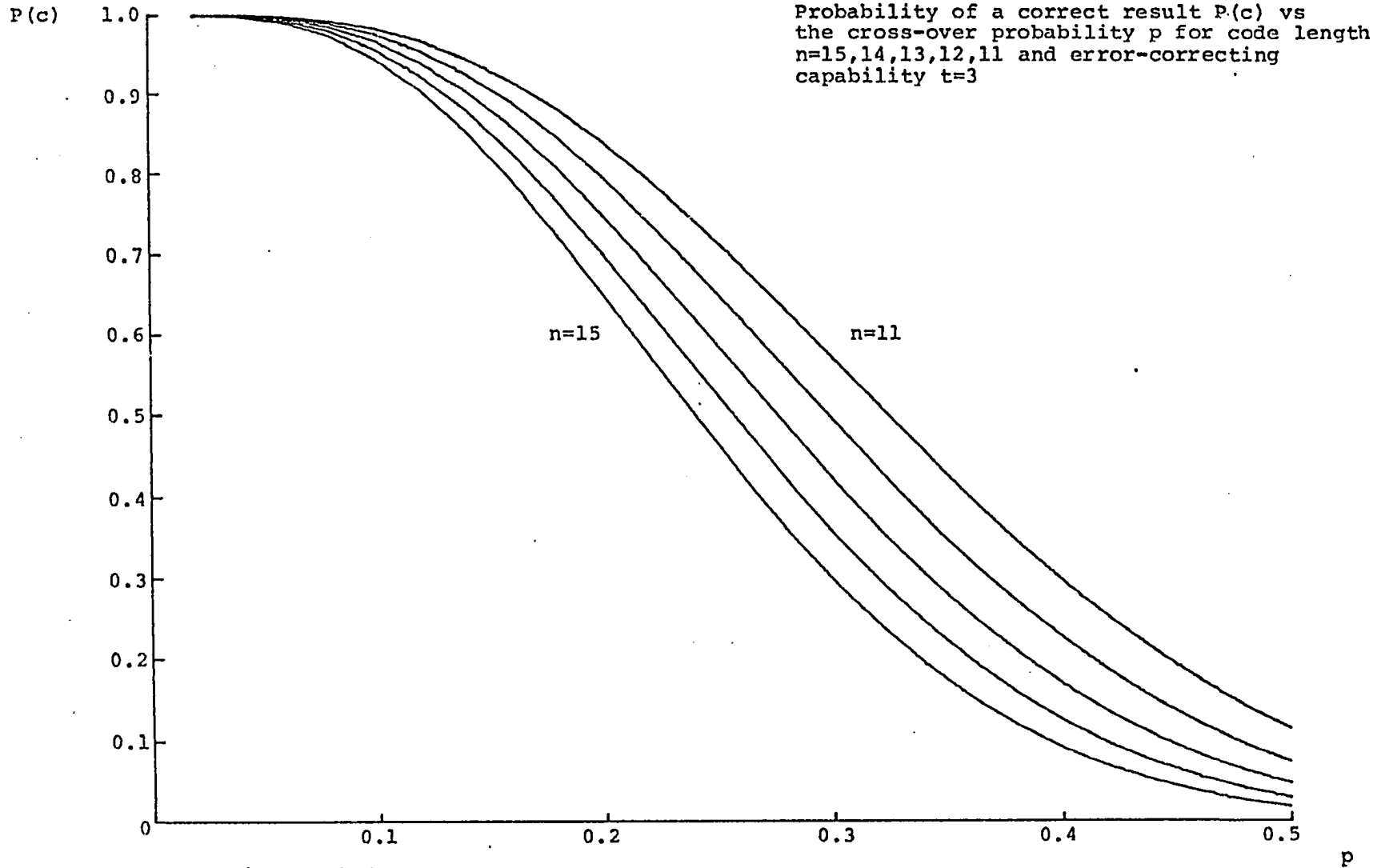

P(c)
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.1
0.2
0.3
0.4
0.5
p
n=15
n=11
Probability of a correct result P(c) vs
the cross-over probability p for code lengths
n=15,14,13,12,11 and error-correcting
capability t=3
Cross-over probability

Figure 1.3

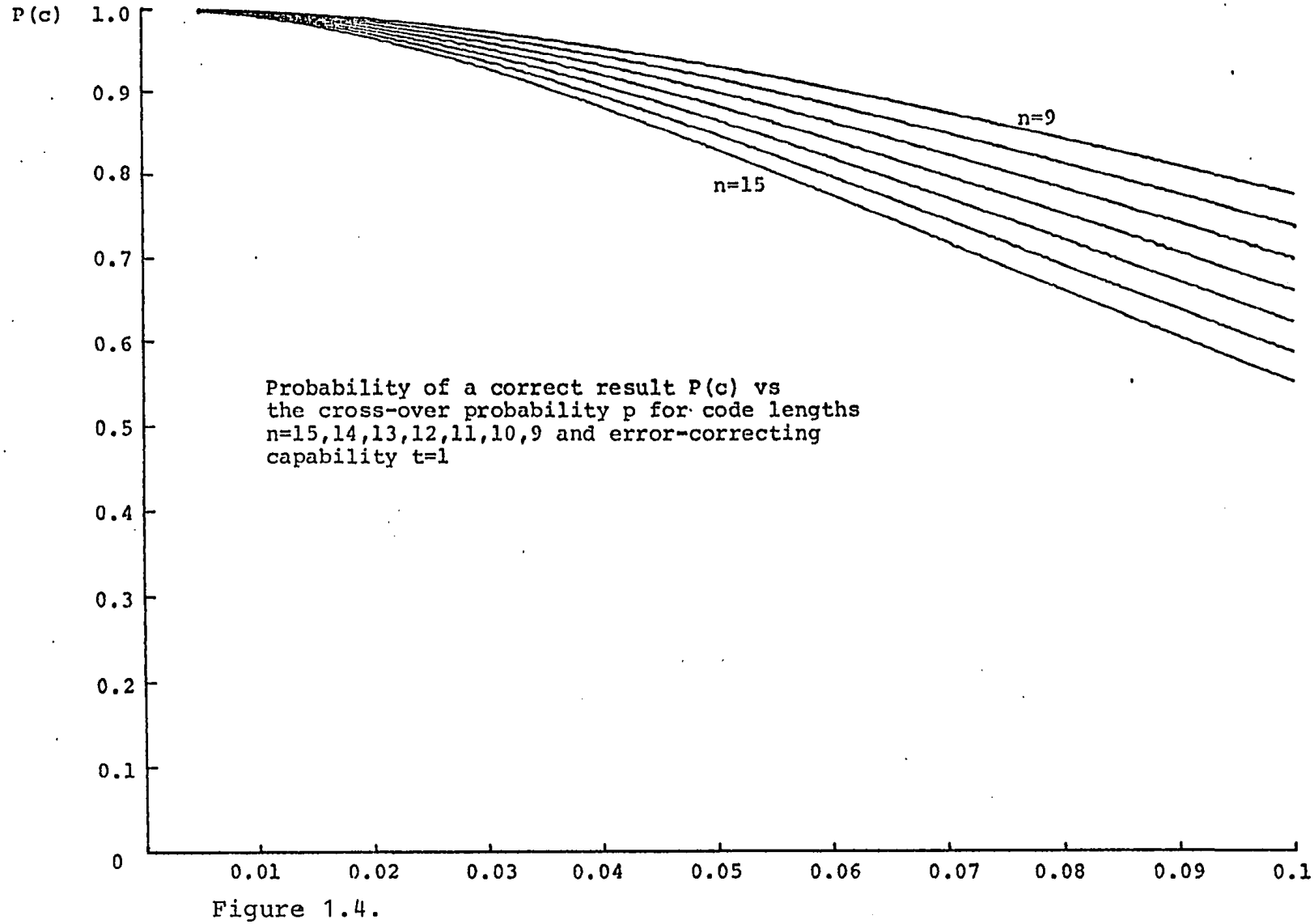
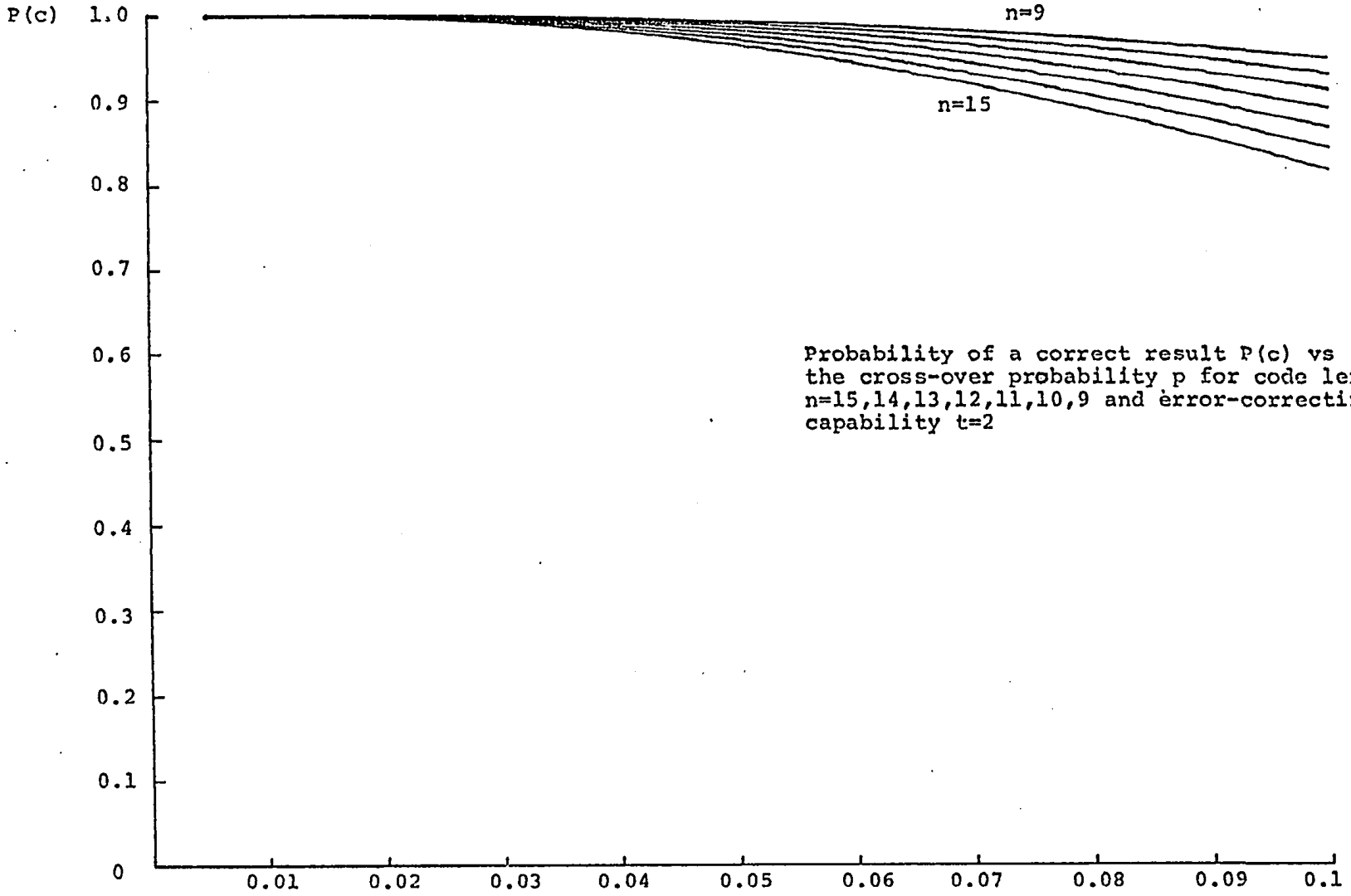
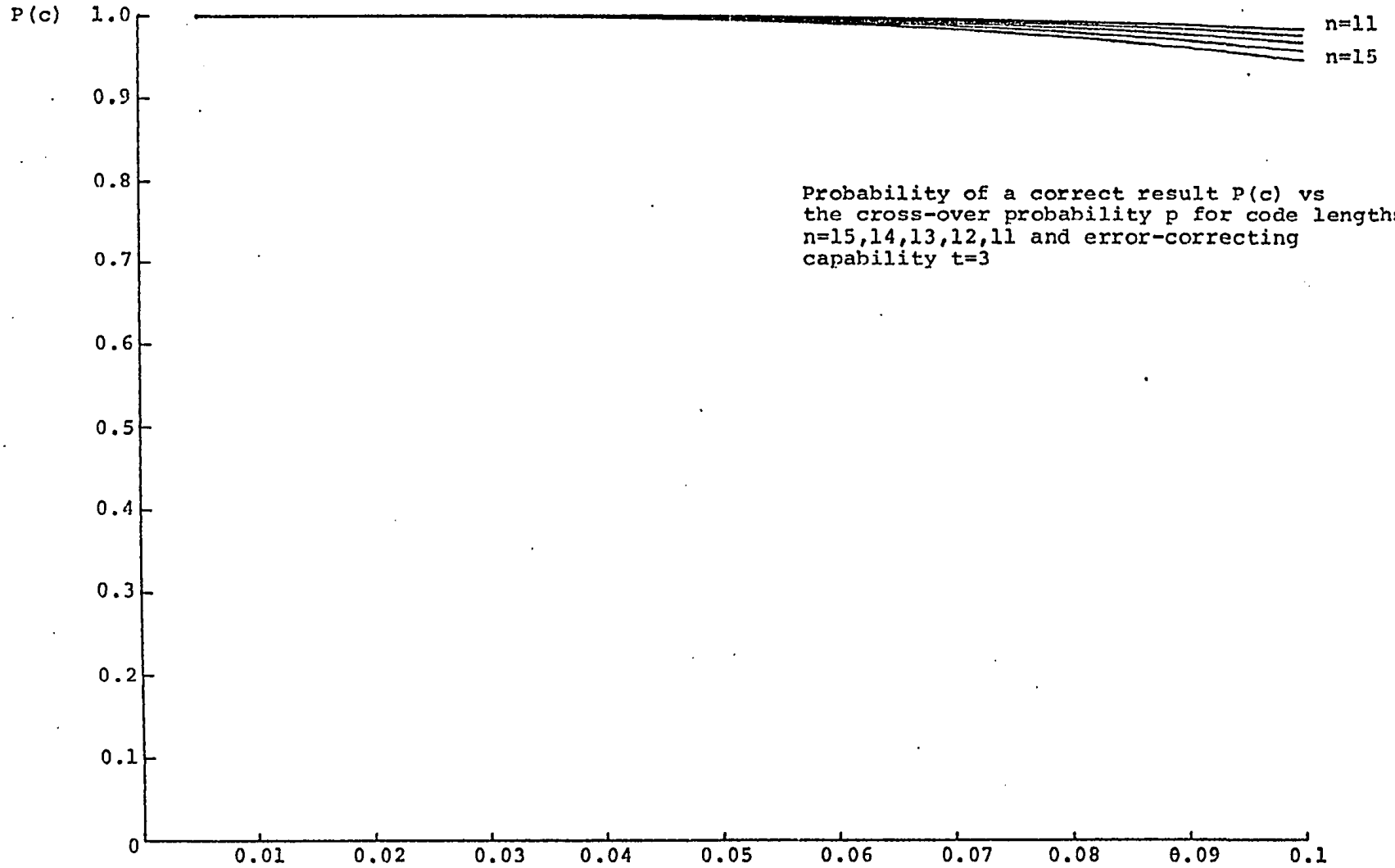

P(c)
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
n=9
n=15
Probability of a correct result P(c) vs
the cross-over probability p for code lengths
n=15,14,13,12,11,10,9 and error-correcting
capability t=1
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1
p
Cross-over probability

Figure 1.4.



Probability of a correct result P(c) vs the cross-over probability p for code lengths n=15,14,13,12,11,10,9 and error-correcting capability t=2

Figure 1.5



Probability of a correct result P(c) vs the cross-over probability p for code lengths n=15,14,13,12,11 and error-correcting capability t=3

Figure 1.6

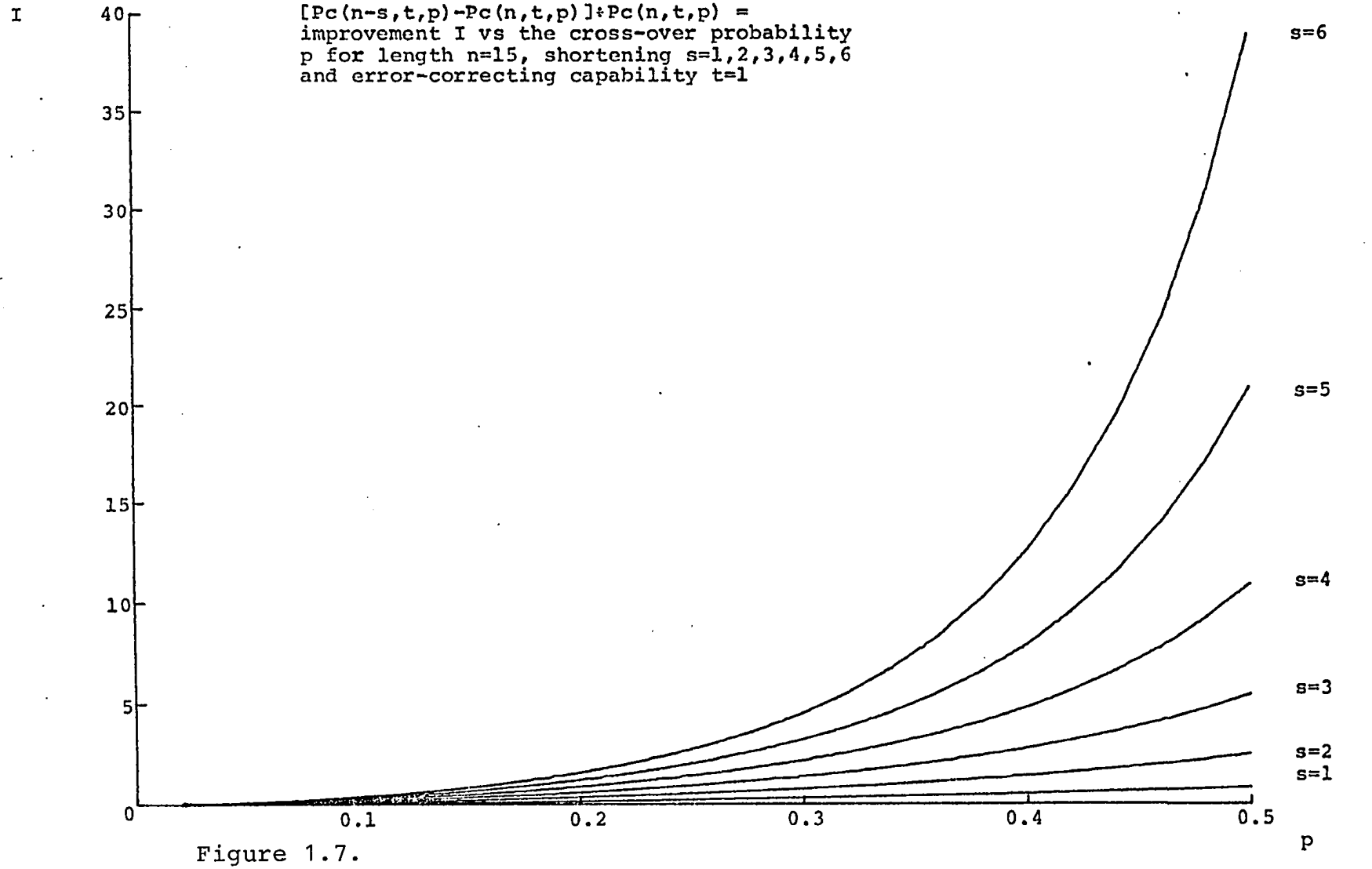


Figure 1.7.

Cross-over probability

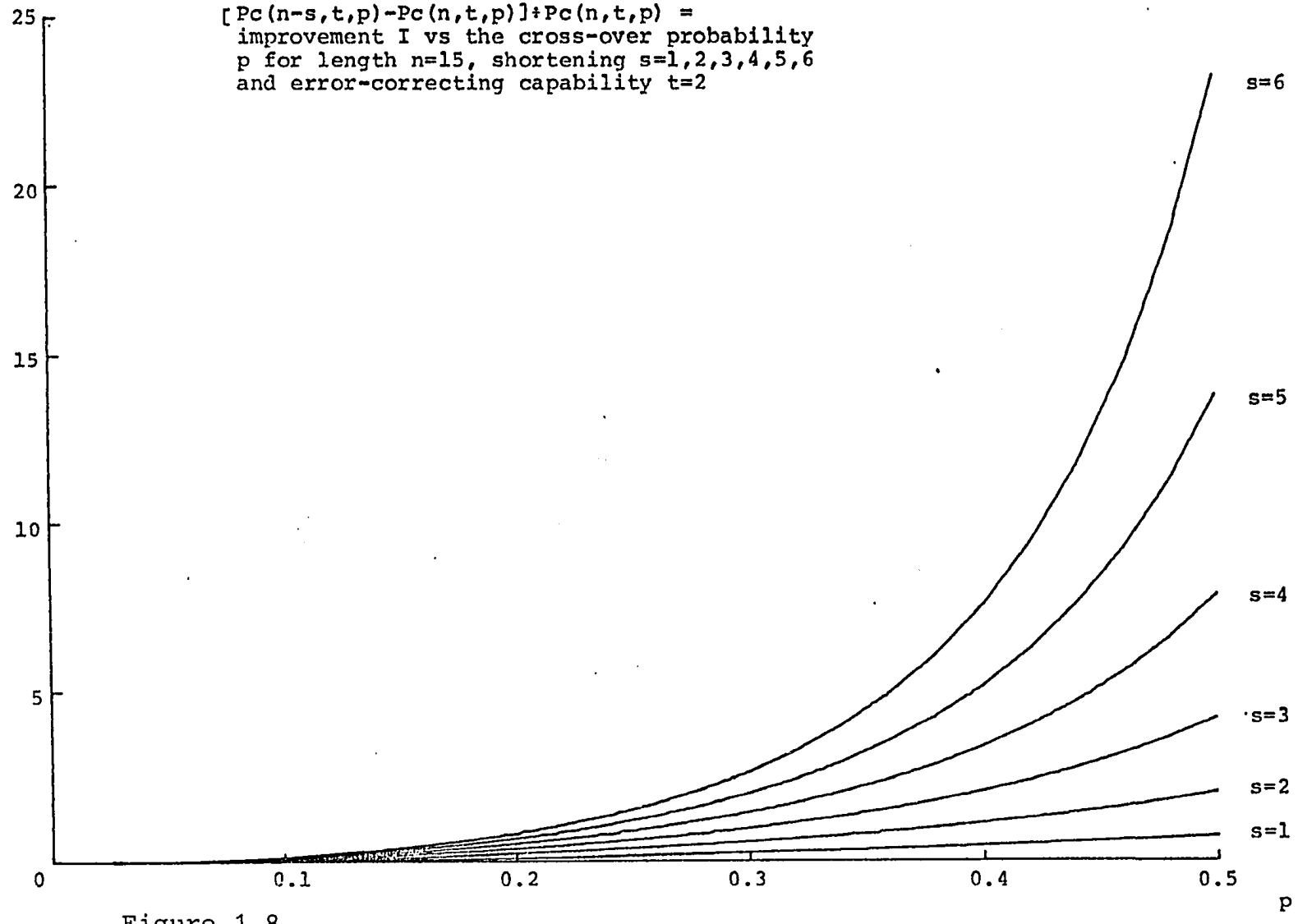

I
25
20
15
10
5
0
0.1
0.2
0.3
0.4
0.5
p
s=6
s=5
s=4
s=3
s=2
s=1
[Pc(n-s,t,p)-Pc(n,t,p)]÷Pc(n,t,p) =
improvement I vs the cross-over probability
p for length n=15, shortening s=1,2,3,4,5,6
and error-correcting capability t=2
Cross-over probability

Figure 1.8.

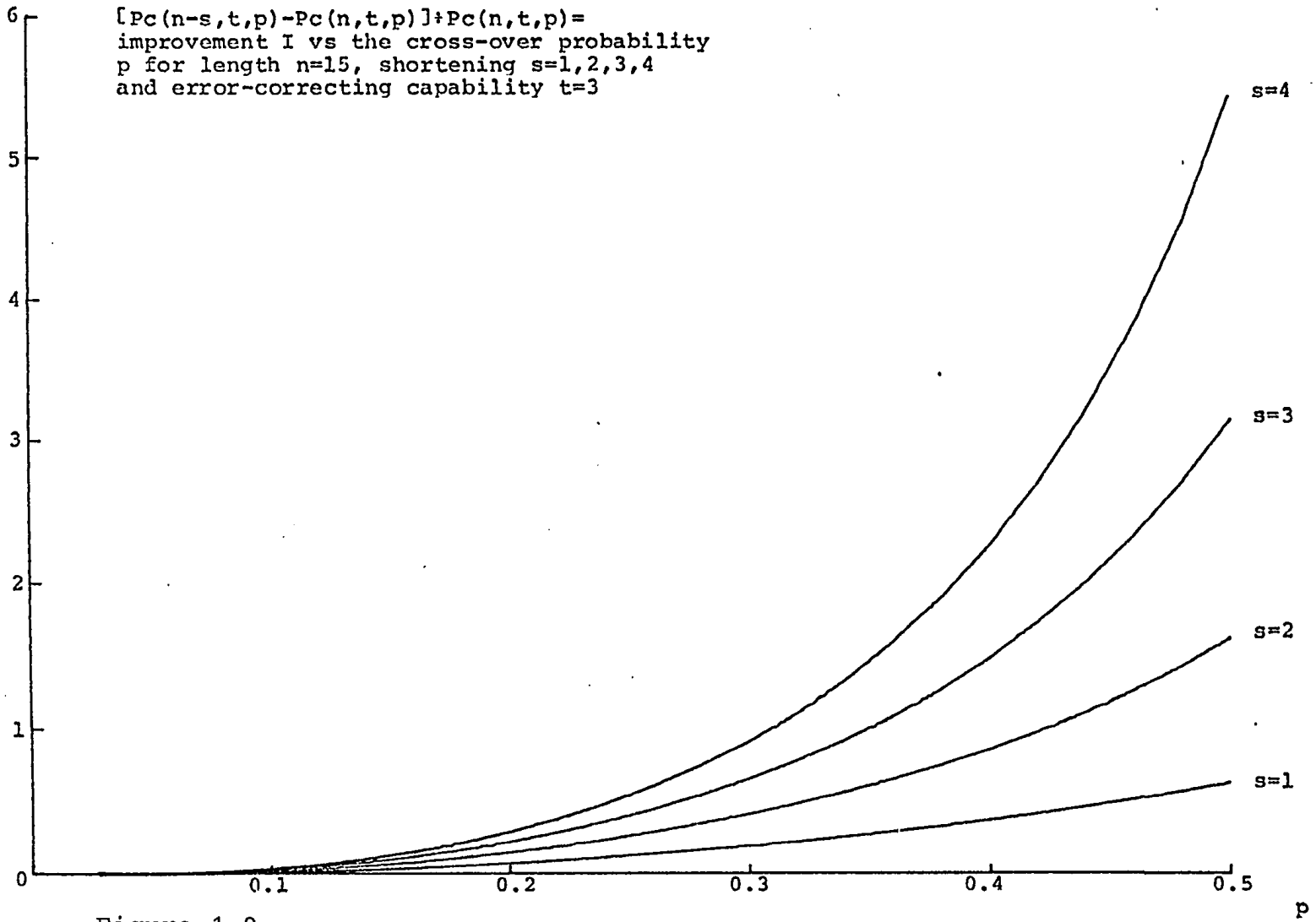

I
[Pc(n-s,t,p)-Pc(n,t,p)]÷Pc(n,t,p)=
improvement I vs the cross-over probability
p for length n=15, shortening s=1,2,3,4
and error-correcting capability t=3
6
5
4
3
2
1
0
0.1
0.2
0.3
0.4
0.5
p
s=4
s=3
s=2
s=1
Cross-over probability

Figure 1.9.

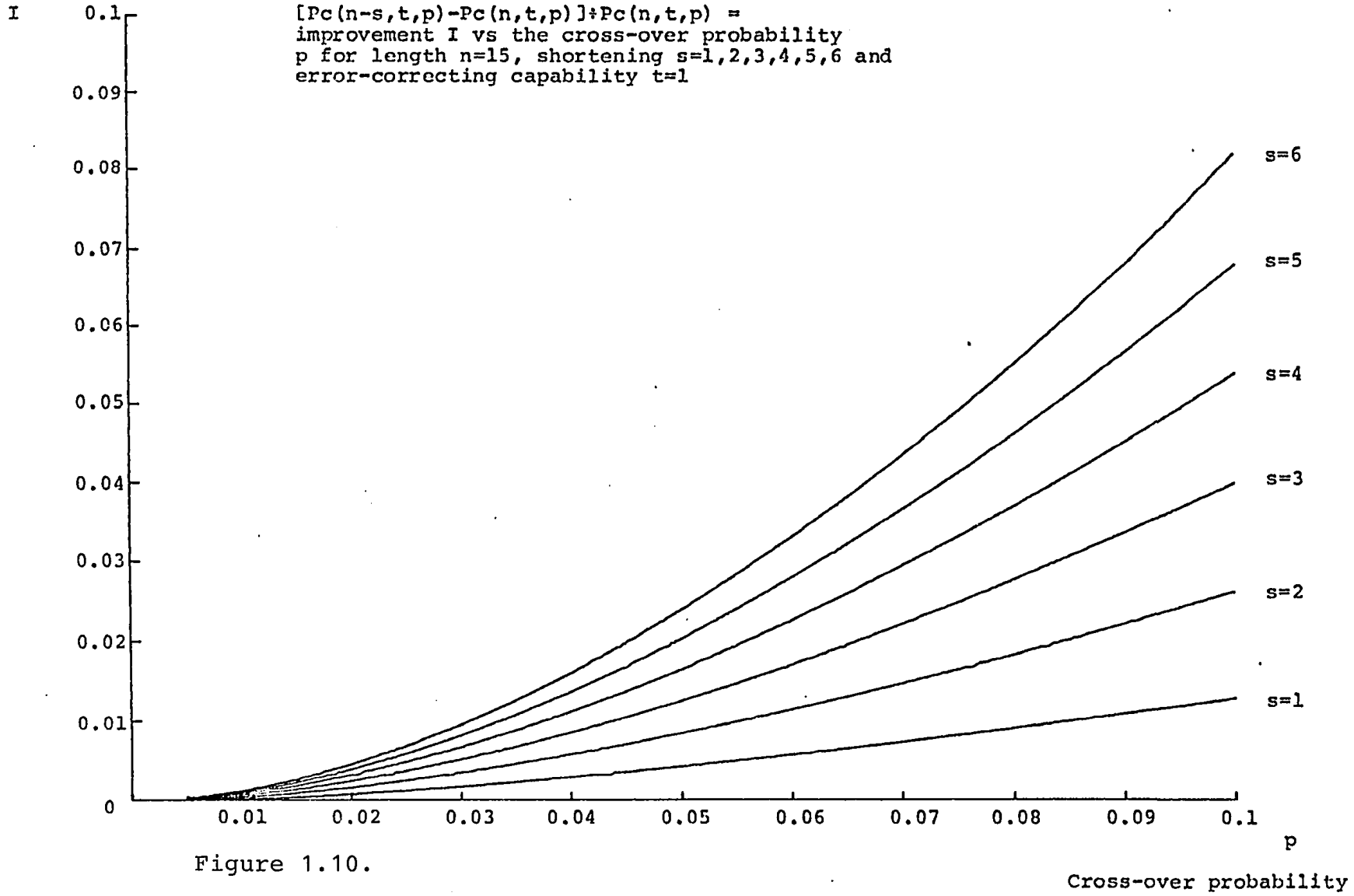

I
0.1
0.09
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
[Pc(n-s,t,p)-Pc(n,t,p)]÷Pc(n,t,p) =
improvement I vs the cross-over probability
p for length n=15, shortening s=1,2,3,4,5,6 and
error-correcting capability t=1
s=6
s=5
s=4
s=3
s=2
s=1
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1
p
Cross-over probability

Figure 1.10.

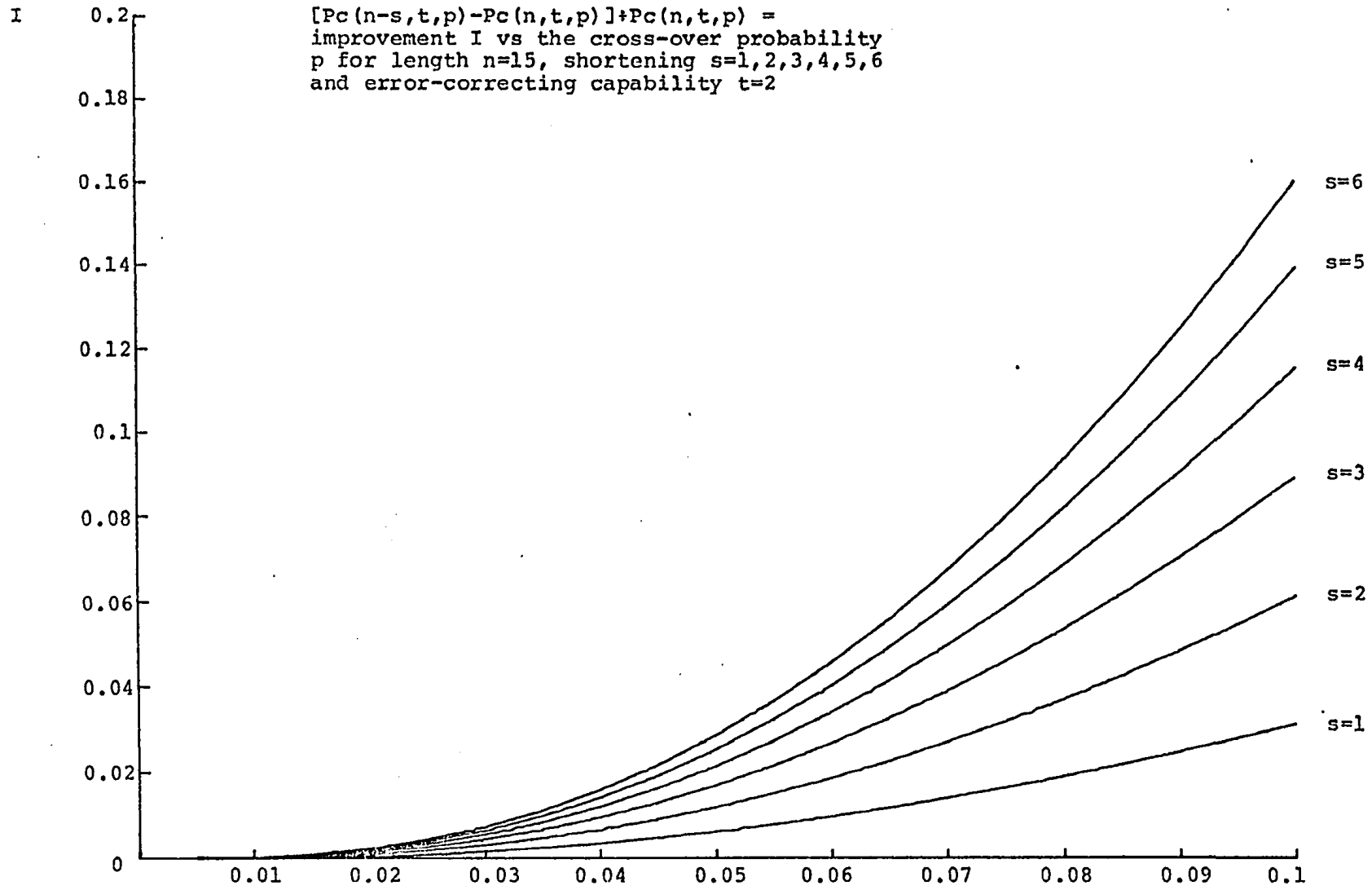

I
0.2
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1
p
s=6
s=5
s=4
s=3
s=2
s=1
[Pc(n-s,t,p)-Pc(n,t,p)]÷Pc(n,t,p) =
improvement I vs the cross-over probability
p for length n=15, shortening s=1,2,3,4,5,6
and error-correcting capability t=2
Cross-over probability

Figure 1.11.

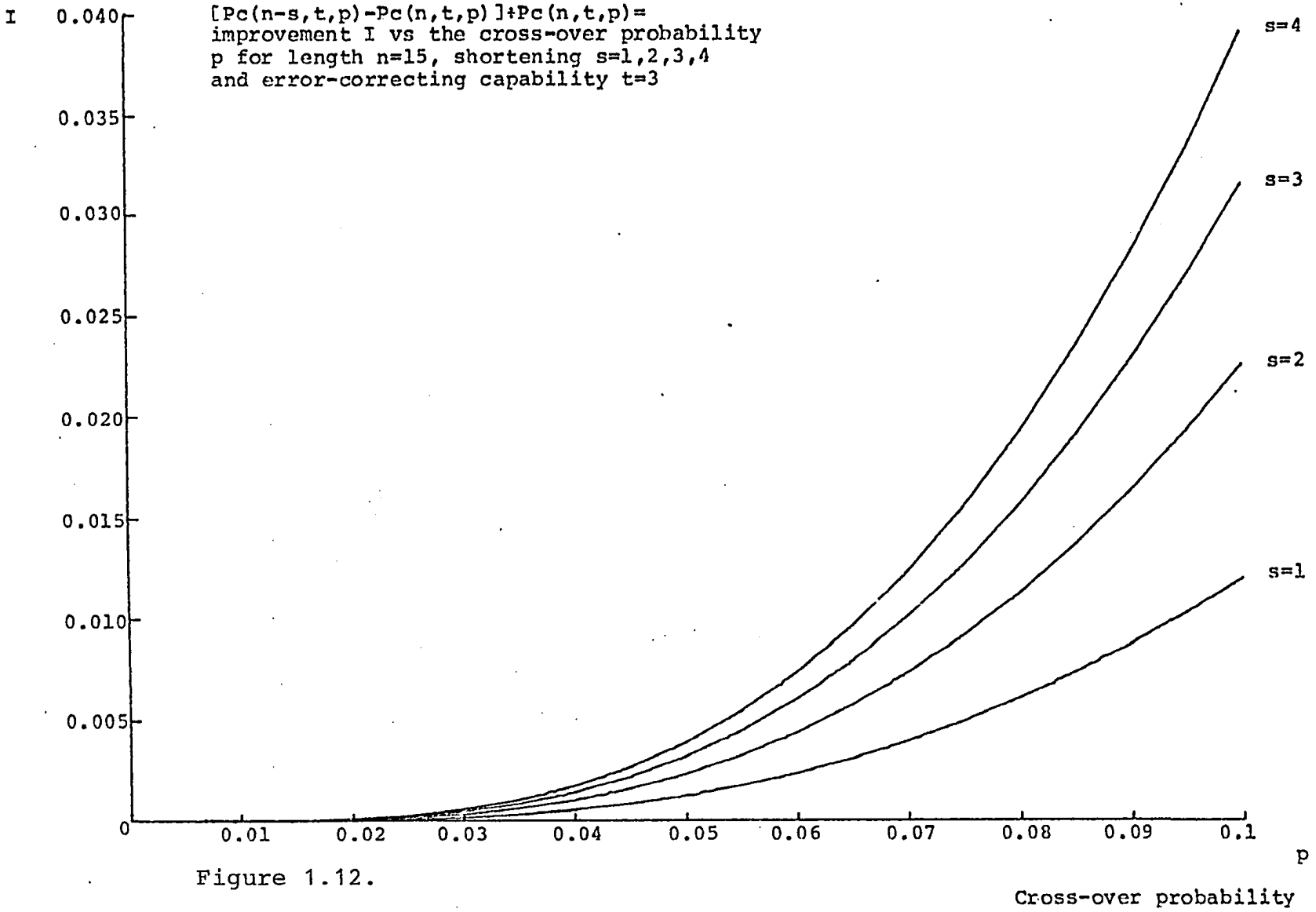

I
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1
p
s=4
s=3
s=2
s=1
[Pc(n-s,t,p)-Pc(n,t,p)]÷Pc(n,t,p)=
improvement I vs the cross-over probability
p for length n=15, shortening s=1,2,3,4
and error-correcting capability t=3
Cross-over probability

Figure 1.12.

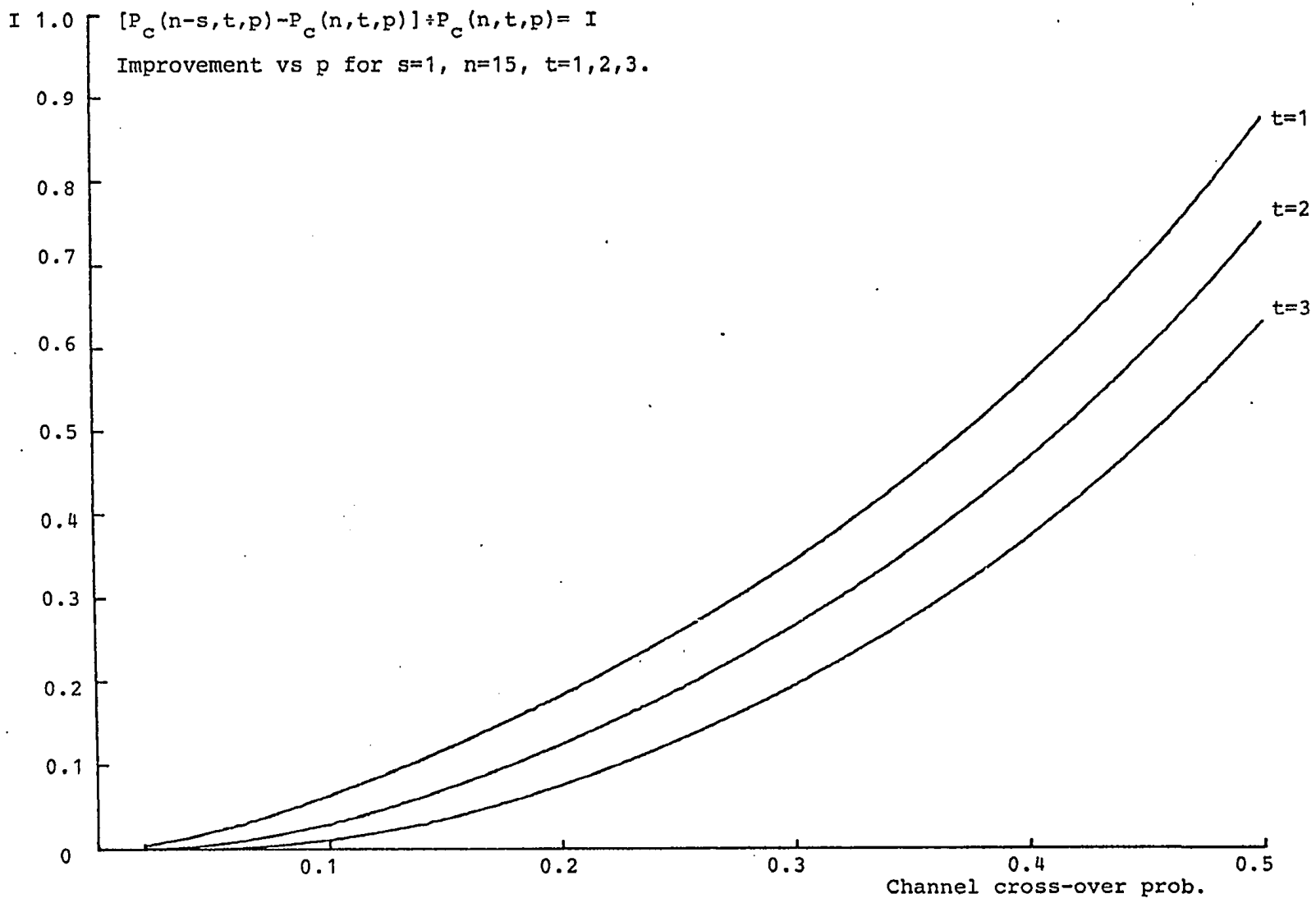

I 1.0
$[P_c(n-s,t,p)-P_c(n,t,p)]\div P_c(n,t,p)= I$
Improvement vs p for s=1, n=15, t=1,2,3.
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
t=1
t=2
t=3
0.1
0.2
0.3
0.4
0.5
Channel cross-over prob.

Figure 1.13.

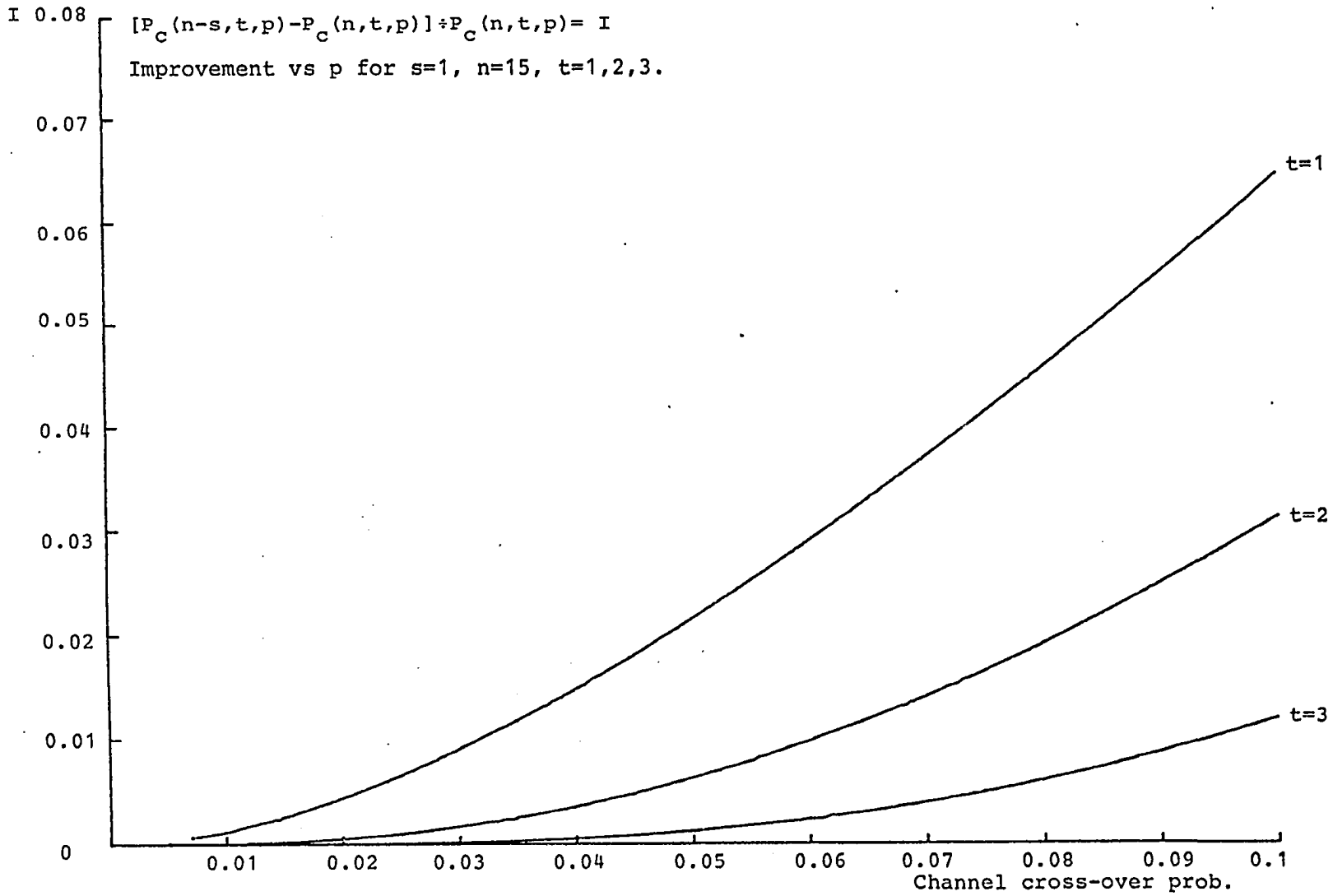

I 0.08
$[P_c(n-s,t,p)-P_c(n,t,p)]\div P_c(n,t,p)= I$
Improvement vs p for s=1, n=15, t=1,2,3.
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
t=1
t=2
t=3
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1
Channel cross-over prob.

Figure 1.14.

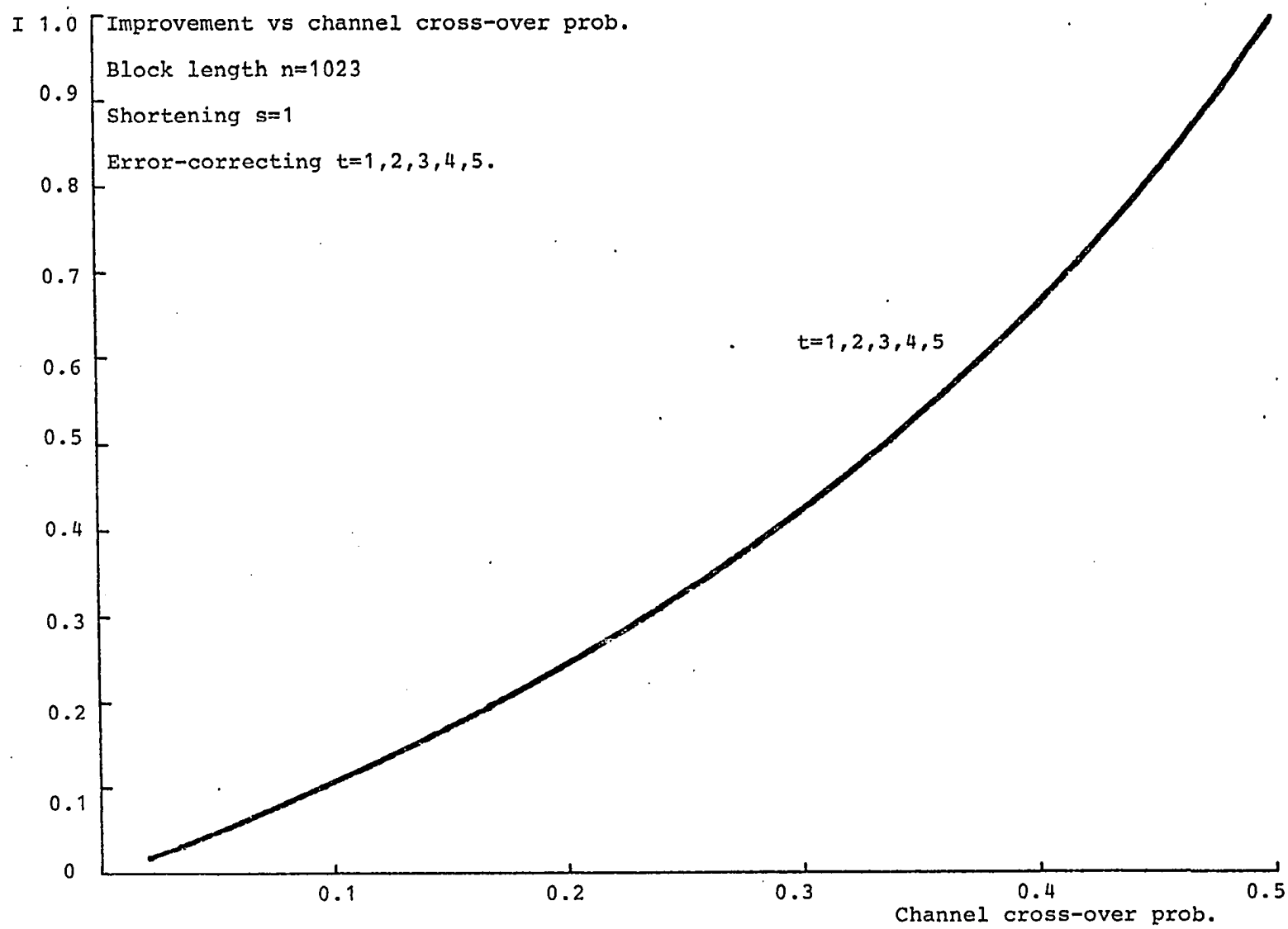

Improvement vs channel cross-over prob.
Block length n=1023
Shortening s=1
Error-correcting t=1,2,3,4,5.
I
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
t=1,2,3,4,5
0.1
0.2
0.3
0.4
0.5
Channel cross-over prob.

Figure 1.15.

CHAPTER 2

MAJORITY LOGIC DECODING

2.1 Introduction

MLD (majority logic decoding) is one of the simplest decoding algorithms known as of today, being very easy to implement. Unfortunately not all codes are efficiently decoded by MLD and in actual fact very few codes are majority logic decodable as opposed to the rest.

In this chapter, the basic concepts of MLD will be introduced in the next section followed in 2.3 by practical circuits for the implementation of an MLD decoder. Also in 2.3 is a comparison of MLD and BDD in terms of decoding time and circuit complexity. Refinements on the MLD algorithm will be in 2.4 and MLD codes are treated in 2.5.

2.2 Majority Logic Decoding

MLD is basically a decoding strategy that can be applied to all codes but with various degrees of success.

The main ideas behind MLD is to form a set of orthogonal check sums on a particular digit so that the digit can be estimated by taking a majority vote on the set of equations. To define the nature of these check sums it is necessary to return to some elementary coding theory.

Consider a linear (n,k) code V. This code is uniquely characterised by its generator matrix G. The generator matrix is formed by a set of k linearly independent n-tuples and a codeword belonging to V is a linear combination of those code vectors.

Corresponding to the generator matrix G of size k x n, there also exists a parity check matrix H of dimension (n-k) x n such that

$$G\,H^{t} = 0 \ . \qquad (2.1)$$

This parity check matrix can be expressed as

$$H = [\ P^{t} : I\ ] \ , \qquad (2.2)$$

where P is the parity matrix of the code. P should chosen in such way as to guarantee the error correcting capability of the code.

A word V(x) belonging to V and a received sequence R(x) are related by

$$R(x) = V(x) + E(x) \ , \qquad (2.3)$$

where E(x) is the error polynomial. In order to perform error correction, E(x) has to be evaluated from R(x). For this reason the syndrome, represented by

$$S = R\,H^{t}\,, \tag{2.4}$$

has to be found. (2.4) can be written differently by the simple fact that if no errors occured, that is if V(x)=R(x), the value of the syndrome is zero. Therefore

$$S = E\,H^{t}\,, \tag{2.5a}$$

or more specifically

$$\begin{bmatrix} s_{k+1} \\ s_{k+2} \\ \vdots \\ s_{n} \end{bmatrix} = [P^{t}:I] \begin{bmatrix} e_{1} \\ e_{2} \\ \vdots \\ e_{n} \end{bmatrix}, \tag{2.5b}$$

where the e_i's are the noise digits.

From (2.5b) it is obvious that the noise digits on the parity check bits are checked by one and only one parity check and that the information noise digits are obtained from the parity matrix P.

For the purpose of MLD, a set of equations is

orthogonal on a particular e_i if e_i is checked by all the equations of the set and if all other entries occur only once in the complete set.

It can be said that if there are d-1 parity check equations, orthogonal on a single e_i, then the code is 1-step majority logic decodable. Here 1-step refers to the fact that the error digit can be evaluated in one step, hence only one majority element used in the decoder.

Example 2.1

Consider the (15,7) BCH code with d=5. In this code it is possible to form (d-1) estimate on e_0. By using the matrix H,

$$H_{(15,7)} = \left[\begin{array}{ccccccc:c} 1 & 1 & 0 & 1 & 0 & 0 & 0 & \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & I \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & \end{array}\right] \qquad (2.6)$$

the following equations are obtained:

$$\begin{aligned} s_0 &= e_0+e_1+e_3+e_7 \ , \\ s_1 &= e_1+e_2+e_4+e_8 \ , \\ s_2 &= e_2+e_3+e_5+e_9 \ , \\ s_3 &= e_3+e_4+e_6+e_{10} \ , \\ s_4 &= e_0+e_1+e_3+e_4+e_5+e_{11} \ , \\ s_5 &= e_1+e_2+e_4+e_5+e_6+e_{12} \ , \\ s_6 &= e_0+e_1+e_2+e_5+e_6+e_{13} \ , \\ s_7 &= e_0+e_2+e_6+e_{14} \ . \end{aligned} \qquad (2.7)$$

These relations can be reduced to

$$
\begin{aligned}
s_1 &= e_0+e_1+e_3+e_7, \quad (s_0) , \\
s_2 &= e_0+e_2+e_6+e_{14}, \quad (s_7) , \\
s_3 &= e_0+e_4+e_{12}+e_{13}, \quad (s_5+s_6) , \\
s_4 &= e_0+e_8+e_9+e_{11}, \quad (s_1+s_2+s_4) ,
\end{aligned}
\tag{2.8}
$$

where 4 is d-1. Therefore the (15,7) is completely orthogonalisable in one step, as the four equations are orthogonal on e_0.

It is also important to notice that if d-1 orthogonal relations are obtained, the decoding algorithm will correct all errors of weight t or less, where t=(d-1)/2. This is easy to visualise if one looks at (2.8) and if one adds the dth relation, namely $s_0=e_0+e_0=0$. If two errors affecting two check sums are present, the majority, i.e. three out of five, will still give the right answer.

In the case of a cyclic code, MLD is applicable as long as the set of equations is orthogonal on a particular e_i. By rotating the received word cyclically, it is possible to evaluate all the information bits and that with the same circuitry. To see what an MLD circuit can be, consider figure 2.1 where such a decoder is shown.

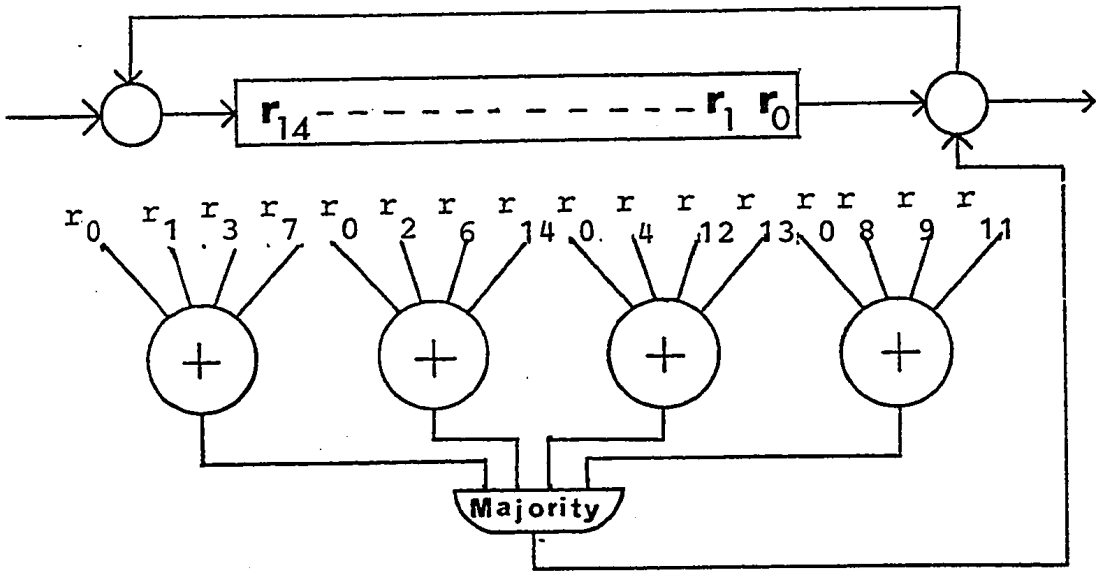


Figure 2.1, MLD decoder for the (15,7).

Keeping in mind that there exists an extra equation which is always zero, it is clear that the decoder of figure 2.1, implementing (2.8), is capable of correcting all error patterns of weight one and two.Furthermore, as will be seen in chapter 3, it can correct some error patterns of weight greater than two, which is the maximun error weight guaranteed correctable.

In some instances, the d-1 estimates will be orthogonal on a sum or sums of different e's and getting

a single e_i out the process in one step is impossible. To obviate this difficulty the sum or sums of e's on which the set or part of the set of equations is orthogonal is or are estimated through a majority vote and then this sum or these sums is or are used as extra relation(s) and a new parity check matrix H' is obtained. If the set of relations obtained from H' is orthogonal on a single error digit e_i then the process is complete and the code is said to be 2-step MLD. A code of this sort would require two levels of majority gating but would still be relatively simple to build. Its error correcting capability would still be the same, that is it would correct errors of weight up to and including t. An example of a 2-step majority logic decodable code is given next.

Example 2.2

Consider the (7,4) Hamming code with d=3. The parity check matrix H for this code is:

$$H_{(7,4)} = \left[\begin{array}{cccc:c} 1 & 0 & 1 & 1 & \\ 1 & 1 & 1 & 0 & \mathrm{I} \\ 0 & 1 & 1 & 1 & \end{array}\right] \qquad (2.9)$$

from which the following equations are obtained:

$$\begin{aligned} s_1 &= e_0+e_2+e_3+e_4 \ , \\ s_2 &= e_0+e_1+e_2+e_5 \ , \\ s_3 &= e_1+e_2+e_3+e_6 \ . \end{aligned} \tag{2.10}$$

By selecting two sets, $A_1=\{e_0,e_1\}$ and $A_2=\{e_0,e_2\}$ it is possible to get estimates B_1 and B_2 for each of them, through a majority vote in the following fashion:

$$B_1 \begin{cases} s_1' = e_0+e_1+e_2+e_5 \ , & (s_2) \ , \\ s_2' = e_0+e_1+e_4+e_6 \ , & (s_1+s_3) \ , \end{cases} \tag{2.11}$$

and

$$B_2 \begin{cases} s_1'' = e_0+e_2+e_3+e_4 \ , & (s_1) \ , \\ s_2'' = e_0+e_2+e_1+e_5 \ , & (s_2) \ . \end{cases} \tag{2.12}$$

B_1 and B_2 are now orthogonal on e_0 and therefore an estimate of e_0 can be obtained by majority vote.

Here again it is clear that by considering the obvious sum of $s_0=e_0+e_0=0$, besides the sums B_1 and B_2 that all single errors can be corrected by this 2-step majority logic. A circuit for such a decoder is shown in figure 2.2.

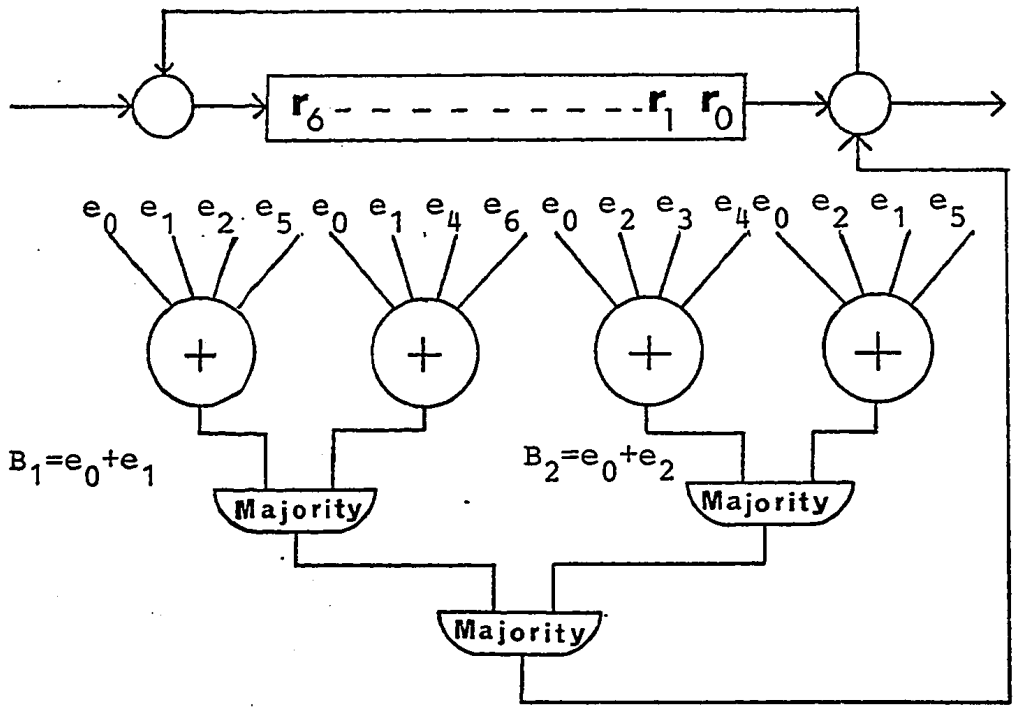


Figure 2.2, A 2-step MLD decoder for the (7,4)

From the arguments used in the justification of 2-step orthogonalisation, it is possible to generalise for L-step MLD. In 2-step MLD, a matrix H' is produced, from which a set of relations orthogonal on e_i can be obtained. If H' does not yield the desired relations, those obtained can be used to form a new matrix H'' and eventually there might exist an H^L containning the equations orthogonal on e_i. In that case the code is said to be L-step orthogonalisable, thus requiring L levels of majority gating. It is understandable that as L gets

large the complexity of the decoding circuit increases, and eventually the circuit required is too complex to be economically implemented. However as there exist d-1 orthogonal estimates obtainable in L steps, the decoding algorithm can correct errors of weight up to and including t, where t=(d-1)/2.

At this point it is important to say that not all codes are L-step orthogonalisable. Some codes, which are orthogonalisable, will be presented in section 2.5. The most interesting type of majority logic decoding is the 1-step process, because of its simplicity and the relatively low cost of its implementation. In this context it is possible to upper-bound the performance of the 1-step MLD decoding of an (n,k) code with minimum distance d. If $\bar{d}$ is the minimum distance of the dual code then t_1, the error correcting capability using 1-step MLD is

$$t_1 \leq \frac{n-1}{2(\bar{d}-1)} \quad . \tag{2.13}$$

The proof of (2.13) is simple [25]. In the null space of the code the minimum weight is $\bar{d}$, so that there is at least $\bar{d}$ digits in each sum. One digit will appear in all the sums while the other $\bar{d}-1$ will be in only one of the

sums. There are n-1 digits besides the one on which the equations are orthogonal and it is therefore possible to construct at the most $(n-1)/(\bar{d}-1)$ relations orthogonal on that digit. t_1, the code error correcting capability with 1-step MLD is therefore half of that. If the code error correcting capability t is equal to t_1, then the code is 1-step MLD. If not, using 1-step MLD would result in less than optimal decoding but might still be considered because of practical limitations.

In the case of L-step decoding, the number of errors t_L, that L-step MLD can correct is

$$t_L \leq \frac{n}{\bar{d}} - \frac{1}{2} \quad , \text{ d even} , \tag{2.14a}$$

$$t_L \leq \frac{n+1}{\bar{d}+1} - \frac{1}{2} , \quad \text{d odd} , \tag{2.14b}$$

From (2.14a) and (2.14b), it is clear that L-step MLD is more powerful than 1-step, as it corrects roughly twice as many error patterns than a 1-step procedure. These results, (2.13, 2.14a, 2.14b), even show that it is impossible to orthogonalise certain codes like, for instance, the Golay code, most quadratic residue codes

and most Reed-Solomon codes.

Going back to 1-step MLD, the next section will investigate the practical implementation of 1-step MLD decoders.

2.3 Practical Implementation of 1-step MLD

There are two basic ways of implementing 1-step MLD. One could calculate the noise digits and add them onto the received sequence. This is what has been done in section 2.2. On the other hand it is possible to evaluate the information digits directly, by modifying the relations very slightly. Consider the following set of relations,

$$\begin{aligned} C_0 &= v_0 \ , \\ C_0 &= v_1+v_3+v_7 \ , \\ C_0 &= v_2+v_6+v_{14} \ , \\ C_0 &= v_4+v_{12}+v_{13} \ , \\ C_0 &= v_8+v_9+v_{11} \ , \end{aligned} \tag{2.15}$$

where C_0 is the first information bit and the v's are

elements of the codeword. To decode, the received sequence R(x) is substituted to the codeword V(x) and, as in the previous case of (2.8), if no more than two errors are present, then C_0 can be evaluated correctly. As the code is cyclic, $C_1, C_2, \ldots, C_{k-1}$ can be obtained by cyclically shifting the codeword and reusing the same equations. A typical circuit for this is shown in figure 2.3.

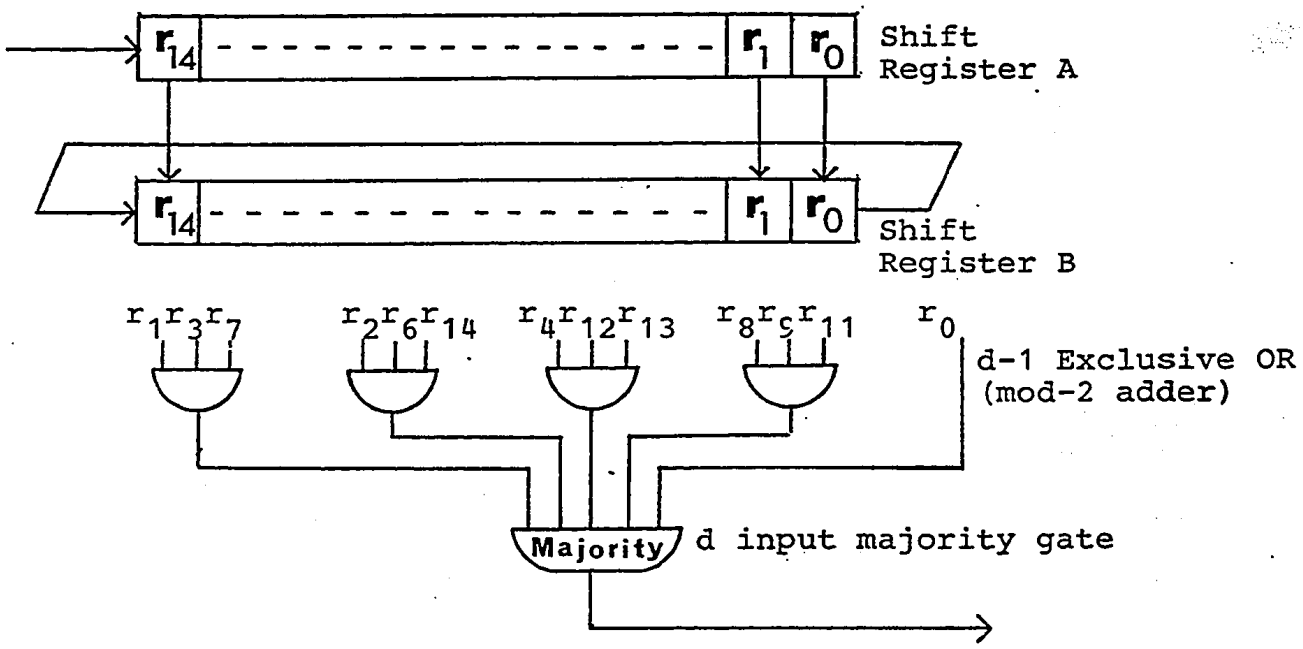


Figure 2.3, A 1-step MLD decoder for the (15,7)

The operations performed by the MLD decoder are as follows:

1) The data serially enters the 15-stage shift register A. Assuming a X nS logic , it takes X nS/bit x 15 bits=15X nS for the shift register A to be full. Using serial input parallel output SR (shift register) it then takes another X nS to transfer the content of SR A into SR B, yielding a turnover time of 16X nS, which is the maximum time allowable for decoding if the throughput is not to be slowed down.

2) Once the data is in SR B, the check sums can be evaluated. A three input mod-2 adder can be made out of two exclusive OR gates (figure 2.4), with a propagation delay of 6 nS per gate level, hence a total time of 12 nS. This is so because of two logic levels inside the adders, each requiring 3 nS of delay [26].

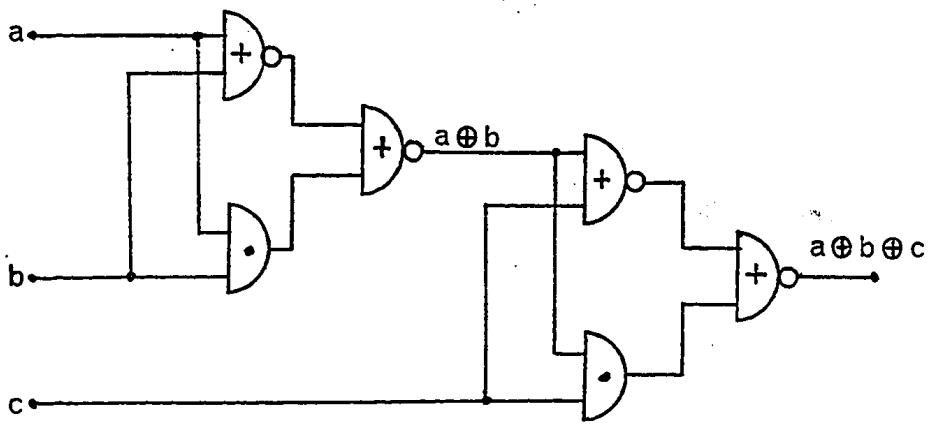


Figure 2.4, A 3-input mod-2 adder.

3) Then a majority vote is taken and for commercially available unit [27], the propagation delay is 180 nS, by far the longest operation.

4) This process has to be repeated k times for the k information digits. Therefore it implies k-1 shifts, each taking X nS. In this case there is a 6X nS delay involved.

Using this procedure the total decoding time for k bits is k(180 nS + 12 nS) = 7(192 nS) = 1,344 nS plus the 6X nS shift delay. Without considering the constraint on the maximum allowable decoding time imposed by the shift register arrangement, a 50 nS logic (X=50 nS) will be assumed. The total MLD decoding time is then 1.7 uS.

Parallel operation is possible if the relations are cyclically incremented, that is $c_0=v_1+v_3+v_7$ would become $C_1=v_2+v_4+v_8$. Doing that, all the C_i's can be estimated at the same time, at the cost of k-1 additional circuits and also the extra error correcting capability inherent

to 1-step MLD, as described in chapter 3 , would be lost. However a very fast decoding time of 192 nS would be possible and the k-1 shifts would be eliminated.

To put 1-step MLD in the proper context a comparison with Berlekamp's iterative decoding [12] for BCH codes is given. Figure 2.5 shows a Berlekamp decoder and figure 2.6 shows details of the central galois field processor.

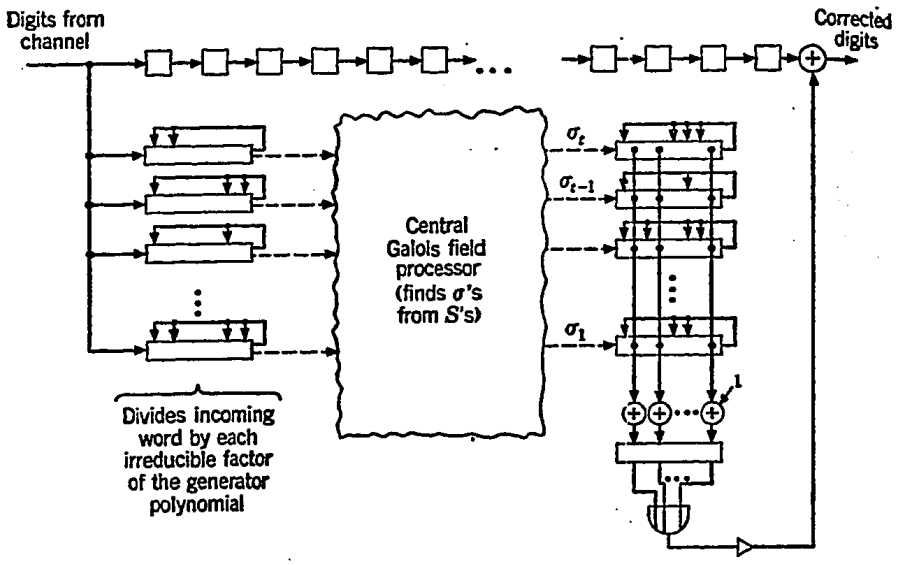


Figure 2.5, Berlekamp's decoder [28].

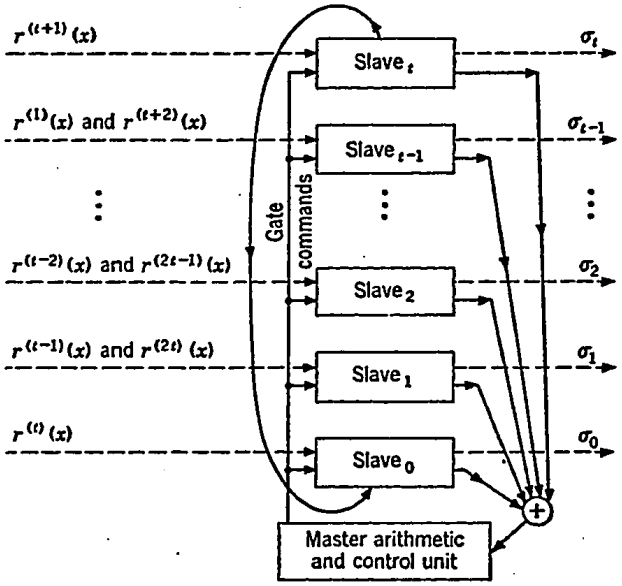


Figure 2.6, Central galois field processor [29].

In this procedure, for double error correcting BCH codes, the CGFP (central galois field processor) has to perform some calculations over the galois field. The CGFP gets $R^1(x)$ and $R^3(x)$ which are the remainders of the received word divided by $M^1(x)$ and $M^3(x)$, the minimal polynomials of α and α^3. From these, the CGFP must find the error locator polynomial. Still in the case of double error correcting codes, the CGFP will compute

$$\begin{aligned} s_1 &= R^1(\alpha) \ , \\ s_2 &= R^1(\alpha^2) \ , \\ s_3 &= R^3(\alpha^3) \ , \end{aligned} \qquad (2.14a)$$

and then will estimate

$$\sigma_2 = \frac{(s_1 s_2 + s_3)}{s_1} \qquad (2.14b)$$

or set $\sigma = 0$ if S_1 and S_3 equal zero.

The tasks performed by the dividing circuit and the CGFP amount to m-1 multiplications for the S's plus t other multiplications per decoded block. The Chien searcher unit performs n multiplications per decoded block plus nt input additions.

In the case of the (15,7) code 20 multiplications and 30 additions have to be done. The galois field over which the computations have to be done is $GF(2^4)$ and due to its reasonable size fast multiplication time can be acheived [30]. In actual fact, multiplications can be done in one clock pulse for that particular case. This would indicate that the overall decoding time is 1 uS, that is 20 multiplications x 50 nS/multiplication, and the time taken for additions can be neglected.

Three procedures have been investigated so far, MLD

with cyclic shifts, parallel MLD and Berlekamp's BCH decoding, all using the same 50 nS logic. The results are tabulated in figure 2.7.

	MLD+ Shift	MLD parallel	BCH
Speed	1.7 uS	192 nS	1 uS
Circuit	Simple	Fairly simple	Complex
Error-correction	up to t + more	up to t	up to t

Figure 2.7, Results of comparison.

Parallel MLD is the fastest at a reasonable level of complexity. However that applies only for small n's and small d's. The slowest element in MLD is the majority gate. As d gets large, the gate or the arrangement of gates is becoming slow and cumbersome. Furthermore, summing the check sums is also slower and more complex as the number of components increases. On the other hand BCH decoding will increase in complexity but will maintain a fairly good speed. It would therefore seem that MLD would be attractive if restricted to codes of moderate lengths

but above all to codes with moderate distances.

2.4 Refinements on the MLD algorithm

Besides being very simple, the MLD algorithm is very flexible. It can be modified to cater for the erasure channel and also it can accept and use reliability estimates.

In certain cases, the binary erasure channel will be used as a model, (figure 2.8).

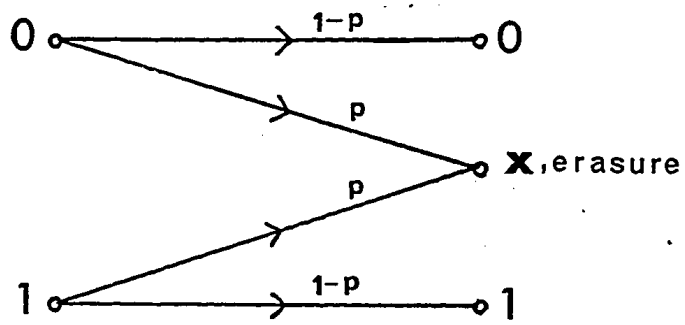


Figure 2.8, Binary erasure channel.

When a channel of this type is used, the error locations are known although their magnitude, 1 or 0 in the binary case, are still unknown. If the minimum distance of a code is d=2t+1, and if erasures occur, then the distance

can be rewritten as $d=2t'+e+1$, where $t' < t$ and e represents the number of erasures the code can cope with [31].

The MLD algorithm can be easily modified to take care of erasures. When a position r_j in the received sequence has been declared an erasure by the estimator, it is simply not considered, that is to say r_j is crossed out of the check sum in which it is present. The concept of erasure is that information can be obtained from the demodulator and can be put to good use. This concept can be taken a step further [32] with the use of reliability estimates. In such a scheme, each received digit has a reliability factor attached to it, so that the decoder can adopt different strategies depending on the reliability vector, i.e. the set of reliability estimates of all digits. One strategy of interest is the one that decodes the least reliable digit and goes to the second least reliable digit and so on, until all the information digits are decoded. Soft decisions of that type can really improve the performance of a majority logic decodable code.

It is not always possible to form d-1 orthogonal

estimates for a particular code and that in any number of steps. In fact, codes which are majority logic decodable only form a small group as compared to the rest of the known codes. MLD being so attractive due to its simplicity, many people tried to find ways by which MLD could be applied to more codes. This was done along two main lines. One approach which will be treated in section 2.5 was to find codes which were majority logic decodable. Another method was to modify the present MLD algorithm and to adapt it to the decoding of any linear code. Work in this direction has been done mainly by Rudolph [33,34,35].

One of the proposed schemes is making use of non orthogonal parity check equations [33] to decode a large class of otherwise not MLD decodable codes. Given a code, its parity check matrix H is reduced to H_0, where H_0 is a submatrix consisting of the b rows of H in which $h_{i0} \neq 0$. Then if R(x) is the received sequence,

$$\sum_{j=0}^{n-1} h_{i_k j}\, r_j = 0 \quad , \; k=1,2,3,..,b \; , \tag{2.15}$$

and finally r_0 can be estimated by,

$$r_0 = -h_{i_k 0}^{-1} \sum_{i=1}^{n-1} h_{i_k j}\, r_i \quad , \; k=1,2,3,..,b \; . \tag{2.16}$$

It has also been shown [36] that such a scheme can decode all single error correcting codes.

Rudolph [34] states that every linear code can be decoded by a 1-step threshold decoding algorithm using non orthogonal parity checks, an exponentiator operator and a linear threshold element. In the binary case, the exponentiator and the threshold element can sometimes be replaced by a majority element and always by a weighted majority element [35].

The generalised threshold decoding is not easily applicable because of its complexity. However the weighted majority logic decoding is interesting in the sense that any decoding rule for any linear binary code can be realised in 1-step weighted MLD.

2.5 Majority logic decodable codes

A majority logic decodable code is a code with a minimum distance d, which allows the formation of d-1 orthogonal estimates on an error digit e_i. These equations can be obtained in one step in the case of 1-step MLD or in L steps in the case of L-step MLD.

Among the codes which are 1-step MLD, there is a class of codes, namely the DS (difference set) codes [37], which is of interest. The main feature about these codes is that they are 1-step MLD although they are not as powerful as some other cyclic codes.

Given a set P, which is a perfect simple difference set, of the form $P=\{0,l_1,l_2,\ldots,l_2s\}$, it is possible to obtain a generator polynomial which will generate a DS code with the following parameters:

$$\begin{aligned} n &= 2^{2s}+2^{s}+1 \ , \\ n-k &= 3^{s}+1 \ , \\ d &= 2^{s}+2 \ . \end{aligned} \qquad (2.17)$$

As an example of a DS code, the (21,11) code which is 1-step MLD can be mentioned.

Binary codes which are generated by $g(x)=1+x^2+x^{2a+1}$, where $n=(2a+1)(a+1)$, $k=(2a+1)a$, $a\geq 1$, are single error correcting 1-step MLD codes [38]. They compare favourably with the TW (Townsend Weldon) codes as far as single error correcting codes are concerned.

Also of interest are the codes generated by $g(x)=1+x^{2a}+x^{2a+1}+x^{4a+1}$, with $n=4a+4a+1$, $k=4a$, and $d=4$ [39]. These codes are 1-step majority logic decodable.

Before going to L-step MLD codes there is a very useful result, which although applicable to MLD in general, is most relevant to 1-step MLD. If an MLD code with odd minimum distance is considered, then by taking the even subset of the codewords, i.e. multiplying $g(x)$ by $(1+x)$, it is possible to form an extra check sum orthogonal on a particular digit [40]. The new code is therefore 1-step MLD with d orthogonal estimates. The implications of this are twofold. The extra sum of the new code could either be used for correction of erasures or for detection of non correctable error patterns of weight t+1.

Classes of codes which are L-step MLD, L>1, are more numerous. The Hamming codes, which are a special case of the BCH codes with parameters $(2^m-1, 2^m-m-1)$, $m=1,2,\ldots,m$, and $d=3$, can all be L-step orthogonalised with L not greater than m-1 [41]. Their dual codes, the maximal length codes can also be orthogonalised completely.

Other codes which are of interest are the RM (Reed Muller) codes. The first order RM codes of the form $(2^m, m+1)$ with $d=2^{m-1}$ can all be 2-step orthogonalised [41].

Finite projective geometry codes and Euclidian geometry codes are extension of the RM codes. These codes can be completely orthogonalised, but their complexity puts them beyond the scope of this section. Suffice to say that perfect difference set codes and maximal length codes are both subclasses of the projective geometry codes.

Finally, although no general results seem to be available as far as majority logic decoding of negacyclic codes is concerned, at least one of these codes has been found to be 1-step MLD [42].

This section enumerated certain classes of MLD codes but is by no means all inclusive and thorough.

CHAPTER 3

1-STEP MAJORITY LOGIC DECODING

3.1 Introduction

The definition of BDD (bounded distance decoding) was given earlier in chapter 1. Basically BDD does not decode a code completely, that is to say does not use all of the available syndromes. For example, the (15,7) BCH double error correcting code will correct all errors of weight 0,1,and 2 if BDD is used. This means a total of 121 possible cases while the total number of available syndromes is 256, leaving 135 unused syndromes.

A procedure like 1-step MLD corrects not only all error patterns of weight t or less but also error patterns of weight greater than t. The total number of correctable error patterns is equal to 2^{n-k} which is the number of syndromes. However, unlike in the case of a standard array type of decoding, every correctable error pattern of weight greater than t is not necessarily one of high probability of occurrence [43]. For this reason, the extra error correcting capability of 1-step MLD was

investigated both theoretically and experimentally.

In this chapter, section 3.3 deals with the theoretical aspect of estimating correctable error patterns of weight greater than t, in particular those of weight t+1, because of their high probability of occurrence. Section 3.4 is devoted to the results of simulations of 1-step MLD decoding of certain codes. The performance of 1-step MLD is then compared with BDD and a standard array type of decoding by means of the decoding error probability.

3.2 An MLD Algorithm

The work carried out in section 3.3 is based on a particular decoding algorithm which shall be explained in detail. This algorithm is well known but its principles of operation are essential to the understanding of the material in the following section.

Referring to section 2.1, an (n,k) code , with minimum distance d, is 1-step MLD if d-1 orthogonal estimates can be formed on a single e_i or C_i, where e_i is an element of the noise sequence and C_i is an element of

the information sequence.

Consider the (15,7) code, with orthogonal equations as described in (2.8) and (2.15). Upon receiving the sequence

$$R(x) = V(x) + E(x) \ , \tag{3.1a}$$

$$R(x) = r_0+r_1x +r_2x^2+\ldots+r_{n-1}x^{n-1}, \tag{3.1b}$$

the decoder evaluates an estimate $\hat{C}_0$ of C_0 of the information polynomial

$$C(x) = c_0+c_1x +c_2x^2+\ldots+c_{k-1}x^{k-1}, \tag{3.2}$$

by means of (2.15). Clearly, if no more than two errors affected (2.15), a majority vote will yield the correct value of C_0. The next thing to be computed is

$$x^{-1}\left[R(x) + \hat{C}_0 g(x)\right] = r'_{-1}x^{-1}+r'_0+r'_1x +r'_2x^2+\ldots+r'_{n-1}x^{n-1}, \tag{3.3}$$

and from (3.3)

$$R'(x)= r'_0+r'_1x +r'_2x^2+\ldots+r'_{n-1}x^{n-1} \ , \tag{3.4}$$

can be obtained. It is interesting to see that r_0 of (3.1b) is now missing in (3.4) as it has been shifted out, thereby cancelling any further influence of that particular bit. Also in (3.4), r'_{n-1} is equal to zero,

making the whole operation similar to a left cyclic shift since the addition of $\hat{C}_0 g(x)$ always results in setting r_0 of (3.1b) equal to zero.

As the entire word has been shifted by one position, (2.15), the set of relations for C_0 can now be reused for C_1 and here again

$$x^{-1}\left[R^1(x) + \hat{C}_1 g(x)\right] = r''_{-1}x^{-1} + r''_0 +$$

$$+r''_1 x^1 + r''_2 x^2 + \ldots + r''_{n-1}x^{n-1}, \qquad (3.5)$$

is computed and a new quantity

$$R''(x) = r''_0 + r''_1 x + r''_2 x^2 + \ldots + r''_{n-1}x^{n-1}, \qquad (3.6)$$

is now formed.

In $R''(x)$, r''_{n-1} and r''_{n-2} are zero as expected. Furthermore, the effect of $\hat{C}_0$ and $\hat{C}_1$ has been cancelled out by the shift operation. (2.15) can now be used again, in the same fashion as before, giving $\hat{C}_2$ which is an estimate of C_2.

This decoding procedure can be applied recursively for estimating all the C_i's, i going from 0 to k-1. To repeat an argument which has been used before, but which

is central to 1-step MLD, it is clear that at any stage of the process , that is in the evaluation of the C_i's, the procedure will evaluate them correctly if no more than (d-1)/2 check sums are affected.

3.3 Some Theoretical Results

Consider a 1-step MLD (n,k) code V with minimum distance d and with orthogonal relations of the form:

$$\begin{aligned} C_0 &= s_0 = v_0 \ , \\ C_0 &= s_1 \ , \\ C_0 &= s_2 \ , \\ &\vdots \\ C_0 &= s_{d-1} \ . \end{aligned} \tag{3.7}$$

In connection with the check sums of (3.7), a new quantity, ρ, can be defined, ρ_i being the number of digits associated with the check sum S_i. Furhtermore, a theorem on the values of these ρ 's can be stated.

Theorem 3.1

$$\rho_t \triangleq \rho_1+\rho_2+\ldots\ldots+\rho_{d-1} \leq n-2 \ . \tag{3.8}$$

Proof:

Theorem 3.1 states explicitely that for a 1-step MLD code, at least one v_i, $i \neq 0$, is missing from the orthogonal relations of (3.7). Assuming that d is odd, adding the d sums of (3.7) will give:

$$C_0 = v_0+v_1+v_2+\ldots\ldots+v_{n-1} \quad . \tag{3.9}$$

The value of C_0 would be dependent on the weight of the codeword, that is to say that if the weight of the codeword is odd, C_0 is equal to 1 and if the weight is even, C_0 is 0. Since this is impossible, because the code is a group code, at least one v_i, $i \neq 0$, has to be missing.

From theorem 3.1, interesting corollaries can be obtained regarding the correctability of error patterns of weight t+1.

Corollary 3.1

There exists at least one error pattern of weight t+1 which 1-step MLD can correct.

Proof:

According to the decoding procedure described in section 3.2, an error pattern of the form

$$E_0(x) = 1 + x^{a1} + x^{a2} + \ldots + x^{at} , \qquad (3.11a)$$

can be decoded properly if one a_i is affecting the missing v_i in the orthogonal relations of (3.7). The weight of the error pattern in (3.11a) is t+1, but as one v_i is missing in the sums, $\hat{C}_0$, the estimate of C_o, is decoded properly. Furthermore, r_0, which was in error, is simply shifted out , reducing the overall weight of the error pattern to only t, thus making it a correctable pattern [44].

Corollary 3.2

Assuming only one v_i missing in (3.7), there exist at least $\binom{n-2}{t-1}$ error patterns of weight t+1, which 1-step MLD can correct.

Proof:

This follows directly from corollary 3.1. In this particular case, the proper estimation of C_0 is the only condition for the correct decoding of the entire information sequence. The error pattern (3.11a) has one digit fixed in the first position and t other digits. One of these t digits is affecting the missing v_i , and the

remaining t-1 digits can be distributed over n-2 positions, therefore giving a total number of possibilities equal to $\binom{n-2}{t-1}$.

Using corollaries (3.1) and (3.2), another result can be stated regarding the 1-step majority logic decodability of binary perfect codes.

Corollary 3.3

Binary perfect codes are not 1-step majority logic decodable.

Proof:

This is easily verified by the fact that, in a binary perfect code, all the syndromes are used to correct error patterns of weight t or less. There is no syndrome left for the correction of any pattern of weight greater than t. Since corollary 3.1 states that in 1-step MLD, there ought to be at least one such a pattern, it implies that corollary 3.3 is true.

As an example of this, consider the single error

correcting Hamming codes which are a class of binary perfect codes. These codes are not 1-step MLD although they can be orthogonalised in L steps.
e.g. The (7,4) Hamming code which is 2-step MLD and the (15,11) Hamming code which is 3-step MLD.

In order to estimate the number of error patterns of weight t+1 which 1-step MLD can correct, it is convenient at the beginning to express $E_0(x)$ as

$$E_0(x) = 1 + x^{a}1 + x^{a}2 + \ldots + x^{a}t \quad , \qquad (3.11b)$$

which is similar to (3.11a), having an error fixed in the zeroth position while the remaining t errors are distributed anywhere in the other n-1 positions.

The condition for the correction of such an error pattern is, as stated before, that $\hat{C}_0$, the estimate of C_0, should be decoded correctly. Therefore, only the evaluation of $\hat{C}_0$ is of concern in determining the correctability of (3.11b).

The first check sum in (3.7), i.e. $S_0=v_0$, with $\rho_0=1$, is always affected by (3.11b). If N_0' is the number of Eo(x) for which $\hat{C}(x)\neq C(x)$, then, with reference to (3.7), N_0' is given by:

$$N_0' = \sum \rho_{i_1} \rho_{i_2} \rho_{i_3} \cdots \rho_{i_t} \quad . \tag{3.12}$$

In (3.12) the summation is over all i_j and $i_1 < i_2 < \ldots < i_t$, and ρ_{i_j} is as defined before. As there are altogether $\binom{n-1}{t}$ error patterns $E_0(x)$, the following theorem can be stated regarding N_0 the number of correctable error patterns $E_0(x)$.

Theorem 3.2

$$N_0 = \binom{n-1}{t} - N_0' \quad . \tag{3.13}$$

When the orthogonal sums of (3.7) are not known, a bound for N_0 would be useful. To get such a bound, the ρ's of theorem 3.1 are expressed as follows:

$$\rho = 2t\, m' + b' \quad , \quad 0 \le b' < 2t \quad , \tag{3.14}$$

and the worst case is considered. Using the Lagrangian multiplier technique, it can be shown that N_0', the number of patterns which are not correctable, is maximum when, in the orthogonal check sums of (3.7), b' of the 2t check sums have each (m+1) v_i's and (2t-b') check sums have each m v_i's. N_0' can now be expressed as

$$N_0' \leq B' \triangleq \binom{b}{t}(m+1)^t + \binom{b}{t-1}(m+1)^{t-1}\binom{2t-b}{1}m +$$

$$+ \binom{b}{t-2}(m+1)^{t-2}\binom{2t-b}{2}m^2 + \ldots + \binom{2t-b}{t}m^t \quad , \qquad (3.15)$$

since $\rho \leq (n-2)$, and the values of m and b are given by

$$n-2 = 2t\,m + b \quad , \; 0 \leq b < 2t \quad . \qquad (3.16)$$

Using (3.15) in theorem 3.2, a similar bound can be obtained.

Theorem 3.3

$$N_0 \geq B \triangleq \binom{n-1}{t} - B' \quad , \qquad (3.17)$$

where $\binom{n-1}{t}$ is the total number of possible error patterns $E_0(x)$ of (3.11b).

At this point it is interesting to note that using theorem 3.3 in the case of double error corecting codes, N_0 is as follows:

$$
\begin{aligned}
N_0 &\geq n^2-4/8 , \quad b=0 , \\
N_0 &\geq n^2-1/8 , \quad b=1 , \\
N_0 &\geq n^2/8 , \quad b=2 , \\
N_0 &\geq n^2-1/8 , \quad b=3 .
\end{aligned}
\tag{3.18}
$$

In (3.18), as n increases, B tends to $n^2/8$ which is about 25% of the total number of patterns $E_0(x)$ of (3.11b).

Having obtained a bound on N_0 with respect to $E_0(x)$, it would be interesting to have similar bounds on $E_1(x)$, $E_2(x),\ldots, E_{n-t-1}(x)$. Consider the following error patterns

$$E_1(x) = x + x^{a1} + x^{a2} +\ldots+ x^{at} \quad , \tag{3.19}$$

which have weight t+1 and an error in the first position. Such a pattern $E_1(x)$ will be correctable if and only if $\hat{C}_0=C_0$ and $\hat{C}_1=C_1$, thereby ensuring that $\hat{C}(x)=C(x)$, because the error in the first position would have been shifted out and therefore would not affect the estimation of the subsequent C_i's.

Theorem 3.1 stated specifically that at least one v_i was missing in the check sums orhtogonal on C_0. In section 2.2 and 2.3 the cyclic nature of the check sums was explained and therefore if $v_\alpha, \alpha \neq 0$, is missing in the sums on C_0, then $v_{\alpha+1}$ will be missing in the sums on C_1.

There exists some error patterns $E_1(x)$ containing both x^α and $x^{\alpha+1}$, the missing v_i's in C_0 and C_1 respectively and such patterns would be correctable because they would not affect more than t check sums in the estimation of C_0 and C_1. This leads to theorem 3.4.

Theorem 3.4

$$N_1 \geq \binom{n-4}{t-2} , \quad t \geq 2 , \tag{3.20}$$

where N_1 is the number of correctable patterns of (3.19).

Proof:

The previous paragraph proved the existence of correctable error patterns $E_1(x)$ if they contained x^α and $x^{\alpha+1}$. If $\alpha=1$, then n-3 positions are left in which t-1 errors could be distributed, giving a total of $\binom{n-3}{t-1}$. On the other hand α can be greater than 1 and in that

case there are n-4 positions in which t-2 errors could be distributed, giving a new total of $\binom{n-4}{t-2}$ possible cases. The code length n is usually greater than or equal to t+2 so that $\binom{n-4}{t-2} \leq \binom{n-3}{t-1}$, thus proving theorem 3.4.

By using the same type of arguments used in proving theorem 3.4, it is possible to extrapolate these results to the more general case of

$$E_\sigma(x) = x^\sigma + x^{a1} + x^{a2} + \ldots + x^{at} \quad , \tag{3.21}$$

which has a weight of t+1 and an error always in the σ th position. Definning N_σ to be the number of $E_\sigma(x)$ of (3.21) for which $\hat{C}(x)=C(x)$, the following theorem can be obtained.

Theorem 3.5

$$N_\sigma \geq \binom{n-2\sigma-2}{t-\sigma-1} \quad , \quad t \geq \sigma+1. \tag{3.22}$$

In this particular theorem, $n \geq t+\sigma+1$ which is generally true.

Theorem 3.5 is general but in the special case of $\sigma=0$, theorem 3.3 is preferable because it gives a more realistic estimate. This is quite reasonable since the former relies on the missing v_i whereas the latter uses

the representation of the orthogonal sums. In this context it is worthwhile mentionning that, for the (15,7) BCH code with d=5, which is one of the examples in section 3.4, N_0 as given by theorem 3.2 is 37, N_0 from theorem 3.3 is greater than or equal to 28 and finally if theorem 3.5 is used N_0 is greater than or equal to 13. Theorem 3.2 gives the exact result but requires knowledge of the check sums. Theorem 3.3 is however less pessimistic than theorem 3.5 and should therefore be used instead of the latter in the case of $\sigma=0$. This leads to a combination of both theorem 3.3 and 3.5 as expressed in the following:

Theorem 3.6

$$N \geq \beta \triangleq \binom{n-1}{t} - B' + \sum_{\sigma=1}^{t-1} \binom{n-2\sigma-2}{t-\sigma-1} , \qquad (3.23)$$

where N is the estimate of the total number of error patterns of weight t+1 correctable by 1-step MLD.

To conclude this section, the case of the DS (difference set) codes which only have even weighted words will be discussed.

An (n,k) DS code is cyclic and also 1-step MLD, having an even minimum distance d=2t+2. The distance d and the length n are related through (d-1)(d-2)+1=n, so that in the d-1 orthogonal check sums similar to those of (3.7) there will be exactly (d-2) v_i's. The decoding procedure described in section 3.2 is also applicable to these codes.

Consider again the error patterns $E_0(x)$ of (3.11b) which have weight t+1 and an error in the zeroth position. If $E_0(x)$ is such that $\hat{C}_0=C_0$ then as seen before $\hat{C}(x)=C(x)$. While there exist $\binom{n-1}{t}$ error patterns of $E_0(x)$ of (3.11b), $(d-2)^t\binom{d-1}{t}$ of those are only detectable. In a DS code all patterns $E_0(x)$ will either be detectable or correctable so that M_0, the number of correctable $E_0(x)$ is:

$$M_0 = \binom{n-1}{t} - (d-2)^t \binom{d-1}{t}, \tag{3.24a}$$

or

$$M_0 = \binom{n-1}{t} - (2t)^t \binom{2t-1}{t}. \tag{3.24b}$$

For the same n,k and d, M_0 for the DS code is less than or equal to M_0 for another corresponding code, since in the case of DS codes, (n-1) v_i's are equally

distributed among d-1 orthogonal check sums so that the number of detectable $E_0(x)$ which are only detectable is maximised.

3.4 Some Experimental Results

In this chapter and especially in the previous section the extra error correcting capability of 1-step MLD has been demonstrated. This extra error correcting capability is inherent to the procedure itself and does not require any modification to the decoding algorithm hence to the decoding circuitry.

Section 3.3 also showed clearly that parallel operation, as proposed in section 2.3 for increasing the decoding speed, will not correct error patterns of weight greater than t, therefore sacrificing simplicity and extra error correcting capability to speed.

In this section, the complete 1-step MLD decoding of a few codes will be presented together with their decoding error probability. This probability will be given for a BDD algorithm and for a 1-step MLD procedure.

The table in figure 3.1 shows the (15,7,2) BCH code and all its shortenings when they are subjected to a 1-step MLD procedure and to a BDD decoding. In the case of the (15,7) and its subsequent shortenings, 1-step MLD decodes the codes completely that is to say uses all the available syndromes.

|E(x)|	15,7 MLD BDD	14,6 " "	13,5 " "	12,4 " "	11,3 " "	10,2 " "	9,1 MLD BDD
0	1 1	1 1	1 1	1 1	1 1	1 1	1 1
1	15 15	14 14	13 13	12 12	11 11	10 10	9 9
2	105 105	91 91	78 78	66 66	55 55	45 45	36 36
3	72	68	62	55	47	41	50
4	47	46	48	52	58	69	69
5	9	28	30	36	43	45	57
6	6	6	19	25	25	30	22
7	1	1	3	6	13	13	12
8	0	1	2	3	3	2	0

Figure 3.1, Decoding table.

In figures 3.2 to 3.9, graphs of the decoding error probability of the (15,7) and its shortenings are shown

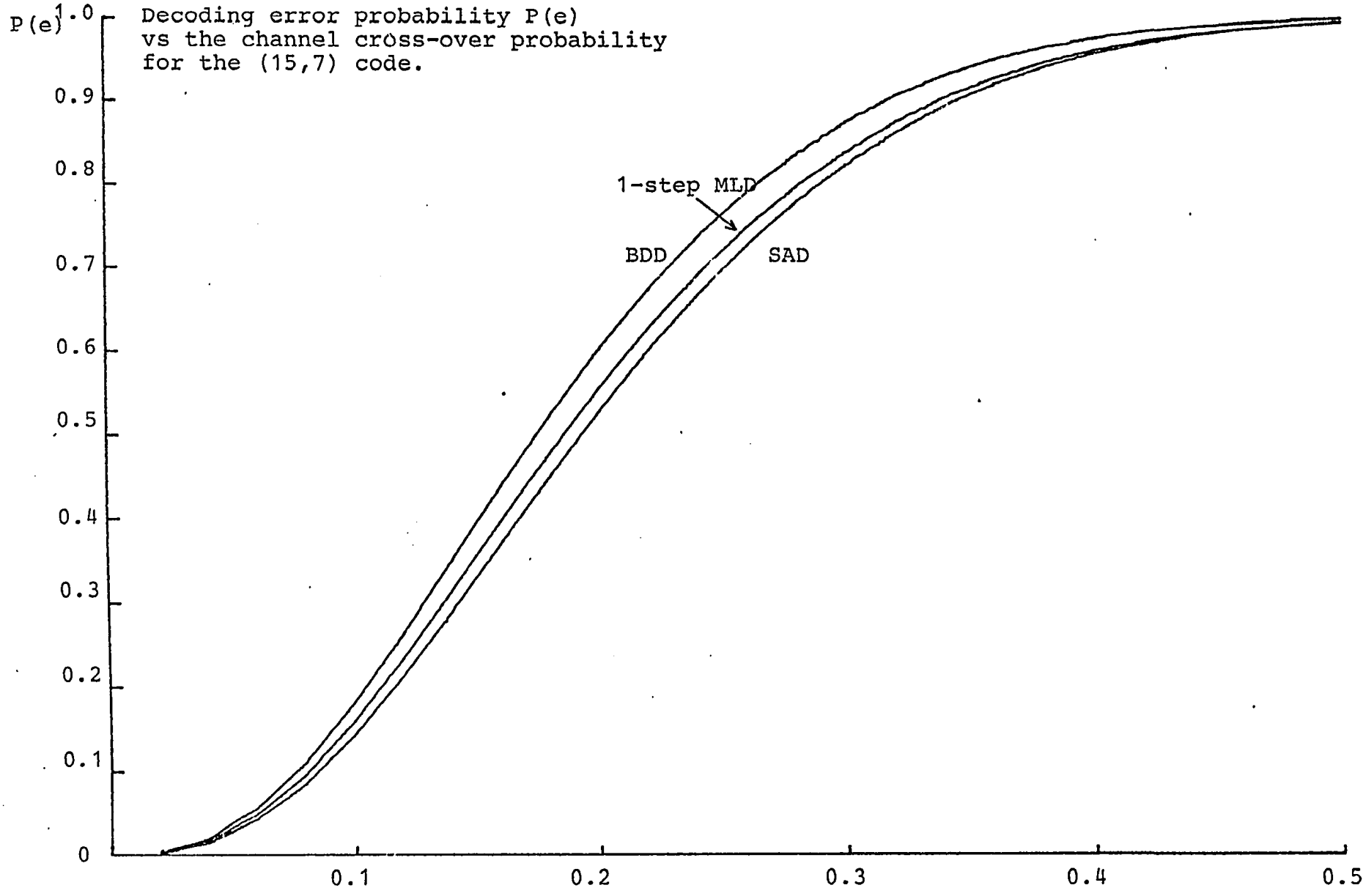

P(e)
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Decoding error probability P(e)
vs the channel cross-over probability
for the (15,7) code.
1-step MLD
BDD
SAD
0.1
0.2
0.3
0.4
0.5
Channel cross-over prob.

Figure 3.2.

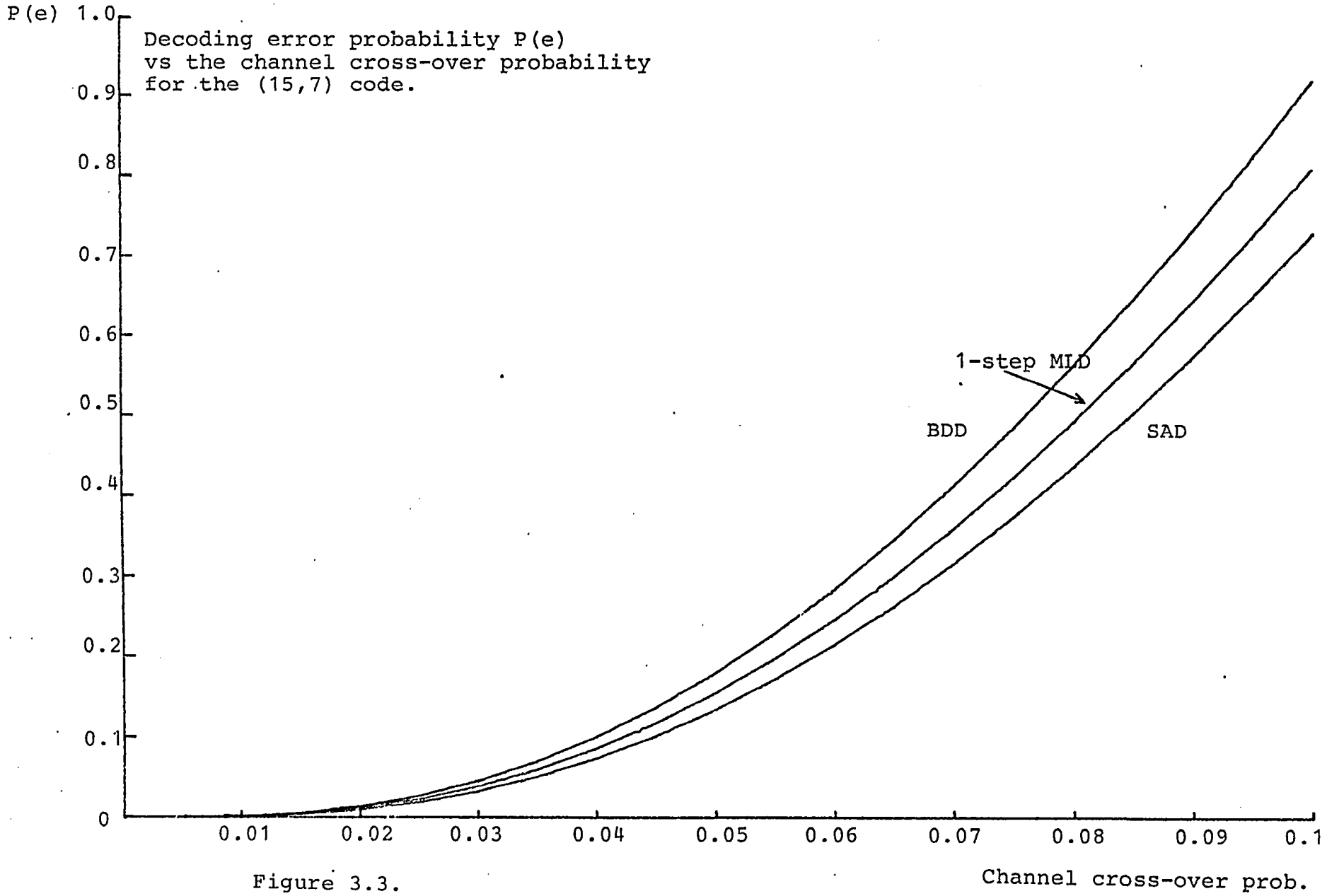

P(e) 1.0
Decoding error probability P(e)
vs the channel cross-over probability
for the (15,7) code.
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1-step MLD
BDD
SAD
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1
Channel cross-over prob.

Figure 3.3.

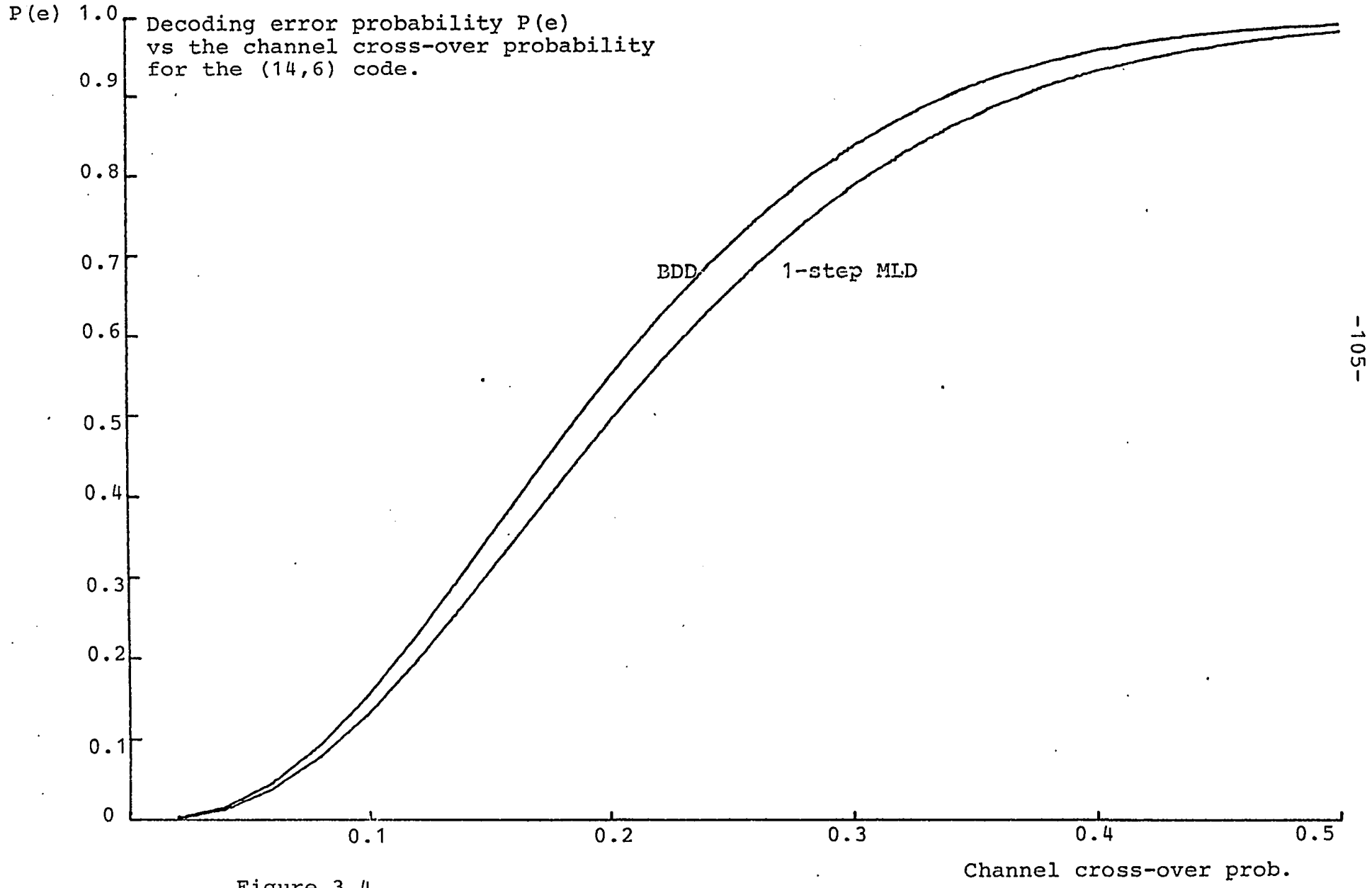

P(e)
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Decoding error probability P(e)
vs the channel cross-over probability
for the (14,6) code.
BDD
1-step MLD
0.1
0.2
0.3
0.4
0.5
Channel cross-over prob.

Figure 3.4.

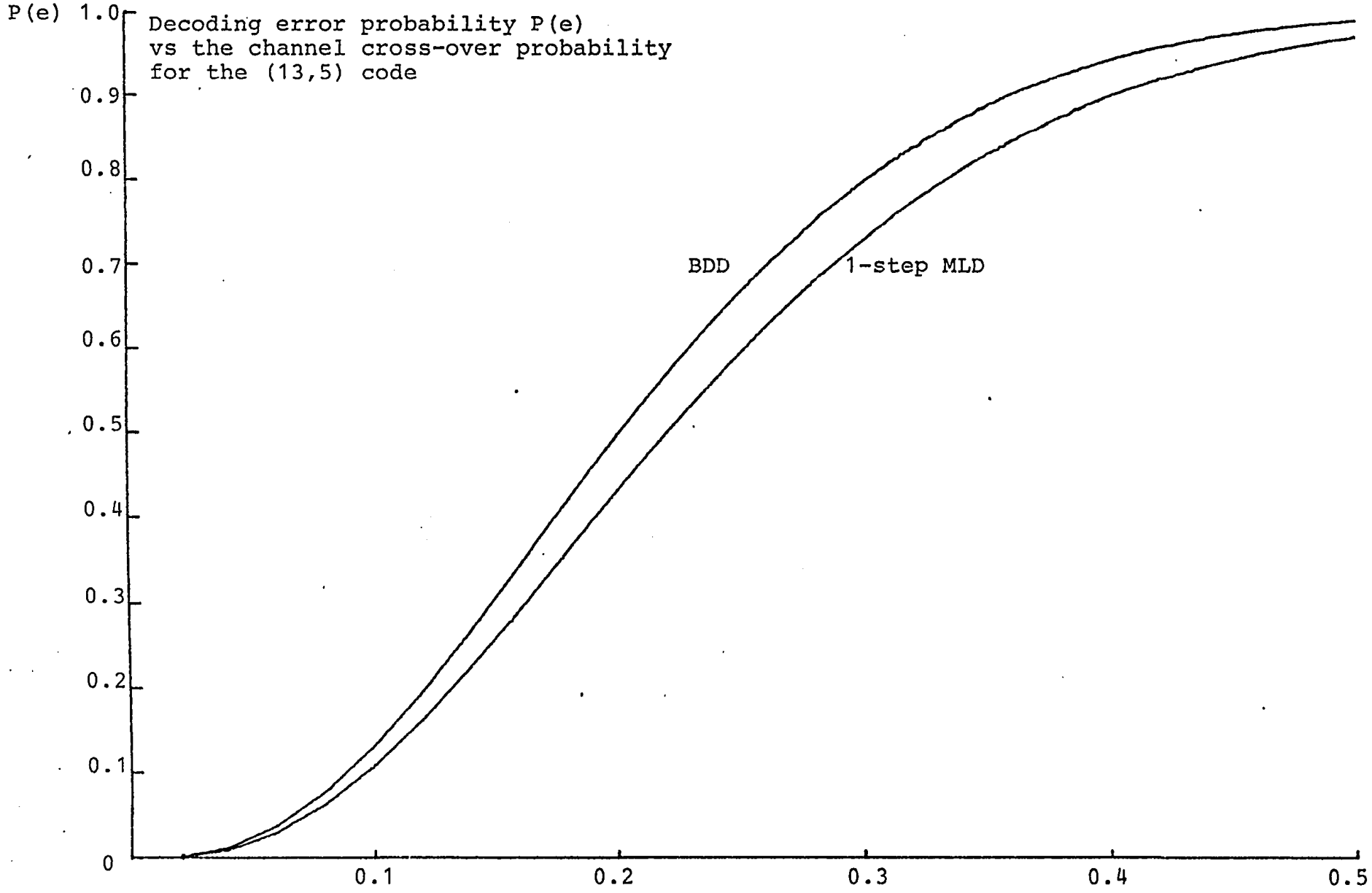

P(e)
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Decoding error probability P(e)
vs the channel cross-over probability
for the (13,5) code
BDD
1-step MLD
0.1
0.2
0.3
0.4
0.5
Channel cross-over prob.

Figure 3.5.

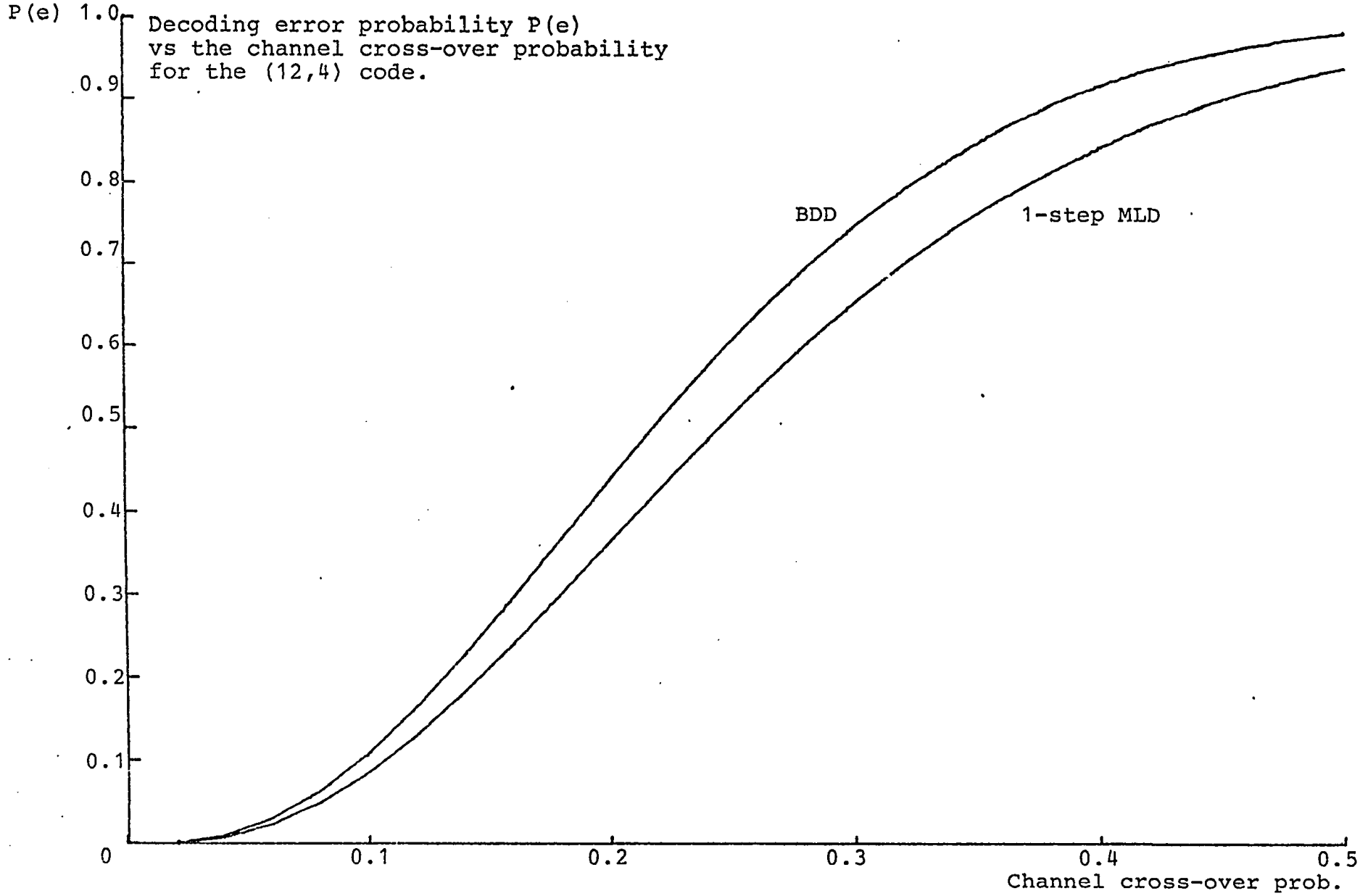

P(e)
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Decoding error probability P(e)
vs the channel cross-over probability
for the (12,4) code.
BDD
1-step MLD
0.1
0.2
0.3
0.4
0.5
Channel cross-over prob.

Figure 3.6.

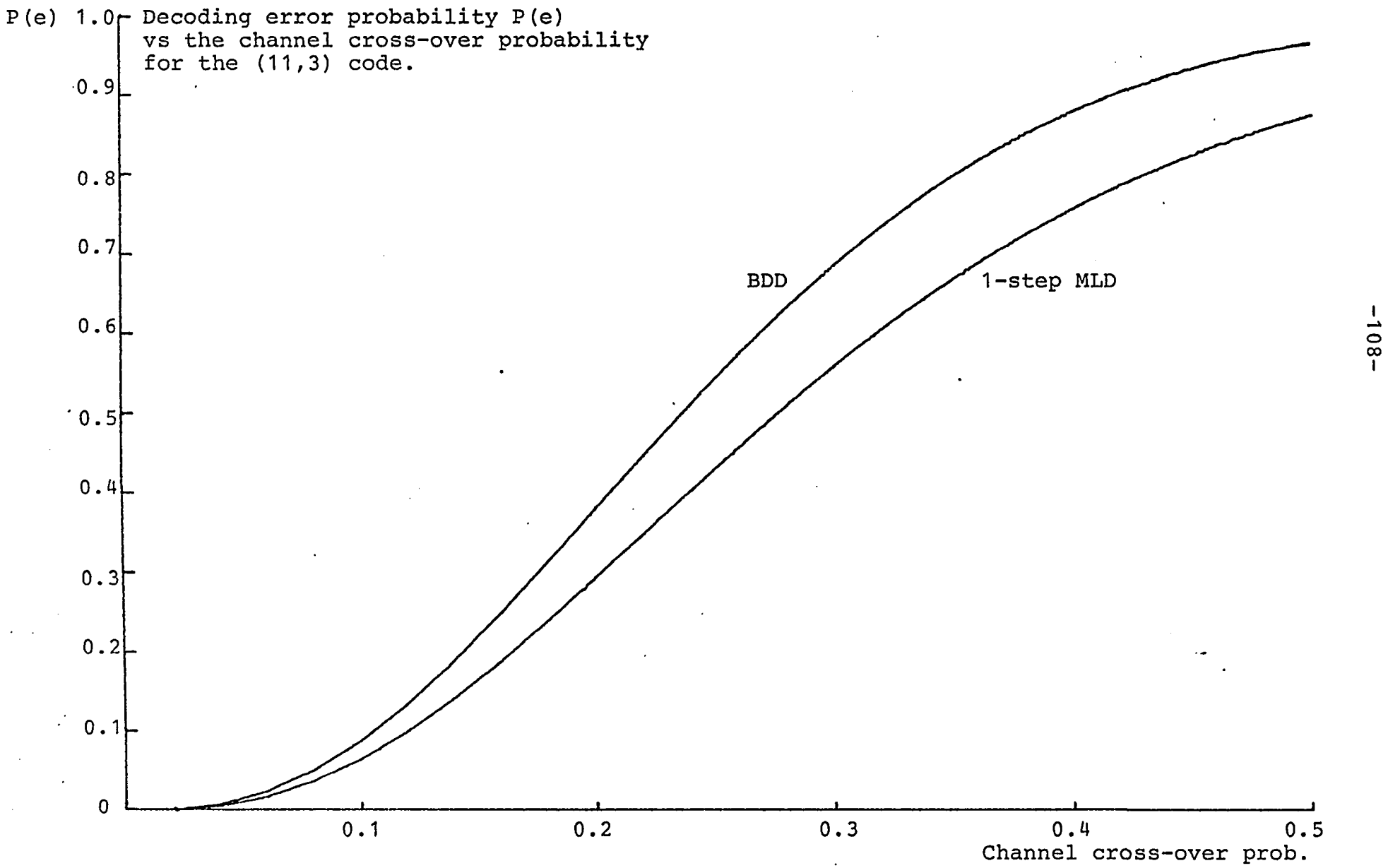

P(e)
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Decoding error probability P(e)
vs the channel cross-over probability
for the (11,3) code.
BDD
1-step MLD
0.1
0.2
0.3
0.4
0.5
Channel cross-over prob.

Figure 3.7.

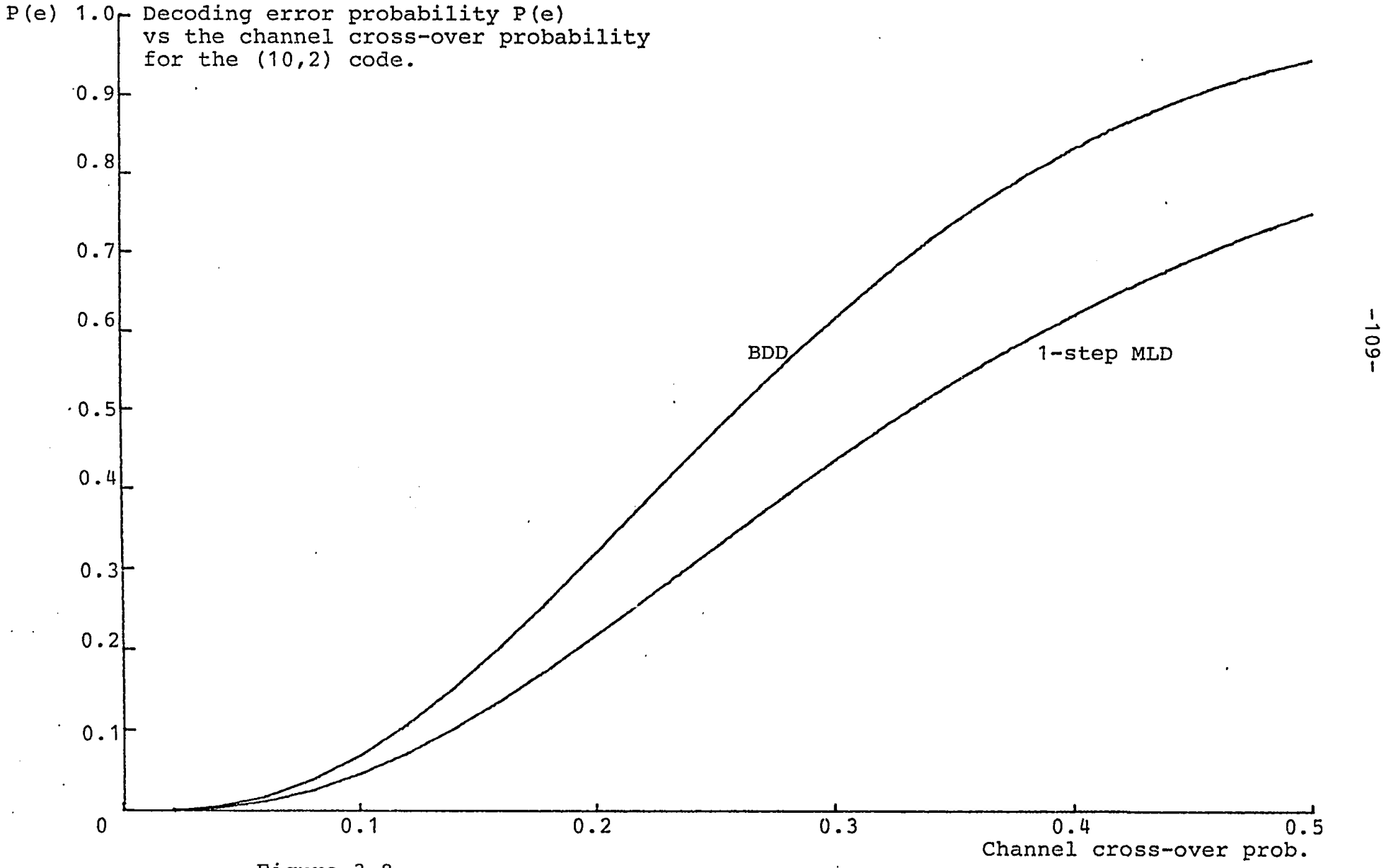

P(e) 1.0
Decoding error probability P(e)
vs the channel cross-over probability
for the (10,2) code.
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
BDD
1-step MLD
0.1
0.2
0.3
0.4
0.5
Channel cross-over prob.

Figure 3.8.

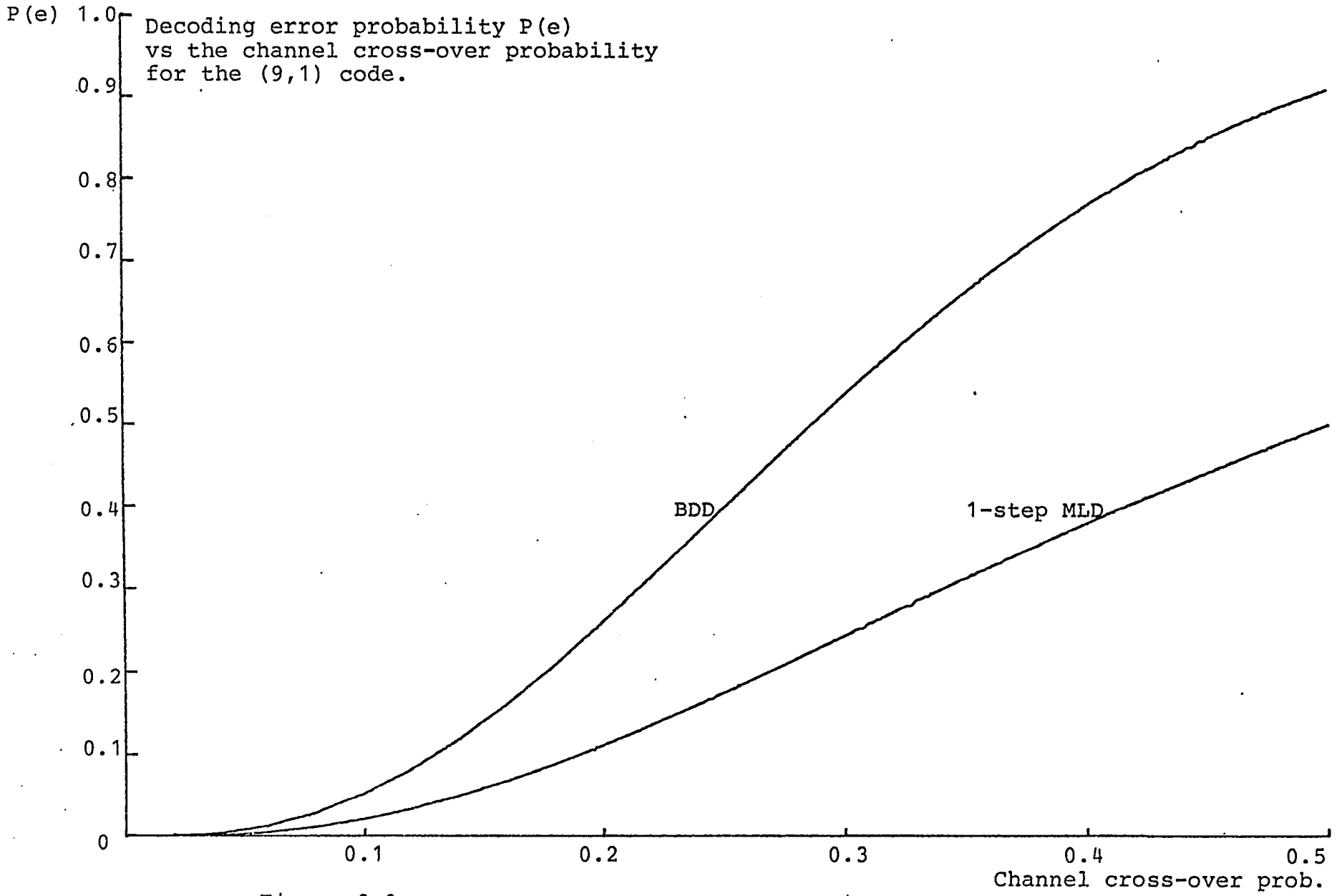

P(e)
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Decoding error probability P(e)
vs the channel cross-over probability
for the (9,1) code.
BDD
1-step MLD
0.1
0.2
0.3
0.4
0.5
Channel cross-over prob.

Figure 3.9.

for the two types of decoding mentionned before. In all cases they show clearly that 1-step MLD yields a lower decoding error probability than BDD (bounded distance decoding). For the (15,7) code which is 1-step MLD, figures 3.2 and 3.3 show three curves, the third one being the best performance obtainable using a decoding procedure using the properties of the standard array. This decoding algorithm would correct all error patterns of weight 0,1,2 and 135 error patterns of weight 3.

In all graphs, BDD yielded a higher decoding error probability. This is so because BDD does not make use of all the available syndromes. The decoding error probability can be expressed as:

$$P(e)= 1-\left[(1-p)^n+ (1-p)^{n-1}\, p\binom{n}{1} +\ldots.+ (1-p)^{n-t}\, p^t\binom{n}{t} + \right.$$

$$\left. + A_{t+1}\, (1-p)^{n-t-1}\, p^{t+1} + A_{t+2}\, (1-p)^{n-t-2}\, p^{t+2}+..+A_n(1-p)p^{n-1}\right]. \tag{3.25}$$

In the case of BDD, A_{t+1} to A_n are zero. If the code is perfect it does not matter, and moreover binary perfect codes are not 1-step MLD, (corollary 3.3). However when the code is not perfect,

$$1+\binom{n}{1}+\binom{n}{2}+\binom{n}{3}+\ldots.+\binom{n}{t}\neq 2^{n-k}\ , \tag{3.26}$$

and therefore A_{t+1} to A_n are not equal to zero. That is where 1-step MLD has an advantage over BDD although it is not felt at small p's, p being the channel cross over probability.

On the other hand, while not correcting error patterns greater than t, BDD detects the presence of some of them. This is so when the solution of the error locator polynomial cannot be found in the galois field over which the code is generated.

Consider a code V, ($n=2^m-1$, $k=n-2m$). The code is double error correcting binary BCH. A standard array for such a code would look like figure 3.10.

Weight	
0	V_1 V_2 ..... V_{n-1}
1	Minimum weight 4
2	Minimum weight 3
3	Minimum weight 3

Figure 3.10, Standard array for code V.

Suppose that A_d is the number of codewords of weight d, then the number of n-tuples of weight 3 with all of the cosets with leaders of weight $2=\binom{5}{2}A_5=10A_5$. Therefore the number of n-tuples of weight 3 which are detectable $=\binom{n}{3} -10A_5$.

If m is even,

$$A_5= \frac{n(n-3)^2}{120} . \tag{3.27}$$

The number of E(x) of weight 3 which detectable by BDD is then

$$= \frac{n(n-1)(n-2)}{6} - \frac{n(n-3)^2}{12} , \tag{3.28a}$$

or

$$= \frac{n}{12} (n^2 -5) , \tag{3.28b}$$

and the fraction of E(X) of weight 3 which are detectable is

$$= \frac{n^2 -5}{2(n^2 -3n +2)} . \tag{3.29}$$

As n gets large, (3.29) tends to 1/2.

To relate this to the previous section where some theoretical results where presented, the case of the

(15,7) can be of interest. As stated before 1-step MLD can correct some error patterns of weight t+1, where t is the error correcting capability of the code. For the (15,7) t+1 is 3 and 72 patterns of weight 3 can be corrected.

Position	0	1	2	3	4	5	6	7	8	9	10
No.	37	17	7	3	2	1	1	1	1	1	1

Figure 3.11, Correctable error patterns of w=3.

Figure 3.11 shows the distribution of correctable error patterns of weight t+1=3 for the (15,7). If the check sums are known, using theorem 3.2 will give N_0, the number of correctable error patterns of (3.11b), as being 37. On the other hand, if the check sums are not known, theorem 3.3 should be used. N_0 would then be greater than or equal to 28. Theorem 3.3 is therefore a more conservative and a more pessimistic estimate than theorem 3.2. The more complete theorem 3.6 would give N, the total number of correctable error patterns of weight t+1=3, as being greater than or equal to 29.

In the context of the previous remarks, it is

interesting to note that if permutation decoding [15] is applied to the (15,7) code, 90 error patterns of weight 3 will be corrected.

A code is said to be permutation decodable if and only if $k/n<1/t$. This condition is satisfied by the (15,7) code which is therefore permutation decodable.

In such a code, the weight of $[R(x)$ modulo $g(x)]$ is smaller than or equal to t if and only if $E(x)$ is contained in the first n-k positions. If, in the case of an error pattern of weight t or less, the weight of $[R(x)$ mod $g(x)]$ is greater than t, indicating that the errors are not contained in the first n-k positions, then there exists an α such that $\left|[(X^{\alpha}R(x))\ \text{mod}\ g(x)]\right|$ is smaller than or equal to t. X^{α} denotes that $R(x)$ is cyclically shifted α times.

Obviously the existence of α is not guaranteed if the error pattern has a weight greater than t. The procedure used in the decoding of the (15,7) was that, for a given error pattern E_1, all the possible shifts of $R(x)$ were considered. If, after performing the test for each value of shifting, the smallest weight was t+1, i.e.

3, and occured only when the errors were confined in the first n-k positions, then E_1 was said to be correctable. As said before, when applied to the (15,7) code, this procedure corrected 90 patterns of weight 3 as compared to 72 for 1-step MLD.

Finally, 1-step MLD has been applied to the (15,5) three error correcting code. This code is not 1-step MLD and the check sums used only guaranteed single error correction. Figure 3.12 shows how the correctable error patterns are distributed for both BDD and 1-step MLD using non orthogonal estimates and for a standard array type of decoding.

|E(x)|	Number of error patterns		
	BDD	1-step MLD	SAD
0	1	1	1
1	15	15	15
2	105	76	105
3	455	182	455
4	-	241	448
5	-	213	-
6	-	159	-
7	-	107	-
8	-	29	-
9	-	1	-

Figure 3.12, Decoding of the (15,5).

The decoding error probability of the (15,5) for those three algorithms is shown in figure 3.13. As the code is not 1-step MLD, the performance obtained using 1-step MLD is not as good as BDD for small values of p. However when a very simple method of decoding is required then 1-step MLD could be envisaged.

To conclude this section, the case of the (7,3) 1-step MLD single error correcting code with d=4 is considered. The (7,3) corrects all single errors and

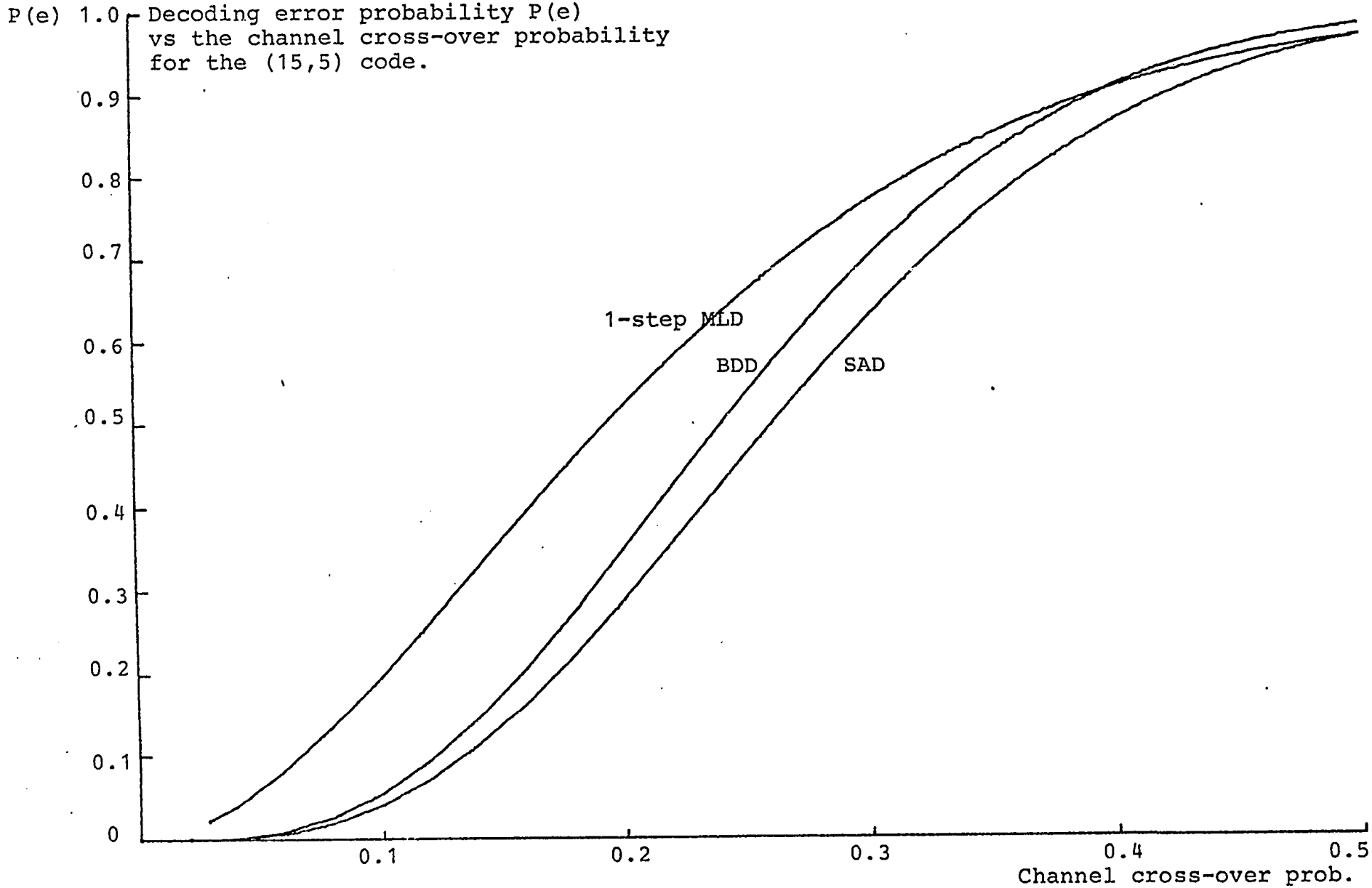

P(e) 1.0
Decoding error probability P(e)
vs the channel cross-over probability
for the (15,5) code.
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1-step MLD
BDD
SAD
0.1
0.2
0.3
0.4
0.5
Channel cross-over prob.

Figure 3.13.

none of greater weights. The code detects however error patterns of weight greater than t as shown in figure 3.14.

|E(x)|	No. of detected patterns	%
2	21	100
3	3	0.9
4	24	70
5	0	0
6	7	100
7	1	100

Figure 3.14, Detection power of the (7,3).

Codes with even minimum distance perform very well for error detection and could be considered when a scheme using forward error correction and error detection followed by retransmission is used.

CHAPTER 4

Microprocessor Implementation of an MLD Decoder

4.1 Introduction

This chapter deals with the feasibility of practically implementing an MLD decoding algorithm with a microprocessor instead of using conventional hard wired logic. The former alternative has both advantages and drawbacks over the latter as will be seen later on. From this, a choice can be made for possible applications of coding, and a few examples of some systems where a microprocessor could be used, are given.

4.2 Definition of a microprocessor

A microprocessor is basically the integration on a single silicon chip of some or all of the necessary circuitry of the CPU (central processing unit) of a digital computer. Figure 4.1 [45] shows a simple block diagram of such a machine. Roughly speaking the microprocessor is made out of an ALU (arithmetic logic unit), a number of registers (internal memory), an

instruction decoder and a timing circuit. Most of these elements are linked by an internal bi-directional data bus, which is used for transfer of information, namely addresses, data and control signals, inside the microprocessor.

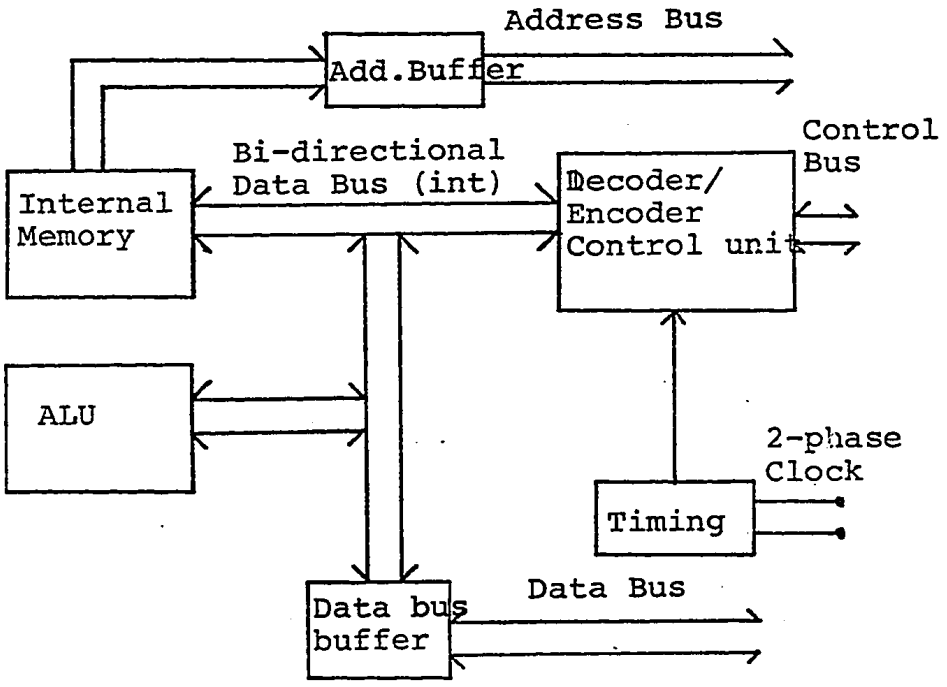


Figure 4.1, Microprocessor block diagram.

Under normal operating conditions, the microprocessor sends out an address to memory and receives an instruction. It then executes the instruction, that is to say performs certain tasks on the data and then, after the completion of the sequence goes to the next instruction. The microprocessor thus performs

exactly like the CPU of a digital computer and if interconnected with input/output facilities and memory, figure 4.2 [46], it then becomes a microprocessor based computer, often known as a microcomputer [47,48].

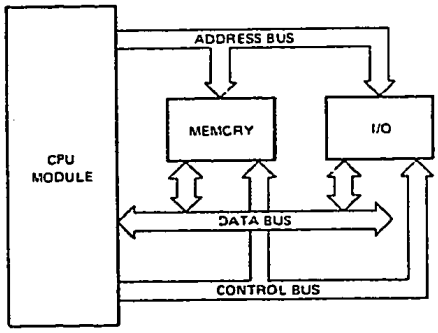


Figure 4.2, Microcomputer block diagram

On data obtained from its input ports, the microcomputer performs tasks which are equivalent to logical operations, and communicates the results via its output ports. Therefore these stored programme sequences can replace conventional hardwired logic, that is gates and flip flops.

4.3 Reasons for using microprocessors

There are three basic reasons why one would use a

microprocessor,

1) The time and cost involved in developing a given product (in this case an MLD decoder) are reduced significantly. The configuration of a microcomputer is pretty standard thus making the hardware designer's task a lot simpler. On the other hand writing a programme, the execution of which will replace the hard wired logic, is much easier than designing a complex circuitry. This is illustrated in figure 4.3 [49], which shows the development cost vs the complexity of the circuit for different options.

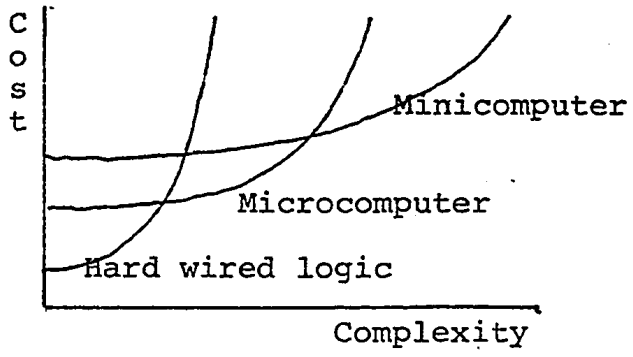


Figure 4.3, Cost vs complexity

2) Because of the software nature of the implementation, it lends itself very well to changes during the prototyping states. Very often, features which would have been too

expensive to include in a hard wired design, can now be implemented at a marginal cost. Therefore the microprocessor is providing a greater flexibility as well as enhanced capabilities.

3) The reliability of a microprocessor based unit is greater than that of a hard wired circuit. This is mainly due to the fact that a microcomputer will reduce the number of interconnections required. Normally a 1Kbit x 8 bit ROM (read only memory) will replace at least 512 gates or roughly 50 IC chips [50]. A 16-pin IC chip usually introduces 36 connections . Therefore, eliminating 50 IC chips means 1800 connections will not have to be made. As interconnections are one of the major sources of trouble in hard wired logic, it is normal to expect an increase in reliability when a microcomputer is used.

On the other hand a microprocessor system has its own drawbacks. The main reason why hard wired logic is still a strong contender is basically speed. Typically, the microprocessor based MLD decoder could not handle high rates like 56 Kbits/s, let alone those in the

megabit region. What it means is that basically software is slower than dedicated hardware for any given task. In the beginning of this section, hardware was no match for the microprocessor because of this very lack of flexibility which gives it its speed.

Besides being slowed down by software the microprocessor suffers from another shortcoming as far speed is concerned and that is integration technology. Due to the complexity of the functions to be put on a silicon chip, MOS (metal oxide semiconductor) technology had to be used because it yielded very high packing density and low power consumption. Unfortunately MOS is much slower than TTL (transistor transistor logic) schotky bipolar technology although manufacturers keep on refining and improving it. As a comparison, the Intel 4040 which is of a P-MOS construction has an instruction cycle time of 10.8 microseconds [51], and the newer N-MOS Intel 8080 reduced it down to 1.3 microseconds [52].On the other hand, the Intel 3000 series, which is a bit slice microprogrammable microprocessor made with the TTL schotky bipolar technology will go as far down as 100 nanoseconds and straight TTL logic is around 50 nanoseconds cycle time.

While there is no doubt as to whether or not the technology will improve the speed of the microprocessor, one could also argue that the speed of conventional logic will also improve. Therefore it would appear that there is a choice betweeen the flexibility offerred by the software oriented microprocessor and the speed of hard wired logic. In cases where speed is not critical and is within the range of operation of a microprocessor, the flexibility and the reliability obtained by the use of such a unit should be more than enough to make it a good choice.

4.4 Basic operations

There are basically five tasks that the microprocessor has to perform in order to decode the received the binary sequence if an MLD algorithm is used.

1) Input the n bit word,
2) Form and evaluate the parity-check equations, which are of the form of (3.7).
3) Take a majority vote,
4) Output the decoded bit
5) Shift the word and repeat 2,3 and 4 as many times

as there are information bits to be decoded.

Although a microcomputer can do all these, it is interesting to see how well it does them and how easy it is for it to do them. Most fixed word microprocessors are not microprogrammable, that is they have a fixed instruction set. In this context it would be appropriate to see if there are instructions that, if added in future machines, would improve the situation.

First of all, item 1 and 4, namely input/output operations are pretty standard in all microprocessors. Generally the I/O's are parallel operations and a UART (universal asynchronous receiver transmitter) can be used to interface them with the lines serial data.

To form the parity-check equations is fairly simple. The codeword being in the accumulator, a mask is applied (figure 4.4) and a jump on even or odd parity can be used to perform the evaluation of the check-sum. This operation can be repeated for the other check-sums, the only thing to change being the mask and also the word has to be reloaded into the accumulator. Here again, referring to figure 4.4, it is clear that changing the

mask from 00100110 to 01101000 would give an equation with v_1 , v_2 and v_4 as components as opposed to v_2 , v_5 and v_6 obtained previously.

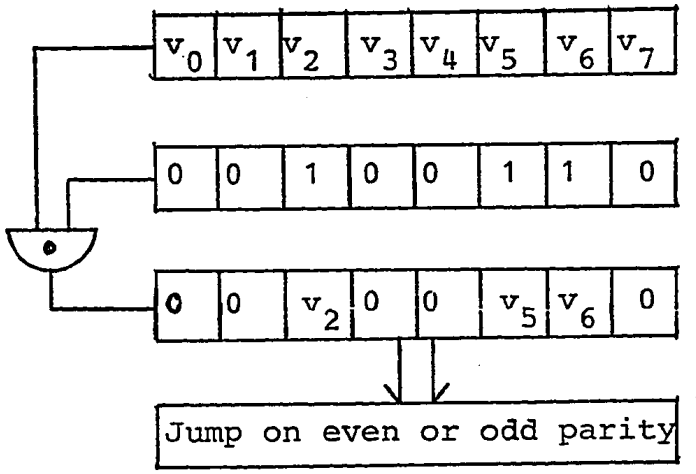


Figure 4.4, Check sum evaluation.

Using the stack or some internal registers, it is possible to store the result of the check-sums for a majority vote to be taken later on.

The majority vote is perhaps the most awkward thing to perform on a microprocessor. It usually calls for adding all the results of the check-sums, then performing a subtraction and finally testing if greater than zero or

not. Normally, an MLD code would have d-1 estimates besides the d_{th} estimate of $v_0=v_0$, therefore the amount to be subtracted is (d-1)/2. This is only a description of what has been used in the simulation. Some other techniques could be used and might be better than this one. Nevertheless, a simpler method for majority vote would be useful.

The last operation to be performed is a shift and is very simple to do since most microprocessors have shift instructions in both left and right directions.

4.5 The ideal microprocessor

All the previous discussion supposed that the codeword was equal to or shorter than the computer word length. Most microprocessors in use today have an eight bit word length, contradicting the above assumption. To overcome this difficulty, the codeword has to be broken into multiples of the processor word length, hence many transfers between memory locations containing the codeword and the accumulator will result, and that at any stage of the decoding process.

This is where a bit-slice machine comes in very handy. The processor word length can be tailored exactly to the codeword, thus saving those multiple transfers. Besides this, two advantages are present, namely speed and the possibility to microprogramme the instructions. As most bit-slice microprocessors are built in schotky bipolar or ECL (emitter coupled logic) technology, the speed of operation is very fast. Furthermore microprogramming adds another level of flexibility by giving the user the opportunity to define his own instruction set. In MLD decoding applications, it would be useful to have instructions implementing tasks such as,

1) giving access to individual bits in the codeword. This would help forming the check-sums especially if used with a form of stack and mod-2 adder onto which could be dumped the different components of the equation (figure 4.5), and the output of which would be the result of a mod-2 addition of all the components of the same equation.

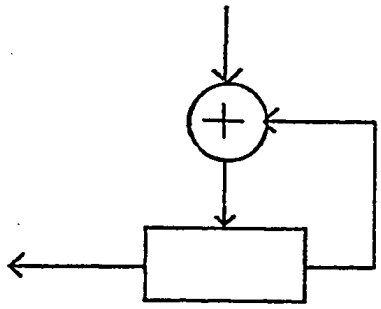


Figure 4.5, Stack and mod-2 adder

2) taking a majority vote on the result of all the check-sums. This could be implemented in a microprogramme and simply calling the instruction could perform the vote onto a predetermined portion of the accumulator, whilst putting the result on the stack or in some other registers.

Other than that, the user always has the choice of microprogramming his MLD decoding routine directly, without taking the option of defining an instruction set. This would be a very hard way of doing it but would provide a fast execution.

4.6 Some practical examples

To conclude this section, it would be interesting to look at the architecture of systems where a microprocessor could be advantageously used.

Consider the case of a low rate (2.4 or 4.8 Kbits/s) remote terminal. If the line is noisy, some coding could be used and if implemented with a microprocessor the system could look like the one in figure 4.6.

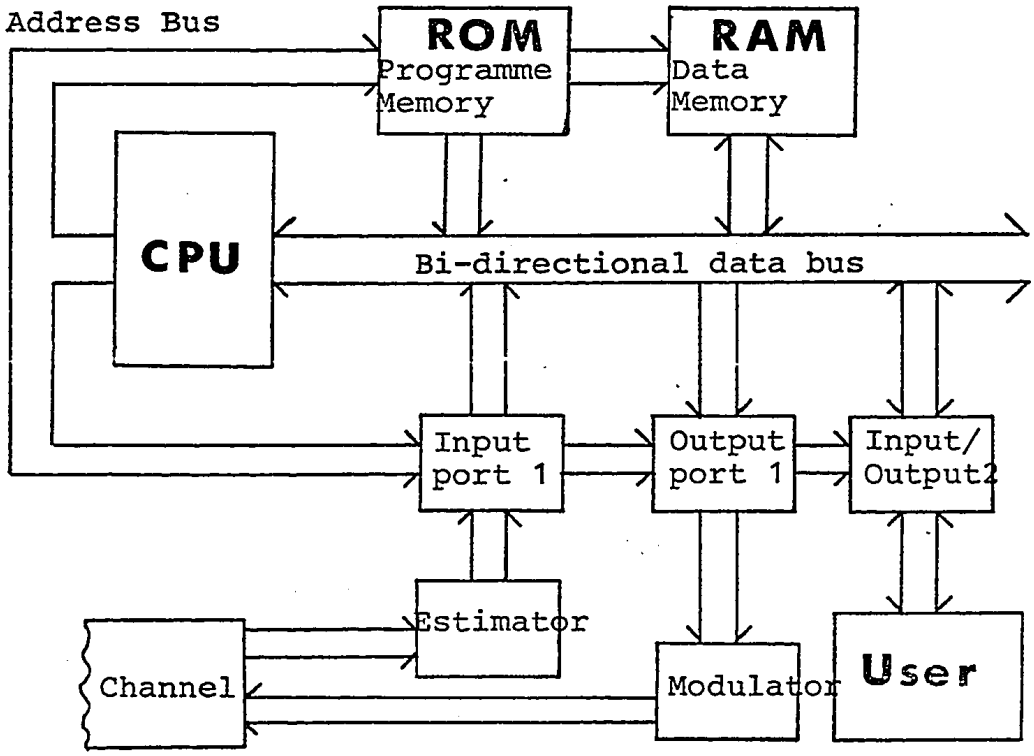


Figure 4.6, Communication facility with coding.

The arrangement described in figure 4.6 is quite

standard. Binary data is obtained by the CPU from the estimator via the input port 1. The codeword received is decoded according to instructions contained in the ROM programme memory. Intermediate results are stored in RAM memory and finally the decoded output is sent to the user via the bi-directional port 2.

Any data originating from the user is transmitted to the CPU via this same port 2. There, it is coded and then sent to the modulator via output port 1, thus completing the process of two-way communication.

Another example of possible application of microprocessor implemented coding is in the case of a large memory bank serving different users . Figure 4.7 describes such a system, where the memory is made out entirely of high capacity semiconductor chips. The word length is n bits and the dedicated microprocessor will decode it into k bits for transmission to different users. Obviously data can also be stored and in that case the microprocessor would code and store the data in the memory at the address specified by the user's computer.

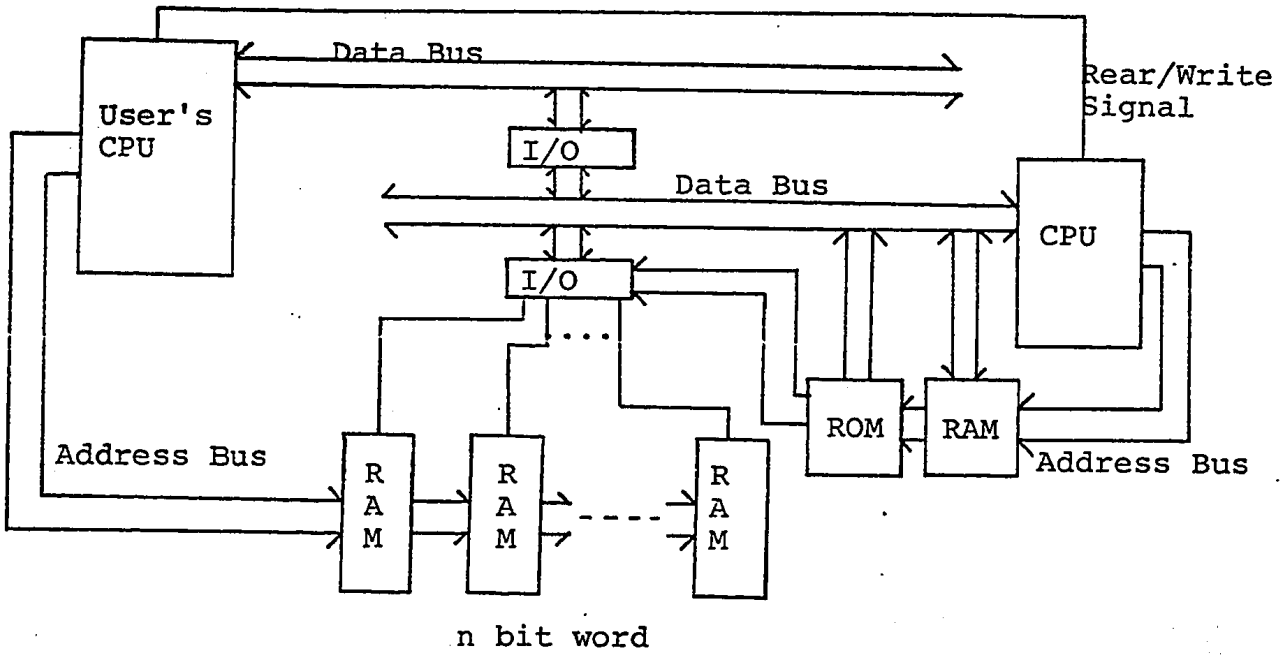


Figure 4.7, A mass storage memory

At the moment it may be too early to see the trends emerging but already a Viterbi decoder designed to operate in real time has been implemented using the Intel 3000 series [53] mentionned earlier. To summarise, the microprocessor could popularise the use of coding and make it a technique widely used by the communication engineer.

CHAPTER 5

CONCLUDING REMARKS

The main point of this thesis was in chapter 3 where 1-step MLD has been investigated. It has been established that 1-step MLD is a very attractive procedure for practical implementation. This is so mainly because of the simplicity of the algorithm as shown in section 2.2 and 2.3 and also because of the extra error correcting capability inherent to the procedure as shown in section 3.3 and 3.4.

The task of estimating the number of error patterns of weight greater than t which 1-step MLD can correct is a difficult one indeed. For this reason, error patterns of weight t+1, due to their higher probability of occurence than those of greater weights, were considered first. Even there, the usefulness of the investigation is restricted to error patterns of the form $E_0(x)$, that is always having an error in the zeroth position. For patterns like these, the correction of C_0 is the only condition on the correctability of the patterns. However when patterns of the form $E_1(x)$ are considered, the

correction of C_0 and C_1 is the condition on their correctability. If A_0 is the set of patterns for which C_0 is properly estimated and A_1 the set of patterns for which C_1 is properly estimated, due to the reshuffling of the check sums between the estimation of C_0 and C_1, it is yet impossible to determine the intersection of the sets A_0 and A_1. The same remarks also applies in the case of patterns of the form $E_\sigma(x)$ where $C_0, C_1, \ldots C_\sigma$ have to be estimated properly for $E_\sigma(x)$ to be correctable. Furhter research in this area would be needed, in particular the investigation of the parity check matrix of the code as a way of estimating the extra correctable error patterns, should prove interesting.

Theorem 3.5 and its special case theorem 3.4 of section 3.3, are very loose estimates relying only on the missing v_i and not on the check sums themselves. For example, in the case of the (15,7) code, N_1, the number of correctable $E_1(x)$, is 17. Theorem 3.4 would give the value of N_1 to be greater than or equal to 1. This definitely needs refining.

An interesting possibility of estimating the extra correctable error patterns would be to shorten the code

to its generator polynomial and then simulate. If some sort of rules were derived to determine the properties of the full length code from its generator polynomial, it would be a very practical way of doing the estimation of these correctable patterns.

Finally, a list of all correctable error patterns of weight greater than t, for the (15,7) BCH code, is to be found in appendix 1.

Appendix 1

Complete 1-step MLD of the (15,7) code

This appendix is basically a list of all the error patterns of weight greater than t that 1-step MLD, when applied to the (15,7) code, can correct.

The patterns are listed by weight and only powers are given instead of the binary form. Therefore 101000000010000 would be found as 0,2,10.

Weight 3

0,1,3	0,1,5	0,1,7
0,1,10	0,2,5	0,2,6
0,2,10	0,2,14	0,3,5
0,3,7	0,3,10	0,4,5
0,4,10	0,4,12	0,4,13
0,5,6	0,5,7	0,5,8
0,5,9	0,5,10	0,5,11
0,5,12	0,5,13	0,5,14
0,6,10	0,6,14	0,7,10
0,8,9	0,8,10	0,8,11

0,9,10	0,9,11	0,10,11
0,10,12	0,10,13	0,10,14
0,12,13	1,2,6	1,3,6
1,3,7	1,3,11	1,5,6
1,5,11	1,5,13	1,5,14
1,6,7	1,6,10	1,6,14
1,7,11	1,8,11	1,9,10
1,9,11	1,10,11	1,10,12
2,3,7	2,4,12	2,6,7
2,6,12	2,6,14	2,10,11
2,10,12	3,5,13	3,7,8
3,7,13	4,6,14	4,8,9
5,9,10	6,10,11	7,8,11
8,9,12	9,10,13	10,11,14

Weight 4

0,1,2,6	0,1,3,6	0,1,3,7
0,1,3,11	0,1,6,7	0,1,6,14
0,1,7,11	0,1,8,11	0,1,9,11
0,2,3,7	0,2,4,12	0,2,6,7
0,2,6,12	0,2,6,14	0,3,7,8
0,3,7,13	0,4,6,14	0,4,8,9
0,5,9,10	0,7,8,11	0,8,9,12

1,2,3,7	1,2,4,7	1,2,4,12
1,2,7,8	1,3,4,8	1,3,5,13
1,3,7,8	1,3,7,13	1,4,8,9
1,5,9,10	1,7,13,14	1,8,9,12
2,3,8,9	2,4,6,14	2,4,8,9
2,4,8,14	2,5,9,10	2,6,10,11
2,11,13,14	3,5,9,10	3,7,11,12
4,8,12,13	7,9,11,12	7,12,13,14
8,10,12,13	9,11,13,14	

Weight 5

0,1,3,5,13	0,2,6,10,11	0,8,10,12,13
1,7,9,11,12	2,8,10,12,13	2,8,11,12,13
3,7,11,12,13	7,10,11,12,13	8,11,12,13,14

Weight 6

0,1,7,9,11,12

0,2,8,11,12,13

0,3,7,11,12,13

0,8,11,12,13,14

1,3,9,12,13,14

1,6,7,11,13,14

Weight 7

3,5,7,9,11,13,14

In the case of error patterns of weight greater than t, it is interesting to see that extra protection was given to positions 0,1,2,..., in that order of importance. This extra protection is dependent on the order of estimating the C_i's which was, in the procedure outlined in this thesis, 0,1,2,... . This accounts for the decreasing amount of protection given to the codeword digits, starting from the zeroth position and on.

This extra prorection is not necessarily restricted to particular digits. Some other positions i could also be protected. This could be done by simply shifting the word to the left by i positions and estimating the corresponding information polynomial B(x) which is not C(x). Then B(x) is multiplied by g(x), shifted to the right by i positions and divided by g(x) to yield C(x).

REFERENCES

[1] C.E. Shannon, "A mathematical theory of communications," Bell System Technical Journal, Vol. 27, pp. 379-423, pp. 623-656, 1948.

[2] J.A. Heller and I.M. Jacobs, "Viterbi decoding for satellite and space communications," IEEE Transactions on Communications Technology, Vol. Com-19, No. 5, pp. 835-848, October 1971.

[3] "Analysis and measurements of the error correcting properties of the Viterbi algorithm," Electronic Applications Bulletin, Vol. 33, No. 2, pp. 45-58, N.V. Phillips Gloeilampenfabrieken, Eindhoven, October 1975.

[4] R.T. Chien, "Block coding techniques for reliable data transmission," IEEE Transactions on Communications Technology, Vol. Com-19, No. 5, pp. 743-751, October 1971.

[5] G.D. Forney Jr., "Burst correcting codes for the classic bursty channel," IEEE Transactions on Communications Technology, Vol. Com-19, No. 5, pp. 772-781, October 1971.

[6] K. Brayer, "Error correction code performance on HF, troposcatter and satellite channels," IEEE Transactions on Communications Technology, Vol. Com-19, No. 5,

pp.772-781, October 1971.

[7] R.W. Hamming, "Error detecting and error correcting codes," Bell System Technical Journal, Vol. 29, pp.147-160, 1950.

[8] A. Hocquenghem, "Codes correcteurs d'erreurs," Chiffres, Vol. 2, PP.147-156, 1959.

[9] R.C. Bose and D.K. Ray-Chauduri, "On a class of error correcting binary group codes," Information and Control, Vol. 3, pp. 68-79, March 1960.

[10] W.W. Peterson and E.J. Weldon Jr., "Error correcting codes," MIT Press, 2nd edition, pp. 52-56, 1972.

[11] Reference 10, pp.283-307.

[12] E.R. Berlekamp, "Algebraic coding theory," McGraw-Hill Book Co., pp. 178-199, 1968.

[13] Reference 10, pp. 310-356.

[14] Shu Lin, "An introduction to error correcting codes," Prentice-Hall, pp.87-111, 1970.

[15] J. Macwilliams, "Permutation decoding of systematic codes," Bell System Technical Journal, Vol. 43, pp. 485-505, January 1964.

[16] E.R. Cacciamani, "The Spade system as applied to data communications and small earth station operation," Comsat Technical Review, Vol. 1, No. 1, pp. 171-181, Autumn 1971.

[17] A.J. Viterbi, "Convolutional codes and their performance in communication systems," IEEE Transactions on Communications Technology, Vol. Com-19, No. 5, pp. 751-772, October 1971.

[18] G.D. Forney Jr., "Coding and its applications in space communications," IEEE Spectrum, pp. 47-58, June 1970.

[19] I. Matsushita, Y. Tsuji and Y. Sohma, "Applications of error correcting codes to TDMA satellite communications," International Conference on Communications, Seattle, pp. 31,7-31,12, 1973.

[20] "Error control products," Linkabit Co., sales pamphlet.

[21] R.T. Chien, "Memory error control: beyond parity," IEEE Spectrum, pp. 18-23, July 1973.

[22] R.T. Chien, "Coding theory and the computer," Proceedings of the 6th Allerton Conference, pp. 175-177.

[23] L. Levine and W. Meyers, "Semiconductor memory reliability with error detecting and correcting codes," Computer, pp. 43-50, October 1976.

[24] J.J. Stiffler, "Editorial of special issue on coding theory," IEEE Transactions on Communications Technology, Vol. Com-19, No. 5, pp. 741-742, October 1971.

[25] E.J. Weldon Jr., "Some results on majority logic decoding," Error Correcting Codes, edited by H. Mann, John Wiley and Sons, pp. 149-161, 1968.

[26] "Designing with TTL integrated circuits," Texas Instruments Electronics Series, McGraw-Hill Book Co., pp.128-131, 1971.

[27] "Mcmos handbook," Motorola inc., pp. 5,8-5,15, pp. 12,9, 1974.

[28] Reference 12, pp. 136.

[29] Reference 12, pp. 196.

[30] T.C. Bartee and D.I. Schneider, "Computation with finite fields," Information and Control, Vol. 6, pp. 79-98, 1963.

[31] Reference 10, pp. 305-307.

[32] C.E. Sundberg, "One-step majority logic decoding with symbol reliability information," IEEE Transactions on Information Theory, pp. 236-242, March 1975.

[33] L.D. Rudolph, "Threshold decoding of cyclic codes," IEEE Transactions on Information Theory, Vol. IT-15, No. 3, pp.414-418, May 1969.

[34] L.D. Rudolph, "A class of majority logic decodable codes," IEEE Transactions on Information Theory, Vol. IT-13, pp. 305-307, April 1967.

[35] L.D. Rudolph and W.E. Robbins, "One-step weighted majority decoding," IEEE Transactions on Information Theory, pp. 446-448, May 1972.

[36] S.W. Ng, "On Rudolph's majority logic decoding algorithm," IEEE Transactions on Information Theory, Vol. IT-16, pp. 651-652, September 1970.

[37] Reference 14, pp. 154-161.

[38] S.G.S. Shiva, R. Provost and S.E. Tavares, "A class of binary single error correcting codes," Conference Records, IEEE ICC, pp. 36,15-36,19, 1971.

[39] S.G.S. Shiva, "Class of majority logic decodable codes with a minimum distance of 4," Electronics Letters, Vol. 12, No. 22, pp. 599-600, October 1976.

[40] S.G.S. Shiva and S.E. Tavares, "On binary majority logic decodable codes," IEEE Transactions on Information Theory, Vol. IT-20, pp. 131-133, January 1974.

[41] J.L. Massey, "Threshold decoding," MIT Press, pp. 86-103, 1963.

[42] C.V. Chinh, "A study of negacyclic codes," M.A.Sc. thesis in electrical engineering, University of Ottawa, pp. 4146, 1974.

[43] C. Laferriere and S.G.S. Shiva, "On 1-step majority logic decoding," IEE Proceedings, Vol. 124, No. 6,

June 1977.

[44] Reference 10, pp. 333-334.

[45] A. Sabatier, "Microprocessor, mesure, regulation et automatisme," Mesure Regulation et Automatisme, pp. 62, September 1975.

[46] "Intel 8080 microcomputer user's manual," Intel Co., pp. 31, September 1975.

[47] "Microprocessor applications manual," Motorola Semiconductor Products inc., McGraw-Hill Book Co., 1975.

[48] A. Osborne, "An introduction to microcomputers," A. Osborne & associates inc., Vol. 1, 1976.

[49] Notes of ELG 7182-7183, "Topics in computer architecture," M. Krieger, University of Ottawa, 1976-77.

[50] "Intel MCS-40 user's manual for logic designers," Intel Co., 2nd edition, 3rd printing, pp. iii-vii, March 1975.

[51] Reference 48, pp. v.

[52] Reference 46, pp. 5,13.

[53] J. Conan and R. Oliver, "Hardware and software implementation of the Viterbi decoding algorithm for convolutional codes," Proceedings of the International Symposium on Mini and Micro Computers, pp. 190-193,1976.