JPL PUBLICATION 78-107

Approximate Maximum Likelihood Decoding of Block Codes

H. J. Greenberger

February 15, 1979

National Aeronautics and Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

JPL PUBLICATION 78-107

Approximate Maximum Likelihood Decoding of Block Codes

H. J. Greenberger

February 15, 1979

National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

The research described in this publication was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under NASA Contract No. NAS7-100

ABSTRACT

Approximate maximum likelihood decoding algorithms, based upon selecting a small set of candidate code words with the aid of the estimated probability of error of each received symbol, can give performance close to optimum with a reasonable amount of computation. By combining the best features of various algorithms and taking care to perform each step as efficiently as possible, a decoding scheme was developed which can decode codes which have better performance than those in use today and yet not require an unreasonable amount of computation. The discussion of the details and tradeoffs of presently known efficient optimum and near optimum decoding algorithms leads, naturally, to the one which embodies the best features of all of them.

CONTENTS

Figures

Tables

SECTION 1

INTRODUCTION

The quest for practical error correcting schemes with better performance at low signal-to-noise ratios has lead to the realization that one must use codes of high Q (distance × rate) and decode them with algorithms whose probability of error approach or equal the theoretical minimum for the code being used. For codes whose words are equally likely and which are transmitted over a memoryless channel, the optimum decoder is a correlator which calculates the distance between the received vector and all possible code words and selects as the best estimate of the transmitted code word that one which is closest to it. Because of the amount of computation required, it is unreasonable to decode directly in this way all but the smallest codes, whose error correcting power is relatively weak. Many schemes have been suggested which reduce the computational complexity while maintaining the desired performance. These schemes can be roughly divided into two groups: those that use code structures for which highly efficient decoding algorithms, equivalent to optimum decoders, are known (for example, convolutional codes decoded using the Viterbi algorithm, see Reference 1-1); and those that use non-optimum decoding algorithms whose performance is close to optimum but whose complexity is much less than the optimum algorithm.

One class of non-optimum decoding algorithms, which select a small number of candidate code words to be correlated with the received vector, can approach maximum likelihood performance even at low signal-to-noise ratios. The best of these algorithms take advantage of a preliminary sorting of the bits of the received vector according to absolute magnitude in order to arrange them in order of their estimated probability of being correct. These algorithms then select candidate code words such that, on the average, those bits which are most probably correct almost always keep the same value as the hard limited received vector, while those which are considered unreliable change sign more often. Each of the algorithms discussed here utilizes the constraints of the parity check equations in a different way in order to generate a set of code words while adhering to this general principle.

One of the first algorithms of this kind was developed by D. Chase (Reference 1-2). He suggested perturbing the hard limited received code work $\underline{Y}$ by adding it, modulo 2, to a test pattern $\underline{T}$ to obtain a new sequence $\underline{Y}'$. This new sequence is decoded algebraically to find the unique code word (if one exists) within a distance $\lfloor (d-1)/2 \rfloor$ (d = minimum distance between code words) of $\underline{Y}'$. If all the binary sequences of weight $\lfloor d/2 \rfloor$ and of length n (n = number of bits in code word) are used for test patterns, all code words within a distance d-1 of $\underline{Y}$ will be in the set $\{\underline{Y}' = \underline{Y} + \underline{T}\}$. This two-fold increase over the error correcting capability of a conventional binary decoder makes up for the loss of non-optimum decoding and, at high signal-to-noise ratios, the performance asymptotically approaches that of a maximum likelihood decoder. This scheme is not practical, however, since, except for short codes of small minimum distance, the number of test sequences is very large. The information on the reliability of the received

symbols can reduce this number to a practical value, for example, by discarding test patterns with many ones in bit positions corresponding to reliable bits. In this and in a later paper (Reference 1-3), Chase describes methods of constructing sets with a reasonable number of patterns which not only approach maximum likelihood performance at high signal-to-noise ratios, but do so also at low signal-to-noise ratios.

L. Baumert, R. McEliece and G. Solomon (References 1-4 and 5-2) have done much work using sets of erasure patterns, of which only a small amount has been published. In this technique a set of bits, equal to the number of redundant bits in the code, is erased. A candidate code word is then generated by reconstructing these bits from the unerased ones. Each erasure pattern generates another candidate to be correlated with the received vector.

The reasoning behind using erasure patterns rather than error patterns is that the redundancy of binary codes is much greater than its error correcting power. For instance, the (128,64) BCH code used as an example in this report has 64 redundant bits but can correct only 10 errors. On the other hand, to correctly decode a received word, this scheme must cover all hard decision errors. Using error masks, a number or errors, up to the correcting power of the code, may remain exposed. The efficiency of both schemes is dependent upon the set of masks used and much thought has gone into finding methods of generating efficient sets. This will be discussed further in later sections describing each decoding scheme in detail.

If only the single most probable erasure pattern is used, the probability of an error remaining in the unerased bits is considerably greater than the probability of decoding wrongly using a maximum likelihood decoder. B. Dorsch (Reference 1-5) noted that the unerased bits have a high probability of being correct and maximum likelihood performance could be approached by generating candidate code words by assuming error patterns of low weight in these bits.

A decoding algorithm that tries all error patterns of weight w or less in the unerased bits must generate and check

$$\sum_{l=1}^{w} \binom{k}{l}$$

candidate code words. The large number of patterns required is somewhat offset by the ease in generating them. However, a slight modification results in a considerably more efficient algorithm.

If one does not erase all of the redundant bits, then the ensuing redundancy can be used to eliminate many error patterns in the remaining bits. Combining erasures and redundancy was implicit in Forney's error and erasure algorithm (Reference 1-6) which can be thought of as shortening the code a bit at a time by successively erasing the most unreliable bits and algebraically decoding the shortened code. This scheme does not work well, however, at low signal-to-noise ratios.

One that does was developed by E. Berlekamp (Reference 1-7) for correcting a combination of burst errors and a small number of random errors and has been adapted here for random errors in linear codes. With a redundancy of r bits, on the average only 1 out of 2^r error patterns satisfies the r parity check equations. Increasing the number of unerased bits increases the total number of error patterns and, since the added bits are of lower reliability, the probability of errors of higher weight also increases. For small values of r, the first factor predominates and the computational efficiency of the algorithm increases significantly.

If the maximum weight of the error in the unerased bits that need be considered is 2, then the generation of error patterns consistent with the parity check equations is considerably simplified. For the (128,64) BCH code many of the more probable error patterns of weight 3 must be tried in order to approach maximum likelihood performance, especially when redundant bits are left unerased. On the other hand, Baumert's and McEliece's algorithm, which uses many erasure masks, can approach maximum likelihood performance without considering errors in the unerased bits at all. This suggests that these algorithms can be combined into a hybrid scheme which uses a small number of erasure masks, a few bits of redundancy, and consideration of error patterns of weight 2 or less to achieve comparable performance. With a suitable selection of parameters such an algorithm is computationally more efficient than any one of its ancestors and is a possible competitor, in terms of complexity versus performance, to Viterbi decoding of convolutional codes.

An alternate approach to selecting a set of candidate code words is to disregard some of the channel information and make an optimum decision on what remains. This method works well at low signal-to-noise ratios where many bits have a high probability of error and can be disregarded with only a small penalty in performance. Such an algorithm can be put into an iterative form. Starting with an initial estimate of a set of independent bits, the estimates can be improved by considering bits related to the initial set through the parity check equations of the code. These redundant bits are considered one at a time, in order of increasing probability of error. Each bit improves the estimate of the previous bits, and the ones with a higher probability of error have a smaller effect than those with a lower probability of error. As poorer and poorer bits are examined, they perturb the previous estimate less and less and a point is reached where the algorithm may be stopped with a high probability of being close to the optimum estimate.

The code that was chosen to compare the various algorithms was the (128,64) BCH code of rate 1/2 and minimum distance 22. It was chosen because it is one of the shortest codes whose maximum likelihood performance is at least 1 dB better than that of the rate 1/2 constraint length 7 convolutional code. Figure 1-1, which compares the performance of a number of rate 1/2 codes, shows this clearly. These curves were obtained by Baumert and McEliece (Reference 1-4), who estimated the maximum likelihood performance by increasing the number of candidate code words generated by their algorithm until most of the wrong decodings were identifiably maximum likelihood errors. That is, the code word chosen as the best estimate had a higher correlation with the received vector than the known transmitted code word.

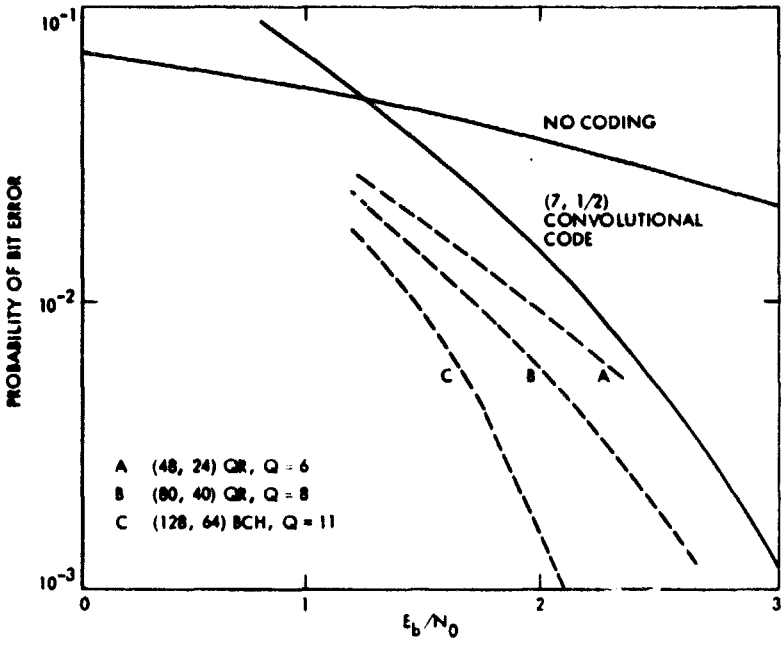

Fig. 1-1. Estimated Maximum Likelihood Performance of Several Rate 1/2 Block Codes.

The results on the estimated performance of various decoding schemes were made using computer-generated bit error patterns. A random error vector was added to the vector (1,1,...,1,1), corresponding to the all zero code word, and the components of the resulting vector sorted according to decreasing absolute magnitude. The location of the symbols with negative values indicated the pattern of errors that would be made by a bit by bit hard decision decoder. If the decoding algorithm could erase or correct all these errors, then a correct decoding was assumed. This gives a lower bound to the performance of the decoder since, in order to decode correctly, the error pattern must be erased or corrected. This does not guarantee, however, that a maximum likelihood error will not occur. This bound was found to be quite close to the actual performance of the decoding algorithm when operating more than 0.5 dB away from the maximum likelihood curves.

Since the received vectors in most cases were not actually decoded, the probability of information symbol error also had to be estimated. Given the probability of code word error, a lower bound on the bit probability of error could be found by assuming all errors result in estimating code words located the minimum distance from the transmitted code word. For the (128,64) BCH code of minimum distance 22,

$$\frac{\text{Pr error (symbol)}}{\text{Pr error (word)}} \geq \frac{22}{128} = \frac{1}{5.8}$$

This ratio is asymptotically correct at high signal-to-noise ratios. At low signal-to-noise ratios in the range considered here, a ratio of 1/5 yields a good estimate of the symbol probability of error.

SECTION 2

AN ITERATIVE ALGORITHM APPROXIMATING MINIMUM PROBABILITY OF SYMBOL ERROR DECODING

2.1 INTRODUCTION

Decoding algorithms that minimize the probability of symbol error are relatively rare because of the more complex form of such a decoder. One of the first attempts to reduce the amount of computation was that of L. Bahl, J. Cocke, F. Jelinick, and J. Raviv (Reference 2-1), who used a trellis to represent the possible states of the decoder. The amount of computation required, however, was n (n = block length or decoding constraint length) times that of a Viterbi decoder for the same code. The next step was taken by Hartmann and Rudolph (Reference 2-2), who showed how to transform the decoding equations into the space of the dual code to a form that required much less storing of intermediate results. Their algorithm still required n times the computation of a minimum probability of code word error algorithm.

The algorithm of Rudolph and Hartmann consists of generating a test statistic for each bit which is compared to a threshold. The statistic takes the form of the sum of a large number of terms, one for each code word in the dual code. By sorting the bits of the received vector according to probability of error, the order of the terms can be permuted so that the terms contributing significantly to the sum are considered first. The optimum estimate is then approached after only a small fraction of the terms have been summed (Reference 2-3).

The ordering of the terms is a natural consequence of sorting the received bits according to increasing probability of error and decoding in the space of the dual code. It can be thought of as converting the algorithm to an iterative form. The k most reliable bits are independent and their value is initially estimated by hard limiting the received vector. The estimate can be improved by successively looking at bits related to them through the parity check equations of the code. These redundant bits are examined in order of increasing probability of error. Each bit, as it is examined, improves the estimate of the previous ones. However the redundant bits with a higher probability of error have a smaller effect than those with a lower probability of error. As poorer and poorer bits are examined, they perturb the previous estimate less and less and a point is reached where the algorithm may be stopped with a high probability of being close to the optimum estimate.

The algorithm, in its present form, is not as efficient as the ones described in subsequent sections for a number of reasons. The computation primarily consists of the products and sums of real numbers rather than of binary vector additions, and common with the other minimum probability of symbol error algorithms, it requires a separate calculation for each symbol.

2.2 A DESCRIPTION OF THE ALGORITHM

As each symbol is received, the likelihood ratio,

$$\phi_m = \frac{Pr(r_m|C_m = 0)}{Pr(r_m|C_m = 1)}$$

is calculated and mapped into the region (-1, +1) by the transformation

$$\rho_m = \frac{1 - \phi_m}{1 - \phi_m} .$$

When the entire code word has been received, the symbols are sorted according to increasing probability of error (or equivalently, magnitude of ρ_m) so that the least reliable bits are to the right. The columns of the parity check matrix of the code are then permuted to the same order as the symbols. By using row operations only, the permuted parity check matrix can be reduced to a form which has a triangle of zeros in the upper right hand corner. The first p rows of this matrix represent the dependence between the k + p symbols with the least probability of error. If the remaining n - (k + p) symbols are considered erased, then an "optimum" decision, in the sense of minimum probability of symbol error, can be made using only this portion of the matrix. This form of the matrix also leads to an iterative algorithm. Starting with p = 0 and increasing p by one each iteration, successively poorer received bits are considered in estimating the transmitted symbols, until for p = n - k the "true" optimum estimate is reached.

The decoding rule, for minimum probability of symbol error, using the method of decoding in the dual space of the code, is from equation (13) in Hartmann and Rudolph (Reference 2-2).

$$m = \sum_j \prod_l \rho_l^{\,C'_{jl} \bullet \delta_{ml}} \underset{H_0}{\overset{H_1}{\gtrless}} 0$$

where

$$\delta_{ml} = 1 \text{ if } m = l$$

$$\delta_{ml} = 0 \text{ if } m \neq l$$

C'_{jl} is the lth bit of the jth code in the dual code, whose code words are formed by a linear combination of the rows of the parity check matrix. When estimating the mth bit, the term ρ_l is included in the product if the lth bit of $C'_j = l$ and $m \neq l$, or the lth bit of $C'_j = 0$ and $m = l$.

As an initial estimate of the transmitted symbols (p = 0), let $\hat{C}_m = 1$ if $\rho_m > 0$ and $C_m = 0$ if $\rho m \leq 0$. The first iteration uses only a single parity check equation so that there are only two words in the dual code--the all zero code word and the one equal to the first row of the parity check matrix. The all zero vector contributes to Λ_m the term P_m, which is the initial estimate, and the single parity check equation contributes a single product term of the ρ'*ls*.

The algorithm is then iterated, each time adding another parity check equation and taking into consideration the "best" of the remaining received bits. At each iteration the number of terms in the sum is doubled. Because of the reduced form of the parity check matrix, with zeros in the upper right hand corner, the terms Λ_m from previous iterations are not changed when a new row of the matrix is added and the new terms can be added directly to the previous stage's estimate. At each iteration the estimate of each bit is improved and the bit probability of error decreases.

2.3 AN EXAMPLE USING THE (23,11) GOLAY CODE

A detailed example of this algorithm will be given for the (23,11) Golay code. The block length of this code is long enough to see the convergence of the algorithm to the optimum solution as the number of iterations increase and yet short enough that a complete decoding can be done in a reasonable time. In order to estimate the performance of the algorithm at a given signal-to-noise ratio, a large number of received vectors were generated and fully decoded. Only the 12 most reliable bits are estimated by the algorithm. The remainder are calculated through the parity check matrix. This forces the estimate of the transmitted vector to be a code word. The estimate of these bits was stored after each iteration and the code word was considered correctly decoded when the 12 bits indicated as most reliable were estimated correctly.

Since the code is linear, the code space looks the same when viewed from any code word. Therefore, it can be assumed, without loss of generality, that the all zero code word is transmitted. The received word can be represented by a vector of dimension 23, $y = (y_1, y_2, \ldots, y_{23})$. Each element of the vector is of the form $y_i = 1 + n_i$ where n_i is a sample of a zero mean Gaussian process of variance

$$\sigma^2 = \frac{N_o}{2E_bR} .$$

As a numerical example, consider a received code word which contains four hard errors and therefore cannot be decoded correctly using algebraic decoding. In addition, one of the errors is among the 12 best bits so that the initial estimate of this algorithm would also be in error. A received vector (SNR = 1.0 d) and the corresponding likelihood ratios are tabulated in Table 2-1.

Table 2-1. Amplitude, Likelihood Ratio, and Transformed Likelihood Ratio for Bit Locations

m	1	2	3	4	5	6	7	8	9	10	11	12
A_m	.99	1.25	.44	-.63	.52	.25	.01	.97	.36	-.90	1.13	.41
Λ_m	.074	.041	.26	3.04	.22	.41	.70	.808	.32	5.62	.054	.28
ρ_m	.86	.92	.39	-.50	.64	.42	.17	.86	.32	-.70	.90	.56

	13	14	15	16	17	18	19	20	21	22	23
A_m	.11	.06	1.12	.33	-.74	-.24	.94	1.86	.71	2.87	1.73
Λ_m	.36	.62	.054	.33	3.90	1.24	.083	.010	.14	.001	.013
ρ_m	.28	.24	.90	.50	-.59	-.11	.85	.98	.75	.99	.97

m = Bit number.
A_m = Amplitude of mth bit.
Λ_m = Likelihood ratio of mth bit.
ρ_m = Transformed likelihood ratio of mth bit.

The first step of the algorithm is to sort the bits according to the absolute values of the transformed likelihood ratios ρ_m. The sorted order, shown in Table 2-2, is:

Table 2-2. New Bit Labeling Ordered by Transformed Likelihood Ratio

Sorted Bit Order	1	2	3	4	5	6	7	8	9	10	11	12
Original Bit Order	22	20	23	2	11	15	8	1	19	21	10	5
Sorted ρ_m	.99	.98	.97	.92	.90	.90	.86	.86	.85	.75	-.10	.64

Sorted Bit Order	13	14	15	16	17	18	19	20	21	22	23
Original Bit Order	17	3	12	9	16	4	6	13	14	7	18
Sorted ρ_m	-.59	.59	.56	.52	.50	-.50	.42	.28	.24	.17	-.11

The original parity check matrix for this code is

```
101001001111100000000000
010100100111110000000000
001010010011111000000000
000101001001111100000000
000010100100111110000000
000001010010011111000000
000000101001001111100000
000000010100100111110000
000000001010010011111000
000000000101001001111100
000000000010100100111110
000000000001010010011111
```

The columns are permuted to the same order as the received symbols to yield

```
000010100010011100011000
000110000010001010011110
000011010001011000011100
000001000000000111111100
000001000011100000101110
000110100001000011010100
000001001000101101000011
010000011010100001010001
010010001100100100000101
110001001110001000000001
111010001100000000101000
```

Reducing this matrix by row operations to form a triangle of zeros in the upper right hand corner yields

```
001110101101100000000000
111100000111010000000000
011011011000101000000000
110010010100011100000000
100001011001001110000000
110011010011000001000000
010001011100000101100000
100001010100101001001000
100011000010101100001000
110000000110100101000010
110001001110001000000001
```

The estimator

$$\Lambda_m^{(i)} = \sum_j \prod_\ell \rho_\ell^{C'_{j\ell} \oplus \delta_{m\ell}}$$

where i = iteration number will be used with this matrix and the sorted ρ_m's. For the ith iteration the code words C'_j are formed by all linear combinations of the first i rows of the matrix.

The initial estimate can be thought of as using the estimator with a dual code consisting of the all zero vector alone. The index j, which indicates the jth code word in the dual code, has only the value 1, and $C'_{j\ell}$ is zero for all values of ℓ. In this case,

$$\Lambda_m^{(o)} = \prod_\ell \rho_\ell^{\delta_{m\ell}} .$$

Since $\delta_{m\ell} = 0$ for $m \neq \ell$ and 1 for $m = \ell$, $\Lambda_m^{(o)} = \rho_m$. When the only code word in the dual code is the all zero one, the code itself contains all possible binary n-tuples. All the bits are independent and the best estimate of any bit depends only upon the likelihood ratio of that bit itself.

The first iteration begins to use the dependence between the bits as expressed by the first parity check equation (the first row) of the H matrix

$$h_1 = (00111010110110000000000)$$

There are only two code words in the dual code; the all zero code word and

h_1 so that $\Lambda_m^{(1)} = \rho_m + \dfrac{\rho_3\ \rho_4\ \rho_5\ \rho_7\ \rho_9\ \rho_{10}\ \rho_{12}\ \rho_{13}}{\rho_m}$ for m = 3, 4, 5, 7, 9, 10, 12, 13

$$= \rho_m + \rho_m\ \rho_3\ \rho_4\ \rho_5\ \rho_6\ \rho_7\ \rho_9\ \rho_{10}\ \rho_{12}\ \rho_{13} \quad \text{otherwise.}$$

Note that each term corresponds to a code word in the dual code. Each term contains the product of the transformed likelihood ratios of all the bits in the code word which are equal to one multiplied or divided by the ratio of the mth bit.

The second iteration uses two parity check equations h_1 and h_2.

$$h_2 = (11110000011101000000000)$$

The four code words in the dual code are

$$C_1 = 0\ h_2 + 0\ h_1 = 00000\ 00000\ 00000\ 00000\ 000$$

$$C_2 = 0\ h_2 + 1\ h_1 = 00111\ 01011\ 01100\ 00000\ 000$$

$$C_3 = 1\ h_2 + 0\ h_1 = 11110\ 00001\ 11010\ 00000\ 000$$

$$C_4 = 1\ h_2 + 1\ h_1 = 11001\ 01010\ 10110\ 00000\ 000$$

At this step the purpose of permuting the columns of the H matrix and reducing it to one with all zeros in a triangle in the upper right hand corner becomes clear. First, the code words in the dual code at the ℓth iteration contain all the code words of the (ℓ-1st) iteration so that

$$\Lambda_m^{(\ell)} = \Lambda_m^{(\ell-1)} + 2^{(\ell-1)}$$

new products. These new products are formed from the code words generated by adding modulo 2 the new parity check equation h_ℓ to all the code words of the previous iteration. Second, all the new products include the transformed likelihood ratio of the (K + ℓ)th bit but not of bits less reliable than this. Thus the th iteration uses the previous estimate and the parity check equation containing the best bit not yet used to obtain an improved estimate.

At each iteration the estimate as to which bits are best may change. This is seen in Figure 2-1 where the $\Lambda_m^{(\ell)}$'s are plotted as a function of iteration number. As more parity check equations are used, the absolute value of Λ (which is a measure of goodness) of the bits whose initial estimate was wrong, decreases relatively rapidly. At the seventh iteration the best 13 bits are correct and the code word would be decoded correctly if the algorithm would be stopped at this point. At the final iteration, using the full decoding algorithm (equivalent to maximum likelihood decoding), the best 17 bits are correct. Of the four bits whose initial estimate was wrong, only the one with the poorest initial estimate was corrected. In cases where there are fewer errors or the likelihood ratios of the correct bits are initially higher, the wrong bits are also corrected, but this is not necessary for correct decoding using this algorithm.

The performance of this code as a function of number of iterations is shown in Figure 2-2. The zero iteration curve illustrates the performance possible by assuming there are no errors in the best 12 bits, while the 11th iteration curve represents the performance using the full decoding algorithm. Note that in the sixth iteration, using $2^6 = 64$ terms in the sum of the estimator, the performance is almost equivalent to the full decoding algorithm which requires $2^{11} = 2048$ terms in the sum.

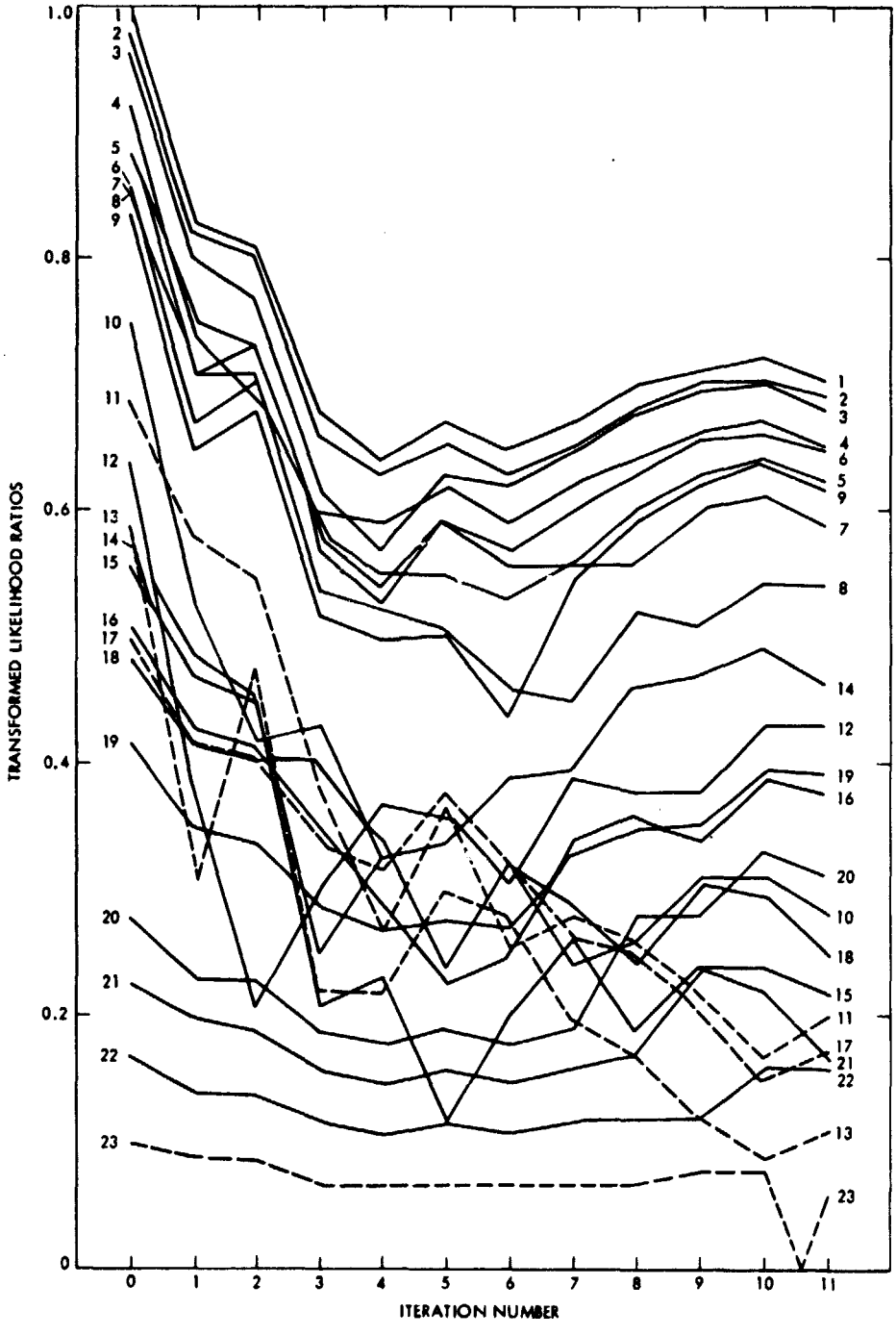

Figure 2-1. Improvement of Bit Estimates as a Function of Iteration Number

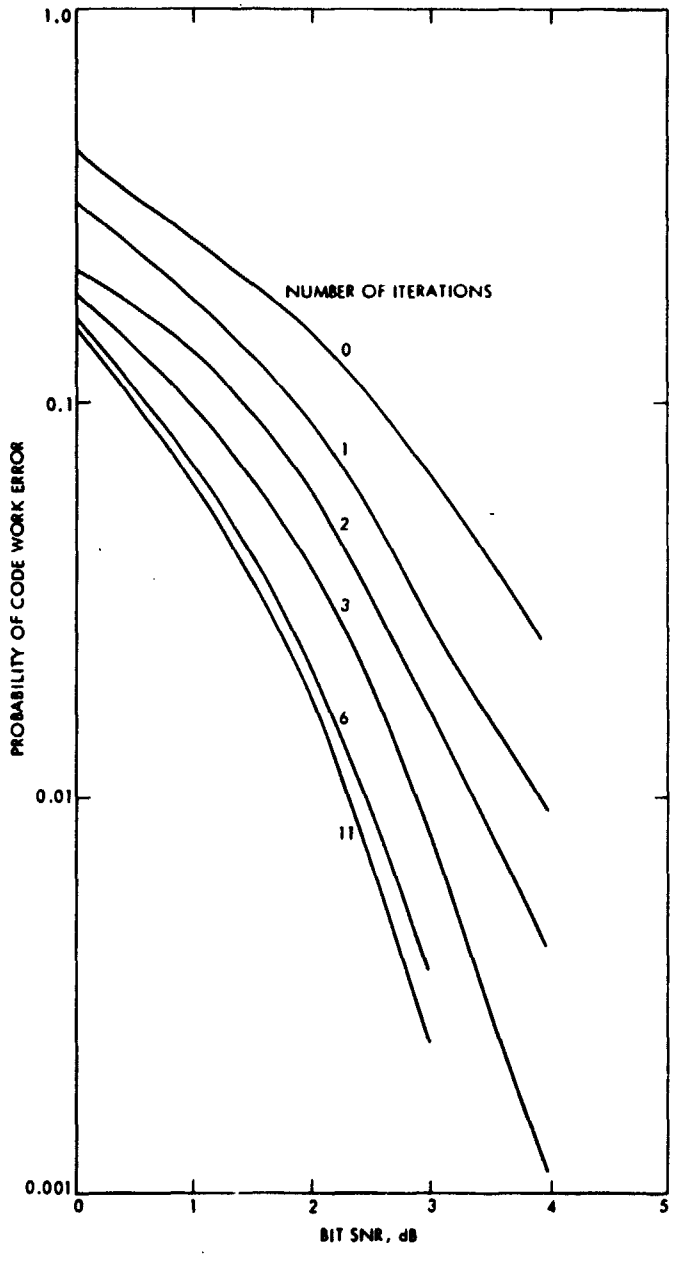

Figure 2-2. Performance of the Algorithm for the (23,12) Golay Code

SECTION 3

ERROR PATTERNS AND ALGEBRAIC DECODING

Algebraic decoding can be used to correct errors within a distance $d_m/2$ (where d_m = the minimum distance of the code) of any error pattern by adding it modulo 2 to the hard limited received vector and then decoding. Proper selection of a set of such patterns, which will be called masks, results in searching a region of space where the errors are most likely for a set of candidate code words. These code words are then correlated with the received vector and the one with the highest correlation is selected as the best estimate of the transmitted code word.

D. Chase (Reference 1-3) suggested using the 2^{12} words of the (23,12) Golay code as masks on the 23 least reliable bits of the (128,64) BCH code. Since any vector of dimension 23 is within a distance 3 of at least one of the Golay code words, any error pattern in the 23 bits will be reduced to 3 or less errors when added to some mask. The larger code can correct up to 10 errors overall. Therefore, depending upon how many errors in the least reliable 23 bits are left uncorrected, patterns of 7 to 10 errors in the most reliable 105 bits can be corrected.

The performance of this decoding algorithm is within 1/2 dB of maximum likelihood (Figure 3-1) and requires 4096 algebraic decodings and correlations. This is considerably more computation than required to obtain comparable performance using the algorithms described in later chapters and is due mainly to three factors. First, this algorithm is based upon detecting errors in contrast to erasing them. For the (128,64) BCH code, the redundancy is 64 bits as compared to an error correcting power of 10 bits and erasing errors is more effective than correcting them. Second, the division into two groups of 23 and 105 was not based upon any property of the large code but only upon the perfection of the Golay code. It may possibly be more efficient to divide the (128,64) BCH code into two more evenly balanced groups. Fewer errors would exist, on the average, in the more reliable group so that, while the less reliable group is larger, one needn't have to correct so many errors in it. Third, the symbols are divided arbitrarily into two groups, one of relatively high and one of relatively low probability of error. Within these groups all bits are considered equal. However, it is more efficient to take into consideration, even approximately, that the bits have a continuous distribution of probability of error.

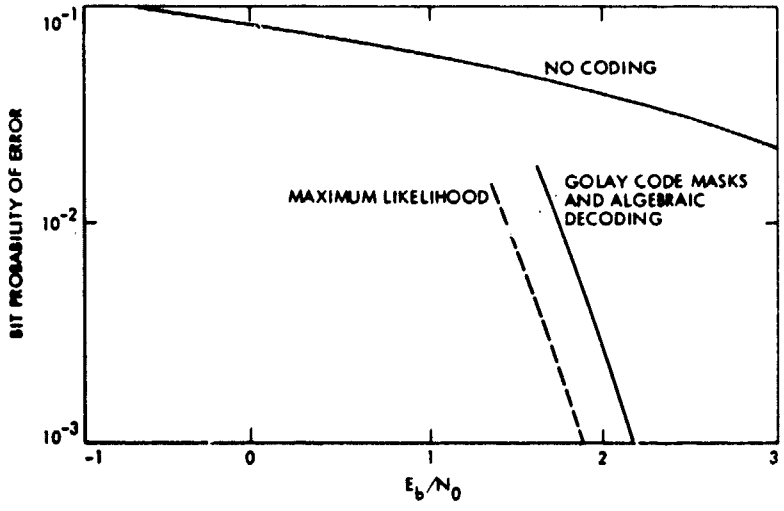

Figure 3-1. Relative Performance of the Chase Decoding Algorithm for the (128,64) BCH Code

SECTION 4

DECODING WITH ERASURES AND ERROR PATTERNS

4.1 INTRODUCTION

For most codes the redundancy is greater than the minimum distance between code words so that it is possible to correct many more erasures than errors. The simplest erasure correcting scheme is one that erases the n-k bits whose estimated probability of error are highest and reconstructs these bits from the k remaining ones by means of the parity check matrix. This scheme will yield the correct estimate only when all the k non-erased bits are estimated correctly by a hard decision decoder. At low signal-to-noise ratios the performance of such a scheme is quite a bit poorer than maximum likelihood but can be improved by expanding the scheme to produce a set of possible code words.

In the k remaining bits which are not erased, there are 2^k possible error patterns. If the decoder correlates with the received vector all of the 2^k possible code words generated by adding these error patterns to the unerased bits, maximum likelihood performance will be achieved. (Note that there is not a one-to-one correspondence between error patterns and code words. If the k-bits are not all independent, there will be no solution when the dependent bits are not consistent with the values of the independent ones and more when they are.) Between the extreme of trying no error patterns and trying them all, one can choose a reasonably sized subset.

To achieve a level of performance of less than 0.5 dB from maximum likelihood approximately 10,000 candidate code words are required. In spite of the large number, this decoding scheme is competitive with the others because of the small number of operations needed to generate each candidate. In this section it will be shown how one should select an efficient set of error patterns and arrange the calculation such that the number of operations will be the minimum possible.

4.2 GENERATING ERASURE PATTERNS ORDERED ACCORDING TO PROBABILITY

Assuming that only a fixed number of error patterns can be tried, it is worthwhile to use the set which contains the ones which are most probable. If the channel is memoryless then the probability of a given error pattern in a block of m bits is

$$\Pr(\underline{e}) = \prod_{i=1}^{m} q_i \prod_{i=1}^{m} \left(\frac{P_i}{q_i}\right)^{e_i}$$

where

e_i = 1 if the ith bit is in the error pattern;
= 0 otherwise
P_i = probability of ith bit being in error
q_i = probability of ith bit being correct.

The first product is independent of the error pattern chosen. Representing it by the constant C_1

$$Pr(\underline{e}) = C_1 \cdot \prod_{i=1}^{m} \left(\frac{P_i}{q_i}\right)^{e_i}$$

taking logarithms

$$\ln Pr(\underline{e}) = \ln C_1 - \sum_{i=1}^{m} e_i \ln \left(\frac{P_i}{q_i}\right).$$

For a Gaussian channel the log likelihood ratio is proportional to the bit amplitude a_i.

$$\ln Pr(\underline{e}) = \ln C_1 - C_2 \sum_{i=1}^{m} e_i a_i.$$

The logarithm is a monotonic function of its argument so that ordering the error patterns by decreasing probability of occurrence is equivalent to ordering them by increasing d_1 where

$$d_1 = \sum_{i=1}^{m} e_i a_i = \underline{e}^t \underline{a}.$$

It is not necessary to calculate the most probable error patterns for each received vector; a set of fixed-error patterns, independent of the received signal-to-noise ratio, can be used with negligible loss of performance. This simplification has been justified by simulation, but it can be demonstrated simply by observing the mean amplitude of the received bits after sorting. An example for a block length of 128 is shown in Figure 4-1. The curves are linear over most of their length, except for a few of the worst and best bits, and change slowly with signal-to-noise ratio. As the signal-to-noise ratio increases, the variation in bit amplitude decreases and approaches one for all bit positions. Thus, the algorithms which depend upon sorting the received bits according to their amplitudes become less efficient.

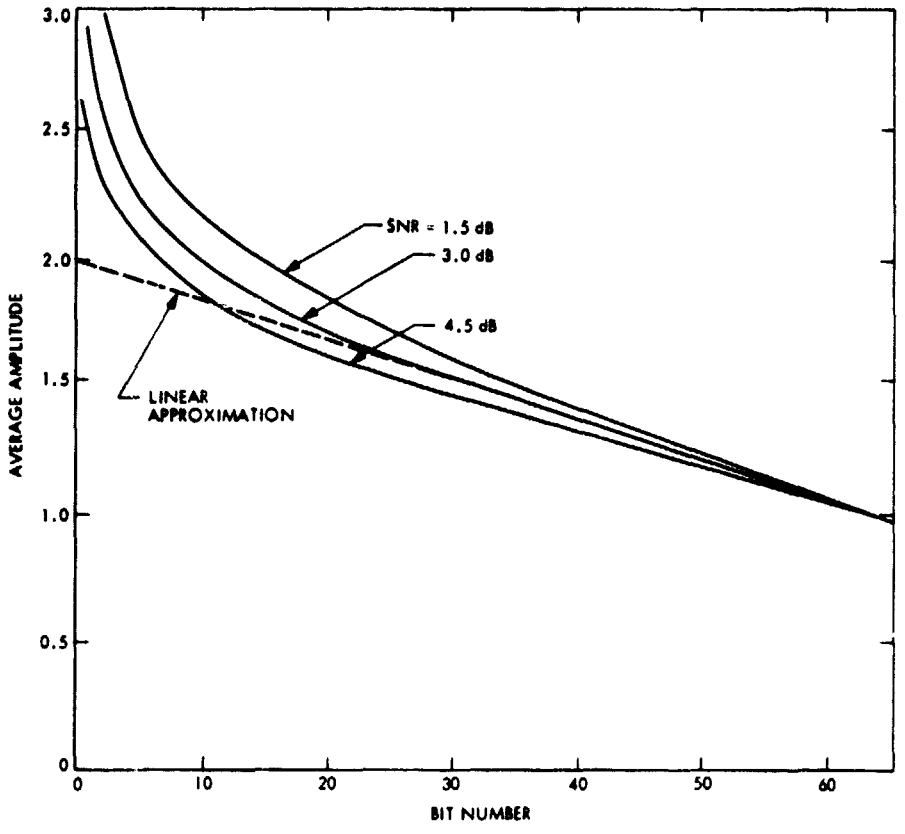

Figure 4-1. Average Bit Amplitude of Received Bits After Ordering Block Length = 128

However, at these signal-to-noise ratios, practically all received vectors are within half the minimum distance between code words so that maximum radius decoding algorithms can be used. For signal-to-noise ratios of 1.0 - 5.0 dB the algorithms discussed here are superior. In this region a good approximation to the mean bit amplitude is

$$a_i = \frac{128 - b_i}{64}$$

where b_i = sorted bit position. The denominator is just a scale factor and may be dropped without affecting the ordering.

Defining an error pattern index f as

$$f = 128n - \sum_{i=1}^{n} b_i$$

where n = number of errors in error pattern. It is easy to generate error patterns according to ascending index. Figure 4-2 illustrates

the number of error patterns as a function of the error pattern index. All single errors have a lower index than any double error, but there are triple errors with lower index than some double errors. There are 2080 single and double error patterns. Checking them all, in order of ascending index, reduces the probability of error, as a function of number of patterns, more slowly than using the set of patterns that allow triple errors. The overall performance for this error and erasure decoding algorithm, as a function of the number of error patterns, is shown in Figure 4-3. To achieve a given level of performance, e.g. within 1/2 dB of a maximum likelihood decoder, the number of error patterns (or candidate code words) required decreases slowly as the signal-to-noise ratio increases.

4.3 GENERATING CANDIDATE CODE WORDS FROM THE ERROR PATTERNS

The generation of candidate code words from a particular error pattern is done in two stages. First the error pattern is checked for consistency with the parity check equations. Then, if it is consistent, a set of code words is generated. Consistency requires that $[P_{1a}]\underline{e} = \underline{c}_1$ In addition to a compare and branch, the number of vector additions needed to check this is equal to the weight of $\underline{e}$, which is one, two or three in the practical cases discussed here. If the set of error patterns is determined beforehand, then they can be arranged in such a way that only one vector addition is needed. In achieving this savings, however, the error patterns are no longer checked in order of ascending error pattern index.

When the error pattern is consistent with the parity check equations, then candidate code words exist. The bits of the candidate code words can be divided into 3 sets:

(1) $\underline{a}_1$, containing the bits which were checked for consistency

$$\underline{a}_1 = \underline{b} + \underline{e}$$

(2) $\underline{a}_{2a}$, a number of arbitrary bits

(3) $\underline{a}_{2b}$, containing the bits calculated from (1) and (2)

$$\underline{a}_{2b} = \underline{c}_2 + [P_{1b}]\underline{e} + [P_3]\underline{a}_{2a}.$$

The first two sets are calculated once, requiring a maximum of three additions (when the weight of $\underline{e} = 3$). This generates the first candidate code word corresponding to $\underline{a}_{2a} = \underline{0}$. The remaining bit patterns of $\underline{a}_{2a}$ can then be generated in Grey code sequence so that the succeeding code words can be calculated by using only a single vector addition per code word.

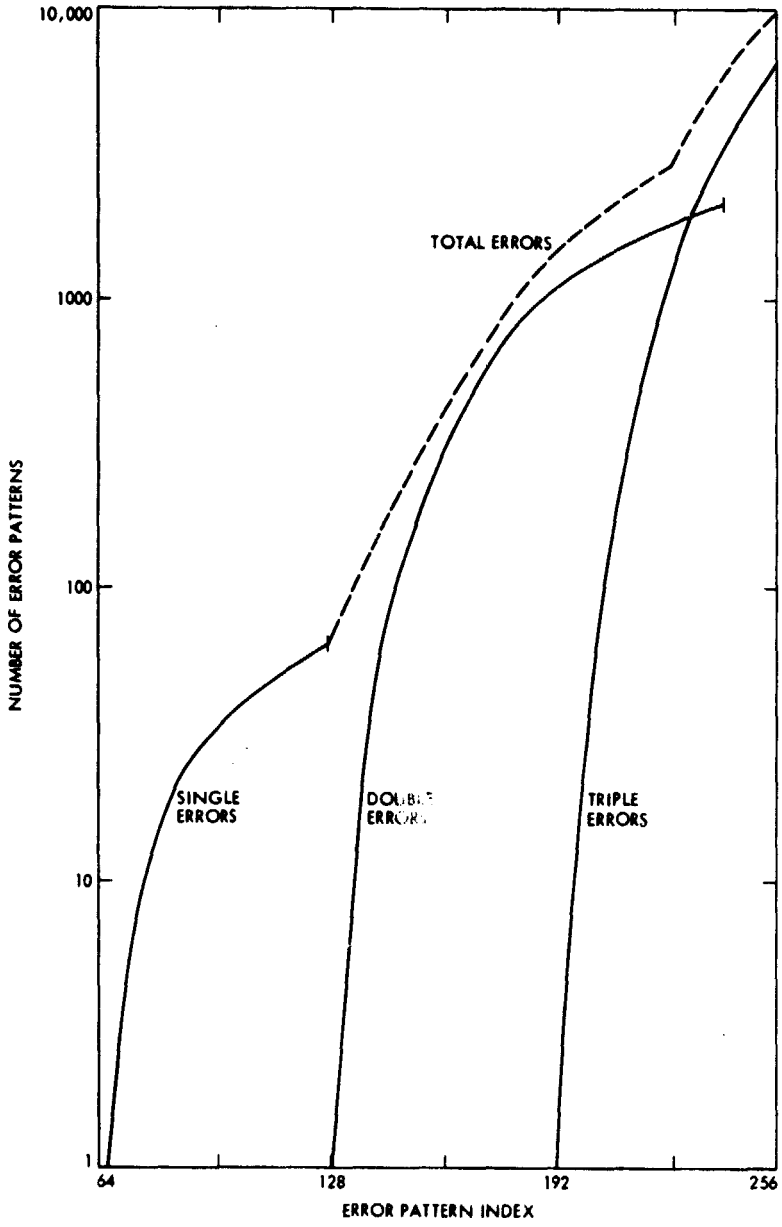

Figure 4-2. Cumulative Number of Error Patterns for a Given Index

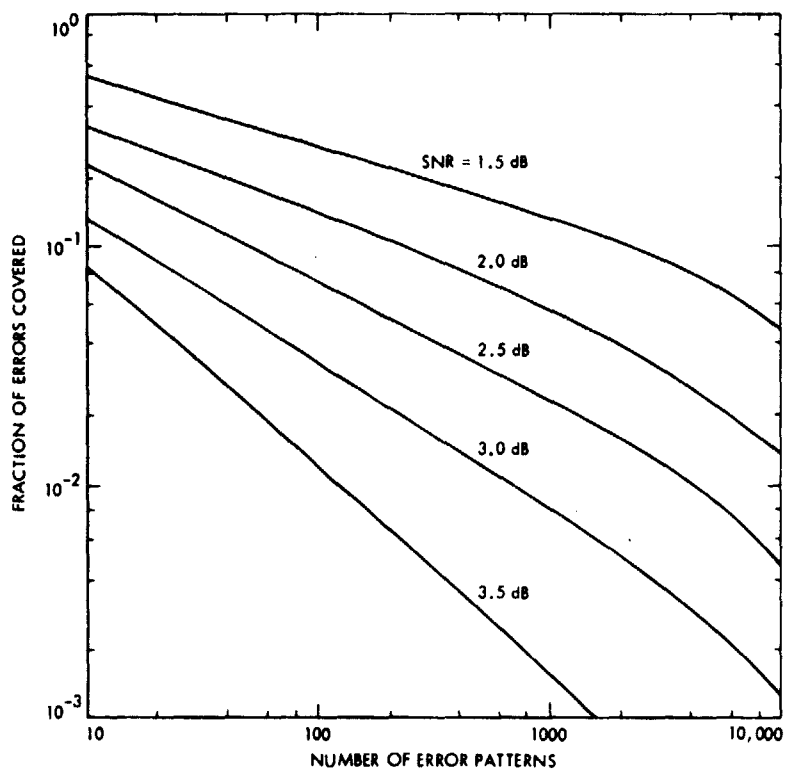

Figure 4-3. Performance of the Error and Erasure Decoding Algorithm

SECTION 5

DECODING USING SETS OF ERASURE PATTERNS

5.1 INTRODUCTION

A more efficient method of generating candidate code words, in the sense of requiring fewer of them to obtain a given level of performance, is to use sets of erasure patterns. For each erasure pattern in the set, the corresponding bits in the received word are erased and then reconstructed by means of the parity check equations. A correct decoding requires that for at least one of the patterns, all of the unerased bits be error free.

One approach to constructing sets of erasure patterns is to try to cover as many error patterns as possible with as few masks as possible. A combinatoric solution to this problem is by means of t-designs (Reference 5-1). Given a set of v elements, a t-design is a collection of subsets of k elements with the property that any subset of t elements is contained in exactly λ blocks. The design is represented as $t - (v, k, \lambda)$ and is sometimes called a tactical configuration. When used to construct erasure masks, v corresponds to the block size and k to the number of erased bits. All errors of weight t or less are covered λ times. The difficulty of this approach is that designs are known only for small t and are therefore useful only for codes of short block length. By relaxing the requirement that all error patterns be covered exactly t times, L. Baumert, R. McEliece and G. Solomon have devised a method of generating masks from sets of code words (Reference 5-2). However, both these methods have the disadvantage that all patterns are treated equally. A more efficient set of masks would consider the probability of a given error pattern occurring rather than just its weight. This criterion leads to a statistical approach for the design of sets of masks.

The first set of such masks was generated by L. Baumert and R. McEliece using a weighing based upon the entropy of the error probability of each bit (Reference 1-4). (The entropy could not be used directly because it violated the constraints on allowable weighing functions, which will be seen in the next section.) This approach will be used here to generate a good set of erasure masks. Starting with a linear weighing, sets of masks with slightly perturbed weighings are generated. The best ones are selected and the process is repeated until it converges to a good weighing function. This approach is possible since the performance varies only slowly with changes in weighing. The set of masks arrived at in this manner was almost identical to that of Baumert and McEliece, which confirmed the accuracy of their intuition.

With 1000 masks, maximum likelihood performance is approached to within a few tenths of a dB. However, the generation of candidate code words from these masks and the received code word requires a relatively long calculation. In the latter part of this section it is shown that the masks can be ordered within a given set so that the number of calculations per candidate code word is considerably reduced.

5.2 GENERATING AN EFFICIENT SET OF ERASURE MASKS

The statistical method generates a set of masks which covers each bit, on the average, a given percent of the time but does not attempt to cover a specific set of errors. The advantage of this method is that it is relatively easy to generate a set with a given number of masks and a wide range of distributions and which, almost always, cover the errors in an efficient manner. The exact distribution is not critical; however, the bits with a probability of error close to zero should hardly ever be covered and those with a probability of error close to one-half should almost always be covered. Also, since the fraction of bits left uncovered by each mask must equal the rate of the code, the entire distribution must also satisfy that constraint. These constraints may be written as:

$$(1) \quad \ell_1 \simeq m \; ; \qquad \ell_m \simeq 0$$

$$(2) \quad \ell_i \geq \ell_j \qquad \text{if } i > j$$

$$(3) \quad \sum_{i=1}^{n} \ell_i = m \cdot n \cdot r$$

where

ℓ_i = the number of times the ith bit is covered

m = number of masks

n = number of bits in code word

r = rate of code

There are an extremely large number of functions that satisfy these constraints. However, it is not difficult to find good ones by trial and error since their performance is relatively insensitive to the exact shape of the function. For example, consider a code of rate 1/2. From the third constraint the area under the function must equal $(m \cdot n)/2$, one half the area enclosed by the graph. One function that satisfies these constraints is linear in percent of bits covered versus bit number, Figure 5.1, curve a. Others, such as b and c, which emphasize the covering of bits with a high or low probability of error, and d, which has a large discontinuity, are also possible. By simulation it has been found that functions with the general shape of b generate masks with best performance. This will be discussed in greater detail in the following section.

If the fractional area under the desired function equals the rate of the code, then one need only find the appropriate scale factor and compensate for the small errors introduced by rounding off the function to integral values. For example, the linear function

$$\ell_i = m\left(1 - \frac{i}{n}\right)$$

satisfies

$$\sum_{i=1}^{m} \ell_i = \frac{m \cdot n}{2} .$$

so that it is suitable for a rate 1/2 code.

Most of the values of ℓ_i are non-integral and truncating or rounding of these numbers may produce a sum not equal to $(m \cdot n)/2$. The slight adjustment required may be made by letting ℓ_i equal the integral part of

$$\ell_i = \left\lfloor m\left(1 - \frac{i}{n+\epsilon}\right) \right\rfloor$$

and varying ϵ until

$$\sum \ell_i = \frac{m \cdot n}{2} .$$

For a general function, however, the fractional area under it does not equal the rate and some distortion must be introduced in order to meet this constraint. One possibility, used by L. Baumert for functions which enclose a fractional area less than the rate, is to multiply the function by a scale factor which will generate values of ℓ_i (for some i) greater than m and then limiting these values to m. The scale factor (and area) is increased until the third constraint is met. This procedure usually results in functions similar to curve b of Figure 5-1 which have been found to yield efficient masks.

Given the normalized distribution which satisfies this constraint, the cover is generated by randomly selecting a mask and placing a one in the ith bit position until ℓ_i ones have been placed. As i approaches n, certain moves will be forced in order to place the proper number of ones and zeros in each mask. These are done first, before the remaining ones are placed randomly. The algorithm will fail if the forced moves require more than ℓ_i ones in the ith position. However, this occurs very rarely in practice and can be remedied by some slight adjustment of the bits in previous positions. In the generation of many sets of masks, these slight deviations from a purely random placement of ones and zeros has not been found to produce any adverse effects.

The linear distribution function is a good starting point for investigating masks suitable for codes of rate 1/2. Using the (128,64) BCH code as an example, a set of 1000 masks were generated and tested against 10,000 received vectors at various signal-to-noise ratios. The received vectors were sorted, hard limited and the bit positions containing errors were noted. The masks were then successively placed over the error pattern and the number of masks tested before the error pattern was completely covered and recorded. No attempt was made to order the masks in any particular way.

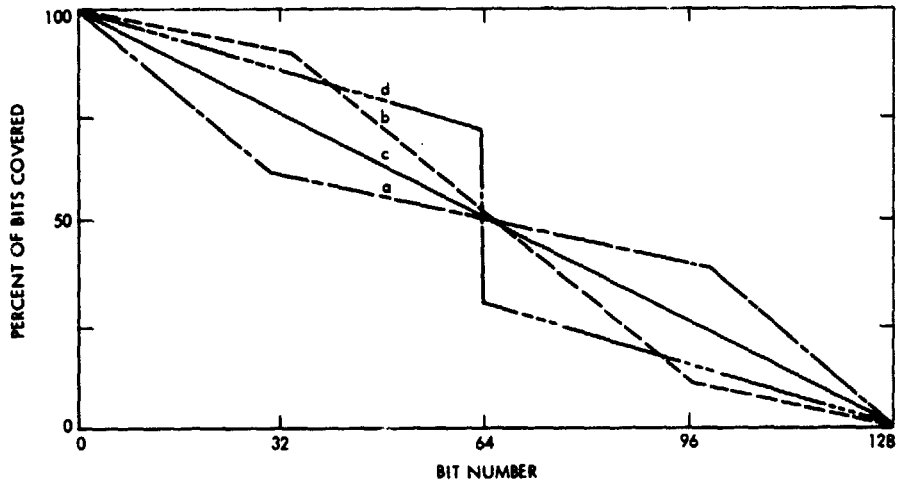

Figure 5-1. Weighing Functions for the (128,64) BCH Code

The resulting curves, seen in Figure 5-2, are an approximation to a lower bound on the probability of error. If the error vector is not covered then the decoder, using these masks, will surely be in error. If the error is covered, the transmitted code word will be among the candidate code words, but there still may exist another code word that has a larger correlation with the received vector. This bound is generally a good approximation except when the decoding algorithm is operating very close to maximum likelihood performance. Using 1000 masks, the performance of the decoding algorithm is approximately 0.15 dB worse then the maximum likelihood.

The linear distribution was selected as one satisfying a number of simple heuristic considerations. That these considerations are valid is demonstrated by the fact that performance equal to that obtained with erasure and error patterns requires only one-tenth the number of candidate code words. It may be, however, that other distributions can generate a set of masks that perform even better. To check this, two distributions symmetric about the linear one, b and c of Figure 5-1, were tried in order to get an idea in which direction to proceed. Their performance, shown in Figure 5-3, and that of others not shown, indicated that better performance can be obtained with weighings which cover the bits with high probability of error more often than the linear weighing.

A family of distributions with this property was then investigated. Rather than using a set of symmetric functions, a set which always covered a given number of bits with the highest probability of error was selected (see Figure 5-4). The performance is only slightly different from that of a similar symmetric distribution and includes, as distribution g, the one originally used by L. Baumert. Using 1000

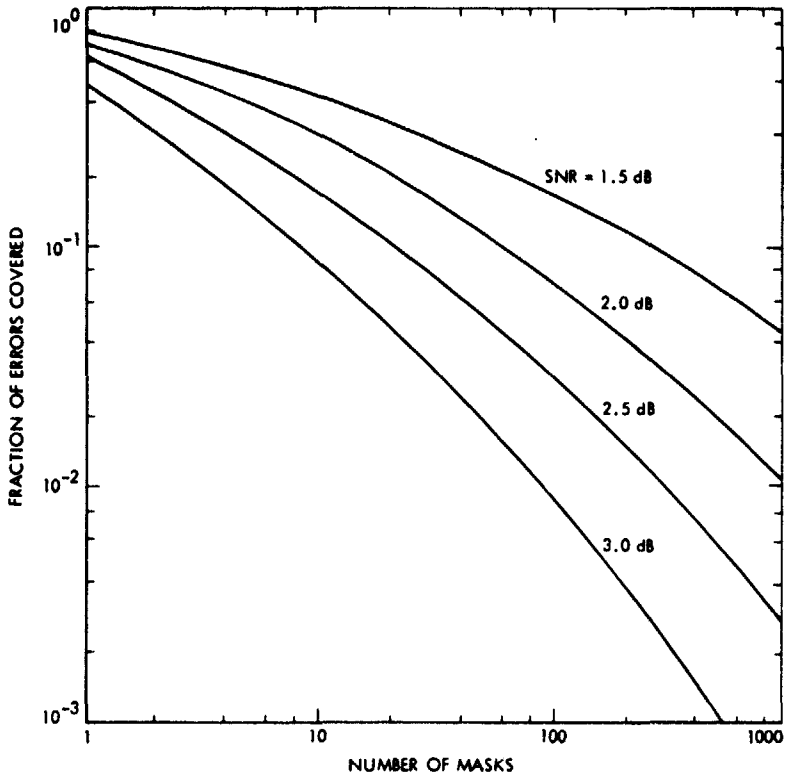

Figure 5-2. Performance of a Set of Masks with Linear Weighing

masks and 10,000 received vectors at an SNR = 2.0, the probability of an error pattern not being covered is

Distribution	a	e	f	g	h	i
Probability error pattern not covered	.020	.015	.011	.011	.021	.030

Distribution g is among the best and was investigated further. Its performance at various signal-to-noise ratios is given in Figure 5-5. Note that the performance curves are very similar to that of the linear distribution with a small displacement. This is another indication that the performance changes slowly with changes in distribution. The performance is only slightly changed when different sets of masks, generated from the same distribution, are used. Figure 5-6 illustrates this point using two sets generated from distribution g.

5.3 ORDERING THE MASKS FOR EFFICIENT DECODING

When a code word is received, the bits are sorted in order of their absolute magnitude and the columns of the parity check matrix

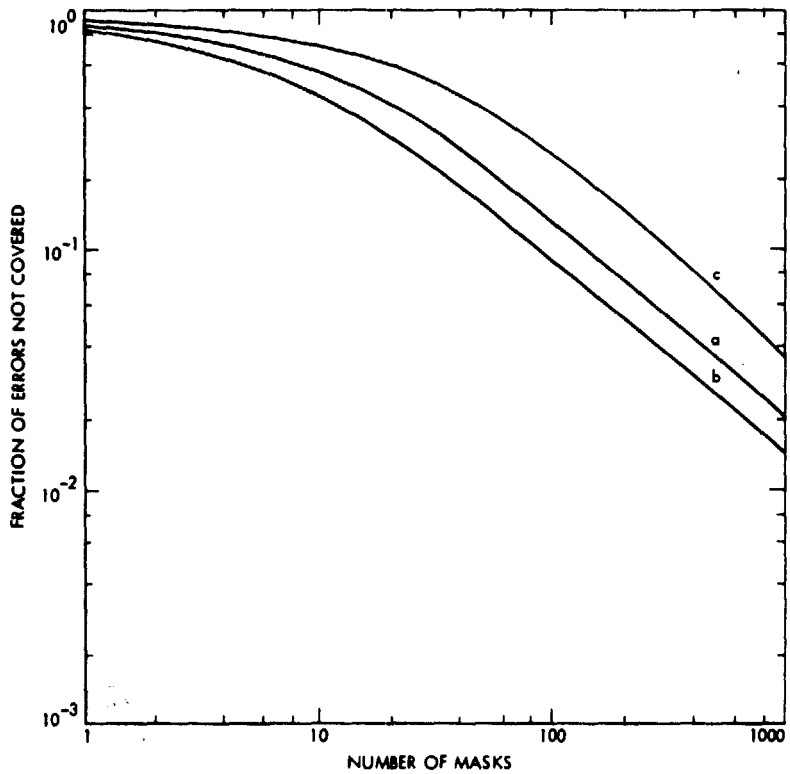

Figure 5-3. Relative Performance of Sets of Masks Using the Weighing Function of Figure 5-1. (SNR = 2.0 dB)

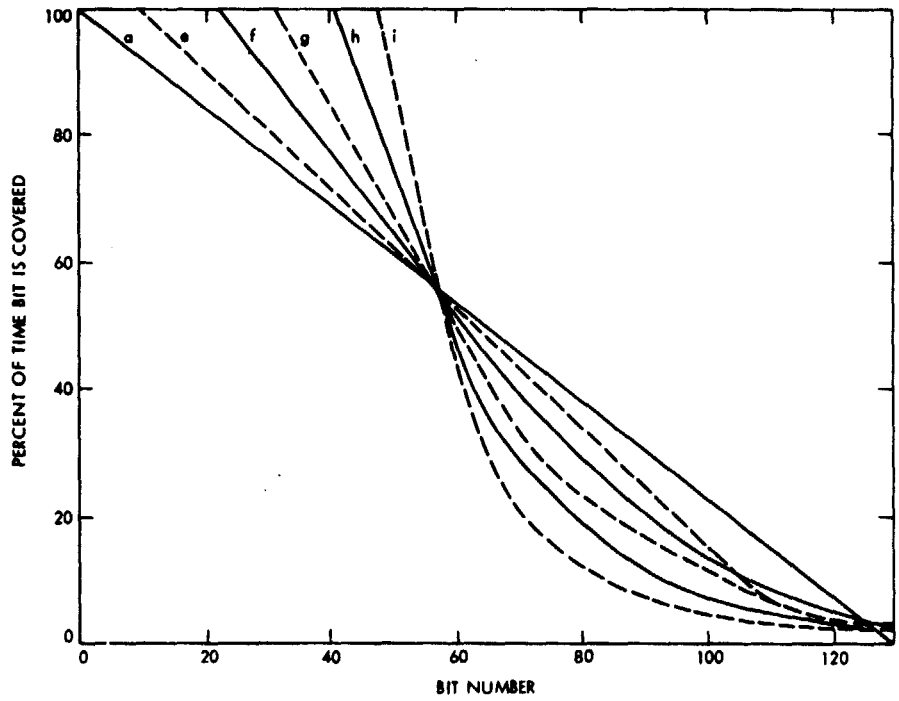

Figure 5-4. Weighing Functions for Rate 1/2 Codes

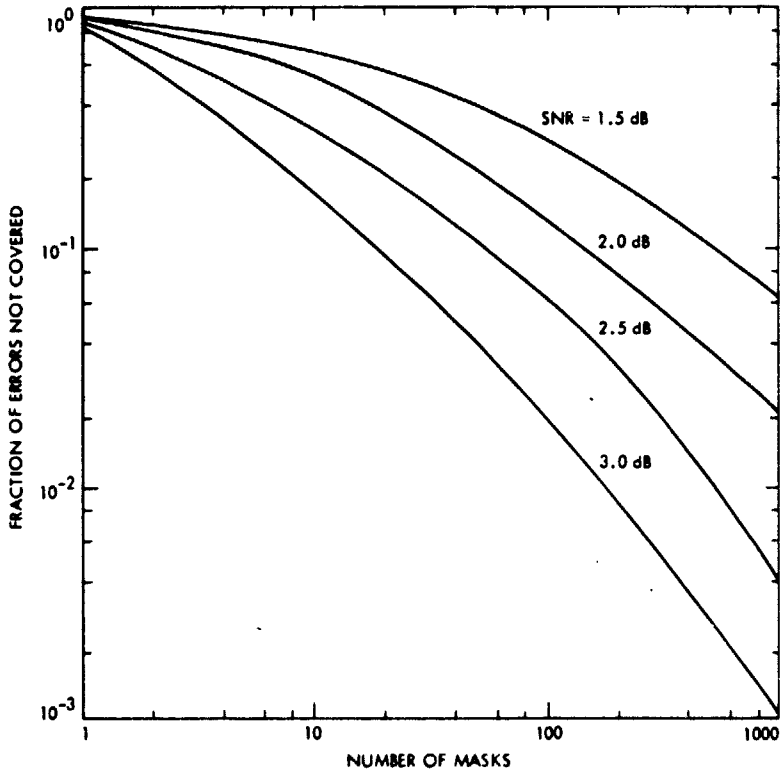

Figure 5-5. Performance Using the Weighing Function of L. Baumert (g)

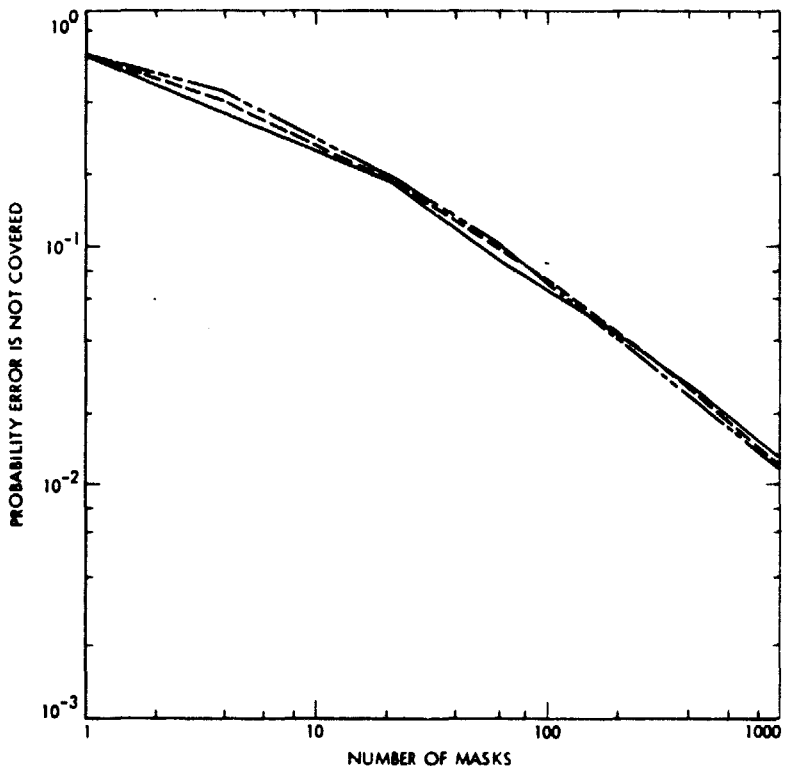

Figure 5-6. Comparison of Performance Using Different Sets of Masks Generated from the Same Weighing Function

are permuted accordingly. Each mask erases n - k bits and these must be determined from the unerased ones in order to generate a candidate code word. This operation requires solving a set of simultaneous equations, or in terms of matrix operations, the reduction of the permuted parity check matrix to standard form. Since each mask erases a different set of bits, the reduction must be done anew for each candidate code word. This calculation is by far the most time consuming and its efficiency determines the overall efficiency of the entire algorithm. It is therefore worthwhile to reduce the computation of the reduction of the parity check matrix to a minimum.

As before, the rate 1/2, (128,64) BCH code will be used as an example. Given the parity check matrix H, for any code word $\underline{c}$, $[H]\underline{c} = \underline{0}$. If H is in reduced echelon form then it can be partitioned into two parts, one of them an identity matrix. Partitioning $\underline{c}$ corresponding to the partition in H

$$[\ P|I\ ] \begin{bmatrix} \underline{c}_1 \\ \hline \underline{c}_2 \end{bmatrix} = \underline{0}$$

or

$$\underline{c}_2 = [P]\underline{c}_1 .$$

In this form the parity check bits $\underline{c}_2$ can be determined directly from the information bits $\underline{c}_1$.

Each mask erases 64 of the 128 bits which must be reconstructed from the remaining 64. This can be done by permuting the erased bits to one side of the matrix, the unerased to the other, and reducing the resulting matrix. (The original matrix P is non-singular. However, it may be that for a particular permutation it is not. This case can be handled in a manner similar to the one described in the Appendix.)

For a general matrix H this would require on the order of $(64)^2/2$ vector operations. However, in this particular case, one can get by with much less.

How much less depends upon the Hamming distance between two successive masks. Since the weight of each mask is identical, transforming one mask into another can be thought of as interchanging pairs of bits from the set of erased bits to the set of unerased bits. Two masks with a Hamming distance d_H between them require $d_H/2$ interchanges. Transforming the parity check matrix corresponding to one mask to a matrix corresponding to the next requires the interchange of $d_H/2$ pairs of columns and the reduction to standard form. The $64 - (d_H)/2$ columns in the erased set which have not been interchanged contain only a single one and are already in reduced form. The number of row operations required to reduce the remaining columns are on the order of $1/2\ (d_H/2)^2$ rather than $1/2(64)^2$.

The average distance between masks in a set depends only upon the distribution that was used to generate the masks and not upon

the particular set used. Given a set of n masks with a distribution (i = number of times the ith bit is covered), the sum of the distances between the ith bit of a particular mask ($m = m_j$) and the ith bit of all other masks is

$$\sum_{\substack{k \\ j \neq k}} d_i(m_j, m_k) = n - \ell_i \qquad \text{if the bit is a 1}$$

$$= \ell_i \qquad \text{if the bit is a 0.}$$

Since there are ℓ_i ones and $n - \ell_i$ zeros, the sum over all masks is

$$\sum_{j} \sum_{\substack{k \\ m_j \neq m_k}} d_i(m_j, m_k) = \ell_i(n-\ell_i) + (n-\ell_i)\ell_i = 2\ell_i(n-\ell_i)$$

Dividing by the number of terms in the summation, the average distance between the ith bits is

$$\bar{d}_i = \frac{2\ell_i(n-\ell_i)}{n(n-1)}.$$

Summing over i gives the average distance between masks

$$\bar{d} = \sum_i \bar{d}_i = 2\sum_i \frac{\ell_i(n-\ell_i)}{n(n-1)}.$$

For the ℓ_i's of distribution D, $\bar{d} = 28$. If all the words were equidistant from each other (desirable from the point of view of covering as many error patterns as possible) then the number of vector additions required to reduce the parity check matrix for each mask would be on the order of

$$\frac{1}{2}\left(\frac{\bar{d}}{2}\right)^2 = 98.$$

The masks are probabilistically generated and the actual distances are distributed about the mean, as shown in Figure 5-7. Given an arbitrary ordering of masks, the number of vector additions required to reduce the parity check matrix m times would be even greater than

$$\frac{m}{2}\left(\frac{\bar{d}}{2}\right)^2.$$

However, if the masks are sorted to an order in which the distances between successive masks are as small as possible, the number of computa-

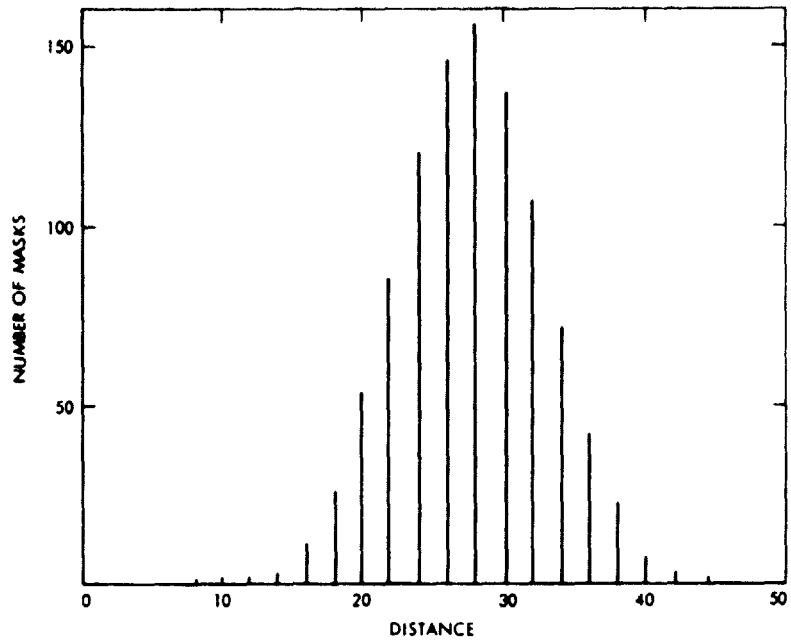

Figure 5-7. Hamming Distance Between 1000 Arbitrary Ordered Masks with Weighing 'g'

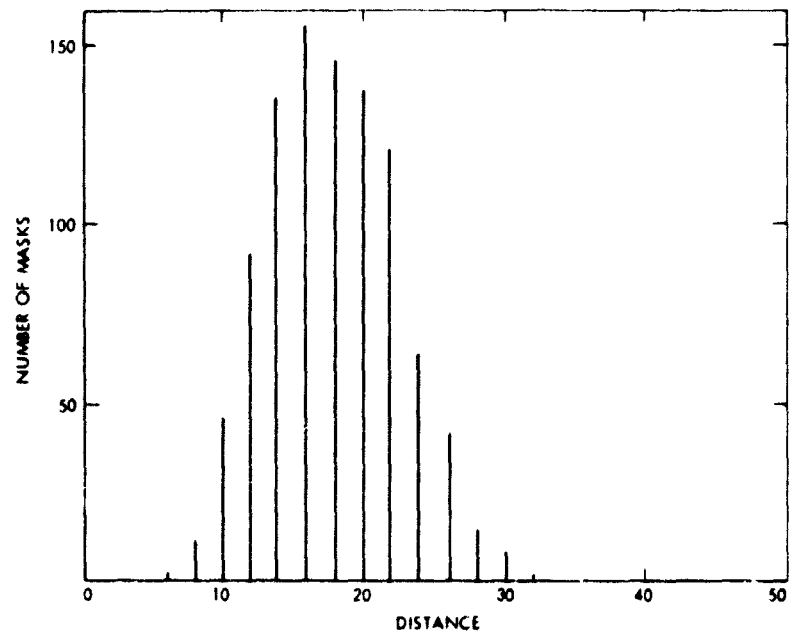

Figure 5-8. Hamming Distance Between Masks After Sorting to Reduce Distance Between Successive Masks

tions will be reduced significantly. A simple sorting algorithm is as follows:

$$\sum_{j} \sum_{\substack{k \\ m_j \neq m_k}} d_i(m_j, m_k) = \ell_i(n-\ell_i) + (n-\ell_i)\ell_i = 2\ell_i(n-\ell_i)$$

Starting with an arbitrary mask, the mask closest to it is placed second. The sort is continued by searching the remaining masks and again placing at the end of the chain the one closest to the mask currently last. The algorithm is continued until all the masks have been ordered.

The resulting distribution of distances between successive masks is shown in Figure 5-8. With a little bit of work, a chain could undoubtedly be constructed which would have successive distances between masks of 20 or less and require only

$$\frac{1}{2}\left(\frac{d}{2}\right)^2 = 50$$

vector additions to convert one reduced matrix to another. With this simplification this decoding method becomes competitive with other schemes discussed in this report.

SECTION 6

DECODING USING BOTH SETS OF ERASURE PATTERNS AND ERROR PATTERNS OF LOW WEIGHT

The most time consuming operation in the decoding scheme using sets of erasure masks is the reduction of the parity check matrix each time a new mask is used. The number of masks required is determined by the desired probability that an error is completely covered by at least one mask. If the requirements are reduced to allow a maximum of one or two errors to remain exposed, the number of masks required is reduced by a large factor. Locating these errors requires a certain amount of computation, but in most cases the total computation required to achieve a given level of performance will be less than in the original algorithm.

In order to estimate the advantages of such a scheme, consider the set of 1000 masks with a performance represented by curve g in Figure 5-4. The curve shows the number of masks required to cover, at least once, all the errors in a received code word a given fraction of the time. This curve is repeated in Figure 6-1 and is accompanied by curves showing the fraction of time a maximum of one, two or three errors are exposed.

To achieve a level of performance equal to 1000 masks and no errors exposed requires 50 masks if one, 6 masks if two, and 2 masks if three errors remain exposed. This scheme will therefore be more efficient if locating single errors requires less computation than reducing the parity check matrix 20 times, double errors 166 times, and triple errors 500 times.

Single errors may be located by assuming one of the exposed bits to be in error and calculating the erased bits, given this assumption, for each of the exposed bits. Assuming the reduced parity check matrix is in the form [H] = [P|I], the calculation of the candidate code words is straightforward. (This implies that the erased bits are all independent. The case for which this is not true is discussed in the Appendix.) The parity check equations can be written as

$$[H]\,\underline{a} = \underline{0}$$

where

$\underline{a}_1$ = exposed bits,

or

$$[P \mid I] \begin{bmatrix} \underline{a}_1 \\ \hline \underline{a}_2 \end{bmatrix} = \underline{0}$$

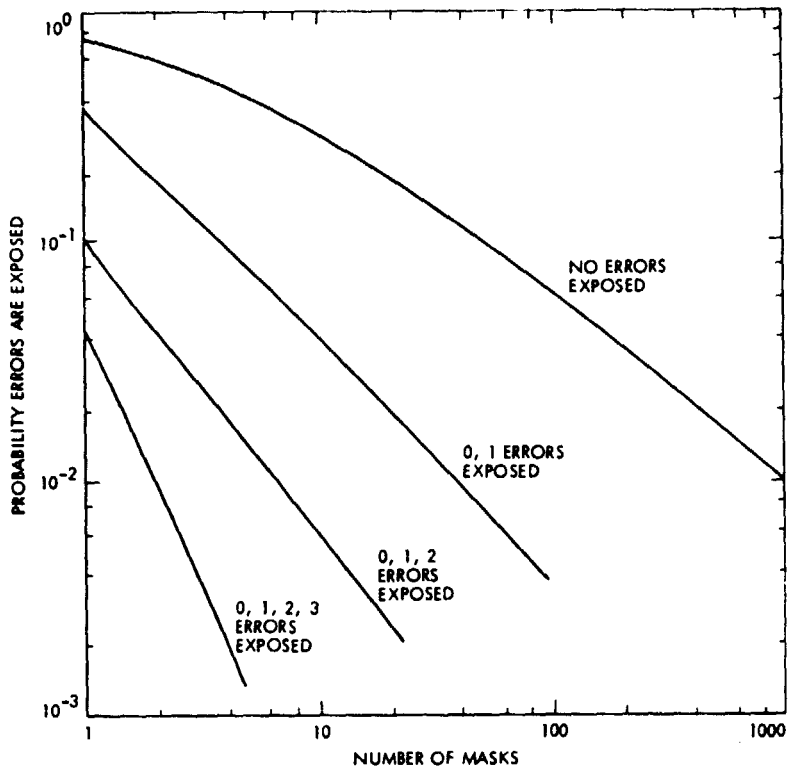

Figure 6-1. Probability of errors not being covered when using masks of 64 bits out of 128

where

$$\underline{a}_2 = \text{erased bits assuming no errors in } \underline{a}_1$$

which is equivalent to

$$\underline{a}_2 = [P]\underline{a}_1.$$

Assuming an error pattern represented by ones in the vector $\underline{e}$, the erased bits are equal to

$$\underline{a}_3 = [P]\underline{a}_1 + [P]\underline{e} = \underline{a}_2 + [P]\underline{e}.$$

The calculation of $\underline{a}_2$ requires a number of vector additions equal to the number of ones in $\underline{a}_1$. (This calculation is also required when decoding using erasure masks only.) Assuming an error pattern $\underline{e}$, $[P]\underline{e}$ equals the sum of the corresponding columns of the matrix [P]. Thus, if there are k independent bits in the code word, to calculate the candidate code words corresponding to no and single errors requires, at most, 2k vector additions.

Considering the previous example of a (128,64) BCH code with 1000 masks and an average distance of 16 between masks, calculating the code word corresponding to each mask requires approximately $1/2(2^8)$ = 128 vector additions. Calculating the candidate code words corresponding to single errors in the exposed bits also requires 2k = 2(64) = 128 vector additions, at most. Since using 1000 erasure masks yields the same performance as using 50 erasure masks and allowing single errors, the second method requires one-tenth the amount of computation as the first.

Extending this comparison to error patterns of greater than one error, there are

$$64 + \binom{64}{2} = 2080$$

single and double error patterns. These can be checked in the time required to reduce the parity check matrix 16 times again giving a savings of about 10 times. The savings can be even greater if error patterns are ordered so that those of higher probability are used first. In this case some triple error patterns will precede some double error patterns. In this case, to get the performance equivalent to all single and double error patterns, only about one-half the number of patterns need to be used, giving a savings of 20 times over using erasure masks only.

SECTION 7

AN EFFICIENT HYBRID ALGORITHM

7.1 INTRODUCTION

The previous chapters have discussed ways to improve the efficiency of previously known decoding algorithms which are based upon selecting a small set of candidate code words. By suitably combining the best features of these algorithms, an algorithm has been developed which is more efficient than any of those previously known. This algorithm uses a small number of erasure masks, assumes errors of low weight in the unerased bits and uses redundancy to reduce the number of error patterns that need to be checked. The input to the algorithm is a vector whose elements are quantized amplitudes of the symbols of the received code word and the output is an estimate of the transmitted code word. A flow chart of the major sections of the algorithm is shown in Figure 7-1.

The first step of the algorithm, as in all the ones previously discussed, is to sort the symbols of the received code word according to their absolute magnitude, permute the columns of the parity check matrix, and reduce it to standard form. (Efficient sorting and matrix reduction algorithms are described in the Appendix.) At this point the algorithm diverges from those previously considered.

7.2 PARTIAL SYNDROME DECODING

When the unerased bits are not all independent, there arises the possibility that certain values of the bits do not satisfy the parity check equations. In previous sections these cases were considered to be a decoding failure. However, it is possible to take advantage of the dependency in order to determine which error patterns in these bits satisfy the parity check equations. Only these patterns need to be used to generate candidate code words. The number of such error patterns of a given weight can be a small fraction of the total number of error patterns of that weight, greatly reducing the number of candidates required for a given level of performance.

Calculating the error patterns which will be consistent with the parity check equations can best be done by considering a portion of the syndrome and determining the error patterns, which, when added to the initial estimate of the received vector, will make that portion equal to zero. Since a code word must satisfy $[H]\underline{c} = \underline{0}$, it will also satisfy this equation for any subset of rows of H. Reducing the parity check matrix and partitioning H as:

$$\left[\begin{array}{c|c} P_1 & 0 \\ \hline P_2 & I \end{array}\right] \left[\begin{array}{c} \underline{c}_1 \\ \hline \underline{c}_2 \end{array}\right] = \underline{0}$$

r bits ——┘ └—— (n-r) bits

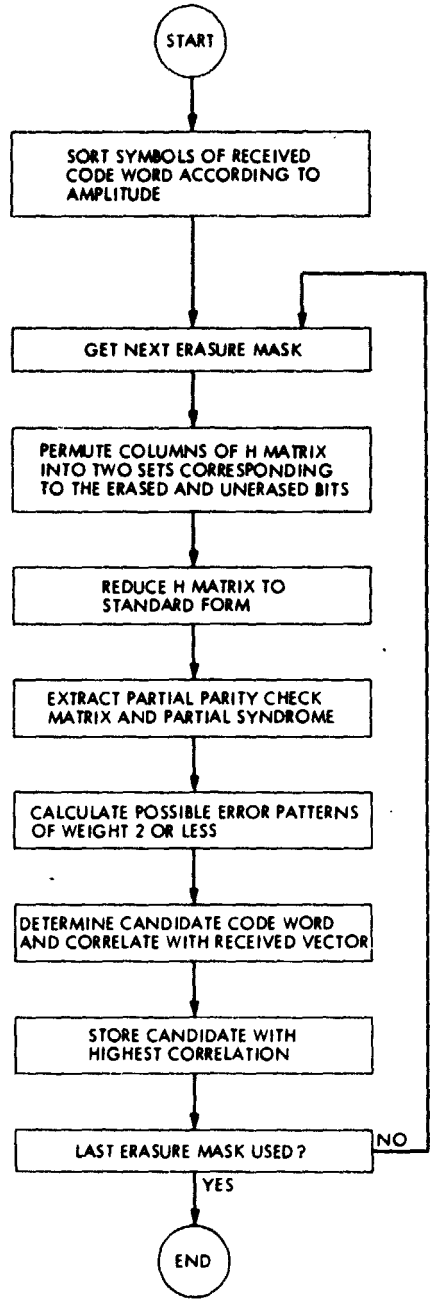

Figure 7-1. Flow chart of Soft Decision Decoding Algorithm

it is seen that $[P_1]\underline{c}_1 = \underline{0}$. Therefore, the first r bits of the code word must also be the solution of a set of homogeneous equations.

For an arbitrary received vector $\underline{a}$, the project $[H]\underline{a} = \underline{s}$ is called the syndrome and specifies the coset containing the possible error patterns in $\underline{a}$. The same notion can be used when considering only the first r bits of the received word. Then $[P_1]\underline{a}_1 = \underline{s}_1$ is the partial syndrome which specifies the possible error patterns in $\underline{a}_1$. Representing $\underline{a}_1$ by $\underline{c}_1 + \underline{e}_1$, where $\underline{e}_1$ is the partial error pattern corresponding to the partial code word $\underline{c}_1$,

$$[P_1]\ [\underline{c}_1 + \underline{e}_1] = \underline{s}_1$$

or

$$[P_1]\underline{e}_1 = \underline{s}_1.$$

Given $[P_1]$ and $\underline{s}_1$, there are a large number of partial error patterns $\underline{e}_1$ that will satisfy this equation. However, for the decoding algorithm to be considered here, it is sufficient to consider patterns of 0, 1 or 2 errors. Note that even though the adjective "partial" is applied to $\underline{a}_1$, $\underline{c}_1$, and $\underline{s}_1$, the remainder of the code word is completely determined from $\underline{c}_1$ by $\underline{c}_2 = [P_2]\underline{c}_1$. The advantage of this approach is that candidate code words are determined only by possible error patterns in the most reliable received bits $\underline{a}_1$. The remaining received bits $\underline{a}_2$ do not enter at all into the calculation and can be considered erasures.

No errors as a possible error pattern can only occur if the partial syndrome equals zero; single errors can occur in those bits whose corresponding columns of $[P_1]$ are identical to $\underline{s}_1$, and double error patterns in those bits whose corresponding columns of $[P_1]$ sum to $\underline{s}_1$. In general, for an error pattern of weight w to be a possibility, the sum of the w corresponding columns of $[P_1]$ must equal $\underline{s}_1$.

To find possible double error patterns, represent the columns of $[P_1]$ by vectors $p_1, p_2, \ldots, p_m$

$$[P_1] = [p_1\ p_2\ p_3\ \cdots\ p_m].$$

A possible double error in bits i and j must satisfy

$$p_i + p_j = \underline{s}_1$$

where

$$i \neq j.$$

Considering the vectors p_i and $\underline{s}_1$ as binary numbers, construct a table of the 2^{r-1} pairs of numbers satisfying the above equation. Under each entry of the table place the index of the columns which have this value. The possible double error patterns are then taken from this table. As an example, consider the reduced parity check matrix of Section 2.3 and the received vector

$$\underline{a}^t = [00000\ 00000\ 10100\ 01000\ 001].$$

The first three row of H generates the partial syndrome

$$[P_1]\underline{a}_1 = \underline{s}_1$$

$$\begin{matrix} 001110101101100 \\ 111101000111010 \\ 011011011000101 \end{matrix} \quad \underline{a}_1 = \begin{matrix} 1 \\ 1 \\ 1 \end{matrix}$$

Table 7-1 shows the possible double error patterns.

Table 7-1. Possible Double Error Patterns

P_i	p_j	i	j	Possible Double Error Patterns
000	111	-	3	-
001	110	8,15	4,10,2	(8,4) (8,10) (8,2) (15,4) (15,10 (15,2)
010	101	1,11,14	5,9,3	(1,5) (1,9) (1,3) (11,5) (11,9) (11,3) (14,5) (14,9) (14,3)
011	100	2,6	7	(2,7) (6,7)

In this example there are only 17 possible double error patterns when considering the partial syndromes, as compared to

$$\binom{15}{2} = 105$$

total pairs of columns. The actual error pattern (11,13) is among these.

The average number of t-error patterns, $E_t(m)$, that can occur in a partial received vector of dimension m and that are consistent with a given partial syndrome can be estimated using a combinatorial argument if it is assumed that, on the average, all possible values for the columns of $[P_1]$ and also all values of the syndrome are equally likely.

Given the partial parity check matrix $[P_1]$ with r rows and m columns, there are 2^r possible values for the syndrome and for the columns of $[P_1]$. The probability that zero errors in the first m bits satisfy the parity check equations is the probability of a zero syndrome, or

$$E_0(m) = \frac{1}{2^r} .$$

Single errors that satisfy $[P_1]$ imply that a column of $[P_1]$ equals the syndrome. If all syndromes are equally likely, then the probability that a given column equals that value is $1/2^r$. For a matrix of m columns

$$E_1(m) = \frac{m}{2^r} .$$

Possible double error patterns are those whose corresponding columns sum to the syndrome. For a given syndrome there are $u = 2^{r-1}$ pairs of values, a and b, whose sum equals that syndrome. If there are n_a columns with value a, and n_b columns with value b, then the contribution of these columns to the total number of error patterns is $n_a n_b$. Considering $k = n_a + n_b$ fixed, the average value of the product $n_a n_b$ is

$$F_2(k) = \frac{\sum_{n_a=0}^{k} \binom{k}{n_a} n_a n_b}{2^k} = \frac{1}{2}\binom{m}{2} .$$

The probability that there will be k columns out of m with values a or b is

$$\frac{\binom{m}{k} 2^k (2u-2)^{m-k}}{(2u)^m} .$$

There are u equally likely pairs, each with an average value of $n_a n_b$ equal to $F_2(k)$. Therefore, the average of the total number of double error patterns is

$$E_2(m) = \frac{u \sum_{k=0}^{m} \binom{m}{k} 2^k (2u-2)^{m-k} F_2(k)}{(2u)^m}$$

Simplifying this expression,

$$\frac{u \sum_{k=0}^{m} \binom{m}{k} (u-1)^{m-k} \frac{1}{2}\binom{m}{2}}{u^m}$$

$$= \frac{u}{2} \sum_{k=0}^{m} \binom{m}{2} \binom{m-2}{k-2} \frac{(u-1)^{m-k}}{u^m}$$

$$= \frac{u}{2} \sum_{\ell=0}^{m-2} \binom{m}{2} \binom{m}{\ell} \frac{(u-1)^{m-2-\ell}}{u^m}$$

$$= \frac{u}{2} \binom{m}{2} \frac{u^{m-2}}{u^m}$$

$$= \frac{1}{2u} \binom{m}{2}$$

$$E_2(m) = \frac{1}{2^r} \binom{m}{2}.$$

In general,

$$E_t(m) = \frac{1}{2^r} \binom{m}{t}.$$

The number of possible t-error patterns is reduced, on the average, by a factor of 2^r given a redundancy of r bits.

Considering only error patterns less than a given weight, increasing the number of redundant bits decreases the number of error patterns that need to be checked but increases the overall probability of error since fewer bits are erased. To improve the performance using this technique, one can either increase the maximum weight of the error patterns that are checked, or one can use the erasure masks and error patterns scheme of Section 6.

Returning to the (128,64) BCH code as an example, the probability of an error of weight greater than t, $0 \leq t \leq 4$, as a function of the number of bits not erased for a signal-to-noise ratio of 2.0 dB is given in Figure 7-2. At this signal-to-noise ratio, in order to closely approach maximum likelihood performance, one must at least test all triple error patterns and some four error patterns if the redundancy is greater than 5 bits. With such a large number of error

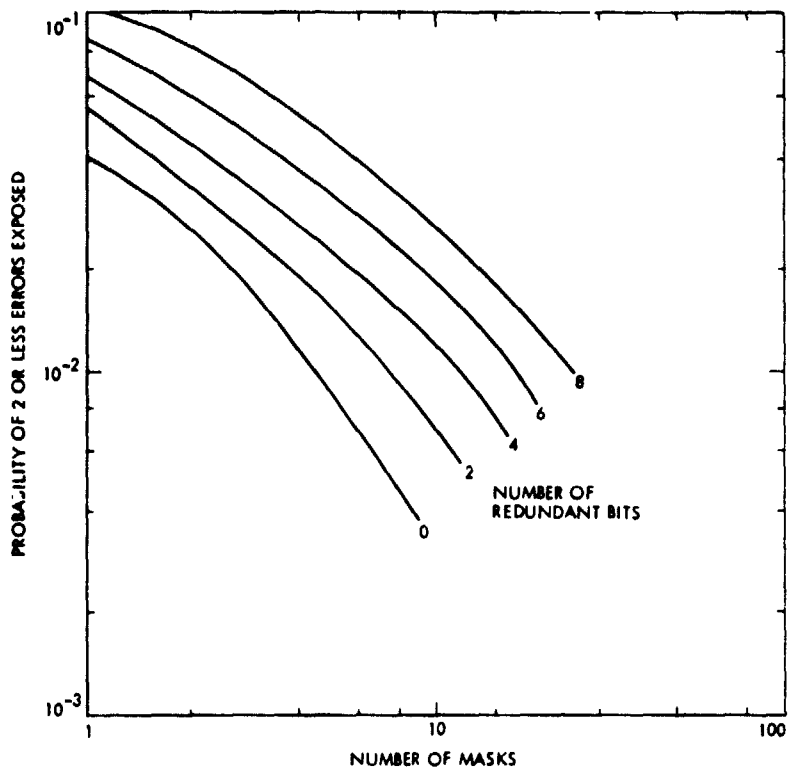

Figure 7-2. Performance as a Function of Number of Redundant Bits and Number of Masks.

patterns, it is more efficient to use sets of erasure masks and test only for double error patterns.

7.3 GENERATING AND TESTING CANDIDATE CODE WORDS

For each of the possible error patterns a code word is generated and correlated with the received vector. The form of correlation which is most suitable is to minimize

$$\sum_i e_i \, |L_i|$$

where $\underline{e} = (e_1, e_2, \ldots, e_n)^t$ is the error vector and L_i is the amplitude of the ith received bit. The errors in the unerased bits are those calculated in the previous step, while those in the erased bits can be found by a small number of vector additions, at most equal to the weight of the error in the unerased bits. Given the error vector, the correlation is calculated by summing those amplitudes for which $e_i \neq 0$. On the average, this will be about half the number of erased bits in the summation.

7.4 AN ESTIMATE OF THE COMPLEXITY OF THE ALGORITHM

As each symbol is detected, it is linked into the sorting table. Nothing more can be done until the entire word is received, at which time the amplitudes are sorted. This requires only one pass through the list which contains 128 + 64 = 192 items assuming the amplitude is quantized to six bits magnitude.

The remainder of the algorithm, as seen in Figure 7-1 is a single loop containing almost all of the required computation. Each pass through the loop is independent and can be done in any order or in parallel. Considering only a single pass, the first operation is to permute the columns of the parity check matrix and reduce the permuted matrix to standard form. This is most easily done by copying the columns, in permuted order, into the area of memory that will be accessed by the reduction algorithm.

The unpermuted H matrix is stored in reduced form so that half the columns contain only a single one. After the initial permutation, on the average, half of these columns correspond to erased bits and can be ignored in the reduction. The number of checks that must be made to see whether there is a zero or one in a particular bit position in the pivotal row is

$$\binom{32}{2} = 496.$$

If half of these are ones, 248 vector additions are required.

The calculation of the correlation requires approximately 30 arithmetic additions per candidate code word. Because of the large number of operations required, a real time decoder using this algorithm should have a special arithmetic unit performing the summations. Ideally, the correlator should work as fast as the candidate code word generator so that each code word may be checked as it is generated. It may be possible to perform an approximate calculation, with negligible loss of performance, directly from the error vector based upon its weight and the position of the errors but this has not yet been investigated.

The redundancy corresponding to the minimum amount of computation requires a number of assumptions. First, a received signal-to-noise ratio of 2.0 dB is still assumed. At this signal-to-noise ratio the bit probability of error of a maximum likelihood decoder is 10^{-3}, which is close to the tolerable limit for most applications. For higher signal-to-noise ratios, the performance of decoding algorithms of this type with a fixed set of parameters generally gets better, that is, approaches the maximum likelihood performance more closely. Second, the level of performance that will be assumed for this signal-to-noise ratio is a probability of 0.9×10^{-3} that an error pattern will not be covered or corrected. With these assumptions the minimum occurs for a redundancy of 6. At this redundancy 20 erasure masks are required to achieve the desired performance.

The two major units of the decoder are the matrix reducer and the candidate code word generator and correlator. Using the parameters mentioned previously, 20 matrix reductions are performed per received code word and approximately 40 candidate code words are generated per matrix reduction. If the matrix reductions are done serially, the time required for these two operations should be the same so that each unit will not have to wait for the other to complete its computation.

Assuming the speed of the algorithm is fixed by the time required for the matrix reduction, the time required to decode a single code word is approximately the time required to reduce the matrix 20 times. A conservative estimate is that the initial reduction can be done in 1000 steps and subsequent reductions in 200 steps, or 5000 steps overall. At a computation rate of 100 nanoseconds per step, one word can be decoded in 0.5 milliseconds. There are 64 information bits per code word so that at this computation rate a data rate of 128 kbits/second can be handled. Higher data rates will require a faster rate of computation or more parallelism while lower data rates can shift some of the computational overhead and bookkeeping from special purpose to general purpose hardware.

SECTION 8

SUMMARY AND CONCLUSIONS

Approximate maximum likelihood decoding algorithms based upon selecting a small set of candidate code words to be correlated with the received vector, can give a close to optimum performance with a reasonable amount of computation. This report describes the search for computationally efficient algorithms of this type. Emphasis has been placed upon the (128,64) BCH code of minimum distance 22 since it is one of the shortest codes whose maximum likelihood performance at low signal-to-noise ratios is better, by more than 1 dB, than the rate 1/2, encoding constraint length 7 convolutional code in wide use today.

The most efficient algorithm found for decoding this code is one which sorts the received bits according to their estimated probability of error and then selects candidate code words by:

(1) using a small number of erasure masks,

(2) assuming errors of weight two or less in the unerased bits,

(3) using six bits of redundancy to reduce the number of error patterns that need be checked, and

(4) using the computationally most efficient algorithm at each step of the decoding.

This algorithm is competitive with the Viterbi decoding of the (7,1/2) convolutional code in number of computations though not in simplicity of the program.

The principle found most useful in developing the algorithm is to take as much advantage as possible of the sorting of the received bits according to their estimated probability of error. This is best illustrated by the increase of efficiency when using weighted masks rather than combinatorially generated ones which consider all erased bits equally. In addition, maximum utilization should be made of the structure of the code. A possible explanation of the relative efficiency of this decoding algorithm is, that in addition to the use of linearity of the code to calculate the erased bits, it is also used to correct errors in the unerased ones. E. Berlekamp's soft decision decoding algorithm (Reference 1-7) also takes advantage of the cyclic and algebraic structure of the code. If a way can be found to use these properties for decoding random errors, it is highly probable that such a scheme will be even more efficient for a given level of performance than those described in this report.

In the algorithms of Sections 5, 6 and 7, sets of weighted erasure masks were used. These sets were first defined by L. Baumert and R. McEliece (Reference 1-4) and were constructed using a random

number generator. For sets with a large number of masks this is the easiest method of constructing them and, as Figure 5-6 shows, it is probably close to the best. However, for sets with a small number of masks, as those of Section 7, a more deterministic method may be superior. Constructing such sets, which have given covering and distance properties, is an interesting combinatorial problem and should be pursued further.

The advantage of the algorithms described in this report over that of the Viterbi decoding of convolutional codes is due mainly to the ability to decode codes of higher Q for a given computational complexity. It is very likely that an exponential increase of complexity will be necessary if an attempt is made to decode larger codes of higher Q just as is now the case with Viterbi decoding. The algorithms discussed here are useful for codes up to the length for which the exponential increase in complexity begins. It would be worthwhile to find the length for which this occurs, for it is at this point that these algorithms operate at their best.

REFERENCES

1-1 Forney, G. D., Jr., "The Viterbi Algorithm," Proc. IEEE, Vol. 61, March, 1973, pp. 268-276.

1-2 Chase, D. "A Class of Algorithms for Decoding Block Codes with Channel Measurement Information," IEEE Trans. Inform. Theory, Vol. IT-18, January, 1972, pp. 170-182.

1-3 Chase, D. and Goldfein, H. D., "Long Block Codes Can Offer Good Performance," Information Theory International Symposium, Cornell, N.Y., October, 1977.

1-4 Baumert, L. D., and McEliece, R. J., "Soft Decision Decoding of Block Codes," The Deep Space Progress Report 42-47, Jet Propulsion Laboratory, Pasadena, CA., August, 1978, pp. 60-64.

1-5 Dorsch, B. G., "A Decoding Algorithm for Binary Block Codes and J-ary Output Channels," IEEE Trans. Inform. Theory, Vol. IT-20, May, 1974, pp. 391-394.

1-6 Forney, G. D., Jr., Concatenated Codes, Chapter 3, The MIT Press, Cambridge, Mass., 1966.

1-7 Berlekamp, E. R., "Long Block Codes Which Use Soft Decisions and Correct Erasure Burst Without Interleaving," Conference Record, National Telecommunications Conference, Los Angeles, Calif., December, 1977, pp. 36:1.1-36:1.2.

2-1 Bahl, L. R., Cocke, J., Jelinek, F., and Raviv, J., "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," IEEE Trans. Inform. Theory, Vol. IT-20, March, 1974, pp.284-287.

2-2 Hartmann, C., and Rudolph, L., "An Optimum Symbol-by-Symbol Decoding Rule for Linear Codes," IEEE Trans. Inform. Theory, Vol. IT-22, September, 1976, pp. 514-517.

2-3 Greenberger, H., "An Iterative Algorithm for Decoding Block Codes Transmitted over a Memoryless Channel," The Deep Space Progress Report 42-47, Jet Propulsion Laboratory, Pasadena, CA., August, 1978, pp. 53-59

5-1 MacWilliams, F. J., and Sloane, N. J. A., The Theory of Error Correcting Codes, North-Holland Publishing Co., New York, N.Y., 1977, pp. 58-64.

5-2 Baumert, L., McEliece, R., and Solomon, G., "Decoding with Multipliers," The Deep Space Network Progress Report 42-34, Jet Propulsion Laboratory, Pasadena, CA., August, 1976, pp. 43-46.

A-1 Knuth, D. E., The Art of Computer Programming, Vol. 3, Sorting Algorithms, Addison Wesley, Reading, Mass., 1966.

APPENDIX

EFFICIENT ALGORITHMS FOR SORTING AND MATRIX REDUCTION

A.1 INTRODUCTION

The sorting of real numbers can be done efficiently by many algorithms all requiring, on the average, n log n operations to sort n bits (Reference A-1). The sorting can be done even more efficiently with neglible loss of performance by quantizing the amplitudes to a fairly large number of levels. Representative of algorithms which sort into a finite number of bins is the linksort, which requires on the order of L + N (L = number of bins, N = number of items to be sorted) operations.

Reducing the parity check matrix to standard form can best be done by Gaussian reduction. In this case the matrix elements can only have the values 0 or 1 and the computation is best done as vector exclusive or addition. In the general case the reduction requires on the order of $n^2/2$ such operations, where n = the rank of the matrix. As seen in Section 5.3, the matrix can be arranged beforehand in such a way so that the number of operations is considerably less than this.

A.2 SORTING THE RECEIVED SYMBOLS ACCORDING TO THEIR RELIABILITY

Common to all the algorithms analyzed here is the sorting of the received symbols according to their reliability. It is assumed that the received signal has been demodulated and reduced to a form in which each symbol is represented by a real number of the form $y_i = s_i + n$ where $s_i = \pm 1$ and n is an independent sample of a zero mean Gaussian process with variance

$$\sigma^2 = \frac{N_0}{2E_bR} = \frac{1}{2\beta R} .$$

This can be shown to be a sufficient statistic; that is, it represents the received signal without loss of information.

A good measure of the reliability of each bit is the absolute magnitude of its log likelihood ratio

$$L_i = \ell n \frac{\Pr(y_i|x_i=+1)}{\Pr(y_i|x_i=-1)} .$$

This measure is intuitively satisfying and, in addition, is required for estimating the transmitted code word. For the memoryless Gaussian channel

$$L_i = \ell n \frac{\frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(y_i-1)^2}{2\sigma^2}\right]}{\frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(y_i+1)^2}{2\sigma^2}\right]} = \left(\frac{2}{\sigma}\right) y_i$$

so that the absolute magnitude of the received symbol is a measure of the reliability of that symbol and the required sorting can be done on the received vector directly.

An efficient algorithm for sorting amplitudes into a small number of bins is the linksort. The version described here sorts integers according to absolute value as required for ordering the received symbols according to their relative reliability. The algorithm requires only a single pass through the data to construct a table and a single pass through the table to output the integers in sorted order along with their input index. The number of memory locations required is the number of items to be sorted plus the absolute value of the largest integer on the data list.

A flow graph of the algorithm is shown in Figure A-1. The variables used in the algorithm are:

n = number of items to be sorted

m = absolute value of largest integer in data list

L_i = list of integers to be sorted, $1 \leq i \leq n$

A_i = output list of integers sorted according to absolute value

P_i = output list of indices associated with each A_i

S_j = special links used internally in algorithm, $0 \leq j \leq m$

The algorithm can be divided into three stages:

(1) Initialize the chain. The S_i are the special links in the chain which point to data of absolute amplitude i. At the beginning of the algorithm each special link points to the next special link in the chain. The value of z can be any number larger than n so as to be able to distinquish between pointers to special links and pointers to data links. If z is

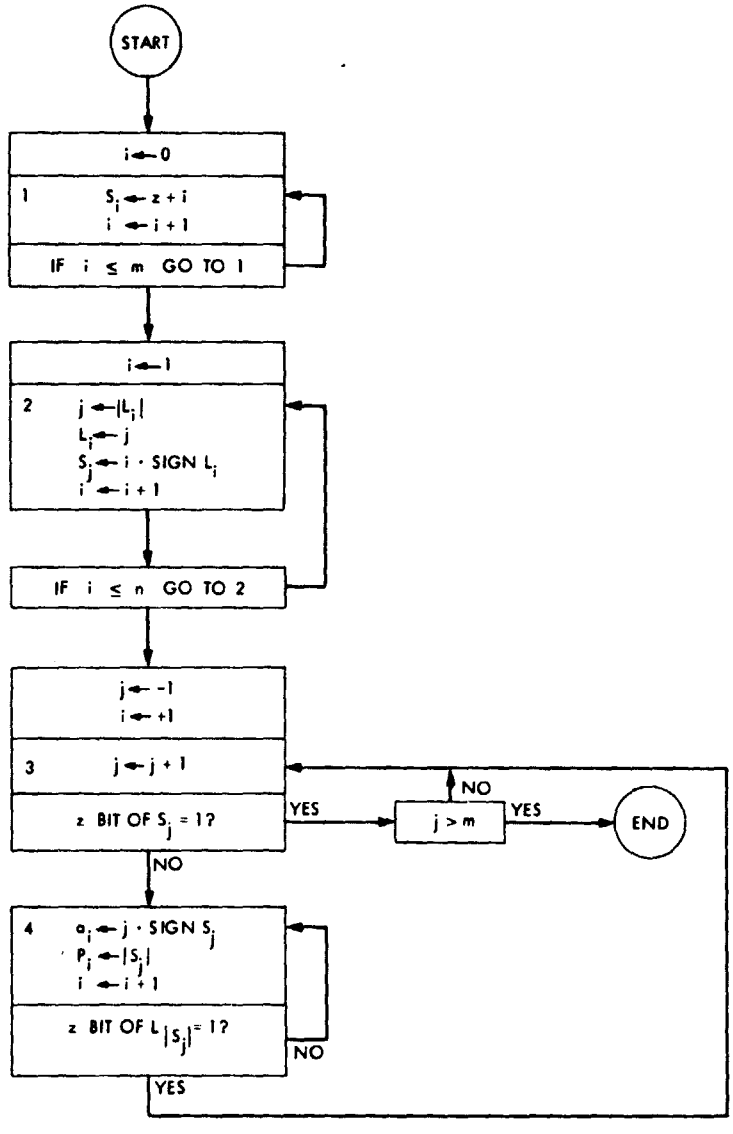

Figure A-1. Flow Chart of the Linksort Algorithm

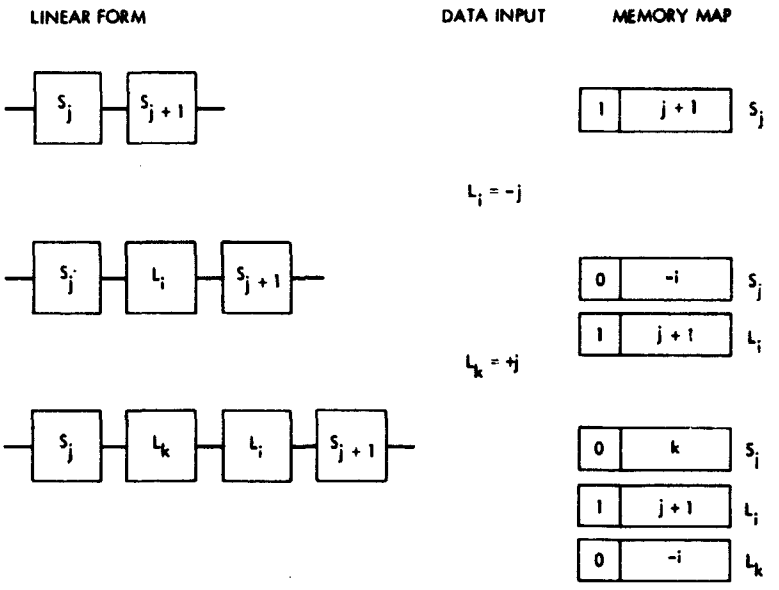

Figure A-2. Schematic Representation of the Links in the Linksort Algorithm

the first power of 2 greater than n, then special links will be characterized by a 1 in the highest order bit. This can be thought of as a special sign bit.

(2) Link data into the chain. The ith data symbol has absolute magnitude j. The contents of memory location S_j are stored in L_i and the index i, along with the sign of the ith data symbol, is stored in S_j. The memory locations can either contain a special pointer, indicated by the z bit pointing to the next amplitude, or an index i, depending on whether or not the amplitude j has been encountered before. These two cases can be schematically represented as in Figure A-2.

(3) Construct the output amplitude and index chains. The chain is read out starting at location S_0. Pointers with the z bit equal to zero correspond to data of amplitude j and position index i. Those with the z bit equal to one correspond to a special link indicating an amplitude increase of one. The algorithm is completed when the last link in the chain is detected.

In order to demonstrate this algorithm, a step-by-step development of the arrays {L} and {S} is shown in Figure A-3. The input list contains 12 numbers with maximum amplitude 6. The value

	MEMORY LOCATIONS																		
DATA	1	2	3	4	5	6	7	8	9	10	11	12	100	101	102	103	104	105	106
INITIAL STATE													101	102	103	104	105	106	107
4	105												101	102	103	104	1	106	107
-3	105	104											101	102	103	-2	1	106	107
0	105	104	101										3	102	103	-2	1	106	107
-6	105	104	101	107									3	102	103	-2	1	106	-4
2	105	104	101	107	103								3	102	5	-2	1	106	-4
3	105	104	101	107	103	-2							3	102	5	6	1	106	-4
5	105	104	101	107	103	-2	106						3	102	5	6	1	7	-4
-4	105	104	101	107	103	-2	106	1					3	102	5	6	-8	7	-4
4	105	104	101	107	103	-2	106	1	-8				3	102	5	6	9	7	-4
6	105	104	101	107	103	-2	106	1	-8	-4			3	102	5	6	9	7	10
0	105	104	101	107	103	-2	106	1	-8	-4	3		11	102	5	6	9	7	10
0	105	104	101	107	103	-2	106	1	-8	-4	3	11	12	102	5	6	9	7	10

Figure A-3. Development of the Chain in the Linksort Algorithm

of 100 is used for z rather than a power of 2 since, in an example, decimal numbers are visually more convenient.

In an actual decoding algorithm many more amplitude levels would be used. The number of memory locations required for the arrays are $3n + q$ (where n = the number of symbols to be sorted and q = the number of amplitude levels). The amount of computation required in building the linked list is independent of q, while reading out the data requires $n + q$ steps. For a code of block length 128, q equal to 64 or 128, corresponding to 6 or 7 bits plus a sign bit, is reasonable. The advantages of fine quantization are that the decoder is much less sensitive to changes in noise and signal power and the quantization loss, which is on the order of 0.2 to 0.25 dB for 3 bit quantization usually used in soft decision decoders, is negligible.

A.3 REDUCING THE PARITY CHECK MATRIX TO STANDARD FORM

The decoding algorithms which assume erasures calculate the erased bits from a linear combination of the unerased ones. By permuting the columns of the parity check matrix which correspond to the erased bits to the right, the matrix may then be partitioned into two parts, a $k \times (n - k)$ partition corresponding to the known bits $[P_1]$ and a $(n - k) \times (n - k)$ partition corresponding to the unknown ones $[P_2]$.

The entire parity check matrix is then reduced by row operations until $[P_2]$ is in triangular form. The parity check equations require

$$[H]\underline{a} = \underline{0} \quad \text{or} \quad [P_1 | P_2] \left[\begin{array}{c} \underline{a}_1 \\ \hline \underline{a}_2 \end{array}\right] = \underline{0} .$$

After reduction this becomes

$$\left[\begin{array}{c|ccc} & 1 & & \\ & & 1 & \\ P_1 & & & 1 \\ & & & \end{array}\right] \left[\begin{array}{c} \underline{a}_1 \\ \hline \underline{a}_2 \end{array}\right] = \underline{0} .$$

Considered as a set of simultaneous linear equations with all the bits of $\underline{a}_1$ known, one can, starting with the first bit of $\underline{a}_2$, calculate the values of the erased bits from those of $\underline{a}_1$ and the previously calculated ones of $\underline{a}_2$. The procedure is straightforward unless there is a zero element along the diagonal of $[P_2]$. This indicates that the bit corresponding to the column containing the zero is independent of the bits in $\underline{a}_1$ and that there exists a dependency relationship among the bits of $\underline{a}_1$ so that not all values are permissible.

In order to handle this case simply, it is worthwhile to reduce the matrix further. By row operations all the off-diagonal terms of the columns of $[P_2]$ which contain a one along the diagonal can be reduced to zero, and all the rows of $[P_2]$ which contain a zero along the diagonal can be reduced completely to zero. The form of the H matrix is now

$$[H] = \left[\begin{array}{c|ccccc} & 1 & & & & \\ & & 0 & & & \\ P_1 & & & 1 & & \\ & & & & 1 & \\ & & & & & 1 \end{array}\right] \leftarrow \text{all zeros in this row}$$

$\uparrow$ possible ones in this column off the diagonal

The matrix can be simplified by permuting the zero row to the top, and the zero column, along with its corresponding bit in $\underline{a}_2$, to the left. The parity check equations are then

$$[H]\underline{a} = \underline{0}; \qquad \left[\begin{array}{c|c|c} P_{1a} & 0 & 0 \\ \hline P_{1b} & P_3 & I \end{array}\right] \left[\begin{array}{c} \underline{a}_1 \\ \hline \underline{a}_{2a} \\ \underline{a}_{2b} \end{array}\right] = \underline{0} .$$

This corresponds to three sets of equations:

(1) $[P_{1a}]\underline{a}_1 = \underline{0}$

(2) $\underline{a}_{2a}$ arbitrary

(3) $\underline{a}_{2b} = [P_{1b}]\underline{a}_1 + [P_3]\underline{a}_{2a}$

The first equation expresses the dependency relationships among the unerased bits which must be satisfied if a solution is to exist. If the bits of $\underline{a}_1$ satisfy these equations, then the number of possible code words with these unerased bits is 2^r (r = the number of bits in $\underline{a}_{2a}$), all of which can be used for candidate code words.

The reduction of the parity check matrix requires a significant portion of the total computation time of the algorithm so that it is worthwhile to seek efficient techniques to perform this calculation. First, permuting the rows and columns need not actually be done. It is sufficient to keep a table of the proper order of the row and column indices. Second, most of the operations in reducing the matrix and generating candidate code words are performed most efficiently as vector additions. At various stages of the algorithm both rows and columns are treated as vectors, and converting from one to the other in the

computer is a time consuming operation. This is not necessary, however, since row operations may be performed on data stored as columns. The flowchart of a matrix reduction algorithm which represents each column of the H matrix as a binary vector stored in a single word of memory is shown in Figure A-4. The algorithm reduces this matrix by row operations without converting from column vector to row vector form.

With this algorithm, the time required to arrange the data for calculation is a minimum and the overall decoding time can be estimated from the number of vector additions that have to be performed.

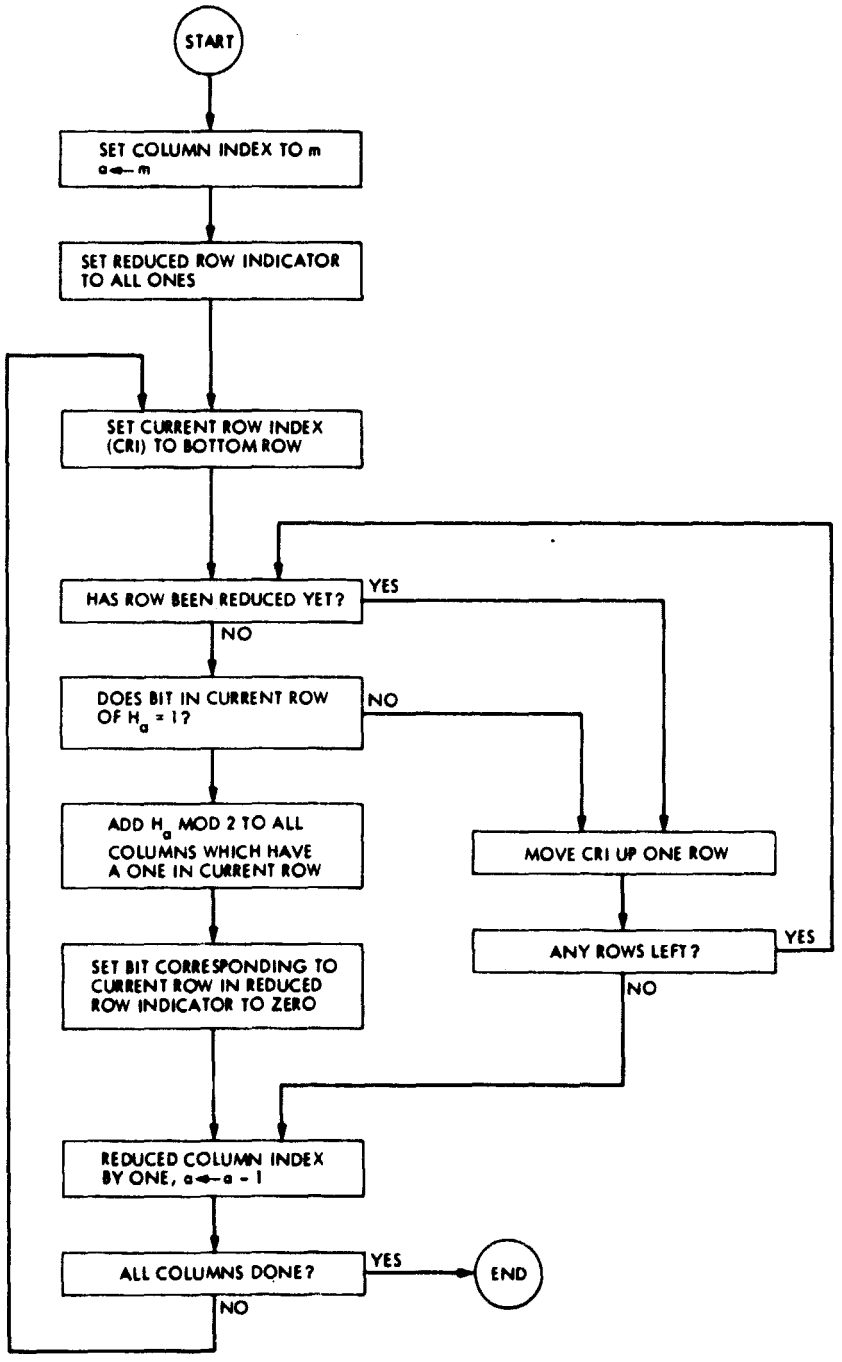

Figure A-4. Matrix Reduction Algorithm