" A binary-ternary code with restricted error extension "

A. Pouliezos

Dissertation submitted in fulfilment of the requirements of the CNAA for the award of the B.Sc. Honours degree in Mathematics and Computing.

Polytechnic of North London, 1975.

Contents

Abstract

This project deals with a method, suitable for computer use, which calculates the best encoding scheme for a 4B3T code.

A 4B3T code is a code used in the transmission of signals over cable. The original message is a string of binary blocks consisting of four digits each and the coded message consists of blocks of three digits from a ternary alphabet (-, +, 0 in our case).

The ternary words are divided into five groups according to their disparity (=sum of signs in a ternary word), the fifth group consisting of the word 000.

Since there are 16 binary and 27 ternary words it is obvious that we cannot have an one-to-one mapping, but some binary words would have to be mapped onto more than one ternary word.
Advantage is taken of this fact so that the accumulated disparity in the transmitted signal is contained near zero.

The requirements for a best encoding scheme are two:
(i) Let two equivalent words be:

$a_1a_2a_3a_4 = x_1x_2x_3$ or $a = x$

$b_1b_2b_3b_4 = y_1y_2y_3$ or $b = y$, where

$a_i, b_i \in \{0,1\}$ $i = 1, 2, 3, 4$

$x, y \in \{+, -, 0\}$ $i = 1, 2$

Then, if x and y differ in one place, we want a and b to differ in at most two places.

(ii) Mean probability of error in a recovered binary digit : minimum.

This is given by :

$$p \sum_i \sum_j i j n_{ij} \; / \; 64$$

where p = probability of error

j = Hamming distance of the binary translations of two neighbouring ternary words (error weighting)

i = probability weighting

n_{ij} = number of neighbours with probability weighting i and error weighting j.

The problem was tackled geometrically and a certain grouping of the ternary words was considered as best. A program was then written to calculate the best mapping of the binary set onto the ternary set, and a multiplicity of solutions was obtained with best mean probability 79/64 = 1.234.

CHAPTER I

Preliminaries

1. Definition of problem and conditions

The problem is one of the many arising in modern telecommunication theory. It is concerned with the best encoding scheme (subject to the conditions) for binary pulse-code-modulation signals over cable.

The original messages consist of blocks of 4 binary digits each. Therefore, the set of the original codewords consists of the 16 binary 4-tuples. It is of general appreciation that the use of alphabets consisting of more than two letters in the encoding procedure can offer certain advantages such as a lower signalling rate and control over the spectral distribution of the energy contained in the line signal. Also redundant words may be used for monitoring line errors [2]. A ternary alphabet consisting of -, + and 0 was chosen for this particular problem. The messages are encoded into messages consisting of ternary blocks of three digits each. All the combinations of the ternary 3-digit words form a set of 27 members.

We define disparity as the sum of the digits of each ternary word expressed in units (e.g. disparity of +-+ = 1, of +++ = 3, of +-0 = 0, e.t.c.).

Since there are only 16 binary words and 27 ternary, 11 binary words will be mapped onto two ternary words and we choose these to be one of positive disparity and one of negative disparity. As mentioned in the abstract, we cannot have an one-to-one mapping from the binary words onto the ternary, and therefore some binary words will have to be mapped onto more than one ternary word. There exists a nice structure in the ternary words which makes it possible to separate them into two sets, one of which contains the non-positive disparity ones (the word 000 is never transmitted). Each set contains 16 words. Since we want to keep the accumulative disparity of the signal as low as possible, it looks reasonable to group the ternary words into sets containing the corresponding positive and negative disparities. Then we will have 6 binary words mapped onto 6 zero disparity ternary words and 10 binary words mapped onto 10 pairs of ternary words. The choice of the appropriate encoding ternary word depends on the accumulative disparity of the signal, since we want to keep it as near as 0 as possible. Therefore, if at a certain stage the accumulative disparity is greater than 0 we will choose the negative disparity word for encoding the binary word, and if it is negative, we will choose the positive. That is, if we can choose, which will not be the case if the encoding ternary word is of 0 disparity.

A transmitted signal is subjected to noise and other factors which can cause an error in it. The total error rate depends not only in the error rate of the ternary digit, but on the mapping also, because one ternary error may result in more than one binary errors. For example, if the translation table has the following two entries :

0000 = 00+

0111 = -0+

then a change in one ternary digit (1st) results in change in 3 binary digits (2nd, 3rd, 4th). Therefore the rate of binary errors can be quite higher than the rate for the ternary errors. (We assume that the occurence of a single ternary error is highly improbable, which makes it possible for us to ignore the possibility of two or three errors occuring in one ternary word).

The ternary words can be divided into 4 groups according to their disparity. The 000 word is not generated but can occur as a result of error.

We define the neighbours of any code word as the set of codewords (valid or not) into which it can be converted by a single ternary error in which :

(a) only one digit is in error,

(b) it is changed into an adjacent value, e.g. + to 0 but not to - .

In this paper only these codes are considered for which: A single ternary error produces at most a double binary error. This is a very severe restriction to our problem since it is expected that "better" codes can be found by removing the above restriction. But unless this is imposed the binary error rate will sometimes be much higher than the ternary error rate and this will have serious effects in circumstances , such as the digital coding of a TV signal , where the signals transmitted appear as dots of different intensity on the screen and errors in the transmitted signal may result in "blots". In this case, our aim is to minimise the possibility of errors occurring at one place of the TV screen. However, the above would not have the same bad effect in a multiplexing situation, such as serial telephone lines, where our aim is to minimise the overall error probability.

Referring to table 2 and 6, we define the probability weighs for neighbouring words (ternary words in this context are thought of as being composed from two modes, positive and negative ; zero disparity words contain both modes) as :

(i) 1/2 between words neighbouring in one mode only;

(ii) 1 between words for which both modes of one are neighbour to one mode of the other or v.v.;

(iii)2 between words for which both modes of one are neighbour to both modes of the other.

This follows the assumption that all modes are equiprobable and therefore relationship (ii) is half as probable than (iii) and (i) half as probable than (ii).

Let p denote the probability of a single error in a ternary word. Suppose there are, n_1 neighbours of probability weighting 2, n_2 neighbours of probability weighting 1 and n_3 neighbours of probability weighting 1/2. Then,

Mean probability of ternary error in a word : $p/16\ (2n_1+n_2+n_3/2)$ ....(1)

For the probability of binary error we must consider the number of binary errors generated by one ternary error. If the binary translations of two ternary neighbours differ in j digits, we shall describe the resulting errors as having <u>error weighting</u> j.

Let n_{ij} be the number of neighbours with probability weighting i and error weighting j, where i = 1/2, 1, 2; j = 1, 2, 3, 4.
Then,

Mean probability of error in a decoded binary digit : $p/64 \sum_i \sum_j i j n_{ij}$ ....(2)

2. Previous attempts and results

Previous attempts for the problem were made by K.W.Cattermole, R.R.Ellis and P.W.Wallace at the Department of Electrical Engineering Science, University of Essex. First attempts did not aim at optimisation and the mapping was arbitrary for easy instrumentation (quadraple errors occurring as well). Later attempts aimed at improvement of the binary error rate and they were concentrated on three main ways of doing it :

(a) Different mapping for reduction of multiple errors;

(b) Different grouping of ternary words;

(c) Optimal interpretation of received words which have been rendered invalid.

The original set of ternary words appears in table 1, type I.
In this case ,

$n_1 = 0,\ n_2 = 51,\ n_3 = 24$

∴ mean probability of ternary error : $p/16(51+24/2) \sim 4p$

For the binary error rate, we note that j ranges from 1 to 4.
We have,

$n_{1,1} = 14,\ n_{1,2} = 24,\ n_{1,3} = 11,\ n_{1,4} = 2$

$n_{1/2,1} = 6,\ n_{1/2,2} = 12,\ n_{1/2,3} = 6$

$\therefore$ mean probability of binary error : p/64(14+2.24+3.11+2.4+1/2(6+2.12+3.6))

= 127p/64 $\sim$ 2p

The above results are easily seen from tables 2 and 3, where the entries in the matrices are i and j respectively. In this example we note that nearly three quarters of the errors are multiple.

Better codes were found by noticing structural properties and the best one found is type II in table 1. The total error weight is 100 and therefore mean probability of binary error is reduced to 1,56, and the maximum error weight j is 2 (table 4). Choosing a set of codewords with fewer neighbours produced code III in table 5.

It can be seen from table 6 and 7 that the number of ternary neighbours was reduced from 75 to 63 and the mean probability of error from 1.56p to 1.32p.

Table 1

4B3T Codes Types I and II

Ternary Word				Binary Word	
Disparity	Designation	Code word in		Code	Code
		Positive mode	Negative mode	I	II
0	A1	+ - 0		1 0 0 1	0 1 0 1
	A2	0 - +		0 0 0 0	0 0 0 1
	A3	- 0 +		0 0 1 0	1 0 1 0
	A4	- + 0		0 0 0 1	0 1 1 0
	A5	0 + -		1 0 0 0	0 0 1 0
	A6	+ 0 -		1 0 1 0	1 0 0 1
±1	B1	+ + -	- - +	1 1 1 0	1 0 0 0
	B2	+ 0 0	- 0 0	1 0 1 1	1 1 0 0
	B3	+ - +	- + -	0 0 1 1	0 1 0 0
	B4	0 0 +	0 0 -	0 1 1 0	1 0 1 1
	B5	- + +	+ - -	0 1 1 1	0 0 0 0
	B6	0 + 0	0 - 0	0 1 0 1	0 1 1 1
±2	C1	+ + 0	- - 0	1 1 0 1	1 1 1 0
	C2	+ 0 +	- 0 -	1 1 0 0	1 1 0 1
	C3	0 + +	0 - -	0 1 0 0	0 0 1 1
±3	D1	+ + +	- - -	1 1 1 1	1 1 1 1
Detected error	E	0 0 0		0 0 0 0	1 1 1 1

PROBABILITY WEIGHS

Ternary neighbours

	A1	A2	A3	A4	A5	A6	B1	B2	B3	B4	B5	B6	C1	C2	C3	D1	E
A1								1	1		1	1					
A2							1		1	1		1					
A3							1	1		1	1						
A4								1	1		1	1					
A5							1		1	1		1					
A6							1	1		1	1						
B1		½	½		½	½							1				
B2	½		½	½		½							1	1			1
B3	½	½		½	½									1			
B4		½	½		½	½								1	1		1
B5	½		½	½		½									1		
B6	½	½		½	½								1		1		1
C1							1	1				1				1	
C2								1	1	1						1	
C3										1	1	1				1	
D1													1	1	1		

Number of neighbours: $\sum_i n_i = 75$

Mean probability of ternary error: $\frac{p}{16}\sum_i i n_i = \frac{63}{16}p \triangleq 4p$

Table 2

Error weights, Code I

								1	2		3	2					
							3		2	2		2					
							2	2		1	2						
								2	1		2	1					
							2		3	3		3					
							1	1		2	3						
		3	2		2	1							2				
	1		2	2		1							2	3			3
	2	2		1	3									4			
		2	1		3	2								2	1		2
	3		2	2		3									2		
	2	2		1	3								1		1		2
							2	2				1				1	
								3	4	2						2	
										1	2	1				3	
													1	2	3		

If E → 0000 then

mean probability of binary error $\frac{p}{64}\sum_i\sum_j i j n_{ij} = \frac{127p}{64} \triangleq 2p$

Table 3

Error weights, Code II

							2	1		2	1					
						2		2	2		2					
						1	2		1	2						
							2	1		2	1					
						2		2	2		2					
						1	2		1	2						
	2	1		2	1							2				
2		2	2		2							1	1			2
1	2		1	2									2			
	2	1		2	1								2	1		1
2		2	2		2									2		
1	2		1	2								2		1		1
						2	1				2				1	
							1	2	2						1	
									1	2	1				2	
												1	1	2		

If E → 1111 then

mean probability of binary error $\frac{p}{64}\sum_i \sum_j ijn_{ij} = \frac{100P}{64} \simeq 1.56p$

Table 4

Table 5

4B3T Code Type III

Ternary Word				Binary Word
Disparity	Designation	Code word in		Code III
		Positive mode	Negative Mode	
0	A1	+ - 0		1 1 0 0
	A2	0 - +		0 1 0 0
	A3	- 0 +		0 0 1 0
	A4	- + 0		1 0 0 1
	A5	0 + -		1 0 1 0
	A6	+ 0 -		1 0 0 0
±1	G1	+ 0 0	0 - 0	1 1 0 1
	G2	+ - +	- - +	0 1 1 0
	G3	0 0 +	- 0 0	0 0 0 1
	G4	- + +	- + -	0 0 0 0
	G5	0 + 0	0 0 -	1 0 1 1
	G6	+ + -	+ - -	1 1 1 0
±2	F1	0 + +	- 0 -	0 0 1 1
	F2	+ + 0	0 - -	1 1 1 1
	F3	+ 0 +	- - 0	0 1 0 1
±3	D1	+ + +	- - -	0 1 1 1
Detected error	E	0 0 0		1 0 0 1

./..

Ternary neighbours, Code III

Ternary word generated

	A 1	A 2	A 3	A 4	A 5	A 6	G 1	G 2	G 3	G 4	G 5	G 6	F 1	F 2	F 3	D 1	E
A1							2	1				1					
A2							1	2	1								
A3								1	2	1							
A4									1	2	1						
A5										1	2	1					
A6							1				1	2					
G1	1	.				.								1	1		1
G2	.	1	.												1		
G3		.	1	.									1		1		1
G4			.	1	.								1				
G5				.	1	.							1	1			1
G6	.				.	1								1			
F1									1	1	1					1	
F2							1				1	1				1	
F3							1	1	1							1	
D1													1	1	1		

number of neighbours $\sum_i n_i = 63$

Mean pr. of ternary error $\frac{p}{16}\sum_i in_i = \frac{63p}{16} \sim 4p$

Table 6

Error weights, Code III

						1	2				1					
						2	1	2								
							1	2	1							
								1	2	1						
									2	1	1					
						2				2	2					
1	2				2								1	1		1
2	1	1												2		
	2	2	1									1		1		1
		1	2	2								2				
			1	1	2							1	1			1
1				1	2								1			
								1	2	1					1	
						1				1	1				1	
						1	2	1							1	
												1	1	1		

If E ⟹ 1001

mean pr. of binary error: $\frac{p}{64}\sum_i \sum_j ijn_{ij} = \frac{85p}{64} = 1.32\,p$

Table 7

CHAPTER II

Theoretical Considerations of the program

1. Number of mappings

It was noticed right from the beginning that the number of different mappings possible for the two sets was too big. If we define "difference" as merely any difference in a digit of a word, then the number of different combinations is :

$$6! \times 3! \times 16!$$

(since the number of different permutations of the 16 binary digits is 16!. Now suppose we consider a certain mapping of the positive ternary set onto the binary set; there are 16! such mappings. But now we can also find different mappings if we fix a mapping of the above 16! ones, and permute either the -1 disparity words in 6! ways or the -2 disparity words in 3! ways - hence the above result).
Now, 16! is approximately 21×10^{12} and the total number of combinations is approximately 8×10^{16}, large enough for any computer not to handle. Of course not all of the above codes are different in a mathematical sense, since, two codes can be said to be equivalent if by a finite number of the following operations, one can be transformed into the other :
(i) the interchange of any two positions in the code symbols. This is equivalent to the permulation of the digits of the words of the code symbols. The so obtained equivalent codes, form the permutation group S_4, with 24 members;
(ii) the complementing of the values of any position in the code symbols.

./..

The above operations are applied to each word of the code. The number of different codes is therefore reduced by a factor of $24.15 \sim 3x10^2$, which leaves the number of different combinations still very large. A complete scan is therefore quite impossible.

2. The binary and ternary sets as groups and their geometrical representation

It is noted that the set of 16 binary words forms a group under the operation addition mod2, with identity element 0000 and inverse of an element the element itself.

This group can be represented geometrically by a 4-D hypercube, as shown in figure 2.

The set of the ternary words also forms a group under the operation addition mod3. It is easier to think of this group, if we change -, 0, + into 0, 1, 2, respectively.

This group can also be represented geometrically by a 3-D cube as shown in figure 1.

3. Intuitive attempts

It was noticed that a change in a single digit of a ternary word changes this word into a word of adjacent disparity. Therefore, it seems logical to try and include in the sets of different disparities binary words as far apart as possible. We can begin by assigning to the words of zero disparity binary words which differ in 4 or 2 places, as follows:

+-0	0101
0-+	1111
-0+	1001
-+0	1010
0+-	0110
+0-	0000

We now find all the binary neighbours of the above binary words:

for 0101 : 1101, 1001 : 0001, 1010 : 1011, 0110 : 0111

0001	1101	1000	0100
0111	1011	1110	0010
0100	1000	0010	1110

for 1111 : 1110, 0000 : 0001

1101	0010
1011	0100
0111	1000

The union of these sets is :

1101, 0001, 0111, 0100, 1011, 1000, 1110, 0010.

We would like to distribute 6 of these words between the ternary words of disparity +1 and -1, and the remaining 4 between the +2, -2 and +3, -3 disparity words. After many trials it was impossible to match the two sets without the induction of multiple errors (more than in 2 places) and this methond was discontinued.

4. Group mappings

A homomorphism is defined as a mapping t from a group G onto a group G′, satisfying ,

(a) for every g in G there exists a unique gt in G′,

(b) (a.b)t = at x bt for every a,b in G (. is the operation of G, x is the operation of G′).

It seemed logical at first to try and construct a homomorphism from the group, say T, of the ternary words, onto the group, say B, of the binary words, and then find a rule which would give the best homomorphism for the problem. It was proved though that a homomorphism is not possible.
For, consider an element a in T, and let the order of a be n, that is $a^n = 1_T$. Let the homomorphism be s. Then,

$$(as)^n = a^n s = 1_T s = 1_B$$

But as is the image of a, and let

$$(as)^m = 1_B \text{ , where } m \leqslant n,\ n = qm+r,\ 0 < r < m.$$

Then,

$$(as)^n = (as)^{qm+r} = (as)^{qm} + (as)^r = 1_B + (as)^r$$

but

$$(as)^n = 1_B \therefore (as)^r = 1_B$$

This is a contradiction since r<m and m is the order of as. Hence the order of the image of a divides the order of a. But all the words in T have order 3 (except identity) and the words in B have order 2.

It was also seen that the two subsets of negative and positive ternary words do not form a group, and this route was discontinued as well.

5. Mappings

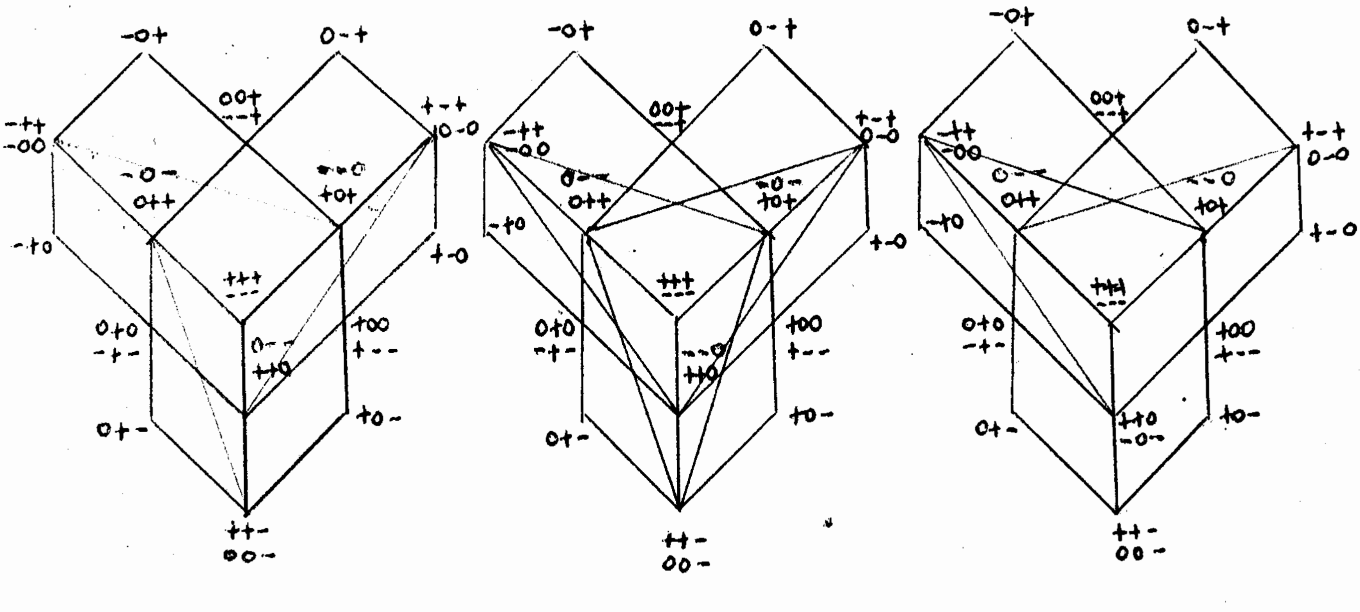
Next step was to try and put the binary words onto the geometrical shape of fig. 1, so as to make the distance between two adjacent words at most 2.
There exist some theorems related to this field, such as the one in relation to optimal binary coding of ordered numbers [1] , which states a method of optimal assigning of binary alphabets on cubes. These theorems are not suitable for encoding in a situation where the words of the encoding set consist of a bigger alphabet than the original words, such as the present case.

It was then noticed that the geometrical representation of the ternary words had a point symmetry about the centre of the cube and therefore it was possible to split the cube into two equal portions not necessarily with all elements distinct.
The elements in common were those of zero disparity. The two portions are as in fig. 4 and fig. 5.

Each of the two figures has 16 elements. It is possible therefore to associate a mapping of the positive words with one figure and then map the other figure on the first one in an optimising way. This optimising way should have as many common edges as possible, because our aim is to have as fewer restrictions to our problem as possible. The number of restrictions is equal to the number of edges in the final mapping, since along each edge the distance of the two words at its ends must be less than or equal to 2.

When the first figure is mapped onto thesecond, seven elements have fixed positions, namely the zero disparity elements and the +3 disparity element. The rest of the elements have also a limited number of places to go to(six for the +1 disparity elements, three for the +2)). So, there are 3!6! different mappings (some of thess mappings are equivalent, because of symmetry).

Let us suppose that we fix the positive subset of the ternary group, and then try to put the negative subset on it, trying to create as less extra edges as possible. It is possible to put the -1 disparity words onto the +1 disparity words without creating a-ny extra lines and this was done as shown in fig.6a.
After some thought it was seen that it is impossible to put the -2 disparity on the +2 words, without creating extra lines. With this in mind the best position for the -2 words was found to be the one in fig.6b.
The other possibilities for the -2 words were:



As it is seen from the preceding figures, we cannot do better than creating three extra edges, and the worst case of creating six extra lines is seen in the middle figure. If we try and associate the negative words in a different way, so that no extra lines are produced from the mapping of the -2 words onto the +2 words, then this would result in more extra lines produced after we tried to put the -1 words onto the +1 words.

In trying to do the above a number of assumptions were made, which though not proved mathematically, seemed intuitively correct. These are:

(i) The optimising way of mapping the two figures will produce the optimum code.
In this case one cannot exclude the possibility of obtaining an optimum code by not mapping the 2 figures in an optimum way but in some other manner.

(ii) The optimising mapping is one which produces as few extra lines as possible.

(iii)It is better not to consider the error word 000 until a code has been found and then try and find the "best" word for 000.

With the final mapping of fig. 6C it was decided to write a program to do an exhaustive search for finding the best code.

Figure 1.
Geometric representation of ternary alphabet.

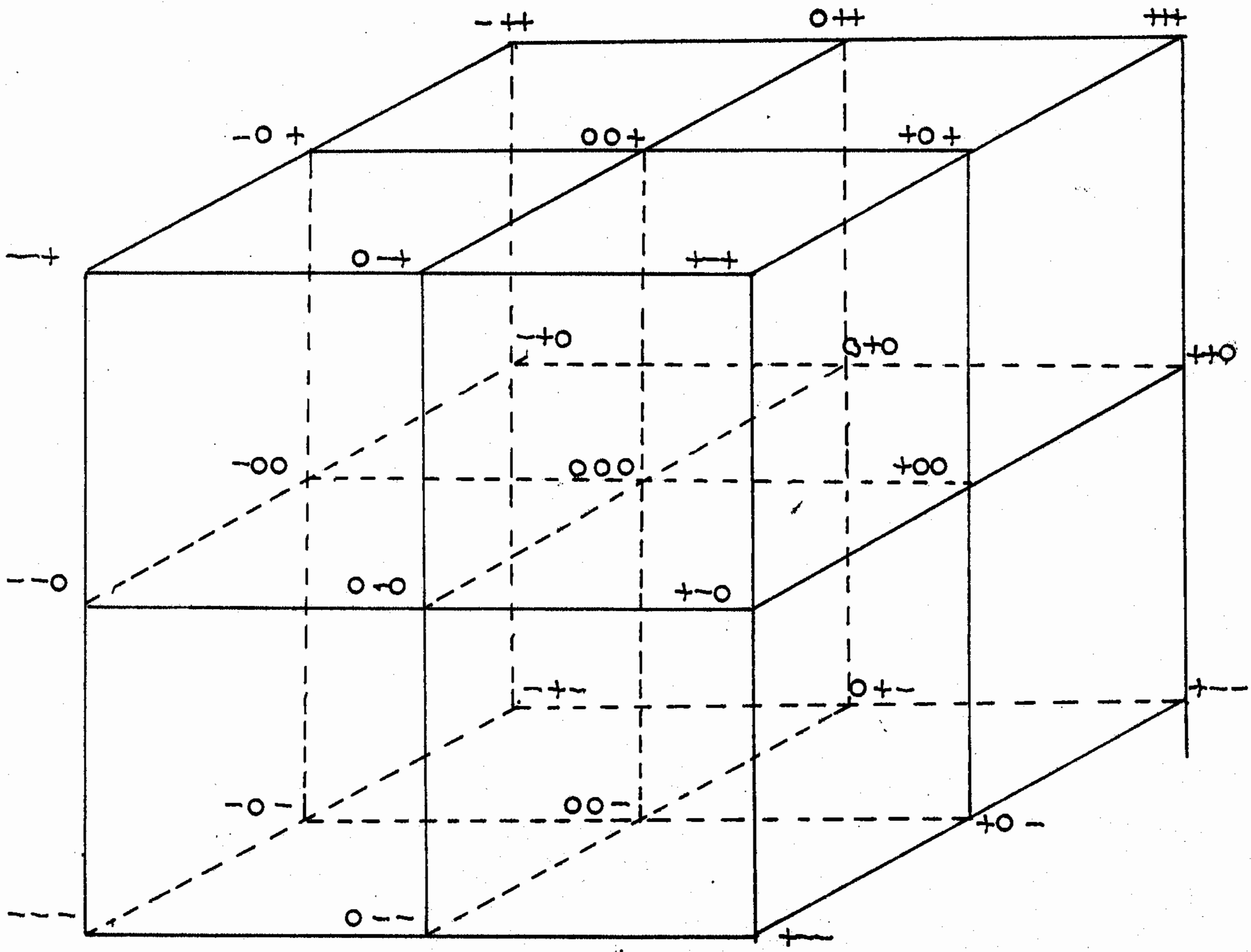
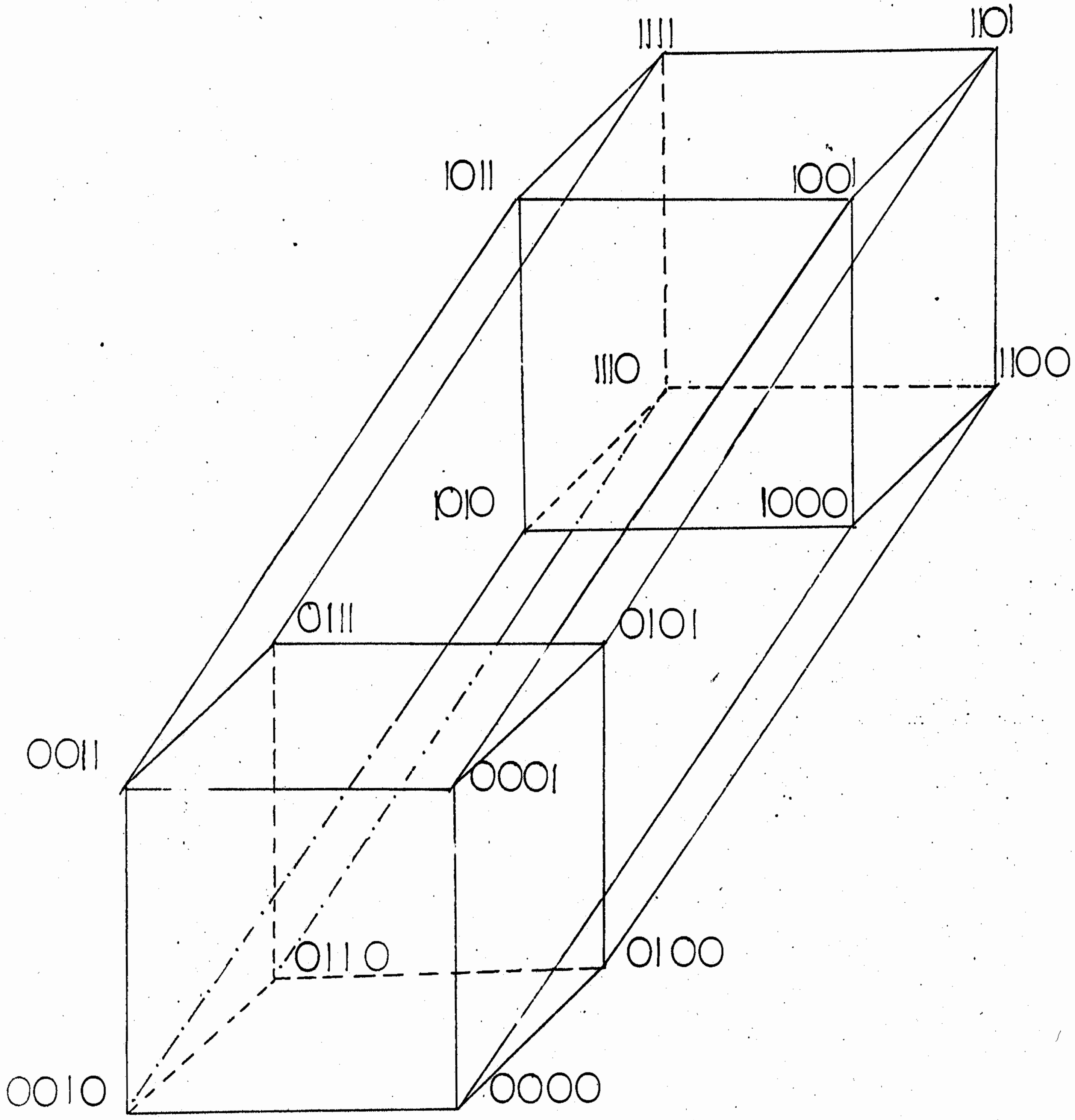


Figure: 2.
Geometric representation
of space of binary 4-tuples.



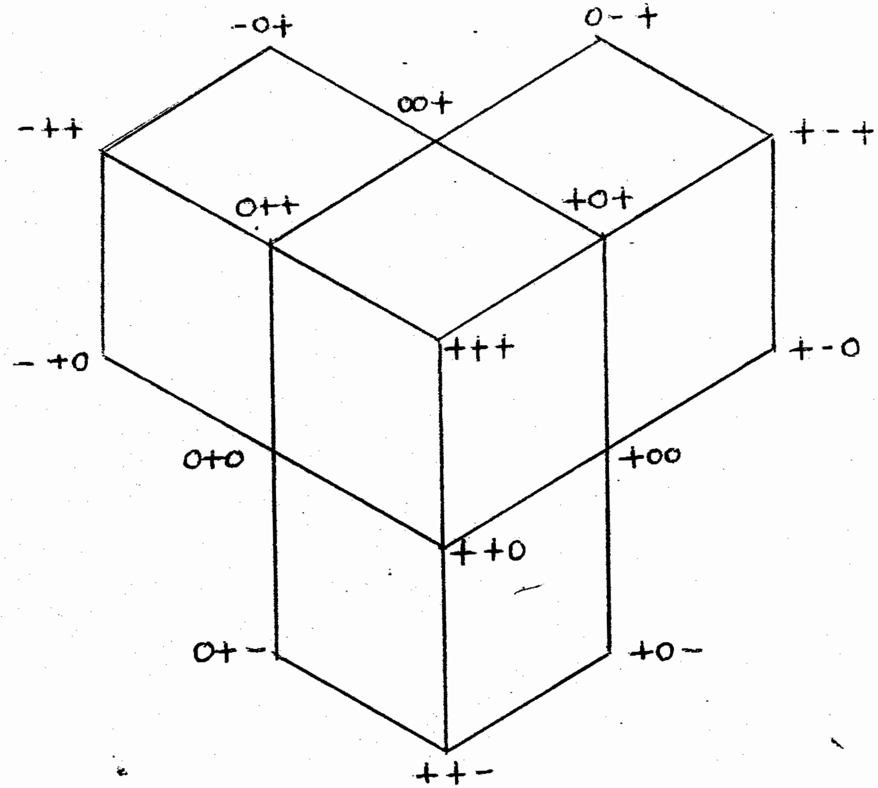


FIGURE 4

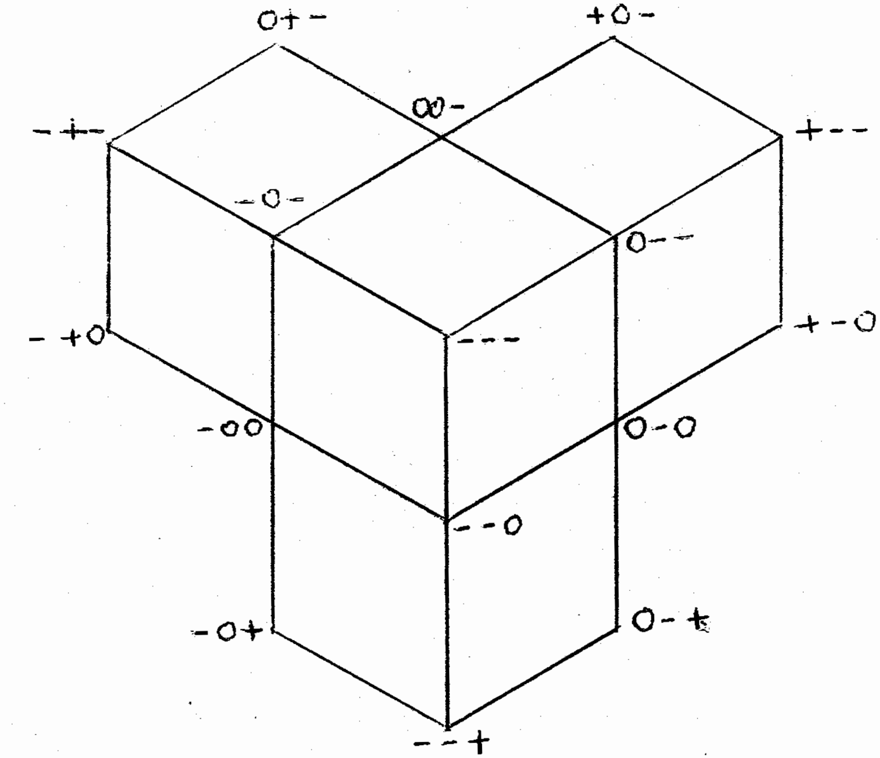
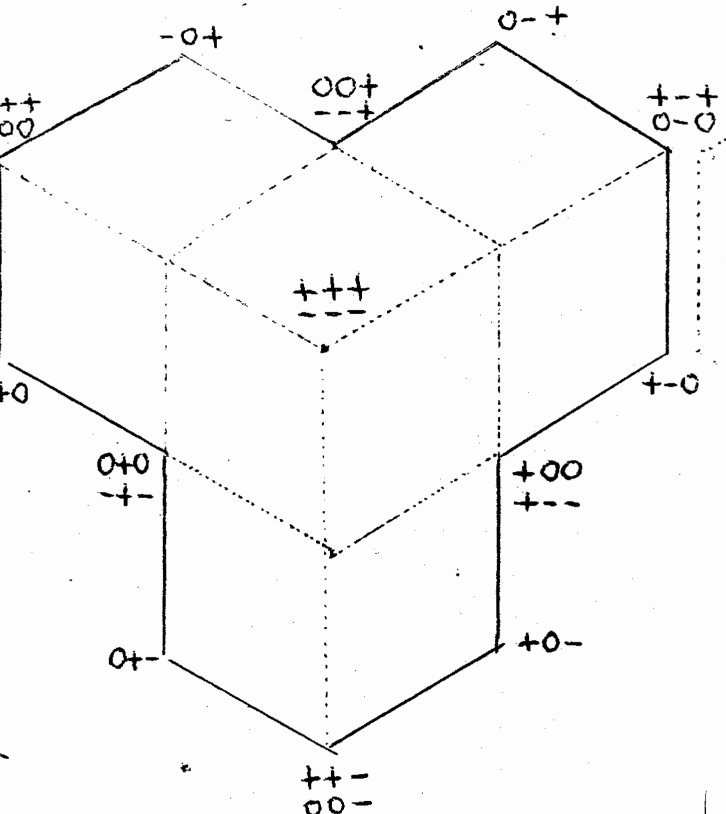
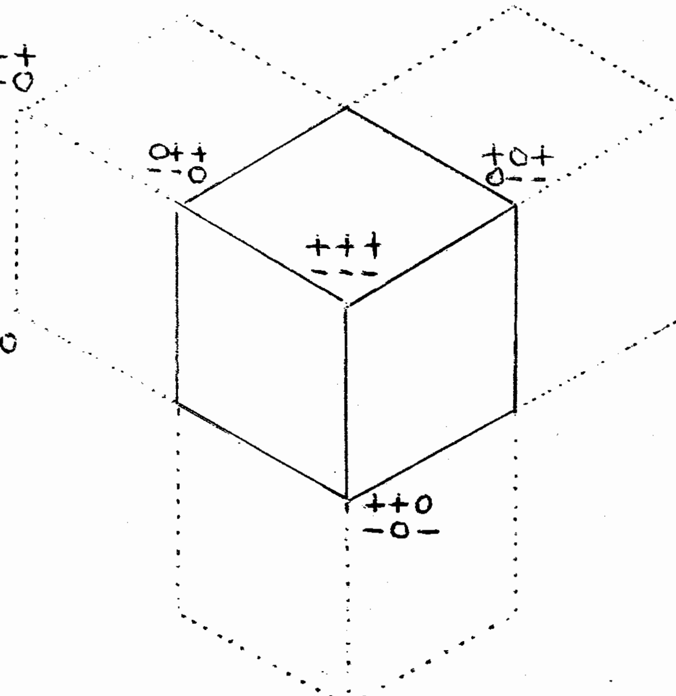


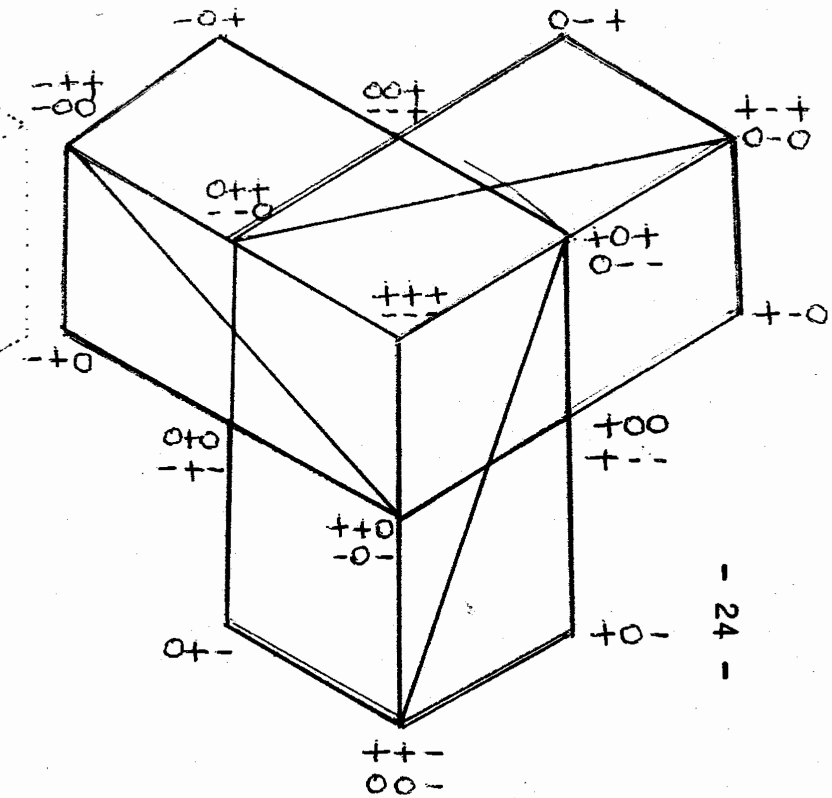
FIGURE 5



a. Mapping of the 0 and +1,-1 disparity ternary words (and +3,-3 also).



b. Mapping of the +2,-2 disparity ternary words



c. Final mapping
Green lines belong to positive words
Red lines belong to negative words

CHAPTER III

The Program

1. General

Though a certain mapping of the negative onto the positive grid was considered the number of different codes is still large (16! not necessarily not equivalent) and special effort was made as to the minimisation of time taken by the program.

2. Outline of the program

The basic task of the program was to compare two binary 4-tuples and find the number of corresponding places in which they differ -their Hamming distance. This operation should be performed 24 times for each code. This is because the figure of the mapping has 27 lines. Since the four centre elements are fixed, this leaves 24 lines. Therefore the need for a fast subroutine is obvious. This subroutine was decided to be written in PLAN (assembly language for ICL). The rest of the program is relatively simple in construction and consists of 12 DO-loops.

3. Data

With reference to fig. 7, the binary numbers for the vertices 1, 2, 3, 4 were given in as data since a certain symmetry was noticed around the 2, 3, 4 vertices.* (It does not matter which number we put for 1, as we can always transform it to any number).

./..

The different sets of data were determined with the following criteria :

(i) The Hamming-distance of words 2, 3, 4 from word 1.

(ii) The Hamming-distance between words 2-3, 3-4, 4-2.

This was considered so as transformations of codes to equivalent ones does not change the Hamming-distance of the corresponding words. Also any cyclic permutation of the data words does not alter the obtainable code as our grid is cyclic symmetric around the origin.

We let d_{ij} be the Hamming-distance of words i and j and W_i the number of 1's in the word i(the weight of word i).
The different sets of data are given in fig. 9. Referring to it, we have :

<u>1.</u> $d_{1,j} = 1$ $(j = 1, 2, 3)$

This leaves us no choice for $d_{2,3}$,$d_{3,4}$, $d_{4,2}$, since $d_{i,j}$=w(word i+wordj) =number of 1's in sum of i,j =2

<u>2. 3. 4.</u>

$d_{1,2} = 2$
$d_{1,3} = 1$
$d_{1,4} = 1$

$d_{3,4} = 2$, since they are both of weight 1.
$d_{2,3} = 1$ or 3
$d_{4,2} = 1$ or 3

This is because all the binary code words of weight 2 and 4 (and 0000) form a group (a subgroup of the group of the binary 4-tuples) and in this subgroup, call it F, the weight of the sum of any two codewords is 2 or 4. Therefore, if $d_{2,3}$, $d_{4,2}$ were

2 or 4, it would mean that both 2 and 3, and 4 and 2 (referring to the number of vertex) would belong to the subgroup F. But this is not so.

5. 6. 7. 9.

$d_{1,2} = 2$
$d_{1,3} = 1$
$d_{1,4} = 1$

Since 2 and 4 are of weight 2, they belong to the already mentioned subgroup F and therefore :

$d_{4,2} \neq 2$ or 4

If $d_{4,2} = 2$, then $d_{2,3} = 1$ or 3, $d_{3,4} = 1$ or 3 (from previous result).

If $d_{4,2} = 4$, then there is only one possibility and that is $d_{2,3} = 1$, $d_{3,4} = 3$ or v.v.

8.10.

$d_{1,j} = 2$
$(j = 2, 3, 4)$

2, 3, 4 will belong to the subgroup F and therefore their relative Hamming distances will be 2 or 4. It is not possible, however, to have two words having distance 4, since for each codeword i in F, of weight 2, there is only one other codeword j of weight 2, such that, weight (i+j) = 4.

The above 10 cases of input data, were considered to be the only ones necessary for the solution of our problem. All other combinations are essentially equivalent to one of the above.

4. Calculation of probability of error and of word corresponding to 000

Since a constant factor of 64 appears in the expression for the mean probability of error in a recovered binary word, ((2), page 7), this was not included in the expression for the calculation of the error in the program.

Since we have a fixed grid we can associate a number, a weight factor, for each edge according to the respective probability weights of positive and negative ternary words. According to the definitions (i), (ii), (iii) in page 6, the weights of the edges are as shown in fig. 8.
The formula for the total error sum is :

$$\sum_{\text{all edges}} (p_i \times d_i) + (\text{error due to choice of } 000)$$

where, p_i = probability weight of edge i

d_i = Hamming distance of binary words at ends of edge i

The word 000 is never transmitted of course but can occur as a result of error due to noise e.t.c. The best binary choice for 000 is calculated after a complete code has been found as follows:
Let ternary words neighbour to 000 be t_i, and the corresponding (to t_i) binary words b_i. Then a binary word E is chosen from the binary group, which minimises :

$$\sum_{j=1,6} d_{E,j}$$

where d_{Ej} = Hamming distance of E and b_j.
It is possible that E is not unique. In this case, both codes, which only differ in E, are printed by the programme.
Only these codes are considered, for which the error sum is less than 85.

./..

5. Programming details

5.1 Subroutine SYG

The subroutine SYG (from SYGrino, greek for compare) is written in PLAN, as this would make it substantially faster than a FORTRAN written subroutine. This required the use of a macro instruction which could perform the mixed language programming (main program written in FORTRAN) and such a macro instruction was fortunately available (called LFORTOPT).
The only problem here was the transfer of the arguments of the subroutine from and to the calling program. This could be achieved either by using two system routines or by locating the core address of the argument and then retrieving their values. The first method was considered safer and just as quick as the second one and so the two routines FPROLOG and FEPILOG were employed.
The rest of the subroutine was quite staight forward.
The execution time of the subroutine was approximately 10^{-6}x5 seconds.

The call to the subroutinr is :

CALL SYG (KUBE (I), KPERM (J), CO (K))

The input arguments are:
KUBE (I) : neighbour binary word, already obtained, of vertex I;
KPERM (J) : current trial binary word for vertex J;
The output argument is :
CO(K) : is the Hamming distance of words I and J.

The subroutine was tested prior to inclusion in the main program and both the listing and test sample are shown in Appendix 1.

5.2 Programming details of main program

The main program is written in FORTRAN and is making use of the subroutine SYG.

The input to the program is in the form of cards. The first four binary words which are allocated to vertices 1, 2, 3, 4 are read in together with the remaining 12 binary words which are put into the array KPERM (which holds the unused binary words). The error sum due to the first four words is also read in.

The program does an exhaustive search of all possible ways of allocating the 12 binary words onto the 12 remaining vertices (No.5-16) in such a way as not to infringe the restrictions.

This search is done by means of 12 do-loops which are nested like shown in the diagram of fig. 9a.

If a word is found that fits a vertex it is put into the array KUBE in the position specified by the number of the vertex. That word is then taken out from the array KPERM by means of moving the words after it one place up. For example, if the arrays KUBE and KPERM are :

pos.	KUBE(16)	KPERM(12)
1	1010	0110
2	1110	0011
3	0000	0101
4	1011	1100
5	1000	1001
6		1111
7		1101
8	a	0111
9	n	0010
10	y	0001
11	t	0100
12	h	1000
13	i	
14	n	
15	g	
16		

./..

before the allocation of a binary word to vertex 6, then they look like

KUBE(16)

pos.	
1	1010
2	1110
3	0000
4	1011
5	1000
6	0001
7	
8	a
9	n
10	y
11	t
12	h
13	i
14	n
15	g
16	

KPERM(12)

0110
0011
0101
1100
1001
1111
1101
0111
0010
0100
1000
1000

after the allocation.

If no word from KPERM can be found to fit a certain vertex I, the program goes back to vertex I-1 and finds another word which will fit it and then continues with this new word.
When all 16 vertices have been allocated, the word corresponding to 000 is found by the method mentioned in III.4. The total error sum is then found by adding the already calculated error sum, due to the obtained codewords (calculated accumulatively), and the error sum due to the choice of codeword corresponding to 000. If the error sum does not exceed 85, the array KUBE together with the binary word corresponding to 000 and the error sum are printed. Otherwise, the program goes on to find another code. When all the different combinations of a given set of data are considered, the next card which contains a new set of words for vertices 1, 2, 3, 4 is read in. There are ten such cards followed by a terminator, which prints the message : "PROGRAM FINISHED".
The total execution time was 993 seconds.

./..

This seems as a rather long time despite the fact that considerable effort was made in order to minimise it. One way of doing this was to try and minimise the number of times the subroutine SYG was called. This could achieved by not allowing the program to complete a set of codewords for which, the already obtained accumulated error sum made it certain that whatever the choice for the remaining codewords, the error sum limit of 85 would be exceeded. A test was therefore inserted after do-loop 8 (since whatever the choice for the first 8 words, it is not impossible not to exceed 85) for the accumulated error sum and if this was greater than a value (shown in fig. 9a) calculated according to the number of codewords found, the program was affected in the same way as if the codeword found at this stage, did not fit the corresponding vertex.
This test was found to reduce the execution time by a considerable amount.

The listing of the program and the results are in Appendix 2. A program by Mr. Markides [3] was used to draw a detailed flowchart in the line printer of the computer, and this is included in this paper separately.

All test runs and final run were done at the LPCU, in the Polytechnic of North London. The computer was an ICL 1900 series.

6. Results

From the computer program output, it is easily seen that the lowest error sum is 79, which makes the mean probability of error in a recovered binary word

$$79p/64 = 1.234\ p$$

Most results (in fact all but two) were obtained with the first set of data, that is with starting values for vertices 1, 2, 3, 4 1010, 1110, 0000, 1011 respectively.
There were all together 12 codes eith error sum 79 (all obtained with above set of input data) half of which only differed in the representation of the error word 000. We are left, therefore, with only 6 "different" codes, which are thought to be equivalent, without having been able to establish the operations which would transform each code into another (referring to the output listing in Appendix 2, we can clearly see that códe (6) can be produced deom (1), if we interchange the first and third digits).

Code (1) is shown on the final mapping in fig.10. Tables 8, 9, 10 also show code (1) and its probability weights and error weights matrices.

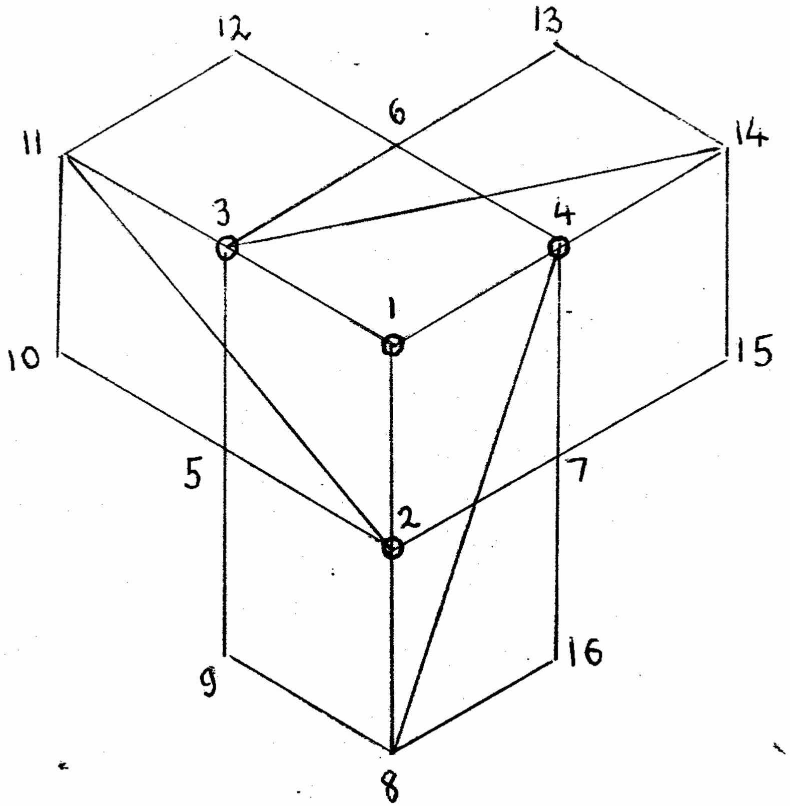


FIGURE 7

Final mapping . numbering of vertices

o fixed vertices

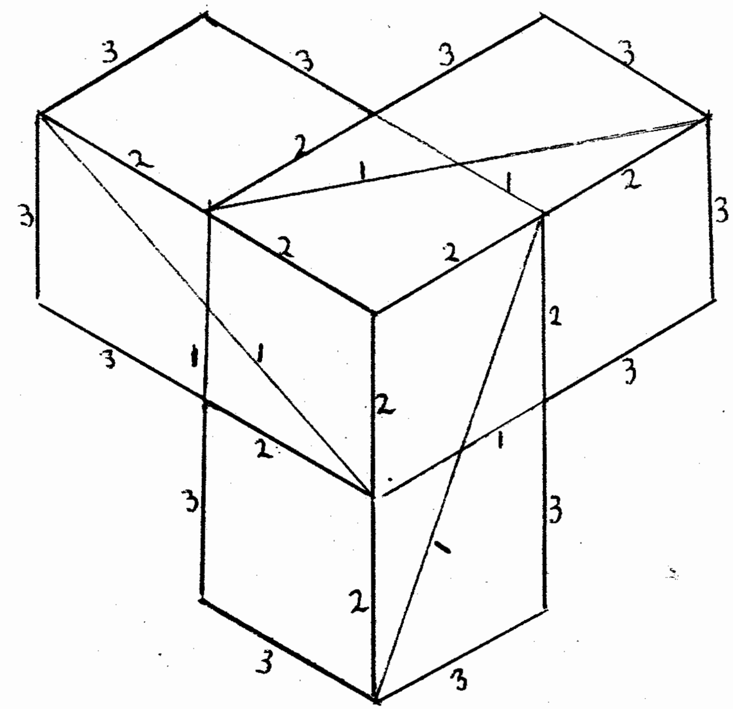
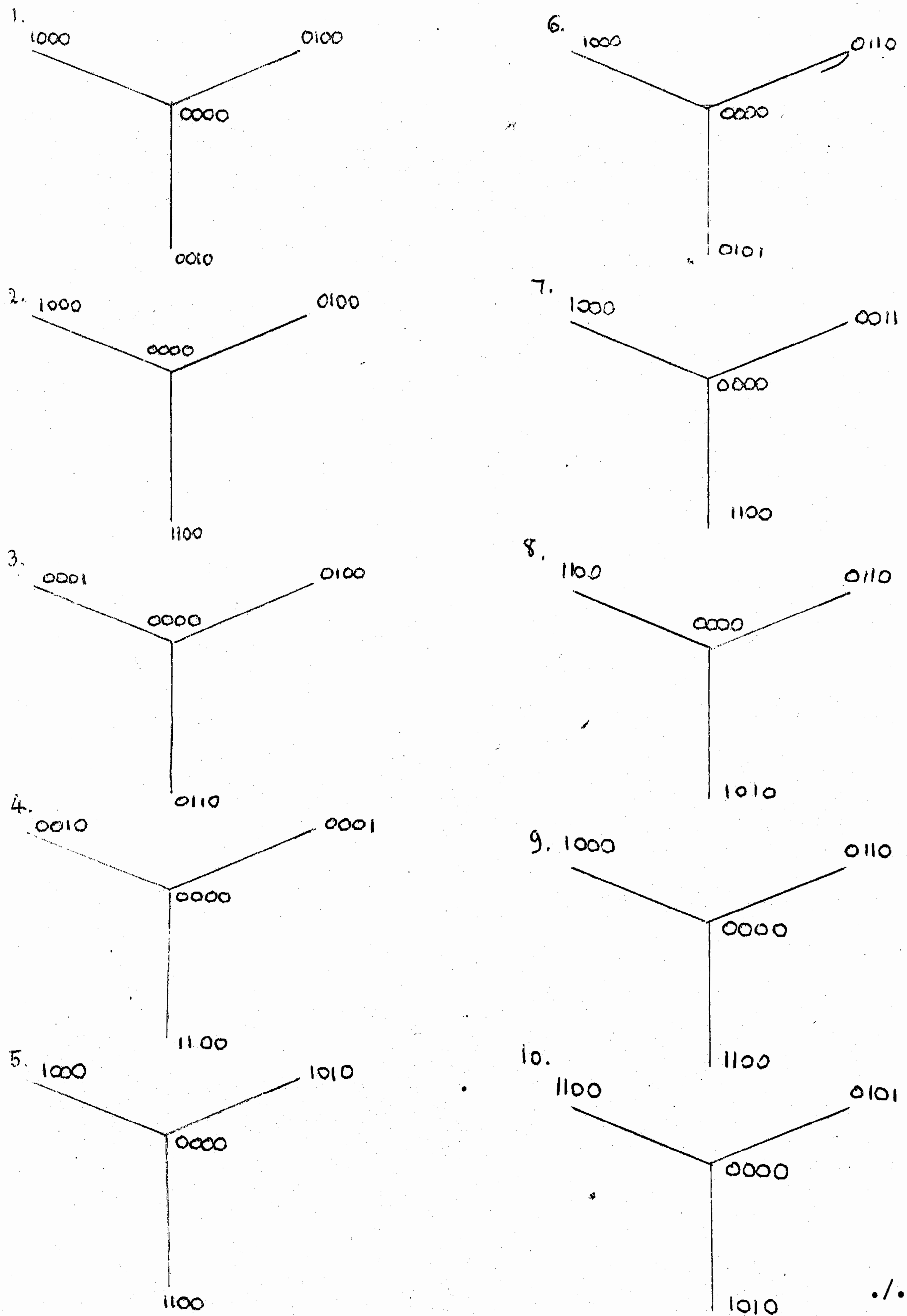


FIGURE 8

Weightings of edges

FIGURE 9

Sets of data



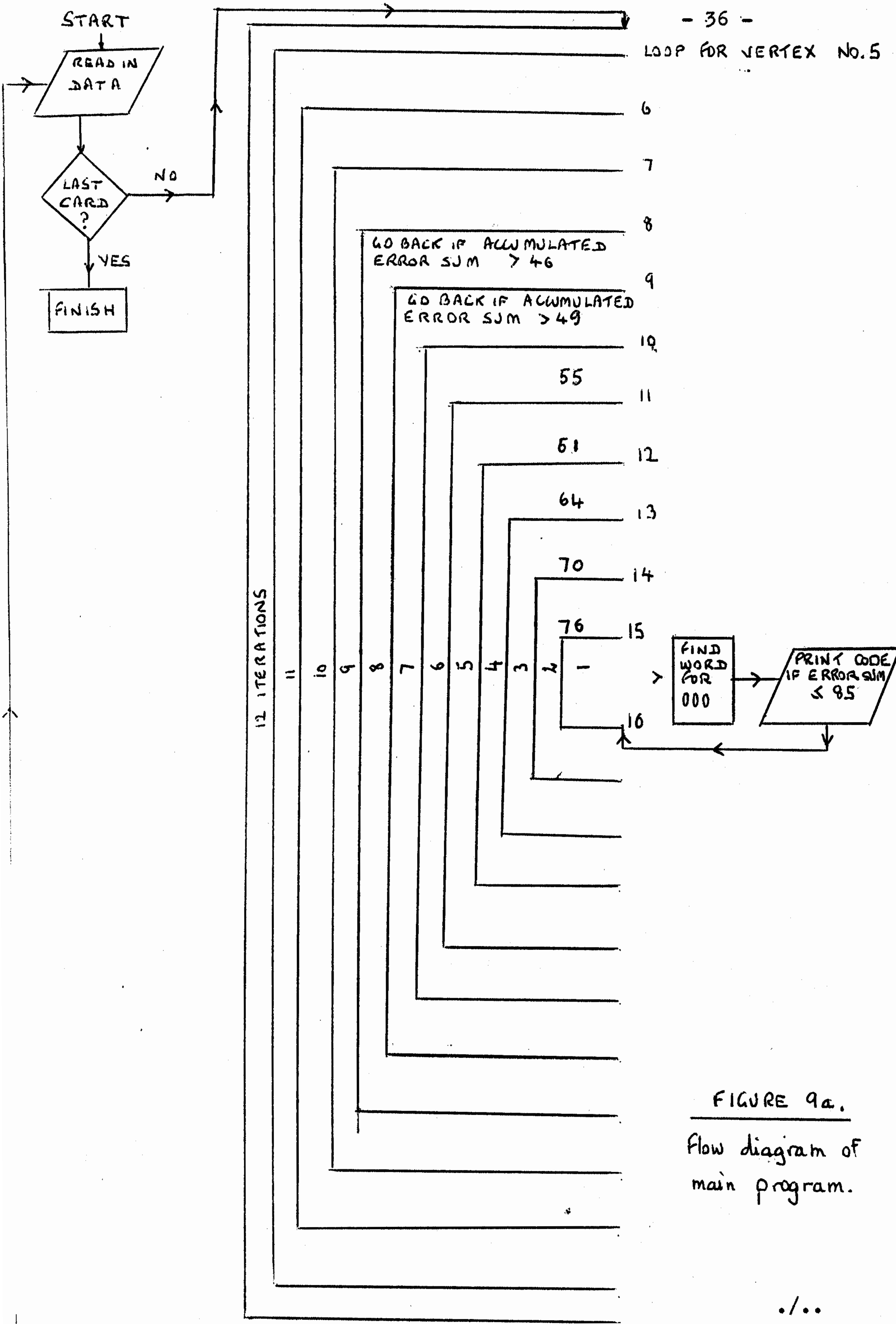


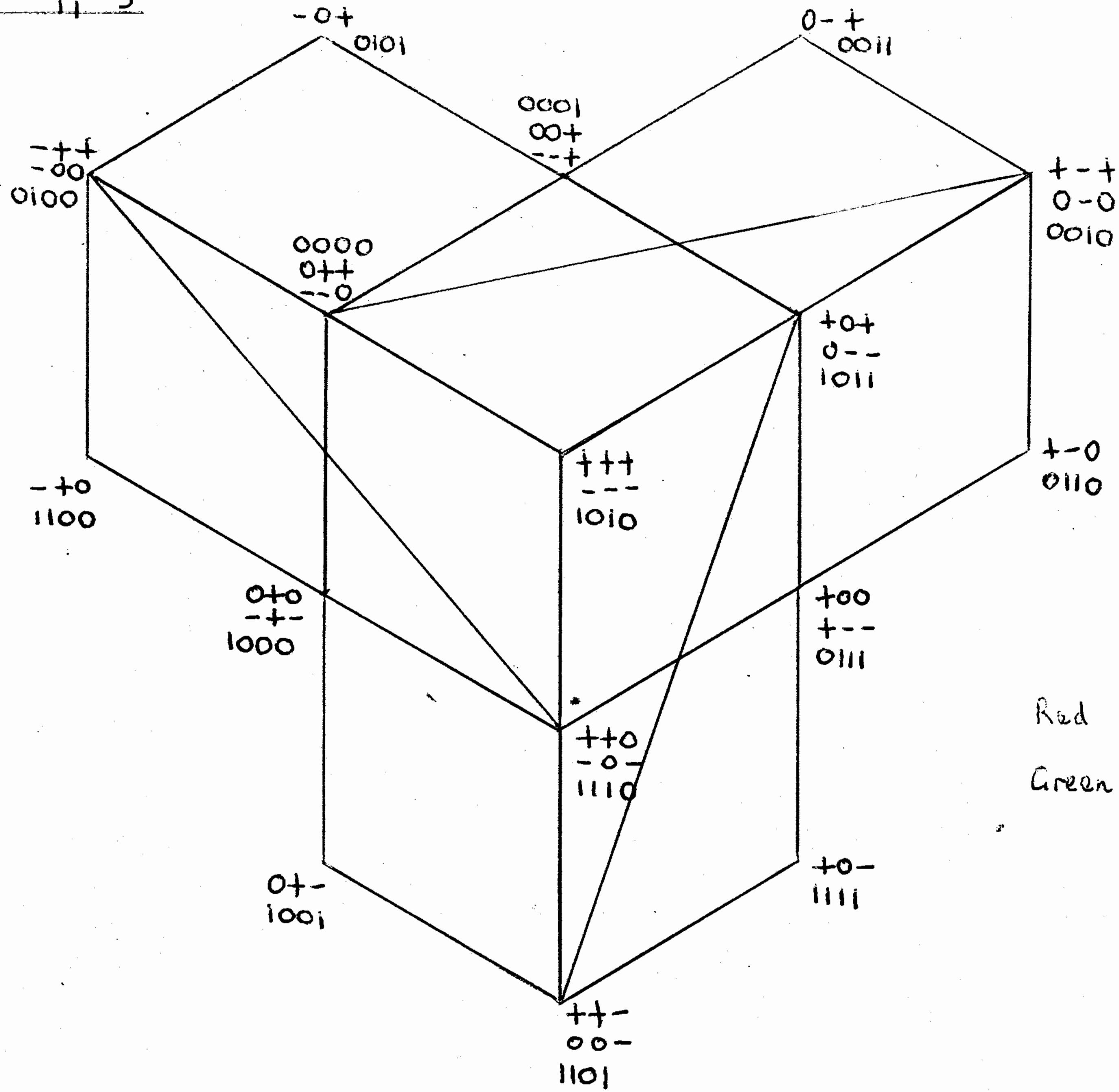
FIGURE 9a.

Flow diagram of main program.

FIGURE 10

Final solution mapping

(Code type IVa)



000 → 0000

Red lines connect neighbouring words

Green lines connect words differing in two places

Table 8

4B3T Code Type IV

Ternary word				Binary word	
Disparity	Designation	Code word in		a	b
		Positive mode	Negative mode		
0	A1	+ - 0		0 1 1 0	1 1 0 0
	A2	0 - +		0 0 1 1	1 0 0 1
	A3	- 0 +		0 1 0 1	0 1 0 1
	A4	- + 0		1 1 0 0	0 1 1 0
	A5	0 + -		1 0 0 1	0 0 1 1
	A6	+ 0 -		1 1 1 1	1 1 1 1
±1	B1	+ + -	0 0 -	1 1 0 1	0 1 1 1
	B2	+ 0 0	+ - -	0 1 1 1	1 1 0 1
	B3	+ - +	0 - 0	0 0 1 0	1 0 0 0
	B4	0 0 +	- - +	0 0 0 1	0 0 0 1
	B5	- + +	- 0 0	0 1 0 0	0 1 0 0
	B6	0 + 0	- + -	1 0 0 0	0 0 1 0
±2	C1	+ + 0	- 0 -	1 1 1 0	1 1 1 0
	C2	+ 0 +	0 - -	1 0 1 1	1 0 1 1
	C3	0 + +	- - 0	0 0 0 0	0 0 0 0
±3	D1	+ + +	- - -	1 0 1 0	1 0 1 0
Detected error	E	0 0 0		0 0 0 1	0 1 0 0

./..

CODE IV AL

PROBABILITY WEIGHS
TERNARY NEIGHBOURS

	A1	A2	A3	A4	A5	A6	B1	B2	B3	B4	B5	B6	C1	C2	C3	D1	E
A1								2	2								
A2									2	2							
A3										2	2						
A4											2	2					
A5							2					2					
A6							2	2									
B1					1	1							1	1/2			1/2
B2	1					1							1/2	1			1/2
B3	1	1												1	1/2		1/2
B4		1	1											1/2	1		1/2
B5			1	1									1/2		1		1/2
B6				1	1								1		1/2		1/2
C1							1	1/2			1/2	1				1	
C2							1/2	1	1	1/2						1	
C3									1/2	1	1	1/2				1	
D1													1	1	1		

number of neighbours: 60

Probability of error: $\frac{51p}{16}$

Table 9

ERROR WEIGHTS

	A1	A2	A3	A4	A5	A6	B1	B2	B3	B4	B5	B6	C1	C2	C3	D1	E
A1								1	1								
A2									1	1							
A3										1	1						
A4											1	1					
A5							1					1					
A6							1	1									
B1					1	1							2	2			2
B2	1					1							2	2			2
B3	1	1												2	2		2
B4		1	1											2	1		0
B5			1	1									2		1		2
B6				1	1								2		1		2
C1							2	2			2	2				1	
C2							2	2	2	2						1	
C3									2	1	1	1				2	
D1													1	1	2		

IF E: 0001

mean probability of error: $p \cdot 79/64 = 1.234p$

Table 10

CHAPTER IV

Conclusion

The results of the computer program were found to be very satisfactory. Indeed the lowest error mean found, 1.234, is below the best known results, as presented by Mr. K.W. Cattermole [4] , and the corresponding translation table does not permit the occurence of more than 2 errors.

It is regretted by the author that no sufficient theoretical work was done as for the proof of the statements in II.4. The results obtained there were very intuitive and with no mathematical proof. Unfortunately there is no mathematical theory behind this type of codes -at least not to the knowledge of the author- and the limited mathematical knowledge of the author did not allow him to provide such a theory. The limited period of time also was also a crucial factor, since the time necessary to equip oneself with the necessary mathematical background for such proofs is considerable.

The overall execution time of the program could be made less if, what was aimed to be found, was not every code with total error sum less than 85, but just the best code that could be found. Therefore, the test which was inserted in the program could be modified, so that it would skip any codes which could not produce a better error sum than the best already obtained.

A way in which this project could be bettered was to try and consider all possible mappings (4320) of the negative words onto the positive ones, and for each one obtain a best code. Of course not all of the 4,320 mappings are different, because of symmetry etc., and the different ones should be considered. In a program suitable for this task, considerable effort should be made in inserting checks which would stop the calculation for a certain mapping if this was sure not to produce a better result than obtained previously (a similar check as mentioned above). Such a program should be quite flexible in the consideration of other codes, different in structure from the one considered in this paper, in that not many changes would be required for the new problem. Unfortunately such a virtue does not exist in the present program.

It is not thought however, that an improvement in the results obtained in this paper can occur by a different mapping of the positive and negative ternary words. The final mapping considered in this paper has also a nice symmetry which intuitively suggest to be the best.

APPENDIX 1

SUBROUTINE SYG

LISTING AND TEST SAMPLE

```
JOB POULIEZOS,H11E415,PR1                    RUN BY GEORGE 2/MK9E ON 09/12/74 AT 15.02
LFORPLAN BIHA
1   9LF3 LCOMPEX BIHA, , , , ,9
****
```

```
PARAMETERS
DCTE(PROGRAM DUMP)
DEND
    PROGRAM DUMP *000013 USED AS     2
```

```
#STEER          LIST,OBJECT
15/03/05         0?/12/74       COMPILED BY XPLF 28

#PROGRAM            /SYG            [SUB TO COMPARE CORR. DIGITS OF 2 NOS
#LOWER
                 LINK(4)
#PROGRAM
        CALL  3  FPROLOG            [SUB TO STORE ARGUMENTS AND LINK ACC.        0  070 3          0 BA
                 8HSYG                                                           1  #63714720
                                                                                 2  #20202020
                 3                                                               3  #00000003
                 /LINK                                                           4  #00000000        LV
        LDX   1  LINK+1                                                          5  000 1 0        1 LV
        ANDX  1  '#00077777'                                                     6  020 1 0        0 LT
        ORX   1  '#00400000'        [KOUNT=4                                     7  021 1 0        1 LT
        LDX   2  LINK+2                                                          8  000 2 0        2 LV
        ANDX  2  '#00077777'                                                     9  020 2 0        0 LT
        ORX   2  '#00400000'                                                    10  021 2 0        1 LT
        LDN   7  0                                                              11  100 7 0        0
TAG     LDCH  5  0(1)                                                           12  024 5 1        0
        LDCH  4  0(2)                                                           13  024 4 2        0
                                                                                14  026 5 0        4
                                                                                15  074 6         17 PR
        BXE   5  4,*+2              [COMPARE DIGITS
        ADN   7  1                                                              16  101 7 0        1
        BCHX  1  *+1                                                            17  064 1         18 PR
        BCHX  2  TAG                                                            18  064 2         12 PR
        LDX   3  LINK+3                                                         19  000 3 0        3 LV
        STO   7  0(3)                                                           20  010 7 3        0
        CALL  3  FEPILOG                                                        21  070 3          0 BA
                 /LINK                                                          22  #00000000        LV
#END
```

```
*      0 AB
----------

*      4 LV
----------
       0 LINK

*      0 R2:LV
-------------

*      0 LP
----------

*      2 LT

----------

                                                        0   #00077777
                                                        1   #00400000

*      0 R2:LP
-------------

*     23 PR
----------
      12 TAG

*      0 UP
----------
```

```
*     0 R2;UP
------------

*     0 UV
---------

*     0 R2;UV
------------

*     0 BA
---------
      0 FPROLOG
      1 FEPILOG
```

```
#FINISH

PLOGRAM ERRORS  NONE
```

TEST SAMPLE

FORTRAN COMPILATION BY #XFAT HK 4C   DATE 09/12/74 TIME 15/03/38

```
LIST
WORK (ED)
LIBRARY (ED,SUBGROUPSRF7)
LIBRARY (ED,SUBGROUPS-RS)
PROGRAM (BILA)
INPUT 1,5 = CR0
OUTPUT 2,6=LP0
COMPACT DATA
COMPRESS INTEGER AND LOGICAL
TRACE 2
END
```

```
          MASTER
C
C         THIS IS A TEST CALLING PROGRAM FOR THE SUB. SYG
C         SUB SYG COMPARES(IN PLAN) TWO 4-DIGIT INTEGERS
C          AND FINDS THE NUMBER OF PLACES IN WHIDH THEY DIFFER
C
          DATA KLM/'####'/
          WRITE(2,103)
      103 FORMAT(1H1,//,20X,'TEST PROGRAM FOR 4B3T',//,10X,'NUMBERS COMPARED
        1',5X,'DIFF. POSITIONS')
        1 READ(1,100)I,J
          IF(KLM.EQ.I)STOP
          CALL SYG(I,J,KOUNT)
          WRITE(2,101)I,J,KOUNT
      101 FORMAT(12X,2(A4,2X),9X,I5)
      100 FORMAT(2A4)
          GO TO 1
          STOP
          END

END OF SEGMENT, LENGTH    57, NAME  NONM
```

```
 0031                     READ FROM(ED,PROGRAM DUMP)
SUBFILE PROGRAM XXXX
SYG             (SEMI-COMPILED)

 END OF SUBFILE
 0001                     FINISH

END OF COMPILATION -  NO ERRORS

S/C SUBFILE:                 7 BUCKETS USED


CONSOLIDATED BY XPCK 12C    DATE      09/12/74   TIME 15/04/14

ICLFORTRAN  (1)     RENAMED  ICLA-DEFAULT(1)

PROGRAM BINA
COMPACT DATA (15AM)
COMPACT PROGRAM (DBM)
CORE                 3643

                    SEG      NONM
                    SEG      SYG
                    SEG      FPROLOG
                    ENT      FEPILOG
```

TEST PROGRAM FOR 4B3T

NUMBERS COMPARED		DIFF. POSITIONS
1110	0010	2
1010	0101	4
1111	1010	2
1110	1110	0
1101	1100	1
1111	1100	2
1000	1111	3

```
15/02/58  0#XPLF   #LPCU
15/03/11  0#XPLF   DISP   COMP OK 0000030#XXXX
15/03/11  0#XPLF   DLTD
15/03/34  0#XFAT   #XPLF
15/04/08  0#XFAT   DLTD   FI #XPCK   #COMP
15/04/08  0#XFAT   CLKD      1
15/04/08  0#XFAT   DLTD
15/04/13  0#XPCK   #XFAT
15/04/34  0#BINA   #XPCK
15/05/03  0#BINA   3648
15/05/03  0#BINA   HALT   LD
15/05/07  0#BINA   DLTD   00
15/05/07  0#BINA   CLKD      4
15/05/07  0#BINA   DLTD

   ACCOUNT CODE      N11L415      DATE         09/12/74      TOTAL MILL TIME     11
   USER NAME         POULIEZOS    START TIME   15/02/30      INPUT RECORDS       61
   JOB NAME          PP1          END TIME     15/05/09      OUTPUT RECORDS     198
   PERIPHERALS USED;                                         MAX. CORE SIZE   18176
--------------------------------------------------------------------------------------
```

APPENDIX 2

MAIN PROGRAM : LISTING AND RESULTS

```
JOB POULIEZOS,M1TA11,PRZ                    RUN BY GEORGE 2/MK9E ON 22/03/75 AT 06.31
LFORTUPT 3IRA,1500,1500, ,R
****
9IIR
```

FORTRAN COMPILATION BY WXFEW MK 4   DATE 22/03/75 TIME 06/51/46

```
LIST
LIBRARY (ED,SUBGROUP-FSCE)
PROGRAM(BIN,)
INPUT 1,5=CR0
OUTPUT 2,6=LP0
COMPACT DATA
COMPRESS INTEGER AND LOGICAL
TRACE 1
END
```

```
      MASTER 14B3T
C
C     PROGRAM TO CALCULATE BEST TERNARY CODE FOR  SPACE OF 16 BINARY
C     4-TUPLES
C
C     ARRAY CUBE HOLDS OBTAINED VERTICES
C     ARRAY KPERM HOLDS UNUSED BINARY 4-TUPLES
C     KOUN1,KOUN2,KOUN3 ARE THE NO OF DIFF. POSITIONS
C     ARRAY CO HOLDS ERROR SUM FOR CORR.VERTEX
C      --LTEST HOLDS ERROR SUM
C
      INTEGER CO
      DIMENSION KUBE(16),KPERM(12),CO(24),KUB(6)
      DATA KLU/'####'/
C
C
      WRITE(2,400)
  400 FORMAT(1H1,10X,'PROGRAM TO CALCULATE BEST ENCODING SCHEME FOR THE
     .16 BINARY 4-TUPLES',//,10X,'USING A CERTAIN MAPPING OF THE POSITIV
     .E  TERNARY CODEWORDS ONTO THE NEGATIVE ONES')
   58 READ(1,200)(KUBE(I),I=1,4),(KPERM(J),J=1,12),LTEST
  200 FORMAT(16A4,I3)
      IF(KUBE(1).EQ.KLU)GO TO 999
      XKEEP=LTEST
      WRITE(2,401) (KUBE(I),I=1,4)
  401 FORMAT(1H ,///////,5X,'STARTING VALUES FOR VERTICES 1,2,3,4 ARE RE
     :SP. ',5X,4(3X,A4),/,3X,'******************************************
     :*********')
      WRITE(2,404)
  404 FORMAT(1H ,//,2X,34('-'),'I',35('-'),'I',17('-'),'I',5('-'),'I',6(
     '-'),'I',/,
     :'7X,'0',18X,'I',15X,'+1,-1',15X,'I',6X,'+2,-2',6X,'I+3,-3IERROR I'
     :,/,2X,'----I',15('-----I'),'------I',/,
     :3X,'A1 I  A2 I  A3 I  A4 I  A5 I  A6 I  B1 I  B2 I  B3 I  B4 I  B5
     : I  B6 I  C1 I  C2 I  C3 I  D1 I  E   I',/,
     :2X,'----I',15('-----I'),'------I',/,
     :6X,'I',5(5X,'I'),1X,'++- I +00 I +-+ I 00+ I -++ I 0+0 I ++0 + +0+
     : I 0++ I +++ I       I',/,
     :2X,'+-0 I 0-+ I -0+ I -+0 I 0+- I +0- I 00- I +-- I 0-0 I --+ I -0
     :0 I -+- I -0- I --0 I --- I 000  I',/,
     :2X,'----I',15('-----I'),'------I')
C
C     OBTAINING POINT (VERTEX) NO 5
C
      DO 5 I5=1,12
      CALL SYG(KUBE(2),KPERM(I5),CO(1))
      IF(CO(1)-2)51,51,5
   51 CALL SYG(KUBE(3),KPERM(I5),CO(2))
      IF(CO(2)-2)52,52,5
```

```
      LTEST=LTEST+CO(3)+CO(1)+CO(2)
      IF(I5-12)53,54,54
   53 DO 55 I55=I5,11
   55 KPERM(I55)=KPERM(I55+1)
C
C     VERTEX NO 6
C
   54 DO 6 I6=1,11
      CALL SYG(KUBE(3),KPERM(I6),CO(3))
      IF(CO(3)-2)61,61,6
   61 CALL SYG(KUBE(4),KPERM(I6),CO(4))
      IF(CO(4)-2)62,62,6
   62 KUBE(6)=KPERM(I6)
      LTEST=LTEST+CO(3)+CO(3)+CO(4)
      IF(I6-11)63,64,64
   63 DO 65 I65=I6,10
   65 KPERM(I65)=KPERM(I65+1)
C
C     VERTEX NO 7
C
   64 DO 7 I7=1,10
      CALL SYG(KUBE(4),KPERM(I7),CO(5))
      IF(CO(5)-2)71,71,7
   71 CALL SYG(KUBE(2),KPERM(I7),CO(6))
      IF(CO(6)-2)72,72,7
   72 KUBE(7)=KPERM(I7)
      LTEST=LTEST+CO(5)+CO(5)+CO(6)
      IF(I7-10)73,74,74
   73 DO 75 I75=I7,9
   75 KPERM(I75)=KPERM(I75+1)
C
C     VERTEX NO 8
C
   74 DO 8 I8=1,9
      CALL SYG(KUBE(2),KPERM(I8),CO(7))
      IF(CO(7)-2)81,81,8
   81 CALL SYG(KUBE(4),KPERM(I8),CO(8))
      IF(CO(8)-2)82,82,8
   82 KUBE(8)=KPERM(I8)
      LTEST=LTEST+CO(7)+CO(7)+CO(8)
      IF(I8-9)83,84,84
   83 DO 85 I85=I8,8
   85 KPERM(I85)=KPERM(I85+1)
C
C     VERTEX NO 9
C
   84 DO 9 I9=1,8
      CALL SYG(KUBE(5),KPERM(I9),CO(9))
      IF(CO(9)-2)91,91,9
   91 CALL SYG(KUBE(8),KPERM(I9),CO(10))
      IF(CO(10)-2)92,92,9
```

```
   92 LTEST=LTEST+(CO(9)+CO(10))*3
      IF(LTEST-46)307,307,98
  307 KUBE(9)=KPERM(I9)
      IF(I9-8)93,94,94
   93 DO 95 I95=I9,7
   95 KPERM(I95)=KPERM(I95+1)
C
C     VERTEX NO 10
C
   94 DO 10 I10=1,7
      CALL SYG(KUBE(5),KPERM(I10),CO(11))
      IF(CO(11)-2)101,101,10
  101 LTEST=LTEST+CO(11)*3
      IF(LTEST-49)308,308,108
  308 KUBE(10)=KPERM(I10)
      IF(I10-7)103,104,104
  103 DO 105 I105=I10,6
  105 KPERM(I105)=KPERM(I105+1)
C
C     VERTEX NO 11
C
  104 DO 11 I11=1,6
      CALL SYG(KUBE(3),KPERM(I11),CO(12))
      IF(CO(12)-2)111,111,11
  111 CALL SYG(KUBE(2),KPERM(I11),CO(13))
      IF(CO(13)-2)112,112,11
  112 CALL SYG(KUBE(10),KPERM(I11),CO(14))
      IF(CO(14)-2)116,116,11
  116 LTEST=LTEST+CO(12)+CO(12)+CO(13)+CO(14)*3
      IF(LTEST-55)309,309,118
  309 KUBE(11)=KPERM(I11)
      IF(I11-6)113,114,114
  113 DO 115 I115=I11,5
  115 KPERM(I115)=KPERM(I115+1)
C
C     VERTEX NO 12
C
  114 DO 12 I12=1,5
      CALL SYG(KUBE(6),KPERM(I12),CO(15))
      IF(CO(15)-2)121,121,12
  121 CALL SYG(KUBE(11),KPERM(I12),CO(16))
      IF(CO(16)-2)122,122,12
  122 LTEST=LTEST+(CO(15)+CO(16))*3
      IF(LTEST-61)310,310,128
  310 KUBE(12)=KPERM(I12)
      IF(I12-5)123,124,124
  123 DO 125 I125=I12,4
  125 KPERM(I125)=KPERM(I125+1)
C
C     VERTEX NO 13
```

```
  124 DO 13 I13=1,4
      CALL SYG(KUBE(6),KPERM(I13),CO(17))
      IF(CO(17)-2)131,131,13
  131 LTEST=LTEST+CO(17)*3
      IF(LTEST-64,311,311,133
  311 KUBE(13,=KPERM(I13)
      IF(I13-4)133,134,134
  133 DO 135 I135=I13,3
  135 KPERM(I135)=KPERM(I135+1)
C
C     VERTEX NO 14
C
  134 DO 14 I14=  ,3
      CALL SYG(KUBE(13),KPERM(I14),CO(18))
      IF(CO(18)-2,141,141,14
  141 CALL SYG(KUBE(3),KPERM(I14),CO(19))
      IF(CO(19)-2)142,142,14
  142 CALL SYG(KUBE(4),KPERM(I14),CO(20))
      IF(CO(20)-2,146,146,14
  146 LTEST=LTEST+CO(18)+3+CO(19)+CO(20)+CO(20)
      IF(LTEST-70)313,313,148
  313 KUBE(14)=KPERM(I14)
      IF(I14-3)143,144,144
  143 DO 145 I145=I14,2
  145 KPERM(I145)=KPERM(I145+1)
C
C     VERTEX NO 15
C
  144 DO 15 I15=1,2
      CALL SYG(KUBE(7),KPERM(I15),CO(21))
      IF(CO(21)-2,151,151,15
  151 CALL SYG(KUBE(14),KPERM(I15),CO(22))
      IF(CO(22)-2)152,152,15
  152 LTEST=LTEST+(CO(21)+CO(22))*3
      IF(LTEST-76,314,314,158
  314 KUBE(15,=KPERM(I15)
      IF(I15-2)153,154,154
  153 KPERM(1,=KPERM(2)
C
C     VERTEX NO 16
C
  154 CALL SYG(KUBE(7),KPERM(1),CO(23))
      IF(CO(23)-2,161,161,16
  161 CALL SYG(KUBE(8),KPERM(1),CO(24))
      IF(CO(24)-2,162,162,16
  162 LTEST=LTEST+(CO(23)+CO(24))*3
      KUBE(16)=KPERM(1)
C
C     A SET OF 16 4-TUPLES HAS BEEN FOUND
C
```

```
C     CALCULATION OF CODEWORD CORR. TO TERNARY 000
C
      XTEST=LTEST
      DO 1000 I1000=1,4
 1000 KUB(I1000)=KUBE(I1000+4)
      KUB(5)=KUBE(11)
      KUB(6)=KUBE(14)
      ZEROS=0.
      DO 3 IZ=1,16
      DO 4 IKU=1,6
      CALL SYG(KUBE(IZ),KUB(IKU),KOUNT)
      IF(KOUNT-2)301,301,302
  301 ZEROS=ZEROS-KOUNT*0.5
    4 CONTINUE
C
C     CALCULATION OF ERROR SUM
C     ERROR SUM LESS THAN 85.PRINTOUT OF SUM
C
      IF(XTEST+ZEROS-85.)2,2,302
    2 FTEST=XTEST+ZEROS
      WRITE(2,201)KUBE(15),KUBE(13),KUBE(12),KUBE(10),KUBE(9),KUBE(16),K
     1UBE(8),KUBE(7),KUBE(14),KUBE(6),KUBE(11),KUBE(5),KUBE(2),KUBE(4),K
     2UBE(3),KUBE(1),KUBE(12),FTEST
  201 FORMAT(1H , X,A4,'I',15(1X,A4,'I'),1X,A4,1X,'I','ERROR SUM = ',F5.
     2,/,6X,'I',15(5X,'I'),6X,'I')
      IF(ZEROS-2.501)1005,1005,302
  302 ZEROS=0.
    3 CONTINUE
 1005 LTEST=LTEST-(CO(23)+CO(24))*3
   16 CONTINUE
C
C     IF VALUE CANNOT BE FOUND TO FIT VERTEX I,REMAKE ARRAY KPERM AND
C     CONTINUE ITERATION
C
      IF(I15-2)157,158,158
  157 KPERM(2)=KPERM(1)
      KPERM(1)=KUBE(15)
  158 LTEST=LTEST-(CO(21)+CO(22))*3
   15 CONTINUE
      IF(I14-3)147,148,148
  147 KEEP1=KPERM(I14)
      DO 149 I149=I14,2
      KEEP2=KPERM(I149+1)
      KPERM(I149+1)=KEEP1
  149 KEEP1=KEEP2
      KPERM(I,4)=KUBE(14)
  148 LTEST=LTEST-(CO(18)*3+CO(19)+CO(20)+CO(20))
   14 CONTINUE
      IF(I13-4)137,138,138
  137 KEEP1=KPERM(I13)
```

```
      KLEP2=KPERM(I139+1)
      KPERM(I139+1)=KLEP1
  139 KEEP1=KLEP2
      KPERM(I13)=KUBE(13)
  138 LTEST=LTEST-CO(17)*3
   13 CONTINUE
      IF(I12-5)127,128,128
      IF(I12-5)127,12,12
  127 KLEP1=KPERM(I12)
      DO 129 I129=I12,4
      KLEP2=KPERM(I129+1)
      KPERM(I129+1)=KEEP1
  129 KEEP1=KLEP2
      KPERM(I12)=KUBE(12)
  128 LTEST=LTEST-(CO(15)+CO(16))*3
   12 CONTINUE
      IF(I11-6)117,118,118
  117 KEEP1=KPERM(I11)
      DO 119 I119=I11,5
      KLEP2=KPERM(I119+1)
      KPERM(I119+1)=KLEP1
  119 KEEP1=KEEP2
      KPERM(I11) =KUBE(11)
  118 LTEST=LTEST-(CO(12)+CO(12)+CO(13)+CO(14)*3)
   11 CONTINUE
      IF(I10-7)107,108,108
  107 KEEP1=KPERM(I10)
      DO 109 I109=I10,6
      KEEP2=KPERM(I109+1)
      KPERM(I10)=KUBE(10)
      KPERM(I109+1)=KEEP1
  109 KEEP1=KLEP2
  108 LTEST=LTEST-CO(11)*3
   10 CONTINUE
      IF(I9-8,97,98,98
   97 KEEP1=KPERM(I9)
      DO 99 I99=I9,7
      KEEP2=KPERM(I99+1)
      KPERM(I99+1)=KEEP1
   99 KEEP1=KEEP2
      KPERM(I9)=KUBE(9)
   98 LTEST=LTEST-(CO(9)+CO(10))*3
    9 CONTINUE
      IF(I8-9,87,88,88
   87 KLEP1=KPERM(I8)
      DO 89 I89=I8,8
      KEEP2=KPERM(I89+1)
      KPERM(I89+1)=KEEP1
   89 KLEP1=KLEP2
      KPERM(I8)=KUBE(8)
```

```
    8 CONTINUE
      IF(I7-10)77,78,78
   77 KEEP1=KPERM(I7)
      DO 79 I79=I7,9
      KEEP2=KPERM(I79+1)
      KPERM(I79+1)=KEEP1
   79 KEEP1=KEEP2
      KPERM(I7)=KUBE(7)
   78 LTEST=LTEST-(CO(5)+CO(6)+CO(5))
    7 CONTINUE
      IF(I6-11)67,68,68
   67 KEEP1=KPERM(I6)
      DO 69 I69=I6,10
      KEEP2=KPERM(I69+1)
      KPERM(I69+1)=KEEP1
   69 KEEP1=KEEP2
      KPERM(I6)=KUBE(6)
   68 LTEST=LTEST-(CO(3)+CO(3)+CO(4))
    6 CONTINUE
      IF(I5-12)57,581,581
  57  KEEP1=KPERM(I5)
      DO 59 I59=I5,11
      KEEP2=KPERM(I59+1)
      KPERM(I59+1)=KEEP1
   59 KEEP1=KEEP2
      KPERM(I5)=KUBE(5)
  581 LTEST=LTEST-(CO(1)+CO(1)+CO(2))
    5 CONTINUE
      GO TO 58
  999 WRITE(2,402)
  402 FORMAT(5H ,//,'***PROGRAM FINISHED***')
      STOP
      END
```

END OF SEGMENT, LENGTH 1570, NAME I4A3T

```
SYG          (SEMI-COMPILED)

 0348                    FINISH

END OF COMPILATION -     NO ERRORS

S/C SUBFILE:               30 BUCKETS USED

ICLF-DEFAULT(    0):       60 BUCKETS USED

ICLF-DEFAULT(    1):       50 BUCKETS USED


CONSOLIDATED BY XPCK 2D    DATE     22/03/75   TIME 06/56/28

ICLFORTRAN  (1)     RENAMED  ICLA-DEFAULT( )

PROGRAM BINA
COMPACT DATA (15AM)
COMPACT PROGRAM (8BM)
CORE               448

              SEG      I4B3T
              SEG      SYG
              SEG      FPROLOG
              ENT      FEPILOG
```

PROGRAM TO CALCULATE BEST ENCODING SCHEME FOR THE 16 BINARY 4-TUPLES

USING A CERTAIN MAPPING OF THE POSITIVE TERNARY CODEWORDS ONTO THE NEGATIVE ONES

STARTING VALUES FOR VERTICES 1,2,3,4 ARE RESP. 1010 1110 0000 1011

0						+1,-1						+2,-2			+3,-3	ERROR		
A1	A2	A3	A4	A5	A6	B1	B2	B3	B4	B5	B6	C1	C2	C3	D1	E		
						++-	+00	+-+	00+	-++	0+0	++0	+0+	0++	+++			
+-0	0-+	-0+	-+0	0+-	+0-	00-	+--	0-0	--+	-00	-+-	-0-	0--	--0	---	000		
0110	0011	0101	1100	1001	1111	1101	0111	0010	0011	0100	1000	1110	1011	0000	1010	0001	ERROR SUM = 79.00	(1)
0110	0011	0101	1100	1001	1111	1101	0111	0010	0001	0100	1000	1110	1011	0000	1010	0100	ERROR SUM = 79.00	
0110	0010	0101	1100	1001	1111	1101	0111	0011	0001	0100	1000	1110	1011	0000	1010	0001	ERROR SUM = 83.50	
0101	0001	0110	1100	1001	1111	1101	0111	0011	0010	0100	1000	1110	1011	0000	1010	0001	ERROR SUM = 84.50	
1111	0001	0110	1100	1001	0101	1101	0111	0011	0010	0100	1000	1110	1011	0000	1010	0001	ERROR SUM = 84.50	
0101	0011	0110	1100	1001	1111	1101	0111	0001	0010	0100	1000	1110	1011	0000	1010	0100	ERROR SUM = 79.00	(2)
0101	0011	0110	1100	1001	1111	1101	0111	0001	0010	0100	1000	1110	1011	0000	1010	0001	ERROR SUM = 79.00	
0011	1001	0010	0100	0101	1111	1101	0111	0001	1000	0110	1100	1110	1011	0000	1010	0100	ERROR SUM = 85.00	
0011	1001	0010	0100	1111	0101	1101	0111	0001	1000	0110	1100	1110	1011	0000	1010	0100	ERROR SUM = 85.00	
0110	0011	0101	1000	1001	1111	1101	0111	0010	0011	0100	1100	1110	1011	0000	1010	0100	ERROR SUM = 83.50	
0110	0011	1001	0100	0101	1111	1101	0111	0010	0011	0000	1100	1110	1011	0000	1010	0100	ERROR SUM = 84.50	
0110	0011	1001	0100	1111	0101	1101	0111	0010	0011	0000	1100	1110	1011	0000	1010	0100	ERROR SUM = 84.50	
0101	0001	0010	0100	1000	1111	1101	0111	1001	0011	0110	1100	1110	1011	0000	1010	0101	ERROR SUM = 85.00	
0101	0001	0010	0100	1000	1111	1101	0111	1001	0011	0110	1100	1110	1011	0000	1010	1111	ERROR SUM = 85.00	

1111	0001	0010	0100	1000	0101	1101	0111	1001	0011	0110	1100	1110	1011	0000	1010	0101	ERROR SUM = 85.00	
1000	0001	0010	0100	0101	1111	0111	1101	1001	0011	0110	1100	1110	1011	0001	1010	0101	ERROR SUM = 85.00	
1000	0001	0010	0100	0101	1111	0111	1101	1001	0011	0110	1100	1110	1011	0001	1010	1111	ERROR SUM = 85.00	
1000	0001	0010	0100	1111	0101	0111	1101	1001	0011	0110	1100	1110	1011	0001	1010	1111	ERROR SUM = 85.00	
1000	0001	0010	0100	1111	0101	0111	1101	1001	0011	0110	1100	1110	1011	0000	1010	0101	ERROR SUM = 85.00	
0101	0011	0110	1000	1001	1111	1101	0111	0001	0010	0100	1100	1110	1011	0001	1010	0100	ERROR SUM = 83.50	
1001	0011	0110	1000	0101	1111	0111	1101	0001	0010	0100	1100	1110	1011	0001	1010	0100	ERROR SUM = 83.50	
1001	0011	0110	1000	1111	0101	0111	1101	0001	0010	0100	1100	1110	1011	0001	1010	0100	ERROR SUM = 83.50	
0101	1000	0011	0100	1100	1111	1101	0111	1001	0011	0010	0100	1110	1011	0001	1010	0001	ERROR SUM = 83.50	
1111	1000	0011	0100	1100	0101	1101	0111	1001	0011	0010	0100	1110	1011	0001	1010	0001	ERROR SUM = 83.50	
0110	0011	1001	1100	0101	1111	1101	0111	0011	0011	0001	0100	1110	1011	0001	1010	0100	ERROR SUM = 79.00	(3)
0110	0011	1001	1100	0101	1111	1101	0111	0011	0011	0001	0100	1110	1011	0001	1010	0001	ERROR SUM = 79.00	
0110	0010	1001	1100	0101	1111	1101	0111	0011	0011	0001	0100	1110	1011	0001	1010	0001	ERROR SUM = 83.50	
1100	1000	0011	0100	0101	1111	0111	1101	1001	0011	0010	0100	1110	1011	0001	1010	0001	ERROR SUM = 83.50	
1100	1001	0011	0100	0101	1111	0111	1101	1000	0011	0010	0100	1110	1011	0001	1010	0100	ERROR SUM = 79.00	(4)
1100	1001	0011	0100	0101	1111	0111	1101	1000	0011	0010	0100	1110	1011	0001	1010	0001	ERROR SUM = 79.00	
0101	0010	1001	1100	0110	1111	0111	1101	0011	0011	0001	0100	1110	1011	0001	1010	0001	ERROR SUM = 83.50	
1111	0010	1001	1100	0110	0101	0111	1101	0011	0011	0001	0100	1110	1011	0001	1010	0001	ERROR SUM = 83.50	
0101	0001	1000	0100	1100	1111	1101	0111	0011	1001	0010	0100	1110	1011	0001	1010	0001	ERROR SUM = 85.00	
1111	0001	1000	0100	1100	0101	1101	0111	0011	1001	0010	0100	1110	1011	0001	1010	0001	ERROR SUM = 85.00	
0101	0001	0010	1100	0110	1111	0111	1101	1001	0011	0001	0100	1110	1011	0001	1010	0001	ERROR SUM = 85.00	
1111	0001	0010	1100	0110	0101	0111	1101	1001	0011	0001	0100	1110	1011	0001	1010	0001	ERROR SUM = 85.00	
0101	0001	1100	0100	0011	1111	0111	1101	1001	1000	0100	0010	1110	1011	0001	1010	0001	ERROR SUM = 84.50	
1111	0001	1100	0100	0011	0101	0111	1101	1001	1000	0100	0010	1110	1011	0001	1010	0001	ERROR SUM = 84.50	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1110	1011	0001	1010	0100	ERROR SUM = 79.00	(5)

(6)

0101	1001	1100	0110	0011	1111	0111	1101	0001	1010	0100	0010	1110	1011	0000	1010	0001	ERROR SUM = 79.00
1100	1000	0101	0110	0011	1111	0111	1101	1001	0001	0100	0010	1110	1011	0000	1010	0001	ERROR SUM = 83.50
1100	1001	0101	0110	0011	1111	0111	1101	1000	0001	0100	0010	1110	1011	0000	1010	0001	ERROR SUM = 79.00
1100	1001	0101	0110	0011	1111	0111	1101	1000	0001	0100	0010	1110	1011	0000	1010	0100	ERROR SUM = 79.00
0011	1001	1100	0010	0101	1111	1101	0111	0001	1010	0100	0110	1110	1011	0000	1010	0100	ERROR SUM = 83.50
0011	1001	1100	0010	1111	0101	1101	0111	0001	1010	0100	0110	1110	1011	0000	1010	0100	ERROR SUM = 83.50
0101	1001	1100	0010	0011	1111	0111	1101	0001	1010	0100	0110	1110	1011	0000	1010	0100	ERROR SUM = 83.50
1100	1001	0011	0110	0101	1111	0111	1101	1000	0001	0010	0110	1110	1011	0000	1010	0100	ERROR SUM = 84.50
1100	1001	0101	0010	0011	1111	0111	1101	1000	0001	0100	0110	1110	1011	0000	1010	0100	ERROR SUM = 83.50
1100	1001	0011	0110	1111	0101	0111	1101	1000	0001	0010	0110	1110	1011	0000	1010	0100	ERROR SUM = 84.50
0010	0001	1000	0110	0101	1111	1101	0111	0011	1011	1100	0110	1110	1011	0000	1010	0101	ERROR SUM = 85.00
0010	0001	1000	0110	0101	1111	1101	0111	0011	1011	1100	0110	1110	1011	0000	1010	1111	ERROR SUM = 85.00
0010	0001	1000	0110	1111	0101	1101	0111	0011	1011	1100	0110	1110	1011	0000	1010	1111	ERROR SUM = 85.00
0010	0001	1000	0110	1111	0101	1101	0111	0011	1011	1100	0110	1110	1011	0000	1010	0101	ERROR SUM = 85.00
0101	0001	1000	0110	0010	1111	0111	1101	0011	1011	1100	0110	1110	1011	0000	1010	0101	ERROR SUM = 85.00
0101	0001	1000	0110	0010	1111	0111	1101	0011	1011	1100	0110	1110	1011	0000	1010	1111	ERROR SUM = 85.00
1111	0001	1000	0110	0010	0101	0111	1101	0011	1011	1100	0110	1110	1011	0000	1010	1111	ERROR SUM = 85.00
1111	0001	1000	0110	0010	0101	0111	1101	0011	1011	1100	0110	1110	1011	0000	1010	0101	ERROR SUM = 85.00
1001	0011	1000	0110	0101	1111	0111	1101	0001	0010	1100	0110	1110	1011	0000	1010	0100	ERROR SUM = 85.00
1001	0011	1000	0110	1111	0101	0111	1101	0001	0010	1100	0110	1110	1011	0000	1010	0100	ERROR SUM = 85.00

STARTING VALUES FOR VERTICES 1,2,3,4 ARE RESP. 0000 0010 1000 0100

****

0						+1,-1						+2,-2			+3,-3	ERROR
A1	A2	A3	A4	A5	A6	B1	B2	B3	B4	B5	B6	C1	C2	C3	D1	E
						++-	+00	+-+	00+	-++	0+0	++0	+0+	0++	+++	
+-0	0-+	-0+	-+0	0+-	+0-	00-	+--	0-0	--+	-00	-+-	-0-	0--	--0	---	000

STARTING VALUES FOR VERTICES 1,2,3,4 ARE RESP. 0000 1100 1000 0100
*** ****** ********* ************************* *********

0						+1,-1						+2,-2			+3,-3	ERROR
A1	A2	A3	A4	A5	A6	B1	B2	B3	B4	B5	B6	C1	C2	C3	D1	E
						++-	+00	+-+	00+	-++	0+0	++0	+0+	0++	+++	
+-0	0-+	-0+	-+0	0+-	+0-	00-	+--	0-0	--+	-00	-+-	-0-	0--	--0	---	000

STARTING VALUES FOR VERTICES 1,2,3,4 ARE RESP. 0000 0110 0001 0100
*** ** *********

0						+1,-1						+2,-2			+3,-3	ERROR	
A1	A2	A3	A4	A5	A6	B1	B2	B3	B4	B5	B6	C1	C2	C3	D1	E	
						++-	+00	+-+	00+	-++	0+0	++0	+0+	0++	+++		
+-0	0-+	-0+	-+0	0+-	+0-	00-	+--	0-0	--+	-00	-+-	-0-	0--	--0	---	000	
0101	1100	1001	1011	1010	1111	1110	0111	1101	1000	0011	0010	0110	0100	0001	0000	1011	ERROR SUM = 84,50

STARTING VALUES FOR VERTICES 1,2,3,4 ARE RESP. 0000 1100 1000 1010

0						+1,-1						+2,-2			+3,-3	ERROR
A1	A2	A3	A4	A5	A6	B1	B2	B3	B4	B5	B6	C1	C2	C3	D1	E
						++-	+00	+-+	00-	-++	0+0	++0	+0+	0++	+++	
+-0	0-+	-0+	-+0	0+-	+0-	00-	+--	0-0	--+	-00	-+-	-0-	0--	--0	---	000

STARTING VALUES FOR VERTICES 1,2,3,4 ARE RESP. 0000 1100 1000 0110

0						+1,-1						+2,-2			+3,-3	ERROR
A1	A2	A3	A4	A5	A6	B1	B2	B3	B4	B5	B6	C1	C2	C3	D1	E
						++-	+00	+-+	00+	-++	0+0	++0	+0+	0++	+++	
+-0	0-+	-0+	-+0	0+-	+0-	00-	+--	0-0	--+	-00	-+-	-0-	0--	--0	---	000

STARTING VALUES FOR VERTICES 1,2,3,4 ARE RESP. 0000 1100 1000 0011

0						+1,-1						+2,-2			+3,-3	ERROR
A1	A2	A3	A4	A5	A6	B1	B2	B3	B4	B5	B6	C1	C2	C3	D1	E

						++−	+00	+−+	00+	−++	0+0	++0	+0+	0++	+++	
+−0	0−+	−0+	−+0	0+−	+0−	00−	+−−	0−0	−−+	−00	−+−	−0−	0−−	−−0	−−−	000

STARTING VALUES FOR VERTICES 1,2,3,4 ARE RESP. 0000 1010 1100 0110

0						+1,−1						+2,−2			+3,−3	ERROR
A1	A2	A3	A4	A5	A6	B1	B2	B3	B4	B5	B6	C1	C2	C3	D	E
						++−	+00	+−+	00+	−++	0+0	++0	+0+	0++	+++	
+−0	0−+	−0+	−+0	0+−	+0−	00−	+−−	0−0	−−+	−00	−+−	−0−	0−−	−−0	−−−	000

STARTING VALUES FOR VERTICES 1,2,3,4 ARE RESP. 0000 1010 1100 0101

0						+1,−1						+2,−2			+3,−3	ERROR
A1	A2	A3	A4	A5	A6	B1	B2	B3	B4	B5	B6	C1	C2	C3	D	E
						++−	+00	+−+	00+	−++	0+0	++0	+0+	0++	+++	
+−0	0−+	−0+	−+0	0+−	+0−	00−	+−−	0−0	−−+	−00	−+−	−0−	0−−	−−0	−−−	000

STARTING VALUES FOR VERTICES 1,2,3,4 ARE RESP. 0000 0101 1000 0110

A1	A2	A3	A4	A5	A6	B1	B2	B3	B4	B5	B6	C1	C2	C3	D1	E	
0						+1,-1						+2,-2			+3,-3	ERROR	
						++-	+00	+-+	00+	-++	0+0	++0	+0+	0++	++-		
+-0	0-+	-0+	-+0	0+-	+0-	00-	+--	0-0	--+	-00	-+-	-0-	0--	--0	---	000	
1011	1010	0011	0011	[illegible]	01[illegible]1	0100	1[illegible]11	1110	00[illegible]0	0001	1101	0101	0110	1000	0[illegible]0	0111	ERROR SUM = 84.50
1100	1010	0011	0011	1011	01[illegible]1	1111	0100	1110	00[illegible]0	0001	1101	0101	0110	1000	0[illegible]0	0111	ERROR SUM = 84.50

PROGRAM FINISHED*

```
06/51,38  0#XFEW  #LICU
06/56,23  0#XFEW  DLTD  FI #XPCK  #XFEW
06/56,24  0#XFEW  CLKD  39
06/56,24  0#XFEW  DLTD
06/56,27  0#XPCK  #XFEW
06/56,40  0#BINA  #XPCK
06/56,48  0#BINA  4.80
06/56,48  0#BINA  HALT  LD
08/24,25  0#BINA  DLTD  00
08/24,26  0#BINA  CLKD  942
08/24,26  0#BINA  DLTD

ACCOUNT CODE    N:1A11      DATE        22/03/75    TOTAL MILL TIME   993
USER NAME       POULIEZOS   START TIME  06/51/29    INPUT RECORDS     364
JOB NAME        PR2         END TIME    08/24/27    OUTPUT RECORDS    732
PERIPHERALS USED;                                   MAX. CORE SIZE    31168
```

Acknowledgements

The author would like to thank his supervisor, Mrs S.P. O'Gorman for her help and guidance in the mathematical aspects of this paper, the Programming Staff at LPCU for their help in the Programming aspects of the problem, and finally Miss H. Mekali, who typed this paper so patiently.

References

Tables 1-7 are taken from a paper by K.W. Cattermole, R.R. Ellis and P.W. Wallace, Department of Electrical Engineering Science, University of Essex, called "A 4B3T Code with improved error properties", Report No. 77, Telecommunication Systems Group.

[1] L.H. Harper, "Optimal assignments of numbers to vertices", J. Soc. Indust. Appl. Math., Vol. 12,No.1, March, 1964, U.S.A.

[2] S.P. O'Gorman, "Minimisation of error extension in pseudo-ternary transmission codes", Page 1.

[3] L. Markides, "Flowcharting Program", final year project, Dept. of Mathematics, Polytechnic of North London, June 1975.

[4] K.W. Cattermole, R.R. Ellis and P.W. Wallace, "A 4B3T Code with improved error properties", Page 5.