UPPER BOUNDS ON THE NUMBER OF BURSTS AT THE OUTPUT OF A VITERBI DECODER

A Thesis

Presented to

the Faculty of the Department of Electrical Engineering

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

by

Narinder K. Goyal

December 1970

ACKNOWLEDGEMENT

The author wishes to acknowledge the encouragement, guidance, and counsel received from Dr. N. M. Shehadeh during the preparation of this thesis.

I would also like to acknowledge the financial support of this research which was sponsored by NASA, Information Systems Division through contract NAS 9 9270.

Narinder K. Goyal

December 1970

UPPER BOUNDS ON THE NUMBER OF BURSTS AT THE OUTPUT OF A VITERBI DECODER

An Abstract of a Thesis

Presented to

the Faculty of the Department of Electrical Engineering

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

by

Narinder K. Goyal

December 1970

ABSTRACT

Burst error statistics at the output of a Viterbi decoder are investigated. Upper bounds on the number of bursts of any length have been derived analytically for a hard decision.

A Viterbi decoder for a hard decision has been simulated on a digital computer. Simulation studies have been made on decoders of different constraint lengths and for different probabilities of error for the Binary Symmetric Channel. Number of bursts of different lengths at the output of these decoders have been counted. The results from the simulation study agree with the upper bounds obtained analytically.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

The Viterbi decoding algorithm has received a great deal of attention recently. This decoding scheme has been shown to be a maximum likelihood decoding scheme and hence, an optimum for decoding convolutional codes. This algorithm performs very well with short constraint lengths.

Simulation results have shown that the errors at the output of a Viterbi decoder occur in bursts, that is, the errors tend to cluster together.[3*] Thus the burst error rate becomes more important than the bit error rate for a Viterbi decoder. Longer bursts at the output cannot be tolerated. In this thesis upper bounds on the number of bursts at the output of a Viterbi decoder have been derived. The validity of the results has been proven by simulating a Viterbi decoder on a digital computer.

In the second and third chapters basic principles of convolutional encoding and Viterbi decoding have been explained. The fourth chapter deals with the communications system model considered for this study. The bounds have been derived in Chapter Five and in Chapter Six the simulation study has been explained in detail.

*These numbers denote references.

CHAPTER II

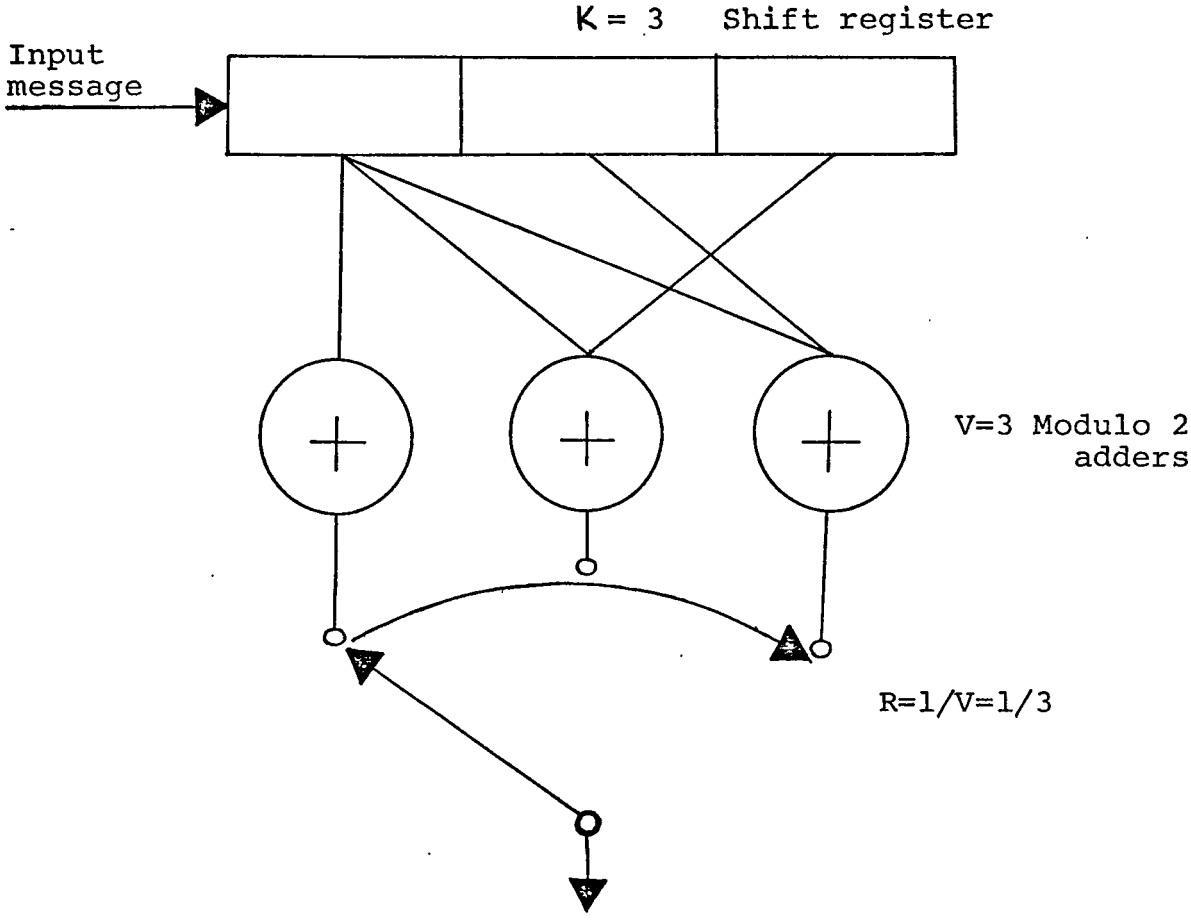
CONVOLUTIONAL ENCODING

A convolutional encoder is a K-stage shift register whose stages are connected to V modulo 2 adders in some prescribed fashion. There is a commutator at the output of the modulo 2 adders. A convolutional encoder with K = 3 and V = 3 is shown in Figure 1. The input digits are shifted one by one into the shift register and for every input digit the commutator makes one revolution producing V output digits per input digit. The rate of the code is then defined as

$$R = 1/V$$

The length K of the shift register is called constraint length of the encoder and is a measure of the duration in which the encoded output digits are affected by any particular input digit.

One way of describing a convolutional encoder is to specify its generator sequence. The generator sequence is defined as the output sequence which results from transmitting a K-bit input sequence consisting of "one" followed by all zeros. Now it is to be noted that if a single "one" is located in some stage S of the shift register the output digit from any modulo 2 adder will be "one" if that adder is connected to stage S or a "zero" otherwise. The length of the generator



Convolutional Encoder

Figure 1

sequence is KV. The first V digits represent the modulo 2 adders which are connected to the first stage, the second V digits represent the adders which are connected to the second stage and so on. For example, the generator sequence of the encoder in Figure (1) is 111 001 010.

The output sequence corresponding to any input message can be obtained by using the generator sequence. It employs the principle of superposition and successive shifts of the generator sequence are added (modulo 2). An example is given in Figure 2.

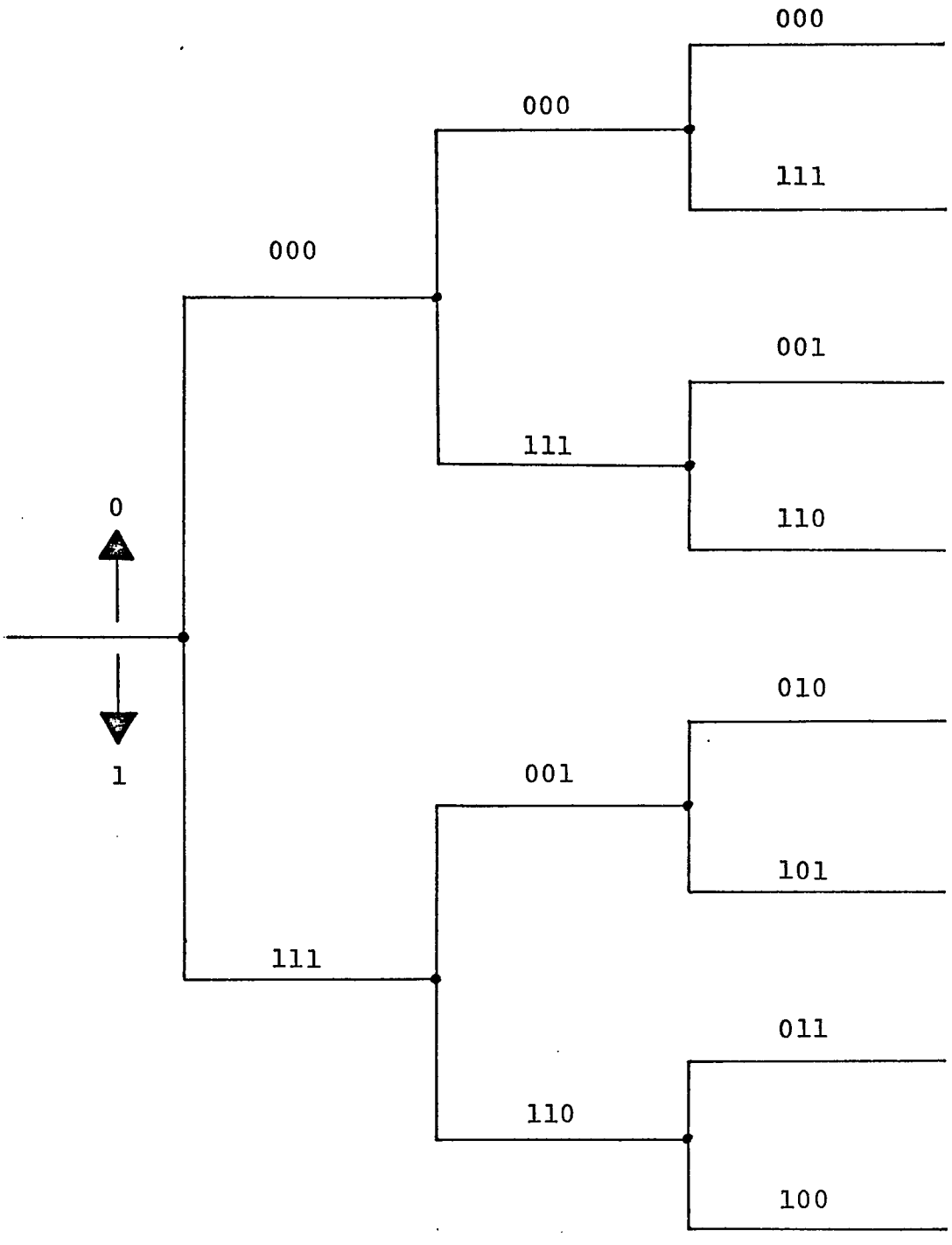
Each bit shifted into the encoder results in one of the two possible V-bit output sequences. One sequence corresponds to shifting in a "one" and the other corresponds to shifting in a "zero". However, the resulting sequence depends on the previous K-1 digits shifted into the encoder. Thus a particular digit not only affects the immediately resulting sequence but also the next K-1 V-bit output sequences.

A convolutional encoder can best be represented by a tree structure as in Figure 3. From every node we have two branches and the digits on the branches represent the output sequences. The tree is so arranged that the upper branch from a node corresponds to shifting in a "zero" and the lower branch corresponds to shifting in a "one" in the encoder. The output sequence corresponding to any message can now be

Input Digits	Generator Sequence							
1	111	001	010					
0		000	000	000				
1			111	001	010			
0				000	000	000		
1					111	001	010	
⋮								
Encoded Output Digits	111	001	101	001	101	----	-----	-------

Determination of encoded output digits corresponding to an input message for the convolutional encoder with generator sequence 111 001 010

Figure 2



Tree structure for convolutional encoder with Generator Sequence 111 001 010

Figure 3

found by following the appropriate path through the tree.

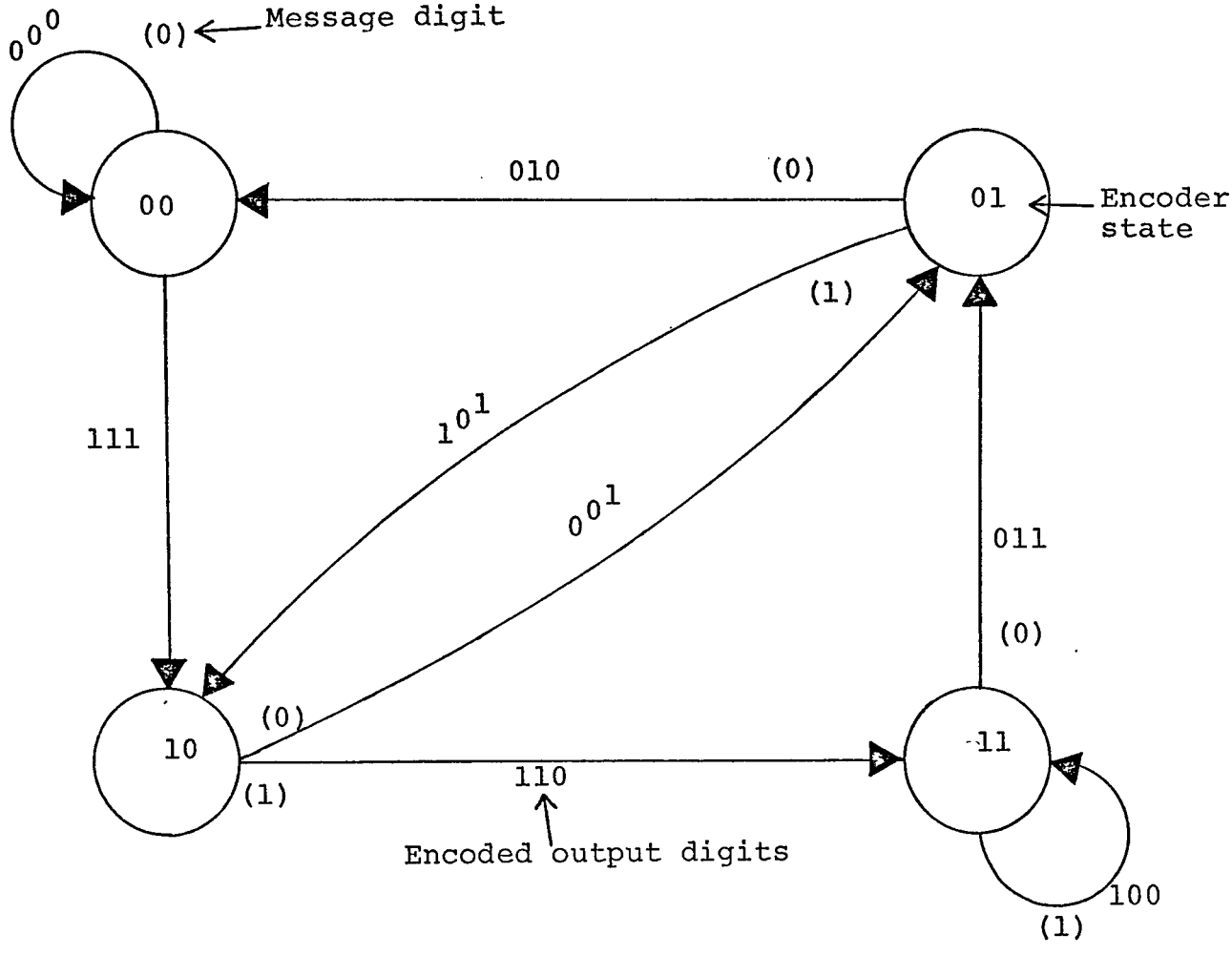
STATE DIAGRAMS

The state of an encoder is defined as the contents of the first K-1 stages in the shift register. As the new digit is shifted into the encoder the most ancient Kth bit is shifted out of the register and so it cannot affect any subsequent sequence. This is the reason that the Kth bit is not considered for defining a state of an encoder. Now if the encoder is in a particular state there are only two possibilities, either we shift in a "zero" or a "one". It means that we can go into only one of the two possible states from a given state.

The state diagram for the encoder discussed in the previous examples is given in Figure 4. As an example, if we are in a state of 01 we can only go to state of 00 or 10. The state 00 results, if we send a zero and state 10 results, if we send a one.

Thus it becomes very easy to determine the output sequence corresponding to any input message by going through the state diagram of the encoder. This in fact is a simplification of the tree structure.

The state diagram can be drawn in a different form which is more convenient for discussing the Viterbi Algorithm. This diagram is called a trellis structure. The states are now



State Diagram

Figure 4

represented as nodes and the transfer from one node to another is represented by an arrow. The trellis structure is given in Figure 5. The important points about the trellis structure are given below. These will be used later on for deriving the upper bound on the number of bursts out of a Viterbi decoder.

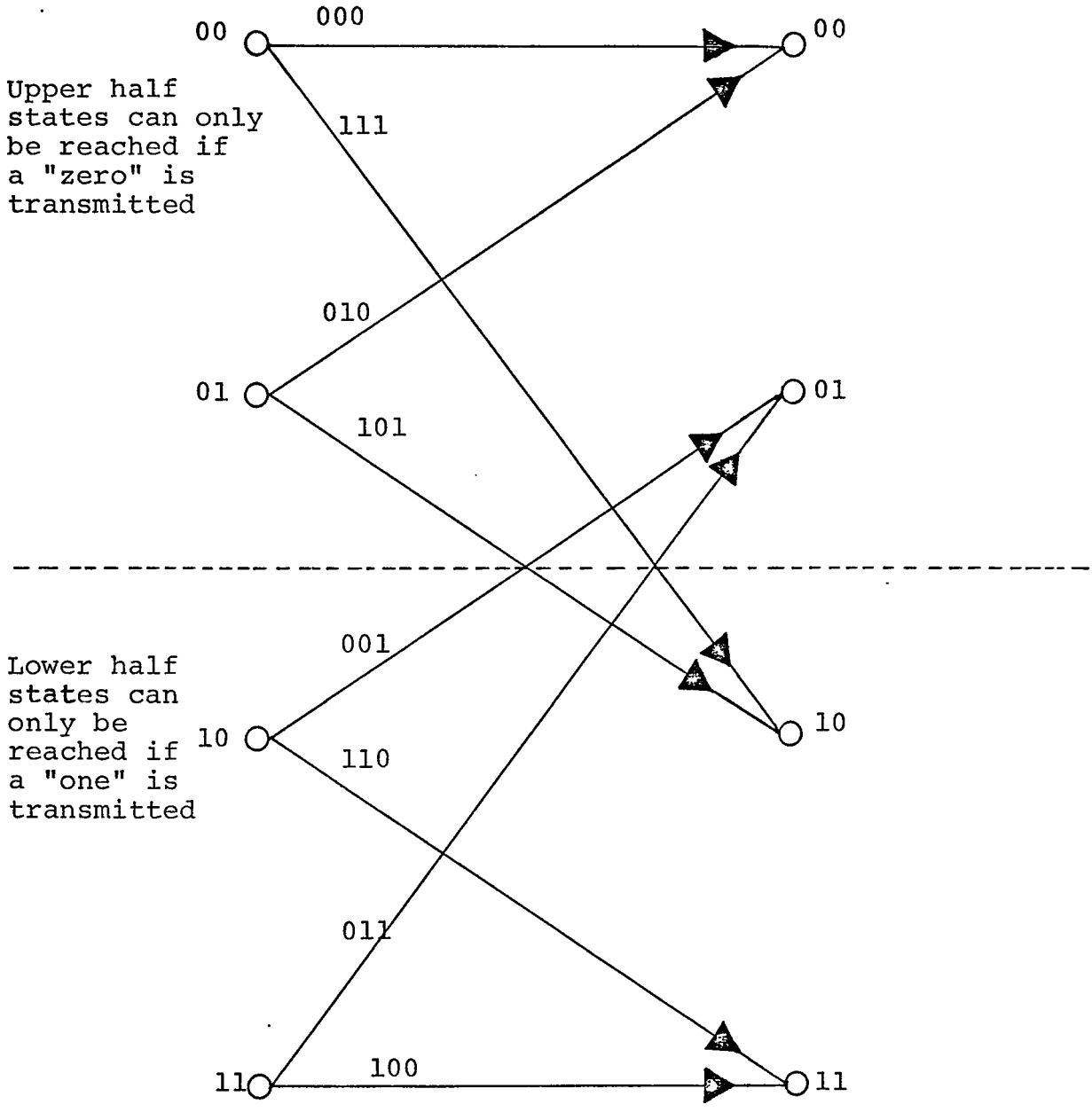
1. There are only two states that can be reached from a given state.

2. A given state can be reached only from one of the two previous states.

3. The moves from state to state are independent of the particular generator sequence. Thus for all encoders of a particular constraint length K the shape of the trellis structure will be the same. However, the output digits on the branches are dependent on the modulo 2 adders connections or the generator sequence.

4. The uppermost of the two paths leaving any node corresponds to shifting in a "zero" and the lower path corresponds to shifting in a "one."

5. There are in all 2^{K-1} states. The upper half states (2^{K-2}) can only be reached if we transmit a zero and the lower half states (2^{K-2}) can be reached only if we transmit a one. This is because we arranged the states in such a way that all states starting with zero were placed at the top and



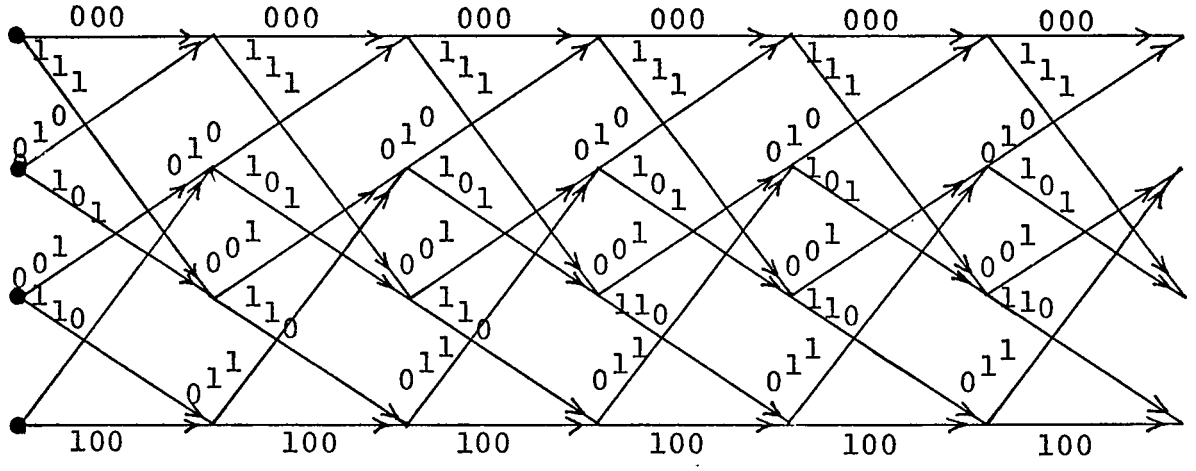
Trellis Structure

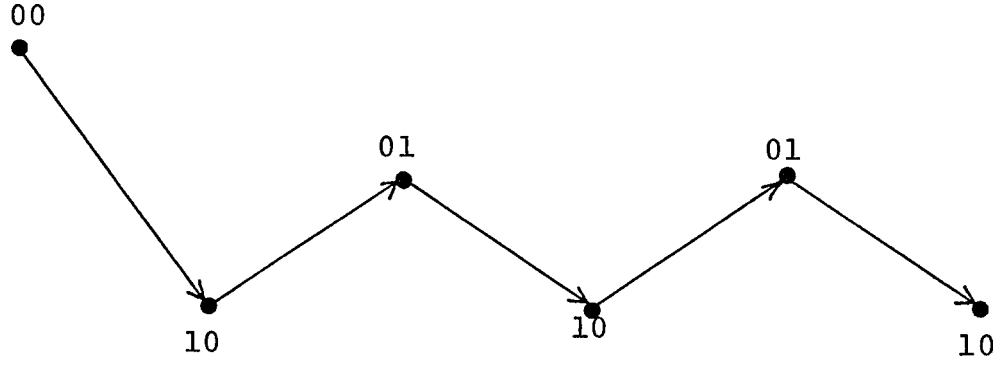
Figure 5

those starting with one were placed below. Since the first digit of the new encoder state has to be the same as the message digit that was just shifted in, the upper half states can only be reached if zero is shifted in and the lower half states can only be reached if a one is shifted in.

6. If we are in one of the lower half states and we send K-1 zeros, the encoder will come to the all zero state, that is, the top most state. This property is very obvious because when the encoder is in one of the lower half states, there is a one in the first stage of the shift register and sending in K-1 zeros pushes this to the Kth stage and all zeros in the first K-1 stages which corresponds to the all zeros state of the encoder.

The output sequence for a given message can be found entirely from the trellis structure. However, for convenience, the structure is expanded to a form given in Figure 6. The encoding procedure then reduces to only going along a path through the expanded version of the trellis structure corresponding to the input message.





Expanded version of trellis structure
and encoding of message digits by tracing a path through it.

Figure 6

CHAPTER III

THE VITERBI DECODING ALGORITHM

The Viterbi decoding algorithm can best be visualized in terms of the expanded version of the trellis structure. We already have seen that encoding operation is nothing but a path through the expanded trellis structure depending on the input message. The decoding procedure is just the reverse of this operation. The noise in the channel changes some of the digits in the encoded sequence and the Viterbi decoding algorithm attempts to find a path through the expanded trellis structure which is as close as possible to the received sequence. The information sequence corresponding to this path is then taken to be the original input message sequence. The decoding algorithm is maximum likelihood decoding and is optimum in the sense that a most probable transmitted message is selected depending upon the received sequence. The Viterbi decoder utilizes the state diagram of the encoder and the received sequence to find the most likely information that was transmitted.

From the trellis structure it can be seen that once any two paths merge into the same state, both paths have identical extensions out of that state and after this state the two paths will correlate equally well with the received sequence. So

we should always be able to discard one of the two paths entering any state otherwise we shall violate the objective of finding a path which correlates best with the received sequence. Thus we discard those paths which can never be candidates for the highest correlated path. This is the idea on which the Viterbi algorithm works.

The main steps in the Viterbi algorithm are as follows:

1. A received branch (V-bit long) is shifted into the decoder. It is compared with each of the two possible branches out of each 2^{K-1} states. The delta scores are generated. A delta score is defined as the number of positions in which the received branch differs from the branch with which it is being compared. There are 2^{K} branches in all which are to be compared with the received sequence.

2. These delta scores for the two paths leaving each state are added to the previous delta scores (initially zero) for that state.

3. Delta scores for the two paths terminating in each of the next 2^{K-1} states are compared and the path having lower score is kept while the other is dropped. If the two scores are equal, one of the paths is arbitrarily dropped. The delta scores for 2^{K-1} states are stored in the score registers.

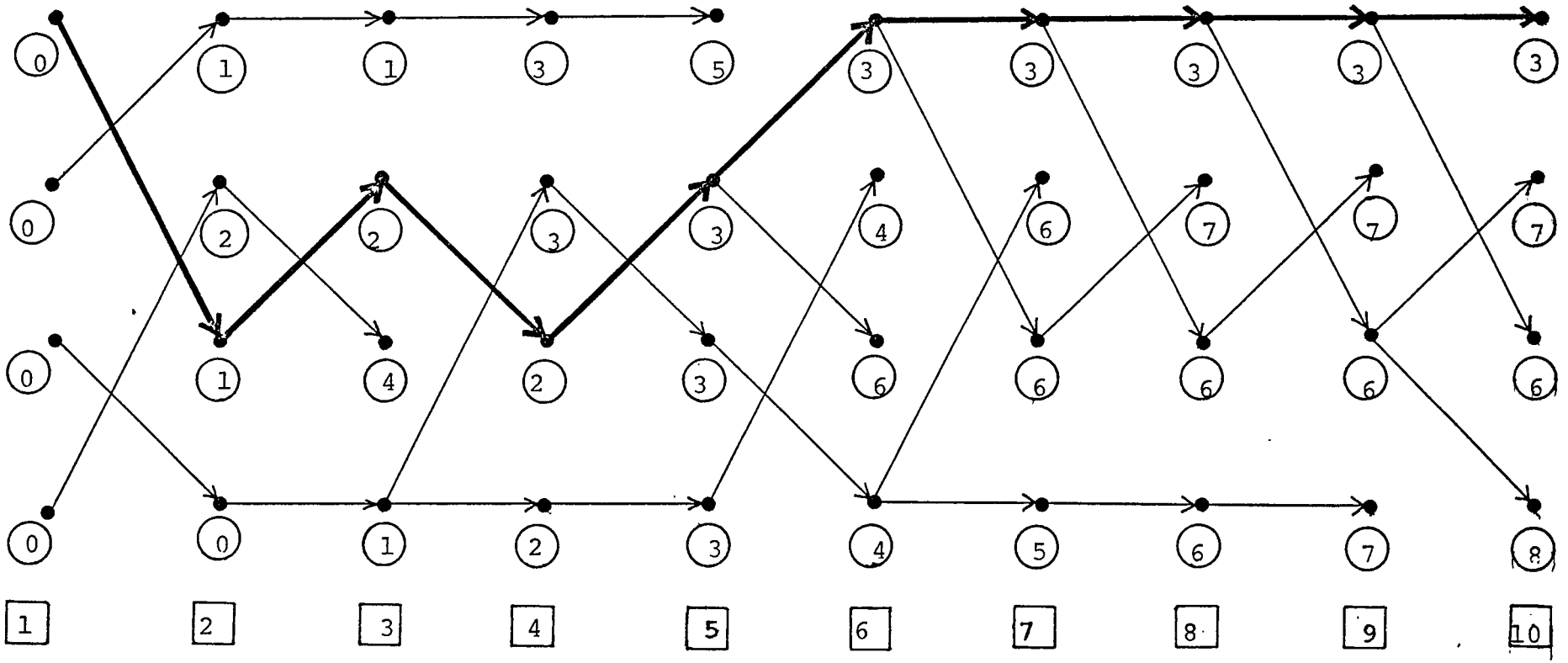
4. The information sequences corresponding to the

paths are called survivor sequences and these are stored in survivor sequence registers for every state.

There should be a scheme for transfer of entire contents of one register to another. For example, if the survivor path terminating in state 00 has come from 01 the new contents of delta score register for 00 state should be the contents of state 01 plus the delta score for branch between 01 and 00. Similarly the contents of survivor sequence register for 01 should go into survivor sequence register of state 00 plus the new bit which resulted in entering the state of 00. The operation of the Viterbi algorithm is explained by an example in Figure 7. The table for the delta score registers and the survivor sequence registers is also given in Figure 8.

As we go on receiving more and more bits the length of the survivor sequence registers goes on increasing. The score registers also have to handle larger numbers. So the question is at what time we make the decision on information bits and how to protect overflow of the registers. It has been observed that if the survivor sequence registers are made B bit long where B is about 5 or 6 times K (constraint length), all registers, with high probability, agree as to the initial few bits in the survivor sequences, even though the channel is too noisy. Thus the bit decision can be delayed B bits where B = 5K or 6K. In case the initial bit is not the same in all the registers

Transmitted sequence	111 001 101 001 010 000 000 000 000 000
Error sequence	001 001 000 100 000 000 000 000 000 000
Received sequence	110 000 101 101 010 000 000 000 000 000



Example for Viterbi Decoding

Figure 7

we shall take the survivor sequence which has the least delta score. Thus a decoded bit will be shifted out of the registers and a new bit will be entered making the decoding process bit by bit and controlling the survivor sequence registers.

The overflow of the delta score registers is controlled by subtracting the lowest delta score from all delta score registers when an overflow condition arises. This is done because we are interested only in the diffence in the delta scores rather than their absolute values.

Step No.	REGISTER CONTENTS							
	SS_{00}	SC_{00}	SS_{01}	SC_{01}	SS_{10}	SC_{10}	SS_{11}	SC_{11}
1	--	-	--	-	--	-	--	-
2	0	1	0	2	1	1	1	0
3	00	1	10	2	01	4	11	1
4	000	3	110	3	101	2	111	2
5	000 0	5	101 0	3	110 1	3	111 1	3
6	101 00	3	111 10	4	101 01	6	110 11	4
7	101 000	3	110 110	6	101 001	6	110 111	5
8	101 000 0	3	101 001 0	7	101 000 1	6	110 111 1	6
9	101 000 00	3	101 000 10	7	101 000 01	6	110 111 11	7
10	101 000 000	3	101 000 010	7	101 000 001	6	101 000 011	8

Table for delta score and survivor sequence registers

SC = Delta score register

SS = Survivor sequence register

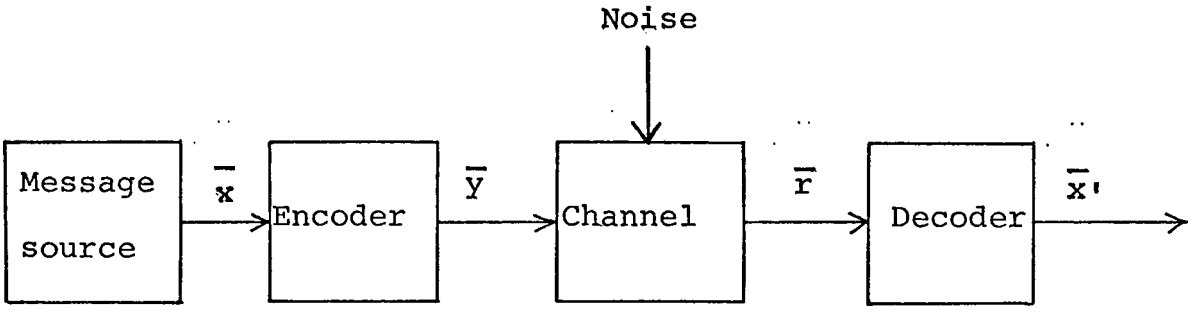
Figure 8

CHAPTER IV

DESCRIPTION OF THE COMMUNICATIONS SYSTEM CONSIDERED FOR THE PRESENT STUDY

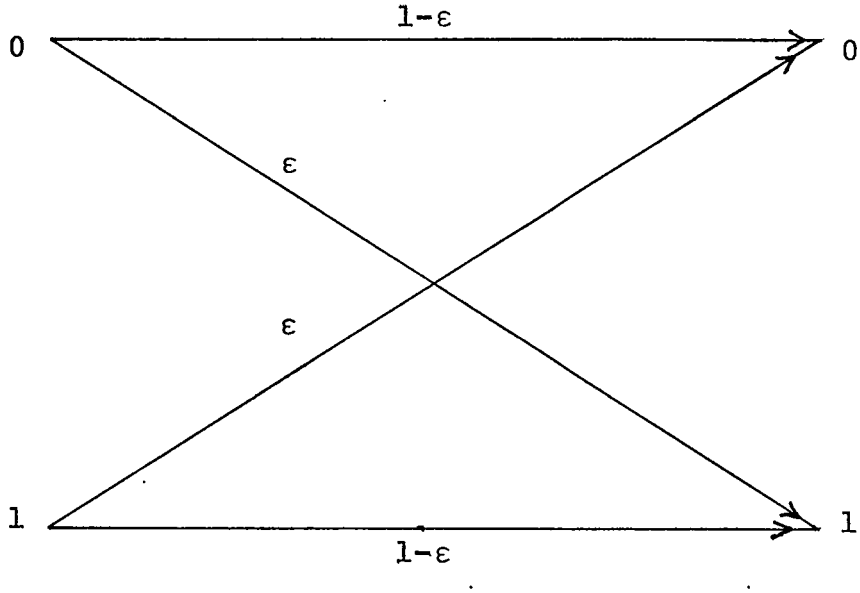
The communications system in which we are interested can be represented by the block diagram in Figure 9.[2] The source produces a sequence $\bar{x}$ of binary digits "0" and "1". This we call the information sequence. This information sequence is sent into the encoder which transforms this sequence to $\bar{y}$. The sequence $\bar{y}$ is sent through the channel. The channel disturbs the input sequence $\bar{y}$ and produces a sequence $\bar{r}$ at the output which is noise corrputed version of sequence $\bar{y}$. The purpose of the decoder is to receive $\bar{r}$ as input and it tries to reproduce the original information sequence. Depending on $\bar{r}$ the decoder produces a sequence $\bar{x}'$ which is taken to be as the original information sequence.

The channel was taken to be as a Binary Symmetric Channel.[2] A Binary Symmetric Channel (abbreviated BSC) can be best represented by a transitional probability diagram of Figure 10. The input to the channel consists of either "0" or "1" (Binary Symbols) and the channel produces binary symbols at its output. Each digit in the input sequence can be received correctly with some fixed probability $(1-\varepsilon)$ and



Communications System

Figure 9



Binary Symmetric Channel

Figure 10

is altered by noise into the opposite digit with probability of ε where $\varepsilon<1/2$. The transitional probabilities are constant and are independent of the value of the symbol being transmitted. The noise effect is thus to reverse the transmitted digits. The probability of error in a BSC can be connected to a practical system by an important parameter called signal to noise ratio.[4,7] For an antipodal binary set of signals, ε is given by

$$\varepsilon = 1/2 - \text{erf}\sqrt{\frac{2E}{N_o}}$$

where

$$\text{erf}(x) = \frac{1}{\sqrt{2\pi}} \int_o^x e^{-y^2/2}\, dy$$

E = Energy per digit.

N_o = One-sided noise spectral density.

E/N_o = Signal to noise ratio.

The plot of ε vs E/N_o in dB gives the performance curve. When the information bits are encoded the number of channel bits becomes V times the number of information bits and this reduces the signal to noise ratio V times, for a given transmitter power.

The encoder is a convolutional encoder and the decoder is a Viterbi decoder. Different decoders of various constraint lengths were tried.

The source was supposed to be producing an all "zeros" sequence and when this all "zeros" sequence goes into encoder we get an all "zeros" encoded sequence. Thus an all "zeros" sequence was taken to be the correct sequence. This can always be done because convolutional codes are group codes and whatever results we get for an all "zeros" sequence should be valid for any other arbitrary sequence. Thus while deriving the bounds the all "zeros" path in the expanded trellis structure was taken to be the correct path.

CHAPTER V

DERIVATION OF UPPER BOUNDS ON THE NUMBER OF BURSTS

As mentioned in Chapter IV the assumption here will be that an all "zero" sequence is transmitted through the channel. So a "one" appearing in the decoded sequence will represent an error in that position. A burst here will be defined as a set of binary digits Z_n, $Z_{n+1}...Z_{n+b-1}$ in the decoded sequence,[3] if and only if:

1) $Z_n = Z_{n+b-1} = 1$

2) (K-1) digits on each side of $Z_n...Z_{n+b-1}$ are all zeros.

3) There is no consecutive sequence of (K-1) zeros within the set Z_n, $Z_{n+1}...Z_{n+b-1}$, where K is the constraint length of the encoder.

The length of the burst is taken as the size of the set which is equal to b. It is to be noted that all digits in a burst need not to be errors. When they are all in errors a burst is called a solid burst.

Example If K = 4

1) ...00010101000... is a burst of length 5

2) ...000111000... is a solid burst of length 3

This particular definition of a burst will be followed while deriving upper bounds on the number of bursts.

UPPER BOUNDS FOR SOLID BURSTS

For an all zero message if the decoder takes the all zeros path, it makes no errors. If it takes any other path, we are making some errors in the decoding process as shown in Figure 11.

If we decode a particular bit into "1", it means that we are in one of the branches of trellis structure which are going downwards and the decoder ends up in one of the lower half states. This follows from the properties of trellis structure discussed in Chapter II. A burst always ends with a "1", so at the end of a burst we should always be in one of the lower half states. Again a burst must be followed by (K-1) zeros. Decoding the next (K-1) bits into zeros puts the decoder in the all zero state, that is, back to the correct path, as shown in Figure 12. It means that if at all nother burst occurs it should start at the uppermost all zero state. So a path which produces a burst separates itself from the all zeros path at some point and again goes back to it at some other point. Every path of this kind gives rise to a particular pattern of burst. For example, for a solid burst of length one there is only one path as shown in Figure 13. Again there is only one path which gives rise to a solid burst of length two and the same is true for any other solid burst of length b. So if at all a solid burst of length b occurs

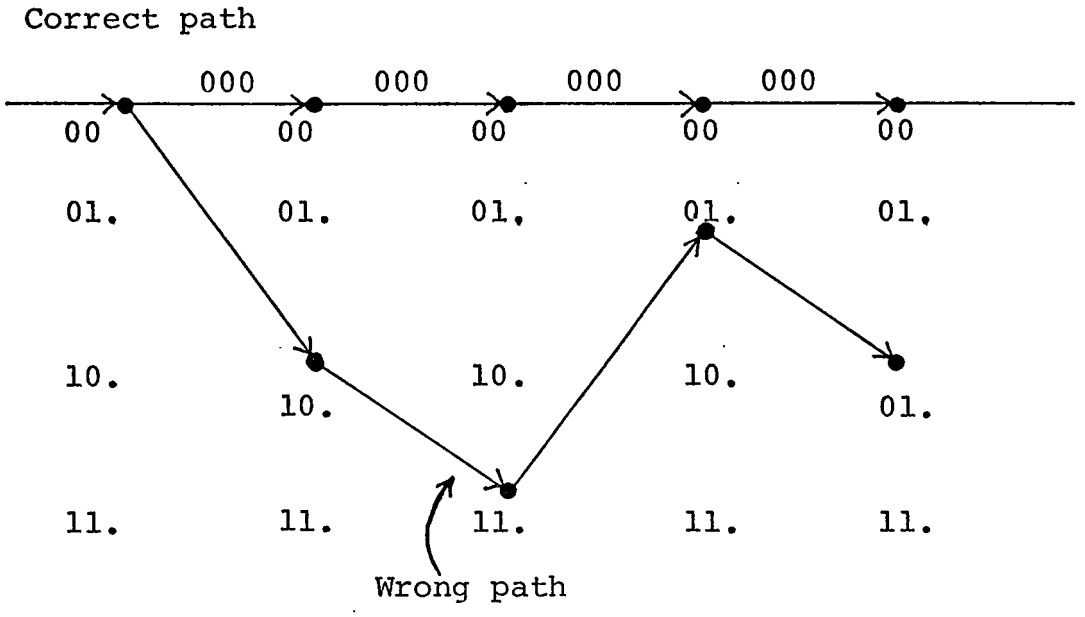


Figure 11

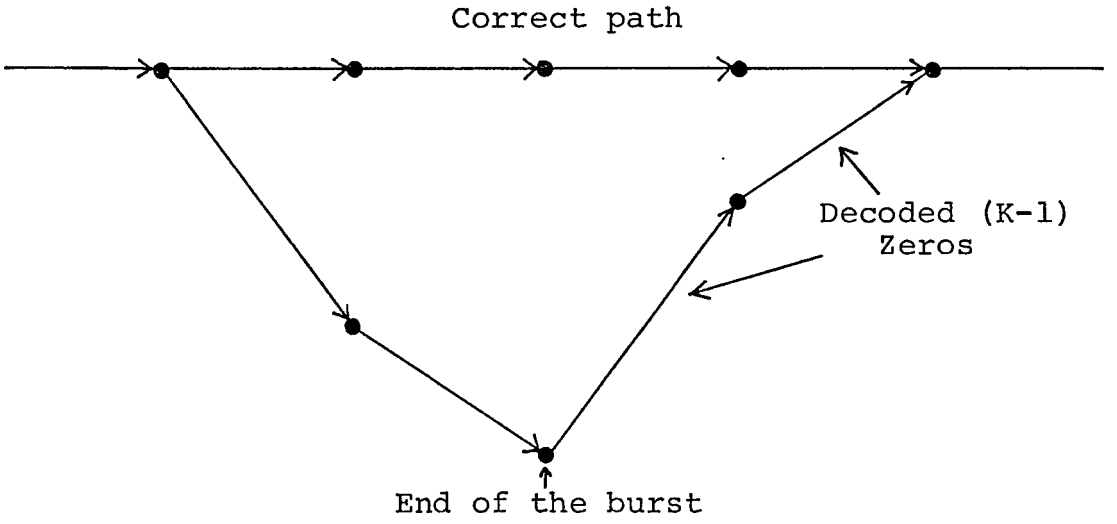
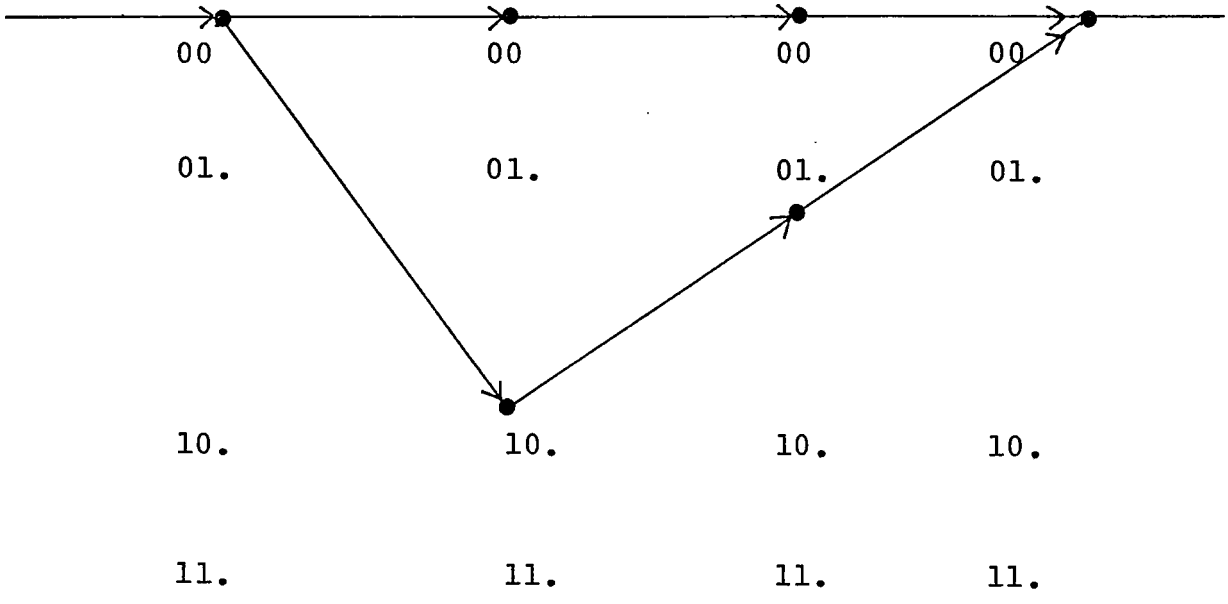


Figure 12



Path giving a solid burst of length 1;
this is the only path which does so.

Figure 13

the decoder should have taken that particular path which produces burst of length b and no other path.

Suppose the weight of the path which corresponds to a solid burst of length b is w. By weight we mean the total number of "ones" in the path. This weight can always be found out from the trellis structure for any solid burst. In order that we choose this path over the all zero path we must make w/2 or more errors in the transmission through the channel. Only then the received sequence will be closer to this path than the all zero path. The errors should be in symbols in which this path differs from the all zeros path. So in order to go into this path we should make w/2 or more errors in w bits. If ε is the probability of error in the BSC, then the probability of making w/2 or more errors in w digits is given by

$$P_b = \sum_{i=w/2}^{w} \binom{w}{i} \varepsilon^i (1-\varepsilon)^{w-i} - 1/2 \binom{w}{w/2} \varepsilon^{w/2} (1-\varepsilon)^{w/2} \qquad (1a)$$

when w/2 is an integer.

The second term in the above expression is for the case when exactly w/2 errors occur resulting in a tie between the all zero path and weight w path so that errors are made only one half of the time.

$$P_b = \sum_{i=w'}^{w} \binom{w}{i} \varepsilon^i (1-\varepsilon)^{w-i} \qquad (1b)$$

When w/2 is not an integer, w' is the integer part of (w/2+1).

P_b is the probability that a solid burst of length b occurs.

It has been shown that the expression for P_b can be upper bounded[8] by

$$P_b < 2^w [\varepsilon(1-\varepsilon)]^{w/2}$$

Suppose we transmit N bits so that N is very large. Then according to the weak law of large numbers we should expect $N\varepsilon$ errors where ε is probability of error in the BSC. Now a solid burst of length b occurs only if we take the loop or path of weight w. The maximum number of such loops we can have (Figure 14) with $N\varepsilon$ errors is given by

$$N_{bmax} = \frac{N\varepsilon}{w/2} = \frac{2N\varepsilon}{w} \quad \text{if } w/2 \text{ is an integer}$$

$$N_{bmax} = \frac{N\varepsilon}{w'} \quad \text{if } w/2 \text{ is not an integer and } w' \text{ is as defined earlier.}$$

A loose upper bound on number of solid bursts of length b, then, can be written as follows.

$$N_b \leq \frac{2N\varepsilon}{w} \quad \text{if } w/2 \text{ is an integer.}$$

$$N_b \leq \frac{N\varepsilon}{w'} \quad \text{if } w/2 \text{ is not an integer.}$$

This is a very loose upper bound. It is to be noted that while deriving this bound we apparently assumed that in every w symbols there will be w/2 errors and that we will always take the wrong path. This means we assumed that a solid burst of

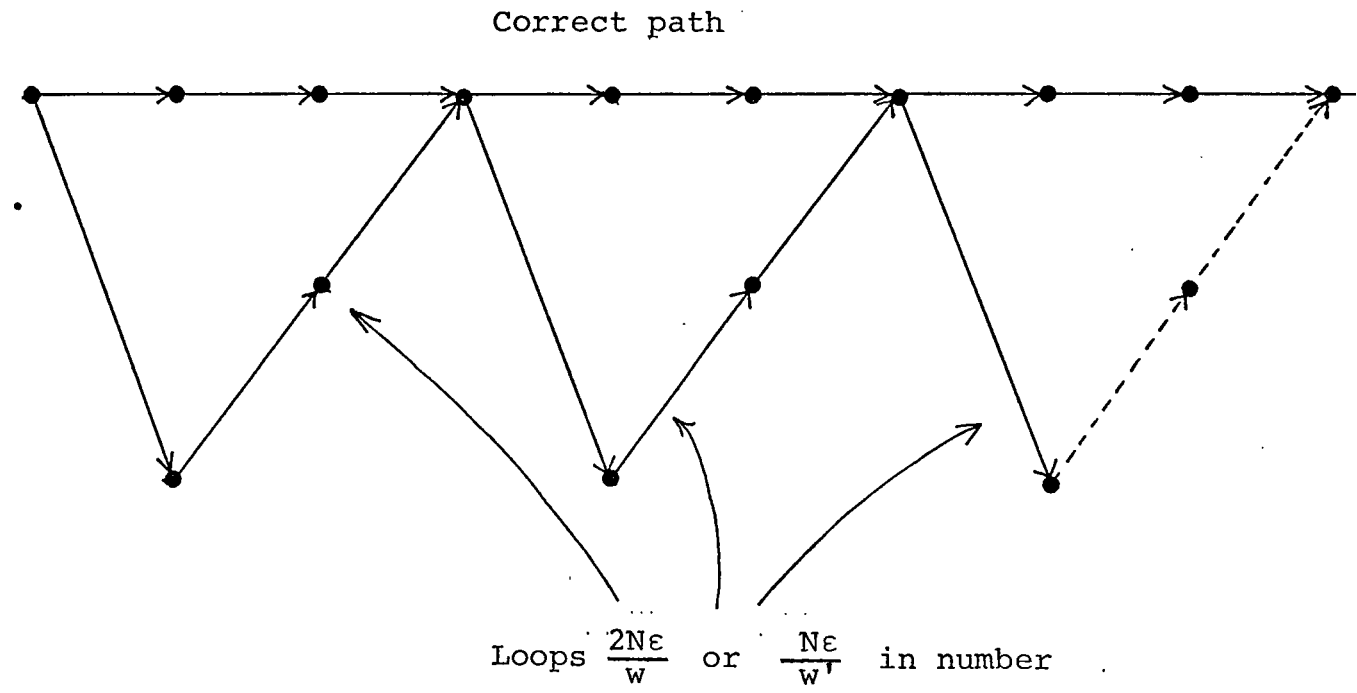

Correct path
Loops $\frac{2N\epsilon}{w}$ or $\frac{N\epsilon}{w'}$ in number

Figure 14

length b is going to occur with a probability of one. However, the probability of occuring of a solid burst of length b is given by P_b as we found out earlier. So weighting this bound by P_b will tighten this upper bound and we get

$$N_b \leq \frac{2N\varepsilon P_b}{w} \quad \text{if } w/2 \text{ is an integer}$$

$$N_b \leq \frac{N\varepsilon P_b}{w'} \quad \text{if } w/2 \text{ is not an integer.}$$

Byt P_b is bounded by

$$P_b < 2^w \, \varepsilon(1-\varepsilon)^{w/2}$$

therefore we have the following upper bound for the solid bursts.

Theorem 1:

The upper bound on the number of solid bursts at the output of a Viterbi decoder is:

$$N_b \leq \frac{2^{(w+1)} [\varepsilon(1-\varepsilon)]^{w/2} \varepsilon N}{w}, \text{ if } w/2 = \text{integer}$$

$$N_b \leq \frac{2^w [\varepsilon(1-\varepsilon)]^{w/2} \varepsilon N}{w'}, \text{ if } w/2 \neq \text{integer} \qquad (2)$$

where

N_b = Number of solid bursts of length b.

w = Weight of the path which gives solid bursts of length b.

ε = Probability of error in the BSC.

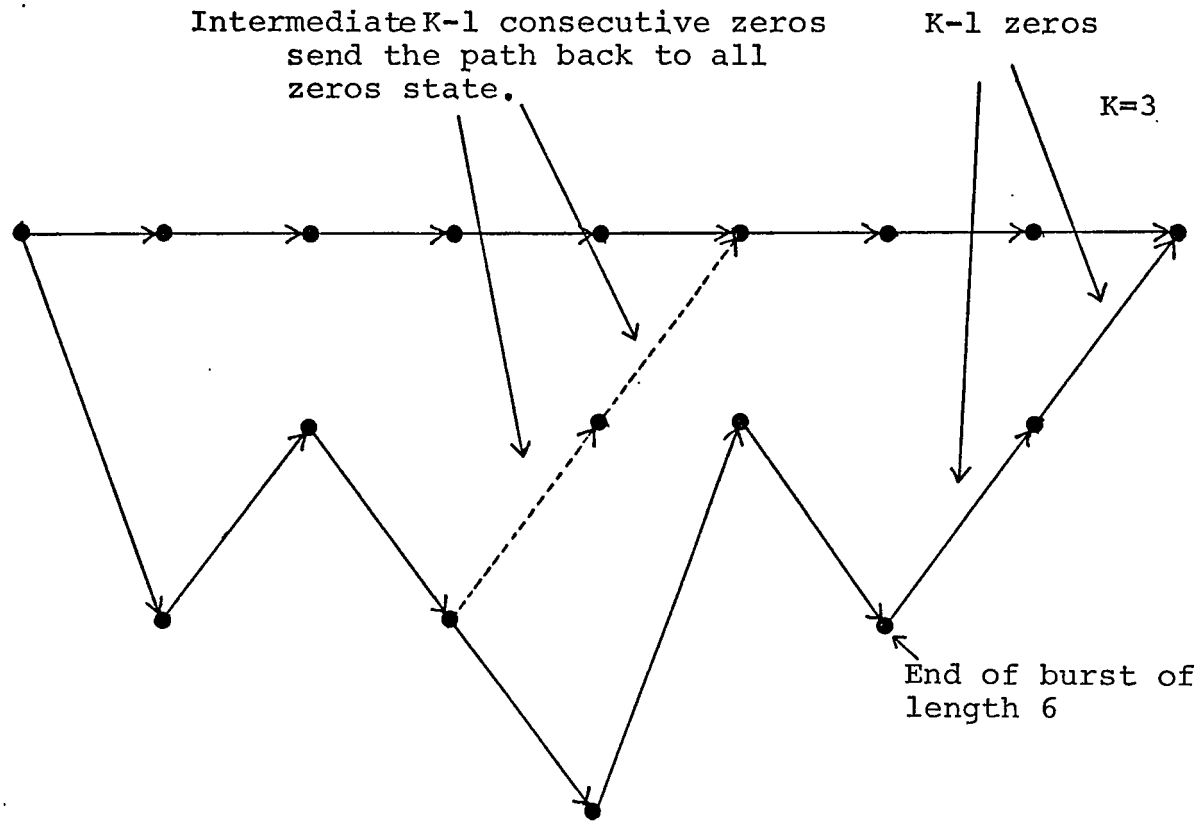
N = Number of channel bits transmitted.

w' = The integer part of (w/2 + 1).

UPPER BOUNDS FOR NON-SOLID BURSTS

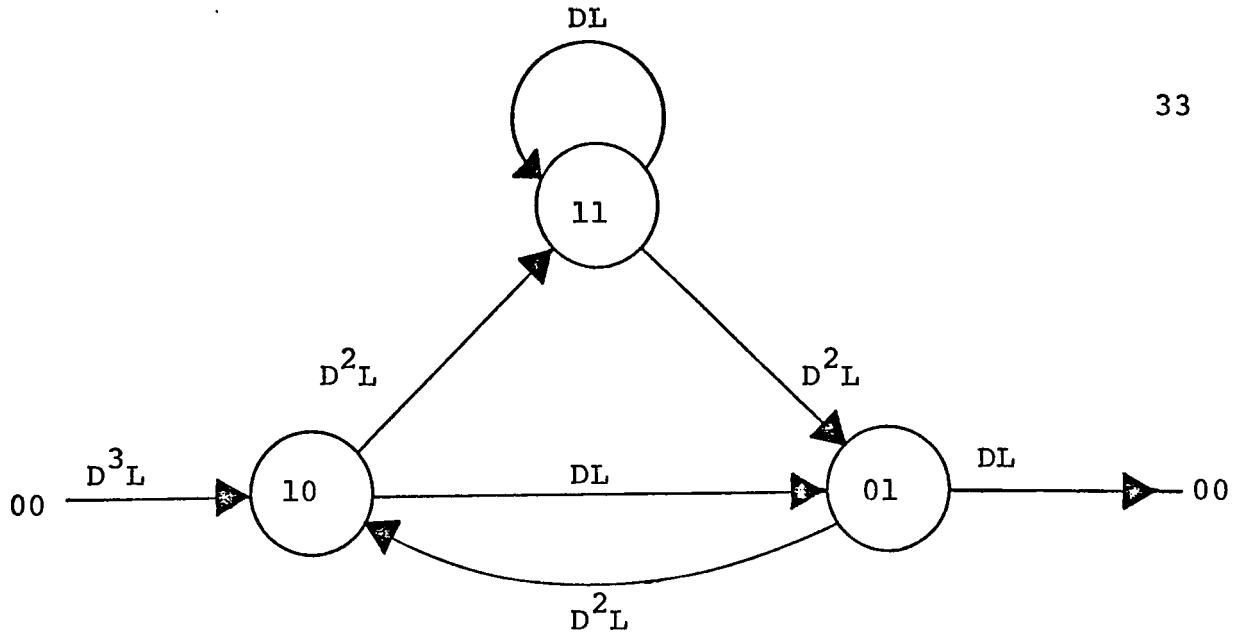
As seen earlier there is only one path which gives rise to a solid burst of length b. For non-solid burst there can be more than one path. For example, for K = 4, ...00010101000... is a non-solid burst of length 5 and so is ...00011011000... . But these two patterns are provided by different paths in the trellis structure. We have also noted that a burst always starts at all zero state and ends in one of the lower half states and after that there should be (K-1) decoded zeros which puts the decoder back in the all zero state. For a burst of length b the length of the path from the point where it separates from all zero path to the point where the end of the burst takes place should be equal to b. After this point the path is (K-1) bits long and merges again with the all zero path. The path that contributes to a burst of length b should be (b+k-1) bits long. This is shown in Figure 15. Now any path which is (b+k-1) bits long cannot have a combination of (K-1) consecutive zeros in the first b places. If it had (K-1) consecutive zeros anywhere in the first b bits it should have gone back to the all zero state. So every path which is (b+k-1) bits long gives rise to a burst of length b.

Now our purpose is to find the number of such paths which separate from all zero path and go back to it at some

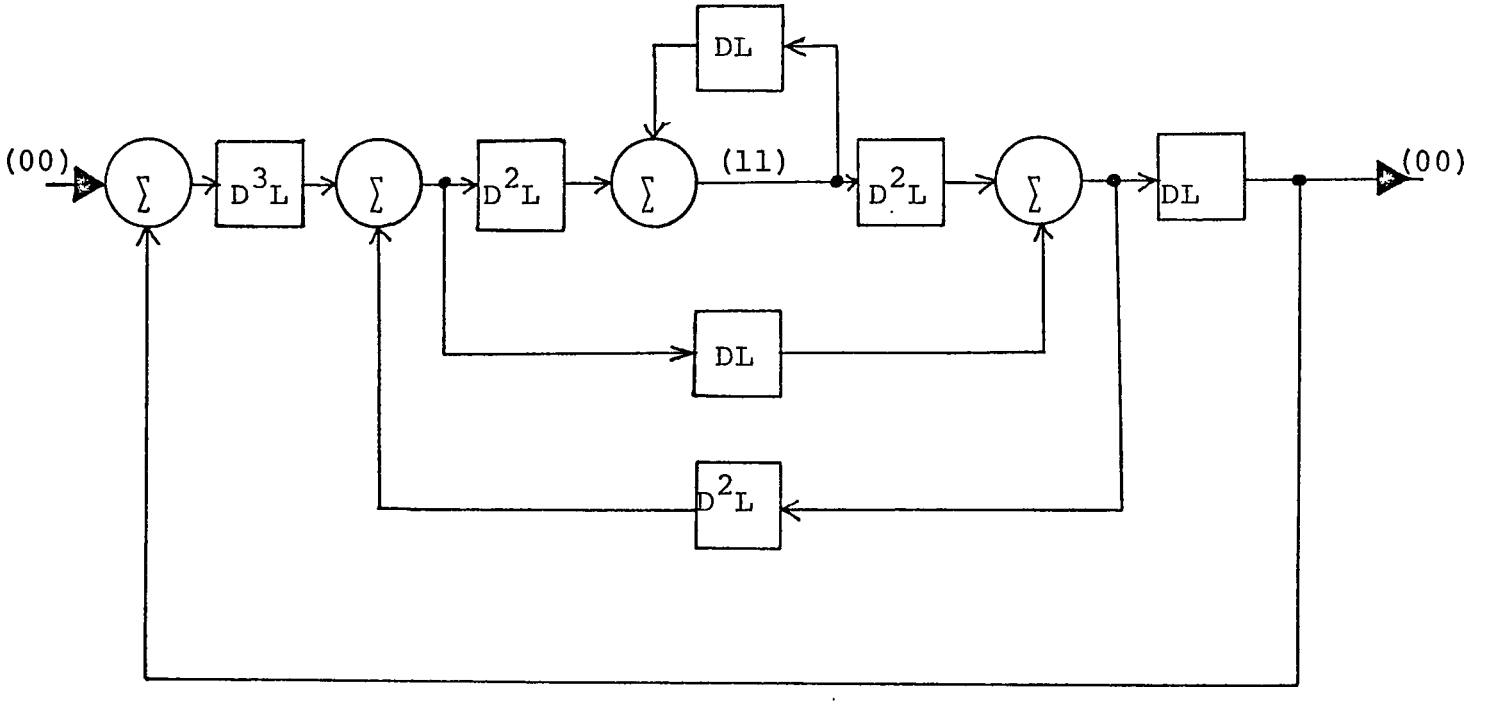


Path of length 8 gives burst of length 6.

Figure 15



Signal Flow Chart for State Diagram
Figure 16



Block Diagram for the State Diagram
Figure 17

other point and are (b+k-1) bits in length. Also we are interested in the weights of these paths. This can be done by considering the state diagram[8] (Figure 4). The state diagram is opened at the self loop on all zeros state and is drawn as given in Figure 16. The paths on the diagram are labelled according to their weights and lengths. The exponent of D gives the weight of the path and the exponent of L gives the length of the path. This diagram is drawn as a signal flow diagram as in Figure 17, and the closed loop transfer function provides us an enumeration of every path. For example, the transfer function for the system in Figure 17 is given by

$$T(D,L) = \frac{\cdots D^8L^4 - D^6L^4 + D^5L^3 \cdots}{1 - DL - D^6L^3 - D^3L^2 - D^8L^2 + D^6L^4 + D^6L^4 - D^5L^3} \qquad (3)$$

Expansion of this gives

$$T(D,L) = D^5L^3 + D^8L^4 + D^9L^5 + D^{10}L^6 + 2D^{11}L^6 + \ldots \qquad (4)$$

which tells us that the path of length 3 has a weight of 5, one path of length 6 has a weight 10 and the other two have weight 11. This expression can be expanded further and knowledge about longer paths can be obtained very easily. So by an investigation of equation (4) we can find out the number of paths of any length and their weights.

Supposing we find from (4) that there are n paths which give rise to a burst of length b, that is, there are n paths

(b+k-1) bits long. Weights of these n paths can be found by equation (4). Let the weights of these paths by $w_1, w_2, \ldots w_n$. These n paths also include a path which gives solid bursts of length b. So we exclude that path and are left with only (n-1) paths and their weights are $w_1, w_2 \cdots w_{n-1}$.

We can go into the jth path only if we make $w_j/2$ or more errors in w_j bits. The probability P_{bj} that we go into the jth path is given by

$$P_{bj} = \sum_{i=w_j/2}^{w_j} \binom{w_j}{i} \varepsilon^i (1-\varepsilon)^{w_j-i} - 1/2 \binom{w_j}{w_j/2} \varepsilon^{w_j/2} (1-\varepsilon)^{w_j/2}$$

if $w_j/2$ = integer

$$P_{bj} = \sum_{i=w'_j}^{w_j} \binom{w_j}{i} \varepsilon^i (1-\varepsilon)^{w_j-i} \qquad \text{if } w_j/2 \neq \text{integer}$$

w'_j = integer part of $(w_j/2+1)$

Again we can upper bound P_{bj} by the expression

$$P_{bj} < 2^{w_j} [\varepsilon(1-\varepsilon)]^{w_j/2} \tag{5}$$

Now whenever we go into one of these (n-1) paths we get a burst of length b. It can happen that two paths are equally close to the received sequence at the same time and thus the events are not disjoint. So the probability P_b that we go into any one of these (n-1) paths is bounded by

$$P_b \leq \sum_{j=1}^{n-1} P_{bj}$$

using the upper bound on P_{bj} given by expression (5) we get

$$P_b < \sum_{j=1}^{n-1} 2^{w_j} [\varepsilon(1-\varepsilon)]^{w_j/2}$$

P_b gives us the probability that a burst of length b will occur.

If we transmit N channel bits through a BSC with probability of error = ε, we can get at the most $\frac{2N\varepsilon}{w_s}$ loops of the form in Figure 14. Where w_s is the smallest weight among w_1, $w_2, \ldots w_{n-1}$ which gives us a loose bound.

$$N_b \leq \frac{2N\varepsilon}{w_s} \qquad \text{if } w_s/2 = \text{integer}$$

$$N_b \leq \frac{N\varepsilon}{w_s'} \qquad \text{if } w_s/2 \neq \text{integer}$$

w_s' = the integer part of $(w_s/2+1)$

as explained before this can be tightened by weighting it by P_b.

Theorem 2:

The upper bound on the number of non-solid bursts at the output of a Viterbi decoder is:

$$N_b \leq \frac{2N\varepsilon \sum_{j=1}^{n-1} 2^{w_j} [\varepsilon(1-\varepsilon)]^{w_j/2}}{w_s} \qquad \text{if } w_s/2 = \text{integer}$$

$$N_b \leq \frac{N\varepsilon \sum_{j=1}^{n-1} 2^{w_j} [\varepsilon(1-\varepsilon)]^{w_j/2}}{w_s'}$$

if $w_s/2 \neq$ integer

where

N_b = Number of non-solid bursts of length b.

N = Number of channel bits transmitted.

n = Number of paths of length (b+k-1).

w_j = Weight of jth path out of all n paths except the one giving solid bursts.

ε = Probability of error in the BSC.

w_s = Smallest weight among $w_1, w_2 \ldots w_{n-1}$.

w_s' = The integer part of $(w_s/2+1)$.

CHAPTER VI

SIMULATION OF THE SYSTEM

The communication system of Figure 9 was simulated on a digital computer. As the all "zeros" sequence is taken to be the transmitted message, the encoder need not be programmed. The modulo 2 sum of the noise sequence and the all "zeros" sequence will give us the noise sequence and so the received sequence $\bar{r}$ will be nothing but the noise sequence. For example, suppose the noise sequence $\bar{n}$ is given as

$$\bar{n} = 0100111000010\ldots$$

and $$\bar{y} = 0000000000000\ldots$$

then the received sequence $\bar{r}$ is given as

$$\bar{r} = \bar{y} + \bar{n} \pmod 2$$

$$\bar{r} = 0100111000010\ldots$$

which is the noise sequence. So for purposes of simulation the noise sequence for different probabilities of erros in the BSC was produced and stored on a tape. This sequence served as input to the simulated Viterbi decoder.

NOISE SIMULATION

Random noise can be generated in a digital computer by producing pseudo-random numbers. There are various techniques for generation of random numbers. Most of the random number

generators are based on a congruence raltion of the form[5]

$$x_{i+1} = ax_i + c \pmod{m} \qquad i = 0,1,2,3,4,\ldots$$

The sequence of integers $x_1, x_2, x_3, x_4 \ldots$ is determined by the choice of x_0, a, c and m, where these four parameters are non-negative integers. Then the hope is that the sequence x_1/m, $x_2/m \ldots$ will appear to be drawn at random from the uniform distribution on (0,1). If c = 0 the generator is called multiplicative type, otherwise it is called a mixed type. The random number generators of mixed type have their statistical behaviour unsatisfactory in some cases. As the noise is simulated using random numbers it is desirable that the random numbers should have very good random properties.

The random number generator chosen was a multiplicative type. It has the following parameters.[3]

$x_0 = 47594118$

a = 23

c = 0

m = 10001009

The formula has passed the simple distribution and auto-correlation tests.[3]

SIMULATION OF BINARY SYMMETRIC CHANNEL

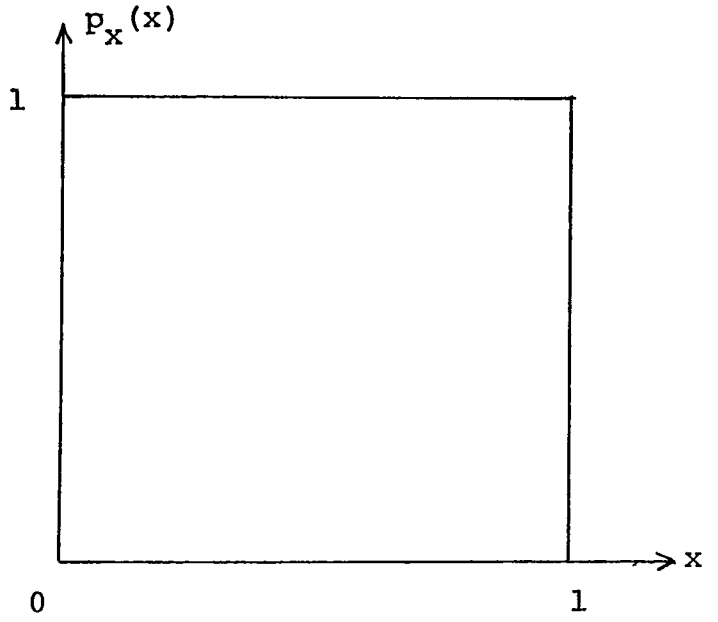
A Binary Symmetric Channel implies a hard decision. Hard

decision means two-level decision, assigning a "zero or a "one". The effect of noise in the channel is to reverse the transmitted digits corresponding to a given probability of error in the BSC. For the purpose of simulation we have to decide the positions of the bits in the encoded sequence which have to be reversed. If we send N bits (N is very large), $N\varepsilon$ bits out of this should be in error or reversed, where ε is the probability of error in the BSC. One of the methods used to accomplish this is given below.

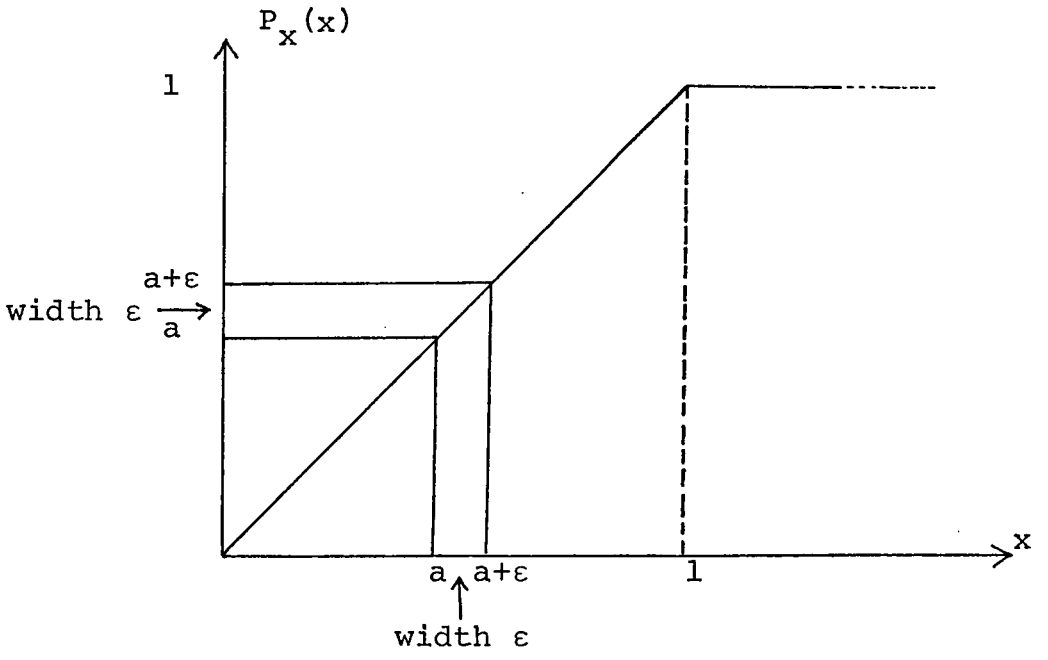
WINDOW METHOD[4]

The random numbers generated are uniformly distributed between 0 and 1. The distribution and density function of such numbers are as shown in Figure 18. If we choose an interval between a and $a+\varepsilon$ where $0<a<1-\varepsilon$, for a true uniform distribution the number of random numbers falling in this interval will be $N\varepsilon$ where N is total number of random numbers generated (N is very large). Also, this number will remain constant no matter where we choose the interval ε between 0 and 1.

In the window method a window is chosen between 0 and 1 and the width of the window is ε which is the probability of error in the BSC. For every input digit sent into the channel a random number is produced. If the random number falls within the window chosen, the digit being transmitted is reversed,



Density function of random numbers uniformly distributed between 0 and 1.



Distribution function of random numbers uniformly distributed between 0 and 1.

Figure 18

otherwise it is left unchanged.

A program for simulation of hard decision noise has been written. The main program uses the subroutine Random in order to produce Random numbers. The flow chart and listing of the program are given in the appendix.

SIMULATION OF VITERBI DECODER

The Viterbi decoding program has been written in FORTRAN IV language. The input to the program consists of the following:

1) Constraint length K.

2) Rate 1/V.

3) The generator sequence.

4) A tape containing the received sequence digits.

For given input values of K and V the program generates the trellis structure of the encoder. It generates all the states and also the sequences on the branches of the trellis structure. These sequences are stored in the memory.

The processing of the input bits is done in blocks. In every block the computer reads 5001 channel bits. It compares these bits, V at a time with the branches on the trellis structure and generates the delta scores. The delta scores are stored in registers. The survivor sequences are stored in survivor sequence registers. Two sets of survivor sequence registers have been provided for every state so that the transfer of contents from one register to another can be

easy. The registers are made 5K bits long so that the delay in decision on bits is 5K. Thus there is no output until 5KV channel bits have been processed. Thereafter, the decoding is bit by bit.

At the end of every block the lowest delta score is subtracted from every delta score register and it controls the overflow of the delta score registers.

A listing of the program and a flow chart are given in the Appendix.

CHAPTER VII

RESULTS AND CONCLUSIONS

For the purpose of simulation systematic and non-systematic codes of constraint length 3 and 5 were considered. The source of these codes were references (9) and (10). Each of the four codes was used with 5 different probabilities of error in the BSC. For each case 100,000 information bits were run and the errors in the decoding process were counted. Probability of error vs signal to noise ratio curves were plotted for each decoder. (See Figures 19 and 20.) The data has also been depicted in Table 1 and Table 2. From Figures 19 and 20 it is observed that non-systematic codes are better than systematic codes. For a constraint length of 3 the gain over systematic code is about 1.2 dB and for K = 5 this gain is about 1.75 dB

The number of bursts in the decoding for each case was counted and also the number of bursts by upper bounds were calculated. These have been shown in Tables 3 through 6. The bounds seem to hold in every case.

Number of bursts vs burst length, diagrams, have been plotted for all decoders in Figures 21 through 29.

The simulation results show that at low probability of error or high signal to noise ratio the non-systematic codes

have very less errors as compared to the systematic codes and the number of bursts is also less. As we go to low signal to noise ratio the total number of errors for the non-systematic codes, of course, remains less than the systematic codes, but the number of longer bursts increases. So at low signal to noise ratio the non-systematic codes give more number of longer bursts than the systematic codes. The number of non-solid bursts also increases as compared to systematic codes.

The upper bounds derived in this work is the first attempt of its kind in the literature. They seem to be reasonably tight. However, it is to be noted that the results are valid only if the number of transmitted bits is very large. Also the application of these bounds is good for short constraint lengths. For large constraint lengths the computation of expression (3) and expression (4) becomes a laborious job.

TABLE 1

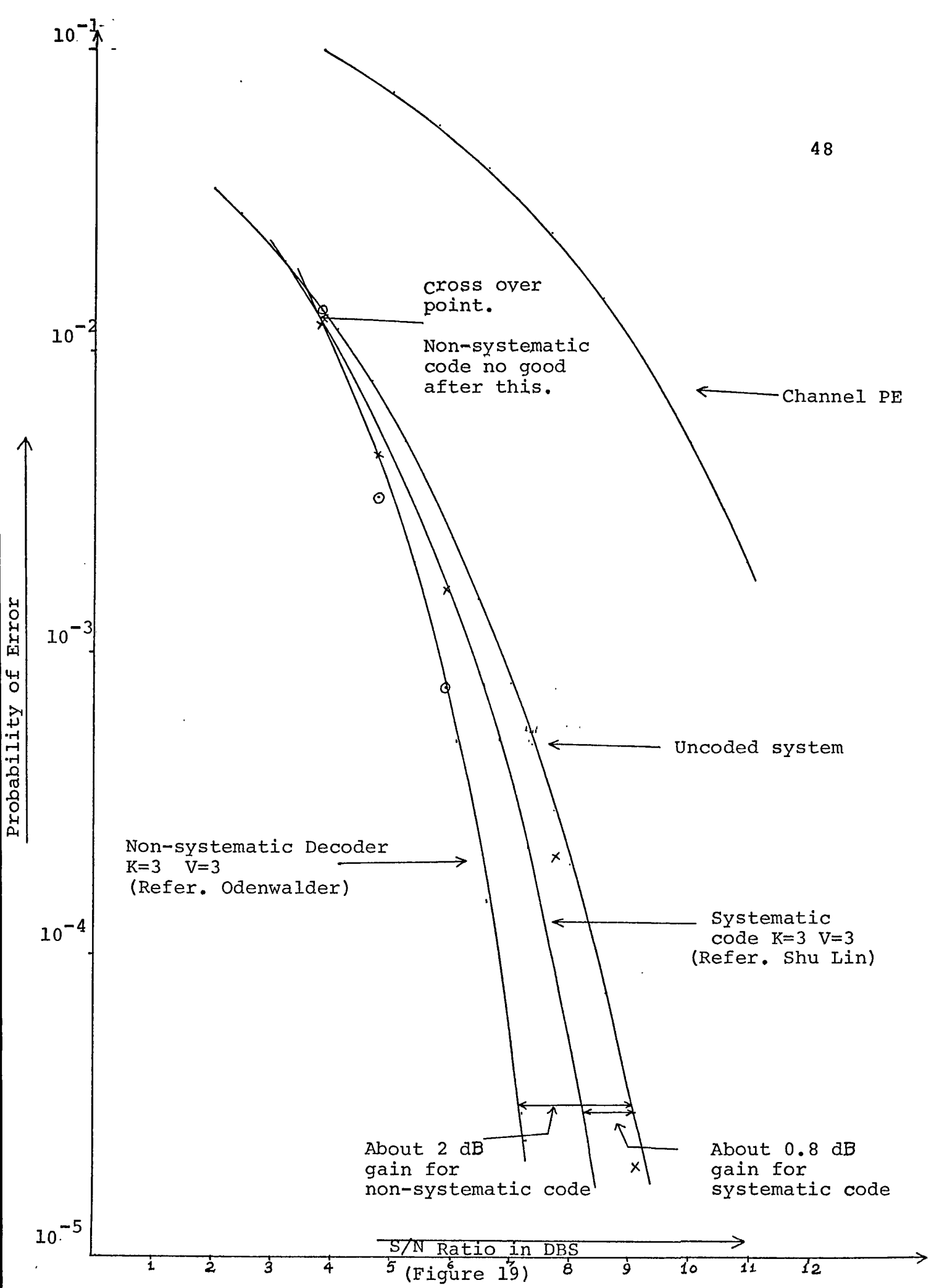
Data for Simulation Study of Systematic and Non-systematic Decoders of Constraint Length 3

Probability of error in BSC P_e	S/N Ratio for the information bits	No. of channel bits transmitted	No. of information bits transmitted	Errors in decoding	Probability of error for the information bits
DECODER SYSTEMATIC K = 3 V = 3 (G.S. 111,001,010)					
0.01	9.1 dB	300,000	100,000	2	2×10^{-5}
0.025	7.78 dB	300,000	100,000	21	2.1×10^{-4}
0.05	5.9 dB	300,000	100,000	162	1.62×10^{-3}
0.075	4.78 dB	300,000	100,000	452	4.54×10^{-3}
0.10	3.8 dB	300,000	100,000	1217	1.217×10^{-2}
DECODER NON-SYSTEMATIC K = 3 V = 3 (G.S. 111,110,111)					
0.01	9.1 dB	300,000	100,000	0	--
0.025	7.78 dB	300,000	100,000	0	--
0.050	5.9 dB	300,000	100,000	77	7.7×10^{-4}
0.075	4.78 dB	300,000	100,000	329	3.29×10^{-3}
0.10	3.8 dB	300,000	100,000	1358	1.358×10^{-2}

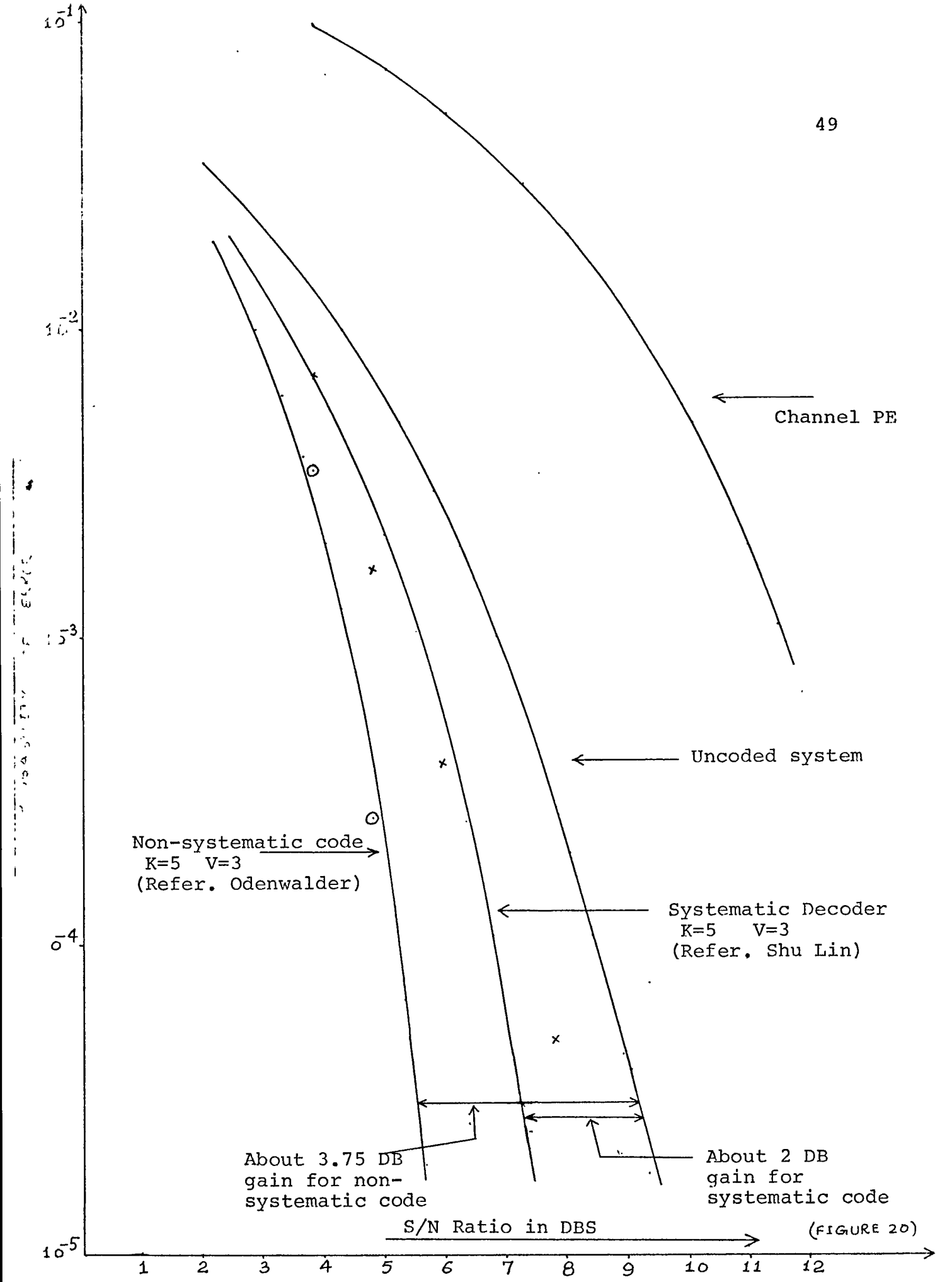
TABLE 2

Data for the Simulation Study of Systematic and Non-systematic Decoders of Constraint Length 5

Probability of error in BSC P_e	S/N Ratio for the information bits	No. of channel bits transmitted	No. of information bits transmitted	Errors in decoding	Probability of error for the information bits
DECODER SYSTEMATIC K = 5 V = 3 (G.S. 111,001,010,010,0001)					
0.01	9.1 dB	300,000	100,000	0	--
0.025	7.78 dB	300,000	100,000	5	5×10^{-5}
0.05	5.9 dB	300,000	100,000	39	3.9×10^{-4}
0.075	4.78dB	300,000	100,000	167	1.67×10^{-3}
0.10	3.8 dB	300,000	100,000	708	7.08×10^{-3}
DECODER NON-SYSTEMATIC K = 5 V = 3 (G.S. 111,110,101,110,111)					
0.01	9.1 dB	300,000	100,000	0	--
0.025	7.78 dB	300,000	100,000	0	--
0.05	5.9 dB	300,000	100,000	0	--
0.075	4.78 dB	300,000	100,000	26	2.6×10^{-4}
0.10	3.8 dB	300,000	100,000	350	3.50×10^{-3}


10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
Probability of Error
cross over point.
Non-systematic code no good after this.
Channel PE
Uncoded system
Non-systematic Decoder K=3 V=3 (Refer. Odenwalder)
Systematic code K=3 V=3 (Refer. Shu Lin)
About 2 dB gain for non-systematic code
About 0.8 dB gain for systematic code
S/N Ratio in DBS
1
2
3
4
5
6
7
8
9
10
11
12

(Figure 19)


10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
Channel PE
Uncoded system
Non-systematic code
K=5 V=3
(Refer. Odenwalder)
Systematic Decoder
K=5 V=3
(Refer. Shu Lin)
About 3.75 DB
gain for non-
systematic code
About 2 DB
gain for
systematic code
S/N Ratio in DBS
1 2 3 4 5 6 7 8 9 10 11 12

(FIGURE 20)

TABLE 3

Comparison of Upper Bounds to the No. of Bursts in the Simulation Study for Systematic Decoder of Constraint Length 3

SYSTEMATIC DECODER

K = 3 V = 3

Length of burst	Type of burst	Number of bursts in simulation	No. of bursts by upper bounds
		$\varepsilon = 0.01$	
1	Solid	2	3
		$\varepsilon = 0.025$	
1	Solid	21	29
		$\varepsilon = 0.05$	
1	Solid	125	157
2	Solid	3	5
3	Solid	2	3
4	Solid	2	1
3	Non Solid	2	5
4	Non Solid	2	2
5	Non Solid	1	1

Table 3 Continued

SYSTEMATIC DECODER

K = 3 V = 3

Length of burst	Type of burst	Number of bursts in simulation	No. of bursts by upper bounds
		ε = 0.075	
1	Solid	352	369
2	Solid	6	30
3	Solid	9	14
4	Solid	4	8
5	Solid	2	3
6	Solid	1	2
7	Solid	1	1
3	Non-Solid	1	1
4	Non-Solid	5	30
5	Non-Solid	2	7

SYSTEMATIC DECODER

K = 3 V = 3

		ε = 0.10	
1	Solid	630	777
2	Solid	30	110
3	Solid	34	60
4	Solid	8	36
5	Solid	5	15
6	Solid	1	10
7	Solid	5	6

Table 3 Continued

$\varepsilon = 0.10$ (continued)			
3	Non-Solid	16	110
4	Non-Solid	7	34
5	Non-Solid	6	27
6	Non-Solid	6	30
7	Non-Solid	1	15
8	Non-Solid	2	9

TABLE 4

Comparison of Upper Bounds to No. of Bursts in the Simulation Study for Non-Systematic Decoder of Constraint Length 3

NON-SYSTEMATIC DECODER

K = 3 V = 3

Length of burst	Type of burst	No. of bursts in simulation	No. of bursts by upper bounds
		$\varepsilon = 0.05$	
1	Solid	7	8
2	Solid	4	8
4	Solid	1	1
5	Solid	1	1
3	Non-Solid	2	3
4	Non-Solid	6	7
5	Non-Solid	1	4
6	Non-Solid	2	2
7	Non-Solid	1	1
8	Non-Solid	1	1
9	Non-Solid	1	1
		$\varepsilon = 0.075$	
1	Solid	59	62
2	Solid	23	62
3	Solid	7	8

Table 4 Continued

		$\varepsilon = 0.075$ (continued)	
3	Non-Solid	7	8
4	Non-Solid	8	10
5	Non-Solid	6	9
6	Non-Solid	5	6
7	Non-Solid	4	5
8	Non-Solid	3	5
		$\varepsilon = 0.10$	
1	Solid	168	181
2	Solid	77	181
3	Solid	16	60
4	Solid	2	36
5	Solid	2	16
3	Non-Solid	27	56
4	Non-Solid	41	45
5	Non-Solid	32	38
6	Non-Solid	17	30
7	Non-Solid	21	30
8	Non-Solid	8	22
9	Non-Solid	7	15
10	Non-Solid	9	11
11	Non-Solid	5	10
12	Non-Solid	1	5

TABLE 5

Comparison of Upper Bounds to the No. of Bursts in the Simulation Study for a Systematic Decoder of Constraint Length 5

Length of burst	Type of burst	No. of bursts in simulation	No. of bursts by upper bounds
SYSTEMATIC DECODER K = 5 V = 3			
$\varepsilon = 0.025$			
1	Solid	5	6
$P_e = 0.05$			
1	Solid	30	58
3	Solid	2	2
4	Solid	1	1
$P_e = 0.075$			
1	Solid	94	132
2	Solid	3	7
3	Solid	5	7
4	Solid	1	1
4	Non-Solid	5	5
5	Non-Solid	1	3
6	Non-Solid	3	3
7	Non-Solid	1	2
9	Non-Solid	1	1

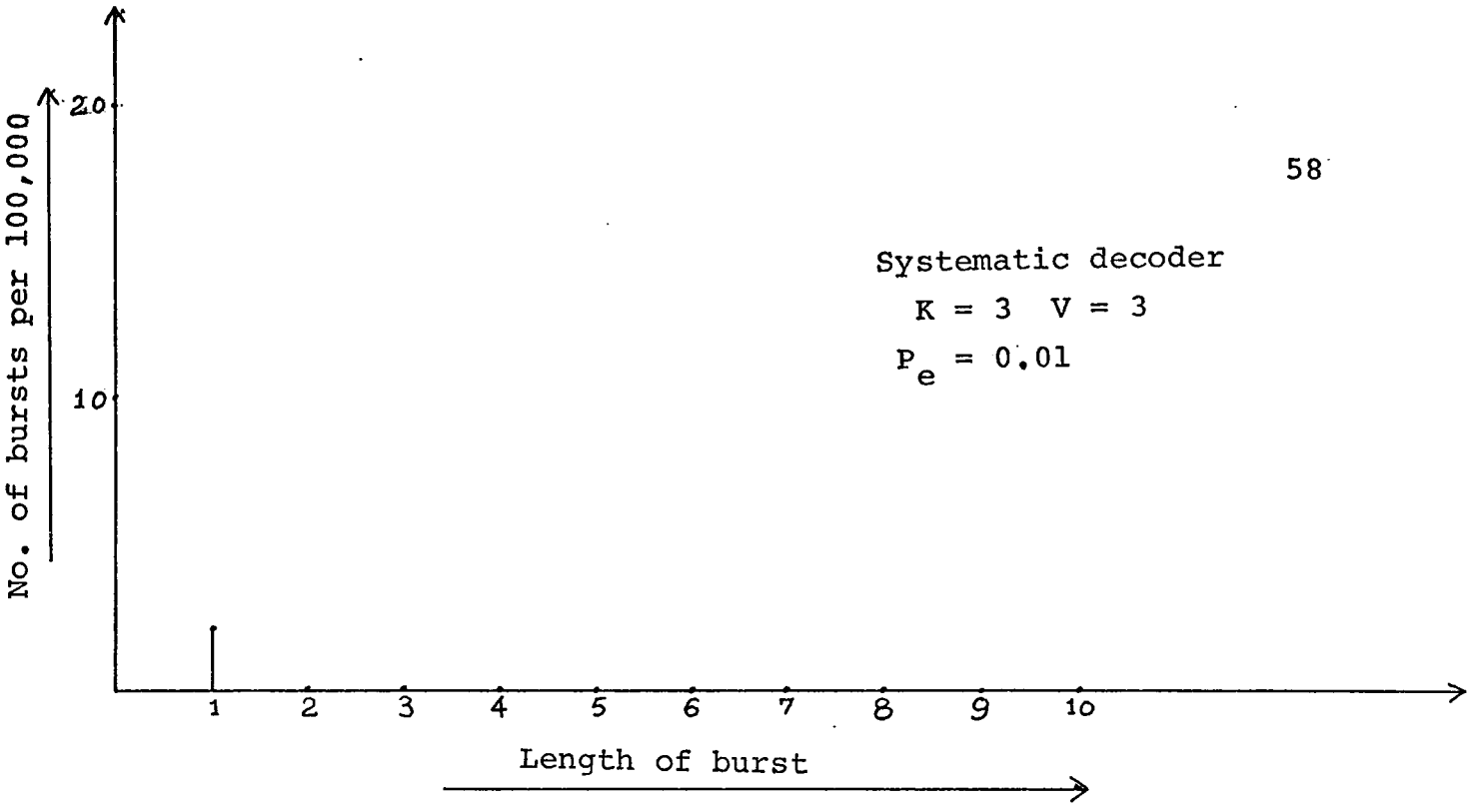
Table 5 Continued

SYSTEMATIC DECODER K=5 V=3			
Length of burst	Type of burst	No. of bursts in simulation	No. of bursts by upper bounds
		$\varepsilon = 0.10$	
1	Solid	336	348
2	Solid	14	36
3	Solid	19	36
4	Solid	3	15
4	Non-Solid	25	32
5	Non-Solid	18	26
6	Non-Solid	11	21
7	Non-Solid	12	19
8	Non-Solid	7	15
9	Non-Solid	9	12

TABLE 6

Comparison of Upper Bounds to the No. of Bursts in the Simulation Study for a Non-Systematic Decoder of Constraint Length 5

Non-Systematic Decoder K = 5 V = 3			
Length of burst	Type of burst	No. of bursts in simulation	No. of bursts by upper bounds
		$\varepsilon = 0.075$	
1	Solid	4	5
2	Solid	3	5
3	Solid	1	5
7	Non-Solid	2	2
		$P_e = 0.10$	
1	Solid	10	12
2	Solid	11	12
3	Solid	6	12
4	Solid	5	12
3	Non-Solid	8	12
4	Non-Solid	1	3
5	Non-Solid	8	10
6	Non-Solid	3	5
7	Non-Solid	4	7
8	Non-Solid	9	10
9	Non-Solid	2	10
10	Non-Solid	6	11
11	Non-Solid	4	11


Systematic decoder
K = 3 V = 3
P_e = 0.01
No. of bursts per 100,000
20
10
1 2 3 4 5 6 7 8 9 10
Length of burst

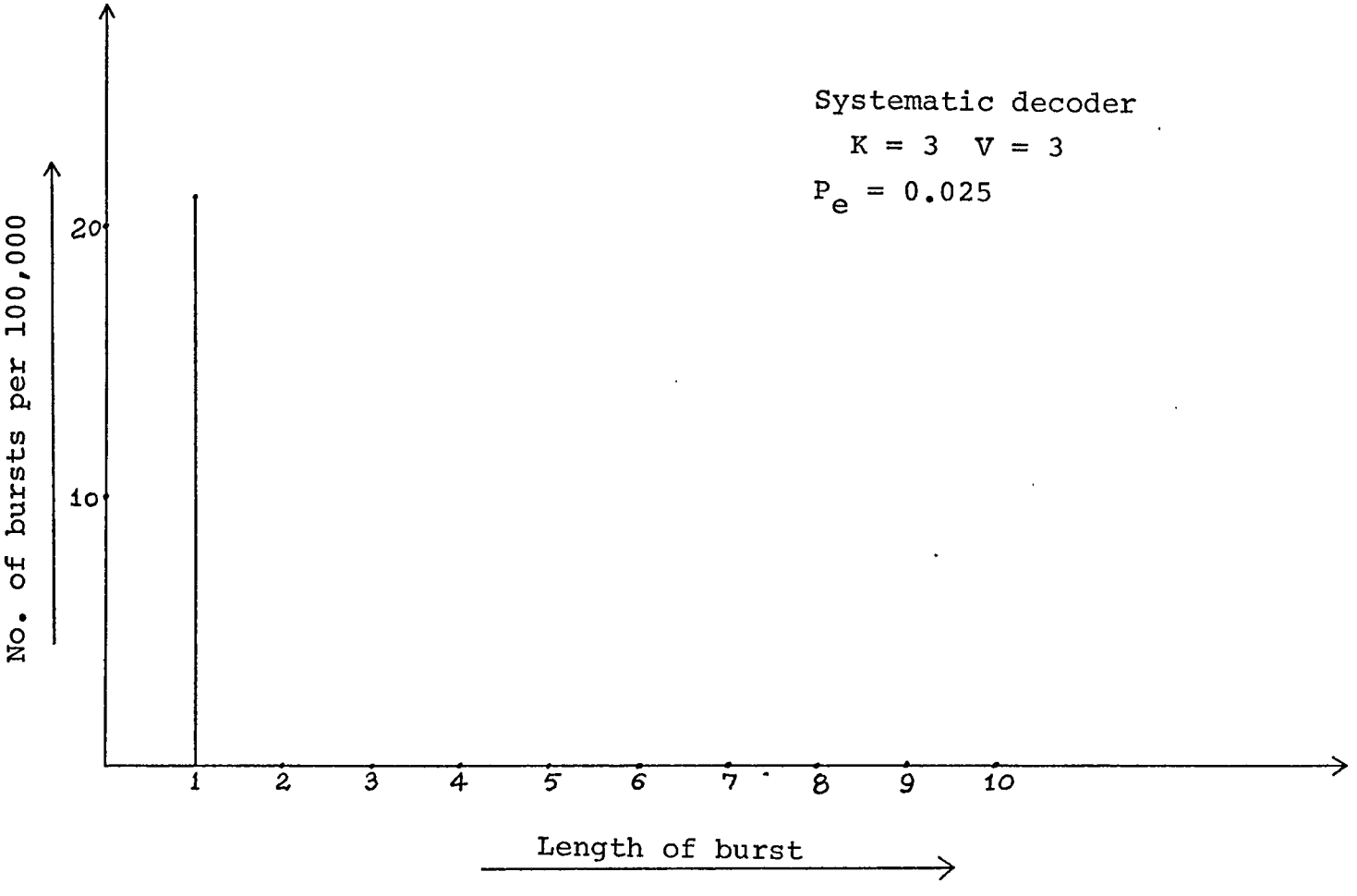
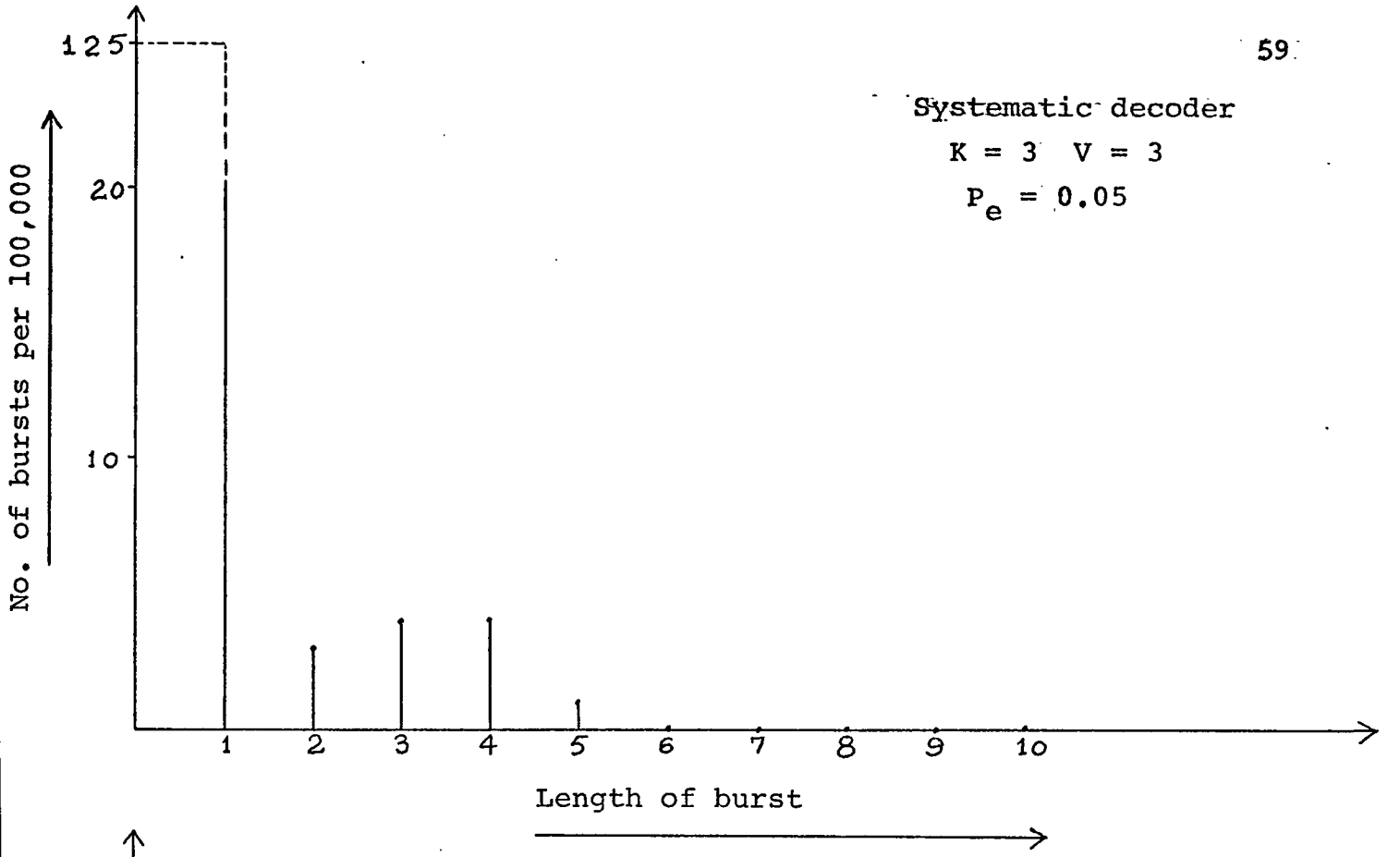

Systematic decoder
K = 3 V = 3
P_e = 0.025
No. of bursts per 100,000
20
10
1 2 3 4 5 6 7 8 9 10
Length of burst

Figure 21


125
20
10
No. of bursts per 100,000
Systematic decoder
K = 3 V = 3
P_e = 0.05
1 2 3 4 5 6 7 8 9 10
Length of burst

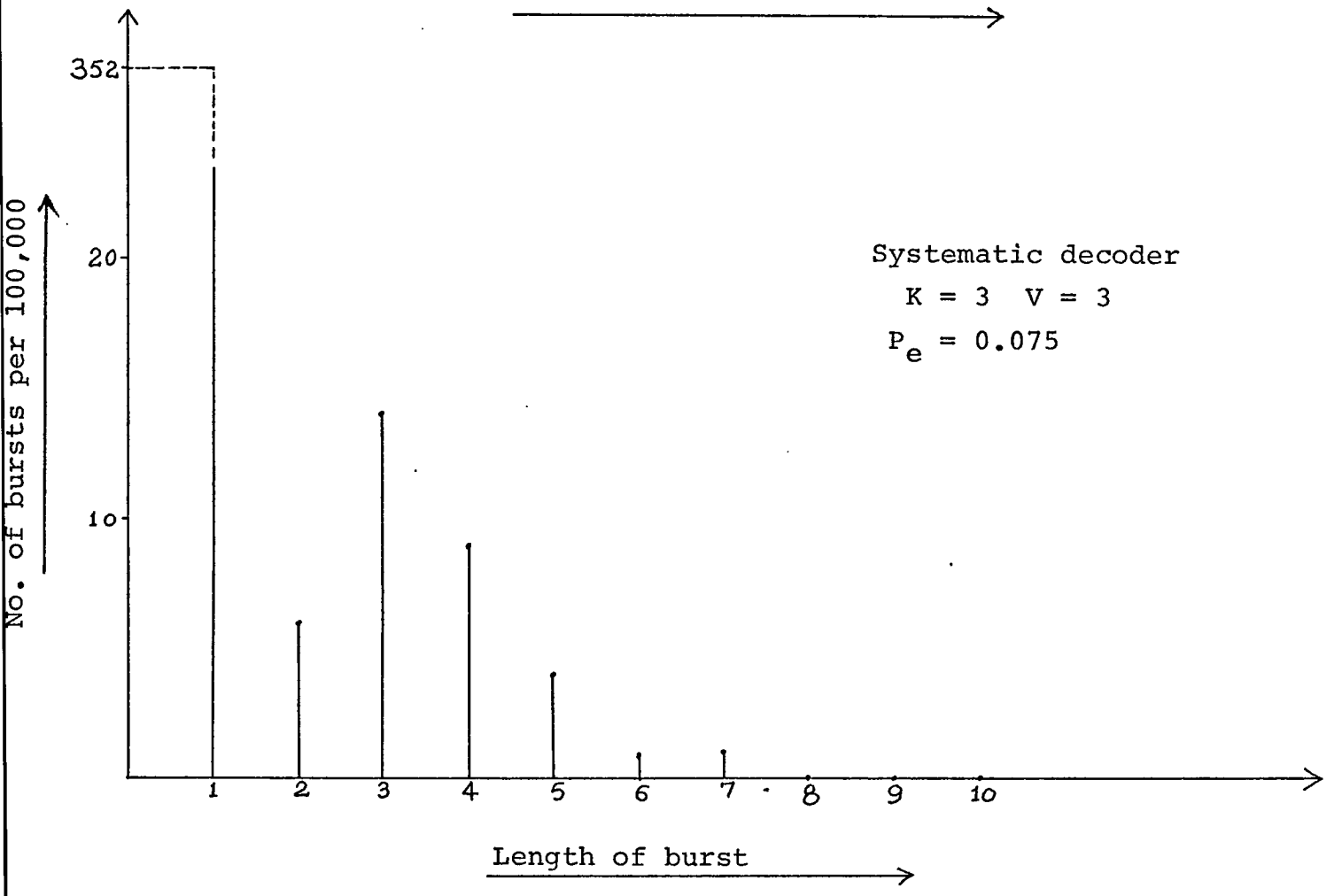

352
20
10
No. of bursts per 100,000
Systematic decoder
K = 3 V = 3
P_e = 0.075
1 2 3 4 5 6 7 8 9 10
Length of burst

Figure 22

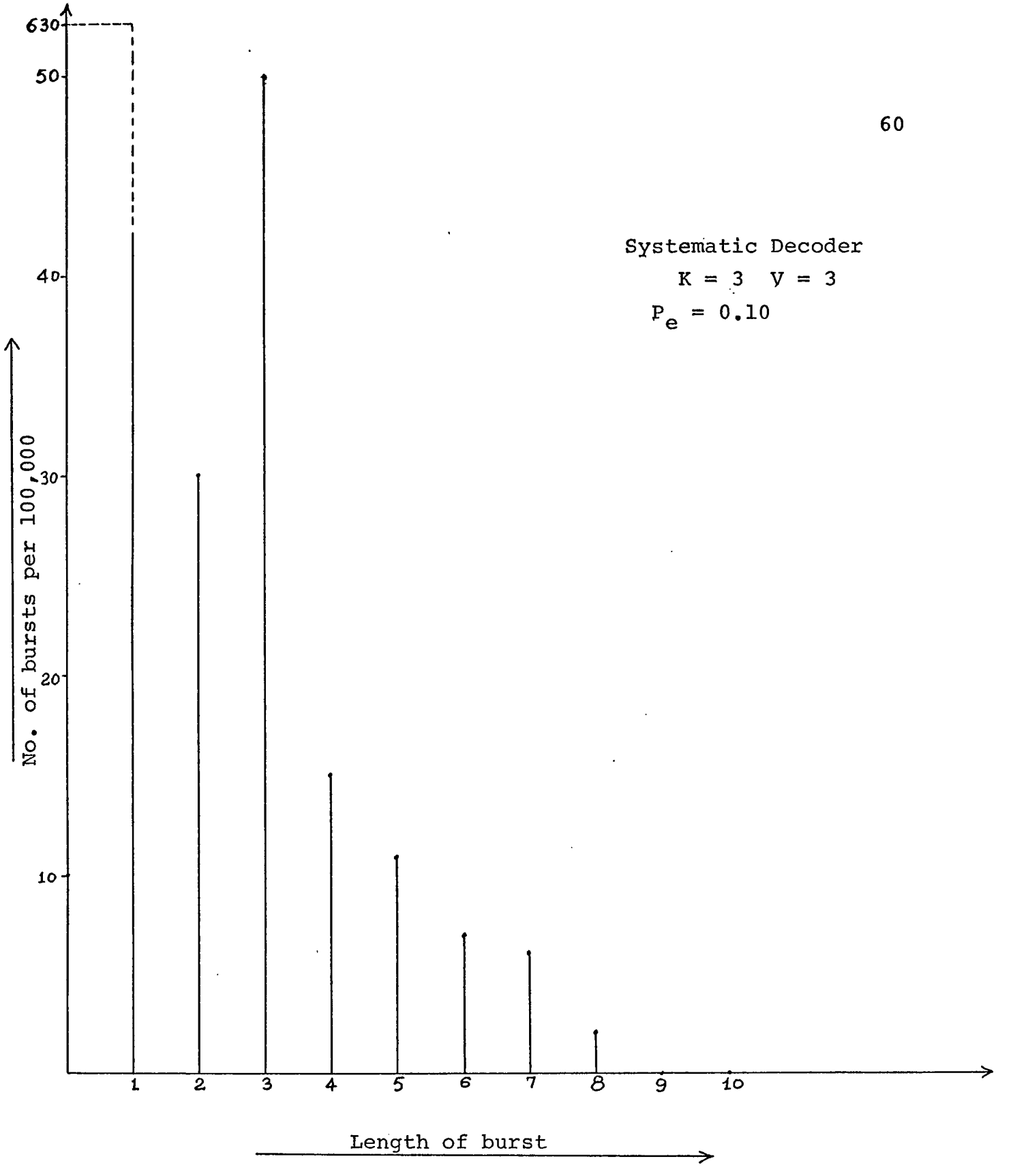
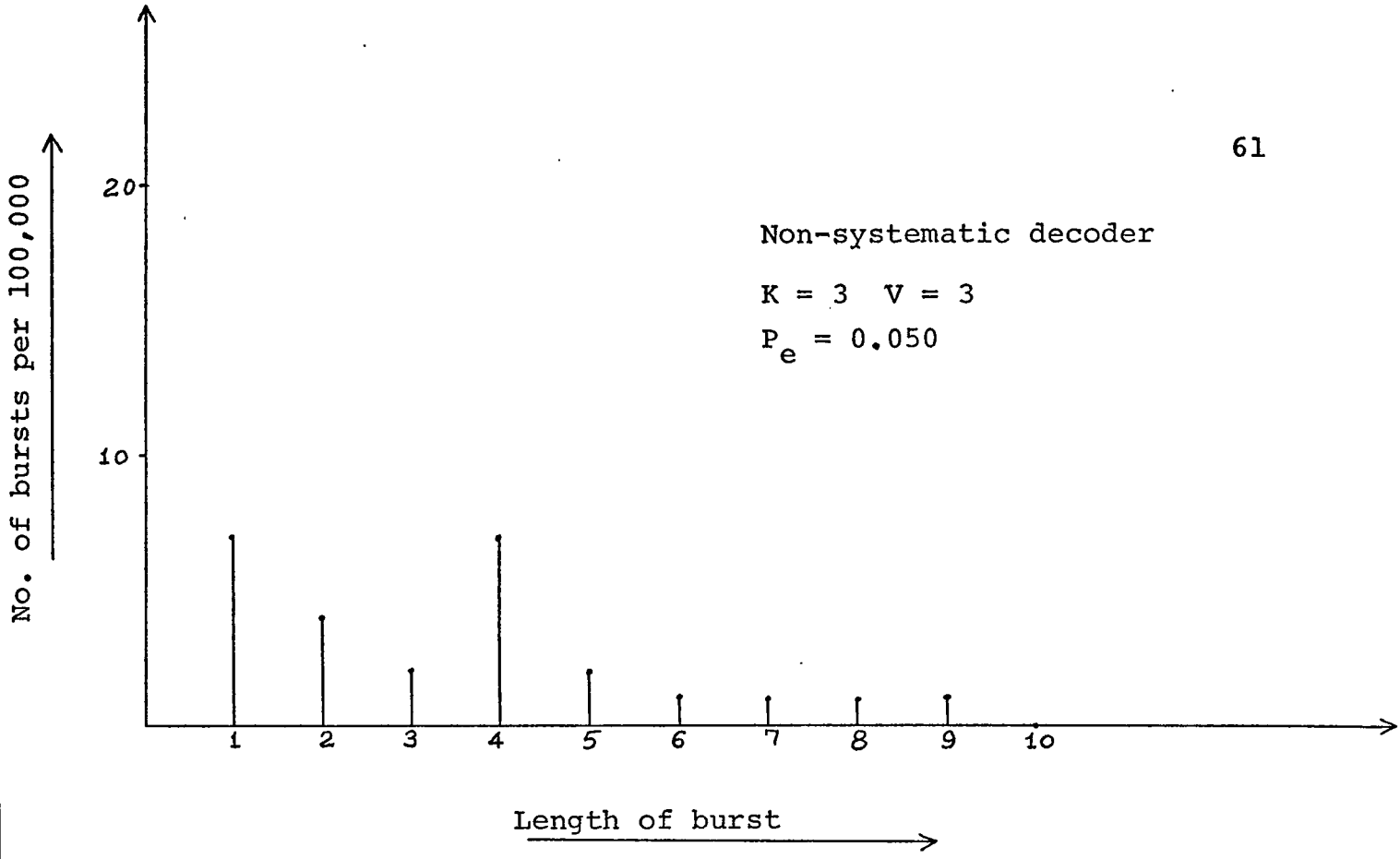

Systematic Decoder
K = 3 Y = 3
P_e = 0.10
630
50
40
30
20
10
No. of bursts per 100,000
1
2
3
4
5
6
7
8
9
10
Length of burst

Figure 23


No. of bursts per 100,000
20
10
Non-systematic decoder
K = 3 V = 3
P_e = 0.050
1 2 3 4 5 6 7 8 9 10
Length of burst

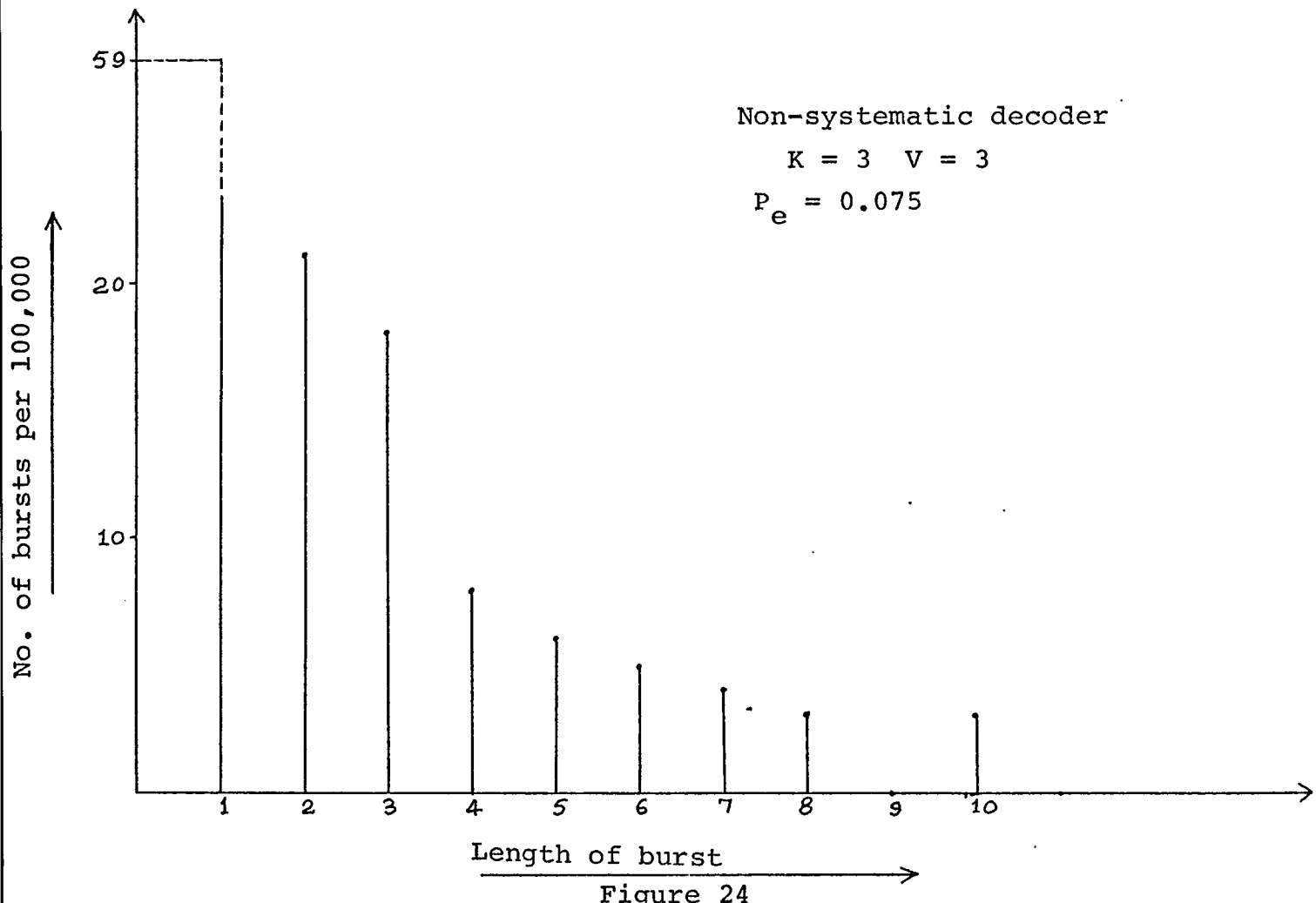

No. of bursts per 100,000
59
20
10
Non-systematic decoder
K = 3 V = 3
P_e = 0.075
1 2 3 4 5 6 7 8 9 10
Length of burst

Figure 24

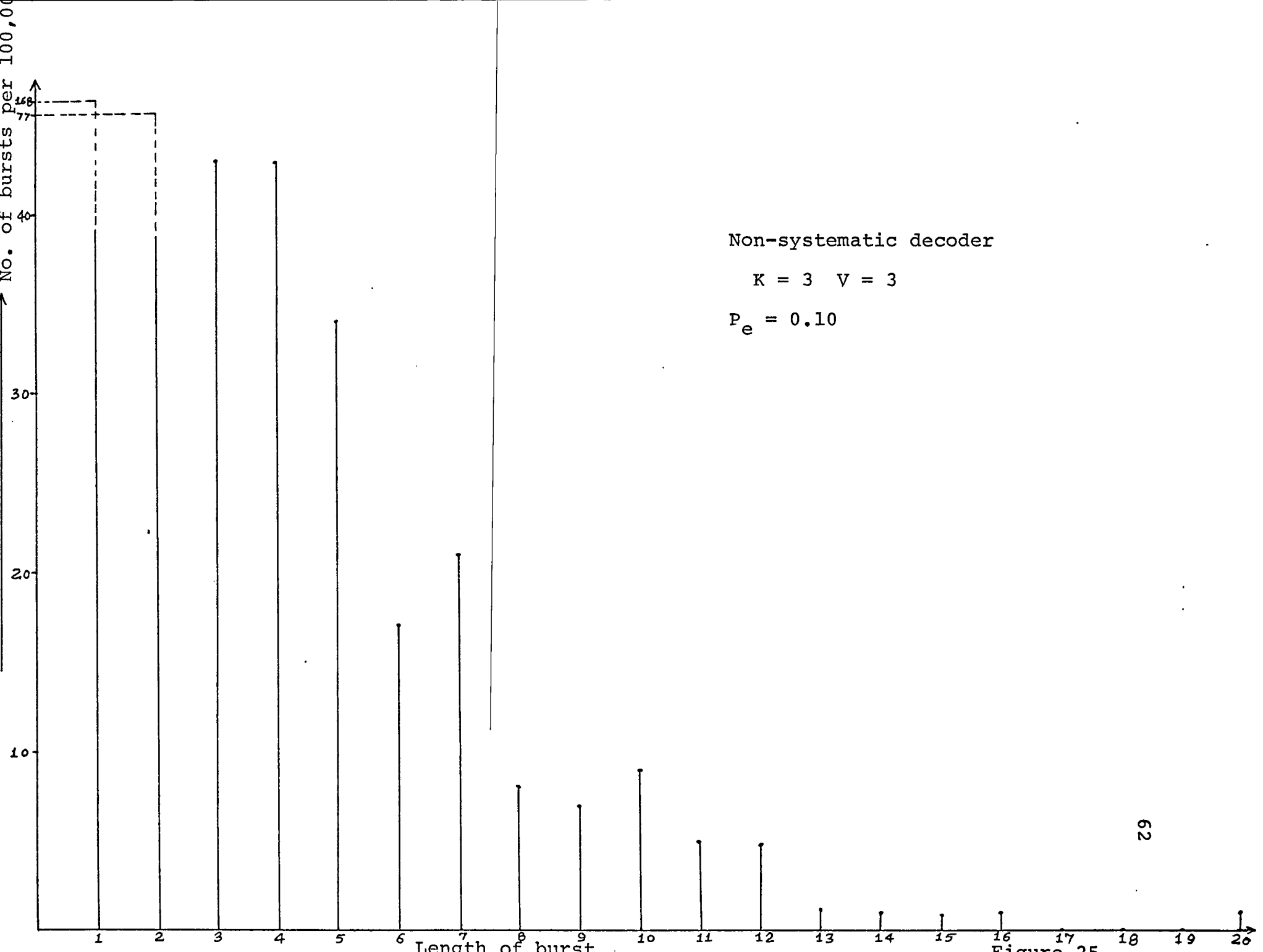
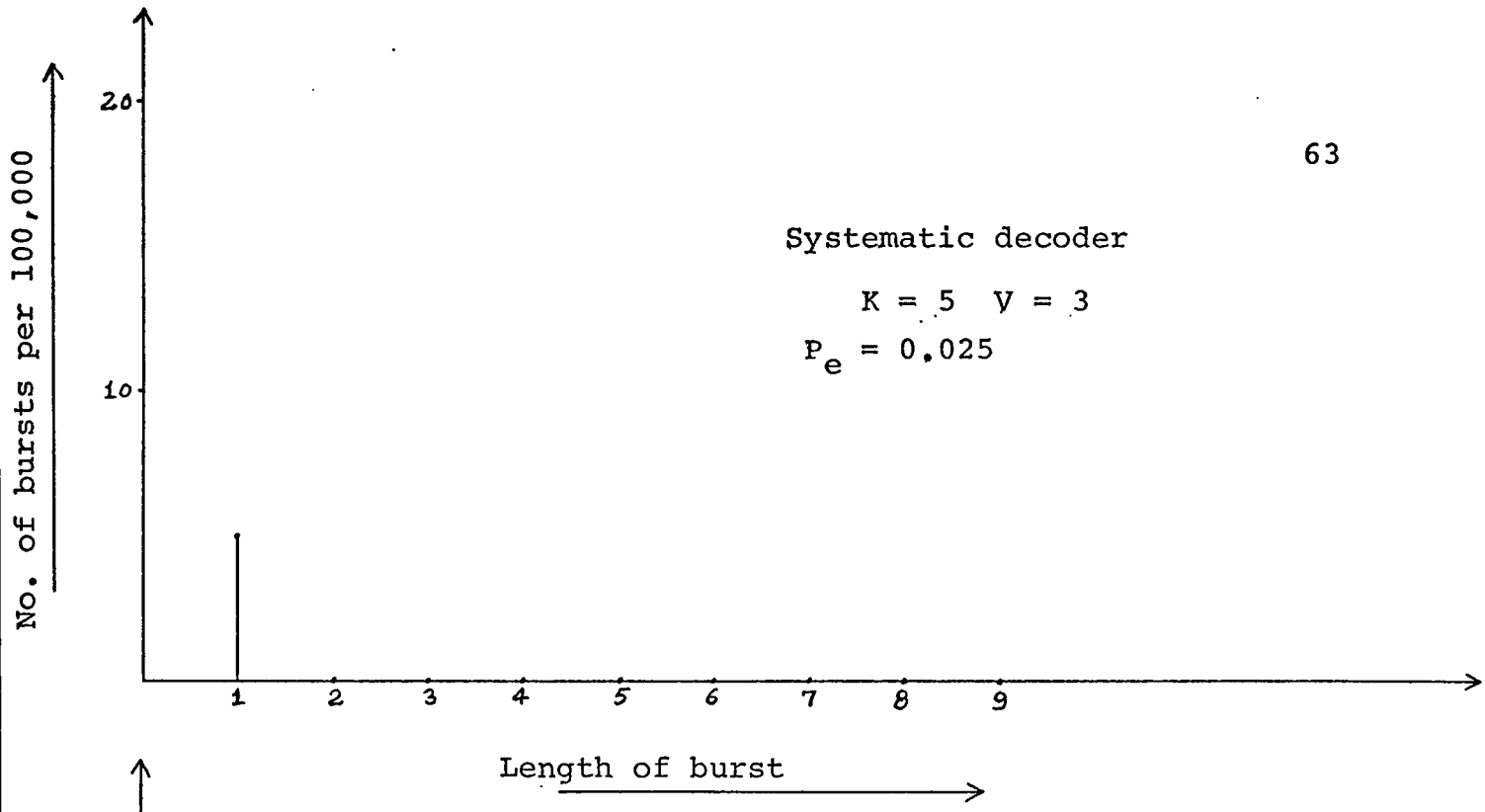

No. of bursts per 100,000
168
77
40
30
20
10
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
Length of burst
Non-systematic decoder
K = 3 V = 3
P_e = 0.10

Figure 25


No. of bursts per 100,000
20
10
Systematic decoder
K = 5 V = 3
P_e = 0.025
1 2 3 4 5 6 7 8 9
Length of burst

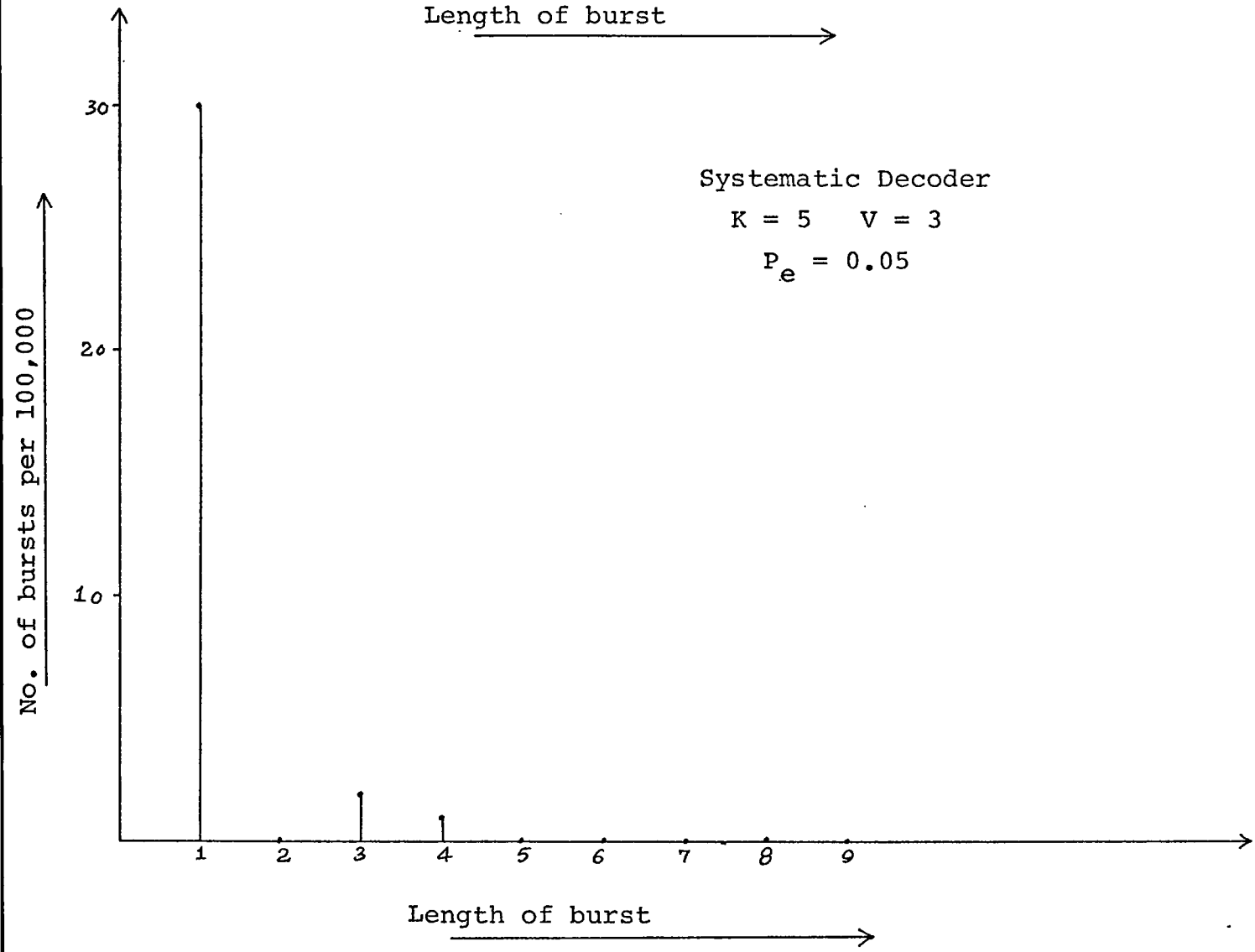
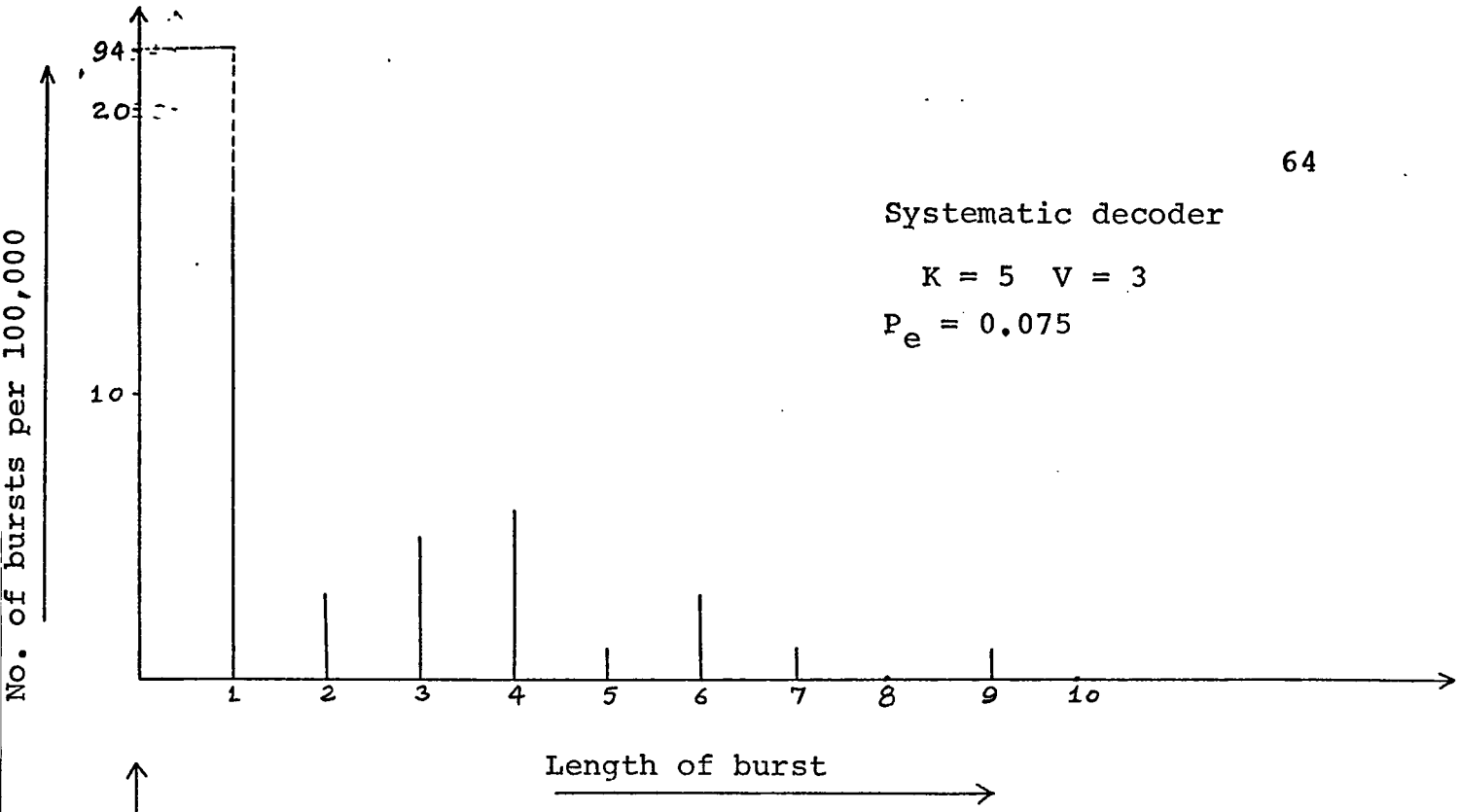

No. of bursts per 100,000
30
20
10
Systematic Decoder
K = 5 V = 3
P_e = 0.05
1 2 3 4 5 6 7 8 9
Length of burst

Figure 26


94
20
10
No. of bursts per 100,000
Systematic decoder
K = 5 V = 3
P_e = 0.075
1 2 3 4 5 6 7 8 9 10
Length of burst

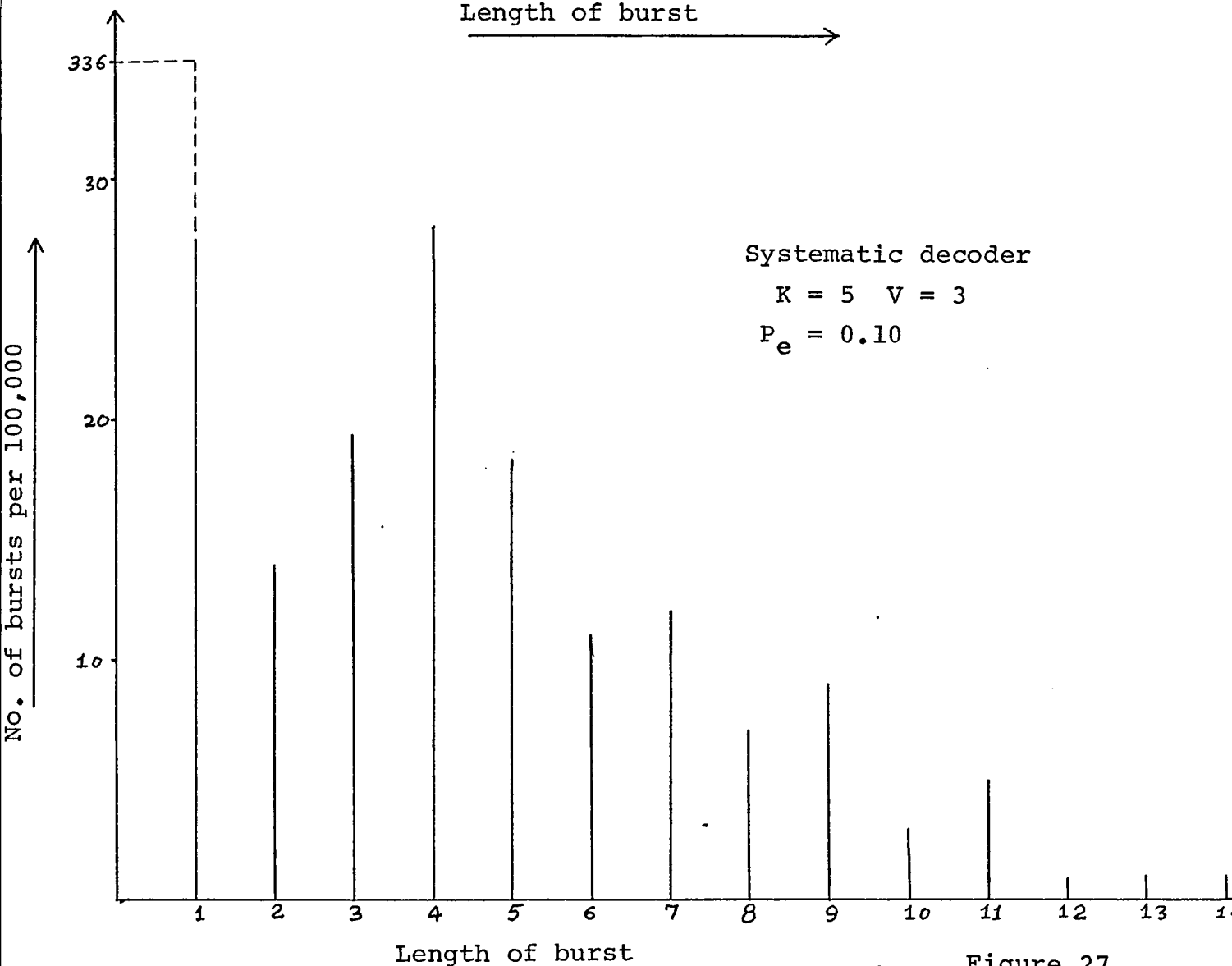

336
30
20
10
No. of bursts per 100,000
Systematic decoder
K = 5 V = 3
P_e = 0.10
1 2 3 4 5 6 7 8 9 10 11 12 13 14
Length of burst

Figure 27

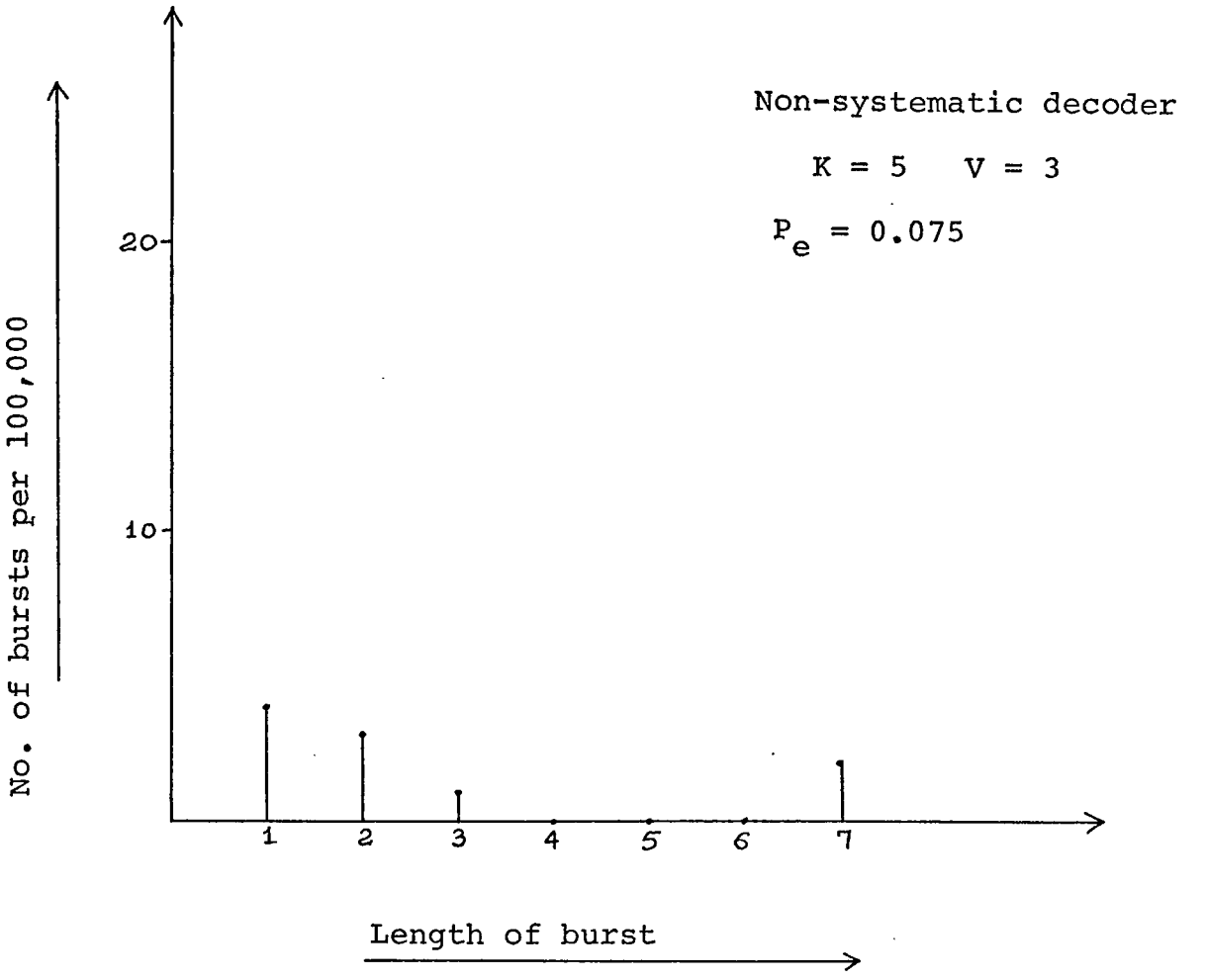

Non-systematic decoder
K = 5 V = 3
P_e = 0.075
No. of bursts per 100,000
20
10
1
2
3
4
5
6
7
Length of burst

Figure 28

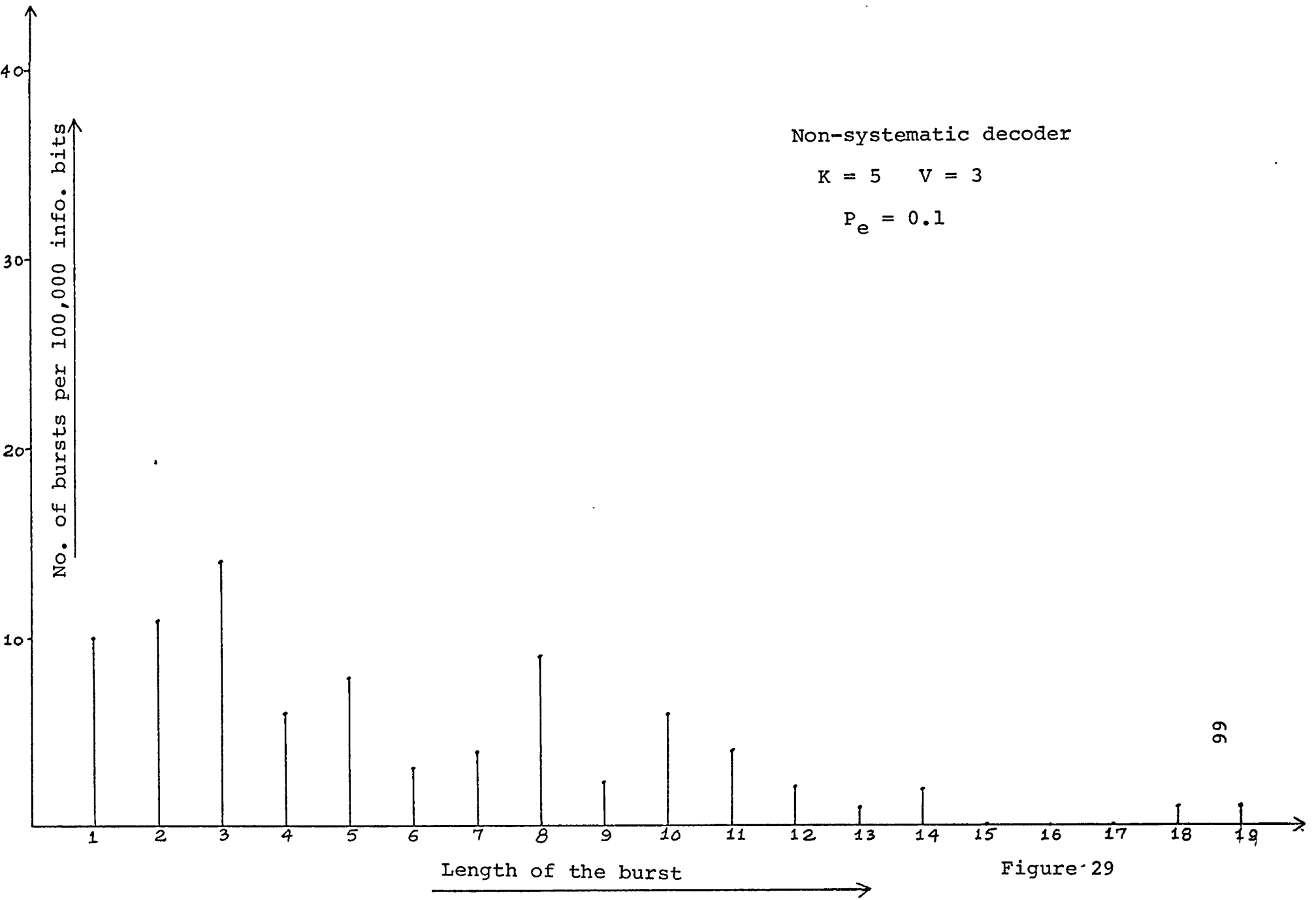

Non-systematic decoder
K = 5 V = 3
P_e = 0.1
No. of bursts per 100,000 info. bits
40
30
20
10
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Length of the burst
99

Figure 29

REFERENCES

1. Batson, B. H., "A Description of the Viterbi Decoding Algorithm," EE70-8008(U), Manned Spacecraft Center, Houston, Texas, May 1970.

2. Gallager, Robert G., "Information Theory and Reliable Communication," John Wiley and Sons, Inc., New York, Chapter 1 and 6, 1968.

3. "Potential Applications of Digital Techniques to Apollo Unified S-Band Communications System," NASA Contract NAS 9-9852, Martin Marietta Corp., Denver, Colorado, February 1970.

4. Taqvi, S. Z. H. and Wai-Hung Ng, "The Simulation of Error-Correction Codes for Space Communication Channels," HASD 642D-823239, Lockheed Electronic Company, pp. 16-18, March 1970.

5. Hull, T. E. and A. R. Dobell, "Mixed Congruential Random Number Generators for Binary Machines," Journal of the Association for Computing Machinery V-11, pp. 31-40, January 1964.

6. Rule, Wilfred P., "Fortran IV Programming," 1968.

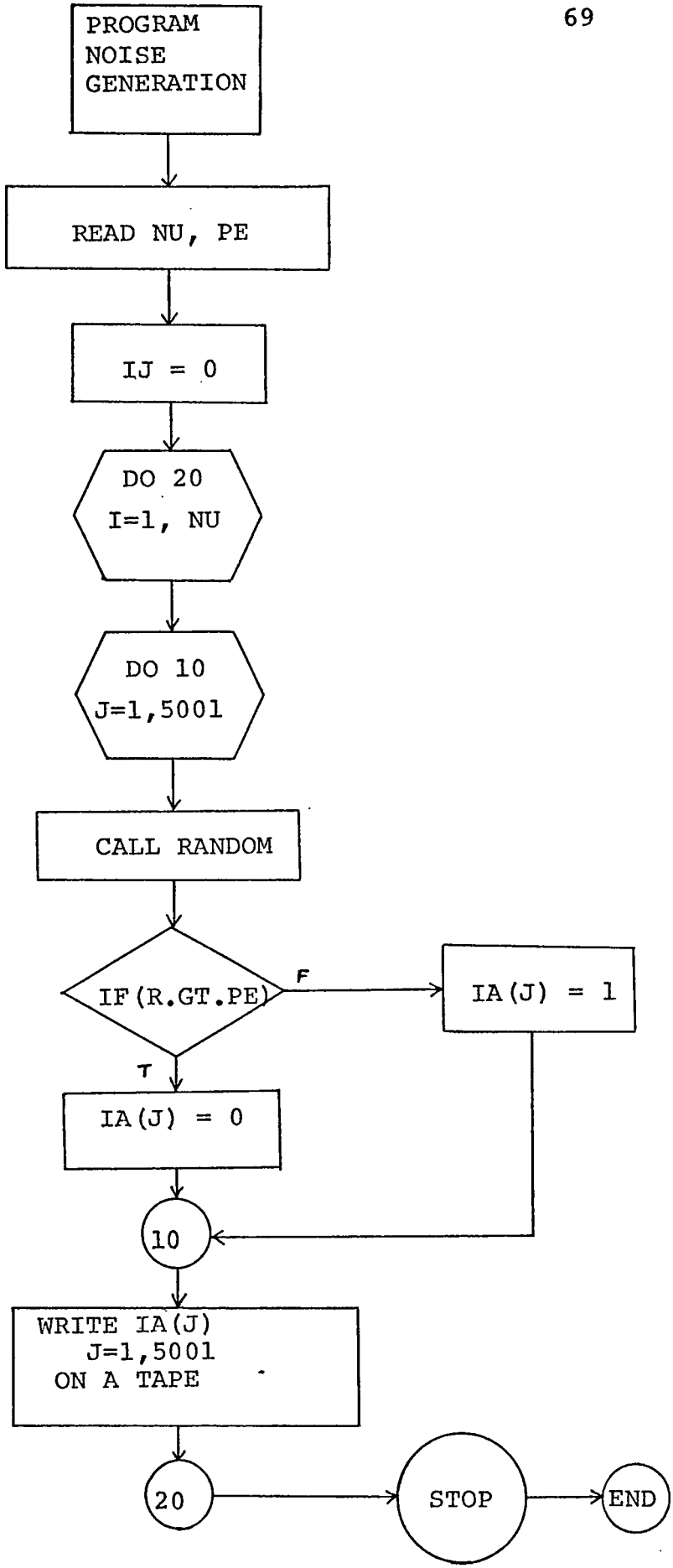
7. Papoulis, A., "Probability, Random Variables, and Stochastic Processes," McGraw-Hill Book Company, New York, Chapter 3, 1965.

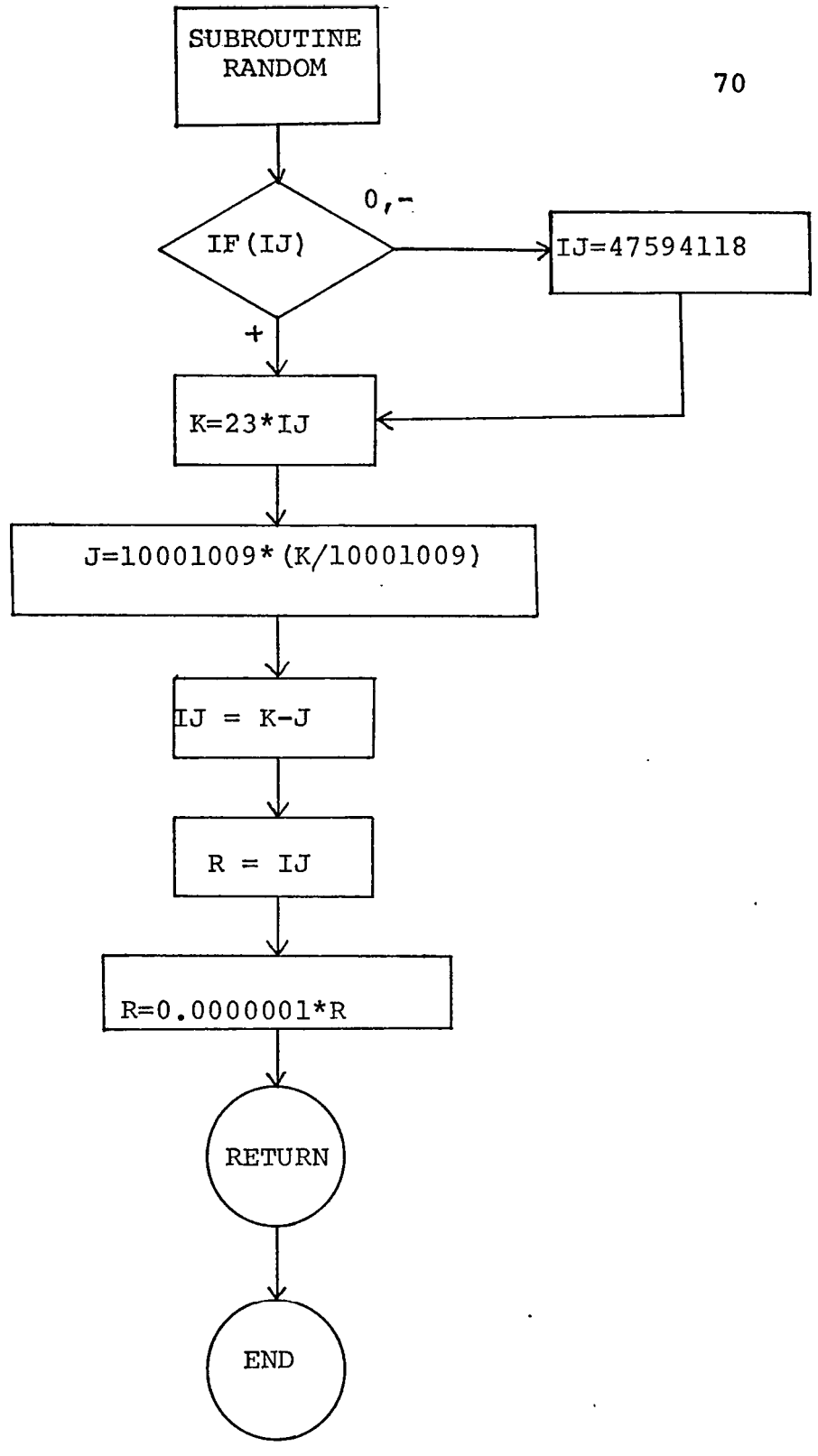
8. Linkabit Corporation, "Convolutional Codes: The Key to Effective Error Control and Efficient Communication", Seminar presented by Linkabit Corporation, 10911 Weyburn Avenue, Los Angeles, California, January 26-29, 1970.

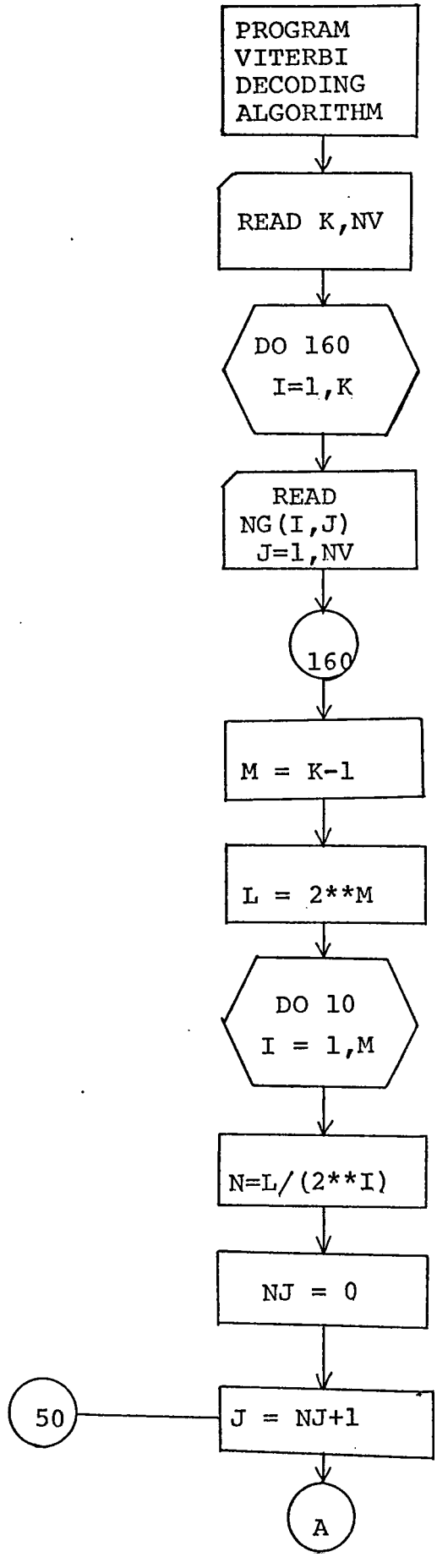
9. Lin, Shu, "Some Results on the Binary Convolutional Tree Code Generators," Department of Electrical Engineering, University of Hawaii, Honolulu, Hawaii.

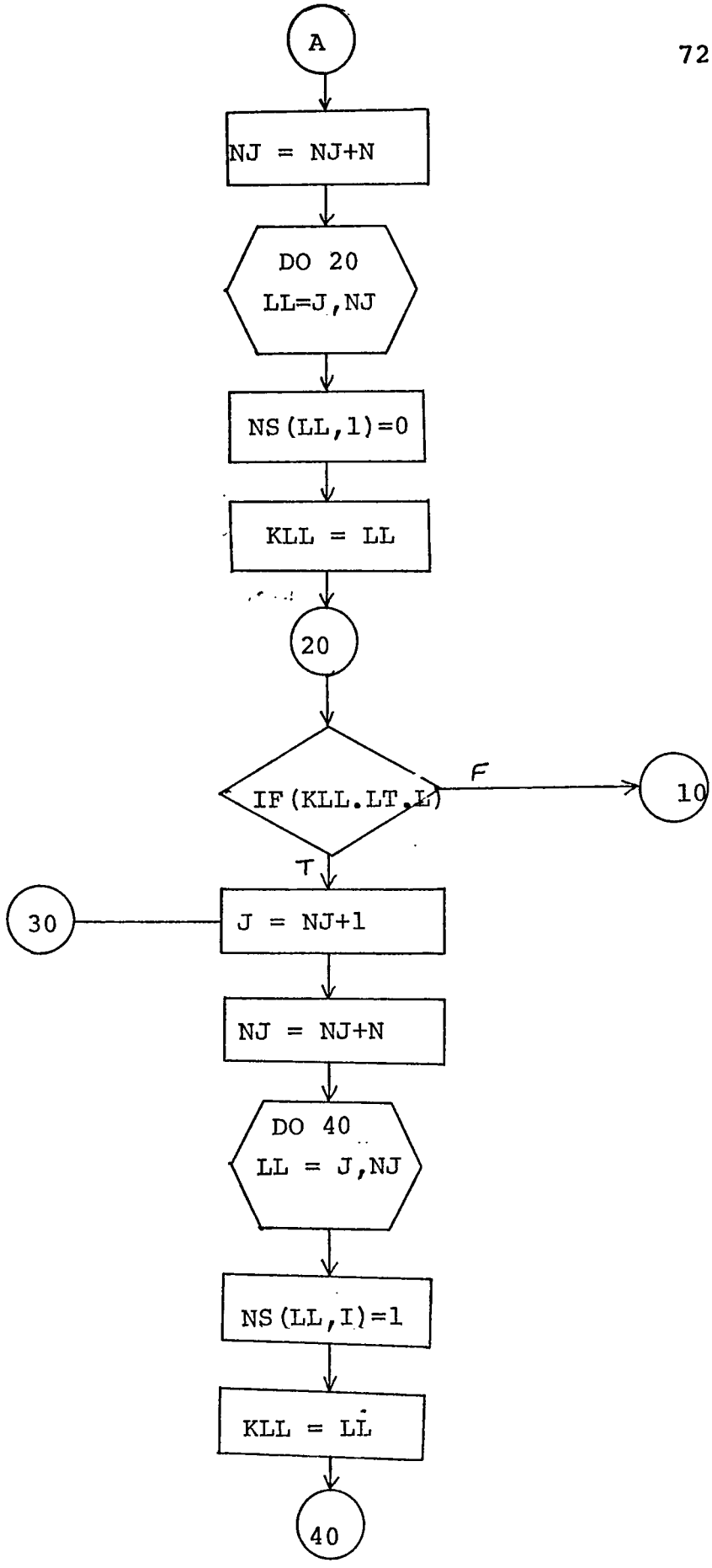
10. Odenwalder, J. P., "Concatenation of Convolutional and Block Codes," Ph.D. Dissertation, UCLA, 1969.

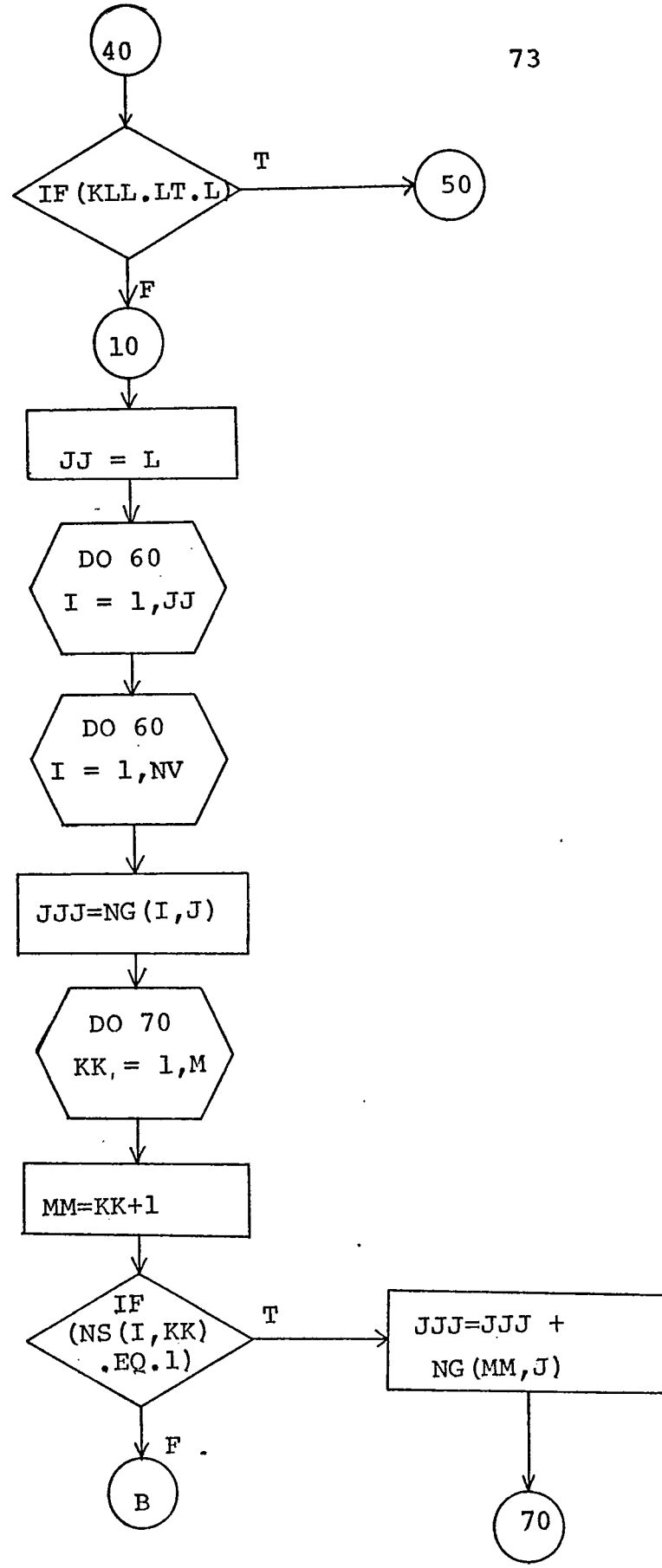
APPENDIX

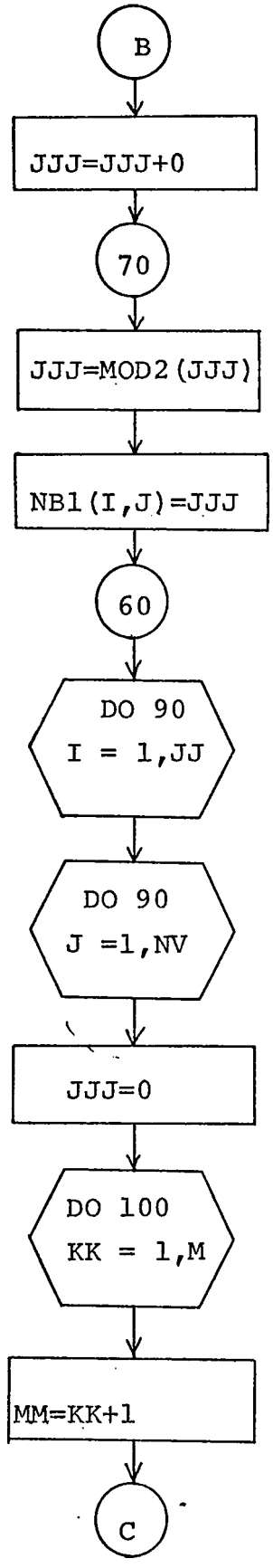

PROGRAM
NOISE
GENERATION
READ NU, PE
IJ = 0
DO 20
I=1, NU
DO 10
J=1,5001
CALL RANDOM
IF(R.GT.PE)
F
IA(J) = 1
T
IA(J) = 0
10
WRITE IA(J)
J=1,5001
ON A TAPE
20
STOP
END

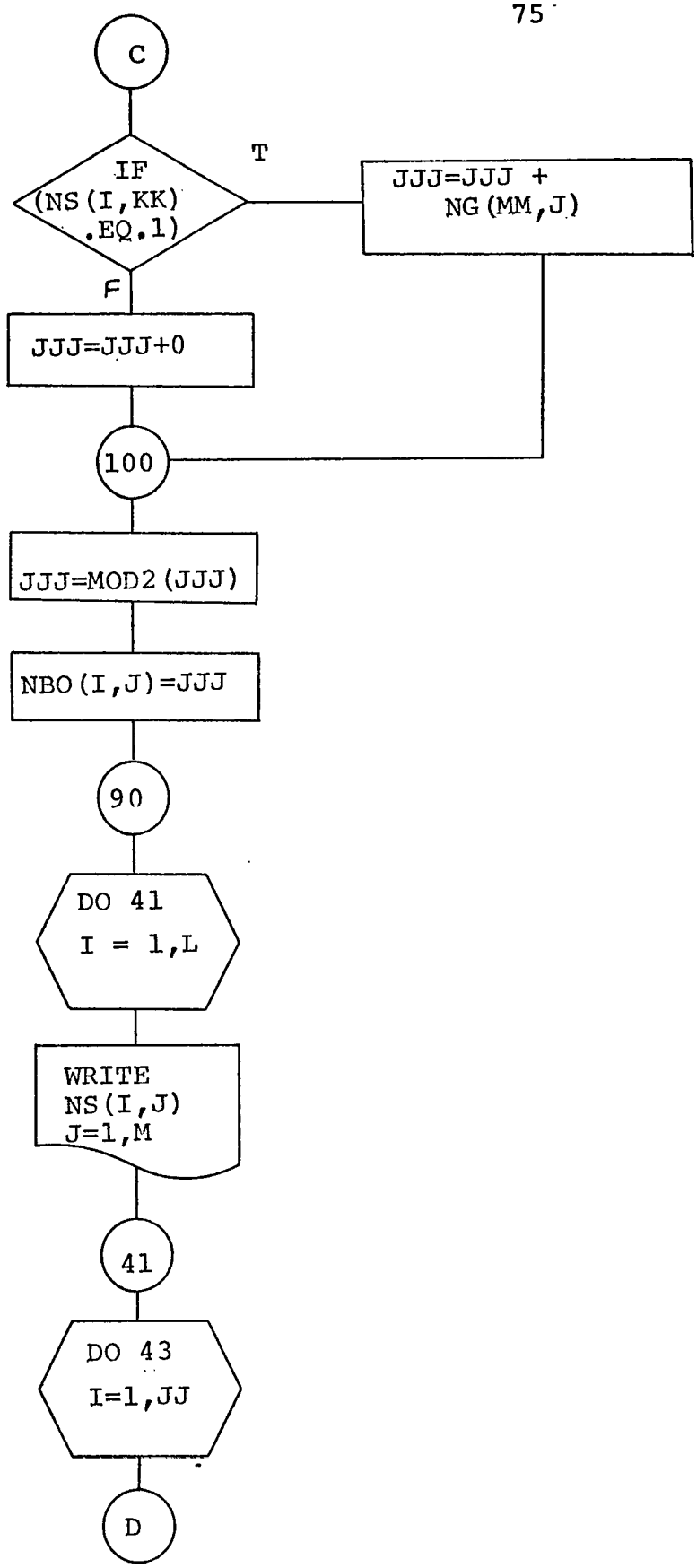

SUBROUTINE RANDOM
IF(IJ)
0,-
IJ=47594118
+
K=23*IJ
J=10001009*(K/10001009)
IJ = K-J
R = IJ
R=0.0000001*R
RETURN
END

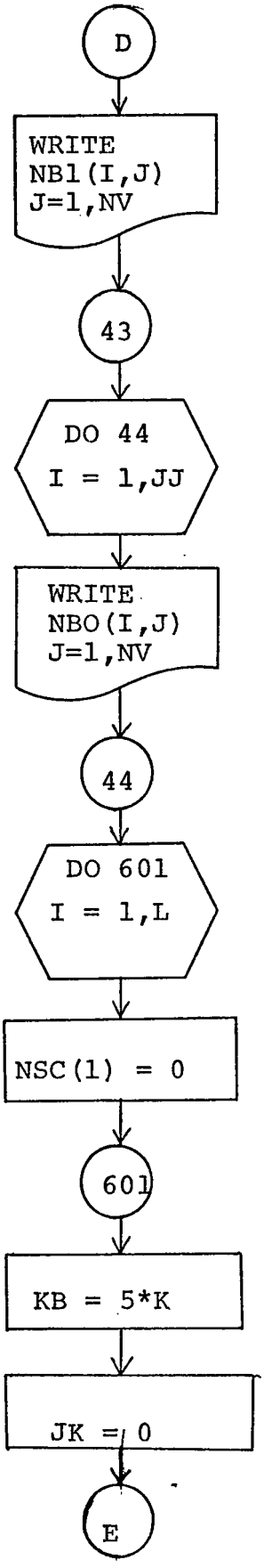

PROGRAM
VITERBI
DECODING
ALGORITHM
READ K,NV
DO 160
I=1,K
READ
NG(I,J)
J=1,NV
160
M = K-1
L = 2**M
DO 10
I = 1,M
N=L/(2**I)
NJ = 0
50
J = NJ+1
A

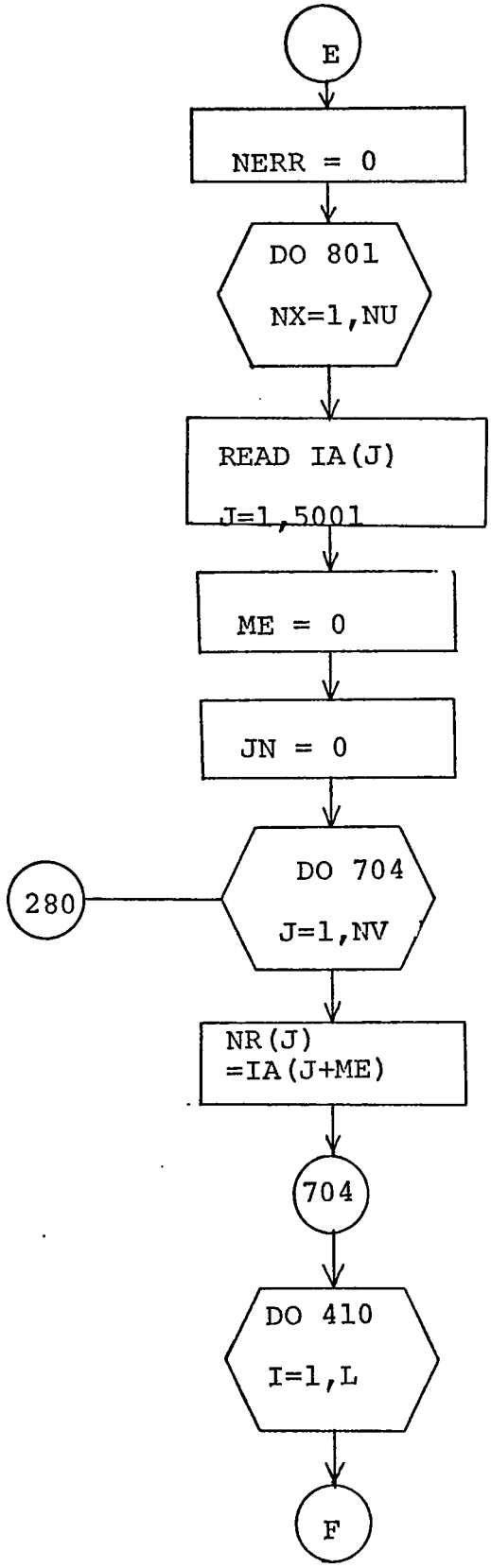

A
NJ = NJ+N
DO 20
LL=J,NJ
NS(LL,1)=0
KLL = LL
20
IF(KLL.LT.L)
F
10
T
30
J = NJ+1
NJ = NJ+N
DO 40
LL = J,NJ
NS(LL,I)=1
KLL = LL
40

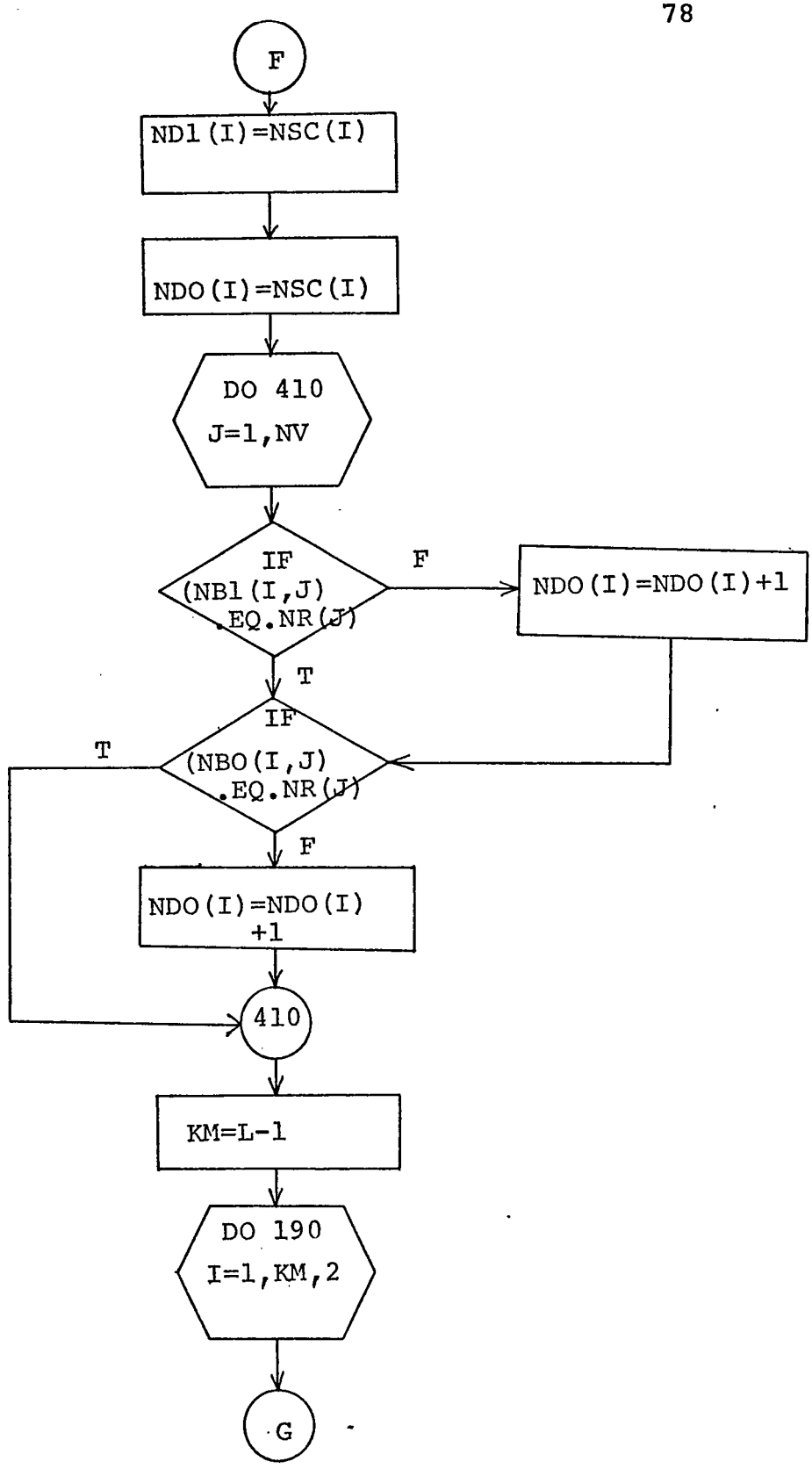

40
IF (KLL.LT.L)
T
50
F
10
JJ = L
DO 60
I = 1,JJ
DO 60
I = 1,NV
JJJ=NG(I,J)
DO 70
KK, = 1,M
MM=KK+1
IF
(NS(I,KK)
.EQ.1)
T
JJJ=JJJ +
NG(MM,J)
F
B
70

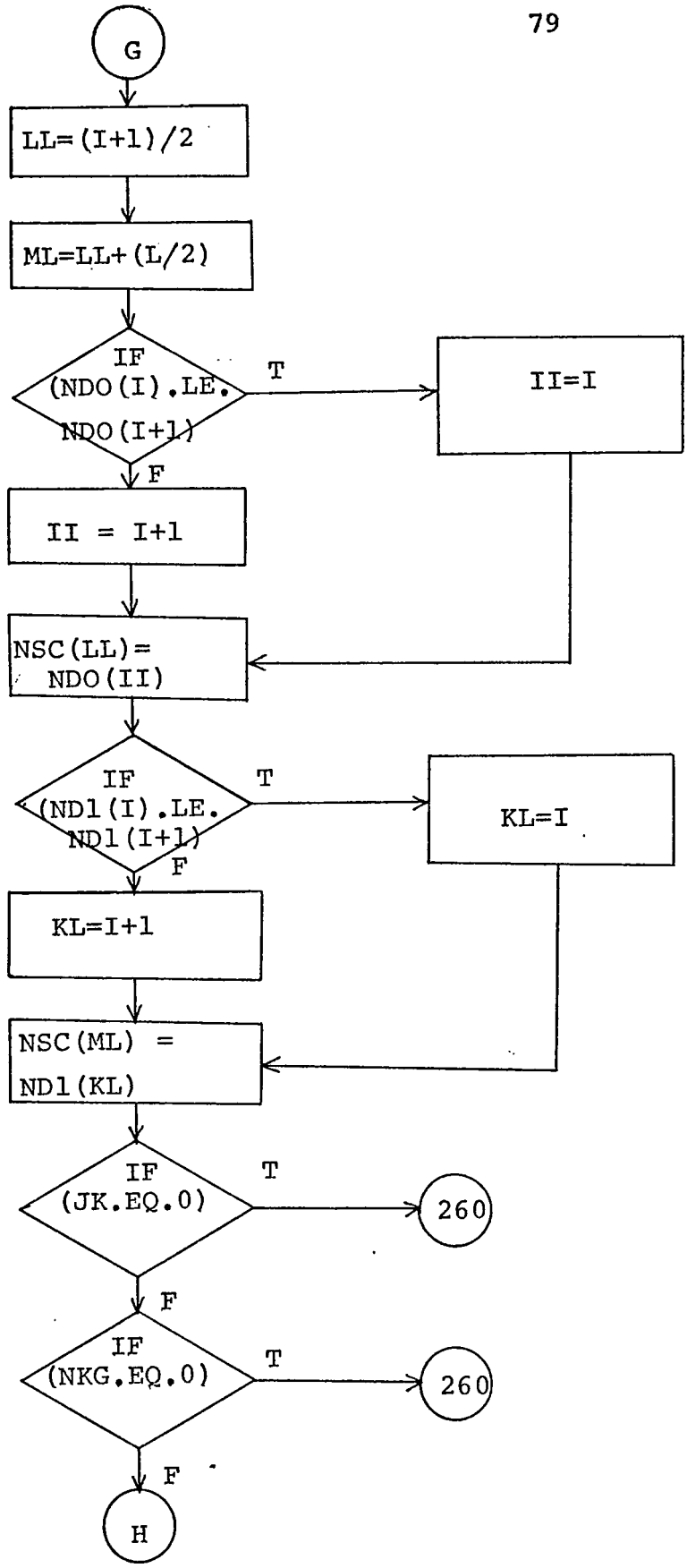

B
JJJ=JJJ+0
70
JJJ=MOD2(JJJ)
NB1(I,J)=JJJ
60
DO 90
I = 1,JJ
DO 90
J =1,NV
JJJ=0
DO 100
KK = 1,M
MM=KK+1
C

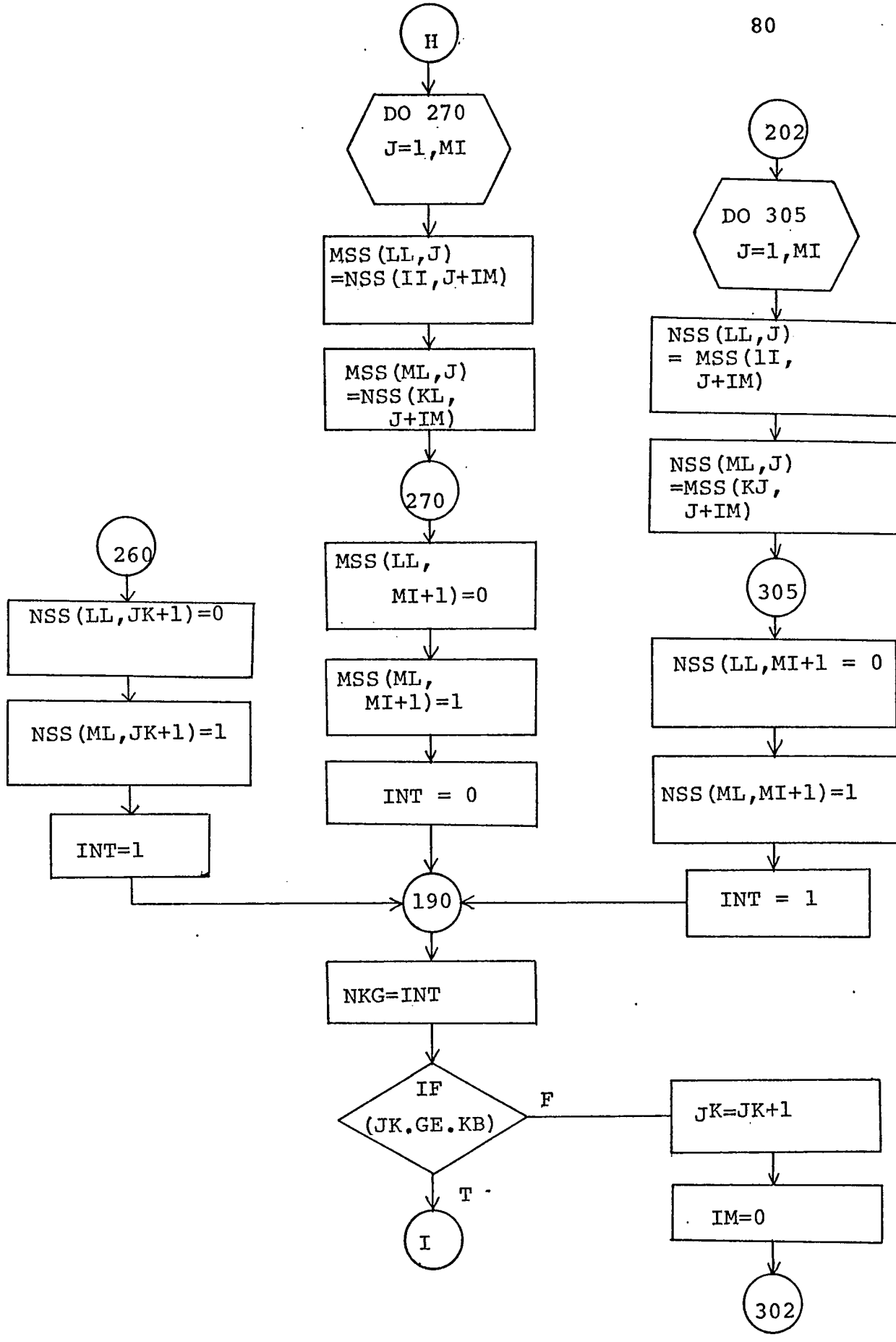

C
IF
(NS(I,KK)
.EQ.1)
T
JJJ=JJJ +
NG(MM,J)
F
JJJ=JJJ+0
100
JJJ=MOD2(JJJ)
NBO(I,J)=JJJ
90
DO 41
I = 1,L
WRITE
NS(I,J)
J=1,M
41
DO 43
I=1,JJ
D

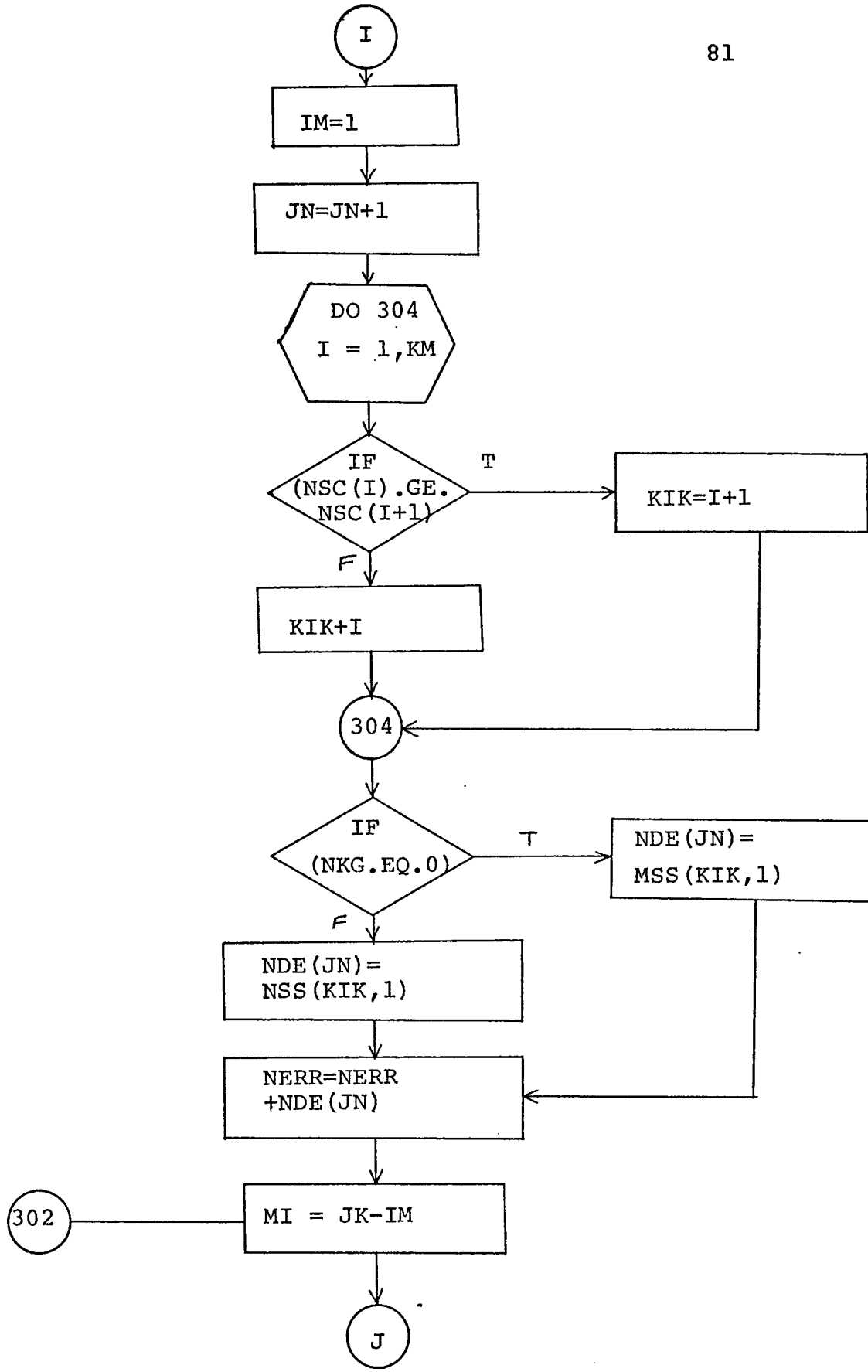

D
WRITE
NB1(I,J)
J=1,NV
43
DO 44
I = 1,JJ
WRITE
NBO(I,J)
J=1,NV
44
DO 601
I = 1,L
NSC(1) = 0
601
KB = 5*K
JK = 0
E

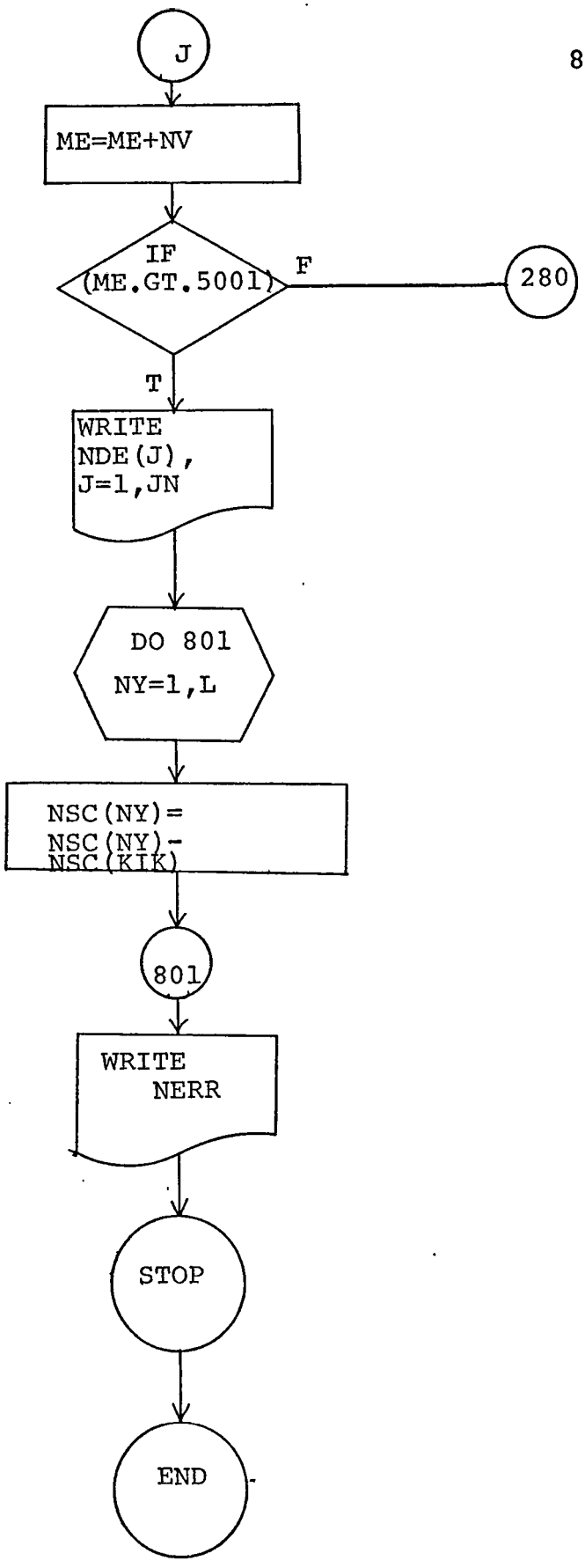

E
NERR = 0
DO 801
NX=1,NU
READ IA(J)
J=1,5001
ME = 0
JN = 0
280
DO 704
J=1,NV
NR(J)
=IA(J+ME)
704
DO 410
I=1,L
F

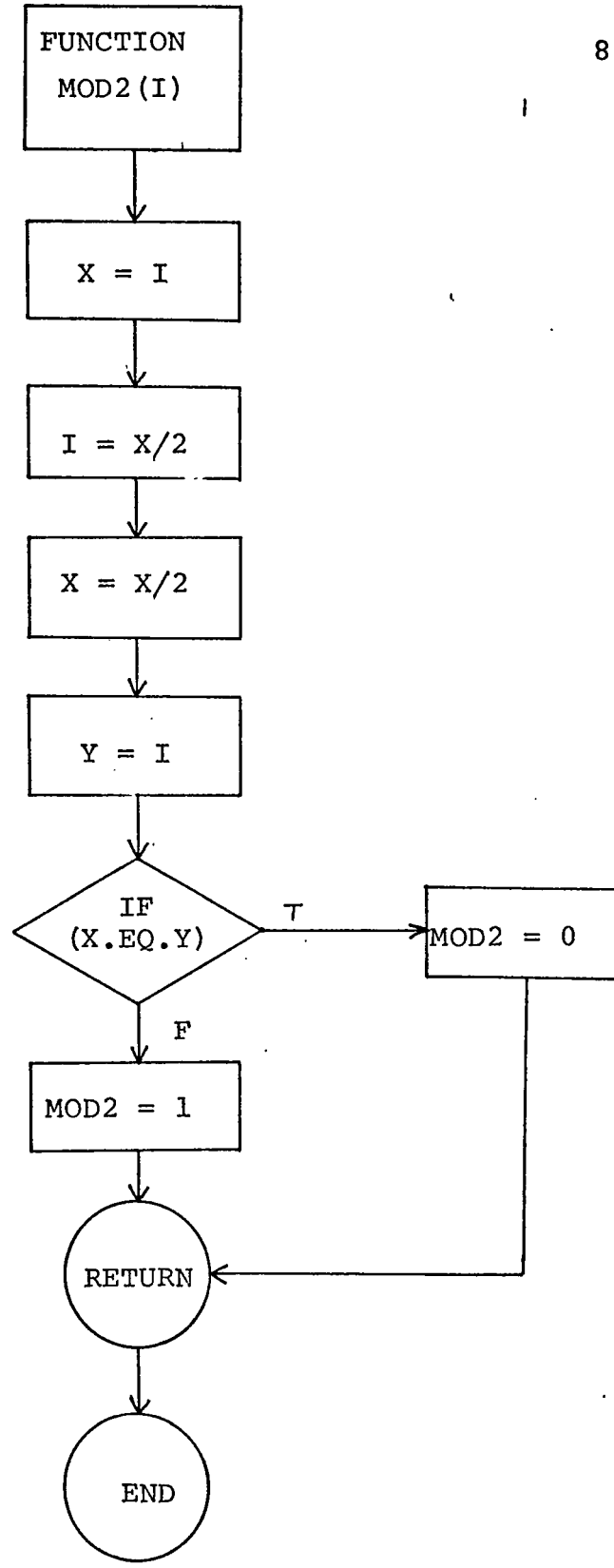

F
ND1(I)=NSC(I)
NDO(I)=NSC(I)
DO 410
J=1,NV
IF
(NB1(I,J)
.EQ.NR(J)
F
NDO(I)=NDO(I)+1
T
IF
(NBO(I,J)
.EQ.NR(J)
T
F
NDO(I)=NDO(I)
+1
410
KM=L-1
DO 190
I=1,KM,2
G


G
LL=(I+1)/2
ML=LL+(L/2)
IF (NDO(I).LE. NDO(I+1)
T
II=I
F
II = I+1
NSC(LL)= NDO(II)
IF (ND1(I).LE. ND1(I+1)
T
KL=I
F
KL=I+1
NSC(ML) = ND1(KL)
IF (JK.EQ.0)
T
260
F
IF (NKG.EQ.0)
T
260
F
H


H
DO 270
J=1,MI
MSS(LL,J)
=NSS(II,J+IM)
MSS(ML,J)
=NSS(KL,
J+IM)
270
MSS(LL,
MI+1)=0
MSS(ML,
MI+1)=1
INT = 0
260
NSS(LL,JK+1)=0
NSS(ML,JK+1)=1
INT=1
202
DO 305
J=1,MI
NSS(LL,J)
= MSS(1I,
J+IM)
NSS(ML,J)
=MSS(KJ,
J+IM)
305
NSS(LL,MI+1 = 0
NSS(ML,MI+1)=1
INT = 1
190
NKG=INT
IF
(JK.GE.KB)
F
T
JK=JK+1
IM=0
I
302


I
IM=1
JN=JN+1
DO 304
I = 1,KM
IF
(NSC(I).GE.
NSC(I+1)
T
KIK=I+1
F
KIK+I
304
IF
(NKG.EQ.0)
T
NDE(JN)=
MSS(KIK,1)
F
NDE(JN)=
NSS(KIK,1)
NERR=NERR
+NDE(JN)
302
MI = JK-IM
J


J
ME=ME+NV
IF
(ME.GT.5001)
F
280
T
WRITE
NDE(J),
J=1,JN
DO 801
NY=1,L
NSC(NY)=
NSC(NY)-
NSC(KIK)
801
WRITE
NERR
STOP
END


FUNCTION
MOD2(I)
X = I
I = X/2
X = X/2
Y = I
IF
(X.EQ.Y)
T
MOD2 = 0
F
MOD2 = 1
RETURN
END

Program for Noise Generation by Window Method

```
      DIMENSION IA(5001)
      READ(5,700) NU,PE
700   FORMAT(I5,F10.3)
      IJ=0
      NERR=0
      DO 701 I=1,NU
      DO 703 J=1,5001
      CALL RANDOM(IJ,R)
      IF(R.GT.PE) GO TO 702
      IA(J)=1
      NERR=NERR+1
      GO TO 703
702   IA(J)=0
703   CONTINUE
      WRITE(2) (IA(J),J=1,5001)
      END FILE 2
      REWIND 2
      DO 704 I=1,NU
      READ(2) (IA(J),J=1,5001)
      WRITE(6,705) (IA(J),J=1,5001)
705   FORMAT(//50I2)
704   CONTINUE
      WRITE(6,706) NERR
706   FORMAT(//////5X,'TOTAL ERRORS IN THIS RUN ARE =',I10)
      STOP
      END
```

```
      SUBROUTINE RANDOM(IJ,R)
      IF(IJ) 10,10,20
10    IJ=47594118
20    K=23*IJ
      J=10001009
      IJ=K-J
      R=IJ
      R=0.0000001*R
      RETURN
      END
```

Program for Viterbi Decoding Algorithm

```
      DIMENSION NB1(32,3),NB0(32,3),NR(3),ND1(64),ND0(64),NSC(64)
      DIMENSION NSS(64,35),MSS(64,35),NDE(2000),IA(5001)
      DIMENSION NS(100,10),NG(10,10)
      READ(5,11) K,NV
11    FORMAT(2I5)
      DO 160 I=1,K
      READ(5,21) (NG(I,J),J=1,NV)
21    FORMAT(10I5)
160   CONTINUE
      M=K-1
      L=2**M
      DO 10 I=1,M
      N=L/(2**I)
      NJ=0
50    J=NJ+1
      NJ=NJ+N
      DO 20 LL=J,NJ
      NS(LL,I)=0
      KLL=LL
20    CONTINUE
      IF(KLL.LT.L) GO TO 30
      GO TO 10
30    J=NJ+1
      NJ=NJ+N
      DO 40 LL=J,NJ
      NS(LL,I)=1
      KLL=LL
40    CONTINUE
      IF(KLL.LT.L) GO TO 50
10    CONTINUE
      JJ=L
      DO 60 I=1,JJ
      DO 60 J=1,NV
      JJJ=NG(1,J)
      DO 70 KK=1,M
      MM=KK+1
      IF(NS(I,KK).EQ.1) GO TO 80
      JJJ=JJJ+0
      GO TO 70
80    JJJ=JJJ+NG(MM,J)
70    CONTINUE
      JJJ=MOD2(JJJ)
      NB1(I,J)=JJJ
60    CONTINUE
      DO 90 I=1,JJ
      DO 90 J=1,NV
      JJJ=0
      DO 100 KK=1,M
      MM=KK+1
      IF(NS(I,KK).EQ.1) GO TO 110
      JJJ=JJJ+0
      GO TO 100
110   JJJ=JJJ+NG(MM,J)
100   CONTINUE
      JJJ=MOD2(JJJ)
```

```
      NBO(I,J)=JJJ
90    CONTINUE
      DO 41 I=1,L
      WRITE(6,42) (NS(I,J),J=1,M)
41    CONTINUE
42    FORMAT(//15I2)
      DO 43 I=1,JJ
      WRITE(6,42) (NB1(I,J),J=1,NV)
43    CONTINUE
      DO 44 I=1,JJ
      WRITE(6,42) (NBO(I,J),J=1,NV)
44    CONTINUE
      DO 601 I=1,L
      NSC(I)=0
601   CONTINUE
      KB=5*K
      JK=0
      NERR=0
      DO 801 NX=1,60
      READ(1) (IA(J),J=1,5001)
      ME=0
      JN=0
280   DO 704 J=1,NV
      NR(J)=IA(J+ME)
704   CONTINUE
      DO 410 I=1,L
      ND1(I)=NSC(I)
      NDO(I)=NSC(I)
      DO 410 J=1,NV
      IF(NB1(I,J).EQ.NR(J)) GO TO 180
      ND1(I)=ND1(I)+1
180   IF(NBO(I,J).EQ.NR(J)) GO TO 410
      NDO(I)=NDO(I)+1
410   CONTINUE
      KM=L-1
      DO 190 I=1,KM,2
      LL=(I+1)/2
      ML=LL+(L/2)
      IF(NDO(I).LE.NDO(I+1)) GO TO 210
      II=I+1
      GO TO 220
210   II=I
220   NSC(LL)=NDO(II)
      IF(ND1(I).LE.ND1(I+1)) GO TO 230
      KL=I+1
      GO TO 240
230   KL=I
240   NSC(ML)=ND1(KL)
      IF(JK.EQ.0) GO TO 260
      IF(NKG.EQ.0) GO TO 202
```

```
      DO 270 J=1,MI
      MSS(LL,J)=NSS(II,J+IM)
      MSS(ML,J)=NSS(KL,J+IM)
270   CONTINUE
      MSS(LL,MI+1)=0
      MSS(ML,MI+1)=1
      INT=0
      GO TO 190
202   DO 305 J=1,MI
      NSS(LL,J)=MSS(II,J+IM)
      NSS(ML,J)=MSS(KL,J+IM)
305   CONTINUE
      NSS(LL,MI+1)=0
      NSS(ML,MI+1)=1
      INT=1
190   CONTINUE
      NKG=INT
      IF(JK.GE.KB) GO TO 301
      JK=JK+1
      IM=0
      GO TO 302
301   IM=1
      JN=JN+1
      DO 304 I=1,KM
      IF(NSC(I).GE.NSC(I+1)) GO TO 404
      KIK=I
      GO TO 304
404   KIK=I+1
304   CONTINUE
      IF(NKG.EQ.0) GO TO 307
      NDE(JN)=NSS(KIK,1)
      GO TO 309
307   NDE(JN)=MSS(KIK,1)
309   NERR=NERR+NDE(JN)
302   MI=JK-IM
      ME=ME+NV
      IF(ME.EQ.5001) GO TO 290
      GO TO 280
290   WRITE(6,330) (NDE(J),J=1,NV)
330   FORMAT(//5012)
      DO 331 NY=1,L
      NSC(NY)=NSC(NY)-NSC(KIK)
331   CONTINUE
801   CONTINUE
      WRITE(6,803) NERR
803   FORMAT(/////5X,'TOTAL ERRORS IN THIS RUN ARE =',I10)
      STOP
      END
```

```
      FUNCTION MOD2(I)
      X=I
      I=X/2.
      X=X/2.
      Y=I
      IF(X.EQ.Y) GO TO 10
      MOD2=1
      GO TO 20
10    MOD2=0
20    RETURN
      END
```