ALGORITHMIC ASPECTS OF SEQUENTIAL DECODING

by John M. Geist

Technical Report No. EE-702

August 1970

Department of

ELECTRICAL ENGINEERING

UNIVERSITY OF NOTRE DAME, NOTRE DAME, INDIANA

ALGORITHMIC ASPECTS OF SEQUENTIAL DECODING

by John M. Geist

Technical Report No. EE- 702

August 1970

Department of Electrical Engineering

University of Notre Dame, Notre Dame, Indiana 46556

This work was submitted to the Graduate School of the University of Notre Dame in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This work was supported by the National Aeronautics and Space Administration under NASA Grant NGL 15-004-026 in liaison with the Flight Data Systems Branch of the Goddard Space Flight Center.

ABSTRACT

Sequential decoding procedures are studied in the context of selecting a high-value path through a metric tree. Several algorithms are considered and their properties compared. For the conventional Fano algorithm, it is noted that the difficulty in decoding depends on the minimum node value along the correct path. This minimum is a random variable and it can be modeled in terms of Markov chains. Some properties of such Markov chains are studied.

A stack algorithm introduced by Zigangirov and independently by Jelinek is presented, and it is shown that this algorithm is essentially equivalent to the Fano algorithm with regard to the set of nodes examined and the path selected, although the description and action of the two algorithms are quite different.

A modified Fano algorithm is introduced, in which the quantizing parameter Δ is eliminated and decisions are based on exact node values rather than on quantized values. While the new algorithm is computationally inferior to the old (in cases studied so far), it is of some theoretical interest since the conventional (quantized) Fano algorithm may be considered to be a quantized version of it.

Extensive computer simulations comparing the Fano algorithm with a quantized Zigangirov-Jelinek algorithm are reported. The conclusion of these comparisons is that at rates near R_{comp} the stack algorithm offers an advantage over the Fano algorithm in decoder speed, but it requires large storage to be available for use by the decoder.

Finally, the possibility of using sequential tree search algorithms for searching more general graphs is investigated.

ACKNOWLEDGEMENT

I am deeply indebted to Professor James L. Massey for his patient support during the course of my research. This work would have been impossible without his guidance, suggestions, and encouragement. My association with him has been one of my most rewarding experiences.

In addition, many of Professor Massey's present and former students have been very helpful; among these, special mention is due Daniel J. Costello, Jr., William F. Hartman, and Jian K. Chang.

I am grateful also to Professor R. J. Leake for some helpful comments and suggestions with respect to the material in Chapter IV.

CONTENTS

CHAPTER I

INTRODUCTION

Communication systems employing convolutional encoding at the input and sequential decoding at the output of certain noisy channels are among the most attractive means of approaching the reliability of communication promised by the coding theorem. In this chapter convolutional codes are discussed briefly, and then sequential decoding is described informally. Some well-known results on sequential decoding are presented and heuristically treated.

1. A. CONVOLUTIONAL CODES

A general single-input rate ½ convolutional encoder is displayed in Figure 1. A single information sequence of symbols from some finite field GF(q) is fed into the encoder from the left and two output sequences are transmitted. We may think of the output sequences as being commutated to form a single transmitted sequence, or as being transmitted over two identical channels. The polynomials

$$G_1(D) = g_0^{(1)} + g_1^{(1)}D + g_2^{(1)}D^2 + \cdots + g_m^{(1)}D^m$$

$$G_2(D) = g_0^{(2)} + g_1^{(2)}D + g_2^{(2)}D^2 + \cdots + g_m^{(2)}D^m$$

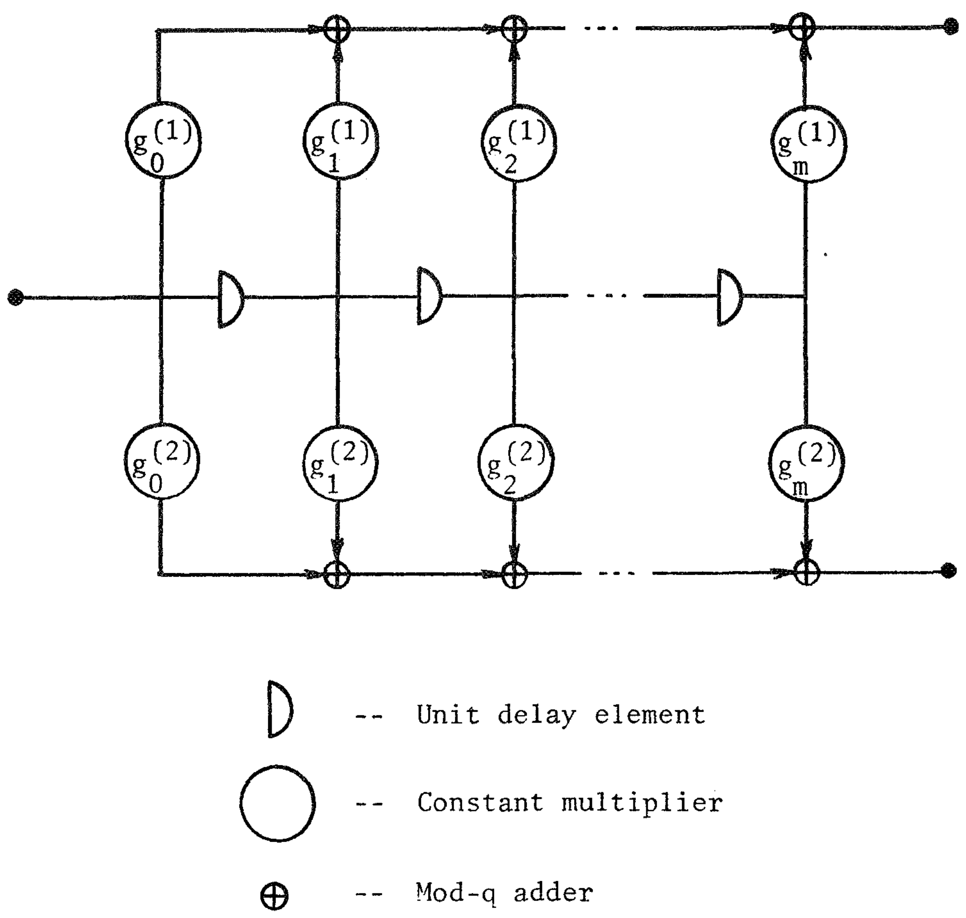


FIGURE 1. A Single-input Rate ½ Convolutional Encoder.

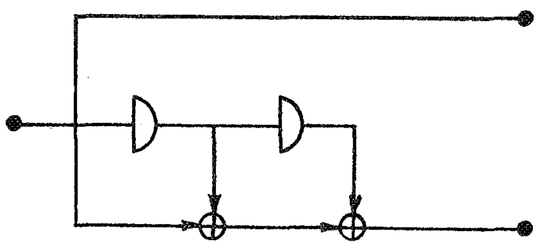


FIGURE 2. A Binary Rate ½ Systematic Encoder of Memory 2.

are called code-generating polynomials and m is called the memory of the encoder. For well-chosen G_1 and G_2 the received sequence, when not too severely corrupted by channel noise, can be processed after reception and the original information sequence recovered.

The dependence of transmitted symbols upon a span of past information symbols gives rise to a natural representation of the output of a convolutional encoder in the form of a tree. This tree is rooted at an origin node which has no predecessor, while every other node has exactly one predecessor. Every node has exactly two successors (for single-input encoders), and there is a one-to-one correspondence between the set of paths through the tree and the set of possible information sequences. As an example, consider the encoder of Figure 2, a rate ½ binary encoder of memory 2 with $G_1(D) = 1$ and $G_2(D) = 1 + D + D^2$. The effect of choosing $G_1(D) = 1$ is that the information sequence appears explicitly in the encoder output. Codes having this property are called systematic. The tree of Figure 3 shows the set of all possible encoder outputs, i.e., the code. At each node in the tree the symbols along the upward and downward emanating branch correspond to the output when a zero and one, respectively, is inserted into the encoder, given that the past information bits fed into the encoder were the bits designated by the path to that node. For example, the information sequence 0 1 0 1 · · · · causes an encoded sequence 00 11 01 10 · · · · , as shown by the bold path in Figure 3.

It is precisely this tree structure of convolutional codes which makes them well-suited to sequential decoding, as we shall see later.

Another graphical model of convolutional codes is suggested by

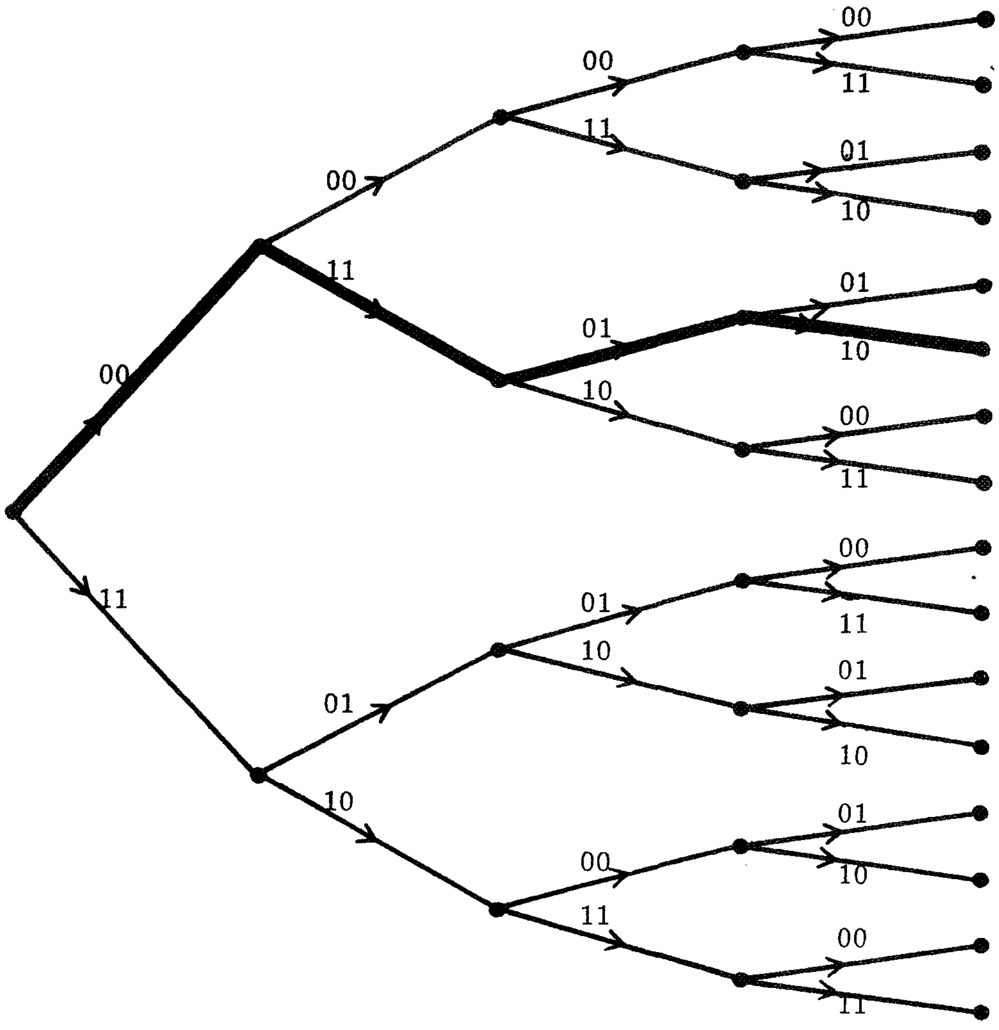


FIGURE 3. Code Tree for the Encoder of Figure 2.

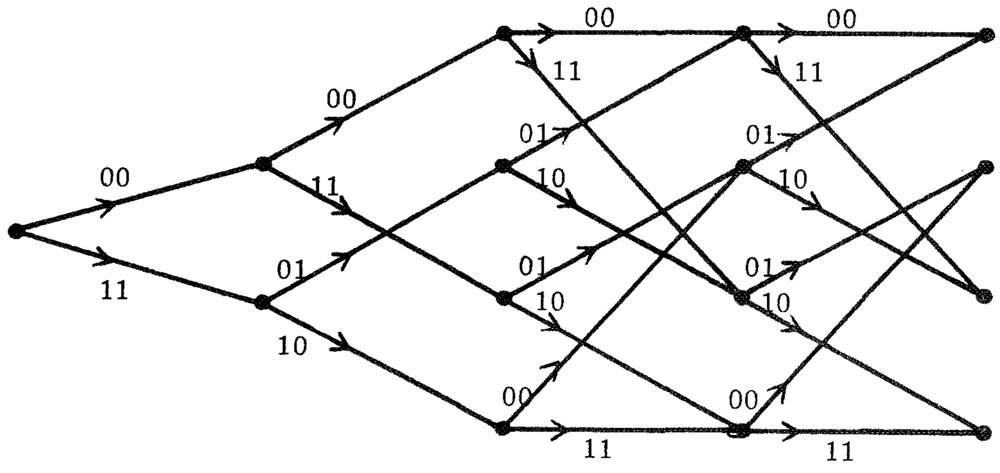


FIGURE 4. Code Trellis for the Encoder of Figure 2.

the fact that the dependence of the encoder outputs upon past information symbols extends only over a finite span of past inputs. If two information sequences $i_0, i_1, \cdots, i_u$ and $i'_0, i'_1, \cdots, i'_u$ differ only in the first u-m+1 places, the portion of the code tree extending out from the node at the end of the path specified by the first sequence is identical to that extending out from the node at the end of the path specified by the second. Thus there is no need to distinguish between these two nodes, and the code tree can be collapsed to a trellis by identifying such nodes. More formally, we may consider the nodes in the trellis to be equivalence classes of nodes in the tree under the relation:

$$S_1 \equiv S_2 \iff \left\{ \begin{array}{l} \text{the encoder state is the same at } S_1 \\ \text{and } S_2\text{; and} \\ \\ S_1 \text{ and } S_2 \text{ are at the same depth in} \\ \text{the tree.} \end{array} \right.$$

The result of collapsing the tree of Figure 3 by identifying nodes at the ends of paths which agree in the last m=2 places is the trellis of Figure 4. Again the output sequence is the sequence of symbols along the path determined by the information sequence, but in the trellis, paths which initially differ will remerge if the corresponding information sequences agree in m consecutive places. The trellis model for convolutional codes was suggested by Forney[1].

As we remarked before, the tree model is the more natural model for convolutional codes when we are studying the action of a sequential decoder. The trellis model is convenient when studying probability of error, however, and in addition we shall see in Chapter IV that one can meaningfully discuss the action of a sequential decoder operating on such a trellis.

1. B. SEQUENTIAL DECODING

In decoding a received sequence, one desires to select from among the transmitted sequences which make up the code, the one which most likely would have been transformed by channel noise into the given received sequence. Of course, one could just as well produce this sequence by eliminating unlikely sequences until only a single sequence remains, if this turns out to be more convenient. Wozencraft[2] noted that in applying this rejection technique to codes having the tree structure discussed before, the number of sequences rejected is a fraction of the total number of codewords which, roughly speaking, grows exponentially with the "earliness" of rejection. More exactly, rejection of a path on the basis of the fact that the transmitted sequence along its first k branches is unlikely to have been corrupted by noise into the corresponding segment of the received sequence is tantamount to rejecting <u>every</u> path beginning with these k branches, namely $^1/_{u^k}$ of the paths in the tree, where u is the number of branches emanating from each node. Wozencraft termed his procedure for exploiting this property "sequential decoding."

Jacobs and Berlekamp[3] have given two conditions which a decoding algorithm must satisfy in order to qualify as a sequential decoding algorithm:

JB1: The decoder performs at least one computation for each node it examines.

JB2: Decisions the decoder makes about searching new parts of the tree are made only on the basis of information about nodes already examined, and not about nodes in the unexplored part of the tree.

The sequential decoding procedures we will be concerned with could be termed "metric-based"; that is, search decisions are based on values, called metrics, which are assigned to every branch in the code tree. These values are, for memoryless channels, generally taken to be:

$$z = \sum_{j=1}^{N} \left[\log \frac{\Pr\{r_j | x_j\}}{f(r_j)} - B \right] \tag{1}$$

where x_j is the jth code symbol on the branch, r_j is the corresponding symbol in the received sequence, $f(\cdot)$ is the nominal received symbol probability function, N is the number of code symbols per branch, and B is a bias term. The bias is chosen in such a way that, on the average, values of branches along the correct path are positive, while values of other branches are negative. Nodes are assigned values equal to the sum of the branch values along the path leading to the node. With our choice of bias, it is clear that on the average, node values along the correct path will increase with depth into the tree, while node values along incorrect paths will decrease. The decoder strategy is then to look for a path of increasing value.

The difficulty the decoder encounters in searching for such a path depends on the various branch values, which are in turn determined by the channel noise. Define C_0 to be the number of computations the decoder performs while examining nodes such that the first branch of the path leading to the node is not the first branch of the correct path. Then C_0 is a random variable, and its statistics are of importance in determining the practicability of sequential decoding. In particular, we would like to know how rapidly the quantity $\Pr\{C_0>N\}$ decreases as N increases. We might hope, for example, that this quantity decreases exponentially with N. The following discussion shows that this is not the case.

Consider using a rate ½ code on a binary symmetric channel with crossover probability p (see Figure 5). Assume that the code is complementary; that is, at every node the symbols on the zero branch emanating from that node are complements of the symbols on the one branch. From (1), the branch metrics have the form:

$$z = \begin{cases} 2\log[2(1-p)] - 2B, & \text{if both symbols agree} \\ \log[2(1-p)] + \log[2p] - 2B, & \text{if one agrees and one disagrees} \\ 2\log[2p] - 2B, & \text{if both disagree} \end{cases}$$

Denote these three values z_0, z_1, and z_2 respectively, and note that $z_2 < z_1 < z_0$ (assuming $0 < p < \frac{1}{2}$). By the group property of convolutional codes, ¼ of the branches at any depth in the tree have branch metric z_0, ½ have branch metric z_1, and ¼ have branch metric z_2. Hence the requirement that the branch metric be negative on the average for branches not on the correct path reduces to:

$$\tfrac{1}{4}z_0 + \tfrac{1}{2}z_1 + \tfrac{1}{4}z_2 = z_1 < 0$$

Moreover we must have $z_0 < 0$ if the average metric on the correct path is to be positive. Hence $z_2 < z_1 < 0 < z_0$. Since the code is complementary, at each node the two exiting branch metrics are either z_0 and z_2 or z_1 and z_1.

Suppose that channel noise causes the node values along the correct path to decrease with depth initially before becoming positive and increasing through the rest of the tree. Let the minimum node value along the correct path be $-Q$, where $Q > 0$. It can be shown that the probability of this event, for large Q, varies as e^{-rQ}, where r is a positive constant. Call that part of the tree composed of paths beginning with the first incorrect branch, the incorrect half-tree. Now by JB2, the decoder must examine every path in the incorrect half-tree at least as far as the first node whose value is less than $-Q$, because it cannot determine before that point which of the paths is increasing. Therefore the number of nodes examined in the incorrect half-tree is at least as great as the number of such nodes which are connected to the origin by paths along which each node has value greater than $-Q$. Let us denote this number $M(Q)$. We now claim that the metric tree configuration which minimizes $M(Q)$, for any Q, is that in which the first incorrect branch has value z_2 and every other branch in the incorrect half-tree has value z_1; that is, the tree shown in Figure 6*. To see that this is so, note that any other metric tree configuration differs from this one in one or both of these ways: (1) the first incorrect branch has value different from z_2; (2) at some nodes in the incorrect half-tree, the exiting branch metrics are z_0 and z_2 instead of z_1 and z_1. Starting with the tree of Figure 6, it is possible to make successive changes, each of which does not decrease $M(Q)$, and which transform the original tree into any tree we please, as deep into the tree as necessary. For example, Figure 7 shows how the tree of Figure 6 can be transformed into that of Figure 7f. In going from a to b, the value of every node in the incorrect half-tree is increased by $z_1 - z_2$; hence $M_b(Q) \geqq M_a(Q)$. To go from b to c, consider interrupting the tree at node A, shifting the part of the tree rooted there one step to the right, and inserting a new branch of value z_0. The nodes in the region enclosed by the dashed line have higher values than their counterparts in tree b, so $M_c(Q) \geqq M_b(Q)$.

* A somewhat similar argument was used by Savage[4], who noted that a tree in which every branch in the incorrect half-tree has value z_2 overbounds $M(Q)$.

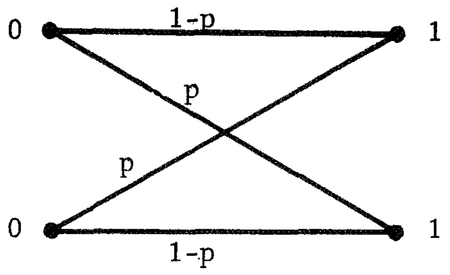


FIGURE 5. The Binary Symmetric Channel.

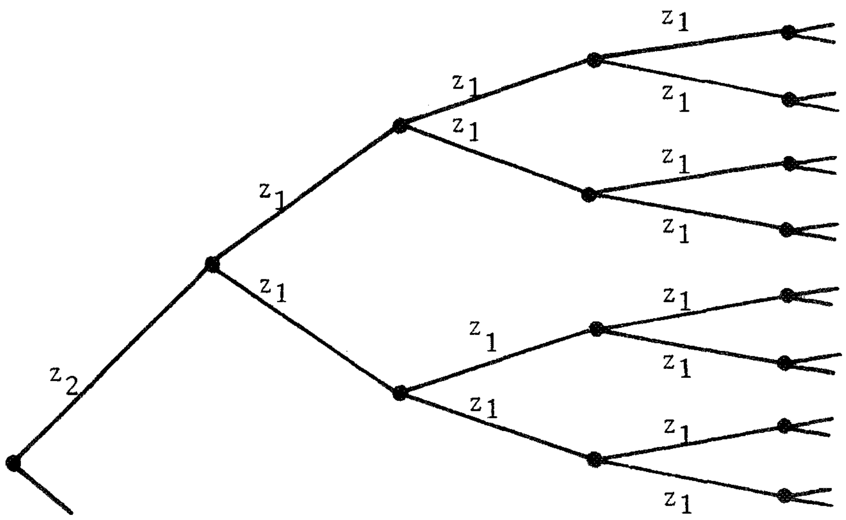


FIGURE 6. The Optimum Configuration of the Incorrect Half-tree.

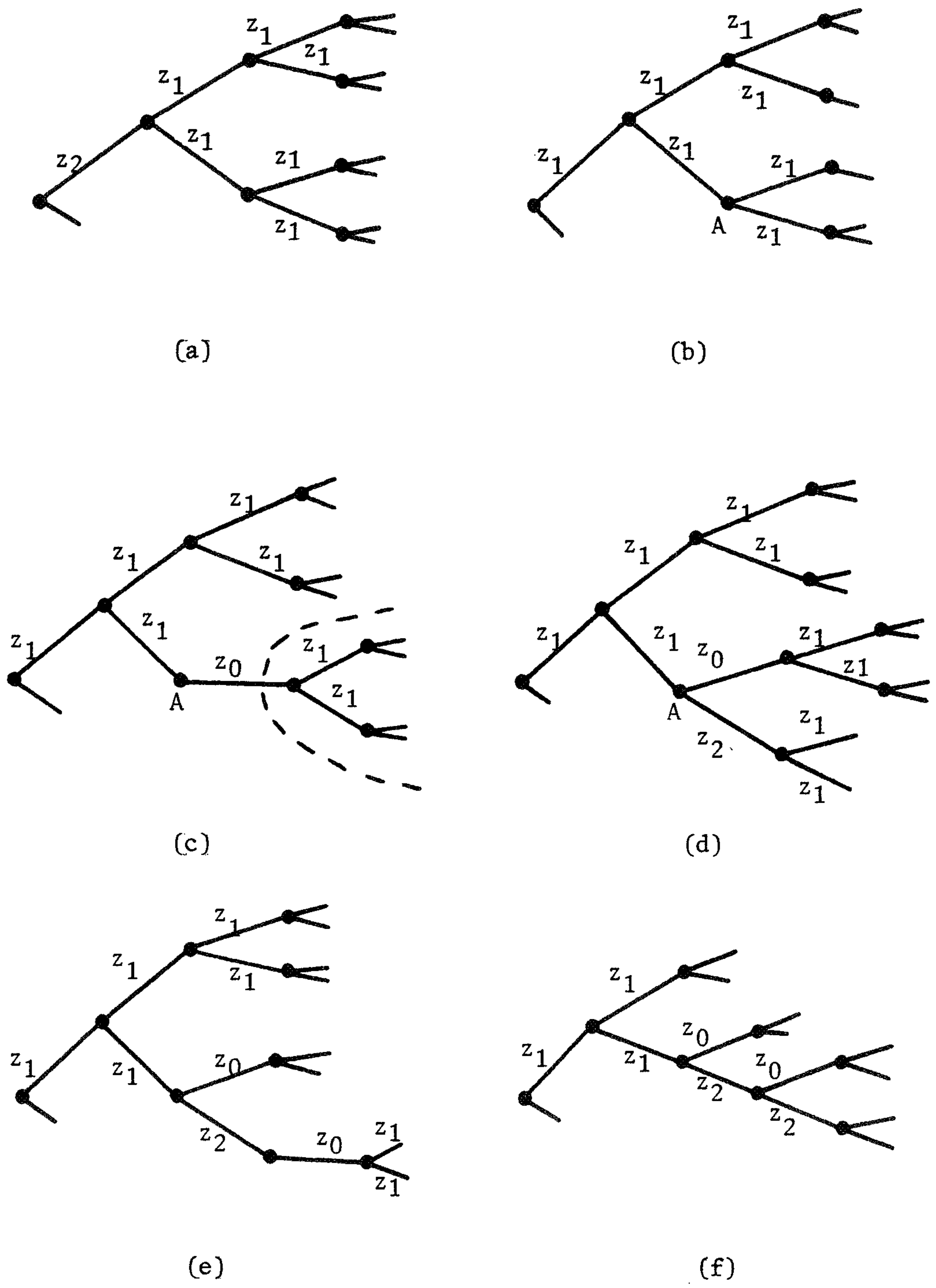


FIGURE 7. Steps in Showing That the Configuration of Figure 6 is Optimum.

Now adding another branch at node A and rooting a tree of branches of value z_1 does not affect the other nodes, so $M_d(Q) \geqq M_c(Q)$. The same process leads through steps e and f, so finally, $M_f(Q) \geqq M_a(Q)$. It is clear that this process can be extended as far into the tree as necessary.

We are now prepared to lower-bound $\Pr\{C_0 \geqq N\}$ for the case under consideration. It is clear that for the tree of Figure 6,

$$M(Q) \geqq 2^{\lfloor -(Q+z_2)/z_1 \rfloor} \geqq k2^{-Q/z_1} \quad \text{for some } k$$

where $\lfloor x \rfloor$ indicates the integer part of x. Now by JB1, $C_0 \geqq M(Q)$. Thus for large Q (and hence large N),

$$\begin{aligned} \Pr\{C_0 \geqq k2^{-Q/z_1}\} &\geq \Pr\{\text{minimum along correct path} = -Q\} \\ &\simeq e^{-rQ} \end{aligned}$$

Putting $N = k2^{-Q/z_1}$ yields $Q = K(\ln N - \ln k)$, and hence

$$\Pr\{C_0 \geqq N\} \geqq e^{-rK(\ln N - \ln k)} \rightarrow N^{-\alpha} \quad \text{as } N \rightarrow \infty.$$

This shows that the combined effects of exponential growth of the number of nodes and exponential decay of having to examine them results in a distribution of computation which decreases at most algebraically. A distribution of this form is called Pareto.

In a more rigorous and less restricted derivation than this one, Jacobs and Berlekamp[3] have shown that for any decoding algorithm satisfying JB1 and JB2, the distribution of computation (conditioned on correct decoding) is essentially Pareto. To state their result precisely, consider operating on a discrete memoryless channel with K inputs and J outputs, transition probabilities $P(j|k)$, and input distribution $\underset{\sim}{Q} = \{Q(k),\ k=1,2,\cdots,K\}$. Define

$$E_0(\rho) = \max_{\underset{\sim}{Q}} \{-\log \sum_{j=1}^{J} [\sum_{k=1}^{K} Q(k)P(j|k)^{1/(1+\rho)}]^{1+\rho}\},\ \rho>0 \tag{2}$$

$$R_\rho = E_0(\rho)/\rho \tag{3}$$

Jacobs and Berlekamp show that the computation required to decode the first Λ branches, $C(\Lambda)$, has a distribution satisfying

$$\Pr\{C(\Lambda) \geqq N\} \geqq N^{-\rho}e^{-O\sqrt{\ln N}}$$

where ρ is the solution of $R = R_\rho$, and $O\sqrt{\ln N}$ is a quantity varying with $\sqrt{\ln N}$. The number Λ depends on N, but the dependence is asymptotically unimportant.

A summary of Jacobs and Berlekamp's argument, with a minor modification allowing the deduction of a lower bound on $\Pr\{C_0 \geqq N\}$ of the form

$$\Pr\{C_0 \geqq N\} \geqq F(N)N^{-\rho} \tag{4}$$

where F(N) is a slowly-varying function of N, is given in Appendix A.

Our derivation of a lower bound to the distribution of computation, as well as Jacobs and Berlekamp's, assume that the tree being searched is infinite in extent. In many practical applications, and in most of the sequel, it is assumed that the tree is finite, consisting of L levels. We now investigate the effect of this truncation on the distribution of computation.

First we modify slightly and generalize the definition of C_0 and take C_j to be the number of computations the decoder performs at the jth node of the correct path and at nodes such that the first j branches of the path leading to the node are the first j branches of the correct path, but the (j+1)st is not. Thus all computations performed in the incorrect part of the tree stemming from the jth node on the correct path, as well as the computations at the jth node on the correct path, are counted in C_j. Now for any $L > 0$, let $C_j(L)$ be the number of computations of the kind counted in C_j which are performed before the decoder makes its first move to any node at depth L. It is clear that $C_j(L) \leqq C_j$ for all j and L. In particular, $C_0(L) \leqq C_0$, and so

$$\Pr\{C_0(L) \geqq N\} \leqq \Pr\{C_0 \geqq N\} \tag{5}$$

for any L and N.

Since the decoder can make only finitely many computations before moving to depth L, there is some N* such that $\Pr\{C_0(L) \geqq N^*\} = 0$. On the other hand, for fixed C_0, the sequence of random variables $C_0(L)$, $L = 1, 2, \cdots$ converges in probability to C_0. The result is that there exists an intermediate range of N for which

$$\Pr\{C_0(L) \geqq N\} \simeq \Pr\{C_0 \geqq N\} \tag{6}$$

so that the lower bound (4) applies to the truncated computation $C_0(L)$ in this range, while for greater values of N, $\Pr\{C_0(L) \geqq N\} = 0$. (This is discussed more precisely in Appendix A.)

Now for a tree of L levels, the total number of computations performed in decoding is

$$\sum_{j=0}^{L-1} C_j(L)$$

A measure of difficulty in decoding that is easy to compute experimentally is the average number of computations per decoded branch,

$$C_{AV}(L) = \frac{1}{L}\sum_{j=0}^{L-1} C_j(L) \tag{7}$$

We now relate $\Pr\{C_{AV}(L) \geqq N\}$ to our known results. If $C_0(L) \geqq NL$, then $C_{AV}(L) \geqq N$. Hence if N is such that NL falls within the range for which (6) holds, then

$$\Pr\{C_0(L) \geqq N\} \approx \Pr\{C_0 \geqq NL\} \tag{8}$$

so that

$$\Pr\{C_{AV}(L) \geqq N\} \geqq N^{-\rho}L^{-\rho}F(NL) \tag{9}$$

by virtue of (4) and (8).

We can also upper-bound the distribution of $C_{AV}(L)$ by noting that if $C_{AV}(L) \geqq N$, then for some j, $C_j(L) \geqq N$. Hence by the union bound,

$$\Pr\{C_{AV}(L) \geqq N\} \leqq \sum_{j=0}^{L-1} \Pr\{C_j(L) \geqq N\} \tag{10}$$

Now since the effect of truncation is enhanced as we move deeper into the tree, we have

$$\Pr\{C_0(L) \geqq N\} \geqq \Pr\{C_j(L) \geqq N\}, \quad 0 \leqq j \leqq L-1 \tag{11}$$

Hence, using (11) in (10), and using (5) in the result,

$$\Pr\{C_{AV}(L) \geqq N\} \leqq L\Pr\{C_0 \geqq N\} \tag{12}$$

To recapitulate, we have shown that the distribution of the average computation per decoded branch in a finite tree is upper-bounded in accordance with (12). Further, for N in a range dependent on L, the existence and limits of which are discussed in Appendix A, the distribution satisfies the Pareto lower bound of (9), which has the same Pareto exponent as the distribution of C_0. It follows that for an intermediate range of N, $C_{AV}(L)$ is Pareto-distributed with the same exponent as C_0.

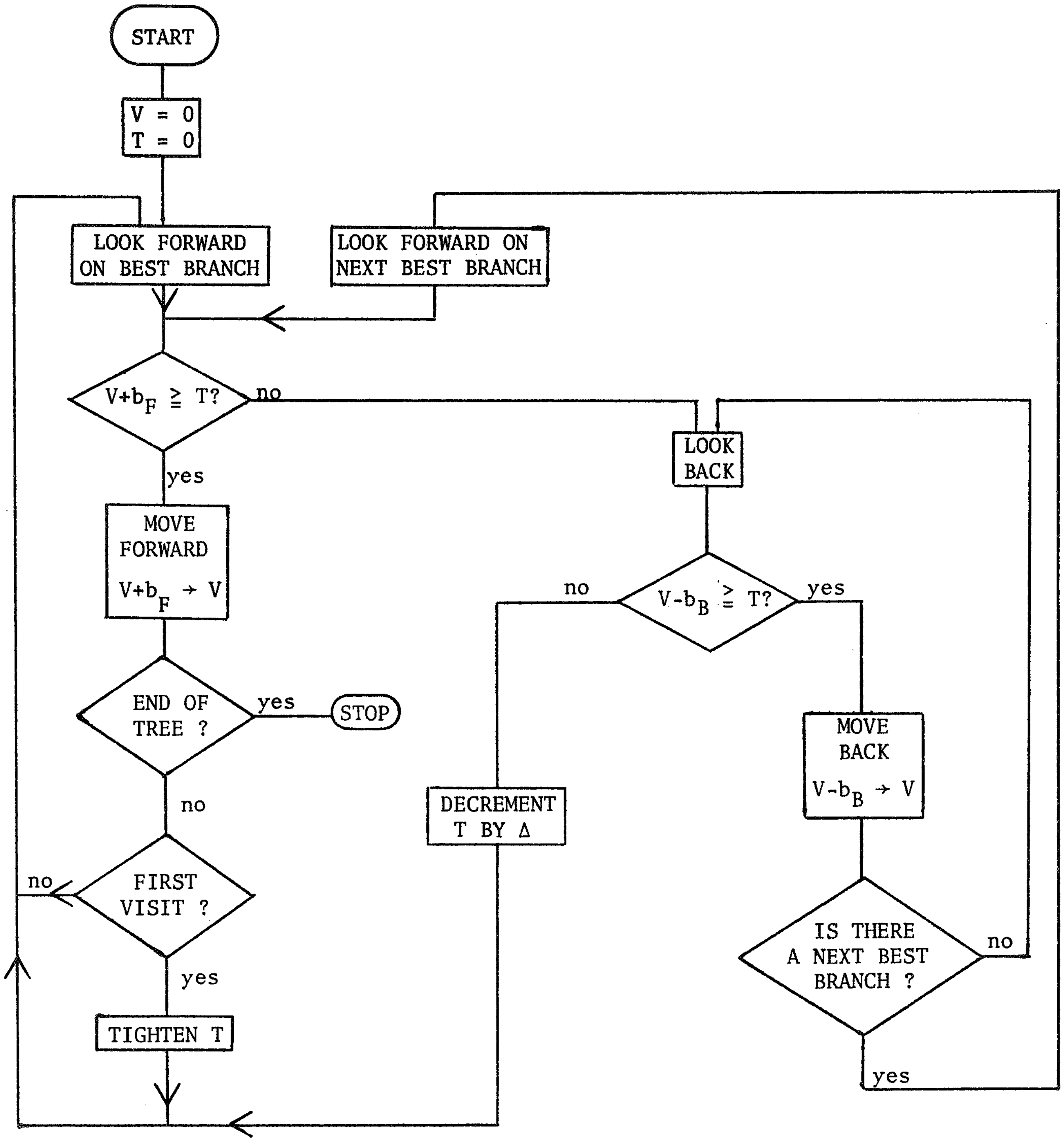
CHAPTER II

ALGORITHMIC PROPERTIES OF SEVERAL SEQUENTIAL DECODING PROCEDURES

In this chapter we investigate and compare some search properties of three sequential decoding algorithms: the standard Fano[5] algorithm; a stack algorithm recently introduced by Zigangirov[6] and Jelinek[7]; and a new modified version of the Fano algorithm which appears to be chiefly of theoretical interest.

2. A. THE FANO ALGORITHM

Subsequent to Wozencraft's introduction of sequential decoding, a number of alternative sequential decoding techniques were introduced, the best known of which is the algorithm proposed by Fano[5] in 1963. The Fano algorithm remains the most popular sequential decoding method, for both practical and theoretical purposes, even today. We assume the reader's familiarity with the algorithm in the form of Figure 8. (Readers unfamiliar with the algorithm will find a discussion in Wozencraft and Jacobs[8] or with a somewhat different viewpoint in Gallager[9].) In Fano's original work, the test for first visit at a node was performed by means of a flag. Gallager has observed ([9], p. 270) that in every case in which the threshold actually has to be



b_F: Forward branch value
b_B: Backward branch value

FIGURE 8. Fano Sequential Decoding Algorithm.

increased upon a new arrival, the threshold would have been tight at the previous node, so that the flowchart of Figure 9 is equivalent to that of Figure 8.

In constructing the metric tree to be searched, we place an imaginary branch of infinite metric leading into the origin node. This branch serves as an origin test since any time the decoder is positioned at the origin and enters the LOOK BACK box of Figure 9, the subsequent test always fails and no attempt is ever made to move back from the origin.

Since the Fano algorithm satisfies conditions JB1 and JB2, the computation a Fano decoder performs must have a distribution which satisfies the lower bound (I-4) (assuming infinite trees). It has been shown that the Fano decoder meets this lower bound asymptotically. Several investigators have contributed to this result, including Savage[4], Falconer[10], and Jelinek[11], who credits an important part of his argument to Yudkin. In addition, Yudkin[12] has shown that the probability of error with the Fano decoder is optimal for sufficiently high rates, and near-optimal for lower rates. These two results, along with the ease of implementation of this procedure, account for the widespread interest the Fano algorithm has attracted.

2. A. 1. Search Properties of the Fano Algorithm

We now describe the set of nodes searched by the Fano algorithm and the path ultimately found by this procedure. The results of this section are due to Massey and Sain[13] and are stated here for reference.

Let s_d be some node in level d of the value tree and let $V(s_d)$ denote the likelihood value at node s_d. Suppose $s_d, s_{d+1}, \ldots, s_L$

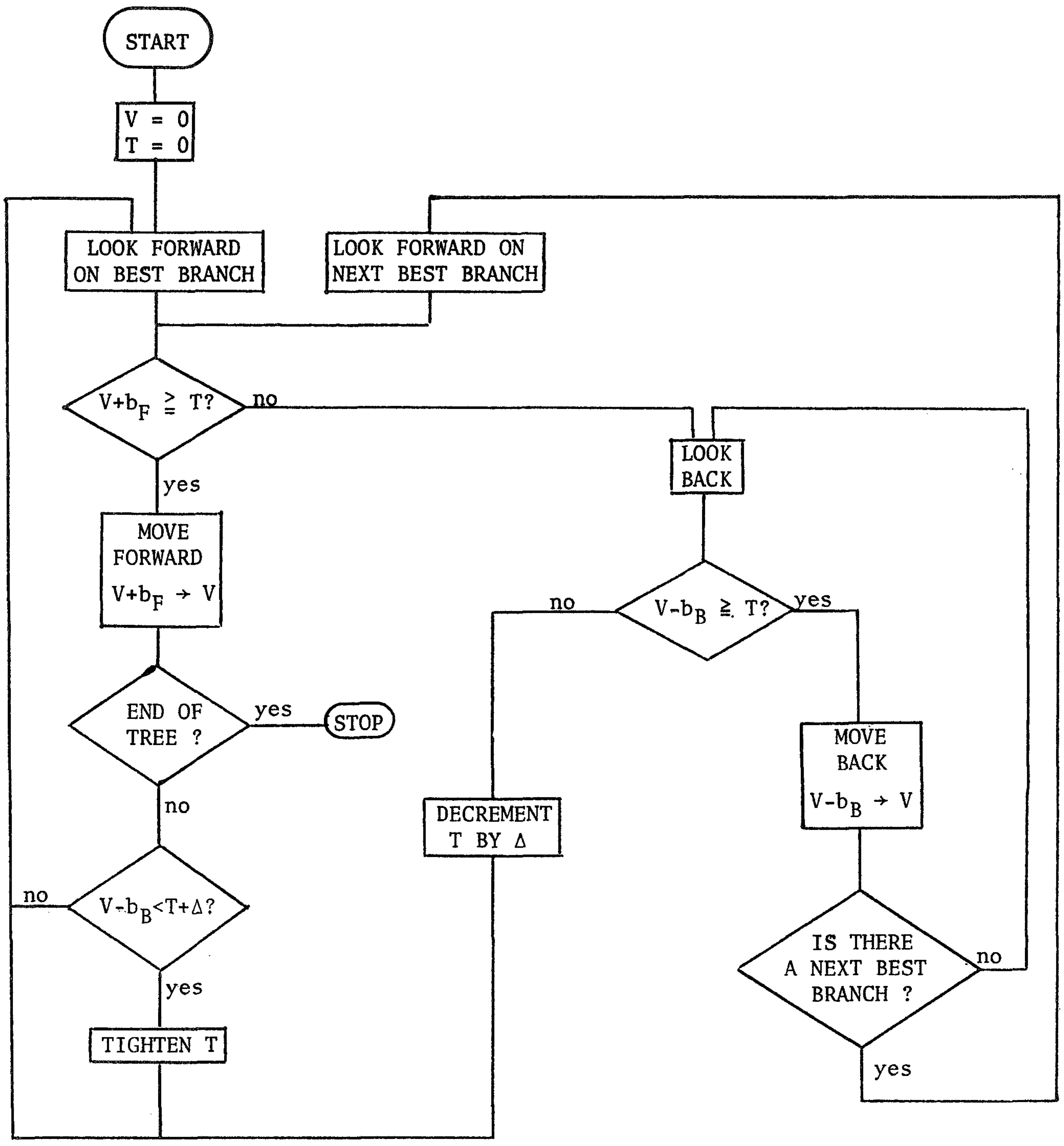


FIGURE 9. Fano Algorithm with Gallager's Threshold-tightening Test.

are the nodes along some path from s_d to the end of the tree. If we say that the path violates some threshold T if for some j, $d \leqq j \leqq L$, $V(s_j) < T$, and that the path satisfies every threshold it does not violate, then the tightest threshold satisfied by the path s_d, s_{d+1}, $\cdots$, s_L is

$$\left\lfloor \frac{1}{\Delta} \min_{d \leqq j \leqq L} V(s_j) \right\rfloor \Delta$$

where $\lfloor x \rfloor$ denotes the greatest integer not greater than x. The path selected by the Fano decoder is the path s_0^*, s_1^*, $\cdots$, s_L^* defined by the following iterative conditions:

F1; s_0^* is the origin node.

F2: For $0 \leqq i < L$, the branch from s_i^* to s_{i+1}^* is the first branch of that path from s_i^* to the end of the tree for which the tightest threshold satisfied is greatest; if two paths s_i^*, s_{i+1}, $\cdots$, s_L and s_i^*, s'_{i+1}, $\cdots$, s'_L are tied, then whichever of the two first branches is ordered higher is the branch from s_i^* to s_{i+1}^*.

Rules F1 and F2 specify a unique path through the tree, and this is the path the decoder will eventually select.

Having determined which path the decoder will choose, we can specify which nodes off that path are ever examined during decoding. For $0 \leqq b \leqq L$, let T_b^* denote the highest threshold satisfied by the final path segment from s_b^* to s_L^*; that is,

$$T_b^* = \left\lfloor \frac{1}{\Delta} \min_{b \leqq j \leqq L} V(s_j^*) \right\rfloor \Delta \qquad (1)$$

Let s_d be any node in level d of the tree not on the final path, and let s_b^* be the deepest node shared by the final path and the path to s_d. Then the path to s_d is $s_0^*, s_1^*, \cdots, s_b^*, s_{b+1}, \cdots, s_d$. The Fano decoder will move to s_d if

$$V(s_k) \geqq T_b^* + \Delta \quad \text{for all } k,\ b+1 \leqq k \leqq d$$

The decoder will not ever be positioned at s_d if

$$V(s_k) < T_b^* \quad \text{for some } k,\ b+1 \leqq k \leqq d$$

In the remaining case, namely $V(s_k) \geqq T_b^*$ for all k, $b+1 \leqq k \leqq d$, but for some such k, $V(s_k) < T_b^* + \Delta$, the decoder will search to s_d if the branch from s_b^* to s_{b+1} is ordered better than the branch from s_b^* to s_{b+1}^*.

Finally, the number of forward looks along the best branch b_1 stemming from a node s_i^* on the chosen path is

$$\left\lfloor \frac{V(s_i^*)}{\Delta} \right\rfloor - \left\lfloor \frac{1}{\Delta} \min_{i \leq j \leq L} V(s_j^*) \right\rfloor + 1$$

If the branches stemming from s_i^* are $b_1, b_2, \cdots, b_n$ in order of value and $s_{i+1}^{(k)}$ is the successor of s_i^* along b_k, then the number of forward looks along b_k, $k > 1$, is

$$\left\lfloor \frac{\min\{V(s_i^*),\ V(s_{i+1}^{(k-1)})\}}{\Delta} \right\rfloor - \left\lfloor \frac{1}{\Delta} \min_{i \leq j \leq L} V(s_j^*) \right\rfloor$$

if $s_{i+1}^{(k-1)} = s_{i+1}^*$ and one greater than this number otherwise.

2. A. 2. Computation Along the Final Path

A sensible definition of a "computation" for the Fano sequential decoder is an entry of either of the LOOK FORWARD boxes of Figure 9. In this section we investigate the number of computations performed at nodes along the final path. From our statements at the end of the last section, it is apparent that we should investigate the behavior of the function

$$Q(s_d^*) = V(s_d^*) - \min_{d \le j \le L} V(s_j^*)$$

where s_d^*, s_{d+1}^*, $\cdots$, s_L^* are the nodes on the segment of the final path extending from s_d^* to the end of the tree. Note that $Q(s_d^*) \geqq 0$.

It will be convenient to consider our branch metrics as being integer-valued. The actual metrics can be scaled and rounded to integers within any desired degree of accuracy, and in fact this would normally be done in a real decoder. We therefore assume that the branch metrics take on values in some finite set of integers $\{r_1, r_2, \cdots, r_M\}$. We study the function Q from two points of view. First, we introduce a graphical technique for computing $Q(s_d^*)$ at every node s_d^* when a particular channel error pattern is given; this method shows the manner in which the occurrence of clusters or bursts of errors compounds the difficulty in decoding. Second we consider $Q(s_d^*)$ as a random variable and study a statistical model for it.

<u>2.A.2.a. $Q(s_d^*)$ for given error patterns.</u> Suppose that $Q(s_d^*)$ is known and we wish to find $Q(s_{d-1}^*)$. Since the error pattern is given, we can find z_d, the value of the branch from s_{d-1}^* to s_d^*. We distinguish two cases:

Case 1: $Q(s_d^*) = 0$. In this case, $V(s_d^*) = \min_{d \leqq j \leqq L} V(s_j^*)$. If $z_d \geqq 0$, then $V(s_{d-1}^*) = V(s_d^*) - z_d \leqq \min_{d \leqq j \leqq L} V(s_j^*)$, so $V(s_{d-1}^*) = \min_{d-1 \leqq j \leqq L} V(s_j^*)$ and hence $Q(s_{d-1}^*) = 0$. If, on the other hand, $z_d < 0$, $V(s_{d-1}^*) > V(s_d^*)$, and therefore

$$\min_{d-1 \leqq j \leqq L} V(s_j^*) = \min_{d \leqq j \leqq L} V(s_j^*) = V(s_d^*).$$ Hence $Q(s_{d-1}^*) = -z_d$.

Case 2: $Q(s_d^*) > 0$. Now $V(s_d^*) > \min_{d \leq j \leq L} V(s_j^*)$. If $z_d \leq 0$, then

$V(s_{d-1}^*) \geq V(s_d^*)$ and hence $\min_{d-1 \leq j \leq L} V(s_j^*) = \min_{d \leq j \leq L} V(s_j^*)$.

Therefore
$$\begin{aligned} Q(s_{d-1}^*) &= V(s_{d-1}^*) - \min_{d-1 \leq j \leq L} V(s_j^*) \\ &= V(s_d^*) - z_d - \min_{d \leq j \leq L} V(s_j^*) \\ &= Q(s_d^*) - z_d. \end{aligned}$$

If $z_d > 0$, there are two possibilities. If $z_d \geq Q(s_d^*)$, then $V(s_{d-1}^*) = V(s_d^*) - z_d \leq \min_{d \leq j \leq L} V(s_j^*)$, and so $V(s_{d-1}^*) = \min_{d-1 \leq j \leq L} V(s_j^*)$ and $Q(s_{d-1}^*) = 0$. But for $z_d < Q(s_d^*)$, $V(s_{d-1}^*) > \min_{d \leq j \leq L} V(s_j^*)$, hence $\min_{d-1 \leq j \leq L} V(s_j^*) = \min_{d \leq j \leq L} V(s_j^*)$ and $Q(s_{d-1}^*) = Q(s_d^*) - z_d$.

Combining these observations and noting that

$$Q(s_L^*) = V(s_L^*) - \min_{L \leq j \leq L} V(s_j^*) = 0$$

we have the following result:

THEOREM 1: For a set of final branch metrics $z_1, z_2, \cdots, z_L$, the values $Q(s_d^*)$ along the final path are determined as follows:

(1) $Q(s_L^*) = 0$;

(2) $Q(s_{d-1}^*) = \max\{Q(s_d^*) - z_d, 0\}$, $1 \leq d \leq L$.

Let us restrict consideration now to cases in which the branch metrics can take on only one positive value; that is, $r_1 > 0$ and $r_j \leq 0$, $2 \leq j \leq M$. Under this assumption, by virtue of Theorem 1, we can quite easily construct a graph of $Q(s_d^*)$ by the following steps:

(1) Draw a horizontal line of length L units, each unit to represent a branch in the final path. The right end of the line represents node s_L^* and the left end represents s_0^*. Each dividing mark along the line represents one node.

(2) For each branch for which $z_d < 0$, label the branch with the value z_d.

(3) At the node preceeding the rightmost labelled branch, erect a vertical line whose height is the absolute value of the label. From the top of this line to the base line, draw a line whose slope is equal to the single positive branch value. This line intersects the base line at some point to the left of the vertical line.

(4) Working from right to left, at the node preceeding each labelled branch, erect a vertical line of appropriate height and draw a slanting line back to the base line. If at any of the labelled branches, slanting lines from nodes to the right extend back past the branch, the vertical line should be measured off from the highest such slanting line, and not from the base line.

(5) The upper boundary of the figure formed when step (4) has been performed for all labelled branches is a plot of $Q(s_d^*)$ versus d.

As an example, consider operating on the binary symmetric channel with $N = 2$, $L = 20$. Branch metrics have the form

$$z = (2-e)\, M_C + e\, M_E$$

where e is the number of discrepancies between the code symbols on the branch and the corresponding received symbols, and M_C and M_E are constants. Suppose $M_C = 1$ and $M_E = -8$, and suppose the error pattern is 00 00 00 00 00 00 00 01 00 00 00 00 00 11 00 10 00 00 01 00. Thus there is a double error in branch 14 and single errors in branches 8, 16, and 19. The graph of $Q(s_d^*)$ from steps (1)-(5) is shown in Figure 10.

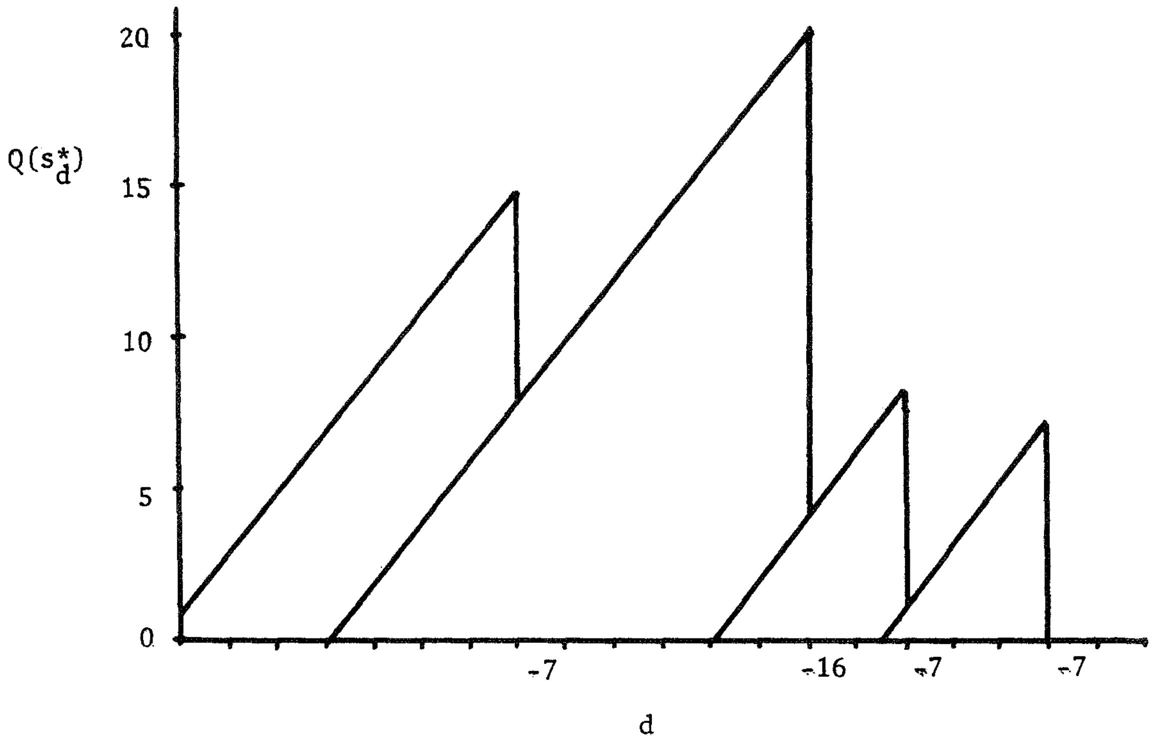


FIGURE 10. Graph of $Q(s^*_d)$ for the Example.

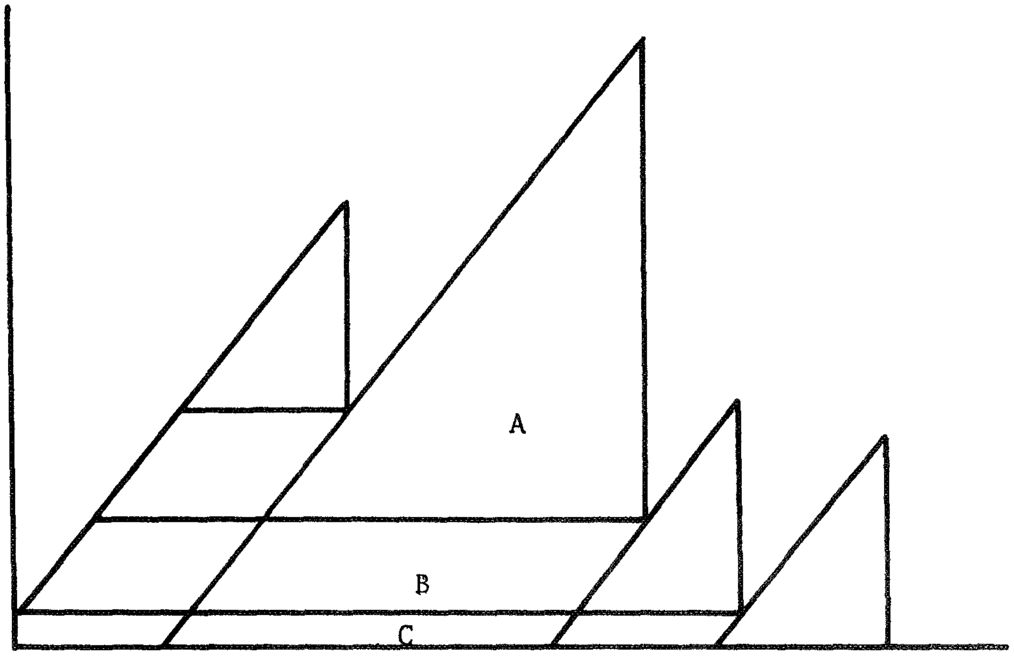


FIGURE 11. Figure Showing Effect of Error Clustering on Computation.

Since the number of final-path computations performed at s_d^* is related linearly (neglecting the effects of quantization) to $Q(s_d^*)$, it is evident that the total number of final-path computations is related linearly to the area of the figure produced in the construction. Moreover, the graph places in evidence the source of various computations, if we partition the figure as shown in Figure 11, by drawing a horizontal line from the bottom of each vertical line. The areas of triangles in the figure represent computations due to the presence of errors, regardless of other errors; if errors were separated by enough error-free branches, only triangles would appear in the graph. The extra computation required by proximity of errors is represented by parallelograms. For example, in Figure 11 the area of triangle A represents computations due to the double error in branch 14. The area of parallelogram B represents computations due to the nearness of the double error to the single error in branch 16. The area of parallelogram C represents computations due to the proximity of these two errors to the error in branch 19.

It is possible to repeal the restriction to metrics which assume only one positive value, but the price is a considerable loss of simplicity. Instead of drawing a line of fixed slope from the top of the vertical lines to the base line, it is necessary to draw a line whose slope over each branch is equal to the (positive) branch value. If the construction is carried out with this modification, a plot of $Q(s_d^*)$ results, and the analogous parts of the partitioned figure admit the same interpretation as above.

2.A.2.b. A statistical model for $Q(s_d^*)$. Suppose we fix some node s_d^* on the final path and investigate the random variable $Q(s_d^*)$. We allow the frame length L to grow without bound, so that s_d^* is arbitrarily far from the end of the tree, and hence the probability density functions of $Q(s_d^*)$ and $Q(s_{d+1}^*)$ are identical. We denote this common density function $P_Q(\cdot)$ and attempt to evaluate it.

We are assuming that the branch values lie in the set $\{r_1, r_2, \cdots, r_M\}$; it will simplify the analysis if we consider separately the positive and negative values. Let $\{n_1, n_2, \cdots, n_K\}$ be the positive values and $\{m_1, m_2, \cdots, m_J\}$ be the absolute values of the negative values, so that $\{r_1,\cdots,r_M\} = \{n_K, \cdots, n_1, -m_1,\cdots,-m_J\}$. Furthermore, let the n_k and m_j be indexed such that $n_1<n_2<\cdots<n_K$ and $m_1<m_2<\cdots<m_J$. Finally, let $p_j = \Pr\{z = -m_j\}$ and $q_k = \Pr\{z = n_k\}$. Thus $\sum_{j=1}^{J} p_j + \sum_{k=1}^{K} q_k = 1$.

For $Q(s_{d+1}^*) \geqq n_K$, $Q(s_d^*) = Q(s_{d+1}^*) - z_{d+1}$. Therefore, for $u \geqq n_K$,

$$\Pr\{Q(s^*) = u\} = \sum_{j=1}^{J} \Pr\{Q(s_{d+1}^*) = u-m_j\}p_j + \sum_{k=1}^{K} \Pr\{Q(s_{d+1}^*) = u+n_k\}q_k$$

Using the fact that $Q(s_d^*)$ and $Q(s_{d+1}^*)$ are identically distributed, we obtain, for $u = n_K$,

$$P_Q(u) = \sum_{j=1}^{J} P_Q(u-m_j)p_j + \sum_{k=1}^{K} P_Q(u+n_k)q_k \qquad (2)$$

Equation (2) suggests a representation in the form of a Markov chain whose typical states are as shown in Figure 12.

For $u < n_K$ the analysis is complicated, but in a fashion similar to the argument preceeding Theorem 1, it can be seen that the distribution $P_Q(u)$ in this range satisfies the equations arising from states

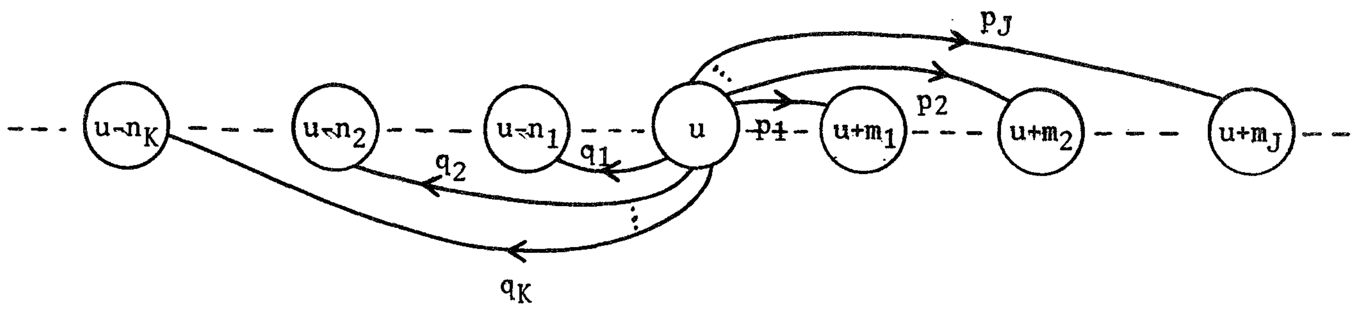


FIGURE 12. A Typical State in the Markov Chain Model for $Q(s_d^*)$.

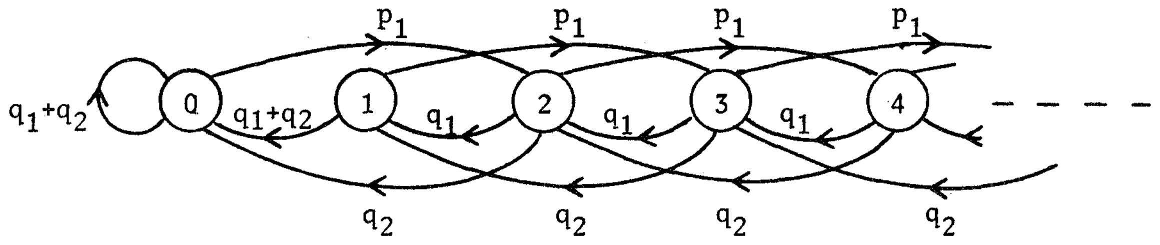


FIGURE 13. Markov Chain Model for $Q(s_d^*)$ for the Example.

identical to the one in Figure 12, except that any transition $u \to u-n_K$ which would "overshoot" the state u=0 is routed instead into the zero state. An example of such a chain with $K=2$, $J=1$, $n_1=1$, $n_2=2$, and $m_1=2$ is shown in Figure 13.

The equilibrium distribution $\{\pi_0, \pi_1, \cdots\}$ of the chain so constructed is the density function of $Q(s_d^*)$; i.e., $\pi_0 = P_Q(0)$, $\pi_1 = P_Q(1)$, etc. Hence if the $\{\pi_u\}$ can be found for this kind of chain, $P_Q(u)$ is determined. A partial solution is furnished by the following theorem:

THEOREM 2: Let $D = \sum_{j=1}^{J} m_j p_j - \sum_{k=1}^{K} n_k q_k$. (3)

$$\text{Then} \quad \sum_{k=1}^{K} q_k \sum_{u=0}^{n_k-1} (n_k-u)\pi_u = -D. \tag{4}$$

Proof: Let $d^+(u)$ be the upward drift from state u and $d^-(u)$ the downward drift. Then

$$d^+(u) = \sum_{j=1}^{J} m_j p_j$$

$$\text{and} \quad d^-(u) = \sum_{k=1}^{K} \min\{u, n_k\} q_k .$$

If $\{\pi_u\}$ is the equilibrium distribution of the chain, then the average values of $d^+(u)$ and $d^-(u)$ with respect to $\{\pi_u\}$ must be equal. That is,

$$\sum_{u=0}^{\infty} d^+(u)\pi_u = \sum_{u=0}^{\infty} d^-(u)\pi_u . \tag{5}$$

Now

$$\sum_{u=0}^{\infty} d^-(u)\pi_u = \sum_{u=0}^{\infty} \sum_{k=1}^{K} \min\{u, n_k\} q_k \pi_u$$

$$= \sum_{k=1}^{K} q_k \sum_{u=0}^{\infty} \min\{u, n_k\} \pi_u$$

$$\sum_{u=0}^{\infty} d^-(u)\pi_u = \sum_{k=1}^{K} q_k\Big(\sum_{u=0}^{n_k-1} u\pi_u + \sum_{u=n_k}^{\infty} n_k\pi_u \Big) \qquad (6)$$

Adding $\sum_{k=1}^{K} q_k \sum_{u=0}^{n_k-1} n_k\pi_u$ and its negative to (6) yields:

$$\sum_{u=0}^{\infty} d^-(u)\pi_u = \sum_{k=1}^{K} q_k \sum_{u=0}^{n_k-1} (u-n_k)\pi_u + \sum_{k=1}^{K} q_k n_k$$

Therefore from (5),

$$\sum_{k=1}^{K} q_k \sum_{u=0}^{n_k-1} (u-n_k)\pi_u = \sum_{u=0}^{\infty} \sum_{j=1}^{J} m_j p_j \pi_u - \sum_{k=1}^{K} q_k n_k = D$$

Changing the sign of both sides yields (4), proving the theorem.

In general, Theorem 2 expresses a linear relation which the equilibrium probabilities of the atypical states must satisfy. In the special case in which K = 1 and $n_1 = 1$, Theorem 2 simplifies to:

COROLLARY: For chains with a single downward transition of unit magnitude,

$$\pi_0 = \frac{D}{q_1} \qquad (7)$$

That is, if there is only one atypical state, π_0 can be found immediately from (7). Then the remaining π_u can be found successively by solving the recurrence equations. Therefore, when there is a single positive metric value and the values can be normalized so that the positive metric has unit value, then the density $P_Q(u)$ can be found exactly for all u.

The Markov chain model for $Q(s_d^*)$ was originally proposed by Massey[14], who also proved the above corollary directly. Theorem 2 arose out of attempts to generalize Massey's technique to apply to chains with more than one atypical state. Such attempts are, however, doomed to failure. For chains of this type with $K > 1$ atypical nodes, it is impossible to find K independent equations by means of these drift-balancing techniques. To see this, note that the equations derived are always linear equations in $\pi_0, \pi_1, \cdots, \pi_{K-1}$ whose coefficients are integer multiples of p_j and q_k. Now suppose all the p_j and q_k are rational. If there were a set of K independent linear drift equations, then the solutions would again be rational. But there exist chains with rational p_j and q_k and irrational π_u. For example, the chain of Figure 14 has $\pi_u = (1-\beta)\beta^u$, $0 \leq u < \infty$, where $\beta = -\frac{1}{2} + \sqrt{7/12}$.

Although it is not always possible to get an exact solution for $\{\pi_u\}$ using the drift-balancing method, one can always solve for $\{\pi_u\}$ numerically by solving a difference equation of the form (2). The solution has the general form:

$$\pi_u = \sum_{i=1}^{m_J+n_K} C_i \beta_i^u \tag{8}$$

where the eigenvalues β_i are the roots of the equation obtained by substituting $\pi_u = \beta^u$ into (2), that is,

$$\sum_{k=1}^{K} q_k \beta^{m_J+n_k} - \beta^{m_J} - \sum_{j=1}^{J} p_j \beta^{m_J-m_j} = 0 \tag{9}$$

By Descartes' Rule of Signs, there are either <u>two</u> positive real roots of (9) or <u>none</u>. Since by inspection $\beta = 1$ is a root, say $\beta_1 = 1$, there must be exactly one other positive real root, say β_2. Now since $\sum \pi_u = 1$, β_1 cannot be active; that is, $C_1 = 0$. The other positive real root, β_2, is active, however, and is the dominant eigenvalue. As $u \to \infty$, $\pi_u \to K\beta_2^u$. Thus we always have $P_Q(u) \to \bar{K}e^{-ru}$ for some $\bar{K}$ and r, a fact

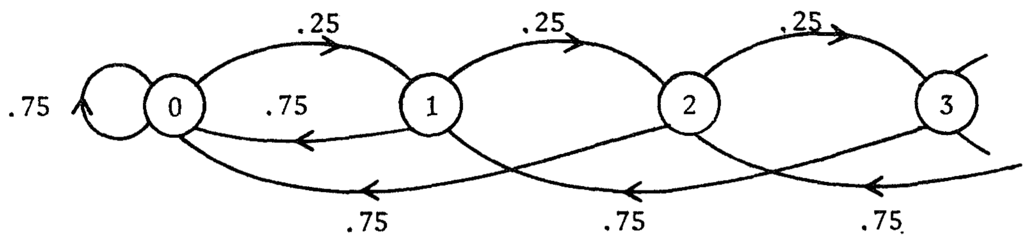


FIGURE 14. A Markov Chain with Rational Transition Probabilities and an Irrational Eigenvalue.

alluded to in Chapter I. (This fact does not depend on the choice of integer-valued metrics, however. See Gallager ([9], pp. 312 ff.) for a proof based on random-walk arguments that is valid for any metric of the form (I-1).)

A class of chains in which it is particularly simple to find the exact solution for the equilibrium probabilities $\pi_0, \pi_1, \pi_2, \cdots$ once the dominant eigenvalue has been obtained, are those with only one upward transition of unit magnitude, $J = 1$, $m_1 = 1$. The chain of Figure 14 is an example of such a chain, and like the chain of Figure 14, all such chains have a single active eigenvalue: $\pi_u = K\beta^u$ for all u. (Since $\sum \pi_u = 1$, $K = 1-\beta$.) To prove this*, we show that ${}^{\pi_{k+1}}/_{\pi_k}$ is independent of k. Suppose we have a long sequence of numbers which represent the trajectory of a particle behaving as specified by the chain of Figure 15a. Let N be the length of the sequence, and let N_k be the number of occurrences of k in the sequence. Then as $N \to \infty$, $N_k/N \to \pi_k$. Now for every adjacent pair (n,0) in the sequence representing a move from state n to state 0 which is an "overshoot" move, that is, one which would have been longer than n steps is there had been room, let us place a symbol E between the n and the 0. This increases N but does not affect any N_k. The new sequence is a sample trajectory from the chain of Figure 15b. Now consider adding after each E in the new sequence j more Es, where j = 0 with probability p, j = 1 with probability p(1-p), $\cdots$, j = i with probability $p(1-p)^i$. Again N has been increased, but no N_k has been changed, and the new sequence is a sample trajectory from the chain of Figure 15c. Now the chain of Figure 15c is exactly the same chain as that of Figure 15a,

* This simple proof was suggested by Massey[15].

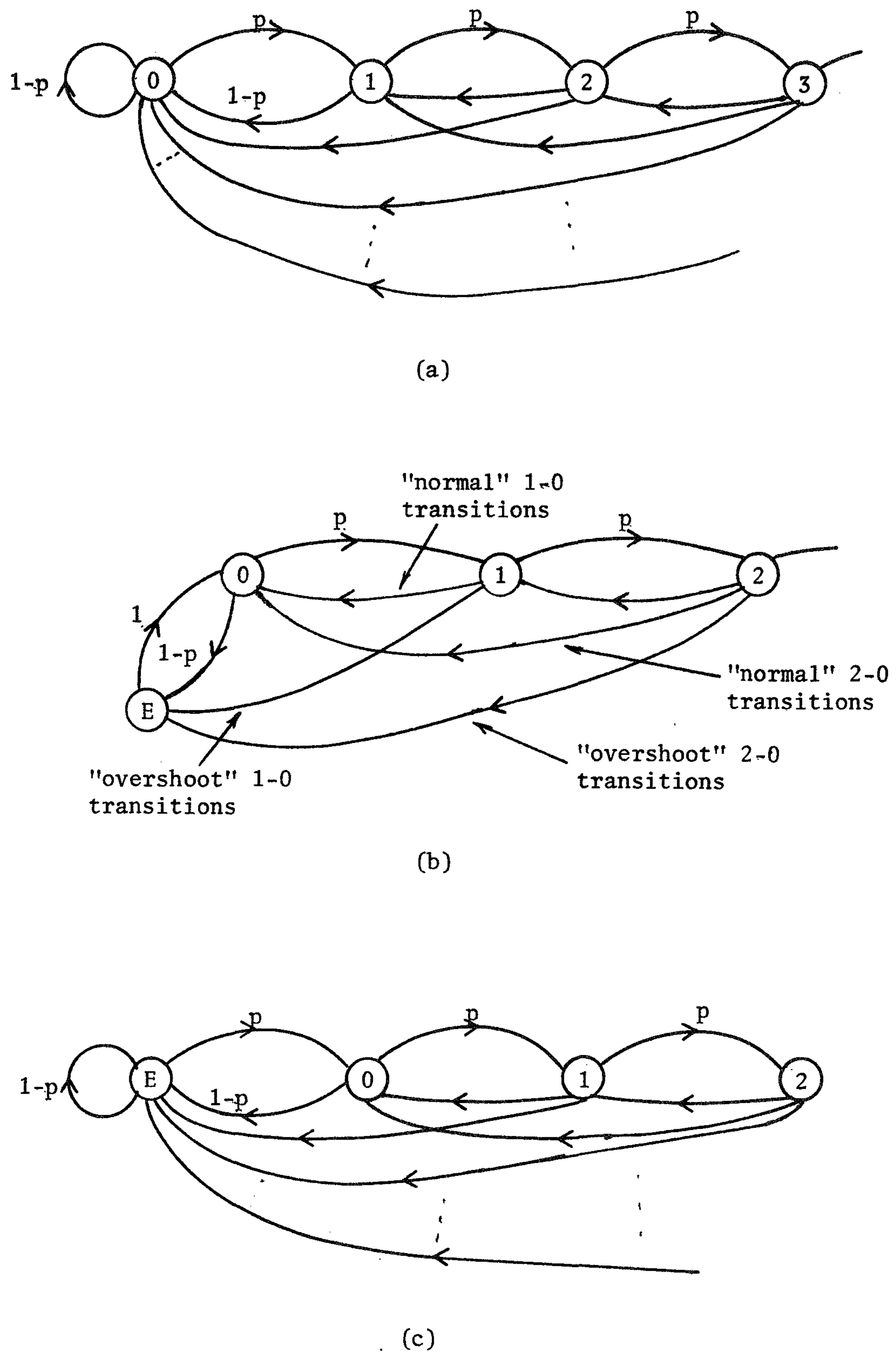


FIGURE 15. Steps in the Proof of the Single Active Eigenvalue Property.

with state E in the former corresponding to state 0 in the latter, and state k in the former corresponding to state k+1 in the latter, $k \geqq 0$. Let N_{ex} denote the length of the sequence we have constructed by adding Es. Looking at the chain of Figure 15a, we have

$$\frac{\pi_{k+1}}{\pi_k} = \frac{N_{k+1}/N}{N_k/N} = \frac{N_{k+1}}{N_k}$$

But from the chain of Figure 15c, we conclude

$$\frac{\pi_{k+2}}{\pi_{k+1}} = \frac{N_{k+1}/N_{ex}}{N_k/N_{ex}} = \frac{N_{k+1}}{N_k}$$

Therefore $\pi_{k+1}/\pi_k = \pi_{k+2}/\pi_{k+1}$ and hence $\pi_u = (1-\beta)\beta^u$ for all $u \geqq 0$.

2. B. THE ZIGANGIROV-JELINEK ALGORITHM

Despite its relative ease of implementation, the Fano algorithm is conceptually a complicated scheme. Recently, in independent work, Zigangirov[6] and Jelinek[7] have suggested a sequential decoding algorithm which is quite simple, and which exhibits more clearly the essential nature of sequential decoding. In addition, when the algorithm is suitably modified, its performance is in many cases superior to that of the Fano algorithm, provided that sufficient memory is available for use by the decoder. In this section we describe the Zigangirov-Jelinek decoder and compare some of its properties to those given previously for the Fano algorithm.

The decoder consists of an ordered list of nodes, called a stack, in which the nodes are listed in decreasing order of likelihood values. Thus the "top" node is (one of) the node(s) of greatest value among the nodes on the stack. The stack is initially loaded with the origin node, whose value is taken to be zero, and is processed according to the following rules:

(1) Compute the values of the successors of the top node and add them to the stack in the places determined by their values.

(2) Delete the node whose successors were just added.

(3) If the new top node is in the last level of the tree, stop. Otherwise, go to (1).

When the algorithm halts, the node at the top of the stack determines the path selected. Taking a computation to be an execution of step (1), the number of computations required to decode a tree is one less than the size of the stack when the decoder halts.

We state two conditions which are satisfied by at least one path through the tree, and then undertake to prove that the Zigangirov-Jelinek decoder selects one of these paths. For a path s_d, s_{d+1}, $\cdots$, s_L, $0 \leqq d \leqq L$, define the minimum value of the path to be $\min_{d \leqq j \leqq L} V(s_j)$. The conditions are:

ZJ1: s_0^* is the origin node.

ZJ2: For $0 \leqq i < L$, the branch leading from s_i^* to s_{i+1}^* is the first branch of one of the paths from s_i^* to the end of the tree having greatest minimum value.

Conditions ZJ1 and ZJ2 do not specify a unique path through the tree as do F1 and F2, because the action in case of ties in ZJ2 is not specified. It will be seen that the problem of resolving ties among path minima with the Zigangirov-Jelinek decoder is quite complicated, depending not only on how ties are resolved in the stack, but upon the values of certain nodes along the paths concerned as well. We will be content to show that the Zigangirov-Jelinek decoder selects one of the paths satisfying ZJ1 and ZJ2.

Let s_d, $0 \leqq d \leqq L$, be a node in the tree, and suppose there is a path $s_d, s_{d+1}, \cdots, s_L$ from s_d to the end of the tree such that

$$V(s_d) \leqq V(s_j), \quad d \leqq j \leqq L$$

Then s_d is called a breakout node*. If

$$V(s_d) < V(s_j), \quad d < j \leqq L \tag{10}$$

then we will call s_d a strict breakout node.

LEMMA: If s_d is a strict breakout node and s_d reaches the top of the stack, then s_d is on the final path (i.e., the path selected by the decoder).

Proof: Let s' be the second node on the stack when s_d reaches the top of the stack. Then

$$V(s') \leqq V(s_d) \tag{11}$$

Let $s_d, s_{d+1}, \cdots, s_L$ be a path from s_d to the end of the tree for which (10) holds. Now after s_d is extended, s_{d+1} is on the stack, and by (10) and (11), s_{d+1} is above s', and hence s_{d+1} is extended before s' is. Thus s_{d+2} appears on the stack, and as before, s_{d+2} is above s', and so on out to s_L. Thus some successor of s_d at the end of the tree appears at the top of the stack, which means that s_d is on the final path, as claimed.

THEOREM 3: The path selected by the Zigangirov-Jelinek decoder is a path satisfying ZJ1 and ZJ2.

---

* This terminology is due to Gallager[9].

Proof: Note first that the minimum value along a path from s_i to the end of the tree is the value of the first strict breakout node among s_i, s_{i+1}, $\cdots$, s_L (s_L is a strict breakout node, so there is at least one).

Let s_0^*, s_1^*, $\cdots$, s_L^* be a path satisfying ZJ1 and ZJ2. Suppose $s_{B_1}^*$ is the first strict breakout node along the path. In Figure 16 let the other paths represent any paths emanating from the nodes shown, and let the nodes $s^{(j)}$ be the first strict breakout nodes along those paths.

Since the path s_0^*, $s_1^*, \cdots$, s_L^* satisfies ZJ2,

$$V(s_{B_1}^*) \geqq V(s^{(j)}), \quad 0 \leqq j \leqq B_1-1 \tag{12}$$

Since $s_{B_1}^*$ is the first strict breakout node,

$$V(s_j^*) \geqq V(s_{B_1}^*), \quad 0 \leqq j \leqq B_1$$

Suppose first that (12) holds with strict inequality for all j. Now $s^{(j)}$ cannot appear on the stack until s_j^* has been extended, and thus until s_{j+1}^* has appeared on the stack. Since $V(s_{j+1}^*) \geqq V(s_{B_1}^*) > V(s^{(j)})$, s_{j+1}^* will reach the top of the stack before $s^{(j)}$ does. (It may reach the top of the stack before $s^{(j)}$ even appears on the stack.) Hence s_{j+2}^* will appear on the stack before $s^{(j)}$ reaches the top of the stack, and so will be above $s^{(j)}$, and so on. Thus $s_{B_1}^*$ will appear on the stack before $s^{(j)}$ reaches the top, and by (12), $s_{B_1}^*$ will reach the top of the stack before $s^{(j)}$ does. Now j is arbitrary, and one of the $s^{(j)}$ or $s_{B_1}^*$ must eventually reach the top of the stack, so it must be $s_{B_1}^*$. Hence by the Lemma, s_0^*, s_1^*, $\cdots$, $s_{B_1}^*$ is the initial segment of the selected path.

In the event that for some j, $V(s_{B_1}^*) = V(s^{(j)})$, then a path from

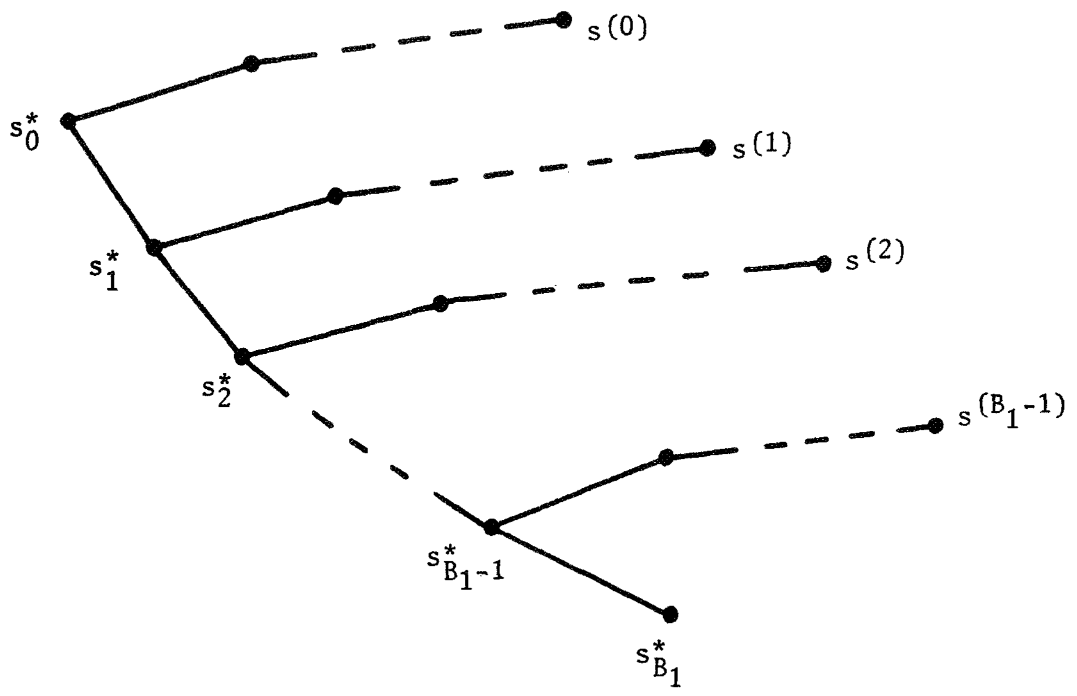


FIGURE 16. The Initial Segment of the Selected Path and Some Branching Paths.

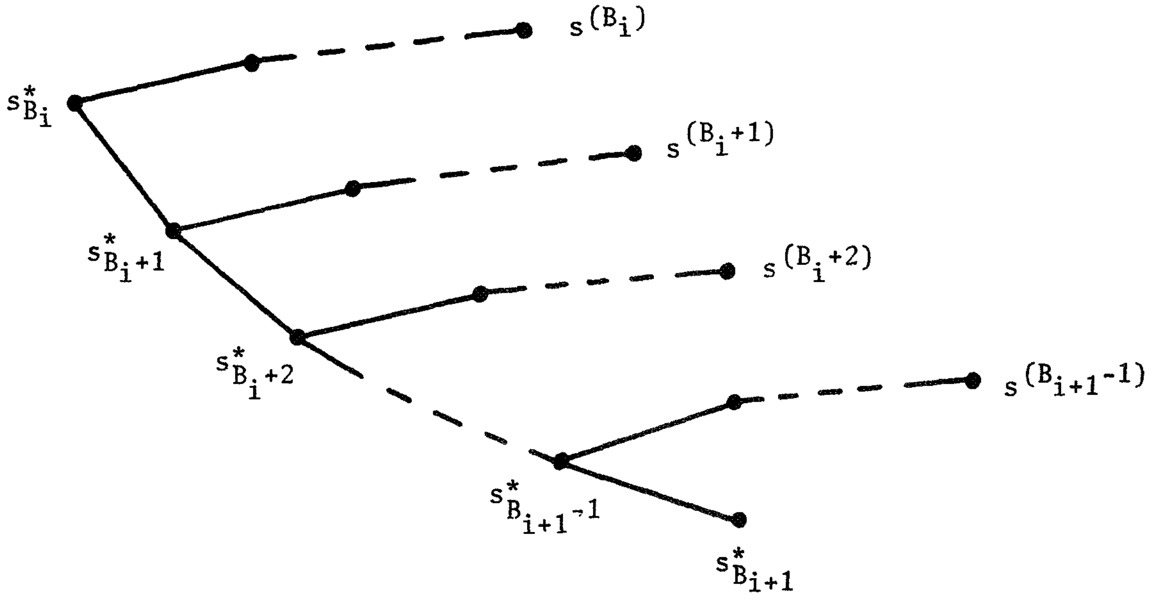


FIGURE 17. An Intermediate Segment of the Selected Path and Some Branching Paths.

s_0^* to the end of the tree passing through $s^{(j)}$ also satisfies ZJ1 and ZJ2. The same argument as above shows that the decoder will select one of the two paths; i.e., either $s^{(j)}$ or $s_{B_1}^*$ will reach the top of the stack first, but which one depends on how ties are resolved in the stack, and on the values of the nodes between s_j^* and $s^{(j)}$ and the nodes between s_j^* and $s_{B_1}^*$.

Now consider Figure 17, which is similar to Figure 16 except that the origin node is replaced by the ith strict breakout node, $s_{B_i}^*$, and $s_{B_{i+1}}^*$ is the (i+1)st strict breakout node. If $s_{B_i}^*$ reaches the top of the stack, the Lemma guarantees that $s_{B_i}^*$ is on the final path, so any processing of nodes which are below $s_{B_i}^*$ when the latter reaches the top of the stack cannot affect the final choice of path. Thus we may consider the processing of these nodes as independent of the processing of the nodes shown in Figure 17. Then the same argument as above applies to show that $s_{B_{i+1}}^*$ reaches the top of the stack before any $s^{(j)}$, barring ties.

Combining the arguments shows that the successive strict breakout nodes along some path satisfying ZJ1 and ZJ2 are nodes on the final path. Since the terminal node s_L^* is a strict breakout node, this shows that the path selected is one which satisfies the conditions, and the theorem is proved.

Now let $s_0^*, s_1^*, \cdots, s_L^*$ be the path selected by the Zigangirov-Jelinek decoder. If s_B^* is the first strict breakout node among the nodes $s_d^*, s_{d+1}^*, \cdots, s_L^*$, then $V(s_B^*) = \min_{d \le j \le L} V(s_j^*)$. Now consider Figure 18, where $s_d^*, s_{d+1}, \cdots, s_{d+k}$ is any other path emanating from s_d^*. Suppose $V(s_{d+j}) > V(s_B^*)$, $1 \leqq j < k$, and $V(s_{d+k}) < V(s_B^*)$. Then

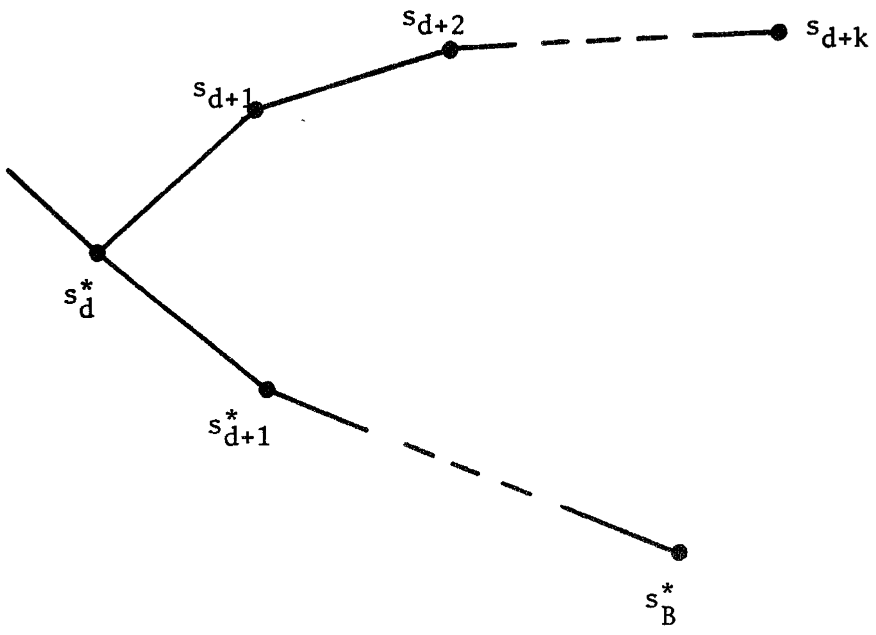


FIGURE 18. A Segment of the Selected Path and a Branching Path.

since s_B^* is on the final path, it eventually reaches the top of the stack, but since each s_{d+j}, $1 \leqq j < k$, has value greater than $V(s_B^*)$, all of these nodes reach the top of the stack before s_B^*. Since $V(s_{d+k}) < V(s_B^*)$, s_B^* reaches the top of the stack before s_{d+k} and since s_B^* is a strict breakout node, s_{d+k} never reaches the top of the stack--there will always be a node s_i^* on the stack such that $V(s_i^*) > V(s_B^*) > V(s_{d+k})$. This proves the following theorem concerning the extending of nodes not on the final path:

THEOREM 4: With the notation of Figure 18,

(1) s_{d+k} reaches the top of the stack if

$$V(s_{d+j}) > \min_{d \leqq i \leqq L} V(s_i^*), \quad 1 \leqq j \leqq k;$$

(2) s_{d+k} does not reach the top of the stack if for any such j,

$$V(s_{d+j}) < \min_{d \leqq i \leqq L} V(s_i^*) \ .$$

The possibility not covered in the theorem, namely $V(s_{d+j}) \leqq \min_{d \leqq i \leqq L} V(s_i^*)$, $1 \leqq j \leqq k$, with equality for some such j, depends, as before, on the resolving of ties in the stack and on intermediate node values.

The Zigangirov-Jelinek algorithm, while of theoretical interest, is impractical because of the time required to keep the stack ordered exactly. Jelinek[7] has proposed a quantized version of the algorithm in which nodes are placed in bins according to their values. That is, for some $H > 0$, a node s is placed in bin k if $kH \leqq V(s) < (k+1)H$. The use of bins rather than an ordered stack obviates the need to search the stack when nodes are to be added. The steps in the modified algorithm are as follows:

(1) Select any node from the highest non-vacant bin, compute the values of its successors, and place them in the proper bins.

(2) Delete the node whose successors were just added.

(3) If any node in the highest non-vacant bin is in the last level of the tree, stop. Otherwise, go to (1).

Extensive computer simulation studies, reported in the next chapter, indicate that the Jelinek algorithm is in most cases of interest superior to the Fano algorithm, provided that large storage capability is provided the decoder.

We state without proof the following theorems, analogous to Theorems 3 and 4, and proved similarly.

THEOREM 5: The path s_0^*, s_1^*, $\cdots$, s_L^* selected by the Jelinek decoder satisfies the following two conditions:

J1: s_0^* is the origin node.

J2: For $0 \leqq i \leqq L-1$, the branch leading from s_i^* to s_{i+1}^* is the first branch of one of the paths from s_i^* to the end of the tree whose lowest-value node belongs in the highest bin.

THEOREM 6: Let s_{d+k} be a node in level d+k of the tree, not on the selected path, and let s_d^* be the deepest node shared by the selected path and the path to s_{d+k}. Let s_{j1}^* be a node such that $V(s_{j1}^*) = \min_{d \leqq j \leqq L} V(s_j^*)$. Then s_{d+k} reaches the top of the stack if every node s_{d+i}, $1 \leqq i \leqq k$, belongs in a higher bin than s_{j1}^*; s_{d+k} does not reach the top of the stack if for any such i, s_{d+i} belongs in a lower bin than s_{j1}^*.

It is evident that J2 is simply a quantized version of ZJ2. Moreover, the quantization of node values into bins of width H corresponds to the quantization by thresholds (1) with the Fano algorithm; F2 and J2 show that, except for the effect of ties, the Fano and Jelinek decoders choose the same path through the tree. In addition, the set of nodes examined by the two decoders is the same, again excepting the effect of ties. As H and Δ are increased, of course, the effect of ties becomes more pronounced.

2. C. A NEW UNQUANTIZED FANO ALGORITHM

In the previous section we described the Zigangirov-Jelinek algorithm and a modification, the Jelinek algorithm, based on quantizing node values. We saw that, except for the effect of quantization, the two algorithms have identical search patterns. In addition, it was noted that the bin width H in the Jelinek algorithm plays the same role as the threshold increment Δ in the Fano algorithm, and that, excluding the effect of ties among quantized node values, the set of nodes examined, and in particular the final paths for these two algorithms coincide.

In view of this it is natural to wonder if there is some algorithm which, like the Zigangirov-Jelinek algorithm, is unquantized, but which, like the Fano algorithm, substitutes a back-and-forth search capability for the requirement of node storage. In this section we give a modification of the Fano algorithm which fills this vacant place among the algorithms.

Before describing the algorithm, we shall make some more remarks about the metric trees which these algorithms are meant to search. Consider drawing contours in the tree, as shown in Figure 19. These contours have an analogous interpretation to that of contours on a

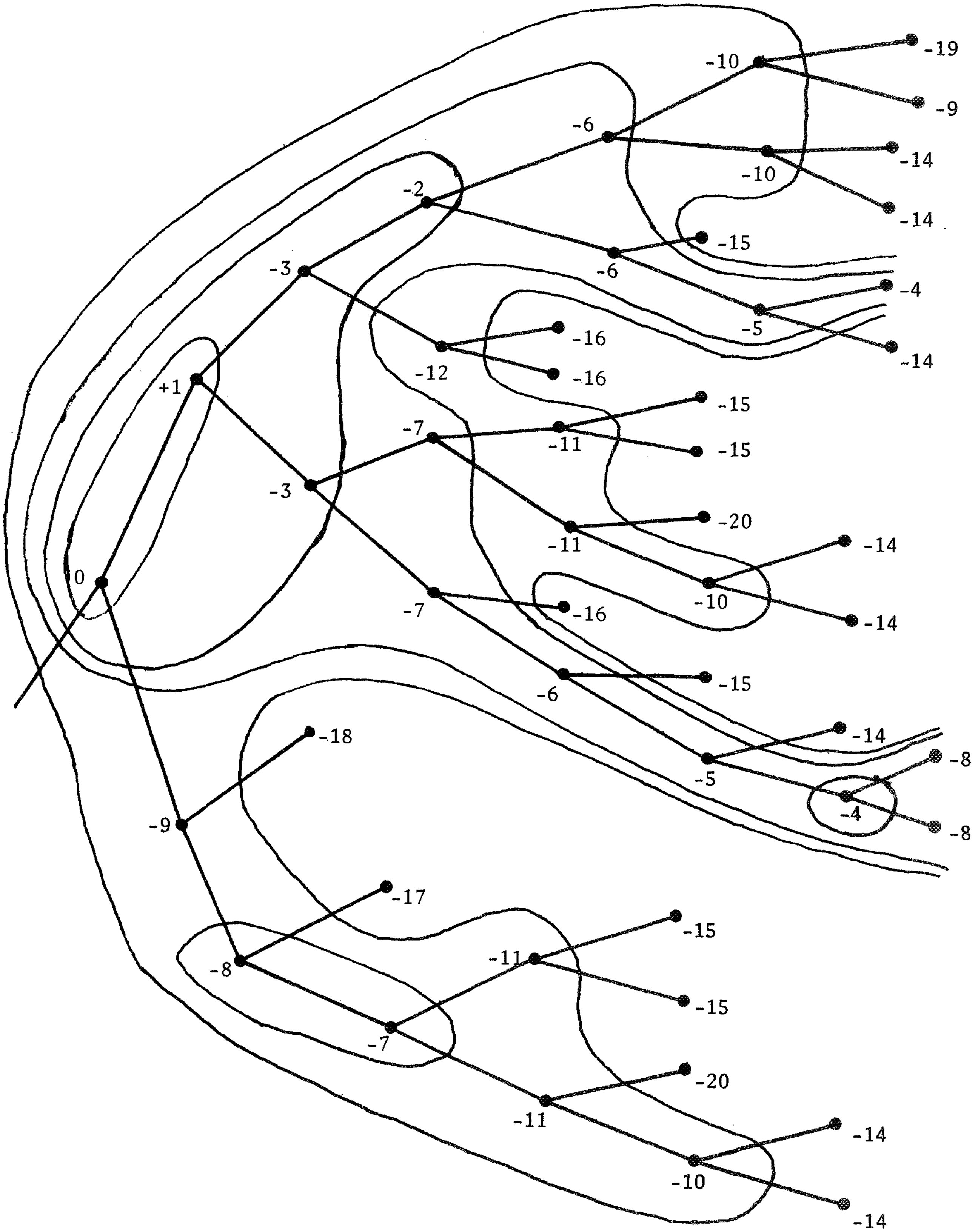


FIGURE 19. A Metric Tree Showing Contours Drawn for Values 0, -4, -8, and -12.

topographical map; namely, each node in the region enclosed by a contour has value no less than the value assigned to the contour. It is clear that among the branches crossed by the contour, exactly one is directed _into_ the region enclosed, and all others are directed _out of_ it. This applies to contours enclosing the origin node as well as all others, since we are adding to the tree a branch of infinite metric leading into the origin node, as discussed on page 19. Now suppose that the contours are drawn at intervals of Δ. A little thought about the Fano algorithm will lead to the following conclusions:

(1) The decoder can move across a contour into a region unimpeded, but once in the region, it cannot move out of it in any direction until the threshold is lowered.

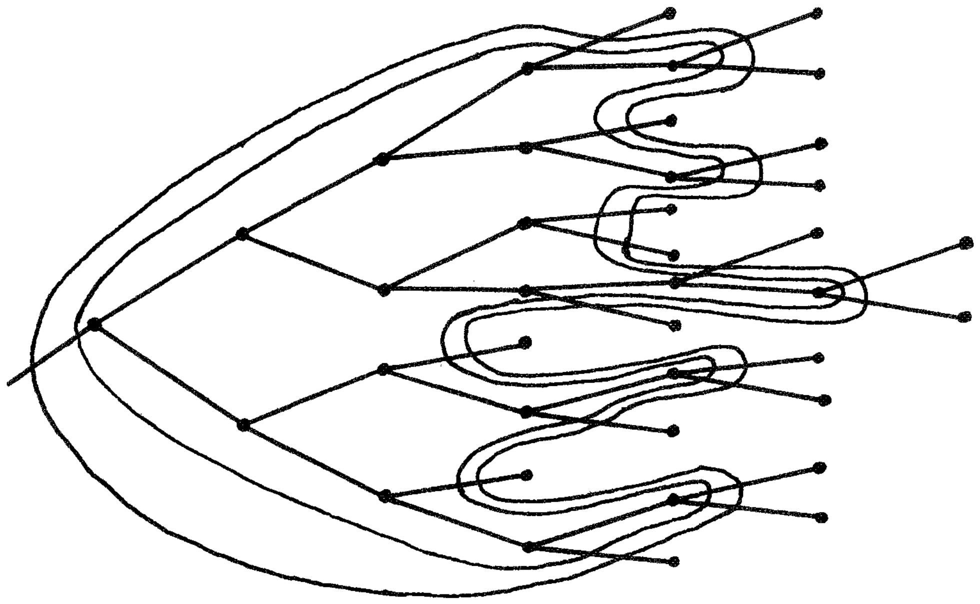
(2) The threshold is tightened the first time the decoder crosses the contour headed into the region.

(3) Before decreasing the threshold and leaving a region, the decoder will visit every node in the region.

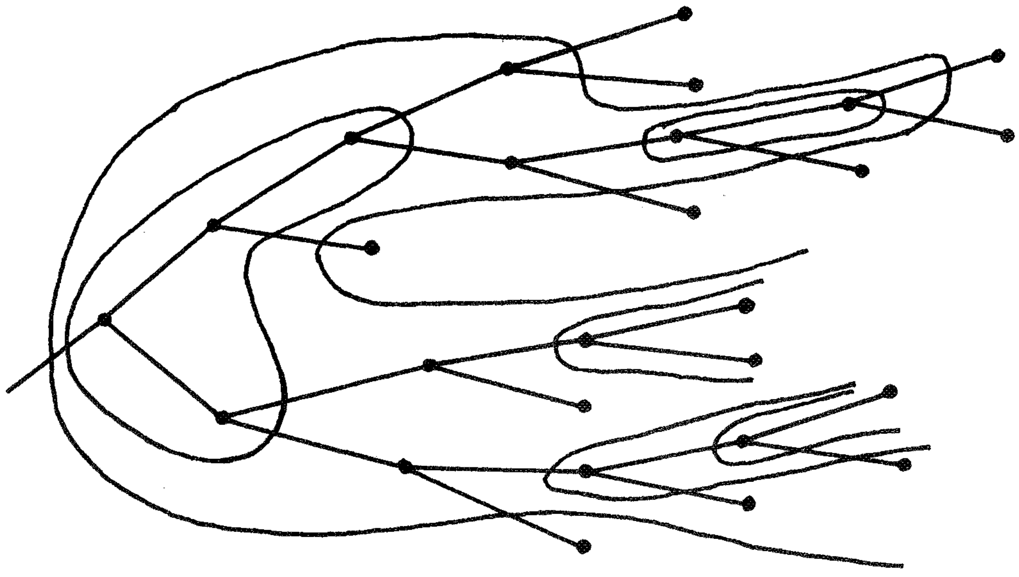
(4) The threshold is lowered only at the node at the end of the branch which leads into the region, i.e., the first node in the region that the decoder visited.

(5) After the decoder has departed from the region enclosed by a contour, that contour has no effect on subsequent searches.

Two annoying possibilities for metric tree configurations are shown in Figure 20. In Figure 20a, there are no nodes in the region between contours T and T-Δ. Hence after the decoder searches the region enclosed by contour T, it lowers the threshold to T-Δ and proceeds to search exactly the same set of nodes again, so that nothing is gained by lowering the threshold to T-Δ. In Figure 20b,



(a)



(b)

FIGURE 20. Two Undesirable Metric Tree Configurations.

on the other hand, a great many nodes, and even new enclosed regions, lie between T and T-Δ, so that lowering the threshold from T to T-Δ makes many new moves possible. When Δ is chosen small, situations like Figure 20a are common, while for large Δ, the configurations of Figure 20b occur quite often.

The new algorithm is based on the principle of avoiding these unpleasant extremes; that is, in lowering the threshold we wish to avoid (1) lowering it so little that no new moves are possible, and (2) lowering it so much that a plethora of new moves is presented. Our approach will again amount to searching in contours in accordance with (1)-(5) above, but the contours will be drawn, not in intervals of Δ, but to correspond to the exact values of the nodes. That is, if T_1 and T_2 are adjacent contours, with $T_1 < T_2$, then in the region between T_1 and T_2 are nodes of value T_1 and only such nodes. For example, the tree of Figure 19 is redrawn in Figure 21, this time with contours corresponding to the exact node values instead of in increments of 4. It is clear that the situation in Figure 20a cannot occur when the contours are so drawn, and that the situation in Figure 20b can occur only if there are many nodes of the same value just outside a contour.

To search the tree, each time a proposed forward move is blocked (that is, the move would lead out of a contour which has not been fully searched), the decoder notes the value of the inaccessible node and saves the largest such value, K. When the region within the contour has been completely searched and the decoder returns to the first node in the region, the threshold must be lowered. The Fano decoder would lower by Δ and undertake to move forward again, without guarantee

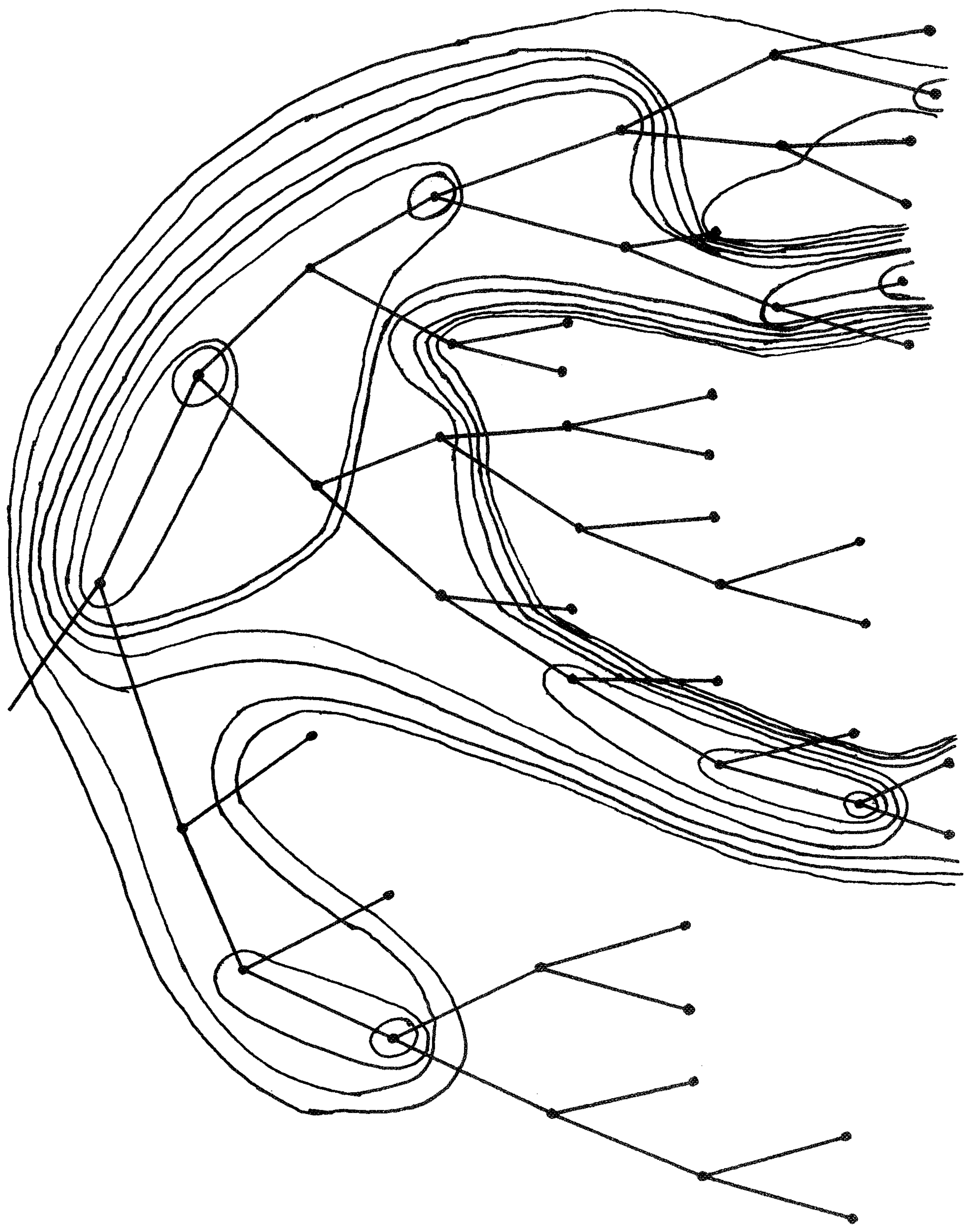


FIGURE 21. The Metric Tree of Figure 19 With Contours Drawn Corresponding to Exact Node Values Between 0 and -10.

of success. The unquantized decoder knows that lowering to K will assure the possibility of a new move forward; but before doing so, it considers the possibility of a backward move. If the value of the predecessor of the present node is greater than K, the threshold is lowered to that value. Recognizing that forward searching with this threshold is fruitless, the decoder immediately moves backward. If, on the other hand, K equals or exceed the value of the predecessor, the threshold is lowered to K and forward searching resumes. In either case, the threshold is lowered just enough to escape from the present contour.

A flowchart of the unquantized Fano algorithm is given in Figure 22. The only differences between this and the original Fano algorithm (Figure 8) are the maintenance of the forward blocking value K, the routine for lowering the threshold, and the absence of the threshold incerment Δ. In the new algorithm, the tightening of T is done exactly; that is, T is set equal to the node value after the first move to a node.

We now study the action of this algorithm in more detail. Following Massey and Sain[13], we first investigate the search properties of the algorithm when applied to "trunks," or degenerate trees in which each node (except the last) has a single successor. In this case, the flowchart can be simplified to that of Figure 23.

Consider the trunk of Figure 24. Each V_j is the value at the corresponding node s_j, and V_0 is taken to be zero. For a fixed j, $0 \leqq j \leqq L$, define $U_k^{(j)} = \min_{j \leq i \leq k} V_i$, $j \leqq k \leqq L$. Then we have $U_{k+1}^{(j)} \leqq U_k^{(j)}$, $j \leqq k \leqq L$. Let $k_1, k_2, \cdots, k_m$ be the values of k such that $U_{k_i}^{(j)} < U_{k_i-1}^{(j)}$. Then $U_{k_1}^{(j)}, U_{k_2}^{(j)}, \cdots, U_{k_m}^{(j)}$ are all the

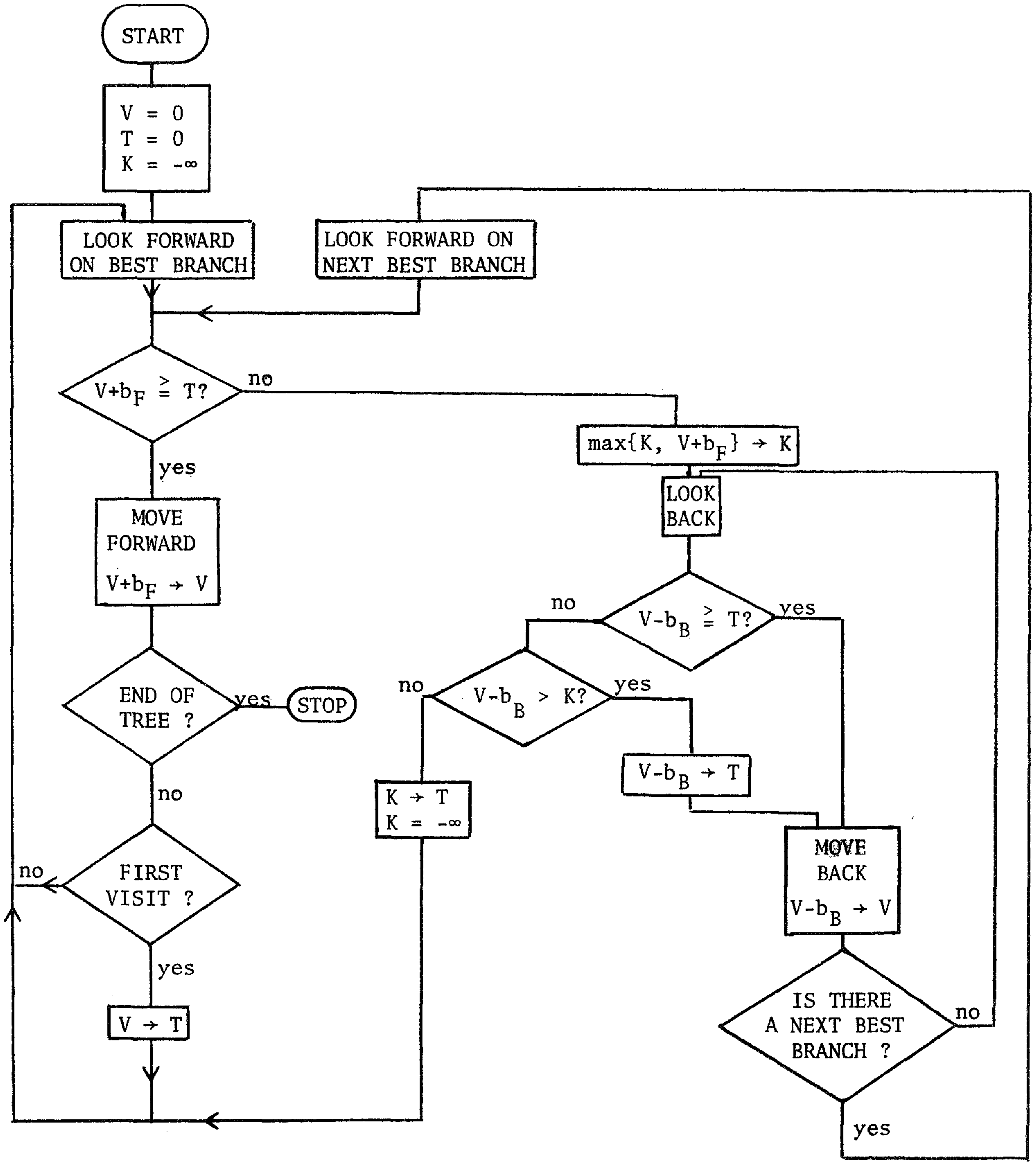


FIGURE 22. Unquantized Fano Sequential Decoding Algorithm.

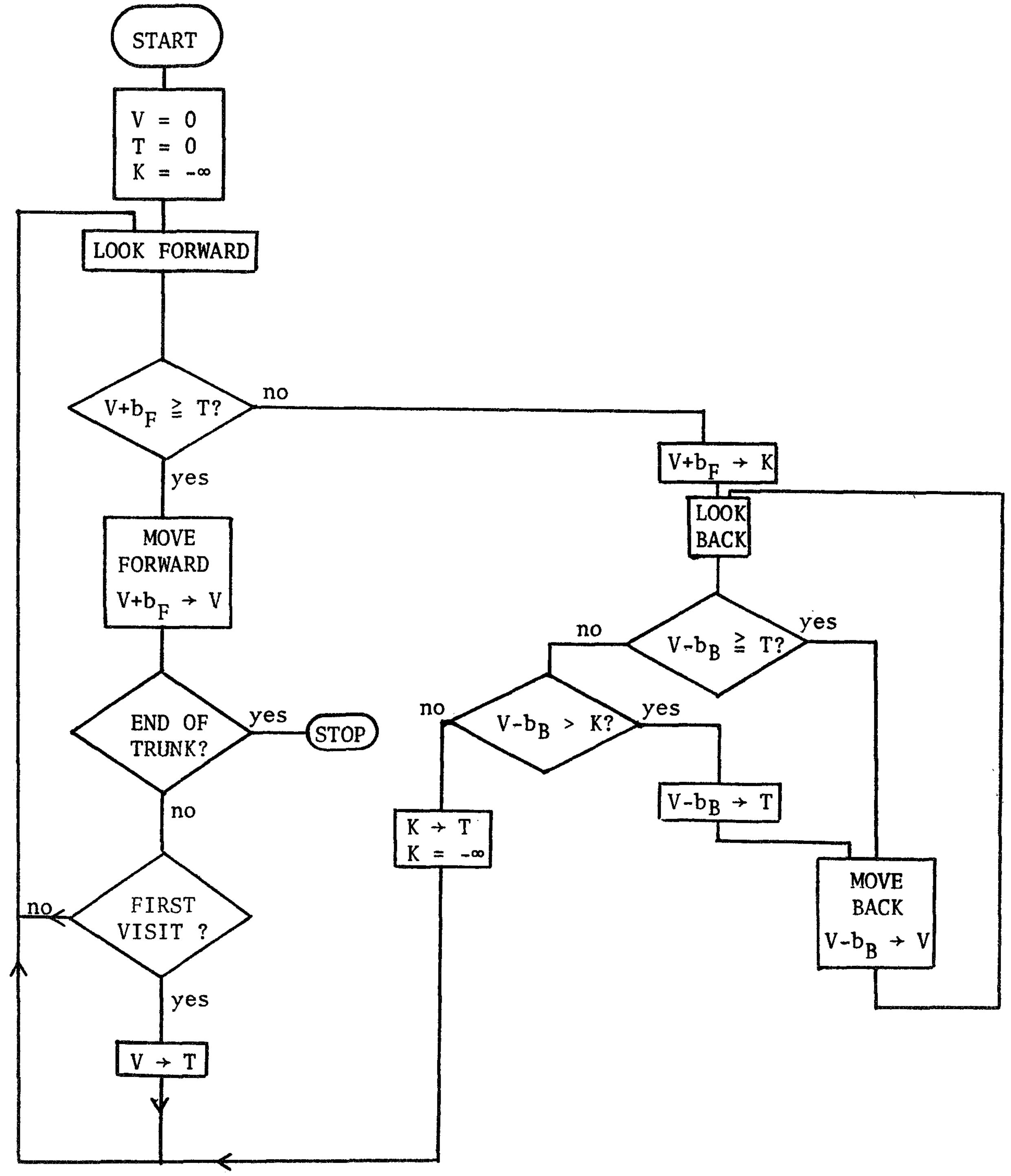


FIGURE 23. Unquantized Fano Trunk Search Algorithm.

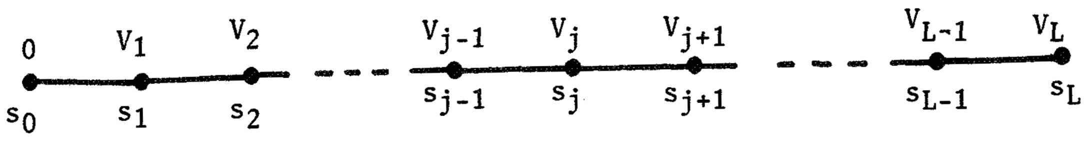


FIGURE 24. A Metric Trunk.

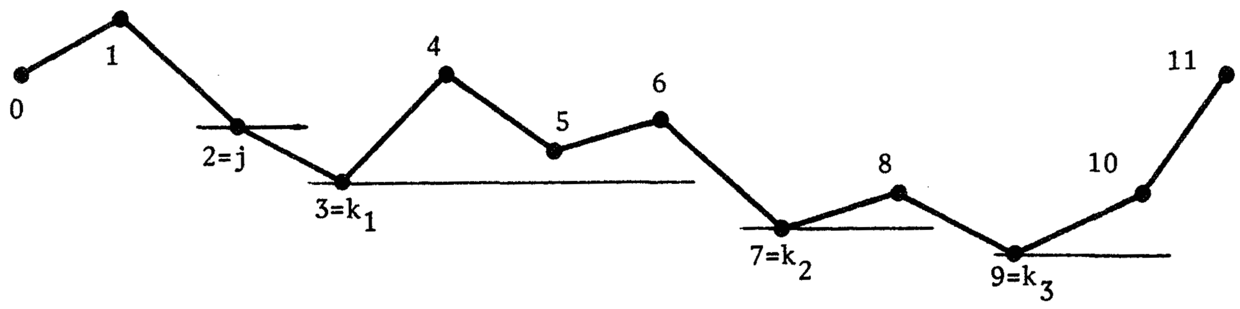


FIGURE 25. Determination of k_i for a Typical Metric Trunk.

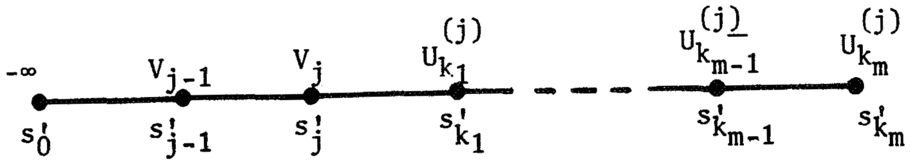


FIGURE 26. A Trunk Equivalent to That of Figure 24 Between s_{j-1} and s_j.

distinct values assumed by $U_k^{(j)}$, and moreover, $U_{k_1}^{(j)} = V_{k_1}, \cdots, U_{k_m}^{(j)} = V_{k_m}$. As a clarifying example, see Figure 25.

LEMMA 1: With respect to the action of the decoder between nodes s_{j-1} and s_j, the trunk of Figure 26 is equivalent to that of Figure 24.

Note: We will refer to the trunk of Figure 24 as P, and to the trunk of Figure 26 as P'. The nodes of P' are labelled with the subscripts of the nodes in P which have the same value, so that subscripts in P' do not indicate the depth of the node.

Proof of Lemma 1: First consider the case when $V_j < V_{j-1}$. Let $k_0 = j$ and $U_{k_0}^{(j)} = V_j$. Upon first moving to s_{j-1} in P and to s'_{j-1} in P', the threshold is tightened; that is, $T = V_{j-1}$. In searching P, a forward look (F look) is made, K is set to V_j, and back looks and moves (B looks and B moves) are made until either the origin is reached or a node of value less than or equal to K is seen, at which time T is set to $K = V_j$ and F searching proceeds, including an F look and an F move from $s_{j-1}$to s_j. Up to this point, two F looks, one with $T = V_{j-1}$ and one with $T = V_j$, and one F move with $T = V_j$ have been made from s_{j-1} to s_j. By inspection, this agrees with the action of the decoder in searching P' up to the first move to s'_j.

Now suppose the action of the decoder on P up to the first move to s_{k_i} agrees with that on P' up to the first move to s'_{k_i}. In P, all nodes between s_{k_i} and $s_{k_{i+1}}$ (exclusive) have values no less than V_{k_i}, and F moves are preferred in case of ties, so the decoder moves to the node preceeding $s_{k_{i+1}}$ without a return to s_{k_i}. At that time, K is set to $V_{k_{i+1}}$ and B searching begins and continues past s_j toward s_0 until the origin is reached or a node of value not exceeding K is seen, when T is decreased to K and F searching resumes, culminating in a move to $s_{k_{i+1}}$. The decoder action between s_{j-1} and s_j between the first move to s_{k_i} and the first move to $s_{k_{i+1}}$ is a B look and a B move from s_j to s_{j-1} with $T = V_{k_i}$ and an F look and an F move from s_{j-1} to s_j with $T = V_{k_{i+1}}$. By inspection, this is the same action as in P' for the corresponding span of moves. By induction, the lemma is proved for the case $V_j < V_{j-1}$.

For $V_j > V_{j-1}$, note that we may have $U_{k_i}^{(j)} \geqq V_{j-1}$ for the first few i. Let i* be the least index i such that $U_{k_{i*}}^{(j)} < V_{j-1}$. We first show that P and P' are equivalent between the (j-1)st and jth nodes up to the first move to $s_{k_{i*}}$. Starting at s_{j-1} with $T = V_{j-1}$, an F look and an F move are made to s_j, and T is set to V_j. Searching to s_{k_1-1} proceeds without return to s_j, but at s_{k_1-1}, K is set to V_{k_1} and the decoder searches back to s_j and looks back to s_{j-1} with $T = V_j$. Since $K \geqq V_{j-1}$, T is lowered to K and F searching begins. This is repeated at each s_{k_i-1} up to the node preceeding $s_{k_{i*}}$. At that point, K is set to $V_{k_{i*}}$ and B searching to s_j is performed as before. This time, however, $K < V_{j-1}$, so T is lowered to V_{j-1}, a B move to s_{j-1} is made, and B searching continues until T is lowered to K. Then F searching resumes and the decoder moves to $s_{k_{i*}}$ for the first time. The total contribution to searching between s_{j-1} and s_j is:

an F look from s_{j-1} to s_j with $T = V_{j-1}$;

an F move from s_{j-1} to s_j with $T = V_{j-1}$;

i* B looks from s_j to s_{j-1} with $T = V_j, V_{k_1}, \cdots, V_{k_{i*-1}}$;

a B move from s_j to s_{j-1} with $T = V_{j-1}$;

an F look from s_{j-1} to s_j with $T = V_{k_{i*}}$;

an F move from s_{j-1} to s_j with $T = V_{k_{i*}}$.

Again by inspection, this is exactly the action of the decoder on P' for the same period. The induction step is exactly as before, proving the lemma for this case.

It remains to consider the case $V_j = V_{j-1}$. Letting $k_0 = j$, in both P and P' there is one F look and one F move from s_{j-1} to s_j with $T = V_{j-1} = V_j$ up until the first move to $s_{k_0} = s_j$. Then by induction as before, the lemma follows for this final case.

From the proof of Lemma 1, we can count the number of operations performed:

LEMMA 2: Using the notation developed above, the numbers of F looks and F moves from s_{j-1} to s_j and B looks and B moves from s_j to s_{j-1} is as follows:

	$V_j < V_{j-1}$	$V_j > V_{j-1}$	$V_j = V_{j-1}$
F looks	m+2	m+2-i*	m+1
F moves	m+1	m+2-i*	m+1
B looks	m	m	m
B moves	m	m+1-i*	m

By invoking Lemma 1 and considering the action of the decoder on trunks of the form of Figure 26, we conclude:

LEMMA 3: If an F move is made from node s_j with threshold T, then the next B move to s_j (if there is one) is also made with threshold T.

Lemma 3, which Massey and Sain[13] call the "superposition property," is significant in that it allows us to extend the analysis to tree searches. Note that a consequence of Lemma 3 is that the threshold is always lowered to a value T at the last point at which it was raised to the value T.

Consider the diverging paths of Figure 27. By Lemma 1, to study the action of the decoder at node s_j we may replace both the primed and unprimed paths by appropriate paths of decreasing node value. Let $k_1, k_2, \cdots, k_m$ be the depths at which $U_k^{(j+1)}$ assumes its values along the unprimed path, and $\ell_1, \ell_2, \cdots, \ell_n$ the depths for the primed path. Then we may replace Figure 27 by Figure 28.

If the decoder reaches s_{k_m} in Figure 28, then it will reach s_L in Figure 27, and if it reaches s'_{ℓ_n} in Figure 28, then it will reach s'_L in Figure 27. Thus to determine which path is chosen, given that the choice is between these two, we must see which of the nodes s_{k_m} or s'_{ℓ_n} is reached first.

Suppose the branch from s_j to s_{j+1} is "better" than the branch from s_j to s'_{j+1}; that is, either $V_{j+1} > V'_{j+1}$ or $V_{j+1} = V'_{j+1}$ and the decoder somehow resolves the tie in favor of s_{j+1}. We distinguish several cases:

If $\underline{V_{k_m} \geqq V_j}$ (which requires $V_{j+1} \geqq V_j$), after the first F move to s_{j+1} the decoder never returns to s_j. Hence the unprimed path is selected, and in fact, no F look is ever made along the primed path. In this case, the minimum along the unprimed path seen from s_j is V_j, while along the primed path it is, of course, at most V_j.

If $\underline{V_j > V_{k_m} \geqq V'_{\ell_n}}$, the threshold must eventually be lowered to V_{k_m}; by Lemma 3, this lowering must take place at or before s_j. After the threshold is so lowered, the decoder proceeds out to s_{k_m} and hence the unprimed path is selected. F moves are made along the primed path

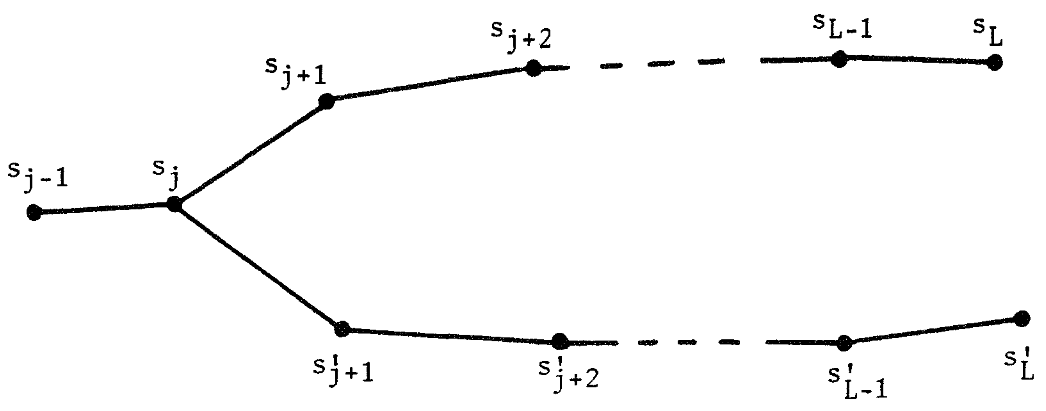


FIGURE 27. Branching Metric Trunks.

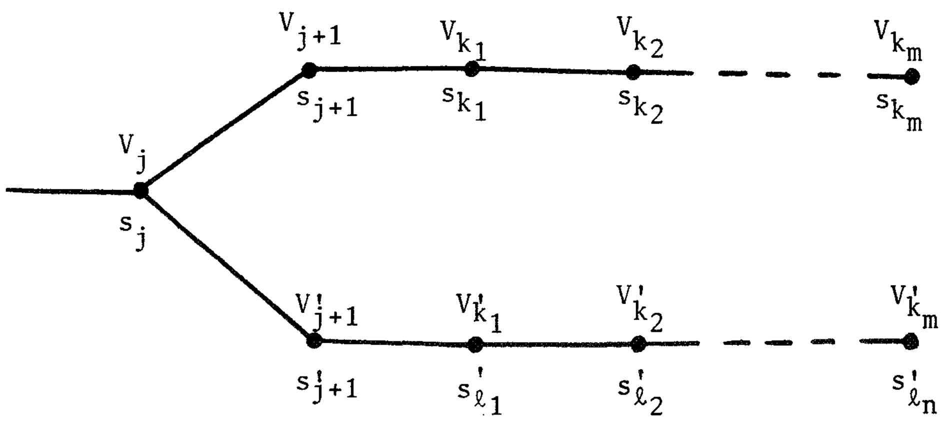


FIGURE 28. Branching Metric Trunks Equivalent to Those of Figure 27.

to all nodes s'_{ℓ_i} such that $V'_{\ell_i} > V_{k_m}$. In this case the minimum seen along the unprimed path from s_j is V_{k_m} and the minimum seen along the primed path is V'_{ℓ_n}.

If $\underline{V_j > V'_{\ell_n} > V_{k_m}}$, the threshold is eventually lowered to V'_{ℓ_n} at or before s_j, and the decoder proceeds out the unprimed path with this threshold in force. Unable to move to the end, it returns to s_j and and proceeds out the primed path (with the same threshold, by Lemma 3), to the end. Therefore, the primed path is chosen, and F moves are made to all nodes s_{k_i} on the unprimed path such that $V_{k_i} \geqq V'_{\ell_n}$. The minimum seen along the unprimed path is V_{k_m} and along the primed path is V'_{ℓ_n}.

If $\underline{V'_{\ell_n} \geqq V_j > V_{k_m}}$, then after the first move to s'_{j+1} no return is made to s_j. Hence the primed path is chosen. Nodes in the unprimed path are examined if their values equal or exceed V_j. The minimum along the primed path is V_j and along the unprimed path is V_{k_m}.

The cases considered above are exhaustive, and we may apply the reasoning to every branching point in the tree, so we have proved the following theorems:

THEOREM 7: The path $s^*_0, s^*_1, \cdots, s^*_L$ through the tree found by the unquantized Fano decoder is defined by the following conditions:

MF1: s^*_0 is the origin node.

MF2: For $0 \leqq i \leqq L-1$, the branch leading from s^*_i s^*_{i+1} is the first branch of that path from s^*_i to the end of the tree having greatest minimum value; if two paths with different first branches are tied, then whichever of the two first branches is better is the branch from s^*_i to s^*_{i+1}.

THEOREM 8: Let s_d be any node in level d of the tree, not on the final path, and let s_b^* be the deepest node shared by the final path and the path to s_d. The decoder will be positioned at s_d sometime during decoding if and only if

$$V(s_k) \geqq \min_{b \leqq j \leqq L} V(s_j^*) \quad \text{for all } k, \; b \leqq k \leqq d$$

with the provision that if equality holds for any k, the branch from s_b^* to s_{b+1} must be better than the branch from s_b^* to s_{b+1}^*.

The only difference between conditions MF2 and F2 is that the latter is quantized. Therefore the Fano algorithm may by considered to be a quantized version of this modified Fano algorithm. This observation carries over to the searching of nodes off the final path as well. In addition, MF2 and ZJ2 differ only in that ties are resolved in a definite manner in MF2 while they are left arbitrary in ZJ2. Therefore, except for the effect of ties, the modified Fano algorithm and the Zigangirov-Jelinek algorithm select the same path. The search patterns for non-final nodes also agree, except for differences due to ties. Therefore this unquantized Fano algorithm is an algorithm of the kind we sought in our remarks at the beginning of the section.

Although it was simple to count the number of looks and moves during a trunk search (Lemma 2), the problem is much more complex when trees are being searched. To count the number of F looks, say, at a node s_d^* on the correct path, we must find the number of distinct values greater than $\min\limits_{d \leqq j \leqq L} V(s_j^*)$ assumed by the $U_k^{(d)}$ along every path emanating from s_d^*. The threshold will be lowered at or before s_d^* once for each of these values, and each such lowering will contribute some F looks along various branches. Because of the complexity, we will omit discussion of the problem in general, but we will consider a special case and compare the computation of the modified algorithm to that of the conventional Fano algorithm in this case.

Suppose the branch metrics assume values in a finite set of integers, and furthermore that all the values are integer multiples of the one of least absolute value; that is, the admissible metric values are $\{r, n_1 r, n_2 r, \cdots, n_N r\}$ where r and all the n_i are integers (positive or negative). As a consequence of this choice of metrics, every node value has the form $V = kr$, k an integer. Suppose we choose $\Delta = |r|$. Then the only difference between a metric tree with contours drawn for the Fano algorithm (i.e., in increments of Δ) and the same tree with contours drawn for the modified Fano algorithm (i.e., at exact node values) is that some contours may appear in the former which do not appear in the latter, namely those arising in situations as in Figure 20a. Consequently the number of F looks made by the unquantized algorithm is no greater than the number made by the Fano algorithm, the only differences being attributable to one of the following reasons:

(1) Reduction of T by Δ does not allow escape from a contour (as in Figure 20a).

(2) Reduction of T by Δ allows a B move, but no new F move. The Fano decoder, after reducing T, makes all possible forward moves before attempting the B move, while the modified decoder makes the B move immediately.

(3) A block near the origin occurs so that T must be lowered at the origin. The Fano decoder lowers by Δ repeatedly until T is low enough, while the unquantized decoder lowers it enough the first time.

Note: If $r > 0$ and all $n_i < 0$, so that r is the only positive metric, (1) cannot occur. If in addition, when T is lowered to the back node value in the modified algorithm, control passes to LOOK FORWARD ON BEST BRANCH rather than to MOVE BACK, the differences in computation counts due to (2) will disappear. Thus we can make the two algorithms almost equivalent in number of computations, being forced to accept differences only due to (3). This applies regardless of what measure of computation we choose.

The superiority of the new algorithm over the conventional one seen above does not extend to the general case. In particular, the choice of Δ given is a poor one from the computational viewpoint. It is possible to choose a larger Δ which usually results in many fewer computations with a negligible degradation in error probability. See Chapter III for further discussion.

Finally, we wish to show that the test "V = T?" before an F move is a sufficient test for the need to tighten the threshold. Consider the first move to s_{j+1} along the trunk of Figure 24, and assume the test works properly up to s_j. Then after the first move to s_j, $T = V_j$. If $V_{j+1} \geqq V_j$, then the first move to s_{j+1} follows immediately. Since $T = V_j$, the threshold will be tightened at s_{j+1}. If $V_{j+1} < V_j$, K is set to V_{j+1} and the decoder moves back until T is reduced to K. Then F searching begins and the first move to s_{j+1} is made with $T = V_{j+1}$. In this case, the test "V = T?" fails at s_j just before the move to s_{j+1}, but there is no need to tighten the threshold. Therefore, if the test works up to s_j then it works up to s_{j+1}. Now since the decoder is started at the origin with $T = V = 0$, if $V_1 \geqq 0$ the test indicates that T must be tightened, and if $V_1 < 0$, T is reduced to V_1

and the first move to s_1 is made. Thus the test works for the first move to s_1, and hence by induction, the test is suitable for trunk searching. By Lemma 3, the analysis extends to tree searching, and so Figure 29 is equivalent to Figure 22. Note that the test "$V = T$?" is a limiting case (as $\Delta \to 0$) of the test "$V < T+\Delta$?" in Figure 9.

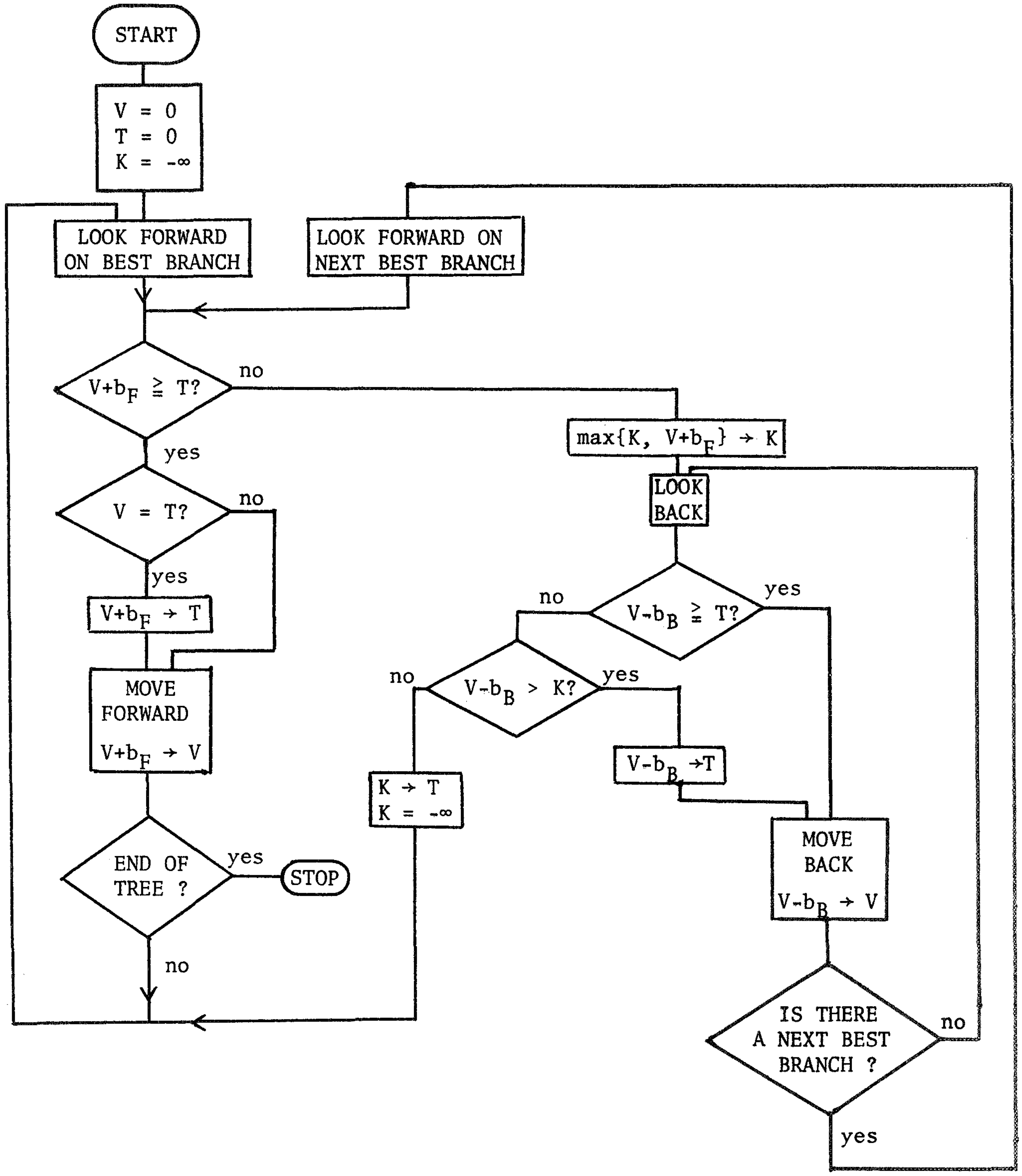


FIGURE 29. Unquantized Fano Sequential Decoding Algorithm with Alternative Threshold-tightening Test.

CHAPTER III

EXPERIMENTAL COMPARISON OF SEQUENTIAL DECODING ALGORITHMS

The Zigangirov-Jelinek, Jelinek, and unquantized Fano algorithms were programmed for the UNIVAC 1107 computer at the University of Notre Dame Computing Center. These programs were used in conjunction with existing programs for the Fano decoder and for simulating channel disturbances to study the performance of the various decoding systems. In this chapter we review the results of this study.

All the results to be presented were obtained using a rate ½ non-systematic convolutional code of memory 35 constructed by Costello[16]. This code has a number of important and useful properties, one of which is large free distance, which makes it well-suited to sequential decoding. In all the frames run in obtaining the data, no decoding error was made with this code, although in several very noisy frames, the computation became prohibitive, and decoding was discontinued, the frames being counted as erasures. The generator polynomials for the code (in octal form) are:

$$G^{(1)} = 533533676737$$

$$G^{(2)} = 733533676737$$

Runs were made on the binary symmetric channel at three noise levels and on the binary-input Gaussian noise channel with three-bit (eight-level) output quantization (see Figure 30). Some parameters of interest for the channels used are given in Table 1. In Table 1, ρ is the solution of $R_\rho = R = \frac{1}{2}$, and $R_{comp} = R_1$.

TABLE 1

Channel	R_{comp}	R/R_{comp}	ρ
BSC, p=.033	0.5593	0.89	1.354
BSC, p=.045	0.4996	1.00	0.998
BSC, p=.057	0.4504	1.10	0.729
Gaussian	0.4913	1.02	0.944

Branch values are computed by adding the values for the two digits on the branch. The digit values are determined by comparing the code symbols with the corresponding received symbol. The digit metrics are listed in Table 2. The values correspond to the terms $\log\frac{\Pr\{r|x\}}{f(r)} + B$ in equation (I-1), rounded to integer values.

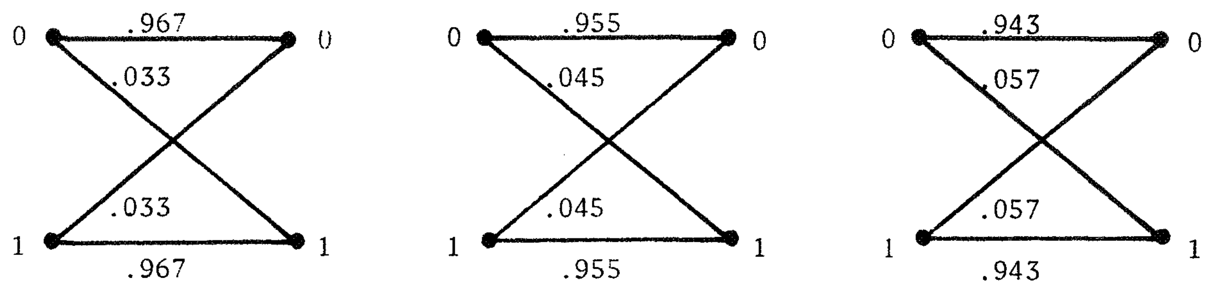
TABLE 2

Binary Symmetric Channel

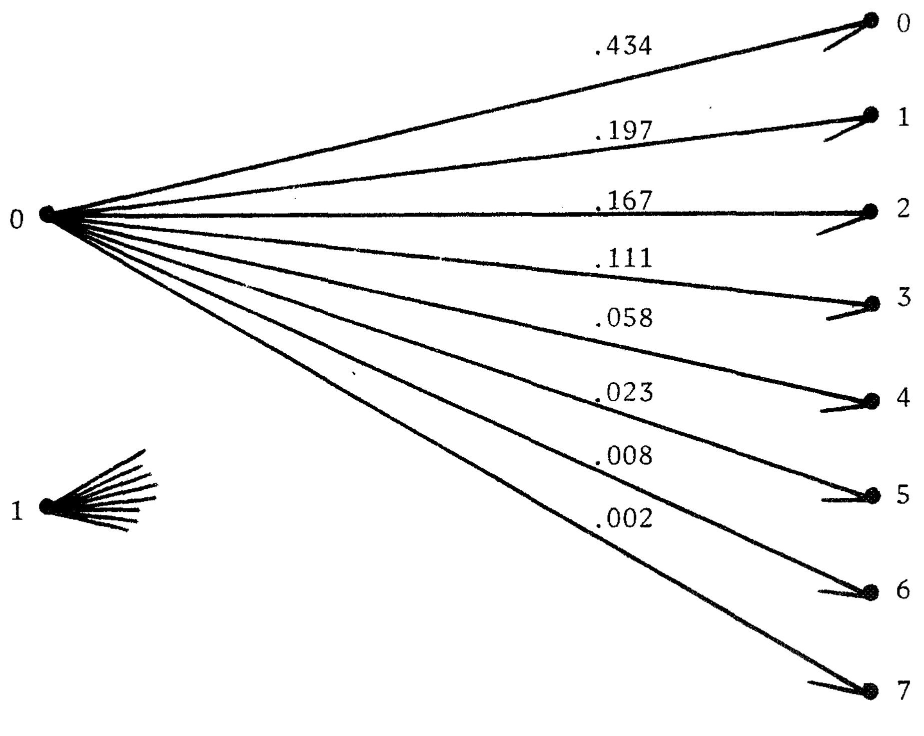
p	Code Symbol	Received Symbol 0	Received Symbol 1
0.033	0	2	-18
	1	-18	2
0.045	0	2	-16
	1	-16	2
0.057	0	4	-35
	1	-35	4

Gaussian Channel

Code Symbol	Received Symbol 0	1	2	3	4	5	6	7
0	4	4	2	0	-8	-20	-34	-58
1	-58	-34	-20	-8	0	2	4	4



(a)



(b)

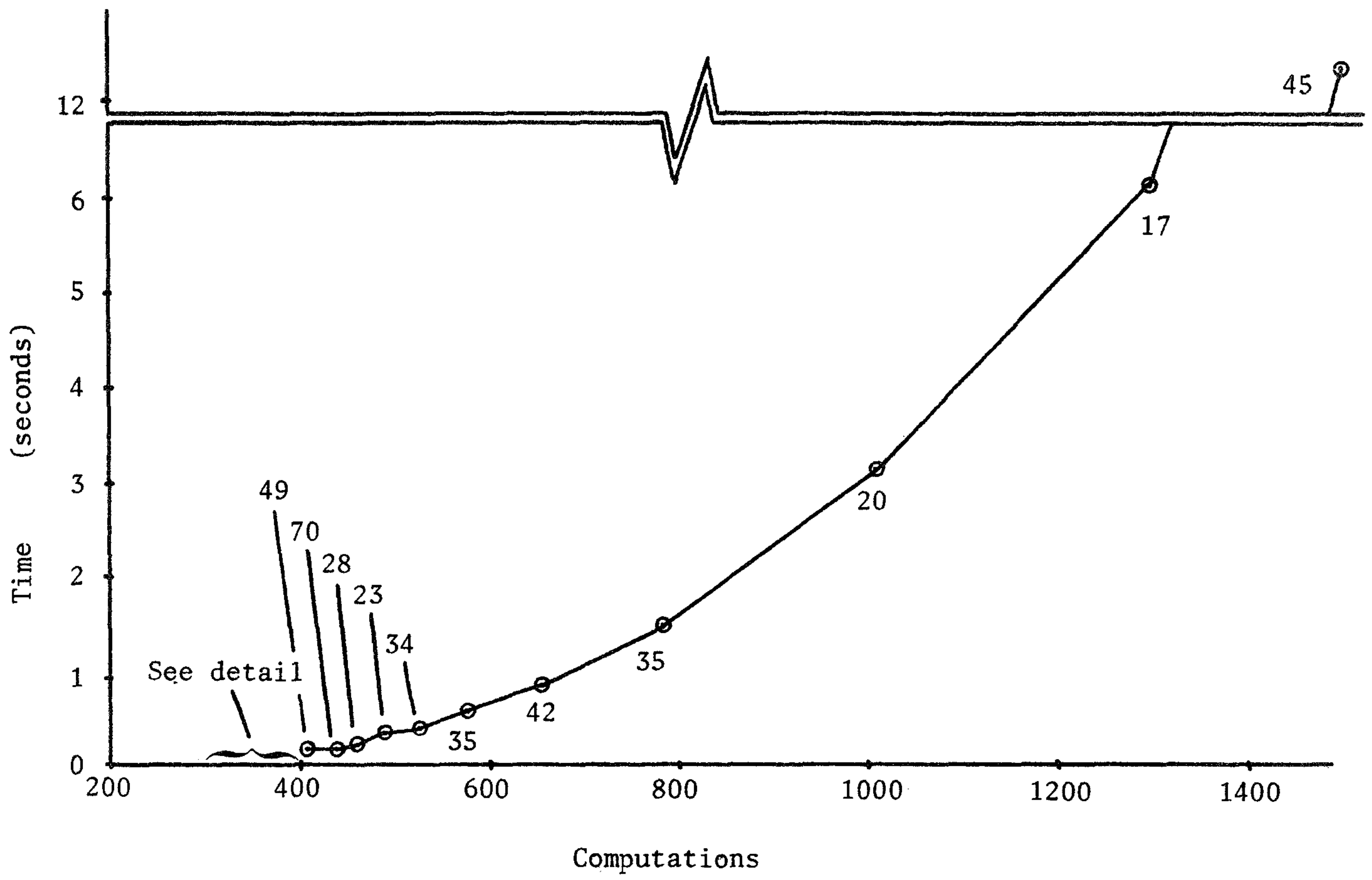
FIGURE 30. Channels Used in the Simulations: (a) Binary Symmetric Channels; (b) 3-bit Quantized Gaussian Channel.

It was assumed throughout that the all-zero sequence was transmitted. This assumption entails no loss of generality, but it offers the decoders an unfair advantage of the 0 branch is preferred in case of ties. Therefore, all decoders were biased to cooose the 1 branch in case of ties. Thus decoding is actually somewhat more difficult than we would expect for a random information sequence.

Computation Time for the Zigangirov-Jelinek Algorithm As pointed out in section II-B, the necessity for a stack search renders the Zigangirov-Jelinek computation time unacceptable, especially since the time increases faster than linearly with the number of computations. Figure 31 exhibits the behavior of computing time with number of computations. Each point on the graph represents average time and computation count for several frames whose computation counts fall in a specified range. The data points are labelled with the number of frames included in the average. These data were taken on the BSC with $p = .045$.

Optimum Bin Spacing and Threshold Increment Extensive work with the Fano decoder* has indicated that the best choice for Δ for rate ½ binary codes is a value equal to the magnitude of the branch metric for a branch having a single discrepancy. This is in agreement with an intuitive feeling that the decoder should be allowed to skip quickly over single errors and undertake extensive searches only in regions of severe noise. Single-error branches have values -16, -14, and -31 on the three BSCs, and the decoder program requires Δ to be a power of 2, so Δ of 16, 16, and 32, respectively, were used in the runs on the BSC.

* This work is beyond the scope of the present report, but some results are given by Costello[16].



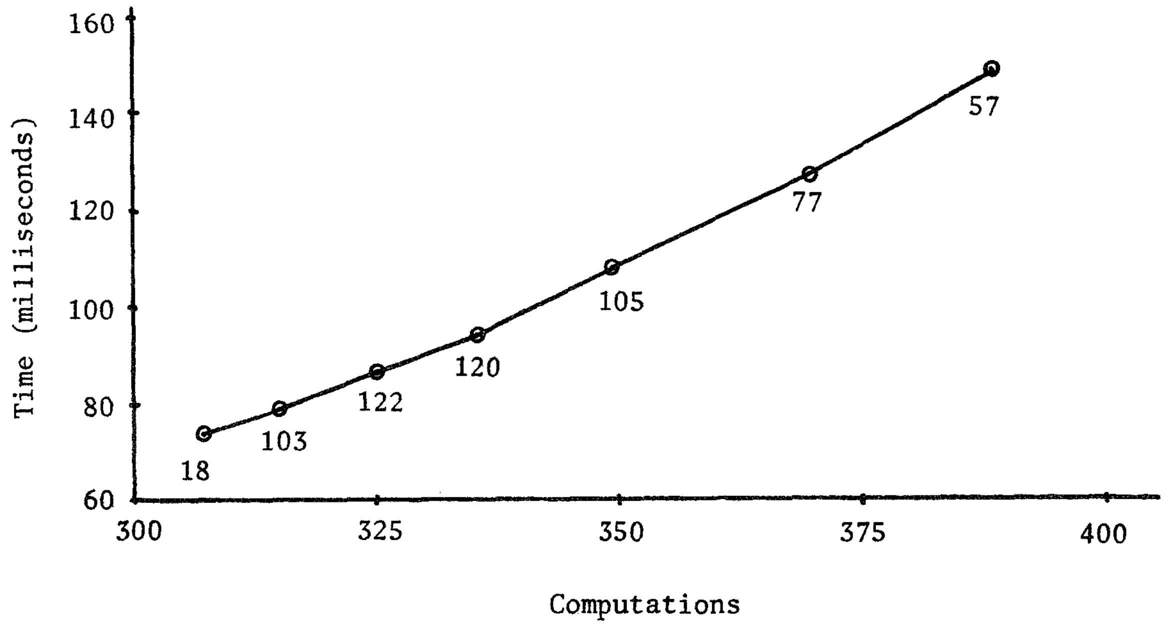


FIGURE 31. Variation of Computing Time with Number of Computations for the Zigangirov-Jelinek Algorithm.

For the Gaussian channel, "single errors" are not well-defined, but since digit metrics are roughly double those used on the first two BSCs, $\Delta = 32$ was used.

In an effort to determine empirically the optimum bin spacing for the Jelinek algorithm, runs were made with $H = 4,8,16$, and 32 on the BSC with $p = .033$ and $p = .045$. (The Jelinek program requires H to be a power of 2 and to be at least as large as the value of a branch which agrees with the received sequence in both digits. See Appendix B for details.) Table 3 gives average values of the number of computations in 100-frame samples at two noise levels. Figure 32 shows the distribution of computation for $H = 4,8$, and 16 at $p = .045$. In Figure 33 the distribution of computation is plotted for $H = 8,16$, and 32 on the Gaussian channel. On the basis of these results we conclude that $H = 4$ is the optimum choice for the first two BSCs and $H = 8$ for the third BSC and the Gaussian channel. These values were used in the remaining runs.

TABLE 3

p	H	Average Computation
0.033	4	353.9
	8	354.3
	16	360.2
	32	411.8
0.045	4	469.0
	8	471.1
	16	517.6
	32	607.3

Distribution of Computation for the Jelinek Algorithm As we saw previously, the computation C required to decode a digit is a random

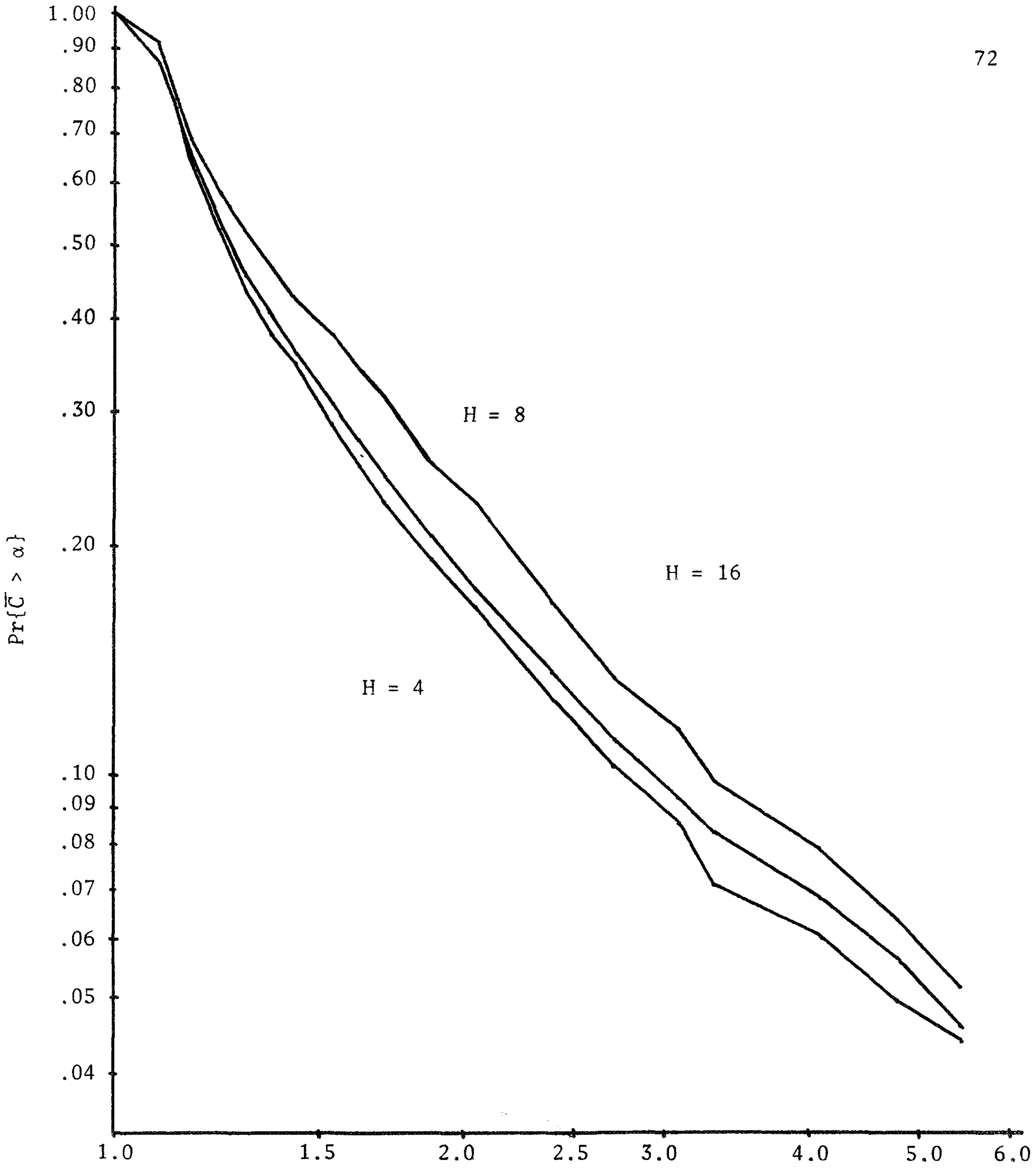


FIGURE 32. Jelinek Algorithm; Distribution of Computation for Various Bin Spacings, Binary Symmetric Channel, p=.045.

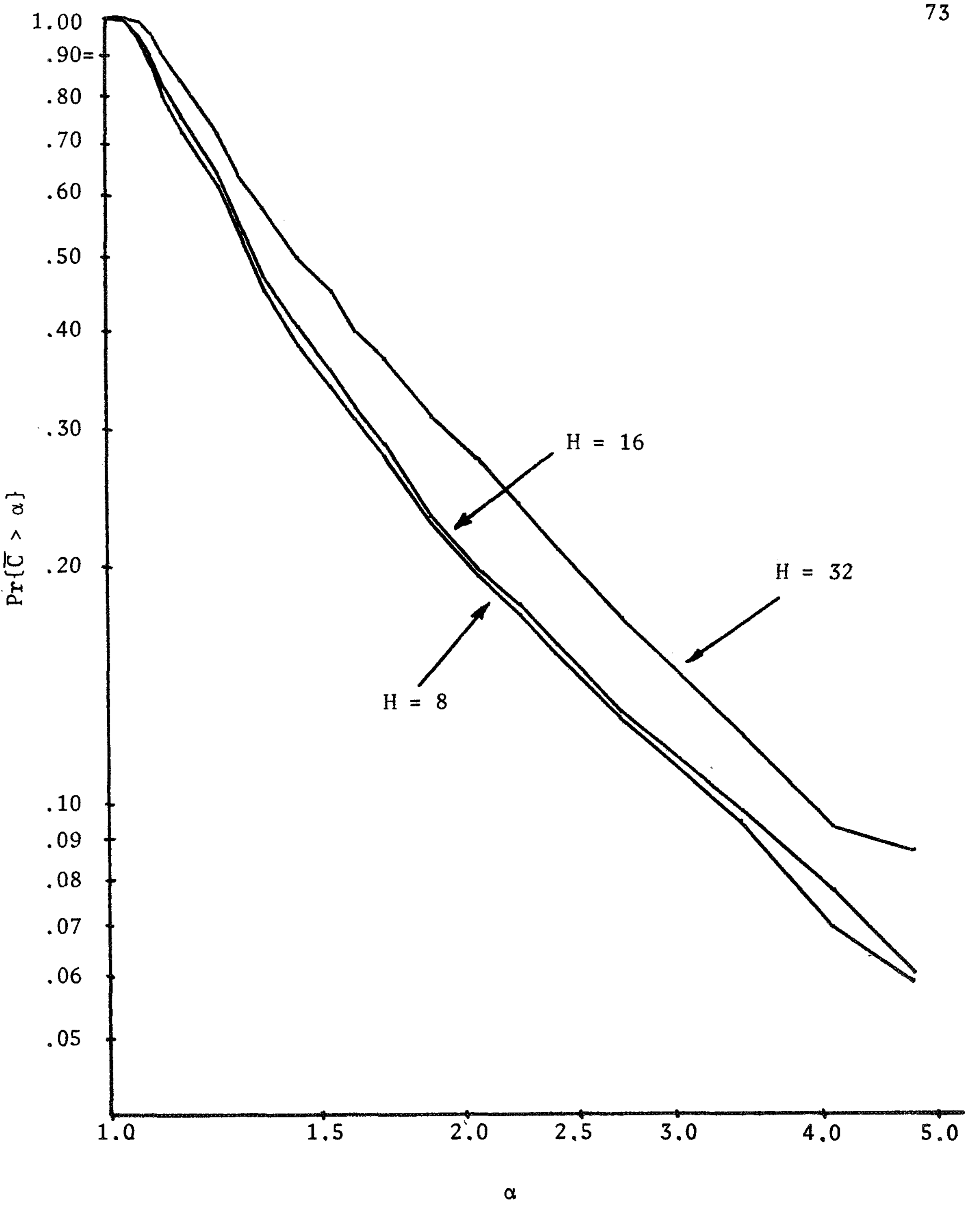


FIGURE 33. Jelinek Algorithm; Distribution of Computation for Various Bin Spacings, Gaussian Channel.

variable whose distribution (asymptotically) has the form:

$$\Pr\{C \geqq N\} = K\ N^{-\rho}$$

where ρ is the solution of $R = R_\rho = E_0(\rho)/\rho$. In Figures 34 (BSC) and 35 (Gaussian channel) are plotted the observed distributions of the average computation per digit, that is, the total computation done in decoding a frame divided by the frame length. The distribution of this random variable $\overline{C}$ differs from that of C, but as we noted in Chapter I, the distributions have the same form over a range of N. Since Figures 34 and 35 are plotted on log-log scale, the curves tend in this range to straight lines whose slopes are the negatives of the values of ρ given in Table 1. These asymptotes are displayed as dashed lines in Figures 34 and 35.

Comparison of Jelinek and Fano Computing Time We will define a computation for the Fano algorithm to be an entry of either of the LOOK FORWARD boxes of Figure 9, and a Jelinek computation to be a execution of setp (1) of the algorithm. Because of the need for repeated visits by the Fano decoder, it always requires more computation to decode a frame than does the Jelinek decoder, so on that basis alone, the Jelinek decoder is superior. However, Jelinek computations are inherently more complicated than Fano computations, since with each node examined, the decoder must store enough information to determine the path back to the origin and to resume searching forward from that node, if necessary. The Fano algorithm, on the other hand, since it only moves from a node to an adjacent node, can easily maintain this information as it proceeds through the tree. The question is, therefore, whether the additional complexity of the Jelinek com-

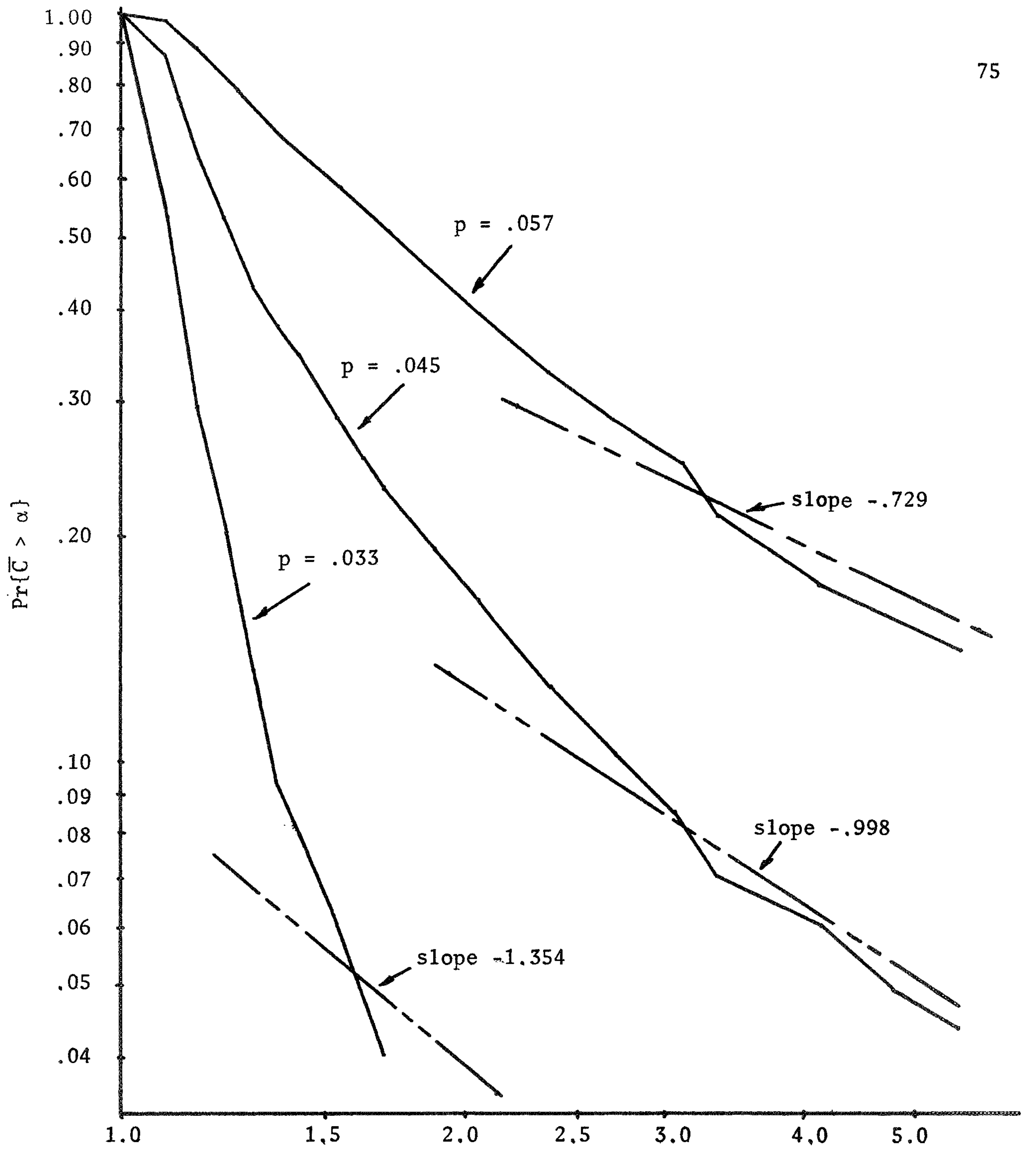


FIGURE 34. Jelinek Algorithm; Distribution of Computation on the Binary Symmetric Channel with Optimum Bin Spacing.

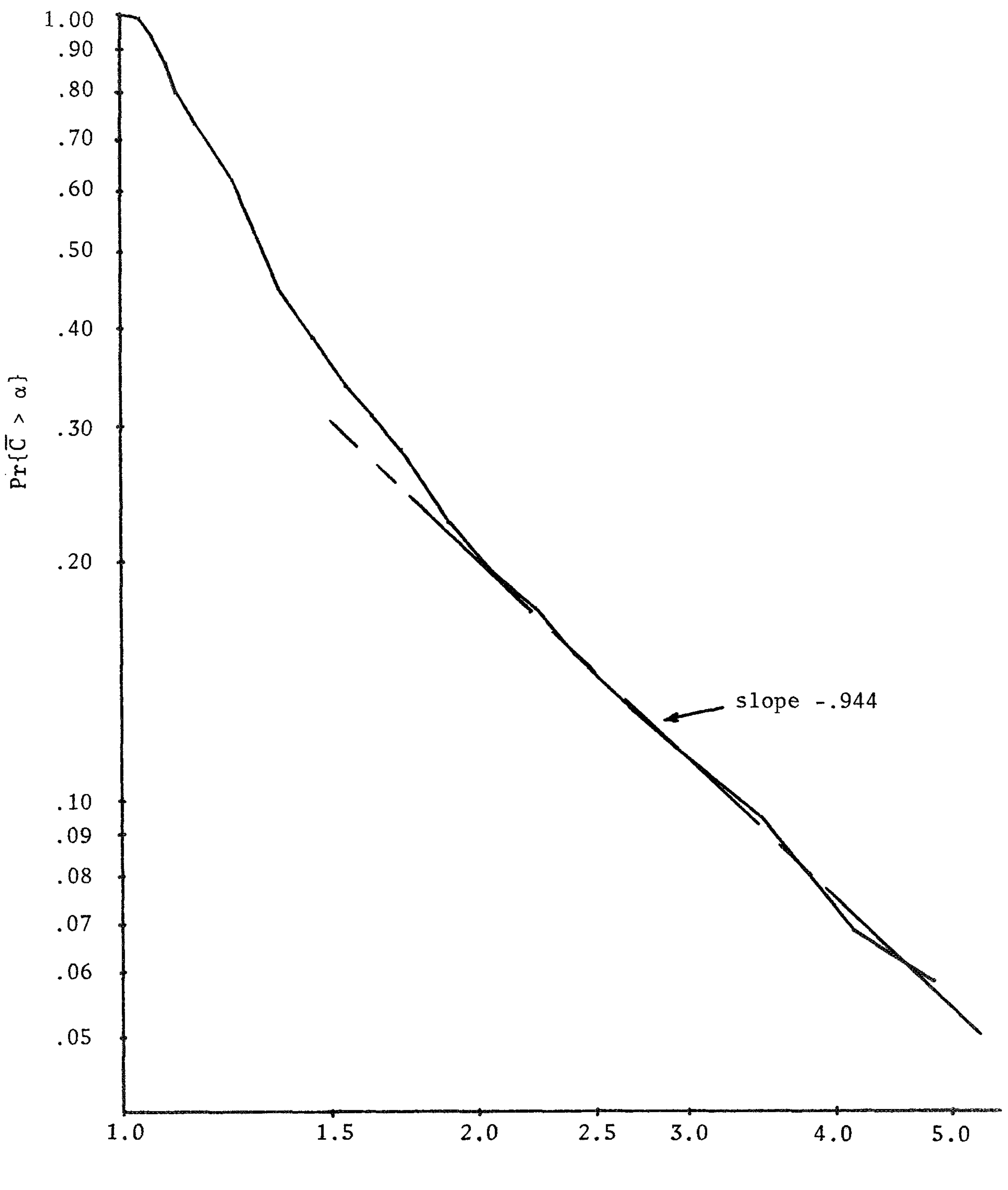


FIGURE 35. Jelinek Algorithm; Distribution of Computation on the Gaussian Channel with Optimum Bin Spacing.

putation is offset by the smaller number of computations required for decoding.

An added complication precludes a simple answer to this question. The number of "forward looks" the Fano decoder must perform grows faster than linearly with the number of nodes examined; that is, the ratio of Fano computations to Jelinek computations is not constant, but increases as the number of Jelinek computations is increased, because of the repeated searches by the Fano decoder. It is therefore possible that, for relatively quiet frames the Fano decoder is superior, while for noisier frames the Jelinek decoder is superior. This is, in fact, the behavior which was observed in the simulations.

We choose as a measure of decoding effort the time to decode a frame. Since the time required is roughly proportional to the number of computations, the distribution of computing time has the same shape as the distribution of computation. The behavior observed in the simulation is plotted in Figures 36 (BSC) and 37 (Gaussian channel).

The reader is cautioned against attributing too much significance to the position of the crossover point in Figures 36 and 37, since it depends strongly on the relative complexity of the two kinds of computations, and may altered by differences in available hardware or programming technique. What can safely be concluded is that the Jelinek decoder is at least competitive with the Fano decoder, and is faster except on fairly quiet frames. The noisier the channel, the less likely are quiet frames, and hence the more likely is the Jelinek decoder to be the faster of the two.

Average Computing Time as a Function of Erasure Probability We have seen that the random variable T, the time to decode a frame,

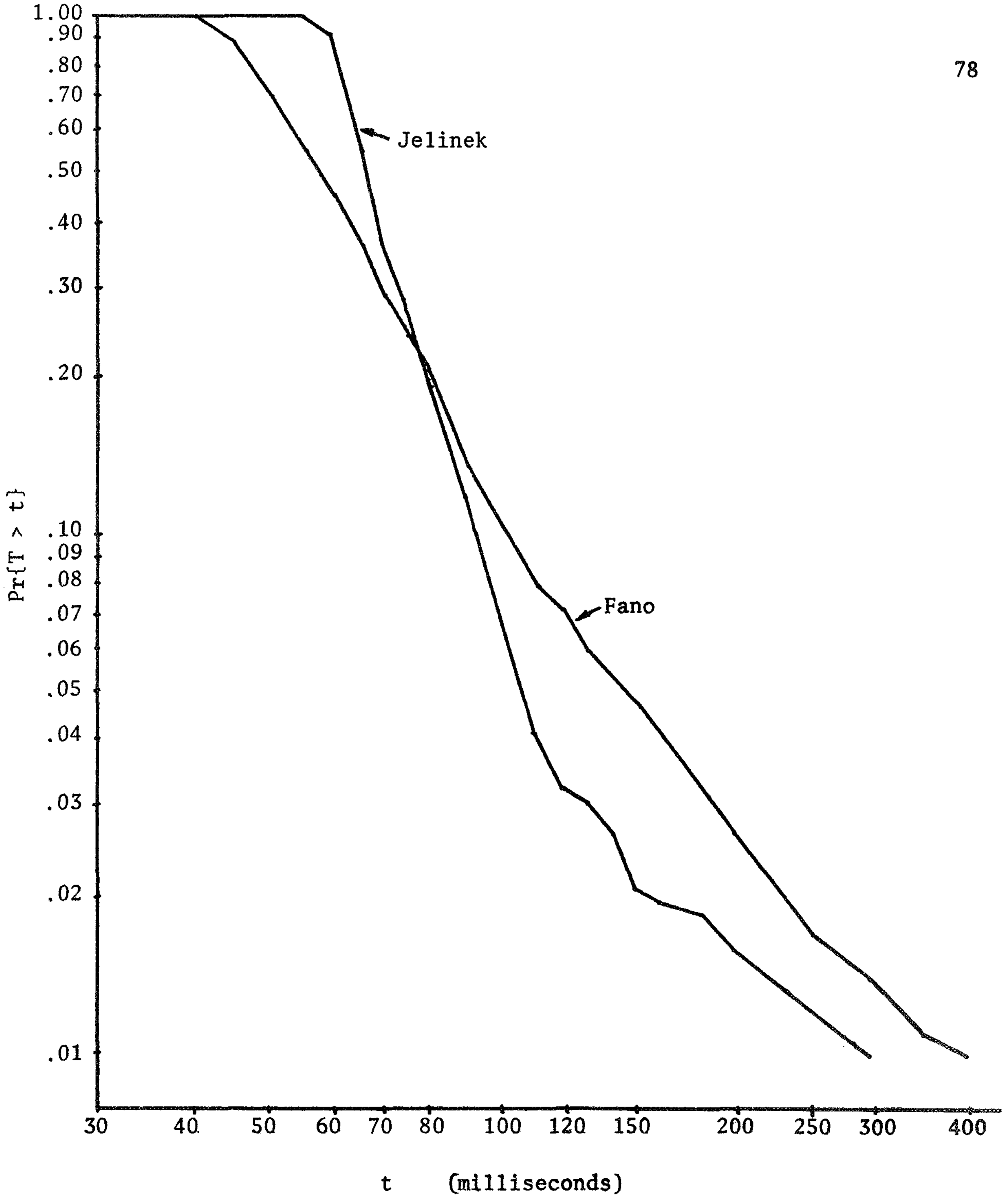


FIGURE 36A. Distribution of Computing Time, Binary Symmetric Channel, p = .033.

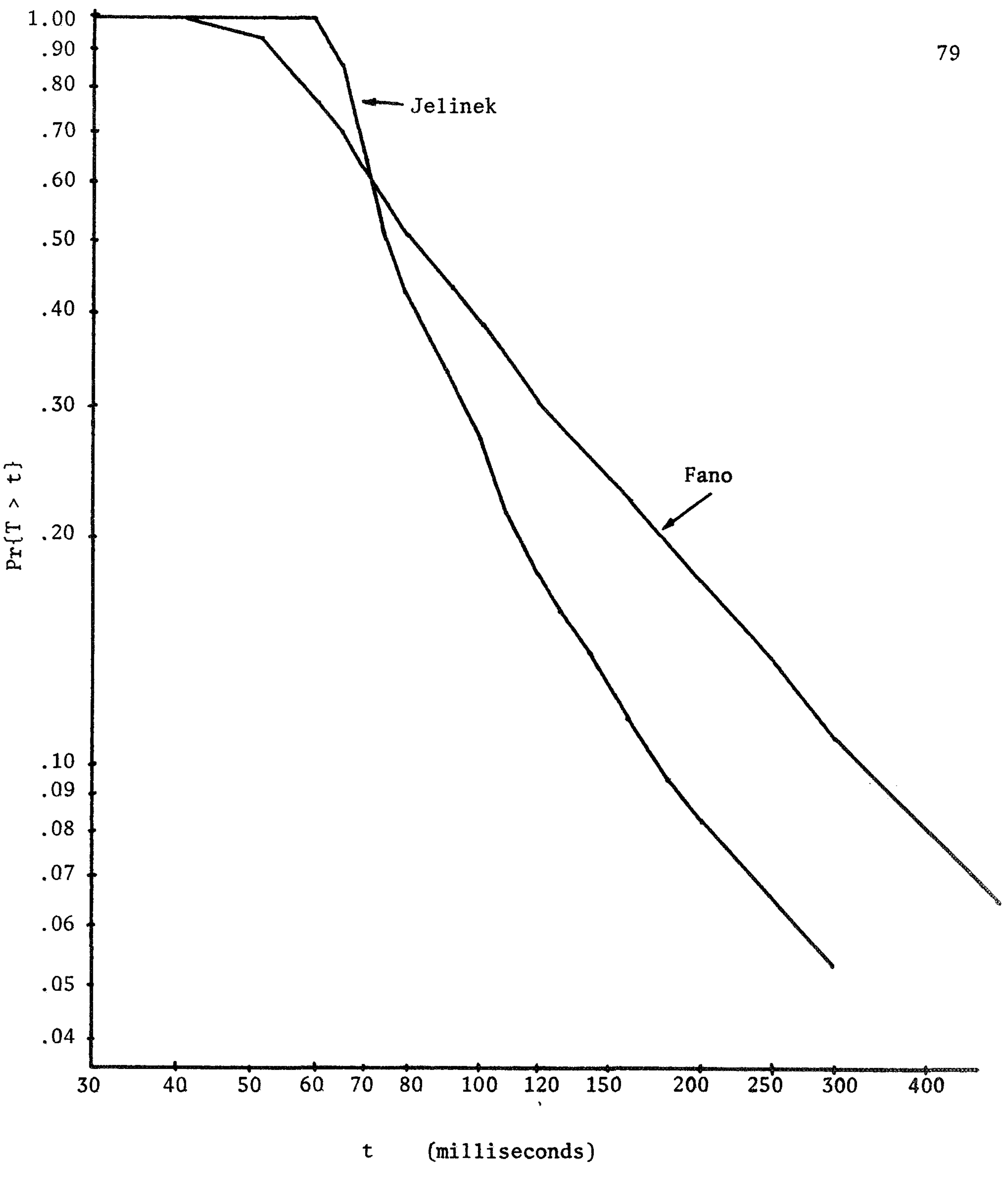


FIGURE 36B. Distribution of Computing Time, Binary Symmetric Channel, p = .045.

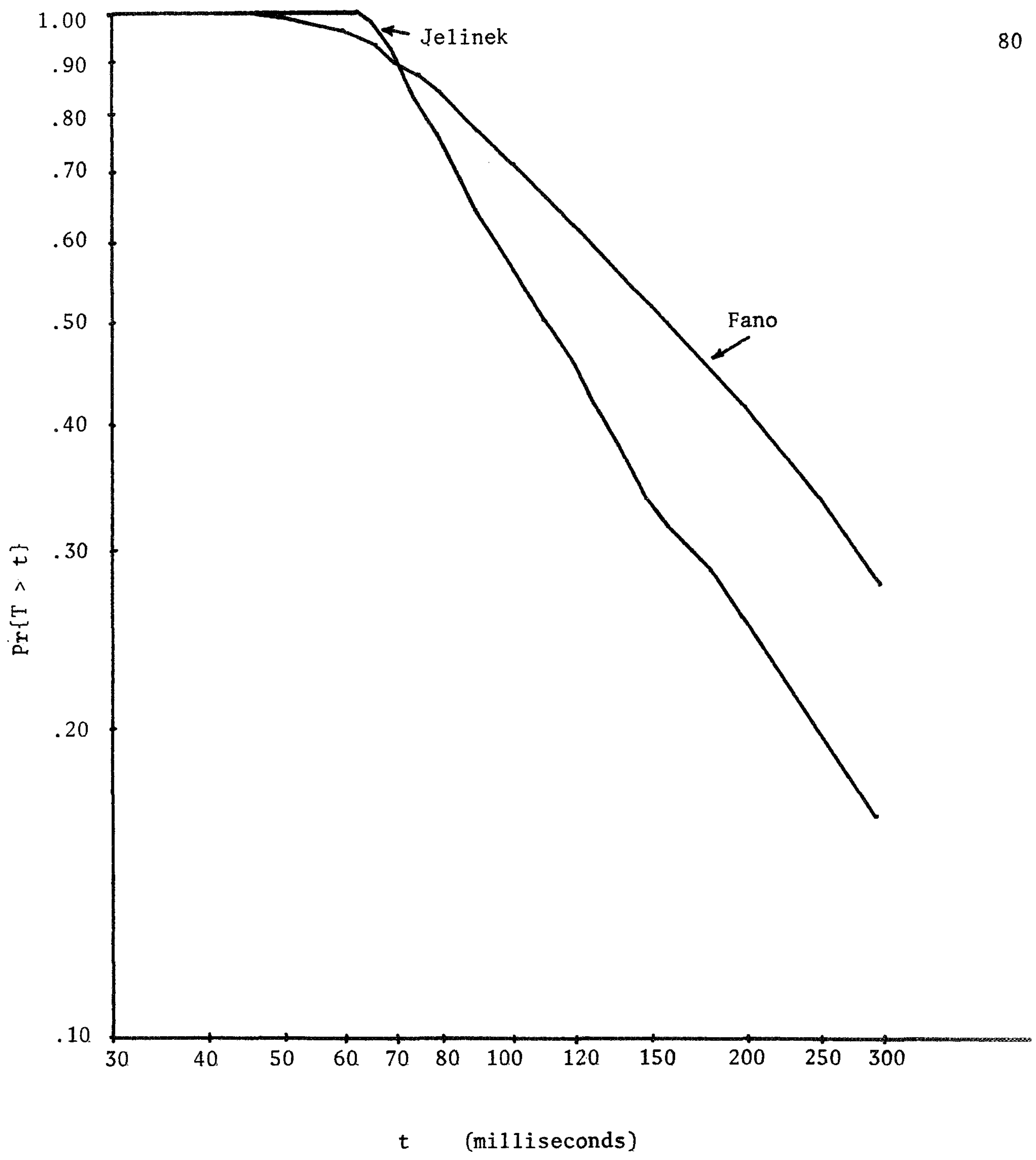


FIGURE 36C. Distribution of Computing Time, Binary Symmetric Channel, p = .057.

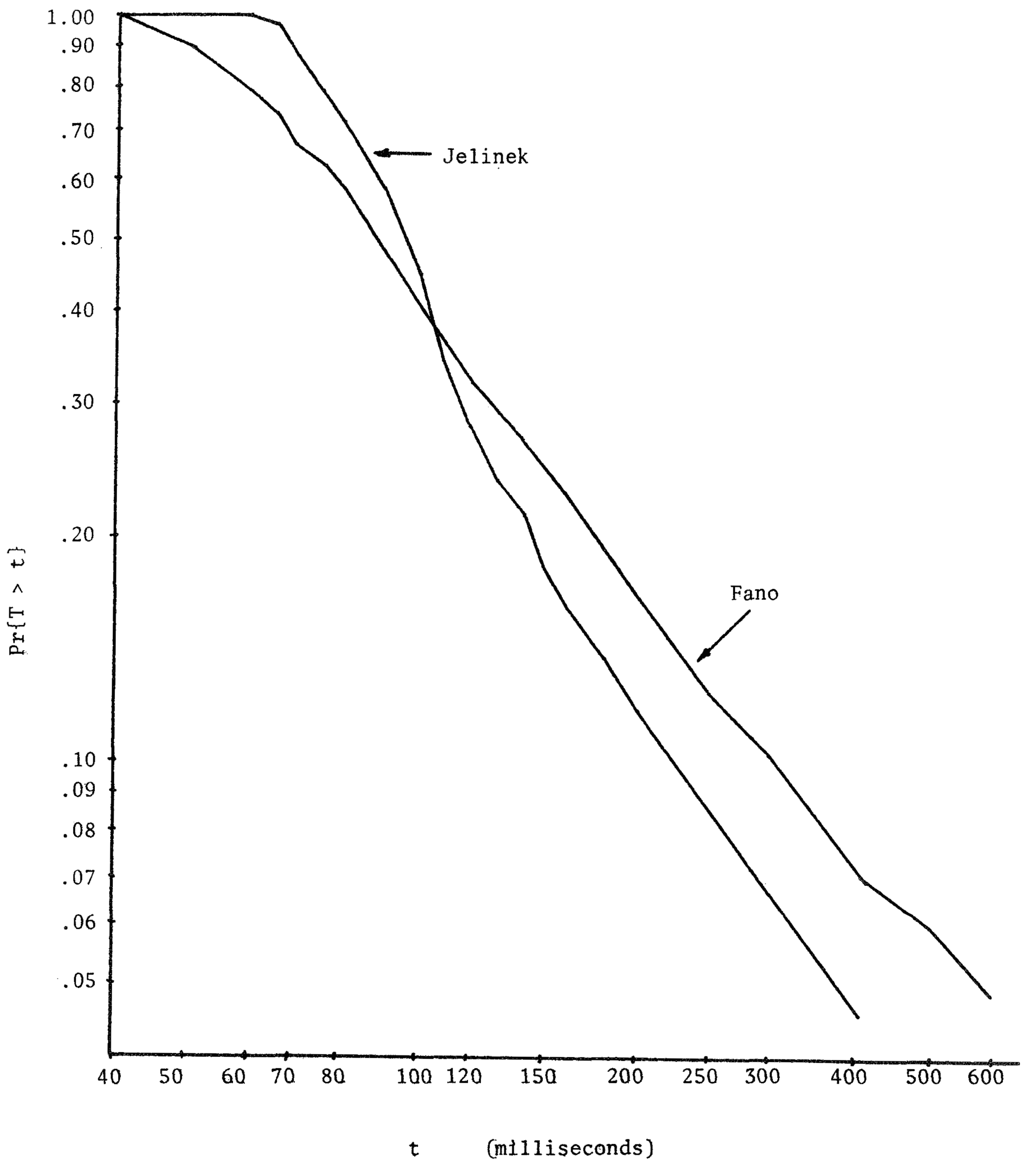


FIGURE 37. Distribution of Computing Time, Gaussian Channel.

exhibits a Pareto distribution for sufficiently large values of the distribution parameter. In any real, on-line communication system, only a finite amount of time can be allowed for decoding each frame, and any frame not completely decoded in the allotted time must be considered an erasure. If t_{max} is the maximum allowable time per frame, then $\Pr\{T > t_{max}\}$ is the erasure probability. Conversely, if a tolerable erasure probability Q_E is prescribed, then the maximum time t_{max} which should be designed into the system can be found by solving $\Pr\{T > t_{max}\} = Q_E$.

With the maximum time fixed, we have a new random variable T^* derived from T as follows: if $T = t$, then $T^* = \min\{t, t_{max}\}$. The average of T^* provides a convenient measure of the performance of the system, and we now investigate the behavior of $\overline{T^*}$ as a function of Q_E.

Suppose we have

$$\Pr\{T > t\} = \begin{cases} f(t), & t \leqq t_p \\ K\, t^{-\rho}, & t \geqq t_p \end{cases}$$

where all that is known about $f(\cdot)$ is that it is monotone non-increasing, that $f(t) = 1$ for all $t \leqq t_0$, and that $f(t_p) = K\, t_p^{-\rho}$. (See Figure 38.)

Assume that Q_E is chosen small enough that the solution of $\Pr\{T > t\} = Q_E$, t_{max}, is greater than t_p. Then $Q_E = K(t_{max})^{-\rho}$, or

$$t_{max} = (Q_E/K)^{-1/\rho} \tag{1}$$

For a given Q_E the distribution of $\overline{T^*}$ is given by:

$$\Pr\{T^* > t\} = \begin{cases} \Pr\{T > t\}, & t < t_{max} \\ 0, & t \geqq t_{max} \end{cases}$$

The density function of T is $p_T(t) = -\frac{d}{dt}\Pr\{T > t\}$, or

$$p_T(t) = \begin{cases} -\frac{df}{dt}(t), & t < t_p \\ K\rho t^{-\rho-1}, & t \geqq t_p \end{cases}$$

Let $\hat{p}_T(t)$ be given by:

$$\hat{p}_T(t) = \begin{cases} p_T(t), & t < t_{max} \\ 0, & t \geq t_{max} \end{cases}$$

Then the density function of T* is:

$$p_{T*}(t) = \hat{p}_T(t) + Q_E\delta(t-t_{max})$$

Therefore the average value of T* is given by:

$$\overline{T^*} = \int_{t_0}^{\infty} t\ p_{T*}(t)\ dt$$

$$= -\int_{t_0}^{t_p} t\ \frac{df}{dt}(t)\ dt \quad + \int_{t_p}^{t_{max}} t\ K\rho t^{-\rho-1}\ dt \quad + \quad Q_E t_{max}$$

Now

$$K\rho \int t^{-\rho}\ dt \quad = \begin{cases} \frac{K\rho}{1-\rho}\ t^{1-\rho}; & \rho \neq 1 \\ K\ \ln t, & \rho = 1 \end{cases}$$

Hence for $\rho \neq 1$,

$$\overline{T^*} = C_1 + \frac{K\rho}{1-\rho} t_{max}^{1-\rho} + Q_E t_{max}$$

Using (1), we obtain

$$\overline{T^*} = C_1 + \frac{K^{1/\rho}}{1-\rho}\ Q_E^{(1-1/\rho)}$$

For $\rho = 1$, $t_{max} = K/Q_E$, so $\overline{T^*} = C_2 - K \ln Q_E$. Therefore,

$$\overline{T^*} = \begin{cases} C_2 - K\ \ln\ Q_E, & \rho = 1 \\ C_1 + \frac{K^{1/\rho}}{1-\rho}\ e^{(1-1/\rho)\ln\ Q_E}, & \rho \neq 1 \end{cases}$$

From this form we can see the behavior of $\overline{T^*}$ as ln Q_E varies; plots for three ranges of ρ are displayed in Figure 39. Note that for $\rho > 1$, the average of T exists, and so $\overline{T^*}$ approaches a finite limit, namely $\overline{T}$. But for $\rho \leqq 1$, $\overline{T}$ fails to exist and the average of T* grows without bound as Q_E is made to decrease.

Experimental observations of the behavior of $\overline{T^*}$ as Q_E is varied are presented in Figure 40. Since the non-asymptotic parts of the distribution of T differ for the two algorithms, the effective value of K is different. Hence in Figure 42B, while both averages increase linearly as log Q_E decreases, the slopes and intercepts are different, showing that the initial superiority of the Fano algorithm quickly disappears, and the Jelinek algorithm becomes the better, its advantage growing rapidly as Q_E is decreased. Thus in the vicinity of R_{comp}, the Jelinek algorithm offers considerable practical advantage. Note, however, that for the low noise case, $p = .033$ ($R = .9R_{comp}$), the advantage goes to the Fano algorithm. In general, the noisier the channel, the more favorably does the Jelinek algorithm perform relative to the Fano algorithm.

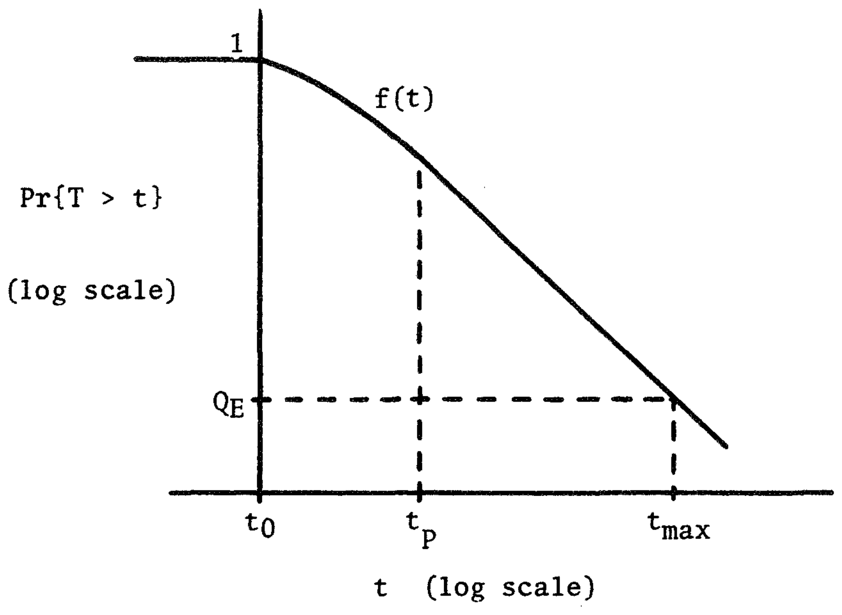


FIGURE 38. An Asymptotic Pareto Distribution.

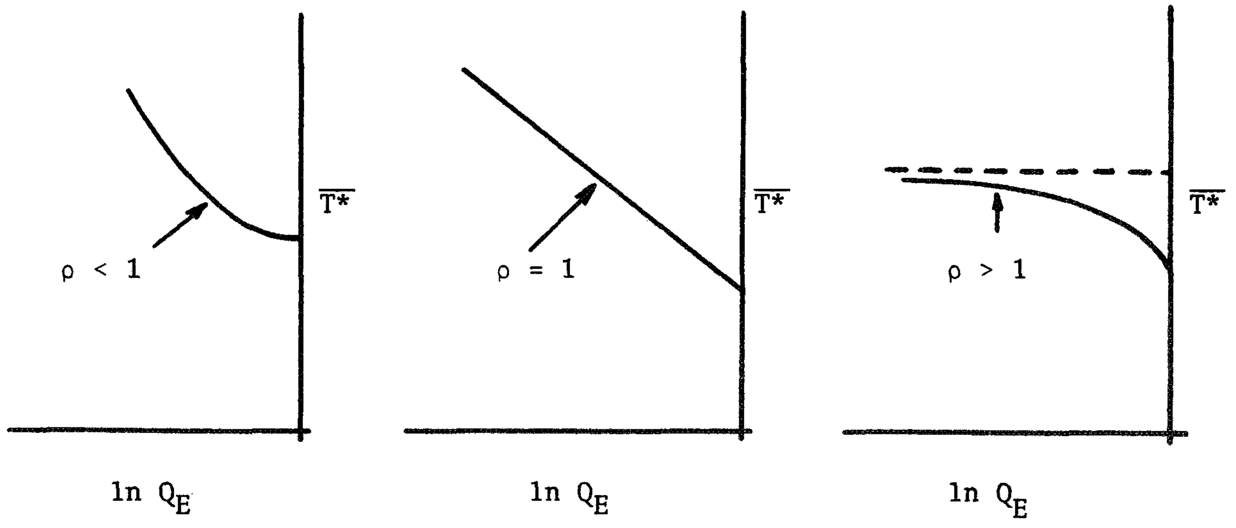


FIGURE 39. Behavior of $\overline{T^*}$ as a Function of Erasure Probability.

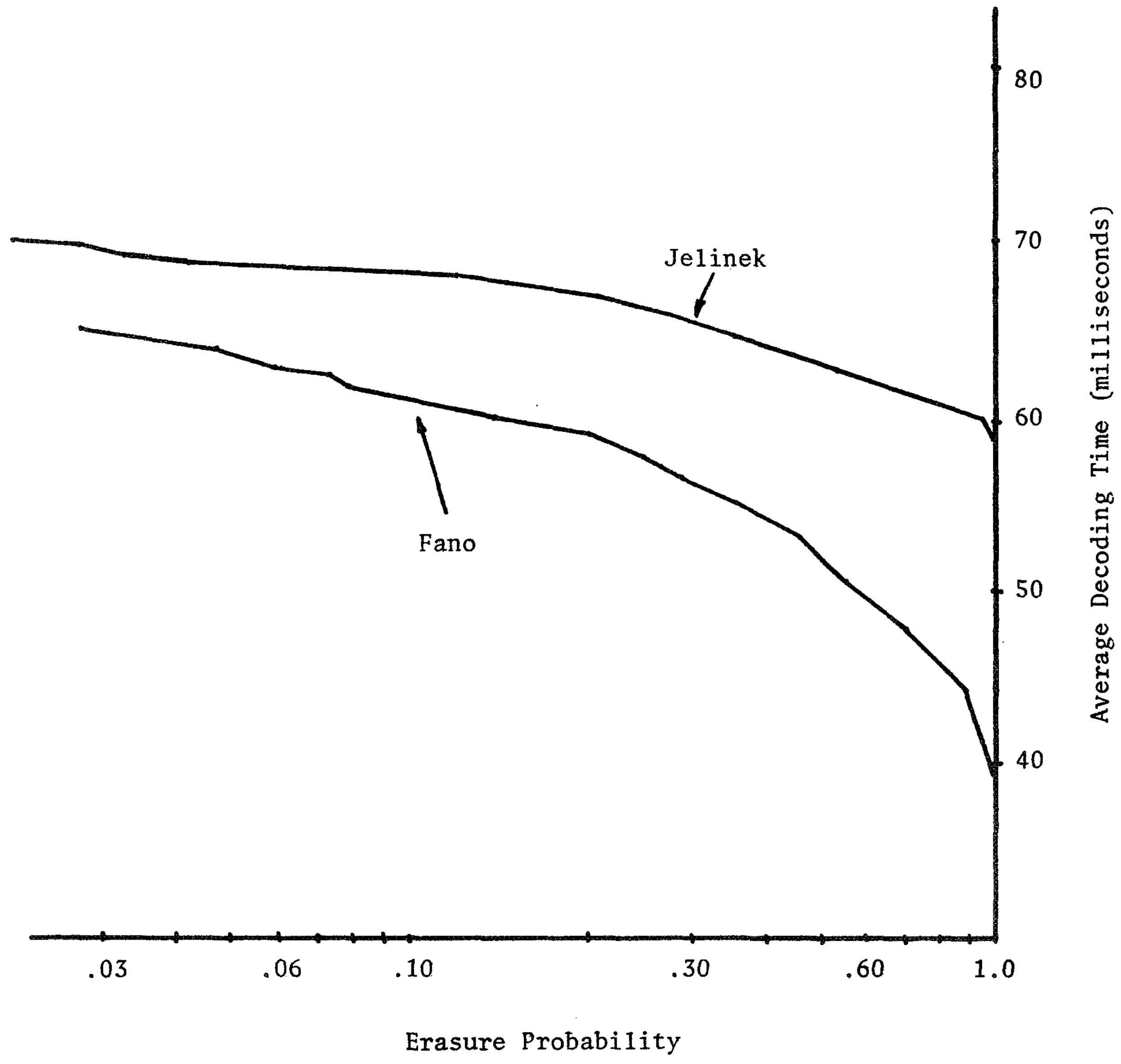


FIGURE 40A. Performance of Time-limited Decoders, Binary Symmetric Channel, p = .033.

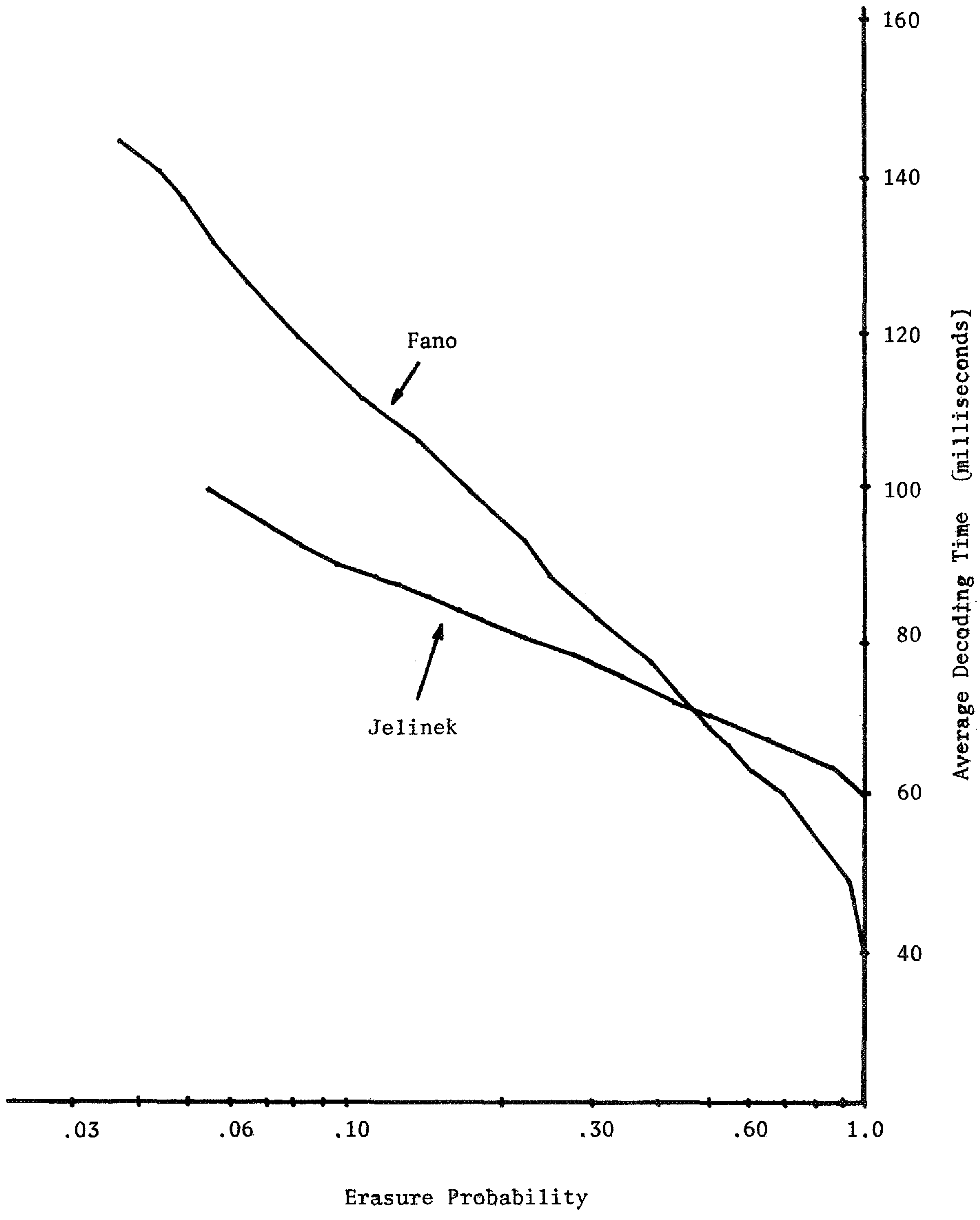


FIGURE 40B. Performance of Time-limited Decoders, Binary Symmetric Channel, p = .045.

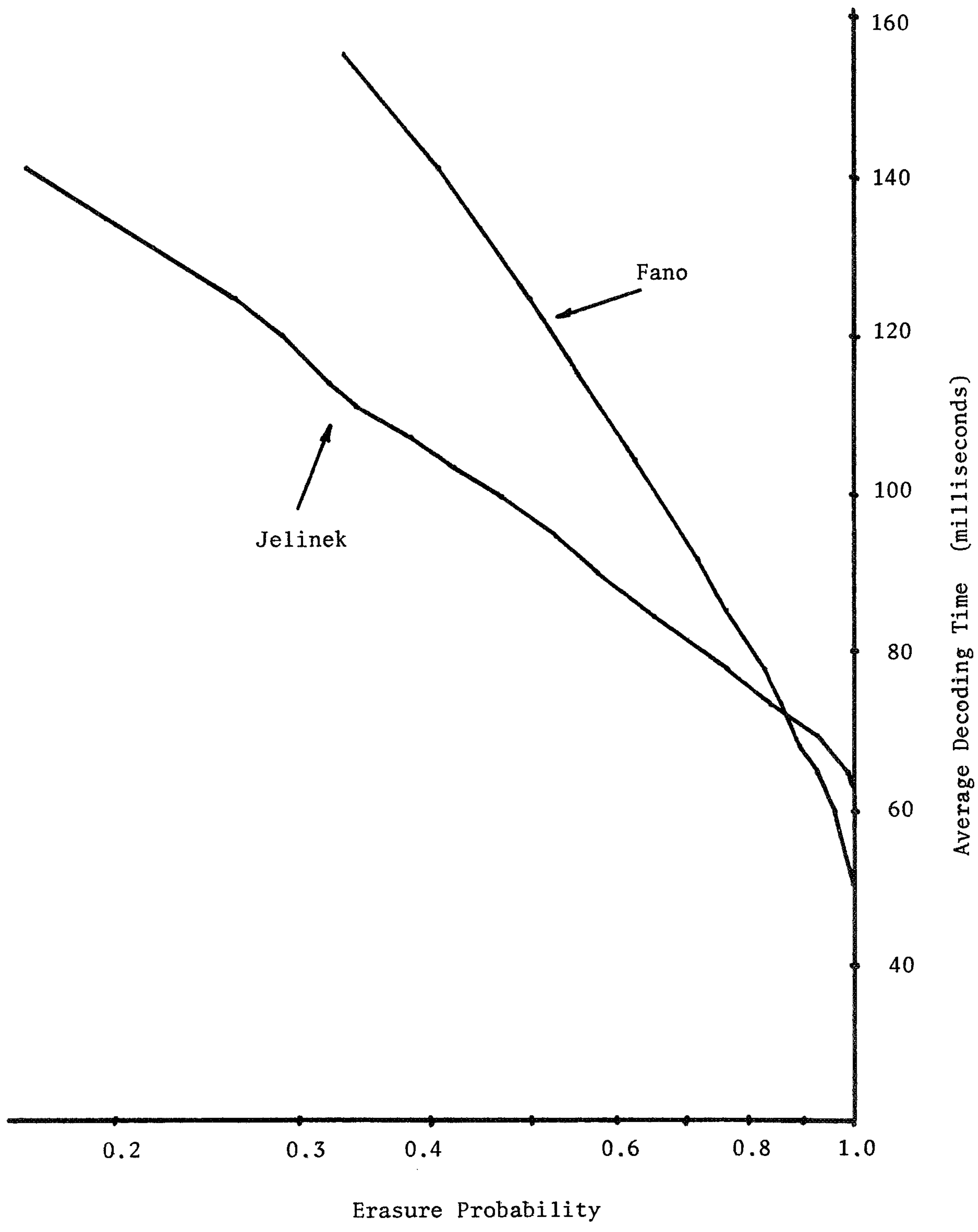


FIGURE 40C. Performance of Time-limited Decoders, Binary Symmetric Channel, p = .057.

CHAPTER IV

APPLICATIONS OF SEQUENTIAL SEARCH ALGORITHMS TO GRAPH-SEARCHING PROBLEMS

Until now we have focused on sequential decoding as a means of finding a near-optimum path through a value tree. In this chapter we consider the problem of finding good paths through more general graphs in which each branch is assigned a value, and we consider how the sequential decoding algorithms we have discussed can be applied to this task. We shall hereafter refer to these procedures as sequential *search* algorithms rather than sequential *decoding* algorithms.

The graphs we will be dealing with in this chapter will exhibit the following six properties:

G1: The graph consists of a finite number of nodes and branches.

G2: The graph is directed; that is, every branch b between two nodes s and s' is assigned a direction, say *from* s *to* s'. In this case, b is said to *emanate* from s and *terminate* at s'; s' is called the *successor* of s along b, and s is called the *predecessor* of s' along b.

G3: There is exactly one node s_0, called the origin node, which has no predecessor.

G4: There is exactly one node s_F, called the final node, which has no successor.

G5: The graph is connected in the sense that for any node s in the graph, $s \neq s_0$, there is at least one path from s_0 to s, and for any node s, $s \neq s_F$, there is at least one path from s to s_F. In particular, there is at least one path from s_0 to s_F, provided that $s_0 \neq s_F$. (Here the term "path from s to s' " means a sequence of branches $b_1, b_2, \cdots, b_N$ such that b_1 emanates from s, b_N terminates at s', and b_i emanates from the node at which b_{i-1} terminates, $i=2, 3, \cdots, N$.

G6: There are no closed paths.*

Conditions G3 and G4 may seem unreasonably restrictive, but for graphs having several starting and ending nodes, we could think of adjoining two additional nodes s_0 and s_F and placing a branch of zero value from s_0 to each starting node, and from each final node to s_F. The resulting graph satisfies G3 and G4, and is equivalent to the original graph for present purposes.

Given a graph satisfying these six conditions, suppose that each branch is assigned a real number called the branch value. The value associated with a path is the sum of the values of the branches comprising the path. Our problem will be to select from among the set of all paths from s_0 to s_F the path of greatest value.

4. A. OPTIMUM GRAPH SEARCHING

The problem of finding an optimum path through a graph has been studied by many investigators and several algorithms have been proposed. See, for example, Dantzig[17], Busacker and Saaty[18], Pollack and Wiebenson[19]. (In the references cited, the term "path length" is used to mean what we have called "path value," while we have used the

* If such a closed path has negative gain, it is part of no optimum path, and if it has positive gain, there is no optimum path (of finite length), so no cases of interest have been excluded by G6.

term "length" to mean the number of branches in the path.)

Under certain conditions, the optimum path can be found by the procedure called "dynamic programming" by Bellman[20]. The trellises discussed in Chapter I admit a dynamic programming search, and as Omura[21] has noted, the Viterbi[22] algorithm for decoding convolutional codes is precisely the dynamic programming search of an appropriate metric trellis.

The algorithms referenced above provide a general solution to the problem of finding and evaluating the optimum path through a graph satisfying G1-G6. The difficulty with this approach is that for fairly large and complex graphs, the amount of computation required becomed intolerable. In the next section, we suggest how procedures which are essentially modifications of the sequential decoding procedures we have seen can be used to search graphs of the kind we are considering and select a path with less effort, at the price of some sacrifice of optimality.

4. B. SUBOPTIMUM GRAPH SEARCHING--SEQUENTIAL SEARCHING

A tree of finite length can be made to satisfy conditions G1-G6 by adjoining an additional final node s_F and several zero-value branches, as discussed on page 90. We will then say that the tree has been terminated. It is clear that an algorithm which searches any graph satisfying G1-G6 can also search a terminated tree. The converse, however, is not evident. Our first task, therefore, will be to see how the sequential search algorithms, which we have heretofore considered as tree-search algorithms, can be used on any graph in the class of graphs satisfying conditions G1-G6.

4. B. 1. The Path Tree of a Graph

Given a graph satisfying G1-G6, the set of all paths from s_0 to s_F has a natural tree structure. This tree, which we shall call the path tree of the graph, can be described as follows: For two paths P_1 and P_2 from s_0 to s_F of respective lengths L_1 and L_2 which have in common the first k_1 branches, then diverge, to remerge again at some node and share the last k_2 branches to s_F, there are two paths in the path tree which coincide in the first k_1 branches, then diverge and remain separate to the end of the tree. In the path tree there are two nodes representing the path node at which P_1 and P_2 remerge, and the portions of the tree from these two nodes to the end of the tree are identical. Thus in the path tree there are as many nodes which represent a given graph node as there are paths in the graph from s_0 to that node. As an example, the graph of Figure 41a has the path tree of Figure 41b. As another example, the code tree of Figure 3 is the path tree of the trellis of Figure 4.

The path trees we obtain in this way do not in general have the property that every non-final node has the same number of successors, nor do all paths through the tree have the same length. However, the sequential search algorithms are still applicable, provided that the searcher can realize when it is at the end of a path. Therefore we can define the action of a sequential search algorithm on a graph to be the action of the algorithm when applied to the graph's path tree.

This at least settles the problem of what it means to search a graph using the Fano or Zigangirov-Jelinek algorithm, but it would be unsatisfactory if we had to actually construct the path tree before performing the search. This is not necessary if certain facts are noted about the algorithms.

For the Fano algorithm, suppose the searcher is allowed to keep track of the path it has followed to its present position. (This is easy to do, and is always done in practice.) Then we can interpret the instruction "LOOK BACK" to mean "Look back along the last branch of the present path," thus resolving the ambiguity of that instruction. In addition, since node values are computed by adding branch values to an accumulated sum V, it is clear that these node values are actually path values, and that if the searcher encounters a node twice from along different paths, different values (in general) will result. Finally, the "FIRST VISIT ?" test should be interpreted as a test for first visit along the present path; in view of the above remarks on node values, it can be seen that Gallager's test (Figure 9) is such a test. We can then conclude that the Fano algorithm will search through the graph exactly as if it were searching the associated path tree. Consequently the path the algorithm will select is the path it would select when searching the path tree, namely, the one which satisfies F1 and F2. (When applying F1 and F2 to graphs, we are obliged to alter them slightly, since it is no longer possible to specify a path by enumerating the <u>nodes</u> along the path--we must list the <u>branches</u> comprising the path.)

Now, turning to the Zigangirov-Jelinek algorithm, we see that the first difficulty does not arise since this algorithm only moves forward. However, since a node may be the successor of more than one node, it is possible that one of the successors of the top node on the stack is already on the stack because of an earlier extension. In this case, the two appearances of the node represent different paths (that is, different nodes in the path tree) and in general different values. If the searcher is allowed to have such multiple occurrences of graph nodes in the stack, then it will search the graph just as if it were searching the path tree, and hence it will select a path satisfying ZJ1 and ZJ2 (when these rules are rephrased as noted before).

Making the analogous argument for the Jelinek and unquantized Fano algorithms, we can see that applying them to graphs yields paths satisfying the corresponding conditions.

Incidentally, these observations show that it is immaterial whether we consider sequential decoding of convolutional codes as a tree search operation or a trellis search operation.

4. B. 2. Using Remergers to Improve Performance

The previous discussion has shown that our sequential search algorithms can be applied to graphs with meaningful results, since the algorithms will search the graphs as if they were searching the associated path tree. If the searcher encounters the same node twice from different paths, it does not realize that the node has been visited before. The question naturally arises whether, if the searcher were able to recognize revisits along new paths, it could use the information to be somewhat more selective in searching. We would like to avoid searching parts of the path tree if it is known in advance that the search will be futile.

For simplicity, we consider only the unquantized algorithms. As an immediate consequence of MF1 and MF2 we may state:

LEMMA: If b_1^*, b_2^*, $\cdots$, b_L^* is the path from s_0 to s_F selected by the unquantized Fano searcher, and s_k is the node at which b_k^*

terminates, then b^*_{k+1}, b^*_{k+2}, $\cdots$, b^*_L is the path the searcher would select if it were started at node s_k and constrained not to move backward from node s_k.

If we assume that ties are resolved in a consistent manner, then an analogous statement can be made about the Zigangirov-Jelinek searcher.

Now let s be a node in the graph and let $\bar{b}_1$, $\bar{b}_2$, $\cdots$, $\bar{b}_N$ be the branches along the path the searcher would select if started as s and prevented from moving back from s. Let b_1, b_2, $\cdots$,b_n and b'_1, b'_2,$\cdots$, b'_m be two different paths from s_0 to s such that $b_j \neq b'_j$, $b_n \neq b'_m$, and $b_i = b'_i$ for $i < j$. (See Figure 42.)

Let $V(s) = \sum_{i=1}^{n} v(b_i)$ and $V'(s) = \sum_{i=1}^{m} v(b'_i)$, where the quantities $v(b_i)$ and $v(b'_i)$ are the branch values; thus V(s) and V'(s) are the values at node s along the unprimed and primed paths, respectively.

THEOREM 9: If the unquantized Fano searcher moves to s along the unprimed path and $V(s) > V'(s)$, then the primed path b'_1, b'_2, $\cdots$,b'_m is not the initial segemnt of the path selected.

Proof: If the searcher never moves to s along the primed path, the claim is obvious. If b_1, b_2, $\cdots$, b_{j-1} is not the initial segment of the selected path, the claim is again obvious. Hence assume that node s_1 in Figure 42 is on the final path, that b_1, b_2, $\cdots$, b_{j-1} is the first segment of that path, and that the searcher eventually reaches s along both the primed and unprimed paths. Then by Theorem 8, if V^*_{min} denotes the minimum value seen along the final path from s_1 to

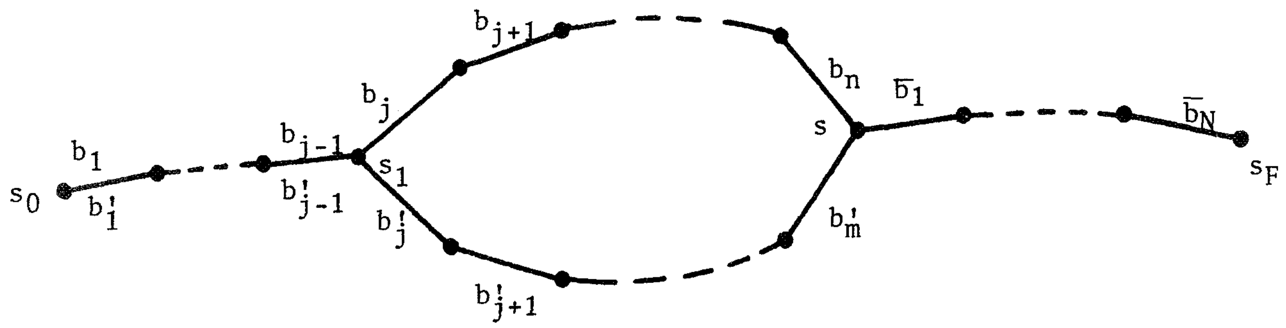


FIGURE 42. Diverging and Remerging Paths Through a Graph.

s_F, we have

$$\min_{j-1 \leqq k \leqq n} \sum_{i=1}^{k} v(b_i) \geqq V^*_{\min} \qquad (5)$$

$$\min_{j-1 \leqq k \leqq m} \sum_{i=1}^{k} v(b'_i) \geqq V^*_{\min} \qquad (6)$$

where if equality holds in (5), then b_j must be better than the jth branch of the selected path, or else it must <u>be</u> the jth branch of the selected path, and similarly for equality in (6).

Let us call the path $b_j, b_{j+1}, \cdots, b_n, \overline{b}_1, \cdots, \overline{b}_N$ simply P, and the other path $b'_j, b'_{j+1}, \cdots, b'_m, \overline{b}_1, \cdots, \overline{b}_N$, P'. Suppose that the minimum node value along P is assumed at or past s; that is,

$$\text{min along P} = V(s) + \sum_{i=1}^{k} v(\overline{b}_i) \quad \text{for some } k \geqq 0$$

Then since $V(s) > V\ (s)$,

$$\begin{aligned} \text{min along P} &\geqq V'(s) + \sum_{i=1}^{k} v(\overline{b}_i) \\ &\geqq \text{min along P'} \end{aligned}$$

By the lemma, if $b'_j, b'_{j+1}, \cdots, b'_m$ were part of the final path, then P' would be the final path from s_1 to s_F. But since the minimum along P exceeds that along P', P' does not satisfy MF2. Hence the primed path is not part of the final path.

Now suppose that the minimum along P is assumed between s_1 and s. Then

$$\text{min along P} = \min_{j-1 \leqq k \leqq n} \sum_{i=1}^{k} v(b_i)$$

Hence by (5),

$$\text{min along } P \geqq V^*_{\min} \qquad (7)$$

But $V^*_{\min}$ is the greatest such minimum, so equality must hold in (7). Now if P is not actually the last segment of the final path. it is because there is another path of minimum value $V^*_{\min}$ whose first branch after s_1 is ordered better than b_j. But if this were so, s would never be reached along the unprimed path, contradicting the assumption. Therefore P must be the last segment of the final path, precluding the appearance of b'_j, b'_{j+1}, $\cdots$, b'_m in that path. This proves the theorem.

Based on Theorem 9, we can make some observations about what a clever unquantized Fano searcher should do in certain circumstances. Suppose that provision is made for storing a value U(s) corresponding to each node s in the graph, and that the searcher is allowed to modify U(s) upon each visit to s. Say that the searcher arrives at node s for the first time with value V. The searcher sets U(s) = V and begins F searching. Suppose that the searcher is eventually forced back to s, then back from s along the path on which it moved to s. Now suppose that later in the process the searcher arrives at s again, this time along a path whose last branch is not the last branch of the previous path to s, and with a different value V'. If V' < U(s), then by Theorem 9, we know that F searching is fruitless, since the present path to s is not the initial part of the path selected. Hence the searcher should just move back immediately, as if the threshold would have been violated by an F move. This way, some futile computations are avoided.

If $V' = U(s)$, although we did not show it, it can be verified that in Theorem 9, if $V'(s) = V(s)$ and the searcher reaches s along the unprimed path first, then the primed path is not part of the final path. Thus the searcher should take the same action as in the case $V' < U(s)$.

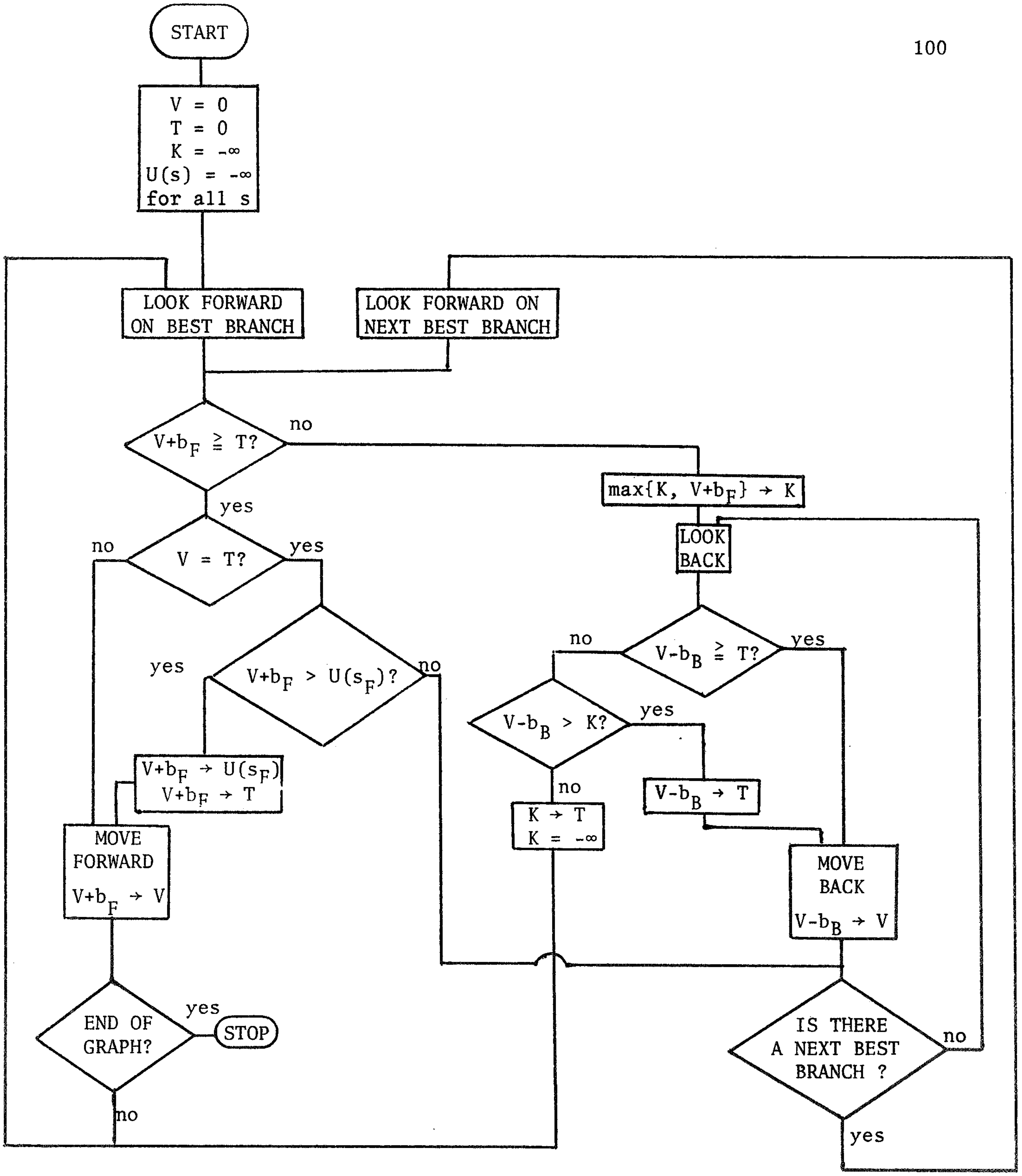
On the other hand, if $V' > U(s)$, we know that the original path to s is not the initial part of the selected path, while the new path to s might be. Thus the prescribed F searching must be done. In addition, the searcher should set $U(s) = V'$.

It is clear therefore that U(s) at any time is the greatest value with which the searcher has ever arrived at node s on any path up to that time. With this in mind, we might initialize all the U(s) to $-\infty$. Then upon each visit to s the searcher will compare the current value V to U(s) to decide whether F searching should be attempted and whether U(s) should be updated.

A flowchart of this Fano graph searcher is given in Figure 43. Unfortunately, the requirement for a storage location for each node examined makes us forfeit one of the principal advantages of Fano-type searches, namely the small memory requirements.

If we are willing to make the assumption that ties are resolved consistently we can prove a theorem analogous to Theorem 9 for the Zigangirov-Jelinek algorithm, and make analogous observations about expediting the search. The corresponding change in the algorithm is that step (1) should be replaced by the following procedure:

(1a) Compute the values of all successors of the top node and put in the stack any successors which have never appeared before. For the remaining successors:



s_F: Successor of present node along b_F

FIGURE 43. Unquantized Fano Graph Search Algorithm.

-- If the node appeared before but has since been deleted, put the new node in the stack only if its value is greater than its value at the previous appearance.

-- If the node appeared before and remains in the stack, reatin only the appearance of higher value.

There seems to be no simple way to implement step (1a) of the algorithm.

4. B. 3. Biasing Branch Values

In our discussion of sequential decoding in Chapter I, we noted that a bias term was added to the branch metrics to make the correct path tend to increase and incorrect paths tend to decrease in value. In order for the sequential search procedures to produce a reasonably good path it is necessary to bias the branch values in such a way that the path satisfying our sets of conditions does in fact have a large terminal value.

Suppose that all the branch values are bounded, say $m \leqq v(b) \leqq M$ for all b. We want to replace each branch value v(b) by a biased branch value $v(b) - B$.

If we choose $B \leqq m$, then each biased branch value is non-negative. Therefore with regard to the sets of conditions, all paths emanating from a node s have the same minimum value, namely the value at node s. Hence the choice of final path is arbitrary with the Zigangirov-Jelinek algorithm and is made only on the basis of comparing single branch values with the unquantized Fano algorithm. There is very little reason to hope that the path selected with this bias is a good path. The process of searching is quite simple, however, requiring only a single computation per branch.

At the other extreme, taking $B \geq M$, each branch value is non-positive so that every path assumes its minimum at the end of the path, i.e., at s_F. For a path $b_1, b_2, \cdots, b_L$ the biased value is $\sum_{i=1}^{L} v(b_i) - LB$; thus a path is selected on the basis of a compromise between length and value. As $B \to \infty$, the searcher selects the shortest path, and if there is more than one, the best of them is chosen. If all paths are of the same length, with any $B \geq M$ the path selected is optimum. The catch is that as B increases, searching becomes more difficult, and for $B \geq M$, virtually every node in the graph is examined, so that the sequential search algorithms offer no improvement over the optimum search algorithms.

This shows that the choice of bias results in a tradeoff between computational difficulty and nearness of the final path to optimality. A reasonable value of B must lie in the range $m < B < M$, but the exact value depends on the particular problem at hand and the level of performance demanded.

4. C. AN EXAMPLE

Figures 44-47 give an example of graph-searching by means of the unquantized Fano algorithm. There are two optimum paths through the graph, shown in heavy lines in Figure 44.

The action of the Fano searcher on the unbiased graph is displayed in Figure 45. Nodes which are encircled are those visited during the search. Arrows along branches indicate F looks along those branches. The path chosen is not optimum, but has a value of -1; the optimum path value is zero.

When all branch values are biased by 1, the Fano searcher selects one of the optimum paths, as shown in Figure 46. In this case, more forward looks are required than before, but the same number of nodes are visited.

When the bias is increased to 2 (Figure 47), the searcher selects the other optimum path. Of course, since this path is shorter than the previous one, it would be possible to force its selection even if it were not optimum.

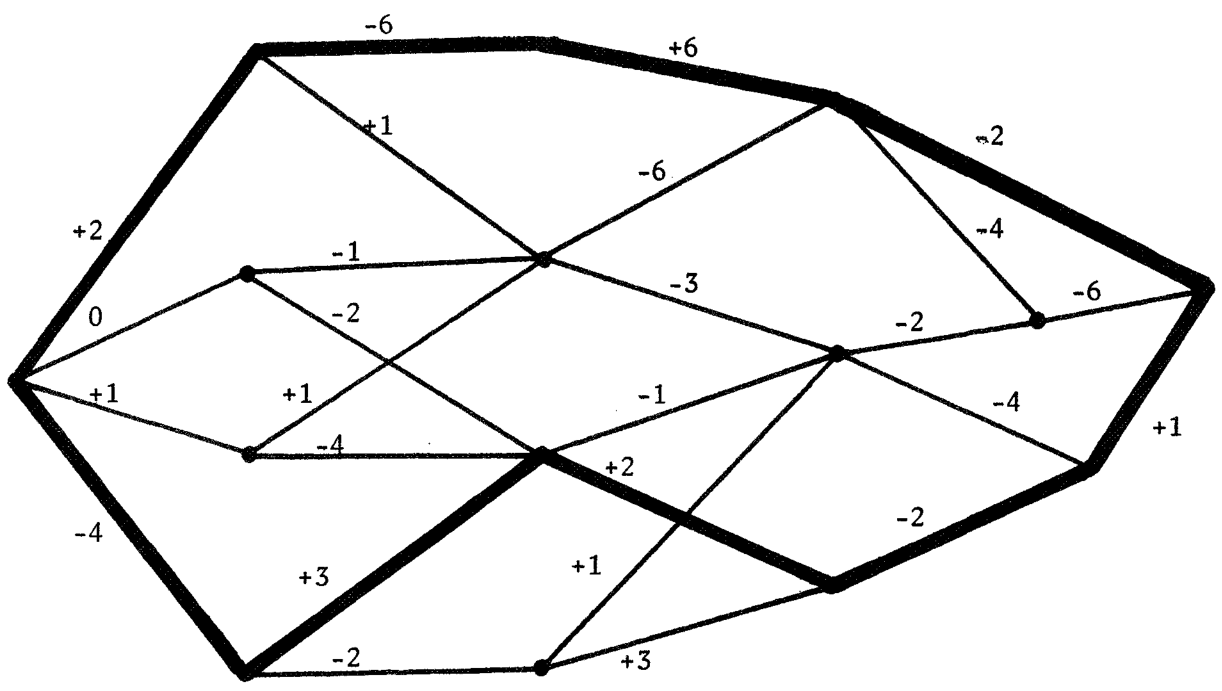


FIGURE 44. A Graph with Two Optimum Paths.

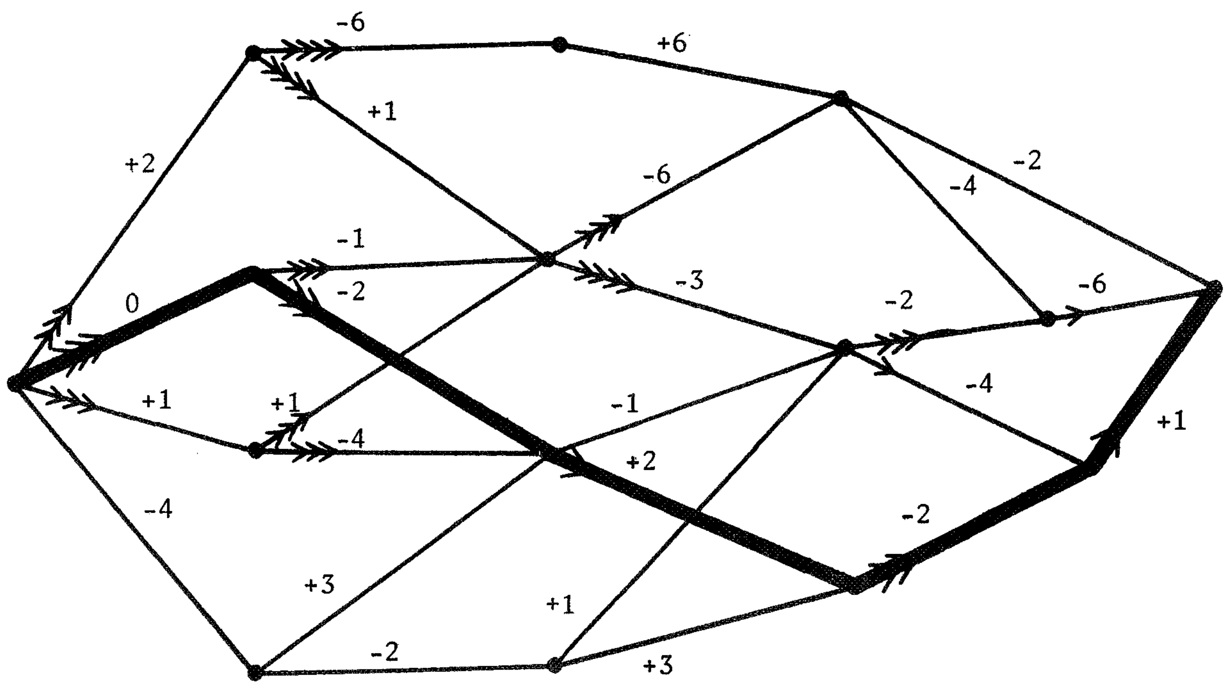


FIGURE 45. Sequential Search of the Graph of Figure 44 with Zero Bias.

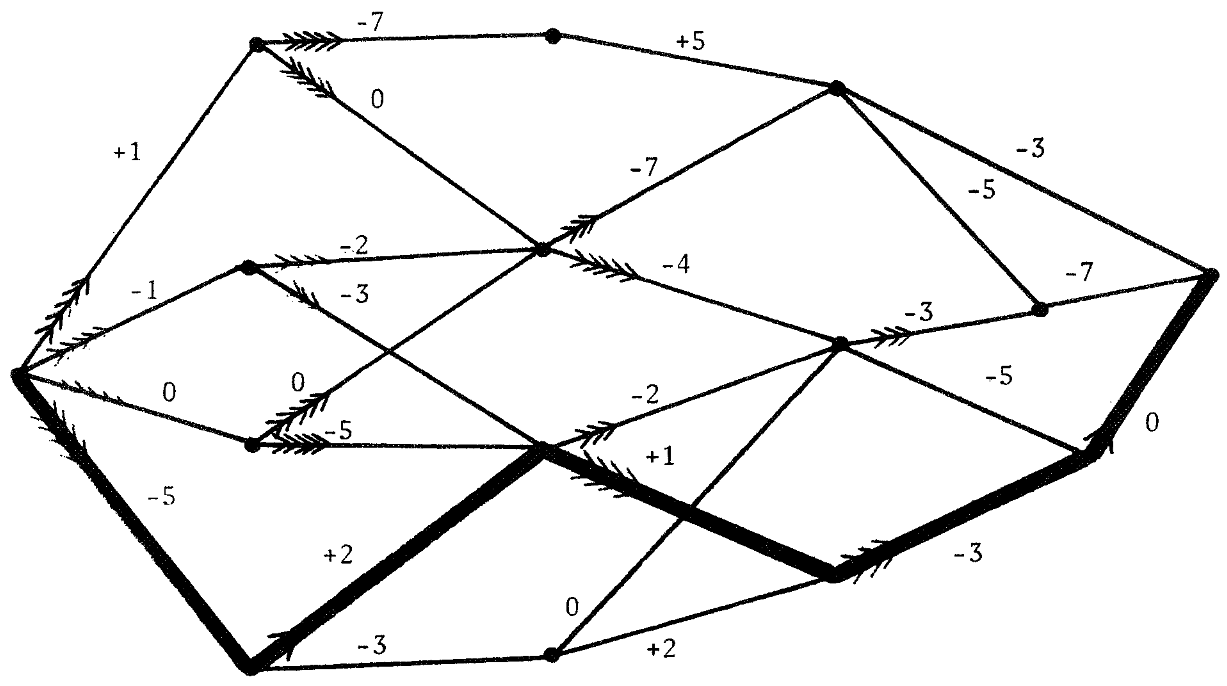


FIGURE 46. Sequential Search of the Graph of Figure 44 with Bias = 1.

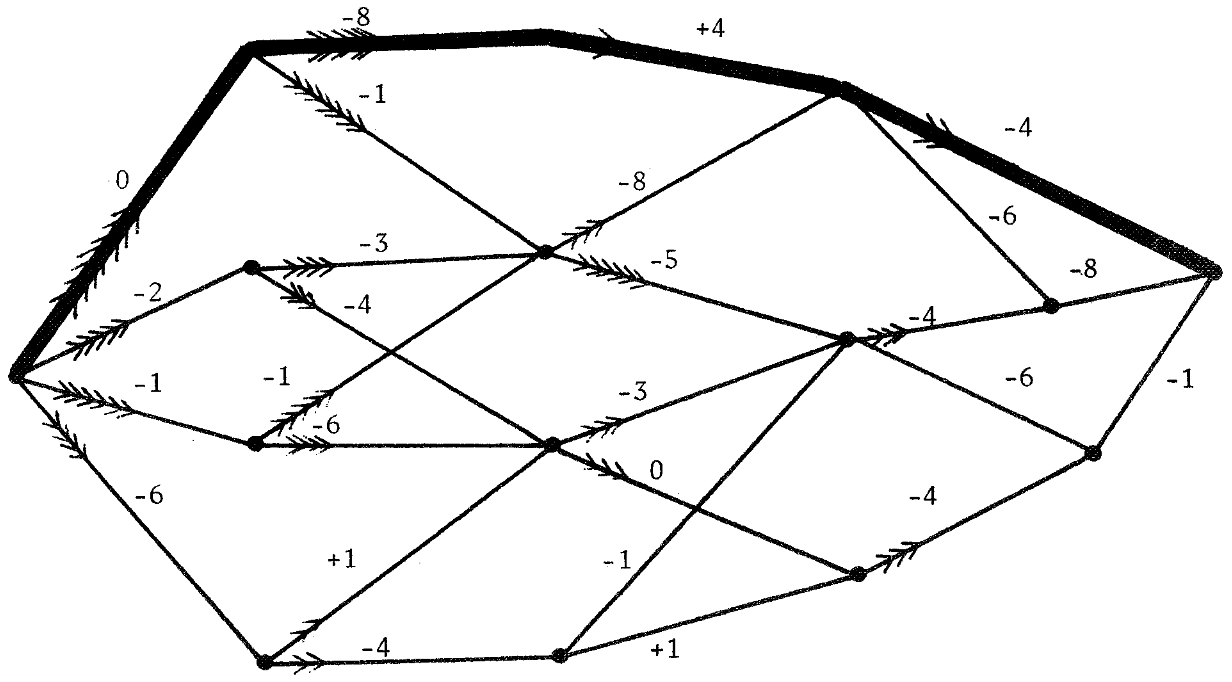


FIGURE 47. Sequential Search of the Graph of Figure 44 with Bias = 2.

APPENDIX A

SUMMARY OF THE LOWER-BOUND ARGUMENT FOR $\Pr\{C_0 \geqq N\}$

In this appendix, we give a brief summary of the argument used by Jacobs and Berlekamp[3], making a minor modification in the argument. Many of the equations and inequalities hold asymptotically; that is, we may have $x \geqq y + O(r)$, where $O(r) \to 0$ as $r \to \infty$. In this case we put $x \overset{\cdot}{\geqq} y$, and it will be clear from context what quantity is growing large. All logarithms (and hence rates) will be taken to the natural base.

Suppose we have a block code of length n and $M = e^{nR}$ codewords. A maximum likelihood list-of-N decoder is one which maps the received sequence into a list of the N most likely transmitted sequences, given the received sequence. A list decoder makes an error if the actual transmitted sequence is not on its list. Shannon, Gallager, and Berlekamp[23] have shown that the probability of error with list decoding is lower-bounded as follows:

$$P_e(N) \overset{\cdot}{\geqq} e^{-nE_{sp}(\mathcal{R})} \qquad (1)$$

where $\mathcal{R} = \frac{1}{n} \ln(M/N) = R - \frac{1}{n} \ln M$ is the list decoding rate and

$E_{sp}(\mathcal{R}) = \underset{\rho \geqq 0}{\text{lub}} \{E_0(\rho) - \rho\mathcal{R}\}$, $E_0(\rho)$ being given by (I-2).

Now suppose $E_0(\rho)$ is convex; if it is not, it can be replaced by its convex hull and the present analysis is unchanged. By differentiating with respect to ρ, we obtain

$$E_{sp}(\mathcal{R}) = E_0(\rho^*) - \rho^*\mathcal{R}$$

where ρ^* is the solution of $\mathcal{R} = E_0'(\rho)$. Suppose we are given ρ and the list size N. It is possible to make $\mathcal{R} \geqq E_0'(\rho)$ by choosing

$$\frac{M}{N} \geqq e^{nE_0'(\rho)} \qquad (2)$$

Then $E_{sp}(\mathcal{R}) \mathrel{\dot{\leqq}} E_0(\rho) - \rho\mathcal{R} \mathrel{\dot{\leqq}} E_0(\rho) - \rho E_0'(\rho)$, a fact which is most easily seen with the aid of a graph. Thus by (1),

$$P_e(N) \mathrel{\dot{\geqq}} e^{-n[E_0(\rho) - \rho E_0'(\rho)]} \qquad (3)$$

Given a tree code of rate R with ν symbols per branch, we obtain a block code of rate R and length n by terminating the tree after n/ν branches (assuming for simplicity that n is a multiple of ν). For a given N, if we choose n to be the least multiple of ν such taat

$$N \mathrel{\dot{\leqq}} e^{n[R - E_0'(\rho)]} \qquad (4)$$

where ρ is the solution of $R = E_0(\rho)/\rho$, then (2) is satisfied and so (3) holds. Thus

$$\begin{aligned} P_e(N) &\mathrel{\dot{\geqq}} e^{-n[E_0(\rho) - \rho E_0'(\rho)]} \\ &= e^{-n[R - E_0'(\rho)]\rho} \\ &\mathrel{\dot{\geqq}} N^{-\rho} \end{aligned}$$

where (4) is used in the last step.

Jacobs and Berlekamp argue as follows. Suppose a certain sequence is received. By JB2, only the first n digits of the received sequence affect the decoder's operation up to depth n/ν. The probability that N computations are performed is at least as great, by JB1, as the probability that at least N paths through the tree are examined to depth n/ν before the correct path is reached, that is, the probability that the correct path lies among the last M-N paths considered, where $M = e^{-nR}$. This last quantity is given by the sum over the last M-N paths examined of the probability of receiving the given sequence conditioned that that path was sent. But this is at least as great as $P_e(N)$ for this received sequence, since $P_e(N)$ is obtained by summing this conditional probability over the M-N paths for which it is minimum. This holds for all received sequences, so if C is the computation to decode the first n/ν branches, then

$$\Pr\{C \geqq N\} \geqq P_e(N) \overset{\bullet}{\geqq} N^{-\rho} \qquad (5)$$

The above argument can be modified to yield a lower bound on the distribution of C_0, the computation required to decode the first branch, rather than a lower bound on C as given by Jacobs and Berlekamp. The solution is to truncate the tree code and form a block code in such a way that only nodes whose examination is chargeable to C_0 are included. Suppose there are u branches emanating from each node. We truncate the tree as before, but also prune the part of the tree starting with the first correct branch, but leaving in the correct path. See Figure 48. The number of codewords in this block code is

$$M_T = u^{n/\nu} - u^{n/\nu - 1} + 1 = \frac{u-1}{u} u^{n/\nu}$$

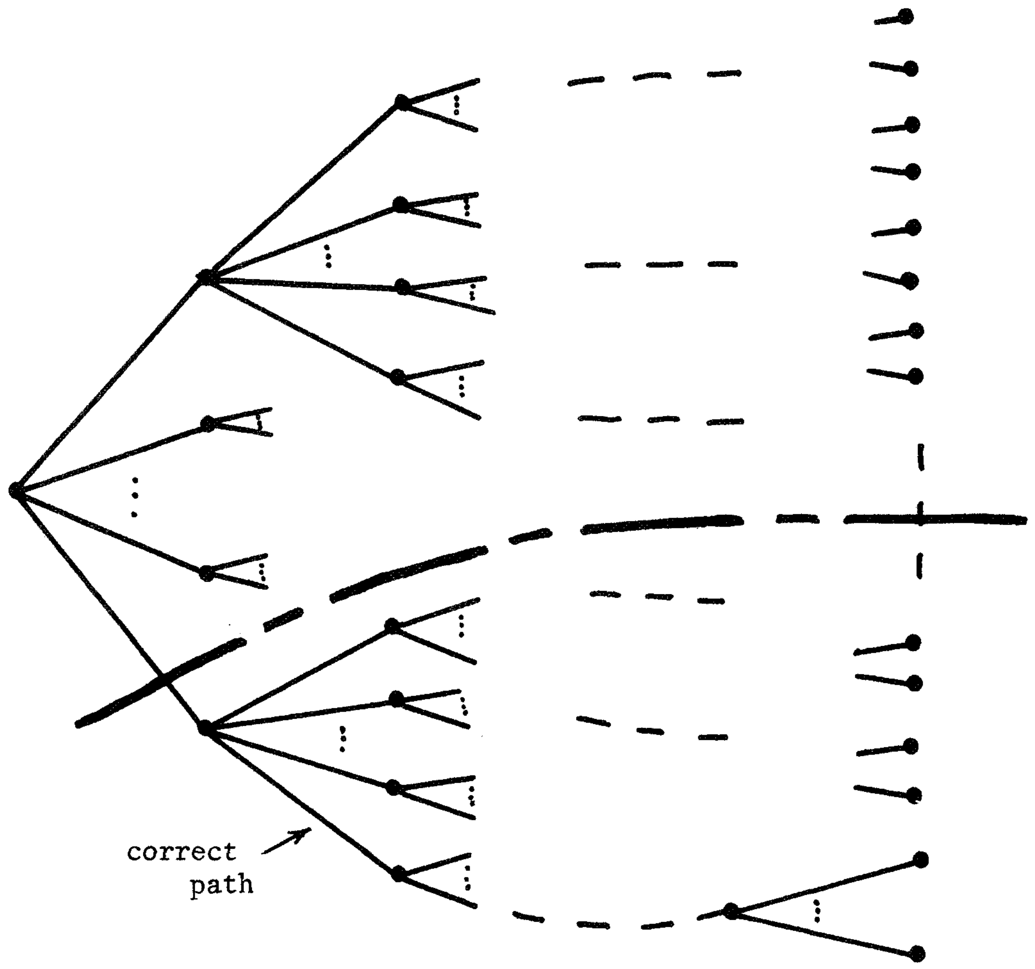


FIGURE 48. The Truncated Tree for Deriving a Lower Bound on $\Pr\{C_0 \geq N\}$. The Set of M_T Codewords in the Truncated Code Consists of All Paths Above the Dashed Line, As Well As the Correct Path.

so that $R_T = \frac{1}{n} \ln M_T \doteq \frac{1}{n}\frac{n}{\nu} \ln u + \frac{1}{n} \ln \frac{u-1}{u} = R + \frac{1}{n} \ln \frac{u-1}{u}$

Since $R_T \doteq R$, the same argument holds; by excluding all nodes whose examination should be charged to decoding other branches than the first, we have not affected the rate asymptotically, so

$$\Pr\{C_0 \geq N\} \overset{\cdot}{\geq} N^{-\rho}$$

Note that from (4), we must be able to choose

$$n \geq \frac{1}{R - E_0'(\rho)} \ln N$$

in order to derive the lower bound on $\Pr\{C_0 \geq N\}$, so that (I-6) holds when

$$L \geq \frac{1}{\nu} \frac{1}{R - E_0'(\rho)} \ln N$$

That is, in applying the lower bound to $C_0(L)$ for finite trees, the range of N for which the bound holds grows exponentially with the tree length L.

APPENDIX B

THE JELINEK DECODER PROGRAM

We now describe in some detail the operation of the Jelinek decoder as we have programmed it on the UNIVAC 1107, in the hope that the techniques may be of use to other investigators.

In order to extend from a node, three items must be known: the value at the node, its depth into the tree, and the encoder state at the node. The encoder state is necessary to produce the code symbols on the branches emanating from the node; the depth is needed to compare these code symbols with the proper span of symbols in the received sequence; and the branch values arising from this comparison must be added to the value of the extended node to yield the new node values. Therefore, for every node the decoder encounters, the value, depth, and encoder state must be saved so that the node can be extended, if necessary.

For the top node on the stack (i.e., one of the nodes in the highest non-empty bin) these items are kept in three accumulators which are given the symbolic labels VALUE, DEPTH, and STATE. For other nodes on the stack, the information is contained in what we call _node descriptions_, which occupy six contiguous words of computer memory, in which are stored, respectively, the node's value, depth, encoder state, a stack pointer, a path pointer, and a flag. (The last three items will be discussed later.)

Initially the top node is the origin node, which is at depth zero, and the encoder state is zero. Thus DEPTH and STATE are initially set to zero. The value of the origin node is usually considered to be zero, but it is convenient to avoid the possibility of encountering nodes of negative value. We therefore bias the node values by initializing VALUE to a sufficiently large number so that, with overwhelming probability, no node placed on the stack has negative value.

As the search through the tree proceeds, the nodes encountered are added to the stack, and the necessary information saved, either as the contents of the three designated accumulators or in stored node descriptions (or both). Once a node description is stored it is never physically deleted from memory, even if the node reaches the top of the stack and is extended. Therefore there are many node descriptions resident in memory representing nodes which are no longer on the stack, by virtue of step (2) of the algorithm. On the other hand, every node on the stack (except possibly the top node) is represented by a node description. The determination of which of the resident node descriptions represent nodes still on the stack, and the ordering of the stack contents into bins, are the functions of the stack pointers and an array called the <u>bin index</u>. The bin index consists of two entries for each bin: the first is the number of nodes in the bin; the second is the address of the first word of the node description corresponding to one of the nodes in the bin. The stack pointer in this node description contains the address of the first word of the node description for another node in the bin, and so on. (The stack pointer in the description of the last node in the bin is meaningless.) Therefore, the contents of bin k can be found successively by using the second entry of the bin index corresponding to bin k and the stack pointers in the node descriptions referenced. In order to place an upper bound on the number of bins and hence on the size of the bin index array, a lower bound must be set on the bin spacing H. We require H to be at least as great as the maximum positive branch value.

To see how decoding proceeds, let us first restrict ourselves to the BSC and to the use of complementary codes, i.e., codes in which the code symbols on the 1 branch emanating from any node are complements of the symbols on the 0 branch*. Then when the top node is extended and the two branches compared to the received sequence, only two outcomes are possible: either (1) one branch agrees with the received sequence in both digits and the other disagrees in both, or (2) each branch disagrees in exactly one digit. We consider the two cases separately.

Case (1). We call this the typical case since almost all extensions of nodes on the correct path and roughly half the extensions of nodes not on the correct path are of this kind. Table 2 shows that the branch value for the branch with two agreements will be +4 (or +8 on the third BSC), so that the value of the corresponding node exceeds the value of the extended node. Since the extended node was in the highest non-vacant bin, and its successor along the +4 (+8) branch belongs in the same or a higher bin, we may take the successor to be

* An equivalent condition is that neither generator begin with a zero.

the new top node and adjust VALUE, DEPTH, and STATE accordingly; there is no need to store its description. For the other successor, however, we must store a node description. The first three items in the description can be gotten easily from the contents of VALUE, DEPTH, and STATE and the branch value. To set the stack pointer, we note that a node of value V belongs to bin k if $kH \leqq V < (k+1)H$. That is, the correct value of k is the integer part of V/H. We restrict H to be a power of 2, say $H = 2^r$, so that k can be found by a simple r-bit shift operation. Having found the bin to which the node belongs, we use the bin index: increase the bin count by one, set the stack pointer of the new node description to the address presently specified in the bin index, and reset the bin index pointer to the address of the first word of the new node description. Thus after insertion, the bin index pointer points to the new description and the stack pointer of the new description points to the description which had been referenced by the bin index pointer. After setting the path pointer and flag, which we discuss later, we are ready to extend again using the updated contents of VALUE, DEPTH, and STATE.

Case (2). Since both branch values are negative, it is likely that there are nodes on the stack in bins higher than the bins to which the two successors belong. Therefore the new top node is not readily available as it is in Case (1) -- we must search for it. First the two new nodes are stored in the same way the one was stored in Case (1). Then the decoder scans down the bin index, starting with the bin to which the extended node belonged, looking for a bin whose count is non-zero. When the first non-empty bin is located, its count is decreased by one, VALUE, DEPTH, and STATE are loaded from the node description referenced by the bin index pointer, and the bin index pointer is reset to the address contained in the stack pointer of that node description. Thus the node which had been the second node in the bin is now referenced by the bin index. The decoder is now ready to extend again.

We turn now to the Gaussian channel, leaving in force the restriction to complementary codes*. It is no longer meaningful to use the terms "agreement" and "disagreement," but from Table 2 we see that there are still two cases: either (1) one branch value is non-negative and the other is negative, or (2) both are negative. It is clear that the same decoder actions described above are applicable on the Gaussian channel.

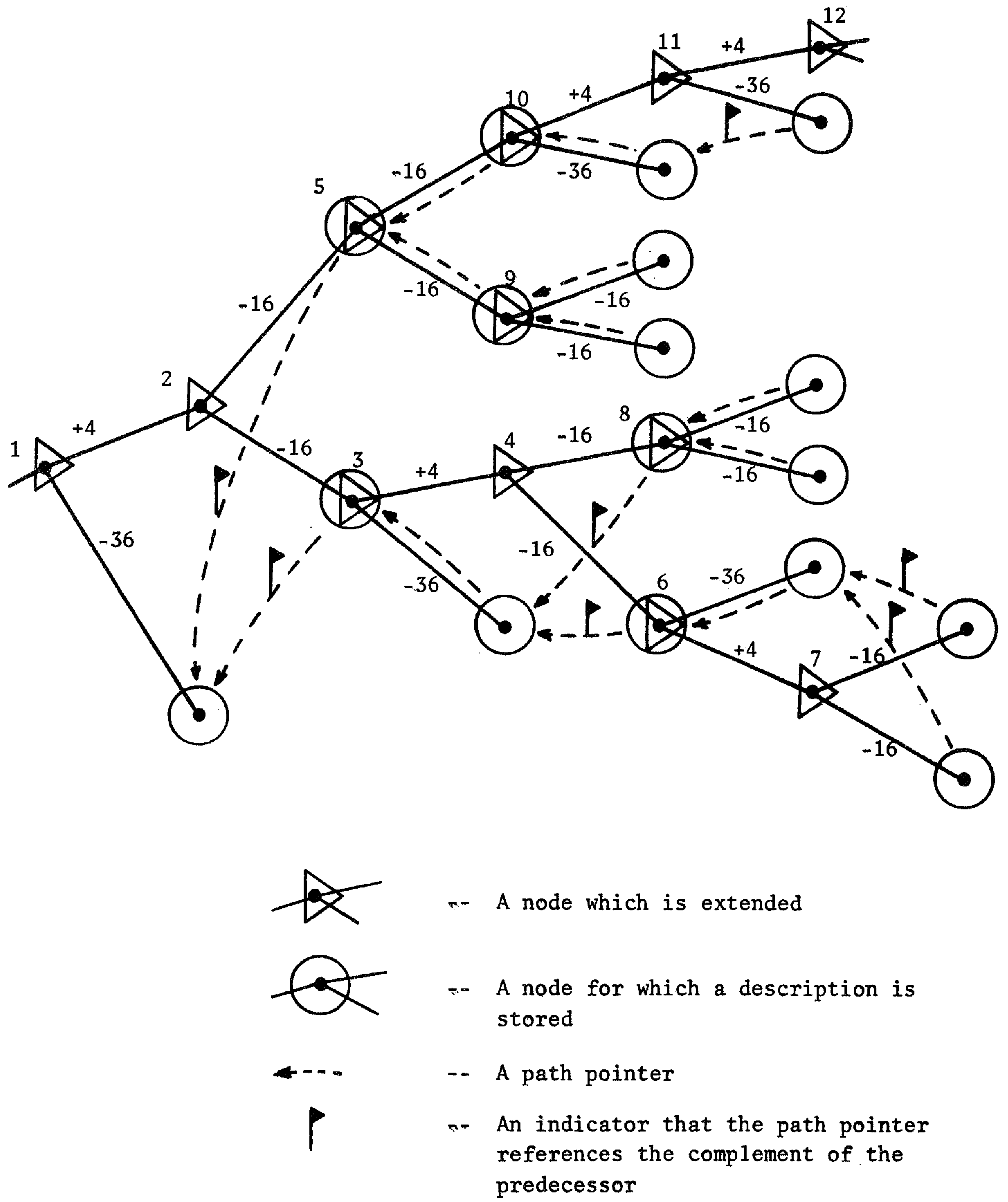
A transmitted frame as programmed consists of 256 branches corresponding to encoded information bits followed by a 35-branch tail corresponding to an encoded memory span of 0s, included to prevent high error probability in the last few information bits. The search in the tail differs from that in the information part of the tree in that only the successor along the 0 branch is considered. The two kinds of computations are still performed: if the branch value is non-negative there is no storage and the successor is extended immediately; if the branch value is negative, the successor is stored and a search for the new top node is undertaken.

If the contents of DEPTH is 291, this indicates that the top node is at the end of the tree and the search is completed. This brings up the problem of recovering the information symbols on the path chosen. If a description had been stored for every node examined, then the

* This is no sacrifice, since only complementary codes would be used in practice.

path pointer in every node description could have been set to the address of the description of its predecessor. Then the path pointers would specify the path from the final node back to the origin. Since typically only one node is stored, this is impossible. However, at every extension at least one node is stored, and therefore for every node encountered, either its predecessor or the complement of its predecessor (or both) is stored. Thus we set the path pointer in each node description to the address of the description of the predecessor if it is stored, or, if it is not, to the address of the description of the complement of the predecessor, and we use the sixth element of the node description, the flag, to indicate whether the node referenced by the path pointer is the predecessor or its complement. Now when the top node is at the end of the tree, the decoder can step back toward the origin using the path pointers, flags, and encoder states to produce the information sequence along the path selected.

Figure 49 illustrates the pattern of extensions and storage by the decoder program for a typical segment of a tree with branch values for the BSC, p = .033. The extensions are numbered sequentially.



-- A node which is extended

-- A node for which a description is stored

-- A path pointer

-- An indicator that the path pointer references the complement of the predecessor

FIGURE 49. A Typical Search by the Jelinek Decoder.

REFERENCES

1. G. D. Forney, Jr., "Final Report on a Coding System Design for Advanced Solar Missions," Codex Corp., Watertown, Mass., Contract NAS2-3637, December 1967, Appendix A.

2. J. M. Wozencraft, "Sequential Decoding for Reliable Communication," IRE Convention Record, 1957, Part 2, pp.11-25.

3. I. M. Jacobs and E. R. Berlekamp, "A Lower Bound to the Distribution of Computation for Sequential Decoding," IEEE Trans. on Information Theory, Vol. IT-13, pp. 167-174, April 1967.

4. J. E. Savage, "The Computation Problem with Sequential Decoding," MIT Lincoln Laboratory Technical Report No. 371, February 1965.

5. R. M. Fano, "A Heuristic Discussion of Probabilistic Decoding," IEEE Trans. on Information Theory, Vol. IT-9, pp. 64-74, April 1963.

6. K. Sh. Zigangirov, "Some Sequential Decoding Procedures," Problemy Peredachi Informatsii, Vol. 2, No. 4, pp. 13-25, 1966.

7. F. Jelinek, "A Fast Sequential Decoding Algorithm Using a Stack," IBM J. Res. Dev., Vol. 13, pp. 675-685, November 1969.

8. J. M. Wozencraft and I. M. Jacobs, Principles of Communication Engineering. New York: Wiley, 1965.

9. R. G. Gallager, Information Theory and Reliable Communication. New York: Wiley, 1968.

10. D. D. Falconer, "A Hybrid Sequential and Algebraic Decoding Scheme," Ph.D. Dissertation, MIT, February 1967.

11. F. Jelinek, "An Upper Bound on Moments of Sequential Decoding Effort," IEEE Trans. on Information Theory, Vol. IT-15, pp. 140-149, January 1969.

12. H. L. Yudkin, "Channel State Testing in Information Decoding," Sc.D. Dissertation, MIT, September 1964.

13. J. L. Massey and M. K. Sain, "Trunk and Tree Searching Properties of the Fano Sequential Decoding Algorithm," Proc. 6th Annual Allerton Conf., pp. 153-160, Univ. of Illinois, October 1968.

14. J. L. Massey and M. K. Sain, "Distribution of the Minimum Cumulative Metric for Sequential Decoding," International Symposium on Information Theory, Ellenville, N. Y., January 1969.

15. J. L. Massey, Private communication.

16. D. J. Costello, "Construction of Convolutional Codes for Sequential Decoding," University of Notre Dame Technical Report No. EE-692, August 1969.

17. G. B. Dantzig, "On the Shortest Route Through a Network," _Mgmt. Sci._, Vol. 6, pp. 187-190, January 1960.

18. R. G. Busacker and T. L. Saaty, _Finite Graphs and Networks_. New York: McGraw-Hill, 1965.

19. M. Pollack and W. Wiebenson, "Solutions of the Shortest-route Problem: A Review," _Operations Res._, Vol. 8, pp.224-230, 1960.

20. R. E. Bellman, _Dynamic Programming_. Princeton, N. J.: Princeton University Press,1957.

21. J. K. Omura, "On the Viterbi Decoding Algorithm," _IEEE Trans. on Information Theory_, Vol. IT-15, pp. 177-179, January 1969.

22. A. J. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," _IEEE Trans. on Information Theory_, Vol. IT-13, pp. 260-269, April 1967.

23. C. E. Shannon, R. G. Gallager, and E. R. Berlekamp, "Lower Bounds to Error Probability for Coding on Discrete Memoryless Channels," _Information and Control_, Vol. 10, pp. 65-103, 522-552.