Sequential Decoding — The Computation Problem*

By J. E. SAVAGE

(Manuscript received August 23, 1965)

Sequential decoding is a technique for encoding and decoding at moderate cost with a decoding reliability which approximates that of the optimum, and expensive, maximum-likelihood decoder. The several known sequential decoding algorithms enjoy a cost advantage over the maximum-likelihood decoder because they allow the level of the channel noise to regulate the level of the decoding computation. Since the average level of the required decoding computation for sequential decoders is small for source rates below a rate R_{comp} *, such a decoder can be realized for these rates with a relatively small logic unit and a buffer. The logic unit is normally designed to handle computation rates which are less than two or three times the average computation rate; the buffer serves to store data during those noisy periods when the required computation rate exceeds the computation rate of the logic unit.*

If the periods of high computation, which are caused by noise, are too frequent or too long, the buffer, which is necessarily finite in capacity, will fill and overflow. Since data are lost during an overflow, continuity in the decoding process cannot be maintained. The decoder, then, cannot continue to decode without error. For this reason, buffer overflow is an important event. In addition, since errors in the absence of overflow are much less frequent than are overflows themselves, the overflow event is of primary concern in the design of a sequential decoder.

This paper presents some recent analytical results concerning the probability of a buffer overflow. In particular, it is shown that this probability is relatively insensitive to both the buffer capacity and the maximum speed of the logic unit for moderate capacities and speeds. By contrast, it is shown that the overflow probability decreases rapidly with a decrease in the source

* The results of this paper are drawn from the author's thesis which has been accepted by the Massachusetts Institute of Technology in partial fulfillment of the requirements for the degree of Doctor of Philosophy. The research reported here was supported by the M.I.T. Research Laboratory of Electronics and the M.I.T. Lincoln Laboratory.

rate and is more than squared by a halving of this rate. These sensitivities are basic to sequential decoding; they exist because the required computation level is large during intervals of high channel noise and grows exponentially with the length of such an interval. It is also shown that the dependence of the overflow probability on the source rate is intimately related to exponents appearing in the coding theorem. In addition, the results presented agree with the limited experimental evidence available.

I. INTRODUCTION

Sequential decoding procedures are important because they achieve, at modest cost, a decoding error rate which approximates the error rate of the optimum and expensive maximum-likelihood decoder. Sequential decoding procedures have this near-optimum performance at modest cost because they allow the level of the channel noise to determine the level of the decoding computation. The level of the decoding computation is a function of the source rate as well as the channel noise and if the source rate is held at less than a computational cutoff rate, R_{comp}, the computation level *on the average* will be small.[1,2,16] Thus, a sequential decoder may be constructed from a logic unit capable of handling two or three times the average computation rate and from a buffer to store data during those noisy periods which require a computation rate which exceeds that of the basic decoding machine. The maximum likelihood decoder, however, always requires a very high computation rate and, in effect, is designed to handle the peak noise levels.

The buffer portion of the decoder stores data during periods of high computation and since it has finite capacity, it will fill and overflow if the high computation intervals are too frequent or too long. If and when a buffer overflow occurs, the decoder cannot continue to decode reliably since data which are important to the continuing decoding process are lost. Consequently, a buffer overflow forces a halt in the decoding process while both the encoding and decoding processes are restarted.

While errors occur after the onset of overflow, they may also occur in the absence of overflow. For a properly chosen code, however, it can be argued that errors in the absence of overflow occur much less frequently than do overflow, themselves. Consequently, it can be argued—and, indeed, it is found in practice—that the buffer overflow event is of primary concern in the design of a sequential decoder.

In this paper, we present some recent results[3] concerning the probability of a buffer overflow. In particular, we show by upperbounding this probability that it is relatively insensitive to machine speed and to the storage capacity of the buffer for moderate speeds and capacities. By

contrast, it is shown that the overflow probability decreases rapidly with a decrease in the source rate and that this probability is more than squared by a halving of the rate. It is found that these sensitivities are basic to sequential decoding and arise because the computation per decoded digit is large during intervals of high channel noise and grows exponentially with the length of such an interval. We show that the dependence of the overflow probability on the source rate is intimately connected with exponents found in the coding theorem.[4,5] In addition, the results represented here agree with the limited experimental evidence available.[6]

We assume throughout this paper that the encoding and decoding are done for a discrete memoryless channel (DMC) characterized by the channel transition probabilities

$$\{P(y_j \mid x_k), \qquad 1 \leqq k \leqq K, \qquad 1 \leqq j \leqq J\}$$

where x_k represents a letter from the channel input alphabet and y_j represents a letter from the channel output alphabet. The results for the DMC apply with qualifications to other channels.

In the following sections we introduce the Fano algorithm,[2] the vehicle for this study of sequential decoding.

II. THE DECODING PROCEDURE

2.1 *Tree Codes*

The Fano algorithm decodes data encoded from tree codes. We assume that this data arrives from a source as a sequence of digits and we make the assumption that these digits are statistically independent and are drawn from the b-letter alphabet, $A = \{a_1, a_2, \cdots, a_b\}$. A sequence of source digits drawn from this alphabet is encoded with a tree code as follows (see Fig. 1): A branch from the first node of the tree is selected which corresponds to the value of the first digit produced by the source. The same is true for the second and later source outputs. Thus, in the example of Fig. 1, the source sequence (1, 0, 2, $\cdots$) with letters from the alphabet {0, 1, 2} selects the sequence (112, 010, 122, $\cdots$) from the tree. The digits on these branches are then transmitted over the DMC.

We assume that each branch of the tree contains l channel symbols so that the source rate in bits per channel transmission is defined as

$$R = \log_2 b/l. \tag{1}$$

A variety of rates can be generated with tree codes.

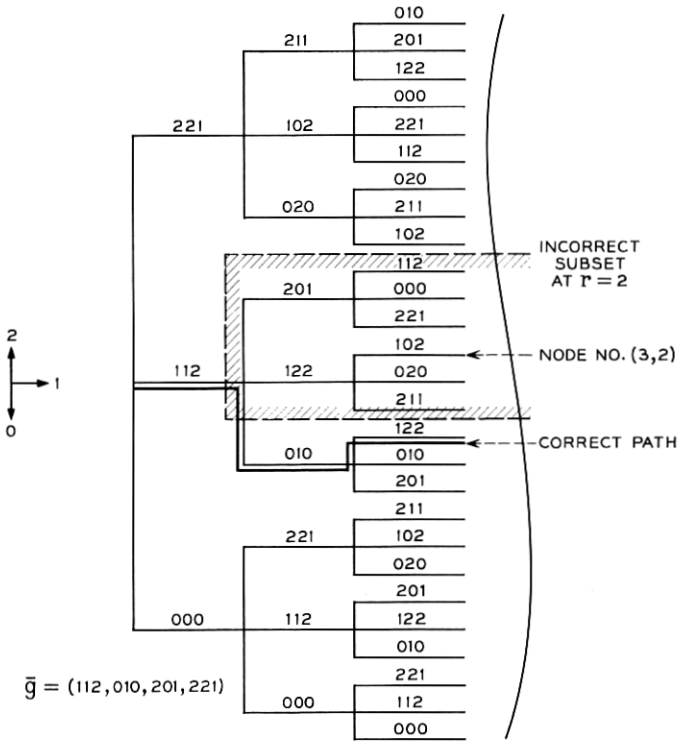

Fig. 1 — Tree code.

A class of tree codes which may be generated with a small amount of equipment is known as convolutional codes. Convolutional codes are generated with shift registers, multipliers and adders. An example of a convolutional encoder is given in Fig. 2 where circles represent multiplication and addition is taken modulo $b = 3$. This example generates the tree code of Fig. 1. Although we do not restrict the results of this paper to convolutional codes, it suffices to say that they can be decoded with sequential decoders with a small resultant error rate.[1,16]

2.2 *The Metric*

The Fano algorithm decodes by comparing the received channel sequence to paths in the tree code in search of a path which "matches well" to the received sequence. A "match" is measured with a "metric." *

* This is not a metric in the mathematical sense since it does not satisfy any of the rules for a metric. The word metric is used here to indicate the relative match (or mismatch) between the received channel sequence and a tree path.

Let $\bar{u}_s$, $\bar{v}_s$ represent a tree path and the received sequence, respectively, where each has s branches. Then, for the purposes of this paper we define the metric between $\bar{u}_s$, $\bar{v}_s$, $d(\bar{u}_s , \bar{v}_s)$, as

$$d(\bar{u}_s , \bar{v}_s) = \sum_{r=1}^{s} \sum_{h=1}^{l} \left[\log_2 \left\{ \frac{P(v_{rh} | u_{rh})}{f(v_{rh})} \right\} - R \right] \tag{2}$$

where u_{rh} , v_{rh} are the hth digits on the rth branches of $\bar{u}_s$, $\bar{v}_s$, respectively. $P(v_{rh}/u_{rh})$ is a channel transition probability. The function (v_{rh}) for $v_{rh} = y_j$ is given as

$$f(y_j) = \sum_{k=1}^{K} p_k P(y_j | x_k) . \tag{3}$$

This function may be viewed as the probability of channel output y_j when the channel inputs are assigned with probabilities $\{p_k\}$. (The function $f(y_j)$ and the probability assignment $\{p_k\}$ are chosen because they fit naturally into the random code bound to be presented later.)

This choice of metric is used because it lends itself to analysis and is a metric with which the Fano algorithm will operate. We now study this metric and observe from a simple combination of terms in (2) that $d(\bar{u}_s , \bar{v}_s)$ is monotonically increasing in increasing $P[\bar{v}_s | \bar{u}_s]$ which is the probability of receiving the sequence $\bar{v}_s$ when the sequence $\bar{u}_s$ is transmitted. This fact plus the fact that all tree paths with the same number of branches are assumed equiprobable imply that $P[\bar{v}_s | \bar{u}_s]$ is propor-

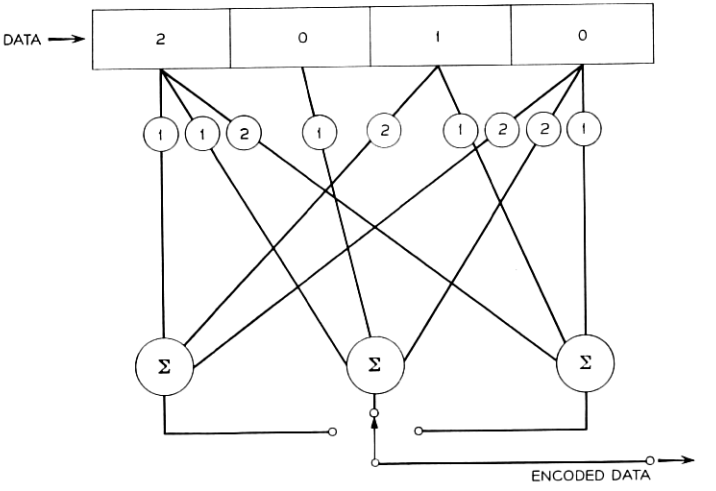

Fig. 2 — Convolutional encoder.

tional from Baye's Rule, to $P[\bar{u}_s \mid \bar{v}_s]$, the *a posteriori* probability of sequence $\bar{u}_s$ given that sequence $\bar{v}_s$ is received. Equivalently, this implies that $d(\bar{u}_s, \bar{v}_s)$ is monotonically increasing in the *a posteriori* probability of tree path $\bar{u}_s$. Thus, as the decoder progresses into the tree we expect $d(\bar{u}_s, \bar{v}_s)$ to increase if $\bar{u}_s$ represents the correct path (see Fig. 3). However, if the decoder branches onto an incorrect path, we expect the path to decrease in probability (for a properly chosen code) and to see $d(\bar{u}_s, \bar{v}_s)$ decrease (see Fig. 3). Although this behavior is typical, occasional noisy intervals will cause the correct path to decrease in metric and searching will be required to distinguish it from incorrect paths.

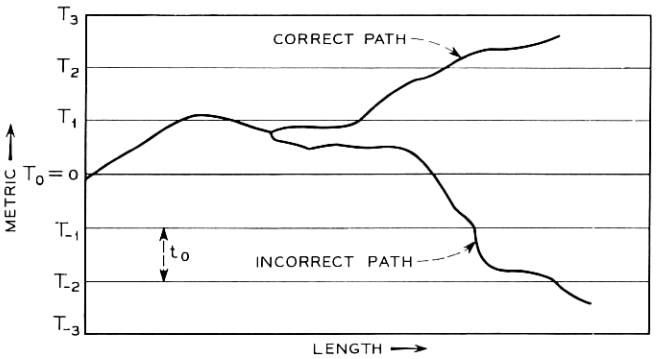

Fig. 3 — Criteria and typical paths.

2.3 *The Fano Algorithm*

The Fano algorithm is a set of rules for searching tree paths using the metric given by (2).* Since the algorithm is designed to find the transmitted tree path, it is programmed to follow a path which grows in metric. A path will be said to grow in metric if it crosses an increasing sequence of thresholds, such as those of Fig. 3.

The decoder is also programmed to search for other paths when the path being followed begins to decrease in metric and crosses a threshold from above. Such a decrease signals the presence of channel noise and indicates that searching will be required to distinguish between the correct path and incorrect paths.

The rules governing such a search, as well as the rules for determining which path to follow when two or more paths increase in metric, are given

* Other metrics with the same properties will also work.

by the flow chart† of Fig. 4. In that chart, a "most probable" branch at a node is that branch for which the increase in the metric is largest.

The "running threshold", which is simply called "threshold" in the flow chart, is really a sequence of thresholds which always lies below the node being examined in the decoder (see Fig. 5). It is used to determine whether the path being extended increases or decreases in metric. In

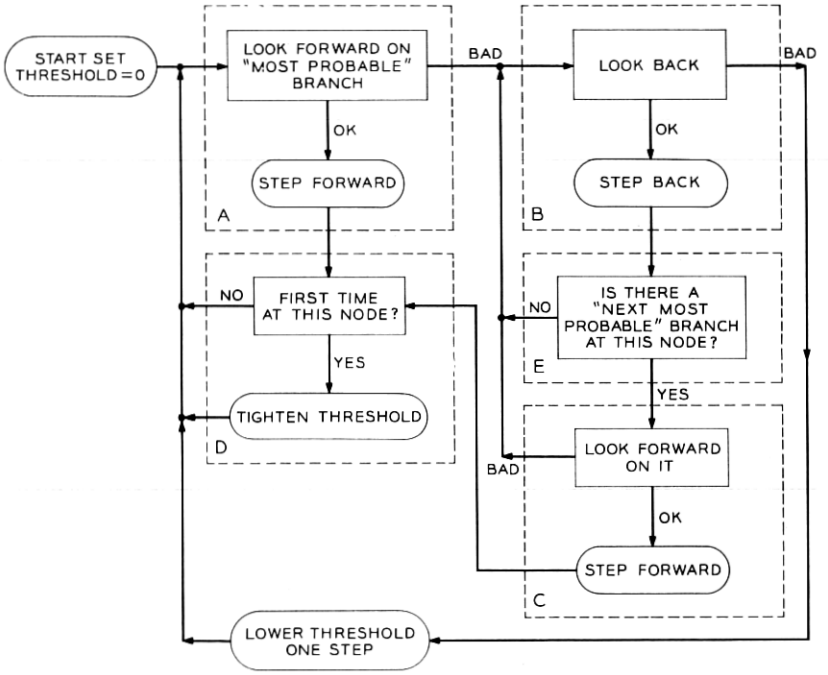

Fig. 4 — Flow chart of the Fano algorithm.

operations A, B, and C the metric on a node is compared with the running threshold. The indicators OK and BAD signify, respectively, that the metric is above or below this threshold.

In operation D, the statement "tighten threshold" means that the running threshold is to be increased until it lies just below the value of the metric on the node reached by the decoder. Notice that the threshold is increased only when a node is reached for the first time. Otherwise looping would occur.

† This chart is based on a chart suggested by Prof. I. M. Jacobs of M.I.T. It is equivalent to the flow chart of Ref. 2.

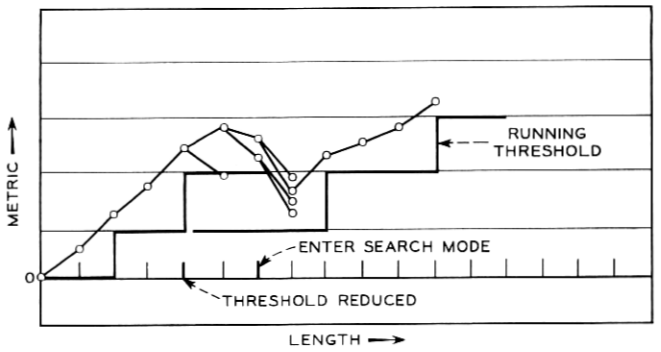

Fig. 5 — Threshold reduction, $b = 2$.

To further clarify the operations of the flow chart, we make the following observations: (*i*) Forward searching on a path whose path metric continues to grow is performed by operations A and D, (*ii*) the searching required after the searched path crosses the running threshold *from above* is performed by operations B, E, C, and D, (*iii*) the running threshold is reduced *only* if it is found that all paths cross this threshold from above (see Fig. 5). This last observation deserves expansion. When the decoder observes that the path under examination violates the running threshold, it looks back, one node at a time, to find a path which it might extend forward. If, after a number of backward and forward moves, the decoder decides that all paths examined violate the running threshold, it reduces the value of this threshold and repeats the search until a path is found which remains above the new lower value of the threshold. (If there is more than one such path, the decoder follows that path which has the "most probable" branches.) The decoder then continues to extend this path.

We now go on to discuss a particular buffer design and to examine the dynamics of the decoding operation. We shall return to the discussion of this section in a following section while discussing a random variable of computation.

2.4 *Dynamics of the Decoder*

A buffer designed to smooth the delay experienced by data arriving at the decoder is shown in Fig. 6. Data arrives from the left, is stored in sections corresponding to tree branches and progresses through the buffer at the rate at which it arrives. Storage is reserved below each branch for tentative source decisions. A safety zone is provided so that,

should a buffer overflow occur, data in this section will be declared unreliable and not released to the user.

Two pointers are shown on the buffer of Fig. 6. With these pointers the decoder operation may be traced. The "search" pointer locates the received branch currently being examined. The "extreme" pointer labels the latest branch ever examined. When the channel is relatively noise-free the two pointers hover at the left-hand side of the buffer. During a noisy interval, searching is required and the search pointer drifts to the right and away from the extreme pointer while the extreme pointer drifts to the right at the data rate. The two pointers become superimposed and move to the left after the noisy period has been passed. (We assume that the decoder has a computation rate which is twice or three times the average required computation rate.)

Should the channel experience a severely noisy period, the search pointer may drift to the far right-hand side of the buffer at which time the decoder will quite probably release an erroneous source decision to the safety zone. This spells trouble because thereafter the decoder searches on incorrect paths and is likely to do a large amount of continuous searching. Additional decoding errors will then be released to the user. This event we call buffer overflow.

Since buffer overflow can be detected by the location of the search pointer, the user can be so informed. However, no known techniques exist for retrieving the decoder from the overflow state once it has entered this state other than a restarting of the decoding process. This

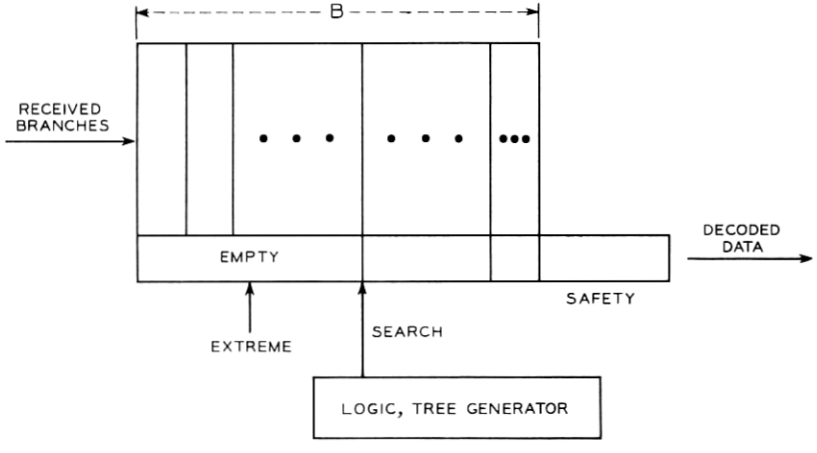

Fig. 6 — Buffer.

implies that either a feedback link has to be available or that periodic restarting is employed. Overflow, then, is a serious event. Since it can be argued the probability of an overflow is generally much larger (for a properly chosen code) than the probability of an error without overflow, it is, therefore, a most important consideration in the design of a sequential decoder.

In the next section we begin the analytical treatment of the overflow probability. Our intent is to indicate the dependence of this probability on the encoder and decoder parameters.

2.5 *Static Computation*

The overflow probability, $P_{BF}(N)$, is defined as the probability that the buffer overflows on or before the Nth source decision is released to the safety zone. It is this probability which is of primary concern in the design of the decoder. Unfortunately, both experimental[7] and analytical[3] investigations of $P_{BF}(N)$ have produced only estimates of this probability and these estimates depend upon a heuristic connection between $P_{BF}(N)$ and probabilities which have either been determined experimentally or bounded analytically. We shall be concerned with the analytical bounds and shall present an interpretation of these bounds.

Since $P_{BF}(N)$ is not amenable to direct analysis we shall be concerned with a random variable of computation which we call "static" computation. This is a computation associated with *a node of the correct path.* We assume that the decoder reaches a node of the correct path, say the gth, and we define static computation, C, as the number of computations required on the gth correct node and on all nodes on paths branching from this correct node except nodes on the correct path. This set of nodes is called the "incorrect subset" associated with the gth correct node. (See Fig. 1 where $g = 2$). A computation on a node is defined as a forward or backward "look" from a node (See the flow chart of Fig. 4).

The analytical results of this paper are concerned with bounds on the cumulative probability distribution of the random variable of static computation C, namely, $P[C \geqq L]$. We shall determine the behavior of $P[C \geqq L]$ with the distribution parameter L. Before we do so, however, we develop an upper bound to C to be used later in developing an upper bound to $P[C \geqq L]$. We begin by labeling nodes in the incorrect subset.

Each node in the gth incorrect subset can be labeled uniquely with a doublet (m, s). We take the index s as a measure of the "penetration"

of a node in the incorrect subset. We say a node has penetration s if it is separated from the correct node by s branches. The correct node itself is at penetration zero. The index m indicates the position of nodes at penetration s counting from the bottom of the incorrect subset (see Fig. 1 where node (3, 2) is shown). We define $M(s)$ as the number of nodes at penetration s and have $1 \leqq m \leqq M(s)$ where

$$M(s) = \begin{cases} 1 & s = 0 \\ (b-1)b^{s-1} & s > 0 \end{cases} \tag{4}$$

and b is the number of branches at a node. (Note that $M(0) = 1$ since the correct node has penetration zero.) Then, each node is uniquely labeled by a doublet (m,s).

To develop an upper bound to the random variable C we continue the discussion of Section 2.3. Assuming that we have reached the gth node of the correct path, defining D as the smallest value of the path metric on the remaining portion of the correct path and letting T_D be the threshold just below D (see Fig. 7), we see from observation (*iii*) of Section 2.3 that no threshold lying below T_D is ever used. A lower threshold would be required if all paths eventually crossed T_D but, by definition, at least one path, the correct path, remains completely above T_D.

Consider a particular incorrect node (m,s) with metric $d_0 + d^*(m,s)$, where d_0 is the value of the metric on the path terminated by the gth correct node and $d^*(m,s)$ is the remainder. If the thresholds are defined by $T_i = it_0$, $t_0 > 0$, $-\infty < i < \infty$, and if $d_0 + d^*(m,s)$ is separated from T_D by k such thresholds (including T_D), then the decoder can

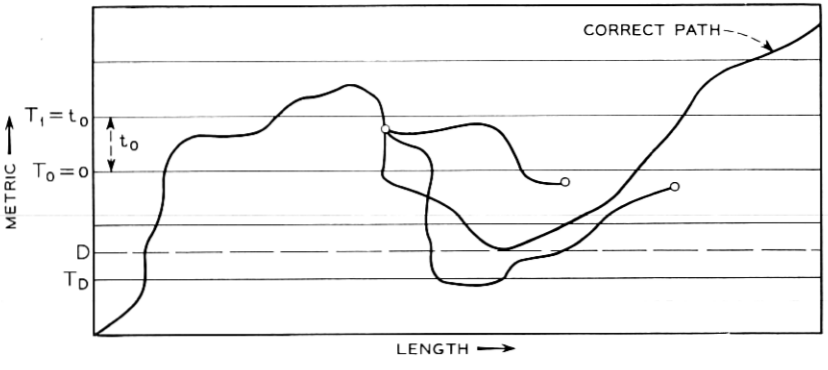

Fig. 7 — Typical path trajectories and the minimum threshold, T_D.

look at node (m,s) with *at most* k thresholds.† It can be shown that with each threshold no more than $(b + 1)$ computations can be performed at node (m,s), i.e., one backward look and b forward looks from (m,s). Since this is the case, we can define a random variable which counts the number of thresholds between the value of the metric on node (m,s) and threshold T_D and we can use this random variable to bound C. To simplify the analysis, however, we shall define a random variable $z_{i,s}(m)$ which allows us to *overbound* the number of thresholds between the value of the metric on node (m,s) and threshold T_D, including T_D. Represent the metric on the correct path of length $g + r$ by $d_0 + d_c(\bar{u}_r, \bar{v}_r)$ where $\bar{u}_r, \bar{v}_r$ are the portions of the correct and received paths extending beyond the gth node, respectively. Then, we have the following definition for $z_{i,s}(m)$:

$$z_{i,s}(m) = \begin{cases} 1 & d^*(m,s) \geqq T_{i-1}, \quad d_c(\bar{u}_r, \bar{v}_r) \leqq T_{i+1}, \\ & \text{some } r \geqq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

We now argue that $\sum_{i=-\infty}^{\infty} z_{i,s}(m)$ overbounds the number of thresholds between the value of the metric on node (m,s), $d_0 + d^*(m,s)$, and threshold T_D, including this threshold. Let d_0, the value of the path metric on the correct path up to and including the gth node, be between thresholds T and $T + t_0$. Then, if the metric on node (m,s) were $T + t_0$ instead of d_0, the number of thresholds with which (m,s) would be examined would be increased. Similarly, if the path metric on the correct path of length $g + r$, $d_0 + d_c(\bar{u}_r, \bar{v}_r)$, were replaced by $T + d_c(\bar{u}_r, v_r)$, the computation on node (m,s) would again be increased. These observations are used to define $z_{i,s}(m)$ in such a way that $\sum_{i=-\infty}^{\infty} z_{i,s}(m)$ overbounds the number of thresholds between (m,s) and T_D.

We have stated that no more than $(b + 1)$ computations are needed for each threshold lying between (m,s) and T_D. Then, we overbound the random variable of static computation on nodes of the gth incorrect subset, C, by

$$C \leqq (b + 1) \sum_{i=-\infty}^{\infty} \sum_{s=0}^{\infty} \sum_{m=1}^{\lceil M(s) \rceil} z_{i,s}(m) . \tag{6}$$

This bound will be used in overbounding $P[C \geqq L]$.

In Section III the analytical results will be delineated and interpreted and the upper bound to $P[C \geqq L]$ will be derived.

---

† This may be deduced from the flow chart of Fig. 4.

III. THE DISTRIBUTION OF STATIC COMPUTATION

Static computation has been defined as the computation performed in a particular incorrect subset of the tree code. A lower bound to the probability distribution of this random variable has been obtained, and is presented elsewhere.[3] An abbreviated derivation of an upper bound to this distribution will be presented in a following section. The essence of the lower bound argument is contained in the following section.

3.1 *Behavior of the Distribution*

It has been found[3,7] that the distribution of static computation, $P[C \geqq L]$ behaves as $L^{-\beta}$ for large L. We shall now present several simple intuitive arguments which explain this behavior.

If noise causes a large dip in the value of the correct path metric in the neighborhood of the gth correct node (see Fig. 7) then, the decoder will not be able to discriminate between the correct path and incorrect paths. Thus, much computation will be required. Since the number of paths in the incorrect subset grows exponentially with penetration into this subset, the number of computations required will grow roughly as an exponential in the length of the correct path dip or the duration of the interval of high channel noise. Hence, the static computation grows exponentially with an interval of high channel noise. On the other hand, an interval of high channel occurs on the DMC with a probability which decreases exponentially with its length. It is the balance between these two exponentials which is responsible for the behavior of the distribution of static computation. Random variables with distributions of this type are known as Paretian random variables and they appear in random walk problems,[6] in the distribution of incomes,[8] in error clustering on the telephone channels[15] and many other places.[14]

3.2 *Random Code Bound on the Distribution*

The technique used in this section to overbound the distribution of computation contains two major steps. In the first step, the distribution is bounded in terms of the moments of computation using a generalization of Chebysheff's Inequality. In the second step, the moments of computation are averaged over the ensemble of all tree codes. Together the two steps generate a random code bound to the distribution. This argument shows the existence of codes having a particular upper bound to their distribution function. The generalized Chebysheff Inequality is stated below.

Lemma 1: Let C be a positive random variable. Then,

$$P[C \geqq L] \leqq \overline{C^p}/L^p, \qquad p \geqq 0. \tag{7}$$

The "tightness" of this inequality is indicated by two examples. (*i*) For the discrete random variable which assumes values 0 and c_0 with probabilities $1 - a$ and a, respectively, the bound is exact when $L = c_0$. (*ii*) For the continuous random variable which assumes values greater than or equal to one with probability density $\beta/c^{\beta+1}$, the exact form of $P[C \geqq L]$ is $1/L^\beta$ and the bound is $\beta/(\beta - p)L^p$ for $p < \beta$. Therefore, as p approaches β the coefficient in the bound becomes indefinitely large while the exponent approaches the true exponent. Since the distribution of *static computation* is Paretian, this same behavior appears in the random code bound derived in this section.

The random variable of computation C has been overbounded by (6). It should be clear that moments of the bound on C will be difficult to evaluate due to the many crossterms. Much of the difficulty is avoided through the use of Minkowski's Inequality[9] which is stated below.

Lemma 2: Let $x_1, x_2, \cdots, x_n$ be a set of positive random variables. Then, for $p \geqq 1$ and for every n *we have*

$$\overline{\left(\sum_{i=1}^{n} x_i\right)^p}^{1/p} \leqq \sum_{i=1}^{n} \overline{x_i^p}^{1/p}. \tag{8}$$

Applying this inequality to the bound on C we have

$$\overline{C^p}^{1/p} \leqq (b+1) \sum_{i=0}^{\infty} \sum_{s=0}^{\infty} \overline{\left(\sum_{m=1}^{M(s)} z_{i,s}(m)\right)^p}^{1/p} + (b+1) \sum_{i=0}^{\infty} \sum_{s=0}^{\infty} \overline{\left(\sum_{m=1}^{M(s)} z_{-i,s}(m)\right)^p}^{1/p}. \tag{9}$$

In this form, moments are taken of the sum of the random variables $z_{i,s}(m)$, $1 \leqq m \leqq M(s)$, with both the threshold T_i and the penetration s fixed. (See the definition of $z_{i,s}(m)$ in (5).)

To further bound $\overline{C^p}$ we make the following expansion for integer values of p where the indices i and s are omitted:

$$\overline{\left(\sum_{m=1}^{M(s)} z(m)\right)^p} = \sum_{m_1=1}^{M(s)} \cdots \sum_{m_p=1}^{M(s)} \overline{z(m_1)z(m_2) \cdots z(m_p)}. \tag{10}$$

Since such an expansion does not hold for noninteger p, we limit our attention hereafter to integer p. We now proceed through several counting arguments to put (10) in a manageable form.

Since the random variable $z(m_1)\, z(m_2) \cdots z(m_p)$ assumes the value 1 only when all implied events occur simultaneously, the expectation in the right-hand side of (10) is the probability of the joint occurrence of all implied events. Now it can be seen for $p = 4$, say, that

$$z(5)z(1)z(16)z(5) = z(16)z(5)z(5)z(1) = z(1)z(5)z(16)$$

since $z(\cdot) = 1$ or 0 (so that $z(5)z(5) = z(5)$). Hence, $z(m_1) \cdots z(m_p)$ is independent of the order of the m_i and equals $z(\theta_1) \cdots z(\theta_t)$ where $\theta_1, \cdots, \theta_t$ are the *distinct* elements among $m_1, m_2, \cdots, m_p$. Consequently we can write*

$$\sum_{m_1=1}^{M(s)} \cdots \sum_{m_p=1}^{M(s)} \overline{z(m_1) \cdots z(m_p)} = \sum_{t=1}^{\mathrm{Min}(M(s),p)} \sum_{\substack{\text{All sets of } t \\ \text{distinct elements} \\ \{\theta_1, \cdots, \theta_t\}}} W(t,p)\, z(\theta_1) \cdots z(\theta_t) \tag{11}$$

where $W(t,p)$ is the number of p-tuples $(m_1, m_2, \cdots, m_p)$ which contain t distinct elements. We now bound $W(t,p)$.

$W(t,p)$ may be viewed as the number of ways of placing one ball in each of p distinguishable cells where the balls are of t different colors and each color must appear at least once. The number of such collections of p balls is less than the number of collections one would have if we include the situations where one or more colors do not appear. This larger number is the number of ways of placing t different elements in each of p distinguishable cells, or t^p. Therefore, $W(t,p) \leqq t^p$.

To underbound $W(t,p)$ we now establish that $W(t,p) \geqq tW(t,p-1)$. Consider $W(t,p-1)$, the number of ways $(p-1)$ balls of t different colors may be placed in $(p-1)$ distinguishable cells with no cell empty. Consider extending the collection by placing one additional ball with one of the t colors in a pth cell. This new collection contains $tW(t,p-1)$ items. It cannot contain more items than does the collection of $W(t,p)$ items because one color appears at least twice and every other color at least once, establishing the desired inequality. Iterating this inequality $(p-t)$ times and observing that $W(t,t) = t!$, we have $W(t,p) \geqq t^{p-t}t!$. The two bounds are summarized in the following Lemma:

Lemma 3: For $t \leqq p$ we have

$$\sqrt{2\pi t}\, e^{-t} t^p \leqq W(t,p) \leqq t^p \tag{12}$$

* The upper limit on t indicates that $(m_1, \cdots, m_p)$ contains no more than the smaller of $M(s)$ and p different elements.

Proof: We use the fact[6] that

$$t! \geqq t^t \sqrt{2\pi t}\, e^{-t}. \qquad \textit{Q.E.D.}$$

The two bounds indicate that $W(t,p)$ grows with t primarily as t^p for large p.

Before we proceed to the next counting argument we motivate our use of this next argument by presenting a result which is too long to be derived here.[3] The probability $\overline{z_{i,s}(\theta_1) \cdots z_{i,s}(\theta_t)}$ is the probability that nodes $(\theta_1, s), \cdots, (\theta_t, s)$, all at penetration s in the gth incorrect subset, simultaneously lie in metric above threshold $T_{i-1} = (i-1)\,t_0$ while the correct path falls below T_{i+1} somewhere following the gth correct node (see (5)). An overbound to this probability is given below. The average of the product of the z's is taken over the ensemble of channel transitions and the *set of all tree codes*. It is at this point that the random code technique is used.

$$\begin{aligned}\overline{z_{i,s}(\theta_1) \cdots z_{i,s}(\theta_t)} \leqq \{2^{t_0[t/(1+t)-\sigma_0]}\, 2^{-it_0[t/(1+t)+\sigma_0]} \\ \cdot 2^{[s\,lt/(1+t)](R_t-R)}\} \cdot \{2^{-\alpha l R_t}\} \left\{\sum_{r_0=1}^{\infty} 2^{-r_0 l[\sigma_0 R-\mu_t(\sigma_0)]}\right\}.\end{aligned} \tag{13}$$

Here α *is the number of branches on the paths terminated by nodes* (θ_1, s), $\cdots$, (θ_t, s), exclusive of branches preceding the gth correct node, and

$$\mu_t(\sigma_0) \triangleq \frac{1}{1+t} \log_2 \sum_{j=1}^{J} f(y_j) \left(\sum_{k=1}^{K} p_k \left[\frac{P(y_j \mid x_k)}{f(y_j)}\right]^{1+\sigma_0}\right)^{1+t} \tag{14}$$

$$f(y_j) \triangleq \sum_{k=1}^{K} p_k P(y_j \mid x_k)$$

where $\sigma_0 \leqq 0$. Also,

$$R_t \triangleq -\frac{1}{t} \log_2 \sum_{j=1}^{J} \left(\sum_{k=1}^{K} p_k P(y_j \mid x_k)^{1/1+t}\right)^{1+t}. \tag{15}$$

The probability assignment $\{p_k\}$ is the assignment given to digits in a code when using the *random code argument*. In the bound of (13) there exists a value of σ_0, $-(t)/(1+t) < \sigma_0 \leqq 0$, for which the sum on r_0 converges, as long as $R < R_t$. Examination of (13) will show that the bound depends on the paths terminated by nodes (θ_1, s), (θ_2, s), $\cdots$, (θ_t, s) only through α, the number of branches which they contain, exclusive of branches preceding the gth correct node. (For example, see Fig. 8 where a set of paths is indicated with checks and the branches which they contain are labeled with 1.) This being the case, we must

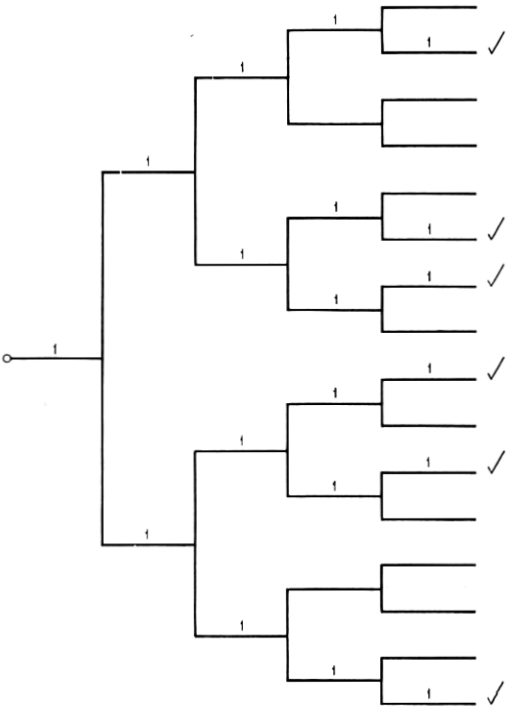

Fig. 8 — Topology of tree paths.

then group together in (13) all paths having the same number, α, of branches. Call this number of paths $N_t(\alpha)$. The following lemma provides a bound on $N_t(\alpha)$.

Lemma 4:
$$N_t(\alpha) \leqq (t-1)!(s-1)^{t-2}2^{\alpha l R}. \tag{16}$$

Proof: The proof is by construction. We first show that

$$N_t(\alpha) \leqq (t-1)!s^{t-2}2^{\alpha l R} \quad \text{for} \quad s \geqq 1.$$

Consider placing the t paths in the tree, one by one. The first of the t paths placed in the incorrect subset of the tree (containing $M(s) \leqq b^s$ paths) may assume no more than b^s positions. A second path connecting with the first, but having d_1 separate branches, may assume any one of b^{d_1} positions since its point of connection to the first path is fixed by its length d_1. A third path with d_2 branches distinct from the first two may connect to either path and terminate in b^{d_2} positions, that is, it can assume no more than $2\ b^{d_2}$ places. The tth path having

d_{t-1} branches may terminate in any one of $b^{d_{t-1}}$ positions; hence, can be situated in no more than $(t-1)\,b^{d_{t-1}}$ places. Thus, given that the second path has d_1 branches distinct from the first, that the third path has d_2 branches distinct from the first and second, etc., the number of arrangements of the t paths cannot exceed $(t-1)!b^{\alpha}$ where

$$\alpha = s + d_1 + \cdots + d_{t-1},$$

the number of branches on these paths. All that remains is to determine the number of ways that values may be assigned to $d_1, d_2, \cdots, d_{t-2}$. (Note that d_{t-1} is fixed given α and $d_1, d_2, \cdots, d_{t-2}$.) Since each number d_i represents a portion of a path, $1 \leqq d_i \leqq s$, we may assign values to $d_1, d_2, \cdots, d_{t-2}$ in no more than s^{t-2} ways. Hence, the number of arrangements of t paths containing α branches cannot exceed $(t-1)!s^{t-2}b^{\alpha}$. Observing that $b = 2^{lR}$ by (1) we have the desired result for $s \geqq 1$. We also have $s \leqq \alpha \leqq st$ since one path contains s branches and the number of branches on all paths cannot exceed st. Now, when $s = 0$, the bound on $N_t(\alpha)$ is zero. We must have $N_t(\alpha) = 1$ when $s = 0$, so that we replace s by $(s+1)$. *Q.E.D.*

Combining all terms we have the following random code bound on the moments of static computation:

$$\begin{aligned}\overline{C^p}^{1/p} \leqq & \left[\sum_{s=0}^{\infty} (s+1)^{1-1/p}\, 2\exp\left(\frac{-sl(R_p-R)}{p(1+p)}\right)\right](2^{t_0(1-\sigma)}pp!)^{1/p} \\ & \times \left\{\left(\sum_{r=1}^{\infty} 2\exp\{-rl[\sigma R - \mu_p(\sigma)]\}\right)^{1/p}\left[\sum_{i=0}^{\infty} 2\right.\right. \\ & \left.\exp\left(-\frac{it_0(\tfrac{1}{2}+\sigma)}{p}\right)\right] \\ & + \left(\sum_{r=1}^{\infty} 2\exp\{-rl[\sigma' R - \mu_p(\sigma')]\}\right)^{1/p}\left[\sum_{i=0}^{\infty} 2\right. \\ & \left.\left.\exp\left(+\frac{it_0(p/(1+p)+\sigma')}{p}\right)\right]\right\}. \end{aligned} \tag{17}$$

It can be shown[3] that σ and σ' can be chosen for $R < R_p$ such that $\sigma > -\frac{1}{2}$, $\sigma' < -(p)/(1+p)$, $\sigma R - \mu_p(\sigma) > 0$ and $\sigma' R - \mu_p(\sigma') > 0$, that is, such that the bound converges for $R < R_p$.

We now have enough results available to draw the central conclusion concerning moments of the random variable of static computation, C. For integer p, $\overline{C^p}$, as an average over the ensemble of all tree codes, is finite for source rates R strictly less than the rate R_p (given by (15)). Since it can be shown that $R_1 \geqq R_2 \geqq R_3 \geqq \cdots \geqq 0$, we have for any

given rate R that moments $\bar{C}$, $\overline{C^2}$, $\cdots$, $\overline{C^k}$ are bounded where k is the largest integer such that $R < R_k$. We cannot determine from this bounding argument whether moments $\overline{C^{k+1}}$, $\overline{C^{k+2}}$, etc. are finite or not.

Returning to Chebysheff's Inequality we overbound the cumulative probability distribution of the random variable of static computation, $P[C \geqq L]$, using the bounds on the moments.

Theorem 1: With a probability of at least 0.9, a tree code drawn from the ensemble of tree codes will have a distribution of static computation, $P[C \geqq L]$, which is bounded by $10\overline{C^k}/L^k$ where k is the largest integer such that $R \leqq R_k$, R is the source rate, R_k is given by

$$R_k = -\frac{1}{k}\log_2 \sum_{j=1}^{J}\left(\sum_{k=1}^{K} p_k P(y_j \mid x_k)^{1/1+k}\right)^{1+k} \tag{18}$$

and $\overline{C^k}$ is the random code bound on the moments of static computation. For any larger k a finite bound on $\overline{C^k}$ is not known.

Proof: Over the ensemble of tree codes $P[C \geqq L]$ is a random variable. Then, if we let x represent $P[C \geqq L]$, we have

$$\bar{x} \geqq \sum_{x \geqq 10\bar{x}} x p(x) \geqq 10\bar{x} P[x \geqq 10\bar{x}]$$

from which we have that $P[x < 10\bar{x}] \geqq 0.9$.

Q.E.D.

This theorem summarizes the major result of this section which is that the distribution of computation decreases as fast as L^{-k} where k is the largest integer such that $R < R_k$. We note that $\overline{C^k}$ becomes indefinitely large as R approaches R_k. This was predicted by the discussion which followed the introduction of Chebysheff's Inequality.

In the next section we interpret the bounds on the distribution of static computation and relate these bounds to the probability of a buffer overflow.

IV. STATIC COMPUTATION AND THE OVERFLOW PROBABILITY

The upper bound to the distribution of computation given above and a lower bound presented elsewhere[3] both are algebraic functions of the distribution parameter, that is, $P[C \geqq L]$ behaves as $L^{-\beta}$, $\beta > 0$, for large L. In the following sections we define a quantity called the "computation exponent" which extracts from $P[C \geqq L]$ its behavior with L. The computation exponent is compared to known exponents on the probability of error and with some experimental data. Also, a heuristic connection between the overflow probability and the distribution of

static computation is established. We begin with a discussion of the computation exponent.

4.1 *The Computation Exponent*

The "computation exponent," $e(R)$, as defined below, is a measure of the tail behavior of $P[C \geqq L]$, that is, its behavior with L for large L.

$$e(R) \triangleq R \left\{ \lim_{L \to \infty} \frac{-\log P[C \geqq L]}{\log L} \right\}. \tag{19}$$

If $P[C \geqq L]$ behaves as $L^{-\beta}$ for large L, then $\beta = e(R)/R$. We consider the computation exponent $e(R)$ rather than β because $e(R)$ is a bounded function while β is not. We now state bounds that have been obtained on $e(R)$ by over- and under-bounding $P[C \geqq L]$. We note that a channel is "completely connected" if all of its transition probabilities are nonzero, i.e., all output symbols can be "reached" from every input symbol, the normal physical situation.

Theorem 2: Codes do not exist for the completely connected DMC which have a computation exponent greater than $\bar{e}(R)$ *where*[3]

$$\bar{e}(R) \triangleq (-\sigma)(R - I_{\min}) \tag{20}$$

and σ, $-1 \leqq \sigma \leqq 0$, *is the solution to*

$$R = \max_k \frac{\gamma_k(\sigma)}{\sigma}. \tag{21}$$

Here, $\gamma_k(\sigma)$ *is given by*

$$\begin{aligned} \gamma_k(\sigma) &\triangleq \log_2 \sum_{j=1}^{J} P(y_j \mid x_k)^{1+\sigma} f(y_j)^{-\sigma} \\ I_{\min} &\triangleq \min_{j,k} \log_2 \frac{P(y_j \mid x_k)}{f(y_j)} \end{aligned} \tag{22}$$

and

$$f(y_j) = \sum_{k=1}^{K} p_k P\left(\frac{y_j}{x_k}\right).$$

Theorem 3: On the DMC, codes exist which have a computation exponent greater than or equal to $\underline{e}(R)$ *where*

$$\underline{e}(R) = pR \tag{23}$$

and $p = 1, 2, 3, \cdots$, *is found from* $R_{p+1} \leqq R < R_p$. R_p *is given by* (18).

The probability assignment $\{p_k\}$ appears in the random code argument and in the definition of the metric through the function $f(y_j)$. Although this will not be done here, one can choose $\{p_k\}$ to maximize $\underline{e}(R)$.

As an example, the two bounds are sketched in Fig. 9 for the Binary Symmetric Channel (BSC) with crossover probability of $p_0 = 0.01$. In that figure we have chosen $p_k = \frac{1}{2}$, $k = 1, 2$. For this probability assignment $\bar{e}(R)$ is zero for R greater than or equal to channel capacity. For other assignments $\bar{e}(R)$ may intercept the rate axis at a rate which exceeds channel capacity.

4.2 *An Experimental Result, A Conjecture and An Interpretation*

Recently a computer simulation was made of the Fano algorithm for a number of channels including the BSC. This simulation study[7] was performed at the M.I.T. Lincoln Laboratory under the direction of K. L. Jordan. Mr. Jordan has generously provided the author with data from a particular simulation of a BSC with crossover probability of $p_0 = 0.01$.

In this simulation, a convolutional tree code of the type described in Section 2.1 with $b = 2$ was used (hence; source rates of $R = 1/l$, l an integer were available). The generator for this tree code was optimized in the manner found in Ref. 1.

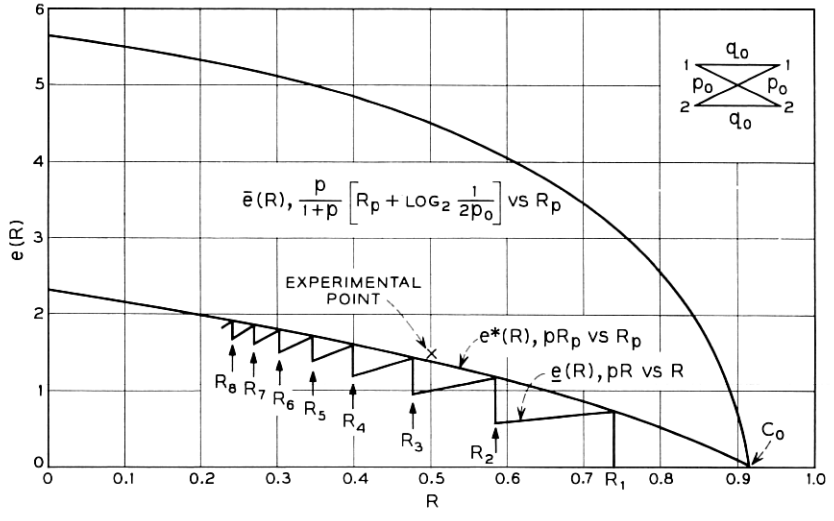

Fig. 9 — Computation exponent bounds for the BSC with $p_0 = 0.01$.

An empirical distribution of a particular random variable of computation was measured. This random variable differs from static computation somewhat but one can argue heuristically that it is within a small multiple of the random variable of static computation when either random variable is large. The random variable measured in the simulation is the number of computations required to advance one node into the tree. For example, when the channel is not noisy, a forward "look" will indicate that a forward move is possible and only one computation will be necessary; however, if the channel is noisy, the decoder may have to do much backward searching before a path is found upon which the point of deepest penetration into the tree can be increased.

The empirical distribution of computation is shown in Fig. 10 for $R = \frac{1}{2}$. The corresponding computation exponent for this rate is shown in Fig. 9. The data from which the empirical distribution was determined represents the transmission of over one million channel digits. Although data at rates $R = \frac{1}{3}, \frac{1}{4}$ was available, it was not deemed reliable and not used because few cases of large computation occurred.

The experimental computation exponent and the derivation of the lower bound to $e(R)$, namely $\underline{e}(R)$, leads one to conjecture a "true" value for the computation exponent.

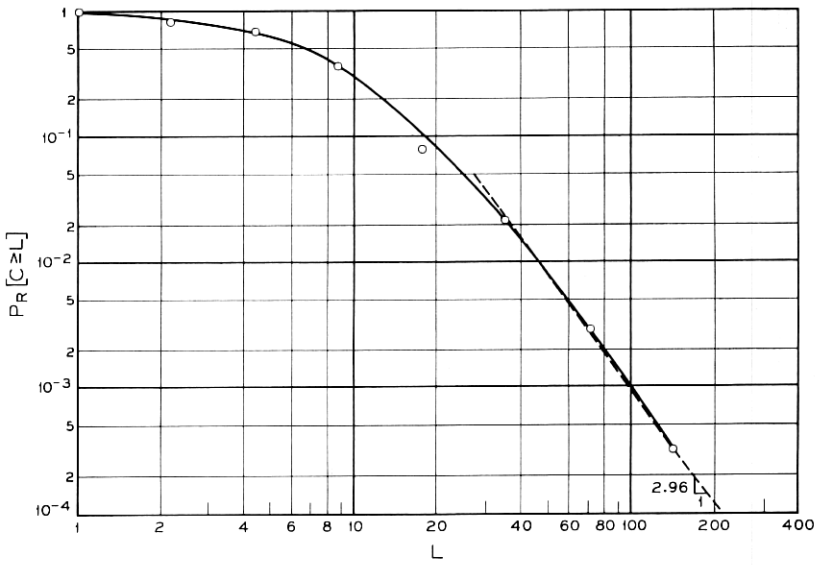

Fig. 10 — Empirical distribution of computation.

Conjecture:† For the metric used in this paper the computation exponent for the Fano algorithm cannot exceed $e^*(R)$ and there exists codes with this computation exponent where $e^*(R)$ is given by

$$e^*(R) = pR_p \quad \text{for} \quad R = R_p \tag{24}$$

Here p assumes all nonnegative, real values and R_p is given by (18). Here we use that probability assignment $\{p_k\}$, for each p that maximizes R_p.

The conjectured exponent for the BSC is shown in Fig. 9. The experimental point and $e^*(R)$ at $R = \frac{1}{2}$ differ by 5 percent, an excellent match.

The conjectured exponent has an interesting interpretation in terms of the probability of error with "List Decoding."[10,11] With list decoding, the decoder makes a list of the k *a posteriori* most probable codewords. If this list does not contain the transmitted codeword, an error is said to occur. Random code bounds have been obtained on this probability of error. This probability of error has an exponent which we call $E_k(R)$ for list size k (see Fig. 11). $E_k(R)$ may be found from an exponent $E_\infty(R)$ by $E_k(R) = E_\infty(R)$ for $R_k^* \leqq R$ where R_k^* is the rate at which $E_\infty(R)$ has slope $-k$ and for $R < R_k^*$, $E_k(R)$ is the tangent to $E_\infty(R)$ at $R = R_k^*$. The rate-axis intercept of this straight line is R_k.

The "list decoding exponent", $E_\infty(R)$, depends on the random code assignment probabilities $\{p_k\}$. If that set $\{p_k\}$ is chosen for each rate which maximizes $E_\infty(R)$, we have the "sphere-packing"[13] exponent. (See Fig. 11.) This is an exponent on the probability of a block decoding error which cannot be exceeded by any block code with any decoding algorithm even when a feedback channel is available. Thus, the "list decoding exponent" and the "sphere-packing exponent" are fundamental.

The conjectured computation exponent, $e^*(R)$, can now be related to $E_\infty(R)$. To find $e^*(R)$ draw a line from R on the rate-axis which is tangent to $E_\infty(R)$. (See Fig. 12.) This line intersects the exponent axis. The rate-axis intercept and the exponent-axis intercept define a point on $e^*(R)$. Using this construction procedure every point on $e^*(R)$ may

† Note added in the preparation of this paper: Recently I. M. Jacobs and E. Berlekamp have underbounded the probability of buffer overflow or undetected error using lower bounds to the probability of error with list decoding. They have found that this bound has a computation exponent which agrees with the conjectured exponent given above and have shown that this bound grows linearly with the number of source digits processed by the decoder before overflow. Also, H. Yudkin has recently upper bounded the moments of static computation for integer and noninteger moments. The computation exponent implied by these bounds establishes the conjecture.

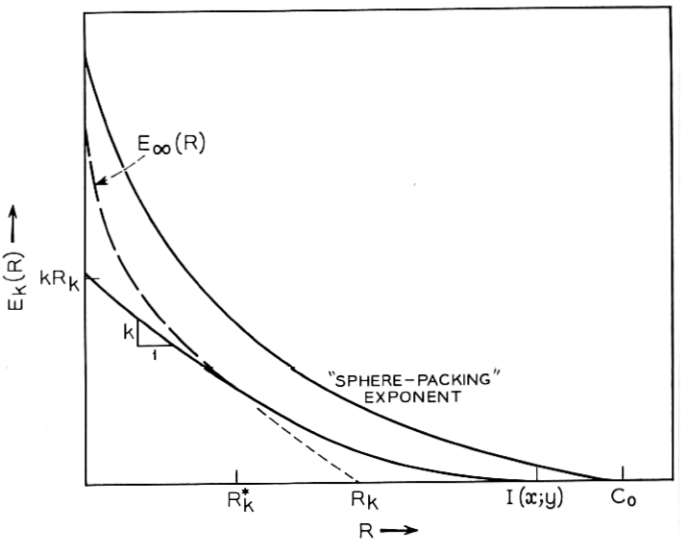

Fig. 11 — List decoding exponent.

be generated. Hence, $e^*(R)$ *may be obtained by a simple and natural construction from* $E_\infty(R)$.

4.3 *Heuristic Connection With The Overflow Probability*

In this section we establish a heuristic connection between $P[C \geqq L]$ and the probability of a buffer overflow $P_{BF}(N)$, the probability of an overflow before the Nth source decision is released to the safety zone.

We begin by noting that $P_{BF}(N)$ is monotone increasing in N, hence that $P_{BF}(N) \geqq P_{BF}(1)$. We first relate $P_{BF}(1)$ and $P_R[C \geqq L]$.

Referring to Fig. 7, an example of a correct path trajectory which causes large static computation, we develop the following argument. If the random variable of static computation, C, is large, most of the computation will be performed on nodes which are close to the reference node. For computation to be performed on nodes distant from the reference node, the correct path must dip sufficiently at some distant point so that it returns at least to the level of the reference node. (Incorrect paths typically decrease in metric.) Since the correct path increases on the average, such a dip must be very large and thus occurs with much smaller probability than do dips close to the reference node.

If most of the static computation is done on nodes close to the reference node, we may associate such computation with dips in the correct

path. Then, since all incorrect subsets in the neighborhood of a path dip will have approximately equal amounts of computation done in them, the total computation due to a correct path dip will be a small multiple of the static computation in a particular incorrect subset in the neighborhood of the dip. Consequently, if about N_{av} incorrect subsets have equal computation in them, say C_0, and $N_{av}C_0$ is enough computation to cause overflow on the first decoded digit, then, heuristically, $P_{BF}(1) \cong P[C \geqq C_0]$. To find C_0, assume that the buffer can store B tree branches of l digits each, that each channel digit arrives in τ_{ch} seconds, and that the decoder can perform one computation in τ_0 seconds. Then, the search pointer will be forced back to the safety zone if more than $lB\tau_{ch}/\tau_0$ computations are needed to decode the first digit. Setting $N_{av}C_0 = lB\tau_{ch}/\tau_0$ we have

$$P_{BF}(1) \cong P[C \geqq lB\tau_{ch}/N_{av}\tau_0] \tag{25}$$

as our heuristic approximation.

If dips in the correct path are infrequent and if the decoder operates at about twice or three times the speed required to do the average computation in real time, then we may approximate $P_{BF}(N)$ by assuming independence of the dips which cause overflow and have

$$P_{BF}(N) \cong NP_{BF}(1) \tag{26}$$

This last argument is weak and should, at best, serve as a rule of thumb.

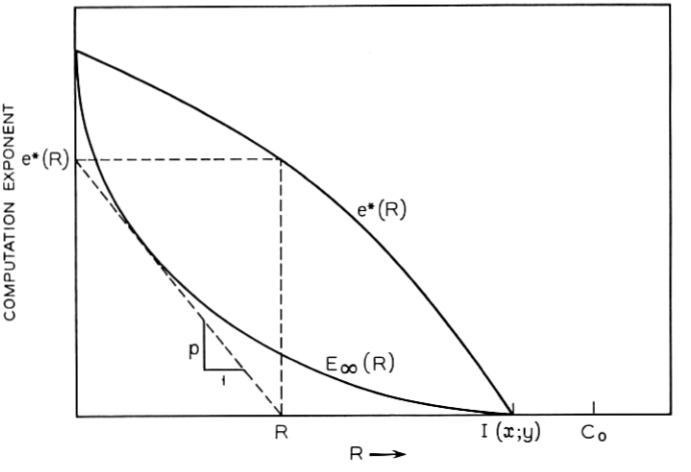

Fig. 12 — Construction for $e^*(R)$.

The argument leading to the connection between $P_{BF}(1)$ and the distribution of static computation is stronger and is partially supported by the experimental evidence cited above.

From (24), (25), (26), and the fact that the distribution of static computation is Paretian, we deduce that the overflow probability, *when it is small*, behaves as $N[N_{av}\tau_0/lB\tau_{ch}]^{\beta}$ where β is related to the computation exponent $e(R)$ by $\beta(R) = e(R)/R$. Thus, it is clear that the overflow probability is relatively insensitive to the buffer size B and the machine speed $1/\tau_0$ but that it depends heavily on the source rate R. (Note that changing the duration of the channel symbols, τ_{ch}, is tantamount to changing the channel and thus $e(R)$.) Since $e(R)$ increases with decreasing rate, we deduce that $\beta(R)$ is more than doubled by a halving of the information rate or that the overflow probability is more than squared.

V. CONCLUSIONS

It has been said that the buffer overflow probability is of primary concern in the design of a sequential decoder. We have examined this buffer overflow probability and have shown that it is relatively insensitive to the buffer capacity and to machine speed for moderate speeds and capacities. We have also indicated that the overflow probability is a strong function of the source rate. In addition, bounds on the dependence of the overflow probability on the source rate were given and related to exponents presented in the coding theorem.

We have argued that the particular sensitivites of the overflow probability exist because the distribution of static computation is an algebraic function of the distribution parameter; i.e., $P[C \geqq L]$ behaves as $L^{-\beta}$, $\beta > 0$, for large L. In turn, it has been observed that such behavior arises because the random variable of static computation assumes exponentially large values with exponentially small probabilities. This exponential growth of computation has been shown to be basic to sequential decoding.[3]

While the probability of overflow is relatively insensitive to many of the machine parameters, these parameters can be so chosen and the source rate can be so restricted that the probability of a buffer overflow can be made very small. To achieve the small overflow probabilities, the source rate for many channels need not be restricted to be less than about 90 percent of R_1.[7] (R_1 is generally known as R_{comp}, the computational cutoff rate defined by Wozencraft.[1] For many channels, R_{comp} is a substantial fraction of channel capacity.)

VI. ACKNOWLEDGMENTS

The author wishes to acknowledge the many helpful conversations during the course of this work with his thesis supervisor Prof. I. M. Jacobs, and his advisors, Profs. C. E. Shannon, J. M. Wozencraft, and R. G. Gallager.

REFERENCES

1. Wozencraft, J. M. and Reiffen, B., *Sequential Decoding,* M.I.T. Press and John Wiley and Sons, Inc., New York, London, 1961.
2. Fano, R. M., A Heuristic Discussion of Probabilistic Decoding, IEEE Trans., *IT-9,* 1963, pp. 64–74.
3. Savage, J. E., The Computation Problem with Sequential Decoding, Ph. D. Thesis, Department of Electrical Engineering, M.I.T., February, 1965, also published as Technical Report No. 371, M.I.T., Lincoln Laboratory, Lexington, Mass., 1965, and as Technical Report No. 439, M.I.T., Research Laboratory of Electronics, Cambridge, Mass., 1965.
4. Shannon, C. E. and Weaver, W., *Mathematical Theory of Communication,* U. of Illinois Press, Urbana, Ill., 1949.
5. Gallager, R. G., A Simple Derivation of the Coding Theorem and Some Applications, IEEE Trans., *IT-11,* 1965, pp. 3–18.
6. Feller, W., *An Introduction to Probability Theory and Its Applications,* 2nd Ed., John Wiley and Sons, Inc., New York, 1957, Chap. 2.
7. Jordan, K. L., Jr., Performance of Sequential Decoding with Efficient Modulation, to be published in IEEE Trans. of the Communication Systems Group.
8. Mandelbrot, B. M., The Pareto-Levy Law and the Distributions of Income, Int. Econ. Rev., *1,* May, 1960.
9. Hardy, G. H., Littlewood, J. E., and Polya, G., *Inequalities,* University Press, Cambridge, England, 1959.
10. Wozencraft, J. M., List Decoding, Quarterly Progress Report 48, Research Laboratory of Electronics, M.I.T., January 15, 1958.
11. Elias, P., List Decoding for Noisy Channels, Technical Report 335, Research Laboratory of Electronics, September 20, 1957.
12. Gallager, R. G., unpublished notes.
13. Fano, R. M., *Transmission of Information,* M.I.T. Press and John Wiley and Sons, New York, 1961.
14. Zipl, G. K., *Human Behavior and the Principle of Least Effort,* Addison-Wesley Press, Reading, Mass., 1949.
15. Mandelbrot, B., Self-Similar Error Clusters in Communication Systems and the Concept of Conditional Stationarity, IEEE Trans., *COM-13,* March, 1965.
16. Wozencraft, J. M. and Jacobs, I. M., *Principles of Communication Engineering,* John Wiley and Sons, Inc., New York, 1965.