[Preprint of paper to be presented at the NATO Advanced Study Institute, "The Impact of Processing Techniques on Communications", Château de Bonas, France, July 11 - 22, 1983 and to appear in the Proceedings of this meeting to be published by Martinus Nijhoff Publishers, The Hague, and edited by J.K. Skwirzynski.]

AN INFORMATION-THEORETIC APPROACH TO ALGORITHMS

James L. Massey

Professor for Digital Systems Engineering
Institute of Telecommunications
Swiss Federal Institute of Technology
Zurich, Switzerland

ABSTRACT

A quite general result, called the Leaf Entropy Theorem, is proved. This theorem gives the relation between the entropy of the leaves and the branching entropies of the nodes in a rooted tree with probabilities. It is shown that this theorem can be used to bound the average number of essential tests performed by a testing algorithm for a discrete random variable. The approach is illustrated by two examples, namely, the sorting of a list and the polling of the stations in a multiple-access communications system.

1. INTRODUCTION

This paper aims to demonstrate that information theory can be a useful tool in the analysis and design of a rather large class of algorithms that are used in communications and elsewhere. Analogous to the manner in which information theory provides both upper and lower bounds on the average data rate of reliable communications systems, so can it also provide upper and lower bounds on the average "computation" (appropriately defined) of algorithms that can perform certain tasks. This presumes, of course, that one can place a probability distribution over the possible values of the "data" that the algorithm must process -- but, as we shall see, such a probability distribution is often suggested by the nature of the problem. The utility of this information-theoretic approach depends both on the reasonableness of the incorporated definition of computation as well as on the appropriateness of an average (rather than worst-case) computational criterion.

The techniques used in this paper are a generalization and extension of those in [1]. The techniques in [1] have also been generalized in a different way by Hartmann et al. [2] to whose paper the reader is referred for an up-to-date bibliography of other work in this area. The approach given here differs from that in [2] principally in the fact that the latter admits a more general complexity measure but requires more detailed information about the algorithm(s) being analyzed. Here, we seek to make useful statements about the average computation of algorithms after only a superficial analysis of their properties.

In the next section, we introduce the terminology and notation for rooted trees with probabilities and prove the Path Length Lemma. In Section 3, we prove the Leaf Entropy Theorem for such trees, which forms the basis of our information-theoretic approach to algorithms. Section 4 introduces the notion of a testing algorithm. Section 5 shows how to use the Leaf Entropy Theorem to obtain upper and lower bounds on the average computation of testing algorithms. Section 6 provides an example of the application of these bounds to a polling problem. Finally, in Section 7, we discuss the capabilities and limitations of this approach to algorithms.

2. ROOTED TREES WITH PROBABILITIES

A tree is a connected undirected graph with no loops. A rooted tree is a tree in which one of the vertices is designated as the root. (In our figures, we will designate the root by attaching the symbol for an electrical ground to that vertex.) If two vertices are adjacent in a rooted tree, the one farther from the root is called a child of the other, which in turn is called the parent of that child. Note that a child has only one parent and that the root is the only vertex with no parent. A vertex with no children is called a leaf. The depth of a leaf is the number of branches on the path from the root to that leaf; the other vertices on this path are called the ancestors of this leaf. Notice that the depth of a leaf equals the number of its ancestors. The non-leaf vertices (including the root) are called nodes.

By a rooted tree with probabilities, we mean a rooted tree to which a nonnegative real number has been assigned to each leaf (and which is called the probability of that leaf) such that the sum over all leaves of these numbers is 1. We then assign to each node a probability equal to the sum of the probabilities of all leaves having this node as an ancestor. It follows that the root has probability 1 and that the probability of a node equals the sum of the probabilities of its children.

In a rooted tree with probabilities, we shall write $p_1, p_2, \ldots, p_L$ for the probabilities of the L leaves and shall write $P_1, P_2, \ldots, P_N$ for the probabilities of the N nodes; we do not exclude the case $L = N = \infty$ when the number of leaves and nodes is countably infinite, but we do assume that there is a finite maximum for the number of children with the same parent. We shall always suppose that the first node is the root so that $P_1 = 1$.

The following simple result, which is (more or less) well known, will be extremely useful in the sequel.

Path Length Lemma: The average depth $E[W]$ of the leaves of a rooted tree with probabilities equals the sum of the node probabilities, i.e.,

$$\sum_{i=1}^{L} p_i w_i = \sum_{j=1}^{N} P_j,$$

where w_i is the depth of the i-th leaf.

Proof: Recall that w_i equals the number of nodes that are ancestors of the i-th leaf. Hence, in the sum of the node probabilities where each term is expressed as the sum of the probabilities of those leaves having that node as an ancestor, the i-th leaf will contribute its probability p_i exactly w_i terms. This proves the lemma.

It is illuminating to interpret the leaf probability p_i as the probability that a one-way journey from the root will terminate on the i-th leaf. Under this interpretation, the node probability P_j is the probability that the j-th node will be visited on the journey and $E[W]$ is the average length of the journey in branches. [To avoid annoying trivialities, we assume hereafter that all nodes (but not necessarily all leaves) have non-zero probability; were this not so, we could prune the tree at the nodes of zero probability to produce an equivalent (for our purposes) tree with all nodes having non-zero probability.] Let q_{jk} denote the probability of that vertex which is the k-th child of node j. [We have chosen q here rather than p or P to denote the vertex probability since this vertex could be either a leaf or another node.] It follows that

$$\sum_{k} q_{jk} = P_j. \tag{1}$$

It follows also that

$$q_{k|j} = q_{jk} / p_j \tag{2}$$

is the probability that the k-th child of node j will be visited on the journey, given that node j is visited.

3. ENTROPIES IN ROOTED TREES WITH PROBABILITIES

In the above interpretation of a rooted tree with probabilities, the leaf entropy, defined as

$$H_{LEAF} = - \sum_{i=1}^{L} p_j \log p_j \ , \tag{3}$$

is just the uncertainty about where the journey will terminate or, equivalently, about which path from the root to a leaf will be followed. [For examples, we shall always use base 2 for our logarithms so that entropies will be in "bits".] The branching entropy at node j, defined as

$$H_j = - \sum_k q_{k|j} \log q_{k|j} \ , \tag{4}$$

is just the uncertainty about which branch will be chosen next, given that the journey has reached node j.

Example 1: For the rooted tree with probabilities of Fig. 1 (where here and hereafter nodes are indicated by squares and leaves by circles), we have $q_{1|1} = q_{2|1} = 1/2$ so that, for the root node, (4) gives

$$H_1 = 1 \text{ bit.}$$

Similarly, we have $q_{1|2} = 1/2$, $q_{2|2} = 1/4$ and $q_{3|2} = 1/4$ so that

$$H_2 = 3/2 \text{ bits.}$$

Because $p_1 = 1/2$, $p_2 = 1/4$, and $p_3 = p_4 = 1/8$, (3) gives

$$H_{LEAF} = 7/4 \text{ bits.}$$

Because $P_1 = 1$ and $P_2 = 1/2$, the path length lemma gives

$$E[W] = P_1 + P_2 = 3/2.$$

This agrees, of course, with the direct calculation of $E[W]$ which, because $w_1 = 1$, $w_2 = w_3 = w_4 = 2$, yields

$$E[W] = (\tfrac{1}{2})1 + (\tfrac{1}{4})2 + (\tfrac{1}{8})2 + (\tfrac{1}{8})2 = 3/2.$$

It follows from (2) and from our definition (4) of the branching entropy at node j that

$$P_j H_j = -\sum_k q_{jk} \log (q_{jk}/P_j)$$

$$= P_j \log P_j - \sum_k q_{jk} \log q_{jk}. \qquad (5)$$

Summing over j now gives

$$\sum_{j=1}^{N} P_j H_j = \sum_{j=1}^{N} P_j \log P_j - \sum_{n=1}^{N} \sum_k q_{nk} \log q_{nk} \qquad (6)$$

where our reason for changing the last index of summation will soon be apparent. We now note that every vertex, <u>except the root</u>, will be the k-th child of the n-th node for exactly one pair (n,k). Thus,

$$\sum_{n=1}^{N} \sum_k q_{nk} \log q_{nk} = \sum_{j=2}^{N} P_j \log P_j + \sum_{i=1}^{L} p_i \log p_i. \qquad (7)$$

Using (7) in (6) and noting that $P_1 \log P_1 = 1 \log 1 = 0$, we obtain the following result.

<u>LEAF ENTROPY THEOREM</u>: The leaf entropy, H_{LEAF}, in a rooted tree with probabilities equals the sum of the node branching entropies weighted by the node probabilities, i.e.,

$$-\sum_{i=1}^{L} p_i \log p_i = \sum_{j=1}^{N} P_j H_j.$$

Example 1 (concluded):

$$\sum_{j=1}^{2} P_j H_j = (1)1 + (\tfrac{1}{2})\tfrac{3}{2} = \tfrac{7}{4} \text{ bits}$$

which, of course, agrees with our previous direct calculation of H_{LEAF}.

The above theorem, which is the main result of this paper, is a reformulation and minor generalization of previous work [1]. The generalization consists in removal of the previous restriction that the rooted tree be a so-called D-ary tree, i.e., that exactly D branches stem from each node in the direction away from the root. The reformulation in tree rather than sequence terminology suggested the above proof, which is a major simplification of the previous proof.

In the sequel, we shall use the above theorem as the link between algorithms and information theory. We note here in passing, however, that this theorem implies one of the fundamental source coding results of information theory [3,p.71], namely, that the average number of letters, E[W], required to code a discrete random variable U into D-ary letters such that no codeword is the prefix of another codeword satisfies

$$E[W] \geqslant \frac{H(U)}{\log D}. \tag{8}$$

This can be seen as follows. Any such "prefix condition" code can be used to label the branches of a D-ary rooted tree whose leaf probabilities are the probabilities of the codewords and hence of the corresponding values of U. Thus, $H_{LEAF} = H(U)$. But, since each node has D children, $H_i \leqslant \log D$. Thus, the Leaf Entropy Theorem gives

$$H(U) = H_{LEAF} \leqslant \log D \sum_{i=1}^{N} P_j = E[W] \log D$$

where, for the last equality, we have invoked the Path Length Lemma. This derivation of (8) completely bypasses the Kraft Inequality [3,p.67], which has been the basis of previous derivations of (8), and could thus be regarded as a more fundamental proof.

4. TESTING ALGORITHMS

Suppose that the input data to some algorithm A can be described as a discrete random variable V. Suppose further that U is a random variable uniquely determined by V. Then V induces a probability distribution on U, which, in particular, permits calculation of the entropy H(U) as

$$H(U) = -\sum_{u} P(u) \log P(u).$$

Example 2: Suppose that V is a list of N distinct integers and that U is an N-tuple whose i-th component is the location in the list of the i-th smallest number. Suppose further that the probability distribution over V is such that U is equally likely to be any of the N! permutations of [1,2,..,N]. Then

$$H(U) = \log (N!).$$

We may suppose that the algorithm A is described by a flowchart. The "branching points" or "test points" in the flowchart may be of two types, which we shall call essential and inessential, respectively. An essential branching point (or test point) is one in which the branch followed after that point is reached cannot be determined without having recourse to the input data; an inessential branching point is one where the branch followed does not depend on the input data. Clearly, an algorithm could always be described by a flowchart with no inessential branching points, although inessential branching points (such as those that determine whether some loop has been performed some prescribed number of times) are often convenient to use in flowcharts.

We may also suppose that the outcome of the essential tests of the algorithm are coded in some convenient fashion, e.g., by the integers 0,1,..,D-1 if the test has D possible outcomes. Let $X_1, X_2, \ldots, X_W$ denote the sequence of essential test outcomes when the algorithm A is applied to the input data V. Note that, in general, W will be a random variable. We shall say that A is a testing algorithm for U if (a) the value of U uniquely determines the outcome of every essential test performed and also determines which (if any) essential test will next be performed, and (b) no further essential tests are performed if and only if the sequence of essential test outcomes to that point uniquely determines the value of U.

It follows from this definition of a testing algorithm that the possible executions of the algorithm define a rooted tree in which each leaf corresponds to a possible value u of U. We now assign to

each leaf the probability $P(u)$ specified by the probability distribution for U to obtain a rooted tree with probabilities such that

$$H_{LEAF} = H(U). \tag{9}$$

This assignment of probabilities to the leaves induces a probability assignment on the nodes such that the node probability P_j is the probability that this node will be reached in the execution of the algorithm; the branching entropy H_j is just the uncertainty about the corresponding essential test performed when this node is reached, and the average depth of the leaves $E[W]$ is the average number of essential tests performed.

Example 3: Consider the special case $N = 3$ of Example 2 when $V = [n_1, n_2, n_3]$ is a list of 3 distinct integers. Fig. 2 shows a testing algorithm for the random variable U (where $U = [2,1,3]$ means, for instance that n_2 is the smallest integer, that n_1 is the second smallest and that n_3 is the largest) in which the only type of test used is a comparison of two integers to see which is greatest. Because the $3! = 6$ values of U are equally likely, the corresponding rooted tree with probabilities is that shown in Fig. 3. We see that the branching entropies are

$$H_1 = H_4 = H_5 = 1 \text{ bit}$$

and

$$H_2 = H_3 = h(1/3) = .918 \text{ bits},$$

where we have introduced the binary entropy function

$$h(x) = -x \log x - (1-x) \log (1-x).$$

By the Path Length Lemma, we see that

$$E[W] = P_1 + P_2 + P_3 + P_4 + P_5 = \frac{8}{3} = 2.667$$

is the average number of essential tests used. Finally, we note that

$$H(U) = H_{LEAF} = \log (3!) = 2.585 \text{ bits}.$$

5. BOUNDING COMPUTATION OF TESTING ALGORITHMS

In our study of testing algorithms, we shall equate a computation with the performance of an essential test, and we shall measure the goodness of an algorithm by the smallness of the average number of essential tests that it uses. Our goal here is two-fold:

(i) to underbound $E[W]$ for all testing algorithms of a specified kind, and

(ii) to overbound $E[W]$ for particular testing algorithms after only a rudimentary analysis of their properties.

A. The Basic Bounds

The following simple result turns out to be surprisingly useful.

Theorem 1: The average depth of the leaves in a rooted tree with probabilities satisfies

$$H_{LEAF}/H_{max} \leqslant E[W] \leqslant H_{LEAF}/H_{min}$$

where H_{LEAF} is the leaf entropy, and where

$H_{max} = \sup_j H_j$ and $H_{min} = \inf_j H_j$ are the maximum and minimum node branching entropies, respectively.

Proof: The definitions of H_{min} and H_{max} imply

$$H_{min} \sum_{j=1}^{N} P_j \leqslant \sum_{j=1}^{N} P_j H_j \leqslant H_{max} \sum_{j=1}^{N} P_j .$$

By the Path Length Lemma, the leftmost and rightmost members of this inequality are $E[W]\, H_{min}$ and $E[W]\, H_{max}$, respectively. By the Leaf Entropy Theorem, the central member is H_{LEAF}. This proves the theorem.

This theorem was given previously in [1] but with the restriction, removed here, that the tree be D-ary.

We now have our first bounds on computation.

Corollary 1: If all testing algorithms for U of a specified type have $H_{max} < H_o$, then the average computation of any such testing algorithm satisfies

$$E[W] \geqslant H(U)/H_o .$$

Corollary 2: If a testing algorithm for U has minimum branching entropy H_{min}, then the average computation of this testing algorithm satisfies

$$E[W] \leqslant H(U)/H_{min}$$

Example 4: Consider again the general sorting problem of Example 2 for a specified, but arbitrary, value of N. Consider all testing algorithms for U that use only tests which compare two integers to see which is greater. Because there are only 2 possible outcomes for any test, $H_{max} \leqslant H_o = 1$ bit for all such algorithms. Thus, Corollary 1 specifies that for any such sorting algorithm

$$E[W] \geqslant \log (N!). \qquad (10)$$

For instance, with N = 64, the bound is

$$E[W] \geqslant 296.0 \text{ comparisons.}$$

The bound (10) has long been familiar in sorting theory where it is often referred to as the "information-theoretic bound".

Example 5: For an arbitrary but fixed N, $N \geqslant 3$, consider the following recursive sorting algorithm (of which the algorithm in Fig. 2 is the N = 3 special case): After the first i numbers have been sorted, determine where the (i+1)-st number should be inserted in this list, first by comparing it to the middlemost number of the sorted list (say, that in position $\lceil i/2 \rceil$ where $\lceil x \rceil$ denotes the smallest integer equal to or greater than x), then to the middlemost number in the reduced by half list to which it belongs, etc., until its proper place has been found. When there are j ($j \geqslant 1$) numbers in the sublist whose middlemost member is used for comparison, the test uncertainty will be $h(\lceil j/2 \rceil/(j+1))$, which achieves its minimum for j = 2, namely h(1/3) = .918 bits. Thus,

$$H_{min} = .918 \text{ bits} \,.$$

for this algorithm. Corollary 2 now gives

$$E[W] \leqslant 1.09 \log (N!). \qquad (11)$$

From (10), we see that this sorting algorithm is inferior by at most 9% to the best sorting algorithm, whatever that might be. For N = 64, (11) gives

$$E[W] \leqslant 322.4 \text{ comparisons.}$$

The sorting bound (11) was given in [1]. In the next section, we sharpen this bound by a refinement of Corollary 2.

B. Sharpened Bounds

Suppose now that the nodes in a rooted tree with probabilities have been divided into two sets S_1 and S_2, respectively. Suppose moreover that

$$H_{1min} = \inf_{j \varepsilon S_1} H_j > H_{2min} = \inf_{j \varepsilon S_2} H_j \tag{12}$$

and let

$$\Delta H_{2min} = H_{1min} - H_{2min}. \tag{13}$$

It then follows that

$$\sum_{j=1}^{N} P_j H_j \geqslant \sum_{j=1}^{N} P_j \left[H_{1min} - \theta_2 \Delta H_{2min} \right] \tag{14}$$

where

$$\theta_2 = \left(\sum_{j \varepsilon S_2} P_j \right) / \left(\sum_{j=1}^{N} P_j \right). \tag{15}$$

Using the Leaf Entropy Theorem and Path Length Lemma in (14) now gives the following result.

<u>Theorem 2</u>: For the partition of the nodes of a rooted tree with probabilities into two classes and with the defintions given in (12), (13) and (15),

$$E[W] \leqslant \frac{H_{LEAF}}{H_{1min} - \theta_2 \Delta H_{2min}}.$$

<u>Example 6</u>: For the tree in Fig. 3 and the choice $S_1 = \{1,4,5\}$ and $S_2 = \{2,3\}$, we have $H_{1min} = 1$ and $H_{2min} = h(1/3) = .918$ bits. Thus, $\Delta H_{2min} = .082$ bits. Also $\theta_2 = (1/2 + 1/2)/(8/3) = 3/8$. Recall from Example 3 that $H_{LEAF} = 2.585$ bits. Theorem 2 now gives the bound

$$E[W] \leqslant 2.667,$$

which is actually an equality as can be seen from Example 3.

In order to apply the bound of Theorem 2 to algorithms, we need a convenient way to find (or to overbound) θ_2. It is instructive first to get a physical interpretation of θ_2 as defined by (15) when applied to the tree of a testing algorithm. We claim that θ_2 is the fraction of time that the essential test performed corresponds to a node in S_2 in many executions of the testing algorithm for independent samples of U. To see this, note that P_j is just the probability that node j is reached in one execution of the algorithm. Thus, in a large number M of executions of the algorithm, this node would be reached about MP_j times, and nodes in S_2 would be reached about

$$M \sum_{j \in S_2} P_j$$

times. The total number of nodes reached would be about

$$M \sum_{j=1}^{N} P_j .$$

Thus, the fraction of nodes reached that are in S_2 would be about θ_2 as given by (15), and this becomes a law-of-large-numbers "certainty" as $M \to \infty$. Actually, we do not need to be more precise about θ_2 here, as we intend only to argue that if $\tilde{\theta}_2$ is any upper bound on the fraction of times that a test in S_2 is used on every particular execution of the algorithm, then surely

$$\theta_2 \leqslant \tilde{\theta}_2$$

and we obtain the following result.

Corollary: If $\tilde{\theta}_2$ overbounds the fraction of essential tests that are in S_2 on every execution of a testing algorithm for U, then the average computation of this algorithm satisfies

$$E[W] \leqslant \frac{H(U)}{H_{1min} - \tilde{\theta}_2 \, \Delta H_{2min}}$$

where H_{1min} and ΔH_{2min} are defined by (12) and (13), respectively.

Example 7: Consider the sorting algorithm of Example 5 and let S_2 consist of all nodes j where $H_{2min} = H_j = h(1/3) = .918$ bits. The branching entropy of all nodes in S_1 is at least $h(2/5) = .971$ bits, as the worst case in S_1 occurs when a number is to be inserted into a sublist of four numbers. Thus, $H_{2min} = .053$. It is readily checked that, for $N \geqslant 8$, the fraction of time that a test in S_2 will be used [i.e., the fraction of times that a number is to be inserted into

a sublist of two numbers] is less than 1/3, i.e.,

$$\tilde{\theta}_2 < 1/3.$$

Using this bound in the Corollary gives, for $N \geq 8$,

$$E[W] \leq 1.05 \log (N!) \tag{16}$$

so that we have substantially improved our upper bound (11) on E[W] for this algorithm. This sorting algorithm is inferior by less than 5% to any other!

It should be obvious that Theorem 2 can be generalized by partitioning the nodes into any number of subsets rather than two. We leave the details to the reader, but remark that bounding the fraction of time that an algorithm spends in each subset generally becomes more difficult as more subsets are used and our motivation is to avoid tedious analysis. We content ourselves in the remainder of this paper to applying the above techniques to an example more in the realm of communications than the sorting example used above.

6. POLLING ALGORITHMS

Consider a multi-access communications system with N sending stations. Let U be the binary N-tuple whose i-th component is a 1 if and only if station i has a message to transmit. We assume that each station independently has probability p of having such a message so that

$$H(U) = Nh(p). \tag{17}$$

A polling algorithm is a testing algorithm of the following type. At each step, some subset of the stations is "enabled" by the central controller, i.e., given permission to send some signal if they have a message to send. The central controller learns from this feedback only whether no station in the enabled set has a message to send (absence of signal)or whether at least one station in the enabled set has a message to send (presence of signal).The central controller continues to poll in this fashion until he has determined precisely which stations have messages to send,i.e.,until he has determined the value of U. Since each test has only two outcomes, $H_o = 1$ bit is an upper-bound on the maximum branching entropy of every essential test performed by the central controller. It follows then from (17) and Corollary 1 that the average number of polls made by the central controller satisfies

$$E[W] \geq Nh(p) \tag{18}$$

for any such polling algorithm.

Example 8: With $N = 30$ stations and $p = .110$ which gives $h(p) = .500$ bits , the bound (18) gives

$$E[W] \geq 15.0 \tag{19}$$

for any polling algorithm. Note that the average number of stations that have messages to send is only $Np = 3.30$.

We shall say that a station belongs to the fully unknown set of stations at some time if that station still has probability p of having a message to send, given the results of all polls conducted up to that time by the central controller. Clearly a station is in the fully unknown set if it has not yet been enabled. Suppose that some subset B of the fully unknown set is enabled, that the response indicates one or more stations have messages (which removes the subset B from the fully unknown set), and that a subset E of B is then enabled. If the response again indicates one or more busy stations, it is easy to see that those stations in B but not in E return to the fully unknown set.

We shall say that a station belongs to a one-plus busy set at some time if the only information about this set of stations derivable from the results of all polls conducted up to that time is that there is at least one station in the set with a message to send. Clearly an enabled subset E of the fully unknown set becomes a one-plus busy set if the response indicates that one or more stations have messages. Suppose that B is a one-plus busy set and that a subset E of B is enabled. If the response indicates that no station has a message, then it is easy to see that those stations in B but not in E now form a one-plus busy set.

These considerations motivate the following polling algorithm that depends on a single positive integer parameter n and that, after each response, has all stations whose status is in doubt either in the fully unknown set F or in a single one-plus busy set B. For convenience, let M be the set of stations found to have a message to send, let E be the set of stations enabled in a poll, and let X denote the response to a poll in the manner that $X = 1$ indicates that one or more of the enabled stations have messages to send, while $X = 0$ indicates that none do.

Step 0: (Initialization) $F \leftarrow \{1,2,..,N\}$, $B \leftarrow \emptyset$, $M \leftarrow \phi$.

Step 1: If $\#(F) < n$, go to step 2. Otherwise, enable the first n stations in F, and $F \leftarrow F - E$. If $X = 0$, return to step 1. If $X = 1$, then $B \leftarrow E$ and go to step 3.

Step 2: If $F = \emptyset$, stop. Otherwise enable all stations in F, and $F \leftarrow \emptyset$. If $X = 0$, stop. If $X = 1$, then $B \leftarrow E$ and go to step 3.

Step 3: If $\#(B) = 1$, then $M \leftarrow M \cup B$, $B \leftarrow \emptyset$, and go to step 1. Otherwise, go to step 4.

Step 4: Enable the first $\lfloor i/2 \rfloor$ stations in B where $i = \#(B)$ [and where $\lfloor x \rfloor$ denotes the integer part of x]. If $X = 0$, then $B \leftarrow B - E$ and to to step 3. If $X = 1$, then $F \leftarrow F \cup (B - E)$, $B \leftarrow E$ and go to step 3.

Let $q = 1 - p$ be the initial probability that a station is idle. Then, for the above algorithm, we have

$$P(X = 0 \mid \text{step } 1) = q^n \tag{20}$$

and

$$P(X = 0 \mid \#(E) = m, \text{ step } 2) = q^m, \; 2 \leq m < n \tag{21}$$

as follows from the fact that E is a subset of the fully unknown set in both step 1 and step 2. Similarly, we find that

$$P(X = 0 \mid \#(B) = i, \text{ step } 4) = \frac{q^{\lfloor i/2 \rfloor} - q^i}{1 - q^i}, \; 2 \leq i \leq n \tag{22}$$

as follows from the facts that $\#(E) = \lfloor i/2 \rfloor$ and that the original probabilities of events are now increased by the factor $1/(1-q^i)$ because of the conditioning that not all i stations in B are idle.

We now choose the parameter n so that the above polling algorithm will give a small average number of polls, $E[W]$, when N is large. We observe first that step 2 is reached only in the "end game" so that the number of tests there will have little influence on $E[W]$. Thus, we choose to ignore step 2 for the moment. To make step 1 efficient, i.e., to maximize the branching entropy of the test performed there, it follows from (20) that we should choose n so that q^n is as close to 1/2 as possible.

Example 8 (continued): For $q = 1 - p = .890$, the choice $n = 6$ gives the best approximation to 1/2, namely

$$q^6 = .497.$$

It follows that all tests performed in step 1 of the polling algorithm have branching entropy

$$H_j = h(.497) = 1.000 \text{ bits} \tag{23}$$

which is very good indeed.

We now turn our attention to step 4 of the algorithm, and we discover that we have been lucky. The fact that

$$q^n \approx \frac{1}{2} \tag{24}$$

ensures that the right side of (22) will also be reasonably close to 1/2 when $p = 1 - q$ is small, as we would expect in a multiple access system appropriate for polling.

Example 8 (continued): For $q = .890$ and $n = 6$, the right side of (22) becomes .471, .373, .442, .529, and .413 for $i = 2,3,4,5$ and 6, respectively. The corresponding branching entropies are .998, .953, .990, .998 and .978 bits, respectively, for these step 4 tests.

Finally, we must face up to those tests performed in step 2 as our polling algorithm goes into its "end game".

Example 8 (continued): For $q = .890$ and $n = 6$, the right side of (21) becomes .890, .792, .705, .627, and .558 for $m = 1,2,3,4$ and 5, respectively. The corresponding branching entropies are .500, .737, .875, .953, and .990 bits, respectively, for these step 2 tests.

We now partition our test nodes into the sets S_1 and S_2 as described in Section 5B, putting into S_1 all test nodes whose branching entropies are at least as great as the smallest branching entropy of a test performed in step 4 of the polling algorithm. It remains only to determine an upper bound $\tilde{\theta}_2$ on the fraction of time that the algorithm will perform tests in S_2 on every execution of the algorithm in order to be able to use the Corollary of Theorem 2 to overbound the average number of polls performed by our algorithm.

Example 8 (concluded): For the specified choice of S_1 and S_2, we see that S_2 contains only those step 2 tests for which the size m of the enabled set is 2 or 3. Moreover, $H_{1min} = .953$, $H_{2min} = .500$ and $\Delta H_{2min} = .453$ bits. It is easy to check that, for $N = 30$, the maximum fraction of tests in S_2 occurs on that execution of the algorithm in which the first four tests are in step 1 and give $X = 0$, the fifth is in step 1 and gives $X = 1$, the sixth in step 4 gives $X = 1$, the seventh in step 2 with $m = 3$ gives $X = 1$, the eighth in step 4 gives $X = 1$, the ninth in step 2 with $m = 2$ gives $X = 1$, the tenth in step 4 gives $X=1$, and the eleventh and last in step 2 with $m = 1$ gives $X = 0$. Thus, $\tilde{\theta}_2 = 3/11$. Using these values in the Corollary of Theorem 2 now gives the upper bound

$$E[W] \leqslant \frac{15.0}{.953 - (3/11)(.453)} = 18.1 \tag{25}$$

on the average number of polls performed by our polling algorithm.

Comparison of the bound (25) to the lower bound (19) on any polling algorithm shows that our simple polling algorithm cannot be far from optimal.

The reader may notice that in fact the polling algorithm analyzed here is an example of "nested group testing" as described elsewhere in this volume [4].

7. DISCUSSION

We have described an information-theoretic approach to algorithms that seeks to obtain useful upper and lower bounds on the performance of an algorithm from only a cursory analysis of the algorithm itself. As our polling example illustrated, the nature of these bounds can sometimes suggest an efficient algorithm for the problem at hand. The avoidance of the need to analyze in detail the computation performed by the algorithm is perhaps the chief virtue of our approach.

There are four limitations to our approach that should be mentioned. The first is that we require a probability distribution over the input data for the algorithm. This is hardly inconvenient as most problems suggest a natural probability distribution. The second limitation is that our computational measure is an average rather than worst-case measure; in most cases, this would seem to be an advantage rather than a limitation, particularly if the algorithm is to be executed many times. The third limitation is more serious, namely, that we identify a computation with a test, and all tests are thus given equal weight regardless of their "complexity". This can sometimes be compensated for by splitting a complex test into several smaller tests, or by combining simple tests performed consecutively into one test, but it remains a drawback of our approach. The fourth and most serious limitation is that the algorithm must be a testing algorithm for a random variable U determined by the data. We have found it rather surprising that algorithms for dividing two integers (when U is taken as their quotient) and for finding the greatest common divisor of two integers (when U is taken as the pair of integers after division by the greatest common divisor) turn out to be testing algorithms. But algorithms for such problems as finding the maximum in a list of N unequal integers are not testing algorithms for any U determined from the data. Although testing algorithms turn up in more places than one might expect, our approach to algorithms will not be very satisfying until it can be generalized to a much broader class of algorithms. We would be delighted if this paper would stimulate someone to make such a generalization.

REFERENCES

1. Massey, J.L., An Information-Theoretic Approach to Data Processing Algorithms, Abstracts of Papers, IEEE Int. Symp. Info. Th., IEEE Cat. No. 74 CHO 883-9 IT, 1974.

2. Fano, R.M., Transmission of Information, M.I.T. Press, Cambridge, Massachusetts, 1961.

3. Hartmann, C.R.P., Varshney, P.K., Mehrotra, K.G., and Gerberich, C.I., Application of Information Theory to the Construction of Efficient Decision Trees, IEEE Transactions on Information Theory, 28, 565, 1982.

4. Wolf, J.K., Principles of Group Testing and an Application to the Design and Analysis of Multi-Access Protocols, this volume, 1983.

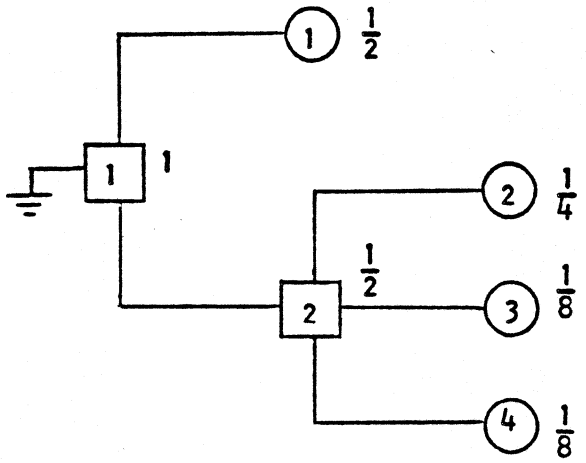

Fig. 1 A rooted tree with probabilities

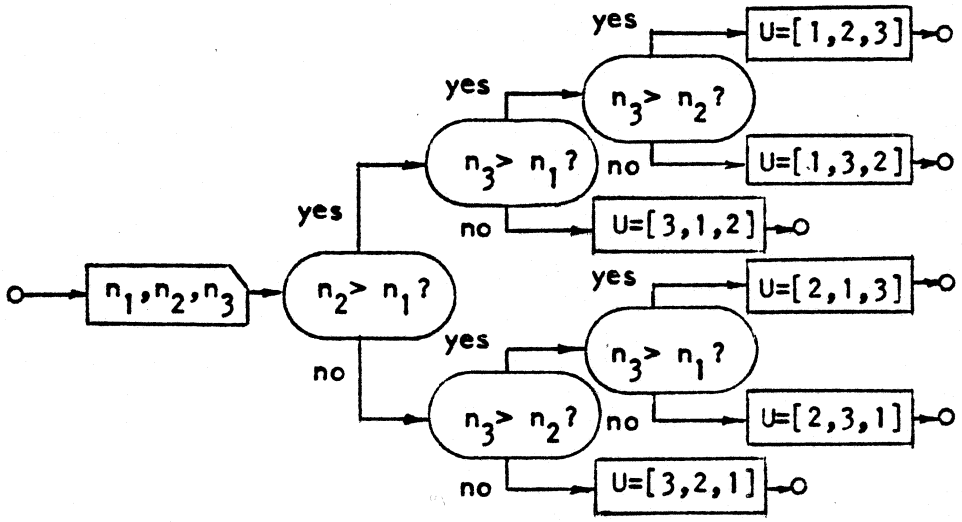

Fig. 2 An algorithm for sorting three distinct integers

Fig. 1

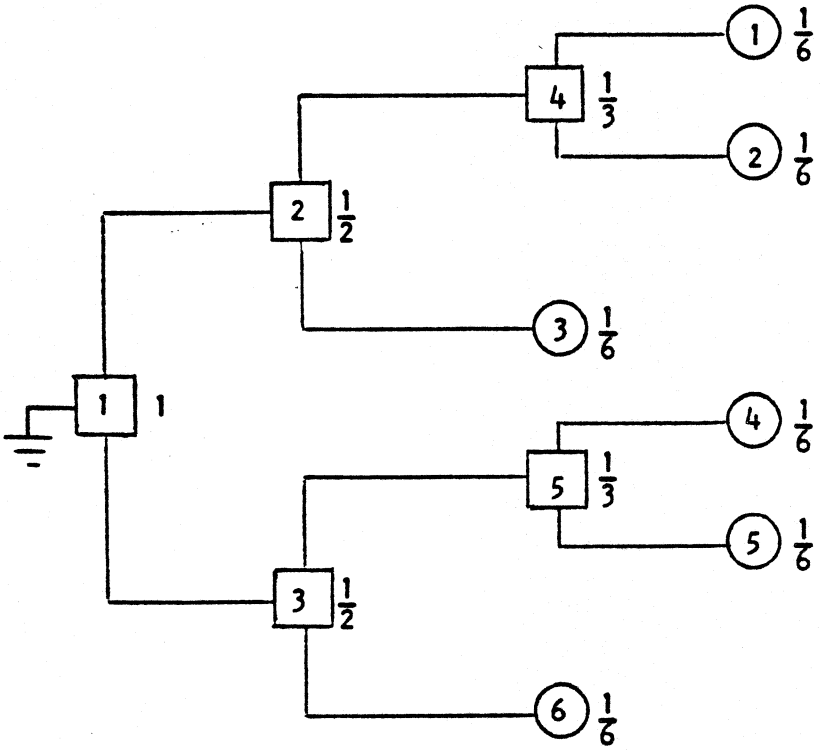

Fig. 3

Fig. 3 The rooted tree with probabilities for the algorithm of Fig. 2 when all values of U are equally likely.